Recurrent neural network weight estimation through backward tuning

Thierry Viéville , Xavier Hinaut , Thalita F. Drumond , Frédéric Alexandre

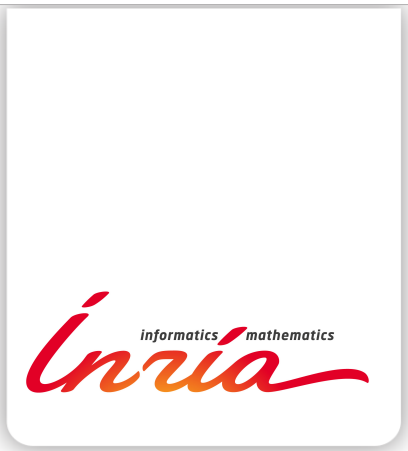
RESEARCH REPORT

N° 9100

October 2017

Project-Team Mnemosyne

ISSN 0249-6399 ISRN INRIA/RR--9100--FR+ENG



Recurrent neural network weight estimation through backward tuning

* Thierry Viéville *, Xavier Hinaut *, Thalita F. Drumond *, Frédéric Alexandre *

Project-Team Mnemosyne

Research Report n° 9100 — October 2017 — 50 pages

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 backward-tuning numerical stabilization 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.

Key-words: recurrent network, machine learning, backward tuning

* Mnemosyne team, INRIA Bordeaux

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération
Bâtiment A 29
33405 Talence Cedex

Estimation des poids d'un réseau récurrent par ajustement rétroactif

Résumé : Nous considérons une formulation alternative de l'estimation du poids dans les réseaux récurrents, proposant une notation integrant une grande quantité d'unités de réseau récurrentes qui aide à formuler ce problème d'estimation. Réutilisant un «bon vieux» principe de la théorie du contrôle, amélioré ici à l'aide d'une heuristique de stabilisation numérique rétroactive, nous obtenons une estimation distribuée du 2ème ordre, numériquement stable et plutôt efficace, sans aucun méta-paramètre à ajuster. La relation avec les techniques existantes est discutée à chaque étape. La méthode proposée est validée en utilisant des tâches d'ingénierie inverse.

Mots-clés : réseaux récurrents, apprentissage automatique, ajustement rétroactif

1 Introduction

Artificial neural networks can be considered as discrete-time dynamical systems, performing input-output computation, at the higher level of generality [49]. The computation is defined by the adjustment of the network connection weights and related parameters[1] In fact, only specific feed-forward or recurrent architectures are considered in practice, because of network parameters estimation, as reviewed now.

In the artificial neural network literature, feed-forward networks parameter learning is a rather 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 [2], though it has been shown at the theoretical and empirical levels that "shallow" architectures are inefficient for representing complex functions [44, 7], or at the cost of huge network sizes as in, e.g., extreme learning [31].

Deep-networks are specific feed-forward architectures [7] which can have very impressive performances, e.g. [22]. The key idea [32] 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 input/output function compact representation (in the sense of [32], i.e., as a hierachical composition of local 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 [5] for an introduction). This problem is highly reduced with specific architectures such as CNN [36].

It also remains restrictive by the fact that the architecture is mainly a pipeline including some parallel tracks or short-cuts, 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). Starting with LeNet-5 [36], different successful architectures in term of performance have been proposed (e.g., AlexNet[35], ZF net [61], Overfeat [47], VGG [50], GoogLeNet [53], Inspection [52], residual nets [26]).

In the brain, more general architectures exist (e.g. with shortcuts between deeper and lower layers, as it happens in the visual system regarding the thalamus [48]) 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, and the need of huge architecture

[1]Other network parameters include the unit leak, intrisic plasticity, parameters of the non-linearity (or activation function). However, in this paper we are going to use a notation allowing us to consider all these parameters as connection weights for an extended set of state variables.

in order to obtain high performances. This is the origin of the present work.

Feed-forward networks are obviously far from the computational capacity of recurrent networks [25, 46, 10]. Therefore, specific multi-layer architectures with 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 [30], or its randomized version as a Boltzman machine, was very specific. For such specific networks, such as bidirectional associative memory [1], specific learning methods apply. Further solutions include Jordan's network [34], Elman's Networks [19], Long short term memory (LSTM) by Hochreiter and Schmidhuber [28]. This latter architecture being very performant [46].

Another track is to consider recurrent networks with a "reservoir" of reccurent units but without explicit weight adjustement [58]. 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 [33] and Liquid State Machines [38], are called "reservoir computing" (see [58] for unification of reservoir computing methods at the experimental level), while extreme learning is based on a closed idea [31]. In such architectures the recurrent weigts of hidden units are not explicitly learned, but 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) [42]. It appears that reservoir computing yields good results [58], but without over-passing recent deep-layer architecture performances [18].

The general problem of learning recurrent neural networks has also been widely addressed as reviewed in [17] for 90's studies and in [39] for recent advances, and methods exist far beyond basic methods such as back-propagation through time, but is still not a well-solved problem.

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, and only on small architectures since we precisely target being able to solve complex computational tasks with reasonable architectures, in order the parameters to be learnable on not so big data [20]. As a consequence, learning issues (e.g., boosting

[23]) are not within the scope of this paper: Neither representation learning [6], nor other complex issues [25] are considered, this contribution being only an alternate tool for variational weight optimization. See [20] for a recent discussion on such issues.

We are also not going to consider biological plausibility in the sense of [8], but will show that the proposed method is compliant with several distributed biological constraints or computational properties: local weight adjustment, backward error propagation, Hebbian like adjustment rules. A more rigorous 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 makes explicit how this notation applies to most of the usual frameworks, while Appendix B compares the method with related recurrent weights estimation methods. We then address the estimation problems and introduce the proposed modified solution, while Appendix C further discuss how it can be used for several estimation problems. In the subsequent section the method is implemented and numerically evaluated. Finally, Appendix D illustrates how certain estimation problem reduce to trivial computation problems, given a suitable units and architecture, while Appendix E reviews how statistical problems can be reduced to an estimation problem compatible with our framework.

This is a short paper with a new proposal for weight estimation, but in link with quite a lot of other issues in the field. This is the reason why the core of the paper is short while several appendices are added.

2 Problem position

Notations. Vectors and matrix are written in bold, only basic linear algebra is used. For instance, $x_n(t)$ stands for the value of the n-th node at time t, $\mathbf{x}_n$ for the whole values of the node along time, $\mathbf{x}(t)$ the whole values of the nodes at time t and $\mathbf{x}$ all network values.

The Heaviside function writes $H(u)$ (considering $H(0) = 1/2$) and the sign function writes $sg(u) = 2\,H(u) - 1$:

$$H(u) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } u > 0 \\ 1/2 & \text{if } u = 0 \\ 0 & \text{if } u < 0. \end{cases}$$

Partial derivatives are written in compact form, e.g., $\partial_{x_n(t)}\,\mathbf{f}(\mathbf{x})$ means $\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n(t)}$, while $\partial_{x_n(t)\,x_{n'}(t')}\,\mathbf{f}(\mathbf{x})$ means $\frac{\partial^2 \mathbf{f}(\mathbf{x})}{\partial x_n(t)\,\partial x_{n'}(t')}$.

The notation $\delta_{\mathcal{P}}$ stands for 1 is the property $\mathcal{P}$ is true and 0 otherwise (e.g., $\delta_{2>1} = 1$).

Other notations are made explicit as soon as used.

A general recurrent architecture.

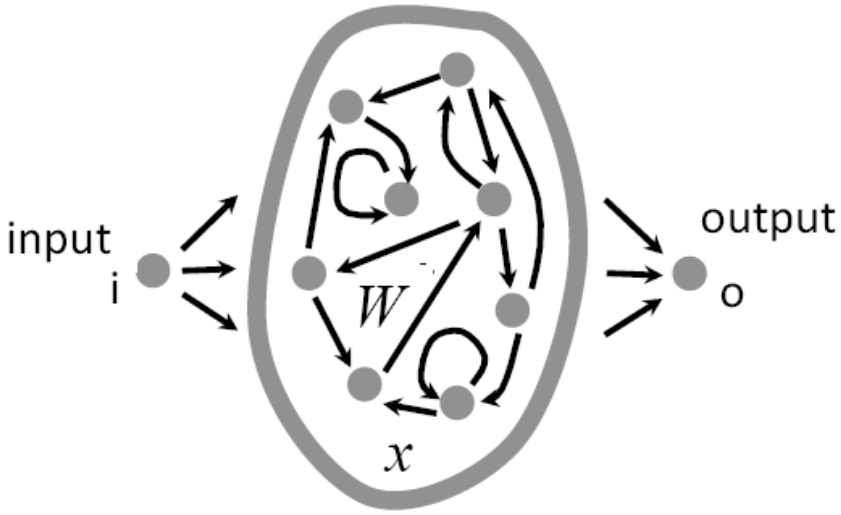


Figure 1: A General recurrent architecture maps a vectorial input sequence $\mathbf{i}(t)$ onto an output $\mathbf{o}(t)$, via an internal state $\mathbf{x}(t)$ of hidden units. It is parameterized by a recurrent parameter matrix $\mathbf{W}$. The dynamics is defined by the network recurrent equations.

As schematized in figure 1, we consider a recurrent network

$$x_n(t) = \Phi_{nt}\left(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots\right)$$

with nodes of the form:

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

i.e., defined as a linear combination of some kernel $\Phi_{ndt}()$. We show in Appendix A that this a very general form (e.g., including when considering the adjustment of unit parameters that are not connection weights).

More precisely, equation (1) elements define:
- 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).
- while $N_0 \leq N$ 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. Here, 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 appendix A for all examples, $0 \leq \partial_{x_{n'}(t')}\Phi_{ndt}() \leq 1$ (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[2], 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[3] for weight estimation to be numerically stable.

The key point here, is that some state variables $\mathbf{x}_n$ are additional intermediate internal variables in order the weight estimation to be a simple linear problem

[2]It is an easy task to introduce non-zero initial conditions as additional network parameter to learn, or consider then as a transient additional random input.

[3]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) [10]. In practice, not only contracting systems can be considered, as 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 [9, 37], which fits with the present requirement.

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. This is made explicit in appendix A.

2. The weight estimation problem writes in a quite simple way, with this reformulation. This is discussed now.

3 Recurrent weight estimation

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

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

for adjustable network parameters or weights $\mathbf{W}$, given state values $\mathbf{x}$ and auxilary variables ε, writing:

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

where $\tilde{x}_n(t)$ is a shortcut for equation (1):

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

while ε_{nt} are Lagrange multipliers, and in most of the cases[4] we use:

$$\rho(\cdots, x_n(t), \cdots) \stackrel{\text{def}}{=} \sum_{nt} \rho_{nt}(x_n(t)).$$

Here $\rho()$ is a cost-function (acting both as supervised or unsupervised variational term) and $\mathcal{R}(\mathbf{W})$ some regularization term, as made explicit in Appendix C. 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 can be used to get the approximate desired output, yield sparse estimation, reduce artifact influence, obtain activity orthogonality, etc (see Appendix C for details).

In a nutshell, $\rho()$ and $\mathcal{R}$ allows one to specify the estimation problem, as a function of the unknows $\mathbf{W}$, $\mathbf{x}$ and ε. Stating the estimation this way, leads us to a simplified form of the Pontryagin's minimum principle, well-known in control

[4]More precisely, here, in the deterministic case, a simple additive criterion is used, while this is not the case for statistical criterion, as further discussed in appendix C and E.

theory [3], and reviwed in the next section. In short, the effective related solution is derived from the normal equations of the proposed criterion.

This formulation is not new and has been formalized, by, e.g. [17]. 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) proposing a 2nd order local estimation mechanism. The relation with other recurrent weights estimation methods is discussed in Appendix B.

Applying standard derivations, the criterion gradient writes:

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

writing :

$$\left\{
\begin{array}{lcll}
\rho'_{nt} & \stackrel{\text{def}}{=} & \partial_{x_n(t)} \rho(\cdots, x_n(t), \cdots) & \\
\phi_{ndt} & \stackrel{\text{def}}{=} & \Phi_{ndt}(\cdots, x_{n'}(t'), \cdots, i_m(s), \cdots) & = \partial_{W_{nd}} \tilde{x}_n(t) \\
\beta_{nt}^{n't'} & \stackrel{\text{def}}{=} & \partial_{x_{n'}(t')} \phi_{n0t} + \sum_{d=1}^{D_n} W_{nd}\, \partial_{x_{n'}(t')} \phi_{ndt} & = \partial_{x_{n'}(t')} \tilde{x}_n(t)
\end{array}
\right.$$

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 $\partial_{\varepsilon_{nt}} \mathcal{L} = 0$ yields $x_n(t) = \tilde{x}_n(t)$. This simply means that $x_n(t)$ is iven by the network equation, i.e., equation (1). Since $\tilde{x}_n(t)$ 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 $\partial_{\varepsilon_{nt}} \mathcal{L} = 0$ 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 $\partial_{x_{n'}(t')} \mathcal{L} = 0$ 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 $t = 0$:

$$\varepsilon_{n't'} = \rho'_{n't'} + \sum_{nt} \beta_{nt}^{n't'} \, \varepsilon_{nt}. \tag{3}$$

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.

This makes the key difference with respect to usual approaches based on gradient back-propagation: Here the output error is back-propagated. This calculation may be recognized as a kind of back-propagation, but it is mathematically different. This method is thus quite different from back-propagation-though-time recurrent network or other standard alternatives.

As mentioned by [17], $\beta_{nt}^{n't'}$ 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 $\partial_{\varepsilon_{nt}} \mathcal{L} = 0$ 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 2nd and 1st order weight adjustment quantities, discussed in the sequel).

Parameter interpretation. We obtain, from equation (3) after some algebra $\varepsilon_{n't'} = \sum_{nt} B_{n't'}^{nt} \, \rho'_{nt}$, with finite summations and for some quantities $B_{n't'}^{nt}$ (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 $\beta_{nt}^{n't'} = 0$, there is no dependency of $x_n(t)$ on $x_{n'}(t')$, i.e. no recurrent connection. 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 derivative. 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.

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 adjustment is a function of hidden activity in the past, the estimation must thus take future information into account in order to properly adapat.

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, considering only a bounded window-time may provide numerically relevant values for on-the-fly backward tuning.

Numerical stability. This back-propagation of tuning error, may suffer from the same curse than back-propagation of gradient, as reviewed in e.g., [29]: Either error explosion (if $|\beta_{nt}^{n't'}| > 1$), or error extinction (if $|\beta_{nt}^{n't'}| < 1$). Based on this remark, the key idea of LSTM [29] is to consider memory carousel (detailled in Appendix A) to guaranty $\left|\beta_{nt}^{n't'}\right| \simeq 1$ and thus a stable back-propagation for at least some recurrent link, but this means that the designer of the network architecture has to consider such predefined units, which is a strong constraint.

In our case, since all kernels are contracting with $\max|\partial_{x_{n'}(t')}\phi_{ndt}| = 1$ we are in a situation where the a-priory numerical conditioning is optimal. We also have the bound, writing $\beta_{max} \stackrel{\text{def}}{=} \max_{nt}|\beta_{nt}^{n't'}|$:

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

without any thinner inequality in the general case. This means that we "must" accept error potential explosion as soon as the weights values are not below one, which can not be a manageable constraint.

To avoid backward explosion or extinction, we are going to introduce another heuristic: We are going to *bias* the backward error given in equation (3). We define:

$$\varepsilon_{n't'} \simeq \rho'_{nt} + g\left(\sum_{nt} \beta_{nt}^{n't'}\, \varepsilon_{nt}\right), \tag{4}$$

considering a function $g(u)$, shown in Fig. 2. It is the identity function except for small vanishing values that are raised using a simple quadratic profile, an huge values saturated by an exponential profile, and providing a continuously derivable

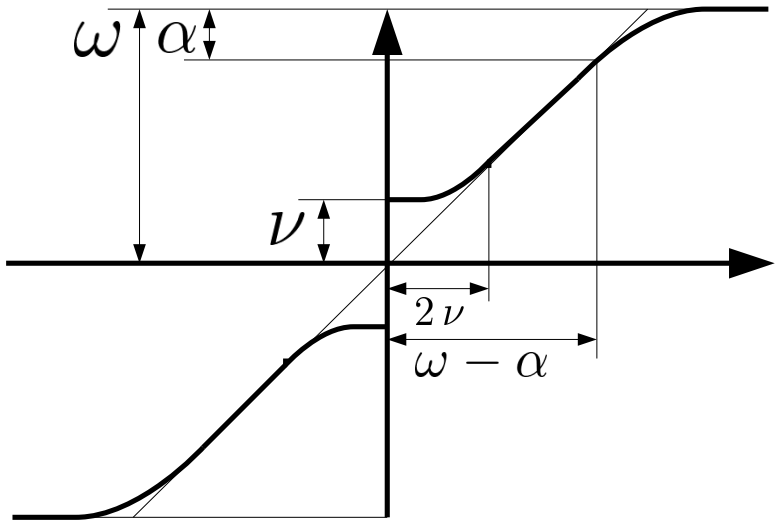


Figure 2: The backward guard profile, defined in (5), with a bias for tiny values and a saturation for huge values.

function. This design choice writes, for fixed meta-parameters ω, α, ν:

$$g(u) \stackrel{\text{def}}{=} sg(u) \begin{cases} \omega - \alpha\, e^{-\frac{|u|-\omega}{\alpha}-1} & \omega - \alpha \leq |u| \\ |u| & 2\,\nu \leq |u| \leq \omega - \alpha \\ \nu + \frac{u^2}{4\,\nu} & |u| \leq 2\,\nu, \end{cases} \tag{5}$$

where $sg()$ is the sign function. To fix these meta-parameters we consider the order of magnitude of the output error:

$$\bar{\rho'} \stackrel{\text{def}}{=} \frac{\sum_{nt,\rho'_{nt}\neq 0} \rho'_{nt}}{\sum_{nt,\rho'_{nt}\neq 0} 1},$$

and a reasonable choice to preserve the numerical conditioning is $\nu = 10^{-6}\,\bar{\rho'}$ and $\omega = 10^{6}\,\bar{\rho'}$, with e.g., $\alpha = 10^{-3}\,\omega$. They are very likely not to be adjusted because they only correspond to order of magnitude of numerical calculation. We have observed that using double precision floating numbers on a standard processor for such kind of calculations corresponds to such rough numbers.

The 2nd order unit weight adjustment

We now have to estimate the weights $\mathbf{W}$ and are left with the last normal equation $\partial_{W_{nd}} \mathcal{L} = 0$ which is not an explicit function of the weights. On track is to use the

gradient to minimize the criterion using a 1st order method, this is discussed in the next sub-section. Interesting enough is the fact that we can also propose a 2nd order method as made explicit and derived now. In other words, we reintroduce a linear estimation of the weights assuming that the criterion is locally quadratic.

We thus propose to use the following 2nd order weight adjustment:

$$\sum_{n''} b_{n'',\,d} = \sum_{n''} \sum_{d'=1}^{D_n} A_{n'',\,d\,d'}\, W_{n''d'} \tag{6}$$

writing, for some κ_{nt}:

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

where:
- $\hat{x}_n(t)$ is best present estimate of $x_n(t)$,
- $\hat{\mathbf{W}}$ is the best estimate of $\mathbf{W}$ at the present step.

This allows us to obtain a new weight value $\mathbf{W}$ solving a linear system of equation for each unit and the closest solution[5] with respect to $\hat{\mathbf{W}}$ is considered.

The derivation[6].

[5]**Minimal distance pseudo-inverse.** We consider:

$$\min_{\mathbf{W}} ||\mathbf{W} - \hat{\mathbf{W}}||, \mathbf{b} = \mathbf{A}\,\hat{\mathbf{W}}$$

which is directly obtained using the singular value decomposition of the symmetric matrix $\mathbf{A} = \mathbf{U}\,\mathbf{S}\,\mathbf{U}^T$:

$$\mathbf{W} = \hat{\mathbf{W}} + \mathbf{A}^\dagger\,(\mathbf{b} - \mathbf{A}\,\hat{\mathbf{W}}),$$

where $\mathbf{A}^\dagger$ is the pseudo-inverse of $\mathbf{A}$.

[6]**Deriving the 2nd order adjustment form.** Let us omit the $\mathcal{R}()$ term and avoid considering weight sharing in this derivation, in order to lighten the notations. The complete derivation would have obviously led to similar results.

Given a desired value estimate $\hat{x}_n(t)$, without loss of generality we can write, for some general quantity $\kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon)$:

$$\mathcal{L}(\mathbf{W}, \mathbf{x}, \varepsilon) = \sum_{nt} \frac{\kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon)}{2}\ (\tilde{x}_n(t) - \hat{x}_n(t))^2\,,$$

yielding for $\partial_{W_{nd}} \mathcal{L}$:

$\sum_t \kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon)\, \phi_{ndt}\, (\tilde{x}_n(t) - \hat{x}_n(t)) + \sum_t \partial_{W_{nd}} \kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon)\, (\tilde{x}_n(t) - \hat{x}_n(t))^2/2 = \sum_t \phi_{ndt}\, \varepsilon_{nt}$.

For a simple least-square criterion, $\kappa_{nt} \in \{0, 1\}$ depending on the fact that the desired output $\bar{o}_n(t)$ is defined or not, and it is straight-forward to verify in this particular case that the proposed 2nd order weight adjustment 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, ϕ_{ndt} is also a function of both the network unknown output and hidden node values.

(i) In our case the output and backward error estimation is ε_{nt}, i.e., we can set $\hat{x}_n(t) \stackrel{\text{def}}{=} \tilde{x}_n(t) - \varepsilon_{nt}$ as a corrected value of the last estimate $\tilde{x}_n(t)$. Given this hypothesis, it is obvious to verify that $\kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon) = 1$ verifies the equation.

(ii) A step further, for a general value $\hat{x}_n(t)$, a sufficient condition now writes:

$$\kappa_{nt}(\mathbf{W}, \mathbf{x}, \varepsilon) = 2\,\varepsilon_{nt}/(\tilde{x}_n(t) - \hat{x}_n(t)),$$

More sophisticated estimated values can be considered[7].

The weight adjustment is local to each unit, providing a true distributed mechanism (unless if weight sharing is considered, because weights from different units are to be estimated together using the proposed equations). This corresponds to a 2nd order minimization scheme. Each step requires $O(N(DT + D^3)$- operation, solving a linear system of equations. The $O(N\,D^3)$ is critical if the network connectivity D is high, and this does not depend on the linear system resolution method (e.g., SVD or Cholesky decomposition). The implemented method stands on the singular-value-decomposition of the matrices $\mathbf{A}_n$.

This offers an alternative to 2nd order adjustment methods such as [39] or other methods reviewed in [25].

In fact, a standard 2nd order adjustment can be derived in closed form[8], directly from the 2nd order criterion derivatives. It is not used here because the computation involves not only the local node parameters, but also the connected node parameters, and the calculation is rather heavy.

---

thus $k_n t$ is proportional to ε_{nt} and must decrease with the prediction error increase. Considering the case (i), it is straightforward to verify that if κ_{nt} is constant, and assuming $\hat{x}_n(t)$ is fixed, we obtain the related least-square linear equations given in (6).

[7]**Improving the best estimate of the state value.** The best estimate of the state value $\hat{x}_n(t)$ given output values $\bar{o}_{n_0}(t)$ is not obtained by the simulation since $\hat{x}_{n_0}(t) \neq \bar{o}_{n_0}(t)$.

If we consider the value obtained by simulation (i.e., the $\tilde{x}_n(t)$ values), corrected by the error estimate thus $\hat{x}_n(t) = \tilde{x}_n(t) - \varepsilon_{nt}$, for a least-square criterion, it is easy to verify that this yields $\hat{x}_{n_0}(t) = \bar{o}_{n_0}(t)$.

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 [45], whereas a bias in the estimation is otherwise expected, since hidden units simulated values and output values are not coherent.

A step further, we propose to retro-propagate the output value through the recurrent network, given weights values $\hat{\mathbf{W}}$, i.e., estimate:

$\hat{x}_n(t) = \arg\min_{x_n(t)} \mathcal{M}, \quad \mathcal{M} = \frac{1}{2} \sum_{n,n \geq N_0\, t} (x_n(t) - \Phi_{nt}\,(\cdots, x_{n'}(t'), \cdots))^2, x_{n_0}(t) = \bar{o}_n(t)$

in words find the state values for which the simulation errors yielding the desired output are minimal. Considering the normal equation $\partial_{x_{n'}(t')} \mathcal{M} = 0$ we obtain the recurrent equation:

$$\hat{x}^k_{n'}(t') = \begin{cases} \bar{o}_{n'}(t') & n' < N_0 \\ \Phi_{nt}\left(\cdots, \hat{x}^{k-1}_{n'}(t'), \cdots\right) - \sum_{nt} \beta^{n't'}_{nt}\, (\hat{x}^{k-1}_{n}(t) - \Phi_{nt}\left(\cdots, \hat{x}^{k-1}_{n'}(t'), \cdots\right)) & N_0 \leq n', \end{cases}$$

i.e., the simulation value is corrected considering a backward propagation of the simulation error.

In fact, it is simple to verify that we implicitly solve a system of NT equations in NT unknowns, the numerical scheme allowing to converge to a solution closed to the simulation values. This has been numerically verified in the experimentation.

It has been implemented as an option in the software in order to help improving the convergence of the recurrent weight adjustment.

[8]**Calculating the standard 2nd order weight adjustment.** The criterion Hessian, omitting the regularization term and weight sharing to lighten the notations, writes:

The 1st order unit weight adjustment

The calculation of $\partial_{W_{nd}} \mathcal{L}$ allows us to propose a 1st order gradient descent adjustment of the weights, providing that $\partial_{\varepsilon_{nt}} \mathcal{L} = 0$ after network simulation and $\partial_{x_{n'}(t')} \mathcal{L} = 0$ after backward tuning.

It 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, unless weight sharing is considered. In that case, this 1st order unit weight adjustment is either to be done globally at the whole node set level, or locally for each unit, but with inter-unit weight adjustment, not discussed here.

A step further, we can enhance this method considering the so-called momentum gradient mechanism (based on a temporal averaging of the gradient values). To this end we consider:

$$\mathbf{g}_k(t) = (1 - 1/k)\, \mathbf{g}_k(t-1) + 1/k\, \partial_{W_{nd}} \mathcal{L}(t), \quad k \in \{1, 2, 4, 8, 16, 32\}$$

in words, a 1st order exponential filtering of the gradient value obtained at time t, and the algorithm is going to compare these 6 options and choose the one with a maximal criterion decrease (avoiding introducing a meta-parameter at this stage). Here we mainly would like to explore several direction of descent if the criterion is

$$\left\{ \begin{array}{rcll} \partial_{\varepsilon_{nt}\varepsilon_{n't'}} \mathcal{L} & = & 0 & \\ \partial_{x_{n'}(t')\varepsilon_{nt}} \mathcal{L} & = & \beta_{nt}^{n't'} & \\ \partial_{W_{nd}\varepsilon_{n't'}} \mathcal{L} & = & \delta_{n=n'}\, \phi_{ndt'} & \\ \partial_{x_{n'}(t')x_{n''}(t'')} \mathcal{L} & = & H_{n't'n''t''}^{nt} & \stackrel{\text{def}}{=} \sum_{nt} \partial_{x_{n'}(t')x_{n''}(t'')} \left(\rho() + \phi_{n0t} + \sum_{d=1}^{D_n} W_{nd}\, \phi_{ndt} \right) \\ \partial_{W_{nd}x_{n'}(t')} \mathcal{L} & = & J_{n't'}^{nd} & \stackrel{\text{def}}{=} \sum_t \varepsilon_{nt}\, \partial_{x_{n'}(t')} \phi_{ndt} \\ \partial_{W_{nd}W_{n'd'}} \mathcal{L} & = & 0, & \end{array} \right.$$

writing $\beta_{nt}^{nt} \stackrel{\text{def}}{=} -1$.

The 1st remark is that $H_{n't'n''t''}^{nt}$ and $J_{n't'}^{nd}$ are not local to one node, whereas the summation involves all nodes connected to the given one. Furthermore if $\rho()$ is not a sum of local terms but a statistical criterion $H_{n't'n''t''}^{nt}$ is a function of the whole network.

Then the standard 2nd order scheme $0 \simeq \nabla \mathcal{L} + \nabla^2 \mathcal{L}\, \delta(\mathbf{W}, \mathbf{x}, \varepsilon)$ writes in our case where $\partial_{\varepsilon_{nt}} \mathcal{L} = \partial_{x_{n'}(t')} \mathcal{L} = 0$:

$$\left\{ \begin{array}{rcrcrcrcl} & & \sum_{n't'} \beta_{nt}^{n't'} & \delta x_{n'}(t') & + & \sum_d \phi_{ndt} & \delta W_{nd} & \simeq & 0_{nt} \\ \sum_{nt} \beta_{nt}^{n't'} & \delta\varepsilon_{nt} & + & \sum_{n''t''} H_{n't'n''t''}^{nt} & \delta x_{n''}(t'') & + & \sum_{nd} J_{n't'}^{nd} & \delta W_{nd} & \simeq & 0_{n't'} \\ \sum_t \phi_{ndt} & \delta\varepsilon_{nt} & + & \sum_{n't'} J_{n't'}^{nd} & \delta x_{n'}(t') & + & \sum_t \phi_{ndt}\, \varepsilon_{nt} & & \simeq & 0_{nd}, \end{array} \right.$$

and $\delta\varepsilon_{nt}$ and $\delta x_{n'}(t')$ can be eliminated in order to obtain a linear equation in δW_{nd}. This however requires the inversion the $\beta_{nt}^{n't'}$ matrix (and its transpose), which is a $O(NT \times NT)$ matrix, not necessarily sparse if the network is fully connected. We thus consider that the resulting calculation is too greedy to be performed at each step of the minimization.

not numerically regular.

This leads to a 1st order adjustment of the weights, i.e. it provides the direction for the weight variation, not its magnitude. In order to manage this issue we very simply automatically adjust a step meta-parameter υ^k, initialized to any reasonnable small value and:

- Calculates: $\tilde{W}_{nd} = \hat{W}_{nd} - \upsilon^k \, \mathbf{g}_k$.
- Performs a rough line-search minimization $\min_{\alpha^k} \mathcal{L}(\alpha^k \tilde{\mathbf{W}} + (1 - \alpha^k) \, \hat{\mathbf{W}})$ (here using the Brent-Dekker method with a 10^{-2} relative precision).
- Updates $\upsilon^k \leftarrow 2 \, \alpha^k \, \upsilon^k$.

In words we look for a weight value between both previous and new values that decreases the criterion, and set the new step value to twice the last optimal value. Each line-search step requires a simulation to compute $\mathcal{L}$.

This is a bit heavy, but it is only a fall-back of the 2nd order adjustment (e.g., for concave parts of the criterion). For the same reason, more sophisticated methods such as conjoint gradient methods (taking into account several subsequent gradient directions in order to infer an approximate 2nd order minimization method) have not been considered.

The complete weight adjustment

Collecting the previous steps the final iterative weight adjustment writes

-1- Perform a forward simulation and a backward tuning, calculating the 1st order gradient and 2nd order elements during the backward estimation.
-2.a- 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.

The 2nd order adjustment also uses a line search, because our experimental observation is that the 2nd order estimation tends to overestimate the local minimum. The 2nd order adjustment is not performed if the connectivity of the network is too high since it has a cubic cost.

Though the algorithm can be implemented in a complete distributed framework, in this preliminary study, the 2nd or 1st order adjustment is global, in order to limit the number of iteration on a simple sequential machine. The complete algorithmic structure is schematized in Fig. 3.

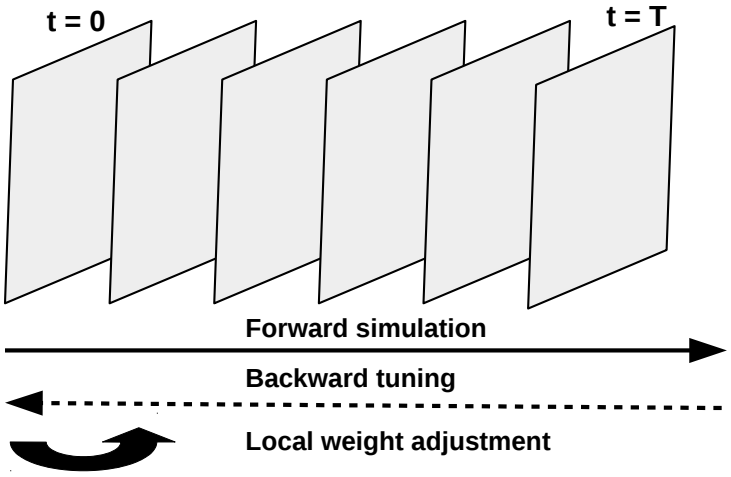


Figure 3: The algorithmic structure of the estimation algorithm: A forward simulation yields the current criterion value, while the backward tuning allows us to obtain the 2nd order and 1st order local weight adjustment elements. The algorithm can be implemented in a complete distributed framework.

4 Experimentation

In this experimental part we study the numerical stability and limit of the method considing toy benchmark problems. 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 we can adjust recurrent network weights.

Software implementation

In order to provide so called reproducible science [54], the code is implemented as a simple, highly modular, fully documented, open source, object oriented, easily forkable, self contained, middle-ware, and is 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. 4.

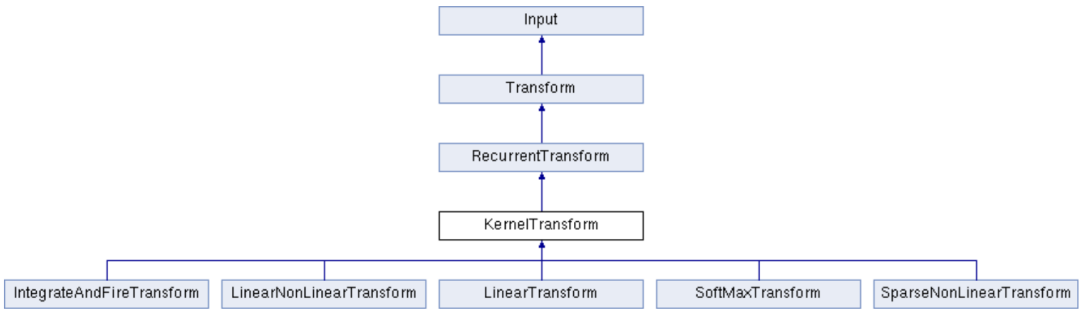


Figure 4: A view of the class-hierarchy: A `Input` simply provides a $x_n(t), n \in \{0, N\{, t \in \{0, T\{$ values, while a `Transform` provides such values given another `Input`, while other objects defined here derive from such an oversimple abstract class, and are precisely defined and discussed in Appendix A.

Regarding the estimations described in Appendix C, the `KernelSupervisedEstimator` class implements, quadratic estimation, bounded and unbounded robust estimation and Boolean estimation, while the `KernelObservableEstimator` class implements some basic stochastic models estimation.

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`, available in the present implementation) being straightforward, using e.g. `swig`.

The first experimental verification is that it is quite simple to define the main unit structures in Appendix section A from `KernelTransform` as claimed in the paper, see Fig 5 for an example with AIF nodes and the code source for the LNL `LinearNonLinearTransform` implementation and the SoftMax `SoftMaxTransform` implementation.

```
unsigned int getKernelDimension(unsigned int n) const
{
  return n < N ? 2 + N + input.getN() : 1;
}
double getKernelValue(unsigned int n, unsigned int d, double t) const
{
  if(n < N) {
    if(d == 0)
      return 0;
    if(d == 1)
      return (1 - zeta(get(n + N, t))) * get(n, t - 1);
    if(d == 2)
      return 1;
    d -= 3;
    if(d < N)
      return zeta(get(d + N, t));
    d -= N;
    if(d < input.getN())
      return input.get(d, t - 1);
    return 0;
  } else
    return d == 1 ? get(n - N, t - 1) - 1 : 0;
}
double getKernelDerivative(unsigned int n, unsigned int d, double t, unsigned int n_, double t_) const
{
  return n < N ?
             (d == 1 ?
              (n_ == n && t_ == t - 1 ? 1 - zeta(get(n + N, t)) :
               n_ == n + N && t_ == t ? -dzeta(get(n_, t)) * get(n, t - 1) : 0) :
              3 <= d && d < N + 3 && n_ == N + d - 3 && t_ == t ? dzeta(get(n_, t)) : 0) :
    d == 1 && n_ == n - N && t_ == t - 1 ? 1 : 0;
}
```

Figure 5: The implementation of the AIF node, translating equation (9) into the notation of equation (1) in the `IntegrateAndFireTransform` object.

Using reverse engineering

As being in a deterministic context, we are going to rely on a reverse engineering setup, in order to evaluate the performances and limit of the method. 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 the transform. This guaranties the existence of an exact solution.

How relevant is it to use such a reverse engineering setup ? On one hand, surprisingly enough perhaps, such networks (at least deep networks [62]) behave with the same order of magnitude of performances, the input being either "meaningful" or not, in the sense it represents data with a semantic or not. We thus can expect simple random input/output tests to be relevant estimation of performance, even for more semantic application. On the other hand, as developed in Appendix D,

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. The key point is thus to see if several kind of nodes can be adjusted with this mechanism. For these reasons we have considered the reverse engineering paradigm as a first test.

In most of the cases, they are several solutions (e.g., in a linear case, up to a permutation of the units, or some linear combination). We consider a root network of $\bar{N}$ units for a sequence of time T, for a $M = 1$ scalar random input, considering either L (for linear), LNL, AIF or SoftMax 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. This choice is related to the fact that the adjustment of the hidden units weights is the key challenge.

In this deterministic case, we observe the following parameters: number of steps to convergence, final precision criterion value, and we also fit an exponential decay curve[9] in order to estimate the decay time-constant and final criterion bias.

Examples of results for different kind of units are reported in Fig. 6, and two typical criterion decay curves are shown in Fig.7.

No surprise, the method converges in each case, while performances depend on the input and weight random draws. The reported result corresponds to the observed variability, as illustrated in Fig. 8. We have never observed run where the estimation fails.

A key point is that the convergence corresponds to an exponential decay profile, and the decay magnitude almost corresponds to the 2nd order adjustment, when the criterion is locally quadratic, while 1st order fall-back mechanism is mainly chosen by the algorithm in the other cases (e.g. concave criterion), again as expected. We only observed that the 2nd order adjustment may generates weights that can, mainly in the linear case, generates divergent sequences. Despite this caveat, the optimization algorithm recovers by reducing the weight variation amplitude, thus

[9] **Exponential decay fit.** The criterion value model to fit is of the form:

$$c(t) \stackrel{\text{def}}{=} \alpha\, e^{-t/\tau} + \beta.$$

The time decay τ is fitted in the least-square sense on $\log(c(t) - c(t-1)) = k - t/\tau$, for $k \stackrel{\text{def}}{=} \log(\alpha\,(1 - e^{1/\tau}))$ and the bias β is fitted, given τ, on $c(t) = (c(t) - c(t-1))/(e^{\frac{1}{\tau}} - 1) + \beta$. More precisely, the least-square problems write:

$$\min_{1/\tau,k} \sum_t^T \gamma^{T-t}\,(k - t/\tau - \log(c(t) - c(t-1)))^2,$$

for an exponential window of width $W = \frac{\log(1-r)}{\log(\gamma)}$ where r is the fraction of data average within this window (typically 90%), while the bias is estimated minimizing:

$$\min_\beta \sum_{t=1}^T \delta_{0<\hat{\beta}(t)<\min_t c(t)}\, \gamma^{T-t}\,(\beta - \hat{\beta}(t))^2, \quad \hat{\beta}(t) \stackrel{\text{def}}{=} c(t) - \frac{c(t-1)-c(t)}{e^{\frac{1}{\tau}}-1},$$

selecting only minimal values in order to guaranty a coherent estimation of this bias.

Node type	LinearNonLinearTransform					
Number of units	2	4	8	16	32	64
Number of Iterations	36	101	78	101	55	101
Minimal criterion value	9.3e-07	3.0e-06	1.0e-06	1.6e-06	9.1e-06	4.5e-06
Exponential decay time	24	23	88	37	20	98
Final bias interpolation	2.2e-08	2.6e-06	5.9e-07	1.6e-06	2.6e-06	4.5e-06
Node type	IntegrateAndFireTransform					
Number of units	2	4	8	16	32	64
Number of Iterations	101	101	101	101	101	101
Minimal criterion value	9.9e-06	4.7e-06	1.1e-06	3.5e-06	7.8e-06	1.4e-06
Exponential decay time	8	36	17	34	56	23
Final bias interpolation	9.1e-06	4.6e-06	1.1e-06	3.7e-06	7.9e-06	1.2e-06

Figure 6: Confirmation of convergence for different type of nodes and different small sized networks, considering random input and random weights for the root network, each number corresponds to one run. The iteration is stopped when the criterion is below 10^{-6}. But we can obtain precision down to 10^{-12} with the proposed implementation, in the linear case. Similar results are available for `SparseLinearNonLinearTransform` and `SoftMaxTransform` node types.

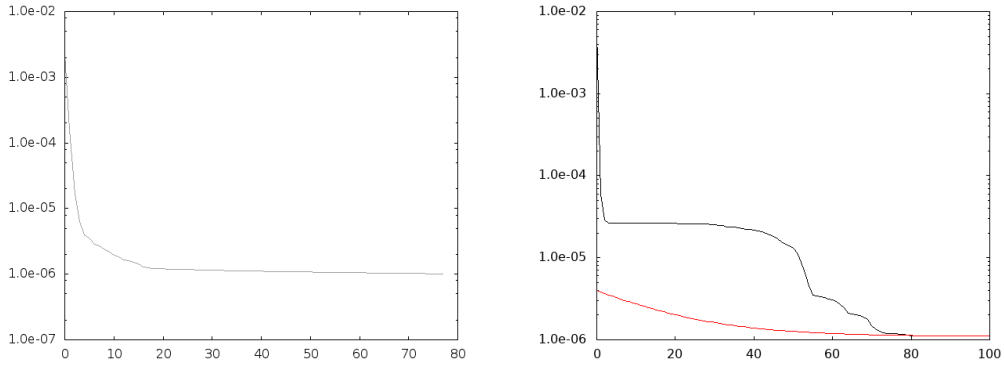


Figure 7: Two examples of criterion decay, here for (Left) a `LinearNonLinearTransform` and (Right) `IntegrateAndFireTransform`, with $N = 8$, in log-coordinates. The left curve is a "standard" curve with a strong decay and then a slow improve of precision. The right curve corresponds to a more erratic behavior with a strong decrease due to the 2nd order mechanism, followed by a "restart" of the optimization after a 1st order search thanks to the gradient momentum heuristic.

.

re-obtaining convergent simulation.

We also never observed backward tuning numerical explosion of extinction in all experiments, probably because of the numerical conditioning of the equation has been optimized, but this is to expected in larger scale experiments.

Node type	LinearNonLinearTransform						
Sample index	1	2	3	4	5	6	7
Number of Iterations	101	4	4	3	4	3	4
Minimal criterion value	1.9e-05	2.6e-06	2.0e-06	4.4e-06	1.3e-06	9.3e-06	6.2e-06

Figure 8: Variability in term of convergence for a `LinearNonLinearTransform`, with $N = 8$, for a standard relative cost of 10^{-5}, when varying the root network input/output sequence and/or the learning network initial weights draw. Some runs may take quite more time if the initial conditions are far from the solution, whereas we always observe convergence.

Using different criterion

A step further, we consider an approximate reverse-engineering input/output sequence, with either additive noise or some spurious outliers with large errors. We already know that as soon as the dynamic is sufficiently rich, even small errors accumulate and the solution exponentially diverges from the exact one. In such a case, two questions are raised.

On one hand, can a robust criterion "resist" to such noise or outliers ? We have tested this by considering both additive noise and outliers as reported in Fig. 9. And we have compared the use of several criterion, discussed in Appendix C: $\mathcal{L}^2$ criterion (i.e. least-square), reweighted $\mathcal{L}^1$ criterion (i.e. unbounded robust criterion), reweighted $\mathcal{L}^0$ criterion (i.e. unbounded robust criterion), standard $\mathcal{L}^1$ related criterion, and standard $\mathcal{L}^0$ biweight criterion. As expected, due to the optimization method that implicitly assumes the criterion to be locally quadratic, reweighted methods outperform usual ones. Furthermore, without any surprise, a $\mathcal{L}^2$ criterion is well adapted to additive noise, while a $\mathcal{L}^1$, or even better a $\mathcal{L}^0$ criterion, is more resistant to outliers (the numerical results related to these observations are not reported here, but the code is available). All together, we just verify that the proposed mechanism behaves as expected in these various situations.

On the other hand, if the deterministic output values diverge, does the output statistics also diverges? The assumption is that though the individual values are very different, the statistical observable (e.g., mean, correlation) can be adjusted. We thus have to compare the KL-divergence between the desired and

	Robustness to noise				
Criterion	2	1	0	a	b
Number of Iterations	4	3	101	4	15
Minimal criterion value	7.8e-05	2.0e-04	1.2e-04	4.5e-04	4.4e-02
	Robustness to outliers				
Criterion	2	1	0	a	b
Number of Iterations	101	9	4	6	3
Minimal criterion value	2.1e-03	2.8e-03	3.8e-03	6.3e-03	5.6e-02

Figure 9: Robust estimation in the presence of (Top) additive normal noise of relative magnitude $\sigma = 0.1$ and (Bottom) outliers with a probability $\pi = 0.05$ and relative magnitude $\sigma = 10$ using a '2' $\mathcal{L}^2$ criterion, '1' reweighted $\mathcal{L}^1$ criterion, '0' reweighted $\mathcal{L}^0$ criterion, 'a' absolute value, 'b' biweight criterion. The quadratic performs better (considering the number of iterations, the final criterion being of the same order of magnitude) in the presence of noise, while robust criterion are must faster in the presence of outliers.

obtained output given the input, as made explicit in Appendix E. In order to perform this test we have considered directly a random output and evaluated if the `KernelObservableEstimator` can be used with the proposed estimation method. This has been verified with a related precision better than 10^{-3} considering a LNL unit, and two simple models:
- Taking the input mean of a given channel $\omega_n(t) = x_n(t)$ into account,
- Taking the input auto-correlation of a given channel $\omega_{n,\tau}(t) = x_n(t)\, x_n(t-\tau)$ into account,
this last couple of tests being very preliminary, while it is a perspective of this work to further investigate in this direction.

Sequence generation

As a final test, let us consider e.g. the Sierpinski sequence[10], which is deterministic aperiodic, and a function of the $O(\sqrt{t})$ previous samples at time t, thus with long term dependency[11].

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

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

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

As discussed in Appendix D, in the general case and without a specific architecture, we need at least $O(\sqrt{T})$ units to generate an unpredictable sequence of length T, without mistakes. Here we have tested with AIF and LNL units and obtained the results reported in Fig. 10. The units have no input, but there is an offset that allows the units to have some spontaneous activity.

Node type	LinearNonLinearTransform						
Number of units	2	3	4	5	6	7	8
Sequence length	6	5	6	5	8	7	8
Number of Iterations	36	2	4	2	18	14	16

Figure 10: Minimal numbers of unit versus sequence length to generate the Sierpinski sequence without any error, for a network of AIF units and the so called binary criterion.

Discussion

These preliminary tests simply demonstrate that the proposed method works, with better performances than 1st order usual estimation methods. The main positive result is that in all cases only one "output" unit is observed while hidden units weights are adjusted without any restriction on the connectivity. Exact estimation can be obtained if a solution exists. The second interesting observation is that it applies to a large class of unit types and criteria.

These results are quite limited. On one hand, due to computer power availability (running on only one machine with no use of GPU), we have only considered tiny network sizes. However, as discussed in the presentation of the method, as being a distributed mechanism, generalization to much larger setup is really feasible, especially because the algorithm ingredients are quite standard. We also can reasonably hope that the good numerical stability will allow the method to scale up on larger networks. On the other hand, we have not proved here that smaller recurrent networks can outperform huge feed-forward deep networks. However, the question could not be raised before, because weight estimation methods were quite limited, which is not the case here.

We thus can propose these first results as a promising track to further revisit how to estimate weights in recurrent networks.

5 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 backward-tuning numerical stabilization 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.

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[12] network unit and is defined by a recurrent equation of the form:

$$\begin{array}{rcl} x_n(t) & = & \gamma_n\, x_n(t-1) \\ & + & \zeta_{[a,b]}\left(\alpha_n + \sum_{n'=0}^{N-1} W_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{M-1} W_{nm}\, i_m(t-1)\right), \end{array} \tag{7}$$

- with either a fixed or adjustable *leak*[13] γ_n, providing $0 < \gamma_n < 1$, and
- optionally *intrinsic plasticity* parameterized by α_n.

The *non-linearity* often[14] writes

$$\zeta_{[a,b]}(u) \stackrel{\text{def}}{=} \tfrac{a+b}{2} + \tfrac{b-a}{2}\,\tanh(\tfrac{2}{b-a}\,u),$$

with $\zeta_{[a,b]}(-\infty) = a, \zeta_{[a,b]}(+\infty) = b, \zeta_{[a,b]}(u) = \frac{a+b}{2} + u + O(u^3)$, while $\zeta'(u) = 1 - \tanh(\frac{2}{b-a}\,u)^2$, $0 < \zeta'(u) \leq 1$, with $\max|\zeta'(u)| = 1$, thus contracting with a correct numerical conditioning. 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 uniformly converges[15], i.e. $\lim_{\epsilon\to 0}\zeta_{\epsilon,[0,+\infty]}(u) = \zeta_{[0,+\infty]}(u)$. See the section on AIF units to see how to adjust, if needed, such a meta-parameter redefining it as a node 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:

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

[13] 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.

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

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

- 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 [5, 18], with an important additional feature : weight-sharing, i.e. the fact that several weights W_{nd} are the same across different nodes. This is taken into account in this paper.

Long short term memory (LSTM) units. Such network unit is defined by a sophisticated architecture [29], described in figure 1. A unit is made of the following nodes:

$$
\begin{array}{rcl}
\text{Unit output:} & & \\
x_n(t) & = & \zeta_{[0,1]}\left(y_n^{out}(t)\right)\, \zeta_{[-1,1]}\left(s_n(t)\right) \\
\text{Unit state:} & & \\
s_n(t) & = & \zeta_{[0,1]}\left(y_n^{forget}(t)\right)\, s_n(t-1) + \zeta_{[0,1]}\left(y_n^{in}(t)\right)\, \zeta_{[-1,1]}\left(g_n(t)\right) \\
\text{Unit gate:} & & \\
g_n(t) & = & \sum_{n'} W^g_{nn'}\, x_{n'}(t-1) + \sum_m W^g_{nm}\, i_m(t-1) \\
\text{Output modulation:} & & \\
y_n^{out}(t) & = & W^o_n\, s_n(t-1) + \sum_{n'} W^o_{nn'}\, y^c_{n'}(t-1) + \sum_m W^o_{nm}\, i_m(t-1) \\
\text{Forgetting modulation:} & & \\
y_n^{forget}(t) & = & W^f_n\, s_n(t-1) + \sum_{n'} W^f_{nn'}\, y^c_{n'}(t-1) + \sum_m W^f_{nm}\, i_m(t-1) \\
\text{Memorizing modulation:} & & \\
y_n^{in}(t) & = & W^i_n\, s_n(t-1) + \sum_{n'} W^i_{nn'}\, y^c_{n'}(t-1) + \sum_m W^i_{nm}\, i_m(t-1)
\end{array}
\tag{8}
$$

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 [24]) of LSTM, while several variants exist. A rather closed mechanism is named gate recurrent unit [15], 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 [46].

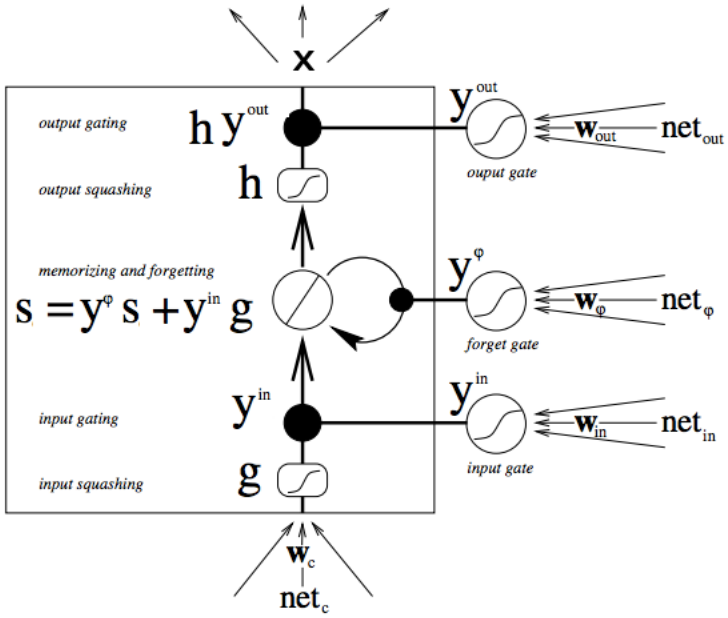


Figure 11: A LSTM unit has three processing stages for bottom to top: The (i) gate g corresponds to as standard LNL unit that (ii) feeds an internal state memory s which value is also driven by a forget (or remember) signal allowing to maintain the previous value, before (iii) the output connected value 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 [29].

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 key-point is that LSTM have, by construction, a real virtue regarding weight adjustment since back-propagation curses (vanishing or explosion) is avoided [46]. 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 other formalism [4] 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 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 [4] and translating in the present notation, at the same degree of generality of LNL networks, we obtain:

$$\begin{array}{rcl} x_n(t) & = & \zeta_{[0,1]}\left(f_n(t)\right)\, x_n(t-1) + \left(1-\zeta_{[0,1]}\left(f_n(t)\right)\right)\, z_n(t) \\ f_n(t) & = & \alpha_n + \gamma_n\, x_n(t-1) \\ z_n(t) & = & \sum_{n'=0}^{N} W'_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{N} W'_{nm}\, i_m(t-1) \end{array} \tag{9}$$

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 [12] for a general discussion on such network computational power and limit). Following [11] (with a tiny change of notation), we consider without loss of generality a discretized form, which writes:

$$\begin{array}{rcl} x_n(t) & = & \gamma_n\, \left(1-\Upsilon_\epsilon\left(x_n(t-1)\right)\right)\, x_n(t-1) \\ & + & \sum_{n'=0}^{N} W_{nn'}\, \Upsilon_\epsilon\left(x_{n'}(t-1)\right) + \sum_{m=0}^{N} W_{nm}\, i_m(t-1), \end{array} \tag{10}$$

where the unit value is over or below the spiking threshold $\theta = 1/2$ (thus spiking or not), while the reset value is 0.

Here, as inspired from [13], we propose to use $\Upsilon_\epsilon(v) \stackrel{\text{def}}{=} \zeta_{[0,1]}\left(\frac{v-1/2}{\epsilon}\right)$, as a mollification of the threshold function[16]:

$$\Upsilon(v) \stackrel{\text{def}}{=} \left\{ \begin{array}{cc} 0 & v < 1/2 \\ 1/2 & v = 1/2 \\ 1 & 1/2 < v \end{array} \right. .$$

To avoid spurious effects when adjusting the weights, we have to find out the best minimal ϵ value for each unit.

[16] Obviously, $\lim_{\epsilon \to 0, v \neq 0} \Upsilon_\epsilon(v) = \Upsilon(v)$, while $\Upsilon'_\epsilon(0) = 1/\epsilon$ and $\int_v |\Upsilon_\epsilon(v) - \Upsilon(v)| = \log(2)/2\,\epsilon$. Here the convergence can not be uniform (since a continuous function converges towards a step function), more precisely $\sup_v |\Upsilon_\epsilon(v) - \Upsilon(v)| = 1/2$ (around $v \simeq 1/2$).

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

$$\begin{array}{rcl} x_n(t) & = & \gamma_n \left[(1-\zeta_{[0,1]}(x_{n_1}(t)))\, x_n(t-1)\right] \\ & + & \alpha_n + \sum_{n'=0}^{N} W_{nn'}\, \zeta_{[0,1]}\left(x_{n'_1}(t-1)\right) + \sum_{m=0}^{N} W_{nm}\, i_m(t-1) \\ x_{n_1}(t) & = & \frac{1}{\epsilon}\left[x_n(t-1) - \frac{1}{2}\right] \end{array}$$

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 [13] where sharpness was considered as a meta-parameter: Here it is a parameter learned on the data. In both cases, 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 D how they can be used to propose trivial solutions to rather complex tasks.

Softmax and exponential probability units. When considering exponential distribution of probability on one hand, or softmax[17] computation on the other hand, one comes to the same equation[18] which writes:

$$\begin{array}{rcl} x_n(t) & = & \frac{e^{z_n(t)}}{\sum_n e^{z_n(t)}} = \exp\left(z_n(t) - \log(\sum_n \exp\left(z_n(t)\right)\right) \\ z_n(t) & = & \alpha_n + \sum_{n'=0}^{N} W'_{nn'}\, x_{n'}(t-1) + \sum_{m=0}^{N} W_{nm}\, i_m(t-1) \end{array} \tag{11}$$

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 [5, 18].

The 1st line is a firmware global equation[19] 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) [5]. We mention this possibility for the

[17] 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.

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

[19] It is worthwhile mentioning that:

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

thus numerically well defined, with no singularity, the transformation being contracting, i.e., $|\partial_{\mathbf{z}}\mathbf{o}| \leq 1$, with $\max|\partial_{\mathbf{z}}\mathbf{o}| = 1$.

completeness of the discussion, making explicit the fact that the present framework includes such equation. However, the estimation problem addressed in RBM completely differs (as being a stochastic estimation paradigm) from 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 [58] 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 further discussed in [21]. 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 [26]. Following [20], 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, [20] 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 Comparison with related recurrent weight estimation methods

In this section we briefly discuss how this method compares with existing methods of recurrent weight methods estimation.

The back-propagation through time (BPTT) is a gradient-based technique used, .e.g., in Elman's Networks [19], where the standard back-propagation algorithm is applied to both the network recurrent layers and through time. It is based on the propagation of the error gradient, and it generally remains on two assumptions that the cost is additive with respect to training examples and that it can be written as a function of the network output (see, e.g., [40]). With respect to this basic method, our method:
- does not rely on the cost gradient propagation, but the error backward propagation (or tuning), while gradients remain local to a unit.
- has been stated including for non additive costs (such as statistical criteria) and for both supervised criterion based on the network output error, or other unsupervised criteria.

Our formulation has been formalized, by, e.g. [17], but without proposing a second order estimation method, considering explicitly the backward tuning of the error with a heuristic to avoid extinction and explosion. Moreover, the fact this formalism has been applied on the formulation propose in section 2 with intermediate variables makes the backward tuning proposal more efficient, than if non linearity and weights linear combination have been mixed.

Furthermore, as made explicit in [60] when comparing back-propagation with contrastive Hebbian learning, or in [17], our backward tuning mechanism corresponds gradient back-propagation up to a change of variable. However contrary to [60] or [29], there is no need to introduce further approximation (such as, e.g, only considering diagonal terms) in order to write the backward propagation rule. This variant is well-founded, simpler to write and seems to be numerically more stable.

A step further, artificial neuron network back-propagation has been related to biological back-propagation in neurons of the mammalian central nervous system (see, e.g., [51]) and it is clear that the propagation of a learning or adaptive error, is more likely to be related to backward tuning of an error, than an energy or criterion gradient minimization. Regarding biological plausibility, our method only involves local distributed adjustments, as a version of back-propagation that can be computed locally using bi-directional activation recirculation [27] instead of back-propagated error derivatives is more biologically plausible, and has been improved by [41]. In its generalized form it also communicates error signals, being inspired by contrastive learning, and using the Pineda and Almeida algorithm [43].

All these methods operate on the current estimate of the derivative of the error, not the backward tuning error defined here, while related to specific cost function.

The proposed method also enjoy an interesting interpretation related to the 2nd order estimation method, as made explicit in footnotes[7] and [6]. Thanks to the simple formulation, and either from the backward tuning of the estimation error in the case of footnote[7] or by direct estimation in the case of footnote [6] we obtain an estimation not only of the output desired value, but also of hidden state desired value. This corresponds to a deterministic estimation / minimization algorithmic scheme : estimation of the desired hidden state value, given the current weight values followed by the local minimization of the criterion adjusting the unit weights.

As it, even if in relation with the usual standard back-propagation method, the proposed method is a real alternative.

C Using this framework in different contexts

In this section we make explicit mechanism of estimation that can make use of the previous variational mechanism.

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, see Fig. 12).

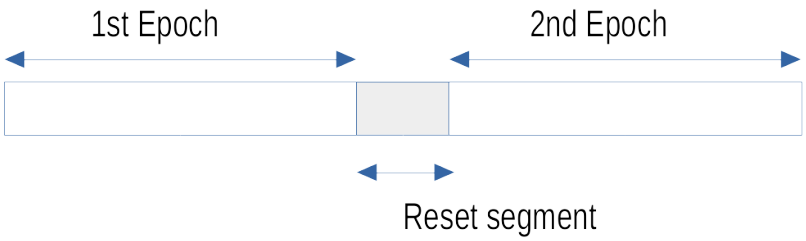


Figure 12: If the supervised learning is performed with different epoch of data, this is equivalent to a unique epoch, providing a reset segment of length R, the maximal recurrent range, is inserted before each new epoch. During reset segment, we set $\kappa_{nt} = 0$.

Least-square adjustment

With respect to desired output $\bar{o}_n(t)$ we can write, in the $\mathcal{L}^2$ case:

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

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, while since $\kappa_{nt} \in [0, +\infty[$ it can also act as error gain, taking related precision into account.

Robust criterion

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. It is

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, as made explicit in Fig 13. This has been addressed, e.g, by [14].

With respect to usual M-estimators[20], we propose here to use reweighted quadratic criterion, i.e., consider a previous estimation $\hat{\mathbf{x}}$ of $\mathbf{x}$, in order to locally work at each step with a least-square criterion, namely:

$$ {}_1\rho_{nt}(x) \equiv \frac{x^2}{\nu + |\hat{x}|} \text{ or } {}_0\rho_{nt}(x) \equiv \frac{x^2}{\nu + \hat{x}^2}, $$

for unbounded and bounded profiles. This is equivalent to use an approximate criterion derivative.

For small values of ν the criterion allows to perform sparse estimations. Anyway, the value of ν is not to be adjusted manually, but can simply be set at a fraction of the criterion minimal value, say $\nu \simeq 10^{-3}\,\rho_{\min}$.

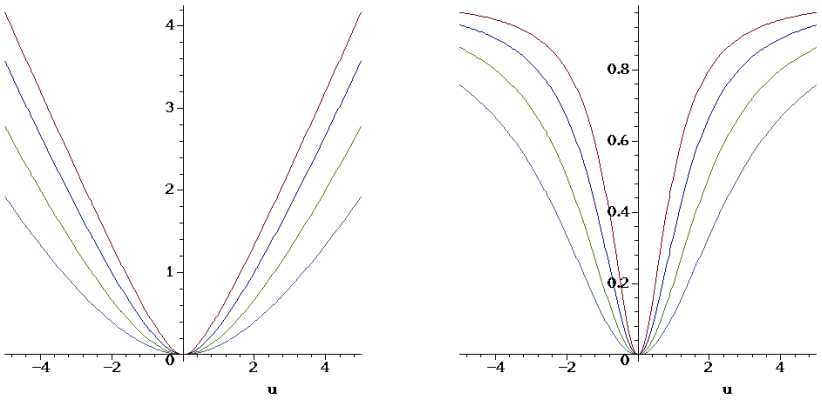


Figure 13: Two examples of M1 profiles. Leftward, an unbounded profile, mollification of the $\mathcal{L}_1$ criterion, thus close to a $\mathcal{L}^2$ criterion for low values, and to a $\mathcal{L}^1$ criterion for high values:
$$\rho_{nt}(x) \equiv \tfrac{x^2}{\nu+|x|} = x^2/\nu + O(|x|^3) = |x| - \nu + O(1/|x|),$$
for $\nu = 1, 2, 4, 8$. Rightward, a bounded profile, mollification of the $\mathcal{L}_0$ criterion:
$$\rho_{nt}(x) \equiv \tfrac{x^2}{\nu+x^2} = x^2/\nu + O(x^4) = 1 + O(1/x^2),$$
for $\nu = 1, 2, 4, 8$.

[20]For instance, for a bounded criterion a usual choice is the twice continuous differentiable biweight profile $\rho_{nt}(x) \equiv (x^6 - 3\,x^4 + 3\,x^2)\,H(1 - |x|) + H(|x| - 1)$, with a hard threshold at $x_{thres} = 1$.

Boolean adjustment

Another aspect of the estimation is the fact that we may have to estimate Boolean value, i.e., with the notations of this paper, requires the values $\hat{x}_n(t)$ to be lower or higher than $1/2$, according to $\bar{o}_n(t)$. Taking a margin ν into account, and considering a quadratic criterion, a natural choice writes:

$$\rho_{nt}(\hat{x}_n(t)) = \tfrac{\kappa_{nt}}{2} q\left(\nu - sg(\bar{o}_n(t) - 1/2)\,(\hat{x}_n(t) - 1/2)\right), \quad q(v) \stackrel{\text{def}}{=} H(v)\,v^2$$

where $sg()$ is the sign function, which is a criterion that vanishes if and only if $|\hat{x}_n(t) - 1/2| > \nu$ and has the right sign, while it behaves as a quadratic criterion otherwise.

Stochastic adjustments

A step further, we may not be interested to perform a deterministic adjustment, but to optimize the output probabilistic distribution with respect to the desired output distributions, as e.g., in [16] for spiking neuron networks adjustment. As developed in appendix E, the weight estimation problem can be related to the following criterion:

$$\rho(\mathbf{x}) \stackrel{\text{def}}{=} \sum_k |\lambda_k| \left| \bar{\Omega}_k - \tfrac{1}{T-\tau_k} \sum_t \omega_k(t) \right|,$$

for some observable $\omega_k(t)$, with average value $\bar{\Omega}_k$, and parameters λ_k made explicit in appendix E and considered here as input.

We may for instance consider mean and auto-correlation as in appendix E, or instantaneous momenta at time t, e.g., mean and variance, considering a Gaussian distribution.

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. [55, 56] 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.

One track is to 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}, as previously discussed.

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.

D Closed forms solution for neural network tasks

Let us illustrate how 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 [29, 24, 39] for certain architectures are trivial for others. In particular, the use of AIF neurons simplifies certain problems, e.g. requiring long short-term memory. We illustrate this point here considering deterministic sequence generation and long-term non-linear transform, and provide explicit simple solutions for those problems.

Generating long term sequential signals

The lever is that it is straightforward to generate a delayed step signal (i.e., equal to 0 before $t = \tau$, and 1 after) using AIF units, e.g.:

$$s_\tau(t) = \tfrac{1}{2}\left(1 - \Upsilon(s_\tau(t-1))\right) s_\tau(t-1) + h_\tau$$

with

$$h_\tau \stackrel{\text{def}}{=} \frac{1}{4\left(1-2^{\frac{1}{2}-\tau}\right)} \in [h_\infty = 1/4, h_1 \simeq 0.85],$$

for which we easily obtain[21] $\Upsilon(s_\tau(t)) = \delta_{t\geq\tau}$.

The numerical limit of this method is the fact that for huge value of τ the parameter precision must be of order $O\left(2^{-\tau}\right)$. To avoid this constraint, either an architecture with several units building a delay line, or with a ramp unit and adaptive thresholds (see next section) can be considered.

From this basic element we can generate a delayed clock signal[22] or another long-term mechanism, such a as flip-flop[23], which is a fundamental building blocks

[21] **Delayed step signal.** Starting with $s_\tau(0) = 0$ this first order recurrent equation yields:

$$s_\tau(t) = 2\,h\,\left(1 - 2^{-t}\right) \in [0, 2\,h],$$

which is an bounded increasing negative exponential profile, for which the parameter h has been chosen to maintain $s_\tau(t) < 1/2, t < \tau$, and reach $s_\tau(t) > 1/2$, for $t \geq \tau$.

[22] **Delayed clock signal.** Modifying the delayed step signal, and adding a memory carousel unit in order to reset the signal after the step and keep it reseted, we obtain:

$$\begin{aligned} c_\tau(t) &= \tfrac{1}{2}\left(1 - \Upsilon(c_\tau(t-1))\right) c_\tau(t-1) + h_\tau\left(1 - \Upsilon(d_\tau(t-1))\right) \\ d_\tau(t) &= d_\tau(t-1) + \Upsilon(c_\tau(t-1)), \end{aligned}$$

with $\Upsilon(c_\tau(t)) = d_\tau(t) = 0, t < \tau$, until $c_\tau(\tau) > 1/2$, As a consequence $d_\tau(\tau+1) = 1$, thus $c_\tau(\tau+1) = 0$, which is a stable fixed point, values remaining constant beyond. Finally we obtain $\Upsilon(c_\tau(t)) = \delta_{t=\tau}$ in this case.

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

$$z(t) = \Upsilon(z(t-1)) + \Upsilon(i_1(t)) - \Upsilon(i_0(t))$$

yielding the following behavior:
- *R-state*: If $i_0(t) < 1/2$ and $i_1(t) < 1/2$ (no-input) and $z(t-1) < 1/2$, then $z(t) = 0 < 1/2$, the

of any digital transform, in conjunction with logic gates such as a xor gate[24].

If we consider a mollification instead of a step function (i.e., replacing Υ withΥ_ϵ in the previous equation), we obtain the behavior for sufficiently large slopes. More precisely[25], for instance, we numerically observed the same qualitative behavior in the delayed step signal case, with $h \in [h_\infty = 0.376, h_1 = 0.5]$, while h is not given in closed form in this case.

Further on this track, it is clear that we can compile any sequential circuit in such networks, which is far from being new. The add-on here is about that the fact we provide explicit solutions, using AIF neurons, with a lower complexity in terms of network nodes than using LSTM units. Let us see two paradigms where this enlighten the problem complexity.

Long term non-linear transform

In many experiments, a variant of a sequence of the form:

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

reset state is maintained.
- *S-state*: If $i_0(t) < 1/2$ and $i_1(t) < 1/2$ (no-input) and $z(t-1) > 1/2$, then $z(t) = 1 > 1/2$, the set state is maintained.
- *R-S transition*: If $i_0(t) < 1/2$ and $i_1(t) > 1/2$ and $z(t-1) < 1/2$, then $z(t) = 1 > 1/2$, flipping to a set state; if it was already in the set state, we still have $z(t) = 2 > 1/2$.
- *S-R transition*:If $i_0(t) > 1/2$ and $i_1(t) < 1/2$ and $z(t-1) > 1/2$, then $z(t) = 0 < 1/2$, flipping to a reset state; f it was already in a reset state, we still have $z(t) = -1 < 1/2$.
- *no instability*: $i_0(t) > 1/2$ and $i_1(t) > 1/2$ contrary to a standard digital RS-latch we simply have $z(t) = \Upsilon(z(t-1))$ providing it was in set of reset state, without any meta-stability.

[24]**Defining the xor function.** It is straightforward to notice that:

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

verifies

$$\begin{aligned} \Upsilon(x_o(t)) &= \Upsilon(x_a(t-1)) \text{ and } \Upsilon(x_b(t-1)) \\ \Upsilon(x_\bullet(t)) &= \Upsilon(x_a(t-1)) \text{ xor } \Upsilon(x_b(t-1)) \end{aligned} \quad ,$$

while other logic gates are easy to build in a similar manner.
A step further the expression

$$x_\dagger(t) = 1/2 - 2\,(\Upsilon(x_a(t-1)) - 1/2)\,(\Upsilon(x_b(t-1)) - 1/2)$$

now considering a multiplication unit, directly calculates the xor function, but does no correspond to some AIF unit.

[25]This is obtained, e.g., by the following piece of maple code:

```
upsilon := (u) -> 1/(1+exp(-4*(u - 1/2)/epsilon)):
c_n := c -> (1 - upsilon(c)) * c / 2 + h:
bounds := [solve(c_n(1/2) = 1/2, h), solve(c_n(0) = 1/2, h)];
```

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 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". In fact, the explicit implementation of such a mechanism on the given example is trivial[26].

What do we learn from this very simple development? While authors have already made explicit the fact that such computations rely on "gate unit" and "memory unit", it seems that "delayed unit" (i.e. learning a time delay) are also basic components. It is also an example of how deterministic computations might become simple, if we introduce a-priory information on the computation, via dedicated units.

Deterministic sequence generation

What is 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 an unpredictable sequence, without any algorithm to generate it, unless copying 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[27]. This solution requires a very large recurrent range, and the numerical precision is limited by the fact that errors accumulate along the recurrent calculation.

On the other hand, feed-forward units of range $R = 1$ solve explicitly the

[26] **An example of long term computation.** One solution writes:

$$\begin{aligned} o_0(t) &= (1 - \Upsilon(c_T(t))\, i(t) + (\Upsilon(c_T(t)) - 1)\, x_a(t)\, x_b(t) + \\ x_a(t) &= (1 - \Upsilon(c_0(t))\, x_a(t-1) + (\Upsilon(c_0(t)) - 1)\, i(t) \\ x_b(t) &= (1 - \Upsilon(c_1(t))\, x_b(t-1) + (\Upsilon(c_1(t)) - 1)\, i(t) \end{aligned}$$

while $c_\tau(t) = \delta_{t=\tau}$ are clock signals, as defined previously, and it is easy to verify that $x_a(t)$ "opens" the memory at time $t = 0$, and stores the previous value otherwise, with a similar behavior for $x_b(t)$, while $o_0(t)$ simply mirror the input until $t = T$, where the expected result is output. Obviously, these are no more AIF units but introduce multiplications between state values

[27] **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 $d > 0$:

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

providing that $\bar{o}_n(1) \neq 0$, 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 $x_n(t) = x'_n(t) - 1$, using now an additional node.

problem using T clock units and N_0 readout units, with $O(N_0\,T)$ weights. This requires no more than $N_0 + T$ units considering binary information[28], and no more that $N_0 + 1$ units if the numerical precision is sufficient and unit threshold adjustable[29]. A step further, considering less than $N_\bullet \stackrel{\text{def}}{=} \sqrt{N_0\,T/R}$ linear or NLN units of range R, we can not generate a solution in the general case[30].

The generation of periodic signal of period T, is a very similar problem, as studied in [45], 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 see that the complexity of signal generation problem highly depends on the kind of "allowed units" and reduces to a trivial problem as soon as suitable operation are allowed. Furthermore, there exist a $R = 1$ network of at most $N_0 + T$ units that exactly solves the problem, without requiring huge precision, while a linear network, a NLN network or a AIF network can generate such a sequence in the general case, with either a closed form solution, or solving a linear system of equation.

---

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

$$\begin{array}{rcll} x_{n_0}(t) & = & \sum_{n=0}^{T-1}(\bar{o}_{n_0}(n) - \bar{o}_{n_0}(n+1))\,\Upsilon(x_{N_0+n}(t)) & 0 \le n_0 < N_0, \text{ writing } \bar{o}_{n_0}(T) \stackrel{\text{def}}{=} 0 \\ x_{N_0}(t) & = & \frac{1}{2}\,(1 - \Upsilon(x_{N_0}(t-1)))\,x_{N_0}(t-1) + h_1 & \\ x_{N_0+n}(t) & = & x_{N_0+n-1}(t-1) & 0 < n < T \end{array}$$

thus providing T delayed step signals such that $\Upsilon(x_{N_0+n}(t)) = \delta_{t>n}$, allowing us to generate the desired sequence combining these signals. If we now consider mollification of he threshold function, the previous system of equation is going to generate a temporal partition of unity. 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.

---

[29]**Long sequence generation with a ramp unit.** If we can consider units of the form:

$$\begin{array}{rcl} x_{n_0}(t) & = & \sum_{n=0}^{T-1}(\bar{o}_{n_0}(n) - \bar{o}_{n_0}(n+1))\,\Upsilon(x_{N_0+n}(t) - \theta_n) \\ x_{N_0}(t) & = & x_{N_0}(t-1) + 1 \end{array}$$

with the ramp unit $x_{N_0}(t)$ precision being of order $O(1/T)$, while we now can introduce adaptive thresholds $\theta_n = n$, it is obvious to verify that we solve the problem with two units.

---

[30]**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 $0 \le n_0 < N_0$ output units and $N_0 \le n < N$ hidden units, using vectorial notations, with the shift operator $\mathcal{S}$ defined as $\mathcal{S}\mathbf{x}(t-1) = \mathbf{x}(t)$, we obtain:

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

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 can not have $N^2\,R+N < N_0\,T$ for this algebraic system of equation to have a solution in the general case.

E Stochastic adjustment of the network weights

Problem position

Let us consider the problem of optimizing the probabilistic distribution $\tilde{p}(\tilde{\mathbf{x}} = \mathbf{x})$ of a network output, as a function of the desired distribution $\bar{p}(\bar{\mathbf{o}} = \mathbf{o})$ of a root network. Since we are in a multi-dimensional and dynamic framework, with continuous values, it is intractable to consider as it the distribution, but only a parametric model of it, and adjust the parameters of this model. The key point is that we are going to consider observables of the output, written

$$\Omega_k \stackrel{\text{def}}{=} \mathbb{E}[\omega_k] \simeq \frac{1}{T-\tau_k} \sum_{t=0}^{T-\tau} \omega_k(t),$$

for an observable $\omega_k(x_n(t) \cdots x_n(t-\tau_k))$ of a given rank τ_k, using the ergodic assumption to approximate numerically its value.

Let us, for instance, consider that the network output mean $\tilde{\Omega}_{n,\bullet}$ and auto-correlation in a $\tau = \{0, \Delta\}$ time window $\tilde{\Omega}_{n,\tau}$, corresponds to the desired statistical paremeters:

$$\begin{aligned} \Omega_n &= \mathbb{E}[\omega_n(t)], \quad \omega_n(t) \stackrel{\text{def}}{=} o_n(t) \\ \Omega_{n,\tau} &= \mathbb{E}[\omega_{n,\tau}(t)] \quad \omega_{n,\tau}(t) \stackrel{\text{def}}{=} o_n(t)\, o_n(t-\tau), \end{aligned}$$

the normalized temporal auto-correlation being:

$$C_{n,\tau} = (\Omega_{n,\tau} - \Omega_n^2)/(\Omega_{n,0} - \Omega_n^2).$$

We thus do not constraint the output desired values directly but only some momentum expectation. We could also have considered higher order momenta, e.g., kewness and kurtosis, or spatial correlations, and so on.

A step further we must introduce the notion of *normalized observable* Υ_k, given a desired values $\bar{\Omega}_k$, thats maps the observable value Ω_k onto a dimension-less normalized monotonic error function. For the mean Ω_n we propose

$$\Upsilon_n = \tanh\left((\Omega_n - \bar{\Omega}_n) \Big/ \mathring{\Omega}_n\right), \mathring{\Omega}_n = \begin{cases} \bar{\Omega}_n & \text{if} \quad |\bar{\Omega}_n| > \epsilon \\ \sqrt{\bar{\Omega}_{n,0}} & \text{if} \quad |\bar{\Omega}_n| < \epsilon, \bar{\Omega}_{n,0} > \epsilon^2 \\ 1 & \text{otherwise,} \end{cases}$$

in words the mean difference is normalized using the expected mean, if not negligible, or the standard deviation if given, though a normalized sigmoid profile. For second order momenta we can consider,.e.g., the normalized correlation, writing:

$$\Upsilon_{n,\tau} = \tanh\left((\Omega_{n,\tau} - \bar{\Omega}_{n,\tau}) \Big/ (\bar{\Omega}_{n,0} - \bar{\Omega}_n^2)\right),$$

providing the denominator does not vanish, normalizing the estimated observable $\Omega_{n,\tau}$ against desired observable values $\bar{\Omega}_k$. Our assumption is that there is always an application dependent pertinent way of defining such a normalization mapping. More generaly we require Υ_k to be a point-wise separating function

$$\Upsilon_k = \rho_{\bar{\Omega}}\left(\Omega_k - \bar{\Omega}_k\right), \text{ with } \rho_{\bar{\Omega}}(u) = 0 \Leftrightarrow u = 0 \text{ and } \rho_{\bar{\Omega}}(u) \in [-1, 1].$$

Considering a general model

Adapting the development given in [57] for binary distribution, we propose to minimize the KL-divergence, considering maximal entropy Gibbs distributions. We are going to propose to adjust the network weights in order to minimize an approximation of the KL-divergence between the desired and simulated distribution.

If we look for a distribution of probability with maximal entropy and which observable ω_k correspond to some expectation values Ω_k, we obtain:

$$p(\mathbf{x}) = \frac{\exp\left(\sum_k \lambda_k \, \omega_k(\mathbf{x})\right)}{Z_p(\lambda)},$$

where the denominator guaranties $\int_{\mathbf{x}} p(\mathbf{x}) = 1$ and is called the partition function[31], topological pressure or free energy. The quantity $Z_p(\lambda)$ has no closed form beyond simple cases, and can be numerically estimated as:

$$Z_p(\lambda) = \int_{\mathbf{x}} \exp\left(\sum_k \lambda_k \, \omega_k(\mathbf{x})\right) \simeq \frac{1}{T-\tau} \sum_{t=0}^{T-\tau} \exp\left(\sum_k \lambda_k \, \omega_k(t)\right)$$

, under the ergodic assumption, τ being chosen for all observable $\omega_k(t)$ to be defined.

Fitting a Gibbs distribution

A step further, it appears that minimizing the KL-divergence between the observed distribution $\bar{p}(\bar{\mathbf{o}})$ and the Gibbs model corresponds to adjust the parameters $\bar{\lambda}$ in order the predicted observable expectation $\Omega_k(\lambda)$ to get as closed as possible to the desired observable expectation $\bar{\Omega}_k$, which is a standard estimation problem (in a nutshell, the trick is to minimize the criterion gradient, not the criterion itself[32]).

[31] **Maximal entropy distribution.** Given expectation Ω_k of observable $\omega_k(t)$ we state that we look for a probability distribution of maximal entropy which corresponds to the observable expectation. This writes, with Lagrangian multipliers λ_k:

$$\min_\lambda \underbrace{\int_{\mathbf{x}} p(\mathbf{x}) \log(p(\mathbf{x}))}_{entropy} + \underbrace{\lambda_0 \left(\int_{\mathbf{x}} p(\mathbf{x}) - 1 \right)}_{normalization} - \underbrace{\sum_k \lambda_k \left(\int_{\mathbf{x}} p(\mathbf{x}) \, \omega_k(\mathbf{x}) - \Omega_k \right)}_{observations}$$

and the functional derivative of this criterion yields:

$$p(\mathbf{x}) = \exp\left(\sum_k \lambda_k \, \omega_k(\mathbf{x})\right) / Z_p(\lambda),$$

as easily obtained from the normal equation derivation, see e.g.:

https://en.wikipedia.org/wiki/Maximum_entropy_probability_distribution#Proof.

[32] **Fitting the Gibbs parameters distribution**. For the sake of completeness, let us detail how such estimation can be performed. If we consider the KL-divergence between the observed distribution $\bar{p}(\bar{\mathbf{o}})$ and the model approximate distribution $q(\mathbf{x})$), we easily derive:

$$\begin{aligned} d_{KL}(\bar{p}(\bar{\mathbf{o}}) \| q(\mathbf{x})) &= \int \bar{p}(\bar{\mathbf{o}}) \, \log\left(\frac{\bar{p}(\bar{\mathbf{o}})}{q(\mathbf{x})}\right) \\ &= \int \bar{p}(\bar{\mathbf{o}}) \, \log\left(\bar{p}(\bar{\mathbf{o}})\right) - \int \bar{p}(\bar{\mathbf{o}}) \, \log\left(q(\mathbf{x})\right) \\ &= -h_{\bar{\mathbf{o}}} - \int \bar{p}(\bar{\mathbf{o}}) \log\left(q(\mathbf{x})\right) \\ &= -h_{\bar{\mathbf{o}}} - \int \bar{p}(\bar{\mathbf{o}}) \left(\sum_k \lambda_k \, \omega_k - \log(Z(\lambda))\right) \\ &= -h_{\bar{\mathbf{o}}} - \sum_k \lambda_k \, \bar{\Omega}_k + 1 \, \log(Z_q(\lambda)) \end{aligned}$$

As a consequence, given a desired output $\bar{\mathbf{o}}$ and a choice of observable ω_k we can estimate the maximal entropy parameters $\bar{\lambda}$.

Statistical weight adjustment from the parametric model

Given of set of desired observable values $\bar{\Omega}_k$, with the corresponding Gibbs model $\hat{p}(\bar{\mathbf{o}})$ parameterized by $\bar{\lambda}$ and adjusted on the reference samples $\bar{\mathbf{o}}$, we now can state the problem of adjusting the network weights. We consider the KL-divergence between the observed distribution $\bar{p}(\bar{\mathbf{o}})$, approximated by the related Gibbs model, and the network simulation $\tilde{p}_{\mathbf{W}}(\tilde{\mathbf{x}})$, parameterized by the network weights $\mathbf{W}$. The network is viewed here as a parametric model of the observed distribution.

Since the network simulation is brought to the desired reference samples distribution, modeled as a Gibbs distribution, we are going to assume that the network simulation can itself be represented by a Gibbs distribution with the goal to adjust the weights in order the related divergence $d_{KL}(\bar{p}(\bar{\mathbf{o}})\|\tilde{p}(\tilde{\mathbf{x}}))$. As before, we can replace the KL-divergence minimization by the minimization of the gradient, i.e., some point separating positive function of the difference between the measured observable Ω_k on the simulation and the desired value $\bar{\Omega}_k$.

---

combining the previous equations, and since the term $h_{\bar{\mathbf{o}}} \stackrel{\text{def}}{=} -\int \bar{p}(\bar{\mathbf{o}}) \log(\bar{p}(\bar{\mathbf{o}}))$ is the observed entropy and is constant with respect to the parameter to estimate, we are left with the following criterion, which in fact corresponds to cross-entropy maximization $\min_\lambda \mathcal{J}$, with:

$$\begin{aligned} \mathcal{J} &= \log\left(Z_q(\lambda)\right) - \textstyle\sum_k \lambda_k \bar{\Omega}_k \\ \partial_{\lambda_k}\mathcal{J} &= \Omega_k(\lambda) - \bar{\Omega}_k \\ \partial_{\lambda_k\,\lambda_l}\mathcal{J} &= \Omega_{kl}(\lambda) \end{aligned}$$

writing $\omega_{kl}(t) = \omega_k(t)\,\omega_l(t)$, and $\Omega_{kl} = \mathbb{E}[\omega_{kl}]$. This computation comes from the fact that:

$$\begin{aligned} Z_q(\lambda) &= \textstyle\int_{\mathbf{x}} \exp\left(\sum_k \lambda_k\,\omega_k(\mathbf{x})\right) \\ \partial_{\lambda_k} Z_q(\lambda) &= \textstyle\int_{\mathbf{x}} \exp\left(\sum_k \lambda_k\,\omega_k(\mathbf{x})\right)\,\omega_k(\mathbf{x}) \\ &= \textstyle\int_{\mathbf{x}} Z_q(\lambda)\,q(\mathbf{x})\,\omega_k(\mathbf{x}) \\ &= Z_q(\lambda)\,\Omega_k(\lambda), \end{aligned}$$

and it is easy to approximate:

$$\begin{aligned} \Omega_l(\lambda) &\stackrel{\text{def}}{=} \textstyle\int_{\mathbf{x}} \frac{\exp\left(\sum_k \lambda_k\,\omega_k\right)}{Z_q(\lambda)}\omega_l(t) \\ &\simeq \textstyle\frac{1}{T-\tau_l}\sum_t \omega_l(t) \end{aligned}$$

under the ergodic assumption.

As a consequence, despite the caveat that $Z_q(\lambda)$ calculation is usually not tractable, this allows us to implement some paradigm that tends to minimize the criterion gradient (since at a criterion minimum, the gradient vanishes):

$$\bar{\lambda} = \arg\min_\lambda \rho\left(\Omega(\lambda) - \bar{\Omega}\right),$$

for some suitable point separating positive function $\rho()$.

This design choice is valid because the topological pressure is convex with respect to λ, so that the criterion is convex [57]. As a consequence, the criterion is minimal when the gradient magnitude vanishes, i.e. is minimal too, while the criterion decreases with the gradient magnitude, thanks to being a convex criterion.

This is the point where we use the normalized observable and propose to minimize:

$$\rho(\mathbf{x}) \stackrel{\text{def}}{=} \sum_k |\Upsilon_k|^d = \sum_k |\rho_k \left(\bar{\Omega}_k - \Omega_k(\mathbf{W})\right)|^d. \tag{12}$$

and have numerically experimented that $d = 1$ seems more efficient than $d = 2$.

Boostrapping the network estimation

Given the previous criterion, we consider a network simulation $x_n(t)$, with the goal to modify the related network weights in order the network output observable Ω_k to be as closed as possible to the desired observable $\bar{\Omega}_k$. To this end, we define desired values $\hat{x}_n(t)$, as follows

$$\min_{\hat{x}_n(t)} \tfrac{1}{2} \textstyle\sum_{nt} (\hat{x}_n(t) - x_n(t))^2 + \sum_k \lambda_k \left[\bar{\Omega}_k - \Omega_k(\mathbf{x})\right]$$

in words: The desired values are the closest values with respect to the present simulation that match the desired observable, which is a standard non-linear constrained least-square 2nd order recurrent scheme (see e.g., [59]). Such desired values may be used to provide local solutions to the estimation problem.

References

[1] Maria Elena Acevedo-Mosqueda, Cornelio Yáñez Márquez, and Marco Antonio Acevedo-Mosqueda. Bidirectional associative memories: Different approaches. *ACM Comput. Surv.*, 45(2):18:1–18:30, March 2013.

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

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

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

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

[6] 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.

[7] 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.

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

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

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

[11] 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.

[12] 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.

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

[14] D. S. Chen and R. C. Jain. A robust backpropagation learning algorithm for function approximation. *IEEE Transactions on Neural Networks*, 5(3):467–479, May 1994.

[15] 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.

[16] Rodrigo Cofre and Bruno Cessac. Exact computation of the Maximum Entropy Potential of spiking neural networks models. Technical report, May 2014. working paper or preprint.

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

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

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

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

[21] Thalita F. Drumond, Thierry Viéville, and Frédéric Alexandre. Using prototypes to improve convolutional networks interpretability. In *31st Annual Conference on Neural Information Processing Systems: Transparent and interpretable machine learning in safety critical environments Workshop*, 2017.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] Geoffrey E Hinton and James L. McClelland. Learning representations by recirculation. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 358–366. American Institute of Physics, 1988.

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

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

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

[31] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489 – 501, 2006. Neural Networks.

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

[33] 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.

[34] 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.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira Weinberger, C. J. C. Burges, L. Bottou, and K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[36] Y. Lecun, L Bottou, Y Bengio, and P Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] 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.

[38] 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.

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

[40] Michael Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[41] R. C. O'Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8(5):895–938, July 1996.

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

[43] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.*, 59:2229–2232, Nov 1987.

[44] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, Mar 2017.

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

[46] 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].

[47] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. dec 2013.

[48] 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.

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

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICRL)*, pages 1–14, sep 2015.

[51] Greg Stuart, Nelson Spruston, Bert Sakmann, and Michael Häusser. Action potential initiation and backpropagation in neurons of the mammalian cns. *Trends in Neurosciences*, 20(3):125 – 131, 1997.

[52] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *Arxiv*, page 12, feb 2016.

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. sep 2014.

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

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

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

[57] Juan Carlos Vasquez, Thierry Viéville, and Bruno Cessac. Parametric Estimation of Gibbs distributions as general Maximum-entropy models for the analysis of spike train statistics. Research Report RR-7561, March 2011. This work corresponds to an extended and revisited version of a previous Arxiv preprint, submitted to HAL as http://hal.inria.fr/inria-00534847/fr/.

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

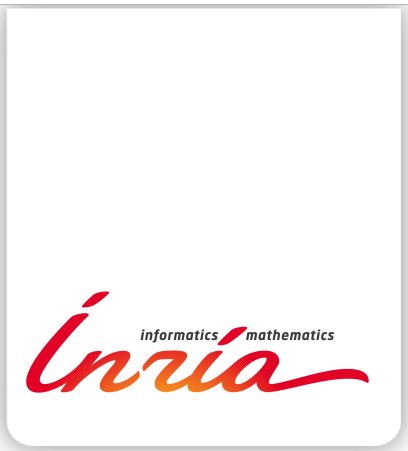
[59] Thierry Viéville and Peter Sander. Using pseudo Kalman-filters in the presence of constraints application to sensing behaviors. Research Report RR-1669, INRIA, 1992.

[60] Xiaohui Xie and H. Sebastian Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural Computation*, 15(2):441–454, 2003.

[61] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science*, 8689:818–833, 2014.

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

Contents



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

351, Cours de la Libération

Bâtiment A 29

33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399