Master research Internship

Bibliographic report

Neural architecture for time management and its contextualization

Domain: Neural and Evolutionary Computing - Machine Learning

Author:
Anthony Strock

Supervisor:
Nicolas P. Rougier
Xavier Hinaut
Mnemosyne

Abstract: In this bibliography review, among all the recurrent neural networks, which naturally deal with time sequences, we are particularly interested in two distinct neural architectures: the Long Short-Term Memory networks (LSTM) and the Echo State Networks (ESN). On the one hand, the LSTM can handle long-term time dependencies, though its training is not biologically plausible. On the other hand, the ESN can handle short-term time dependencies, and, its training is way simpler, it can even be performed online. The aim of this internship is to see what are the properties of the LSTM that can be transferred to the ESN keeping in mind that the ultimate goal is to model how the brain is able to learn. In other words, the training performed must be biologically plausible in the end. In this bibliographic review, we provide a global view on the LSTM, on the ESN, and, on the extensions of both.

Contents

1 Introduction

Artificial neural networks are models that started to be developed in the past century. They are naturally inspired by the biological neural networks and mimic the way the brain is processing information. Recently, these models have shown their strength. With the Deep Learning, the performance in image recognition and in natural language processing has improved significantly [LeCun et al., 2015]. However, there is still progress to be made in order to deal with the temporal aspect of the data. In the following, as in the internship, we will focus on artificial neural networks which allow to learn from sequential data. They are called recurrent neural networks, and, unlike the feedforward neural networks, their connections contain cycles. In practice, the progress on these networks and on their understanding affects two different domains: Computational Neuroscience, where they are used to model the brain and its functions in order to understand its behaviour; and Machine Learning, where they are principally used in order to solve supervised machine learning problems over sequential data.

When humans make a decision, they do it within a temporal context. Moreover, they seem able to keep a track of the time dependencies between events. Their memory capacity being limited, their brain must be able to keep a useful representation of this context. A recurrent neural network can, through its dynamic, produce a representation of the temporal context. It is then natural to ask whether or not it can produce a representation that allows to keep a track of time dependencies.

Handling long-term time dependencies with these networks is recognized as a hard problem [Bengio et al., 1994]. However, the Long Short-Term Memory networks (LSTM), a special kind of deep neural networks, have been shown able to handle relatively long-term time dependencies [Hochreiter and Schmidhuber, 1997]. The activity, after a learning phase, of certain neurons in these networks shows that they are able to extract high level information on the context [Karpathy et al., 2015]. However, their learning phase relies on an unfolding of the networks through the time which is not biologically plausible, and, it can only be performed offline. In contrast to these deep neural networks, there is the Reservoir Computing (RC) and the Echo State Networks (ESN). Training these networks is much less computationally expensive [Jaeger, 2002] and can be performed online. Moreover, it has been shown that these networks can deal with short-term time dependencies [Jaeger, 2001a]. Furthermore, connections between the RC and the dynamic of the brain has been made [Dominey, 1995].

To summarize, whereas the LSTM are able to deal with long-term time dependencies, the ESN are more biologically plausible. The aim of this internship is to propose an approach that combines the best aspects of the LSTM and the ESN, keeping in mind that the goal is to model how the brain learns. The learning algorithm proposed must be biologically plausible.

In this bibliographic review, we start by some preliminaries where we formalize three different common supervised machine learning problems using sequential data, and where we provide the basics on Recurrent Neural Networks. Section 3 and 4 will draw respectively the state of the art concerning the LSTM and the ESN. The last section will consist of a discussion aiming at raising the question that must be taken into account during the internship and giving some beginning line of thought.

2 Preliminaries

2.1 Supervised machine learning problems over sequential data

Let E and F be two sets. Let f be a function from E to F and X be a part of E. X represents the dataset where the values of f are known. The supervised machine learning problems consist in finding back f only from this restriction of f over X.

Formally, we are given $\{(x, y) \in E \times F, x \in X, y = f(x)\}$ and for any $e \in E$ we want to be able to compute the value of $f(e)$.

In the case of sequential data, E is assumed to be a set of sequences which are finite in practice. If there is not any theoretical problems in assuming that the sequences we learn from are infinite, such data can't possibly be represented in a computer without loss. So, in practice, we have $E = \bigcup_{n \in \mathbb{N}} V^n$ where V is the set the sequences take values from.

Three different kinds of supervised machine learning problems over sequential data can be highlighted [Dietterich, 2002]. They are defined the same way than previously, but properties over F and f they insure may differ. Their specificity will be provided in the following, accompanied by concrete examples of application.

2.1.1 Sequential classification

In sequential classification problems we assume that F is a finite set. Loosely speaking, F represents a set of classes, and for each $e \in E$, $f(e) \in F$ represents the class of e.

Intuitively, solving sequential classification problems means being able to associate a class to each sequence. There are several domains where this problem appears. For instance, in voice recognition, you are given sounds recorded by a microphone, which is nothing more than a sequence which takes values in $\mathbb{R}$, and you want to determine whether or not someone is speaking, which word this person is saying. Or also in online handwritten character recognition. You are given a sequence of position over a touch pad, which can be represented as a sequence which takes values in $\mathbb{R}^2$, and you want to determine which character the person has drawn.

2.1.2 Sequential supervised learning

In sequential supervised learning problems we assume that F as the same form than E and that f preserves the length of the sequences. So, formally, $F = \bigcup_{n \in \mathbb{N}} T^n$ where T is a set and $\forall\ n \in \mathbb{N} f(U^n) \subset T^n$.

Intuitively, solving sequential supervised learning problems means being able to change the representation of the sequence you are provided with. You could decide that the value you associate to each value of your sequence means something about this value, but you could also imagine more complex things such as filtering, sorting and more. One clear application appears in Natural Language Processing: the Part of Speech tagging.

2.1.3 Prediction

In prediction problems we assume that $F = V$. As suggested the name of the problem, the goal is here to predict the next value of each sequence. A prediction problem could be instantiated in

a sequential classification problem by transforming the couples $((x_1, ..., x_n), x_{n+1})$ into the couples $((x_1, ..., x_n), (x_2, ..., x_{n+1}))$.

Prediction problems finds numerous application, notably in finance where they desire to predict how the stock market will evolve, or in meteorology where they desire to predict the weather.

2.2 Basics on recurrent neural networks

2.2.1 Formal definition

First of all, in order to deal with the recurrent aspect of the artificial neural networks, a notion of time must be taken into account. In order to give a formal definition we will consider that the time is discrete, the definitions could be extended to continuous time through differential equations. Let us denote T the space of time instants. In our case we suppose $T = \mathbb{N}$. Let us first define what is a neuron and what is an input of a neural network. Then, let us see how to combine them as a network.

A neuron n can be integrally defined by an activation function f_n. A neuron is provided with two quantities varying over time: one input and one output. The function f_n is nothing but the link between the input at time t and the output at time t. We will denote in the following these two quantities respectively as $x_n(t)$ and $y_n(t)$. In classical neurons the link between the input and the output can be written formally for all $t \in T$ as:

$$y_n(t) = f_n(x_n(t)) \tag{1}$$

But there are other kind of neurons. For instance, the leaky integrator neurons, which comes naturally by discretizing a differential equation, extends the classical neurons by adding a leaking rate α. The link between the input and the output becomes for all $t \in T$:

$$y_n(t) = (1 - \alpha)y_n(t - 1) + \alpha f_n(x_n(t)) \tag{2}$$

An input i of a recurrent neural network net is a function $s \in \mathbb{R}^T$. Its output at time t is simply defined as $s(t)$.

A recurrent neural network net can be defined by a number of inputs k, a set of neurons N, a directed graph $G = (V, E)$ such that $V = N \biguplus [\![1, k]\!]$, a function $w \in \mathbb{R}^E$ and O a subset of V. E represents the directed connections between each neuron or input of the network. w associates a weight to each connections. O represents the neurons that will be chosen as output of the neural network. G and O conjointly form the structure of the network.

The output y_{net} of the recurrent neural network net is defined as the function that associates to t, the tuple of outputs at time t of the element in O.

$$\forall t \in T \; y_{net}(t) = (y_o(t))_{o \in O} \tag{3}$$

The dynamics of the neural network *net* is defined given a function $init \in \mathbb{R}^N$ and k inputs $(i_j)_{j \in [\![1;k]\!]}$, where *init* represents the initial states of the neurons in the network. It is controlled via the following equations for all $n \in N$ and for all $t \in T$:

$$y_n(0) = init(n) \tag{4}$$

$$x_n(t) = \sum_{n' \in N, (n',n) \in E} w((n',n)) * y_{n'}(t-1) \; + \sum_{j \in [\![1;k]\!], (j,n) \in E} w((j,n)) * i_j(t) \tag{5}$$

2.2.2 Properties

First of all, a recurrent neural network is a model which can naturally encode a function from the sequences which takes values in $\mathbb{R}^k$ to the sequences which takes values in $\mathbb{R}^{|O|}$. So such models can be naturally used as hypothesis space for the sequential supervised learning problem. However, they can also be used to solve the prediction problem and other supervised machine learning problems over sequential data.

Moreover, several theoretical studies have shown that a lot of things can be modelled with these networks. For instance, under some assumptions, such networks can be used to approximate any dynamical systems [Funahashi and Nakamura, 1993]. They are also able to simulate any Turing machines [Siegelmann and Sontag, 1992]. Furthermore it has even been proved that their non-deterministic version is strictly more expressive than a non-deterministic Turing machine [Cabessa and Villa, 2012].

2.2.3 Learning

In the following we will present learning algorithm that will aim to solve the sequential supervised learning problems. These algorithms can be adapted to solve prediction tasks through the transformation provided earlier or others. In order to learn a suited recurrent neural network these different algorithms will start by fixing what is the structure of the neural network and find the most appropriate weighting through the optimisation of a cost function. More details on their usage can be found in [Jaeger, 2002].

BackPropagation Through Time (BPTT): BackPropagation Through Time (BPTT) is a gradient descent based algorithm proposed in [Werbos, 1990]. It is based on the transformation of the recurrent neural network into a feed-forward neural network through the duplication through the time of its neuron units and on the use of the classical backpropagation algorithm [Rumelhart et al., 1995] on this unfolded network. Loosely speaking, in the BPTT algorithm we estimate the partial derivatives of a cost function with respect to the weights by backward computations in time and through the network.

First of all it can be highlighted that BPTT need to use the entire sequences in order to learn which can be a problem when the goal is to learn online. Truncated version of this algorithm aims to overcome this problem. However, BPTT has some other bad properties such as the one highlighted in [Bengio et al., 1994] where they have shown that convergence and stability against noise can't be both satisfied through this algorithm. Convergence to a local minima cannot even be guaranteed. Another problem comes from the backpropagation itself, the gradient either explode

or vanish through the backward propagation in time. So, long-term dependencies cannot be learn with this algorithm on a classical recurrent neural network.

Real-Time Recurrent Learning (RTRL): Real-Time Recurrent Learning (RTRL) is more straightforward than BPTT. It is also a gradient descent based algorithm, but here no transformation is performed before the computation of the gradient. Loosely speaking again, it approximates the partial derivatives of a cost function with respect to the weights forward in time. However, the problem highlighted in [Bengio et al., 1994] is also valid for RTRL. Moreover RTRL is more computationally costly per time step than BPTT though its computation is local in time, which means that the memory required by this algorithm doesn't depend on the length of the inputs.

Other approaches: There were several other attempts of approach between BPPT and RTRL such as the one in [Robinson and Fallside, 1987] but all of them shared the same bad properties described in [Bengio et al., 1994] that were inherent to the use of a gradient-based scheme. Another completely different approach based on the Extended Kalman Filtering (EKF) emerged later on which gives arguably similar results.

3 Long Short-Term Memory (LSTM)

3.1 Intuition toward LSTM

Intuitively, as each neuron in a recurrent neural network can be provided with a feedback connection, it is natural to imagine that these networks can be able to store information about their past inputs. The activation state of each neuron would play the role of a memory, and this memory would be updated through the dynamic of the network. The more recent these inputs are, the easier it seems to be remembering them. In other words, in such networks, a short-term memory ability seems to be acquired for free.

Although there is no clear practical advantage over using feed-forward neural networks with sliding windows of fixed size, experience shows that this intuition seems to be correct. But, as demonstrated in [Bengio et al., 1994], this short-term memory ability cannot be extended naturally to a long-term memory ability through the same learning algorithms. The convergence and the stability against noise can't be both satisfied by a gradient descent-based learning algorithm.

In [Hochreiter and Schmidhuber, 1997], by pointing out where lies the problem of convergence, they highlight that the Constant Error Caroussel (CEC, a recurrent neural network with linear activation function and feedback weights equals to one) should not suffer from this problem of convergence. Using a CEC insures the gradient to be constant against time, so that it neither vanish nor explode as it is very often the case with others recurrent neural networks. The convergence not being a problem, only stays the stability against noise.

On the basis of this finding, they raise two problems that might cause instability in these networks: the memory of a neuron can be polluted by irrelevant inputs; an irrelevant memory will provide an irrelevant output.

In the end, they propose a novel architecture constructed around the CEC in order to overcome these problems: the Long Short-Term Memory networks (LSTM). The core idea in their construction is the ability to regulate both what should be memorized from the inputs and what should be shared as output from what has been memorized. In order to do so, they do not consider neurons as the

units in the networks, but, memory cells consisting of neurons and (input or output) gates. These gates will provide the desired mechanisms on the memory.

3.2 LSTM's dynamics

As for recurrent neural networks, let us describe first the behaviour of one unit in these networks and then how they are structured as a network.

3.2.1 A memory cell

Let us consider a memory cell c. It is composed of one input gate in and one output gate out which are classical neurons with an activation function f_{in} and an activation function f_{out}. We have for all $t \in T$:

$$y_{out}(t) = f_{out}(x_{out}(t)) \tag{6}$$

$$y_{in}(t) = f_{in}(x_{in}(t)) \tag{7}$$

As an extension of the previous notations, the input and the output of the memory cell c at time t will be denoted respectively as $x_c(t)$ and $y_c(t)$. Now, the output of a cell is dissociated from what the memory cell has memorized. $s_c(t)$ will represent the inner memory of a cell.

The neuron out will act as a filter from the memory of c to the output of c. So, for all $t \in T$:

$$y_c(t) = y_{out}(t)h(s_c(t)) \tag{8}$$

where h is a differentiable function which scale the memory of the cell c before trying to output it.

While the neuron in will act as a filter from the input of c to the memory of c. So, for all $t \in T$:

$$s_c(t) = s_c(t-1) + y_{in}(t)g(x_c(t)) \tag{9}$$

where g is a differentiable function which scale the input of the cell c before trying to memorize it.

3.2.2 A network of memory cells

This time one other thing must be taken in consideration in the network model. We could imagine that some memory cells share the same input gates and output gates. A memory block B of size S is then defined as a set of S memory cells which share the same input and output gates. In order to simplify the notation we will consider two functions $gate_{in}$ and $gate_{out}$ which associate to a memory cell respectively its input and output gate.

Similarly to the recurrent neural networks, an LSTM can be defined by a number of inputs k, a set of memory cells C, a directed graph $G = (V, E)$ such that $V = C \biguplus [\![1, k]\!]$, three functions $w, w_{in}, w_{out} \in \mathbb{R}^E$ and O a subset of V. E represents this time the directed connections between each memory cell or input of the network. w, w_{in} and w_{out} associate a weight to each connection. w_{in} and w_{out} represent respectively the weight given to the input and the output gates. O represents the memory cells that will be chosen as output of the neural network. G and O conjointly form the structure of the network. The inputs and the outputs of a LSTM are defined in the same way.

To take into account the memory blocks, w_{in} and w_{out} satisfies the following properties for all $c, c' \in C$:

$$gate_{in}(c) = gate_{in}(c') \Rightarrow \forall c'' w_{in}(c'', c) = w_{in}(c'', c') \tag{10}$$

$$gate_{out}(c) = gate_{out}(c') \Rightarrow \forall c'' w_{out}(c'', c) = w_{out}(c'', c') \tag{11}$$

The dynamics of the LSTM is defined this time given three function $init \in \mathbb{R}^C$, $init_{in} \in \mathbb{R}^{gate_{in}(C)}$ and $init_{out} \in \mathbb{R}^{gate_{out}(C)}$, and k inputs $(i_j)_{j \in [\![1;k]\!]}$. $init$ represents the initial memory states of the memory cells, $init_{in}$ and $init_{out}$ represents respectively the initial states of the input and the output gates. The dynamic is controlled then via the following equations for all $c \in C$ and for all $t \in T$:

$$s_c(0) = init(c) \tag{12}$$

$$s_{gate_{in}(c)}(0) = init_{in}(gate_{in}(c)) \tag{13}$$

$$s_{gate_{out}(c)}(0) = init_{out}(gate_{out}(c)) \tag{14}$$

$$x_c(t) = \sum_{c' \in C, (c',c) \in E} w((c', c)) * y_{c'}(t-1) \; + \sum_{j \in [\![1;k]\!], (j,c) \in E} w((j, c)) * i_j(t) \tag{15}$$

$$x_{gate_{in}(c)}(t) = \sum_{c' \in C, (c',c) \in E} w_{in}((c', c)) * y_{c'}(t-1) \; + \sum_{j \in [\![1;k]\!], (j,c) \in E} w_{in}((j, c)) * i_j(t) \tag{16}$$

$$x_{gate_{out}(c)}(t) = \sum_{c' \in C, (c',c) \in E} w_{out}((c', c)) * y_{c'}(t-1) \; + \sum_{j \in [\![1;k]\!], (j,c) \in E} w_{out}((j, c)) * i_j(t) \tag{17}$$

3.3 LSTM's training

As there are multiplicative units (the gates), the most popular BPTT cannot be used as is. For this reason, [Hochreiter and Schmidhuber, 1997] decided to perform a slight variant of RTRL instead. The final complexity of this variant is equivalent to the one of BPTT. This variant keeps the good property of RTRL to be local in time. Moreover, it is also local in space unlike BPTT, which means that at each time step the computation of the partial derivative of the cost with respect to a weight doesn't depend on the size of the network. Furthermore, thanks to truncations in the computation, this algorithm insures to keep a constant gradient. Having this constant gradient will insure convergence and the gates will participate in fighting against the noise.

3.4 LSTM's memory abilities

Already in [Hochreiter and Schmidhuber, 1997] LSTM has shown to be able to handle quite long-term dependencies in noisy contexts. Various other works have shown their memory ability in a lot of different contexts. In [Karpathy et al., 2015], they started to give some clues about why LSTM work that well. They have shown that through their memory cells, these networks are able to keep track of meaningful information about the context. For instance, after training their LSTM on the Linux Kernel they saw that some cells were specialized in keeping track of the length of a line, others specialized in detecting the commentaries, others specialized in keeping track of the block depth we are actually in and many others. However, as shown in [Gers et al., 2001], the LSTM has not been well suited to predict the behaviour of chaotic time series.

3.5 Variants and extensions

Several variants of LSTM has been proposed in the literature, and, as shown in [Greff et al., 2015], there is no clear winner between them. One of the first variant was to introduce another gate, the forget gate [Gers et al., 2000]. The purpose of this gate was to give the possibility to a cell to clear its memory. In practice it is done by changing $s_c(t-1)$ in $y_{forget}(t)s_c(t-1)$ in equation 9. Some of other variants consisted in adding information in the inputs of the cells and of the gates. For instance, [Gers and Schmidhuber, 2000] introduced peep-hole connections, where a component linked to the state of the memory of the cells is added to the inputs of the cells and of their gates. Others considered that input gate already carried out upstream the work of the output gate. Intuitively, if the memory isn't contaminated by irrelevant inputs there is no need to filter out the memory before sharing it. [Cho et al., 2014] introduced the Gated Recurrent Unit where the output of a cell was its memory state itself and where there were forget gates which were entirely defined with the input gates. There are also more recent works which try to give a special structure to the LSTM. As for instance in [Kalchbrenner et al., 2015] with their Grid LSTM. LSTM introduced an idea of memory cells, others goes even further and uses real memory itself. A notion of memory neural networks has recently emerged with the MemNN [Weston et al., 2014, Sukhbaatar et al., 2015] and with the Deep Turing Machines [Graves et al., 2014]

4 Echo State Networks (ESN)

4.1 Reservoir Computing (RC) principle

As seen previously, training a recurrent neural network can be an hard task. It often comes to minimizing a cost function through gradient-based techniques. Moreover, when the goal is to be able to handle long-term dependencies, the problem becomes even harder [Bengio et al., 1994]. The LSTM networks are a kind of neural network designed in order to handle such dependencies. However, their training is not biologically plausible because, as several other recurrent neural networks, they use a training which unfold the network through the time. The Echo State Networks (ESN) are a special kind of recurrent neural network, introduced by [Jaeger, 2001a], where the training is particularly easy to perform. In such networks, an unfolding of the network is not necessary. They emerged at the same time than Liquid State Machines (LSM), introduced by [Maass et al., 2002], which are based on the same principle. This principle is now known as the Reservoir Computing (RC) principle [Verstraeten et al., 2007].

The main feature of the Reservoir Computing (RC) paradigm is the way it simplifies tremendously the training. The following intuitions are inspired by [LukošEvičIus and Jaeger, 2009] which draws a global review of the different aspect of the RC paradigm of the last decade.

In the RC paradigm, we start by fixing a recurrent neural network, called the reservoir, which will create a dynamic. The main idea is that this reservoir will be maintained constant through the training. The training will only consist in learning the connections from this reservoir to some other output neuron units. Experience shows that even by taking randomly generated reservoirs, the dynamic created by the reservoir has an inherent ability to create a context that we can learn from. For instance, [Jaeger and Haas, 2004] shows great improvement in chaotic systems prediction.

Several connections have been made between the RC paradigm and the dynamic of the brain [Dominey, 1995, Hinaut and Dominey, 2013]. For instance, the ability to learn through the uncontrolled dynamic of a reservoir could explain how the brain can perform accurate computations

though its physical components are noisy and imprecise. Moreover, in computational neuroscience, these approaches aim to model generic part in the brain[Rigotti et al., 2013].

4.2 ESN's dynamics

As in the reservoir computing principle, an Echo State Networks (ESN) is composed of two parts. The first one, also known as the reservoir, is a classical recurrent neural network where each activation functions are sigmoids, typically tanh. The second one consists in a new set of neurons. Their outputs will be the outputs of the whole network.

Formally, an ESN can be defined by a number of inputs k, two disjoint set of neurons N_R and N_{out}, a directed graph $G = (V, E)$ such that $V = N_R \biguplus [\![1, k]\!]$, a function $w_R \in \mathbb{R}^E$ and a function $w_{out} \in \mathbb{R}^{V \times N_{out}}$. N_R represents the neurons of the reservoir, whereas N_{out} represents the new output of the network. E represents the directed connections between each neuron of the reservoir or input of the network. w_R associates a weight to each of these connections. G and w_R conjointly form the reservoir of the ESN.

Similarly than in a classical recurrent neural network, the dynamics is defined given a function $init \in \mathbb{R}_R^N$ and k inputs $(i_j)_{j \in [\![1;k]\!]}$, where $init$ represents the initial states of the neurons in the network. The update equations are obtained by considering that the only connection added to the reservoir in the ESN are the connections from V to O. Loosely speaking, the reservoir acts as a hidden layer in a classical feed-forward scheme. To keep the same notations, we will denote respectively as $x_n(t)$ and $y_n(t)$ the input and the output at time t of the neuron n. The dynamics is controlled via the following equations for all $n_R \in N_R$, for all $n_{out} \in N_{out}$ and for all $t \in T$:

$$y_{n_R}(0) = init(n_R) \tag{18}$$

$$y_{n_R}(t) = f_{n_R}(x_{n_R}(t)) \tag{19}$$

$$y_{n_{out}}(t) = f_{n_{out}}(x_{n_{out}}(t)) \tag{20}$$

$$x_{n_R}(t) = \sum_{n' \in N_R, (n', n_R) \in E} w_R((n', n_R)) * y_{n'}(t-1) \; + \sum_{j \in [\![1;k]\!], (j, n_R) \in E} w_R((j, n_R)) * y_j(t) \tag{21}$$

$$x_{n_{out}}(t) = \sum_{n' \in N_R} w_{out}((n', n_{out})) * y_{n'}(t) \; + \sum_{j \in [\![1;k]\!]} w_{out}((j, n_{out})) * y_j(t) \tag{22}$$

These equations are given for classical neurons but could be extended naturally to leaky integrator neurons. ESN with these neurons are naturally called ESN with leaky integrator neurons.

4.3 ESN's training

There are two steps in the training of ESN. The first one consists in choosing a reservoir and the second one in learning from this reservoir the best suited output weights. The key idea is that the reservoir chosen in the first step, and more specifically the weights w_R of its connections, will be maintained constant in the second step. There are not yet a lot of theoretical results on what can be learned from a given ESN reservoir. As mentioned in [Jaeger et al., 2007], for Liquid State

Machines (LSM) reservoirs, exact results on what can be approximated are present in the literature. In [Jaeger, 2001a] they give some preliminary results by defining the memory capacity of a reservoir, the principal results being that this memory capacity is bounded by the number of neurons in the network.

4.3.1 Learning the output weights

Once the reservoir is fixed, finding the most appropriated weights from the outputs of the reservoir to the new output of the network can be rephrased as a linear regression problem. That could be obtained by transforming, for all output neurons n and for all $t \in T$, $y_n(t)$ into $y'_n(t) = f_n^{-1}(y_n(t))$. For the sake of simplicity, let us suppose that the activation function of the output neurons are linear and that the network aims to reproduce a given sequence. The problem becomes then:

$$W_{out} Y_R = Y_{target} \tag{23}$$

where W_{out} are the weights we want to learn, $Y_{target} \in M_{|N_{out}|,L}(\mathbb{R})$ is the targeted sequence, $Y_R \in M_{|N_R|,L}(\mathbb{R})$ are the outputs of the neurons of the reservoir.

There are two different philosophies in the resolution of this linear regression problem. The first one considers having access to the inputs of the reservoir and the desired outputs at all time, while the second one considers receiving the inputs and the desired outputs one at a time. The first one requires classical linear regression techniques while the second one could be seen as a way to update previously computed weights to fit better a new observation. Let us denote techniques from the first philosophy as offline linear regressions and the ones from the second one as online linear regressions. Online linear regression techniques are more biologically plausible than offline ones because they do not require an explicit access to the outputs of a reservoir for all the time instant in the past but only for the actual time instant.

In order to perform an offline linear regression, several linear regression techniques could be used. In [Lukoševičius, 2012], we can find recipes in order to have a more successful reservoir computing approach. They state that the most generally recommended linear regression technique in that case is the ridge regression:

$$W_{out} = Y_{target} \left(Y_R\right)^T \left(Y_R \left(Y_R\right)^T + \beta^2 I\right)^{-1} \tag{24}$$

where β is a regularization factor, I is the identity matrix and $(.)^T$ is the transpose operator.

In [LukošEvičIus and Jaeger, 2009], they relate that the principal technique used to perform an online linear regression is based on a stochastic gradient descent on a cost function. This cost function could be for instance a classical Least Mean Squares or a Recursive Least Squares. The second one has better convergence properties but is more computationally costly. There are also other proposal in the literature as for instance the BackPropagation-DeCorrelation[Steil, 2004].

4.3.2 Choosing a reservoir

There are mainly three different approaches in the way the reservoirs are chosen. The first approach and the most naive one consists in taking one randomly satisfying some properties. The second one consists in adapting the reservoir to the inputs it will receive. The last one consists in adapting the reservoir to solve a particular task, in this approach the reservoir is optimized for a given set of inputs and their associated desired outputs. The first approach is called generic, the second unsupervised pre-training and the last one supervised pre-training. We will focus on the most used

one is: the generic one [Lukoševičius, 2012]. More details about the other approaches can be found in [LukošEvičIus and Jaeger, 2009].

Even if we decide to pick a reservoir randomly there are several parameters that must be taken into account. In [Lukoševičius, 2012], we can find an exhaustive list of these parameters. They also give intuition about what are their influence on the learning. For classical ESN there are mainly four things to take into account in the sampling:

- **The size of the reservoir:** Generally, the more neurons there are in the reservoir, the easier it is to learn from it. There must be at least as much neurons as values that are expected to be stored. However, too many neurons can lead to overfitting. In that case, a regularization method should be used.
- **The sparsity of the reservoir:** It is recommended to use reservoirs with few connections. Each neuron must be connected to few other neurons in the reservoir. Moreover, the less connections there is in the reservoir, the more computationally efficient it is to learn from it. Few connections corresponds to few non zero element in the weight matrix.
- **The spectral radius:** It is computed as the maximal eigenvalue of the weight matrix of the connection of the neurons to the others in the reservoir. Intuitively, it is linked to the time an input can stay present in the memory of a reservoir. The bigger it is, the longer it can stay in the memory. But, if it is too high the reservoir becomes very sensitive to its initialization and behave chaotically. For a spectral radius smaller than one there is not, in general, this sensitivity problem. This sensitivity is formalized in [Jaeger, 2001b] with the Echo State Property. They show that under some assumptions having the Echo State Property implies that the spectral radius must be smaller than one.
- **The input scalings:** It affects the way each neuron in the reservoir will be sensitive to the inputs. It is generally done by uniformly sampling a dense matrix.

And if we decide to use leaky integrator neurons, another crucial parameter arises: the leaking rate. It intuitively represents the speed of evolution of the dynamic of the network. [Lukoševičius, 2012] states that it must be chosen adapted to the considered inputs and the desired outputs.

4.3.3 ESN's memory abilities

In [Jaeger, 2001a], they were already providing results on the ability of an ESN reservoir to keep information about its past inputs. Later on in [Jaeger and Haas, 2004], they will show that learning an ESN in the RC paradigm provided good results for chaotic time-series prediction. In [Jaeger, 2012], they show that by choosing smartly the reservoir, the ESN could have a notably long short-term memory. They made this proof of concept on the task of giving further in time the first inputs given to the network. Other works has shown their strength in the natural language processing tasks [Hinaut and Dominey, 2013, Hinaut et al., 2015].

4.4 Variants and extensions

4.4.1 Variants on the reservoir

At the same time than ESN was proposed by [Jaeger, 2001a], Liquid State Machine (LSM) was proposed by [Maass et al., 2002]. The LSM reservoirs are a special kind of spiking neural networks.

These special kind of networks are closer from how the brain works. In these networks, the time is considered continuous and a neuron doesn't provide an output at each step of time, but whenever a potential related to the charge of the neuron reach a certain threshold. Other kinds of reservoirs have been proposed since then. For instance the evolino reservoirs which consist in taking an LSTM as a reservoir.

4.4.2 Towards the handling of long-term time dependencies

Even if [Jaeger, 2012] has shown that ESN can handle long-term time dependencies through a smart choice on the parameters of the reservoir considered. Finding these parameters is not an easy task and even if they are found, it is not completely satisfactory in order to model how the brain learns. [Pascanu and Jaeger, 2011] tries to acquire the ability to handle long-term time dependencies through adding external memory units on a randomly sampled ESN reservoir. For this purpose they propose a model which extends the ESN. In this model, there is one more component, neurons which have a feedback connection to themselves, connections from the reservoir outputs to them and connections from them to the reservoir. All the weights on these new connections are trainable. Intuitively, these new neurons will act as a memory for the reservoir. They show that this model was suited to learn even with noisy and highly variable input signal.

5 Discussion

To summarize this bibliographic review, we saw two different kinds of neural architectures. First, we saw the Long Short-Term Memory (LSTM) networks, a special kind of recurrent neural networks, which are able to keep a meaningful representation of temporal contexts containing relatively long-term time dependencies. However, in order to perform that well, they require a training which implies an unfolding of the network through the time. Such kind of training is obviously not biologically plausible and the brain cannot be modelled in such a way. After, we saw the Echo State Networks (ESN), which, through the Reservoir Computing principle, simplify tremendously the training of the network. In this paradigm, the training consists only in a linear regression which can even be performed online. Moreover, such networks have been able to deal with short-term time dependencies, and, with a good choice on the reservoir, relatively long-term time dependencies.

For the sake of simplicity, to keep the possibility to be biologically realistic, and, to have the best performance, a question arises naturally: Could the functionality of the LSTM networks be transposed into the simplicity of the Reservoir Computing principle? The main goal of this internship will be to try to give an answer to this question. Obviously, there is no guarantee that this can be done simply or that it is even possible. But, by identifying what are the properties of the LSTM networks that makes them able to handle long-term time dependencies, we could expect to be able to find an extended version of the ESN having these properties. One of the main components of the LSTM are the gates, they allow to block the activity of the cell and to keep something in memory for a long time. We could start by thinking about how we could add some of these gates on an ESN. For instance, earlier works on ESN have already tried to give an external memory to the reservoir via additional neuron units communicating with the reservoir. We could try to make this memory stable by using memory cell instead of neurons. But, the training becomes harder than only a linear regression. Would it still be possible to perform the training online? Another feature of more recent versions of LSTM networks is to give the ability to the memory to reset itself. We could imagine

that there is a moment when the reservoir internal state starts behaving improperly. We may would like to be able to reset some part of the reservoir. Could it be done with other external neuron units? One question must be raised for each of these attempts to give the ESN an LSTM networks property: Is the training biologically plausible?

References

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Cabessa and Villa, 2012] Cabessa, J. and Villa, A. E. (2012). The expressive power of analog recurrent neural networks on infinite input streams. *Theoretical Computer Science*, 436:23–34.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Dietterich, 2002] Dietterich, T. G. (2002). Machine learning for sequential data: A review. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 15–30. Springer.

[Dominey, 1995] Dominey, P. F. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274.

[Funahashi and Nakamura, 1993] Funahashi, K.-i. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806.

[Gers et al., 2001] Gers, F. A., Eck, D., and Schmidhuber, J. (2001). Applying lstm to time series predictable through time-window approaches. In *International Conference on Artificial Neural Networks*, pages 669–676. Springer.

[Gers and Schmidhuber, 2000] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[Graves et al., 2014] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Greff et al., 2015] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*.

[Hinaut and Dominey, 2013] Hinaut, X. and Dominey, P. F. (2013). Real-time parallel processing of grammatical structure in the fronto-striatal system: a recurrent network simulation study using reservoir computing. *PloS one*, 8(2):e52946.

[Hinaut et al., 2015] Hinaut, X., Petit, M., Pointeau, G., and Dominey, P. F. (2015). Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks. *Towards embodied artificial cognition: TIME is on my side*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Jaeger, 2001a] Jaeger, H. (2001a). *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik.

[Jaeger, 2001b] Jaeger, H. (2001b). The "echo 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.

[Jaeger, 2002] Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik.

[Jaeger, 2012] Jaeger, H. (2012). Long short-term memory in echo state networks: Details of a simulation study. Technical report, Jacobs University Bremen.

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

[Jaeger et al., 2007] Jaeger, H., Lukoševičius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352.

[Kalchbrenner et al., 2015] Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

[Karpathy et al., 2015] Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Lukoševičius, 2012] Lukoševičius, M. (2012). A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer.

[LukošEvičIus and Jaeger, 2009] LukošEvičIus, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

[Maass et al., 2002] Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

[Pascanu and Jaeger, 2011] Pascanu, R. and Jaeger, H. (2011). A neurodynamical model for working memory. *Neural networks*, 24(2):199–207.

[Rigotti et al., 2013] Rigotti, M., Barak, O., Warden, M. R., Wang, X.-J., Daw, N. D., Miller, E. K., and Fusi, S. (2013). The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590.

[Robinson and Fallside, 1987] Robinson, A. and Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering.

[Rumelhart et al., 1995] Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, Architectures and Applications*, pages 1–34.

[Siegelmann and Sontag, 1992] Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational theory*, pages 440–449. ACM.

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

[Sukhbaatar et al., 2015] Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

[Verstraeten et al., 2007] Verstraeten, D., Schrauwen, B., d'Haene, M., and Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural networks*, 20(3):391–403.

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

[Weston et al., 2014] Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.