Machine Learning With Echo State Networks

Technical Report · March 2020

CITATIONS
0

READS
97

1 author:

Yuanchao Xu
University of Manitoba
1 PUBLICATION **0** CITATIONS

SEE PROFILE

A Review of Machine Learning With Echo State Networks

Yuanchao Xu

June 16, 2020

Contents

Abstract

This report aims at presenting a detailed technical review of Recurrent Neural Networks (RNNs) and in particular Echo State Networks (ESNs), which are networks belonging to the general class of Reservoir Computing. A general overview of RNNs is presented, followed by a detailed discussion of Reservoir Computing and Echo State Networks. We discuss the working principal, construction details, applications and limitations of ESNs. Current research work in ESNs directed towards overcoming some of the limitations faced by researchers using ESNs, is reviewed. One major limitation of ESNs is their dependence on the hyper-parameters. The stable operating region of ESNs in the hyper-parameter space is quite narrow, which makes training these networks a challenging task. The paper reviewed proposes a novel model of ESNs, which prevents the network from entering chaotic regime. The paper also demonstrates the improved performance in the experiments performed over standard benchmarking datasets for nonlinear systems. Finally, we conclude with a discussion of renewed interest in ESNs, in particular, as a computational principle.

Keywords: *ESN, hyper-parameters, EoC, self-normalizing activations*

Chapter 1

Introduction

1.1 Overview

Machine Learning is a field which lies at the intersection of artificial intelligence, statistics and computer science. Its main goal is to extract knowledge from data. It is also referred to as *statistical learning* or *predictive analytics*. Machine learning applications have touched every aspect of modern daily life in recent years. From automatic movie recommendations to interactive digital assistants, to suggestions for which products to buy, to recognizing your friends in your photos, many modern websites and devices have machine learning algorithms at their core. Platforms such as Facebook, Amazon or Netflix have multiple machine learning models working under the hood. Outside of commercial applications, machine learning has contributed greatly to the way data driven research is done today. Machine learning is helping researchers find solutions to diverse scientific questions such as understanding stars, finding distant planets, analyzing DNA sequences, and providing personalized cancer treatments.

This tremendous growth in machine learning applications is fueled by advancements in a particular sub-field of machine learning, called as *Deep Learning*. Deep learning is a term used for machine learning models based on Artificial Neural Networks, with a large number of hidden layers, typically more than two hidden layers. The formal definition of deep learning can be stated as [30]: *A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification.* In this report we will not focus on machine learning techniques other than neural networks, hence the terms machine learning and deep learning are used interchangeably here.

At its core, machine learning is all about automating the decision making process by generalization of known examples. Machine learning can be divided in four broad areas, as described next:

1. In *supervised learning*, the user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired output given an input. In particular, the algorithm is able to create an output for an input it has never seen before, without any help from a human. So the model is used in two phases, *training*, during which the model is shown the desired outputs for a set of inputs (hence the name *supervised*) and *inference*, during which the model predicts output for a previously unseen input. Examples of supervised machine learning tasks include:

    - Identifying the amount from handwritten digits on a cheque: Here the input is a scanned image of a cheque, and the desired output is the actual amount. To create a data set for building a machine learning model, many sample cheque images are collected and are labeled by manually reading the amount.
    - Diagnosis of a tumor (benign/malignant) based on a medical image: Here the input is the image, and the output is whether or not the tumor is benign. The dataset for building a model is a database of medical images, with labels obtained from an expert in the field.
    - Detecting fraudulent financial activity: Here the input is a record of a credit card transaction, and the output is whether it is likely to be fraudulent or not. Collecting a dataset for this task means storing all transactions, and recording if a user reports any transaction as fraudulent.

2. In the second category of ML algorithms, *unsupervised learning*, only the input data is known and there is no known output data available to the algorithm. Unsupervised learning is providing the model with information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of algorithm is to detect patterns in the unlabeled data, or to group unsorted information according to similarities, patterns and differences without any prior training on the data. While there are many successful applications of these methods as well, they are usually harder to understand and evaluate.

3. The third category of ML, *Reinforcement Learning* does not fit into the paradigm of supervised or unsupervised Learning. It is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is learning how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging

cases, actions may act not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics-trial-and-error search and delayed reward-are the two most important distinguishing features of reinforcement learning. Some examples of reinforcement learning can be listed as follows:

- Robotics for industrial automation.
- Business strategy planning.
- Game playing (Atari, Chess, Go etc.)
- Machine learning and data processing.
- Autonomous driving.
- Aircraft control and robot motion control

4. In recent times, a new category of ML called *Semi-Supervised Learning* has emerged. In this type of learning, the algorithm is trained using a combination of labeled and unlabeled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabeled data. First, clustering analysis is used to identify cluster of similar data using an unsupervised learning algorithm and then the existing labeled data is used to label the rest of the unlabeled data in that particular cluster. The typical use cases of such type of algorithm are motivated from the fact that acquisition of unlabeled data is relatively cheap while labeling the said data is very expensive.

Since Supervised learning is the most successful area of ML and it forms the largest proportion of ML applications today, we will refer to the concepts discussed in this report, in the the context of Supervised learning and will not discuss other three categories of ML any further.

1.2 Scope of This Report

This report primarily focuses on a subclass of recurrent neural networks, called as Reservoir computing, in particular Echo State Networks (ESN). The first chapter introduces the machine learning field (section 1.1) and defines the terminology used in the report (section 1.3). Next, recurrent neural networks are discussed in detail (chapter 2). Third chapter describes reservoir computing and various models belonging to this class (chapter 3). The fourth chapter introduces Echo State Networks (chapter 4). Their working principal (section 4.1), limitations (section 4.4) and latest research(section 4.5) are discussed in depth.

1.3 Machine Learning Model Structure

The canonical problem of ML research can be stated as: given a finite subset of the graph of a function or mapping f, determine what f is. The subset of the given graph is called as training set. Elements of its domain are spatial patterns. The ML algorithm must process the training set to come up with a representation for f. This learnt representation is used to compute f on elements of the domain not appearing in the training set. The entire process of ML training and inference is described next, with reference to a simple neural network shown in Figure 1.1 Let us denote by U a set of m training examples

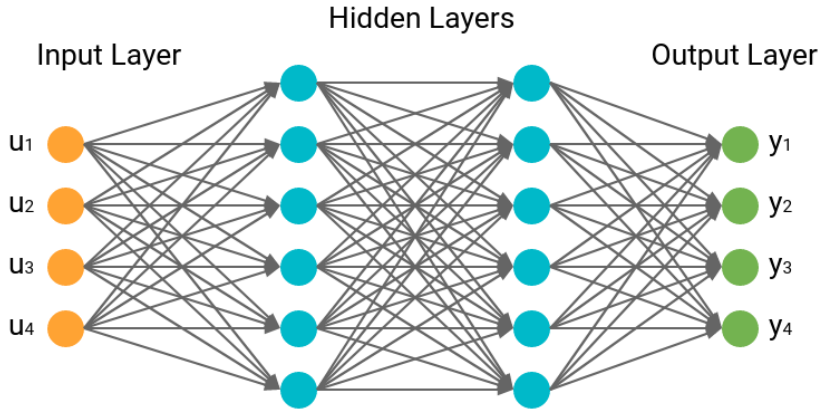


Figure 1.1: Schematic of a simple three layer neural network.

and $\hat{y}$ denote the corresponding set of desired outputs. The objective of the machine learning model is to find a function or mapping f such that $\hat{y}^{(i)} = f(U^{(i)})$ for all $i \in m$. During the training phase, the input set is presented to the network. The dimensionality of the input space is given by $n^{[u]}$, which represents the number of features relevent to the output prediction being made. Thus, a single training example is represented by a $n^{[u]}$ dimensional vector, called as feature vector. This vector passes to the first layer of neural network through connections denoted by arrows. These connections have weights associated with it denoted by a matrix $W^{[1]}$ for the first layer, $W^{[2]}$ for the second layer and so on. If $n^{[1]}$ denotes the number of neurons in the first layer, $W^{[1]}$ is a matrix of size $n^{[1]} \times n^{[u]}$. For brevity we omit the *bias* or *intercept* term in the layer equations. In addition to the weight matrix, the layer is characterised by an *activation function* denoted by σ, which transforms its input in a nonlinear way. So after the first layer, input is transformed to: $a^{[1]} = \sigma(W^{[1]} \times U_i)$

This process is repeated for all the layers of the neural network. Assuming the network has N_L layers, the entire network is characterised by N_L weight matrices and N_L acti-

vation functions. Thus each input is transformed into a $n^{[y]}$ dimensional vector known as output vector. This is known as *forward pass* or *forward propagation*. When the network training begins, all weight matrices are initialised with small random weights. The output obtained with these will not be desired output $\hat{y}^{(i)}$. The output $y^{(i)}$ after the forward propagation is compared with desired output $\hat{y}^{(i)}$ and the difference is characterised by a suitable *cost function* or *loss function*. The objective of the training process is to obtain values for all the weights, which will minimize the cost function. For this purpose, gradient of the cost function is calculated with respect all connection weights. Based on the gradient, the weights are iteratively adjusted to minimize the cost function, using suitable optimization algorithm. One pass of the model through entire training set is called as one *epoch*. The process of training can proceed in number of ways. One way is to present one training sample at a time, selected randomly from the training set. This process is known as (*stochastic gradient descent*). It is very efficient memory-wise since only one example needs to be loaded in the memory, but it takes longest time to complete one epoch. If entire set is presented in a single step, it is called (*batch gradient descent*). The process of presenting input data into multiple batches is known as (*mini-batch gradient descent*). The training process is carried out for multiple epochs, till the cost function is reduced to satisfactory levels.

Chapter 2

Recurrent Neural Networks

2.1 What are RNNs

The primary characteristics of neural networks discussed in previous chapter is that they have no memory. Each input presented to the model is processed independently, with no state information preserved between inputs. While this works fine for many types of data, there is a large class of inputs where the data is in the form of a sequence or temporal series i.e. individual data points are not independent of each other. To process this type of data using neural networks, it has to be shown complete sequence at once, i.e., convert entire sequence into single data point. While this approach can work in some cases, this is not optimal solution due to various reasons.

Firstly, biological systems processing sequential data do not process the entire sequence at once. For example, as you are reading this sentence, you're processing it word by word - or rather, eye saccade by eye saccade[1], while keeping memories of what came before. This gives you a fluid representation of the meaning conveyed by this sentence. Thus, we process information incrementally while maintaining an internal model of what we are processing, built from past information and and constantly updated as new information comes in.

A *recurrent neural network* (RNN) adopts the same principle. It processes sequences by iterating through sequence elements and maintaining a state containing information relative to what it has seen so far. RNN is a type of neural network that has an internal loop. The state of the RNN is reset between processing two different, independent sequences. So one sequence is still considered as a single data point. The difference is that this data point is no longer processed in a single step; rather, the network internally loops over the sequence elements. Formally, a recurrent neural network (RNN)

[1]Eyes do not move continuously along a line of text, but make short, rapid movements (saccades) intermingled with short stops (fixations).

is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs.

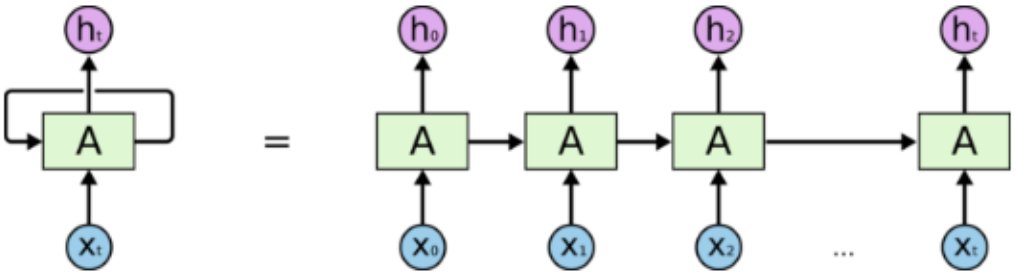


Figure 2.1: Schematic representation of RNN

Figure 2.1 shows a schematic representation of RNN, along with the same network *unrolled* in time. Time, in this context can be physical time or it can be any independent variable of the sequence. For example, in text data, position of the word or character in the text block, takes place of the time. As shown in figure, the state information is carried over from one time step to another, and the output at any step depends upon the input at that timestep and the previous state of the network.
Mathematically, we can represent the working of RNN as:

$$y_t = f_1(X_t, U_t)$$

Where X_t denoted state of the network at time t. It is calculated as

$$X_t = f_2(X_{t-1}, U_t)$$

In other words, the network state implicitly depends on all previous inputs, through the past states. Thus, a recurrent neural network is not just a function or mapping $f : U \to y$, but it is a dynamical system $\delta : X \times U \to X$.
The simple RNN described above is modified in order to make it useful to perform practical tasks. Next, section describes some popular RNN architectures.

2.2 RNN architectures

In practical applications such as text processing, time-series predictions, sequence-to-sequence transformation such as language translation, the network should carry information about previous input. A simple RNN is theoretically able to retain inputs seen

many time-steps before, but such long term dependencies are difficult to learn. This is due to the vanishing gradient problem, an effect that is similar to what is observed with non-recurrent networks (feedforward networks) that are many layers deep. As layers are added to a network, the network eventually becomes untrainable. Just like simple neural networks, training the RNN also involves backpropagation [18]. The difference in this case is that since the parameters are shared by all time steps, the gradient at each output depends not only on the current time step, but also on the previous ones. This process is called backpropagation through time. Vanishing gradients prevents the effect of current state from impacting states in the distant future. The theoretical reasons for this effect were studied by in the 90s [2]. A Long Short Term Memory (*LSTM*) network solves this problem as described below.

2.2.1 Long Short Term Memory (LSTM) Network

The Long Short-Term Memory (LSTM) algorithm was developed by Hochreiter and Schmidhuber [19]. It was the culmination of their research on the vanishing gradient problem. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is their primary feature. Figure 2.2 shows a schematic representation of ideas behind LSTM.

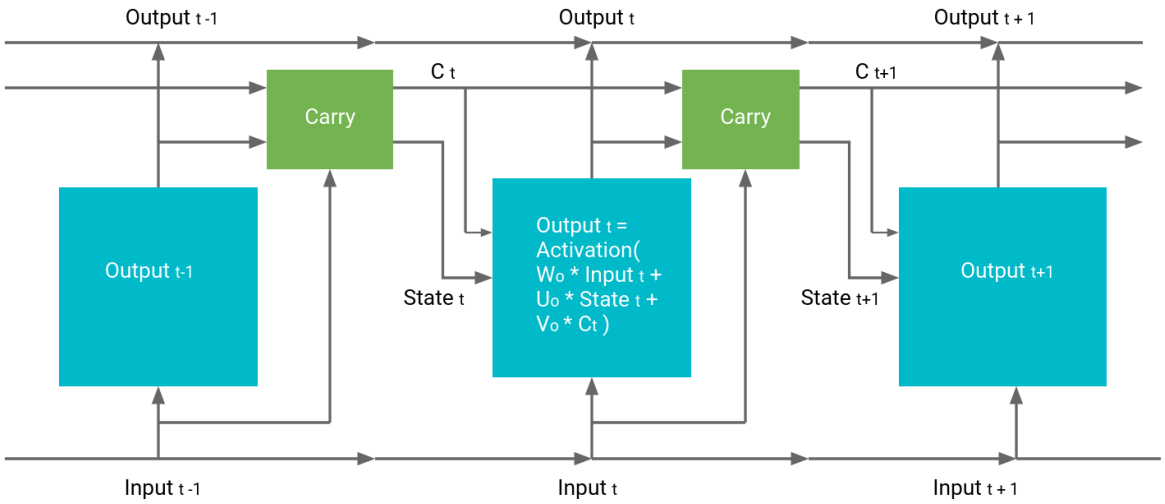


Figure 2.2: Schematic representation of LSTM cell

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram, below the output line. The cell state acts like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are

a way to optionally let information through. They are composed out of sigmoid neurons and a point-wise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means input should be forgotten while a value of one means input should be remembered.

An LSTM has three gates, to protect and control the cell state. To define the working more concretely, the first step in LSTM operation is to decide what information it is going to throw away from the cell state. This decision is made by a sigmoid layer called the *Carry Computation* . It computes the next carry value at each step as follows: $C_{t+1} = C_t * f_t + i_t * k_t$ where i_t, k_t and f_t are calculated by a sigmoid layer, from previous state and current inputs as:

$i_t = X_t * W_i^1 + U_t * W_i^2$

$f_t = activation(X_t * W_f^1 + U_t * W_f^2)$

$k_t = activation(X_t * W_k^1 + U_t * W_k^2)$

At each timestep, output is calculated by:

$y_t = activation(X_t * W_0^1 + U_t * W_0^2 + C_t * V_0)$

Thus, it looks at C_{t-1} and U_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 means the state is retained while a 0 means it is forgotten. For example, in a language model trying to predict the next word based on all the previous ones, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

The next step is to decide what new information it is going to store in the cell state. It combines these two to create an update to the state. In the example of the language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting. It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t. This can be interpreted as forgetting irrelevant information in the carry dataflow. Meanwhile i_t and k_t provide information about the present, updating the carry track with new information, forgetting the things it decided to forget earlier. However, these interpretations are only indicative of what the neural net might be learning, and it's actual operation might be different depending on the learnt weight values.

2.2.2 Gated Recurrent Units

Gated Recurrent Units of *GRU* cells are another popular implementation of RNNs. A gated recurrent unit (GRU) was proposed [8] to make each recurrent unit to adaptively capture dependencies of different time scales. Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells. Figure 2.3 shows schematic representation of a GRU cell. It is easy to notice similarities between the LSTM unit and the GRU from Figure 2.3.

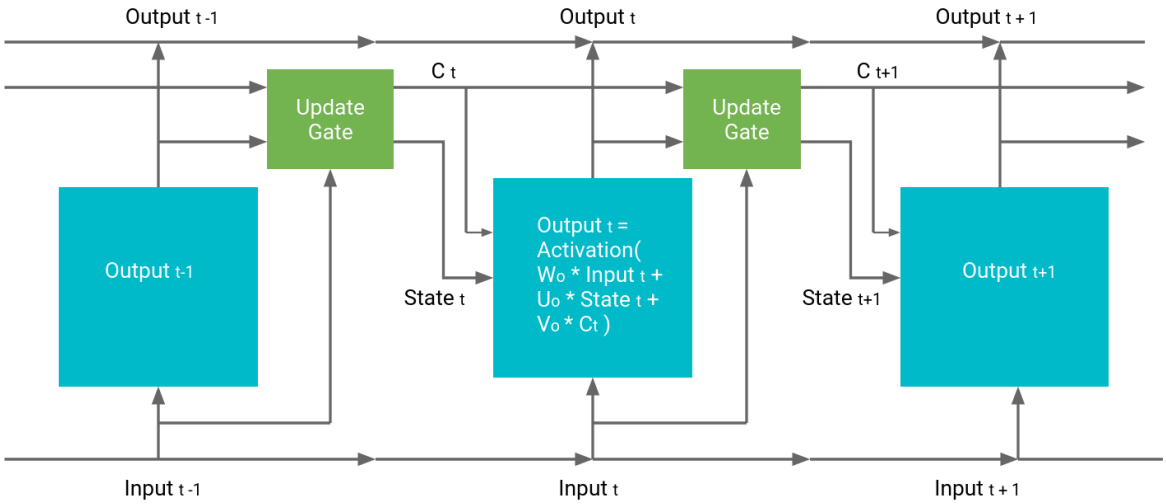


Figure 2.3: Schematic representation of a GRU cell

The most prominent feature shared between these units is the additive component of their update from t to $t + 1$, which is lacking in the traditional recurrent unit. The traditional recurrent unit always replaces the activation, or the content of a unit with a new value computed from the current input and the previous hidden state. On the other hand, both LSTM unit and GRU keep the existing content and add the new content on top of it.

This additive nature has two advantages. First, it is easy for each unit to remember the existence of a specific feature in the input stream for a long series of steps. Any important feature, decided by either the forget gate of the LSTM unit or the update gate of the GRU, will not be overwritten but be maintained as it is. Second, and perhaps more importantly, this addition effectively creates shortcut paths that bypass multiple temporal steps. These shortcuts allow the error to be back-propagated easily without too quickly vanishing (if the gating unit is nearly saturated at 1) as a result of passing through multiple, bounded nonlinearities, thus reducing the difficulty due to vanishing gradients [2]. These two units however have a number of differences as well. One feature of the LSTM unit that is missing from the GRU is the controlled exposure of the memory

content. In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate. On the other hand the GRU exposes its full content without any control. Another difference is in the location of the input gate, or the corresponding reset gate. The LSTM unit computes the new memory content without any separate control of the amount of information flowing from the previous time step. Rather, the LSTM unit controls the amount of the new memory content being added to the memory cell independently from the forget gate. On the other hand, the GRU controls the information flow from the previous activation when computing the new, candidate activation, but does not independently control the amount of the candidate activation being added (the control is tied via the update gate). From these similarities and differences alone, it is difficult to conclude which types of gating units would perform better in general. Although these two units perform comparably to each other, their relative performance might differ depend on the performed tasks.

2.2.3 Other RNN architectures

LSTM and GRU are two most widely used configurations of RNNs. Apart from these two, following different RNN architectures are used for different applications.

- Elman networks and Jordan network
- Hopfield Networks
- Independently Recurrent Neural Networks
- Neural History Compressors
- Recursive Neural Networks
- Second Order RNNs
- Bi-directional RNNs
- Continuous-time Networks
- Multiple Timescales Model Networks
- Neural Turing Machines
- Differentiable Neural Computer
- Neural Network Pushdown Automata
- Memristive Networks

2.3 Limitations of RNNs

In the beginning, Recurrent neural networks (RNNs) had two main issues which lead to difficulty in their training:

1. Vanishing Gradient Problem: RNNs can be viewed as a sequence of neural networks, where the activation of one network is input to the next network. These multiple networks are a replica of the same network in time steps because actually the output activation is again fed as input to the same network. Thus there are multiple networks with same weights, but different inputs and they are linked via activations. When there are many such linked networks (many time steps), the gradients of the activation propagate from next network to previous one and so on. The gradients multiply due to chain rule of differentiation. So if the gradient is close to zero in one network, it is likely that the gradient at the previous network is also very small. Also, as the gradients multiply, they are reduced further. This causes a chain of smaller gradients at each time step network. This is know as the vanishing gradient problem.

   The core problem with timeseries tasks is that the network needs to learn dependencies over long time windows, and the gradients can explode or vanish. To understand the vanishing gradients problem formally, consider an encoder-decoder architecture for machine translation, shown in Figure 2.4 The task is to read an

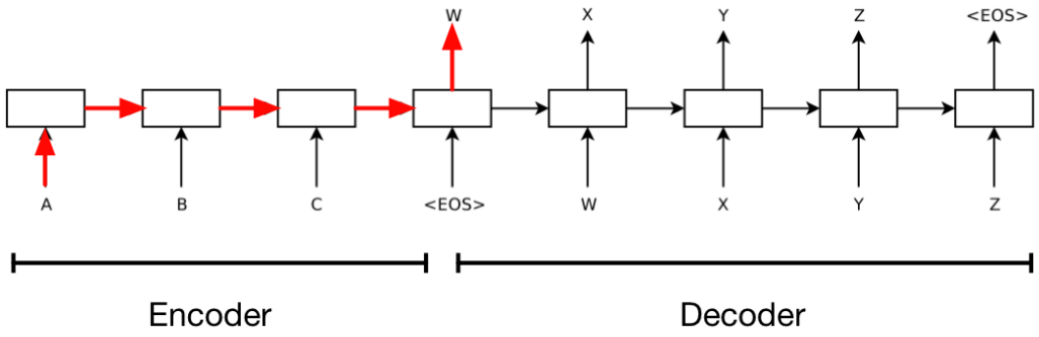


Figure 2.4: Encoder-decoder model for machine translation

English sentence, generate a context vector which stores as much information as possible, and output a French sentence. The information about the first word in the sentence doesn't get used in the predictions until it starts generating translation. Since a typical sentence might be about 20 words long, this means there's a

long temporal gap from when it sees an input to when it uses that to make a prediction. It can be hard to learn long-distance dependencies, for reasons described next. In order to adjust the input-to-hidden weights based on the first input, the error signal needs to travel backwards through this entire pathway (shown in red in Figure 2.4).

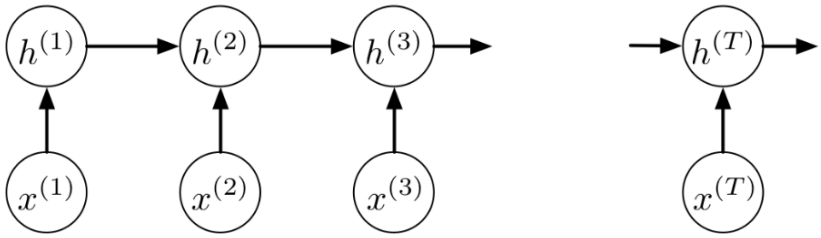


Figure 2.5: Backpropagation through encoder

As shown in the encoder section Figure 2.5, consider the error signal $h(T)$. The network then alternate between the following two backprop rules:

$$\bar{h}^{(t)} = \bar{z}^{(t+1)} w$$
$$\bar{z}^{(t)} = \bar{h}^{(t)} \phi'(z^{(t)})$$

Which gives, upon applying multiple times

$$\begin{aligned} \bar{h}^{(1)} &= w^{T-1} \phi'(z^{(2)}) \dots \phi'(z^{(T)}) \bar{h}^{(T)} \\ &= \frac{\partial h^{(T)}}{\partial h^{(1)}} \bar{h}^{(T)} \end{aligned}$$

Hence, $\bar{h}^{(1)}$ is a linear function of $\bar{h}^{(T)}$. The coefficient is the partial deriva- tive $\frac{\partial h^{(T)}}{\partial h^{(1)}}$. If we make the simplifying assumption that the activation functions are linear, we get

$$\frac{\partial h^{(T)}}{\partial h^{(1)}} = w^{(T-1)}$$

which can clearly explode or vanish unless w is very close to 1. For instance, if $w = 1.1$ and $T = 50$, we get $\partial h^{(T)}/\partial h^{(1)} = 117.4$, whereas if $w = 0.9$ and $T = 50$, we get $\partial h^{(T)}/\partial h^{(1)} = 0.00515$. In general, with nonlinear activation functions, there's nothing special about $w = 1$; the boundary between exploding and vanishing will depend on the values $h^{(t)}$. More generally, in the multivariate case,

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(T-1)}} \cdots\cdots \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}$$

This quantity is called the Jacobian. It can explode or vanish just like in the univariate case, but this is slightly more complicated to make precise. In the case of linear activation functions, $\partial \mathbf{h}^{(t+1)}/\partial \mathbf{h}^{(t)} = \mathbf{W}$, so

$$\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \mathbf{W}^{T-1},$$

This will explode if the largest eigenvalue of $\mathbf{W}$ is larger than 1, and vanish if the largest eigenvalue is smaller than 1. Contrast this with the behavior of the forward pass. In the forward pass, the activations at each step are put through a nonlinear activation function, which typically squashes the values, preventing them from blowing up. Since the backwards pass is entirely linear, there's nothing to prevent the derivatives from blowing up.

To give an intuitive understanding of vanishing gradients, there is a nice interpretation of the problem in terms of the function the RNN computes. In particular, each layer computes a function of the current input and the previous hidden activations, i.e. $\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t1)})$. If we expand this recursively, we get:

$$h^{(4)} = f(f(f(\mathbf{h}^{(1)}, \mathbf{x}^{(2)}), \mathbf{x}^{(3)}), \mathbf{x}^{(4)})$$

This looks a bit like repeatedly applying the function f . Therefore, we can gain some intuition for how RNNs behave by studying iterated functions, i.e. functions which we iterate many times. Iterated functions can be complicated. Consider the innocuous-looking quadratic function

$$f(x) = 3.5x(1 - x)$$

If we iterate this function multiple times (i.e. $f(f(f(x)))$, etc.), we get some complicated behavior, as shown in Figure 2.6

Figure 2.7 shows the function computed at each time step, as well as the function computed by the network as a whole. From this figure, it can be seen which regions have exploding or vanishing gradients.

2. Exploding Gradient Problem: The counterpart of vanishing gradient problem, where the gradients become successively larger and larger at each step called as exploding gradients problem.

3. The vanishing gradient problems in simple RNNs is only partially solved by LSTMs and Gated Units. This made them immensly popular for NLP and other sequence processing tasks. But these units still have a sequential path to follow, as the information flows from older past cells to the current one. The path in these is even

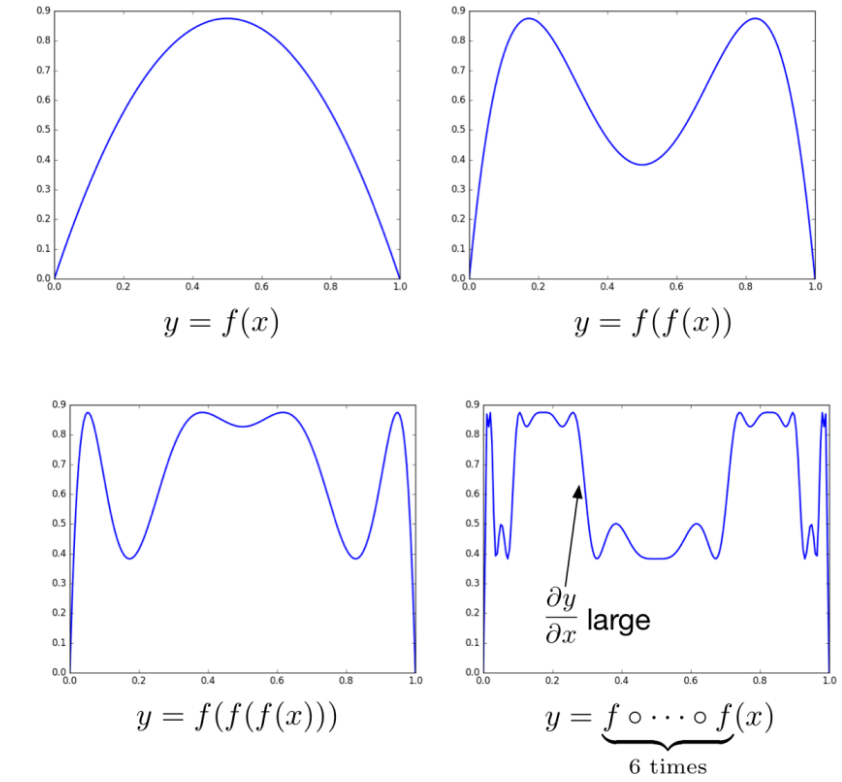


Figure 2.6: Iterations of the function $f(x) = 3.5x(1-x)$

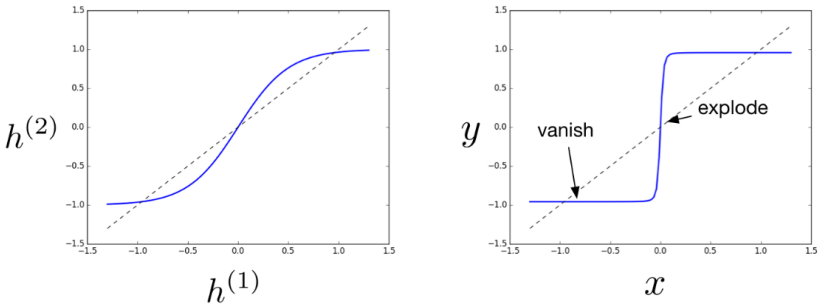


Figure 2.7: (left) The function computed by the RNN at each time step, (right) the function computed by the network

more complicated compared to simple RNNs, because it has additive and forget branches attached to it. Although LSTM, GRU and their derivatives are able to learn a lot of longer term information, they can remember sequences of 100s, not 1000s or 10,000s or more. This brings to attention one problem of RNN that they are not hardware friendly. It takes a lot of resources to train these network fast. Also it takes much resources to run these models in the cloud, and it is not scalable.

All these problems are leading to RNNs being replaced by attantion based models in the recent past.

In ESNs, the idea is that the sparse random connections in the reservoir allow previous states to ”echo” even after they have passed, so that if the network receives a novel input that is similar to something it trained on, the dynamics in the reservoir will start to follow the activation trajectory appropriate for the input and in that way can provide a matching signal to what it trained on, and if it is well-trained it will be able to generalize from what it has already seen, following activation trajectories that would make sense given the input signal driving the reservoir. The advantage of this approach is in the incredibly simple training procedure since most of the weights are assigned only once and at random. Yet they are able to capture complex dynamics over time and are able to model properties of dynamical systems. Thus in ESNs only the weights of readout layer are learnt during training, hence there are no vanishing or exploding gradients problems faced during ESN training.

RNNs were rarely used in practice before the introduction of ESNs. Because these models were trained using gradient descent algorithms to adjust the connections, it lead to vanishing and exploding gradient problems. Also, the algorithms were slow and made the learning process vulnerable to branching errors. Convergence of learning process could not be guaranteed. When ESNs were introduced, they did not suffer from these problems during training and additionally they were easy to implement. They also outperformed all other nonlinear dynamic models. However, with the advent of deep learning, the problems that made RNNs slow and error-prone were solved. Hence the unique selling point of ESNs was lost, although they provided first successful approach to training RNNs. We will discuss ESNs in detail in Chapter 4. But before that, in Chapter 3, we review the field of Reservoir Computing, which is the general class of networks, to which ESNs belong.

Chapter 3

Reservoir Computing

Reservoir computing (RC) is a field comprising of different machine learning techniques that use high-dimensional transient dynamics of an excitable system (*reservoir*), to perform classification or regression tasks. It is primarily applied for temporal information processing. These techniques are inspired from efficient bio-inspired approaches for processing time dependent data. Any dynamical system offering rich dynamics can, in principle, be used to build a reservoir. Various subfields of reservoir computing have emerged to address the demand for increasingly complex real-time signal processing methods. It represents an alternative to standard recurrent neural network models, providing advantage of faster training, over them.

Techniques of RC have been developed in three main types of methods: *Liquid State Machine*, *Echo State Network* and *Backpropagation-Decorrelation learning rule*. Each of these methods aim to promote a new approach to modeling complex dynamic systems in mathematical and engineering fields via an artificial Recurrent Neural Network. Each approach consists of a fixed-weight recurrent network that, when fed by an input dataset, outputs a series of activation states. These intermediate values are then used to train output connections to the second part of the system, which will output a description of original model's dynamics obtained from the data. The first part of the system, the reservoir, is a recurrent NN with fixed weights that acts as a "black-box" model of complex dynamic systems. The second part, known as readout layer, is a classifier layer of some kind, usually a simple linear one, connected by a set of weights to the reservoir. A fundamental property belonging to all these techniques is to have some kind of intrinsic memory effect, due to recurrent connections in the reservoir, whose size depends on the time steps needed to exhaust the effect of the k-th input in reservoir's computed output. During reservoir's construction, one of the major characteristic or hyper-parameter is the activation function used to characterize reservoir units' behavior. In literature, various models of artificial neurons from simple linear models to more elaborate non-linear ones

are found, such as the sigmoidal neuron in the "Echo state" and "Backpropagation-decorrelation's approach", or biological-inspired LIF model mainly employed in "Liquid State" technique.
The main methods which form the field of reservoir computing were conceived in the early 2000s by Herbert Jaeger and Wolfgang Maass independently. In the period after that, a variety of implementations have spawned from RC framework, due to its simplicity and flexibility. In this chapter we discuss Liquid State Machines and Backpropagation Decorrelation, while Echo State Machines is the subject of chapter 4.

3.1 Liquid State Machines

The Liquid State Machine (LSM) was proposed as a computational model that is more adequate for modelling computations in cortical microcircuits, than traditional models, such as Turing machines or attractor based models in dynamical systems. In contrast to these other models, the LSM is a model for real-time computations on continuous streams of data such as spike trains, i.e., sequences of action potentials of neurons that provide external inputs to a cortical microcircuit. In other words, both inputs and outputs of a LSM are streams of data in continuous time. These inputs and outputs are modelled mathematically as functions $u(t)$ and $y(t)$ of continuous time t. These functions are usually multi-dimensional, because they typically model spike trains from many external neurons that provide inputs to the circuit, and many different "readouts" that extract output spike trains. Since a LSM maps input streams onto output streams, it implements a functional or operator (like a filter).
Liquid State Machines have close links to the Echo State Networks, although these two theories were independently developed. The technical approach behind this idea is based on the concept of a "Liquid Computer" which can be explained as follows. Consider a liquid surface subjected to a series of transient perturbations by a sequence of external disturbances (inputs), such as wind, sound, or objects dropped into a liquid. If we consider the surface as being analogous to an attractor neural network, it has only one attractor state - the resting state - and may therefore seem useless for computational purposes. However, the perturbed state of the liquid, at any moment in time, represents present as well as past inputs, potentially providing the information needed for an analysis of various aspects of the environment. In order for such a liquid to serve as a source of salient information about present and past stimuli without relying on stable states, the perturbations must be sensitive to variations in inputs without being chaotic. The manner in which perturbations are formed and maintained would vary for different types of liquids and would determine how useful the perturbations are for anal-

ysis purpose. Limitations on the computational capabilities of liquids are imposed by their time-constant for relaxation, and the strictly local interactions as well as the homogeneity of the elements of a liquid. Neural microcircuits appear to be *ideal liquids* for computing on perturbations because of the large diversity of their elements, neurons and synapses, as well as the large variety of mechanisms and time constants characterizing their interactions, involving recurrent connections on multiple spatial scales.
A mathematical model of such a "liquid computer" is called Liquid State Machine (LSM) and consists of a Reservoir, in this case called liquid, which processes an input time-series $u(t)$ into a liquid state $x(t)$ which integrates influences from inputs at all times prior to t.
The primay characteristic of LSM approach consist of a model of the reservoir based on neurons with activation functions based on biological synapse model theorized by observing natural patterns found in the microcircuits of the brain. One example can be the Leaky Integrate and Fire (LIF) neuron model that in its basic form can be written as

$$I(t)\frac{V_m(t)}{R_m} = C_m\frac{dV_m(t)}{dt} \tag{3.1}$$

This is an evolution of the Integrate and Fire model which represents a neuron as the V-th time derivative of the law of capacitance $Q = CV$. The term "Leaky" refers to $I_{th} = V_{th}/R_m$ that is a threshold for the cell which can fire an output if the input intensity is sufficiently intense or cancel any change in the membrane potential. This type of node has an internal memory comparable with the leaky integrator neuron that competes with the reservoir intrinsic memory effect.

3.2 Backpropagation Decorrelation

Two years after the approaches proposed by Jaeger and Maass, another independent study on recurrent networks was published under the name of *Backpropagation-decorrelation (BPDC) learning rule* [22]. It is proposed as an effective and simple learning rule for online adaptation of recurrent networks, combining backpropagation, virtual teacher forcing, and decorrelation. Since this is applied only to the output weights, it yields a minimal $O(N)$ complexity at very good performance. The proposed learning rule combines three important principles

1. One-step back propagation of errors by means of the virtual teacher forcing.
2. Usage of the temporal memory in the network dynamics which is adapted based on decorrelation of the activations

3. The employment of a non-adaptive reservoir of inner neurons to reduce complexity.

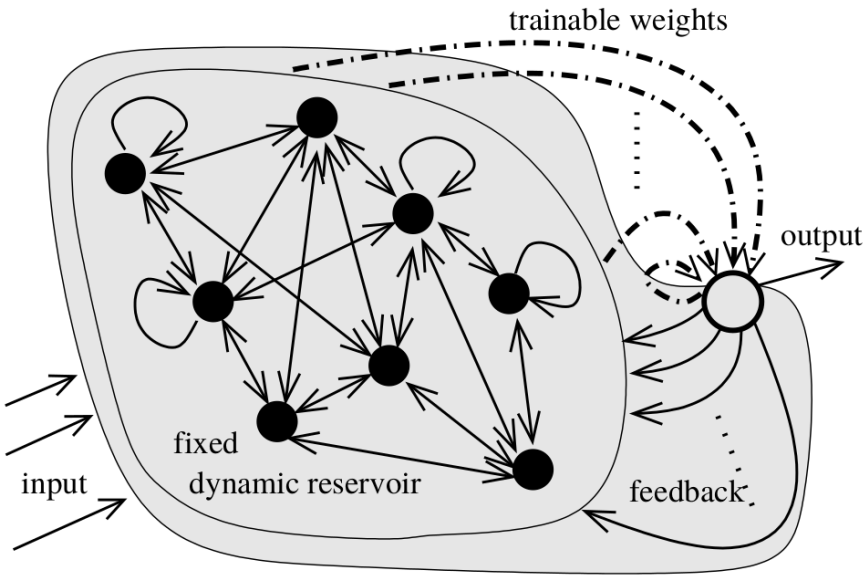


Figure 3.1: Schematic of a system implementing BPDC rule

Figure 3.1 show schematic representation of a system implementing BPDC rule. The state updates are given by

$$x(k+1) = (1 - \Delta t)x(k) + \Delta t W f(x(k)) \tag{3.2}$$

Where k represents discretised time, W is reservoir weight matrix and f is activation function of reservoir units, which is some kind of elementwise sigmoidal function. The units of the dynamical reservoir are triggered by the input signal and they provide a dynamical memory to the system. The system's information processing capacity is maximal when it's states are maximally decorrelated with respect to the given input. The output layer can optimally combine these states to read out the desired output. A compromise between error propagation and decorrelation is implemented by the Backpropagation-Decorrelation rule. The learning rule for the output weights is given by

$$\Delta w_{ij}(k+1) = \frac{\eta}{\Delta t} \frac{f(x_j(k))}{\sum_{i=1}^{N} f(x_s(k))^2 + \epsilon} \gamma_i(k+1) \tag{3.3}$$

where, γ_i is calculates as

$$\gamma_i(k+1) = \sum_{i=1}^{N} \left((1 - \Delta t)\delta_{is} + \Delta t w_{is} f'(x_s(k)) \right) e_s(k) e_i(k+1) \tag{3.4}$$

Here η is the learning rate, ϵ a regularization constant, and $e_s(k)$ are the non-zero error components reservoir unit s at time k, [$e_s(k) = x_s(k) - y_s(k)$] with respect to the teaching signal $y_s(k)$. The second term on rhs of 3.3 implements an approximative decorrelation rule. The γ_i propagates a mixture of the local errors $e_i(k+1)$, $e_i(k)$ and the errors in the last time step $e_s(k)$ weighted by the backpropagation term involving f'.
In most cases the output is taken via a single output neuron. In such cases the update can be computed with complexity of order proportional to $2N$.

3.3 Echo State Networks

Echo State Networks (ESNs) form the main topic of chapter 4. In that chapter, we will discuss construction, training, stability, applications and limitations of ESNs in detail. Here we briefly survey ESNs, in the way of overview of the field.
Echo State Networks have been primarily applied in time-series prediction tasks [27], speech recognition [37], noise modelling [27], dynamic pattern classification [26], reinforcement learning [3], and in language modelling [39]. A variety of extensions/modifications of the classical ESN are found in the literature, e.g. intrinsic plasticity [36], [38], refined training algorithms [27], training with Neuroscale [42], leaky-integrator reservoir units [24], support vector machine [35], setting the reservoir weights using Self-Organizing Maps (SOM) and Scale-Invariant Maps (SIM) [1], filter neurons with delay and sum readout [20], pruning connections within the reservoir [10] etc. There have also been attempts to impose specialised interconnection topologies on the reservoir, e.g. hierarchical reservoirs [23], small-world reservoirs [9] and decoupled sub-reservoirs [43]. Thus, there are many variations of ESNs built in different ways, reported in literature. They can be set up with or without directly trainable input-to-output connections, with or without output reservation feedback, with different neurotypes, different reservoir internal connectivity patterns etc. The output weight can be calculated for linear regression with all algorithms whether they are online or offline. In addition to the solutions for errors with smallest squares, margin maximization criteria, so-called training support vector machines, are used to determine the output values. The fixed RNN acts as a random, nonlinear medium whose dynamic response, the “echo”, is used as a signal basis. The linear combination of this basis can be trained to reconstruct the desired output by minimizing some error criteria.

Chapter 4

Echo State Networks

Echo state networks (*ESNs*) are variants of Recurrent Neural Networks. They belong to the reservoir computing framework. The main characteristics of ESNs can be described as:

- The network consists of input layer, hidden layer (*reservoir*) and readout or output layer.
- The weights between the input and the reservoir W_{in} and the weights of the reservoir units W, are not trainable. They are initialised using suitable random distributions.
- The only trainable weights belong to the output or the readout layer. They are trained on task specific data to reproduce specific temporal patterns.
- The connectivity of the reservoir units is very sparse (typically $< 10\%$).
- The reservoir creates a high dimensional recurrent nonlinear representation or embedding of the input, which is connected to the desired output through trainable weights.
- It is possible to connect the embeddings to different predictive models depending on the desired task output.

It was shown in [40] that, in training methods for RNNs, where all weights (not only the output weights) are adapted, the dominant changes take place in the output weights. ESNs take advantage of this fact by using fixed input and reservoir weights, thereby reducing the number of trainable parameters of the network.
Section 4.1 provides general overview of the field of ESNs. It starts with the mathematical details of the working of an ESNs. Section 4.2 defines echo state property and section 4.2.3 discusses its significance. The methodology for ESN training is discussed

in section 4.1.1 and section 4.1.2 introduces the topic of Deep ESNs. Section 5 reviews a research paper on ESNs with self normalizing activations. Section 4.4 discusses applications and limitations of ESNs. The chapter concludes with discussion about current ESN research areas in section 4.5.

4.1 Details of Echo State Networks

In this section we describe the operation of an ESN in a detailed step by step manner:

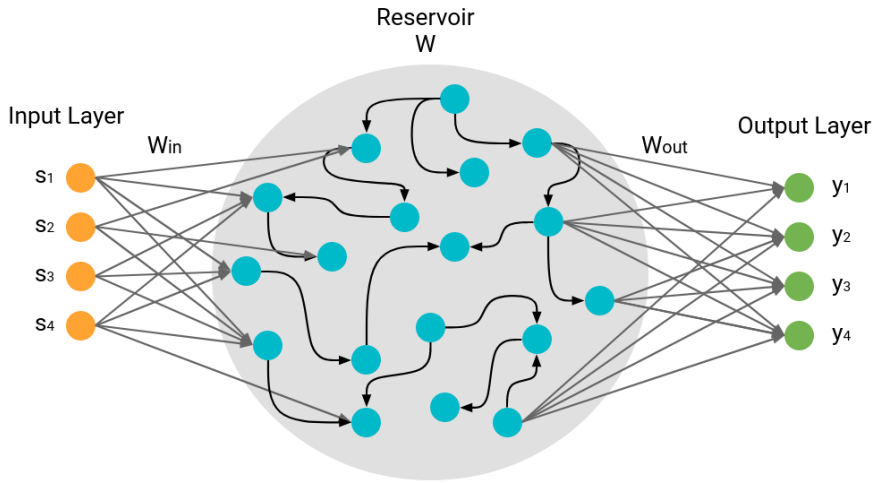


Figure 4.1: Schematic representation of ESN.

1. The input is represented by a matrix having shape (V, T), where T is the number of time steps and V is the number of features. Other parameters characterizing the network are:
   - Size of the reservoir N, specifying the number of units in the reservoir.
   - Sparsity of connections, indicating the fraction of non-zero entries in reservoir weight matrix W.
   - Whether to model a leakage.
   - Number of components after the dimensionality reduction.
2. Input weights W_{in} of dimension $(N \times V)$ are initialised by sampling from a binomial distribution.
3. Reservoir weights W, of dimension $(N \times N)$ are generated by sampling from a random uniform distribution with a given density. Density is the parameter which sets the level of sparsity.

4. The high dimensional state representation x is calculated as a non linear function (typically *tanh*) of the summation of input at the current time step k, (s_k) multiplied by the input weights and the previous state x_{k-1} multiplied by the reservoir weight matrix W.

$$a_k = W x_{k-1} + W_{in} s_k \tag{4.1}$$

$$x_k = \phi(a_k) \tag{4.2}$$

   The activation function ϕ can be:

   (a) *identity* function, for linear networks.

   (b) *tanh*, the hyperbolic tangent function.

5. A representation of the input is created by the reservoir. It can be in the form of a matrix of all calculated states or the mean of all states or the final value of state x_k.

6. This embedding is connected to the desired output, either by using a neural network or other types of predictors, which are trained on specific task of interest.

$$y_k = W_{out} x_k$$

4.1.1 Training ESNs

This section describes the detailed procedure for training an ESN on a given task. The output units are assumed to be sigmoid units and it is assumed that there are output-to-reservoir feedback connections. The procedure can be adapted easily to special cases such as linear output units, no output-to-reservoir feedback connections, or even systems without input (such as the pure sinewave generator), also known as autonomous systems.
Given: A training input/output sequence $(u(1), d(1)), \ldots, (u(T), d(T))$.
Determine: A trained ESN $[W_{in}, W, W_{back}, W_{out}]$ whose output $y(n)$ approximates the teacher output $d(n)$, when the ESN is driven by the training input $u(n)$.
The primarily goal is not to obtain a good approximation of only the teacher output, but more importantly, to produce a good approximation of testing output data from independent test data sets generated by the same (unknown) system which also generated the teacher data. This concerns with achieving a good generalization performance of the trained model, which is the central problem of the statistical learning theory.

1. The first step is to start with an untrained ESN (W_{in}, W, W_{back}) which has the echo state property, and whose internal units can produce desired rich dynamics when excited. The weights can be determined in following manner.

(a) Randomly generate an internal weight matrix W_0.

(b) Normalize W_0 to a matrix W_1 with unit spectral radius by $W_1 = W_0/|\lambda_{max}|$, where $|\lambda_{max}|$ is the spectral radius of matrix W_0.

(c) Scale W_1 to $W = \alpha W_1$, where $\alpha < 1$, whereby W obtains a spectral radius of α.

(d) Randomly generate input weights W_{in} and output backpropagation weights W_{back}.

(e) The untrained network (W_{in}, W, W_{back}) obtained using above steps is **empirically** found to be an echo state network.

2. The network training dynamics are determined in the following manner:

(a) Initialize the network state arbitrarily, e.g., to zero state $x(0) = 0$.

(b) Input training data is provided to the network, for times $n = 0, \ldots, T$, by presenting the teacher input $u(n)$, and desired teacher-forced output $d(n-1)$. The network state is updated by computing

$$x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}d(n)) \tag{4.3}$$

(c) At time $n = 0$, where $d(n)$ is not defined, it is set to $d(n) = 0$.

(d) For each time step greater than an initial washout time T_0, the network state $x(n)$ is collected as a new row into a state collecting matrix $\mathbf{M}$. In the end, one has obtained a state collecting matrix of size $(T - T_0) \times (K + N + L)$.

(e) Similarly, for each step greater than T_0 , the sigmoid-inverted teacher output $\tanh^{-1} d(n)$ is collected into rows of a teacher collection matrix $\mathbf{T}$. This gives a teacher output collection matrix T of size $(T - T_0) \times L$.

3. Compute output weights.

(a) The pseudoinverse of $\mathbf{M}$ is multiplied with $\mathbf{T}$, to obtain a $(K + N) \times L$ sized matrix $(W^{out})^T$ whose i-th column contains the output weights from all reservoir units to the i-th output unit:

$$(W^{out})^T = \mathbf{M}^{-1}\mathbf{T} \tag{4.4}$$

(b) Transpose $(W^{out})^T$ to W_{out} in order to obtain the desired output weight matrix.

4. Exploitation

   The network $(W_{in}, W, W_{back}, W_{out})$ is now ready for use. It can be driven by a novel input sequences $u(n)$, using the update equations:

$$x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}y(n)) \tag{4.5}$$

$$y(n+1) = f^{out}(W^{out}(u(n+1), x(n+1)) \tag{4.6}$$

4.1.2 Deep Echo State Networks

Keeping in line with the trends in other neural network architectures, deep ESNs were proposed in [14], [11] . DeepESNs consist of stacks of recurrent layers. The interest in the study of the DeepESN model is twofold. On the one hand, it allows to shed light on the intrinsic properties of state dynamics of layered RNN architectures. On the other hand it enables the design of extremely efficiently trained deep neural networks for temporal data. Previous to the explicit introduction of the DeepESN model, works on hierarchical RC models focussed on architectures, where different modules were trained for discovery of temporal features at different scales on synthetic data. Ad-hoc constructed modular networks made up of multiple ESN modules have also been investigated in the speech processing area. The advantages of multi-layered RC networks have been experimentally studied on time-series benchmarks in the RC area [33]. Differently from the above mentioned works, [16] addressed fundamental questions pertaining to the true nature of layering such as:

- Necessity of stacking layers of recurrent units
- What is the inherent architectural effect of layering in RNNs (independently from learning)?
- Can we extend the advantages of depth in RNN design using efficiently trained RC approaches?
- Can we exploit the insights from such analysis to address the automatic design of deep recurrent models (including fundamental parameters such as the architectural form, the number of layers, the number of units in each layer, etc.)

Simillar to the standard shallow ESN model, a DeepESN is also composed by a dynamical reservoir component, which embeds the input history into a rich state representation, and a feed-forward readout part, wich exploits the state econding provided by the reservoir to compute the output. The reservoir of a DeepESN is organized into a hierarchy of stacked recurrent layers, where the output of each layer acts as input for the next one, as illustrated in Figure 4.2. In deepESN, at each time step t, the state computation proceeds through the pipeline of recurrent layers, from the first one, which is directly fed

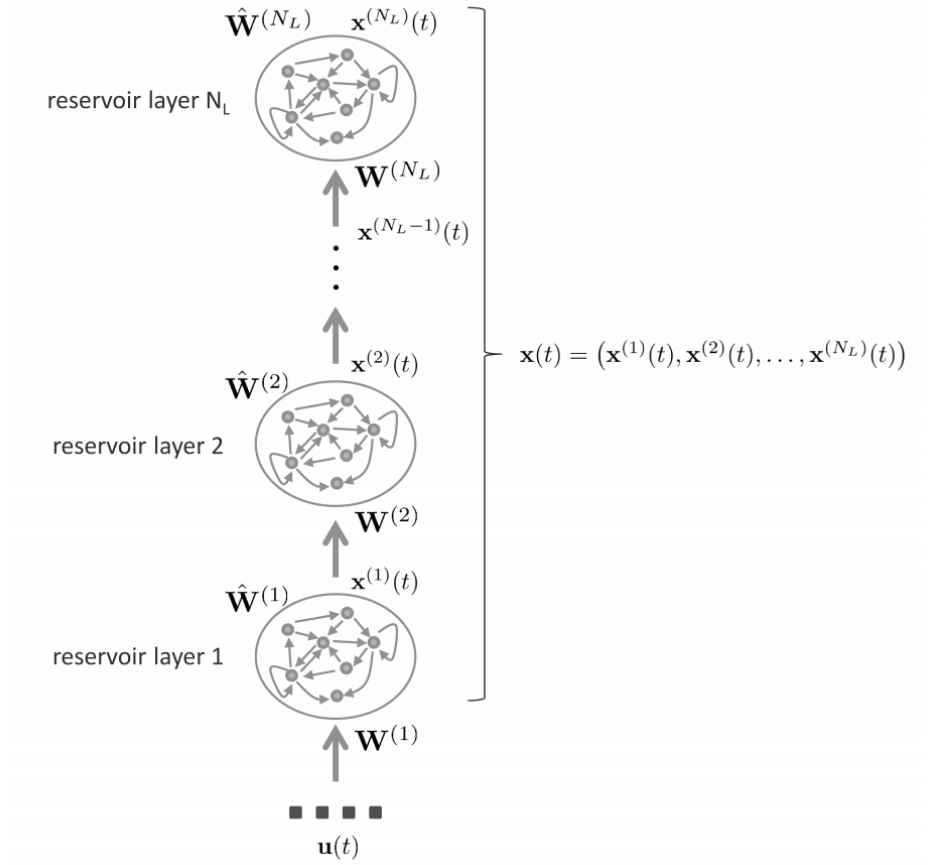


Figure 4.2: Deep Echo State Network Schematic.

by the external input, up to the highest one in the reservoir architecture. Let N_U denote the external input dimension, N_L indicate the number of reservoir layers, N_R denote number of recurrent units in each reservoir layer. Let $u(t) \in R^{N_U}$ denote the external input at time step t, while $x^{(i)}(t) \in R^{N_R}$ be the state of the reservoir layer i at time step t. The superscript (i) indicates that an item is related to the i-th reservoir in the stack. At each time step t, the composition of the states in all the reservoir layers, i.e. $x(t) = x^{(1)}(t), \ldots, x^{(NL)}(t) \in R^{N_R N_L}$, gives the global state of the network.
The computation carried out by the stacked reservoir of a DeepESN can be understood under a dynamical system viewpoint as an input-driven discrete time non-linear dynamical system, where the evolution of the global state $x(t)$ is governed by a state transition function $F = F^{(1)}, \ldots, F^{(N_L)}$, with each $F^{(i)}$ ruling the state dynamics at layer i. Assuming leaky integrator reservior units in each layer and omitting the bias terms for the ease of notation, the reservoir dynamics of a DeepESN are mathematically described as

follows. For the first layer we have that:

$$x^{(1)}(t) = F(u(t), x^{(1)}(t-1)) \tag{4.7}$$
$$= (1 - a^{(1)})x^{(1)}(t-1) + f(W^{(1)}u(t) + \hat{W}^{(1)}x^{(1)}(t-1)) \tag{4.8}$$

while for successive layers $i > 1$ the state update is given by:

$$x^{(i)}(t) = F(x^{(i-1)}(t), x^{(i)}(t-1)) \tag{4.9}$$
$$= (1 - a^{(i)})x^{(i)}(t-1) + f(W^{(i)}x^{(i-1)}(t) + \hat{W}^{(i)}x^{(i)}(t-1)) \tag{4.10}$$

$W^{(1)} \in R^{N_R \times N_U}$ is the input weight matrix, $W^{(i)} \in R^{N_R \times N_R}$ for $i > 1$ is the weight matrix for inter-layer connections from layer $(i-1)$ to layer i, $\hat{W}^{(i)} \in R^{N_R \in N_R}$ is the recurrent weight matrix for layer i, $a^{(i)} \in [0, 1]$ is the leaking rate for layer i and f denotes the element-wise applied activation function for the recurrent reservoir units (typically, $tanh$). The layered reservoir architecture of DeepESN can be interpreted as a constrained version of a shallow reservoir as indicated in Fig. 4.3. Compared to the shallow case with the same total number of recurrent units, in a stacked DeepESN architecture the following connections are removed: from the input to reservoir levels at height > 1 (blue dashed arrows), from higher to lower reservoir levels (green dash dotted arrows), from each reservoir at level i to all reservoirs at levels $> i + 1$ (orange dotted arrows).
The salient features and recent advances in the DeepESN models is summarised in following points:

- The DeepESN model were formally introduced in [14], extending the preliminary work in [11] . Their main characteristic is the hierarchical structure of temporal data representations developed by the layered reservoir architecture. Specifically, the stacked composition of recurrent reservoir layers enables a multiple time-scales representation of the temporal information, naturally ordered along the network's hierarchy.

- The hierarchically structured state representation in DeepESNs was investigated by means of frequency analysis in [6]. The case of recurrent units with linear activation functions was specifically considered in it. Result showed the intrinsic multiple frequency representation in DeepESN states, where, even in the simplified linear setting, progressively higher layers focus on progressively lower frequencies. The potentiality of the deep RC approach was also exploited in predictive experiments, showing that DeepESNs outperform state-of-the-art results on the class of Multiple Superimposed Oscillator (MSO) tasks by several orders of magnitude.

- The fundamental RC conditions related to the Echo State Property (ESP) is generalized to the case of deep RC networks in [5]. A sufficient condition and a necessary

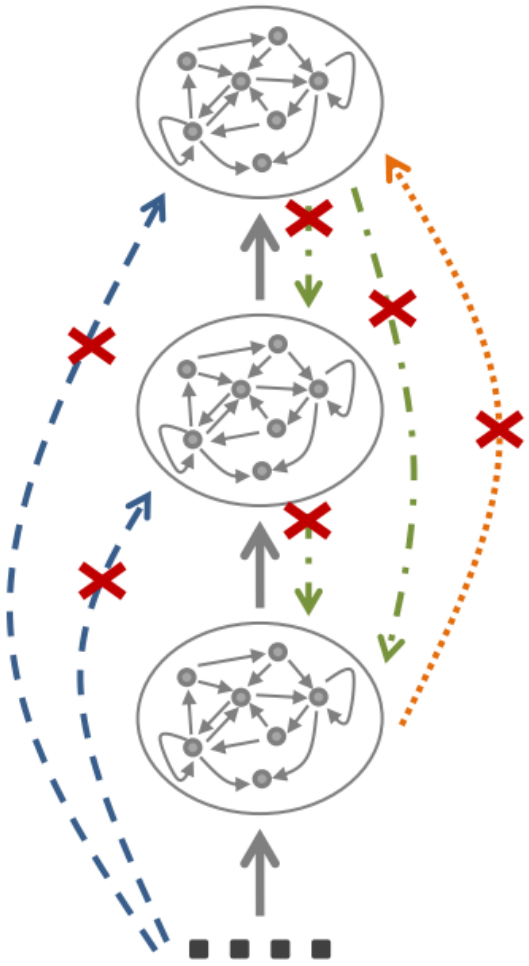


Figure 4.3: DeepESN differnece w.r.t. shallow ESN.

condition for the Echo State Property, is given through the study of stability and contractivity of nested dynamical systems for the case of deep RNN architectures. It provides a relevant conceptual and practical tool for the definition, validity and usage of DeepESN in an "autonomous" way with respect to the standard ESN model.

- The frequency spectrum study of deep reservoirs addressed one of the fundamental open issues in deep learning, namely how to choose the number of layers in a deep RNN architecture. Starting from the analysis of the intrinsic differentiation of the filtering effects of successive levels in a stacked RNN architecture, the work in [15] proposed an automatic method for the design of DeepESNs. The proposed approach allowed to tailor the DeepESN architecture to the characteristics of the

input signals, consistently relieving the cost of the model selection process, and leading to new state-of-the-art results in speech and music processing tasks.

- An extension of the deep RC framework for learning in structured domains has been presented in [13], [12], in the form of the Deep Tree Echo State Network (DeepTESN) model. This model pointed out that it is possible to combine the concepts of deep learning, learning for trees and RC training efficiency, taking advantages from the layered architectural organization and from the compositionality of the structured representations both in terms of efficiency and in terms of effectiveness. On the application side, the experimental results concretely showed that the deep RC approach for trees can be extremely advantageous, beating previous state-of-the-art results in challenging tasks from domains of document processing and computational biology. As regards the mathematical description of the model, the reservoir operation is extended to implement a (non-linear) state transition system over discrete tree structures, whose asymptotic stability analysis enables the definition of a generalization of the ESP for the case of tree structured data. Overall, DeepTESN provides a first instance of an extremely efficient approach for the design of deep neural networks for learning in cases where the input data is represented in the form of tree structures.

- DeepESNs were shown to bring several advantages in both cases of synthetic and real-world tasks. Specifically, DeepESNs outperformed shallow reservoir architectures (under fair conditions on the number of total recurrent units and on the number of trainable readout parameters) on the Mackey-Glass next-step prediction task, on the short-term Memory Capacity task, on MSO tasks, as well as on a Frequency Based Classification task, purposely designed to assess multiple-frequency representation abilities. The DeepESN approach recently proved effective in a variety of domains, including Ambient Assisted Living (AAL), medical diagnosis, speech and polyphonic music processing.

- Following software implementations of the DeepESN models are publicly available:

  - DeepESN Python Library (DeepESNpy)

    `https://github.com/lucapedrelli/DeepESN`

  - Deep Echo State Network (DeepESN) MATLAB Toolbox

    `https://it.mathworks.com/matlabcentral/fileexchange/69402-deepesn`

  - Deep Echo State Network (DeepESN) Octave library

    `https://github.com/gallicch/DeepESN_octave`

4.2 Echo State Property

In order for the ESN principle to work, the reservoir must have the *echo state property (ESP)*. For a network satisfying ESP, the activation state $x(n)$ of the reservoir is a function of the input history $[u(n), u(n-1), \dots]$, i.e., A function E exists such that,

$$x(n) = E(u(n), u(n-1), \dots) \tag{4.11}$$

The name ESN stems from the fact that, for networks satisfying this echo state property, the state $x(n)$ can be considered as an *"echo"* of the input history.

4.2.1 General ESP for all inputs

To formally define ESP, consider a discrete-time neural networks with K input units, N internal network or reservoir units and L output units. The activations of input, reservoir and output units at time step n are denoted as:

$$u(n) = (u_1(n), \dots, u_K(n)) \tag{4.12}$$
$$x(n) = (x_1(n), \dots, x_N(n)) \tag{4.13}$$
$$y(n) = (y_1(n), \dots, y_L(n)) \tag{4.14}$$

In the general case, there are connections from input to output, and within output neurons. Thus output layer is characterised by matrix W_{out} of size $(L \times (K+N))$. Additionally, there can be back connections from output to reservoir, specified through W_{back}, a $(N \times L)$ dimensional matrix. The formula for state update is given by:

$$x(n+1) = f(W_{in}u(n+1) + Wx(n) + W_{back}y(n)) \tag{4.15}$$

where f is the internal unit's activation functions (typically a sigmoid function). The output is computed according to

$$y(n+1) = f_{out}(W_{out}([u(n+1), x(n+1)])) \tag{4.16}$$

where f_{out} is the output unit's output functions and $([u(n+1), x(n+1), y(n)])$ is the concatenation of the input, internal, and previous output activation vectors. With these notations, the Echo State Property can be defined as follows [25]:

Definition: Assuming a network with no output feedback connections, the network is said to have echo states, if the network state $x(n)$ is uniquely determined by any left-infinite input sequence $\bar{u}^{-\infty}$. This means that for every input sequence $\dots, u(n-1), u(n) \in U^{-N}$, for all state sequences $\dots, x(n-1), x(n)$ and $\dots x'(n-1), x'(n) \in A^{-N}$,

where $x(i) = T(x(i-1), u(i))$ and $x'(i) = T(x'(i-1), u(i))$[1], it holds that $x(n) = x'(n)$.

Stated in alternate terms, the echo state property says, "if the network has been run for a very long time (from minus infinity time in the definition), the current network state is uniquely determined by the history of the input and the (teacher-forced) output". Equivalently, for every internal signal $x^i(n)$ there exists an echo function E^i which maps input/output histories to the current state:

$$E^i : (U \times D)^{-N} \to R \tag{4.17}$$

$$E^i(\ldots(u(-1), d(-2)), (u(0), d(-1))) \to x(0) \tag{4.18}$$

The properties of being state contracting, state forgetting, and input forgetting are all equivalent to the network having echo states.

4.2.2 ESP linked to inputs

In many cases, the simple criterion of the spectral radius of the reservoir weight matrix being smaller than unity, is not sufficient to satisfy the echo state property, as discussed in [44]. They have demonstrated this using analytical examples investigating local bifurcation properties of the standard ESNs. The paper proposes updating the formal definition of the echo state property based on new sufficient conditions for ESP given by them, for ESNs using sigmoid or leaky integrator neurons. The paper proves that: A standard recurrent network of the form given by Eq. 4.15 with internal weight matrix W satisfies the echo state property for any input if W is diagonally *Schur stable*[2], i.e., there exists a diagonal $P > 0$ such that $W^T PW - P$ is negative definite. The earlier propositions and theorems involving consider only the reservoir weights, disregarding any impact the driving input might have on the ESP. In other words the ESP was considered to be a property of the resrevoir. In [34], the ESP is considered as a property of reservoir-input pair. The definition of input linked ESP is given as:

Definition: Let $g : U \times X \to X$ be an input driven system, where X is compact and U is complete. A sequence $x_{(-\infty,0]} := (\ldots, x_{-1}, x_0) \subset X$ is said to be compatible with $u_{(-\infty,0)} := (\ldots, u_{-2}, u_{-1}) \subset U$ when $x_{k+1} = g(u_k, x_k)$for all $k < 0$. The input driven system $g : U \times X \to X$ has the echo state property with respect to U if for any given

[1] T is the network state update operator defined by $x(n+h) = T(x(n), y(n), \bar{u}^h)$

[2] A matrix $W \in R^{N \times N}$ is called Schur stable if there exists a positive definite symmetric matrix $P > 0$ such that $W^T PW - P$ is negative definite. If the matrix P can be chosen as a positive definite diagonal matrix, then W is called diagonally Schur stable. The positive definite and negative definite matrices are denoted by $P > 0$ and $P < 0$, respectively.

$u_{(-\infty,0)} \subset U$ and sequences $x_{(-\infty,0]}, y_{(-\infty,0]} \subset X$, both compatible with $u_{(-\infty,0)}$, the equality $x_0 = y_0$ holds.

The motivation for defining ESP with respect to input signal can be summarized as below:

In addition to the reservoir characteristics, the echo state property is intrinsically tied to the characteristics of the driving input. It is possible that for a specific reservoir, given inputs of some kind, initial states may not be washed out, while for others they might be. Therefore, the ESP may not be considered as the property of a reservoir alone, but a property of the pair (reservoir, "admissible inputs"). The available definitions and conditions relating to the ESP characterize the admissible inputs by their value range. The input are constrained to take values from compact set U, from which the ESP becomes a property of a pair (reservoir, U). This is the commonly used mathematical treatment setting so far, even though it becomes irrelevant for many applications of reservoir computing. To understand the limitations associated with specifying admissible inputs solely through their range, consider a standard discrete-time reservoir RNN with a *tanh* activation function. From the functional form of *tanh*, it is seen that, it is relatively more contractive for larger-amplitude neural activations as compared to small amplitude ones because the slope of the *tanh* is greatest around zero. Hence, larger arguments become more strongly contracted by the asymptotic regions of *tanh* function. Thus, when a *tanh* reservoir is driven by large-amplitude input, the reservoir neurons will become highly excited, the *tanh* quenches strongly, which results in an overall initial condition forgetting. However, for small amplitude input, the washing out of initial condition may not happen. Specifically, for a constant zero input the ESP might be lost. In a practical application, the relevant input range contains zero. Which means $0 \in U$, and the constant zero signal is an admissible input, which has to be accommodated into ascertaining the ESP. This is unrealistic as one will never encounter the constant-zero input in most applications. But this leads to unnecessarily strict constraints on the reservoir weight matrix because the ESP also has to be guaranteed for the zero input signal. In RC literature, the weight matrix is scaled to ensure the ESP (which leads to suboptimal scaling of the weight matrix). In some studies, a weight matrix scaling that formally aborts the ESP is used but it still works well nonetheless, because the input signal is strong enough. In some other published work, it results in the incorrect scaling of the reservoir weight matrix to the border of chaos by setting it such that the ESP for zero input gets lost. These factors make it necessary to have an alternative definition of the ESP that respects the nature of the expected input signals in more detail than just through fixing their range. [34] provides such an alternative definition, by defining the ESP for a specific single-input signal, $\{u_n\}_{n \in Z}$. This definition covers general input

driven dynamical systems in addition to RNNs, provided their state space is compact. From this single-input-signal-based definition of the ESP, they derive a general $0-1$ law (that if the ESP is obtained for a particular input signal, then with probability 1, it is also obtained for other inputs from the same source).

The state evolution in a input driven system (IDS) is studied through the orbits or solutions. Any sequence $\{\theta_n\} \subset X$ is called an entire solution of the IDS $g : U \times X \to X$, if there exists some $\{u_n\} \subset U$ such that $\theta_{n+1} = g(u_n, \theta_n)$ for all $n \in Z$.

In [34], the following proposition is proved, which is result of ESP with regard to input space, in terms of entire solutions:

Proposition: Suppose $g : U \times X \to X$ is an input driven system that has the echo state property with respect to U. If $x_{(-\infty,0]}$, $y_{(\infty,0]} \subset X$ are both compatible with $u_{(\infty,0)}$, then $x_k = y_k$ for all $k \leq 0$. As a consequence, for any input sequence $\{u_n\}_{n\in Z}$, there exists at most one entire solution.

With this, a definition of ESP with respect to inputs is given as [34]:

Definition: An input-driven system $g : U \times X \to X$ is said to have the echo state property with respect to an input sequence $\{u_n\}$ if there exists exactly one entire solution - if $\{\upsilon_n\}$ and $\{\theta_n\}$ are entire solutions, then $\upsilon_n = \theta_n$ for all $n \in Z$.

Having Echo State Property means that the network, for a given input sequence, will asymptotically produce the same sequence of states, following the same trajectory (which can be very complex), in response to that input, regardless of initial conditions, i.e., it will "forget" any initial internal state. For networks not driven by inputs, or autonomous systems, it is possible to have multiple attractors simultaneously for solving specific tasks that involve some degree of memory. In [7] a generalisation of the ESP for multistable, nonautonomous systems driven by inputs is proposed. It is demostrated that existence of a unique local point attractor in phase space coincides with the presence of ESP. A system driven by input $u \in U$ is said to have echo index n if it admits a proper decomposition into n uniform local point attractors. The system is then said to have the *n-Echo State Property* (n-ESP) for input u. It is proved that the ESP corresponds to existence and uniqueness of a global pullback attractor with specific features. The paper also reports numerical experiments showing different numbers of local point attractors and highlighting their input-driven nature.

The details of the $0-1$ law mentioned above, are explained next. Consider an input driven system $g : U \times X \to X$ with an input obtained as a realization of a U-valued stationary ergodic process $\{\xi_n\}$ defined on a probability space (Ω, F, P) - that is, for each ω and n, $\xi_n()$ takes values in the set U. Each realization $\{\xi_n()\}$, where $\omega \in \Omega$ gives rise to a separate nonautonomous system and hence has its own natural association $\{X_n()\}$. We thus consider $\{X_n\}$ to be a set-valued stochastic process. With these, the following

Lemma is proved in [34].

Lemma: Consider an IDS $g : U \times X \to X$ and a U-valued ergodic process $\{\xi_n\}$ defined on (Ω, F, P). For each realization $\{\xi_n(\omega)\}, \omega \in \Omega$, define the process $\phi(n, m, x) := g_{n1,} \dots g_{m,\omega}(x)$, on X, where $g_{n,}() := g(\xi_n(),) : X \to X$. Let $\{X_n\}$ be the set-valued stochastic process where $\{X_n()\}$ is the natural association of the process ϕ ($X_n() \in \mathbf{H}X$). Define $\gamma : \mathbf{H}_X \to \mathbb{R}$ by

$$\gamma(A) := 0 : diam(X) = 0$$
$$:= 1 : otherwise$$

Then $\{\gamma(X_n)\}$ is an ergodic stochastic process. Along with the following theorem.

Theorem: Let $\{\xi_n\}$ be a U-valued ergodic process defined on (Ω, F, P) and $g : U \times X \to X$ an input-driven system. Then the set of all $\omega \in \Omega$ such that g has the echo state property, that is, the subset of Ω, $\{\omega : g$ satisfies ESP with $\{\xi_n(\omega)\}\}$, has either probability 1 or 0.

4.2.3 Significance of Echo State Property

As discussed earlier, the dynamics of an ESN is governed by an update equation of the form

$$d(n) = E(u(n), u(n-1), \dots, d(n-1), d(n-2), \dots) \tag{4.19}$$

where E is a possibly highly complex and nonlinear function of the previous inputs and system outputs. Eq. 4.19 is a general equation for stationary deterministic system. It can be simplified in a case, for example, where E is linear and has only finitely many arguments (i.e., the system has finite memory), for many practical engineering applications. The task of finding a model for an unknown system given by Eq. 4.19 amounts to finding a good approximation to the system function E. To facilitate notation, assuming an ESN with linear output units, the network output of the trained network can be written as a linear combination of the network states, which in turn are governed by the echo functions.

$$d(n) = E(u(n), u(n-1), ..., d(n-1), d(n-2), \dots) \tag{4.20}$$
$$\approx y(n) \tag{4.21}$$
$$= \sum W^i_{out} x^i(n) \tag{4.22}$$
$$= \sum W^i_{out} E^i(u(n), u(n-1), ..., y(n-1), y(n-2)) \tag{4.23}$$

It becomes clear from Eqs.4.20-4.23 how the desired approximation of the system function E can be interpreted as a linear combination of echo functions E^i. This interpretatation of the system approximation task interprets the network states as echo

states. The arguments of E and E^i are identical in nature and both are collections of previous inputs and system outputs. Without echo states it is not possible to mathematically understand the relationship between network output and original system (which the network is trying to model) output, nor would the training algorithm work.

4.3 Universal function approximation property

It has been proved in ESN literature [17] that ESNs of the form $x_k = \phi(a_k)$ are universal function approximators, provided that they have the Echo State Property(ESP). In this subsection we first sketch the outline of that proof. A reservoir filter can be approximated by an an approximant for the reservoir system generating that filter [17]. Thus, a density statement in a space of operators between infinite dimensional spaces can be proved in a space of functions with finite dimensional variables and values. Which means simple approximating reservoir filters can be found for any reservoir system with fading memory property. Since ESNs are natural generalizations of neural networks in a dynamic learning system, they can be used as such approximators. The Euclidean norm for finite dimensional spaces $\mathbf{x} \in \mathbb{R}^n$, is defined as $||\mathbf{x}|| := (\sum_{i=1}^{n} x_i^2)^{1/2}$. The symbol $B_{||.||}(O, M)$ represents the open balls with Euclidean norm, for $M > 0$, with $I_n := B_{||.||}(O, 1)$.

Theorem: Let $U : I_n^{\mathbb{Z}_-} \to (\mathbb{R}^d)^{\mathbb{Z}_-}$ be a causal and time-invariant filter that has the fading memory property. Then, for any $\epsilon > 0$ and any weighting sequence w, there is an echo state network:

$$\mathbf{x}_t = \sigma(A\mathbf{x}_{t-1} + C\mathbf{z}_t + \zeta) \tag{4.24}$$

$$\mathbf{y}_t = W\mathbf{x}_t \tag{4.25}$$

whose associated generalized filters $U_{ESN} : I_n^{\mathbb{Z}_-} \to (\mathbb{R}^d)^{\mathbb{Z}_-}$ satisfy that

$$|||U - U_{ESN}|||_\infty < \epsilon \tag{4.26}$$

Proof: A non-homogeneous state-affine system is a reservoir system specified by:

$$\mathbf{x}_t = p(\mathbf{z}_t)\mathbf{x}_{t-1} + q(\mathbf{z}_t) \tag{4.27}$$

$$\mathbf{y}_t = W_1\mathbf{x}_t \tag{4.28}$$

where, $\mathbf{z}_t \in I_n := B_{||.||}(0, 1)$, $\mathbf{x} \in \mathbb{R}^{N_1}$ for $N_1 \in \mathbb{N}$ and $W_1 \in \mathbb{M}_{d,N_1}$. The polynomials $p(\mathbf{z}_t)$ and $q(\mathbf{z}_t)$ have degrees r and s with matrix coefficient have form:

$$p(\mathbf{z}) = \sum_{i_1,\ldots,i_n \in \{0,\ldots,r\}} z_1^{i_1} \ldots z_n^{i_n} A_{i_1,\ldots,i_n}, A_{i_1,\ldots,i_n} \in \mathbb{M}_{N_1}, \mathbf{z} \in I_n \tag{4.29}$$

$$q(\mathbf{z}) = \sum_{i_1,\ldots,i_n \in \{0,\ldots,s\}} z_1^{i_1} \ldots z_n^{i_n} B_{i_1,\ldots,i_n}, B_{i_1,\ldots,i_n} \in \mathbb{M}_{N_1,1}, \mathbf{z} \in I_n \tag{4.30}$$

For $L > 0$ amd

$$0 < K < \frac{L}{L+1} < 1 \tag{4.31}$$

for any $\epsilon_1 > 0$, there exists a state-affine system (SAS) filter $U^{p,q}_{W_1}$, for which:

$$|||H_U - H^{p,q}_{W_1}|||_\infty < \epsilon_1$$

where, H_U and $H^{p,q}_{W_1}$ are reservoir functionals associated to U and $U^{p,q}_{W_1}$ respectively. This implies that

$$|||U - U^{p,q}_{W_1}|||_\infty < \epsilon_1$$

To show that SAS filter $U^{p,q}_{W_1}$ can be approximated by a fiter generated by ESN, define the map

$$F_{SAS} : \overline{B_{||.||}(0, L)} \times I_n \to \mathbb{R}^{N_1} \tag{4.32}$$

$$(\mathbf{x}, \mathbf{z}) \mapsto p(\mathbf{z})\mathbf{x} + q(\mathbf{z}) \tag{4.33}$$

The choice of K ensures that the map F_{SAS} is a contraction since $K < 1$ and $||F_{SAS}||_\infty < L$. This implies that the map F_{SAS} maps into $\overline{B_{||.||}(0, L)}$

$$F_{SAS} : \overline{B_{||.||}(0, L)} \times I_n \to \overline{B_{||.||}(0, L)}$$

Additionaly,

$$L_1 := ||F_{SAS}||_\infty < L$$

The uniform density on compacta of the family of feedforward neural networks with one hidden layer guarantees that for any $\epsilon_2 > 0$, there exists $N \in \mathbb{N}$, $G \in \mathbb{M}_{N,N_1}$, $C \in \mathbb{M}_{N,n}$, $E \in \mathbb{M}_{N_1,N}$, and $\zeta \in \mathbb{R}^N$, such that the map defined by

$$F_{NN} : \overline{B_{||.||}(0, L)} \times I_n \to \mathbb{R}^{N_1} \tag{4.34}$$

$$(\mathbf{x}, \mathbf{z}) \mapsto E\sigma(G\mathbf{X} + C\mathbf{z} + \zeta) \tag{4.35}$$

satisfies,

$$||F_{NN} - F_{SAS}||_\infty = sup\{||F_{NN}(\mathbf{x}, \mathbf{z}) - F_{SAS}(\mathbf{x}, \mathbf{z})\} < \epsilon_2$$

This, with reverse triangle inequality proves, that

$$||F_{NN}||_\infty < ||F_{SAS}||_\infty + \epsilon_2$$

This guarantees that $||F_{NN}||_\infty < L$ if ϵ_2 is chosen such that $\epsilon_2 < L - L_1$. Thus,

$$F_{NN} : \overline{B_{||.||}(0, L)} \times I_n \to \overline{B_{||.||}(0, L)}$$

The continuity of the map F_{NN} implies that the corresponding reservoir equation has the existence of solutions property and that we can hence associate to it a (generalized) filter $U_{F_{NN}}$. This allows to conclude that the (unique) reservoir filter $U_{F_{SAS}}$ associated to the reservoir map F_{SAS} is such that

$$|||U_{FNN} - U_{FSAS}|||_\infty < \epsilon_2/(1-K)$$

Consider now the readout map $h_{W_1} : \mathbb{R}^{N_1} \to \mathbb{R}^d$ given by $h_{W_1}(\mathbf{x}) := W_1\mathbf{x}$ and let $U^{h_{W_1}}_{F_{NN}} : (I_n)^{\mathbb{Z}_-} \to (\mathbb{R})^{\mathbb{Z}_-}$ be the filter given by $U^{h_{W_1}}_{F_{NN}}(\mathbf{z})_t := W_1 U_{F_{NN}}(\mathbf{z})_t, t \in \mathbb{Z}_-$. Analogously, let $U^{h_{W_1}}_{F_{SAS}}(\mathbf{z})_t := W_1 U_{F_{NN}}(\mathbf{z})_t, t \in \mathbb{Z}_-$, noticing that $U^{h_{W_1}}_{F_{SAS}} = U^{p.q}_{W_1}$. Thus, for any $\epsilon_2 > 0$ we can find a filter of the type $U_{F_{NN}}$, that satisfies:

$$|||U^{p,q}_{F_1} - U^{h_{W_1}}_{F_{NN}}|||_\infty \leq ||W_1||_2 |||U_{F_{SAS}} - U_{F_{NN}}|||_\infty \leq ||W_1||_2 \epsilon_2/(1-K)$$

Consequently, for any $\epsilon > 0$, if we first set $\epsilon_1 = \epsilon/2$ and we then choose

$$\epsilon_2 := min\left\{\frac{\epsilon(1-K)}{2||W_1||_2}, \frac{L-L_1}{2}\right\}$$

Thus, it is evident that

$$|||U - U^{h_{W_1}}_{F_{NN}}|||_\infty \leq |||U - U^{p,q}_{W_1}|||_\infty + |||U^{p,q}_{W_1} - U^{h_{W_1}}_{F_{NN}}|||_\infty \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

To conclude the proof it suffices to show that the filter $U_{F_{NN}}$ can be realized as the reservoir filter associated to an echo state network of the type presented in the statement. It can be accomplished by defining a new reservoir map F_{ESN} with the architecture of an echo state network. Let $A := GE \in \mathbb{M}_N$ and define

$$F_{ESN} : D_N \times I_n \to \mathbb{R}^N \tag{4.36}$$

$$(\mathbf{x}, \mathbf{z}) \mapsto \sigma(A\mathbf{x} + C\mathbf{z} + \zeta) \tag{4.37}$$

The set D_N in the domain of F_ESN is given by

$$D_N := [-1,1]^N \cap E^{-1}(\overline{B_{||.||}(0,L)})$$

where, $E^{-1}(\overline{B_{||.||}(0,L)})$ denotes the preimage of the set $\overline{B_{||.||}(0,L)} \subset \mathbb{R}^{N_1}$ by the linear map $E : \mathbb{R}^N \to R^{N_1}$ associated to the matrix $E \in \mathbb{M}_{N_1,N}$. This set is compact as $E^1(\overline{B_{||.||}(0,L)})$ is closed and $[1,1]^N$ is compact and hence D_N is a closed subspace of a compact space which is always compact. Additionally, D_N is also convex because $[1,1]^N$ is convex and $E^1(\overline{B_{||.||}(0,L)})$ is also convex because it is the preimage of a convex set by a linear map, which is always convex. The image of F_{ESN} is contained in D_N. First, as the squashing function maps into the interval $[1,1]$, it is clear that

$$F_{ESN}(D_N, I_n) \subset [1,1]^N$$

$\mathbf{x} \in E^{-1}(\overline{B_{||.||}(0,L)})$ for any $\mathbf{x} \in D_N$, amd hence $E\mathbf{x} \in \overline{B_{||.||}(0,L)}$. Since F_{NN} maps into $\overline{B_{||.||}(0,L)}$, we can ensure that for any $\mathbf{z} \in I_n$, the image $F_{NN}(E\mathbf{x}\mathbf{z}) = E\sigma(GE\mathbf{x} + C\mathbf{z} + \zeta) = E\sigma(A\mathbf{x} + C\mathbf{z} + \zeta) \in \overline{B_{||.||}(0,L)}$, or equivalently

$$F_{ESN}(\mathbf{x},\mathbf{x}) = \sigma(A\mathbf{x} + C\mathbf{z} + \zeta) \in E^{-1}(\overline{B_{||.||}(0,L)})$$

These relations imply that

$$F_{ESN}(D_N, I_n) \subset D_N$$

and hence,

$$F_{ESN} : D_N \times I_n \to D_N$$

These, alongwith the continuity of the map F_{ESN} allow us to conclude that the corresponding reservoir equation has the existence of solutions property and that we can hence associate to it a (generalized) filter $U_{F_{ESN}}$. Let $W := W_1 E \in \mathbb{M}_{d,n}$ and define the readout map $h_{ESN} : D_N \to \mathbb{R}^d$ by $h_{ESN}(\mathbf{x}) := W\mathbf{x} = W_1 E\mathbf{x}$. Denote by U_{ESN} any generalized reservoir filter associated to the echo state network system (F_{ESN}, h_{ESN}) that, by construction, satisfies $U_{ESN}(\mathbf{z})_t := h_{ESN}(U_{F_{ESN}}(\mathbf{z})_t) = WU_{F_{ESN}}(\mathbf{z})_t$, for any $\mathbf{z} \in I_n$ and $t \in \mathbb{Z}_-$.
It can be shown that the map $f : D_N = [1,1]^N \cap E^1(\overline{B_{||.||}(0,L)}) \to (\overline{B_{||.||}(0,L)})$ given by $f(\mathbf{x}) := E\mathbf{x}$ is a morphism between the echo state network system (F_{ESN}, h_{ESN}) and the reservoir system (F_{NN}, h_{W_1}). The reservoir equivariance property holds because, for any $(\mathbf{x},\mathbf{z}) \in D_N \times I_n$:

$$f(F_{ESN}(\mathbf{x},\mathbf{z})) = E\sigma(A\mathbf{x} + C\mathbf{z} + \sigma) = E\sigma(GE\mathbf{x} + C\mathbf{z} + \sigma) = F_{NN}(E\mathbf{x},\mathbf{z}) = F_{NN}(f(\mathbf{x}),\mathbf{z})$$

Thus, it implies that all the generalized filters U_{ESN} associated to the echo state network are actually filters generated by the system (F_{NN}, h_{W_1}). This means that for each generalized filter U_{ESN} there exists a generalized filter of the type $U^{h_{W_1}}_{F_{NN}}$ such that $U_{ESN} = U^{h_{W_1}}_{F_{NN}}$. This concludes the proof of universal function approximation property of ESNs.
To prove the ESP in ESNs, it is sufficient [27] to show that the network satisfies the contractivity condition, i.e., for each x and y in the domain of the squashing function ϕ, it must be true that:

$$\|\phi(x) - \phi(y)\| \leq \|x - y\| \tag{4.38}$$

4.4 ESN Applications and Limitations

Echo State Networks and reservoir computing in general is well adapted for solving temporal classification, regression or prediction tasks. They give high performance, at fast

speeds due to less number of trainable parameters. It is however crucial that the *natural* time scale of the reservoir is tuned to be of the same order of magnitude as the important time scales of the application. Several successful applications of reservoir computing to both theoretical and real world engineering applications have been reported. Abstract applications include dynamic pattern classification, autonomous sine generation or the computation of highly nonlinear functions on the instantaneous rates of spike trains.
For practical applications, in robotics, ESNs have been used to control a simulated robot arm, to model an existing robot controller, to perform object tracking and motion prediction, event detection, motor control etc. ESNs have been used in the context of reinforcement learning. Also, applications in the field of Digital Signal Processing (DSP) have been quite successful, such as speech recognition or noise modeling. The use of reservoirs for chaotic time series generation and prediction have been reported. In many area's such as chaotic time series prediction and isolated digit recognition, reservoir computing techniques can outperform state-of-the-art approaches.
Although ESNs are not a relatively new research idea, there are still many open issues and future research directions. From a theoretical standpoint, a proper understanding of network dynamics and measures for these dynamics is very important. A theory on regularization of the readout and the reservoir dynamics is currently missing. The influence of hierarchical and structured topologies is not yet well understood. Bayesian inference with reservoir computing needs to be investigated. Intrinsic plasticity based reservoir adaptation needs also be further investigated. From an implementation standpoint, the concept can be extended way beyond RNN, and any high-dimensional dynamical system, operating in the proper dynamical regime can be used as a reservoir.
The theory of ESNs do not offer any immediate indications on how to construct reservoirs that have certain *desirable* properties given a particular task, nor is there consensus on what these desirable properties would be. Therefore, a lot of the current research on reservoirs focuses on reducing the randomness of the search for a good reservoir and offering heuristic or analytical measures of reservoir properties that predict the performance. The random connectivity of the ESN reservoir does not give a clear insight in what is going on in the reservoir.
The original appeal of ESNs was that their fast training due to small number of trainable parameters, they do not suffer from bifurcations, and are easy to implement. On a number of benchmark tasks, ESNs have performed better compared to other methods of nonlinear dynamical modeling. However, with the advent of Deep Learning, the problems faced by gradient descent based training of recurrent neural networks are mostly solved and the original unique selling point of ESNs, stable and simple training algorithms, is lost. Moreover, deep learning methods for RNNs have been successfully applied for highly

complex modeling tasks especially in natural language and speech processing. Reaching similar levels of complexity with ESNs would demand reservoirs of inordinate size.

4.5 Current Interest in ESNs

ESNs and other methods of reservoir computing can be a good alternative for cheap, fast and adaptive training when the modeled system is not too complex. There are many applications in signal processing, for example in biosignal processing, remote sensing or robot motor control for which this is particularly true.
During the last decade, ESNs have shown promise as a computational principle that blends well with non-digital computational substrates. Dedicated hardware realizations of ESNs and reservoir computers could enable speed gains and power savings. Some of the novel research fields using ESNs and reservoir computers are listed below.

- **Optical microchips:** Use of passive photonic silicon reservoirs as a generic computational platform for diverse digital and analogue tasks is described in [29].
- **Mechanical nano-oscillators:** In [4] a device based on a network of masses coupled by linear springs and attached to a substrate by non-linear springs, is described. The device operates on reservoir computing principle, to efficiently solve benchmark problems of computing the parity of a bit stream and classifying spoken words.
- **Memristor-based neuromorphic microchips:** Hierarchical composition of a set of interconnected memristive networks into a larger reservoir is shown to perform significantly better than monolithic memristive networks on tasks such as waveform generation tasks [21].
- **Carbon-nanotube/polymer mixtures:** [32] demonstrates application of the reservoir model as a simple computational layer on physical substrates consisting of single-walled carbon nanotubes and polymer mixtures to extract exploitable information from them.
- **Artificial soft limbs:** Soft machines dynamics exhibit a variety of properties such as nonlinearity, memory and potentially infinitely many degrees of freedom. [28] demonstrate use of these dynamics for real-time computation to emulate nonlinear dynamical systems.

Such novel computational materials and the microdevices made from them often lack numerical precision and ways to emulate classical logical switching circuits are unknown. But, nonlinear dynamics can be elicited from suitably interconnected ensembles of such elements offering new avenues for building and training physical reservoirs using ESN methods.

Chapter 5

Review of the paper by Verzelli et al. [41]

This section presents a detailed review of research paper [41] concerning techniques to address the problem of stability and chaotic behaviour of ESNs. The ESNs have a narrow operating region in their hyper-parameter space in which their performance is maximum. The operation of ESNs marginally outside this region can be unstable and chaotic. Hence, it is critical for successful application of ESNs, to find this stable operating region in a reliable and efficient way. The performance gain obtained by hyper-parameter tuning also needs to be considered from the point of view of *memory-nonlinearity trade-off*. Networks with linear activations can propagate the effect of past inputs over a wide time range. Increasing the nonlinearity of activation leads to reduction in the ability to remember past inputs. This phenomenon is known as memory nolinearity tradeoff in ESNs.
The primary hyper-parameters of ESNs which characterize their operating regime are [26]:

- *The number of units in the reservoir* [N], typically 100 - 1000 times the input dimensionality.
- *The input scaling* [α_{in}], which is used to scale the randomly-generated input matrix W_{in}. The elements of W_{in} obey a uniform distribution between -α_{in} and α_{in}.
- *The spectral radius* [ρ] of the weight matrix of the reservoir units, W.
- *The degree of sparseness*, given by the fraction of non-zero elements in W.

The narrow region of hyper-parameter space, which maximises ESNs' performance is denoted as edge of criticality *(EoC)*. A hyper-parmater configuration outside such a region might lead the ESNs into fully developed chaos and produce unreliable computations. Characterizing such a region requires some techniques such as cross-validation, ad-hoc criteria, and consideration of memory-nonlinearity trade-off. In the paper, the authors propose a nonlinear activation function which ensures that the network never enters chaotic region. The ESN using this activation is henceforth referred as a *spher-*

ical reservoir. It is proved in the paper that spherical reservoirs are universal function approximators. The authors demonstrate that the network can never become chaotic as long as it satisfies certain conditions. The paper proposes a memory metric to characterize the ability of the network to retain memory of past inputs. The authors also present the results of its performance on various nonlinear system performance benchmarks. We discuss each of these, in following paragraphs.

5.1 Self-normalizing Activation Function

In the paper [41], the authors propose a self-normalizing activation function for reservoir units, which is given by:

$$a_k = W x_{k-1} + W_{in} s_k \tag{5.1}$$

$$x_k = r \frac{a_k}{\|a_k\|} \tag{5.2}$$

The above equation specifies that the pre-activation a_k should normalized and then multiplied by a scalar r. Thus, each successive state lies on the surface of a hypersphere with radius r, in the state space. In other words, the pre-activations are projected onto the surface of the hypersphere by the activation function, as shown in in Fig. 5.1.

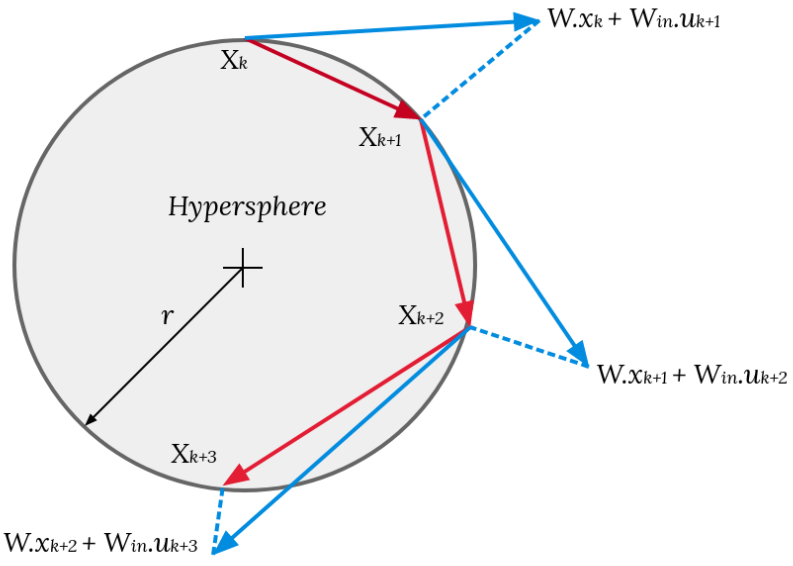


Figure 5.1: Behavior of the proposed model in a 2-dimensional case

The salient characteristics of this activation function can be stated as:

1. The normalization Eq. 5.2 projects the network pre-activation a_k onto an $(N-1)$ dimensional hyper-sphere of radius r.

2. The normalization operation in Eq. 5.2 is not element-wise. It depends on activations of all units of the reservoir.

3. This activation function guarantees that the spectral radius of the reservoir matrix can vary in a wide range, without compromising network stability.

4. The maximum Lyapunov exponent is always zero, implying that the network always operates on the EoC[1].

Proposition 1. *[41] The ESNs with proposed self-normalizing activation function are shown to satisfy the contractivity condition, hence possess the universal function approximation property, if provided*

$$\sigma_{min}(W) \geq 1 + \frac{\|W_{in}\|\|s_{max}\|}{r} \tag{5.3}$$

where $\sigma_{min}(W)$ is the smallest singular value of matrix W and $\|s_{max}\|$ denotes the largest norm associated with an input.

Proof. Let $\widehat{x} := x/\|x\|$ and $\widehat{y} := y/\|y\|$. Taking the square of the norm, by applying properties of norms and the given inequality in the proposition, we have

$$\|x_{k+1}\| = \|W \cdot x_k + W_{in} \cdot s_k\| \geq \|W \cdot x_k\| - \|W_{in} \cdot s_k\| \geq \sigma_{min}(W)r - \|W_{in}\|\|s_{max}\| \geq r,$$

which means that 5.1 map states outside the hyper-sphere of radius r.

And then one gets

$$\begin{aligned}
\|\phi(x) - \phi(y)\|^2 &= \|r\widehat{x} - r\widehat{y}\|^2 \\
&= r^2(\|\widehat{x}\|^2 + \|\widehat{x}\|^2 - 2\widehat{x} \cdot \widehat{y}) \\
&= r^2(2 - 2\widehat{x} \cdot \widehat{y}) \\
&\leq r^2 \left(\frac{\|x\|}{\|y\|} + \frac{\|y\|}{\|x\|} - 2\widehat{x} \cdot \widehat{y} \right) \\
&\leq \|x\|\|y\| \left(\frac{\|x\|}{\|y\|} + \frac{\|y\|}{\|x\|} - 2\widehat{x} \cdot \widehat{y} \right) \\
&= \|x - y\|^2.
\end{aligned}$$

□

[1]EoC or *Edge of Criticality* is a region in hyperparameter space separating ordered and chaotic regimes. The computational capability of ESNs - prediction accuracy and memory capacity, is maximized at EoC.

Which ensures that network satisfies the contrativity condition. Since this is a sufficient, but not necessary condition, it is possible to have stable operation of ESNs without satisfying this condition. In general, as long as the norm of W is large enough compared to the signal variance, the ESN is very unlikely to enter chaotic region. Since the value of r does not have an impact on network stability, as long as condition 5.3 is satisfied, it is set to 1 in further analysis.

5.2 Network state dynamics

The network state dynamics are discussed for an autonomous network (e.g. a pure sine wave generator) and for input driven network.

5.2.1 The autonomous case

For the case where the ESN is operating in absence of any input, i.e. $s_k = 0$, the system state dynamics are given by

$$\phi(x) := \frac{Wx}{\|Wx\|} \tag{5.4}$$

At time-step n, the system state is given by

$$\begin{aligned} x_n &= \phi(x_{n-1}) \\ &= \phi(\phi(x_{n-2})) \\ &= \phi(\phi(\phi(\dots(x_0)))) \\ &= \phi^n(x_0) \\ &= \frac{W^n x_0}{\|W^n x_0\|} \end{aligned} \tag{5.5}$$

This implies that, for the autonomous case, evolution of the system through n time-steps is equivalent to updating the state by performing n matrix multiplications and projecting the outcome only at the end. Also, the state at time n remains same even if the reservoir weights matrix W is scaled to aW, indicating that the spectral radius of the matrix does not affect the network dynamics in the autonomous case.

5.2.2 Edge of criticality

While selecting the hyper-parameters of ESNs, it is desirable to have the system operating close to the EoC, since it is in this region that their performance is optimal [31] with the network performance having rich dynamics without entering the chaotic region. In

the paper, the authors demonstrate **analytically** that the proposed recurrent model cannot enter a chaotic regime for the case of network with infinite reservoir dimension and **empirically** for the case of input driven reservoirs with finite number of units. To prove these claims, it is shown that for a reservoir with large number of units N, the maximum local Lyapunov exponent[2] is always non-positive.
The Jacobian matrix of 5.2 is given by:

$$J_{ij} = \frac{\partial}{\partial x_i}\phi_j(x) \tag{5.6}$$

$$= \sum_{l}^{N} \frac{\partial \phi_j}{\partial a_l}\frac{\partial a_l}{\partial x_i} \tag{5.7}$$

$$= \frac{W_{ij}}{\|a\|}\left(1 - \frac{a_i a_j}{\|a\|^2}\right) \tag{5.8}$$

It can be observed that, for asymptotically large networks ($n \to \infty$), we have $a_i/\|a\| \to 0$ for each i. This reduces the Jacobian matrix to $J(x) = W/\|Wx\|$. As we are considering the case with $r = 1$, each successive state lies on the surface of a hypersphere of radius 1. Thus, we know that norm of each state vector $\|x\| = 1$. Since spectral radius ρ of a square matrix W is the largest absolute value of its eigenvalues, we can write, from definition of eigenvalues and eigenvectors:

$$Wx_k = \rho x_k \tag{5.9}$$

$$\|Wx_k\| = \|\rho x_k\| \tag{5.10}$$

$$= \rho\|x_k\| \tag{5.11}$$

$$= \rho \tag{5.12}$$

This allows us to approximate the norm of Wx with its SR $\rho = \rho(W)$,

$$J \approx \frac{W}{\|Wx\|} \approx \frac{W}{\rho} \tag{5.13}$$

The global behavior of 5.2 is characterised by the maximum LLE [31], defined as:

$$\Lambda := \lim_{n\to\infty} \frac{1}{n} \log\left(\prod_{k}^{n} \rho_k\right) \tag{5.14}$$

[2]The Lyapunov exponent or Lyapunov characteristic exponent of a dynamical system is a quantity that characterizes the rate of separation of trajectories which are infinitesimally close initially. If the system is conservative (i.e., there is no dissipation), a volume element of the phase space will stay the same along a trajectory. Thus the sum of all Lyapunov exponents must be zero. If the system is dissipative, the sum of Lyapunov exponents is negative, which is ensured when the maximum LLE is non-positive.

where ρ_k is the spectral radius of the Jacobian at time-step k. This implies $\Lambda = 0$ as $N \to \infty$. Thus, since the maximum LLE is zero the network cannot become chaotic.
To show that $\Lambda = 0$ still holds for spherical reservoirs with a finite number of units N, the maximum LLE for the Jacobian Eq. 5.8 is numerically computed and reported in Figs. 5.2-5.3 Thus, it can be observed that the largest Lyapunov exponent is always

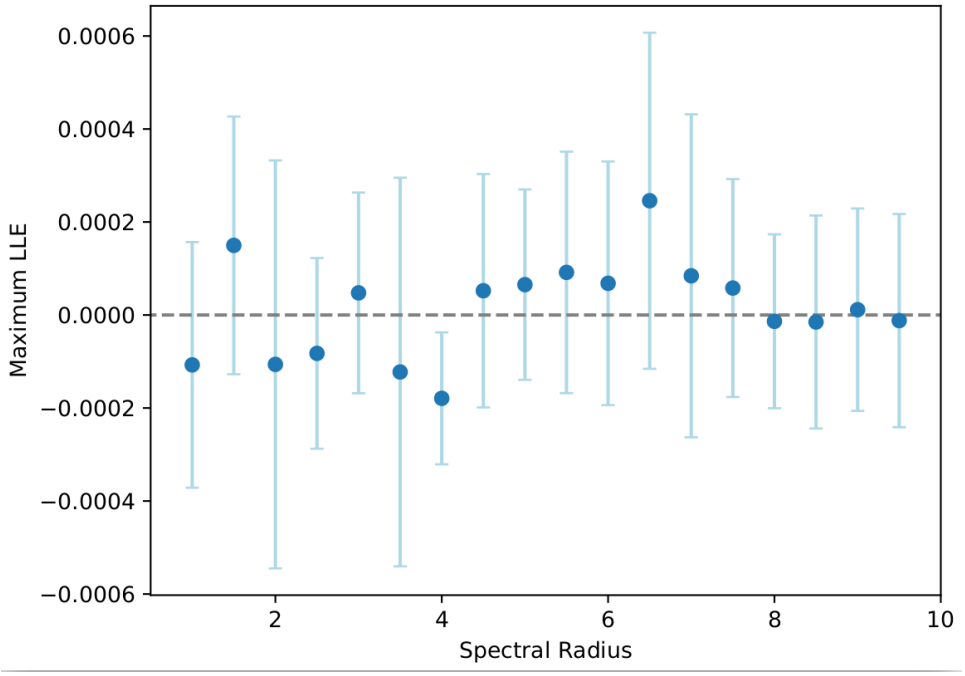


Figure 5.2: LLEs for different values of SR for $N = 500$ network.

close to zero, irrespective of the spectral radius.

5.2.3 Input driven case

For input driven case, effective input is defined as, $u_k = W_{in} s_k$ and Eqn. 5.1 becomes:

$$a_k = W x_{k-1} + u_k \tag{5.15}$$

The state at timestep n is shown to be:

$$x_n = M^{(n,0)} x_0 + \sum_{k=1}^{n} M^{(n,k)} u_k \tag{5.16}$$

where,

$$M^{(n,k)} = \frac{W^{(n-k)}}{\Pi_{l=k}^{n} N_l} \tag{5.17}$$

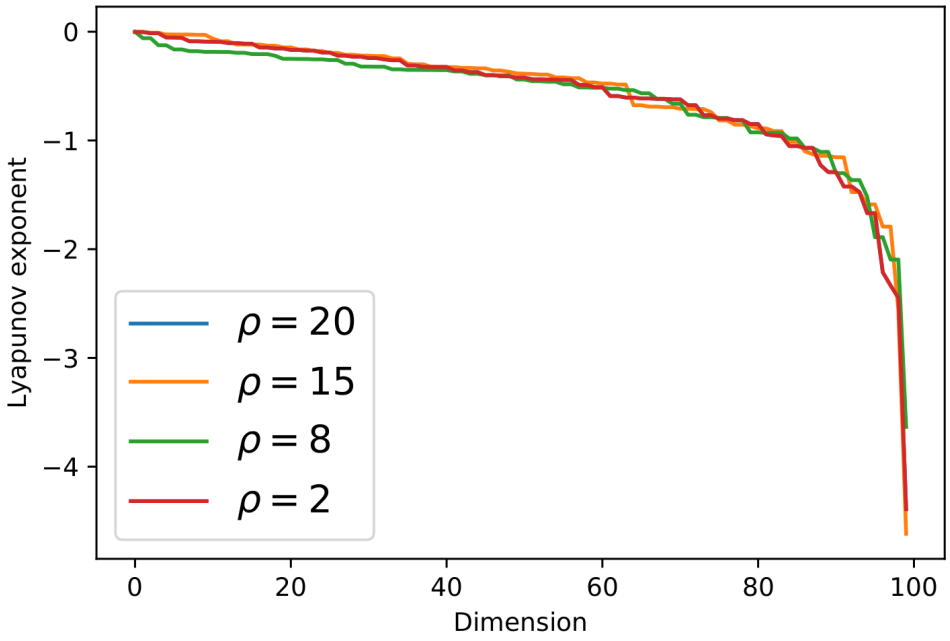


Figure 5.3: Lyapunov spectrum for $N = 100$.

and,

$$N_k = \|Wx_{k-1} + u_k\| \tag{5.18}$$

Equation 5.16 indicates that the network's final state can be interpreted as a combination of all previous inputs. The next sections discuss the memory of the network and memory loss.

5.2.4 Memory of past inputs in spherical reservoirs

To analyse the ability of the proposed spherical reservoir model to retain the memory of past inputs, the authors define a metric which quantifies the impact of past inputs on current state x_n. This metric, called as *memory of the network* $\mathbf{M}_\alpha(k|n)$ is defined as the influence of input at time k on the network state at time n.

$$\mathbf{M}_\alpha(k|n) = \left(\frac{\alpha}{\alpha+1}\right)^{n-k} \tag{5.19}$$

In this equation, α is defined as $\alpha = \rho/\delta$, where δ represents the amount by which the network state is modified by the driving input, compared to the case of autonomous network. Eqn. 5.19 goes to 0 indicating there is no impact of the input on the states for the case where $\alpha \to 0$, and it tends to 1 for $\alpha \to \infty$. This implies that for an infinitely large ρ, the network can perfectly remember all values of past inputs, regardless of their occurrence in the past. Eqn. 5.19 does not depend on values k and n, but only on

their difference (elapsed time). The value of $\mathbf{M}_\alpha(k|n)$, decreases with increasing value of elapsed time, meaning that inputs far away in past have less impact on the current state.

5.2.5 Memory loss

The memory loss between states x_m and x_n with respect to signal at present time step k is found to be:

$$\Delta\mathbf{M}_\alpha(k|m,n) = \left(1-\frac{1}{\alpha+1}\right)^n \left(1-\frac{1}{\alpha}\right)^k \left(\left(1-\frac{1}{\alpha+1}\right)^r - 1\right) \leq 0 \tag{5.20}$$

which can be interpreted as having $\Delta\mathbf{M}_\alpha(k|m,n) \to 0$, for large values of spectral radius, indicating that the model eliminates memory losses of past inputs.
The comparative difference in impact of past inputs vs recent inputs on the present state is given by the expression:

$$\Delta\mathbf{M}_\alpha(k,l|n) = \left(1-\frac{1}{\alpha+1}\right)^{n-k} \left(1-\left(1-\frac{1}{\alpha+1}\right)^{k-l}\right) \geq 0 \tag{5.21}$$

This expression implies that the network is able to preserve the information about the network state from all inputs observed in past.

5.3 Performance on memory tasks

The proposed memory metric is compared with corresponding metrics derived for a linear network and a network using hyperbolic tangent (*tanh*) activation, by numerical computing $\mathbf{M}_\alpha(k)$ for 100 timesteps, over all three models. The results, shown in 5.4a indicate that the linear model performs best in terms of memory, while the *tanh* model is not able to perform well beyond 20 timesteps. This is consistent with expectations based on the memory-nonlinearity tradeoff. The spherical reservoir performs close to the linear network in terms of memory, and it also contains rich dynamics of nonlinear networks, thus combining best features of both configurations. The paper describes experiments conducted to evaluate the ability of the spherical reservoirs to reproduce inputs seen in the past using benchmarking datasets for nonlinear dynamic systems on all 3 models. A brief description of each task/dataset is as follows:

- White noise: For this task, uniformly distributed white noise in the interval $[-1, 1]$ was fed to the network.
- Multiple Superimposed Oscillators: This task consists of training the networks on a combination of 3 sinusoids:

$$u(k) = sin(0.2k) + sin(0.311k) + sin(0.42k) \tag{5.22}$$

This task is challenging due to simultaneous presence of multiple frequency scales in the inputs signal.

- Lorenz system: The Lorenz system is given by equations:

$$\dot{x} = \sigma(y - x) \tag{5.23}$$

$$\dot{y} = x(\rho - z) - y \tag{5.24}$$

$$\dot{z} = xy - \beta z \tag{5.25}$$

This system is chaotic for $\sigma = 10$, $\beta = 8$ and $\rho = 28$. The system is solved for these parameters and $\dot{x}$ values are used to train the networks.

- Mackey-Glass system: The Mackley-Glass system is given by:

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t - \lambda)}{1 + x(t - \lambda)} \tag{5.26}$$

- Santa Fe Laser dataset: The Santa Fe laser dataset is a chaotic time series obtained from laser experiments.

For all experiments, the desired output y_k is the input $u_{k-\tau}$ from 0 to 100 timesteps in past (τ is from $0 - 100$). 1000 reservoir units were used for all models. For spherical reservoir, input was scaled by a factor of 0.01 and spectral radius was set to $\rho = 15$. For other two models, inputs were not scaled and spectral radius $\rho = 0.95$ was used.
The performance of the networks was evaluated using an accuracy metric based of *Normalised Root Mean Square Error (NRMSE)*, defined as:

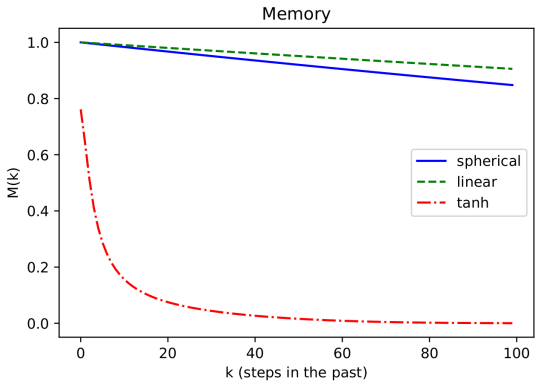
$$\gamma = max\left[1 - NRMSE, 0\right] \tag{5.27}$$

$$NRMSE = \sqrt{\frac{\langle \|y_k - \hat{y}_k\|^2 \rangle}{\langle \|y_k - \langle \hat{y}_k \rangle\|^2 \rangle}} \tag{5.28}$$

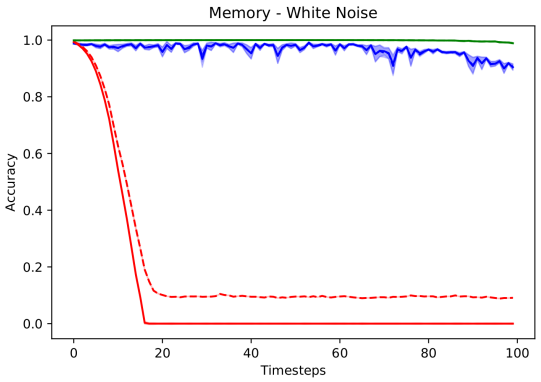
The results on five different memory tasks were reported for train data (*solid line*) and test data (*dashed line*) for all three models, as shown in Fig. 5.4

- Spherical reservoir (*blue curve*)
- Linear activation (*green curve*)
- Hyperbolic tangent activation (*red curve*)

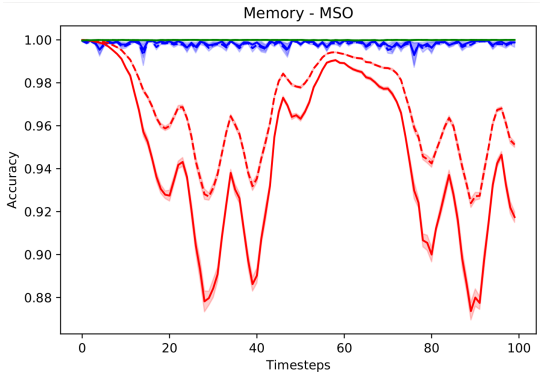
The plot shows that for white noise task, multiple superimposed oscillator task, Lorenz series task and Santa Fe laser dataset, networks using the spherical reservoir have a performance comparable with linear networks, while *tanh* networks fails to correctly reconstruct the signal over the entire domain k. Only for the Mackey-Glass system the spherical reservoir network performance drops to the level of *tanh* nonlinear model.



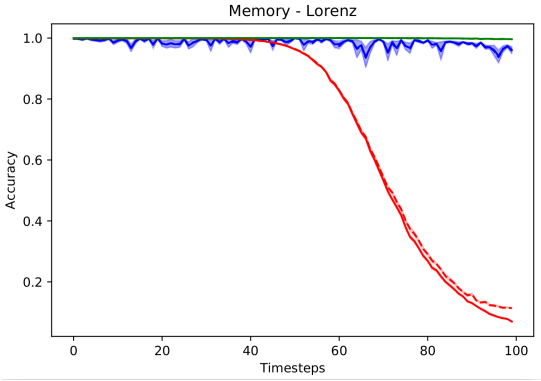
(a) Memory Decay



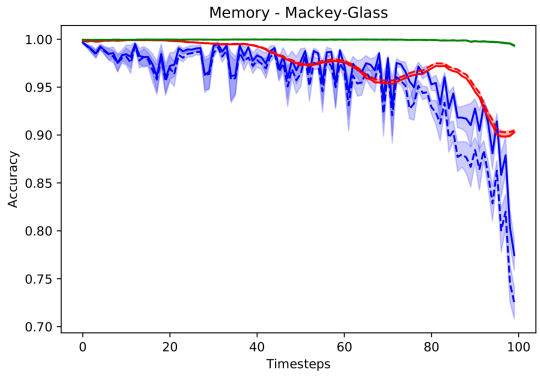
(b) White noise



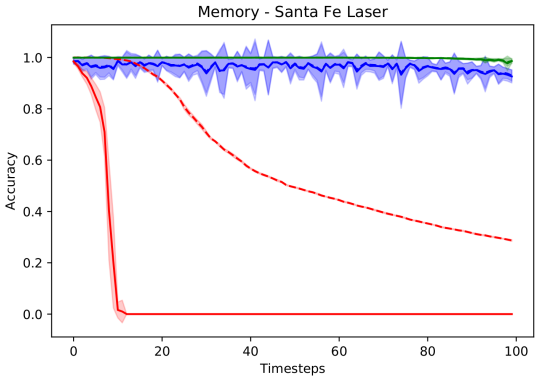
(c) Multiple Superimposed Osc.



(d) Lorenz series



(e) Mackey-Glass System



(f) Santa Fe Laser dataset

Figure 5.4: Memory decay and experiment results.

Bibliography

[1] S. Basterrech, C. Fyfe, and G. Rubino. Self-organizing maps and scale-invariant maps in echo state networks. *11th International Conference on Intelligent Systems Design and Applications (ISDA)*, 2011.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks 5*, 5(2), 1994.

[3] K. Bush and C. Anderson. Modeling reward functions for incomplete state representations via echo state networks. *In Proceedings of the International Joint Conference on Neural Networks, Montreal, Quebec*, 2005.

[4] Coulombe J C, York M C A, and Sylvestre J. Computing with networks of nonlinear mechanical oscillators. *PLOS ONE*, 12(6), 2017.

[5] Gallicchio C. and Micheli A. Echo state property of deep reservoir computing networks. *Cognitive Computation*, 9(3):337–350, 2017.

[6] Gallicchio C., Micheli A., and L. Silvestri. Local lyapunov exponents of deep rnn. *Proceedings of the 25th European Symposium on Artificial Neural Networks (ESANN)*, pages 559–564, 2017.

[7] Andrea Ceni, Peter Ashwin, Lorenzo Livi, and Claire Postlethwaite. The echo index and multistability in input-driven recurrent neural networks. *https://arxiv.org/abs/2001.07694*, 2020.

[8] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[9] Z. Deng and Y. Zhang. Collective behavior of a small-world recurrent neural system with scale-free distribution. *IEEE Transactions on Neural Networks*, 18(5):1364–1375, 2007.

[10] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, and M. Nuttin. Pruning and regularization in reservoir computing. *Neurocomputing*, 72:1534–1546, 2009.

[11] C. Gallicchio and A. Micheli. Deep reservoir computing: A critical analysis. *Proceedings of the 24th European Symposium on Artificial Neural Networks (ESANN)*, pages 497–502, 2016.

[12] C. Gallicchio and A. Micheli. Deep tree echo state networks. *Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN)*, pages 499–506, 2018.

[13] C. Gallicchio and A. Micheli. Deep reservoir neural networks for trees. *Information Sciences*, 480:174–193, 2019.

[14] C. Gallicchio, A. Micheli, and L. Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–89, 2017.

[15] C. Gallicchio, A. Micheli, and L. Pedrelli. Design of deep echo state networks. *Neural Networks*, 108:33–47, 2018.

[16] Claudio Gallicchio and Alessio Micheli. Deep echo state network (deepesn): A brief survey. *https://arxiv.org/pdf/1712.04323.pdf*, 2019.

[17] Lyudmila Grigoryeva and Juan-Pablo Ortega. Echo state networks are universal. *Neural Networks*, 108:495–508, 2018.

[18] G. E. Hinton, D. E. Rumelhart, and R. J. Williams. Learning internal representations by backpropagating errors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1, 1985.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997.

[20] G. Holzmann and H. Hauser. Echo state networks with filter neurons and a delay and sum readout. *Neural Networks*, 32(2):244–256, 2009.

[21] Burger J, Goudarzi A, Stefanovic D, and Teuscher C. Hierarchical composition of memristive networks for real-time computing. *IEEE/ACM International Symposium*, pages 33–38, 2015.

[22] Steil J J. Backpropagation-decorrelation: online recurrent learning with o(n) complexity. *Technical report, Neuroinformatics Group*, 2004.

[23] H. Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. *Technical report, Jacobs University*, 2007.

[24] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimisation and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20:335–352, 2007.

[25] Herbert Jaeger. The "state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[26] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

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

[28] Nakajima K, Hauser H, and Li T. Information processing via physical soft body. *Scientific Reports*, 5, 2015.

[29] Vandoorne K, Mechet P, and Van Vaerenbergh T. Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature Communications*, 5(3541), 2014.

[30] Deng L and Yu D. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(1):199–199, 2014.

[31] Lorenzo Livi, Filippo Maria Bianchi, and Cesare Alippi. Determination of the edge of criticality in echo state networks through fisher information maximization. *IEEE transactions on neural networks and learning systems*, 29(3):706–717, 2017.

[32] Dale M, Miller J F, Stepney S, and Trefzer M A. Evolving carbon nanotube reservoir computers. *International Conference on Unconventional Computation and Natural Computation*, pages 49–67, 2016.

[33] Z. K. Malik, A. Hussain, and Q. J. Wu. Multilayered echo state machine: a novel architecture and algorithm. *IEEE Transactions on cybernetics*, 47(4):946–959, 2017.

[34] G. Manjunath and H. Jaeger. Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks. *Neural Computation*, 2013.

[35] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural Computation*, 19:757–779, 2007.

[36] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7):1159–1171, 2008.

[37] M. D. Skowronski and J. G. Harris. Minimum mean squared error time series classification using an echo state network prediction model. *Science*, 304:78–80, 2004.

[38] J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20:353–364, 2007.

[39] M. H. Tong, A. D. Bicket, E. M. Christiansen, and G. W. Cottrell. Learning grammatical structure with echo state network. *Neural Networks*, 20:424–432, 2007.

[40] Schiller U.D. and Steil J. J. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63(C):5–23, 2005.

[41] Pietro Verzelli, Cesare Alippi, and Lorenzo Livi. Echo state networks with self-normalizing activations on the hyper-sphere. *Scientific Reports*, 9(1):1–14, 2019.

[42] T. Wang and C. Fyfe. Training echo state networks with neuroscale. *In 2011 International Conference on Technologies and Applications of Artificial Intelligence*, 2011.

[43] Y. Xue, L. Yang, and S. Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20:365–376, 2007.

[44] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. Re-visiting the echo state property. *Neural Networks*, 2012.

Appendices

Appendix A

Fisher memory in dynamic systems

A dynamic system such as a recurrent network could store information about recent input sequences in its transient dynamics, even if the network does not have information-bearing attractor states. To explain it with an analogy, consider the surface of a liquid, perturbed by impact of an object on the surface. Even though this surface has no attractors, transient ripples on the surface encode information about past objects that were thrown in it. For networks and other dynamical systems, Fisher information theory provides a framework to construct a measure of memory traces. The measure should estimate the efficiency with which the network state encodes the history of input signals. A given past signal history $\{s(n-k)|k \geq 0\}$ induces a conditional probability distribution $\{P(x(n)|s)\}$ on the network's current state. This history can be considered as a temporal vector s whose k-th component is s_{n-k}. The Fisher memory matrix (FMM) between the present state $x(n)$ and the past signal is then defined as

$$\mathbf{J_{k,l}}(s) = \left\langle -\frac{\partial^2}{\partial s_k \partial s_l}\{logP(x(n)|s)\} \right\rangle_{P(x(n)|s)} \tag{A.1}$$

A.1 Motivation

The statistical difference between the two probability distributions, $p(x)$ and $q(x)$ is measured by Kullback-Leibler (KL) divergence, which is defined as

$$D_{KL}(p\|q) = \int dx.p(x).log\frac{p(x)}{q(x)} \tag{A.2}$$

For a multidimensional family of distributions $p(x|s)$, parameterized by the vector s, the KL divergence between two infinitesimally close distributions s and $s+\delta s$ is approximated by the Fisher information matrix Eq. A.1, so that $D_{KL}[p(x|s)\|p(x|s+\delta s)] \approx \delta s^T \mathbf{J}(\mathbf{s})\delta s$. In the special case when $p(x|s)$ is a family of Gaussian distributions whose mean $\mu(s)$

depends explicitly on s, and whose covariance C is independent of s, the KL-divergence and Fisher information matrix expressions take the form

$$D_{KL}p(x|s^1)\|p(x|s^2) = \frac{1}{2}[\mu(s^1) - \mu(s^2)]^T C^{-1}[\mu(s^1) - \mu(s^2)] \tag{A.3}$$

$$\mathbf{J_{k,l}}(s) = \frac{\partial \mu(s)^T}{\partial s_k} C^{-1} \frac{\partial \mu(s)}{\partial s_l} \tag{A.4}$$

In the situation where the mean μ of the family only depends linearly on the parameter s, the Fisher information Eq. A.4 becomes independent of s, and the quadratic approximation to the KL-divergence through the Fisher information becomes exact. The network state at time n is given by $x(n) = \sum_{k=0}^{\infty} W_k v s_k + \sum_{k=0}^{\infty} W_k z(n-k)$. Since the noise is Gaussian, the conditional distribution $P(x(n)|s)$ is also Gaussian, with mean $\mu(s) = \sum_{k=0}^{\infty} W_k v s_k$ and noise covariance matrix $C_n = \epsilon \sum_{k=0}^{\infty} W^k W^{kT}$. The mean is linearly dependent on the signal, and the noise covariance is independent of the signal, so $\mathbf{J}$ is independent of the signal history and takes the form

$$\mathbf{J_{k,l}} = v^T W^{kT} C_n^{-1} W^l v \tag{A.5}$$

This matrix is called as the Fisher Memory Matrix (FMM). To understand the significance of individual matrix elements, consider a scenario in which a single input pulse s_k enters the network at time $n-k$. Then the network state at time n has the solution $x(n) = W^k v s_k + \sum_{m=0}^{\infty} W^m z(n-m)$. The Fisher information that $x(n)$ retains about this pulse is a single number given by the diagonal element $\mathbf{J_{k,k}}$ of the FMM. Thus the digonal of the FMM is a memory curve, called the Fisher Memory Curve (FMC). It captures the decay of the memory trace of a single pulse. Next, consider a scenario in which two pulses s_k and s_l enter the network at times $n-k$ and $n-l$. The Fisher information between $x(n)$ and both pulses is a 2×2 symmetric matrix whose diagonals are $\mathbf{J_{k,k}}$ and $\mathbf{J_{l,l}}$ and whose off diagonal elements are $\mathbf{J_{k,l}}$. Thus the off diagonal elements of the FMM capture the interference between two pulses entering the system at k and l timesteps in the past. Such interference can arise physically through the collision of signals entering the network at the two different times. In general, the Fisher information that an output variable x contains about an input s can be thought of as a signal to noise ratio. This interpretation is especially precise in the situation where x is linearly proportional to sum of s and some additive Gaussian noise. For example, in the one dimensional case $x = g.s + z$, where g is a scalar input-output gain and z is a Gaussian random variable with variance ϵ. Then the Fisher information that x contains about s is simply g^2/ϵ, the ratio of the squared signal gain to the noise variance. Similarly, at any given time, the Fisher information that the input vector $vs + z$ to the network has about the signal s is given by $v.v/\epsilon$, which is again a squared signal gain to noise ratio. Finally, k timesteps

after a pulse has entered the network, it is embedded in the network state in the direction $W^k v$ (the signal gain). $\mathbf{J_{k,k}}$ is essentially this squared signal gain divided by the noise covariance C_n . When measured in units of the input SNR $1/\epsilon$, $\mathbf{J_{k,k}}$ represents the fraction of the input SNR remaining in the network state x about an input pulse entering the network k timesteps in the past.