# Reservoir Computing

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.

2014

Reservoir Computing

by

Felix Grezes

Advisor: Pr. Andrew Rosenberg

**Introduction:** Even before Artificial Intelligence was it's own field of computational science, men have 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 limited 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 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 field will be reviewed.

# Contents

**1 History of Neural Networks** **1**

1.1 Appeal and Historical Recap . . . . . 1

1.2 Feed-Forward Networks: Definitions and Theory . . . . . 2

1.3 Recurrent Neural Networks . . . . . 6

**2 The Reservoir Computing Paradigm** **9**

2.1 Reservoir Models . . . . . 10

2.2 Reservoir Computing Theory . . . . . 12

**3 How the Reservoir Computing Paradigm is used** **15**

3.1 Neuroscience . . . . . 15

3.2 Physics and Machine Learning . . . . . 18

3.3 Natural Language Processing . . . . . 18

3.4 Other Randomized Networks . . . . . 18

**4 Conclusion and my own future work** **19**

**Bibliography** **21**

# 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 () logical operation, since the perceptron is only a linear classifier. Still, research in the field stagnated until Werbos proposed the backpropa-

gation 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: $f(x) = \varphi(w \cdot x)$ , with $\varphi$ the chosen activation function, and $w, x$ the same as in the perceptron, i.e. respectively the weight vector and the input vector. Figure 1.1 gives a visualization of the artificial neuron model.

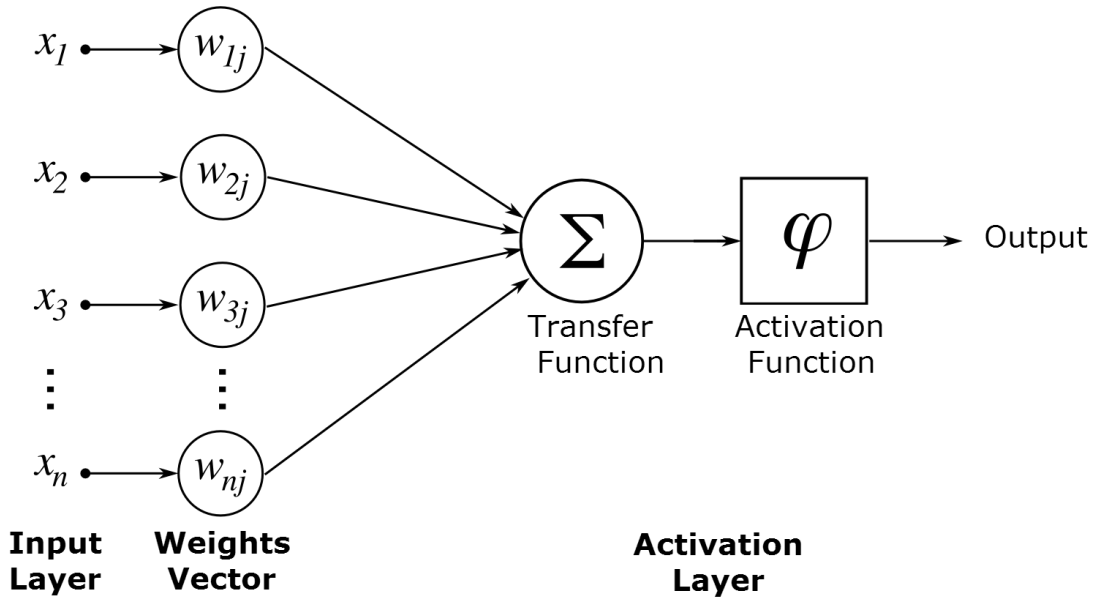


Figure 1.1: Model of an Artificial Neuron

The transfer function is almost always summation. 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 who are continuously differentiable. Research is also done on more esoteric activations, such as the cosine or radial functions [3].

Since the perceptron alone can only learn linear functions, and because it is natural to do so, neurons can be combined into networks, as neurons are the brain. The first type of network architecture we consider is the feedforward network, in which the graphical representation of the network does not contain cycles.

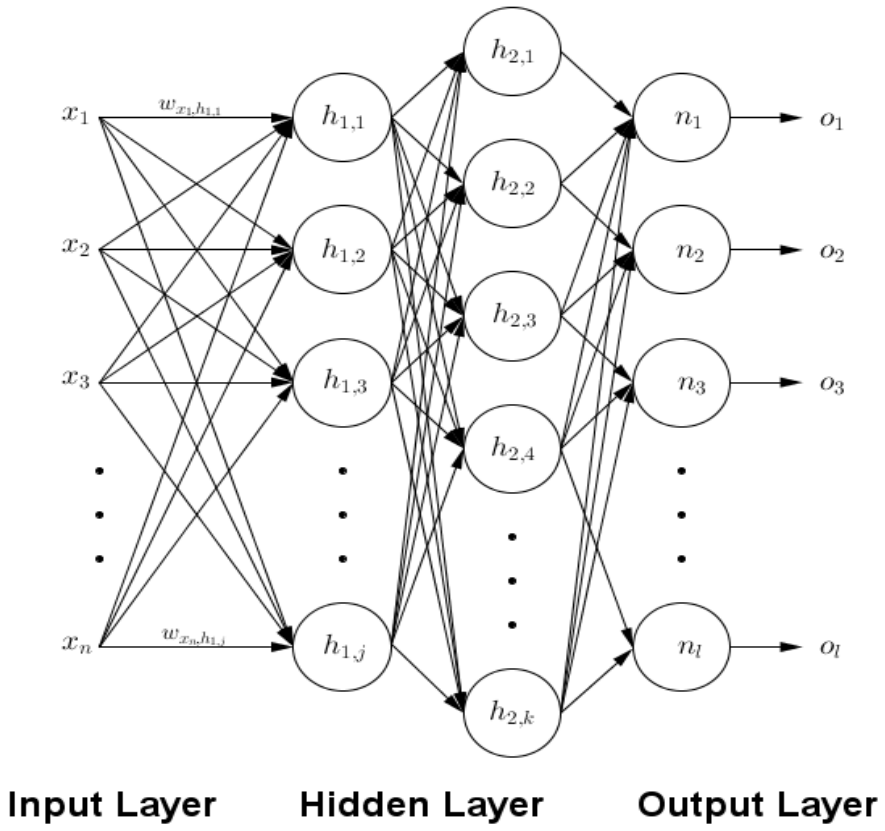


Figure 1.2: Model of an Feedforward Neural Network

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.

With the theoretical hurdles passed, 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, popularized by Paul Werbos in 1974 [6].

The backpropagation algorithm is a variant of gradient-descent. It requires a known, desired output for each input value in order to calculate the gradient of the error and is therefore considered to be a supervised learning method. It also requires the activation function used by the neurons of the network to be differentiable.

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 to a parameter alpha (learning rate).

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

The algorithm is made possible by using the chain rule from calculus to iteratively compute gradients for each layer. 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. Because each update is multiplied by the learning rate (smaller than 1), and propagated to the next layer where the update is again reduced by the learning rate, if the network is too large and the learning rate inadequate, then the weights on the hidden layers closest to the input values might update too slowly for practical uses. Still, with the advances in computing power, and with multi-threaded techniques, the backpropagation algorithm has been used in recent years for speech recognition, image manipulation, medical diagnostics and more. (see 'backpropagation' + 'field' on Google scholar for examples)

## 1.3 Recurrent Neural Networks

For better or for worse, 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, cortices and more, with electrical signals traveling in all directions. This type of network in which the graphical representation contains cycles are called Recurrent Neural Networks (RNNs).

But 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 $f(x)$, then to $f(f(x))$ 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. The following picture gives a graphical example of a small RNN.

(pic)

Of course, the increased complexity of the network architecture comes at a cost. New techniques for training have to be devised, and in practical cases are almost always slower than feedforward network training algorithms. Theoretical mathematical results from feedforward networks must be proven again for RNNs, if possible. For example, one important theoretical result came in 1991, when Siegelmann and Sontag [7] 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 computational power of RNNs.

**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 [6]. 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. In this unfolded, feed-forward network, the weights connected to each iteration of a same original neuron are tied.

(pic)

The remarks concerning the slow convergence of standard backpropagation carry over to BPTT, even more so since the size the network grows every iterations. This has limited the

used of BPTT to small networks, in the order of tens or hundreds of nodes.

# Chapter 2

# The Reservoir Computing Paradigm

In this context of difficult progress for RNNs, 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, which had predecessors in computational neuroscience (Peter Dominey 1995) and subsequent ramifications in machine learning as the Backpropagation-Decorrelation learning rule proposed by Schiller and Steil in 2005, is now increasingly often collectively referred to as Reservoir Computing (RC).

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 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, 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 RNN itself are set at the start of the learning and do not change. (DRAW PIC of influence of error) Despite being a relatively new concept, the Reservoir computing paradigm has already been successfully applied to a range of scientific fields. The following are different brands of reservoir methods. Most of these techniques were developed independently and have only been united under the term Reservoir since 2008. Each brand has its own history and mindset.

**Echo State Networks**

The Echo State Networks (ESNs) method, created by Herbert Jaeger and his team, represents one of the two pioneering reservoir computing methods. Having observed that if a RNN possesses certain algebraic properties, 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 linear 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. ESNs usually use sigmod neurons, but more biologically inspired models such as leaky integrator neurons have also been used.

**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. 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 bio-inspired topologies and metric constraints for neuron connections in the reservoir, instead of the randomized network of ESNs. This usually makes LSMs more complicated to implement, and less useful for practical purposes.

**Temporal Recurrent Networks**

We should also mention Peter Dominey's decade long research on cortico-striatal circuits in the human brain. His research aims at elucidating complex neural structures rather than theoretical computational principles, but he 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). Only in 2008 have computational researchers become aware of Dominey's work.

#### Other Exotic Networks

As mentioned previously, the idea of treating the reservoir and the readout layer separately was also devised by the Schiller and Steil. They proposed an algorithm called Backpropagation-Decorrelation as a new RNN training method, boasting fast convergence and good practical results.

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.

## 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 (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.

**Separability Property**

Another important aspect of good reservoirs is the capacity to separate two different inputs, i.e. the output activity should differ. However this contrasts with the previously stated Echo State Property; if the two signals only vary in the very beginning, than the reservoir will eventually stabilize to the same state. Thus the separability property applies to a fixed number of timesteps. Balancing the two properties is a task-specific problem. To increase the separability of a reservoir, a good heuristic to to make the network connections sparse and random. This will make the activation signals within the network decoupled and varied.

**Topology**

A larger number of neurons in the network will mean more activation signals, allowing for finer grain classification for example. Concerning the particular topology of the network, it has been noted that there is substantial variation in ESN performance among randomly created reservoirs. However, in a study of different topologies were tested, such as scale-free,small world or biologically inspired; but none tested to "perform significantly better than simple random networks". Still the difference in performance amongst these random networks indicates that similar approaches might be useful.

### Unsupervised reservoir adaptation

One simple test to measure the memory capacity of a reservoir is to implement the task of recreating the input as output, with a time given time delay. This tests that the reservoirs has a the desired memory capacity. Other tests including choosing reservoirs with minimal pair-wise activation of neurons,

# Chapter 3

# How the Reservoir Computing Paradigm is used

This survey will first present recent scientific papers that have used the reservoir computing paradigm in the fields of applied physics, machine learning theory, computational neuroscience and robotics. It will then transition to papers more closely related to my own field, natural language processing.

## 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. [8, 9] (and many more) 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 [10] 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 [11] 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. [12, 13] 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 [13] 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 [14]. 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.

## 3.2 Physics and 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.

(passage on how RC can be used to solve Multi-objective problems)

(passage on how the paper Constructing Optimized Binary Masks offers an architecture for reservoirs better suited to hardware implementation)

## 3.3 Natural Language Processing

## 3.4 Other Randomized Networks

It should be noted that the idea of treating the readout layer separately from the network can be applied to more traditional feed-forward networks. (passage on extreme learning machines)

# Chapter 4

# Conclusion and my own future work

# 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 hyper-hill 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, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, 1990.

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

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

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

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

[11]

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

[13] 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.

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