Long Short-Term Memory Recurrent Neural Networks for Signature Verification

by

CONRAD TIFLIN

A thesis submitted in fulfilment to the requirements

for the degree of

Magister Scientiae

in the Department of Computer Science

University the Western Cape

Promoter: Prof. Christian W. Omlin

December 2003

Long Short-Term Memory Recurrent Neural Networks for Signature Verification

CONRAD TIFLIN

Keywords

Long Short-Term Memory, Recurrent Neural Networks, Time Series Modelling, Long-term Dependencies, Signature Verification, ROC Curves.

Abstract

Handwritten signature verification is defined as the classification process that strives to learn the manner in which an individual makes use of the muscular memory of their hands, fingers and wrist to reproduce a signature. A handwritten signature is captured by a pen input device and sampled at a high frequency which results in time series with several hundred data points. A novel recurrent neural network architecture known as long short-term memory was designed for modelling such long time series. This research investigates the suitability of long short-term memory recurrent neural networks for the task of on-line signature verification. We design and experiment with various network architectures to determine if this model can be trained to discriminate between authentic and fraudulent signatures. We further determine whether the complexity of a signature impacts on the performance level of the network when applied to fraudulent signatures. We also investigate the performance level of the network when varying the number of signature features.

The results obtained are a clear indication that long short-term memory recurrent neural networks can model handwritten signatures. Our experiments also show that complex signatures are more difficult to forge. Also, the network's ability to discriminate between authentic and fraudulent signatures improves when we increase the size of the signature feature vector.

Declaration

I declare that *Long Short-Term Memory Recurrent Neural Networks for Signature Verification* is my own work, that it has not been submitted for any degree or examination in any other university, and that all the sources I have used or quoted have been indicated and acknowledged by complete references.

CONRAD TIFLIN

December 2003

Acknowledgments

I wish to convey my most profound appreciation to my supervisor Prof. C.W. Omlin for his valuable contribution and expertise. His motivation and guidance contributed significantly to the completion of this dissertation.

Further we would like to express our appreciation to Dr. Douglas Eck formally from IDSIA (Istituto Dalle Molle di Studi sull'Intelligenza Artificiale) currently an Assistant Professor in the Department of Computer Science and Operations Research at the University of Montreal, Canada for his assistance and the use of the LSTM RNN implementation known as *rete*. We would also like to convey our gratitude to Dr. J.G.A. Dolfing from the Philips Research Laboratories in Aachen, Germany for the use of the signature database. We are grateful to the Telkom-Siemens Centre of Excellence for funding which enabled me to pursue this research.

Finally a sincere debt of gratitude to my family for their guidance, support and inspiration.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recurrent neural networks (RNNs) are difficult to understand due to their high dimensional, nonlinear and dynamic nature [34]. They are, however, ideally suited for modelling dynamical systems with hidden states whose output is typically recorded in the form of time series. The two primary objectives of time series analysis are prediction of future values, i.e. given observations y(1), y(2), ..., y(n), predict future values y(n+1), y(n+2), ... and time series classification. Various machine learning methods have been applied to time series prediction and classification tasks such as speech recognition [38], electroencephalogram classification [60] and dynamic gesture recognition [9].

A considerable amount of research has been carried out in the area of handwritten signature verification using both offline and online approaches. The best results reported in the literature have been attained for on-line signature verification where the entire signing process is monitored; thus, dynamic signature information is available. In this thesis, we apply a novel RNN known as long short-term memory (LSTM) for time series classification that is able to discriminate between genuine signatures and forgeries. The high sampling frequency of on-line signatures

results in time series with several hundred data points. LSTM RNNs were designed for modelling such long series.

1.1 Research Hypotheses

An individual's handwritten signature is a dynamic time-varying process and can be captured and represented as a time series,

$$[x(t), y(t), p(t), \theta_x(t), \theta_y(t)]$$

where $x(t)$ and $y(t)$ represent the x and y pen-tip co-ordinates, p(t) the pen-tip pressure, $\theta_x(t)$ and $\theta_y(t)$ the polar angles of the pen. The complexity of signatures is established through visual inspection and fall into the following categories: easy, moderately easy, and difficult to forge.

The hypothesis of our research is that LSTM RNNs can be trained to discriminate between genuine and fraudulent signatures. Secondly, we believe that the complexity of a signature impacts on the performance level of a LSTM RNN when applied to casual, skilled and forensic forgeries. Lastly, an increase in the number of significant features used to discriminate between genuine and fraudulent signatures of varying complexity results in a diminished misclassification rate of fraudulent signatures by the network.

1.2 Technical Objectives

The primary aim of this thesis is to apply Long Short-Term Memory Recurrent Neural Networks to the task of on-line signature verification, which to our knowledge has not yet been attempted.

Traditional RNNs have difficulties in dealing with time series that contain long-term dependencies. The result of experimentation conducted on these networks will indicate if this can be done, as well as the suitability of this modelling technique in discriminating between these signature exemplars. In this thesis, we further seek to determine how the complexity of a signature affects the performance of the model. Finally, we will identify how signature features affect the performance of the model on casual, skilled and forensic forgeries.

1.3 Methodology

Handwritten signature reproduction is based on muscular movement. The sequence of states of the muscles involved in this process is not available, i.e. they are hidden and hence verification can only be applied to the output of these processes. Various methods including hidden Markov models (HMMs) and support vector machines (SVMs) have been applied to this problem with varying degrees of success [39, 66]. However, since signature classification involves modelling of hidden states, recurrent neural networks are computationally better suited than other adaptive models. HMMs do not contain continuous internal states, i.e. hidden states must be modelled explicitly, whereas feedforward neural networks and SVMs contain no internal states at all [1].

We have applied LSTM to handwritten signatures and adjusted various network parameters to obtain the desired results through training. Experimentation was conducted to investigate if a single LSTM RNN is capable of discriminating between genuine and fraudulent signatures. We then trained three LSTM RNNs to model signatures belonging to various complexity groups. Finally, we varied the number of signature features to determine how it impacts on the performance level of the

network. These results are reported in receiver operating characteristic curves that estimate the performance of this learning system.

1.4 Accomplishments

We applied LSTM RNNs to the problem of on-line handwritten signature verification. We give the reader insight into various time series modelling approaches and supply sufficient motivation for using LSTM RNNs as a solution to our problem statement. The performance levels attained by the network were sufficiently satisfactory to prove the feasibility of applying LSTM RNNs to dynamic signature verification and also provided sufficient evidence to validate our hypotheses.

1.5 Thesis Outline

The content of this thesis is arranged as follows: In Chapter 2, we present various neural network time series prediction techniques as well as some application domains; we conclude with problems pertaining to modelling long time series. In Chapter 3, we present the theory of long short-term memory recurrent neural networks and discuss a few of its applications. We then present our application of long short-term memory recurrent neural networks to on-line signature verification in Chapter 4, as well as various aspects of this problem domain. In Chapter 5, we discuss the results of our experiments and conclude with possible directions for future research.

Chapter 2

Time Series Prediction with Artificial Neural Networks

2.1 Introduction

Human beings are capable of gathering vast amounts of sensory data from their surroundings which in turn enables them to formulate logical decisions. This data can be represented as a time series which the brain organizes and performs complex operations that allow us to predict and classify sequences in nature. Artificial neural networks (ANNs) are simple mathematical models devised in an attempt to emulate some of these human brain functions.

Recurrent neural networks (RNNs) are ideally suited for modelling dynamical systems with hidden states. The output of such processes are typically recorded in the form of time series. The two primary objectives of time series analysis are prediction of future values, i.e. given observations y(1), y(2), ..., y(n), predict feature values y(n+1), y(n+2), ... and time series classification such as in the recognition of phonemes for speech processing [38].

Various machine learning methods have been applied to time series prediction and

classification tasks such as electroencephalogram classification [60] and dynamic gesture recognition [9]. In our presentation, we will focus on the classification of signature time series data using RNNs. Throughout this thesis, we assume that time is quantized into discrete steps, since a handwritten signature extends for a fixed number of time steps.

We give a brief introduction to neural networks followed by various neural network topologies that have been applied to time series prediction tasks. We conclude with an explanation of the long-term dependency problem that is inherent in traditional RNNs.

2.2 Neural Networks Overview

ANNs are subdivided into two classes, namely feedforward and recurrent. Feedforward neural networks (FNNs) [56] contain no explicit feedback connections. Conventional FNNs are able to approximate any finite function as long as there are enough hidden nodes to accomplish this [23].

ANNs store knowledge in synaptic weights. During this learning process, these weights are adjusted in order to approximate a desired input-output mapping with a some degree of accuracy.

Two types of learning methods exist: supervised or associative learning and unsupervised or self-organizing learning. The former requires an input pattern along with matching output patterns which is given by an external teacher whereas the latter requires input patterns from which it develops its own representation of the input stimuli.

2.3 Feedforward Neural Networks

A multi-layer feedforward neural network (FFNN) (Figure 2.1) consists of layers of neurons which are connected via weights which may either be fixed or random values. These networks also comprise of one or more hidden layers containing hidden units which are able to extract higher-order statistics [23]. This is particularly valuable when the size of the input layer is large. Activation functions in the network may be those whose output is a nonlinear differential function, e.g. sigmoid function, of its inputs and hence suitable for gradient descent learning.

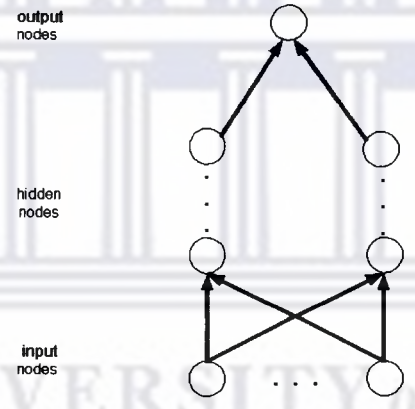

Figure 2.1: **A FeedForward Neural Network**

The backpropagation learning algorithm and the generalised delta rule are common gradient descent approaches that are used to train these static networks. The error backpropagation algorithm is defined according to [23] as follows:
Training examples are presented to a neural network in the form $(\overline{x}, \overline{t})$ where $\overline{x}$ is a vector of input data, and $\overline{t}$ is the vector of desired teacher signals.

- We construct a feedforward network with n_{in} inputs, n_{hidden} units, and n_{out} output units.
- We initialise the networks synaptic weights to a random value.
- We repeat the following steps until the termination condition is met:
  - For each $(\overline{x}, \overline{t})$ in the training set, propagate the input forward through the network.
  - backpropagate the error through the network:
    * For each network output unit k, calculate its error term δ_k

      $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
    * For each hidden unit h, calculate its error term δ_h

      $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$
    * The following weight update rule is then applied

      $\Delta w_{ji} = \eta \delta_j x_{ji}$

Two basic methods exist for backpropagation learning [23]: They are sequential or stochastic mode and batch mode backpropagation. The former learning approach updates weights after each training example is presented to the network, whereas the latter approach requires that the entire training set be presented to the network followed by the weight updates.

Conventional FFNNs are not able to deal with time-varying input since they are static learning devices, i.e. the number of input nodes in the network are fixed based on the task at hand. They can, however, be adapted to deal with temporal relationships as we will see in the next section.

2.4 Time Delay Neural Networks

Time delay neural networks (TDNN) is an architecture developed by [63] specifically for speech recognition. The purpose of this architecture is to have a network that contains context which is able to represent sequences. In TDNNs, context or short-term memory is represented as input history. The number of time steps that a TDNN is able to process depends entirely on the time window that stores the input sequence. [40] notes that the buffer size limits the length of longest sequence which can successfully be differentiated by these networks. When dealing with time series problems, we must have a good idea of what the size of the longest sequence in the dataset will be.

The reason for this is simply that, in TDNNs, a fixed input window size has to be selected based on the longest sequence length. If this prior information is not known, the sequence of course cannot be stored in the time window, thus making processing of the sequence impossible. The number of input to hidden layer weights increases with increasing window size. The increased number of parameters increases the time complexity of the learning algorithm.

The input layer of a TDNN consists of a sliding window whose weight vector is shared amongst other inputs. The output of the activation units is computed by taking a weighted sum of the nodes residing in the input window over a time period and then applying a squashing function to it. TDDNs are trained with the conventional error-backpropagation learning algorithm.

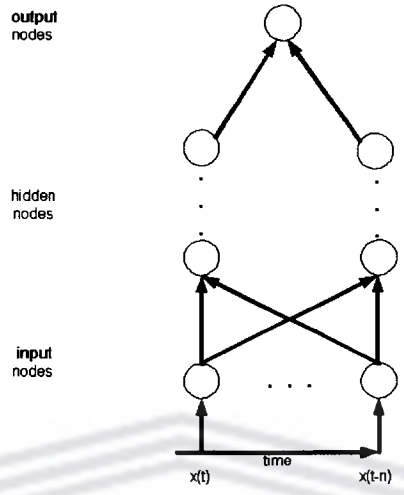

Figure 2.2: **A Time Delay Neural Network**

TDNNs have been applied to a control problem in [35]. Data was recorded from a model-based controller for a robot performing a peg-into-hole assembly task, see Figure 2.3.

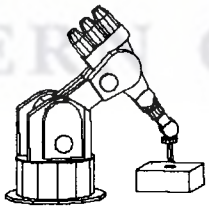

Figure 2.3: **The Robot Controller:** This figure illustrates the peg-into-hole task.

The data set consisted of 3000 single samples each of which represents an insertion of a peg into a hole. The samples comprise three forces F_x, F_y, F_z, three torques about a point M_x, M_y, M_z, three translational velocities V_x, V_y, V_z and three angular

velocities ω_x, ω_y, ω_z. These samples are represented as signals in Figure 2.4.

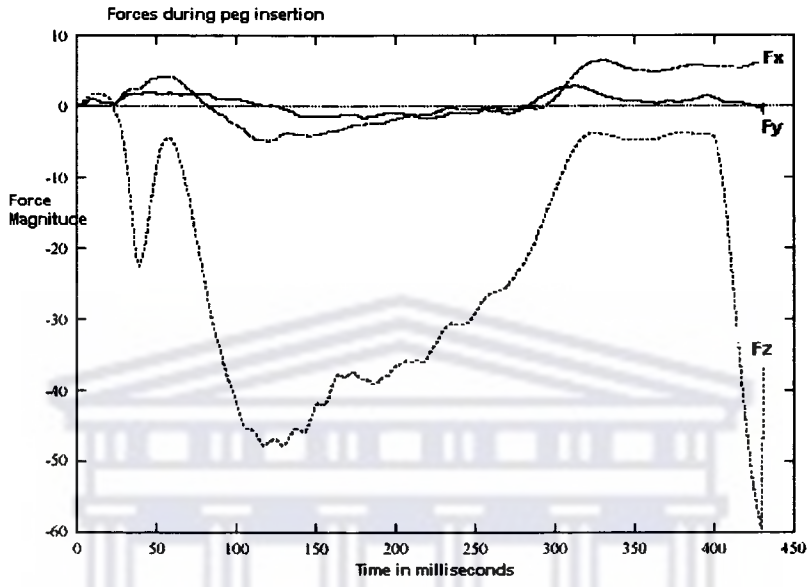

Figure 2.4: **Force signals recorded during peg insertion**

The TDNN's purpose was to approximate a mapping of $force/torque \rightarrow$ velocity. An error rate of 0.072 for the output V_x , 0.125 for the output V_y, and 0.021 for the output V_z was achieved, which indicates that a TDNN is able to model such a task.

2.5 Finite Impulse Response and Infinite Impulse Response Neural Networks

Finite impulse response (FIR) and infinite impulse response (IIR) neural networks, in Figure 2.3, were developed by [3] specifically for the task of nonlinear time series prediction. They are based on the traditional FFNN architecture with the exception that each weight is replaced by a FIR/IIR linear filter. FIR networks contain time delays and they do not have recurrent connections, whereas the IIR networks have

connections that are locally recurrent. Both of these networks are still globally feed-forward in nature.

The FIR filter allows input to be stimulated for a finite period of time, which results in the output activation of the filter also being produced for a finite period of time. An important point that [64] mentions is that FIR networks are functionally equivalent to TDNNs.

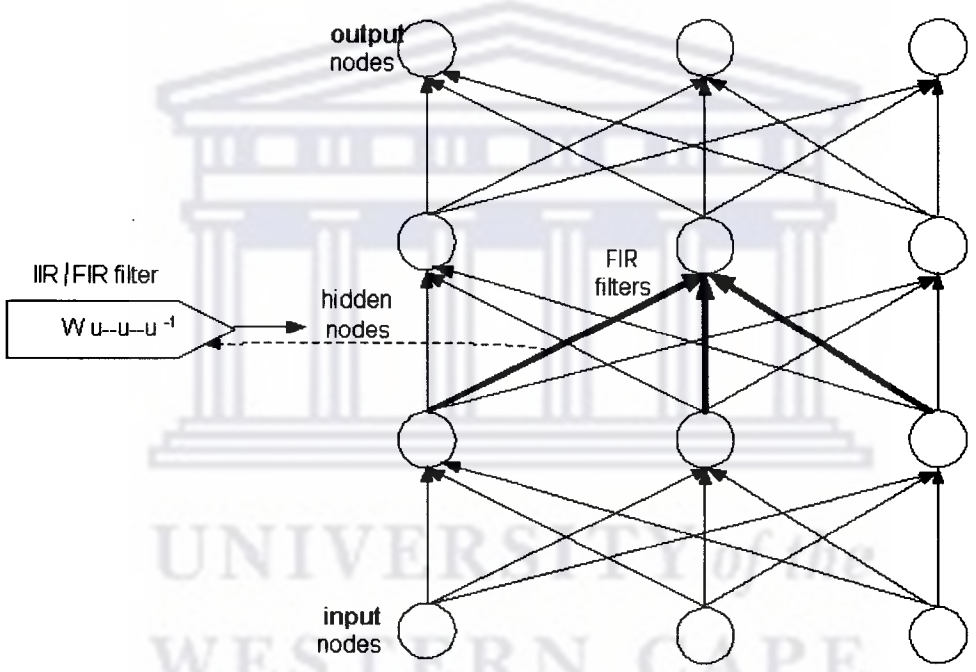

Figure 2.5: **A FFN network with FIR/IIR synapses**

This FIR filter produces an output, y(k), that represents the weighted sum of the current and past inputs, x(k).

$$y(k) = \sum_{n=0}^{T} w(n)x(k-N)$$

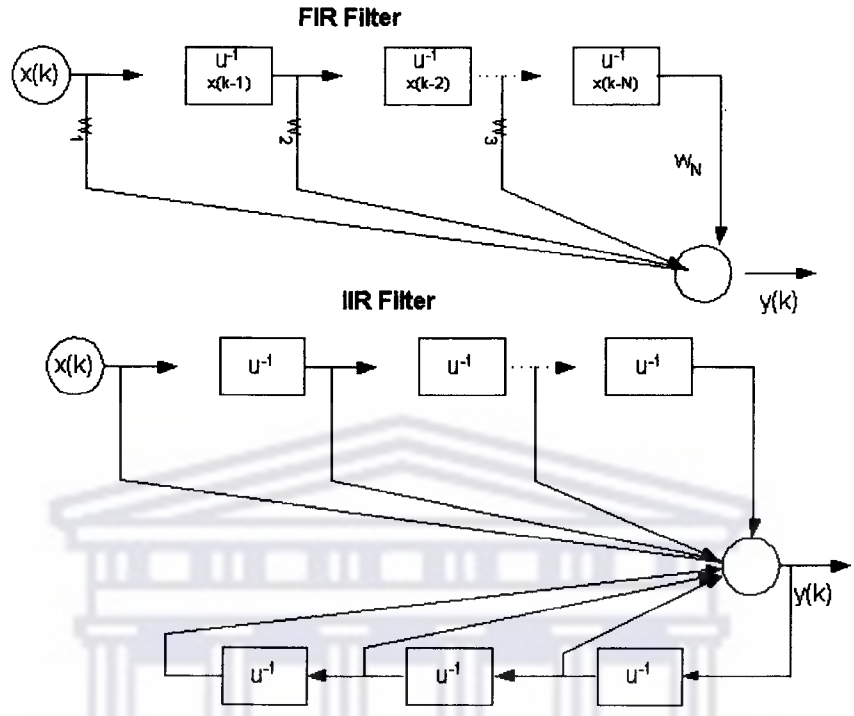

Figure 2.6: **FIR/IIR filters**

This is then passed through a squashing function, i.e. sigmoid function which results in the activation of the neuron.

$$z(k) = f(y(k))$$

The IIR filter has the form:

$$y(k) = \sum_{n=0}^{T} a(n)x(k-n) + \sum_{m=0}^{M} b(m)y(k-m)$$

Figure 2.6 illustrates FIR and IIR filters which contain weighted tapped delay lines. The unit delay operator u^{-1} represents the input at a given time step, i.e. $x(k-1) = u^{-1}x(k)$. The learning algorithm for these networks is known as temporal backpropagation. The reader may consult [64] and [3] for its derivation.

[3] tested the performance of both FIR and IIR networks on a time series generated by the following function:

$$y(t) = \sin\left\{\pi\left[\frac{\beta_1(q^{-1})}{1} - \alpha_1(q^{-1}) - \alpha_2(q^{-2})x(t)\right]\right\} \tag{2.1}$$

where $x(t)$ is a zero mean white noise source, low-pass filtered with a cut-off frequency of 7 rad/sec, with $\alpha_1 = 0.8227$, $\alpha_2 = -0.9025$, and $\beta_1 = 0.99$. [3] notes that these parameters highlight the dynamics of the system and its nonlinearity. This problem stems from nonlinear control systems which occurs in a wide range of applications used in engineering and science. Some examples include nonlinear circuits, mechanical systems, robotics, chemical processes, flight control, jet engine control, evolutionary systems and biological systems. The time series generated by Equation 2.1 approximates a particular real-world control system.

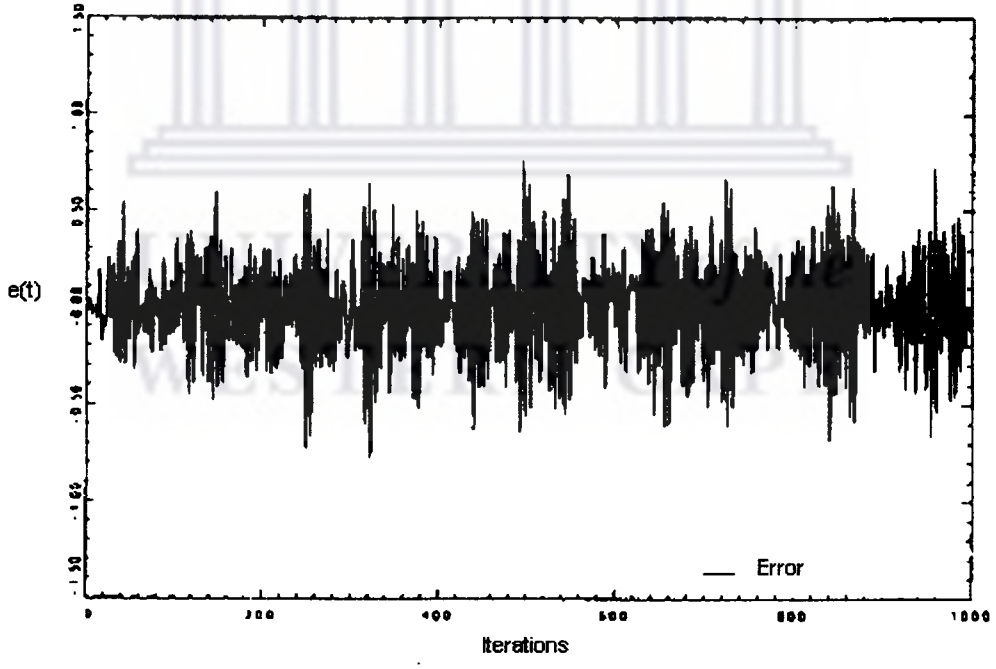

Figure 2.7: **FIR Testing Mean Square Error:** This figure illustrates the testing mean square error of 0.0664.

The FIR and IIR networks were trained by [3] on 5×10^6 data points generated from the above equation. The test set consisted of unseen generated data of 1000 points. A testing mean square error of 0.0664 in Figure 2.7 was obtained for the FIR network. For the IIR network, a testing mean square error of 1.2×10^{-5} was achieved as depicted in Figure 2.8.

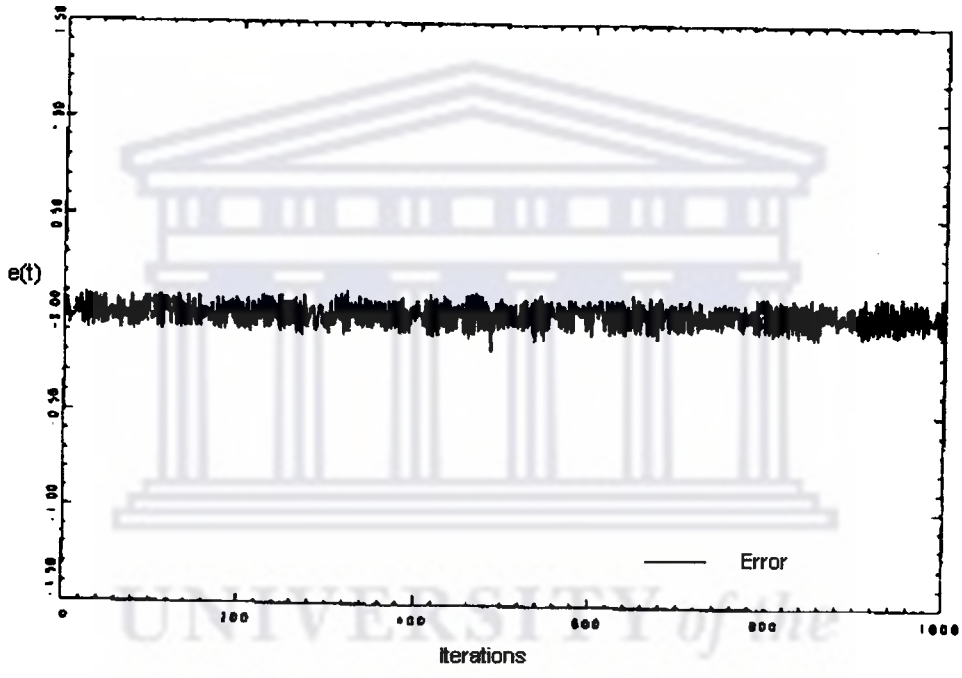

Figure 2.8: **IIR Testing Mean Square Eerror:** This figure illustrates the testing mean square error of 1.2×10^{-5}.

[3] concludes that the IIR network achieves a lower error rate thus making it a more efficient model than a FIR network for this given task. The reason for this is that networks which have local-feedback connections, i.e. IIR networks, perform better than those with only local feedforward connections, i.e. FIR networks [3].

2.6 Bi-directional Neural Networks

A bi-directional computation is able to improve future value prediction by adding values that are predicted from the past [68]. The reason for this is that past values are related to those that occur in the future [69]. The bi-directional neural network thus makes use of past and future values.

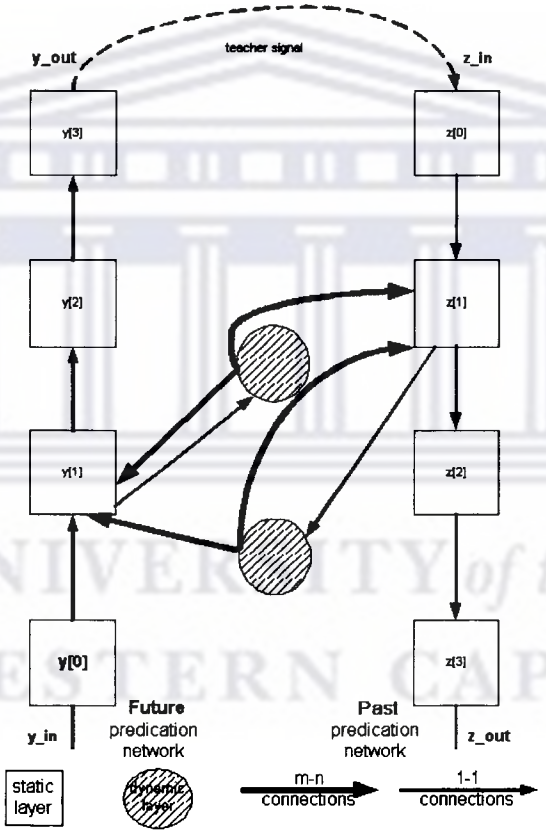

Figure 2.9: **A Bi-Directional Neural Network Model:** A pattern which fed into the future prediction network flows in the opposite direction to a pattern which had been fed into the past prediction network.

A pattern which fed into the future prediction network flows in the opposite direction to a pattern which had been fed into the past prediction network. This model, Figure 2.9, consists of an input layer, $y^{[0]}$, two hidden layers $y^{[1]}$ and $y^{[2]}$ and an output layer, $y^{[3]}$. [68] further notes that for a pattern flowing in a given direction is computed as follows:

$$y_i^{[0]} = [y_{in}]_i,$$
$$y_i^{[1]} = f1(\sum_j w_{ij}^{[1]} y_j^{[0]} + \sum_j w_{ij}^{[F]} s_j^{[F]} + \sum_j w_{ij}^{[P]} s_j^{[P]}),$$
$$\tau ds_j^{[F]}/dt + s_j^{[F]} = y_j^{[1]},$$
$$y_i^{[2]} = f_2(\sum_j w_{ij}^{[2]} y_j^{[1]}),$$
$$[y_{out}]_i = y_i^{[3]} = f_3(\sum_j w_{ij}^{[3]} y_j^{[2]}),$$
$$f_1(x) = f_2(x) = \frac{1}{1+\exp(-x)},$$
$$f_3(x) = x,$$

where F represents the future prediction network and P the past prediction network. During training, the future prediction network weights are updated according to the real-time recurrent learning algorithm. The function for minimizing the error is defined as:

$$e_f = \sum_t \sum_i \{[y_{out}(t)]_i - d_i^{[F]}(t)\}^2$$

where $d^{[F]}$ is simply the desired teacher signal.

The error and the weight updates for the past prediction network are computed in the same way. This bi-directional network as well as a uni-directional network have been applied to sunspots[1] data in [68] which is one of the most popular data sets often used for time series prediction tasks. The data set consisted of 280 years (A.D. 1700-1979) of normalised annual data.

[1]Sunspots are dark spots found on the surface of the sun, which are really areas of intense magnetic energy which tend to cool down the area residing within it, thus resulting in a darker appearance compared with the surrounding solar atmosphere.

The training set consisted of the first 100 years and the rest was used for testing purposes. In the future prediction network, $x(t)$ is fed as input to $y_{in}(t')$ and $x(t+a)$ is assigned a teacher signal at $y_{out}(t')$. In the past prediction network, $x(t+a)$ is fed as input to $z_{in}(t')$ and $x(t)$ is assigned a teacher signal at $z_{out}(t')$, where $x(t)$ is simply that sunspot input data at time t (in years), a is the prediction step and t' is the time step of the network.

Both networks consisted of 4 layers, i.e. 1 input node, 9 hidden layer 1 nodes , 9 hidden layer 2 nodes, and 1 output node. A learning rate of 0.005 was used to avoid instability. The results of the experiments conducted by [68] is as follows:

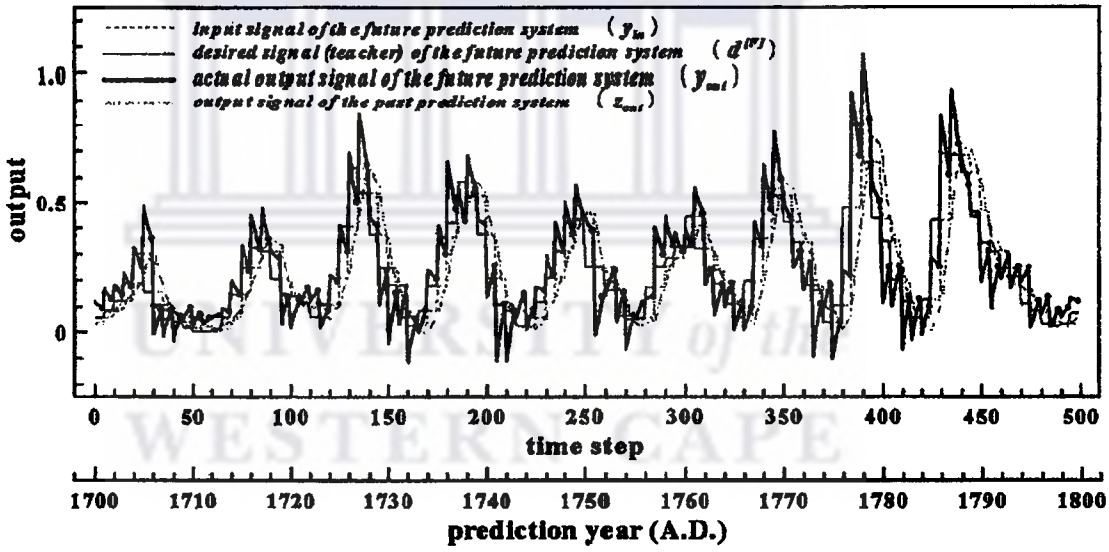

Figure 2.10: **The Future Prediction Network:** This figure illustrates the response of the input and output neurons.

Figure 2.10 and Figure 2.11 [68] shows the response of the input and output neurons for the bi-directional neural network.

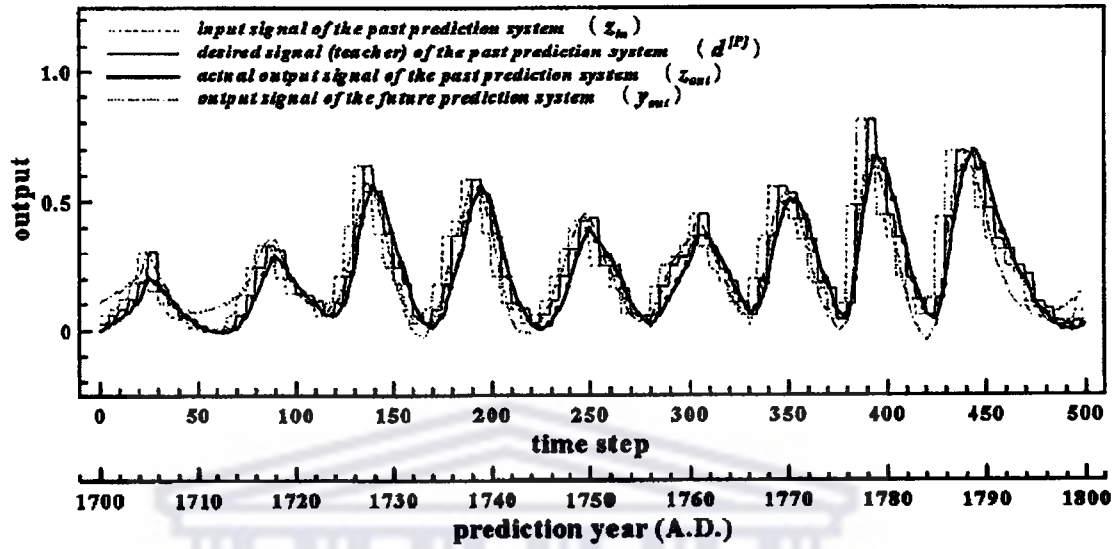

Figure 2.11: **The Past Prediction Network:** This figure illustrates the response of the input and output neurons.

[68] measured the models generalisation performance by an index known as the average relative variance (ARV). The ARV is defined as :

$$ARV = 1/\sigma^2 T \sum_{t=1}^{T} (DesiredOutput(t) - ActualOutput(t))^2$$

A perfect prediction has an ARV = 0, while an average one yields an ARV = 1. It can be seen in Figure 2.12 that the prediction quality of the bi-directional model is better than the standard uni-directional network. This future-past information integration helps the bi-directional model to predict future points more accurately (see Figure 2.11) and it results in better performance rates versus that of conventional uni-directional models [68].

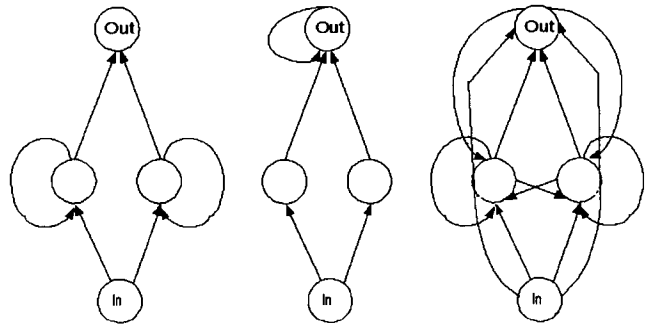

Figure 2.13: **Elman RNN, Jordan RNN and a Fully RNN:** This Figure illustrates RNNs without explicitly showing time delays.

the weight space that produce the minimum margin of error. Learning in recurrent networks is accomplished by finding the minimum of an error function E over all sequences which measures the difference between desired target outputs t_k and actual outputs a_k.

$$E(t) = 1/2 \sum (t_k(t) - a_k(t))^2$$

The error at a time t is calculated for a particular pattern; thus, E(t) represents the sum of all the errors over all the patterns residing in the dataset. The weights are then updated according to the following rule:

$$\Delta w = -\eta E(t)$$

where η represents the learning rate constant which determines the step size in the gradient descent search. With a small learning rate, a network will take a considerable period of time to converge to the desired solution if one exists. Too large a learning rate may result in divergence; the learning rate parameter is increased, the settling time of the network also increases which is the result of overshooting the

solution. After the error signals have been calculated, they are added together and contribute to one big change for each weight, this is known as batch learning. An alternative approach is on-line learning which allows the weights to be updated after each pattern is presented to the network.

RNNs have been applied to speech recognition with promising results. [48] applied RNNs to speech recognition whose dataset contain missing speech values. In experiments performed, incomplete speech values were replaced by estimated ones. An Elman RNN was used to estimate these missing values in the speech input vector. This is known as imputation[2].

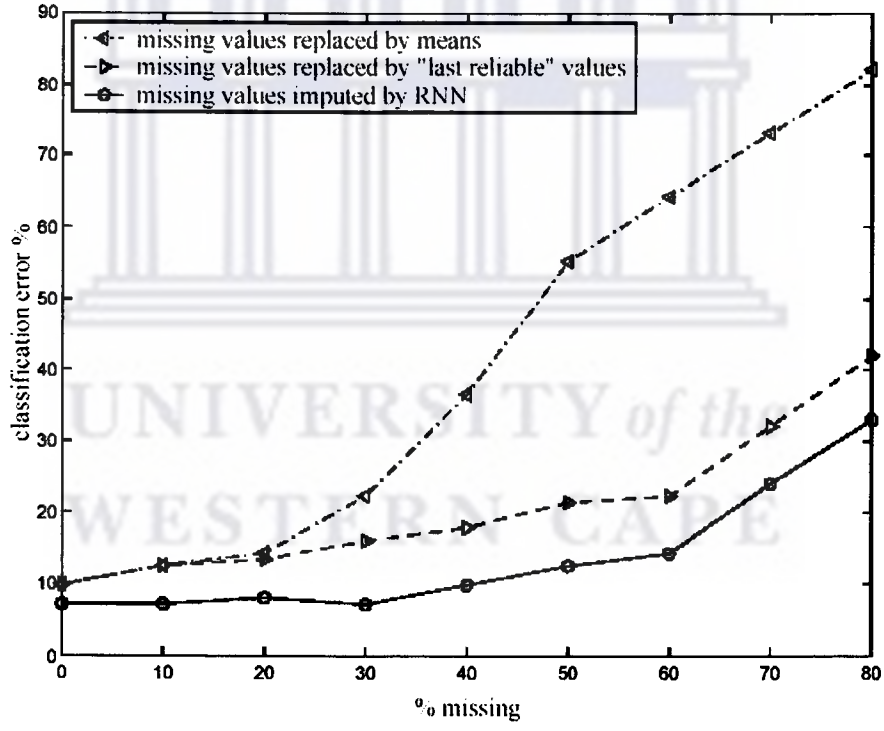

Figure 2.14: **The RNNs Classification Performance:** This figure illustrates that the RNN used for imputation and classification produced the best results.

[2]Imputation is a technique in which missing features are replaced by estimated values to allow the recognition process proceed in a normal way.

The same network was used to classify these speech signals. The dataset consisted of 30 male speakers, which were split into training and testing sets. This was obtained from the TIDIGIT database [48]. The network contained 20 features per time frame from a Hamming window which overlapped by 50%. The RNN consisted of 20 inputs, 65 hidden nodes, and 11 ouput units. In Figure 2.14 we can clearly see that the RNN used for imputation and classification produced the best results.

Common approaches based on gradient descent learning for recurrent networks include backpropagation-through-time (BPTT) [56] and real-time recurrent learning (RTRL) [67] which follows.

2.7.1 Backpropagation through time

This learning algorithm was first proposed by [56] and is based on the conventional error backpropagation algorithm. The backpropagation through time (BPTT) learning algorithm computes the error gradient on a RNN that is unfolded in time. This is accomplished by creating a copy of the network for each time step. The weights are then shared amongst these copies.
The total error of the unfolded network is defined by:

$$E = \sum_{t_1}^{t_n} \sum_j 1/2(e_i^t)^2$$

where e_i^t = error of node i at time t = $d_i^t - a_i^t$

e_i^t will = 0 if d_i^t is not specified.

The learning procedure computes weight updates as follows:

1. The forward pass for the given data is performed, and the error for each time step t is computed.

2. The error is backpropagated in order to calculate the local gradients for time step t.

$$\delta_i^t = -\delta E/\delta I_i^t = g'(I_i^t)e_i^t \; for \; t_1 = t_n$$

otherwise

$$\delta_i^t = g'(I_i^t)(e_i^t + \sum_j w_{ij}\delta_j^{t+1})$$

where $g(I_i^t)$ represents the squashing function.

3. The weight change is then computed:

$$\Delta w_{ij} = -\alpha \; \delta E/\delta w_{ij} = \alpha \; \delta \sum_{t_1}^{t_n-1} \delta_i^{t+1}\xi_j^t$$

where ξ_j^t represents the input to node j at time step t.

2.7.2 Real-time recurrent learning

Real-time recurrent learning (RTRL) [23, 26, 67] is another gradient descent learning approach for training RNNs. This learning algorithm calculates the derivatives of states and outputs with respect to all weights in the network. This means that the network is not unfolded in time.

We define input units as: $I = x_k(t)$, where $0 \leq k \leq m$, hidden or output units as: $U = y_k(t)$, where $0 \leq k \leq n$ and arbitrary units are indexed by: $z_k(t) = x_k(t) \; if \; k \in I \; or \; y_k(t) \; if \; k \in U$.

Let W represent the weight matrix which contains n rows and n+m columns and w_{ij} will represent a weight from unit i to unit j.

The network activation for a given unit is:

$$net_k(t) = \sum_{I \in U \cup J} w_{kl} z_i(t)$$

where t denotes a given time step.

The network activation is then passed through a squashing function:

$$y_k(t+1) = f_k(net_k(t))$$

A teacher signal may not be assigned for each input signal, i.e. a target is provided only for the last input in the sequence. An error defined over the output units needs to be time dependent. The reason for this is that if no target exists at a particular time step, an error produced at the output layer will be undefined or zero.
The output unit error is therefore defined as:

$$e_k(t) = d_k(t) - y_k(t) \text{ for } k \in T(t) \text{ or } 0 \text{ elsewhere}$$

where T(t) is simply the set of indices in U where there exists a teacher signal $d_k(t)$.
The cost or error function for a given time step is defined as:

$$E(t) = 1/2 \sum_{k \in U} e_k(t)^2$$

This error function needs to be minimized over all past steps of the network.

$$E_{total}(t_0, t_1) = \sum_{t_0+1}^{t} {}_1E(t)$$

The total error is now the sum of the current error and the error of the previous time steps. It then follows that E_{total} is the sum of the gradient for the preceding time steps and the current time step.

$$\nabla_w E_{total}(t_0, t+1) = \nabla_w E_{total}(t_0, t) + \nabla_w E(t+1)$$

where ∇_w is simply the gradient of w. For every sequence that is presented to the network we can compute the weight change Δ_w.

$$\Delta_{w_{ij}}(t) = -\mu \partial E(t) / \partial w_{ij}$$

So each weight within the network is adjusted by:

$$\sum_{t=t_0+1}^{t1} \Delta_{w_{ij}}(t)$$

2.8 The Vanishing Gradient Problem

According to [6], a task will exhibit long-term dependencies if the computation of a teacher signal at a given time step depends on the input signal presented at a much earlier instance. This means that current activation states within the network influence states in the distant future.

RNN's are appropriate tools for modelling short sequences; however, training is unlikely to converge when sequences have long-term dependencies [6]. [6] further notes that the vanishing gradient problem is really the main reason why gradient descent learning is not powerful enough to discover the temporal relationship that exists between current and past inputs.

[27] analysed the problem which these networks suffer from and explains it as follows: Using the conventional BPTT algorithm devised by [67], the premise is based on the fact that we initially have a fully connected RNN whose hidden and output unit indices range from 1 to n. We note that the local error flow for arbitrary unit u at a given instant will be backpropagated for q time steps to unit v. This then results in scaling[3] the error by the following component:

$$\partial\vartheta_v(t-q)/\partial\vartheta_u(t) = f'_v(net_v(t-1))w_{uv} \; for \; q=1$$

and

$$\partial\vartheta_u(t) = f'_v(net_v(l-q))\sum_{t=1}^{n} \partial\vartheta_l(l-q+1)/\partial\vartheta_u(t)w_{lv} \quad \text{for} \quad q>1.$$

[3]Adjusting the weights by a small amount at a time so as to reduce the error of the network.

Thus, with l_q = v and l_o = u, we have:

$$\partial\vartheta_v(t-q)/\partial\vartheta_u(t) = \sum_{t_1=1}^{n}\cdots\sum_{t_{q-1}=1}^{n}\prod_{m=1}^{q} f'_{l_m}(net_{lm}(t-m))w_{l_m l_{m-1}}$$

where $\prod_{m=1}^{q} f'_{l_m}(net_{lm}(t-m))w_{l_m l_{m-1}}$ results in the total error flowing back into time.

Thus, if the absolute value $\left|f'_{l_m}(net_{lm}(t-m))w_{l_m l_{m-1}}\right| > 1.0$, then the error increases without bound and conflicting signals arriving at unit u will result in network instability and oscillating weight magnitude.

Additionally, if the absolute value

$$\left|f'_{l_m}(net_{lm}(t-m))w_{l_m l_{m-1}}\right| < 1.0$$

the error tends to vanish.

Since f_{lm} represents a sigmoid function, the upper bound of f'_{lm} is 0.25. If $y^{l_{m-1}}$ is kept constant and $\neq 0$, then $\left|f'_{l_m}(net_{lm}w_{l_m l_{m-1}}\right|$ will have upper bound values where

$$w_{l_m l_{m-1}} = 1/y^{l_{m-1}}\coth(1/2net_{lm})$$

tends to zero for $\left|w_{l_m l_{m-1}}\right| \to \infty$ and is smaller than 1.0 for $\left|w_{l_m l_{m-1}}\right| < 4.0$. With a conventional sigmoid transfer function, the error flow will therefore diminish when the weights have absolute values below 4.0. This occurs mostly in the initial stages of the training phase.

[27] further notes that this local error flow results in the global error flow diminishing as well. It can be seen from the above analysis that the gradient descent learning technique is inadequate to deal with this problem. We will now review some solutions that various researchers' have devised, most of which is described in [27] and [28].

2.8.1 Time Constants

[45] proposed the idea of time constants. A time constant affects the changes in a networks unit activations. [59] proposed an alternative approach in which the activation of a feedback unit is updated by additions of a past activation value with the current network input.

2.8.2 Ring's Approach

[55] determined that when conflicting error signals enter a unit in a network, particular error signals promote in increasing the activity of the unit by adding a higher order unit which will influence the appropriate connections. The dilemma that was confronted by this approach is that bridging gaps of n time steps may involve the addition of n units.

2.8.3 Searching without Gradients

Network weights are randomly initialised until the resulting network is able to classify all the training patterns correctly. It has been shown by [29] that simple weight guessing solves several popular tasks faster than RNN learning algorithms. Other proposed methods include probabilistic target propagation and adaptive sequence chunkers, which can be found in [28].
We will now direct our focus on a gradient based-method which [27] devised.

2.8.4 Long-Short Term Memory

[28] developed this model after theoretically analysing the long-term dependency problem. This model is a gradient descent based method which truncates the networks gradient. This model is described in detail in Chapter 3.

2.9 Summary

Neural networks have been successfully applied to time series modelling, i.e. time series analysis and prediction. In this chapter, we have defined what time series modelling is. We then introduced the theory of neural networks and discussed how various architectures have been applied to numerous problem domains. We then extended the idea to networks that contain feedback connections as well as learning algorithms adapted to deal with it. Furthermore, we highlighted a short-coming of recurrent neural networks which was evident when dealing with extremely long time series. Finally, we concluded by defining the long-term dependency problem as well as a few solutions discussed in literature. This then motivates and explains why we will be using the modelling approach discussed in Chapter 3.

Chapter 3

Long Short-Term Memory Recurrent Neural Networks

The theory of long short-term memory (LSTM) recurrent networks was first introduced by [27] in the early 1990's. It is a gradient based method specifically used for modelling time series with long-term dependencies. Since the work described in this thesis makes use of LSTM RNNs, we have included a chapter on the topic.

3.1 Introduction

In the past, it used to be extremely difficult to train recurrent neural networks from examples because their parameters often settle in sub-optimal solutions which only take into account short-term dependencies and not long-term dependencies [16]. A novel RNN architecture and learning algorithm proposed in [27] overcomes the problem of learning long-term dependencies. The LSTM RNN architecture allows an error to be backpropagated through time further than any other method that exists, with the exception of the echo state approach to training RNNs [33].

LSTM enforces constant error flow over extremely long temporally extended patterns. We will first introduce traditional LSTM RNNs [27], followed by LSTM with forget gates then finally move on to LSTM RNNs with peephole connections, [18]. Forget gates allow the network to process continuous input signals that do not have clear beginning and end markers which indicate the start and end points of a pattern. Peephole connections allow the gates to inspect and use information in their decisions which is contained in the networks hidden layer. The result of this improves the networks performance. LSTM RNNs have better temporal generalization capabilities than time window based methods such as TDNNs. Temporal generalisation means that the temporal distance of unseen events in the training set are recognised and processed in the verification phase. Time window approaches suffer from the fact that relevant events must be learned for each position in the window that they occur.

3.2 Traditional LSTM RNNs

In [27], empirical evidence demonstrates that LSTM can learn temporal patterns with long-term dependencies which traditional RNNs struggle to learn. A LSTM RNN (Figure 3.1) consists of an input layer, a recurrent hidden layer and an output layer. It is similar in structure to a conventional fully RNN in Figure 2.13, with the exception that the hidden layer is replaced by a memory block layer. Each memory block is further subdivided into memory cells and gating units. The input to a cell is connected to every gating unit. The gating units control input and output access to every single cell residing within a memory block and bridge the connection to every cell within the hidden layer. When the gating units of a cell produce an activation close to zero, no erroneous input enters that cell and thus does not contribute to the state of the cell.

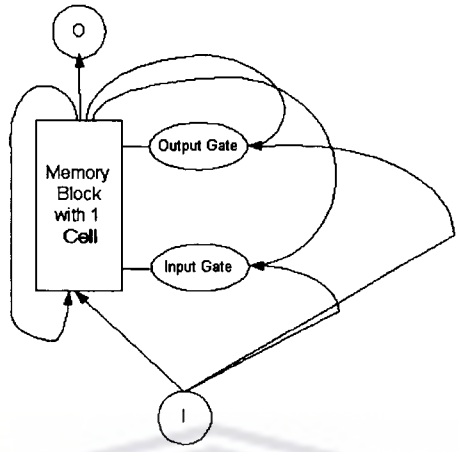

Figure 3.1: **A LSTM RNN**

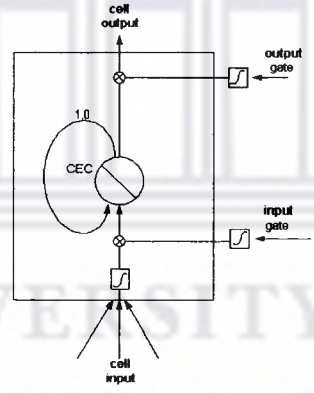

Figure 3.2: **A Memory Block with One Cell**

The memory cell in Figure 3.2 contains a unit known as the constant error carousel (CEC). The CEC is a linear function and has a weighted connection of 1.0 fed back to itself. The CEC produces an activation which reflects the state of the cell at a given instant. When the CEC does not receive any input, its activation state will be maintained over time.

3.2.1 Forward Propagation

In [27], only discrete-time steps are considered. A memory block is denoted by j and v denotes a memory cell within block j. net_c represents the input to a cell, net_{in} the input to the input gate and net_{out} the input to the output gate. w_{lm} are the weights on the connection from unit m to unit l.

The input gate activation:

The input to the cell is passed through a sigmoid function and is then multiplied by the activation of the input gate. Its main function is to govern the flow of the input layer activation to the cell.

$$net_{in_j}(n) = \sum_m w_{(in_j)m} y^m(n-1) \; ; \; y_j^{in}(n) = f_{in_j}(net_{in_j}(n)) \tag{3.1}$$

The output gate activation:

The output of the cell also passes through a sigmoid function and is then multiplied by the activation of the output gate. The output gate thus governs the flow of the output layer activation from the cell.

$$net_{out_j}(n) = \sum_m w_{(out_j)m} \; y^m(n-1) \; ; \; y_j^{out}(n) = f_{out_j}(net_{out_j}(n)) \tag{3.2}$$

The gating units make use of a sigmoid function with range [0,1].

$$f(x) = \tfrac{1}{1+e^{-x}} \tag{3.3}$$

The input to the cell is calculated as follows:

$$net_{c_j^v}(n) = \sum_m w_{c_j^v m} y^m(n-1) \tag{3.4}$$

This activation is scaled by a sigmoid function with range [-2,2].

$$g(x) = \tfrac{4}{1+e^{-x}} - 2 \tag{3.5}$$

The state of the memory cell at a given instant is calculated as follows:

$$s_{c_j^v}(0) = 0 \ ; \ s_{c_j^v}(n) = s_{c_j^v}(n-1) + y^{in_j}(n) g(net_{c_j^v}(n)) \tag{3.6}$$

for all n $\geq$ 0.

The output from the cell:

$$yc_j^v(n) = y^{out_j}(n) h(s_{c_j^v}(n)) \tag{3.7}$$

where h is a sigmoid function with range [-1,1]:

$$h(x) = \frac{2}{1+e^{-x}} - 1 \tag{3.8}$$

The output of the network:

$$net_k(n) = \sum w_{km} y^m(n-1); \ y^k(n) = f_k(net_k(t)) \tag{3.9}$$

where f_k represents the sigmoid function expressed in Equation 3.3.

3.3 LSTM RNNs with Forget Gates

LSTM RNNs fail to determine how to correctly handle long time series such as those that emanate from dynamical systems [17]. If any training pattern has no clear beginning and end, the internal state values of a given memory cell can grow ad infinitum. These beginning and end markers of a cell allow the cells state to be reset. [17] proposed a solution to this problem by modifying the traditional LSTM RNN to forget the unit activations which represents the short term memory within the network. Forget gates are introduced to circumvent this problem as shown in Figure 3.3. The forget gates task is to gradually reset a memory block. The traditional LSTM CEC self recurrent connection of 1.0 is replaced by a forget gate activation y^γ. The forget gate activation, y^γ is computed in the same way as in Equations 3.1 and 3.2 respectively.

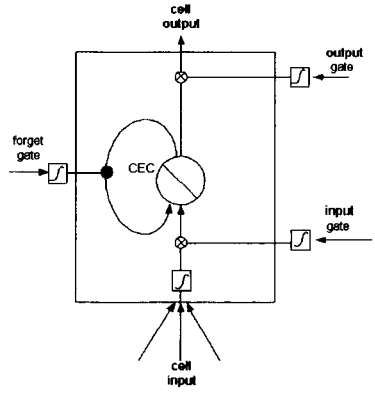

Figure 3.3: **A Memory Block with Forget Gates**

The forget gate activation is as follows:

$$net_{\gamma_j}(n) = \sum_m w_{\gamma_j m}\ y^m(n-1)\ ;\ y_j^{\gamma}(n) = f_{\gamma j}(net_{\gamma j}(n)) \tag{3.10}$$

where $f_{\gamma j}$ is a sigmoid function as in Equation 3.3.

The cell state now changes slightly, which now includes the forget gate activation:

$$s_{c_j^v}(0) = 0\ ;\ s_{c_j^v}(n) = y_j^{\gamma}(n) s_{c_j^v}(n-1) + y^{in_j}(n) g(net_{c_j^v}(n)) \tag{3.11}$$

3.4 LSTM RNNs WITH PEEPHOLE CONNECTIONS

The limitation of LSTM RNNs with forget gates is that there exists no explicit connection from the CEC which it controls. This results in essential information being lost since the forget gate can only directly observe the output of a cell. The networks performance is then adversely affected by not being able to inspect the CEC. Weighted peephole connections from the CEC to all the gates residing within the same memory block are therefore introduced. These peephole connections (Figure 3.4) shield the CEC from unwanted information during the forward and backward phases.

Peephole connections are able to utilize the cells contents when decisions need to be made.

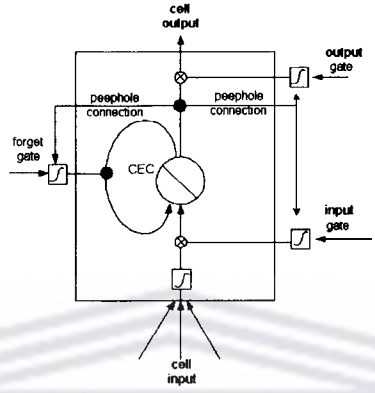

Figure 3.4: **A Memory Block with Peephole Connections**

The input and forget gate activation with peephole connections are as follows:

$$net_{in_j}(n) = \sum_m w_{in_{jm}} y^m(n-1) + \sum_{v=1}^{s_j} w_{in_j} c_j^v s_{c_j^v}(n-1) \ ,$$
$$\mathrm{y}^{in_j}(n) = f_{in_j}(net_{in_j}(n)) \tag{3.12}$$

$$net_{\gamma_j}(n) = \sum_m w_{\gamma_{jm}} y^m(n-1) + \sum_{v=1}^{s_j} w_{in_j} c_j^v s_{c_j^v}(n-1) \ ,$$
$$\mathrm{y}^{\gamma_j}(n) = f_{\gamma_j}(net_{\gamma_j}(n)) \tag{3.13}$$

The output gate activation with peephole connections:

$$net_{out_j}(n) = \sum_m w_{out_{jm}} y^m(n-1) + \sum_{v=1}^{s_j} w_{out_j} c_j^v s_{c_j^v}(n-1) \ ,$$
$$\mathrm{y}^{out_j}(n) = f_{out_j}(net_{out_j}(n)) \tag{3.14}$$

3.5 Learning in LSTM RNNs

Learning in LSTM RNNs is based on a fusion of truncated BPTT and a modified version of RTRL which was devised in [27]. This learning algorithm includes forget gates and peephole connections which has been extended by [18] to deal with

continuous time-series and precise timing.

During this training phase, the multiplicative gates learn to open and close, thus allowing the next set of input in enter the cell.

The sum of square errors is defined as follows:

$$E(n) = 1/2 \sum_k e_k(n)^2;\ e_k(n) := d^k(n) - y^k(n)$$

where d^k represents the desired teacher signal and y^k represents the actual output of the network.

The learning procedure is as follows:

Step1:

The errors are backpropagated by engaging the following procedure:

Compute the derivative of the output units:

$$\delta_k = f_k^{'}(net_k)e_k;$$

For all memory blocks j compute the derivatives of the output gates:

$$\delta_{out_j} = f_{out_j}^{'}(net_{outj})(\sum_{v=1}^{s_j} s_{c_j^v} \sum_k w_{kc}^v \delta_k)$$

For the v-th cells in the j-th block compute the cell state error:

$$e_{s_{c_j^v}=y^{out_j}(\sum_k w_{kc_j^v}\delta_k)}$$

Note that the Kronecker delta which is the discrete version of the delta function defined by:

$$\delta_{ij} = 1/2\pi i \oint_\gamma z^{i-j-1} dz$$

Step2:

Update the weights within the network:

Output unit weight updates:

$$\Delta w_{km} = \alpha \delta_k y^m$$

For all memory blocks j

Output gate weight updates:

$$\Delta w_{out,m} = \alpha \delta_{out} \hat{y}^m; \ \Delta w_{out,c_j^v} = \alpha \delta_{out} s_{c_j^v};$$

Input gate weight updates:

$$\Delta w_{in,m} = \alpha \sum_{v=1}^{s_j} e_{s_{c_j^v}} dS_{in,m}^{jv}$$

For all peephole connections, v'

$$\Delta w_{in,m} = \alpha \sum_{v=1}^{s_j} e_{s_{c_j^v}} dS_{in,c_j^{v'}}^{jv}$$

Forget gate weight updates:

$$\Delta w_{in,c_j^{v'}} = \alpha \sum_{v=1}^{s_j} e_{s_{cj}^v} dS_{\gamma,m}^{jv}$$

For all peephole connections, v'

$$\Delta w_{\gamma,c_j^v} = \alpha \sum_{v=1}^{s_j} e_{s_{c_j^v}} dS_{\gamma,c_j^{v'}}^{jv}$$

Update the cells for the v-th cells in j-th block:

$$\Delta w_{c_j^v,m} = \alpha e_{s_{c_j^v}} dS_{cm}^{jv}$$

where the partial derivative dS=0.

This learning algorithm is of course local in space and time. It has a computational complexity per time step and weights of $O(1)$, that means $O(n^2)$ where n represents the number of hidden units. This is in essence determined by the network topology.

3.6 Applications Of LSTM RNNS

We will now discuss a few applications of LSTM RNNs.

3.6.1 Blues Improvisation

[12] showed that LSTM RNNs are able to learn a form of blues music and are able to compose novel melodies with the same style. [12] notes that the compositions of music have a distinct global temporal structure in the form of nested periodicities. Music therefore has some notes that are more distinctive than others. A LSTM RNN is able to successfully learn to predict notes at time t+1 by making use of input data at times $\leq t$.

3.6.2 Automatic Speech Recognition

Since traditional RNNs suffer from the vanishing gradient problem (see Section 2.8), they are unable to learn correlations between inputs and errors which span long time intervals [13]. This then leads to a general failure to find long-term dependencies spanning several phones[1]. Traditional RNNs are not able to discover transition probabilities among sequences of words at a very slow timescale. The authors note that even at faster timescales, time warping and co-articulation effects tend to stretch phones which in turn blur their boundaries. LSTM seeks to address these problems that traditional RNNs face. A LSTM RNN maps every frame of an acoustic speech signal onto a set of fixed phone targets. The training involved using a collection of hand-labelled data. Two LSTM RNNs are used: the first network estimates the frame-level phone probability; the second network computes a mapping of the phone predictions into words, i.e. when the network is trained, it predicts sequences of words from sequences of phones which has been obtained

[1] A small unit of speech sound that assists to distinguish one word from another.

from the first network. Results indicate that LSTM RNNs performs well at the frame-level phone prediction.

3.6.3 Named Entity Recognition

Involves identifying atomic elements of information in text, such as names, locations and monetary values, etc. [21] trained a LSTM RNN on English and German atomic elements. A self-organising map for sequences is used to generate representations for the lexical items presented to the network [21]. The network is trained to output a vector which represents a particular tag, i.e. for the tag O a vector representation of 0100000 is produced. Promising results were yielded. The reader can consult [21] for the results.

3.7 Summary

Long Short-Term Memory RNNs are extensions of RNNs. They are able to overcome the long-term dependency problem which traditional RNNs suffer from. This chapter provided a brief overview on how LSTM RNNs work and briefly highlighted some tasks that LSTM has been applied to. This will be useful in understanding the next chapter which applies LSTM RNNs to signature verification.

Chapter 4

An Application: Automatic Signature Verification

4.1 Introduction

In this chapter, we will present a brief overview of the science of biometrics. We will then introduce signature verification to give a clear distinction between on- and off-line techniques and various modelling approaches that have applied to the former. We will then use signature verification time series as our testbed to indicate how well LSTM RNNs perform on these long time series.

4.2 The Science of Biometrics

Biometrics is the field of study that is primarily concerned with identifying the unique physical characteristics or behaviour of an individual in order to grant or deny them access to some type of computer resource or system. The science of biometrics can be divided into two classes, those that comprise physiological characteristics and those that comprise behavioural ones. The former include physical parts of the human body that can be used to uniquely identify an individual, whereas the

latter deals with the measurement of a behaviour performed by the central nervous system. A brief discussion of these characteristics will enable the reader to have clear perspective.

4.2.1 Physiological characteristics

Fingerprint Recognition

The scanning of a fingerprint is performed by detecting ridges and valleys found on the surface tip of a human finger to identify an individual. A 3-dimensional image is captured and then converted into a usable digitized greyscale image. Preprocessing such as binarization is performed on this greyscale image. A unique feature vector is then generated and fed into a classifier (i.e. neural network). Finally, the result of the classification process is the identity of the fingerprint.

Face Recognition

An individual's facial features are used for authenticating his/her identity. The majority of face recognition systems tend to use either eigenfaces or local feature analysis. In the eigenfaces technique, eigenvectors are computed for the covariance matrix of a training set of images which represent the features. Any image residing in the training set can therefore be reconstructed identically which in turn makes authentication possible.

Retinal Scanning

The patterns of veins that reside in the layer at the back of the eye can be used for authentication. A small laser beam scans the retina and an image is constructed from the descanned reflected light. The image for each individual will thus be unique. Authentication can therefore be carried out by comparing a test image against a constructed model of an image of an individual.

4.2.2 Behavioural characteristics

The behavioural characteristics are those aspects that describe a person's behaviour.

Speaker Verification

The acoustic speech signal contains unique information about the speaker, such as vocal tract length, vocal tract shape, vibration of vocal cords, etc. A model is then constructed to represent the speech signal.

Keystroke Dynamics

This behavioural biometric is also known as typing rhythms. This method analyses the way a user types at a terminal by monitoring keyboard input at a rate of 1,000Hz. No enrolment and verification phase is necessary; since the individual's keystrokes are monitored, so there is no need for the individual to detract from regular work flow.

4.3 Signature Verification as a Behavioural Biometric Technique

The science of Handwritten Signature Verification (HSV) falls within the category of behavioural biometrics. These behavioural characteristics are dictated by a combination of psychological and physical actions that are inherently unique to each individual.

HSV is defined as the classification process which ultimately strives to learn the manner in which an individual makes use of the muscular memory of their hands, fingers and wrist to reproduce a signature. The two core components of HSV are the quality of the signature exemplars and the process of verifying them.

4.3.1 Signatures

[31] states that signatures have at least three attributes which include form, movement and variation; of which movement is probably the most important. The reason is that movement is produced by muscles of the fingers, hand and wrist and these muscles are controlled by nerve impulses which are in turn controlled by the brain without any particular attention to detail [31].

[20] shows that a high degree of variability exists across a large population of individual signers; in some cases, the appearance of a signature might be extremely complex while others appear to be simple.

Handwritten signatures can be identified from two groups of distinct features: static features are concerned with the overall shape, e.g. length to width ratio of the signature; signature dynamics represents the manner in which a signature is produced during the acquisition process, e.g. pen tip pressure, positional information etc. It is important to highlight that no two genuine signatures of an individual are identical as they tend to vary in their statics and dynamics [20]. As a result, the process of verifying human signatures is a challenging pattern recognition problem. Signatures are a unique way of accurately identifying individuals. A good model will thus yield a robust, accurate and non-intrusive method to the biometrics toolbox.

4.3.2 The Process of Verification

The verification of signatures can either be done manually or computationally. The former requires a human to physically analyse the signature. The dilemma which we are confronted with in a commercial environment is that there is a tendency to conduct minimal checking when a purchase is made via a credit card: either no checking is done or a paper copy of your signature is compared very briefly to the one residing on the card without much thought going into it. As a result, credit card

fraud becomes that much simpler to accomplish.

Computational techniques are seen to be more complicated but provide a far greater level of security. These techniques fall into one of two categories, namely that of off-line and on-line signature verification.

To measure the performance level of signature verification modelling technique, results are traditionally reported in terms of the false rejection rate (FRR) or type I error and the false acceptance rate (FAR) or type II error, [50]. We define these terms to enable the reader to interpret the results of the on-line signature verification techniques presented in Section 4.4.2. In our experiments we report the results in terms of the true acceptance rate (TAR) versus the FAR (see Sections 4.5.5 and 4.6.2).

The FRR is the probability that a genuine signature will be incorrectly classified as a forgery by the model. The FAR is the probability that a fraudulent signature will be incorrectly classified as a genuine signature by the given model. At the point where the FRR and FAR intersect we have the Equal Error Rate (EER). The lowest EER is therefore achieved by a adjusting the FRR and FAR respectively, and by reducing the FRR by adjusting the decision threshold, a higher FAR is obtained.

4.4 Signature modelling Techniques

An important decision in HSV design is the choice of modelling technique. This modelling segment is an important component of the HSV technique, since it ultimately distinguishes genuine signatures from those that are fraudulent. In this section, we will review a few modelling approaches that have been applied to on-line HSV.

4.4.1 Off-line Signature Verification

Off-line HSV [2, 4, 22, 25, 57, 58, 71] involves optically scanning a signature, using a scanner, that was written on a piece of paper. The signature acquisition phase therefore leads to binary images of signature that have been captured. These binary images are then normalised to discard any irrelevant information, e.g. noise produced by the scanning process. Most methods then make use of a neural network classifier whose input layer size is dependent on the size of the signature image. For a further in-depth discussion on off-line techniques the references in this section may be consulted.

4.4.2 On-line Signature Verification

On-line HSV systems capture an individual's signature via a graphical tablet and stylus or even an instrumented pen, which digitizes and stores the signature on computer. The time-varying signals that can be sampled from the signature include position, pen tip pressure and orientation in space. The digitized signature can then be represented as a time series and can thus be analysed using well founded modelling techniques. We will briefly discuss a few of these modelling techniques.

Dynamic Time Warping

Dynamic Time Warping (DTW) is a well-known technique which is ideally suited for quantifying the similarities of handwritten signatures, [43, 46]. DTW takes two sequences and aligns them by calculating the distance between them.

There exists various approaches which determine the similarity between time series. [10] states that the Euclidean distance measure may not always produce a correct measure of similarity between two patterns that are very easily distorted along the time axis. It is for that reason that the DTW approach is adopted.

[10] further notes that DTW allows an elastic shifting of the x-axis so as to detect the same shapes. The authors mention that a pitfall of the DTW approach is that performance seems to be limited on large datasets. Apart from HSV, DTW has been successfully applied in the fields of chemical engineering, [51] and ECG pattern matching [62].

The DTW algorithm [10], is defined as follows:
Given two time series A and B of lengths n and m we have, $A = a_1, ..., a_n$ and $B = b_1, ..., b_m$. In order to align these two patterns an n-by-m matrix has to be constructed. The (ith, jth) element of this matrix contains the distance $d(a_i, b_j)$ between two points a_i and b_j. We then have a warping path $W = w_1, ..., w_k$, which defines a mapping between patterns A and B. One is then required to find the path that minimises the warping cost.

$$DTW(A, B) = min\left\{1/k\sqrt{\Sigma_{k=1}^{K} W_k}\right\}$$

were k is used to compensate for warping paths of varying length. A further explanation of the DTW algorithm can be found in [10].
Figure 4.1 illustrates the result of applying DTW to a signal. In HSV, DTW is mainly being used to compare the similarity of signatures, [46]. In [46], no Normalisation is performed on the signature set since the assumption is made that the signers reproduce their signatures consistently. DTW essentially aligns various signature features such as velocities and accelerations. In [46], an EER of 2.6% is achieved. For an in depth overview on how DTW is applied to two-dimensional curves the reader can consult [46].

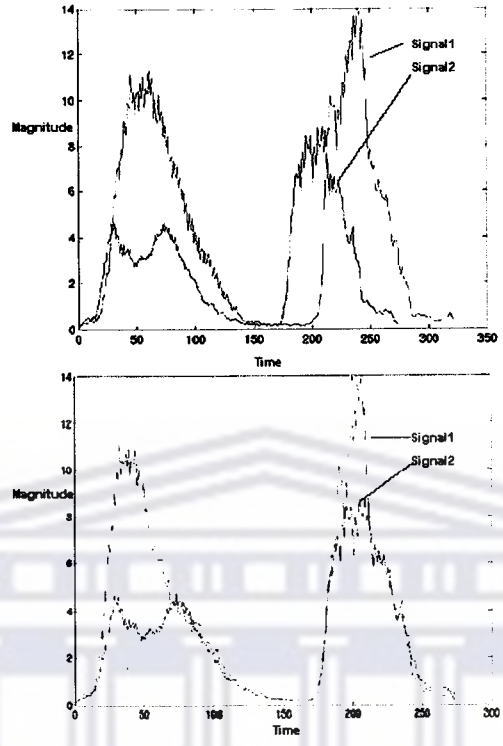

Figure 4.1: **Dynamic Time Warping:** This figure illustrates how two signals are aligned.

Hidden Markov Models

Hidden Markov models (HMMs) have been used quite extensively in the area of speech recognition [32] and bioinformatics [36] with a great deal of success. They have also been applied to HSV and have produced notable results [37, 54, 66]. It is important to note that the vanishing gradient problem is also present for HMMs as has been shown in [8]. A Markov model is defined as a stochastic process over a set of states, $S_1, ..., S_n$ with transitions occurring between these states. A transition matrix defines what the probability of the next state will be for the time step that

follows, which of course depends on the current state.

So we have:

$$\mathrm{A} = \begin{bmatrix} p_{11} & \cdots & p_{1n} \\ . & \cdots & . \\ p_{n1} & \cdots & p_{nn} \end{bmatrix}$$

where A represents the transition matrix with p_{lm} indicating the probability of making a transition from state l to state m. Since this process is stochastic, any given probability p_{lm} must be greater than 0 as well as the sum of each row of probabilities equal to 1.

A zeroth order Markov Model is similar to a multinomial probability distribution and contains no context.

$$P(h_t = S_l) = P(h_{t'} = S_l)$$

where S are simply the model states.

A first order Markov Model on the other hand contains a context size of 1 and is represented by a matrix of probabilities:

$$P(h_t = S_l) = P(h_{t-1} = S_l), \text{ for } 1 \leq l \leq N, mn$$

We finally have a v-th order Markov Model, where the length of the context depends on the state transition probabilities. It follows from this that HMMs are simply Markov chains in which we have hidden states. A compact way of defining it is as follows:

$$\text{So the HMM } \lambda = (S, \prod, A, B)$$

where S: hidden states, $\prod$: starting state probabilities, A: transition matrix probabilities and B: a probability density function which may be discrete or continuous.

An example of applying a HMM to a trivial problem is to compute the probability

of a coin toss either being heads or tails for a given time period, since it impossible to determine the result exactly.

Training involves determining the likely hidden state sequence that can best explain a sequence that was observed. This learning approach is known as the Viterbi algorithm, whose roots stem from dynamic programming.

[37] applied hidden Markov models to an on-line signature verification. The system created a universal prototype of signatures from a database. During the features extraction phase, 21 global features and only a single local feature were considered. Global features referred to the pen-down segments of the signer or the signature as a whole. Local features referred to the equally spaced sub-segments or every signatures sample point. The global features included the total signature time, time of pen down, root mean square speed, average horizontal speed, integrated absolute centripetal acceleration, length-to width ratio, horizontal span ratio, 8 directional histograms, 4 directional change histograms, x,y speed correlation and the first moment of the moment generating function. The local feature only included the slope of the tangent at each point. Each new signature type was assigned a distance from the prototype along a few measurements. Model parameters were estimated from a list of valid signatures. To model a signature the system made use of the handwriting tangent and its derivative as a vector. The model was trained using the Viterbi algorithm. A EER of 2.5% was achieved due to a 1% FRR and a 5% FAR.

The main advantage of HMMs over other methods is that negative training samples are not needed for a HSV system to function efficiently [66].

Artificial Neural Networks

Neural Networks are robust classifiers and are used today in many application domains such as medical diagnosis, seismic event prediction and stock market prediction. The reader can consult [23] for a concise introduction to neural networks.

In [42], a standard graphics tablet was used along with a pressure sensitive stylus. Signatures were collected from a single subject, it consisted of 1000 genuine signatures and 450 skilled forgeries which were reproduced by 18 trained forgers. [42] notes that very little preprocessing was carried out on the data. This was namely linear time normalisation and signal time resampling. The main aim of this phase was to transform a signature into a sequence which can be fed into a neural network as input. Linear time normalisation scales two signatures in terms of its timescale, i.e. two signatures with differing time frames will be scaled in such a way that the time it takes to reproduce it is equivalent. Signal time resampling is carried out by linearly interpolating a signatures spatial co-ordinates (x(t),y(t)).

The absolute velocity $|v(n)|$ is then computed as the only feature. This is calculated as follows from each component of the signature time series:

$$|v(n)| = \frac{\sqrt{\Delta x(n)^2 + \Delta y(n)^2}}{\Delta t(n)}$$

where $\Delta x(n) = x(n+1) - x(n)$, $\Delta y(n) = y(n+1) - y(n)$ and $\Delta t(n) = t(n+1) - t(n)$. A feature vector of a signature was therefore represented as a sequence of absolute velocity values.

Three neural network architectures were investigated, namely a Bayes feedforward neural network (BFNN), a TDNN and a input-oriented neural network (IONN). The BFNN are similar to conventional feedforward networks with the exception that during training, the backpropagation algorithm computes the minimum mean squared error approximation to the Bayes discriminate function. [42] mentions that this minimizes the networks misclassification error probability. The input layer consisted of 100 input neurons into which the entire feature vector was fed. The hidden and output layers contained 5 and 1 neurons respectively. The TDNNs (see section 2.4) input was buffered using a feature vector size of 20. It consisted of two hidden layers; hidden layer one contained three neurons and hidden layer two contained

7 neurons. The output layer contained a single neuron. IONNs are also similar to conventional FFNNs in that the following constraints are placed on the weighted connections to and from a node: $w_{i,j}^{(k)}$ connecting neuron i of layer k-1 and neuron j of layer k is constrained to be the same for a fixed i and any j, i.e. $w_{i,j}^{(k)} = w_{i,k}^{(k)}$ for any j and k, [42]. After training and testing on set of 100 genuine signatures and 18 forgeries the equal misclassification error rates for the BFNN, IONN and the TDDN were 2,67%, 3.82% and 6.39% respectively.

On-line systems yield the best results since dynamic features increase the discriminative capability of the classifier simply on the basis that more relevant information is available from the signing process. The problem inherent in most off-line systems is that one is able to trace a signature with the result that it will be accepted. To our knowledge no published research application has been conducted based on applying RNNs to HSV. RNNs, however, been applied to handwriting recognition [41], which is not a biometric.

4.5 On-line Handwritten Signature modelling

It is within the process of HSV that we extract the attributes associated with the behavioural characteristics of an individual and model it, using a classifier which conclusively determines the identity of the subject in question. This modelling process can hence be divided into five major phases: data acquisition, preprocessing, feature extraction, classification, and performance evaluation. The task of modelling a handwritten signature involves the processes illustrated in Figure 4.1.

4.5.1 The Signature Data Acquisition Phase

A graphics tablet and stylus or digitizer is generally used to capture a handwritten signature as a 5-dimensional vector (see Figure 4.2). This vector consists of a pair x

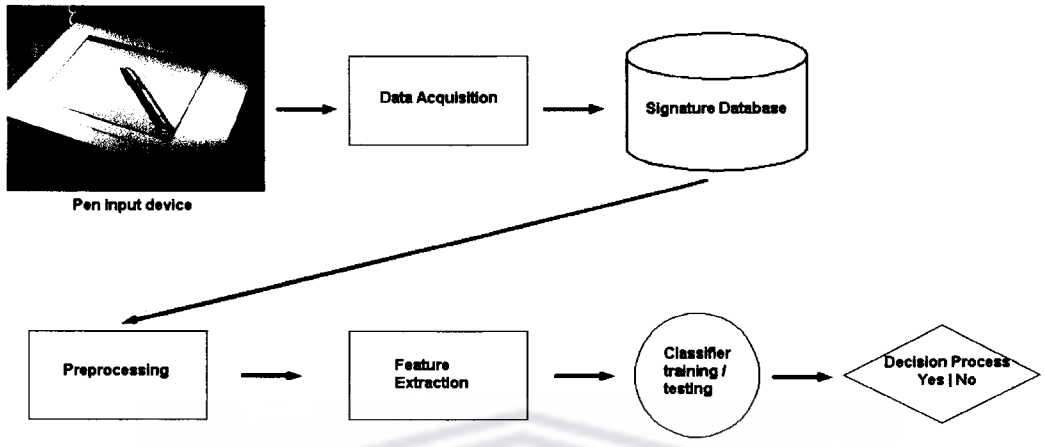

Figure 4.2: **The On-line Signature Verification Modelling Process:** a signature is modelled using the process depicted in this figure

and y co-ordinates, axial pen tip pressure and polar angles of the stylus. The x and y co-ordinates represent the positional information of the signature. The axial pen tip pressure indicates the amount of force that the signer is applying to the tablet surface with the stylus. While signing the polar angles θ_x and θ_y, simply indicate the position at which the stylus is being held at a given instant in space (see Figure 4.3). A digitizer samples a signature at a rate of approximately 100Hz to 200Hz.

4.5.2 The Signature Preprocessing Phase

When an individual signs his/her signature on a graphics tablet, nothing prohibits them from signing in a certain position or a particular size, i.e. a signature may be slanted, small or large etc. Even if a common baseline is given for the signer to start from, chances are that this might not be accurate or consistent. Normalisation of a signature with respect to size and orientation is therefore carried out. In essence,

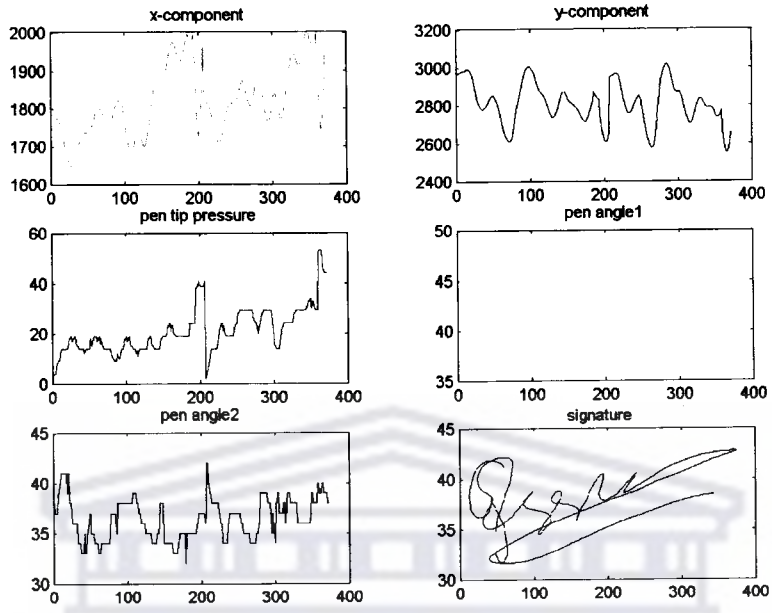

Figure 4.3: **Digitised Signature Signals:** A signature is reduced into these components.

preprocessing seeks to take the signature signals and convert them into a derivable form. Normalisation as defined in [66] seeks to make a signature invariant to rotation, scale and position.

Low-Pass Filtering

During the signature acquisition phase, the analog output from the tablet is converted into binary values. During this process, a sampling phenomenon known as aliasing may occur that results in errors and also reduces the accuracy of the collected data. Aliasing is the process when a high frequency signal assumes the identity

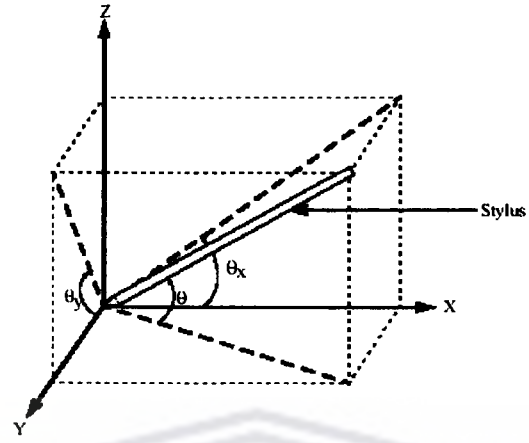

Figure 4.4: **Polar Angles:** The spatial position of the stylus while signing

of a lower frequency signal. This normally occurs when an input signals frequency components is equal to or greater than half of the given sampling rate. One finds aliased signals at higher frequencies, but they are converted by the sampling process to a false frequency below half the sampling rate, e.g. given a sampling rate of 200Hz, a signal at 160Hz will be aliased 40Hz which is really the false lower frequency. A Fast Fourier Transform is applied to the data to minimise the likelihood of aliasing. Figure 4.4, illustrates the result low-pass filtering using filter widths of 10,30, and 50. We apply this to user 18 in our signature database as an example.

Rotational Invariance

This ensures that a signature will not deviate from a given baseline. A signature is rotated by an angle θ, Figure 4.3, which aligns it through a centroid. According to [66], an angle θ of corrective rotation about the centroid of (x, y) co-ordinate pairs is computed. The signature is then normalised by rotating the signature by a certain

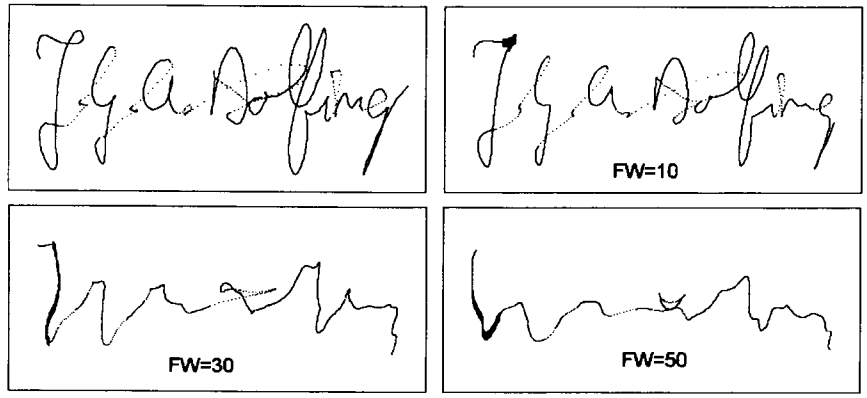

Figure 4.5: **Low-Pass Filtering:** A Fast Fourier transform is applied to the data to minimise the likelihood of aliasing.

angle θ. For a in-depth algorithm on how this is computed, the reader may consult [66].

Translation Invariance

This procedure compensates for the fact that a signer may not necessarily always sign on the exact place when given a common baseline. It can be carried out by taking the smallest x and y component vectors and subtracting them from the rest of the positional components.

Scaling Invariance

Generally, signature exemplars may be of different sizes, since it is well known that an individual cannot reproduce the same signature more than once, [20]. This is achieved by transforming a signal into a particular aspect ratio.

4.5.3 Feature Extraction

A feature is defined by [22] as a measurement derived from a given input signal that is to be subjected to classification. The correct number of features should also be selected without discarding vital information. Various features then combine to form a feature vector. The entire feature space consists of n feature vectors.

4.5.4 Classifier Training and Testing

The training procedure involves feeding a pattern through a classifier. The classifier learns which class the given pattern belongs to, given sufficient positive and negative examples.

During classification an input signal is preselected into a one of two classes, which is based on the analysis of significant features. It then constructs a suitable decision boundary by partitioning a feature space into classes. If a given input signal activation falls within a particular class, then it is assigned that particular class label.

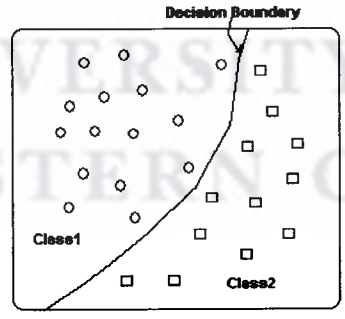

Figure 4.6: **Two Class Classification:** This figure illustrates the decision space of a two class problem.

HSV is in essence a two-class classification problem. An individual's signature is either genuine or it is a fake produced by a fraudster.

THRESHOLD DETERMINATION

Once the classifier produces an activation at the output layer, we need to decide on the authenticity of signature. Class A is assigned a value of 1 and class B a value of 0. Once the behaviour of the activations are known, a single global or local decision threshold can be chosen based on what activation is produced by the suspect signature.

4.5.5 PERFORMANCE EVALUATION

The objective of learning classifications from sample data is to classify and successfully predict on unseen data [2]. In classification problems a particular classification rule, e.g. if x is then greater than 0.5 then x belongs to class 1, will always lead to some sort of misclassification or error, i.e. positive and negative examples are presented to a model and classifies it incorrectly. [2] notes that this is the result of having features common to two or more classes.

The true acceptance rate (TAR) is the probability of detecting a genuine signature and the false acceptance rate (FAR) is the probability that a forgery will be misclassified as a genuine signature (as mentioned in Section 4.3.2).

The performance of a learning system is evaluated using a receiver operating characteristic (ROC) curve. The ROC curve graphically represents the trade off between the TAR and FAR for every possible threshold.

4.6 LSTM RNNs

This section describes the results of our experiments. We present the result of our experiments in the form of RMSE learning curves and ROC curves which plot the TAR versus the FAR.

4.6.1 The Signature Database

It is important to note that a good signature database can represent a real-world scenario and plays an important role in the development of a HSV schema.
Our training and testing data originate from the same database that was used in [11]. The database contained signatures of 51 individuals, 45 male and 6 female which was collected over a long period of time. The total genuine signature count was 1530. The number of amateur, home-improved and over-the-shoulder fraudulent signatures totaled 3000. Additionally, there were also 240 forensic forgeries; each individual contributed 30 genuine, 30 fraudulent home-improved and over-the-shoulder signatures; for 6 of the individual signatures four forensic document examiners contributed 60 professional forgeries each.

Figure 4.7: **Signature Complexities:** This figure illustrates the signatures that we will use in our experiments for the three types of complexity classes.

These individuals (see Figure 4.7) were examples of signatures of varying complexity, i.e. 'easy', 'moderately easy' and 'difficult' to forge [11]. Signature complexity is based on the visual inspection of the signature in question, i.e. is a signature complex in its visual appearance.

Figure 4.8: **Sample Signatures from Database:** The first column contains genuine signatures, the second column over-the-shoulder forgeries, and the third column home-improved forgeries.

4.6.2 Signature Modelling Experiments

For each experiment, we trained a fully connected LSTM RNN architecture in which every node is connected to every other node. Forget gates (Section 3.3) and peephole connections (Section 3.4) were added to each memory block. The network architectures used in our experiments were determined by extensive testing, i.e. a given architecture was selected, training and testing was carried out, finally the process was repeated until the desired network architecture was obtained. We selected the networks that converged to the desired solution in the least amount of time. We found training a large network on the entire dataset to be computionally expensive, i.e. to reduce the RMSE to 0.1 it took approximately two weeks on an amd athlon barton 2167mhz processor. Ten sets of experiments were conducted on signature features and signature complexity. The results of each experiment fall within a 95%

confidence interval; thus we can safely conclude that they are a consistent representation of the other sets and thus these results are statistically significant.

Our dataset is randomly divided into training and testing sets and are normalised according to the standard normal distribution, i.e. z-distribution, with a mean of 0 and a standard deviation of 1. In the experiments that involve smaller sample sizes, this transformation is applied to the data which then forms a new statistical distribution which is representative of a larger sample set. The magnitude of the initial random weights was chosen to be 0.1. In all our experiments the following is discussed:

- Network Input
- Network Architecture
- Learning

We derived two additional signature features namely, velocity and path tangent. This is described in the next section.

Feature Derivation

From the original sampled signature signals in Figure 4.2, we derived two more to be used in our experiments.

Velocity

[22] notes that an experienced forger might succeed in duplicating the static aspect of a signature but will find it difficult to do the same for the dynamical information. The velocity at a point t can be calculated from the positional components as:

$$V(t) = \sqrt{(V_x(t)^2 + V_y(t)^2)}$$

where t=0,..,n.

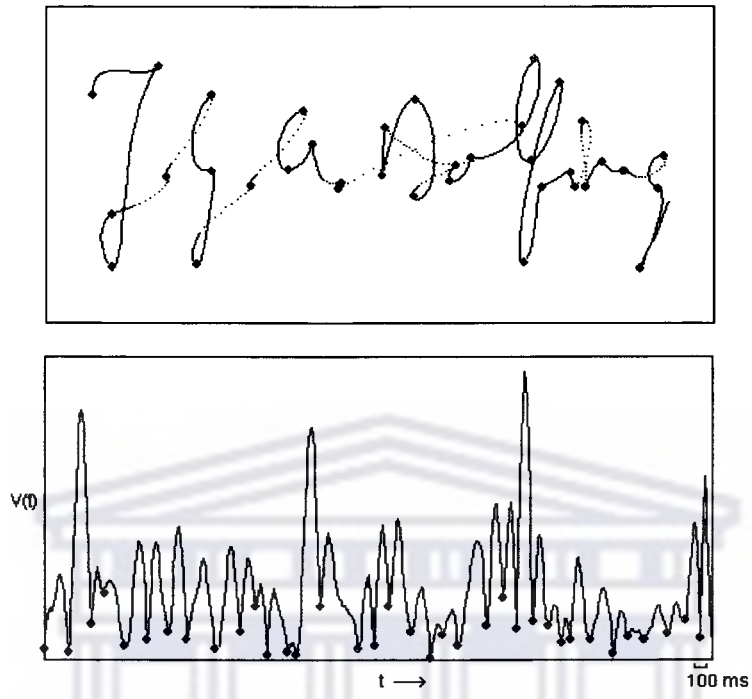

Figure 4.9: **Velocity Signal with Minima:** The figure illustrates the velocity profile and its minima of a given signer which is sampled at a rate of 100Hz.

Path Tangent

This is simply the directional component of the velocity signal. The path tangent is calculated by:

$$T_\theta = tan^{-1} V_x / V_y$$

Modelling the entire Signature Database using a Single LSTM RNN

This section covers the design and implementation of the signature modelling experiment that was conducted to determine if a single LSTM RNN is capable of modelling signatures for all users that exists in the signature database. This is accomplished by training the network to a desired error rate. The entire signature database, i.e. 4530

signatures, is used in this experiment. This dataset was split into 70% for training and the remaining 30% unseen samples for testing.

Numerous experiments were conducted to determine what network architecture is sufficient to accomplish the task given that 5 sampled signature components, as discussed in Section 4.5.1, are used as input. A LSTM RNN with an input layer of 5 neurons into which the normalised signature data is input. The hidden layer contains 12 memory blocks which consists of 4 memory cells each. The output layer contained a single neuron whose activation is squashed through a 8020 sigmoid function in the range of [0.2,0.8]. The network contained 4482 weights and 55 unit activations.

The learning procedure is on-line in which the weights within the network are updated after the presentation of each pattern. A small learning rate of 0.0001 and a high momentum rate of 0.9 was chosen to minimize the instability of the network. Figure 4.10 illustrates the learning curve of the network which was terminated once an average testing RMSE of 0.1 was attained. A class label is only assigned at the end of a pattern and this indicates what class the pattern belongs to. Hence, an error is only generated at the end of each pattern. If the error is generated at each time step, i.e. each point in the pattern has a corresponding class label, a forger would only need to forge a single point in the pattern in order to forge the signature.

A testing RMSE of 0.082 and a training RMSE of 0.05 was obtained after 1480 epochs. These results are based over an average of 10 runs and are a clear indication that LSTM can model these signatures. Also, based only on the testing RMSE value we can conclude that the network is able to generalize fairly well on the testing set.

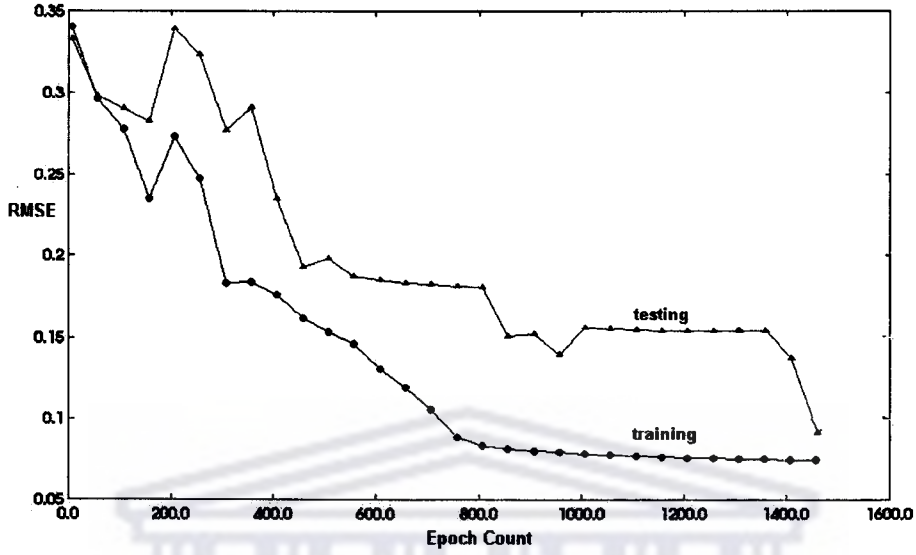

Figure 4.10: **Average Training and Testing RMSE:** The figure illustrates the RMSE which quantifies both the differences between the actual and the predicted output of the network by computing the average errors across 2 or more time series.

Signature Complexity Experiments

This section covers the design and implementation of the signature complexity experiments that were conducted to determine what effect the complexity of a signature has on the performance level of the network when considering casual, skilled and forensic forgeries. It must be stressed that no comparison will be made against other proposed methods since a common signature database is not available.

Three different experiments were conducted for the three varying complexities of signatures, i.e. casual, skilled, and forensic, using two users from each category. Three LSTM RNNs were trained per signature complexity level. As mentioned in Section 4.5.1, all signatures consist of various features of which we have used 5, namely x and y pen tip co-ordinates, pen-tip pressure and polar angles of the stylus. The networks used for the experiments are LSTM RNNs with an input layer of 5 neurons into

which the normalised signature data is input. The hidden layer contains 8 memory blocks that consists of 4 memory cells each. The output layer for the networks that were trained on easy and moderately easy complexity signatures contained a single neuron whose activation is squashed through a sigmoid function in the range of [0,1]. The network that was trained on the difficult complexity signature set consisted of a 8020 sigmoid function, which has a range of [0.2,0.8]. The value 0.8 will represent 1 and the value 0.2 will represent 0. The network contained 2262 weights and 39 unit activations. The learning procedure is on-line in which the weights within the network are updated after the presentation of each pattern. A small learning rate of 0.0001 and a high momentum rate of 0.9 was chosen to minimize the instability of the network.

Statistics	**Easy**	**Moderate**	**Difficult**
Average Training RMSE	0.070	0.1	0.072
Average Testing RMSE	0.095	0.099	0.093
Epoch Count	2003	2602	1153
Output unit activation function	sig[0,1]	sig[0,1]	sig[0.2,0.8]

Table 4.1: **LSTM RNN Learning Statistics**

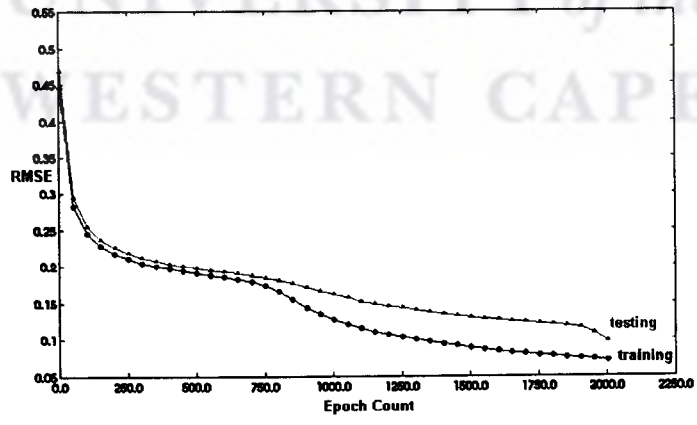

Figure 4.11: **RMSE for Easy Complexity Level**

Figures 4.11, 4.12 and 4.13 show the learning curves of the networks for the 3 complexity levels and Table 4.1 indicates the precise values. Learning is terminated once the average RMSE reaches 0.1.

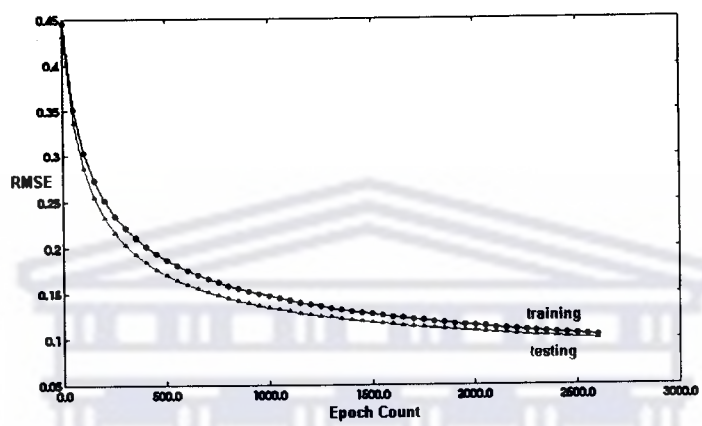

Figure 4.12: **RMSE for Moderately Easy Complexity Level**

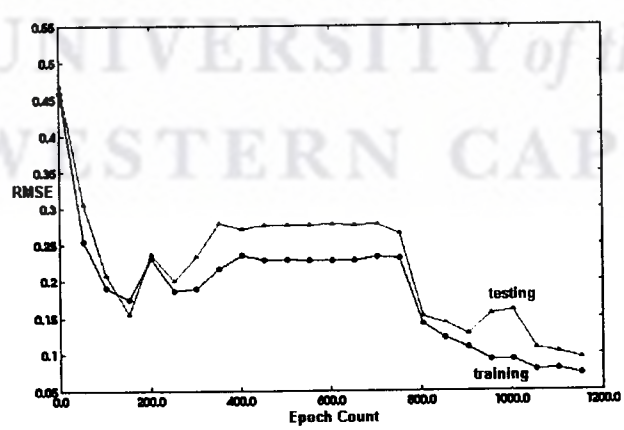

Figure 4.13: **RMSE for Difficult Complexity Level**

Table 4.2 represents the results obtained from the analysis of the testing when evaluating the classifiers performance using ROC curves. For our evaluation the ROC curves utilize TAR and FAR.

Forgery Type	**Easy**		**Moderately**		**Difficult**	
	TAR	FAR	TAR	FAR	TAR	FAR
Casual	95%	4.5%	91.7%	5.3%	92.3%	0%
Skilled	91.7%	5.3%	100%	5.6%	100%	0%
Forensic	92.3%	6.7%	100%	5.6%	100%	0%

Table 4.2: **Signature Complexity Performance Summary:** This table illustrates the performance rate of the network on signatures of varying complexity.

Forgery Type	**Easy**	**Moderately**	**Difficult**
	C.I	C.I	C.I
Casual	0.990 to 1	0.838 to 1	0.948 to 1
Skilled	0.838 to 1	0.920 to 1	1 to 1
Forensic	0.948 to 1	1 to 1	1 to 1

Table 4.3: **Confidence Intervals for Signature Complexity Experiments:** This table illustrates the corresponding confidence intervals (C.I) for experiments in Table 4.2.

The results in Table 4.2 suggest that complex signatures are more difficult to forge. Even forensic experts had difficulties in forging the dynamic aspects of signatures, eg. velocity of the signing process. This leads to a better discrimination performance for more complex signatures if we use dynamic features for classification. This can be seen from the decreasing FAR for the three categories of signatures across the three complexity levels for those signatures. It should be noted that for casual and skilled signatures, the FAR values from easy to moderately easy signatures actually increase, but by such a small margin as to be insignificant, 0.9 and 0.3 respectively. The forensic signatures performance provide the best example in that FAR values decrease across all complexity categories for the signatures. This

can be explained by the fact that forensic forgeries are very accurate in terms of the shape of a genuine signature, i.e. x and y co-ordinates, because of slow careful forging, but are in turn extremely poor with respect to the dynamic information such as pen-tip pressure and pen-tip angles. As a result, the forensic examiner finds it easier to forge an easy signature and hence provide a greater propensity for the model to misclassify these forgeries as genuine. By the same token, the forger finds it more complex to forge a difficult signature and hence our model perfectly classifies each forgery. The overall results show a relatively low misclassification rate across all complexities with an average of 5.5% misclassification for easy and moderaly easy signatures and 0% misclassification for all difficult signatures. This low misclassification rate can be attributed to the fact that dynamic features when forged result in poor accuracy and hence make them easier to detect.

Signature Feature Experiments

This section covers the design and implementation of the signature feature discrimination experiments that were conducted to determine the effect that varying amounts of significant features has on the performance level of the network. The features used in the various features sets are as follows: for 3 features; x, y co-ordinates and pen-tip pressure, for 5 features; x, y co-ordinates, pen-tip pressure and the two pen-tip angles θ_x and θ_y, for 7 features; x, y co-ordinates, pen-tip pressure, the two pen-tip angles θ_x and θ_y, pen-tip velocity V(t) and path-tangent T_θ. Discussion for Network connectivity is the same as that in section 4.6.3.

Three different experiments were conducted for the three signature feature sets, i.e. 3, 5 and 7 features. The data set consisted of 15 users of varying signature complexities. Within the 3 feature sets the x and y co-ordinates represented the static features while the remaining features were dynamic in nature. Also of note is the

fact that within the 7 feature set both velocity and path-tangent are derived features where velocity is computed using the x and y co-ordinates and path-tangent is then calculated based on the velocity.

Three identical networks were utilized to conduct the experiments based on 3, 5 and 7 features. The networks used for the experiments are LSTM RNNs with an input layer of 3, 5 and 7 neurons respectively, into which the normalised signature data is input. The hidden layer contains 10 memory blocks that consists of 4 memory cells each. The output layer for the networks that were trained contained a single neuron whose activation is squashed through a sigmoid function in the range of [0.2,0.8]. The network contained 2948, 3174, 3408 weights and 45, 47, 49 unit activations for 3, 5 and 7 inputs respectively.

Once again a small learning rate of 0.0001 and a high momentum rate of 0.9 was chosen. The network was trained to an RMSE of 0.1. The test results in Table 4.3 for the 3 feature sets were obtained from the ROC curves in Figures 4.14 - 4.16 at their respective operating points and are as follows:

Forgery Type	**3 Features**		**5 Features**		**7 Features**	
	TAR	FAR	TAR	FAR	TAR	FAR
Casual	88.5%	34.1%	89.2%	21%	91.7%	14.6%
Skilled	90.5%	26.6%	91.2%	16.6%	94.9%	10.7%
Forensic	75%	25%	86.6%	16.3%	90%	14.3%

Table 4.4: **Signature Feature Set Performance Summary:** This table illustrates the performance rate of the network on feature vector sizes of 3, 5 and 7.

Forgery Type	**3 Features**	**5 Features**	**7 Features**
	C.I	C.I	C.I
Casual	0.982 to 1	0.962 to 1	0.931 to 1
Skilled	0.920 to 1	0.936 to 1	0.983 to 1
Forensic	1 to 1	0.995 to 1	1 to 1

Table 4.5: **Confidence Intervals for Signature Feature Experiments:** This table illustrates the corresponding confidence intervals (C.I) for experiments in Table 4.4.

It can be seen in Table 4.3 and Figures 4.13 - 4.15 that an increase in signature features from 3 to 7 results in a decrease in the FAR and an increase in the TAR of the network. Firstly, it should be reiterated dynamic features are extremely difficult to reproduce [22]. Added to this is the fact that the same two features within each feature set is static in nature and therefore any additional features used across the 3 feature sets will be dynamic. As a result, the skill of the signer will not be taken into account within this analysis as all signers regardless of skill have an equal propensity to incorrectly reproduce these dynamic features. Our conclusion is then that if more dynamic features are used, the likelihood of the network misclassifying a forgery is dramatically reduced. This can be observed when looking at the feature set containing 3 features. In this case, the static features outweigh the dynamic features 2:n-2, where n represents the total number of features; as a result, the misclassification rates are much higher than when 5 and 7 features are considered. The rationale for this is that the network has fewer dynamic features to use for its classification process and hence fewer instances of incorrectly reproduced data from the forger to correctly classify the forgery as such. It can then be noted that by increasing the dynamic features from 3 to 5 in the 5 and 7 feature set respectively results in a decrease of the FAR, i.e. As per

Table 4.3 within the casual signers category the FAR fell from 34.1% to 21% to 14.6% when dynamic features totalled 1, 3 and 5 respectively. The same trend can also be noted across the skilled and forensic categories. To obtain the ROC curves a descision threshold value was varied as mentioned in section 4.5.5.

The ROC curves for casual forgeries, skilled forgeries and forensic forgeries based on 3,5 and 7 feature sets are as follows:

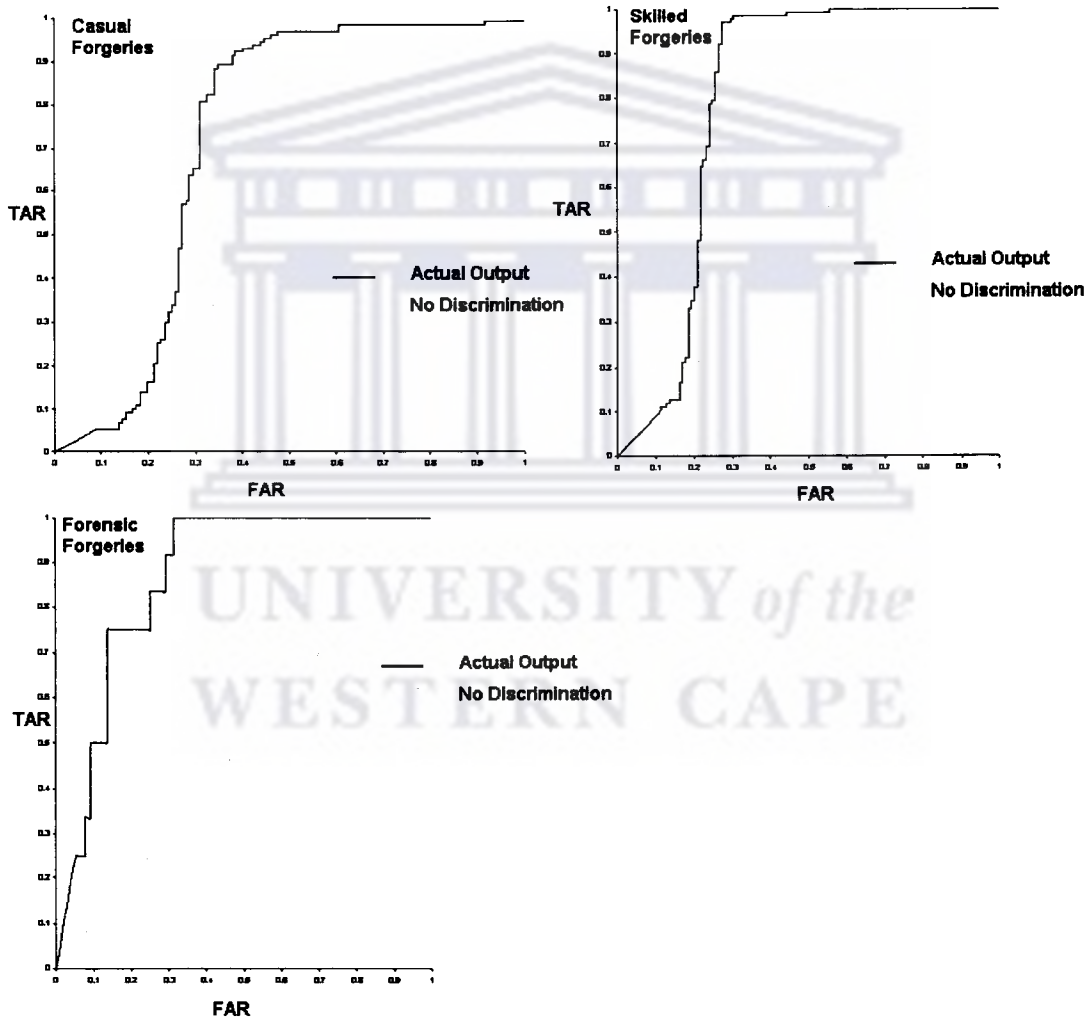

Figure 4.14: **ROC Curves:** 3 features

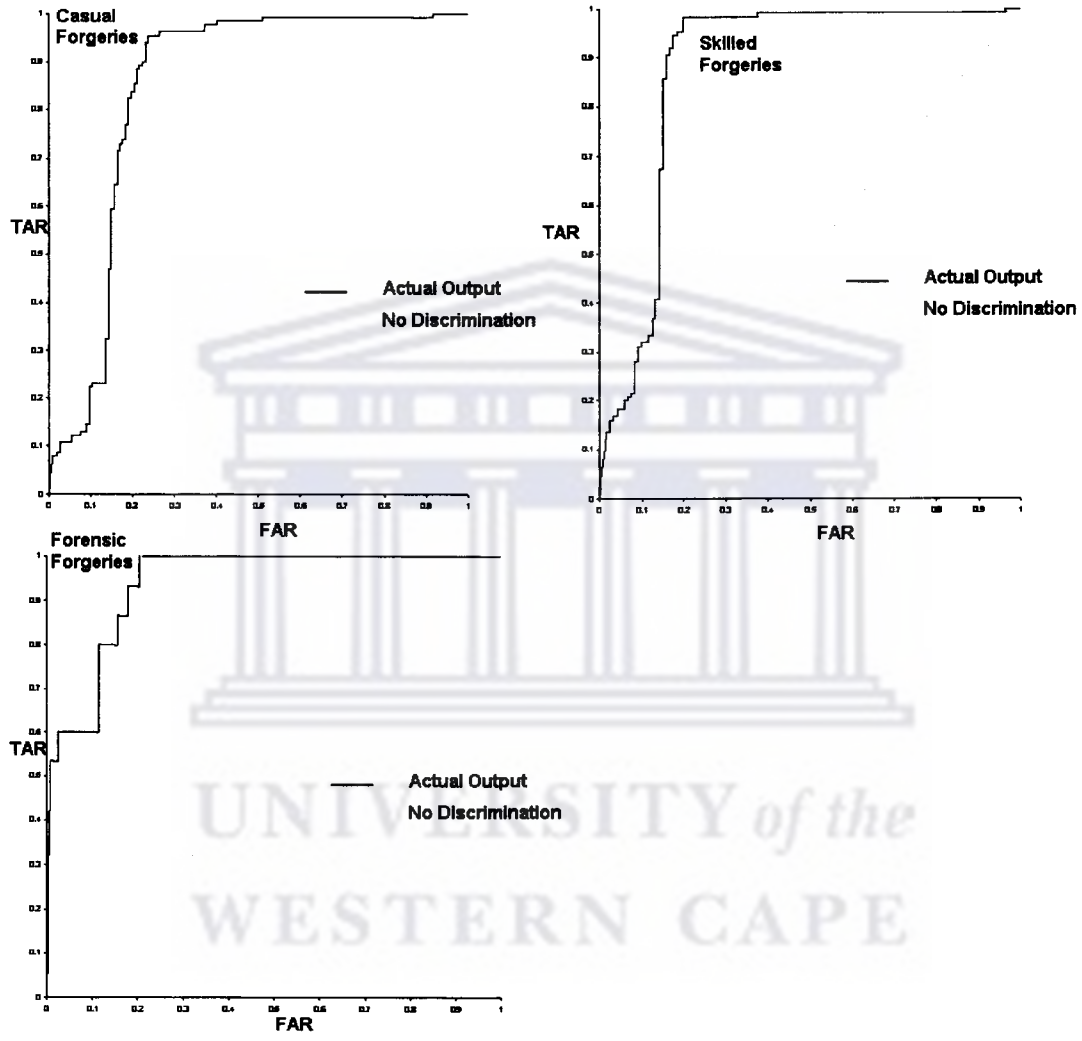

Figure 4.15: **ROC Curves:** 5 features

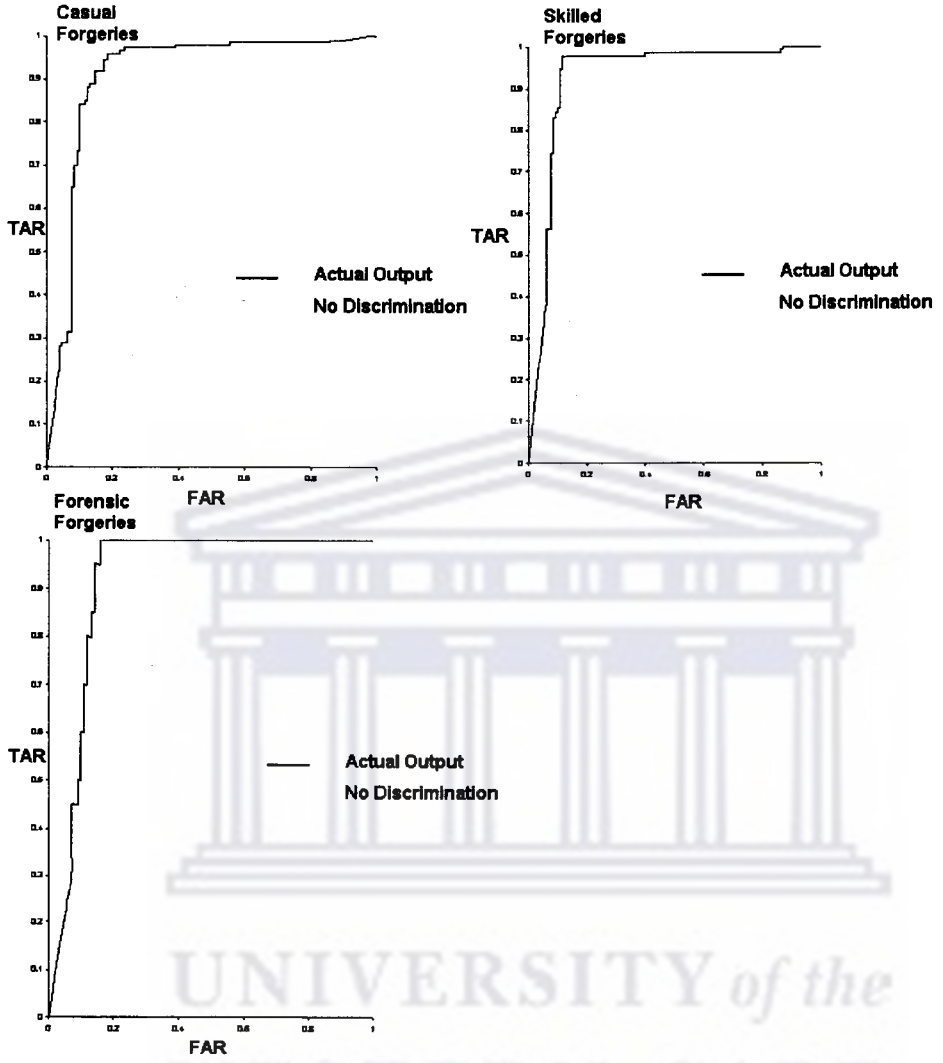

Figure 4.16: **ROC Curves:** 7 features

4.7 Summary

In Chapter 4, we have presented the results of our experiments, which have enabled us to deduce a number of conclusions based on LSTM RNNs to the task of modelling signature time series data.

Chapter 5

Conclusions and Directions for Future Research

5.1 Conclusion

The main focus of this thesis has been to investigate LSTM RNNs and its application to handwritten signature time series data. To our knowledge, LSTM RNNs and traditional RNNs, have not yet been applied to the task of signature verification. The most probable reason for this is that signature time series are long therefore traditional RNNs fail to learn such time series due to the long-term dependency problem.

In Chapter 4, we have presented the results of our experiments. These results have enabled us to draw a number of conclusions of relevance to HSV and LSTM RNNs. We have shown that a single LSTM RNN is able to model a entire signature database with a high degree of accuracy. To avoid instability within the network, the learning rate needs to be kept small and the momentum rate needs to be kept large. We have further proved that the signature complexity affects the performance level of a LSTM RNN when considering casual, skilled and forensic

forgeries. We also have experimented with feature sets of varying sizes and determined that the performance level of the network improves as the number of significant features increase. Experiments also show that dynamic signature features are the most difficult to forge since the forger places more emphasis on correctly reproducing the static signature features. Finally, this study provided us with a first time look at the application of LSTM RNNs to HSV since to our knowledge no published research has been carried out.

5.2 Directions for future research

5.2.1 Kalman Filters

[49] determined that LSTM training algorithms applied to time series depends on the instantaneous estimation of the gradient, i.e. the derivatives of the error function only take into account the distance between the current output and the corresponding desired target signal, which means that no prior gradient information is utilised.

A Decoupled Extended Kalman Filter (DEKF) [15, 24], on the other hand, computes the solution by making use of the derivatives at each time step in a sequence. The best curve fit for a set of data is found by minimising the standard deviation, i.e. the average distance between the data and the curve.

[49] applied LSTM with DEKF to predict symbols of a continual symbolic input stream with long-term dependencies that was not initially segmented into subsequences. [49] then found that LSTM RNNs combined with DEKF resulted in a faster convergence rate and improved the overall network performance when compared to the original gradient descent based algorithm.

It would certainly be useful to apply this improved algorithm to signature modelling and for that matter any task involving long time series, since the network is able to settle to a more accurate solution in a shorter space of time.

5.2.2 Growing LSTM RNNs

[53] proposed a method for growing LSTMs (GLSTM) RNNs in order to improve learning in signal prediction tasks. [53] notes that an important issue in ANNs topology design is that of determining a method for finding the number of hidden nodes in a network required for a given task, which results in satisfactory performance. This is known as the constructive or growing technique. [53] further notes that growing methods are specifically designed to automate the process of network topology estimation. This is accomplished by modifying both the synaptic weights as well as the network connectivity during training. [53] applied GLSTM and conventional LSTM RNNs to a forecasting problem in the biomedical domain. Numerous experiments were carried out using central nervous system control signals such as: the signals produced from the heart rate controller, the peripheric resistance controller and so on. The results indicated that GLSTM clearly outperforms conventional LSTM.

Since this task is similar to any other signal processing task such as signature verification, it would indeed be very useful. Trying to determine the number of hidden nodes in a network, to a large extent, is an art form. The approach that we used to determine our network topology in our experiments was merely that of observation followed by careful fine tuning. Training and fine tuning a network on such a large amount of data is extremely time consuming. Although this method proposed by [53] might add to the overall computation time, it will possibly contribute to reducing the element of guessing.

Bibliography

[1] *Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, State of the Art Recurrent Neural Networks.*[Online]. Available: www.idsia.ch/juergen/rnn.html [November 2003].

[2] R. Abbas, *Backpropagation Networks Prototype for Off-line Signature Verification*, Masters Thesis, Department of Computer Science, Royal Melbourne Institute of Technology, March, 1994.

[3] A.D. Back and A.C. Tsoi, *FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modelling*, Neural Computation, vol. 3, pp 375-385, 1991.

[4] H. Baltzakis and N. Papamarkos, *A new signature verification technique based on a two-stage neural network classifier*, Engineering Applications of Artificial Intelligence, vol. 14, pp 95-103, 2001.

[5] Y.J. Bae and M.C. Fairhurst, *Parallelism in Dynamic Time Warping for Automatic Signature Verification*, Third International Conference on Document Analysis and Recognition, vol 1, page 426, 1995.

[6] Y. Bengio, P. Simard, and P. Frasconi, *Learning long-term dependencies with gradient descent is difficult*, IEEE Transactions on Neural Networks, 5(2):157-166, 1994.

[7] Y. Bengio and P. Frasconi, *Credit assignment through time: Alternatives to backpropagtion*, Advances in Neural Information Processing Systems 6, pp 75-82, 1994.

[8] Y. Bengio and P. Frasconi, *Diffusion of Context and Credit Information in Markovian Models*, Journal of Artificial Intelligence Research, vol. 3, p. 249-270, 1995.

[9] K. Boehm, W. Broll, and M. Sokolewicz, *Dynamic Gesture Recognition using Neural Networks; A Fundament for Advanced Interaction Construction*,SPIE Conference Electronic Imaging Science and Technology, San Jose California, 1994.

[10] S. Chu, E. Keogh, D. Hart, and M. Pazzani, *Iterative Deepening Dynamic Time Warping for Time Series*, Proceedings of the Second SIAM International Conference on Data Mining, Part IV: Time Series Analysis, USA, 2002.

[11] J. Dolfing, E. Aarts, and J. van Oosterhout, *On-line Signature Verification with Hidden Markov Models*, Proceedings. Fourteenth International Conference on Pattern Recognition, vol. 2, pp 1309-12, 1998.

[12] D. Eck and J. Schmidhuber, *Finding Temporal Structure in Music: Blues Improvisation with LSTM Reccurent Networks*, Neural Networks for Signal Processing XII, pp 747756, 2002.

[13] D. Eck, A. Graves and J. Schmidhuber, *A New Approach to Continuous Speech Recognition Using LSTM Recurrent Neural Networks*, Technical Report IDSIA-14-03, IDSIA, www.idsia.ch/techrep.html, 2003.

[14] M.C. Fairhurst and E. Kaplani, *Strategies for exploiting signature verification based on complexity estimates*, Ph.D. Thesis, Department of Electronics, University of Kent, 1997.

[15] L. A. FeldKamp and G.V. Puskorious, *Training controllers for robustness: multi-stream Decoupled Extended Kalman Filters*, IEEE Transactions on Neural Networks, pp 279-297, Vol. 5, 1994.

[16] P. Frasconi, M. Gori and Y. Bengio, *Recurrent Neural Networks for Adaptive Temporal Processing*, Proceedings of the 6th Italian Workshop on Parallel Architectures and Neural Networks, 1993.

[17] F. Gers, J.Schimdhuber, *Learning to forget: Continual prediction with LSTM*, Neural Computation, 12(10):2451-2471, 2000.

[18] F. Gers, *Long-Short Term Memory in Recurrent Neural Networks*, Ph.D. Thesis, Lausanne, EPFL, 2001.

[19] F. Gers, N. Schraudolph, J. Schmidhuber, *Learning Precise Timing with LSTM Recurrent Networks*, Journal of Machine Learning Research 3, pp 115-143, 2002.

[20] G. Gupta and A. McCabe, *A Review of Dynamic Handwritten Signature Verification*, Department of Computer Science, James Cook University, Townsville, Australia, 1997.

[21] J. Hammerton, *Named Entity Recognition with Long Short-Term Memory*, Proceedings of the Seventh Conference on Natural Language Learning, Edmonton, Canada, pp 172-175, 2003.

[22] T. Hastie, E. Kishon, *A Model for Signature Verification*, Department of Electrical Engineering, Standford University, 1992.

[23] S. Haykin, *Neural Networks: A comprehensive foundation*, 2nd edition, Prentice Hall, 1998.

[24] S. Haykin, *Kalman filtering and neural networks*, Wiley, 2001.

[25] B. Herbst and H. Coetzer, *On an Offline Signature Verification System*, Proceedings of the 9th Annual South African Workshop on Pattern Recognition, pp 39-43 , 1998.

[26] J. Hertz, A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company, Inc., Redwood City, CA, 1991.

[27] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation 9(8):1735-1780, 1997.

[28] S. Hochreiter, Y. Bengio, and J. Schmidhuber, *Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies*, A Field Guide to Dynamical Recurrent Networks, Editors: John F. Kolen and Stefan C. Kremer, chapter 14, pages 231-234, Wiley-IEEE Press, 2001.

[29] S. Hochreiter and J. Schmidhuber, *LSTM can solve hard long time lag problems*, Advances in Neural Information Processing Systems 6, pp 473-479, Cambridge, 1997.

[30] J. Higashino, *Signature verificaiton system on neuro-computer*,Proceedings of 11th IAPR International Conference on Pattern Recognition, vol. 3, pp 517-521, 1992.

[31] O. Hilton, *Signatures - Review and New View*, Journal of Forensic Sciences, vol. 37.[Online]. Available: http://www.astm.org [March 2002].

[32] H. Iwamida, S. Katagiri, E. McDermott, and Y. Tohkura, *A hybrid speech recognition system using HMMs with an LVQ-based codebook*, Proceedings of the International Conference on Acoustics Speech and Signal Processing, pp 489-492,1990.

[33] H. Jaeger, *The "echo state" approach to analysing and training recurrent neural networks*, Technical Report, GMD Report 148, German National Research Center for Information Technology, 43 pages, 2001.

[34] H. Jaeger, *Beautiful Beast: Recurrent Neural Networks.*[Online]. Available: http://www.ais.fraunhofer.de/INDY/herbert/ESNTutorial/CompleteTutorial.pdf [July 2003].

[35] M. Kaiser, *Time-Delay Neural Networks for Control*, 4th International Symposium on Robot Control, Capri, Italy, 1994.

[36] K. Karplus, C. Barrett,and R. Hughey, *Hidden Markov Models for Detecting Remote Protein Homologies*, Department of Computer Science, University of California, 1998.

[37] B. Kashi, J. Hu, W.L. Nelson and W. Turin, *Hidden Markov Model approach to on-line handwritten signature verification*, International Journal on Document Analysis, vol. 1, pages 319-330, 1998.

[38] K. Kasper, H. Reininger, D. Wolf and, H. W üst, *Fully Recurrent Neural Networks for Phoneme Based Speech Recognition*, University of Frankfurt.

[39] A.A. Kholmatov., *Biometric Identity Verification Using On-Line Signature Verification*, Graduate School of Engineering and Natural Sciences, Master of Science, Sabanci University, 2003.

[40] J.F. Kolen and S.C. Kremer (Eds.), *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE, 2001.

[41] S. Lee and E. Lee, *Integrated Segmentation and Recognition of Connected Handwritten Characters with Recurrent Neural Networks*, Document Recognition III, Proceedings of the International Society for Optical Engineering, Vol. 2660, pp 251-261, 1996.

[42] L. Lee, *Neural Approaches for Human Signature Verification*, Proceedings of the Third International Conference on Document Analysis and Recognition, Vol. 2, pp 1055-1058, 1995.

[43] R. Martens and L. Claesen. *On-line Signature Verification based on dynamic time-warping*, Proceedings of the 13th International Conference on Pattern Recognition, 1996.

[44] M. L. Forcada , *Neural Networks: Automata and Formal Models of Computation*, Ninth International Conference on Artificial Neural Networks, Tutorial 6, 1999.

[45] M. Mozer, *Induction of multiscale temporal structure*, In J. E. Moody, S. J. Hanson, R. P. Lippmann (Eds.), Advances in Neural Information Processing Systems 4, pp 275-282, 1992.

[46] M. E. Munich and P. Perona, *Continuous Dynamic Time Warping for translation-invariant curve alignment with applications to signature verification*, International Conference on Computer Vision, vol. 1, page 108, 1999.

[47] M. Munich and P. Perona, *Visual Signature Verification using Affine Arc-length*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2, pp 180-186, 1999.

[48] S. Parveen and P.D. Green, *Speech Recognition with Missing Data using Recurrent Neural Nets*, Neural Information Processing Systems, 2001.

[49] J. A. Prez-Ortiz, F. A. Gers, D. Eck, and J. Schmidhuber, *Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets*, Neural Networks, 2003.

[50] R. Plamadon and G. Lorette, *Automatic Signature Verification and Writer Indentification-The State of the Art*, Pattern Recognition, vol. 22, no. 2, pp 107-131, 1989.

[51] M. Qian and R. Srinivasan,*Classifying process transitions using dynamic time warping*, Presented in the Regional Symposium on Chemical Engineering, Kuala Lumpur, Malaysia, Oct 28-30, 2002.

[52] L. R. Rabiner, *A tutorial on hidden markov models and and selected applications in speech recognition*, Proceedings of the IEEE, vol. 77, no. 2, pp 257-285, 1989.

[53] E. S. L. Ribeiro, *Incremental Construction of LSTM Recurrent Neural Networks*, Technical Report, UPC-LSI, June, 2002.

[54] G. Rigoli and A. Kosmala, *A Systematic Comparision between On-line and Off-line Methods for Signature Verification with Hidden Markov Models*, International Conference on Pattern Recognition, 1997.

[55] M.B. Ring, *Learning sequential tasks by incrementally adding higher orders*, Advances in Neural Information Processing Systems 5, Morgan Kaufmann Publishers, pp 115-122, 1993.

[56] D.E. Rumelhart, J.L. Mcclelland and The PDP research group, Eds. *Parallel Distributed Processing* MIT Press, Cambridge, MA, 1986.

[57] R. Sabourin, G. Genest and F.J. Prêteux, *Off-line Signature Verification by Local Granulometric Size Distributions*, IEEE Transactions on Pattern ANALYSIS and Machine Intelligence, Vol. 19, No. 9, 1997.

[58] R. Sabourin and G. Ginest, *An Extended-Shadow-Code Based Approach for Off-line Signature Verification: Part-II-Evaluation of Several Multi-Classifier Combination Strategies*, IEEE, 1995.

[59] G. Sun, H. Chen, and Y. Lee, *Time warping invariant neural networks*, Advances in Neural Information Processing Systems 4, California, 1993.

[60] A. Tsoi, D. So and, A. Sergejew, *Classification of Electroencephalogram using Artificial Neural Networks*, Advances in Neural Information Processing Systems 6, pp 1151-1158, 1994.

[61] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[62] H. Vullings, M. Verhaegen, and H. Verbruggen *ECG Segmentation Using Time-Warping*, IDA97, pages 275–285, 1997.

[63] A. Waibel, *Modular construction of time-delay neural networks for speech recognition*, Neural Computation, 1(1):39-46, 1989.

[64] A.E. Wan, *Finite Impulse Response Neural Networks with applications in time series prediction*, Ph.D. Thesis, Department of Electrical Engineering, Standford University, 1993.

[65] A.E. Wan, *Finite Impulse Response Neural Networks for autoregressive time series prediction*, Department of Electrical Engineering, Standford University, 1993.

[66] T. Wessels, *Hidden Markov Models for On-line Signature Verification*, Masters Thesis, University of Stellenbosch, 2001.

[67] R. Williams and D. Zipser, *A learning algorithm for continually running fully recurrent neural networks*, Neural Computation, 1(2):270-280, 1989b.

[68] H. Wak and J. Zurada, *Time Series prediction by a Neural Network Model Based on the Bi-directional Computation Style*, Time Series Prediction I, pp 2225-2230, International Joint Conference on Neural Networks, 2000.

[69] A. Weigend, B. Huberman and D. Rumelhart: *Predicting the future: A connectionist approach*, International Journal of Neural Systems, vol.1, pp 193-209, 1990.

[70] P.J. Werbos, *Backpropagation through time: what it does and how to do it*, International Proceedings of the IEEE, 78(10):1550-1560, 1990.

[71] B. Zhang, M. Fu and H. Yan, *Handwritten Signature Verification based on Neural 'Gas' Based Vector Quantization*, International Conference on Pattern Recognition, page 44, 1998.