# Reservoir Computing: A New Paradigm for Neural Networks

by

Felix Grezes

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2019

Reservoir Computing: A New Paradigm for Neural Networks

by

Felix Grezes

Advisor: Pr. Andrew Rosenberg

**Introduction:** Even before Artificial Intelligence was its own field of computational science, humanity has tried to mimic the activity of the human brain. In the early 1940s the first artificial neuron models were created as purely mathematical concepts. Over the years, ideas from neuroscience and computer science were used to develop the modern Neural Network. The interest in these models rose quickly but fell when they failed to be successfully applied to practical applications, and rose again in the late 2000s with the drastic increase in computing power, notably in the field of natural language processing, for example with the state-of-the-art speech recognizer making heavy use of deep neural networks.

Recurrent Neural Networks (RNNs), a class of neural networks with cycles in the network, exacerbates the difficulties of traditional neural nets. Slow convergence limiting the use to small networks, and difficulty to train through gradient-descent methods because of the recurrent dynamics have hindered research on RNNs, yet their biological plausibility and their capability to model dynamical systems over simple functions makes then interesting for computational researchers.

Reservoir Computing emerges as a solution to these problems that RNNs traditionally face. Promising to be both theoretically sound and computationally fast, Reservoir Computing has already been applied successfully to numerous fields: natural language processing, computational biology and neuroscience, robotics, even physics. This survey will explore the history and appeal of both traditional feed-forward and recurrent neural networks, before describing the theory and models of this new reservoir computing paradigm. Finally recent papers using reservoir computing in a variety of scientific fields will be reviewed.

# Contents

# Chapter 1

# History of Neural Networks

## 1.1 Appeal and Historical Recap

Explaining and reproducing human thought have always interested scientists and philosophers, but only with the discovery of the neuron in the 1890s by Golgi and Ramón y Cajal, and with the advent of the computer after 1950 has it finally become a feasible goal. Today artificial intelligence is a major area of research (again to both scientists and philosophers), and artificial neural networks an important paradigm of AI.

An early and promising model was the perceptron, proposed in 1957 by Rosenblatt [1] of the Cornell Aeronautical Laboratory. Modeling a single neuron and using simple algorithms, the perceptron is able to learn a number of functions of its inputs. The output of the perceptron is computed as follows: $f(x) = 1$ if $w \cdot x + b > 0$, and 0 otherwise, where $w \cdot x$ is the dot-product of the input $x$ with the weight vector $w$ (i.e. the weighted sum of the inputs), and $b$ is a bias term which acts a threshold.

However in 1969 Minsky and Papert [2] showed that not all functions can be learned by the perceptron, most famously the XOR (eXclusive OR) logical operation, since the perceptron is only a linear classifier. Still, research in the field stagnated until Werbos proposed the

backpropagation algorithm in 1975, which solved the XOR problem by training over multiple layers of neurons. By the mid 1980s the study of artificial neural networks became a fully established field, with dedicated journals and conferences.

## 1.2 Feed-Forward Networks: Definitions and Theory

The fundamental building block of a neural network is the neuron. In essence the neuron is simply a model for a multivariate function whose input variables are weighted by a weight vector. Mathematically: $f(x) = \varphi(w \cdot x + b)$ , with $\varphi$ the chosen activation function, and $w, x, b$ the same as in the perceptron, i.e. respectively the weight vector, the input vector and the bias term. Figure 1.1 gives a visualization of the artificial neuron model.

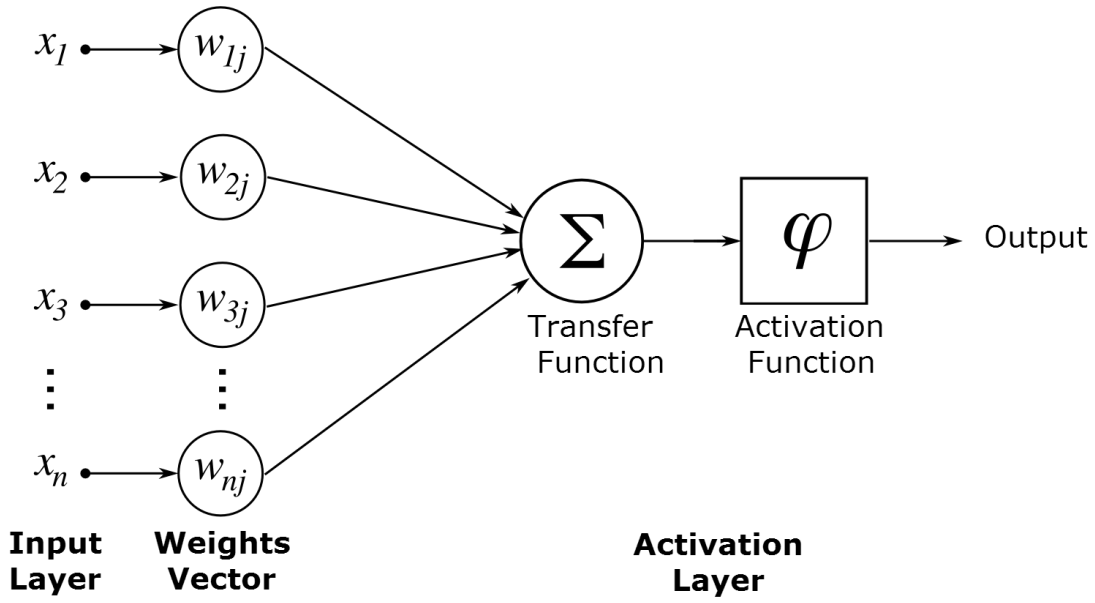


Figure 1.1: Model of an Artificial Neuron

The activation function is usually chosen to be non-linear, as to allow the neuron, and

the networks built from neurons, the power to approximate any mathematical function (see Cybenko-Hornik results below). Some popular choices are, threshold similarly to the perceptron, tanh and sigmoid which are continuously differentiable. Research is also done on more esoteric activations, such as the cosine or radial functions [3].

However, the perceptron alone can only learn linear functions. Combining mathematical into networks, similar to how biological brains are a network of neurons, can help overcome this linear limitation, as will be shown in following paragraph. The first type of network architecture we consider is the feedforward network, in which the network does not contain cycles. Figure 1.2 shows a graphical representation.

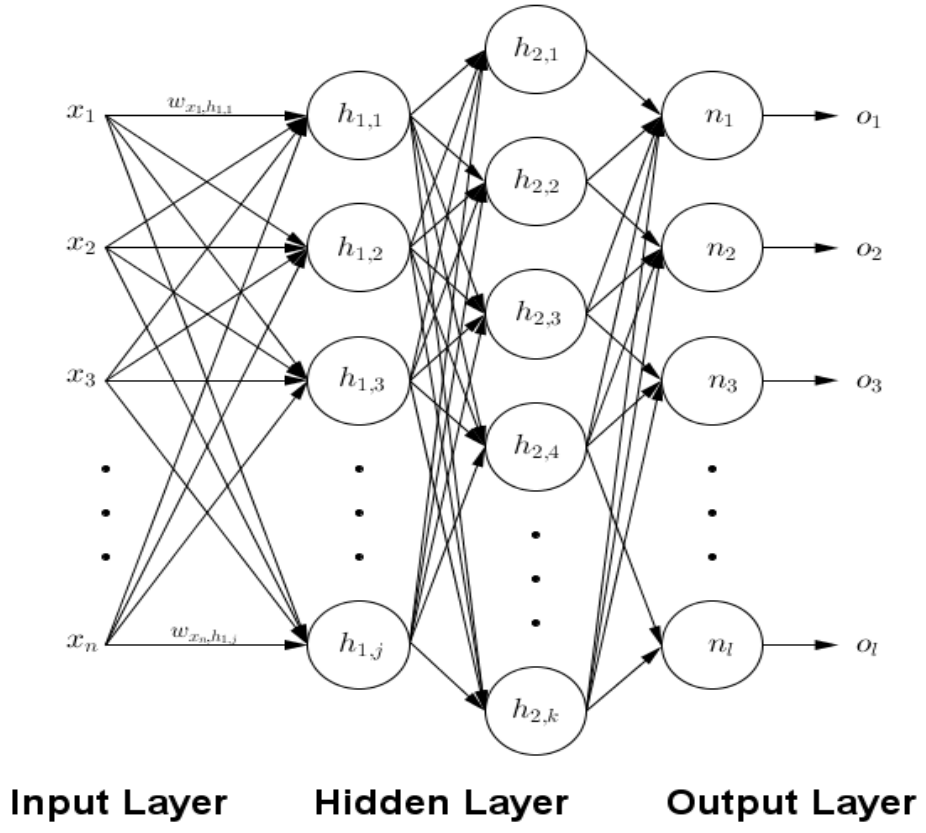


Figure 1.2: Model of an Feedforward Neural Network

In a landmark paper, Cybenko [4] proved in 1989 that a feedforward network containing a single hidden layer or neurons with sigmoidal activation could approximate any continuous function of $\mathbb{R} \to \mathbb{R}$, renewing interest in the field. In 1991 Hornik [5] proved the same for any activation function that is continuous, non-constant, bounded, and monotonically-increasing, showing that it is the multilayer feedforward architecture that provides the universal approximation power of neural networks.

Armed with this theoretical knowledge, research focused on the practical application of feedforward networks. To approximate functions whose analytical form is unknown or non-existent, the proper weights have to be applied to the neuron inputs. The search for these weights is the called the training of the network, and algorithms that perform this task are often called 'learning' or 'teaching' algorithms if applied in a supervised context. The most successful of these learning algorithms is the backpropagation method, discovered by Paul Werbos in 1974 [6] but only popularized by Rumelhart et al. in 1986 [7].

The backpropagation algorithm is a variant of gradient-descent.

It requires the activation function used by the neurons of the network to be differentiable. It also requires a cost function that can be written as an average over cost functions for individual training examples.

The steps of the algorithm are as follows:

1. Forward propagation of the input values to obtain the activation of each neuron.

2. Backwards propagation of the error for each node, in order to calculate the delta (weight update) for each weight. Computing the delta is done by using the calculus chain rule to calculate the partial derivative of the error with respect to a weight.

3. Update each weight according to the gradient and learning rate.

4. Repeat until the error over the data is below a threshold, or the gradient converges, or for a fixed number of iterations .

The algorithm is made possible by using the chain rule from calculus to iteratively compute gradients for each layer, and relies on 4 equations:

1. An equation for the error in the output layer: $\delta^L$, a vector whose components are given by: $\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$. C is the cost/error, the $\partial C/\partial a_j^L$ measures how fast the cost is changing as a function of the $j^{th}$ output activation,$z^L$ is the vector of weighted input to the neurons in layer L, the final term $\sigma'(z_j^L)$ measures how fast the activation function is changing at this neuron.

2. An equation for the error $\delta^l$ in terms of the error in the next layer $\delta^{l+1}$ given by: $\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$. Here $\odot$ is the Hadamard product for matrices in which elements are pair-wise multiplied [8], $(w^{l+1})^T$ is the transpose of the weight matrix for the $l + 1^{th}$ layer.

3. An equation for the rate of change of the cost with respect to any bias in the network: $\frac{\partial C}{\partial b_j^l} = \delta_j^l$, where $b_j^l$ is the bias of the $j^{th}$ neuron of layer $l$.

4. An equation for the rate of change of the cost with respect to any weight in the network: $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$. This lets us compute the partial derivative in term of quantities $\delta_j^l$ and $a_k^{l-1}$ we already know how to compute.

See [9] for an in-depth analysis and proof of the backpropagation algorithm.

Since no assumptions are made over the training dataset, it is possible to use batch, stochastic

or on-line training. In on-line and stochastic learning, each propagation is followed immediately by a weight update. In batch learning, many propagations occur before updating the weights. On-line learning is used for dynamic environments that provide a continuous stream of new patterns. Stochastic goes through the data set in a random order in order to reduce its chances of getting stuck in local minimum. Stochastic learning is also much faster than batch learning since weights are updated immediately after each propagation. Yet batch learning will yield a much more stable descent to a local minimum since each update is performed based on all patterns.

The main limitation of the backpropagation algorithm is the long time needed for proper training. One reason for this can be the vanishing gradient problem.

This fundamental limitation of multi-layered neural networks was first described by Hochreiter in his 1991 thesis [10] and further investigated in two following papers [11, 12]. By observing the gradient of the $l^{th}$ layer of an $L$ layer network, which is given by (in vector form):

$\delta^l = \sum'(z^l)(w^{l+1})^T \sum'(z^{l+1})(w^{l+2})^T \cdot \sum'(z^L) \bigtriangledown_a C.$

Here, $\sum'(z^l)$ is a diagonal matrix whose entries are the $\sigma'(z)$ values for the weighted inputs to the $l^{th}$ layer. The $w^l$ are the weight matrices for the different layers. And $\bigtriangledown_a C$ is the vector of partial derivatives of C with respect to the output activations. The important factors here are $\sigma'(z)$ which get repeatedly multiplied by the weights $w$. If the absolute value of these products are all smaller than 1, then the absolute value of the gradient will be close to 0. Unfortunately, because the activation functions are generally squashing functions with values converging to $-1$ or $-1$ as they approach $-inf$ and $+inf$ respectively, so as $w$ gets

very large or very small, $\sigma'(z) = \sigma'(wa + b)$ goes to 0. Because each layer adds an extra product to the computation of the gradient, the weights furthest from the output are the slowest to change. This has been verified empirically [9, 13, 14].

Despite this issue, deep neural networks have enjoyed success in recent years. Effective pre-training techniques have shown that careful choices in initial weights can lead to efficient learning [15], and specialized architectures such as convolutional nets which are biologically inspired networks in which the weights are tied and connections sparse [16, 17]. Nonetheless these techniques remain specialized workarounds for specific tasks and domain, and are not appropriate for all tasks.

## 1.3 Recurrent Neural Networks

Part of the appeal of neural networks is the parallel with the human brain. However, the network architecture of neurons within the brain is decidedly not a feedforward architecture. In the human brain, billions of neurons are combined in separated lobes and cortices, with electrical signals traveling in all directions. This type of network in which contains cycles are called Recurrent Neural Networks (RNNs). These RNNs are useful because they have superior theoretical computational power. A feedforward network approximates a mathematical function, whereas RNNs approximate dynamical systems. Dynamical systems are essentially functions with an added time component, the same input input can result in a different output at different timesteps. In our case, we consider discrete-time systems, defined by a function $f : X \to X$, with $X$ the configuration space, and the state of the system evolving from state $x \in X$ to $f(x + 1, f(x))$ etc.

It is the presence of cycles in the network that allow these dynamical changes to occur. In theory, RNNs are capable of "remembering" input values for some time by preserving them in some form within the activations of nodes in the network. A properly trained RNN is capable of learning context sensitive information without having to engineer task-specific data representations as input to the network (e.g. triphones or n-grams in common NLP tasks), though these can still be used with RNNs. Additionally, the study of RNNs is essential to any type of research done on modeling or simulating a human or animal brain. Figure 2.1 gives a graphical example of a small RNN.

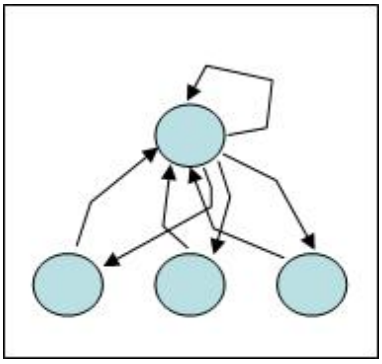

Figure 1.3: Model of a a small Recurrent Neural Network. Image from [18]

Of course, the increased complexity of the network architecture comes at a cost. New techniques for training have to be devised (see section on Backpropagation Through Time below), and in practical cases are almost always slower than feedforward network training algorithms. The universal approximator theorem must be proven again for RNNs. This important result came in 1991, when Siegelmann and Sontag [19] proved that RNNs, of finite size and with sigmoidal activation function for the nodes, have the capacity to simulate a universal Turing machine, confirming the superior computational power of RNNs over simple feedforward networks.

### Backpropagation through Time

For practical applications, a successful training technique for RNNs has been Backpropagation Through Time (BPTT), which was independently derived by numerous researchers but popularized in 1990 by Paul Werbos [20]. It is an adaptation of the well-known backpropagation training method known from feedforward networks. The feedforward backpropagation algorithm cannot be directly transferred to RNNs because the error backpropagation pass presupposes that the connections between units induce a cycle-free ordering. The solution of the BPTT approach is to "unfold" the recurrent network in time, by stacking identical copies of the RNN, and redirecting connections within the network to obtain connections between subsequent copies. This gives a feedforward network, which is compatible with the backpropagation algorithm. Figure 1.4 gives a visual explanation of the basic unfolding idea of BPTT.

In this unfolded, feed-forward network, the weights are identical between the copies. The forward pass of one training batch consists of stacking a new layer at each timestep, i.e. at timestep $n$, the current output $f(x_n)$ is computed from the current input $x_n$ and the previous internal state at time $n-1$. The classical backpropagation algorithm is straightforwardly adapted to this unfolded architecture. The teaching data now consists a single input-output time series. The changes over the tied weights is averaged after each training batch. The remarks concerning the slow convergence of standard backpropagation carry over to BPTT, even more so since the size the network grows for every iteration. This has limited the used of BPTT to small networks, in the order of tens or hundreds of nodes. Another drawback of this batch training method is that the entire input-output time series must be used, which

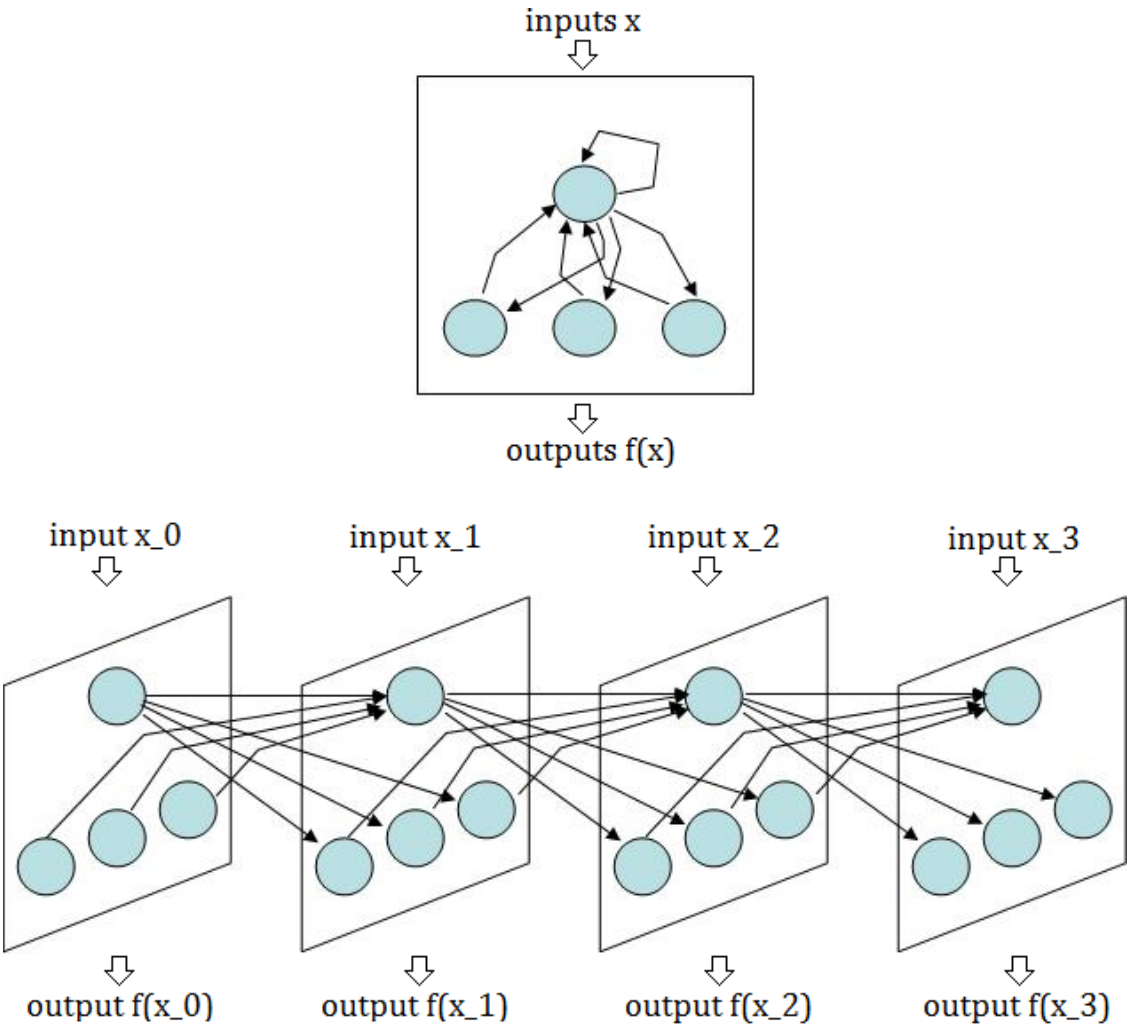


Figure 1.4: Visualization of Backpropagation Through Time unfolding the RNN.

excludes any kind of online adaptation. A possible solution to both the long computation time and the need for the entire time series is to truncate the past history and use a only $p$ past inputs, in effect doing mini-batches of training. However this limits the memory capacity of network and prevents time dependencies longer than $p$ to be modeled.

The shortcomings of the BPTT algorithm hold for other classical RNN training methods.

- The algorithms are computationally expensive, limiting their use to small networks.
- Gradient-descent methods suffer from vanishing gradients, making it impossible to guarantee converge. This also leads to dependencies requiring long-term memory becoming hard to learn since gradient information exponentially dissolves over time.

# Chapter 2

# The Reservoir Computing Paradigm

In 2001 a fundamentally new approach to RNN design and training was proposed independently by Wolfgang Maass under the name of Liquid State Machines and by Herbert Jaeger under the name of Echo State Networks. This approach is now often referred to as the Reservoir Computing Paradigm. Reservoir computing also has predecessors in computational neuroscience (see Peter Dominey's work in section 3.1) and in machine learning as the Backpropagation-Decorrelation learning rule proposed by Schiller and Steil in 2005 [21]. As Schiller and Steil noticed, when applying BPTT training to RNNs the dominant changes appear in the weights of the output layer, while the weights of the deeper layer converged slowly. It is this observation that motivates the fundamental idea of Reservoir Computing: if only the changes in the output layer weights are significant, then *the treatment of the weights of the inner network can be completely separated from the treatment the output layer weights.*

## 2.1 Reservoir Models

Reservoir computing methods differ from traditional RNN learning techniques by making a conceptual and computational separation between the reservoir, i.e. the inner neurons and

weights of the network, and the readout, the neurons and weights that produce the output. More specifically, in traditional supervised learning the error between the desired output and computed output will potentially influence the weights of all the network. By contrast, in the Reservoir Computing paradigm, the error will only influence the weights of the readout layer. The weights of the reservoirs connections are set at the start of the learning and do not change. Figure 2.1 give an graphical overview of the reservoir model.

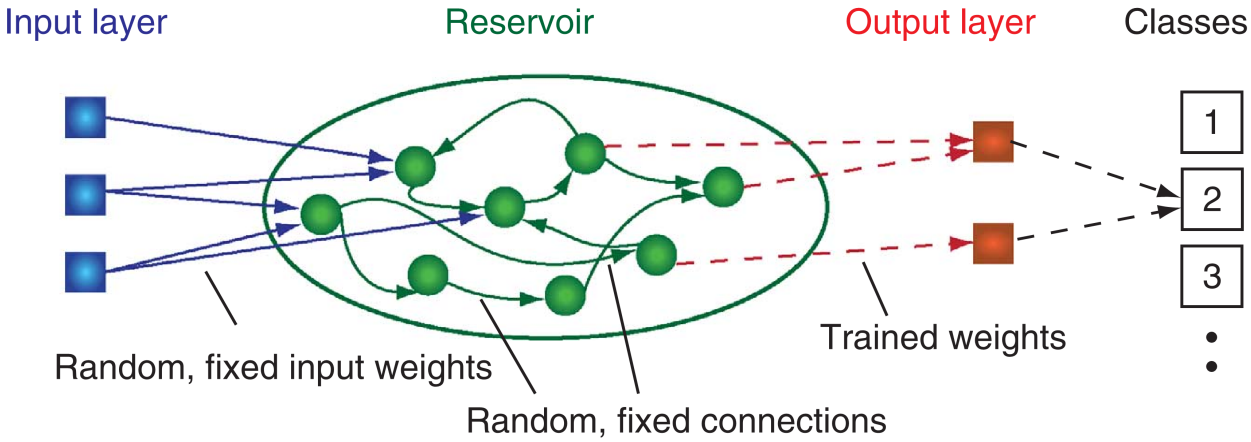


Figure 2.1: Model of a Reservoir Neural Network. Image from *Information Processing using a Single Dynamical Node as Complex System* by Appeltant et al. [22]

Despite being a relatively new concept, the reservoir computing paradigm has already been successfully applied to a range of scientific fields, including but not limited to neurosciences, speech processing and robotics. See chapter 3 for a review of papers using reservoir computing. The following sections detail different brands of reservoir methods: Echo state Network, Liquid State Machines, and other networks that fall into the reservoir category. Most of these techniques were developed independently and have only been united under the term reservoir since 2008. Each brand is a specialized version of reservoir computing, with its own name, history and motivations.

### Echo State Networks

The Echo State Networks (ESNs) method, created by Herbert Jaeger and his team [23], represents one of the two pioneering reservoir computing methods. Having observed that if a RNN possesses certain behavioral properties (separability and echo state, see Section 2.2), then it is possible to achieve high classification performance on practical applications simply by learning a linear classifier on the readout nodes, for example using logistic regression. The untrained nodes of the RNN are part of what is called the dynamical reservoir, which is where the name Reservoir Computing comes from. The Echo State names comes from the input values echoing throughout the states of the reservoir due to its recurrent nature. Because ESNs are motivated by machine learning theory, they usually use sigmoid neurons over more complicated biologically inspired models.

### Liquid State Machines

Liquid State Machines (LSMs) are the other pioneer method of reservoir computing, developed simultaneously and independently from Echo State Networks, by Wolfgang Maass [24]. Coming from a computational neuroscience background, LSMs use more biologically realistic models of spiking integrate-and-fire neurons and dynamic synaptic connection models, in order to understand the computing power of real neural circuits. As such LSMs also use biologically inspired topologies for neuron connections in the reservoir, in contrast to the randomized connections of ESNs. However, creating a network that mimics the architecture of the cerebral cortex requires some engineering of the weight matrix; and simulating spiking neurons, which are dynamical systems themselves in which electric potential accumulates until it fires leaving a trail of activity bursts, is slow and computationally intensive.

Both these facts make the LSM method of reservoir computing more complicated in their implementation and have not commonly been used for engineering purposes.

**Other Types of Reservoir Networks**

As mentioned previously, the idea of treating the reservoir and the readout layer separately was also devised by Schiller and Steil [21]. They proposed an algorithm called Backpropagation-Decorrelation as a new RNN training method, boasting fast convergence and good practical results, providing a conceptual bridge between traditional BPTT which applies weight update to the whole network, and the reservoir computing paradigm which focus solely on the output layer weights.

We should also mention that Peter Dominey's was probably the first to properly state the reservoir computing principles by observing that 'there is no learning (adaptation) deep within the neural network, and that these connections are randomized (in the pre-frontal cortex in his case)' [25]. Dominey called these networks Temporal Recurrent Networks, and only recently have computational researchers become aware of the similarities with reservoir networks.

The Reservoir Computing paradigm can be extended beyond neural networks. Taking the idea of a reservoir and echoes quite literally, an experiment was set up where the inputs were projected into a bucket of water, and by recording the waves bouncing around the liquid's surface, the authors were able to successfully train a pattern recognizer. Another exotic idea for an untrained reservoir is an E.Coli. bacteria colony, with chemical stimuli as input and protein measures as output. These experiments show that the reservoir computing paradigm may be suitable to harness computational power with unexpected hardware material.

Despite having different names, all the approaches above (ESNs, LSMs, BPDC, Temporal Recurrent) are now referred to as reservoir computing, and we will be using this term throughout the rest of the paper.

## 2.2 Reservoir Computing Theory

While reservoir computing techniques can rely on randomized networks to obtain good performance, there is no guarantee that it is optimal. In fact 'random' is almost by definition the opposite of 'optimal'. While no single technique will work for all tasks, some general rules exist to create reservoirs with good behavior.

**Echo State Property**

An important element for Reservoir Computing to work, coming from the ESNs approach, is that the reservoir should have the echo state property. This condition in essence states that the effect of a previous state and a previous input on a future state should vanish gradually as time passes, and not persist or worse, get amplified. For practical purposes, the echo state property is assured if the matrix W of reservoir weights is scaled so that its spectral radius $\rho(W)$ (i.e., the largest absolute eigenvalue) is close to or inferior to 1. Intuitively, the spectral radius is a crude measure of the amount of memory the reservoir can hold, the small values meaning a short memory, and the large values a longer memory, up to the point of over-amplification when the echo state property no longer holds. Since one incentive to use RNNs is their capacity to have a memory of the inputs, it is important that any method that guarantees the echo state property does not reduce the memory capacity to nothing. A practical test to measure this memory capacity is: can the reservoir output recreate the

input, with a k steps delay?

**Separability Property**

Another important aspect of good reservoirs is the capacity to separate two different inputs, i.e. two different input signals should not have the same output signals. A good heuristic measure of a reservoir's separability power is to compute the distance between different states caused by different input sequences. One method of achieving good separability is to make the matrix defining the network connections sparse, making the activation signals within the network decoupled from each other. Larger networks also lead to more varied activations, and higher separability power.

**Topology**

The question as to what is the best way to arrange the neurons in the network is still an unresolved one. A larger number of neurons in the network will mean more activation signals, allowing for finer grain classification and higher performance (see [26] for an example on how size can improves performance). But, concerning the particular topology of the network, it has been noted that there is substantial variation in ESN performance among randomly created reservoirs [27]. After all randomness is almost by definition the opposite of optimal. As mentioned previously, a sparse matrix helps with the performance, but the optimal topology for a reservoir network remains unknown. In a 2004 study [28], different possible topologies were tested, such as scale-free, small world or biologically inspired. Scale-free and small world networks are both widely studied graph topologies that are frequently observed in nature. However, none tested to "perform significantly better than simple random networks" on two dynamical system prediction tasks. Still the difference in performance amongst these

random networks indicates that similar approaches might be useful.

# Chapter 3

# How the Reservoir Computing Paradigm is used

The universality of the reservoir computing paradigm has lead to its application in a diversity of scientific fields. Since 2009, over a thousand papers have been published on the subject [29]. Among these fields are computational neuroscience [30, 31, 32, 33], reservoir computing theory [34, 35, 36, 37], speech processing [26, 38, 39] and physical implementation [22, 40, 41]. This survey focuses on a select few of these papers with the goals of showing the diversity of reservoir computing applications and discussing the strength and weaknesses of these approaches. Lastly, a quick review of a similar randomized network technique, named Extreme Learning Machines [42, 43], will be given to show the broad scope of the random network principle of reservoir computing.

## 3.1 Neuroscience

Usage of the Reservoir Computing paradigm in the sciences at large can be classified in two broad categories: use as a powerful machine learning tool, or use as a biologically feasible mechanism.

In research that predates the formulation of the reservoir computing paradigm, Dominey et al. [44, 45] argue that the brain exhibits reservoir-like processes, i.e. random connection between neurons and linear or simple learning on only the output layer. The learning algorithm described by Dominey is a version of the Least Mean Squares algorithm, in which the output weights are updated by gradient-descent in order to minimize the mean square of the error.

Continuing in this vein of research, in 2011 Nature Neuroscience published a paper by Bernacchia et al [31] which shows how the brain could predict expected rewards over multiple timescales by processing the available information through a reservoir network. By measuring the activity of cortical neurons in monkeys performing a gaming task of matching pennies, the authors observed that some neurons firing indicated that the monkey expected a reward immediately, while others indicated an expected reward in the more distant future. The timescale of these neuronal responses ranged from hundreds of milliseconds to tens of seconds. To replicate this phenomenon, the authors implemented a reservoir neural network of 1000 neuron,a similar size to that measured on the animals. The activity of neurons evolves according to $\frac{dv}{dt} = J \cdot v(t) + h \cdot Rew(t)$ where $v$ is the vector of neuron activity, $J$is the synaptic connectivity matrix of their interactions and $h$is a vector representing the relative strength of the reward input $Rew(t)$ to each neuron. The matrix J of the connection weights was created to be sparse and random. By broadly distributing the connection weights, the authors allows the network to respond to inputs on a wide variety of timescales. In addition, the connection matrix was made to be normal ($J^*J = JJ^*$). This ensures [46] that the network dynamics are robust to small changes of the connection strengths. It

was then observed that the activity of neurons in the reservoir displayed similar patterns that observed on live monkeys. The paper concluded that animals may be able to process information on multiple timescales thanks to the reservoir's recurrent network capability to maintain multiple memory traces at once.

Partnering with Dominey, the recent work of Hinaut et al. [32, 33] also uses reservoir computing as a explanation to a biological process. Hinaut explores how language is learned, working at the frontiers of neuroscience, natural language processing and robotics. In the paper from May of 2014 [33] Hinaut and his colleagues argue in that human language can be learned through general associative mechanisms in a stimulus rich environment, in contrast to the Chomsky's innatism that believes that the child's stimulus environment is too poor and that language can only be learned via a highly specialized universal grammar system. The authors aim to show child's social environment contains enough non-linguistic information that help with the acquisition of phoneme, word, sentence meaning. Through simple interactions with an iCub robot, they showed that the robot is not only capable of learning the grammatical structure of sentences, but also able to produce sentences describing the human's actions. For those two tasks, comprehension and production, the neural language model they used was a random reservoir network with bio-inspired neurons. The reservoir is modeled after the human prefrontal cortex: the reservoir corresponds to the cortex, and the readout layer corresponds to the striatum. The reservoir is composed of leaky neurons with sigmoid activation. The following equation describes the internal update of activity in the reservoir: $x(t+1) = (1\alpha)x(t) + \alpha f(W_{res}x(t) + W_{in}u(t+1))$ where $x(t)$ represents the reservoir state at time $t$; $u(t)$ denotes the input that time;$\alpha$ is the leak rate; and $f$ is the

hyperbolic tangent (tanh) activation function. $W_{in}$ is the connection weight matrix from inputs to the reservoir and $W_{res}$ represents the recurrent connections between internal units of the reservoir. The readout activity is defined by the weight matrix $W_{out}$ which is multiplied to $x(t)$. This readout matrix was trained by linear regression with bias and pseudoinverse method described by Jaeger in 2001 [23]. The results of these experiments is a robot capable of learning that sentences like "John hit Mary" and "Mary was hit by John" have the same meaning and same grammatical structure: agent(John), object(Mary), predicate(hit).

## 3.2 Machine Learning

The reservoir computing paradigm is general and powerful enough to also be applied as tool, rather than as an explanation for natural phenomena. In 2012 Oubbati et al. [37] adapted the reservoir paradigm to multi-objective problems, in which more than one objective function need to be optimized together. These types of problems are perhaps more naturally occurring than single-objective problems, after all most human decision need to balance multiple goals, for example social/economic, speed/quality. Unfortunately multi-objective problems are also much harder, if not impossible to solve exactly, since the size of all the Pareto (non-dominatied) optimal solutions is potentially infinite. In this paper, the authors apply the reinforcement learning framework to the multi-objective problematic, and use a technique called Adaptive Dynamic Programming (ADP) [47], which was developed to solve one of the commons pitfalls of multi-objective reinforcement learning algorithms, the curse of dimensionality. The ADP tool approximates the expected reward using a system called 'Critic'. The reward can be seen as the agent's preference over the objectives,

and the expectation is the agent's belief related to how actions impact outcomes. The authors state that the critic is usually a neural network, and requires powerful computational tools to be effective. This is where the reservoir computing paradigm becomes useful. The reservoir estimates several rewards simultaneously and provides their gradients, which are required for the agent to adapt its behavior to the multiple objectives, while being a lot less computationally intensive than traditional neural networks, since reservoir only optimized the output layer of the network. The authors test their hypothesis by implementing this Reservoir-Computing-ADP in a simulated environments, in which an agent must balance three different objective functions. It was shown that the agent was able to estimates several utilities simultaneously, reaching different Pareto optimal solutions depending on how the objectives were weighted.

## 3.3 Speech Processing

One of the attractive traits of recurrent neural networks is their capacity to process temporal information. One domain where such temporal information is important is in speech processing. For example, for an automatic speech recognizer to work properly, a sound uttered at time $t$ will influence the recognition probabilities of phonemes both before and after. To allow traditional feed-forward networks to process these temporal dependencies, solutions such as 'context-dependent' phones and N-grams have been engineered. An N-gram is a contiguous sequence of n items extracted from text or speech. These can be phonemes, syllables, letters or words depending on the task. Feeding these N-grams into a feedforward network allows it to process some contextual information. However the range of the context

is fixed by preprocessing of the data, and cannot be learned by network. More generally, to avoid the the 'curse of dimensionality', it is assumed that the simplifying Markov property holds, i.e.that future states only depend on the near past.

The memory capacity of RNNs allows reservoir networks to store this information and use it without needing preprocessing, though it certainly is compatible with such techniques. In the field of natural language processing, reservoir computing research has been led by B. Schrauwen and Jean-Pierre Martens of Ghent University. Amongst the many papers produced by the group ([48, 49]), an important one for the popularity of the reservoir computing paradigm in the speech recognition domain was published in 2010 by Triefenbach, Jalalvand and the aforementioned Schrauwen and Martens [26]. This particular paper explores the task of continuous phoneme recognition on the popular TIMIT dataset [50], with the goal of comparing the performance of single a reservoir network against the novel hierarchical reservoirs architecture, as shown in figure 3.1.

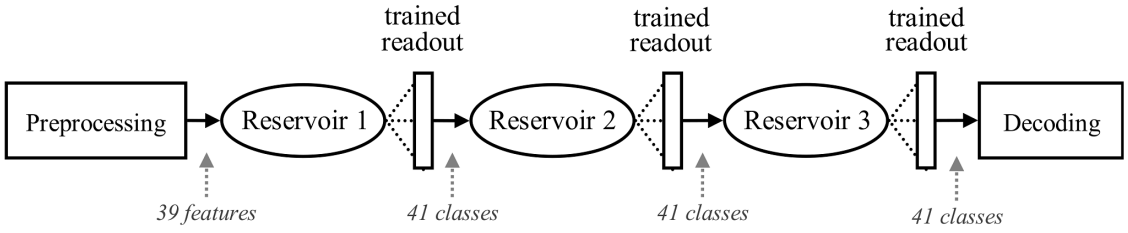


Figure 3.1: The reservoir architecture proposed by Treifenbach et al. Image from [26]

The authors argue that additional reservoirs can correct some of the errors made by the previous reservoir. And indeed, in their implementation they observed that the second layer induces a significant improvement of the correction error rate by 3-4%. They also observed that additional layers beyond the second only provided minimal gain. The overall perfor-

mance of reservoir networks for the task was comparable to other state of the art systems such as Hidden Markov Models and Deep-Belief nets.

## 3.4 Physical Implementation

If reservoir networks are to be used in practical applications, implementing them using dedicated hardware would further improve the computational speed of the system over traditional RNNs optimization methods. In recent work published by Nature, Appeltant et al. [22, 41] explore this challenge, with feasibility, performance and resource-efficiency in mind. The authors note that implementing a randomized network architecture, the type of reservoir that is usually coded in simulations, would be a difficult task in hardware. Many components would be needed, one for each reservoir neuron, and the construction process would have to adapt itself every time a new random architecture was devised. In addition to the construction difficulties, not all random networks are efficient, as discussed under **Topology** in section 2.2, leading to potentially wasted material. To answer these problems, the authors propose to implement a reservoir computer in which the usual reservoir structure of multiple connected nodes is replaced by a dynamical system comprising a nonlinear node subjected to delayed feedback. The delayed feedback creates multiple values that act as 'virtual nodes', whose values are connected to the readout layer, and whose connecting weights are trained using standard reservoir computing procedures. Figure 3.2 gives an overview of the proposed reservoir computer architecture.

For the non-linear node, the authors chose the Mackey-Glass oscillator whose state$X$ evolves according to $\dot{X}(t) = -X(t) + \frac{\eta \cdot [X(t-\tau)+\gamma \cdot J(t)]}{[1+X(t-\tau)+\gamma \cdot J(t)]^p}$ with $\eta, \gamma, p$ tunable parameters. $J(t)$ is the

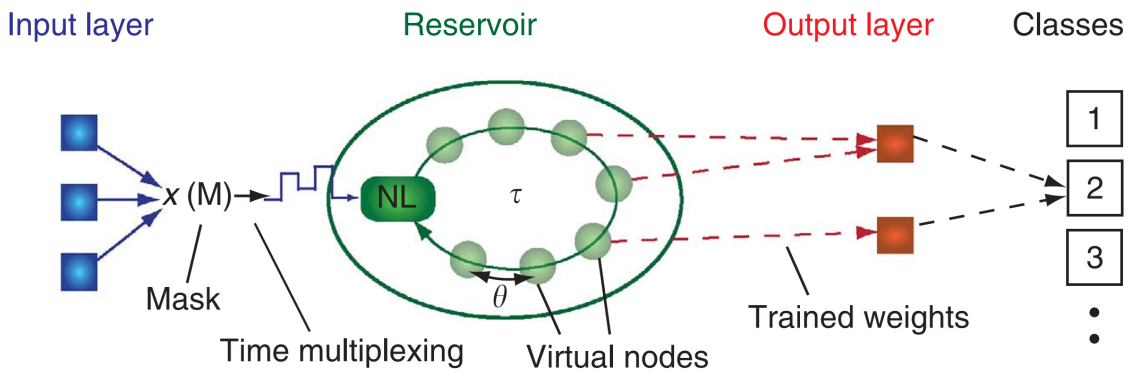


Figure 3.2: Scheme of the Reservoir Computer proposed by Appeltant et al. Image from [22]

output of the mask, a preprocessing of the input, where $\tau$ is the delay of the feedback loop, $\theta$ is $\frac{\tau}{N}$ with $N$ the number of virtual nodes.

The authors tested the validity of this model on two tasks, a spoken digit recognition task and the NARMA task introduced in [51] which has become a benchmark for reservoir computing (see [52] for an example). On both the performance is close to or better than that obtained previously with traditional, fully randomized reservoir networks, indicating the potential of this approach.

## 3.5 Other Randomized Networks

It should be noted that the idea of treating the readout layer separately from the network has also been applied to more traditional feed-forward networks. Under the name of 'Extreme Learning Machines' (ELM), Huang et al. [42, 43] developed a tool very similar to the basic theory of reservoir computing. The weights of the hidden layers of a feedforward network can be randomized, and with proper training of the output weights, good results can be obtained. The authors proves several theorems concerning these ELMs, notably that they are

universal function approximators. Toy problems such as noisy function approximation, and real world tasks such as automated medical diagnostic prediction are tackled with success, comparing favorably to traditional feedforward techniques like backpropagation or support vector machines, showing the promise of the Reservoir / ELM approaches to neural network machine learning, with ELM learning non-temporal classifiers and reservoirs approximating dynamical systems with temporal dependencies.

# Chapter 4

# Conclusion and Future Work

The reservoir computing paradigm has already proven itself as a powerful tool in numerous scientific domains, both at an biological explanation of animal neural processes, and as a computational tool in tasks requiring complex temporal information processing.

## 4.1 Conclusion

In conclusion:

- The simple reservoir computing paradigm states that recurrent neural networks can be efficiently trained by optimizing only the weights connected to the output neurons, leaving the weights of internal connections unchanged.
- The reservoir computing paradigm offers a theoretically sound approach to modeling dynamical systems with temporal dependencies.
- The paradigm also promises to be much more computationally efficient than traditional RNN approaches that aim at optimizing every weight of the network.
- Reservoir computing techniques have successfully been applied to classical artificial intelligence problems, providing state-of-the-art performance on engineering problems,

and offering explanations on biological brain processes.

Despite these advances, reservoir computing is still a new research topic. Because the paradigm is so general, many options are available to research: the type of neuron, the architecture of the network, the training method for the output weights; no consensus exists and is most certainly task-dependent. It also remains to be seen if the paradigm can be applied to large-scale technological ventures, as currently only a small number of research groups are working with reservoirs. Today neuro-scientists focus on which parts of the brain display reservoir-like patterns and how to model these parts; while computational scientists apply the paradigm to new technical fields, exploring and optimizing the different options mentioned above.

## 4.2 Future Work

As a member of the Speech Lab @ Queens College [53], my work centers on computational speech analysis, with a focus on understanding how prosody and intonation communicate information. As an NSF IGERT [54] fellow, I am also interested in multi-disciplinary work, possibly working with biomedical data. The reservoir computing paradigm in its general classification power is well adapted to multi-disciplinary work.

I have already explored toy problems using reservoir networks, implemented using the OGER [55] Python toolbox. Figure 4.1 shows my results on a simple task of replicating a noisy input with a time delay of 5 steps, which even a small reservoir is capable of doing perfectly, showcasing the memory capacity of the paradigm.

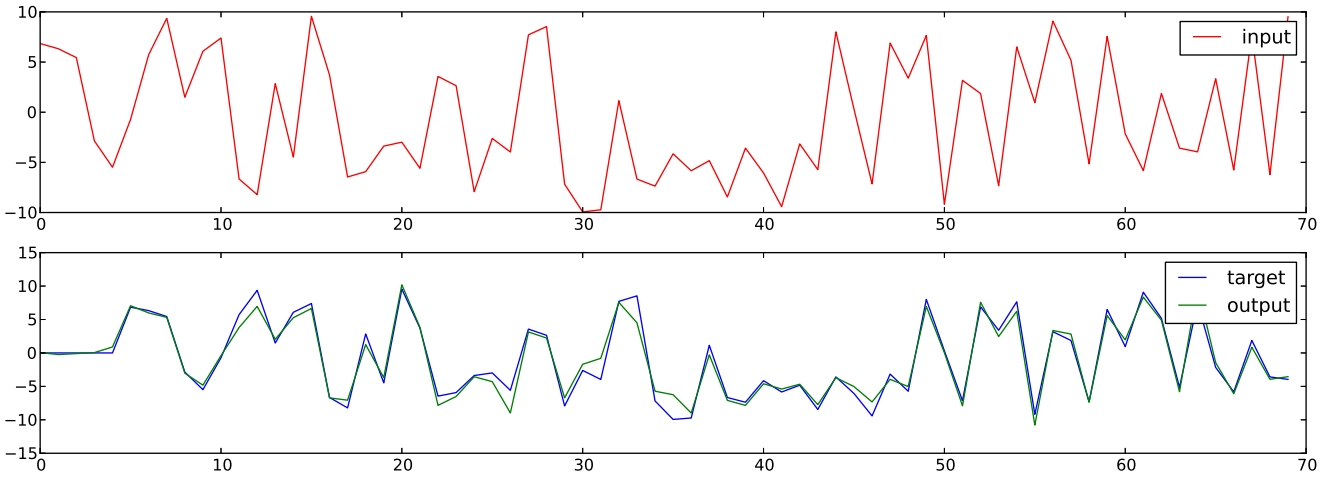


Figure 4.1: Results of a simple 50 neuron random reservoir on a toy task. Note: the results are artificially worsened for visualization purposes

# Bibliography


[1] F. Rosenblatt, "The perceptron–a perceiving and recognizing automaton," 1957.

[2] M. Minsky and P. Seymour, "Perceptrons," 1969.

[3] S.-W. Lee and C. Moraga, "A cosine-modulated gaussian activation function for hyperhill neural networks," in *, 3rd International Conference on Signal Processing, 1996*, pp. 1397–1400 vol.2, 1996.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, 1989.

[5] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, 1991.

[6] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," 1974.

[7] D. E. Rumelhart, Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[8] E. Million, "The hadamard product," 2007.

[9] M. Nielsen, "Neural networks and deep learning," 2014.

[10] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

[11] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[12] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[13] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[14] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

[15] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1139–1147, 2013.

[16] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, ACM, 2008.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[18] J. L. McClelland, "Explorations in parallel distributed processing. chapter 8: Recurrent backpropagation." `http://web.stanford.edu/group/pdplab/pdphandbook/handbookch9.html`.

[19] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Applied Mathematics Letters*, 1991.

[20] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, 1990.

[21] J. J. Steil, "Backpropagation-decorrelation: online recurrent learning with o (n) complexity," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, 2004.

[22] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature communications*, 2011.

[23] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks - with an erratum note," 2001.

[24] T. Natschläger, W. Maass, and H. Markram, "The "liquid computer": A novel strategy for real-time computing on time series," *Special Issue on Foundations of Information Processing of TELEMATIK*, 2002.

[25] P. F. Dominey and F. Ramus, "Neural network processing of natural language: I. sensitivity to serial, temporal and abstract structure of language in the infant," *Language and Cognitive Processes*, 2000.

[26] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens, "Phoneme recognition with large hierarchical reservoirs," in *Advances in Neural Information Processing Systems*, Neural Information Processing System Foundation, 2010.

[27] F. Jiang, H. Berry, and M. Schoenauer, "Supervised and evolutionary learning of echo state networks," in *Parallel Problem Solving from Nature–PPSN X*, 2008.

[28] B. Liebald, "Exploration of effects of different network topologies on the esn signal cross-correlation matrix spectrum," masters, School of Engineering & Science at International University Bremen, 2004.

[29] G. Scholar, "Reservoir computing articles since 2014," 2014.

[30] P. Buteneers, B. Schrauwen, D. Verstraeten, and D. Stroobandt, "Real-time epileptic seizure detection on intra-cranial rat data using reservoir computing," in *Advances in neuro-information processing*, pp. 56–63, Springer, 2009.

[31] A. Bernacchia, H. Seo, D. Lee, and X.-J. Wang, "A reservoir of time constants for memory traces in cortical neurons," *Nature Neuroscience*, 2011.

[32] X. Hinaut, "On-line processing of grammatical structure using reservoir computing," *Artificial Neural Networks and Machine Learning ICANN 2012*, 2012.

[33] X. Hinaut, M. Petit, G. Pointeau, and P. F. Dominey, "Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks," *Frontiers in Neurorobotics*, 2014.

[34] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz*, vol. 26, no. 4, pp. 365–371, 2012.

[35] M. Hermans and B. Schrauwen, "Recurrent kernel machines: Computing with infinite echo state networks," *Neural Computation*, vol. 24, no. 1, pp. 104–133, 2012.

[36] S. P. Chatzis and Y. Demiris, "Echo state gaussian process," *Neural Networks, IEEE Transactions on*, vol. 22, no. 9, pp. 1435–1445, 2011.

[37] M. Oubbati, T. Oess, C. Fischer, and G. Palm, "Multiobjective reinforcement learning using adaptive dynamic programming and reservoir computing," *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2012.

[38] A. Jalalvand, K. Demuynck, and J.-P. Martens, "Noise robust continuous digit recognition with reservoir-based acoustic models," in *2013 International Symposium on Intelligent Signal Processing and Communication Systems, Proceedings*, pp. 204–209, 2013.

[39] A. Jalalvand, F. Triefenbach, K. Demuynck, and J.-P. Martens, "Robust continuous digit recognition using reservoir computing," *COMPUTER SPEECH AND LANGUAGE*, 2014.

[40] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific reports*, vol. 2, 2012.

[41] L. Appeltant, G. Van der Sande, J. Danckaert, and I. Fischer, *Constructing optimized binary masks for reservoir computing with delay systems*. 2014.

[42] G.-b. Huang, Q.-y. Zhu, and C.-k. Siew, *Extreme learning machine: Theory and applications*. 2006.

[43] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: a survey," *International Journal of Machine Learning and Cybernetics*, 2011.

[44] P. F. Dominey, "Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning," *Biological Cybernetics*, 1995.

[45] P. F. Dominey, M. Hoen, J.-M. Blanc, and T. Lelekov-Boissard, "Neurological basis of language and sequential cognition: evidence from simulation, aphasia, and erp studies," *Brain and Language*, vol. 86, pp. 207–225, 08/2003 2003.

[46] L. N. Trefethen and M. Embree, "Spectra and pseudospectra: The behavior of nonnormal matrices and operators," 2005.

[47] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: an introduction," *Computational Intelligence Magazine, IEEE*, 2009.

[48] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: A case study," *Inf. Process. Lett.*, 2005.

[49] A. Jalalvand, "Connected digit recognition by means of reservoir computing," *Interspeech*, 2011.

[50] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "DARPA TIMIT acoustic phonetic continuous speech corpus CDROM," 1993.

[51] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *Neural Networks, IEEE Transactions on*, 2000.

[52] M. Lukosevicius and H. Jaeger, "Survey: Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, no. 3, p. 127149, 2009.

[53] "Speech lab @ queens college." `http://speech.cs.qc.cuny.edu/`.

[54] "From data to solutions: Integrative graduate education and research traineeships." http://www.cs.columbia.edu/igert/index.shtml.

[55] "Organic environment for reservoir computing." http://organic.elis.ugent.be/organic/engine.