論文 / 著書情報
Article / Book Information

題目(和文)	適応フィルタにおけるエコーステートネットワークの貯留槽適応と読み出し部の低コストオンライン学習
Title(English)	Reservoir Adaptation and Low Cost Online Readout Training for Echo State Network in Adaptive Filtering
著者(和文)	YUENYONGSUMETH
Author(English)	Sumeth Yuenyong
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第9634号, 授与年月日:2014年9月25日, 学位の種別:課程博士, 審査員:西原　明法,國枝　博昭,山田　功,府川　和彦,杉山　将,田中　聡久
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第9634号, Conferred date:2014/9/25, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Tokyo Institute of Technology

Reservoir Adaptation and Low Cost Online Readout Training for Echo State Network in Adaptive Filtering

YUENYONG Sumeth

June 2014

Tokyo Institute of Technology

Department of Communication and Integrated Systems

Reservoir Adaptation and Low Cost Online Readout Training for Echo State Network in Adaptive Filtering

A thesis by

YUENYONG Sumeth

Advisor: Professor NISHIHARA Akinori

Submitted in partial fulfillment for the degree of

Doctor of Philosophy

June 2014

Acknowledgment

I would like to thank my adviser, professor Akinori Nishihara for accepting me into this laboratory. His continued support and insights were very valuable throughout my entire time as a PhD student. I must also thank him for his help with my personal problems in recent months as well.

Thanks to my mom for her unwavering support of my decision to come to Japan, even if it meant that she will be alone for several years. And thank to my wife for coming here with me, and for always supporting me no matter how many arguments we have. It's not easy being married to a graduate student, I am sure.

Thanks to Anzai san and Ma Yanna, for helping me with administrative stuffs, filling out forms in Japanese, and getting an apartment. I really appreciate your help with everything.

Finally, thanks to the people of Japan, from whom my scholarship ultimately came from.

Abstract

This thesis deals with a new type of neural networks called Echo States Networks (ESN) and their use in nonlinear adaptive filtering. ESN use sparse, recursive and randomly generated recurrent networks as "reservoirs". The states of the neurons in the reservoir can be tapped into using a "readout" in order to closely approximate any desired response. The weights in the reservoir are not trained, and adaptation of the readout weights is as simple as training of linear filters. This makes ESN very promising as nonlinear adaptive filters, compared to traditional Recurrent Neural Network (RNN) where training is difficult with local minima.

However, despite easy training, ESN have two disadvantages. Firstly, the reservoirs are randomly generated, and the parameters involved in their generation requires manual tuning in order for ESN to perform well. Proper tuning is problem specific and require experience on the part of the user. Secondly, the covariance matrix formed by the reservoir states has extremely high eigenvalue spread, on the order of millions or more. Therefore online readout training requires the use of RLS algorithm instead of the much cheaper LMS. This is made worse by the fact that the reservoir size required for satisfactory performance in many problems is in the hundreds, making online adaptation by RLS very expensive.

This thesis proposes the solution to the above two problems. For the first problem, we propose the adaptation of the reservoir by maximizing the mutual information between the neurons states and the desired response. For the second problem, we propose the use of a specific type of feed-forward neural network as readouts, leading to reduced number of taps to be adapted by the RLS algorithm, and much reduced computational cost for online training of ESN with large reservoirs.

Publications

Journal Papers:

1. Sumeth Yuenyong and Akinori Nishihara. "Evolutionary Pre-Training for CRJ-Type Reservoir of Echo State Networks" conditionally accepted by *Neurocomputing.*

2. Sumeth Yuenyong and Akinori Nishihara. "A hybrid gradient-based and differential evolution algorithm for infinite impulse response adaptive filtering" *International Journal of Adaptive Control and Signal Processing* (2013)

Conference Papers:

1. Sumeth Yuenyong and Akinori Nishihara. "Training Recurrent Neural Network for Nonlinear Adaptive Channel Equalization with Differential Evolution" *Proceedings of 2013 RISP International Workshop on Nonlinear Circuits, Communication and Signal Processing*, vol.1, no.1, pp. 409-411. 2013.

2. Sumeth Yuenyong and Akinori Nishihara. "IIR Adaptive System Identification based on Particle Swarm Optimization with Improved Cost Function" *Proceedings of 2011 Signal Processing Symposium* vol.1, no.1, pp. 716-720. 2011.

Contents

List of Figures

List of Tables

List of Symbols

$\mathbf{u}(k)$: input vector

$\mathbf{x}(k)$: state vector

N: number of neurons in a reservoir

$\mathbf{W}$: reservoir weights matrix

$\mathbf{W}_{\text{in}}$: input weights matrix

$\mathbf{P}$: matrix with $\mathbf{x}(k)$ in each row

$\mathbf{w}_{\text{out}}$: output weights vector

$d(k)$: desired response

$u(k)$: scalar input

$y(k)$: output

μ: step size parameter

ϵ: generic regularization

η: scaling of regularizing noise in state update

L: length of training signals

ρ: sparsity of $\mathbf{W}$

v: input scaling

$J()$: reservoir pre-training cost function

$\max(|\lambda_{\mathbf{W}}|)$: magnitude of largest eigenvalue of $\mathbf{W}$, spectral radius

λ (by itself): forgetting factor of RLS $\mathbf{\Phi}$: covariance matrix formed by a reservoir's states

Chapter 1

Introduction

Linear finite impulse response (FIR) filters have enjoyed great success in adaptive filtering because of their simplicity, well-understood properties, and low cost learning algorithms [Widrow and Stearns, 1985]. However, there are times when a nonlinear and/or recursive structure must be employed in order to achieve the best possible performance for the problem at hand. Such situations include: nonlinear system identification [Nelles, 2001], prediction of signals generated by a nonlinear and/or recursive process [Kantz and Schreiber, 2004; Haykin and Li, 1995], and nonlinear channel equalization [Patra et al., 1999]. Nonlinear filter structures range in their complexity, training difficulty and applicability. This thesis deals with a very general type of nonlinear structure called Echo State Networks (ESN), which was proposed as an alternative to Recurrent Neural Networks (RNN)

We begin this chapter by briefly discussing common nonlinear filter structures. Next, RNN will be introduced, and discussed why training of RNN is a difficult and high cost problem. We then present ESN as an alternative to RNN and explain in details the theory as well as practical training of ESN. After that, we discuss how ESN have their own new set of challenges. This thesis is about overcoming those challenges. Finally, the chapter concludes with the contribution of this research and the structure of the rest of the thesis.

1.1 A Taxonomy of Nonlinear Filter Structures

1.1.1 Hammerstein and Weiner Models

The simplest type of nonlinear structure are the Hammerstein and Weiner models [Schetzen, 1981] where a linear filter is preceded or followed by a static nonlinearity such as the structure shown in Figure 1.1. While simple, this class of nonlinear structure is limited in applicability. The system/signal in question must be able to be expressed as a cascade of a linear system with some nonlinear function(s). Moreover, the nonlinearity must be manually chosen. The advantage of this type of structure is that its training complexity is basically the same as a linear filter.

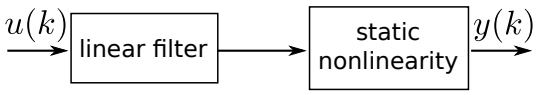

Figure 1.1: A Weiner nonlinear model.

1.1.2 Volterra and Wiener Series

Volterra and Wiener series [Koh and Powers, 1985] can be though of as nonlinear extension of Taylor series. A filter's input $u(k)$ and its output $y(k)$ is related by Volterra series expansion

$$\begin{aligned} y(k) = w_0 &+ \sum_{l_1=0}^{\infty} w_0(l_1)u(k-l_1) \\ &+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty} w_2(l_1,l_2)u(k-l_1)u(k-l_2) \\ &+ \sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\sum_{l_3=0}^{\infty} w_3(l_1,l_2,l_3)u(k-l_1)u(k-l_2)u(k-l_3) \\ &+ \ldots, \end{aligned}$$

where the terms w_i are called kernels. The problem with the Volterra series representation is one of complexity. As can be seen by the above equation, it is an infinite sum of infinite order, and in practice one must settle for a truncated series of some finite order. Moreover, the number of filter coefficients grows exponentially with the order, thus even a relative low order Volterra filter can have a very large number of coefficients. For this reason, in practical use the order of Volterra series is usually limited to only second order.

1.1.3 Kernel Methods

The idea of the kernel method [Príncipe et al., 2011] is to transform an input vector $\mathbf{u}(k)$ via a nonlinear map into a higher-dimensional feature space where the original nonlinear input-output relationship can be described by a linear filter. This is illustrated in Figure 1.2. The linear filter in the feature space's output due to a new input $\mathbf{u}(k)$ in the original space can be described by

$$y(k) = \omega(k)^T \phi(\mathbf{u}(k)) = \mu \sum_{j=1}^{k} e(j)\sigma(\mathbf{u}(j), \mathbf{u}(k))$$

where σ is the kernel function, ω is the filter's coefficient vector in feature space and e is the error produced by the filter. This is made possible by the kernel trick that allows one to avoid actually having to compute the transformed input $\phi(\mathbf{u})$. The difficulty in using this method is that the kernel σ must be appropriately chosen. Both the Kernel Methods and Volterra series cannot model nonlinear

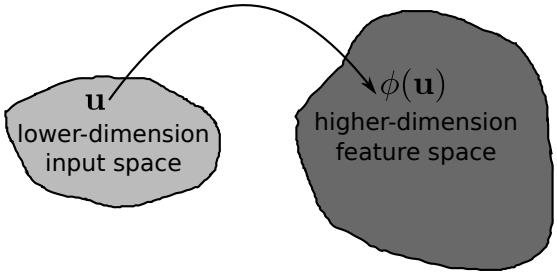

Figure 1.2: Nonlinear mapping from input to feature space.

systems with feedback. This can easily be seen by simple inspection that they contained no feedback term.

1.1.4 Neural Networks

A neural network consists of interconnections of individual computing elements called neurons. A diagram of a neuron is depicted in Figure 1.3. A neuron computes a sum of its inputs multiplied by their corresponding connection weights and feeds the result into an activation function f to produce its output. The output $y(k)$ of a neuron is given by

$$y(k) = f(\mathbf{w}^T \mathbf{u}(k)), \tag{1.1}$$

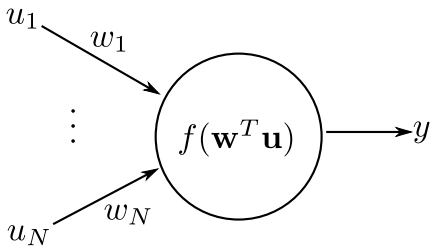

Figure 1.3: A neuron. In the rest of this thesis, a neuron is denoted simply by a blank circle.

where $\mathbf{w}$ is the neuron's input weights vector, $\mathbf{u}$ is its input vector and f is a scalar activation function.

A neural network is formed when multiple neurons are connected together. The topology of the connection allows for classification of neural networks into two types, feed-forward and recurrent neural networks (FNN and RNN). In FNN, there is no feedback connection and the signals only flow one way from input to output. FNN are also known as multi layer perceptron (MLP) when the activation function f is a step function. It has been proven that FNN are universal approximators of functions without feedback [Hornik et al., 1989]. FNN are quite popular in signal processing applications, when they are fed by a tap delay line [Haykin, 2005], as shown in Figure 1.4. However, due to the lack of feedback connection, this type of neural networks share the same disadvantage as Voltera series and kernel filters, that they cannot approximate nonlinear systems with feedback.

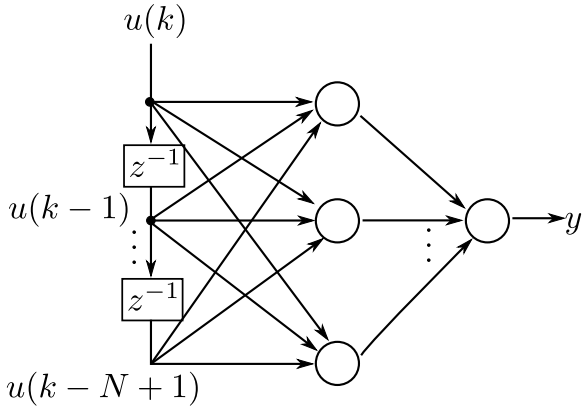

Figure 1.4: A feed-forward neural network with tap delay line.

Neural networks with any feedback connection can be classified as RNN. However, this term is rather generic, as it applies to any network which contains a

feedback connection. There are RNN with specific structure, activation functions and training regime which are known under their own names such as Hopfield and Jordan networks, Boltzmann machines, etc. In this work, we assign the term RNN to mean fully recurrent neural networks, where every neuron is connected to every other neurons and also to itself, and the activation function is a smooth function. A diagram of RNN is shown in Figure 1.5[1]. The input weights to the network are denoted by the matrix $\mathbf{W}_{\text{in}}$. The connection weights between the neurons themselves, which can be viewed as feedback connections, are denoted by the matrix $\mathbf{W}$. The input to the network at time k is denoted by the vector $\mathbf{u}(k)$.

RNN are dynamical systems, where the concept of network states applies. The state of RNN at time k are the outputs of each neuron. We denote the state by the vector $\mathbf{x}(k)$. The state is updated from time $k-1$ to k by the state update equation

$$\mathbf{x}(k) = f\left[\mathbf{W}\mathbf{x}(k-1) + \mathbf{W}_{\text{in}}\mathbf{u}(k)\right]. \tag{1.2}$$

The output $y(k)$ of the entire network can be tapped from any one element of the state vector.

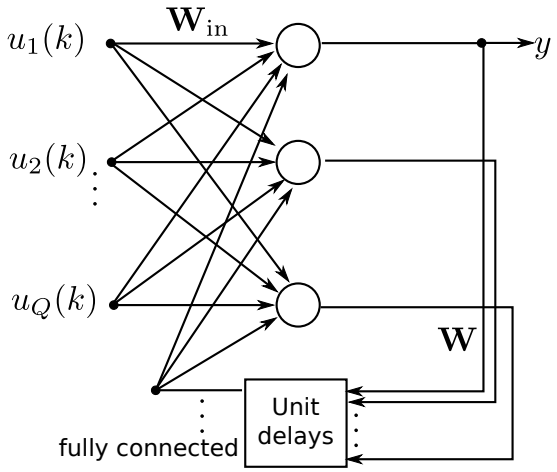

Figure 1.5: A fully recurrent neural network.

RNN have also been shown to be universal function approximators [Jin et al., 1995]. Moreover, by considering the state variables as feedback, it can be shown that the output of RNN can be mapped to the nonlinear autoregressive moving average (NARMA) nonlinear representation [Nerrand et al., 1993]. The output of

[1]For RNN, since they have feedback, there is no need for a delay line like in FNN, instead, the inputs are "fanned out" to all the neurons

a NARMA system can be written as

$$y(k) = f(y(k-1), ..., y(k-d_y), u(k), ..., u(k-d_u)), \tag{1.3}$$

where f is any nonlinear function and d_y, d_u are the orders of feedback and input vectors, respectively. This is a general nonlinear difference equation which can represent any discrete time nonlinear system, much the same way as linear ARMA can represent any discrete time linear system. The ability of RNN to approximate any nonlinear function with feedback make them the most general nonlinear filter structure. However, their use in adaptive filtering is rather limited because of the difficulty and computational cost to train them well. We will discuss these in the next section.

1.2 Training of RNN

Neural networks are usually trained with some sort of gradient based algorithm. It is well known that training of neural networks is a nonlinear optimization problem with local minima. The local minima can trap any gradient based algorithm and cause the trained network to have sub-optimal performance. Both FNN and RNN suffer from this problem.

However, compared to FNN, RNN are even harder to train. In the case of RNN, the gradient with respect to connection weights are more expensive to calculate, on the order of $O(N^4)$ [Williams and Zipser, 1989], compared to $O(N^2)$ for FNN. Moreover, the error surface of RNN can be more complicated. The study in [Horn et al., 2009] clearly demonstrated this. They fed a white noise input into a single neuron with one input and one feedback weight and recorded the output. Then the exact same input was fed to neurons with different weights values and their outputs recorded. The mean squared error (MSE) between these outputs and the output of the original neuron forms an error surface. It was discovered that even with this small, matched-structure case, the error surface can be very complicated with multiple local minima. In contrast, large, practical FNN can have unimodal error surface under certain conditions [Bianchini et al., 1995].

The same study also proposed a way to overcome the local minima problem in RNN. By inspection of 2D error surfaces generated as described above, they found that the majority of local minima occur because of saturation caused by large

values of weights. Therefore they proposed adding a regularization term which is the sum of the weights, to the MSE to steer the gradient descent away from large weight values. In the case of a single neuron this leads to 99% convergent rate to the global minimum. However, we note that the error surfaces they studied come from matched structure case, which is of course not applicable to practical use of RNN. Moreover, even for the matched structure case, the same study reported that when the number of neurons is increased to just two neurons, the rate that the global minimum is reached drops significantly.

The high cost of gradient calculation can be addressed by using the pipelined RNN [Zhao and Zhang, 2009; Zhao et al., 2011; Mandic and Chambers, 1999], which is a modular structure where each unit consists of a small recurrent network in order to reduce the computational cost of training large RNN. Still, such structure has local minima and it introduces the problem of delay between each module which needs to be compensated.

Another problem with gradient-based training of RNN is that the gradient tend to vanish, leading to extremely slow of even stagnant weights update [Hochreiter, 1998], especially for problems that require long memory. Their proposed solution is to use special units called "Long Short Term Memory" (LTSM) in addition to normal neurons in RNN. However, pipelined RNN and LTSM have not been combined, therefore the user of RNN has to choose between non vanishing gradient and lower computational cost.

At present, the best first order training algorithm reported for RNN is the APRL algorithm [Atiya and Parlos, 2000], which was shown to be much better than direct weight update following the output gradient. Although the weight update in APRL is different, it is still based on gradient descent and therefore suffer from the same local minima problem. The use of second order methods based on the Extended Kalman Filter to train RNN seem to yield better results that first order methods [Jaeger, 2002]. However, these methods are really computationally expensive since the Hessian has to be calculated or estimated, in addition to the gradient. It is simply trading more calculation for improved performance, and the problem of local minima still exists.

1.2.1 Training of RNN with Global Optimization

The difficulties discussed above had lead to efforts to train RNN using global optimization (GO) algorithms such as Particle Swarm Optimization or Adaptive Simulated Annealing [Krusienski and Jenkins, 2004; Ingber, 1996]. These algorithms can search multi-modal surfaces, and do not require calculating the gradient. Applying a GO algorithm to train small RNN for adaptive equalization problem had been shown to produce lower symbol error rate than using gradient descent [Yuenyong and NISHIHARA, 2013].

This may sound like GO algorithms should be able to solve the problem of training RNN. However, GO algorithms do have disadvantages of their own. These algorithms are probabilistic and heuristic in nature, meaning that there is no proof of convergence except for very specific conditions. Moreover, these algorithms are population based and the size of the population required to produce good result grows with the number of neurons in the network to be trained.

Furthermore, these algorithms need the MSE to be estimated and cannot work directly on the instantaneous error values[2]. This means that they do not work on a sample-by-sample basis, but are window-based in which all the outputs in a window must be recalculated every time the weights change. This and the fact that they are population based makes the computational cost of GO algorithms in training RNN impractical but for very small networks containing only a few neurons. Thus, just like gradient based methods, there is no guarantee of consistently reaching the global minimum for most practical RNN.

Despite the disadvantages of GO algorithms, they are the only option when one is faced with a multi-modal function which the derivative cannot be easily calculated, or does not exist at all. We shall encounter such a function later in this thesis, and we will use a GO algorithm called Differential Evolution (DE) to optimize that function. The details of DE can be found in Appendix A.

1.3 Echo State Network

Now we come to the main subject matter of this thesis, Echo State Network (ESN). This is a new type of neural network proposed in [Jaeger, 2001] to overcome the

[2]GO algorithms are quite sensitive to noise in the cost function. Thus variations in the instantaneous error produced by a filter would cause them to misconverge.

difficulties of RNN training.

ESN is based on employing large, randomly generated and sparsely connected recursive networks as "reservoirs" in a sense that the outputs of a reservoir's neurons provide a rich pool of signals that can be tapped into by using a "readout" layer in order to closely approximate any desired response.

Figure 1.6 compares RNN vs ESN. In RNN, the neurons are fully connected ($\mathbf{W}$ is a dense matrix) and the output $y(k)$ is taken from one of the neuron. In ESN, the neurons are sparsely connected ($\mathbf{W}$ is a sparse matrix), and $y(k)$ is formed by linear combination of the neurons states. The number of neurons N is usually large for ESN. There are examples which use hundreds or even thousands of neurons. For comparison, N in the case of RNN is usually limited to less than 100. For the input weights $\mathbf{W}_{\text{in}}$, apart from the matrix being larger to accommodate large N, there is no structural difference between that of RNN and ESN[3].

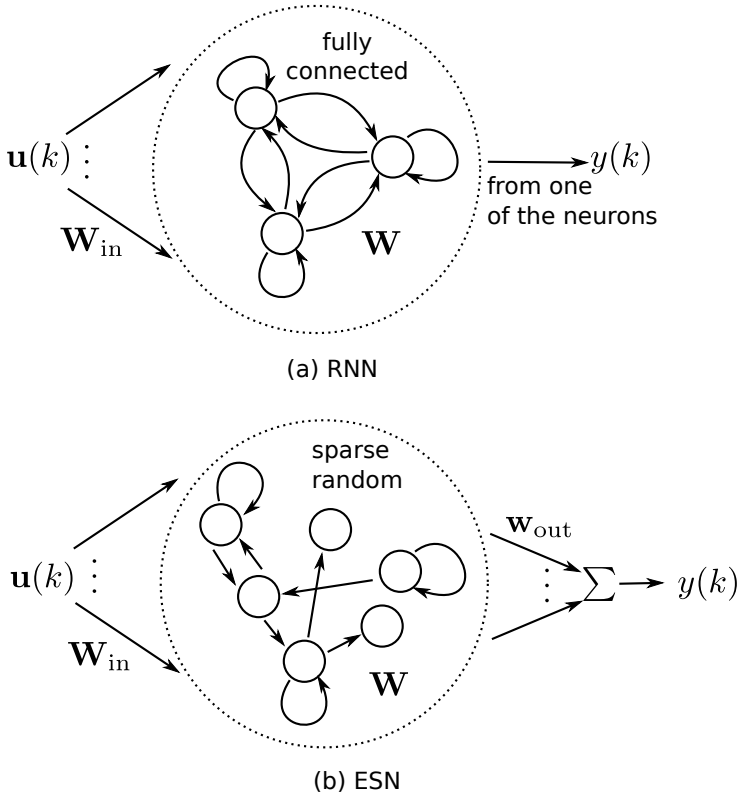

Figure 1.6: Comparison between (a) RNN and (b) ESN.

[3]In the ESN literature, sometime the word reservoir means only the neurons and $\mathbf{W}$. In this work, we also include the input weights, thus reservoir in this thesis means $\mathbf{W}, \mathbf{W}_{\text{in}}$ and the neurons

As mentioned, the output of ESN is formed by taking a linear combination of the reservoir state, this can be written as

$$y(k) = \mathbf{w}_{\text{out}}^T \mathbf{x}(k), \tag{1.4}$$

where $\mathbf{w}_{\text{out}}$ is the output or the readout weights. Apart from a linear combination, one can also use a nonlinear neuron to produce the output, which can be written as

$$y(k) = \tanh\left[\mathbf{w}_{\text{out}}^T \mathbf{x}(k)\right]. \tag{1.5}$$

The key point in ESN is that the matrices $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ never need to be adapted. The only parameter that is adapted or trained is the readout weight $\mathbf{w}_{\text{out}}$. By inspection of (1.4), we can make an analogy to FIR filter where $y(k)$ is a linear function of the filter weights. Therefore, $\mathbf{w}_{\text{out}}$ in (1.4) can be trained using any method that can train linear FIR filters[4]. In the case where a nonlinear neuron is used as the readout, the MSE can be made linear with respect to $\mathbf{w}_{\text{out}}$ by applying $\tanh^{-1}$ to the desired response, we shall have more to say about this when we discuss ESN training later in this chapter.

It is also possible to have feedback connection from the output back into the reservoir. In such case, the feedback connections are dense, and they are randomly initialized and not trained, just like $\mathbf{W}_{\text{in}}$. Using feedback connection is a technique which may improve the performance when the task at hand requires long memory. However the impact of feedback connections is not well understood in ESN theory since it was developed assuming no feedback connections. Therefore, in this work we shall exclusively use ESN without feedback connections.

The ESN approach to adaptive filtering allows one to utilize the power of RNN to perform nonlinear filtering without all the difficulties that come with their training as explained in Section 1.2. Further, it is possible to use results from linear adaptive filter to analyze the filter since the readout layer and the weights adapted are linear. Therefore, we can compute information such as excess MSE, convergent speed, etc., when a certain algorithm is applied to train the readout. This essentially turns difficult to analyze, nonlinear and recursive RNN into well understood and simple FIR filters.

[4]This is only a theoretical statement. In practice the LMS algorithm can not be used to train $\mathbf{w}_{\text{out}}$ well. The reason will be discussed later in this chapter.

1.4 Theory of ESN

Before we discuss training and challenges involved in using ESN for adaptive filtering. It is important for understanding to know the theory behind ESN, which we discuss in this section.

1.4.1 Echo State Property

The key to ESN is the "echo states property", which has the following formal definition. For all left-infinite input sequence $\mathbf{u}(k)$ where $k = ..., -2, -1, 0$ and for all state sequences $\mathbf{x}(k), \mathbf{x}'(k)$, it holds that

$$\mathbf{x}'(k) = \mathbf{x}(k) \forall k \leq 0 \tag{1.6}$$

Intuitively, if the network had been running for a very long time, the current network state is uniquely determined by the history of input, regardless of the initial state. In order to have echo states, it is nessesary for the network to satisfy any one of the following three equivalent properties: state contracting, state forgetting, and input forgetting. We explain these three properties now, even though in practical use of ESN it is not essential to understand them, to prepare for later when we investigate ESN in the complex domain. These definitions are taken from [Jaeger, 2001]. Here, we try to explain them in a more intuitive manner. For a highly rigorous mathematical approach, please see the reference.

Definition 1 The following are all equivalent to the network having echo states. Let d denote Euclidean distance. We introduce a network state update operator T and write $\mathbf{x}(k+h) = T(\mathbf{x}(k), \mathbf{u}^h)$ to mean the network state that resulted from repeated application of (1.2) where a network with state $\mathbf{x}(k)$ is fed with input sequence $\mathbf{u}(k+1), ..., \mathbf{u}(k+h)$. The set A is some closed set.

1. The network is called *state contracting* if for all right-infinite input sequences $\mathbf{u}^{+\infty}$ there exist a null sequence[5] $(\delta_h)_{h>0}$ such that for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$, and for all input sequence prefixes $\mathbf{u}^h = \mathbf{u}(k), ..., \mathbf{u}(k+h)$ it hold that $d(T(\mathbf{x}, \mathbf{u}^h), T(\mathbf{x}', \mathbf{u}^h)) < \delta_h$, where d is the Euclidean distance on $\mathbb{R}^N$.

[5] A null sequence is a sequence which has a limit of zero.

2. The network is called *state forgetting* if for all left-infinite input sequences $..., \mathbf{u}(k-1), \mathbf{u}(k)$ there exists a null sequence $(\delta_h)_{h>0}$ such that for all states $\mathbf{x}, \mathbf{x}' \in A$, for all $h \geq 0$, and for all input sequences suffixes $\mathbf{u}^h = \mathbf{u}(k-h), ..., \mathbf{u}(k)$ it holds that $d(T(\mathbf{x}, \mathbf{u}^h), T(\mathbf{x}', \mathbf{u}^h)) < \delta_h$.

3. The network is called input forgetting if for all left-infinite input sequence $\mathbf{u}^{-\infty}$ there exist a null sequence $(\delta_h)_{h>0}$ such that for all $h \geq 0$, for all input sequence suffixes $\mathbf{u}^h = \mathbf{u}(k-h), ..., \mathbf{u}(k)$, for all states $\mathbf{x}, \mathbf{x}'$ end-compatible[6] with $\mathbf{u}^h$, it holds that $d(\mathbf{x}, \mathbf{x}') < \delta_h$.

Intuitively, state contracting means that two networks with different initial states fed with the same input will have their states coming closer and closer to each other as $k \to \infty$. The state forgetting property basically means the same thing, that as the length h approaches infinity, the state difference between two networks with initial states $\mathbf{x}, \mathbf{x}'$ approaches zero, thus, the networks has "forgotten" their initial states. Finally, the input forgetting property means that the effect on the state from previous inputs approaches zero as the network is fed by the new inputs.

In essence, the echo state property means that the effect of current state $\mathbf{x}(k)$ and current input $\mathbf{u}(k)$ on future state $\mathbf{x}(k+h)$ approaches zeros as h approaches infinity.

The definitions above, while formal, are difficult to check in practice. Fortunately, for networks that use the standard tanh activation function, the existence of echo states can easily be checked by verifying that the spectral radius of $\mathbf{W}$ is less than one. The spectral radius of $\mathbf{W}$ is given by $\max(|\lambda_{\mathbf{W}}|)$, where $\lambda_{\mathbf{W}}$ denotes the eigenvalues of $\mathbf{W}$. The reason behind this quick check for echo states can be seen from the following proof.

$$\begin{aligned} d(T(\mathbf{x}, \mathbf{u}), T(\mathbf{x}', \mathbf{u})) &= d(\tanh(\mathbf{W}_{\text{in}} + \mathbf{W}\mathbf{x}), \tanh(\mathbf{W}_{\text{in}} + \mathbf{W}\mathbf{x}')) \\ &\leq d(\mathbf{W}_{\text{in}} + \mathbf{W}\mathbf{x}, \mathbf{W}_{\text{in}} + \mathbf{W}\mathbf{x}') \\ &= d(\mathbf{W}\mathbf{x}, \mathbf{W}\mathbf{x}') \\ &= ||\mathbf{W}(\mathbf{x} - \mathbf{x}')|| \\ &\leq \max(|\lambda_{\mathbf{W}}|) \times d(\mathbf{x}, \mathbf{x}') \end{aligned} \quad , \qquad (1.7)$$

[6]End-compatible means that the state can be reached by feeding a particular input sequence to the network.

assuming that the spectral radius is less than one, the final line shows that the distance between the states $\mathbf{x}, \mathbf{x}'$ after state update is less than that before state update. This satisfies the state contacting property and according to Definition 1, it is equivalent to the network having echo state. The key step in the proof is going from the first to second line, where the fact that tanh is bounded between $[-1, 1]$ allows dropping it from the distance calculation.

In order to easily ensure the existence of echo states, in this thesis we shall exclusively use tanh as the activation function. Unless stated otherwise.

1.4.2 How Does ESN Works?

Up to this point we have discussed what is an Echo State Network and the echo states property. But why would ESN work at all? It seems remarkable that a randomly generated structure can be tapped by a linear combiner to produce any desired response. In order to illustrate how ESN work, consider the most general nonlinear system description where the desired response is some nonlinear function of the entire input history

$$d(k) = f(..., \mathbf{u}(k-1), \mathbf{u}(k)). \tag{1.8}$$

We would like to approximate the above $d(k)$ using some linear combination of the state $\mathbf{x}(k)$ of ESN. If echo states exist, then there is also a one-to-one correspondence between the current state and the input history. The current state can be written as

$$\mathbf{x}(k) = \mathbf{e}(..., \mathbf{u}(k-1), \mathbf{u}(k)), \tag{1.9}$$

where $\mathbf{e}$ is a vector-valued "echo[7]" function that uniquely maps the entire input history to the current state, (1.9) is valid due to the echo states property, allowing one to one mapping between input history and current state.

It can be seen that the above two equations are similar, thus we may be able to approximate $d(k)$ using the elements of the state vector. In order to do this we

[7]We follow the name used in [Jaeger, 2001].

write

$$\begin{aligned} d(k) &= \mathbf{f}(..., \mathbf{u}(k-1), \mathbf{u}(k)) \\ &\approx y(k) \\ &= \sum_{i=1}^{N} w_i e_i \\ &= \mathbf{w}_{\text{out}}^T \mathbf{x}(k). \end{aligned} \tag{1.10}$$

The last line of the above equation is exactly the output of ESN when a linear combination is used as the readout.

In essence, ESN work because the echo function maps uniquely the entire input history to the current states thanks to the echo states property. We can then take a linear combination of the current state to approximate as closely as possible the desired response d, due to the close similarity between (1.8) and (1.9). This similarity also suggests that other functions apart from a linear combiner can be used to produce the output as well. In fact, in Chapter 4 we shall use a particular type of FNN as the readout.

1.5 Training of ESN

Training of ESN involves adapting $\mathbf{w}_{\text{out}}$, which can be done either offline or online. We will now discuss offline training. First assume that the readout is a linear combiner, whose output can be written as

$$y(k) = \mathbf{w}_{\text{out}}^T \mathbf{x}(k).$$

Given the input $u(k)$ and the desired response $d(k)$ both of length L, we would like to minimize in the MSE sense the error

$$e(k) = d(k) - y(k) = d(k) - \mathbf{w}_{\text{out}}^T \mathbf{x}(k). \tag{1.11}$$

In the offline setting, this is a linear regression problem. The readout weights $\mathbf{w}_{\text{out}}$ can be computed by solving

$$\mathbf{P}\mathbf{w}_{\text{out}} = \mathbf{d}, \tag{1.12}$$

where $\mathbf{P}$ is a matrix whose rows are the state vector at each time sample

$$\mathbf{P} = \begin{bmatrix} \mathbf{x}(1)^T \\ \vdots \\ \mathbf{x}(L)^T \end{bmatrix}, \tag{1.13}$$

and $\mathbf{d}$ is a vector of desired response. Since L is generally larger than N the system in (1.12) does not have an exact solution. Linear regression looks for the least squares solution to (1.12), which can be obtained by calculating the pseudo inverse of $\mathbf{P}$.

$$\mathbf{P}^T\mathbf{P}\mathbf{w}_{\text{out}} = \mathbf{P}^T\mathbf{d}. \tag{1.14}$$

and solving for $\mathbf{w}_{\text{out}}$. Since linear regression belongs to statistic, not adaptive filtering which solves the normal equation. We now show that they are essentially the same. From (1.13), the product $\mathbf{P}^T\mathbf{P}$ can be written out as

$$\mathbf{P}^T\mathbf{P} = \begin{bmatrix} \sum_{k=0}^{L} x_1(k)x_1(k) & \cdots & \sum_{k=0}^{L} x_1(k)x_N(k) \\ \vdots & \ddots & \vdots \\ \sum_{k=0}^{L} x_N(k)x_1(k) & \cdots & \sum_{k=0}^{L} x_N(k)x_N(k) \end{bmatrix}, \tag{1.15}$$

which has the same form as the time-average correlation matrix [Haykin, 2005]. Similarly, $\mathbf{P}^T\mathbf{d}$ can be written out as

$$\mathbf{P}^T\mathbf{d} = \begin{bmatrix} \sum_{k=0}^{L} x_1(k)d(k) \\ \vdots \\ \sum_{k=0}^{L} x_N(k)d(k) \end{bmatrix}, \tag{1.16}$$

which has the same form as the time-average cross-correlation vector. Where in both of these, the lag on the tap delay line is replaced by different positions of $\mathbf{x}(k)$ like in "array" adaptive filters. Therefore, solving (1.14) for $\mathbf{w}_{\text{out}}$ is the same as solving the normal equation in linear FIR filter. Both linear regression and solving the normal equation arrives at the same solution for $\mathbf{w}_{\text{out}}$.

When the readout is a tanh neuron, the output is given by

$$y(k) = \tanh\left[\mathbf{w}_{\text{out}}\mathbf{x}(k)\right]. \tag{1.17}$$

Therefore, (1.12) is changed to

$$\tanh\left[\mathbf{P}\right]\mathbf{w}_{\text{out}} = \mathbf{d}. \tag{1.18}$$

But this would be a nonlinear regression problem. In order to avoid this, we may perform linear regression using the input to the tanh function of the output neuron as the data and the transform desired response $\tanh^{-1}\left[d(k)\right]$ as the target. This new regression problem can be written as

$$\mathbf{P}\mathbf{w}_{\text{out}} = \tanh^{-1}\left[\mathbf{d}\right]. \tag{1.19}$$

Doing this is justified since tanh() is invertible and one-to-one. It is clear that $\mathbf{w}_{\text{out}}$ which produces $\mathbf{w}_{\text{out}}^T\mathbf{x}(k)$ that closely approximates $\tanh^{-1}[d(k)]$ will also produce $\tanh\left[\mathbf{w}_{\text{out}}^T\mathbf{x}(k)\right]$ that closely approximates $d(k)$.

Another way to see this is to look at how the errors in the two cases are related. The "original error" is $e(k) = d(k) - \tanh[\mathbf{w}_{\text{out}}\mathbf{x}(k)]$. When we are solving (1.19), the "transformed error" that we are minimizing is $e'(k) = \tanh^{-1}[d(k)] - \mathbf{w}_{\text{out}}\mathbf{x}(k)$. Thus $e(k)$ and $e'(k)$ are related by $e(k) = \tanh[e'(k)]$ or $e^2(k) = \tanh^2[e'(k)]$. Thus, when $[e'(k)]^2$ is minimized, $e^2(k)$ is also minimized since $\tanh^2()$ is an even function with the minimum at 0.

Regularization of Training

A point to note about the training process of ESN is that, to produce the state vectors that fill the rows of $\mathbf{P}$, the state update in (1.2) has to be modified as

$$\mathbf{x}(k) = \tanh[\mathbf{W}_{\text{in}}\mathbf{u}(k) + \mathbf{W}\mathbf{x}(k-1)] + \eta\mathbf{a}, \tag{1.20}$$

the difference from (1.2) is the addition of the $\eta\mathbf{a}$ term, which serve as a regularization, η is a scalar factor and $\mathbf{a}$ is a noise vector drawn from the standard normal distribution. If unregularized state update as in (1.2) is used, the rows of $\mathbf{P}$ may be so similar to each other that (1.12) becomes an ill-conditioned problem, leading to very large output weights that generalize poorly to new inputs. Throughout this thesis, we fix the value of η to be 0.0001.

The training process of ESN may be summarized as follows:

1. Set the sparsity of $\mathbf{W}$.
2. Sample the elements of $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ from a uniform distribution in the domain $[-1, 1]$.
3. Scale the elements of $\mathbf{W}$ such that the spectral radius is a specific value less than 1.
4. Scale the elements of $\mathbf{W}_{\text{in}}$ by a scalar input scaling parameter v between zero and one.
5. Feed the network with $u(k)$, apply (1.20) and collect $\mathbf{x}(k)$ of each time step k into the rows of $\mathbf{P}$, collect each desired response sample $d(k)$ into the rows of $\mathbf{d}$.

6. Discard the top L_{washout} rows from both $\mathbf{P}$ and $\mathbf{d}$.

7. Solve (1.12) using any least squares method.

8. When the trained network is used to process new inputs, use the unregularize state update (1.2). The regularized state update (1.20) is only used for training.

The reason why the top L_{washout} rows of both $\mathbf{P}$ and $\mathbf{d}$ has to be discarded is because echo states take some time to establish, so the initial states should be "washed out" and not used in training. Exactly how many time steps it takes is not a hard fact, but from our experience as little as 10 samples is enough in certain applications. However, to be on the safe side and avoid any possible confusion if echo states have been established or not, we choose to fix $L_{\text{washout}} = 200$ in all simulations in this thesis unless explicitly stated otherwise.

It can be seen that the parameters involved in generating the reservoir are: the spectral radius, the input scaling, and the sparsity of $\mathbf{W}$. We will illustrate in Chapter 2 that the performance of ESN is sensitive to the choice of these parameters. The current practice in using ESN to set these parameters manually by trial and error, as well as by the user's experience.

For online training, the first 4 steps of the above procedure to generate the reservoir remains the same. The state vector can be updated with the unregularized state update equation (1.2), since in online training one is no longer solving the linear system in (1.12). To see how the readout can be trained online, we can make an analogy with linear FIR filters. It is clear to see that $\mathbf{x}(k)$ is analogous to the tap input signal and $\mathbf{w}_{\text{out}}$ is analogous to the filter's tap weights. Thus we can use any training algorithm for linear filters to train $\mathbf{w}_{\text{out}}$ online by replacing $\mathbf{u}(k)$ in the linear algorithm with the state vector $\mathbf{x}(k)$ and replace the tap delay line update with state update. To accommodate the washout period, the adaptation algorithm can be turned on only after L_{washout} time steps has passed (the state vector starts updating from $k = 0$ however), which can easily be checked in practice.

1.5.1 State Vector Augmentation

An easy way to improve the performance of ESN is state vector augmentation [Jaeger, 2003]. That is, before storing the state vector of each time instant $\mathbf{x}(k)$

into the rows of $\mathbf{P}$ for training, one can apply an arbitrary transformation to it, such as

$$\mathbf{x}'(k) = \left[u(k), \mathbf{x}(k), u^2(k), \mathbf{x}^2(k)\right], \tag{1.21}$$

for example, where $\mathbf{x}'(k)$ is the "augmented" state vector. There is no theory regarding state vector transformation, and its use in ESN take a see-what-works approach. However we note that the use of the input $u(k)$ is quite popular, as well as a squared version of the state vector itself. All of which is included in the above transformation.

State vector augmentation improves the performance at the cost of larger "effective" reservoir size. As can be seen from (1.21), the length of $\mathbf{x}'(k)$ is $2N + 2$ instead of N, therefore the size of $\mathbf{P}$ as well as the number of output weights also must be increased to match the length of the augmented state vector. Thus one must consider whether or not to augment the state vector by comparing the performance gain versus using a larger reservoir with un-augmented or "plain" state vectors.

1.5.2 Example

Next we present an example to illustrate ESN training. Consider a system identification problem where the task is to identify [Narendra and Parthasarathy, 1990]

$$y(k) = \frac{y(k-1)y(k-2)(y(k-1)+0.25)}{1+y^2(k-1)+y^2(k-2)} + u(k), \tag{1.22}$$

where the input $u(k)$ is a uniform noise in [0, 0.5]. We shall later refer to this problem as "NARMA2". We consider two different parameters sets for the reservoirs, we call these reservoir 1 and reservoir 2. In both sets, the reservoir size is $N = 20$, the sparsity of $\mathbf{W}$ is 5% and the spectral radius is 0.8. The difference between reservoir 1 and reservoir 2 is that the first has input scaling of 1.0, while the latter has input scaling of 0.3.

The reservoirs are fed with the input and the state vector $\mathbf{x}$ is updated using (1.20). At each time instant k, the current state vector $\mathbf{x}(k)$ is saved into the k^{th} row of the matrix $\mathbf{P}$, while the desired response sample $d(k)$ is saved into the k^{th} row of the vector $\mathbf{d}$. Once the entire input has been fed into the reservoir, the top 200 rows of $\mathbf{P}$ and $\mathbf{d}$ were discarded. Then the readout was obtained by solving (1.12) using pseudo inverse.

After training, the state vector is re-initialized to all zeros, another 500 samples was input for testing, the state update was done using (1.2) and the output $y(k)$ was calculated using (1.4). The plot of the test outputs produced by the two reservoirs and the desired response for this example in shown in Figure 1.7. The top panel shows the output of reservoir 1 with input scaling of 1.0, while the bottom panel shows the output of reservoir 2 with input scaling of 0.3. It can be seen that the output of reservoir 2 matched the desired response more closely than that of reservoir 1. This shows that a reservoir's parameters have significant effect on its performance.

Another point we would like to make about this example is that, if one calculate the eigenvalue spread of the covariance matrix formed by the reservoirs state vectors, the value comes out to be about 1.6×10^{16}. We have discussed earlier that the state vector is analogous to a FIR filter's tap input vector. Thus, online training of $\mathbf{w}_{\text{out}}$ is severely affected by this very large eigenvalue spread.

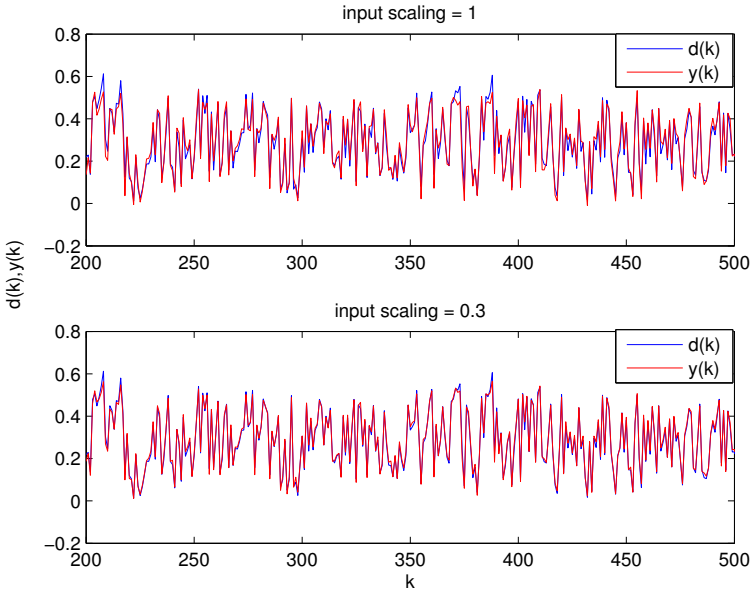

Figure 1.7: Result of ESN training example.

1.5.3 Challenges in ESN Training

ESN offer an intriguing alternative to RNN due to its simple training scheme. However, it is not without its own drawbacks. As the above example illustrates, there are two main challenges one faces. The first challenge comes from the random

generation of $\mathbf{W}, \mathbf{W}_{\text{in}}$, which involves parameters that must be chosen manually. Parameter choice has significant effect on a reservoir's performance. We elaborate more on this in Chapter 2.

The second challenge in ESN arises when it is used as an adaptive filtering structure, which in practice must be trained online. As can be seen from (1.4), the output equation of ESN resembles that of linear FIR filters, this suggests that any algorithm that can be used to train linear FIR filters online can be used to train $\mathbf{w}_{\text{out}}$ in the same manner. Theoretically speaking this is true, however, the problem is that the eigenvalue spread of the covariance matrix formed by the reservoir states $\mathbf{x}$ in ESN is extremely high. This eigenvalue spread is enormous compared to linear filter standard that the LMS algorithm can not be used to train $\mathbf{w}_{\text{out}}$ successfully[8]. One must resort to using the RLS algorithm which is insensitive to eigenvalue spread of the input, but, as the reservoir size in ESN can be large (100 neurons or more), the computational cost of RLS can become impractically high. This is the second challenge in ESN as adaptive filters.

1.6 Thesis Contribution and Organization

The contribution of this thesis is that it addresses these two challenges. For the first challenge, we propose a method to automatically optimize a reservoir's parameters so that it performs optimally given a particular problem. This method is presented in Chapter 3. For the second challenge, we propose using a new type of readout structure that significantly reduces the number of coefficient that must be adapted by the RLS algorithm for large reservoirs, greatly reducing the computational cost in online training. This is presented in Chapter 4.

The organization of this thesis as follows. In Chapter 2, ESN will be compared with RNN trained with different methods, in order to establish that ESN can perform at least as well, or better than RNN trained with the best training methods. It will be shown by experiments that the performance of ESN is sensitive to its parameters, setting the stage for our reservoir optimization method which will be presented in Chapter 3.

In Chapter 3, we present a method that adapts the reservoir parameters to specific input and desired response. We call this "pre-training" in order to distin-

[8]We will illustrate this fact in Chapter 4.

guish it from readout training. Pre-training of the reservoir allows one to avoid the trial and error process in manually choosing the parameters. We then extend the ESN concept to the complex domain, and finally finish the chapter by extending the reservoir pre-training method presented to the complex domain as well.

In Chapter 4, we deal with the problem of online training. We show that the eigenvalue spread problem lead to much higher MSE when the readout is trained using LMS, compared to when it is trained with RLS. We then investigate an $O(N)$ algorithm that is generally regarded in the ESN literature to be effective at online training and show that is its performance is actually only slightly better than LMS. We then propose a new readout structure and conduct many experiments to confirm its effectiveness as well as compares the computational cost against linear readout given the same reservoir size.

To conclude this introductory chapter, we started by outlining some nonlinear filter structures and showed that RNN is more general than others due to its recursive nature. We discussed the reasons why RNN are still not very widely used in adaptive filtering due to the difficulties in their training. We introduced ESN, discussed its theoretical foundation, and gave an example of how it can be used for nonlinear adaptive filtering problems. Next we discussed the challenges in using ESN, the solution to which is the main contribution of this thesis. Finally, we ended the chapter by presenting the organization of this thesis.

1.7 ESN vs Deep Learning

This section is an aside, it may be skipped without loss of understanding.

Currently, another hot topic in neural network is Deep Learning (DL) [Bengio, 2009], which compared to ESN, have a larger number of publications. In order to avoid any possible confusion whether ESN may be a subset of DL, in this section we briefly discuss DL and see whether there is any relationship to ESN.

DL derives its name from learning of FNN with many layers, hence "deep". Theoretically, a single layer FNN can represent any static function, however the number of neurons required may be very large. A network with multiple layers may be able to represent the same function with much lower number of weights. Less number of weights mean less amount of training data is required. Another advantage of using a deep architecture is that each layer can represent different

level of abstraction, the use of deep architecture allows for composing of lower level of abstraction (output of previous layers) for the construction of higher level of abstraction. For a photo for example, lower abstraction level are the raw pixel, edges, histogram, etc. The higher level of abstraction may be the objects in the photo, even higher is what the person in the photo is doing.

The difference between DL and ESN are clear in terms of the application domain, architecture employed and training methodology.

1. Domain: DL is used in machine learning to discover pattern in the training data in order to allow for highly abstract (compared to raw data) classification or representation such as recognizing a photograph. The environment in these tasks is static, that is "time" is not present in the data. ESN on the other hand is a dynamical system, its use is to approximate temporal nonlinear functions.

2. Architecture: DL use FNN architecture. Although each layer may consists of recurrent Restricted Boltzman Machine (RBM), but the overall flow of signals is only from input to output of the entire network. Furthermore, RBM learns only static function. ESN on the other hand is fully recurrent and learns temporal functions. At present, extending the concepts of DL to recurrent or dynamical networks is still an open question [Bengio, 2009].

3. Training: DL networks are trained layer by layer. When each layer consists of RBM, this can take a long time and is not unimodal. In ESN training consists of solving an overdetermined linear system which is fast and unimodal.

It is true that in some application such as speech classification, both ESN and DL can be used and DL would probably gives better results because there is abstraction (speech feature as input and classification label as output). However in adaptive filtering the data is temporal and there is no abstraction (the output is as "raw" as the input). It is clear that for our purpose here, ESN is neither a subset of, nor is replaceable by DL. A literature search reveals that DL has never been applied to adaptive filtering.

Chapter 2

Performance Comparison between RNN and ESN

In Chapter 1, we argued that ESN can potentially outperform RNN due to their simple, unimodal training. In this chapter, we compare the performance of RNN against ESN in nonlinear system identification, time-series prediction and equalization tasks. The RNN are trained with an algorithm that was shown to be much better than simple gradient descent. Furthermore, we also trained the RNN with a hybrid algorithm that utilizes both global optimization and the aforementioned training algorithm. The performance of ESN is shown to be better than RNN in all problems considered in this chapter. We also show that while ESN are very simple to use, their performance is sensitive to the choice of parameters. The spectral radius and the input scaling in particular significantly effect the performance of ESN. A parameter sweep was conducted for different problems to demonstrate this fact.

2.1 Training of RNN

The first training algorithm proposed for RNN was the RTRL algorithm [Williams and Zipser, 1989]. It is based on direct gradient calculation from the RNN state update and output equations. While this gradient is exact, its complexity is $O(N^4)$, making RTRL limited in applicability to situations where small network sizes would suffice. Back propagation through time [Werbos, 1990] (BPT) is a very popular method to overcome the high complexity of RTRL. By "unfolding"

RNN through time, each time instance that the network has been running becomes another layer in a feedforward network that stretches back to $k = 0$. BPT essentially transforms RNN into (highly) multi-layer FNN, the training of which many algorithms with $O(N^2)$ complexity are available. Obviously, the BPT approach only make sense if the network would not be run for a very long time, otherwise, any computational saving gained from using a $O(N^2)$ algorithm would be lost due to the shear size of the network.

The shortcoming of RTRL had lead to many studies which proposed new algorithms for training RNN [Werbos, 1990; Toomarian and Barhen, 1991; Sun et al., 1992; Schmidhuber, 1992], with lower complexity and better performance than RTRL. The study in [Atiya and Parlos, 2000] showed that these different approaches to RNN training are essentially different ways of solving the same underlying matrix equations. The same study also proposed a new training algorithm now known as APRL that we will apply in this chapter.

Regardless of what training algorithm is used, RNN training suffers from one major problem: the error surface of RNN is highly multi-modal. The study in [Horn et al., 2009] showed that even a single neuron with one input weight and one feedback weight can have local minima. Since the problem they consider was just training of one network to match the weights of another with exactly the same connection architecture, it is likely that the local minima problem would be worse in practical use of RNN, where the structure of the network has no relation to the problem.

The local-minima problem is also well-known for linear IIR filters, for which numerous methods based on GO algorithms had been proposed to overcome it, such as [Chen et al., 2001; Karaboga, 2005; Krusienski and Jenkins, 2004]. A GO based approach to RNN training for nonlinear equalization problem was proposed in [Yuenyong and NISHIHARA, 2013]. However as discussed in Chapter 1, most GO algorithms are heuristic and probabilistic, meaning that convergence to the global minimum point is not guaranteed. Moreover, they are also population-based and the size of the population required for good search capacity grows with the dimension of the parameter space. Because neither gradient-based or GO algorithm can be 100% successful at locating the global minimum on their own, a hybrid approach where a GO algorithm is combined with a gradient-based algorithm is an interesting new way of training RNN.

One such hybrid algorithm for training linear IIR filters based on combining DE and gradient descent was proposed in [Yuenyong and Nishihara, 2013] and was shown in be superior than either algorithm by itself. Extending along this line, in this chapter we apply the same principles as in that work to train RNN using a hybrid APRL-DE algorithm[1], the result of which serves as a baseline which represent the best trained RNN, against which the performance of ESN can be compared to.

2.2 Experiments - Comparing RNN vs ESN

The purpose of this section is to compare the performance of RNN against ESN using three different problems in adaptive filtering: system identification, time-series prediction, and equalization, all of which are nonlinear. RNN are trained using both APRL alone and by using the hybrid APRL-DE algorithm. ESN are trained as described in Section 1.4.2. The measure of performance for system identification and time-series prediction is given by the normalized mean squared error (NMSE), defined by

$$\text{NMSE} = \frac{E[(y(k) - d(k))^2]}{\sigma_d^2}, \tag{2.1}$$

where σ_d^2 is the variance of the desired response. Note that the signals in the above equation are independently generated as "test" signals, different from the input and desired response used to train the networks. As a rule, we exclude the first L_{washout} samples when evaluating the NMSE values, both for RNN and ESN. This is to make sure that the echo states property had been established. The performance measure for the equalization problem is given by the symbol error rate (SER), evaluated by simulating the transmission of 10000 test symbols after the network has been trained.

2.2.1 System Identification

For this problem we take the same NARMA2 system from Chapter 1, reproduced here

$$y(k) = \frac{y(k-1)y(k-2)(y(k-1)+0.25)}{1+y^2(k-1)+y^2(k-2)} + u(k),$$

[1]For the details of the APRL algorithm, please see [Atiya and Parlos, 2000], the details of DE can be found in appendix A.

where the input $u(k)$ is a uniform noise in $[0,\ 0.5]$. We consider offline training with the number of training samples $L = 500$. Four different cases were simulated: RNN trained with APRL and APRL-DE, and two ESN with different parameters (ESN1 and ESN2). The parameters of the APRL algorithm were $\mu = 0.2, \epsilon = 0.001$, taken from [Atiya and Parlos, 2000].

The parameters of the DE algorithm were $F = 0.8, CR = 0.3, NP = 50$, taken from our previous study on training RNN with DE [Yuenyong and NISHIHARA, 2013]. For ESN1, both $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ was sampled from the range $[-1,\ 1]$, the spectral radius was 0.8, the sparsity ρ of $\mathbf{W}$ was 5%, and the input scaling was one. ESN2 had exactly the same parameters except that ρ was 10%.

The RNN had 10 neurons, while the ESN had 20[2]. The initial weights for APRL was initialized from the range $[-1,\ 1]$ and the initial population for APRL-DE was also initialized from the range $[-1,\ 1]$. Both APRL and APRL-DE were run for 50 epochs for each trial with early stopping for the APRL algorithm if the NMSE goes up for 5 consecutive epochs. The simulation in each case was repeated for 20 independent trials. The results are shown in Table 2.1 and Figure 2.1. It can be seen that both ESN1 and ESN2 had lower NMSE than RNN trained with either APRL or APRL-DE, and that changing only one parameter had significant effect on the performance of ESN.

2.2.2 Time-Series Prediction

This task is the prediction of the Mackey-Glass time series [Jaeger and Haas, 2004] that is governed by the following differential equation

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1 + x(t-\tau^{10})} - 0.1x(t), \tag{2.2}$$

where τ is an adjustable integer parameter, which was set to 17 to make the time-series chaotic. The initial condition $x(0)$ was randomly initialized in the range $[0,\ 1]$. To generate a discrete-time sequence, (2.2) was solved numerically and sampled at integer values of time. For the simulations, other than the change of input and desired response, all other parameters were exactly the same as the

[2]It may seems like it's not a fair comparison between RNN and ESN as the later has more neurons. However, in terms of the number of trained parameters, RNN in these simulations have 100, while ESN only have 20.

Table 2.1: Average NMSE over 20 trials for the NARMA2 and MG problems for the four different cases considered.

	APRL	APRL-DE	ESN1	ESN2
NARMA2	0.0358	0.0309	0.0254	0.0118
MG	0.0172	0.0117	0.0046	2.31×10^{-5}

system identification problem. The results are shown in Table 2.1 and Figure 2.1[3]. It can be seen that results follow the same trend as in the NARMA2 problem.

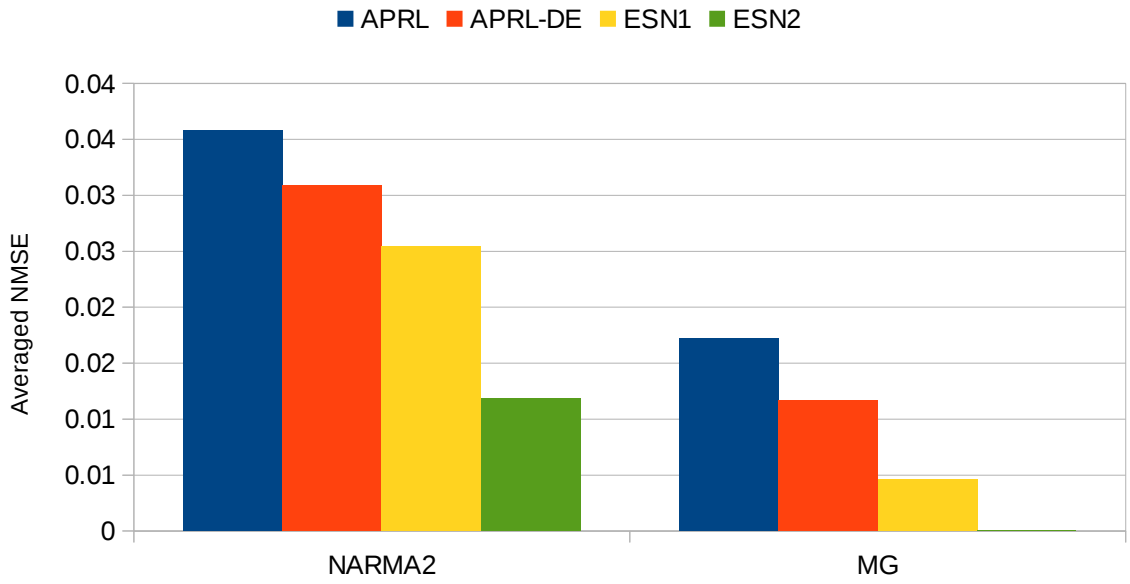

Figure 2.1: Performance of APRL, APRL-DE and the two ESN's for NARMA and MG problems.

2.2.3 Channel Equalization

Channel equalization is another area where the use of recurrent structures give much better performance than FNN [Cha and Kassam, 1995]. For this problem we considered the following non-minimum-phase, nonlinear channel given by [Kechriotis et al., 1994]

$$\begin{aligned} v(k) &= 0.3482u(k) + 0.8704u(k-1) + 0.3482u(k-2) \\ y(k) &= v(k) + 0.2v^2(k), \end{aligned} \tag{2.3}$$

[3]The bar for ESN2 of MG problem is not visible at this scale.

where $u(k)$ is the channel input and $y(k)$ is the output. The delay for the desired response was one. The input signal was a binary alphabet from the set $\{-0.5, 0.5\}$. Following [Kechriotis et al., 1994], we set the number of neurons to be $N = 2$ for RNN. Keeping the same number of neurons for ESN as in the previous two problems seems to be too much different from the RNN size, so the reservoir size was changed to $N = 10$. In terms of the number of trained parameters, RNN now have 8, while ESN have 10. Training length was increased to 2000 samples. After training is completed, 10000 more samples was transmitted in order to measure the SER. The parameters for the training algorithms were the same as the previous two problems. We ran 20 independent trials at each SNR value and plotted the average SER for each as shown in Figure 2.2. At low SNR values, APRL and APRL-DE are indistinguishable, with their SER curves started to deviate at around 12 dB. The SER curves of both ESN1 and ESN2 are below the other two for all SNR values.

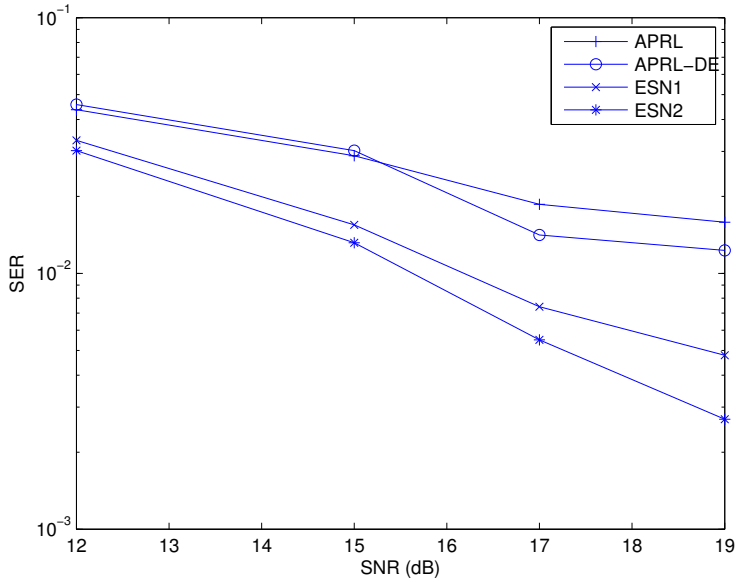

Figure 2.2: Symbol error rate for the equalization problem.

These experiments have shown that ESN have comparable or better performance than the best-trained RNN over different classes of problems, and that changing the sparsity of $\mathbf{W}$ has significant effect on their performance. For the remaining of this chapter, we will demonstrate that the other two parameters: the spectral radius and input scaling, also have significant effect on the performance as well.

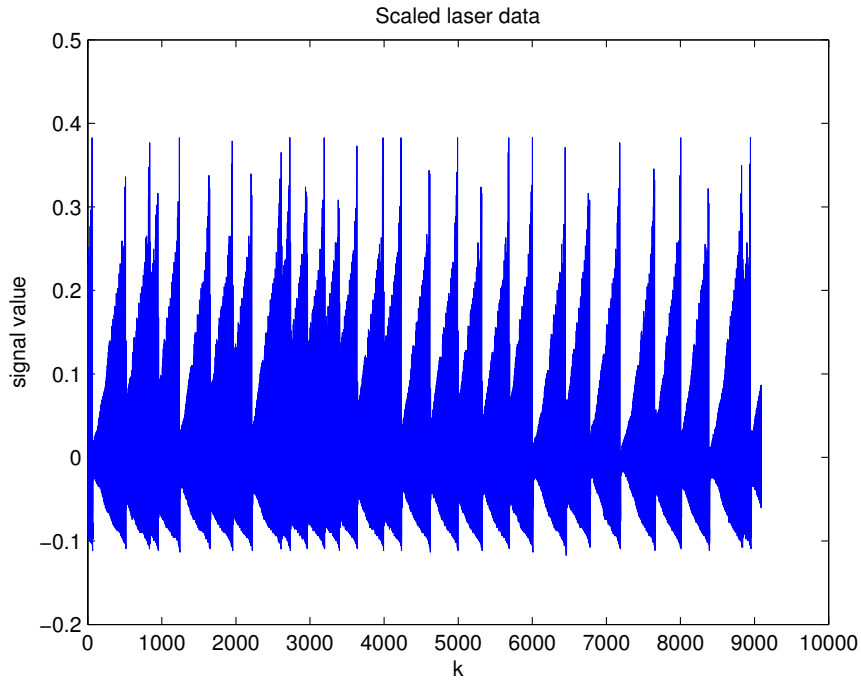

Figure 2.3: Nonlinear laser data.

2.3 Parameter Study for ESN

In order to demonstrate that the performance of ESN is also sensitive to the spectral radius and input scaling, we fixed the sparsity of $\mathbf{W}$ to 5% and sweep the spectral radius in the set $\{0.5, 0.55, ..., 0.95\}$ and the input scaling in the set $\{0.5, 0.55, ..., 1.0\}$. At each grid point in this 2-D parameter space, we repeatedly train and test the networks using four different problems to get the average NMSE over 20 independent trials. The number of neurons was fixed at 10 in all simulations.

Figure 2.4 shows the contour NMSE plot for the NARMA2 and MG problems we considered in the previous section. Figure 2.5 shows the plots of the equalization problem and another problem which is the prediction of nonlinear laser data from [Gershenfeld and Weigend, 1994], shown in Figure 2.3. We scaled the values of the original time-series so that the maximum magnitude is less than one.

For the NARMA2 problem, the contour plot reveals that the performance is sensitive to both the spectral radius and input scaling, the surface of the NMSE plot is highly complex, having disjoint global minimum areas with NMSE=0.01. There are also several local minima where NMSE=0.02. The NMSE corresponding

to the worse possible parameter is 0.04, which is roughly 4 times that of the best possible value.

For the laser problem, the global minimum area is concentrated on the top of the parameter space. Thus generally, larger input scaling yield better performance, and spectral radius has relatively little impact compared to the NARMA2 case. Given a particular input scale, varying the spectral radius yield a difference of only around 0.02 in NMSE value. The difference between the best and worse parameter is less than the NARMA2 case, only around 2 times.

The MG problem has the most complicated surface, and also has the smallest area for the global minimum, which is located at roughly 0.8 for input scaling and 0.9 for spectral radius. Also, the difference between the best and worse parameters is the largest, with the worse parameter giving almost 10 times the NMSE value of the best parameters.

The equalization problem has the simplest surface of all. It also follows roughly the same trend as the laser problem: the performance is better with larger input scaling, with spectral radius having little or no impact at all. Moreover, the difference between the NMSE values of the best and worse parameters is the smallest at 0.08.

From these 4 plots, it can be seen that in some problems, the performance of ESN is highly sensitive to both the input scaling and spectral radius. Whereas in others, the performance is only sensitive to the input scaling. In such problems, the NMSE surface tend to be simpler, with large global minimum area, and the difference between the best and worse parameter are small. In problems with complicated surface, the global minimum areas are smaller, and the difference between the best and worse parameters are much larger.

It is important to keep in mind that the situation here is simplified, since the sparsity of the reservoir is fixed. As seen in the previous section, the value of the sparsity also effects the performance. However even in this simplified case, it can be seen that the performance of ESN, depending on the problem at hand, can be sensitive to the input scaling and spectral radius. Since in practice we would have no prior knowledge which parameter the performance is sensitive to, as well as what the shape of the NMSE surface looks like, it would be desirable to have some method to "adapt" the reservoir parameters in order to ensure the best performance of ESN regardless of the problem under consideration.

2.4 Conclusion

In this chapter, we have shown by experiments on different adaptive filtering problems that ESN have better performance than the best trained RNN. We have also shown by a parameter study that the performance of ESN, depending on the problem at hand, can be quite sensitive to all of the reservoir parameters. The NMSE values achieved by reservoirs with good parameters can be many times less than those obtained by reservoirs with bad parameters. Therefore, how to determine the best parameters given a particular problem, without having to tune them manually by trial and error, is a worthwhile question in ESN. This is one of the main subject matter of the thesis and is presented in the next chapter.

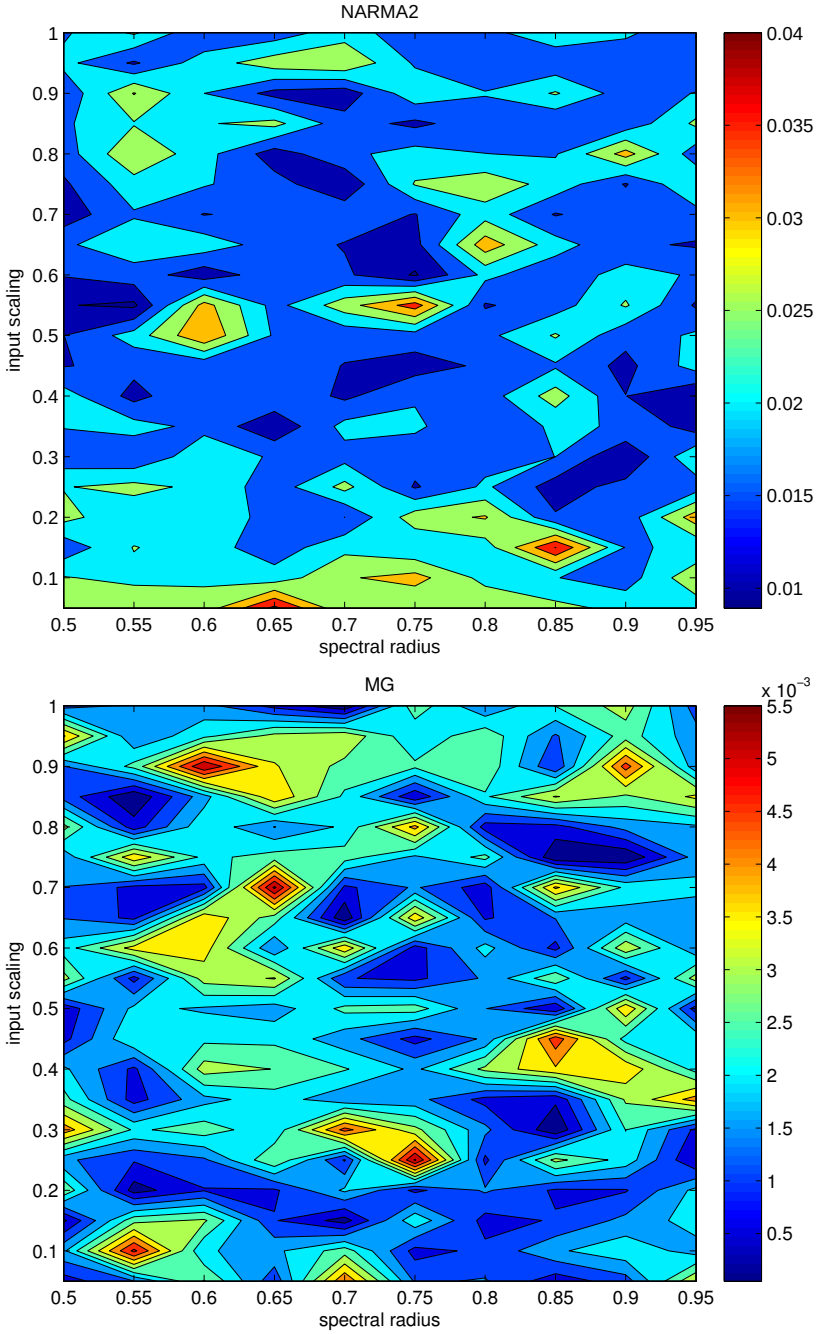

Figure 2.4: NMSE surfaces of the NARMA2 and MG problems.

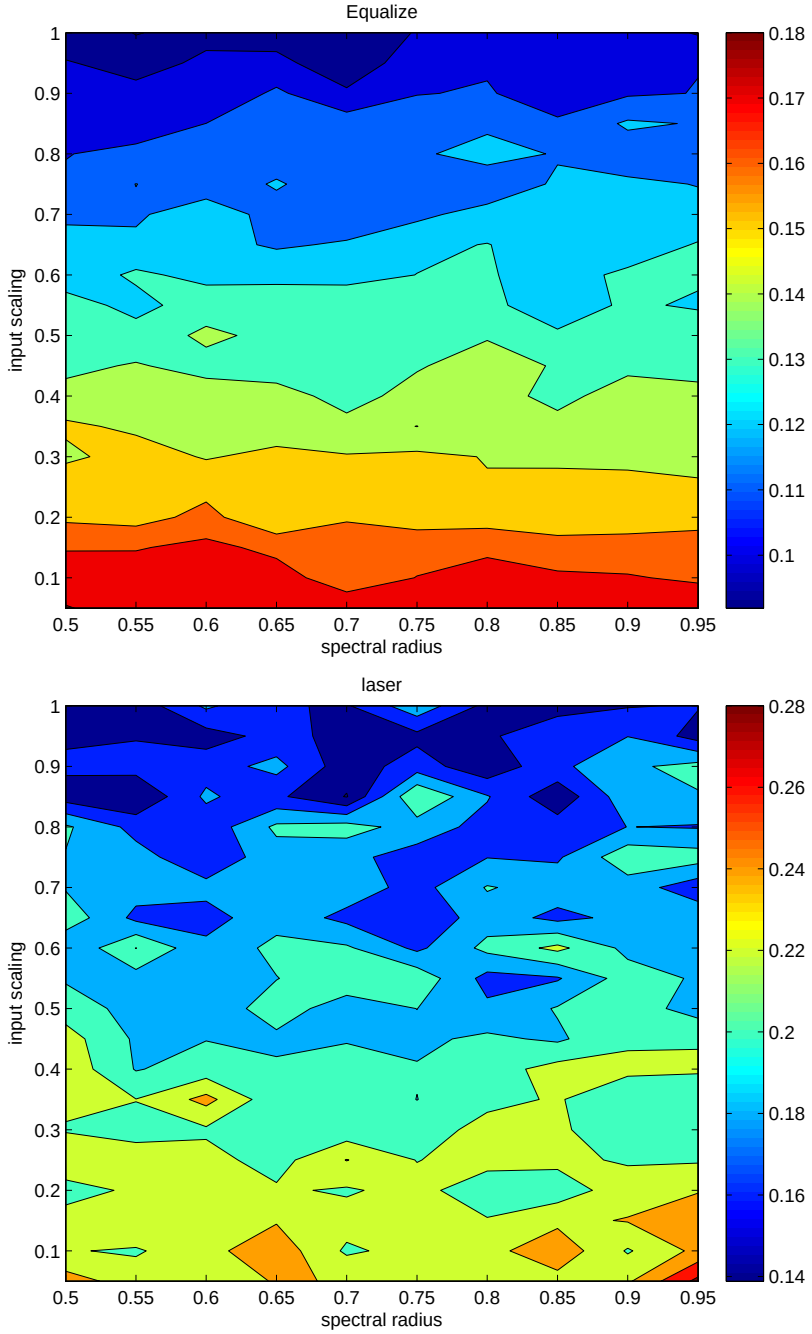

Figure 2.5: NMSE surface of the equalization and laser problems.

Chapter 3

Reservoir Adaptation for ESN

3.1 Toward Reservoir Adaptation

The results in the previous chapter show that although ESN works with randomly initialized reservoir, the final performance is sensitive to the choice of reservoir parameters. We have shown that all parameters involved in reservoir generation: the spectral radius, the reservoir sparsity and the input scaling, can effect the performance of ESN. It would be desirable to have a function that can accurately predicts the performance of a reservoir from its parameters. Ideally, the minimization or maximization of such function should lead to lowest possible MSE when the readout is trained, we call such optimization "pre-training", to distinguish it from "training" of the readout weights. We also say reservoir adaptation to mean the same thing as pre-training.

The obvious choice for a cost function, is to use the MSE itself, or its normalized version NMSE. This is called supervised pre-training. However, in order to evaluate this cost function, the readout must be trained, and then tested. This requires two blocks of signals, one for training of the readout, and one for testing it to compute the MSE or NMSE. This process must be repeated for each point in the parameter space to be evaluated, which means repeatedly solving (1.12). Therefore supervised pre-training has high computational cost.

On the other hand, unsupervised cost functions are computed only from the states of the reservoir, without training the readout. Thus, such functions has to somehow predict a reservoir's performance from its states. And since the states of a reservoir is determined by its parameters and its input signal, optimizing

the cost function is equivalent to optimizing a reservoir's parameters for a given task. Since the readout is never trained, unsupervised pre-training has much lower computational cost than supervised pre-training.

Reservoirs are nonlinear dynamic systems, thus a function defined on their states would be highly multi-modal, with derivative that is difficult, if not impossible, to calculate. Therefore, for the purpose of adapting the reservoir, we use the DE algorithm. This algorithm, and evolutionary algorithms in general, are affected by the randomness of reservoir generation, which results in a different one being generated each time even when exactly the same parameters are used. It was demonstrated in [Ozturk et al., 2007] that reservoirs generated with the same parameters can have significantly different performance.

Since the performance of ESN is linked to the states of the reservoir which is in turn linked to the cost function defined on the states, random generation ultimately acts as noise to the algorithm used to adapt the reservoir and degrades the quality of the final result. Recently, Deterministically Constructed Cycle Reservoir with Regular Jumps (CRJ) was proposed in [Rodan and Tiňo, 2012] and shown to have better performance than classical reservoirs[1] when its parameters are set by brute force supervised pre-training. CRJ reservoirs are completely deterministic, meaning that there is only one possible reservoir that can be generated from a given parameters set. This is preferable for the purpose of optimization than classical reservoirs. In this chapter and the next, will we focus mainly on CRJ reservoirs.

In this chapter, we will first present the CRJ reservoir. Next, we will review existing works on reservoir adaptation, present our pre-training cost function and demonstrate its effectiveness by using it to pre-train reservoirs for different problems. The results will be compared against that of classical reservoirs with sparsity set to 10% and both the spectral radius and input scaling set to the middle of their respective ranges, which is 0.5 for the input scaling and 0.75 for the spectral radius. These parameter setting for the classical reservoirs reflect their actual use where one has no a priori knowledge of the proper parameters. We call this parameter set for classical reservoirs "standard parameters". For the rest of this chapter, whenever we say classical reservoir, we mean classical reservoir with standard parameter.

[1] By "classical reservoirs", we mean those generated as described previously in Chapter 1.

We will then extend the pre-training cost function to the complex domain and compare the performance of pre-trained CRJ reservoirs against classical reservoirs for complex valued time-series prediction problem.

3.2 The CRJ Reservoir

The CRJ reservoir is illustrated in Figure 3.1. It has a clockwise ring topology with one-directional weight r_c connecting adjacent neurons, and bi-directional "jumps" r_j every l neurons. The N input weights are denoted by a vector $\mathbf{v}$ where each element v_i is assigned $-v$ if the i^{th} decimal digit of π is equal to or less than 4 and $+v$ otherwise. The entire CRJ reservoir is completely characterized by 4 numbers: r_c, r_j, l, v. The three parameters r_c, r_j, v take on real values from 0 to 1 while l takes on integer values from 1 to $\lfloor N/2 \rfloor$.

The reason for using the digits of π is to provide a deterministic way of assigning plus or minus signs to v for each of the input weights, so that for the same number of weights, the same sequence of pluses and minuses will be generated every time, in contrast with randomly generated reservoirs, where the signs of the input weights or the weights themselves are sampled from a probability distribution. It is not necessary to use π, any irrational number such as $e, \sqrt{2}$ will also work.

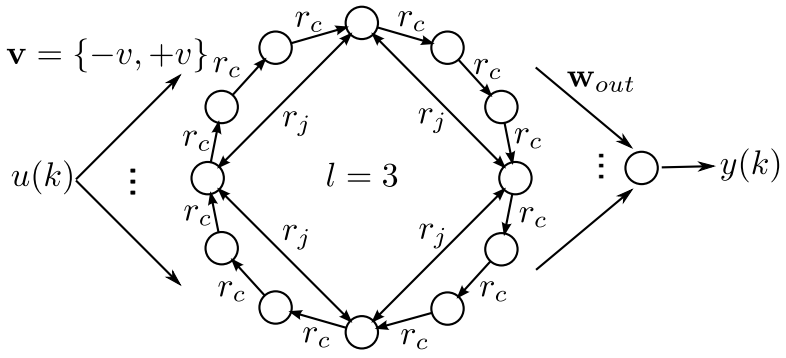

Figure 3.1: A CRJ reservoir with 12 neurons and jump size of 3.

3.3 Predicting Reservoir Performance

In this section we review existing studies on the relationship between reservoir parameters and performance. A method based on *kernel quality* proposed in [Maass et al., 2005] involved feeding the reservoir with k different inputs, and

collecting the resulting states $\mathbf{x}(n_0)$ where n_0 is some fixed time after the input was presented, row-wise into a matrix $\mathbf{M}^{k\times N}$. If the rank of $\mathbf{M}$ equals k then the reservoir is said to have linear separation property. This basically tests if the reservoir is capable of reaching independent states when fed by different input sequences. Thus this method is useful for time series classification tasks.

Various research directions in ESN was briefly outlined in [Jaeger, 2005], most notably the problem of very high eigenvalue spread (EVS) of the covariance matrix formed by the reservoir's states, which is a quantity that ideally should be minimized[2]. However, high EVS seems unavoidable because each neuron in the reservoir is fed by the same input thus their states will always have some dependency on each other. We have tried to minimize EVS by minimizing the condition number of $\mathbf{P}$. While this does lead to a lower (but still very high by linear filter standard) EVS, it only slightly improve the performance of the reservoir. Moreover, decreasing the EVS too much seems to adversely effect the reservoir performance.

Recently [Song and Feng, 2010] studied the relationship between the reservoir connectivity (sparsity) and its performance for time-series prediction tasks. Their criterion, the omega-complexity index, seems to be successful at predicting a reservoir's performance based on its connectivity. But their method applies only to time-series prediction and difficult to apply to CRJ reservoirs where the connectivity is only coarsely adjustable through l.

The study in [Ozturk et al., 2007] proposed viewing the elements of $\mathbf{x}(k)$ of each time instant as samples of an underlying random variable. Their cost function is evaluated by first calculating the Renyi's squared entropy values $H_2(\mathbf{x}(k))$, $k = 1, ..., L$ and then taking the average of the values. This cost function is maximize by first linearizing the ESN state update equation around zero state, and then distribute the poles of the resulting system evenly inside the unit circle. This procedure also has the effect of distributing the eigenvalues of $\mathbf{W}$ evenly in the unit circle as well. However, the eigenvalues of $\mathbf{W}$ for the CRJ reservoir are located *on* a circle with radius less than 1, providing a counter example. Moreover, each element of $\mathbf{x}(k)$ actually belongs to different random variables since they are the outputs of different neurons. Moreover, linearizing around zero means that the

[2]As discussed in the introduction, extremely large EVS is a problem when trying to train ESN online. We will deal with this issue in Chapter 4.

nonlinear power of the reservoir is not taken into consideration, since tanh is very close to linear around zero.

The edge of chaos is a region in the state space of a dynamical system at which it operates at the boundary between chaotic and non-chaotic behaviour. It is often claimed that at the edge of chaos, dynamical systems have high computational power [Bertschinger and Natschläger, 2004]. The edge of chaos is detected by measuring the Lyapunov exponent, a task that is difficult in itself. A method to approximate the Lyapunov exponent of non-autonomous dynamical systems was proposed in [Verstraeten et al., 2007]. Application of this method to classical reservoirs showed that the optimal reservoir coincide with positive Lyapunov exponent, which supports the edge of chaos hypothesis. However, for CRJ reservoirs, the optimal one coincide with negative Lyapunov exponent, when corresponds to non-chaotic behavior [Rodan and Tiňo, 2012]. Moreover, [Verstraeten et al., 2007] did not offer any criterion for optimizing the Lypunov exponent except that it must be positive, since different problems can have different optimal Lypunov exponent, this is not a cost function that can be maximized or minimized.

The study in [Verstraeten, 2009] suggested that the magnitude of the smallest singular value of $\mathbf{P}$ (MSS) is a good predictor of a reservoir's performance, but did not provide any optimization attempt using this criterion. We tested this cost function and found, for the CRJ reservoir, that increasing the MSS does correlate with lower NMSE, but only up to a certain point. Beyond that, larger value of MSS can actually lead to a higher NMSE. Since the optimal value of MSS is different for different problems, it cannot be used for optimizing reservoir parameters.

Finally, it is often stated that the computational power of ESN is maximized when its states are maximally decorrelated or independent [Steil, 2004]. In other words, the columns of $\mathbf{P}$ should be as independent from each other as possible. A reservoir with maximally independent states would give the lowest possible NMSE when the readout is trained. We show by a counter example that this is not necessarily true in Section 3.1.

After reviewing the existing works on the relationship between reservoir parameters and performance, we found that a usable cost function, especially one that can be applied to CRJ reservoirs, is not available. Therefore, we propose one of our own in the next section.

3.4 Proposed Pre-training Cost Function

Our proposed cost function is based on a well-known quantity and is easy to calculate. It is the average mutual information (MI) between each column of $\mathbf{P}$ and the desired response vector $\mathbf{d}$. The idea is that producing the desired response vector $\mathbf{d}$[3] is basically taking a linear combination of each column of $\mathbf{P}$, this can easily be seen in the linear system formulation of ESN training (1.12). Therefore, it is reasonable to assume that in order for the reservoir to perform well, there should be high degree of dependency between the columns of $\mathbf{P}$ and $\mathbf{d}$. This dependency is maximized when the average MI between them is maximum.

MI [Steuer et al., 2002; Estévez et al., 2009] measures the degree of (nonlinear) dependency between two random variables. The state of each neuron and the desired response can be regarded as random variable pairs, where MI can be calculated. The MI between two random variables X and Y is defined by (Here, we follow the convention of denoting random variables by uppercase letters, and we use ";" instead of "," to separate the random variable in order to reserve "," for joint pdf.)

$$I(X;Y) = \int\int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy. \tag{3.1}$$

where $p(x,y)$ is the two-dimensional probability density function (pdf) and $p(x)$ and $p(y)$ are the two one-dimensional pdf's.

In practice MI must be estimated from samples since we have no knowledge of the pdf's. There are many methods available for the estimation of MI with different accuracy and computational cost ranging from simple histogram, adaptive partitioning, kernel density, k-nearest neighbor (KNN), wavelets-based, etc. The study in [Walters-Williams and Li, 2009] surveyed the many methods of MI estimation. It was found that the top three most accurate methods are KNN, B-spline and wavelet-based, and the least accurate one is the histogram.

In general, one would prefer a method with high accuracy, however, the pre-training cost function we shall define below is not based on a single value of MI but an *average* of many MI values. Using a relatively inaccurate method such as the histogram, some MI value will be overestimated, while some will be underestimated. These estimation errors tends to cancel out as the average is

[3] Or, when a nonlinear neuron is used as the readout, the desire response transformed by $\tanh^{-1}$

taken. Furthermore, we are interested in the relative values of average MI between reservoirs with different parameters, not the absolute values, when we pre-train the reservoir. As long as their relative size is correct, the estimation error does not matter. Therefore, for our purpose here, it is not necessary to use a very accurate method for MI estimation. We confirmed this fact by comparing the end results produced by using the histogram, adaptive partitioning, and supervised pre-training using MSE as the cost function. There was no statistically significant difference between the three.

Then, in this work we will use the histogram method because it has the lowest computational cost of all MI estimation algorithms. Histogram based estimation of (3.1) is given by

$$\tilde{I}(X;Y) = \log(L) + \frac{1}{L}\sum_{ij} k_{ij} \log \frac{k_{ij}}{k_i k_j}, \tag{3.2}$$

where k_{ij} is the number of samples that lie within the ij bin of the 2D histogram. Similarly, k_i and k_j are respectively the number of samples that lie with the i and j bins of the 1D histograms. We used $L = 500$ sample pairs for the estimation in (3.2) in order to compromise between low variance of the estimation while maintaining acceptable computational cost. The optimal number of bins of the histograms depends on the pdf's which we are trying to estimate [Moddemeijer, 1989]. For this reason, we must settle for a good guess for the number of bins. For the 1D histograms, we set the number of bins to $\left\lceil \sqrt{L} \right\rceil$, and for the 2D histogram, we set it to $\left(\left\lceil \sqrt{3}L \right\rceil\right)^2$. These values were obtained from the default setting of a MI toolbox [Brown et al., 2012].

We are now ready to define our pre-training cost function which we denote as J

$$J(r_c, r_j, v, l) = \frac{\sum_{i=1}^{N} \tilde{I}(X_i; D)}{N}, \tag{3.3}$$

where X_i denote the random variable that is the state of each neuron in the reservoir and D is the desired response. The cost function $J(r_c, r_j, v, l)$ can be evaluated as follows

The proposed cost function can be evaluated as follows:

1. Generate a CRJ reservoir with the parameter set $\{r_c, r_j, v, l\}$, as described in Section 3.2.

2. Feed the reservoir with the input signal, update the states according to (1.2).

3. Store the states for each time instant k row-wise into $\mathbf{P}$. and each desired response sample into the elements of $\mathbf{d}$. Repeat 2-3 until all L samples have been used.

4. Discard the top 200 rows of $\mathbf{P}$ as the initial washout period, in order to make sure that the echo state properties is established.

5. Calculate MI values between each columns of $\mathbf{P}$ and $\mathbf{d}$ using (3.2).

6. Take the average of the MI values, the result is the cost function value $J(r_c, r_j, v, l)$.

An advantage of using MI to construct the cost function is that it should be robust to noise. In order to show that, we will use the chain rule of MI which states that

$$I(X; Y, Z) = I(X; Z) + I(X; Y|Z), \tag{3.4}$$

where X, Y, Z here are just some dummy random variables.

In the system identification configuration, where the desired response is corrupted by an independent additive measurement noise. The MI between this corrupted desired response and the states of ESN can be written by considering this corrupted desired response as a joint random variable of the true desired response and the noise

$$I(\text{observed plant output}, X_i) = I(D, \epsilon; X_i), \tag{3.5}$$

where ϵ is the noise random variable, D is the true desired response and D, ϵ denote the joint random variable of desired response and noise.

By using the chain rules of mutual information, we can rewrite the previous equation as

$$I(D, \epsilon; X_i) = I(D; X_i) + I(\epsilon; X_i|D). \tag{3.6}$$

Since the true desired response has no effect on the states of ESN, we can drop the "given D" from the last term and write

$$I(D; X_i) + I(\epsilon; X_i|D) = I(D; X_i) + \cancelto{0}{I(\epsilon; X_i)} = I(D; X_i), \tag{3.7}$$

where in the end we used the fact that the MI between some random variable and noise must be zero, since noise contains no information. This shows that the MI is the same in both noise-free and noisy case. Similar arguments can be made for

the equalization configuration, by moving the noise term to the state variable X_i, since the effect of noisy input to the states can be modeled as additive noise to the true state variable.

The situation in the time-series prediction configuration is a little more complicated, since both the input and the desired response are corrupted by noise. Since the desired response is a one step ahead sample and the noise added to the original time-series is assumed to be white, we can model the added noises to the input and desired response as two independent noise variables. That is, the MI between the noisy state variable and the noisy desired response can be written as the MI between joint random variables

$$I(D, \epsilon_1; X_i, \epsilon_2), \tag{3.8}$$

where D and X_i are respectively the noise-free desired response and state of neuron i and ϵ_1, ϵ_2 are respectively the noise random variable added to the clean desired response and clean state.

We can show that $I(D, \epsilon_1; X_i, \epsilon_2) = I(D; X_i)$ by applying the chain rule of MI twice. The first application gives

$$\begin{aligned} I(D, \epsilon_1; X_i, \epsilon_2) =& I(D; X_i, \epsilon_2) + I(D; \epsilon 1 | X_i, \epsilon_2) \\ =& I(D; X_i, \epsilon_2) + \cancelto{0}{I(D; \epsilon_1)} \end{aligned}, \tag{3.9}$$

where we can change $I(D; \epsilon 1 | X_i, \epsilon_2)$ to $I(D; \epsilon_1)$ because ϵ_1 is independent from the joint random variable X_i, ϵ_2 and of course the MI between anything with a noise is zero.

By apply the chain rule of MI to the remaining term $I(D; X_i, \epsilon_2)$, we get

$$\begin{aligned} I(D; X_i, \epsilon_2) =& \cancelto{0}{I(D; \epsilon_2)} + I(D; X_i | \epsilon_2) \\ =& I(D; X_i) \end{aligned}, \tag{3.10}$$

where we can replace $X_i | \epsilon_2$ by X_i because the true state is not dependent on the noise term. Thus we have shown that J is noise robust for the time-series prediction configuration as well.

3.4.1 Maximizing The Cost Function

In order to pre-train a reservoir, we have to maximize J with respect to the four parameters $\{r_c, r_j, v, l\}$. It can be seen that the evaluation of J involves estimation

of MI using (3.2 for which the derivative is not defined. Thus a gradient-based method is not applicable to maximize J. A derivative-free method is needed for this task. We used the DE algorithm, even though the required population size may be larger than some other evolutionary algorithm, for example CMAES [Hansen, 2011], because it can easily handle mixed-integer problems, as in this case since l takes on only integer values. Using a derivative free method is the norm when dealing with reservoirs [Ishu et al., 2004; Bush and Tsendjav, 2005; Jiang et al., 2008].

We set the two parameters of DE to $F = 0.8$, $CR = 0.8$ and the population size $NP = 15$. The maximum number of iterations to run is 100. The original DE/rand/1 version was used without any modification. There are more advanced version of DE such as [Lin et al., 2000; Huang et al., 2007], but since the problem at hand here is only 4 dimensional and static, we decided to use the original version of DE, to keep things as simple as possible.

DE is a minimizing algorithm, but our problem here is that of maximization, therefore we minimize $1/J$ instead, so that the algorithm does not need to be modified. The formal formulation of the reservoir pre-training problem is therefore

$$\min_{r_c, r_j, v, l} \left(\frac{1}{J} \right). \tag{3.11}$$

In using any evolutionary algorithm, a search domain for each parameter being optimized needs to be defined. Since the CRJ reservoir already places limits on the values of its parameters (see Section 3.2), we can specify the same limits to the DE algorithm. The search domain is therefore $[0, 1]$ for r_c, r_j, v and 1 to $\lfloor N/2 \rfloor$ for l.

The search domain could have been made smaller by constrained optimization since certain combinations of r_c, r_j produce reservoirs whose spectral radius is higher than 1, violating the condition for the existence of echo states. Such reservoirs will not work so the corresponding region in the parameter space need not be searched. However, calculating the spectral radius involves solving the eigenvalues problem for large matrices which is an expensive operation. Fortunately J seems to take on small values for such reservoirs, so there is no need to implement the constraint on the values of the spectral radius into the optimization of J.

3.5 Experimental Results

For the first part of the experimental results, we compared CRJ reservoirs pre-trained with our cost function versus classical reservoirs of the same size with standard parameters. CRJ Reservoirs are pre-trained using 500 samples of input and desired response. Then to evaluate the performance, the readouts are trained and tested using different blocks of 2000 samples. The final performance measure is the average of NMSE over 30 independent trials. For the first three problems, we shall employ state vector augmentation as in (1.21) to illustrate that pre-training is beneficial even if state vector augmentation is used.

In order to show that the difference between the mean of NMSE values reported is statistically significant, we calculated the p-values using Welch Two Sample t-test.

3.5.1 System Identification and Time-Series Prediction

The first problem is the identification of a tenth order system (NARMA10)

$$d(k+1) = 0.3d(k) + 0.05d(k)\left[\sum_{i=0}^{9} d(k-i)\right] + 1.5u(k-9)u(k) + 0.1, \quad (3.12)$$

where the input $u(k)$ is a uniform random sequence in the range $[0, 0.5]$. The plots showing the output and desired response of the classical reservoir and our pre-trained reservoir is shown in Figure 3.2. It can be seen that the output of the pre-trained CRJ reservoir matches the desired response more closely than that of the classical reservoir. The average NMSE of pre-trained CRJ reservoirs are 0.0476, 0.0443 and 0.0360 respectively for reservoir of size 100, 200 and 300. The average NMSE of classical reservoirs are 0.1266, 0.0916 and 0.0830 for the same sizes.

This problem presents us with an opportunity to provide a counter example to the claim that a reservoir with maximally independent states will give the lowest possible NMSE, which was discussed in Section 3.3. To quantify the notion of maximally independent, we use the average of pair-wise MI between each neuron in the reservoir, i.e., the MI between each pair of the columns of $\mathbf{P}$, we call this quantity Average Pair-wise MI (APMI). Since MI measures degree of dependence, a reservoir with maximally independent states has the lowest APMI.

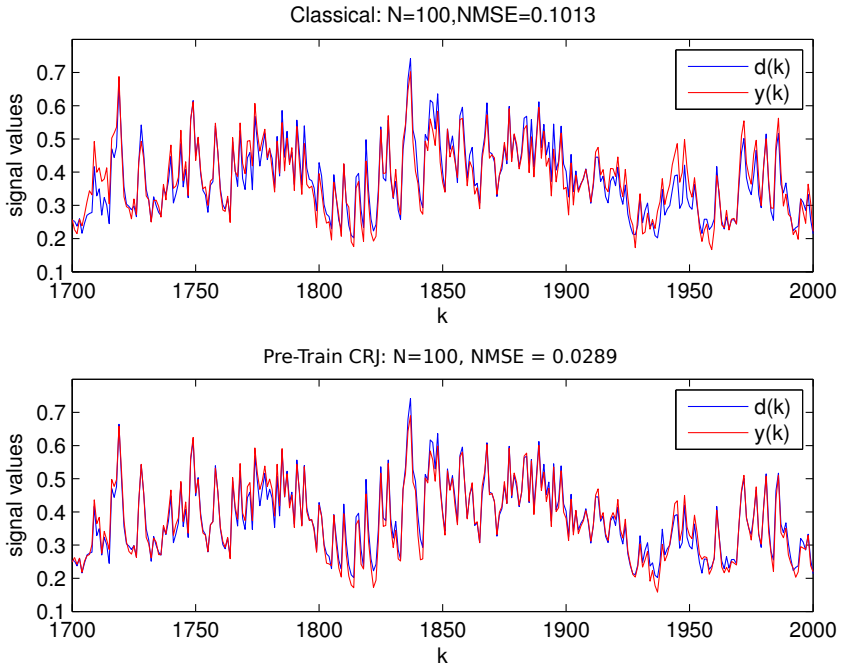

Figure 3.2: One trial of last 300 samples of the test output and desired response of the classical and pre-trained CRJ reservoirs for the NARMA10 problem with $N = 100$.

For a counter example, take a reservoir of size $N = 100$ with parameters

$$\{r_c, r_j, v, l\} = \{0.8882, 0.5247, 0.05, 33\}\,.$$

This reservoir has APMI of 0.0880, but the average NMSE of this reservoir is 0.1146. On the other hand, our pre-trained reservoir for this problem has APMI of 0.1529 but it has average NMSE of 0.0443. This shows that a reservoir with higher degree of independence between its states is not necessarily the best performing one.

Next, for the time-series prediction problem we use the same laser time-series introduced in Chapter 2. We do not show the plot for for this problem, because due to low value of NMSE it would be very difficult to discern the difference between the classical reservoirs and pre-trained CRJ ones. The average NMSE of pre-trained CRJ reservoirs are 0.0072, 0.0069 and 0.0066 respectively for reservoir of size 100, 200 and 300. The average NMSE of classical reservoirs are 0.0088, 0.0078 and 0.0077 for the same sizes. Table 3.1 summarize the results for the NARMA10 and laser problems. It can be seen that the pre-trained CRJ achieved

lower NMSE than the classical reservoirs in all cases, the difference is especially large for the NARMA10 problem.

3.5.2 Memory and Nonlinear Mapping Task.

The next problem is memory and nonlinear mapping. The task is to reconstruct a delayed and nonlinear versions of the input signal:

$$y_{p,d_l}(k) = \text{sign}[\beta(k-d)] \times |\beta(k-d_l)|^p, \tag{3.13}$$

where $\beta(k-d_l)$ is the product of two delay successive inputs,

$$\beta(k-d_l) = u(k-d_l) \times u(k-d_l-1), \tag{3.14}$$

and the input $u(k)$ is a uniform noise in range $[-0.8, 0.8]$. The parameter d_l controls the amount of delay, and p controls the amount of nonlinearity. Since there are extra parameters in this case, the reservoir size is limited to only $N = 100$ for the experiments. We pre-trained the CRJ reservoir using the signals generated by setting $p, d_l = 3$ and used the obtained parameters for simulations with other values of p, d_l. The results are shown in Table 3.3. It can be seen that the pre-trained CRJ reservoir performed better than the classical one for all values of d_l and p and especially for larger values of d_l.

Table 3.1: Average NMSE results for NARMA10 system identification and laser prediction problems.

		$N = 100$	$N = 200$	$N = 300$
NARMA10	classical	0.1266	0.0916	0.0830
	pre-trained CRJ	0.0476	0.0443	0.0360
laser	classical	0.0088	0.0078	0.0077
	pre-trained CRJ	0.0072	0.0069	0.0066

Table 3.2: P-values of the result presented in Table 3.1.

	$N = 100$	$N = 200$	$N = 300$
NARMA10	1.913×10^{-10}	6.799×10^{-7}	3.914×10^{-7}
laser	2.824×10^{-11}	2.790×10^{-5}	2.550×10^{-5}

Table 3.3: NMSE results for the delayed-nonlinear mapping problem. $N = 100$.

$p \backslash d_l$		2	3	4	5	6
2	classical	0.2240	0.2813	0.3689	0.5726	0.8968
	pre-trained CRJ	0.1489	0.1639	0.1786	0.2159	0.4177
3	classical	0.4214	0.4830	0.5395	0.7338	0.9188
	pre-trained CRJ	0.3392	0.3717	0.3824	0.4136	0.5737
4	classical	0.5662	0.6379	0.7115	0.8220	1.0068
	pre-trained CRJ	0.5018	0.5395	0.5489	0.5755	0.6889
5	classical	0.6870	0.7531	0.7945	0.8951	1.0243
	pre-trained CRJ	0.6286	0.6730	0.6727	0.6938	0.7957
6	classical	0.7242	0.9150	0.9605	1.0054	1.0582
	pre-trained CRJ	0.7158	0.7774	0.7774	0.7830	0.8826

3.5.3 MIMO System Identification

In this problem, we consider identification of the following MIMO system described by the nonlinear state equations:

$$\begin{aligned} x_1(k+1) &= 0.5x_1^{2/3}(k) + 0.3x_2(k)x_3(k) + 0.2u_1(k) \\ x_2(k+1) &= 0.5x_2^{2/3}(k) + 0.3x_3(k)x_1(k) + 0.5u_1(k) \\ x_3(k+1) &= 0.5x_3^{2/3}(k) + 0.3x_1(k)x_2(k) + 0.5u_2(k), \\ y_1(k+1) &= 0.7(x_1(k+1) + x_2(k+1)) \\ y_2(k+1) &= 1.5x_1^2(k+1) \end{aligned} \tag{3.15}$$

where x_1, x_2, x_3 are the three states of the system, u_1, u_2 are in the inputs and y_1, y_2 are the two outputs. The input are again randomly generated from the range $[0\ 0.5]$.

CRJ reservoirs were proposed only for the single input case, to apply it to this problem, we extend it to the MIMO configuration. Assume that the number of

Table 3.4: P-values of the result presented in Table 3.3.

$p \backslash d_l$	2	3	4	5	6
2	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$	2.30×10^{-15}	5.37×10^{-15}
3	3.68×10^{-10}	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$	1.00×10^{-15}	$< 2.2 \times 10^{-16}$
4	5.30×10^{-10}	3.45×10^{-15}	7.96×10^{-16}	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
5	1.06×10^{-4}	1.49×10^{-6}	4.24×10^{-14}	2.00×10^{-15}	$< 2.2 \times 10^{-16}$
6	2.16×10^{-5}	3.26×10^{-6}	3.36×10^{-8}	2.32×10^{-13}	3.06×10^{-14}

inputs is Q and that the number of outputs is R. We change the input vector $\mathbf{v}$ in the SISO case into a matrix $\mathbf{W}_{\text{in}} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_Q]$ where Q is the number of inputs. Each of the $\mathbf{v}_i$ is generated in exactly the same way as $\mathbf{v}$ in the SISO case, with exception of each one having its own scalar parameter. Thus, MIMO CRJ reservoir are characterized by $\{N, r_c, r_j, v_1, ..., v_Q, l\}$. The MIMO classical reservoir has an extra input scaling parameter from the SISO case considered thus far, for which we again use the middle of the range 0.5.

On the output side, the output weights is now represented by the matrix $\mathbf{W}_{\text{out}}$ instead of the vector $\mathbf{w}_{\text{out}}$. Each row i of $\mathbf{W}_{\text{out}}$ is now the output weights vector of the i output.

In RNN, the MIMO configuration requires that the MSE be redefined to be a total error, which must take into account all of the R errors and all elements of $\mathbf{W}_{\text{out}}$ must be adapted together. In ESN, since the reservoir is not adapted during readout training, the total error need not be defined and each of the R readouts can be trained independently from each other. In other words, the adaptation of the weights of a readout has no effect to other readouts. Therefore, MIMO readouts can be trained simply by solving (1.12) R times, each using a different desired response ($\mathbf{P}$ is of course the same). The obtained output weights vectors can then simply be pasted into the rows of $\mathbf{W}_{\text{out}}$.

Since the R readouts are independent from each other, the pre-training cost function J can also be easily extended to the MIMO case by averaging the MI over different outputs as well as different neurons. That is

$$J(r_c, r_j, v_1, ..., v_Q, l) = \frac{\sum_{r=1}^{R} \sum_{i=1}^{N} \tilde{I}(X_i; D_r)}{RN}. \tag{3.16}$$

Thus, in principle at least, the proposed pre-training cost function can be extended

Table 3.5: Average NMSE over 30 independent trials for the MIMO system identification problem.

	NMSE on y_1	NMSE on y_2	sum of NMSE
classical	0.1129	0.1109	0.2238
pre-trained CRJ	0.0579	0.0863	0.1442

to as many outputs as desired. The drawback is that the number of MI values that must be estimated grows with the number of outputs.

We compare the classical reservoir with standard parameters against a CRJ reservoir pre-trained with the above MIMO cost function. The number of neurons for both reservoir types was 20. Training and testing of the readouts was done using blocks of 2000 samples. The CRJ reservoir parameters obtained by pre-training was $\{r_c, r_j, v_1, v_2, l\} = \{0.5455, 0.3678, 0.2265, 0.05, 2\}$. The result is shown in Table 3.5. The average NMSE is individual channels, as well as the sum of both channel, are lower for the pre-trained CRJ reservoir than for the classical reservoir. The p-value of the y_1 channel is 1.80×10^{-4} and the p-value of the y_2 channel is 1.25×10^{-4}. We did not calculate the p-value of the sum of NMSE.

3.5.4 Channel Equalization

In this section, we consider the same channel equalization problem from Chapter 2. As in the previous problem, we compared the pre-trained CRJ reservoir against classical reservoir with standard parameters. The reservoir size remained at $N = 20$.

We pre-trained the reservoir once using signals from the 12 dB SNR environment, and used the obtained parameters for the simulations at all SNR values. We simulated 30 trials at each SNR value and calculated the average SER over 10000 symbols for each. The SER results of this experiment is shown in Figure 3.3. At all SNR values, the SER of the pre-trained CRJ reservoir is lower than that of the classical one. As expected, the higher the SNR, the larger the difference becomes. The p-values at different SNR environments are as follows: 1.74×10^{-8} at 12 dB, 8.01×10^{-12} at 15 dB, 2.81×10^{-12} at 17 dB and 1.50×10^{-11} at 19 dB. The success of the pre-training algorithm in this problem also demonstrated that our cost function still works in a noisy environment as was mathematically shown in

Section 3.4.

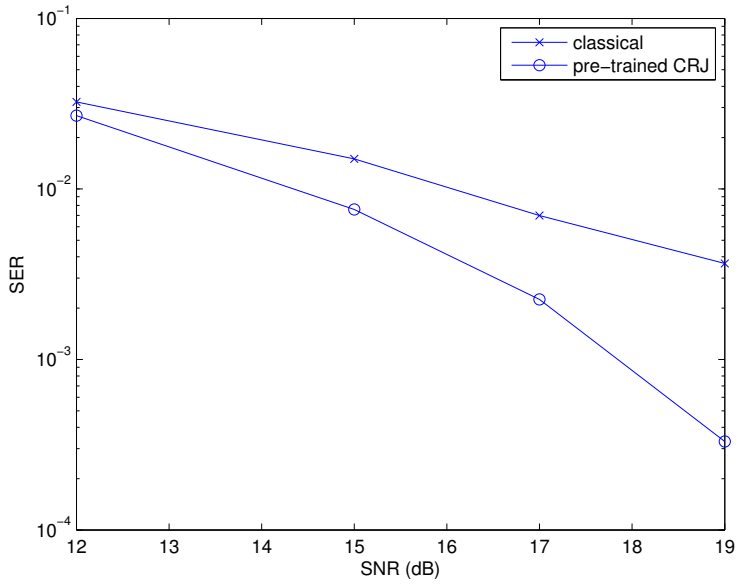

Figure 3.3: Symbol error rates for classical and pre-trained CRJ reservoirs.

In order to conclude this section, we summarize all the CRJ parameters discovered by pre-training in Table 3.6 and 3.7.

3.6 Pre-Training in The Complex Domain

In this section, we investigate the use of ESN in the complex domain and also extend the pre-training cost function to the complex domain as well.

Just like real valued linear filters can be extended to the complex domain, ESN can work in the complex domain as well. However, compared to the real value case, complex valued ESN are rare in the literature. At the time of this writing, there are only three studies that investigate ESN in the complex domain [Seth et al., 2007; Xia et al., 2008, 2011].

These studies make ESN complex valued by simply making $\mathbf{W}$ and $\mathbf{W}_{\text{in}}$ complex valued and using complex value algorithms to train the readout instead of the real valued version. The results in the studies showed that ESN work just as well in the complex domain. However, none of them give theoretical foundation for complex valued ESN the same way as [Jaeger, 2001] did for the real valued case.

Table 3.6: CRJ parameters discovered by pre-training for the NARMA10, laser, and mapping problems.

Problem	$N = 100$	$N = 200$	$N = 300$
NARMA10	$r_c = 0.7200$	$r_c = 0.7211$	$r_c = 0.6085$
	$r_j = 0.5304$	$r_j = 0.3890$	$r_j = 0.4723$
	$v = 0.0501$	$v = 0.3220$	$v = 0.0534$
	$l = 18$	$l = 9$	$l = 6$
laser	$r_c = 0.4970$	$r_c = 0.5014$	$r_c = 0.4974$
	$r_j = 0.3235$	$r_j = 0.1073$	$r_j = 0.2150$
	$v = 0.9994$	$v = 1.0$	$v = 1.0$
	$l = 41$	$l = 44$	$l = 43$
mapping	$r_c = 0.6681$	$r_c = \text{N/A}$	$r_c = \text{N/A}$
	$r_j = 0.3277$	$r_j = \text{N/A}$	$r_j = \text{N/A}$
	$v = 0.2103$	$v = \text{N/A}$	$v = \text{N/A}$
	$l = 8$	$l = \text{N/A}$	$l = \text{N/A}$

Table 3.7: CRJ parameters discovered by pre-training for the equalization and MIMO problems.

Problem	$N = 20$
Equalization	$r_c = 0.0296$
	$r_j = 0.0473$
	$v = 0.9987$
	$l = 4$
MIMO	$r_c = 0.9882$
	$r_j = 0.4420$
	$v_1 = 0.0583$
	$v_2 = 0.0120$
	$l = 1$

In this section, we try to give such theoretical foundation by extending the materials in [Jaeger, 2001] to the complex domain. In order to do this we utilized the isomorphism between real and complex number to extend the properties of echo states given in the introduction to the complex domain. This lay the foundation for complex valued ESN, which is currently missing from the literature.

3.6.1 Proofs of Echo States Definitions

Recall the three definitions of echo state in Chapter 1. In what follows, we will explain the proof of each definition in an intuitive manner.

The proofs are based on the set $D\{(\mathbf{x}, \mathbf{x}')\}$ of state pairs that are compatible[4] with some input sequence $\mathbf{u}^{\infty}$ and the Euclidean distance between state vectors $d(\mathbf{x}, \mathbf{x}')$. By definition of echo state, these state pairs must be the same, such that D contains only identical pairs $(\mathbf{x}, \mathbf{x}')$. Intuitively, if two networks both have echo states, then when fed by the same input sequence, they must eventually reach the same states as $k \to \infty$.

The first proof is that "state contracting mean echo states", is done by contradiction. Assume that the networks have no echo state but are state contracting. Then there exists $(\mathbf{x}, \mathbf{x}') \in D$ such that $d(\mathbf{x}, \mathbf{x}') > \delta$ when two networks are fed with $\mathbf{u}^{\infty}$. This means that if there are no echo states, the networks can reach different states such that $\mathbf{x} \neq \mathbf{x}'$ when fed by $\mathbf{u}^{\infty}$. But this contradicts the state contracting property which assert that the state difference is always contracting, which means that $d(\mathbf{x}, \mathbf{x}')$ must eventually become smaller than any δ.

The second proof is "state contracting means state forgetting". This is established by linking to the first proof. Assume that the networks are not state forgetting. This means that there exist some right-hand-infinite input sequence $\mathbf{u}^{+\infty}$ such that for two networks with difference current states $\mathbf{x}, \mathbf{x}'$, $d(T(\mathbf{x}, \mathbf{u}^{+\infty}), d(T(\mathbf{x}', \mathbf{u}^{+\infty})) > \delta$. That is, the effect of the difference in current states $\mathbf{x}, \mathbf{x}'$ does not vanish as $k \to \infty$. This violates the state contracting property. Therefore, if the network is state forgetting, it must also be state contracting and has echo states.

The third proof is "input forgetting means echo states". Assume that the networks do not have echo states. Then there exists an input sequence $\mathbf{u}^{-\infty}$ end-compatible with states $\mathbf{x}, \mathbf{x}'$ such that $d(\mathbf{x}, \mathbf{x}') > \delta$, which violates the input

[4]Recall that compatible means that the state can be reached when the network is fed by that input sequence.

forgetting property itself. Thus if the networks are input forgetting, they must also have echo states.

It can be seen that the proofs above are based on the set D containing state pairs expressed as vectors of real number, and the Euclidean distance between two real vectors $d(\mathbf{x}, \mathbf{x}')$. Therefore, if we can express the state vector of a complex reservoir as a vector of real numbers, show that the Euclidean distance between a pair of complex vectors is the same as their "real expression", and that the state sequence produced by a complex ESN and its real expression is the same, then the proofs above extend automatically to the complex domain without any modification.

3.6.2 Isomorphism of State Update

Our tool in expressing a complex reservoir as a real one is the isomorphism between $\mathbb{C}^N$ and $\mathbb{R}^{2N}$. Reproduced here for convenience is the state update equation from Chapter 1,

$$\mathbf{x}(k) = f\left[\mathbf{W}\mathbf{x}(k-1) + \mathbf{W}_{\text{in}}\mathbf{u}(k)\right]. \tag{3.17}$$

Assume that the matrices and vectors in (3.17) are complex-valued and that f is defined by

$$f = \tanh\left[\Re(\mathbf{c})\right] + j\tanh\left[\Im(\mathbf{c})\right], \tag{3.18}$$

where $\Re()$ and $\Im()$ are respectively real and complex value operators. The vector $\mathbf{c}$ is just a dummy vector.

The key is to express matrix to vector multiplication in the complex domain in terms of real matrix and vector, this is achieved by the following isomorphism where $\mathbf{C}$ is a complex valued matrix

$$\begin{aligned} \mathbf{C}_{\text{real}} &= \begin{bmatrix} \Re(\mathbf{C}) & -\Im(\mathbf{C}) \\ \Im(\mathbf{C}) & \Re(\mathbf{C}) \end{bmatrix}, \\ \mathbf{z}_{\text{real}} &= \begin{bmatrix} \Re(\mathbf{z}) \\ \Im(\mathbf{z}) \end{bmatrix} \end{aligned} \tag{3.19}$$

where the subscript "real" explicitly denote real matrices and vectors. Using (3.19), an $N \times 1$ complex state vector can be written as an $2N \times 1$ real vector. Furthermore, using f in (3.18) and the isomorphism above, we can rewrite the complex valued state update (3.17) as a real valued one

$$\mathbf{x}_{\text{real}}(k) = f\left[\mathbf{W}_{\text{real}}\mathbf{x}_{\text{real}}(k-1) + \mathbf{W}_{\text{in,real}}\mathbf{u}_{\text{real}}\right]. \tag{3.20}$$

This works because the product $\mathbf{Cz}$ is exactly the same, up to isomorphism with $\mathbf{C}_{\text{real}}\mathbf{z}_{\text{real}}$, and choice of f means that the same result will be produced, again up to isomorphism whether f is applied to a complex vector or to a real vector defined by (3.19), because the tanh function is applied independent to the real and imaginary parts anyway.

With this expression, one can simply replace the complex valued $\mathbf{u}(k), \mathbf{x}(k), \mathbf{W}$ and $\mathbf{W}_{\text{in}}$ by their corresponding real valued ones with twice the dimensions. The state sequence produced will be the same, up to the isomorphism as if complex valued quantities are used. Therefore, the ESN dynamic will remain the same, and we can replace state pairs $\mathbf{x}, \mathbf{x}'$ of the proofs in the previous section with $\mathbf{x}_{\text{real}}, \mathbf{x}'_{\text{real}}$ without effect the mechanics of the proofs since the dimensions of the state vectors played no role.

The next task is to show that the distance between two complex vectors and that between their two corresponding real vectors as defined by (3.19) are the same. That is

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{x}_{\text{real}}, \mathbf{y}_{\text{real}}) = d\left(\begin{bmatrix}\Re(\mathbf{x})\\ \Im(\mathbf{x})\end{bmatrix}, \begin{bmatrix}\Re(\mathbf{y})\\ \Im(\mathbf{y})\end{bmatrix}\right), \tag{3.21}$$

where $\mathbf{x}$ and $\mathbf{y}$ here denote some complex vectors. In order to show this, we will show that the norm squared of the distance between the two complex vectors and that between their real expressions are the same.

Proof Let the subscript R and I denote respectively the real and imaginary parts. The normed square distance between the real vectors is

$$\begin{aligned} d^2\left(\begin{bmatrix}\Re(\mathbf{x})\\ \Im(\mathbf{x})\end{bmatrix}, \begin{bmatrix}\Re(\mathbf{y})\\ \Im(\mathbf{y})\end{bmatrix}\right) &= (y_{1_R} - x_{1_R})^2 + \cdots + (y_{L_R} - x_{L_R})^2 + \\ &\quad (y_{1_I} - x_{1_I})^2 + \cdots + (y_{L_I} - x_{L_I}) \quad , \\ &= \sum_{k=1}^{L} y_{k_R}^2 - 2x_{k_R}y_{k_R} + x_{k_R}^2 + y_{k_I}^2 - 2x_{k_I}y_{k_I} + x_{k_I}^2 \end{aligned} \tag{3.22}$$

and that of the complex vectors is

$$
\begin{aligned}
d^2(\mathbf{x},\mathbf{y}) &= (\mathbf{y}-\mathbf{x})^H(\mathbf{y}-\mathbf{x}) \\
&= \sum_{k=1}^{L} \left[y_{k_R} - x_{k_R} - j(y_{k_I} - x_{k_I})\right]\left[y_{k_R} - x_{k_R} + j(y_{k_I} - x_{k_I})\right] \\
&= \sum_{k=1}^{L} y_{k_R}^2 - y_{k_R}x_{k_R} + jy_{k_R}x_{k_I} - jy_{k_R}x_{k_I} \\
&\quad - x_{k_R}y_{k_R} + x_{k_R}^2 - jx_{k_R}y_{k_I} + jx_{k_R}y_{k_I} \\
&\quad - jy_{k_I}y_{k_R} + jy_{k_R}x_{k_I} + jy_{k_I}x_{k_R} - jx_{k_I}x_{k_R} \\
&\quad + y_{k_I}^2 - y_{k_I}x_{k_I} - y_{k_I}x_{k_I} + x_{k_I}^2 \\
&= \sum_{k=1}^{L} y_{k_R}^2 - 2x_{k_R}y_{k_R} + x_{k_R}^2 + y_{k_I}^2 - 2x_{k_I}y_{k_I} + x_{k_I}^2 = (3.22). \quad \blacksquare
\end{aligned}
\tag{3.23}
$$

Therefore the Euclidean distance is unchanged by the isomorphism in (3.19).

We have shown that both complex state vectors and the distance between them can be expressed in terms of real quantities without effecting the dynamics of ESN. In other words, an real ESN based on the isomorphism and the corresponding complex ESN fed with the same input will produce exactly the same state sequence. And since the Euclidean distance remain unchanged by the isomorphism, the echo states property as well as its three equivalent definitions hold. Moreover, the proofs in the previous section rely only on state vectors pairs and the Euclidean distance between them, therefore they apply to the equivalent real vectors constructed from the complex ones using (3.19). It follows logically then that the theory of ESN applies in the complex domain as well.

An issue when using complex ESN is how to ensure echo state property. The easy check that the spectral radius must be less than one only applies to the real ESN with standard tanh activation, the reason for that was explained in back in Section 1.4.1. It does not work, for example with the true complex tanh defined by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{3.24}$$

because it contains singularity. For most complex activation functions surveyed in [Özdemir et al., 2011], if we were to try to follow the same proof as in (1.7), it would fail because there is no way to guarantee that the distance between the state vector pair after the activation function have been applied is less than or equal

to that before the application of the activation function because the function is not bounded by one. However, f in (3.18) applied to real numbers *is* bounded by one. Since we have already shown that the state sequence produced by a complex ESN and it isomorphic real one are the same, this means that the isomorphism allows us to extend the proof in (1.7) to the complex domain when the activation function is (3.18).

3.6.3 Complex-Valued CRJ Reservoir

Next, we extend the CRJ reservoir to the complex domain by making r_c, r_j complex valued. The input scaling v can be left as a real number and of course the jump length l must be a real number. The complex valued CRJ reservoir is then characterized by 6 real numbers: $\Re(r_c), \Im(r_c), \Re(r_j), \Im(r_j), v, l$.

In order to pre-train this complex reservoir, we must also extend the cost function (3.3) to the complex domain by defining the complex valued MI. There are several ways in doing so, such taking the MI of the real part only, or taking the sum of real MI of the real and imaginary parts. But since we have already defined real vectors from complex ones in (3.19), the same isomorphism can be used to define the complex MI as

$$\tilde{I}_{\text{complex}}(X;Y) = \tilde{I}\left(\begin{bmatrix}\Re(X)\\ \Im(X)\end{bmatrix}, \begin{bmatrix}\Re(Y)\\ \Im(Y)\end{bmatrix}\right). \tag{3.25}$$

Thus the complex version of (3.3) is

$$J(\Re(r_c), \Im(r_c), \Re(r_j), \Im(r_j), v, l) = \frac{\sum_{i=1}^{N} \tilde{I}_{\text{complex}}(X_i; D)}{N}. \tag{3.26}$$

3.6.4 Complex-Valued Time-Series Prediction

In order to test (3.26) we performed an experiment on complex-valued time-series prediction. The time-series to be predicted is the real-world wind vector data [Mandic, 2012] which contains wind magnitudes and directions expressed as complex numbers. We compare our pre-trained complex-valued CRJ reservoir against classical complex-valued reservoir with standard parameters. The wind data is shown in Figure 3.4. We consider reservoir size of $N = 100$. The training and testing signal lengths were both 2000 samples. Pre-training was again done with

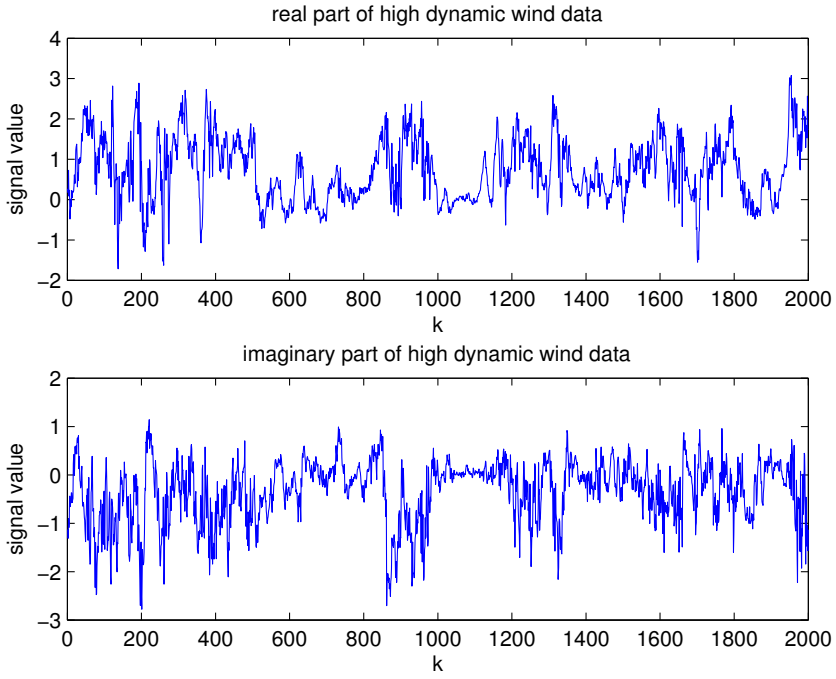

Figure 3.4: High dynamic wind time-series.

500 samples. Here, the performance are measured by the ensemble average of the standard prediction gain p

$$p = 10 \log_{10} \frac{\sigma_u^2}{\sigma_e^2}, \tag{3.27}$$

where σ_u^2 and σ_e^2 are the variances of the input signal and the prediction error, respectively. The complex CRJ parameters discovered by the pre-training algorithm are

$$\{r_c, r_j, v, l\} = \{0.1027 + 0.0121j, 0.0742 + 0.0089j, 0.2962, 10\}\,.$$

Classical complex reservoir with standard parameters achieved an average prediction gain of 7.0783 dB for this problem, while pre-trained complex CRJ reservoir got 7.4281 dB. The p-value is 3.90×10^{-11}. The result shows that our pre-training method is also effective in the complex domain.

3.7 Computational Cost

Now we present the computational cost of our method to give an idea how much computation is involve in pre-training. In order to do so we measure the time pre-training takes for reservoir of different sizes. The simulations are performed

on a PC with 2.93 Ghz core i7 processor and 4 GB of RAM. The environment is MATLAB 2013a. The number of samples pair used for pre-training was 500. Figure 3.5 shows the time taken to complete pre-training for different reservoir sizes from 20 to 200 using MATLAB and C++. We used the Armadillo library [Sanderson, 2010] which provides matrix vector multiplication as well as many MATLAB-like functionality in C++ .

It can be seen that the computational cost seems a little high, however, to put this in perspective, the time it takes to do supervised pre-training with $N = 100$ under the same conditions using MATLAB is 311.62 seconds[5].

ESN generally have large reservoirs, therefore even the C++ implementation can take a significant amount of time. This can potentially be reduced by considering the nature of the cost function. It may not be necessary to calculate the MI between every pair of neuron states and the desired response. It is the average of the MI values that is being maximized, not individual values. As is well known in statistics, the sample mean approaches the true population mean as the sample size increases. Therefore, it should be possible to evaluate J sufficiently accurate enough for optimization by only considering a subset of neurons in the reservoir. In other words, to calculate the MI values only for some neurons N' which can probably be much smaller than N for large reservoirs.

Another source of speedup may come from the algorithm itself. Profiling our code, we found that most of the time is spent in the histogram function, which at first seems strange since forming the histogram is just counting and involve very little calculation. However, we found that the memory access pattern of the arrays that hold the histogram values is not contiguous, i.e., there are big jumps in what element of the arrays is accessed. Non-contiguous memory access is well-known to have a large overhead. Thus, optimizing the memory access of the histogram routine can lead to significant speedup.

3.8 Conclusion

We start this chapter by introducing the CRJ reservoir and discussed why, for the purpose of pre-training, it is better to use a deterministic reservoir than a

[5]Using the DE algorithm, brute force search as in [Rodan and Tiňo, 2012] would take even longer.

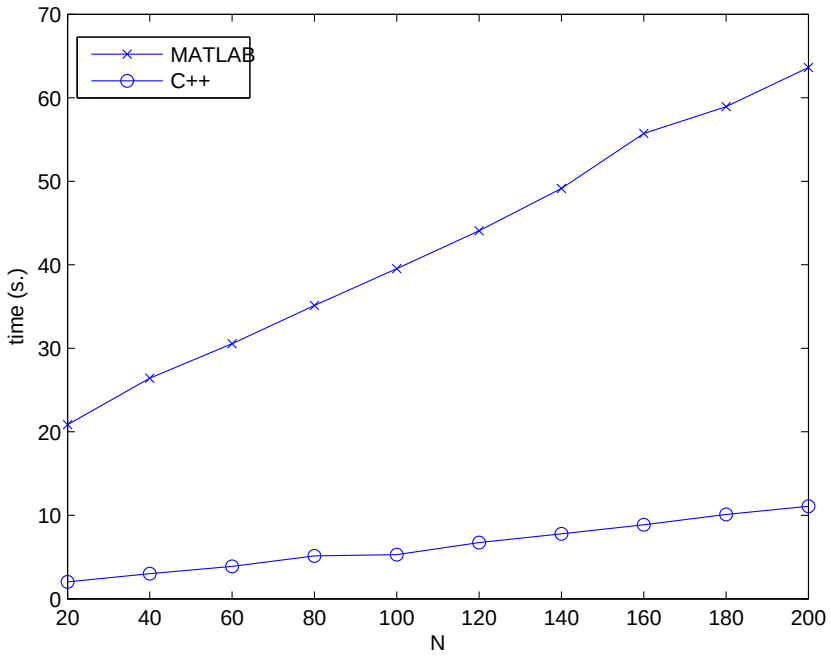

Figure 3.5: Time taken for pre-training with different reservoir sizes.

classical one. Then we review existing works on prediction of a reservoir's performance based on its parameters and showed by eliminating existing works one by one that a cost function that can reliably related a reservoir's parameter to its performance that can be used as a optimizable cost function is still missing. Next, we proposed the new cost function for pre-training that is based onthe mutual information between the states of each neuron and the desired response. This is a semi-supervised cost function in a sense that the desired response is utilized, but training of the readout is not performed. We showed mathematically that the proposed cost function is noise-robust, something that is very important for adaptive filtering applications. We then conducted many experiments to illustrate the effectiveness of pre-training using the proposed cost function. Reservoirs pre-trained with our method have better performance than classical reservoirs with standard parameters in almost all cases.

The theory of ESN was then extended to the complex domain by utilizing the isomorphism between real and complex number to express complex matrix vector multiplication using real matrices and vectors. Then the CRJ reservoir and the pre-training algorithm was extended to the complex domain. Complex time-series prediction experiment was conducted to verify the effectiveness of complex-valued

pre-training. Finally, the chapter ended with discussion on the computational cost of the algorithm and suggest come possible improvements for further research.

Chapter 4

Online Readout Training

In this chapter, we shift our attention to online training of ESN readout. Since in the off-line case the dominant training algorithm is ordinary least squares, the natural algorithm to use for the online case is the Recursive Least Squares (RLS) algorithm. The readout weights $\mathbf{w}_{\text{out}}$ can be adapted online using RLS as if they are taps of a linear filter. However, the complexity of RLS is $O(N^2)$ per sample, and for large reservoirs the computational cost of using RLS to train the readout can be very high.

For linear filters, the least mean squares (LMS) algorithm [Widrow and Stearns, 1985] is the workhorse. Its complexity is $O(N)$, it is very easy to implement and free from numerical issues like the RLS algorithm. Its drawback however is that it is sensitive to the eigenvalue spread of the error surface. When the eigenvalue spread is large, LMS produces large misadjustment, and its convergence along the small eigen-directions can be prohibitively slow [Haykin, 2005]. As shown by the example in Chapter 1, the covariance matrix formed by the neuron states in ESN has extremely large eigenvalue spread, on the order of 10^8 or more, using LMS for online training of ESN readout leads to unsatisfactory performance.

This extremely large eigenvalue spread problem is well known in the ESN literature [Schrauwen et al., 2007; Jaeger, 2005]. It can be reduced, somewhat, by injecting noise into the state update equation as in (1.20). As was discuss in Section 1.5, it is common practice in ESN to add noise during training, but the purpose of doing so is to avoid very large values of weights when the readout is trained, getting lower eigenvalue spread is merely a by product. Adding too much noise can negatively effect the performance of ESN. In fact, the size of η in (1.20)

is a trade of between performance and small output weights. From experience, we found that adding a reasonable level of noise, with η in (1.20) around 0.0001-0.001, significantly reduce the magnitude of the trained readout weights with minimal impact on performance. This level of noise however does not reduce the eigenvalue spread enough, but if one tries to reduce the eigenvalue spread further by making η too large, the performance will be effected too much that it makes no sense to do so.

An attempt to properly reduce the eigenvalue spread using Hebbian [Gerstner and Kistler, 2002] and Anti-Hebbian learning [Plumbley, 1993; Hebb, 2005] on $\mathbf{W}$ is reported in [Jaeger, 2005] but it was not successful. In this study we have also tried to reduce the eigenvalue spread by minimizing the condition number of $\mathbf{P}$ using the DE algorithm. In doing so we were able to reduce the eigenvalue spread by several orders of magnitudes, but it was still well above 10^6. Moreover, the performance of ESN was negatively effected even when offline training was used. At present it seems that very high eigenvalue spread is something that is unavoidable when training the readout online, as trying to reduce it too much means sacrificing performance too much that any gain to be had from smaller misadjustment is nullified.

Despite being a major problem, in the ESN literature online training had received relatively little attention, due to the fact that the majority of applications of ESN are in offline settings [Skowronski and Harris, 2007; Sacchi et al., 2007], or are online but the sample rate is not so high so computational cost is not a problem [Plöger et al., 2004; Salmen and Ploger, 2005]. However in adaptive filtering, low cost online training is a must. In this chapter we will investigate ways to reduce the computational cost in readout training, not by deriving a new algorithm, but by reducing the number of taps that must be trained by the RLS algorithm given a certain reservoir size. We believe that this is the right approach given the state of the ESN literature at present and from our own investigation.

4.1 Effect of Eigenvalue Spread on Readout Training

In this section, we investigate how serious of a problem this high eigenvalue spread is for readout training. For the test problem, we use the same NARMA10 system

from Chapter 3.

$$d(k+1) = 0.3d(k) + 0.05d(k)\left[\sum_{i=0}^{9} d(k-i)\right] + 1.5u(k-9)u(k) + 0.1,$$

where the input $u(k)$ is a uniform noise in [0, 0.5]. We employed classical reservoir with standard parameters and $N = 100$. We train the readout online using $L = 1000$ and evaluate the NMSE over the last 100 samples. For the training algorithms, we used the standard versions of NLMS, VLLMS[Kamenetsky and Widrow, 2004] and RLS. The parameters of NLMS was set as $\mu = 0.5$ and $\epsilon = 0.1$. The parameters for VLLMS are set the same as in the reference, except for μ, which we set at the maximum stable value of 0.1. The parameters[1] of RLS was set at $\lambda = 1$ and $\epsilon = 0.0001$. Setting the forgetting factor to 1 is justified because the problem under consideration here is stationary. We average the NMSE produced by the three algorithms over 100 trials to compare their performance.

We are concerned with the eigenvalues of the covariance matrix formed by the reservoir states $\mathbf{x}$, calculated using time average of L samples given by

$$\mathbf{\Phi} = \frac{1}{L}\sum_{k=1}^{L} \mathbf{x}(k)\mathbf{x}(k)^T$$

From Chapter 1, the eigenvalue spread of this problem is on the order of 10^{16}. The three algorithms produced NMSE's as follows: 0.6670 for NLMS, 0.6520 for VLLMS and 0.2813 for RLS. The training squared error curves are shown in Figure 4.1. The top panel compares NLMS vs RLS and the bottom panel compares VLLMS vs RLS. It can be seen that MSE curve of RLS lies below both of NLMS and VLLMS.

From this example, it is clear that the extremely large EVS make LMS type algorithms ineffective compared to RLS for online readout training. This is not an issue for relatively problems where the error produced by NLMS is already low enough, or problems that requires small reservoir sizes that one can just use RLS. It becomes a problem when the reservoir size is large and the performance specification calls for RLS, as it may be unacceptably costly. From here on, we assume that RLS is required, and try to reduce the computational cost by reducing the number of trained taps.

[1]When we are talking about the RLS algorithm, λ denotes the forgetting factor, not an eigenvalue.

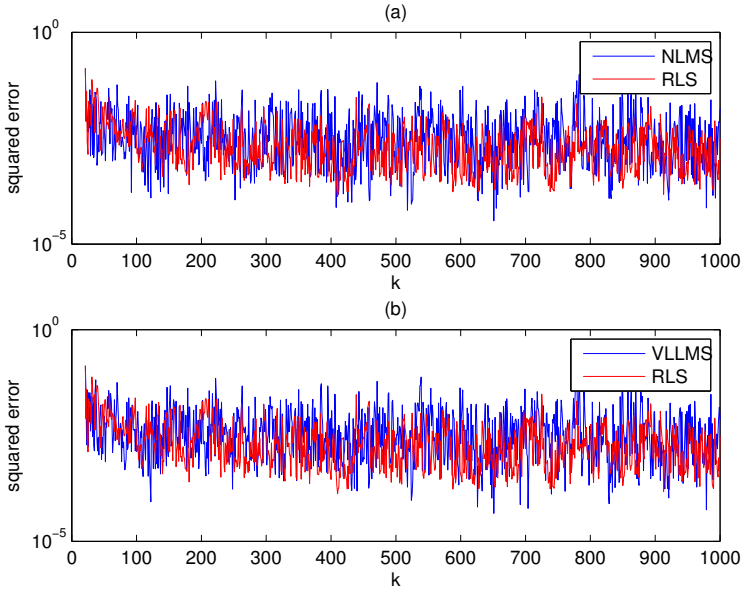

Figure 4.1: (a) Comparison of MSE curves between NLMS and RLS, (b) between VLLMS and RLS.

4.2 Backpropagation Decorrelation Learning

Before we investigate ways to reduce the number of taps that must be trained by the RLS algorithm, we first look at a training algorithm that has $O(N)$ complexity that is well known in both RNN and ESN literature. This is to see if there exists an $O(N)$ algorithm that can train the readout online well.

The study in [Steil, 2004] proposed a RNN training algorithm based on simplification of the APRL algorithm and an observation that under adaptation using APRL of a SISO network, the weights going into the output neuron changes rapidly, while the weights of the rest of the network changes much more slowly. The idea that lead to the derivation of this method is to simplify APRL using instantaneous correlation matrix instead of the full one and to use rank one adjustment instead of actual matrix inverse. The resulting algorithm called Backpropagation Decorrelation (BD) is then applied only to the output weights in RNN. This similarity between the principles of BD and ESN is interesting and begs the question whether BD could be used for readout training. In fact, in the ESN literature BD is considered an option for online readout training. So in this section we investigate it to see what kind of performance it has, and whether or not it is really a suitable method.

BD was derived for fully connected RNN. As shown in Figure 4.2, if we consider the output neuron alone, then the rest of the RNN network can be considered as the reservoir, much that same way as in ESN. Thus if we ignore the sparsity difference between ESN and RNN then a BD structure of size $N+1$ is structurally the same as ESN structure of size N with output feedback connections. This is reason why BD is considered to be applicable to ESN readout training.

Applying BD to train ESN readout had also been considered in [Küçükemre, 2006]. That study compared various versions of RLS algorithms, along with BD and LMS algorithm, for training of the readout under two system identification problems and one noise cancellation problem. ESN structures with feedback were used throughout, hence, BD was directly applicable. It was found that, for all problems considered, BD was no better LMS for online training.

We have also conducted our own experiment to investigate the performance of BD using the same problem as in Section 4.1. The experiment yielded final NMSE of 0.6695 when the readout was trained using BD. Comparing this to the values in the previous section, it can be seen that our result agrees with those from [Küçükemre, 2006].

From these results, we conclude that it is probably more promising to try to reduce the number of taps to be trained rather than trying to derive an $O(N)$ algorithm for online readout training. In other words, due to the huge eigenvalue spread of $\mathbf{\Phi}$, there is probably no alternative to RLS for good online readout training, since it is a natural counterpart to the least squares method in offline training and both LMS and BD were shown here to be insufficient. What we can do then is to try to reduce the number of taps that the RLS algorithm has to work with.

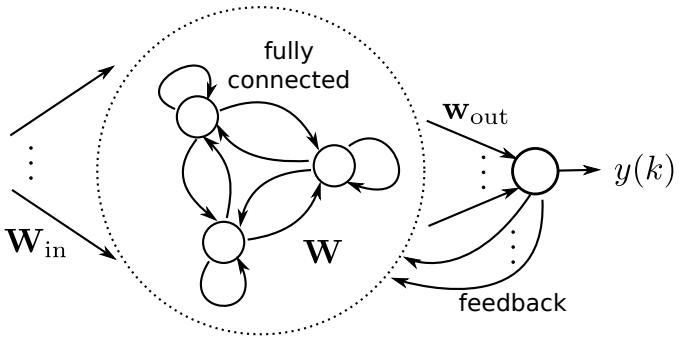

Figure 4.2: The network structure used by Backpropagation Decorrelation.

4.3 Extreme Learning Machine As Readouts

In the ESN literature, there are some works that explored using FNN as readout [Bush and Anderson, 2005; Babinec and Pospíchal, 2006]. The focus of these research had been to improve the performance of ESN, since FNN are much more powerful than linear or single nonlinear neuron readouts. However as it is well known, training of FNN face the same challenges as training of RNN which makes the result suboptimal. Because of this, often FNN readouts give inferior or only slightly increased performance than simple linear combination, not enough to justify the added complexity

For our purpose of reducing the computational cost of online readout training, we can also employ FNN as readouts. At first, this may seem to do the exact opposite, however, by using a particular type of FNN, and the right number of neurons in such network, the computational cost can actually be lower than training linear readouts with RLS, given a particular reservoir size.

We will employ a particular type of feed-forward neural network called Extreme Learning Machines (ELM) proposed by [Huang et al., 2006]. It is a counterpart of ESN for feed-forward networks. ELM are single-hidden-layer feed-forward networks, depicted in Figure 4.3. The input weights on the left hand side of the hidden neurons are fully connected, randomly initialized and fixed, only the output weights on the right hand side are adapted. The output neuron usually uses linear activation function, so it can be considered as just a linear combiner. The adaptation of the output weights can be done offline using least squares, or online using RLS just like the output weights of ESN. ELM have been proven to be a universal function approximator and had been utilized in many application where normal feed-forward neural networks would be used with improved results [Li et al., 2005; Huang et al., 2012]. This is because, like ESN, the outputs of ELM are linear function of the output weights, so their training is optimal and unimodal. This property eliminate the shortcomings of normal feed-forward networks, therefore, ELM can be used as readouts of ESN without destroying the advantage of using ESN in the first place. Moreover, the complex valued case is covered, since ELM is also available in the complex domain [Li et al., 2005].

The computational gain to be had by using ELM as readouts can be understood by inspecting Figure 4.4. The output weights $\mathbf{w}_{\text{out}}$ now belongs to the output network, with the number of taps equal to the number of neurons M in the ELM.

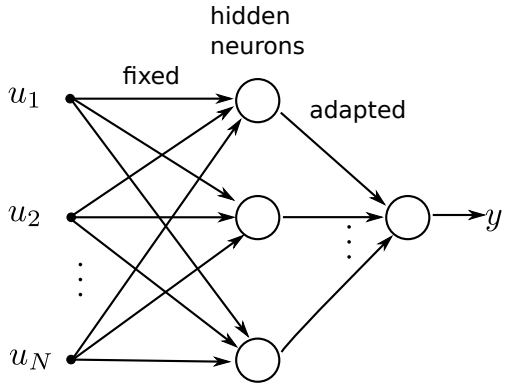

Figure 4.3: Extreme Learning Machine architecture.

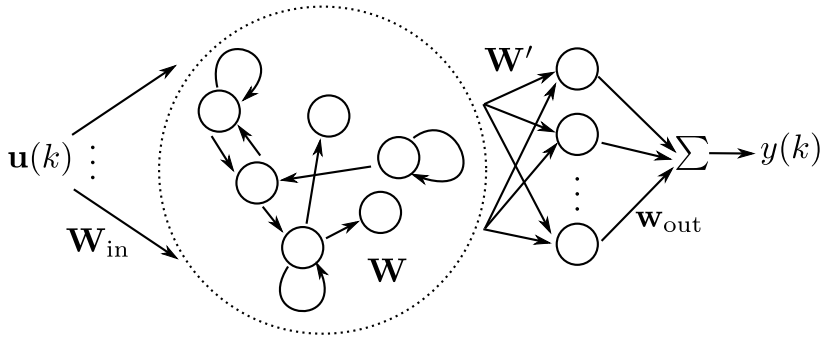

Figure 4.4: ESN combined with ELM readout.

Thus, if the required M for acceptable performance is significantly less than the number of neurons in the ESN reservoir N, then the number of taps that must be adapted by the RLS algorithm is significantly decreased, theoretically without sacrificing any performance since ELM can approximate any linear readout.

4.3.1 Theory of ELM

Assume ELM with M neurons and a single output, the output $y(k)$ can be written as

$$\sum_{i=1}^{M} w_i f(\mathbf{w}_i' \mathbf{u}(k) + b_i) = y(k), \tag{4.1}$$

where $\mathbf{u}(k)$ is the input vector, $\mathbf{w}_i'$ is the i$^{\text{th}}$ row of the input weight matrix $\mathbf{W}'$, f is the activation function which we shall also use tanh, b_i is the i$^{\text{th}}$ bias weight and w_i are the output weights. Now consider training samples of length L, the goal is to try to approximate the desired response as closely as possible for each

sample k

$$\sum_{i=1}^{M} w_i f(\mathbf{w}_i' \mathbf{u}(k) + b_i) = d(k), \quad k = 1, ...L, \tag{4.2}$$

which can be written as an overdetermined system of linear equations

$$\mathbf{H}\mathbf{w}_{\text{out}} = \mathbf{d}, \tag{4.3}$$

where

$$\mathbf{H} = \begin{bmatrix} f(\mathbf{w}_1' \mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_N' \mathbf{x}_1 + b_N) \\ \vdots & \cdots & \vdots \\ f(\mathbf{w}_1' \mathbf{x}_L + b_1) & \cdots & f(\mathbf{w}_N' \mathbf{x}_L + b_N) \end{bmatrix} N \times L. \tag{4.4}$$

One can see the similarity in form between (4.3) and (1.12) for offline ESN training. They are both based on the same principle of combining the states of the neurons to approximate the desired response as closely as possible in the least square sense. This means that the output weights of ELM can be trained online using RLS, just like when linear readout is used. Therefore, the training simplicity of ESN is not compromised by using ELM readouts, the same RLS algorithm can be used, the difference is that it is now working on the output weights of ELM instead of the weights of a linear readout.

The key to reducing the computational cost is the number of neurons needed in ELM readouts to produce acceptable results. Just like ESN needs relatively large reservoir compared to RNN to perform the same task, ELM also needs relatively large number of hidden neurons compared to traditionally trained feed-forward networks. However we are optimistic here since the computational cost of readout training with ELM is $O(M^2)$, even when M is a significant portion of N , something like $N/2$, the computation saving will be a factor of four thanks to the square. Thus, we expect this approach to be effective at reducing the computation cost of online readout training. In the next section, we will conduct experiments to demonstrate this. We will use ELM structure with no bias connections, the activation function of the hidden neurons is tanh and the output layer is just a linear combiner.

4.4 Experiments - Large Reservoirs

For the experiments, we will compare linear readouts trained using RLS and ELM trained with exactly the same RLS algorithm with the same parameters. We will

consider all real-valued problems we have seen where the use of large reservoir size would make sense, and we will consider both regular state vectors and augmented ones. For the mapping problem, since there are many possible parameter choices, we choose a moderately difficult parameter with p, d in (3.13) equals to 3 and 5 respectively, we call this particular setting "mapping35". In all cases we fixed the parameters of RLS to be $\lambda = 1$ and $\epsilon = 0.0001$. From here on, we refer to linear readout trained with RLS simply by "RLS", and we refer to ELM readout, also trained by RLS, as "ELM", to avoid confusion as both types of readouts are trained by the same algorithm. We trained the readout for 2000 samples, then stop training and evaluate the NMSE using another 1000 samples. All reported NMSE values are averages of 100 independent trials. We randomly initialize the input weights of ELM in the range $[-1\ 1]$. We used the CRJ reservoir, where the reservoir size in all cases is $N = 200$, the reservoir parameters were taken from Table 3.6, except for the MG problem where we perform a fresh pre-training to get the parameters. For each problem, we incrementally increase M, with the starting value of $M = 0.1N$. This is increased in increments of 10 until $M = 0.25N$. We consider the "acceptable" level of performance of ELM readouts to be when the average NMSE produced by ELM is no more than 10% in excess of those produced by RLS.

The result for the non-augmented case is shown in Table 4.1. When $M = 40$, the ELM readouts were able to achieved acceptable NMSE for all problems. The interesting point is that, the NMSE of ELM can actually become lower than that of RLS. The most notable case is the laser problem where even with a small $M = 20$, the NMSE value of ELM is already lower than that of RLS by 20.8%. At $M = 30$, ELM also surpass RLS for the MG and mapping35 problems. If M is increased further than 40, the same trend holds for the NARMA10 problem, for example at $M = 50$, the NMSE value produced by ELM for the NARMA10 problem is 0.1312. That the NMSE of ELM can become lower than RLS is due to the fact that ELM's are more powerful than linear readout.

The laser problem seems to response very well, much more than other problems, to ELM readout. The reason for this may be because ELM is a nonlinear readout and ESN had been shown to have troubles with approximating static, highly nonlinear function [LukošEvičIus and Jaeger, 2009], and laser devices are highly nonlinear. The use of ELM probably compensate for this weak spot of ESN.

Next, we repeat the experiments, this time considering augmented state vector as (1.21). The "effective" reservoir size then becomes $2N+2$. The result is shown in Table 4.2. We have also plot the NMSE as a function of M for the non-augmented cases in Figures 4.7 through 4.10 and for the augmented cases in Figures 4.11 through 4.14.

Comparing Tables 4.2 to Table 4.1, we see that state vector augmentation leads to better performance, as expected. The same trend as in the non-augmented case held for the laser and MG problems, where relatively small M was enough for acceptable NMSE. The NARMA10 problem required $M = 80$ for the NMSE to reach acceptable levels. The only problem where the NMSE was not acceptable at $M = 0.25N$ was the mapping35 problem, which needed $M = 120$, producing NMSE of 0.3989.

4.4.1 Computational Saving

In order to demonstrate the computational saving achieved by using ELM readouts, we compare the real time taken to perform 20 trials of training and testing, using $M = 30$ for the non-augmented case, and $M = 70$ for the augmented case. The reason we do not compare the computational cost in terms of multiplications per cycle, as is usually done, is because ELM readouts have the cost of evaluating the tanh activation functions for the ELM's neurons. This cost is not included in the number of multiplications. Therefore, it would be better to compare the computational costs using real time taken on the same hardware, since this captures the total cost. The reason we used $M = 30$ and $M = 70$ is because these are the average number ELM neurons required to reach acceptable performance, across the four test problems, for the non-augmented and augmented case, respectively. Figures 4.5 and 4.6 respectively show the computation times for the non-augmented and augmented case. From Figure 4.5, it can be seen that the computation time of ELM stayed below 5 seconds as N was increased from 20 to 200, while the computation time of RLS went up to almost 30 seconds for the same range of N. The computational saving when $N = 200$ is 84.1%. The computational saving for the augmented case in Figure 4.6 is even greater as the effective reservoir size is now $2N + 2$. When $N = 200$, the computational saving is 94.28%.

Table 4.1: NMSE results when the state vector is not augmented

	RLS	$M = 20$	$M = 30$	$M = 40$
NARMA10	0.1430	0.2220	0.1679	0.1500
laser	0.0775	0.0590	0.0294	0.0181
MG ($\times 10^{-4}$)	2.838	3.419	2.545	2.387
mapping35	0.5073	0.5046	0.4936	0.4861

Table 4.2: NMSE results when the state vector is augmented.

	RLS	$M = 40$	$M = 50$	$M = 60$	$M = 70$	$M = 80$	$M = 90$	$M = 100$	120
NARMA10	0.1195	0.1558	0.1479	0.1324	0.1221	0.1160	0.1039	0.0933	N/A
laser	0.0222	0.0173	0.0137	0.0120	0.0112	0.0105	0.0104	0.0103	N/A
MG ($\times 10^{-4}$)	2.373	2.508	2.335	2.303	2.291	2.287	2.275	2.269	N/A
mapping35	0.4069	1.0165	0.9781	0.9360	0.6272	0.4967	0.4625	0.4615	0.3989

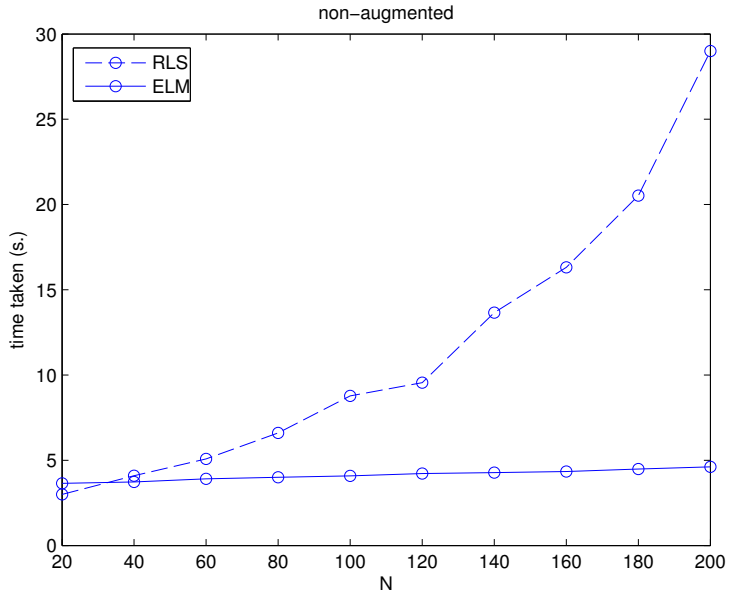

Figure 4.5: Real time taken to perform 20 trials of training and testing for RLS and ELM as a function of the reservoir size N. Number of neurons in ELM, $M = 30$ is the average to achieve satisfactory performance across the 4 test problems in the non-augmented case.

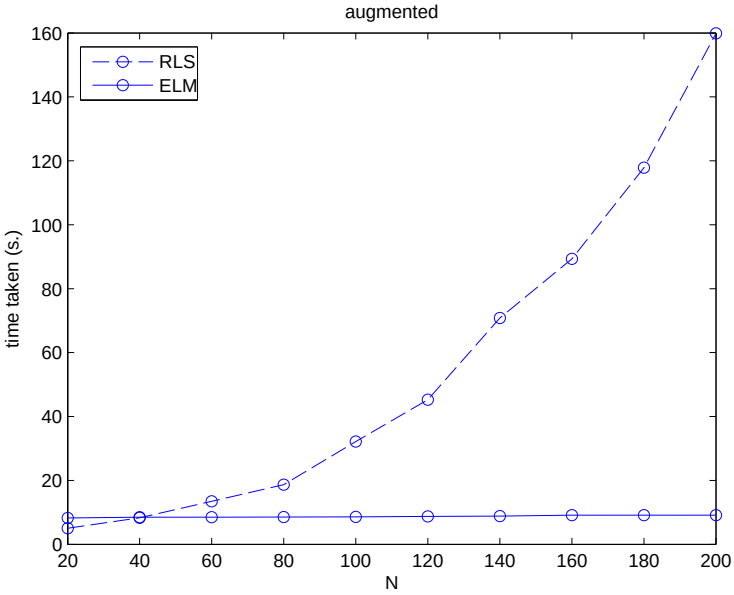

Figure 4.6: Real time taken to perform 20 trials of training and testing for RLS and ELM as a function of the reservoir size N. Number of neurons in ELM, $M = 70$ is the average to achieve satisfactory performance across the 4 test problems in the augmented case.

4.5 Conclusion

We started out this chapter by investigating the effect of extremely high eigenvalue spread of the reservoir state matrix $\mathbf{P}$ has on the online learning algorithms. By comparing the learning curves of LMS and RLS on an example system identification problems, it was found that LMS has very slow convergence and high final error when compared to RLS algorithm, as shown in Figure 4.1. This confirms the well-known fact in the ESN literature that the eigenvalue spread of $\mathbf{P}$ can be extreme high, even with noise added to the state update. This prevents LMS from being used successfully for online readout training.

We then investigated a learning algorithm called back-propagation decorrelation (BD) which was claimed in [Steil, 2004] to exhibit similar level of performance as the APRL algorithm for RNN, and has only $O(N)$ complexity when applied only to the output weights. However, when we actually applied BD to the ESN structure, we found that the final error achieved by BD is only slightly better than using LMS and still much worse than RLS. The same observation had also been made in [Küçükemre, 2006] using test problems that are different from our own.

From the poor performance of LMS and BD, we concluded that there is no choice but to use the RLS algorithm for online training. Therefore in order to reduce the computational cost, the number of taps to be trained must be reduced. Following this direction, we investigated the use of ELM as a readout. ELM is ESN's counterpart for feed-forward networks and can be trained without local minima using RLS. Computation saving is achieved by making the number of neurons in the ELM structure M significantly smaller than the reservoir size N.

We tested all large-reservoir problems both with non-augmented and augmented state vector. It was found that the NMSE values of ELM can actually become lower than that of RLS, given a large enough M. From the results, we see that in order to reach satisfactory NMSE, $M = 30$ is required on average for the non-augmented case. For the augmented case, on average $M = 70$ is required. Compared to the cost of training linear readouts with RLS, using ELM achieved average computational saving of 84.1% and 94.28%, respectively for the non-augmented and augmented case. These big saving, coupled with being able to reach lower NMSE values, make ELM a very much better choice than linear readout, especially for large N.

4.6 Epilogue

As an epilogue to this chapter, we would like to summarize other methods we have tried for online readout training that we have found to be lacking. Since online training of ESN have received relatively little attention, there may be some researchers considering these methods we have already tried.

The first thing that came to mind when faced with the huge eigenvalue spread of $\mathbf{\Phi}$ was to use the variable leaky LMS algorithm [Kamenetsky and Widrow, 2004] (VLLMS), sine the eigenvalue spread "seen" by it is much lower than the true eigenvalue spread due to the leakage factor. However, in order to have proper convergence, the leakage factor must eventually become zero. We observe the using VLLMS does accelerate initial convergence, but as the leakage factor becomes zero, the excess NMSE at the end is the same as using LMS.

Based on the observation that the eigenvalues of $\mathbf{\Phi}$ have few large ones and the rest are very small, we tried using the low rank adaptive filter [Strobach, 1996] (LRAF) to adapt the readout. LRAF is basically RLS with reduced computational complexity of $O(Nr)$ where r is the rank, the number of eigenvalues and eigenvectors used to reconstruct the reduced rank covariance matrix $\mathbf{\Phi}_r$ from the full rank $\mathbf{\Phi}$. If the rank r required for good performance is small, then large computational saving can be had. Unfortunately, we found that the rank required for the same "acceptable performance" as presented in this chapter can be as large as $N/2$, especially for augmented state vectors. In such case the computational saving is only a factor of two. Moreover, the NMSE values produced by low rank filters did not become lower than those produced by the full rank ones, despite the finding in [Scharf and Tufts, 1987] which suggests that it can happen given the distribution of eigenvalues that we observed.

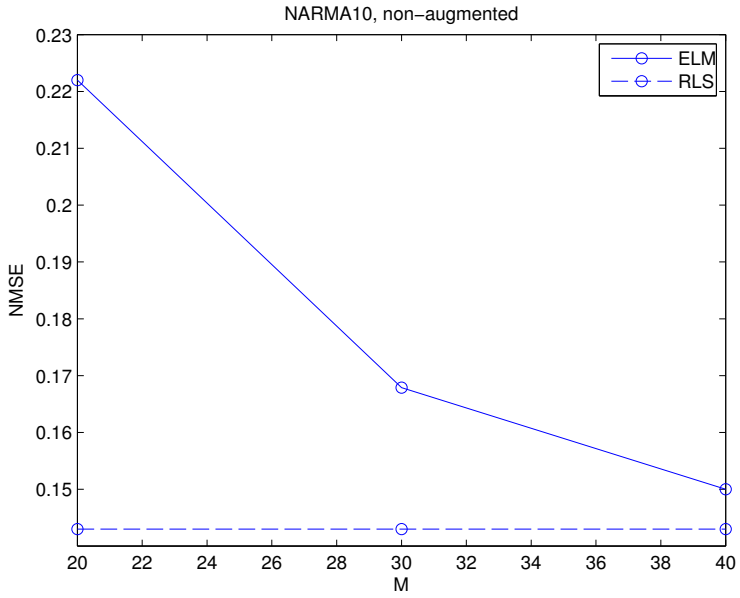

Figure 4.7: NMSE's of RLS and ELM for the NARMA10 problem, non-augmented case.

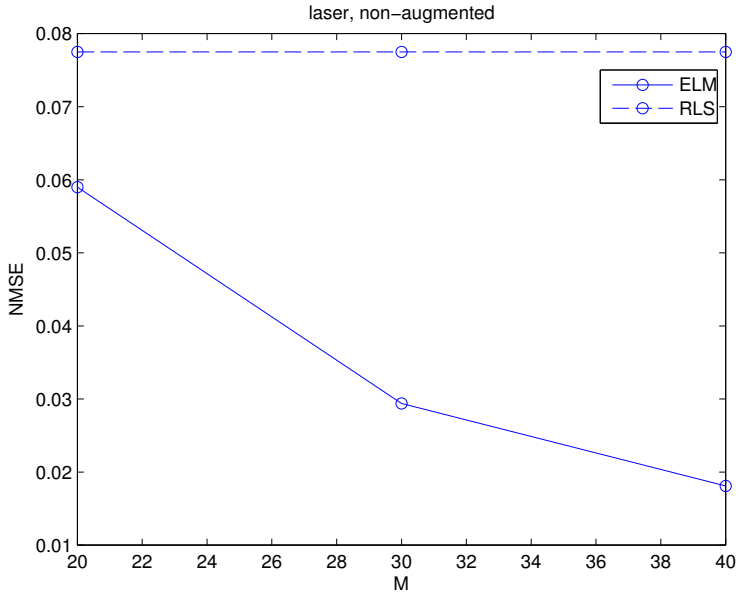

Figure 4.8: NMSE's of RLS and ELM for the laser problem, non-augmented case.

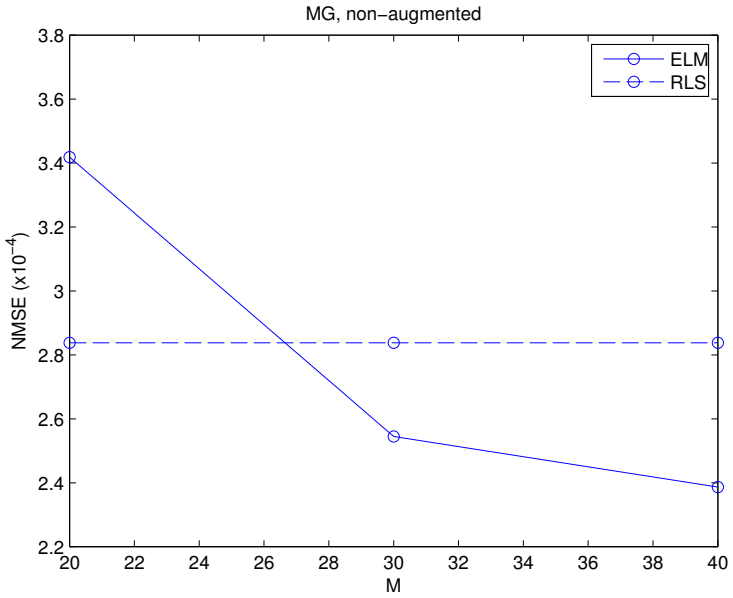

Figure 4.9: NMSE's of RLS and ELM for the MG problem, non-augmented case.

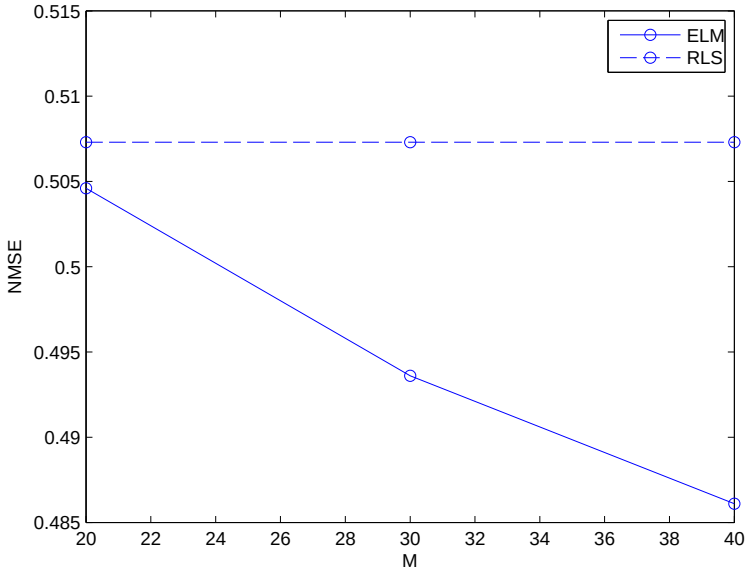

Figure 4.10: NMSE's of RLS and ELM for the mapping35 problem, non-augmented case.

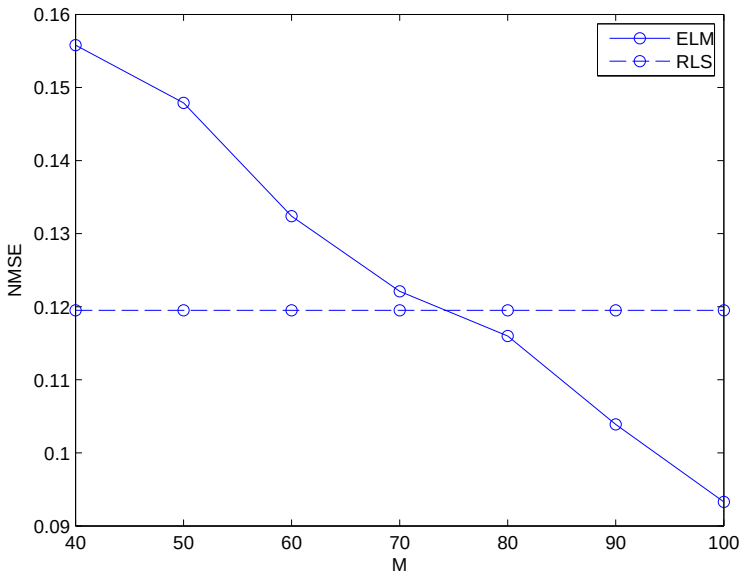

Figure 4.11: NMSE's of RLS and ELM for the NARMA10 problem, augmented case.

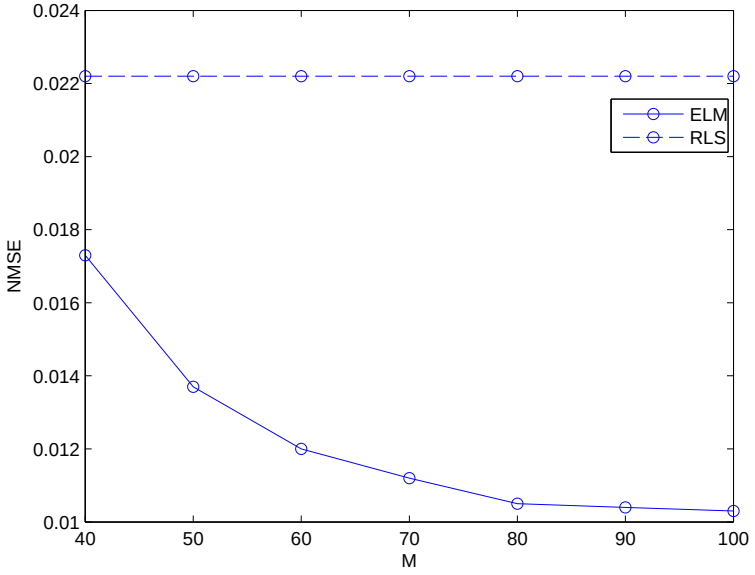

Figure 4.12: NMSE's of RLS and ELM for the laser problem, augmented case.

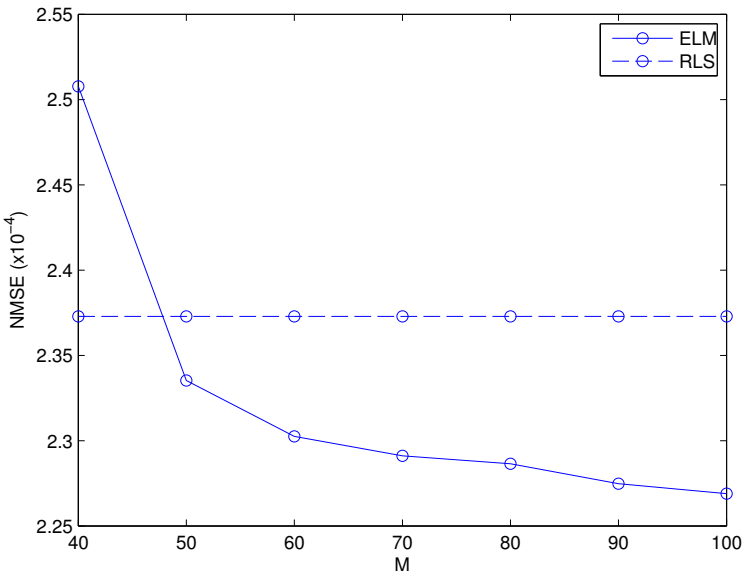

Figure 4.13: NMSE's of RLS and ELM for the MG problem, augmented case.

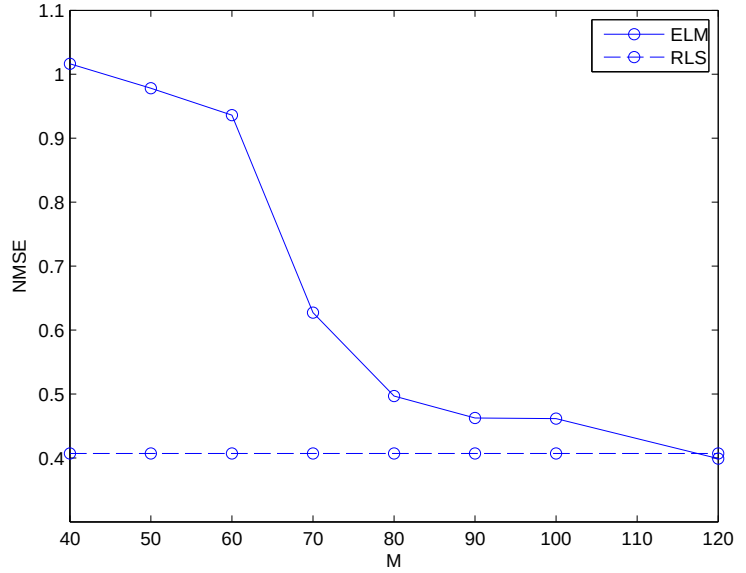

Figure 4.14: NMSE's of RLS and ELM for the mapping35 problem, augmented case.

Chapter 5

Conclusion

This thesis has presented pre-training of ESN reservoirs and low cost online readout training. In the first two chapters, we introduced ESN and demonstrated that the reservoir parameters have significant impact on ESN performance.

In Chapter 3, we discussed why it is better, from the viewpoint of pre-training, to use a deterministic reservoir instead of classical ones. We then presented our pre-training cost function based on the average mutual information between neuron states and the desired response. We mathematically showed that this cost function works in noisy environment, and demonstrated its effectiveness in pre-training of reservoirs in different adaptive filtering problems. Then, we extend the theory of ESN to the complex domain using isomorphism between real and complex vectors and matrices. We finished the chapter by extending pre-training to the complex domain.

In Chapter 4, we showed that the huge eigenvalue spread of the covariance matrix formed by the reservoir states make online training with LMS ineffective. We investigated an algorithm that is generally considered to be effective at online training and showed that it was not the case. We showed that there is no alternative to the RLS algorithm for online training of ESN. In order to reduce the computational cost of online training for large reservoir sizes, we proposed the use of ELM as the readout structure. Using ELM, the number of coefficients that must be adapted by RLS was reduced significantly, leading to huge saving in computational cost due to the $O(N^2)$ nature of the RLS algorithm.

5.1 Further Research

The proposed pre-training algorithm, though quite effective, still has high computational cost. As shown in Figure 3.5, even a C++ implementation needs more than 10 seconds to pre-train reservoirs with 200 neurons. Since very large reservoirs are quite common for ESN, speeding up the pre-training algorithm is a must to make it more practical.

To this end, we note that it may not be necessary to calculate the MI between every pair of neuron states and the desired response. It is the average of the MI values that is being maximized, not individual values. As is well known in statistics, the sample mean approaches the true population mean as the sample size increases. Therefore, it should be possible to evaluate J sufficiently accurate enough for optimization by only considering a subset of neurons in the reservoir. In other words, to calculate the MI values only for some neurons N' which can probably be much smaller than N for large reservoirs.

Another source of speedup may come from the algorithm itself. Profiling our code, we found that most of the time is spent in the histogram function, which at first seems strange since forming the histogram is just counting and involve very little calculation. However, we found that the memory access pattern of the arrays that hold the histogram values is not contiguous, i.e., there are big jumps in what element of the arrays is accessed. Non-contiguous memory access is well-known to have a large overhead. Thus, optimizing the memory access of the histogram routine can lead to significant speedup. We propose a further research objective of being able to pre-train reservoirs with 1000 neurons in under 1 second. This should be fast enough to make pre-training practical for most applications.

An interesting area for further research is to extend pre-training to problems where the desired response is not available, most notably for blind signal processing such as blind source separation or blind deconvolution/equalization. As we have shown in Chapter 3, simply maximizing the independence between neurons states, the obvious thing to do without a desired response, does not lead to the best performing reservoirs. A new cost function that uses only the input and reservoir states is needed for pre-training in blind signal processing problems, for example some high order statistics on the neurons states.

Closely related to ESN are what's called Liquid States Machine (LSM). These are basically ESN which use more biologically realistic model of neurons which

can be regarded as the "type 3" neurons after type 1 perceptrons and type 2 tanh neurons. LSM works on the same principles as ESN but are potentially more powerful due to the higher fidelity of the neurons they used. Type 3 neural networks are presently used for realistic simulation of the brain or other nervous systems [Bugmann et al., 1997; Maass and Bishop, 2001]. The use of type 3 neural networks in general and LSM in particular for engineering applications are only beginning to be investigated [Jaeger et al., 2007; Verstraeten et al., 2005]. It would be interesting to see if the any of the results presented here can be extended to LSM.

Appendix A: Differential Evolution

Differential Evolution (DE) [Storn and Price, 1997] belongs to the family of evolutionary algorithms. But it stands out due to its simplicity, robustness and good performance. Like most GO algorithms, DE is population based, where each member of the population is vector of filter coefficients. There are two control parameters: F, CR. The algorithm consists of adapting target vector $\mathbf{v}_{\text{target}}$ according to the flowchart in Figure 1 and (1-3). The vectors $\mathbf{v}_b, \mathbf{v}_i, \mathbf{v}_j$ are distinct and randomly chosen from other vectors, excluding $\mathbf{v}_{\text{target}}$. A temporary vector $\mathbf{v}_{\text{temp}}$ is generated by (1) while the crossover operation is described by (2), where "rand" denotes uniform random number and l is the vector length. The output of the crossover operation is a trial vector, which is compared to the original target vector. The one with the lower cost is saved for the next iteration, as shown by (3). One iteration is completed when each vector in the current population has taken turn being the target vector. The parameters F, CR must be appropriately chosen for DE to perform best. Tuning of DE's parameters is not part of the scope of this study, see [Das and Suganthan, 2011] for more details about DE and tips on setting the parameters.

$$\mathbf{v}_{\text{temp}} = \mathbf{v}_b + F(\mathbf{v}_j - \mathbf{v}_k), \tag{1}$$

$$v_{\text{trial},l} = \begin{cases} v_{\text{temp},l} \text{ rand} < CR \\ v_{\text{target},l} \text{ otherwise} \end{cases} \quad 1 \leq l \leq l, \tag{2}$$

$$\mathbf{v}_i(k+1) = \begin{cases} \mathbf{t}_i(k) & J(\mathbf{t}_i) < J(\mathbf{v}_i(k)) \\ \mathbf{v}_i(k) & \text{otherwise} \end{cases} . \tag{3}$$

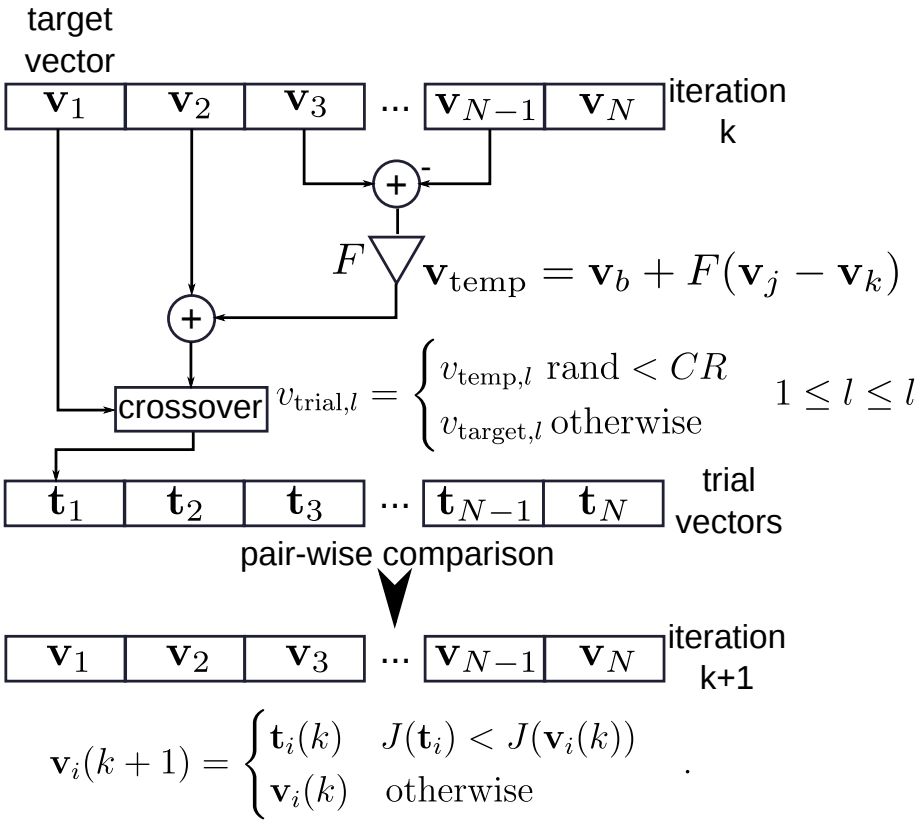

Figure 1: Flowchart of the DE algorithm.

Bibliography

Atiya, Amir F and Parlos, Alexander G. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *Neural Networks, IEEE Transactions on*, 11(3):697–709, 2000. 7, 24, 25, 26

Babinec, Štefan and Pospíchal, Jiří. Merging echo state and feedforward neural networks for time series forecasting. In *Artificial Neural Networks–ICANN 2006*, pages 367–375. Springer, 2006. 68

Bengio, Yoshua. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 21, 22

Bertschinger, Nils and Natschläger, Thomas. Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–1436, 2004. 39

Bianchini, Monica; Frasconi, Paolo, and Gori, Marco. Learning without local minima in radial basis function networks. *Neural Networks, IEEE Transactions on*, 6(3):749–756, 1995. 6

Brown, Gavin; Pocock, Adam; Zhao, Ming-Jie, and Luján, Mikel. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research*, 13:27–66, 2012. 41

Bugmann, Guido; Christodoulou, Chris, and Taylor, John G. Role of temporal integration and fluctuation detection in the highly irregular firing of a leaky integrator neuron model with partial reset. *Neural Computation*, 9(5):985–1000, 1997. 83

Bush, Keith and Anderson, Charles. Modeling reward functions for incomplete state representations via echo state networks. In *Neural Networks, 2005.*

IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on, volume 5, pages 2995–3000. IEEE, 2005. 68

Bush, Keith and Tsendjav, Batsukh. Improving the richness of echo state features using next ascent local search. In *Proceedings of the artificial neural networks in engineering conference*, pages 227–232, 2005. 44

Cha, Inhyok and Kassam, Saleem A. Channel equalization using adaptive complex radial basis function networks. *Selected Areas in Communications, IEEE Journal on*, 13(1):122–131, 1995. 27

Chen, S.; Istepanian, R., and Luk, B.L. Digital IIR filter design using adaptive simulated annealing. *Digital Signal Processing*, 11(3):241 – 251, 2001. ISSN 1051-2004. doi: 10.1006/dspr.2000.0384. URL `http://www.sciencedirect.com/science/article/pii/S1051200400903841`. 24

Das, Swagatam and Suganthan, Ponnuthurai Nagaratnam. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, 2011. 85

Estévez, Pablo A; Tesmer, Michel; Perez, Claudio A, and Zurada, Jacek M. Normalized mutual information feature selection. *Neural Networks, IEEE Transactions on*, 20(2):189–201, 2009. 40

Gershenfeld, Neil and Weigend, Andrea. Santa fe time series competition data, 1994. URL `http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html`. 29

Gerstner, Wulfram and Kistler, Werner M. Mathematical formulations of hebbian learning. *Biological cybernetics*, 87(5-6):404–415, 2002. 64

Hansen, Nikolaus. Cma-es - a stochastic second-order method for function-value free numerical optimization, October 2011. URL `https://www.lri.fr/~hansen/msrc-cmaes-nov-2011.pdf`. 44

Haykin, Simon and Li, Liang. Nonlinear adaptive prediction of nonstationary signals. *Signal Processing, IEEE Transactions on*, 43(2):526–535, 1995. 1

Haykin, Simon S. *Adaptive Filter Theory, 4/e*. Pearson Education India, 2005. 4, 15, 63

Hebb, Donald Olding. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005. 64

Hochreiter, Sepp. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998. 7

Horn, Jason; De Jesús, Orlando, and Hagan, Martin T. Spurious valleys in the error surface of recurrent networksanalysis and avoidance. *Neural Networks, IEEE Transactions on*, 20(4):686–700, 2009. 6, 24

Hornik, Kurt; Stinchcombe, Maxwell, and White, Halbert. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 4

Huang, Fu-zhuo; Wang, Ling, and He, Qie. An effective co-evolutionary differential evolution for constrained optimization. *Applied Mathematics and computation*, 186(1):340–356, 2007. 44

Huang, Guang-Bin; Zhu, Qin-Yu, and Siew, Chee-Kheong. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006. 68

Huang, Guang-Bin; Zhou, Hongming; Ding, Xiaojian, and Zhang, Rui. Extreme learning machine for regression and multiclass classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(2):513–529, 2012. 68

Ingber, Lester. Adaptive simulated annealing (asa): Lessons learned. In *Control and cybernetics*. Citeseer, 1996. 8

Ishu, K; van der Zant, Tijn; Becanovic, Vlatko, and Ploger, P. Identification of motion with echo state network. In *OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04*, volume 3, pages 1205–1210. IEEE, 2004. 44

Jaeger, Herbert. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001. 8, 11, 13, 51, 53

Jaeger, Herbert. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002. 7

Jaeger, Herbert. Adaptive nonlinear system identification with echo state networks. *Networks*, 8:9, 2003. 17

Jaeger, Herbert. Reservoir riddle: Suggestions for echo state network research. In *Proceedings of International Joint Conference on Neural Networks*, pages 1460–1462, 2005. 38, 63, 64

Jaeger, Herbert and Haas, Harald. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. 26

Jaeger, Herbert; Lukoševičius, Mantas; Popovici, Dan, and Siewert, Udo. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007. 83

Jiang, Fei; Berry, Hugues, and Schoenauer, Marc. Supervised and evolutionary learning of echo state networks. In *Parallel Problem Solving from Nature–PPSN X*, pages 215–224. Springer, 2008. 44

Jin, Liang; Nikiforuk, Peter N, and Gupta, Madan M. Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks. *Automatic Control, IEEE Transactions on*, 40(7):1266–1270, 1995. 5

Kamenetsky, Max and Widrow, Bernard. A variable leaky lms adaptive algorithm. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, volume 1, pages 125–128. IEEE, 2004. 65, 76

Kantz, Holger and Schreiber, Thomas. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004. 1

Karaboga, Nurhan. Digital IIR filter design using differential evolution algorithm. *EURASIP Journal on Applied Signal Processing*, 2005(8):1296–1276, 2005. 24

Kechriotis, G; Zervas, E, and Manolakos, ES. Using recurrent neural networks for adaptive communication channel equalization. *Neural Networks, IEEE Transactions on*, 5(2):267–278, 1994. 27, 28

Koh, Taiho and Powers, E. Second-order volterra filtering and its application to nonlinear system identification. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 33(6):1445–1455, 1985. 2

Krusienski, D.J. and Jenkins, W.K. Particle swarm optimization for adaptive IIR filter structures. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 965 – 970 Vol.1, june 2004. doi: 10.1109/CEC.2004.1330966. 8, 24

Küçükemre, Ali Uygar. *Echo state networks for adaptive filtering*. PhD thesis, University of Applied Sciences, 2006. 67, 75

Li, Ming-Bin; Huang, Guang-Bin; Saratchandran, Paramasivan, and Sundararajan, Narasimhan. Fully complex extreme learning machine. *Neurocomputing*, 68:306–314, 2005. 68

Lin, Uung-Chien; Hwang, Kao-Shing, and Wang, Feng-Sheng. Plant scheduling and planning using mixed-integer hybrid differential evolution with multiplier updating. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 593–600. IEEE, 2000. 44

LukošEvičIus, Mantas and Jaeger, Herbert. Survey: Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3): 127–149, 2009. 71

Maass, Wolfgang and Bishop, Christopher M. *Pulsed neural networks*. MIT press, 2001. 83

Maass, Wolfgang; Legenstein, Robert, and Bertschinger, Nils. Methods for estimating the computational power and generalization capability of neural microcircuits. *Advances in Neural Information Processing Systems*, 17, 2005. 37

Mandic, Danilo P. Complex valued and quaterion valued 2d and 3d wind modelling and prediction, 2012. URL `http://www.commsp.ee.ic.ac.uk/~mandic/research/wind.htm`. 57

Mandic, Danilo P and Chambers, Jonathon A. Toward an optimal prnn-based nonlinear predictor. *Neural Networks, IEEE Transactions on*, 10(6):1435–1442, 1999. 7

Moddemeijer, Rudy. On estimation of entropy and mutual information of continuous distributions. *Signal Processing*, 16(3):233–248, 1989. 41

Narendra, Kumpati S and Parthasarathy, Kannan. Identification and control of dynamical systems using neural networks. *Neural Networks, IEEE Transactions on*, 1(1):4–27, 1990. 18

Nelles, Oliver. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer, 2001. 1

Nerrand, Olivier; Roussel-Ragot, Pierre; Personnaz, Léon; Dreyfus, Gérard, and Marcos, S. Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms. *Neural computation*, 5(2):165–199, 1993. 5

Özdemir, Necati; İskender, Beyza B, and Özgür, Nihal Yılmaz. Complex valued neural network with möbius activation function. *Communications in Nonlinear Science and Numerical Simulation*, 16(12):4698–4703, 2011. 56

Ozturk, Mustafa C; Xu, Dongming, and Príncipe, José C. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007. 36, 38

Patra, Jagdish C; Pal, Ranendra N; Baliarsingh, Rameswar, and Panda, Ganapati. Nonlinear channel equalization for qam signal constellation using artificial neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(2):262–271, 1999. 1

Plöger, Paul G; Arghir, Adriana; Günther, Tobias, and Hosseiny, Ramin. Echo state networks for mobile robot modeling and control. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 157–168. Springer, 2004. 64

Plumbley, Mark D. Efficient information transfer and anti-hebbian neural networks. *Neural Networks*, 6(6):823–833, 1993. 64

Príncipe, José C; Liu, Weifeng, and Haykin, Simon. *Kernel Adaptive Filtering: A Comprehensive Introduction*, volume 57. John Wiley & Sons, 2011. 3

Rodan, Ali and Tiňo, Peter. Simple deterministically constructed cycle reservoir with regular jumps. *Neural Computation*, 24(7):1822–1852, July 2012. 36, 39, 59

Sacchi, Rodrigo; Ozturk, Mustafa C; Principe, Jose C; Carneiro, Adriano AFM, and da Silva, Ivan Nunes. Water inflow forecasting using the echo state network:

a brazilian case study. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 2403–2408. IEEE, 2007. 64

Salmen, Matthias and Ploger, Paul G. Echo state networks used for motor control. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1953–1958. IEEE, 2005. 64

Sanderson, Conrad. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010. 59

Scharf, Louis L and Tufts, Donald W. Rank reduction for modeling stationary signals. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3): 350–355, 1987. 76

Schetzen, Martin. Nonlinear system modeling based on the wiener theory. *Proceedings of the IEEE*, 69(12):1557–1573, 1981. 2

Schmidhuber, Jürgen. A fixed size storage o (n3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4 (2):243–248, 1992. 24

Schrauwen, Benjamin; Verstraeten, David, and Van Campenhout, Jan. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, pages 471–482, 2007. 63

Seth, Sohan; Ozturk, Mustafa C, and Principe, Jose C. Signal processing with echo state networks in the complex domain. In *Machine Learning for Signal Processing, 2007 IEEE Workshop on*, pages 408–412. IEEE, 2007. 51

Skowronski, Mark D and Harris, John G. Noise-robust automatic speech recognition using a predictive echo state network. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(5):1724–1730, 2007. 64

Song, Qingsong and Feng, Zuren. Effects of connectivity structure of complex echo state network on its prediction performance for nonlinear time series. *Neurocomputing*, 73(10):2177–2185, 2010. 38

Steil, Jochen J. Backpropagation-decorrelation: Online recurrent learning with o (n) complexity. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 843–848. IEEE, 2004. 39, 66, 75

Steuer, Ralf; Kurths, Jürgen; Daub, Carsten O; Weise, Janko, and Selbig, Joachim. The mutual information: detecting and evaluating dependencies between variables. *Bioinformatics*, 18(suppl 2):S231–S240, 2002. 40

Storn, Rainer and Price, Kenneth. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997. 85

Strobach, Peter. Low-rank adaptive filters. *Signal Processing, IEEE Transactions on*, 44(12):2932–2947, 1996. 76

Sun, Guo-Zheng; Chen, Hsing-Hen, and Lee, Yee-Chun. Green's function method for fast on-line learning algorithm of recurrent neural networks. In *Advances in neural information processing systems*, pages 333–340, 1992. 24

Toomarian, Nikzad Benny and Barhen, Jacob. Adjoint-functions and temporal learning algorithms in neural networks. In *Advances in neural information processing systems*, pages 113–120, 1991. 24

Verstraeten, David. *Reservoir Computing: computation with dynamical systems*. PhD thesis, Ghent University, 2009. 39

Verstraeten, David; Schrauwen, Benjamin, and Van Campenhout, Jan. Recognition of isolated digits using a liquid state machine. In *Proceedings of SPS-DARTS*, volume 2005, pages 135–138. Citeseer, 2005. 83

Verstraeten, David; Schrauwen, Benjamin; d'Haene, Michiel, and Stroobandt, Dirk. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007. 39

Walters-Williams, Janett and Li, Yan. Estimation of mutual information: A survey. In *Rough Sets and Knowledge Technology*, pages 389–396. Springer, 2009. 40

Werbos, Paul J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 23, 24

Widrow, Bernard and Stearns, Samuel D. *Adaptive signal processing*, volume 15. IET, 1985. 1, 63

Williams, Ronald J and Zipser, David. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. 6, 23

Xia, Yili; Mandic, Danilo P; Hulle, M, and Principe, Jose C. A complex echo state network for nonlinear adaptive filtering. In *Machine Learning for Signal Processing, 2008. MLSP 2008. IEEE Workshop on*, pages 404–408. IEEE, 2008. 51

Xia, Yili; Jelfs, Beth; Van Hulle, Marc M; Príncipe, José C, and Mandic, Danilo P. An augmented echo state network for nonlinear adaptive filtering of complex noncircular signals. *Neural Networks, IEEE Transactions on*, 22(1):74–83, 2011. 51

Yuenyong, Sumeth and Nishihara, Akinori. A hybrid gradient-based and differential evolution algorithm for infinite impulse response adaptive filtering. *International Journal of Adaptive Control and Signal Processing*, 2013. 25

Yuenyong, Sumeth and NISHIHARA, AKINORI. Training recurrent neural network for nonlinear adaptive channel equalization with differential evolution. In *Proceedings of 2013 RISP International Workshop on Nonlinear Circuits, Communication and Signal Processing*, volume 1, pages 409–411, 2013. 8, 24, 26

Zhao, Haiquan and Zhang, Jiashu. Nonlinear dynamic system identification using pipelined functional link artificial recurrent neural network. *Neurocomputing*, 72(13):3046–3054, 2009. 7

Zhao, Haiquan; Zeng, Xiangping, and He, Zhengyou. Low-complexity nonlinear adaptive filter based on a pipelined bilinear recurrent neural network. *Neural Networks, IEEE Transactions on*, 22(9):1494–1507, 2011. 7