Master thesis report

Statistical physics of learning in recurrent neural networks

Submitted by

Samantha Fournier
Student of the master
Physics of Complex Systems

Under the guidance of

Pierfrancesco Urbani
Researcher at
Institut de Physique Théorique
CEA, Orme des Merisiers Bat 774, 91191 Gif-sur-Yvette

Date: June 2023

Contents

Introduction

Artificial Neural Networks (ANN) are computing devices inspired from the brain, in which neurons (i.e. real variables) are connected between themselves with certain weights, and we wish to tune those weights to perform certain tasks. ANN were first invented acknowledging that the brain works in an entirely different way than the computer does. The aim was therefore to understand and eventually harness the brain's computational power.

To give some historical context, the first model of a biological neuron – the perceptron – was invented in 1943 by McCulloch and Pitts, and then improved upon by Rosenblatt in 1957. The multi-layer perceptron (MLP) was created soon after, by stacking layers of neurons with a feed-forward architecture. Here, the analogy with the brain was somewhat lost because biological neural networks do not have a feed-forward architecture (for the most part). But this was, first, a legitimate attempt, and second, justified by the MLP's performances. Contrary to its single-layer counterpart, the MLP could indeed do non-linear classifications. And the simpler setting of a feed-forward architecture allowed for an efficient training method with Gradient Descent (GD), called the Backpropagation algorithm. This algorithm (dating back to the 70's) uses the network's forward architecture to solve the uneasy task of computing gradients of synaptic weights in the hidden layers of neurons. With the advent of the Internet in the 90's and the multitude of data it provided for tuning a large number of parameters, the MLP took off and became – to this day – ubiquitous in Artificial Intelligence.

This was all well and good from a performances standpoint, but what about understanding how the brain works? As mentioned previously, biological neural networks do not have a feed-forward architecture: connections can go backwards and, in particular, form loops. This type of neural networks is called Recurrent Neural Networks (RNN) and is much harder to train. Indeed, one usually trains neural networks by defining a cost function parameterized by the synaptic weights, and then minimizes this cost function with GD. But the presence of loops induces instabilities in the gradients: on a loop, gradients can be either amplified or diminished, leading to diverging or vanishing gradients.
Some algorithms were however invented to train RNNs with GD. Because of loops, one has in principle to take into account the echoing effects of synaptic modifications, and thus needs to compute gradients through time. The two main algorithms that accomplish this are Real-Time Recurrent Learning (RTRL) and Back-Propagation

Through Time (BPTT). Both algorithms are based on the realization that every RNN is forward in time. RTRL is an on-line technique that deals with the indirect effects of weight modifications by simulating them forward in time; while BPTT lets the network evolve in batches, and propagates backwards the errors at the end of each batch to update the synaptic weights. Nevertheless, it is fair to say that these algorithms are not very efficient, and are not biologically plausible.

In order to make progress in our understanding of how the brain works, a new line of research at the frontier between neuroscience and physics is being carved. The goal for physicists working on this topic is to invent algorithms for training RNNs that are biologically plausible. In that spirit, recent advances have been made.
In 2009, Sussillo and Abbott developed a procedure to train a RNN that exhibits spontaneous chaotic activity to generate desired patterns [1, 2]. Rather than progressively decreasing an error function as in GD, their approach is to keep the error small from the start, so that delayed effects due to the loopy architecture of the network can be neglected. They coined their method First-Order Reduced and Controlled Error (FORCE). Although they worked with neurons modelled as smooth functions, Nicola and Clopath [3] recently showed that the FORCE algorithm also works for *spiking neurons*, which are more realistic. Even more recently in 2017, Miconi [4] created a learning algorithm that checks a lot of boxes for biological plausibility: it uses a Hebbian learning rule (the same that the brain supposedly uses) and it works with sparse and delayed rewards.
However, these algorithms have been discovered heuristically, and a thorough theoretical analysis remains to be performed. Here lies an opportunity for the physicist specialized in statistical mechanics to step in and add insights to the discussion.

Indeed, ANNs find a natural formulation in the language of statistical mechanics because, at its core, training an ANN amounts to minimizing a cost function in a space of high dimensions. The first to make the link with statistical physics was Hopfield, with his seminal work of 1982 and 1984 [5, 6]. Hopfield reformulated the problem of training a RNN as a spin-glass problem. He showed how one could write the *symmetric* synaptic weights of the network, so as to embed memories as fixed point attractors. The memories could then be retrieved by initializing the network with a noisy version of that memory, meaning in its basin of attraction.

The objective for this internship is therefore to study algorithms to train RNNs with the tools of statistical physics. Specifically, we will study the FORCE learning algorithm. We will try to write a simple model of a RNN, to then analyze it with Dynamical Mean Field Theory (DMFT). The work performed during this internship has lead to the creation of an article that is in the process of being published (see ref.[12]).

Chapter 1

Methods

1.1 Standard Model

A recurrent neural network (RNN) is usually modelled by a set of dynamical equations for the activities $x_i \in \mathbb{R}$ of the neurons, akin to their membrane potential. In the simplest setting, one considers $i = 1, ...N$ neurons evolving according to

$$\tau \frac{dx_i}{dt} = -x_i(t) + \frac{g}{\sqrt{pN}} \sum_{j=1}^{N} J_i^j r_j(t), \tag{1.1}$$

where $r_i = \tanh(x_i)$ is a non-linear function of the x_i, modelling the neuronal firing rate. J_i^j is the synaptic strength between neuron i and j. The J_i^j's are fixed and independently identically distributed as $J_i^j \sim \mathcal{N}(0, 1)$. p is the sparsity of the matrix J. τ is the neuronal time constant assumed to be the same for all neurons. g is the coupling strength.

In the thermodynamic limit, the system undergoes a transition from a non-chaotic to a chaotic state at $g_c = 1$. The matrix $\frac{g}{\sqrt{pN}} J$ indeed belongs to the Gaussian orthogonal ensemble (GOE) so the distribution of its eigenvalues will converge to a semi-circle of radius g as $N \to \infty$. The fixed point $\underline{x} = \underline{0}$ is therefore linearly stable if $g < 1$, and if $g > 1$, it has been shown that there are positive Lyapunov exponents [7].

As mentioned in the **Introduction**, the tricky part about training this network is the presence of feedback loops. One way to go around this issue is to add a linear readout to the network and train the readout weights – with no feedback. Such networks are called Liquid-State or Echo-State Machines. They use the random, chaotic and non-linear part of the network as a "reservoir" to do transformations or classifications on inputs, but they are not well-suited for pattern generation [2]. Instead, one would like to solve the truly recurrent problem – where the output is fed back into the network – to do pattern generation. In doing so, one has to deal with the fact that a weight modification that first appeared to bring the output closer to its desired value, can later on have uncontrolled effects, detrimental to training.

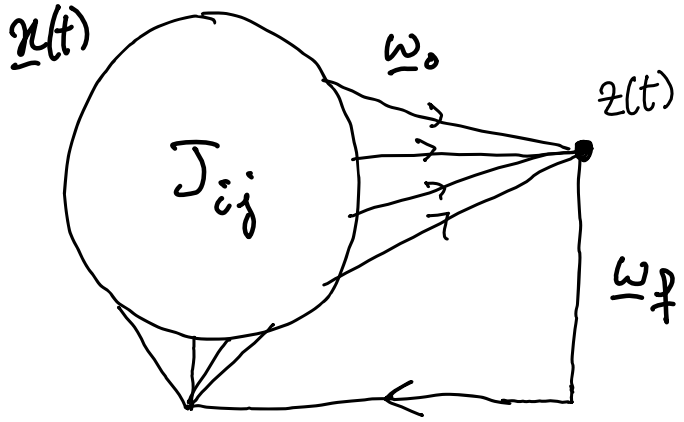

Figure 1.1: Standard Network configuration for FORCE learning.

Sussillo and Abbot found a simple and elegant solution to the problem, which we will discuss next.

1.2 FORCE Learning

A simple way to look at the activity of the whole network is to add a linear readout, defined as

$$z(t) = \sum_{i=1}^{N} w_{oi} r_i(t).$$

This output neuron $z(t)$ is then fed back into the network (see fig.1.1), so the dynamical equations now read

$$\tau \frac{dx_i}{dt} = -x_i(t) + \frac{g}{\sqrt{pN}} \sum_{j=1}^{N} J_i^j r_j(t) + w_{fi} z(t) \tag{1.2}$$

where $\underline{w}_o$, $\underline{w}_f$ are respectively the output and feedback weights. The w_{fi}'s are chosen randomly and independently from a uniform distribution in $[-1, 1]$. To make things easier, only the output weights $\underline{w}_o$ are trained, although one could also train the whole matrix of connections J [2]. The goal of training is to learn a pre-defined function f, so that after learning $z(t) \simeq f(t)$.

The idea behind the FORCE learning algorithm is that the quantity $z(t) = f(t) + \epsilon(t)$ is fed back into the network, so delayed effects due to the loopy nature of the network will be of order $\epsilon(t)$. Therefore, if the error $e(t)$ that drives weight modifications is much larger than $\epsilon(t)$, delayed effects can be neglected and weights can be updated by a simple gradient descent. The proposed rule to update the output weights $\underline{w}_o$ thus reads

$$\underline{w}_o(t) = \underline{w}_o(t - dt) - \eta(t) e(t) \underline{r}(t) \tag{1.3}$$

and will work only if $e(t) \gg \epsilon(t)$ during the whole training phase.

Defining $z^+(t-dt) = \underline{w}_o^T(t-dt)\underline{r}(t)$ and $e(t) = z^+(t-dt) - f(t)$, we get using eq.(1.3) that

$$z(t) = \underline{w}_o(t) \cdot \underline{r}(t) = z^+(t-dt) - \eta(t)e(t)\underline{r}^T(t)\underline{r}(t) = f(t) + \epsilon(t),$$

so

$$\epsilon(t) = e(t)[1 - \eta(t)\underline{r}^T(t)\underline{r}(t)],$$

and delayed effects will be negligible if $1 - \eta(t)\underline{r}^T(t)\underline{r}(t) \ll 1$, so for a well-chosen learning rate $\eta(t)$. In fact if $\eta(t) = 1/\underline{r}^T(t)\underline{r}(t)$, then $\epsilon(t) = 0$ at all times. But this is not good because if $z(t)$ always matches $f(t)$ perfectly, the system never explores the phase space enough to find a stable attractor, and therefore never really learns $f(t)$. Instead in ref. [2], Sussillo suggests to initialize $\eta(t)$ close to $1/\underline{r}^T(t)\underline{r}(t)$ so that $\epsilon(t) \ll e(t)$ from the start – but to then relax this condition, letting $\eta(t)$ decrease slowly as $|e(t)| \to 0$ and learning proceeds. The proposed η-schedule is

$$\tau\frac{d\eta}{dt} = -\eta^2(t) + \frac{|e(t)|^\gamma \eta(t)}{\tau}, \tag{1.4}$$

where γ is a parameter tuned empirically. This was the first version of the learning algorithm (a.k.a. FORCE-I). While it has the advantage of being simple and somewhat plausible for biological systems, the FORCE-I learning algorithm is very slow to converge and sensitive to initialization.

In order to find a way of improving performances, Sussillo and Abbott looked at the averaged correlation matrix of network activities $C = \langle \underline{r}(t)\underline{r}^T(t)\rangle_t$, and noticed that the system's trajectory during learning was dominated by a few principal components of C. They consequently guessed that learning could be made faster in the directions of large eigenvalues and proposed to replace $\eta(t)$ by multiple learning rates, given by the matrix $P(t)$. The update rule for $P(t)$ is given by eq.(1.5), which constitutes the optimized version of the algorithm, a.k.a. FORCE-II [1, 2].

$$\begin{aligned} P(0) &= I/\alpha \\ P(t) &= P(t-dt) - \frac{P(t-dt)\underline{r}^T(t)\underline{r}(t)P(t-dt)}{1 + \underline{r}^T(t)P(t-dt)\underline{r}(t)} \end{aligned} \tag{1.5}$$

where I is the identity matrix. Defined in this way, $P(t)$ converges to $(\sum_t \underline{r}^T(t)\underline{r}(t) + \alpha I)^{-1}$, i.e. the inverse of C plus a regularization term [8]. Assuming t is large enough for $P(t)$ to have converged to the aforementioned value, the learning rate in the direction of the eigenvector $\underline{a}$ is $P_a(t) = (\lambda_a t + \alpha)^{-1}$, where the λ_a's are the eigenvalues of C. Thus we see that learning is divided into two stages [1]. If $t < \alpha/\lambda_a$, then $P_a(t) \approx 1/\alpha > 0$ and learning serves to keep the error $\epsilon(t) = z(t) - f(t)$ small. But once $t > \alpha/\lambda_a$, $P_a(t) \to 0$, so w_{oa} reaches a stationary value and learning in the

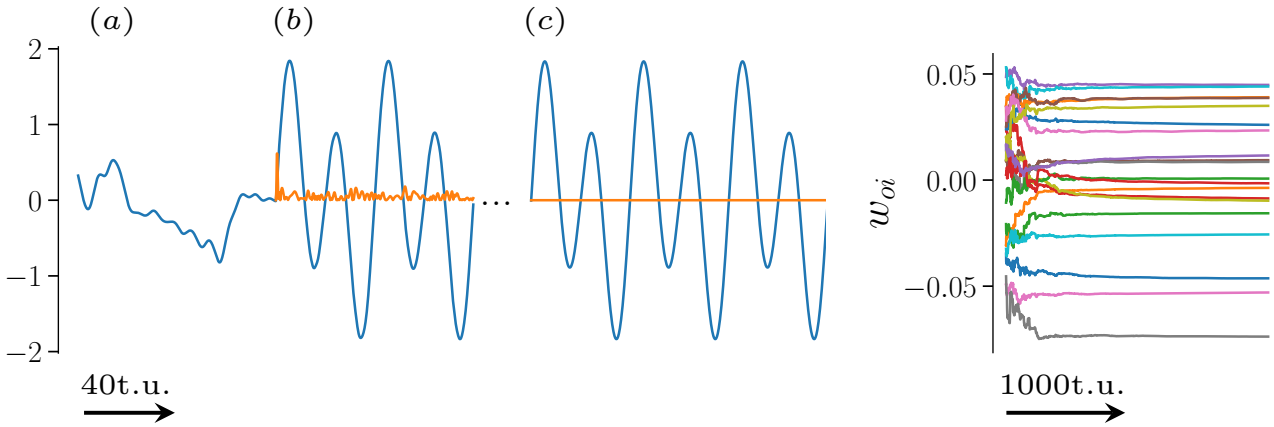

Figure 1.2: Example of FORCE-II learning. *Left panel:* network output $z(t)$ in *blue*, weight update $\|\underline{dw}_o(t)\| = \|\underline{w}_o(t) - \underline{w}_o(t-dt)\|$ in *orange*. (a) prior to training, the output $z(t)$ is chaotic. (b) during training, $z(t)$ closely matches $f(t)$. The weight update is initially big to bring $z(t)$ close to $f(t)$, then decreases as the weights $\underline{w}_o(t)$ reach a stationary value. In the plot, $\|\underline{dw}_o(t)\|$ has been re-scaled by a factor 20. (c) after training, the weights $\underline{w}_o$ are fixed and the network autonomously outputs the function $f(t)$. *Right panel:* trace of 20 randomly picked weights w_{oi} during training. The parameters used for the simulation are: $N = 1000$, $p = 0.1$, $\tau = 1$, $dt = 0.1$, $g = 1.5$ and $\alpha = 1$. The function learned is $f(t) = 0.67\sin(0.05\pi t) + 1.34\sin(0.1\pi t)$. Training lasted 2000 time units (t.u.), at the end of which $\|\underline{dw}_o(t)\| \approx 10^{-5}$.

direction $\underline{a}$ stops. So the advantage of introducing one learning rate P_a for every eigen-direction $\underline{a}$ is that learning will be fast in the directions where λ_a is large, while learning in the directions with small λ_a will keep $\epsilon(t)$ small and ensure stability. In passing, we can note that the constant α also influences the speed of learning: the smaller α, the faster the learning – but if α is too small, weight updates can overshoot and learning can become unstable.

The FORCE-II algorithm is consequently much more efficient than its previous version. But it is less biologically plausible since, to update w_{oi}, it requires non-local information in the form of the correlations of the activities of the whole network.

In practice, eq.(1.2) is Euler-discretized and learning proceeds until $\underline{w}_o$ reaches a stationary value. The network needs to be chaotic prior to training ($g > 1$). The more chaotic, the better, as learning will converge faster – up to a point ($g \approx 1.6$) where the network is so chaotic that learning fails [2]. The function $f(t)$ to be learned must be of sufficient amplitude and frequency to suppress chaos during learning [9].

1.3 Dynamical Mean Field Theory (DMFT)

The appropriate tool from statistical physics to study RNNs is Dynamical Mean Field Theory (DMFT), which maps the high-dimensional dynamical equations describing the neurons' activities onto an effective single-neuron picture that is exact in the $N \to \infty$ limit. This section gives a summary of the method based on ref.[10].

Consider first the one-dimensional Langevin equation

$$\frac{dx}{dt} = -\frac{\partial H_0}{\partial x} - \frac{\partial H_J}{\partial x} + \eta(t), \tag{1.6}$$

where H_J is the part of the Hamiltonian with quenched disorder, and $\eta(t)$ is a Gaussian noise with average 0 and two-times correlation function $\langle \eta(t)\eta(t')\rangle = D_0(t,t')$. Eq.(1.6) can equivalently be written

$$\mathcal{L}_0(x) + \mathcal{L}_J(x) = \eta(t), \tag{1.7}$$

with $\mathcal{L}_0(x) = \partial_t x + \partial_x H_0$ and $\mathcal{L}_J(x) = \partial_x H_J$.

Let's compute the generating functional using the Martin–Siggia–Rose–Janssen–deDominicis path-integral formalism.

$$\begin{aligned} Z_J &= \int \mathcal{D}x\, P(x) = \int \mathcal{D}x\mathcal{D}\eta\, P(\eta)\delta(\mathcal{L}_0(x) + \mathcal{L}_J(x) - \eta(t)) \\ &= \int \mathcal{D}x\mathcal{D}\hat{x}\mathcal{D}\eta\, P(\eta)\exp\left\{ i\int_t \hat{x}(t)[\mathcal{L}_0(x) + \mathcal{L}_J(x) - \eta(t)]\right\} \\ &= \int \mathcal{D}x\mathcal{D}\hat{x}\exp\left\{ -\frac{1}{2}\int_{t,t'} \hat{x}(t)D_0(t,t')\hat{x}(t') + i\int_t \hat{x}(t)[\mathcal{L}_0(x) + \mathcal{L}_J(x)]\right\}, \end{aligned}$$

where we first wrote the δ in exponential form and then performed the average over η. Now, averaging over the disorder J

$$\overline{Z_J} = \int \mathcal{D}x\mathcal{D}\hat{x}\, e^{-\frac{1}{2}\hat{x}D_0\hat{x} + i\hat{x}\mathcal{L}_0(x)}\overline{e^{i\hat{x}\mathcal{L}_J(x)}},$$

where we used a short-hand notation for the time integrals, e.g.

$$\hat{x}D_0\hat{x} = \int_{t,t'} \hat{x}(t)D_0(t,t')\hat{x}(t').$$

Assuming we can write $\overline{\exp\{i\hat{x}\mathcal{L}_J(x)\}} = \exp\{\Delta(x,\hat{x})\}$, with $\Delta(x,\hat{x}) = -\frac{1}{2}\hat{x}D_1\hat{x} + i\hat{x}\mathcal{L}_1 + \dots$, the average over the disorder J will re-normalize D_0 and $\mathcal{L}_0$. We thus end up with an effective Langevin equation

$$\mathcal{L}_0(x) + \mathcal{L}_1(x,\hat{x}) = \xi(t), \tag{1.8}$$

with $\langle \xi(t)\xi(t')\rangle = D_0(t,t') + D_1(x,\hat{x})$.

This computation can easily be generalized to the case where x is multidimensional, and we end-up with an effective single-site Langevin equation (see appendix A).

Chapter 2

Results

2.1 Exactly Solvable Model

The Standard Model described by eqs. (1.1) can be solved with Dynamical Mean Field Theory (DMFT), but the solution goes through a sampling step and is therefore not exact (see [11]). Is it possible to simplify eqs.(1.1) to write an exactly solvable model ? If so, is the simplified model still relevant – or did it lose essential physical properties ?

What complicates the DMFT analysis of eqs.(1.1) is the presence of the tanh function, which has biological grounds but is hard to deal with analytically. Thus, we posit that it can be replaced by any other non-linear function, and we choose to replace it by a quadratic function, which is simpler. After doing this, we also need to change the confining term $-x_i(t)$ in eqs.(1.1) into $-\mu(t)x_i(t)$, where the function $\mu(t)$ ensures that the dynamics of the x_i's is not diverging. The Exactly Solvable Model thus created reads

$$\tau \frac{dx_i}{dt} = -\mu(t)x_i(t) + \frac{\hat{g}}{N}\sum_{j,k=1}^{N} J_i^{jk} x_j(t)x_k(t) + H_i(t), \tag{2.1}$$

where $H_i(t)$ is a possible input current and the J_i's are GOE random matrices with

$$J_i^{jk} = J_i^{kj}, \quad \overline{J_i^{jk}} = 0, \quad \overline{(J_i^{j\neq k})^2} = 1, \quad \overline{(J_i^{jj})^2} = 2.$$

This model is always chaotic, whatever the value of $\hat{g}$.

With DMFT (see section 1.3 for a summary of the method), the system of equations (2.1) is mapped onto the following single-neuron equation

$$\tau \frac{dx}{dt} = -\mu(t)x(t) + \xi(t) + H(t), \tag{2.2}$$

where $\xi(t)$ is a gaussian noise whose statistical properties can be computed (see appendix A) and the computation yields

$$\overline{\xi(t)} = 0, \quad \overline{\xi(t)\xi(t')} = 2\hat{g}^2 C^2(t,t'),$$

with $C(t,t') = \frac{1}{N}\sum_i x_i(t)x_i(t') = \langle x(t)x(t')\rangle_\xi$ being the two-times correlation function of the x_i's. The dimensionality of the system of equations (2.1) has thus been reduced, at the cost of introducing memory into the system.

Now, multiplying eq.(2.2) by $x(t')$ and averaging over the noise ξ, one gets

$$\partial_t C(t,t') = -\mu(t)C(t,t') + \int_0^{t'} ds\, 2\hat{g}^2 C^2(t,s)R(t',s) + \langle H(t)x(t')\rangle_\xi, \tag{2.3}$$

where $R(t,t') = \langle \frac{\delta x(t)}{\delta \xi(t')}\rangle_\xi$ is the response function of the x_i's, and we used the fact that $\langle \xi(t)x(t')\rangle_\xi = \int_0^{t'} ds\, 2\hat{g}^2 C^2(t,s)R(t',s)$ (see [10]).

Taking a derivative of eq.(2.2) with respect to $\xi(t')$ and averaging over the noise ξ, we then get

$$\partial_t R(t,t') = -\mu(t)R(t,t') + \delta(t,t') + \left\langle \frac{\delta H(t)}{\delta \xi(t')} \right\rangle_\xi. \tag{2.4}$$

The DMFT analysis of the model introduced in this section therefore yields a closed system of equations for the correlation function C and the response function R. Provided one specifies the form of $\mu(t)$ and $H(t)$, these equations can be integrated numerically with a forward Euler scheme, and they give access to an exact solution in the $N \to \infty$ limit.

The theoretical advances obtained with the Exactly Solvable model of eqs.(2.1) are thus substantial – but these gains are useless if the aforementioned model does not accurately describe a RNN. To see this, we tested if we recovered – with the Exactly Solvable model – standard results obtained with the model of eqs.(1.1). These results are discussed in the next two sections.

2.2 Hebbian Driving

Recently in [11], Clark and Abbott considered the Standard model of eqs.(1.1) and added an input current of the form

$$H_i(t) = \sum_{j=1}^{N} A_i^j(t)x_j(t), \tag{2.5}$$

where the connections $A_i^j(t)$ follow the dynamical equation

$$p\frac{dA_i^j(t)}{dt} = -A_i^j(t) + \frac{k}{N}x_i(t)x_j(t), \tag{2.6}$$

with p the synaptic time constant and k the plasticity strength. Eqs.(2.5) and (2.6), together with (1.1), describe coupled neuronal-synaptic dynamics, where the synaptic connections fluctuate around a random fixed value J_i^j with an amplitude $A_i^j(t)$

that evolves according to an Hebbian rule. Indeed, according to eq.(2.6), the connection $A_i^j(t)$ increases if the product $x_i(t)x_j(t) > 0$ (i.e. if neurons i and j have similar activities) and decreases otherwise, which is in keeping with the biological rule "neurons that fire together wire together".

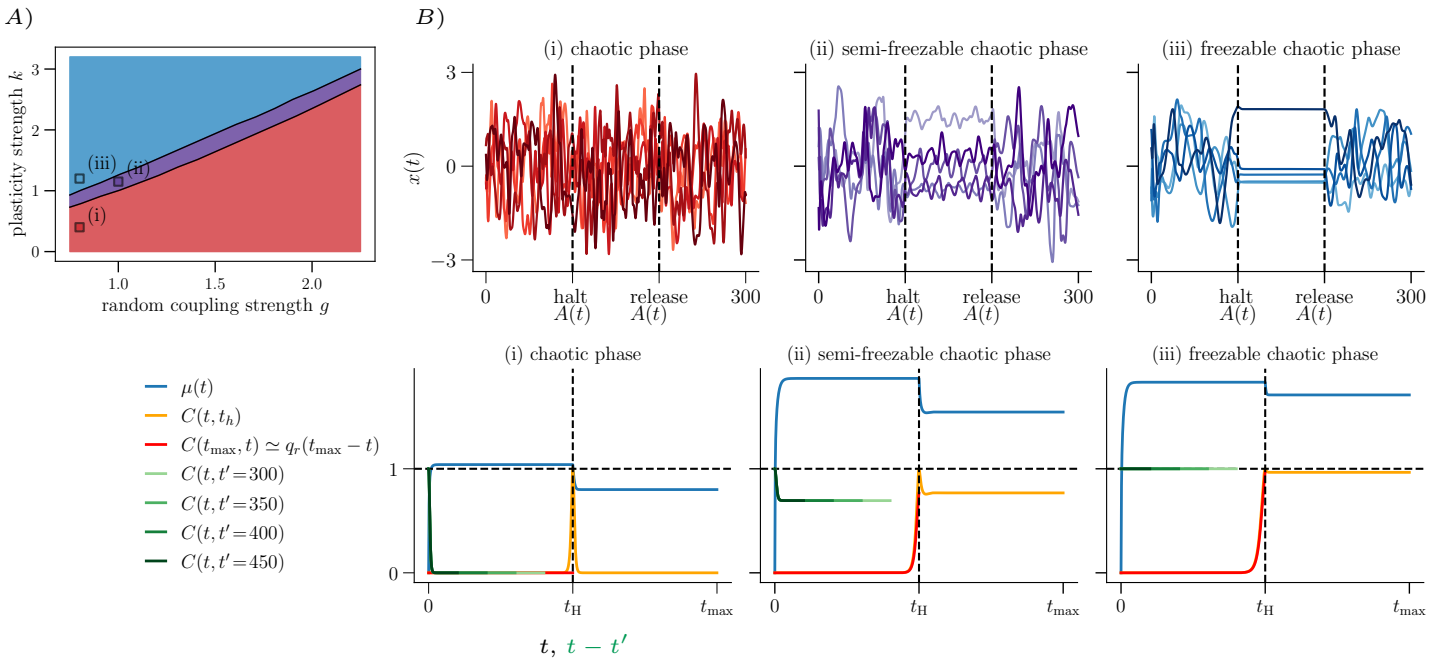

Figure 2.1: *Upper panel:* A) The phase diagram for the Exactly Solvable model with random couplings modulated by an Hebbian plasticity. B) Traces of 5 randomly picked x_i's. The simulations were done with $\tau = 1$, $dt = 0.01$ and $p = 2.5$. The values of g and k match those of the corresponding points in the phase diagram. In plots (i) and (iii), $N = 100$, while $N = 200$ in plot (ii). *Lower panel:* the behavior of a set of dynamical correlation functions as extracted from the numerical integration of the DMFT equations. $t_{\max} = 500$ and $t_H = 250$. The *green* lines represent $C(t,t')$ as a function $t - t'$ and we use them as a proxy for $q_{\rm EA}$.

Adding this Hebbian plasticity to the Standard model, Clark and Abbott found that – for $g > 1$ – if one lets the connections $A_i^j(t)$ evolve up to a halting time t_H, and then clamp them to their value $A_i^j(t_H)$, they could distinguish between three phases, depending on the value of k.

- if k is sufficiently small, the activity of the x_i's after halting the synaptic dynamics of the A_i^j's remains chaotic, and this is the Chaotic Phase (CP)
- if k is sufficiently large, the x_i's go to a fixed point and their activities become constant. This is called the Freezabe Chaotic Phase (FCP)
- and for intermediate values of k, the activity of the x_i's after the halting time t_H remains chaotic, but the x_i's at $t > t_H$ are not completely uncorrelated

from their value at t_H. The space of visited configurations is smaller than in the CP and is somewhat centered around the state $\underline{x}(t_H)$. This is called the Semi-Freezabe Chaotic Phase (SFCP)

We applied the same synaptic dynamics of eqs.(2.5) and (2.6) to the Exacty Solvable model described by eq. (2.1) and found a similar phenomenology (see the upper panel of fig. 2.1). Let's now write the DMFT equations for the correlation and response functions we get under Hebbian driving.

The dynamical equation (2.6) can be integrated, which yields

$$A_i^j(t) = A_i^j(0) + \frac{k}{pN}\int_0^t ds\, e^{-(t-s)/p} x_i(s)x_j(s).$$

So, assuming $A_i^j(0) = 0$, we get

$$\langle H_i(t)x_i(t')\rangle_\xi = \frac{k}{p}\int_0^t ds\, e^{-(t-s)/p}\langle x_i(s)x_i(t')\rangle_\xi \frac{1}{N}\sum_{j=1}^N x_j(s)x_j(t) = \frac{k}{p}\int_0^t ds\, e^{-(t-s)/p} C(t,s)C(t',s)$$

$$\left\langle \frac{\delta H_i(t)}{\delta \xi_i(t')}\right\rangle_\xi = \frac{k}{p}\int_{t'}^t ds\, e^{-(t-s)/p}\left\langle \frac{\delta x_i(s)}{\delta \xi_i(t')}\right\rangle_\xi \frac{1}{N}\sum_{j=1}^N x_j(s)x_j(t) = \frac{k}{p}\int_{t'}^t ds\, e^{-(t-s)/p} C(t,s)R(s,t').$$

We still need to specify the form of the confining function $\mu(t)$. In this case, we choose to impose spherical constraints, meaning that

$$\frac{\|\underline{x}(t)\|}{N} = C(t,t) = 1.$$

Consequently, we obtain the following system of equations for the correlation and response function

$$\begin{cases} \partial_t C(t,t') = -\mu(t)C(t,t') + \int_0^{t'} ds\, 2\hat{g}^2 C^2(t,s)R(t',s) + \frac{k}{p}\int_0^t ds\, e^{-(t-s)/p} C(t,s)C(t',s) \\ \partial_t R(t,t') = -\mu(t)R(t,t') + \delta(t,t') + \frac{k}{p}\int_{t'}^t ds\, e^{-(t-s)/p} C(t,s)R(s,t') \\ \mu(t) = \int_0^{t'} ds\, 2\hat{g}^2 C^2(t,s)R(t',s) + \frac{k}{p}\int_0^t ds\, e^{-(t-s)/p} C(t,s)C(t',s), \end{cases} \tag{2.7}$$

where the equation for $\mu(t)$ is obtained by taking the $t' \to t$ limit and imposing $C(t,t) = 1$ in the equation for the evolution of $C(t,t')$. The system of eqs.(2.7) can be integrated numerically with a forward Euler scheme. A set of correlation and response functions are plotted in the lower panel of fig.2.1.

In order to distinguish between the three phases, we introduce the order parameter

$$q_{\text{EA}} = \lim_{t,\, t'\to\infty,\, t-t'\to\infty} C(t,t').$$

As can be seen by the green traces in the lower panel of fig.2.1 which we use as a proxy for q_{EA}, we have that

- $q_{\text{EA}} = 0$ in the CP
- $0 < q_{\text{EA}} < 1$ in the SFCP
- $q_{\text{EA}} = 1$ in the FCP.

Some progress can be made in describing analytically the three phases by writing asymptotic solutions of the system of eqs.(2.7). See [12] for more a more detailed explanation.

2.3 FORCE learning with the Exactly Solvable model

As mentioned in section 1.2, the Standard model can learn to generate patterns with an algorithm called FORCE. Can the Exactly Solvable model learn to generate patterns as well? We'll first look at numerical simulations of the FORCE algorithm, and then write a DMFT analysis of FORCE learning. We'll focus on FORCE-II learning as it is more efficient.

2.3.1 Numerical simulations

In this setting, we define

$$H_i(t) = w_{fi} z(t), \tag{2.8}$$

$$\text{with } \; z(t) = \frac{1}{N} \sum_{i=1}^{N} w_{oi} x_i(t). \tag{2.9}$$

We choose $w_{fi} = 1$ for $i = 1, ... N$ for simplicity, but the analysis developped in this section can be generalized to case where the w_{fi}'s are chosen randomly. The w_{oi}'s are the output weights and we wish to tune them with the FORCE learning algorithm, so that – after training – $z(t)$ reproduces a desired function $f(t)$. The FORCE algorithm is adapted to the Exactly Solvable model by replacing all the r_i's in eqs.(1.3) and (1.5) by x_i's, and sometimes re-scaling by a factor $1/N$ to have the proper scaling for the DMFT analysis below. We get the following Euler-discretized iterative procedure

$$\begin{cases} x_i(t+dt) = x_i(t) + dt \left[-\mu(t) x_i(t) + \frac{\hat{g}}{N} \sum_{j,k=1}^{N} J_i^{jk} x_j(t) x_k(t) + z(t) \right] \\ z^+(t) = \frac{1}{N} \underline{w}_o^T(t) \underline{x}(t+dt) \\ e(t+dt) = z^+(t) - f(t+dt) \\ P(t+dt) = P(t) - \frac{1}{N} P(t) \underline{x}^T(t+dt) \underline{x}(t+dt) P(t) / [1 + \frac{1}{N} \underline{x}^T(t+dt) P(t) \underline{x}(t+dt)] \\ \underline{w}_o(t+dt) = \underline{w}_o(t) - e(t+dt) P(t+dt) \underline{x}(t+dt), \end{cases} \tag{2.10}$$

with $P(0) = I/\alpha$. I is the identity matrix and α a learning constant.

We choose to work here with a confining term of the form

$$\mu(t) = C(t,t) = \frac{1}{N}\sum_{i=1}^{N} x_i(t)x_i(t).$$

Implementing the procedure (2.10) numerically, we find that the Exactly Solvable model is able to learn to generate simple periodic functions. In all numerical simulations presented in this section, we work with $dt = 0.01$, $N = 100$, $\alpha = 0.001$ and $\hat{g} = 0.7\sqrt{3/4}$.

In the left panel of fig.2.2, we consider the task of learning a constant function, and periodic perturbations around a constant value. Specifically, we choose

$$f(t) = A + Bg(t,\underline{\omega}) \tag{2.11}$$

$$\text{with } g(t,\underline{\omega}) = \frac{1}{\sqrt{a^2+b^2}}\left(a\sin(2\pi\omega_0 t) + b\sin(2\pi\omega_1 t)\right) \tag{2.12}$$

and where $A = 1$, $B \in \{0,\, 0.1,\, 1\}$, $a = 0.6$, $b = 1.2$ and $\underline{\omega} = \underline{\omega}^* = \{\omega_0 = 0.1,\, \omega_1 = 0.2\}$. In the right columns of fig.2.2, we plotted the trace of some w_{oi}'s during training. Learning has converged once the w_{oi}'s reach a stationary value. So, in the left panel, we see that the larger the amplitude B, the longer the learning time.

In the right panel of fig.2.2, we instead consider the task of learning a periodic function of varying frequency. We choose to learn the same function as in eqs.(2.11) and (2.12) but with $A = 0$, $B = 2$ and $\underline{\omega} \in \{0.5\,\underline{\omega}^*,\, \underline{\omega}^*,\, 2\,\underline{\omega}^*\}$. We note that there seem to be an optimum value of $\underline{\omega}$ at around $\underline{\omega}^*$, where learning is the fastest.

The goal of learning is that the output weight $\underline{w}_o$ reaches a stable attractor where the network outputs $z(t) \approx f(t)$. The attractor reached is never completely stable (some Lyapunov exponents are always positive). But the time it takes to move away from the stable attractor once learning is halted is increased the longer the training time. To show this, we plotted in the left panel of fig.2.3 the trajectory of $\underline{w}_o(t)$ during training, projected on the first two principal components (PCs) of the averaged correlation matrix of network's activities $C = \langle \underline{x}(t)\underline{x}^T(t)\rangle_t$. The function learned here is $f(t) = 3\sin(t/2)/2$. We see that $\underline{w}_o$ reaches quickly a fixed point attractor (after learning for just 8 periods of f). In the right panel of fig.2.3, we plot the periodic error $\epsilon(n)$ defined as

$$\epsilon(n) = \int_{nT}^{(n+1)T} ds\, |z(s) - f(s)|^2, \tag{2.13}$$

where T is the period of f.

We look at the performances after learning for $n_h \in \{8,\, 40,\, 80\}$ periods of f. We see that $\epsilon(n)$ grows exponentially fast once learning is halted, with an exponent that is smaller the larger n_h, so the larger the training time. For training times sufficiently large for $\underline{w}_o(t)$ to have reached a stable fixed point attractor, we see some

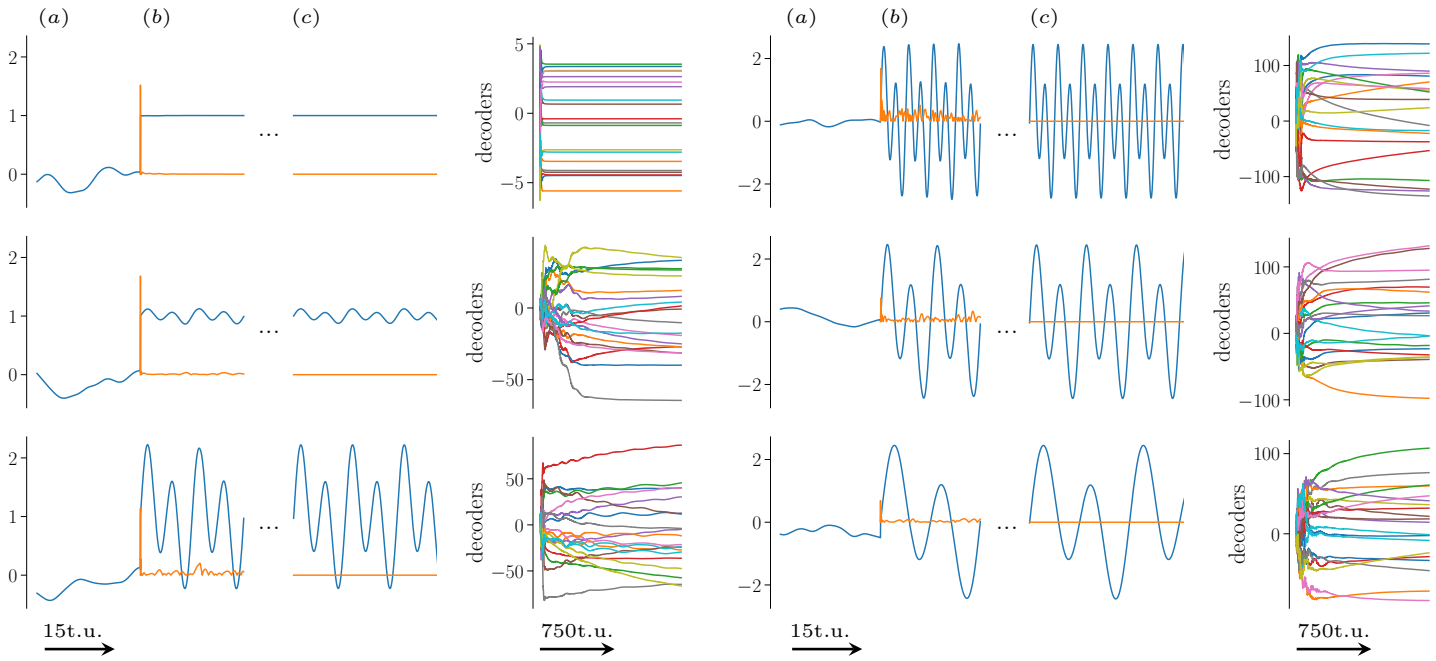

Figure 2.2: The Exactly Solvable model can learn to generate a variety of simple periodic functions with the FORCE learning algorithm. *Left panel:* learning to generate a constant function, and a periodic perturbation of varying amplitude around a constant value. (a) pre-training phase. The activity of the network's output $z(t)$ in *blue* is chaotic. (b) training phase. $z(t)$ reproduces the function $f(t)$ that the network is learning. In *orange* is shown the weight update $\|\underline{dw}_o(t)\| = \|\underline{w}_o(t) - \underline{w}_o(t-dt)\|$. The weight update is initially big to bring $z(t)$ close to $f(t)$ instantly, then $\|\underline{dw}_o(t)\| \to 0$ as learning proceeds. $\|\underline{dw}_o(t)\|$ has here been re-scaled by a factor 0.1. (c) post-training phase. The output weights $\underline{w}_o$ are fixed and the network autonomously outputs $z(t) \simeq f(t)$. In the *right column* is shown the traces of some randomly picked decoder weights w_{oi}'s during training. *Right panel:* same as *Left panel* but when learning a simple periodic function of varying frequency. In all simulations, learning lasted 1500 t.u.

fluctuations in performances due to finite size effects. Notice in particular that the *orange* curve, where training is halted after $n_h = 40$, is below the *blue* one, where training is halted after $n_h = 80$.

In conclusion, the Exactly Solvable model has the capacity to learn to generate simple periodic functions with the FORCE algorithm, which is an important property that the Standard model of RNNs also shares.

2.3.2 DMFT analysis of FORCE learning

With the Exactly Solvable model, we are able to write exact DMFT equations. We now take on the task of performing the DMFT analysis of the FORCE learning

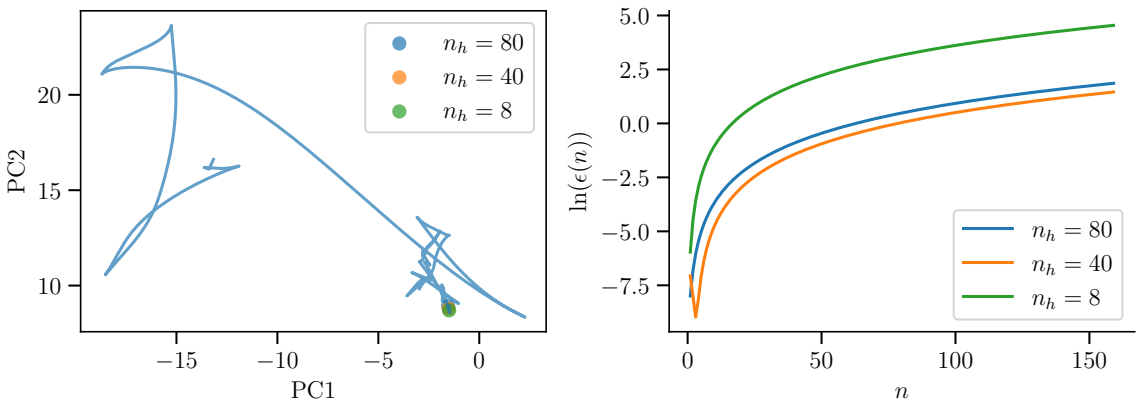

Figure 2.3: *Left panel:* trajectory of $\underline{w}_o(t)$ during training, projected on the first two principal components (PCs) of the averaged correlation matrix of network's activities $C = \langle \underline{x}(t)\underline{x}^T(t)\rangle_t$. The colored dots correspond to the state of $\underline{w}_o(t)$ after learning for $n_h \in \{8, 40, 80\}$ periods of f. The explained variance of the first two PCs is $\simeq 0.961$. *Right panel:* periodic error $\epsilon(n)$ after learning for different training times.

procedure for our simplified model.

Recalling that

$$C(t,t') = \langle x(t)x(t')\rangle_\xi$$
$$R(t,t') = \left\langle \frac{\delta x(t)}{\delta \xi(t')} \right\rangle_\xi ,$$

we can write the discretized DMFT equations of the dynamical system of eq.(2.2) with $H(t) = z(t)$ and $\mu(t) = C(t,t)$. We get the following system of equations

$$\begin{cases} C(t+dt,t') - C(t,t') = dt\left[-C(t,t)C(t,t') + \frac{3g^2}{2}\sum_{i=0}^{t'/dt} C^2(t,idt)R(t',idt) + z(t)m(t')\right] \\ C(t+dt,t+dt) - C(t,t) = 2dt\left[-C(t,t)^2 + \frac{3g^2}{2}\sum_{i=0}^{t/dt} C^2(t,idt)R(t,dt) + z(t)m(t) - dtC(t,t)z(t)m(t)\right] \\ \qquad + dt^2\left[\frac{3g^2}{2}C^2(t,t) + C^3(t,t) + z^2(t) - 3g^2C(t,t)\sum_{i=0}^{t/dt} C^2(t,idt)R(t,idt)\right] \\ R(t+dt,t') - R(t,t') = -\mu(t)R(t,t')dt + \delta_{t/dt,t'/dt} \\ m(t+dt) - m(t) = dt\left[-\mu(t)m(t) + z(t)\right] \end{cases} , \tag{2.14}$$

with $C(0,0) = 1$, $R(0,0) = m(0) = z(0) = 0$ and $2\hat{g}^2 = 3g^2/2$.
The function $m(t)$ is the magnetization, defined as

$$m(t) = \frac{1}{N}\sum_{i=1}^{N} x_i(t) = \langle x(t)\rangle_\xi . \tag{2.15}$$

And the initial conditions come from the fact that we take the $x_i(0)$'s extracted from a Gaussian measure with 0 mean, we take the $w_{oi}(0)$'s uncorrelated from the $x_i(0)$'s, and we impose $\|\underline{x}(0)\| = N$.

We now need an equation for the evolution of

$$z(t) = \frac{1}{N}\underline{w}_o^T(t)\underline{x}(t) = \langle w_o(t)x(t)\rangle_\xi.$$

From the system of equations (2.10), we get

$$z(t+dt) = z^+(t) - e(t+dt)\mathcal{P}(t+dt, t+dt, t+dt), \tag{2.16}$$

where we have denoted

$$\mathcal{P}(t,t',t'') = \frac{1}{N}\underline{x}^T(t)P(t')\underline{x}(t''). \tag{2.17}$$

We can find an equation for $z^+(t)$ recursively. We get

$$z^+(t+dt) = -\sum_{i=1}^{t/dt} e(idt)\mathcal{P}(t+dt, idt, idt)\,, \quad t \geq dt. \tag{2.18}$$

Finally, we can write a recursive relation for $\mathcal{P}(t,t',t'')$

$$\begin{cases} \mathcal{P}(t,0,t') = \frac{1}{\alpha}C(t,t') \\ \mathcal{P}(t,s+dt,t') = \mathcal{P}(t,s,t') - \mathcal{P}(t,s,s+dt)\mathcal{P}(s+dt,s,t')/[1+\mathcal{P}(s+dt,s,s+dt)]. \end{cases} \tag{2.19}$$

The system of equations (2.14), together with eqs.(2.16), (2.18) and (2.19), form a closed set of DMFT equations that can be integrated numerically. These equations describe the dynamics of the system in the $N \to \infty$ limit.

We now show the results of numerically integrating the DMFT equations describing the dynamics of FORCE-II learning for the Exactly Solvable model. In the figures shown below, we work with $dt = 0.1$ and $\alpha = 0.001$. We checked in appendix B that the results of the numerical integration of the DMFT equations were consistent with the simulations.

The numerical implementation is rather straight-forward and the running time of the algorithm is small since we are just propagating equations. But the numerical integration is limited by the available memory, because we need to store all the entries of the 3-tensor $\mathcal{P}(t,t',t'')$.

Figure 2.4 shows a qualitative picture of FORCE algorithm's performances in the $N \to \infty$ limit. The function learned is a simple sinus function $f(t) = \sin(t)$. We see again that the longer the training time (i.e. the larger n_h), the closer the *colored*

curve $z(t)$ follows the *black* curve $f(t)$ after learning is halted.

Figure 2.5 instead shows a more precise view of the algorithm's performances. The function learned is $f(t) = 3\sin(t)/2$. The quantity $|z(t) - z^+(t - dt)|$ is proportional to $\|\underline{dw}_o(t)\|$ so it decreases during training and the smaller its value at the end of learning, the better the convergence of the algorithm. In the right panel of fig.2.5, we see that $|z(t) - z^+(t - dt)|$ decreases exponentially fast during learning. And in the left panel, we see that the periodic error $\epsilon(n)$ also decreases exponentially fast during training, and then grows exponentially fast once learning is halted, with an exponent that is larger the smaller n_h.

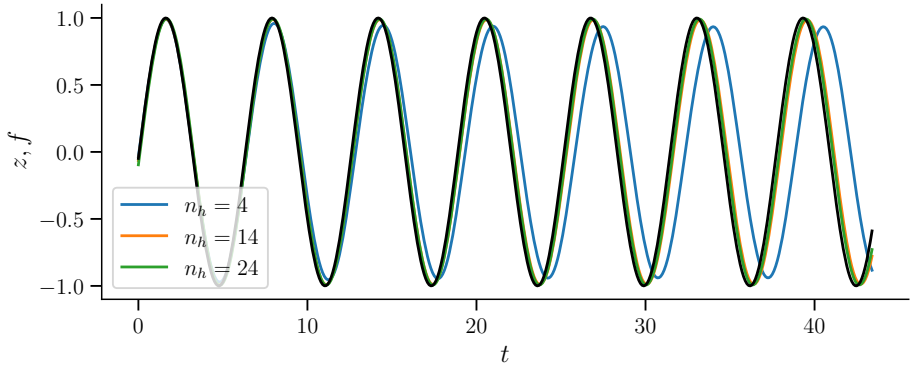

Figure 2.4: Qualitative view of the performances of FORCE-II learning in the $N \to \infty$ limit. *black curve:* function learned $f(t) = \sin(t)$. *colored curves:* network output $z(t)$, when learning is halted after a varying number n_h of periods of f.

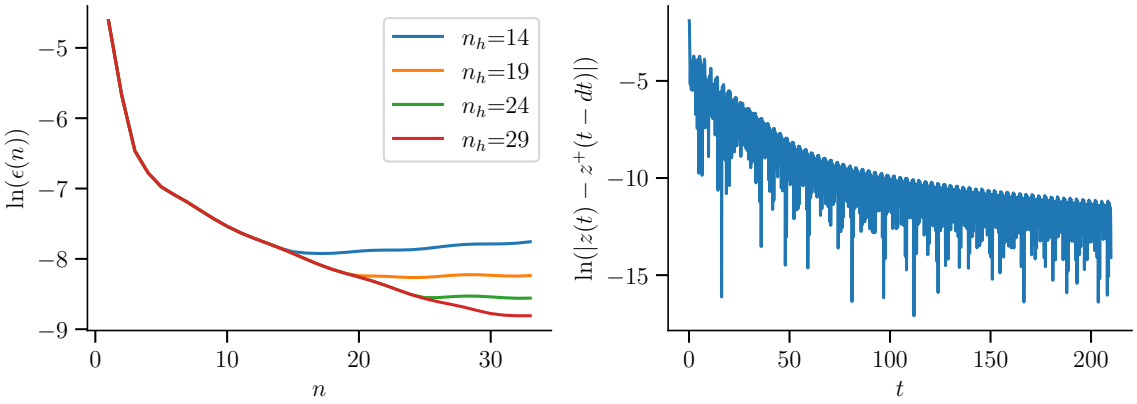

Figure 2.5: Performances of FORCE-II learning in the $N \to \infty$ limit. *left panel:* periodic error $\epsilon(n)$ during and after learning, for different halting times. *right panel:* $|z(t) - z^+(t - dt)| \propto \|\underline{dw}_o(t)\|$ during training. The numerical integration lasted in total 210 t.u.

In conclusion, we find that the Exactly Solvable model is also able to learn to generate simple periodic patterns in the $N \to \infty$ limit. This result was not a given,

as only finite size simulations had been done previously with the Standard model [1, 2]. This result is interesting from a biological standpoint because biological neural networks are made of billions of neurons. So in this case, the $N \to \infty$ limit is a better model than finite size simulations of $N = 100$ or 1000 neurons.

Discussion

To recall the train of thought followed in this work, we first started by studying the Standard model of recurrent neural networks (RNNs), which is the simplest model one can write on biological grounds. Although already quite simple, this model is not exactly solvable with Dynamical Mean-Field Theory (DMFT) because of the presence of a tanh function. Therefore, we did something that is a hallmark of statistical physics: we further simplified the Standard model, and then hoped that we didn't lose some of its essential physical properties in the process. Specifically, we assumed that the relevant features of the Standard model are that it is a chaotic and high-dimensional system. In that respect, we posited that the specific form of the tanh function wasn't crucial, and we replaced it by a simpler quadratic non-linearity. We coined the resulting network the Exactly Solvable model.

Then, we had to check that the newly found model describes appropriately a RNN. We checked that the Exactly Solvable model has the same phenomenology under Hebbian driving as the Standard model. And we checked that the Exactly Solvable model is also able to learn to generate patterns. These two results allow us to corroborate the idea that a RNN can – at first glance – be approximated by a simple chaotic and high-dimensional system. Henceforth, we can use our simplified model to derive new and interesting analytical results.

With the formalism developed in section 2.3.2, we now have access to the phase diagram of FORCE learning, meaning that we can find under which conditions learning is possible. We know that there needs to be a proper interplay between the level of chaos in the system and the function $f(t)$ being learned. Indeed, the level of chaos needs to be sufficient to generate $f(t)$ – but if it is too high, learning fails [2]. And the "proper" amount of chaos also depends on the amplitude and period of $f(t)$ [9].
In that respect, it is interesting to note that the Hebbian driving discussed in section 2.2 is a powerful modulator of the level of chaos in the system. So one can ask if the optimal level of chaos for FORCE-learning the task at hand could be reached by previously tuning the synaptic weights with an Hebbian driving.
In section 2.3.2, we looked at the dynamics of $z(t)$ and $z^+(t)$, but we could also write equations describing the dynamics of the output weights $\underline{w}_o(t)$ during learning. Doing so would allow us to look at the space of good configurations for the generation of $f(t)$, and how such a configuration is reached during training.
Lastly, the formalism developed in section 2.3.2 can easily be generalized to the case

with multiple readout neurons $z_l(t)$. Generalizing, one could look at the competition between the different output neurons, specifically: Can the network learn to generate multiple functions? And if not, how does the network forget? This phenomenon is known as *catastrophic forgetting* in the Neural Networks community, and it describes the tendency of artificial neural networks to forget previously learned information upon learning new information.

One objective when studying RNNs – besides finding new and potentially better ways to do computations – is to understand how biological networks work. The goal here is to come-up with learning algorithms that are biologically plausible. In this work, we studied in depth the FORCE-II learning algorithm. This algorithm is very efficient for pattern generation but it is not biologically plausible, because the matrix P of learning rates is essentially proportional to the inverse correlation matrix of the whole network. Therefore to update one weight, the algorithm needs non-local information like the activities of all the neurons in the network. Instead, our current understanding of how biological networks work tells us that synaptic modification only requires local information. One example of such a biological learning rule is the Hebb rule, which states that the modification of the synaptic weight J_i^j is only based on the activities of neurons i and j.
Consequently, following the quest for understanding how the brain learns, the same reasoning as the one developed in this work could be applied to more biologically plausible learning algorithms. Namely, one could try to write an exactly solvable model for *spiking neurons*. The neurons in this work were indeed modelled as smooth functions, but we know that biological neurons are highly non-linear: when the membrane potential reaches a certain threshold value, a sharp electrical potential called a spike is generated. Models to describe such neurons have been developed [13], and the idea would again be to simplify one of these models to make it solvable with DMFT, while retaining its physical properties.
Another interesting direction towards biological plausibility would be to study reinforcement learning algorithms, which work by rewarding desired outcomes and punishing undesired ones. One reinforcement learning method that successfully trains RNNs is called node-perturbation [14, 15], and it solves the *credit assignment* problem (i.e. finding which weight modification is responsible for the change in the output) by applying small perturbations to neuronal activities and keeping an "eligibility trace" of the resulting changes. A variant of node-perturbation that is even more biologically plausible – but for which we have far less theoretical understanding – is called Miconi's algorithm [4]. Contrary to node-perturbation, it utilizes Hebbian learning and delayed gratification, which is reminiscent of the effect that dopamine has on the brain.

Finally, we conclude by saying that the Standard model and the Exactly Solvable model studied in this work have a wide array of applicability that goes far beyond RNNs. Indeed, this type of dynamical equations are also studied in social networks with evolving ties, gene expression dynamics and ecosystems with variable interspecies interactions. Furthermore, the specific setting of FORCE learning studied

in this work could be applicable to many other complex systems, from cell and human populations to financial markets. These systems can indeed be made of chaotic constituents having to monitor the feedback loop from their environment.

Appendix

A Effect of the quenched disorder term with DMFT

Let's compute the effect of the quenched disorder term with DMFT in eq. (2.1). With the notations of section 1.3 generalized to N dimensions, we define the actions

$$\mathcal{L}_0 = \partial_t \underline{x}(t) + \mu(t)\underline{x}(t) - \underline{H}(t)$$
$$\mathcal{L}_J = -\frac{\hat{g}}{N}\sum_{j,k=1}^{N} J_i^{jk} x_j(t)x_k(t).$$

We need to compute

$$i\underline{\hat{x}}(t).\mathcal{L}_J = -i\sum_i \int_t \left[\frac{2\hat{g}}{N}\sum_{j<k} J_i^{jk}\hat{x}_i(t)x_j(t)x_k(t) + \frac{\hat{g}}{N}\sum_j J_i^{jj}\hat{x}_i(t)x_j^2(t)\right].$$

$J_i^{j\neq k}$ and J_i^{jj} are independent so the averages can be computed separately. The computation yields

$$\overline{\exp i\underline{\hat{x}}(t).\mathcal{L}_J} = \exp\left\{\frac{1}{2}\frac{2\hat{g}^2}{N^2}\int_{t,t'}\sum_i i\hat{x}_i(t)i\hat{x}_i(t')\sum_{j,k} x_j(t)x_j(t')x_k(t)x_k(t')\right\}$$
$$= \int \mathcal{D}Q\,\delta(NC(t,t') - \underline{x}(t).\underline{x}(t'))\delta(ND(t,t') - i\underline{\hat{x}}(t).i\underline{\hat{x}}(t'))e^{\frac{1}{2}2\hat{g}^2N\int_{t,t'}D(t,t')C^2(t,t')}$$
$$= \int \mathcal{D}Q\mathcal{D}l\exp\left\{N\int_{t,t'}\left[il_1(C(t,t') - x(t)x(t')) + il_2(D(t,t') - \hat{x}(t)\hat{x}(t')) + \frac{1}{2}2\hat{g}^2D(t,t')C^2(t,t')\right]\right\},$$

where $Q = (C,\, D)$ and $l = (l_1,\, l_2)$.

Now we use the saddle-point approximation. Taking a derivative with respect to the Q's, we get

$$il_1 + 2\hat{g}^2C(t,t')D(t,t') = 0 \;\Rightarrow\; l_1 \propto D(t,t') = 0 \quad \text{because of causality}$$
$$il_2 + \hat{g}^2C(t,t')^2 = 0 \;\Rightarrow\; il_2 = -\hat{g}^2C(t,t')^2.$$

Therefore

$$\overline{\exp i\underline{\hat{x}}(t).\mathcal{L}_J} = \exp \int_{t,t'} \left[\cancel{il_2 D(t,t')} - il_2 \hat{x}(t)\hat{x}(t')) + \cancel{\frac{1}{2} 2\hat{g}^2 D(t,t') C^2(t,t')} \right]$$
$$= \exp \left\{ -\frac{1}{2} \int_{t,t'} \sum_i \hat{x}_i(t) \left(2\hat{g} C^2(t,t') \right) \hat{x}_i(t') \right\}.$$

Now all the sites are decoupled in the generating functional so the system is equivalent to the single-site effective Langevin equation

$$\partial_t x = -\mu(t)x(t) + \xi(t) + H(t),$$

with $\overline{\xi(t)} = 0$ and $\overline{\xi(t)\xi(t')} = 2\hat{g}^2 C^2(t,t')$.

B Consistency between the numerical integration of the DMFT equations and simulations

To check that the DMFT equations derived in section 2.3.2 were correct, we compared the result of their numerical integration to simulations (see fig. 6 below).

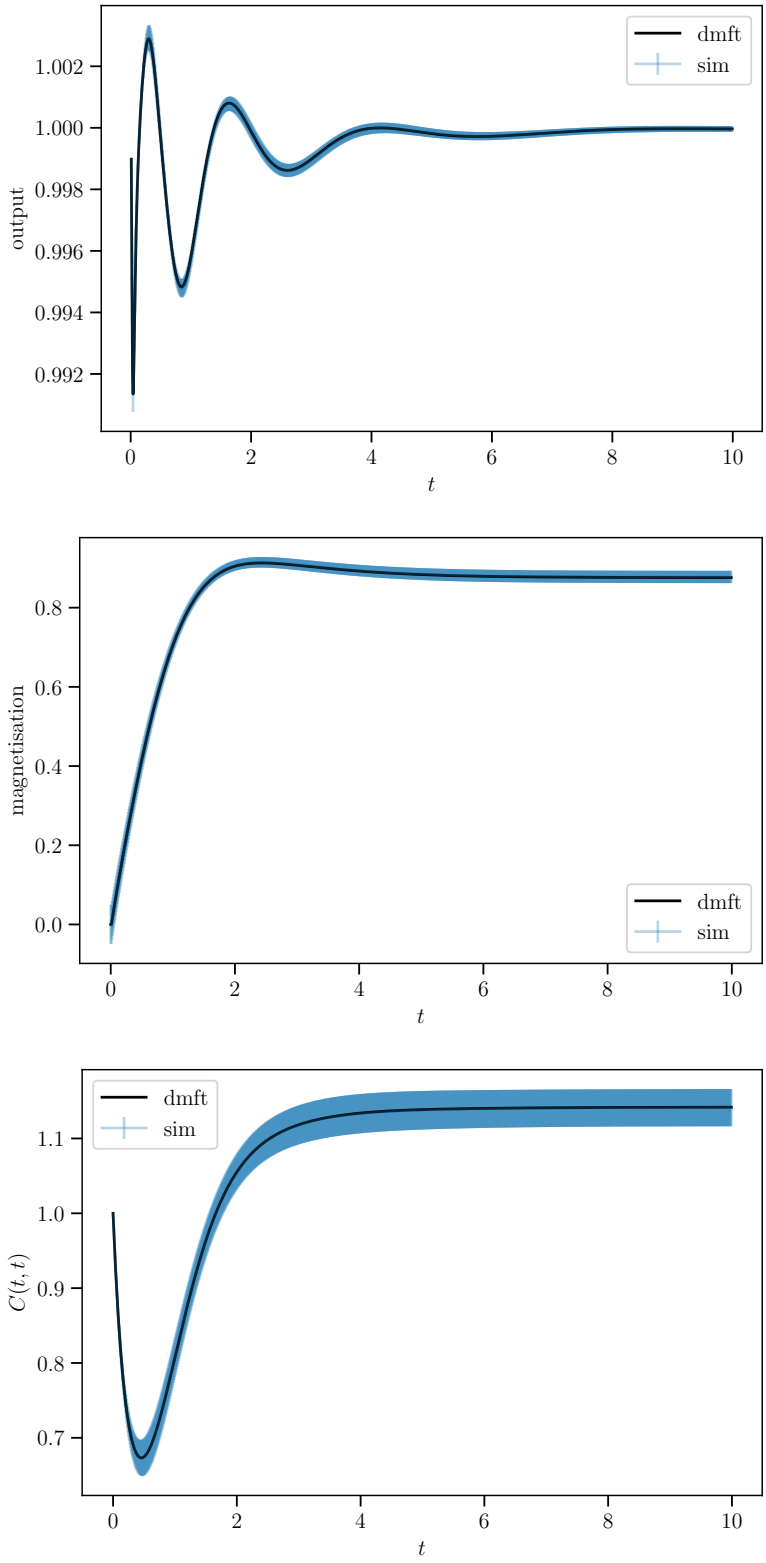

Figure 6: Consistency between DMFT equations and simulations. *black curve:* numerical integration of the DMFT equations derived in section 2.3.2. *blue curves:* results of simulations. The main curve corresponds to the mean over 100 samples and the error bars correspond to the standard deviation. We used $N = 400$ for the simulations, $f(t) = 1$, $dt = 0.01$, $g = 0.5$ and $\alpha = 0.001$.

References

[1] Generating Coherent Patterns of Activity from Chaotic Neural Networks.
D. Susillo and L.F. Abbott, Neuron **63**, p.544-57 (2009)

[2] Learning in Chaotic Recurrent Neural Networks.
D. Susillo, PhD thesis, Columbia University (2009)

[3] Supervised learning in spiking neural networks with FORCE training.
W. Nicola and C. Clopath, Nature Communications **8**, 2208 (2017)

[4] Biologically plausible learning in recurrent neural networks reproduces neural dynamics observed during cognitive tasks.
T. Miconi, eLife **6**, e20899 (2017)

[5] Neural networks and physical systems with emergent collective computational abilities.
J. J. Hopfield, Proceedings of the National Academy of Sciences USA **79**, p.2554-2558 (1982)

[6] Neurons with graded response have collective computational properties like those of two-state neurons.
J. J. Hopfield, Proceedings of the National Academy of Sciences USA **81**, p.3088-92 (1984)

[7] Chaos in Random Neural Networks.
H. Sompolinsky, A. Crisanti and H. J. Sommers, Physical Review Letters **61**, p.259-262 (1988)

[8] Adaptive Filter Theory, S. Haykin (2002)

[9] Stimulus-Dependent Suppression of Chaos in Recurrent Neural Networks.
K. Rajan, L.F. Abbott and H. Sompolinsky, Physical Review E **82**, 011903 (2010)

[10] Spin-Glass Theory for Pedestrians.
T. Castellani and A. Cavagna, J. Stat. Mech., p.05012 (2005)

[11] Theory of Coupled Neuronal-Synaptic Dynamics.
D.G. Clark and L.F. Abbott, arXiv:2302.08985 (2023)

[12] Statistical physics of learning in high-dimensional chaotic systems.
S. Fournier and P. Urbani, arXiv:2306.01477 (2023)

[13] Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting.
M. Izhikevich, MIT Press (2007)

[14] Gradient Learning in Spiking Neural Networks by Dynamic Perturbation of Conductances.
I. R. Fiete and H. S. Seung, Physical Review Letters **97**, p.048104 (2006)

[15] Model of birdsong learning based on gradient estimation by dynamic perturbation of neural conductances.
I. R. Fiete 1, M. S. Fee and H. S. Seung, Journal of Neurophysiology **98**, p.2038-57 (2007)