Recurrent neural network weight estimation though backward tuning

June 5, 2017

Abstract

We consider another formulation of weight estimation in recurrent networks, proposing a notation for a large amount of recurrent network units that helps formulating the estimation problem. Reusing a "good old" control-theory principle, improved here using a continuation heuristic, we obtain a numerically stable and rather efficient second-order and distributed estimation, without any meta-parameter to adjust. The relation with existing technique is discussed at each step. The proposed method is validated using reverse engineering tasks and non-trivial numerical computations.

Introduction

Artificial neural networks can be considered as discrete-time dynamical systems, performing input-output computation, at the higher level of generality [37]. However, only specific feed-forward or recurrent architectures are considered in practice, because of weight estimation, as reviewed now.

In the artificial neural network literature, feed-forward networks parameter learning is a well-solved problem. For instance, the back-propagation algorithms, based on specific architectures of multi-layer feed-forward networks, allows one to propose well-defined implementation [1], though it has been shown at the theoretical and empirical levels, that "shallow" architectures are inefficient for representing complex functions [6].

Deep-networks are specific feed-forward architectures [6] which can have very impressive performances, e.g. [19]. The key idea [27] is that, at least for threshold units with positive weights, reducing the number of layers induces an exponential complexity increase for the same input/output function. On the reverse, it is a reasonable assumption, numerically verified, that increasing the number of layers yields a compact representation of input/output functions. One drawback is related to weights supervised learning in deeper layers, since readout layers may over-fit the learning set, the remedy being to apply unsupervised learning on deeper layers (see [4] for an introduction). This problem desapears with specific architectures such as CNN.

It also remains restrictive by the fact that the architecture is mainly a pipeline, including some parallel tracks, while each layer is a feed-forward network (e.g. a convolutional neural layers) or with a very specific recurrent connectivity (e.g., restrained Boltzman machines). In the brain, more general architectures occur (e.g. with shortcuts between deeper and lower layers, as it happens in the brain regarding the pulvinar [36]) and each layer is a more general recurrent network (e.g., with short and long range horizontal connections). Breaking this pipe-line architecture may overcome the problem of deeper layer weight adjustment.

Feed-forward networks are obviously far from the computational capacity of recurrent networks. Therefore, specific multi-layer architectures without recurrent links within a layer and specific forward/backward connections between layers have been proposed instead. The first dynamic neural model, the model by Hopfield [26], appeared much later, and was very specific. Further solutions include Jordan's network [29], Elman's Networks [16], Long short term memory (LSTM) by Hochreiter and Schmidhuber [24].

Another track is to consider recurrent networks without supervised weight adjustment [41]. Units in such architectures are linear or sigmoid artificial neurons, including soft-max units, or even spiking neurons. Such network architectures, such as Echo State Networks [28] and Liquid Sate Machines [31], are called "reservoir computing" (see [41] for unification of reservoir computing methods at the experimental level).

In such architectures the recurrent parameters of hidden units is not explicitly learned, whereas recurrent weights are either randomly fixed, likely using a sparse connectivity, or adjusted using unsupervised learning mechanism, without any direct connection with the learning samples (though the hidden unit statistics, for instance, is sometimes adjusted in relation with the desired output). In the case of temporal mechanisms, i.e. using spiking neurons (e.g. in the model of [33]), the unsupervised learning mechanism of the recurrent weights is a form of synaptic plasticity, usually STDP (Spike-Time-Dependent Plasticity), a temporal Hebbian unsupervised learning rule, biologically inspired. It appears that simple methods yield good results [41], but without over-passing recent deep-layer architecture performances [15].

The general problem of learning recurrent neural networks has also been widely addressed as reviewed in [14] for 90's studies and in [32] for recent advances, and methods exist far beyond basic methods such as backpropagation through time.

In the present paper, we revisit the general problem of recurrent network weight learning, not as it, but because it is related to modern issues related to both artificial networks and brain function modeling. Such issues include: Could we adjust the recurrent weights in a reservoir computing architecture ? Is it possible to consider deep-learning architecture, with more general inter and intra layers connectivity ? Would it be possible to not only use some specific recurrent architecture as exemplified here, but to learn also the architecture itself (i.e. learn the weight value and learn if the connection weight has to be set to zero, cutting the connection) ?

We are not going to address more than weight adjustment in this paper. For instance learning issues (e.g., boosting [20]) are not within the scope of this paper: Neither representation learning [5], nor other complex issues [22] are considered, this contribution being only an alternate tool for variational weight optimization. See [17] for a recent discussion on such issues.

We are also not going to consider biological plausibility in the sense of [7], but will show that the proposed mehod is compliant with several distributed biological constraints or computational properties: local weight adjustment, backward error propagation, Hebbian like adjustment rules. A more rigourous discussion about the link with computational neuroscience aspects is however beyond the scope of this work.

In the next section we choose a notation to state the estimation problem, and Appendix A make explicit how this notation applies to most of the usual frameworks. We then adress the estimation problem and introduce the modified solution we propose, while Appendix B further discuss how it can be used for several estimation problems. In the subsequent section the method is implemented and numerically evaluated, while Appendix C expalined why certain estimation problem ar not considered here because reducing to trivial computation problems, given an architecture.

Problem position

A general recurrent architecture.

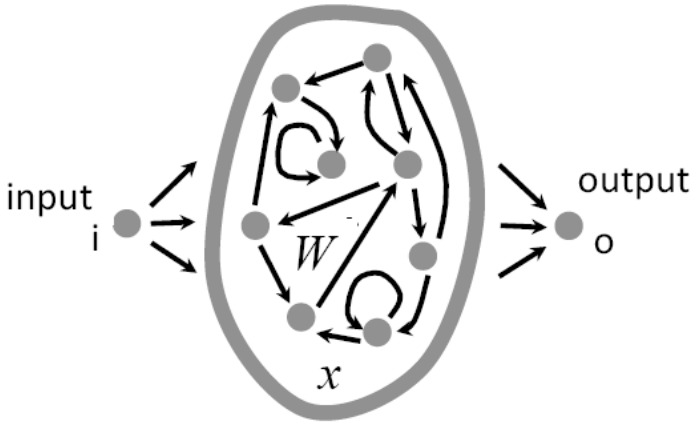

Figure 1: A General recurrent architecture maps a vectorial input sequence $\mathbf{i}(t)$ onto an output $\mathbf{o}(t)$, via an internal state $\mathbf{x}(t)$ of hidden units. It is parameterized by a recurrent weight matrix $\mathbf{W}$. The dynamics is defined by the network recurrent equations.

As schematized in figure 1, we consider a recurrent network with nodes of

the form:

$$\begin{array}{rcl} x_n(t) & = & \Phi_{n0t}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right) \\ & + & \sum_{d=1}^{D_n} W_{nd}\, \Phi_{ndt}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right) \\ o_n(t) & = & x_n(t), n < N_0 \end{array} \quad (1)$$

with:
- N nodes of value $x_n(t)$ indexed by $n \in \{0, N\{$,
  - with a maximal state recurrent causal range of R and with either,
  - $t-R \leq t' < t$ (i.e., taking into account previous value up to R time-steps in the past) or
  - $t' = t$ and $n < n'$ (i.e., taking into account present value, of subsequent nodes, in a causal way).
  - Here $N_0 \leq N$ of these nodes are output;
- M input $i_m(s)$ indexed by $m \in \{0, M\{$, $t - S \leq s < t$,
- $1+D_n$ predefined kernels $\Phi_{ndt}\,()$ for each node, defining the network structure;
- $\sum_n D_n$ static adjustable weights W_{nd}, defining the network parameter.

Considering equation (1) we notice that :

- The distinction between output or hidden node is simply based on the fact that we can (or not) observe the $o_n(t)$ node value. By convention and without loss of generality, output nodes are the $N_0 \leq N$ first ones.
- Though, in order to keep compact notations, we mixed node with either
  - *unit firmware* parameter-less function, i.e. with $\Phi_{n0t}()$, or
  - *unit learnware* linear combination of elementary kernels, i.e. with $\sum_d W_{nd}\, \Phi_{ndt}()$,

  in all examples these two kinds of node will be separated. This constraint is not mandatory, but will help clarifying the role of each node.
- A given state value depends either on previous time values ($t - R \leq t' < t$) or subsequent indexed nodes ($t' = t$ and $n < n'$), yielding a causal dependency in each case.
- By design choice, as made explicit in the sequel in all examples, $0 \leq \frac{\partial \Phi_{ndt}()}{\partial x_{n'}(t')} \leq 1$ (non-decreasing contractive non-linearity), is verified. This constraint is not mandatory, but will help at the numerical conditioning level.
- We further assume, just for the sake of simplicity, that initial conditions are equal to zero, i.e., $\mathbf{x}(t) = 0, t < 0$ and $\mathbf{i}(s) = 0, s < 0$.
- We also assume that the dynamic is regular enough[1] for weight estimation to be numerically stable.

[1]Here, we assume that input and output are bounded, while the system is regular enough for the subsequent estimation to be numerically stable. Chaotic behaviors likely require very different numerical methods (taking explicitly the exponential dependency on previous value variations into account) [9]. In practice, not only contracting systems can be considered, as

The key point here, is that we have introduced intermediate internal state variables in order the weight estimation to be a simple linear problem as a function of these additional variables (and at the cost of higher dimensional problem).

The claim of this paper is that this choice of notation has two main consequences developed in the next sections:

1. All known computational networks architecture can be specified that way.
2. The weight estimation problem writes in a quite simple way, with this reformulation.

This will thus help us to revisit the recurrent weight estimation problem.

Formalizing the recurrent weight estimation

We implement the recurrent weight estimation as a variational problem, i.e. define weights as:

$$\mathbf{W} = \arg\,\min_{\mathbf{W},\mathbf{x}} \mathcal{L}(\mathbf{W},\mathbf{x}), \tag{2}$$

writing:

$$\begin{array}{lll} \mathcal{L}(\mathbf{W},\mathbf{x}) \stackrel{\text{def}}{=} & & \\ \quad \sum_{nt} & \rho_{nt}(x_n(t)) & \text{desired values} \\ - \quad \sum_{nt} & \varepsilon_{nt}\left(\hat{x}_n(t) - x_n(t)\right) & \text{network dynamic constraint} \\ + \quad \mathcal{R}(\mathbf{W}) & & \text{regularization} \end{array}$$

and represent the dynamic network recurrent equation via the notation of equation (1):

$$\left\{ \begin{array}{lll} \hat{x}_n(t) & \stackrel{\text{def}}{=} & \Phi_{n0t}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right) \\ & + & \sum_{d=1}^{D_n} W_{nd}\, \Phi_{ndt}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right) \end{array} \right.$$

while ε_{nt} are Lagrange multipliers.

Here $\rho_{nt}()$ is a cost-function (acting both as supervised and unsupervised variational term, as detailed in the next section) and $\mathcal{R}(\mathbf{W})$ a regularization term. The cost function includes both the term attached to the data, i.e., the fact that output values have a desired values, and regularization. These ingredients are going to be used in the sequel to control approximate desired output, yield sparse estimation, reduce artifact influence, obtain activity orthogonality, etc.

Stating the estimation this way, leads us to a simplified form of the Pontryagin's minimum principle, well-known in control theory [2]. The effective related solution is derived from the normal equations of the proposed criterion.

---

soon as the observation times are not too large with respect to cumulative rounding errors. As far as computing capabilities are considered, systems at the edge of chaos (but not chaotic) seem to be interesting to consider [8, 30], which fits with the present requirement.

This formulation is not new and has been formalized, by, e.g. [14]. Here we restate it at a higher level of generality, with two new aspects: (i) making explicit the role of the Lagrange multiplier (also called adjoint state in this context) for hidden units and (ii) embedding this mechanism in more general learning schemes.

Applying standard derivations, the criterion gradient writes:

$$\begin{aligned}
\nabla_{\varepsilon_{nt}} \mathcal{L} &= \hat{x}_n(t) - x_n(t) \\
\nabla_{x_{n'}(t')} \mathcal{L} &= -\varepsilon_{n't'} + \rho'_{n't'} + \sum_{nt,\ \substack{t' < t \leq t' + R \\ or\ t' = t, n < n'}} \beta^{nt}_{n't'}\, \varepsilon_{nt} \\
\nabla_{W_{nd}} \mathcal{L} &= \sum_{n'', W_{n''d} = W_{nd}} \sum_t \phi_{n''dt}\, \varepsilon_{n''t} + \nabla_{W_{nd}} \mathcal{R}
\end{aligned}$$

writing :

$$\left\{
\begin{array}{rcll}
\varepsilon_{nt} & \stackrel{\text{def}}{=} & \varepsilon_{nt} + \rho'_{nt} & \\
\rho'_{nt} & \stackrel{\text{def}}{=} & \rho'_{nt}\left(\hat{x}_n(t)\right) & \\
\phi_{ndt} & \stackrel{\text{def}}{=} & \Phi_{ndt}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right) & = \nabla_{W_{nd}} \hat{x}_n(t) \\
\beta^{nt}_{n't'} & \stackrel{\text{def}}{=} & \frac{\partial \phi_{n0t}}{\partial x_{n'}(t')} + \sum_{d=1}^{D_n} W_{nd} \frac{\partial \phi_{ndt}}{\partial x_{n'}(t')} & = \nabla_{x_{n'}(t')} \hat{x}_n(t)
\end{array}
\right.$$

The sum $\sum_{nt,\ t' < t \leq t' + R\ or\ t' = t, n < n'}$ encounters for previous values and subsequent node values. This sum includes terms with $\beta^{nt}_{n't'} \neq 0$, i.e. terms for which there is a recurrent connection from the node of index n at time t onto the node of index n' at time t'. We simply write $\sum_{nt}$ in the sequel, without any risk of ambiguity.

The sum $\sum_{n'', W_{n''d} = W_{nd}}$ encounters for weight sharing, i.e., the fact that weights from different units may be constrained to have the same value. We will simply write $\sum_{n''}$ in the sequel, without any risk of ambiguity.

Let us now review and discuss how we can implement such a minimization.

The minimization steps

Forward simulation

The equation $\nabla_{\varepsilon_{nt}} \mathcal{L} = 0$ yields $x_n(t) = \hat{x}_n(t)$, which simply corresponds to equation (1). Since $\hat{x}_n(t)$ depends on previous values at time $t' < t$, it provides a closed-form formula to evaluate $x_n(t)$ from the beginning to the end. This simply corresponds to the fact that the dynamic is simulated. This step depends on the weights W_{nd} but not on the Lagrange multipliers ε_{nt}. At the end of the step the equality $\nabla_{\varepsilon_{nt}} \mathcal{L} = 0$ is obtained, and the criterion value itself does not

depends on ε since the constraints are verified. As a consequence, the criterion value $\mathcal{L}$ can be calculated during this step.

The forward simulation complexity corresponds to the network simulation and is of order $O(NDT)$ with a memory resources of $O(NT)$ since we must buffer the calculated output, for subsequent calculations.

Backward tuning

The equation $\nabla_{x_{n'}(t')} \mathcal{L} = 0$ also provides a closed-form formula to evaluate $\varepsilon_{n't'}$ as a linear function of subsequent values $\varepsilon_{nt}, t > t'$, so that the calculation is to be done from the last time $t = T - 1$ backward to the first time $t = 0$.

This is the key feature of such a variational approach, allowing backward tuning, i.e., take into account the fact that adjusting the system parameters for a node n at time t is interdependent with the state of subsequent computations.

If $\beta^{nt}_{n't'} = 0$, there is no dependency of $x_n(t)$ on $x_{n'}(t')$, and in the absence of recurrent connection $\varepsilon_{n't'} = 0$ and $\varepsilon_{n't'}$ only depends on the state cost function ρ'_{nt}.

As mentioned by [14], $\beta^{nt}_{n't'}$ is nothing more than the first order approximation of the backward dynamics, technically the product of the weight matrix with the system Jacobian.

This backward computation is local to a given unit in the sense that only efferent units (i.e., units this unit is connected to) are involved in the computation of the related Lagrange parameter. This step depends on both weights and output values, and the equality $\nabla_{\varepsilon_{nt}} \mathcal{L} = 0$ is obtained at the end.

The backward tuning step has the same order of magnitude in terms of calculation $O(NDT)$ and memory resources of $O(NT)$ (in fact of $O(NR)$, because the obtained result may be immediately re-used to compute the weight adjustment gradient or 2nd order system).

Parameter interpretation. We obtain, after some algebra:

$$\varepsilon_{n't'} = \rho'_{nt} + \sum_{nt} \beta^{nt}_{n't'} \varepsilon_{nt} = \sum_{nt} B^{nt}_{n't'} \rho'_{nt}$$

with finite summations and for some quantities $B^{nt}_{n't'}$ (not made explicit here) which are unary coefficient polynomial in $\beta^{nt}_{n't'}$. This made explicit the fact $\varepsilon_{n't'}$ is a linear function of subsequent errors, i.e., a backward tuning error.

If the unit has no recurrent connection, i.e. is a not a function of other units, then $\varepsilon_{n't'} = \rho'_{nt}$ is simply related to the cost function. In the least-square case (i.e. if $\rho_{nt} = \frac{1}{2}(x_{nt} - \bar{o}_{nt})^2$), then $\rho'_{nt} = x_{nt} - \bar{o}_{nt}$ is the output error.

It must be noted, that this method is quite different from back-propagation-though-time recurrent network estimation or other standard alternatives. If the calculation may be recognized as a kind of back-propagation, it is mathematically different.

Numerical stability. However, as reviewed in e.g., [25] this back-propagation of tuning error, will suffer form the same curse than back-propagation of gradient: Either error explosion (if $|\beta^{nt}_{n't'}| > 1$), or error vanishing (if $|\beta^{nt}_{n't'}| < 1$). In

our case, since all kernels are contracting we have the bound, writing $\beta_{max} \stackrel{\text{def}}{=} \max_{nt} |\beta^{nt}_{n't'}|$:

$$0 \leq |\beta^{nt}_{n't'}| \leq \beta_{max} \leq 1 + \textstyle\sum_d |W_{nd}|$$

without any thinner inequality in the general case. Based on this remark, the key idea of LSTM [25] is to consider memory carousel as reviewed in the previous section to guaranty $|\beta^{nt}_{n't'}| \simeq 1$ and thus a stable back-propagation for at least some recurrent link. In our framework this means that it is the responsibility of the designer of the network architecture to consider nodes with such property.

Here, we are going to introduce another heuristic and *bias* the backward error in order to avoid both error explosion and vanishing. To this end, we define:

$$\kappa_{n't'} = \rho'_{nt} + g_\epsilon \left(\sum_{nt} \beta^{nt}_{n't'} \, \kappa_{nt}, \beta_{max} \right), \tag{3}$$

considering a function $g_\epsilon(u, \beta_{max})$ yielding both amplification of small errors and saturation of larger error, while at the algorithm convergence we need to have $g(u) = u$ in order to yield an unbiased estimation. Here is a continuation parameter, with $\epsilon = 1$ at the minimization start, while $\epsilon \to 0$ with algorithm convergence. It is adjusted by the minimization algorithm, which looks for a convergence at $\epsilon = 1$ and then decreases ϵ while tracking the minimum during ϵ decrease.

A simple choice is to consider an exponential saturation. Taking into account the previous requirements, for $0 < \epsilon < 1$, writes:

$$g_\epsilon(u, \beta_{max}) \stackrel{\text{def}}{=} sg(u) \, \left(1 + \beta_{max} \, \tfrac{1-\epsilon}{\epsilon}\right) \left(1 - e^{-\frac{\epsilon}{\beta_{max}} |u|}\right)$$

and it is obvious to verify that:

$$\begin{array}{rcll} g_\epsilon(u, \beta_{max}) & = & u + O(\epsilon) & \text{convergence to identity when } \epsilon \to 0 \\ & = & \frac{u}{\beta_{max}} + O(u^2) + O(1-\epsilon) & \text{amplification around } \epsilon = 1 \\ & < & \frac{1}{\epsilon} & \text{saturation for non negligible } \epsilon \\ & \leq & \left((1-\epsilon) + \frac{\epsilon}{\beta_{max}}\right) u & \text{concave profile} \end{array}$$

The convergence to identify is uniform for bounded values of u and the amplification leads to $\left|\frac{\partial \dot{g}()}{\partial \kappa_{nt}}\right| \leq 1$ but as closed as possible to 1, avoiding any propagation explosion, with at least some non negligible backward tuning. With the decrease of ϵ we may and must relax this constraint, but can reasonably expect the backward tuning error to decrease with the criterion convergence, the related quantities to remain bounded.

Real-time aspects. Such a formulation is definitely not "real-time", since we "go back in time". It is however, the only solution for hidden layers to be tuned, since the output is a function of hidden activity in the past.

However, in a real-time paradigm, it must be noted that each computation is also local in *time*: It only depends on values in a "near future" within a time range equal to the system time range. In other words, at a given time we obtain the value with a lag equal to system time-range. It is an interesting perspective of this work to explore if, in a rather stationary context, it may

provide numerically relevant values for on-the-fly backward tuning.

The 1st order unit weight adjustment

The calculation of $\nabla_{W_{nd}} \mathcal{L}$ yields a Hebbian weight adaptation rule (as the sum of products between an output unit error term ε_{nt} (combining the supervised error and the backward tuning multiplier) and an input quantity ϕ_{ndt}. This rule applies to both output unit of index $n < N_0$ with a desired output and hidden units of index $N_0 \leq n$ that indirectly adapt their behavior to optimize the output, via the backward tuning values. The gradient calculation is local to a given unit and average over time, through another $O(NDT)$ computation.

This leads to a local 1st order adjustment of the weights, i.e. it provides the direction for the weight variation, not its magnitude.

To numerically adjust this magnitude we have to perform a 1D minimization, i.e., $\min_{\upsilon^k} \mathcal{L}(\mathbf{W}(\upsilon^k), \tilde{\mathbf{x}})$, with:

$$W_{nd}(\upsilon^k) = \tilde{W}_{nd} - \upsilon^k \nabla_{\tilde{W}_{nd}} \mathcal{L}, \text{ with } 0 < \upsilon^k$$

where $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{x}}$ correspond to the previous estimation of the weights and activity. This minimization can be local to each unit. The key point is that we do not know any reasonable upper-bound for υ^k. As a consequence standard golden section search or Brent-Dekker methods do not directly apply, and we have to numerically find such an upper-bound first, and it appeared that which requires additional calculation steps. It is also known that it is inefficient to precisely find the minimum since the gradient value varies with W_{nd} so that the result is anyway biased. Taking all this into account, we obtain the 1st order unit weight adjustment heuristic:

-0- Starts with the last known value of υ^k or very small value (here 10^{-6}).
-1- Computes $\mathcal{L}(W_{nd}(\upsilon^k))$.
-2.1- If it decreases, register this better value, and set $\upsilon^k = 2\,\upsilon^k$ for the next step,
-2.2- else set $\upsilon^k = \upsilon^k/3$.
-3- Repeat -1- unless steps -2.2- leads to a negligible value of υ^k.

Each step requires a simulation to compute $\mathcal{L}$.

It is easy to verify that this tiny heuristic converges and provides an approximation of the minimum. It also provides an upper-bound for usual minimum search methods, but it has be numerically observed, at the global minimization level, that adding such search methods is useless (it slows-down the minimization with only a negligible gain in precision, if any).

As far as backward tuning stability is concerned the value of ε_{nt} is replaced by its approximation κ_{nt}.

The 1st order unit weight adjustment could either be done globally at the whole node set level, or locally for each unit, the criterion itself being decomposable on each unit. In this latter case, the global convergence is guaranty only with infinitesimal gradient steps.

The 2nd order unit weight adjustment

Due to the simplicity of the approach, we can write a 2nd order weight adjustment:

$$\nabla_{W_{nd}} \mathcal{L} = 0 \Rightarrow \sum_{n''} b_{n'',\,d} = \sum_{n''} \sum_{d'=1}^{D_n} A_{n'',\,d\,d'}\, W_{nd'}$$

writing, for some κ_{nt}:

$$\begin{cases} b_{n,\,d} & \stackrel{\text{def}}{=} & \sum_t \phi_{ndt}\ (\varepsilon_{nt} + \kappa_{nt}\ (\hat{x}_n(t) - \phi_{n0t})) + \nabla_{W_{nd}} \mathcal{R}(\tilde{\mathbf{W}}), \\ A_{n,\,d\,d'} & \stackrel{\text{def}}{=} & \sum_t \kappa_{nt}\, \phi_{ndt}\, \phi_{nd't}. \end{cases}$$

This allows to obtain a new weight value $\tilde{\mathbf{W}}$ solving a linear system of equation for each unit, where $\hat{x}_n(t)$ is calculated taking the previous or initial weight value $\hat{\mathbf{W}}$ into account.

Here indeed, the value of κ_{nt} corresponds to the backward tuning stability proposed heuristic.

The chosen form is related to the 2nd order Hessian of the criterion[2] and generalizes the readout least-square estimation of weights[3] in a neural network.

The weight adjustment is local to each unit, providing a true distributed mechanism. This corresponds to a 2nd order minimization scheme. Each step requires $O(N(DT+D^3)$ operation, solving a linear system of equations. The implemented method is a Cholesky decomposition with a fallback onto a singular-value-decomposition if the $\mathbf{A}_n$ is not strictly positive.

[2] The criterion Hessian, omitting the regularization term, writes:

$$\nabla_{W_{nd} W_{n'd'}} \mathcal{L} \quad = \quad \delta_{n=n'} \sum_t \rho''_{nt}\, \phi_{ndt}\, \phi_{nd't}$$

Here the notation $\delta_{\mathcal{P}}$ stands for 1 is the property $\mathcal{P}$ is true and 0 otherwise.

A step further, for the sake of completeness we also make explicit (writing $\beta^{nt}_{nt} \stackrel{\text{def}}{=} -1$):

$$\begin{cases} \nabla_{\varepsilon_{nt}\varepsilon_{n't'}} \mathcal{L} & = & 0 \\ \nabla_{x_{n'}(t')\varepsilon_{nt}} \mathcal{L} & = & \beta^{nt}_{n't'} \\ \nabla_{W_{nd}\varepsilon_{n't'}} \mathcal{L} & = & \delta_{n=n'}\, \phi_{ndt'} \\ \nabla_{x_{n'}(t')x_{n''}(t'')} \mathcal{L} & = & \sum_{nt} \rho''_{nt}\, \beta^{nt}_{n't'}\, \beta^{nt}_{n''t''} \\ & + & \left[\frac{\partial^2 \phi_{n0t}}{\partial x_{n'}(t') x_{n''}(t'')} + \sum_{d=1}^{D_n} W_{nd}\, \frac{\partial^2 \phi_{ndt}}{\partial x_{n'}(t') x_{n''}(t'')} \right] \varepsilon_{nt}, \\ \nabla_{W_{nd}x_{n'}(t')} \mathcal{L} & = & \sum_t \rho''_{nt}\, \beta^{nt}_{n't'}\, \phi_{ndt} + \frac{\partial \phi_{ndt}}{\partial x_{n'}(t')}\, \varepsilon_{nt}, \end{cases}.$$

[3] For a simple least-square criterion of the form:

$$\mathcal{L} = \sum_{nt} \frac{\kappa_{nt}}{2}\, (\hat{x}_n(t) - \bar{o}_n(t))^2$$

where $\kappa_{nt} \in \{0, 1\}$ depending on the fact that the desired output $\bar{o}_n(t)$ is defined or not, it is straight-forward to verify the proposed 2nd order weight adjustment holds, and reduces to an exact linear system of equation, in the absence of recurrent links of the given unit, since ϕ_{ndt} is only function of the input. Otherwise, we only ϕ_{ndt} is also a function of both the network unknown output and hidden node values. For output node value the $\bar{o}_n(t)$ desired value could be enforced, limiting recurrent perturbation and yielding ϕ_{ndt} values closed to the ideal value, which is interesting in reverse-engineering estimation, i.e. when an exact solution is expected [34], whereas a bias in the estimation is otherwise introduced.

Taking all this into account, we obtain the following 2nd order unit weight adjustment heuristic:

-0- Calculates $\tilde{\mathbf{W}}$ and starts with $\upsilon^k = 1$.
-1- Computes $\mathcal{L}(\upsilon^k \tilde{\mathbf{W}} + (1 - \upsilon^k)\, \hat{\mathbf{W}})$.
-2.1- If it decreases, register this better value,
-2.2- else set $\upsilon^k = \upsilon^k / 2$.
-3- Repeat -1- unless steps -2.2- leads to a negligible value of υ^k.

In words we look for a weight value between both previous and new values that decreases the criterion. Each step requires a simulation to compute $\mathcal{L}$.

The complete weight adjustment

Collecting previous steps the final iterative weight adjustment writes

-1- Performs a forward simulation and a backward tuning, calculating the 1st order gradient and 2nd order elementsduring the backward estimation.
-2.a- Attempt to perform a 2nd order weight adjustment.
-2.b- If it fails, attempt to perform a 1st order weight adjustment.
-3- Repeat -1- unless steps -2.b- fails.

This weight adjustment has to be performed by continuation using the modified numerically stable backward tuning calculation.

The present proposal stands on the fact that we have been able to derive a local 2nd order adjustment based on renormalized backward tuning. The 1st order fall-back method could indeed have been enhanced using either the so called momentum gradient mechanism (based on a temporal averaging of the gradient values), or the partial conjoint gradient methods (taking into account several subsequent gradient directions in order to infer an approximate 2nd order mimimization method. We also propose here an alternative to 2nd order adjustment methods such as [32] or other methods reviewed in [22].

Experimentation

In this experimental part we are going to both study the numerical stability and limit of the method and compare to existing non-trivial benchmark problems. Here, supervised learning is targeted since it is a direct way to evaluate the method efficiency and robustness. Let us remember that we do not evaluate learning performances here, only the way me can adjust recurrent network weights. We only study the estimation convergence here, not the learning properties (such as generalization, capacity, ...).

Software implementation

In order to provide so called reproducible science [38], the code is implemented as a very simple, highly modular, fully documented, open source, object oriented,

easily forkable, self contained, middle-ware, available here: https://vthierry.github.io/mnemonas. A minimal set of standard mechanisms (random number generation, histogram estimation, linear system resolution, system calls) is used. The main part of the implementation hierarchy is show in Fig. 2.

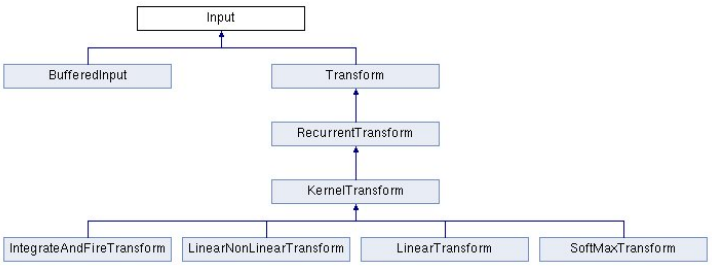

Figure 2: A view of the class-hierarchy: A `Input` simply provides a $x_n(t), n \in \{0, N\{, t \in \{0, T\{$ values, while a `Transform` provides such values given another `Input`, while other objects defined here derive from such an oversimple abstract class.

For run-time performances and inter-operability with different programming languages a `C/C++` implementation (with the compilation scripts) is proposed, the wrapping to other programming languages (e.g., `Python`) being straightforward, using e.g. `swig`.

The first experimental verification was that it was very simple to define the main unit structures reviewed in section A from `KernelTransform` as claimed in the paper.

Numerical stability and limit of the method

Regarding this first issue, as being in a deterministic context, we are going to rely on a reverse engineering setup: An input/output learning sequence is going to be generated by a input/output root network of $\bar{N}$ units and another learning network with random initialization is going to re-estimate a transform. This guaranty the existence of an exact solution.

How revelant is it to use such a reverse engineering setup ? On one hand, surprisingly enough perhaps, such networks (at least deep networks [42]) behave with the same order of magnitude of performances, the input being either "meaningful" in the sense it represents data with a semantic or not. We thus can expect simple random input/output tests to be relevant to more specific application. On the other hand we are going to develop in the next section how several "challenging" tests are in fact highly dependent on the chosen architecture, with often trivial solution, as soon as the hidden architecture is well chosen.

Considering random input/output with some statistics, in most of the cases, they are several solutions (e.g. up to a permutation of the units, or some linear combination in a linear case, ...). We consider a root network of $\bar{N}$ units for a

sequence of time T, for a $M = 1$ scalar input, considering either L (for linear), LNL or AIF units, with random weights (drawn from a Gaussian distribution with 0 mean and $\sigma \simeq 1/N$ standard deviation, which is know to guaranty a stable non-trivial dynamic). Only the unit of index $n = 0$ is considered as output units, i.e., $N_0 = 1$, the $N - 1$ remainder units activity being hidden to the estimation.

In this determistic case, we observe two main parameters the final precision criterion value $\mathcal{L}$ and the number of steps to convergence S.

A step further, we also study exact solution or approximation, we are going to consider learning network with $N \leq \bar{N}$ units, i.e. networks that do not generate the exact solution. We already know that as soon as the dynamic is sufficiently rich, even small errors accumulates and the solution exponentially diverges from the exact one. In such a case the question is wether the input/output statistics also diverges. We thus have to compare the KL-divergence between the desired and obtained output given the input. The fact we chose $N_0 = 1$ makes this estimation tracktable since it is a 1D distribution. OUI MAIS JUSTE HISTOGRAMME QUID DE LA DEPENDENCE EN i(t-R) ?

Comparison with existing estimation problems

Let us now discuss how our validation method compares with usual benchmark for recurrent network estimation.

Conclusion

We consider another formulation of weight estimation in recurrent networks, proposing a notation for a large amount of recurrent network units that helps formulating the estimation problem. Reusing a "good old" control-theory principle, improved here using a continuation heuristic, we obtain a numerically stable and rather efficient second-order and distributed estimation, without any meta-parameter to adjust. The relation with existing technique is discussed at each step. The proposed method is validated using reverse engineering tasks and non-trivial numerical computations.

A Major examples fitting this architecture.

The notation of equation (1) seems to be the most general form of usual recurrent networks. Let us state this point by considering several examples of units, and make explicit how we decompose them in term of nodes.

Linear non-linear (LNL) units. Such network unit corresponds to the most common[4] network unit and is defined by a recurrent equation of the form:

$$\begin{array}{rcl} x_n(t) & = & \gamma_n\, x_n(t-1) \\ & + & \zeta_{[a,b]}\left(\alpha_n + \sum_{n'=0}^{N-1} W_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{M-1} W_{nm}\, i_m(t-1)\right), \end{array} \tag{4}$$

- with a fixed of adjustable *leak*[5] γ_n, $0 < \gamma_n < 1$, and
- optionally *intrinsic plasticity* parameterized by α_n.

The *non-linearity* often[6] writes $\zeta_{[a,b]}(u) \stackrel{\text{def}}{=} \frac{a+b}{2} + \frac{b-a}{2}\tanh(\frac{2}{b-a}\,u)$, with $\zeta_{[a,b]}(-\infty) = a, \zeta_{[a,b]}(+\infty) = b, \zeta_{[a,b]}(u) = \frac{a+b}{2} + u + O(u^3)$, while $\zeta'(u) = 1 - \tanh(\frac{2}{b-a}\,u)^2$, $0 < \zeta'(u) \leq 1$, thus contracting. We mainly have $[a,b] = [0,1]$ or $[a,b] = [-1,1]$ depending on the semantic interpretation of the $x_n(t)$ variable.

Another form of non-linearity is a rectified linear unit (or ReLU), i.e., $\zeta_{[0,+\infty]}(u) \stackrel{\text{def}}{=} \max(0,u)$. This function is not derivable at $u = 0$. It is however very easy, to consider a mollification (called softplus) e.g., $\zeta_{\epsilon,[0,+\infty]}(u) \stackrel{\text{def}}{=} \epsilon \log\left(1 + e^{\frac{u}{\epsilon}}\right)$ which is an analytic smooth approximation which uniformaly converges[7], i.e. $\lim_{\epsilon\to 0} \zeta_{\epsilon,[0,+\infty]}(u) = \zeta_{[0,+\infty]}(u)$. See the section on AIF units to define how to adjust such meta-parameter, as unit parameter.

For adjustable leak we need three nodes to fit within the proposed notations:

$$\begin{array}{rcl} x_n(t) & = & x_{n_1}(t) + \zeta_{[a,b]}\left(x_{n_2}(t)\right) \\ x_{n_1}(t) & = & \gamma_n\, x_n(t-1) \\ x_{n_2}(t) & = & \alpha_n + \sum_{n'=0}^{N} W_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{N} W_{nm}\, i_m(t-1) \end{array}$$

and it is easy to verify that this second form fits with equation (1), since:
- The 1st line corresponds to a parameter-less $\Phi_{n0t}\,()$ kernel (unit firmware).
- The 2nd and 3rd lines correspond to linear combinations of elementary kernels $\Phi_{ndt}\,()$ selecting another state or input variable (unit learnware).

With this example, we see that the proposed approach is to introduce two additional intermediate variables $x_{n_1}(t)$ and $x_{n_2}(t)$ related to each linear combination of weights or other parameter.

With a fixed leak (i.e., if the value γ_n is known) the LNL unit decomposes into two nodes, a parameter less node combining $x_n(t)$ and $x_{n_1}(t)$, and the linear combination defined for $x_{n_2}(t)$.

This equation is also valid for the main auto-encoder architectures, and for convolution networks [4, 15], with an important additional feature : weight-sharing, i.e. the fact that several weights W_{nd} are the same across different nodes. This is going to be taken into account in the sequel.

[4] See also a dual form related to AIF, in the sequel, with an alternate insertion of the non-linearity.

[5] Here $\gamma = 1 - \frac{\Delta T}{\tau}$ stands for the leak of each unit, writing ΔT the sampling period, τ the continuous leak and using an basic trivial Euler discretization scheme, the $\zeta()$ profile being re-normalized accordingly.

[6] If the model corresponds to a rate, i.e., a firing probability, we can use the logistic sigmoid, which writes $\zeta_{[0,1]}(u) = \frac{1}{1+e^{-4\,u}} = \frac{1+\tanh(2\,u)}{2}$.

[7] Since $\forall u, |\zeta_{\epsilon,[0,+\infty]}(u) - \zeta_{[0,+\infty]}(u)| \leq \log(2)\,\epsilon$.

Long short term memory (LSTM) units. Such network unit is defined by a sophisticated architecture [25], described in figure 1. A unit is made of the following nodes:

$$
\begin{array}{rcl}
\text{Unit output:} & & \\
x_n(t) & = & \zeta_{[0,1]}\left(y_n^{out}(t)\right)\, \zeta_{[-1,1]}\left(s_n(t)\right) \\
\text{Unit state:} & & \\
s_n(t) & = & \zeta_{[0,1]}\left(y_n^{forget}(t)\right)\, s_n(t-1) + \zeta_{[0,1]}\left(y_n^{in}(t)\right)\, \zeta_{[-1,1]}\left(g_n(t)\right) \\
\text{Unit gate:} & & \\
g_n(t) & = & \sum_{n'} W^g_{nn'}\, x_{n'}(t-1) + \sum_m W^g_{nm}\, i_m(t-1) \\
\text{Output modulation:} & & \\
y_n^{out}(t) & = & W^o_n\, s_n(t-1) + \sum_{n'} W^o_{nn'}\, y^c_{n'}(t-1) + \sum_m W^o_{nm}\, i_m(t-1) \\
\text{Forgetting modulation:} & & \\
y_n^{forget}(t) & = & W^f_n\, s_n(t-1) + \sum_{n'} W^f_{nn'}\, y^c_{n'}(t-1) + \sum_m W^f_{nm}\, i_m(t-1) \\
\text{Memorizing modulation:} & & \\
y_n^{in}(t) & = & W^i_n\, s_n(t-1) + \sum_{n'} W^i_{nn'}\, y^c_{n'}(t-1) + \sum_m W^i_{nm}\, i_m(t-1)
\end{array}
\tag{5}
$$

The first two nodes are parameter-less additive and/or multiplicative combination of non-linear functions of the reminding four nodes, which are themselves linear combination of the incoming signal gate and the input, forgetting and output modulatory signals.

The present notation corresponds to the most general form (e.g., with peephole connections [21]) of LSTM, while several variants exists. A rather closed mechanism is named gate recurrent unit [13], and is based on the same basic ideas of modulatory combination, but with a simpler architecture. We do not make explicit the equations for all variants of LSTM here, just notice that they correspond to some of the very best solutions for high performance recurrent network computation [35].

However, in our context, instead of reusing such a complex unit as it, the design choice is to consider the non standard nodes (i.e., unit output and unit state) as modular nodes that could be combined with NLN at different level of complexity, depending on the task. At the implementation level we are not going to provide LSTM units as black boxes but an object-oriented framework allowing to adjust the network architecture to the dedicated task.

A keypoint is that LSTM have, by construction, a real vertu regarding weight adjustment since back-propagation curses (vanishing or explosion) is avoided [35]. A strong claim of this paper is that we can efficiently adjust the recurrent network weights even if we do not use (or only use) LSTM but simpler units also.

Strongly-Typed Recurrent Neural units. This formalism [3] carefully considers the signal type in the sense of parameters of different physical origins (e.g., Volts and meter), that cannot be simply mixed. This approach allows unary and binary functions on vectorial values of the same type, transformation from one orthogonal basis to another (thus using orthogonal matrices only) and component-wise product (i.e., modulatory combination). The authors

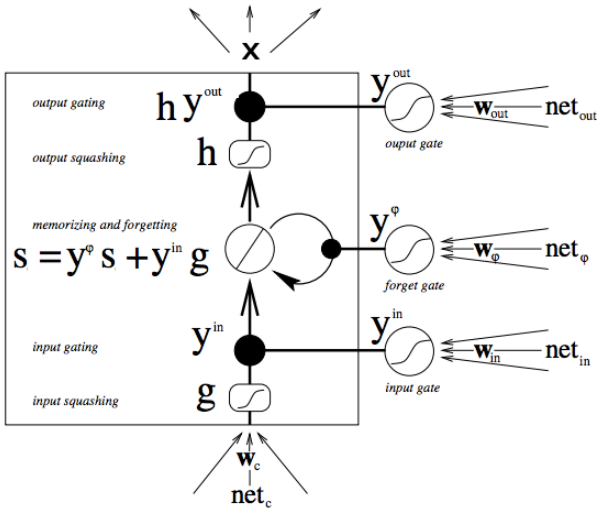

Figure 3: A LSTM unit has three processing stages for bottom to top: The (i) gate g corresponds to as standard LNL unit that (ii) feeds an internal state memory s which value is also driven by a forget (or remember) signal allowing to maintain the previous value, before (iii) the output connected value x diffuse (or not) the result in the network. The LSTM mechanism is thus based on three ingredients, (a) the use of modulatory connection (i.e., with a multiplication by a number between 0 and 1 in order to control the signal gain), (b) a memory "carrousel" (i.e., an equation that could be of the form $s_n(t) = s_n(t-1)$ in order to maintain a signal, during a long short-term delay), and (c) the use of several modulatory signals. From [25].

show that strongly-typed gradients better behaved and that, despite being more constrained, strongly-typed architectures achieve lower training and comparable generalization error to classical architectures. Considering a strongly-typed LNL unit, following [3] and translating in the present notation, at the same degree of generality of LNL networks, we obtain:

$$\begin{array}{rcl} x_n(t) & = & \zeta_{[0,1]}\left(f_n(t)\right)\, x_n(t-1) + \left(1-\zeta_{[0,1]}\left(f_n(t)\right)\right)\, z_n(t) \\ f_n(t) & = & \alpha_n + \gamma_n\, x_n(t-1) \\ z_n(t) & = & \sum_{n'=0}^{N} W'_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{N} W'_{nm}\, i_m(t-1) \end{array} \tag{6}$$

The first line is the firmware combination of the unit forgetting mechanism, this value being defined in the 2nd line, while the 3rd line performs the linear combination of other network values. It is an interesting alternative to usual approach, embedable in our notation.

Approximation of *leaky integrate and fire* (AIF), current-driven, spiking-neuron unit. Let us also discuss how to cope with spiking networks (see [11]

for a general discussion on such network computational power and limit). Following [10], we consider without loss of generality a unit threshold, discretized form, which writes:

$$\begin{array}{rcl} x_n(t) & = & \gamma_n \left((1 - H_\epsilon \left(x_n(t-1) - 1 \right)) \, x_n(t-1) \right. \\ & + & \left. \sum_{n'=0}^{N} W_{nn'} \, H_\epsilon \left(x_{n'}(t-1) - 1 \right) + \sum_{m=0}^{N} W_{nm} \, i_m(t-1) \right), \end{array} \tag{7}$$

where $H_\epsilon(v)$ approximates the Heaviside function $H(v)$. More precisely, from [12], we consider a mollification of the Heaviside function $H_\epsilon(v) \stackrel{\text{def}}{=} \zeta_{[0,1]}\left(\frac{v}{\epsilon}\right) = \frac{1}{1+e^{-\frac{4\,v}{\epsilon}}}$. Obviously, $\lim_{\epsilon \to 0} H_\epsilon(v) = H(v), with H(0) = 1/2$. Here the convergence can not be uniform (since a continuous function converges towards a step function). To avoid spurious effects when adjusting the weights, we have to find out for each unit the best minimal ϵ value.

As far as the unit architecture is concerned, it is a simple variant of LNL unit, with different kernel function, and different poitionning of the non-linearity. The key point is that this so called BMS formulation fits with the present approach:

$$\begin{array}{rcl} x_n(t) & = & \gamma_n \left[(1 - \zeta_{[0,1]}(x_{n_1}(t))) \, x_n(t-1) \right] \\ & + & \sum_{n'=0}^{N} W_{nn'} \, \zeta_{[0,1]} \left(x_{n'_1}(t-1) \right) + \sum_{m=0}^{N} W_{nm} \, i_m(t-1) \\ x_{n_1}(t) & = & \frac{1}{\epsilon} \left(x_n(t-1) - 1 \right) \end{array}$$

Here $\omega \stackrel{\text{def}}{=} \frac{1}{\epsilon}$ is now a parameter to estimate, in order each unit to be a suitable approximation of a spiking activity. This differs from [12] where sharpness was considered as a meta-parameter: Here it is a parameter learned on the data. In both cases, the limit of the trick is that we need $\epsilon \to 0$, which means that the transformation is very sharp, limiting the numerical stability. This is going to be investigated at the numerical level.

The use of such units is very interesting in practice and we review in appendix **??**

Softmax and exponential probability units. When considering exponential distribution of probability on one hand, or softmax[8] computation on the other hand, one comes to the same equation[9] which writes:

$$\begin{array}{rcl} x_n(t) & = & \frac{e^{z_n(t)}}{\sum_n e^{z_n(t)}} = \exp\left(z_n(t) - \log(\sum_n \exp\left(z_n(t) \right) \right) \\ z_n(t) & = & \alpha_n + \sum_{n'=0}^{N} W'_{nn'} \, x_{n'}(t-1) + \sum_{m=0}^{N} W_{nm} \, i_m(t-1) \end{array} \tag{8}$$

with $\sum_n x_n(t) = 1$ in relation with the so-called partition function $Z(t) = \sum_n \exp\left(z_n(t) \right) > 0$.

This kind of unit, in addition to NLN units, or LSTM units form the basic components of deep-learning architectures [4, 15].

[8] The relation with a max operator comes from the fact that:

$$x_n(t) \stackrel{\text{def}}{=} \frac{e^{\frac{z_n(t)}{\epsilon}}}{\sum_n e^{\frac{z_n(t)}{\epsilon}}} \Rightarrow \lim_{\epsilon \to 0} \sum_n x_n(t) \, z_n(t) = \max_n \left(z_n(t) \right).$$

In words the softmax weighted sum of values approximates these values maximum.

[9] See, e.g., `https://en.wikipedia.org/wiki/Softmax_function`.

The 1st line is a firmware global equation[10] which is a function of all units value of the same layer.

We encounter such a construction in restricted Boltzmann machine (RBM) (also using LNL network with the logistic sigmoid, but in a context of stochastic activation of the units in this case) [4]. We mention this possibility for the completeness of the discussion, making explicit the fact that the present framework includes such equation. However, the estimation problem addressed in RBM completly differs (as being a stochastic estimation paradigm) strongly differs form the deterministic estimation considered here, the key difference being the fact we want relevant results event on small data sets.

Other aspects of the proposed notation It is also straightforward to verify that the reservoir computing equations [41] also fit with this framework, as being a particular of LNL network, since they simply correspond to a recurrent reservoir of interconnected units, plus a read-out layer.

Since there is no restriction on the architecture, depending on the choice of the kernels, it also can represent a two-layers non-linear network, or even better a multi-layers deep network. The trick is simply to choose kernels corresponding to the desired inter-layer and intra-layer connectivity.

A step further, in a given architecture, we can adjust both the number of layers and the choice between one or another computation layer. This aspect if futher discussed in [18]. We also would like to consider not only a sequence of layers, but a more general acyclic graph of layers, noticing that shortcuts can strongly improve the performance thanks to what is called residual-learning [23]. Following [17], the key-point is that we want to have this structural optimization as a parameter continuous adjustment and not a meta-parameter combinatory adjustment. The proposal is thus to consider an architecture with *versatile layers* where the choice of the non-linearity is performed via a linear combination, obtained with sparse estimation, thus acting as a soft switch. Furthermore, adding shortcuts allows to define an adjustable acyclic graph with the output as supremum and the input as infimum. On the reverse, [17] points out that any acyclic graph can obviously be defined in this framework. Of course, we do not expect this method to generate the best acyclic graph and combination of modules, but to improve an existing architecture by extending usual optimization to the exploration of structural alternatives.

B Using this framewrok in different contexts

In this section we review classical mechanism of estimation that can make use of the previous estimation mechanism.

[10] It is worthwhile mentioning that:

$$\frac{\partial o_n(t)}{\partial z_{n'}(t)} = o_n(t)\ (\delta_{n=n'} - o_{n'}(t)) \in [0,1], \delta_{n=n'} = \left\{ \begin{array}{ll} 1 & n = n' \\ 0 & \text{otherwise} \end{array} \right. ,$$

thus numerically well defined, with no singularity, the transformation being contracting, i.e., $\left|\frac{\partial \mathbf{o}}{\partial \mathbf{z}}\right| \leq 1$.

Considering a supervised learning paradigm.

If we focus on a supervised learning paradigm, we consider learning sequences of size T with desired output $\bar{\mathbf{o}}(t), 0 \leq t < T$, corresponding to the input $\bar{\mathbf{i}}(t)$, in order to adjust the weights.

This setup includes without loss of generality the possibility to use several epochs (i.e., several sequences): They are simply concatenated with a period of time with state reset at the end of each epoch, in order to guaranty to have independent state sequences.

With respect to desired output $\bar{o}_n(t)$ we can write:

$$\rho_{nt}(\hat{x}_n(t)) = \frac{\kappa_{nt}}{2}(\hat{x}_n(t) - \bar{o}_n(t))^2$$

On one hand, we choose $\kappa_{nt} > 0$ if $\bar{o}_n(t)$ is defined (output node) and $\kappa_{nt} = 0$ otherwise (hidden unit, missing data, or segmentation of the sequence in different epochs, see Fig. 4), while since $\kappa_{nt} \in [0, +\infty[$ it can also act as error gain, taking related precision into account.

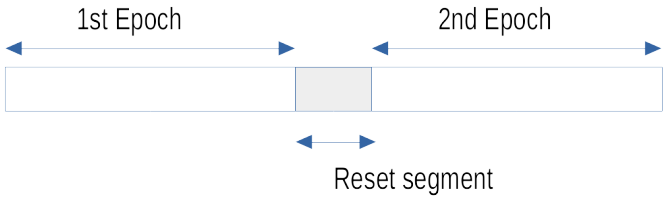

Figure 4: If the supervised learning is performed with different epoch of data, this is equivalent to a unique epoch, providing a reset segment of length R, the maximal recurrent range, is inserted before each new epoch. During reset segment, we set $\kappa_{nt} = 0$.

One aspect of the estimation is related to robustness, i.e., being able to take into account the fact that errors and artifacts may occur in the learning set, implemented here as a M-estimator, i.e., not a least-square function but another alternative cost function, with a smaller slope for higher values.

Considering static estimation.

The present framework stands for dynamic estimation of a temporal sequence. It can also simply be applied to a static estimation at the final time step $T-1$ considering $\bar{o}_n(T-1)$ only the previous values $o_n(t)$ being unconstrained. In that case the value T corresponds to the number of iteration to obtain the desired estimation. In a non-recurrent architecture this value is easy to derive from the architecture, it corresponds to the number of computation steps. In a recurrent architecture, the situation is more complex since computation loops have to converged, and the number of computation steps is an explicit parameter, unless

the system is tuned to converge to a fixed point, while considering $T \to +\infty$ which is a rather straightforward extension of the present work.

Considering constrained architecture and weights values.

It is precious to also introduce constraints on the connection weights. Typical constraints include:
- sparse connectivity, which reduces the total amount of computation, and allows internal sub-assemblies to emerge,
- positive or negative weight values (corresponding to excitatory or inhibitory connections).

The design choice of the kernels allows us to constraint the network connectivity. It is possible to specify partial connectivity allowing to distinguish different layers (e.g. hidden layers not connected to input and/or output). This may be, for instance, a 2D-topography with local horizontal connections, or several layers with, e.g., either point to point, or divergent connectivity between layers.

However, if the architecture itself has to be learned, the present framework may be used in another way: Starting from a given connected network and performing a sparse estimation, may lead to a result with zero weight values for connections not present in the estimated architecture, and non zero values otherwise. This is a sparse estimation, i.e. not only minimizing the metric not only with respect to the weights values, but also with respect to the fact that some weights have either zero or non-zero values, i,e, with respect connection sets. Sparse estimation methods (see e.g. [39, 40] for a didactic introduction) can be used to this end.

One application could be modulatory weighted connections, allowing to enhance or cancel sub-parts of the network connectivity.

In our case we may simply choose, for some meta-parameters ν_{nd}:

$$\mathcal{R}(\mathbf{W}) = \sum_{nd} \frac{\nu_{nd}}{\epsilon + |\hat{W}_{nd}|} W_{nd}^2$$

where $\hat{W}_{nd}$ stands for the best a-priory or previous estimation of the weight. This leads to a reweighted least-square criterion, where small weights value minimization is reinforced, up to 0, yielding sparse estimation.

The case where we consider excitatory or inhibitory connections (i.e., weight values that only positive or negative), or the case where the weights are bounded, is managed at the implementation level, as a hard constraint in the minimization. Very simply, if the value is beyond the bound it is reprojected on on the bound. This may lead to a sub-optimal estimation, but avoids the heavy management of Karush-Kuhn-Tucker conditions.

As an example, let us consider the adjustable leak γ_{nt}, $0 \leq \gamma_{nt} \leq 0.99 \simeq 1$ of a NLN unit. If the minimization process yields a negative value, the value is reset to zero (it means that we better have no leak). If the minimization process yields an unstable value higher than one, it is reset to, say, 0.99 to be sure the system will not diverge.

Considering un-supervised regularization.

In order to find an interesting solution, we have to constraint the hidden activity to be estimated. Interesting properties includes sparseness, orthogonality, robustness and bounds.

Sparse activity (i.e., with a maximal number of values closed or equal to zero), which is known to correspond to unit assemblies tuned to a given class of input statistics, can be specified as a reweighted least-square criterion again, for some meta-parameters κ_{nd}:

$$\rho_{nt}(x_{nt}) = \frac{\kappa_{nd}}{\epsilon + |\hat{x}_{nt}|}\, x_{nt}^2$$

where $\hat{x}_{nd}$ stands for the previous estimation, with an initial value equal to κ_{nd}.

Orthogonality of hidden unit activities, in order to avoid redundancy and maximize the dynamic space dimension in the recurrent network, can also be specified, the same way as :

$$\rho_{nt}(x_{nt}) = \kappa_{nd} \sum_{n' \neq n} (\sum_{t} x_{nt}\, \hat{x}_{n't})^2$$

again as as, now not local but global, reweighted least-square criterion, now minimizing the dot products between unit activities, thus minimal when orthogonal.

Another aspect concerns the fact we may have to control the activity bound, e.g., a weak constraint of the form $x_{nt} \preceq b$. Following the same heuristic, we may introduce a cost of the form:

$$\rho_{nt}(x_{nt}) = \kappa_{nd}\, e^{k\,(x_{nt} - b)}$$

with $k > 0$ in order to have a fast increasing function as soon as the bound is violated.

C Closed forms solution for neural network tasks

Let us make explicit here the type of used units has a strong influence on the difficulty of the task. Here we consider deterministic tasks only. The remark is that tasks considered as quite complex [25, 21, 32] for certain architectures are trivial for others. In particular, the use of AIF neurons dramatically simplifies certain problems. We ilustrate this point here considering deterministic sequence generation and long-term non-linera transform.

Deterministic sequence generation

Let us make explicit the complexity of the task of generating a deterministic time sequence $\bar{o}_n(t), n \in \{0, N_0\{, t \in \{0, T\{$, with a recurrent network of

$N \geq N_0$ units of range R. This could be, e.g. the Sierpinski sequence[11], which is deterministic aperiodic, and a function of the previous and $O(\sqrt{t})$ previous samples at time t, thus with long term dependency[12], or an unpredictable sequence without any algorithm to generate it, unless the copy of all sample (i.e., with a maximal Kolmogorov complexity).

On one hand, $O(N_0)$ independent linear recurrent units of range $R = T$, solves the problem of generating an exact sequence of $N_0\,T$ samples, in closed form[13]. This solution requires a very large recurrent range, and the numerical precision is limited by the fact errors accumulate along the recurrent calculation.

On the other hand, feed-forward units of range $R = 1$ solve explicitly the problem using clock units and readout, with $O(N_0\,T)$ weights. This requires no more than $N_0 + T$ units considering binary information[14], and no more that $N_0 + 1$ units if the numerical precision is sufficient[15]. In both cases, the weight estimation reduces to a simple readout estimation (i.e., the linear estimation of output weights, given the predefined hidden unit activity). A step further,

---

[11] This corresponds to the Sierpinski triangle read from left to right and from top to down in sequence.

[12] The Sierpinski sequence is generated by recurrent equations of the form:

$$\begin{array}{rcll} x_0(t) & = & -1 + 2\,(x_1(t) \bmod 2) & x_0(t) \in \{-1, 1\} \\ x_1(t) & = & 1 + \delta_{0<k_t<l_t<t}\,(x_1(t-l_t) + x_1(t-l_t-1) - 1) & \text{Pascal triangle sequence} \\ l_t & = & l_{t-1} + \delta_{k_{t-1}=0} & l_t = O(\sqrt{t}) \\ k_t & = & \delta_{k_{t-1}=l_{t-1}}\,(k_{t-1}+1) & 0 \leq k_t < l_t \end{array}$$

[13] **Long range sequence generation.** Let us consider units of the form:

$$x_n(t) = \textstyle\sum_{d=1}^{d=T-1} W_{nd}\,x_n(t-d) + W_{n0},$$

thus with $N_0\,T$ weights. Since $x_n(t) = 0, t < 0$, providing $\bar{o}_n(1) \neq 0$, we immediately obtain $W_{n0} = \bar{o}_n(1)$ and for $d > 0$:

$$W_{nk} = (\bar{o}_n(k+1) - W_{n0} - \textstyle\sum_{d=1}^{d=k-1} W_{nd}\,\bar{o}_n(t-d))/\bar{o}_n(1),$$

thus a closed-form solution. If $\bar{o}_n(1) = 0$ we simply have to generate the sequence, say, $\bar{o}'_n(t) = \bar{o}_n(t) + 1$ and add a second unit of the form $x_n(t) = 1 + x'_n(t)$, using now $2\,N_0$ nodes.

[14] **Long sequence generation with delay lines.** Let us consider N_0 readout units and T clock units of the form:

$$\begin{array}{rcll} x_{n_0}(t) & = & \sum_{n=0}^{T-1} \bar{o}_{n_0}(n+1)\,H(x_{N_0+n}(t)-1) & 0 \leq n_0 < N_0 \\ x_{N_0}(t) & = & W_0\,x_{N_0}(t-1) + W_2\,(1 - H(x_{N_0}(t))) & \\ x_{N_0+n}(t) & = & x_{N_0+n-1}(t-1) & 0 < n < T \end{array}$$

considering $H(0) = 1/2$ by prolongation. As soon as $W_2 > 2, W_0 < 2/W_2$ we easily obtain:

$$\mathbf{x}_{N_0} = \{0, W_2/2 > 1, \cdots, (W_0)^{t-1}\,W_2/2 < 1, \cdots\}$$

thus $H(x_{N_0}(t)-1) = \delta_{t=1}$ and $H(x_{N_0+n}(t)-1) = \delta_{t=n-1}$. In words, we generate T clock signals with one non-zero value at time $n = t+1$, allowing to generate the desired sequence combining these signals. The unit of index N_0 is a recurrent unit generating a trigger signal, all other units being feed-forward units. If we now consider not Heaviside profiles, but sigmoid $h(u) = \left(1 + e^{-4*u}\right)^{-1}$, the previous system of equation is going to generate a temporal partition of unity. More precisely we obtain, with say $W_2 = 2$ and $W_0 = 1/4$:

$$\mathbf{x}_{N_0} = \{0, 1, u_t < 1, \cdots\}$$

while numerically $u_t \to u_\infty \simeq 0.419$, the equation fixed point. Since $\mathbf{x}_{N_0+n}$ are simple shifts of $\mathbf{x}_{N_0}$, the clock units obviously span the output signal space and output units can easily linearly adjust there related combination to obtain the desired values.

[15] **Long sequence generation with a range unit.** We simply consider units of the form:

$$\begin{array}{rcll} x_{n_0}(t) & = & \sum_{n=0}^{T-1} \bar{o}_{n_0}(n) & (H(x_{N_0}(t) - (n-1)) \\ & & & -H(x_{N_0}(t) - n)) \quad = \{\cdots, \bar{o}_{n_0}(t), \cdots\} \quad 0 \leq n_0 < N_0 \\ x_{N_0}(t) & = & x_0(t-1) + 1 & = \{\cdots, t, \cdots\} \end{array}$$

where $H(x_{N_0}(t) - (n-1)) - H(x_{N_0}(t) - n) = \delta_{n=t}$.

considering $N = O(\sqrt{N_0\,T/R})$ linear or NLN units of range R, as developed in [34] in a special case, we may generate a solution in the general case[16]. The number of required unit can not be higher than $N_0 + T$ as discussed previously.

The generation of periodic signal of period T, is a very similar problem, as studied in [34], for $N = N_0$. In a nutshell, we simply must add equations such that $\mathbf{x}(T) = \mathbf{x}(0)$ to guaranty the periodicity.

From this discussion, we would like to point out several remarks:
- the complexity of signal generation problem highly depends on the kind of "allowed units" and reduces to a trivial problem as soon as clock-like signals are used;
- there always exist a $R = 1$ network of at most $N_0 + T$ units that exactly solves the problem, so that this is still an reverse engineering problem;
- a linear network, NLN network or AIF network can generate such a sequence in the general case;
- the use of BMS (a.k.o. spiking neurons) ease the problem statement but does not change qualitatively the solution.

Long term non-linear transform

In many experiment a sequence of the form:

time :	0	1				T
input:	a	b	$*$	$\cdots$	$*$	$*$
output:	$*$	$*$	$*$	$\cdots$	$*$	$a\,b$

where a and b are variable input, $*$ are random distractors and $a\,b$ the desired delayed output (here a product, but it could be another calculation such as a XOR[17]). Such setup combines several non-trivial aspects, long short term memory, distractor robustness, and operation which may not explicitly hardwired in the network, presently a product. The LSTM approach was shown to be particularly efficient for such computation, because of the notion of "memory carousel". The explicit implementation of such a mechanism is however elemen-

[16] **Long sequence generation with fully connected network.** Considering the linear network system:

$$x_n(t) = \sum_{m=1}^{m=N} W_{nmr} \sum_{r=1}^{r=R} x_m(t-r) + W_{n0},$$

with $0 \leq n_0 < N_0$ output units and $N_0 \leq n < N$ hidden units, using vectorial notations, with the shift operator $\mathcal{S}$ defined as $\mathcal{S}\mathbf{x}(t-1) = \mathbf{x}(t)$, we obtain:

$$\mathcal{S}^{\begin{smallmatrix} N_0\,T \updownarrow \\ (N-N_0)\,T \updownarrow \end{smallmatrix}} \begin{pmatrix} \bar{\mathbf{o}} \\ \bar{\mathbf{x}} \end{pmatrix} = \mathbf{W} \begin{pmatrix} \bar{\mathbf{o}} \\ \bar{\mathbf{x}} \end{pmatrix} + W_0$$

where $\bar{\mathbf{o}}$ are the desired output. It is a bi-linear system of $N\,T$ equations in $N^2\,R + N$ independent unknowns, i.e., the weights, while the $(N - N_0)\,T$ hidden values are entirely specified as soon as the weights are given. In terms of number of degree of freedom we must have $N^2\,R + N > N_0\,T$ for this algebraic system of equation to have a solution in the general case.

[17] **Defining the xor function.** Using the same method as before, with $W_1 > 1$:

$$\begin{aligned} x_\bullet(t) &= W_1\,H(x_a(t-1)-1) + W_1\,H(x_b(t-1)-1) + -2\,W_1\,H(x_o(t)-1) \\ x_o(t) &= x_a(t-1) + x_b(t-1) - 1 \end{aligned}$$

allows us to obtain $H(x_\bullet(t)-1) = H(x_a(t-1)-1)$ xor $H(x_b(t-1)-1)$.

tary[18] while several variants of long-term mechanism can easily be defined, such a flip-flop[19]

References

[1] D.J. Amit. *Modeling brain function—the world of attractor neural networks*. Cambridge University Press, New York, NY, USA, 1989.

[2] K.J. Astrom. Theory and application of adaptive control: a survey. *Automatica*, 19:471–486, 1983.

[3] David Balduzzi and Muhammad Ghifary. Strongly-typed recurrent neural networks. *CoRR*, abs/1602.02218, 2016.

[4] Yoshua Bengio. *Learning Deep Architectures for AI*. Now Publishers Inc, Hanover, Mass., October 2009. 02705.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2012.

[6] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

[7] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arxiv:1502.0415*, page 10, feb 2016.

[8] N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16:1413–1436, 2004.

[9] B. Cessac. A View of Neural Networks as Dynamical Systems. *International Journal of Bifurcation and Chaos*, 20(06):1585–1629, June 2010. 00020.

[18] **Long term computation.** One solution writes:

$$\begin{array}{rcl} o_0(t) & = & H(c_T(t)-1)\,x_a(t)\,x_b(t) + H(1-c_T(t))\,i(t) \\ x_a(t) & = & H(1-c_0(t))\,x_a(t-1) + H(c_0(t)-1)\,i(t) \\ x_b(t) & = & H(1-c_1(t))\,x_b(t-1) + H(c_1(t)-1)\,i(t) \\ c_\tau(t) & = & W_0\,c_\tau(t-1) + (\frac{1}{\tau+1/2} + (1-W_0)\,c_\tau(t-1))\,(1-H(c_\tau(t))) \end{array}$$

for some value $W_0 < 3/4$. It is easy to verify that:
1/ $\mathbf{c}_\tau = \{0, \underbrace{t/(\tau+1/2)}_{t<\tau}, \underbrace{(\tau+1)/(\tau+1/2)}_{t=\tau}, \underbrace{W_0^{t-\tau}\,(\tau+1)/(\tau+1/2)}_{\tau<t}\}$ so that $H(c_\tau(t)-1) = \delta_{t=\tau}$ is a clock signal.
2/ $x_a(t)$ "opens" the memory at time $t=0$, and stores the previous value otherwise, since $H(1-z) = (1-H(z-1))$ is simply a "negation", with a similar behavior for $x_b(t)$.
3/ $o_0(t)$ simply mirror the input until $t=T$, where the expected result is output.

[19] **Defining a flip-flop latch.** Let us defined a SR-latch (i.e., a flip-flop) with:

$$z(t) \quad = \quad W_1\,H(z(t-1)-1) + W_1\,H(i_1(t)-1) - W_1\,H(i_0(t)-1)$$

with $1 < W_1$, considering the binary signal $H(z(t-1)-1)$ and yielding the following behavior:
- *R-state*: If $i_0(t) < 1$ and $i_1(t) < 1$ (no-input) and $z(t-1) < 1$, then $z(t) = 0 < 1$, the reset state is maintained.
- *S-state*: If $i_0(t) < 1$ and $i_1(t) < 1$ (no-input) and $z(t-1) > 1$, then $z(t) = W_1 > 1$, the set state is maintained.
- *R-S transition*: If $i_0(t) < 1$ and $i_1(t) > 1$ and $z(t-1) < 1$, then $z(t) = W_1 > 1$, flipping to a set state; if it was already in the set state, we still have $z(t) = 2\,W_1 > 1$.
- *S-R transition*:If $i_0(t) > 1$ and $i_1(t) < 1$ and $z(t-1) > 1$, then $z(t) = W_1 - W_1 < 1$, flopping to a reset state; f it was already in a reset state, we still have $z(t) = -W_1 < 1$.
- *no instability*: $i_0(t) > 1$ and $i_1(t) > 1$ contrary to a standard digital RS-latch we simply have $z(t) = z(t-1)$ providing it was in set of reset state, without any meta-stability. If now we consider a sigmoid instead of a step function, [A COMPLETER]

[10] Bruno Cessac. A discrete time neural network model with spiking neurons. Rigorous results on the spontaneous dynamics. *Journal of Mathematical Biology*, 56(3):311–345, 2008. 00004 56 pages, 1 Figure, to appear in Journal of Mathematical Biology.

[11] Bruno Cessac, Hélène Paugam-Moisy, and Thierry Viéville. Overview of facts and issues about neural coding by spikes. *J. Physiol. Paris*, 104(1-2):5–18, February 2010.

[12] Bruno Cessac, Rodrigo Salas, and Thierry Viville. Using event-based metric for event-based neural network weight adjustment. page 18 pp. Louvain-La-Neuve : I6doc.com, April 2012. 00000.

[13] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[14] Yann Le Cun. *A Theoretical Framework for Back-Propagation*. 1988.

[15] Li Deng. Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing*, 7(3-4):197–387, 2014. 00003.

[16] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[17] Thalita F. Drumond, Thierry Viéville, and Frédéric Alexandre Alexandre. Not-so-big data deep learning: a review. 2017. in preparation.

[18] Thalita F. Drumond, Thierry Viéville, and Frdric Alexandre. From shortcuts to architecture optimization in deep-learning. 2017.

[19] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, August 2013. 00578.

[20] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research*, 4:933–969, 2003.

[21] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *J. Mach. Learn. Res.*, 3:115–143, March 2003.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

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

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[26] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Sciences, USA*, 79:2554–2558, 1982.

[27] Johan Hstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, June 1991. 00000.

[28] H. Jaeger. Adaptive nonlinear system identification with Echo State Networks. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*2002, Advances in Neural Information Processing Systems*, volume 15, pages 593–600. MIT Press, 2003.

[29] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. *Proceedings of the 8th Annular Conference Cognitive Science Society*, pages 531–546, 1986.

[30] Robert Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323 – 334, 2007. Echo State Networks and Liquid State Machines.

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

[32] James Martens and Ilya Sutskever. Learning Recurrent Neural Networks with Hessian-Free Optimization.

[33] Hélène Paugam-Moisy, Régis Martinez, and Samy Bengio. Delay learning and polychronization for reservoir computing. *Neurocomputing*, 71:1143–1158, 2008.

[34] Horacio Rostro-Gonzalez, Bruno Cessac, and Thierry Viéville. Exact spike-train reproduction with a neural network model. *Journal of Computational Neuroscience*, 2010. submitted.

[35] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[36] S. Murray Sherman and R. W. Guillery. The role of the thalamus in the flow of information to the cortex. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 357(1428):1695–1708, December 2002.

[37] Hava T. Siegelmann and Eduardo D. Sontag. Turing Computability With Neural Nets. 1991.

[38] Meropi Topalidou, Arthur Leblois, Thomas Boraud, and Nicolas P Rougier. A long journey into reproducible computational neuroscience. *Frontiers in Computational Neuroscience*, 9:30, 2015.

[39] Joel A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory*, 50:2231–2242, 2004.

[40] Joel A. Tropp. Just relax: Convex programming methods for subset selection and sparse approximation. Technical report, Texas Institute for Computational Engineering and Sciences, 2004.

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

[42] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. nov 2016.