Supervised Learning in Recurrent Spiking Neural Networks of LIF neurons

Javier Prieto

Abstract

Artificial neural networks are theoretical models of the nervous system used to investigate, among other things, the phenomenon of learning. Although neurons have long been known to communicate via spikes, sudden surges and decreases of their electrostatic potential, the study of learning has been limited until very recently to smoothed, simplified versions of these dynamics. In the present work we give a brief survey of the reasons why learning in so-called recurrent spiking networks has proven difficult for so long and show through numerical simulations that these networks are nonetheless able to learn a few simple tasks if their parameters are chosen correctly.

I. INTRODUCTION

Originally conceived as a model of the central nervous system of animals, neural networks have attracted the interest of researchers in distant fields, such as physics, psychology, neurobiology, and computer science. Recent improvements in computing power have opened the possibility of using them in the fields of machine learning and artificial intelligence, where they have proven useful in many tasks where traditional approaches fail [14]. These networks, however, differ substantially from their biological predecessors, both in design and purpose.

In the present work we will focus on the problem of training recurrent spiking neural networks (RNN) of leaky integrate-and-fire (LIF) neurons using the FORCE algorithm [22]. These terms will be carefully defined in the following subsections. This report is organized as follows: in the current section we give a brief overview of different neuron models (I A) and network architectures (I B); and define the problem of training a recurrent neural network, describe the challenges that it poses and discuss the key features of the algorithm that we will use to address them (I C). In section II, we describe our implementation of an artificial neural network (II A) and the algorithm used to train it (II B). In section III) we present and discuss the results of the numerical experiments we have carried out. These constitute the core of our work. We end this report with some brief conclusions in section IV.

A. Neuron models

Animal cells are contained in an insulating lipidic membrane. This membrane is pierced by numerous proteic ion channels that open or close selectively to let or impede the flow of ions across the membrane, thus reducing the effective electric resistance of the cell. All animal cells behave as capacitors to some extent, maintaining an electrostatic potential gradient across their membrane. However, neurons are the only ones that use this as an information encoding mechanism.

Although the electrostatic potential may vary across the cell's cytoplasm, it is convenient

to neglect this and characterize a neuron by a single parameter, v, the average potential in the soma (the body of the neuron) with respect to the extracellular medium, where the potential is zero by convention. This approximation is called the single-compartment model and the potential thus obtained is called the membrane potential [5]. Its order of magnitude can be estimated from simple thermodynamic arguments. Assuming the electrostatic energy of a single ion ev is of the order of its thermal energy kT, its value should be around 27 mV for warm-blooded animals. Indeed, typical experimental values range from -90 mV to 50 mV [5].

For a single-compartment neuron, a dynamical equation for $v(t)$ can be derived from elementary circuit theory:

$$c\frac{dv}{dt} = -I_m + I_b \tag{1}$$

where c and I_m are the capacitance of the neuron and current across the membrane, respectively. The inhomogeneous term I_b accounts for any external sources of current, e.g. an electrode. By convention, I_m is positive when flowing from the neuron towards the extracellular medium [5].

What determines I_m? Electric current can only flow through ion channels. These open or close depending on several factors, including the membrane potential v itself and chemical signals from the extracellular medium, such as neurotransmitters. Thus, in order to solve equation 1, the dynamics of the channels have to be specified. Many ion channels can be characterised a conductance g_i and a reversal potential E_i, satisfying that when $v = E_i$, all channels of that type are closed and no current flows through them. Small deviations from equilibrium have a linear effect on the current. Thus, we can write the total current as $I_m = \sum_i g_i(v - E_i)$, where the sum is taken over channel types. The conductances g_i will in general vary over time depending on the factors cited above. Much of the rich dynamics of neurons originates from nonlinearities in these time-dependent conductances [5].

The most relevant feature of these dynamics is the action potential or spike, a sudden surge in the membrane potential followed by a rapid decrease and a short refractory period

during which the potential remains approximately constant. The physical principles giving rise to this phenomenon are well understood and models similar to the one outlined above agree with observation to a high degree of accuracy. The most complete example of these family of models is perhaps the Hodgkin-Huxley model [11], which earned its proponents the Nobel Prize in Physiology in 1963.

Although theoretically appealing, this model involves a set of differential equations for four state variables (the voltage plus three other variables describing the state of the ion channels) that cannot be solved analytically. For that reason, simplified models have been proposed. Some notable examples include Izhikevich neurons [12], which simplify the Hodgkin-Huxley model making use of bifurcation theory; theta neurons [8], an application to neuroscience of a more general model of systems exhibiting bursting dynamics; or leaky integrate-and-fire (LIF) neurons [15], whose simplicity make them a good choice for the present work.

LIF neurons are modelled as simple RC circuits with capacitance c and conductance g, satisfying

$$c\frac{dv}{dt} = g(v_{rest} - v) + I_b(t) \qquad \text{if } v < v_{th} \tag{2}$$

plus the prescription that when the potential goes above a certain threshold v_{th}, a spike is fired.

Equation 2 governs the subthreshold dynamics. The membrane current through all ion channels is grouped into a single term $g(v_{rest} - v)$, usually denoted *leaky* current. The term $I_b(t)$ accounts for any external current sources. In the absence of external current, the potential relaxes to its rest value v_{rest} with a characteristic membrane time constant, $\tau_m = c/g$. If the potential goes above the specified threshold, a spike is fired and the potential is put to v_{reset} for a refractory time, τ_{ref}.

Spikes in LIF neurons are thus modeled as plain Dirac deltas. The sequence of spikes produced by a neuron is called a spike train. If spikes are fired at times t_k, the expression for

the spike train will be $S(t) = \sum_{t=t_k} \delta(t - t_k)$. This rough approximation works extremely well because spikes are stereotypical events whose fine grained structure is irrelevant for neuronal communication.

B. Network architectures

Having built a model neuron we can now turn to the problem of connecting several of them to form a neural network. In biological networks, neurons have long, wire-like extensions called axons and dendrites. However, these differ substantially from wires in that there is always a physical gap between them. This gap is called the synapse. A pre-synaptic neuron communicates with its post-synaptic neurons by releasing neurotransmitters in the synaptic cleft. This happens whenever the pre-synaptic neuron fires a spike. Depending on the type of neurotransmitter, a neuron can be excitatory, if the effect is to raise its post-synaptic neurons' potential and therefore the probability that they, in turn, fire a spike; or inhibitory, if the opposite happens.

A key feature of neuronal communication is that an action potential does not induce the same amount of current in every post-synaptic neuron. Some pairs of neurons couple to each other stronger than others. The strength of this coupling is different from the outset, that is, from embryo development, and it changes over the course of a lifetime. We shall characterise the strength of the synapse from neuron j to i by a synaptic weight w_{ij}. Note that a negative weight is indicative of inhibition. The structured modification of these weights forms the basis of learning, as we will discuss in the next section.

How do neurons encode information? This is a question that remains open [5]. One proposal is that spike timing (the sequence of values t_k) is irrelevant and information depends only upon the firing rate $r(t)$, that is, the number of spikes per unit time. This hypothesis originates from the fact that neural activity is inherently noisy [16]. Under this assumption, spike trains can be thought of as stochastic processes. All the information is encoded in the slow-varying $r(t)$. This is the idea behind Poisson networks. In this model, the inter-spike timing (ISI) of a single neuron, that is, the time between any two consecutive spikes, is taken

to follow a Poisson distribution of parameter r, $t_{k+1} - t_k \sim Pois(r)$. These networks have been studied extensively [5].

Taking this hypothesis one step further, if single spikes do not matter, one could forget about them altogether and consider only the behaviour of r as a function of time. Firing rate networks are based on this assumption [5]: instead of simulating single spikes, only $r(t)$ is computed. This firing rate is then fed to post-synaptic neurons by means of a sigmoid function, like the logistic function or hyperbolic tangent. This accounts for the empirical fact that the firing rate tends to increase linearly with the incoming current but only up to some saturation point, since the neuron cannot produce more spikes per second than $1/\tau_{ref}$.

Spiking networks, on the other hand, simulate the action potentials, making them a more realistic model of the brain, at the expense of being computationally more demanding. A detailed description of this model for the particular case of LIF neurons is given in section II. Suffice for now to say that the model must be supplemented with a function mapping the spike trains to the firing rates, since it is the latter what couples to the potential of the post-synaptic neurons (see equation 5).

Both in firing rate and spiking networks, the synaptic weight matrix $\mathbf{w}$ is usually taken to be sparse. This accounts for the empirical observation that the probability of any two neurons being connected is usually low [5]. $\mathbf{w}$ might be further required to satisfy Dale's Law, which states that any given neuron must be either inhibitory or excitatory to all its post-synaptic neurons [7]. This amounts to imposing that all the entries in a given column of $\mathbf{w}$ are either positive or negative. However, as we will discuss later, this constraint is difficult to accommodate in training algorithms and is therefore sometimes dropped [1, 18, 22].

When $\mathbf{w}$ is the adjacency matrix of a directed acyclic graph, the network is said to have a feed-forward architecture. In simpler words, in a feed-forward network (FFN) the neural circuit has no closed loops. Thus, neurons are arranged in layers, each one connected only to neurons in the preceding (pre-synaptic) and succeeding (post-synaptic) layer. The first and last layers are called input and output layer, respectively. The input layer contains the neurons receiving signals from the outside, e.g. sensory stimuli or another network's output; while the output layer is the one producing the response of the network to a given input

signal, e.g. a motor action. With this architecture, information flows from the input layer to the output layer in a sequential way, meaning that the state of a neuron in a given layer at any time does not depend on the state of its post-synaptic layers at previous times.

Recurrent networks (RNNs) on the other hand are allowed to have cycles. This is more realistic from the biological point of view, since this kind of feedback loops are observed in almost any biological neural circuit, but it also poses a series of difficulties for learning. These will be explored in the following subsection.

C. The problem of learning

In order for a neural network to reproduce some desired dynamics or perform a certain task, its synaptic weights must be tuned appropriately. We call *training* or *learning* the process of finding these weights by applying a suitable algorithm. Learning can be divided in three broad categories: supervised, unsupervised and reinforcement learning. We give a brief explanation of the three before turning to the concrete example of supervised learning in FFNs and RNNs.

In supervised learning, we (the *teacher*) are able to define a mapping between *input* stimuli and desired *target* behaviours. Training consists in finding the weights that minimize some loss function measuring the difference between the target for a given input and the actual behaviour produced by the network. Examples of this paradigm are classification tasks: in the training phase, the teacher provides the network with a set of inputs along with their true class. After training, the network should be able to classify new inputs without the teacher's assistance.

Unsupervised learning deals with situations where we do not have this kind of prior information matching inputs and targets. Thus, the loss function we supply the network with depends upon the inputs alone. The network is expected to infer some structure in the data that is not known beforehand. An classic example of unsupervised learning would be the problem of finding clusters of points in Euclidean space. In this case, the points come with no class tags (prior information) but they might be nonetheless grouped in discrete

clusters informed by their pairwise distances.

Finally, in reinforcement learning the network is said to take *actions*, with no initial data provided from the outside. These actions are then assigned a *reward*. This reward may also depend on a set of variables the network has no access to which are used to model the state of the environment. The goal of the network is to maximize the expected reward. Therefore, the network needs to explore configuration space and constantly adapt its strategy to that end [23].

Training feed-forward networks in a supervised manner is rather simple. To fix ideas, assume that the network's behaviour can be described by some function of time, $z(t)$, that we shall call its *output*; and the loss function is the mean squared error between output and target:

$$\mathcal{L}(z, g) = \frac{1}{T} \sum_{t=0}^{T} (z(\mathbf{w}, t) - g(t))^2 \tag{3}$$

where the dependence of the output on the set of weights has been made explicit. Hence, the goal is to find the weights w_{ij} minimising $\mathcal{L}$. This can be achieved by imposing that the gradient of the loss function vanishes:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial z} \frac{\partial z}{\partial w_{ij}} = 0 \qquad \text{for every } w_{ij} \tag{4}$$

granted that the dependence of z on the weights is differentiable. A simple yet powerful algorithm for solving this set of equations is BackPropagation [20]. Its name comes from the fact that the error is propagated backwards from the output layer to the input layer by successively applying the chain rule.

However, this approach fails when tried on recurrent networks. Why is that? First, we note that an RNN is equivalent to a FNN with infinite layers, each corresponding to a point in time, with the input layer being the whole network at $t = 0$. This trick is known as *unrolling* the RNN [19]. Using this idea, we can apply BackPropagation to the unrolled

network. Because the network state at any given time depends on all states in its past history, applying the chain rule in this case gives rise to a product of jacobian matrices where the number of factors grows exponentially in time. Thus, this term "explodes" very quickly along the directions with eigenvalues greater than one, and vanishes equally fast otherwise, making it computationally difficult to evaluate [19].

A simple solution to this problem developed in the field of machine learning are Long Short-Term Memory (LSTM) networks [10]. Their architecture is designed precisely to circumvent the vanishing/exploding gradients problem, by means of a carefully designed architecture. LSTMs have become the machine learning standard for RNNs given their good performance in a wide arrange of tasks [3]. However, this solution is not satisfactory from the biological point of view, since (i) the architecture is chosen *ad hoc*; and (ii) they do not exhibit spontaneous activity in the absence of external stimuli, as opposed to real neural networks.

A step towards a solution came from what has been called *reservoir computinig* [13, 17]. The intuition behind that term is the following: the network is a dynamical system $\dot{\mathbf{x}}(t) = F(\mathbf{w}\mathbf{x}(t))$. Like biological networks, it is capable of self-sustained activity. When perturbed by an external stimulus $f(t)$, the network dynamics change in a way that can be understood as either storing or performing a computation on the stimulus (or possibly both). The result $g(t)$ of this operation can then be recovered from the network state $\mathbf{x}(t)$ by a set of readout weights as $g(t) = \mathbf{w}^{\mathbf{out}}\mathbf{x}(t)$. The key insight of this approach is that only the readout weights need to be modified, while the recurrent weights can be chosen at random. As long as the dynamics they produce are rich enough, the network will be able to perform the desired computations and memory storage operations required for learning. What is meant exactly by *rich* is a matter of ongoing discussion, but the notion that the spontaneous activity of the network needs to be chaotic or, at least, highly heterogeneous, seems to be crucial [13].

The major breakthrough in the problem of supervised learning with biologically plausible RNNs was the development of First-Order Reduced and Controlled Error (FORCE) learning [22]. Building upon the reservoir computing paradigm, it introduced a slight but important modification: adding feedback signal from the network output $z(t)$ back to the network. This allows the network to sample its own internal dynamics and correct them in order to

perform the desired computation. This option had been considered earlier but discarded since feeding back the error usually made the dynamics unstable and spoiled the possibility of learning. The novelty introduced by FORCE consists on reducing the error very rapidly in the first iterations and keeping it low throughout the whole learning phase, solving the feedback problem.

Since its publication eight years ago, FORCE has been successfully used to train rate networks [1]. Spiking networks on the other hand have traditionally proven harder to train. One of the main reasons for this is the lack of a differentiable measure of the error for the inherently discontinuous dynamics of spike trains. Hence, gradient-descent methods are doomed when applied to spiking neurons. Some other frameworks have been used, but their range of applicability is limited to tasks where the target behaviour can be expressed as a closed-form differential equation [18]. The first successful examples of FORCE being used to train spiking networks came about as recently as last year [6, 18]. We follow their lead in implementing FORCE for a recurrent network of LIF neurons. Although the FORCE algorithm could in principle be implemented in several ways, we chose to do it by means of a learning rule called Recursive Least Squares, again following [18, 22]. This learning rule will be described in the following section.

II. MODEL

A. Spiking network

We consider a network of N LIF neurons all having the same membrane relaxation time τ_m and resting potential v_{rest}, recurrently connected via the synaptic matrix $\mathbf{w}$. A fraction $0 < 1 - p < 1$ of the entries in this matrix are zero, while the rest are drawn independently from a normal distribution with mean 0 and variance G^2/Np, where G is a scaling parameter used to control the chaotic behaviour of the network (see section III).

The stimulus $f(t)$ is fed into the network through a matrix of input weights, $\mathbf{w^{in}}$. The network output $z(t)$ is read from the network state through a set of readout weights, $\mathbf{w^{out}}$,

and fed back into the network via a set of feedback weights $\mathbf{w^{fb}}$. The sets of weights $\mathbf{w^{in}}$, $\mathbf{w^{out}}$ and $\mathbf{w^{fb}}$ are drawn independently from a uniform distribution in the interval $[-1, 1]$.

For a network receiving stimulus from N_{in} channels, the model is encapsulated in the following system of ODEs:

$$\tau_m \frac{dv_i}{dt} = v_{rest} - v_i + \sum_{j=1}^{N} \left(w_{ij} + w_i^{fb} w_j^{out} \right) r_j + \sum_{j=1}^{N_{in}} w_j^{in} f_j + RI_b \qquad \text{if } \; v \leq v_{th} \tag{5}$$

plus the prescription that whenever $v_i > v_{th}$, the i-th neuron fires a spike and its potential is kept equal to v_{reset} for time τ_{ref}. The term RI_b represents a constant, positive current chosen so as to drive the equilibrium potential of the uncoupled neuron upwards from v_{rest} to v_{th}. This is needed in order to induce spontaneous activity in the network.

The variables r_i in equation 5 are the neurons' firing rates. If a neuron produces spikes at times t_i^k, then its firing rate is calculated by applying a double exponential filter to the spike train defined by:

$$\tau_r \frac{dh_i}{dt} = -h_i + \frac{1}{\tau_d} \sum_{t_i^k \leq t} \delta(t - t_i^k) \tag{6}$$

$$\tau_d \frac{dr_i}{dt} = -r_i + h_i \tau_d \tag{7}$$

where τ_r is the synaptic rise time, that is, the typical time it takes for the ion channels in the post-synaptic neuron to open after the pre-synaptic neuron has fired; and τ_d is the synaptic decrease time, measuring the typical time for the ion channels to close again. The smoothing effect of these parameters in the spike train can be appreciated in figure 1.

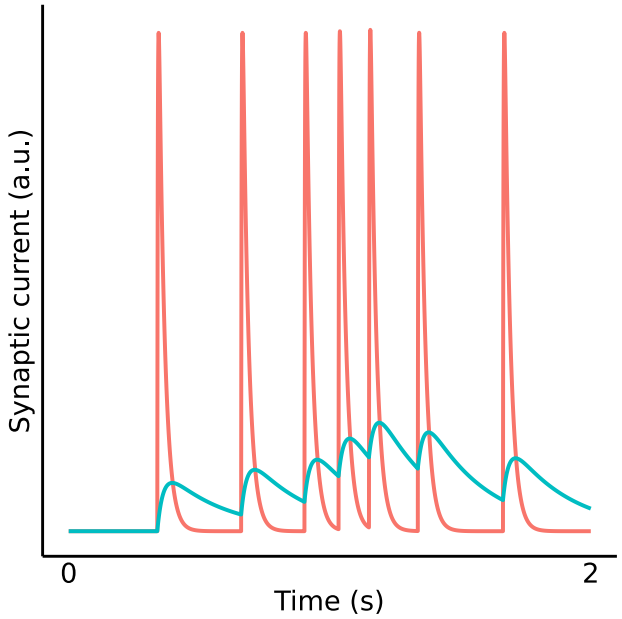

FIG. 1: The same spike train is filtered with different synaptic time constants. Larger synaptic times result in smoother signals, making the network trajectory more similar to that of continuous models such as firing rate networks. For the red trace, $\tau_d = 20$ ms and $tau_r = 2$ ms; and for the blue trace, $\tau_d = 200$ ms and $\tau_r = 20$ ms.

B. Learning algorithm

We used the RLS algorithm in order to find the optimal readout weights $\mathbf{w^{out}}$ without modification of the recurrent weights $\mathbf{w}$. The weights computed with this algorithm are an unbiased estimator of the optimal weights, that is, the ones minimising the mean squared error between output and target. See [9] for a detailed description of the algorithm and its properties. RLS is suitable for FORCE learning because it reduces the error in the output early in time. This is due to the fact that it works on line, that is, while the simulation runs.

The RLS learning rule updates the readout weights every Δt seconds as:

$$\mathbf{w^{out}}(t) = \mathbf{w^{out}}(t - \Delta t) - e(t)\mathbf{r}^T(t)\mathbf{P}(t) \tag{8}$$

with all the weights initially set to zero, $e(t) = \mathbf{w^{out}}(t - \Delta t)\mathbf{r}(t) - g(t)$ the error before the

weight update, and $\mathbf{P}$

$$\mathbf{P}(t) = \mathbf{P}(t - \Delta t) - \frac{\mathbf{P}(t - \Delta t)\mathbf{r}(t)\mathbf{r}^T(t)\mathbf{P}(t - \Delta t)}{1 + \mathbf{r}(t)\mathbf{P}(t - \Delta t)\mathbf{r}(t)} \tag{9}$$

with $\mathbf{P}(0) = \lambda^{-1}\mathbf{1}$.

The matrix $\mathbf{P}$ is a running estimate of the inverse of the correlation matrix $\Sigma_{ij} = \mathrm{Corr}(r_i, r_j)$ plus a regularization term $\lambda\mathbf{1}$. λ takes the role of a learning rate: a larger value makes weight changes smaller during the first iterations. Some heuristic arguments for the choice of λ can be found in [22].

We note that, in general, the resulting effective synaptic matrix, $\mathbf{w} + \mathbf{w^{fb}}\mathbf{w^{out}}$, satisfies neither the sparseness condition, nor Dale's Law. Trying to impose those constraints by hand as the algorithm runs would probably spoil learning, so it is not clear how to accommodate them. See [1, 18] for two tentative approaches.

The simulation is carried out by numerically integrating equations 5-7 using the Euler (first order Runge-Kutta) method, while applying the RLS rule of equation 8 every $\Delta t/dt$ steps in the learning phase. Unless otherwise stated, the network parameters are those summarised in Table I. The simulation was written in Armadillo [21], a C++ library for fast, parallel matrix multiplication, while the analysis of the results was done with MATLAB.

III. RESULTS

A. Analysis of the spontaneous activity

First, we wanted to investigate the effect of the coupling parameter G in the spontaneous activity of the network. To that end, we computed the population-averaged autocorrelation function (ACF) of the firing rates, $\mathbf{r}(t)$ at different time lags for several values of G:

Neuron		Network	
τ_m	20 ms	N	2000
v_{th}	-40 mV	p	0.1
v_{reset}	-65 mV	G	0.015
v_{rest}	$-65\ mV$	Integration	
RI_b	15 mV	λ	5 s
τ_r	20 ms	dt	0.5 ms
τ_d	200 ms	Δt	2.5 ms

TABLE I

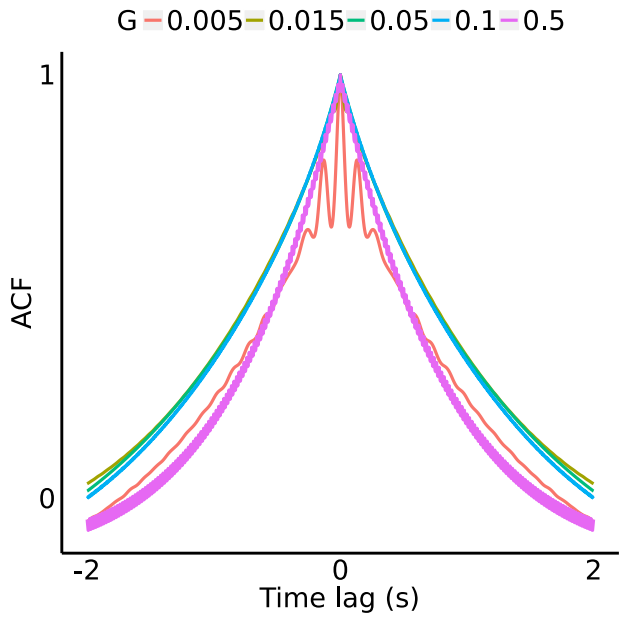

FIG. 2: The spontaneous activity (no input or feedback) of the network was simulated for different values of the coupling parameter G. For each value of G, the simulation was 10 seconds long. The firing rates $\mathbf{r}(t)$ were recorded and their autocorrelation functions (ACF(τ)) at different time lags computed and averaged over neurons (see equation 10). Values of G in the intermediate range of those explored yielded larger autocorrelation timescales.

$$\mathrm{ACF}(\tau) = \frac{1}{N}\sum_{i=1}^{N} \frac{\langle r_i(t-\tau) r_i(t)\rangle - \langle r_i(t)\rangle^2}{\langle r_i^2(t)\rangle - \langle r_i(t)\rangle^2} \tag{10}$$

where angular brackets denote time average.

The width of the ACF gives the typical timescales at which the network operates. It is crucial that the network has large timescales for learning to happen. If the spontaneous activity were completely uncorrelated, the effect of a stimulus in the reservoir would fade almost immediately, rendering memory impossible. We found that values of G ranging from 0.015 to 0.5 resulted in the largest full width at half maximum (see figure 2). We chose to do our experiments with the smallest value in that range because the resulting output was less noisy, so target dynamics were reproduced more faithfully.

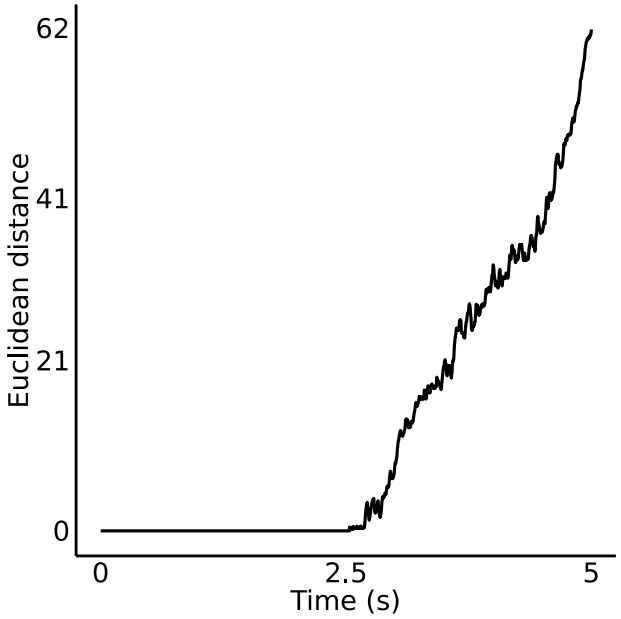

FIG. 3: A simulation of the network with no input or feedback was run for 2.5 seconds. At that time, the network state was perturbed by deleting a single spike chosen at random. The simulation was then run for the perturbed and unperturbed states for another 2.5 seconds. In the figure, the Euclidean distance between both neural trajectories $\mathbf{r}(t) \in \mathbf{R}^N$ is shown. The two states are seen to diverge in time, indicating that the network dynamics tend to amplify small perturbations.

A necessary condition for the network activity to be heterogeneous is that small perturbations lead to large divergences in the neural trajectory $\mathbf{r}(t) \in \mathbf{R}^N$. An informal way of testing this is perturbing the network state by deleting a single spike and then comparing the perturbed trajectory, $\widetilde{\mathbf{r}}(t)$, with the unperturbed one [16, 18]. Having set $G = 0.015$, we ran a new simulation and performed this test. The result of the test is shown in fig-

ure 3. It can be seen that the Euclidean distance between the unperturbed and perturbed trajectories $d(\mathbf{r}(t), \widetilde{\mathbf{r}}(t)) = \sqrt{\sum_{i=1}^{N} \left(r_i(t) - \widetilde{r}_i(t)\right)^2}$ grows in time immediately after the spike deletion, confirming the network dynamics amplify rather than damp small perturbations. As discussed above, this is key for the reservoir computing approach to work.

B. Function approximation

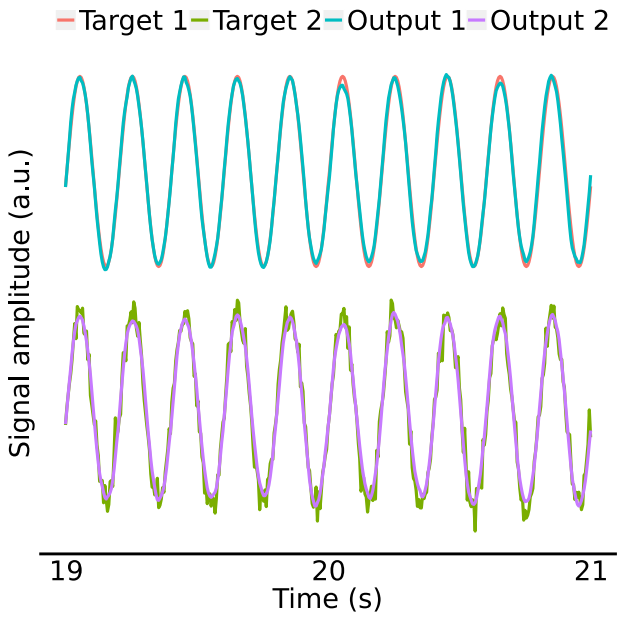

FIG. 4: The network was trained to produce a 5 Hz sinusoidal signal (top traces) and that same signal corrupted with uncorrelated Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 0.1$ (bottom traces) for 20 seconds using the RLS learning rule. After that time, the last readout weights were kept and 20 more seconds of simulation were carried out to test the performance of the trained network. The network was able to reproduce the target signal to a high degree of accuracy, indicating that it had learned the desired dynamics. After a few more seconds (not shown) a slight drift in the output's frequency was perceptible.

Having chosen an appropriate value for G, we checked that the network could produce some target dynamics autonomously, that is, again in the absence of any stimulus. Failing to do so would mean that the recurrent dynamics are too poor for learning to happen. A simple choice for these target dynamics usually found in the literature [18, 22] are sinusoidal

functions.

Figure 4 shows that the network was able to reproduce a 5 Hz sine wave with a high degree of accuracy, even when it was corrupted with Gaussian noise. The training phase took 20 seconds of simulation time to complete. After that time, the simulation was run with no further modification of the readout weights to check that the network kept producing the desired dynamics on its own. We noted however that the frequency of the output signal drifted slightly for greater times (not shown).

C. Detection task

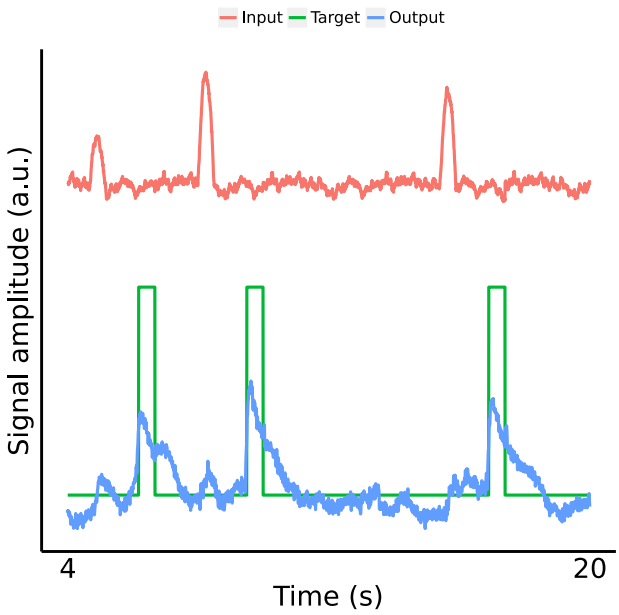

FIG. 5: The network was trained to detect the presence of a stimulus of variable amplitude over a noisy background generated by an Ornstein-Uhlenbeck process with mean $\mu = 0$, standard deviation $\sigma = 0.1$ (both in units of the signal's amplitude) and correlation time $\tau = 300$ ms. The training set consisted of 100 trials, presented in a random order. Learning was tested on a different set of 100 trials. In the figure, stimulus (top trace), target (bottom, green) and output (bottom, blue) for four consecutive test trials are shown. Note that the stimulus is absent in the third trial, hence the network correctly produces no response. Although the network does not reproduce the target waveform exactly, the decision signal is nonetheless distinguishable and the network did learn the detection task.

Having shown that the network could learn some simple dynamics we turned to the problem of training it to do more complex tasks. These were inspired in laboratory experiments done with model animals, such as rodents or monkeys. Experimentalists train these animals to respond to stimuli in specific ways, frequently involving memory and decision-making. Electric activity of single neurons is recorded during the experiment and is later interpreted in relation to the behavioural outcomes that were observed.

The first experiment we considered is a detection task described in previous work [4], where it was modeled using a firing rate network. In each trial, the network was presented a stimulus consisting of half a cycle of a 1 Hz sine wave of variable amplitude (see figure 5) over a noisy background in half the trials, and no stimulus (only noise) in the other half. The order of the trials was chosen at random. For each trial, the network had to produce a square 500 ms signal if the stimulus was present and no response if it was absent.

Two additional input channels were fed into the network, one indicating the onset of a trial and another one indicating the time when the decision had to be expressed. These channels simulate visual or auditory cues that are commonly used in behavioural experiments with the same purpose. We noted that their use was crucial, since the network did not learn the task if they were absent. The functions used to encode input and target were chosen to be the same as in [4].

After 100 training trials, the network was presented a different set of 100 test trials. Both in the learning and training phase, random delays ranging from 1 to 2 seconds were added between trials. It was observed that the network was not able to fully reproduce the target waveform in the testing phase (see figure 5) but that there was nonetheless a response. In order to confirm that this response was significant, we extracted the maximum output amplitude $z(t)$ at the time windows where the decision had to be communicated (last 500 ms of each trial) and compared the distribution of these values between trials where the stimulus was present with those where it was absent. The results are shown in figure 6. A ROC analysis yielded a value for the area under the curve (AUC) of 0.98, indicating the difference was indeed significant.

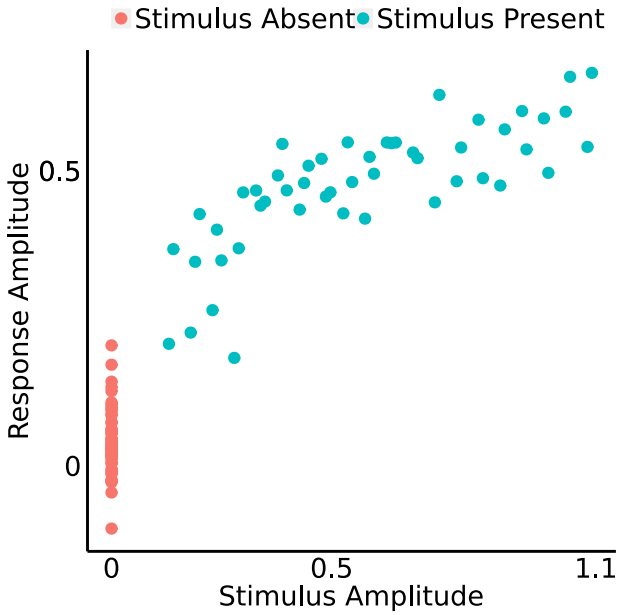

FIG. 6: Stimulus amplitude and response amplitude for the detection task test trials. The response amplitude is measured as the maximum of the output signal $z(t)$ during the response window, i.e. the last 500 ms of each trial. The distribution of the response for trials where the stimulus was absent (red) and trials where it was present (blue) are significantly different (AUC = 0.98).

D. Discrimination task

Having checked that the network was able to detect the presence of stimuli, we wanted to see if it was also capable of discriminating them. To that end, we trained the network to perform a task usually denoted temporal XOR. It consists on telling whether two stimuli are identical or not. The stimuli are presented at different times, thus requiring that the network stores the duration of the first one in memory at least until the second one is presented. Many cognitive experiments done with animals follow this pattern [2]. The stimuli were encoded as square signals of equal amplitude, each having a fixed length of either 100 ms or 300 ms, with a delay in between (see figure 7). The network had to respond with half a cycle of a 1 Hz sine wave, its sign depending on whether the inputs were equal (up state) or different (down state). We chose this encoding scheme after [1]. As in the detection task, channels indicating the onset of the trial and the decision were used.

In the first experiment, the delay between consecutive trials, the inter-stimulus delay and

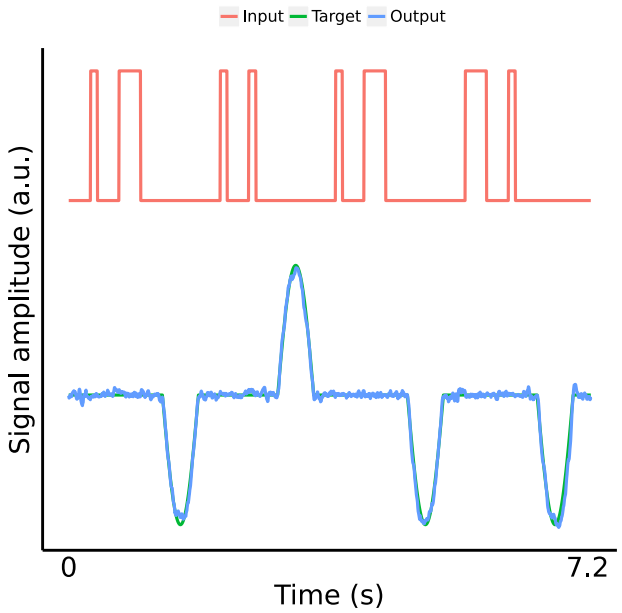

FIG. 7: The network was trained to perform the temporal XOR task on a set of 100 trials with inter-trial, inter-stimulus and stimulus-response delays all fixed and equal to 300 ms. As in the previous task, trials were presented in a random order and learning was tested with a different set of 100 trials. In the figure, stimulus (top trace), target (bottom, green) and output (bottom, blue) are shown for four consecutive test trials. Note that the target signal is barely visible since there is almost perfect overlap between it and output. The network failed to correctly classify only 3% of the 100 test trials.

the delay between stimulus presentation and decision, were all kept constant and equal to 300 ms. With this setting, the network was able to correctly classify 97 of the 100 trials in the test set. In order to check if this result was robust to randomness in the timing structure of the trials, we did a second experiment introducing random delays ranging from 1 to 2 seconds, both between consecutive trials and between input presentation and decision without further training under the new condition. The performance of the network dropped to 69 out of 100 correct decisions in the test set, notably lower but still significantly ($p < 5 \cdot 10^{-5}$) above chance.

In order to test the limit timescale at which the network could retain a stimulus in memory, we did a set of experiments increasing the inter-stimulus delay in the temporal XOR task. For each experiment, the delay was the same in the training and testing phase, with no randomness in any time interval. The network's performance stayed above 90% for

inter-stimulus delays shorter than 1 second (see figure 8), but it dropped for longer delays, ceasing to be significantly different from chance at the $p < 0.01$ level for delays greater than 3.5 seconds.

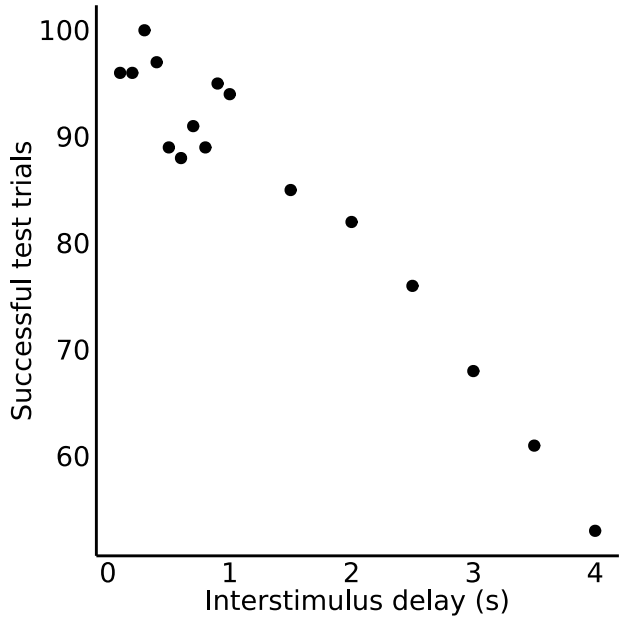

FIG. 8: The network was trained to perform the temporal XOR task on a set of 100 trials for different values of the inter-stimulus delay. For delays below 1 second, the network was able to classify correctly around 90 out of the 100 test trials. For larger delays, performance dropped, indicating a failure to retain the first stimulus in memory. Performance for delays longer than 3.5 seconds is not significantly different from chance ($p > 0.01$).

Finally, we wanted to investigate how the network encoded its decision in the temporal XOR task. To that end, we computed the principal components of the neural trajectory $\mathbf{r}(t) \in \mathbf{R}^N$ across the whole test phase. We found that the first two principal components were indeed easy to interpret (see figure 9). The first one was selective to the stimulus, with the interesting result that of having encoded the temporal input code into an amplitude code in the output, while the second one tracked the decision.

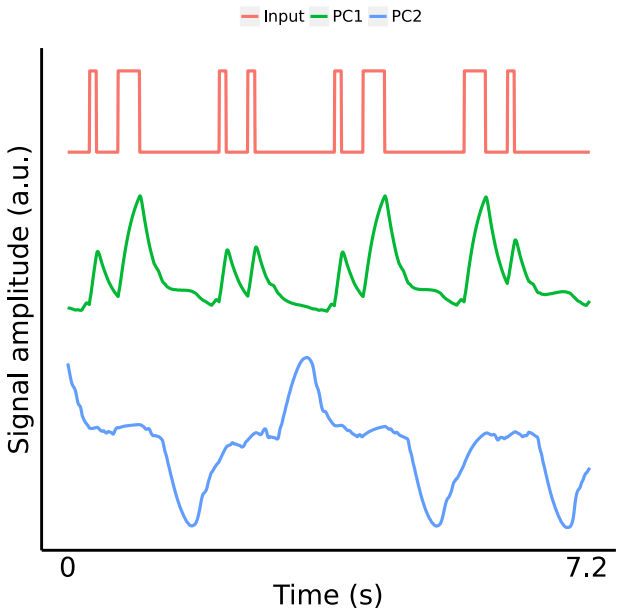

FIG. 9: Stimulus (top trace) and two first principal components of the neural trajectory $\mathbf{r}(t)$ (middle and bottom traces) for the same four consecutive test trials shown in figure 9. The principal components were computed pooling together all the data from the 100 test trials. The first principal component encodes stimulus duration as amplitude, while the second principal component correlates with the decision.

IV. CONCLUSIONS

Artificial neural networks provide both a powerful tool in the field of machine learning and a rigorous framework for investigating the nature of animals' brains. In the latter area, numerical simulations of biologically plausible networks have been an active field of research for several decades. However, the study of learning has until recently been confined to either firing rate networks or feed-forward architectures, given the difficulty of training recurrent networks of spiking neurons. In the present work, we have shown that these networks can be trained, adding to the recent developments in the literature [1, 6, 18]. Further research is needed in order to fully understand the dynamics of these networks and how the model parameters, in particular the nature of the regime of the untrained initial network, impact the final results.

[1] L. Abbott, B. DePasquale, and R. Memmesheimer. Building functional networks of spiking model neurons. *Nature neuroscience*, 19(3):350–355, 2016.

[2] O. Barak, D. Sussillo, R. Romo, M. Tsodyks, and L. Abbott. From fixed points to chaos: three models of delayed discrimination. *Progress in neurobiology*, 103:214–222, 2013.

[3] T. Breuel. Benchmarking of lstm networks. *arXiv preprint arXiv:1508.02774*, 2015.

[4] F. Carnevale, V. de Lafuente, R. Romo, O. Barak, and N. Parga. Dynamic control of response criterion in premotor cortex during perceptual detection under temporal uncertainty. *Neuron*, 86(4):1067–1077, 2015.

[5] P. Dayan and L. F. Abbott. *Theoretical neuroscience*, volume 806. Cambridge, MA: MIT Press, 2001.

[6] B. DePasquale, M. M. Churchland, and L. Abbott. Using firing-rate dynamics to train recurrent networks of spiking model neurons. *arXiv preprint arXiv:1601.07620*, 2016.

[7] J. C. Eccles, P. Fatt, and K. Koketsu. Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones. *The Journal of physiology*, 126(3):524, 1954.

[8] G. B. Ermentrout and N. Kopell. Parabolic bursting in an excitable system coupled with a slow oscillation. *SIAM Journal on Applied Mathematics*, 46(2):233–253, 1986.

[9] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 2002.

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

[11] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[12] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[13] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks - with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2010.

[14] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness`, 2015.

[15] L. Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J Physiol Pathol Gen*, 9:620–635, 1907.

[16] M. London, A. Roth, L. Beeren, M. Häuser, and P. E. Latham. Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, 466(7302):123–127, 2010.

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

[18] W. Nicola and C. Clopath. Supervised learning in spiking neural networks with force training. *arXiv preprint arXiv:1609.02545*, 2016.

[19] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[20] D. E. Rumelhart, J. L. McClelland, P. R. Group, et al. *Parallel distributed processing*, volume 1. IEEE, 1988.

[21] C. Sanderson and R. Curtin. Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26–32, 2016.

[22] D. Sussillo and L. F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

[23] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.