Dynamical system approach to explainability in recurrent neural networks

Alexis Dubreuil

Institut de la Vision, Sorbonne Universités, INSERM, CNRS, F-75012 Paris, France

alexis.dubreuil@gmail.com

Résumé

Les technologies basées sur l'IA, notamment les machines neuronales, sont souvent qualifiées de boîtes noires, limitant leur déploiement dans toute une gamme d'applications. Ici nous présentons des méthodologies qui permettent d'ouvrir ces boîtes noires. Elles se basent sur le formalisme de la théorie des systèmes dynamiques qui a été initialement mis à profit pour comprendre les calculs dans les réseaux de neurones biologiques. Nous décrivons des travaux qui appliquent ces méthodes pour la modélisation en neurosciences et pour le traitement automatique du langage. Ceci nous permet, à partir d'exemples concrets, d'illustrer comment l'explicabilité peut contribuer aux développements de l'IA.

Mots-clés

explicabilité, apprentissage profond, systèmes dynamiques, neurosciences, Traitement Automatique du Langage

Abstract

AI based technologies and neural machines in particular are often referred to as black-boxes, which limits their deployment in various fields. Here we present methodologies to open these black-boxes. They rely on the formalism of dynamical system theory which has initially been leveraged to understand neural computations in brain circuits. We describe works that applied these methodologies for modeling in neuroscience and for natural language processing, allowing us to discuss concrete examples demonstrating the potential of explainability in fostering further developments in AI.

Keywords

explainability, deep-learning, dynamical systems, neuroscience, Natural Language Processing

1 Introduction

AI based technologies are developing at a fast pace, in particular thanks to recent progress in harnessing neural network based machines. Further expanding the scope of applications requires the functioning of AI systems to be explainable in a human understandable format so that they can be trusted, for instance in safety-critical applications such as self-driving cars. While many levels of explainability can be defined based on the deployment context of a technology [4], a necessary step for neural network based machines is to get access to their "inner workings" [26]. However, the functioning of neural networks is particularly difficult to grasp as they typically are high-dimensional non-linear systems with thousands or millions of parameters tuned by a learning algorithm. Here we review efforts that have been undertaken over the last ten years to reverse-engineer these neural machines. The reviewed works focused on recurrent neural networks (RNN), a generic neural architecture, known to be particularly well suited to deal with time varying inputs and outputs such as those in natural language processing tasks (see e.g. [35]). In sections 2 and 3, we introduce the theoretical concepts, based on dynamical system theory [1], that have been developed since the 80's to understand neural computations. In section 4 we present methodological tools that leverage this conceptual framework to reverse-engineer RNN. In section 5 we illustrate how explainability enables AI approaches to foster interactions between theoretical and experimental neuroscience. In section 6 we show how these methodologies have been used to understand how RNN solve natural language processing tasks. Finally in discussion we outline the potential of these methodologies for scientific applications of AI, practical applications of AI and theoretical investigations of neural computations.

2 RNN as dynamical systems

Recurrent neural networks are fully connected networks, i.e. each neuron a priori receives inputs from all the other neurons in the network (Fig. 1a). Given that they typically receive inputs that extend over time, they are often referred to as deep-in-time neural networks as illustrated in Fig. 1b. A typical way of formalizing the time evolution of the state of a RNN composed of N neurons, $\vec{x}^t \in \mathbb{R}^N$, is through

$$\vec{x}^{t+1} = W_{rec}\Phi\left(\vec{x}^t\right) + W_{in}\vec{u}^t \tag{1}$$

where $W_{rec} \in \mathbb{R}^{N\times N}$ is the recurrent connectivity matrix, $W_{in} \in \mathbb{R}^{N\times N_{in}}$ connects input neurons to recurrent neurons, $\vec{u}^t \in \mathbb{R}^{N_{in}}$ models the activity of input neurons at time t, and $\Phi(.)$ is a non-linear function, typically the hyperbolic tangent or the rectified-linear function. The results

1. see [31] for a friendly introduction to dynamical system theory

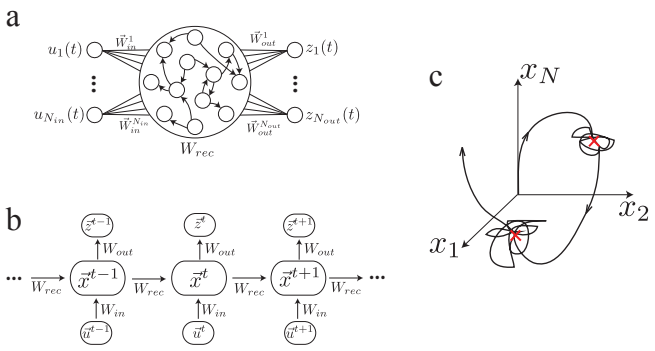

Figure 1. a. Schematic of a recurrent neural network (RNN). b.Typical representation of a RNN seen as a deep-in-time neural network. c. The activity of a RNN while it solves a task and computes on its inputs can be represented as a trajectory in the state-space defined by the activity of each individual units. Red crosses represent stable activity states that could be used to store computationally relevant variables. Transitions between such states would then be triggered by inputs.

of computations in the RNN are read out by readout neurons with activity $\vec{z}^t \in \mathbb{R}^{N_{out}}$:

$$\vec{z}^t = W_{out}\Phi(\vec{x}^t) \tag{2}$$

where $W_{out} \in \mathbb{R}^{N_{out} \times N}$ connects recurrent neurons to the readout neurons.
These equations describe so called Vanilla RNN or Elman networks. Note that sometimes the time evolution is defined as $\vec{x}^{t+1} = \Phi\left(W_{rec}\vec{x}^t + W_{in}\vec{u}^t\right)$. These two formulations can be shown to be equivalent up to a change of coordinates [7]. With the addition of a leak term to eq. (1), the time evolution of the network can be expressed in continuous time as

$$\tau\dot{\vec{x}}(t) = -\vec{x}(t) + W^{rec}\Phi(\vec{x}(t)) + W^{in}\vec{u}(t) \tag{3}$$
$$\vec{z}(t) = W^{out}\Phi(\vec{x}(t)) \tag{4}$$

This is the standard equation for the dynamics of a rate model used for the modeling of biological neural networks [10] and whose behavior has been studied extensively as we illustrate below. RNN are thus high-dimensional ($N \gg 1$) non-linear dynamical systems whose activity $\vec{x}(t)$ can be conveniently thought of as a trajectory in a state-space of dimension N as illustrated in Fig. 1c. When the network is performing a task, these trajectories will reflect the computations performed by the RNN.

3 Recurrent computations on inputs

Neural trajectories are the results of two factors, the input drive $W^{in}\vec{u}(t)$ and the recurrent activity $W^{rec}\Phi(\vec{x}(t))$. In biological and artificial neural networks, neural trajectories are typically attracted towards low-dimensional sub-spaces of dimension $D \ll N$ [9; 2], such that recurrent activity can typically be described by D coordinates or latent variables representing the overlap between the neural activity $\vec{x}(t)$ and particular directions in state-space. The connectivity structure W_{rec} determines the shape of the sub-spaces on which these latent variables evolve. In the next two sub-sections we present classical examples illustrating these features and explicit how latent variables can be related to computations. In the last sub-section we present a class of RNN for which the high-dimensional dynamics eq. (3) can be reduced to a D-dimensional dynamical system governing the time evolution of latent variables, making this class of networks particularly amenable to reverse-engineering.

3.1 Point attractors in Hopfield networks

As a first example, we consider Hopfield networks that have been proposed as models of associative memory [19; 20]. A set of P memories $\vec{\xi}^\mu \in \{-1, +1\}^N$, $\mu = 1, ..., P$ are stored in a RNN by choosing the connectivity matrix

$$W_{rec} = \vec{\xi}^1\vec{\xi}^{1T} + ... + \vec{\xi}^P\vec{\xi}^{PT} \tag{5}$$

where T denotes the transpose operation. If the number of memories P is not too large, neural trajectories evolve towards one of P fixed-points (the dynamical landscape associated to eq. (3) is said to contain P attractors). The dynamics of the network during the retrieval of a memory μ can be effectively reduced to a one-dimensional system over a latent variable representing the overlap between the network state and the memory pattern $\kappa_\mu = \langle\Phi(\vec{x}(t)), \vec{\xi}^\mu\rangle$ [11]

$$\dot{\kappa}_\mu = -\kappa_\mu + F\left(\kappa_\mu\right) \tag{6}$$

where $F(.)$ is a sigmoid-shaped function. When the RNN is cued with an input that triggers neural activity in the direction $\vec{\xi}^\mu$, neural activity sets up in a fixed point characterized by $\kappa_\mu \simeq 1$, $\kappa_{\nu\neq\mu} \simeq 0$ and the memory μ is said to be retrieved.

3.2 Continuous attractors

In the previous example, the recurrent dynamics is attracted towards a set of P discrete points randomly spread throughout state-space. Activity of biological and artificial neural networks can also evolve on continuous sub-spaces such as line attractors (D=1) [29; 21] or plane attractors (D=2) [36; 2]. In [29], a line attractor (illustrated in Fig. 2b) is imprinted in the network's dynamics by choosing a rank one recurrent matrix with a carefully chosen singular value

$$W_{rec} = \vec{m}\vec{n}^T \tag{7}$$

that generates activity $W_{rec}\Phi(\vec{x}(t))$ along the direction $\vec{m}$. The N-dimensional dynamics of such a network can thus be reduced to the one-dimensional dynamics of a latent variable κ. Given the particular structure of the connectivity matrix, inputs that trigger neural activity along the direction $\vec{n}$ will be kept in network activity for a time that can be much longer than the single neuron time constant τ, while inputs triggering neural activity along directions orthogonal to $\vec{n}$ will be forgotten from network activity on a time-scale

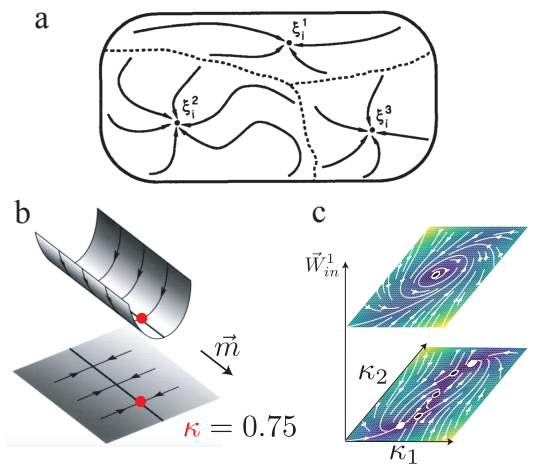

Figure 2. a. Schematic projection of the state space for a Hopfield network storing three memories (taken from [18]). b. Schematic representation of a line attractor. The activity of the network can be thought of as a ball evolving on the half-pipe representing the energy function of the system. The location of this ball on the line composed of marginally stable fixed-points encodes a scalar value (taken from [29]). c. Dynamical landscapes of a 2-D reduced system (see eq. (10)) obtained from a trained rank-2 RNN composed of $N = 1024$ neurons. Filled white circles denote stable fixed-points, empty circles denote unstable fixed-points, color coded is the absolute speed at which latent variable trajectories evolve and the white arrows depict the directionality of the latent variables flow. Inputs to the network can modify the effective couplings between latent variables and thus reconfigure their dynamical landscape. (taken from [13])

τ (cf Fig. 2B). Such latent variables that encode analog values and maintain them on long-time scales can store various computationally relevant information such as the position of an animal [36] or the sentiment associated with a text in a language processing network [21] (see section 6).

3.3 Reduced dynamics over latent variables

The above examples illustrate how latent variables can be used to relate patterns of neural activity to their role in computations. As shown by recent theoretical works [24; 12; 5; 13], the dynamics of low-rank RNN, whose connectivity matrix can be written as

$$W_{rec} = \sum_{k=1}^{D} \vec{m}_k \vec{n}_k^T \tag{8}$$

can be analytically reduced to a dynamics over D latent variables ($D \ll N$). Using tools from statistical physics, the neural activity can be expressed as [24]

$$\vec{x}(t) = \sum_{k=1}^{D} \kappa_k(t)\vec{m}_k + W_{in}\vec{v(t)} \tag{9}$$

where the latent variables are projections of neural activity onto the connectivity vectors $\vec{m}_k$ ($\kappa_k(t) = \langle \vec{x}(t), \vec{m}_k \rangle$) and $\vec{v(t)}$ corresponds to inputs $\vec{u}(t)$ low-pass filtered with a time constant τ. The full N-dimensional dynamics eq. (3) is then fully characterized by a D-dimensional dynamical system [12], that can be concisely expressed as

$$\tau \dot{\kappa_j}(t) = -\kappa_j(t) + \sum_{k=1}^{D} \tilde{\sigma}_{jk}^{rec} \kappa_k(t) + \sum_{k=1}^{N_{in}} \tilde{\sigma}_{jk}^{in} v_k(t) \tag{10}$$

where the effective couplings $\tilde{\sigma}$ are non-linear functions of the κ_k's and inputs $\vec{u}$ (see [12; 13] for explicit expressions of the $\tilde{\sigma}$'s). In particular an input u_k can have two qualitatively different effects, it can either be integrated by a latent variable κ_j through $\tilde{\sigma}_{jk}^{in} v_k(t)$ or it can modulate the effective couplings between latent variables and inputs by controlling the value of one of the $\tilde{\sigma}$. In Fig. 2c we show two dynamical landscapes associated with a single system of two-latent variables, for different applied inputs. At the bottom, the dynamical landscape is composed of two stable fixed-points (white dots) towards which latent variables are attracted. As an input is applied onto the RNN, the effective dynamics of the latent variables is reconfigured and they will oscillate on a limit cycle. Trained RNN make use of this principle to solve tasks [12; 13].

As we will see below, low-rank networks, whose dynamics can be reduced in a principled manner, are particularly well suited to provide access to their inner workings.

4 Reverse-engineering methodologies for RNN

4.1 Characterizing the dynamical landscape by linearization around fixed-points

According to the principles exposed in section 2, understanding computations in a RNN requires to characterize the dynamical landscape associated with the set of trained parameters $\{W_{rec}, W_{in}, W_{out}\}$. Towards such a characterization, Sussillo and Barak [32] developed a numerical procedure to find the fixed-points (or slow points, $\dot{\vec{x}}(t) \simeq 0$) of a trained RNN (Fig. 3a). Then by studying the linearized dynamics of trained RNNs around these slow points they could reveal important computational properties of these systems. For instance by doing so on a RNN trained on a 3-bit flip-flop task (Fig. 3b), they could identify multiple stable fixed-points that enable the system to remain in various states important for solving the task (black crosses in Fig. 3b). They also found saddle-points (fixed-points with some unstable directions, green crosses) that allows inputs to the network to induce transitions between stable fixed-points. By visualizing the locations of these fixed-points and the neural trajectories (blue and red thin lines) in a relevant three dimensional sub-space (identified by principal component analysis on network trajectories concatenated across multiple task trials), they could provide a concise description of how the network solves the task. Various authors have used this approach to reverse-engineer RNN [8; 21] and a tensorflow toolbox is available to find fixed-points and characterize the linear

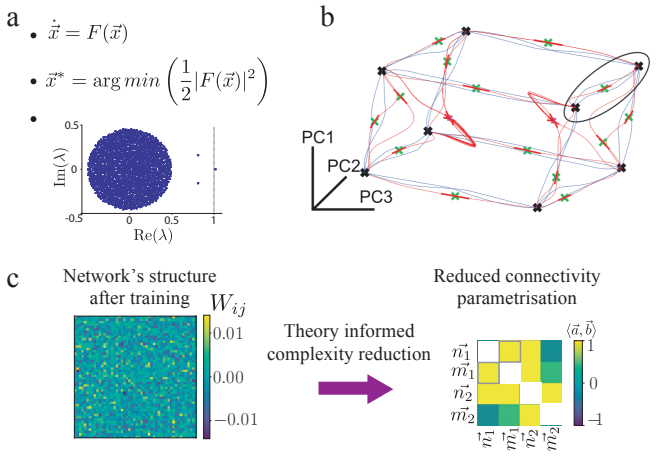

Figure 3. a. Numerical procedure to characterize the dynamical landscape of RNN. For a generic dynamical system defined by a function F(.), the first step is to locate the fixed-points (or slow points) of the trained RNN, then to characterize the linearized dynamics around these fixed points, which can be done by analyzing the eigenspectrum of the stability matrix associated to each fixed point (points beyond the dashed lines are associated with state-space directions that are unstable). b. Visualization of neural trajectories (blue and red thin lines) for a network that solves a 3-bit flip-flop task : a network is excited by three input neurons whose activity takes non-zero values at random time points, three output neurons are trained to represent the sign of the last activation of the inputs neurons (taken from [32]). Black crosses denote the location of stable fixed-points in which the network settles in between inputs. Possible input-triggered transitions between network states are allowed by the presence of saddle points (green crosses) whose direction of instability connects (red thick lines) stable fixed-points. c. Illustration of the analytical procedure to reduce the dimensionality of trained RNN. After training one obtains a $N \times N$ connectivity matrix, the complexity of this matrix is reduced by computing averaged quantities that determine RNN dynamics, e.g. the overlaps between the recurrent connectivity vectors $\vec{m}$ and $\vec{n}$.

dynamics around these fixed-points [15].

4.2 Reducing the dimensionality of trained RNN

Here we present two methodologies that have been proposed to reduce the dimensionality of trained RNN and to describe their computations in a simple language.

In [27], the authors first dissect a trained RNN using an approach similar to what has been described in the above section and noticed that recurrent activity mainly lies in a two-dimensional sub-space. Then, inspired by the knowledge distillation approach [6], they trained a two-units RNN to reproduce the projected two-dimensional activity of the original network and obtained a simpler two-dimensional dynamical system that performs the task equally well than the original network. This allowed them to synthesize the behavior of the network by describing the dynamics of only two variables.

In [12], the authors trained RNN with a connectivity matrix of fixed rank D as in eq. (8). To do so, they reparametrized learning so that a loss function is minimized over the parameters $\Theta = \{\vec{m}_k, \vec{n}_k, W_{in}, W_{out}\}_{k=1,\dots,D}$ instead of $\{W_{rec}, W_{in}, W_{out}\}$ as is usually done. In the training procedure, the rank D was treated as an hyper-parameter, and for each task they identified the minimal rank that allows the task to be solved. They then used the theoretical insights presented in section 3.3 to concisely summarize the dynamics of their RNN with a dynamical system of minimal dimensionality. This analytical characterization of the reduced system allows to relate properties of neural trajectories to the connectivity structure through the mathematical expression of the functional couplings $\tilde{\sigma}$, which depends on the statistical structure of the connectivity vectors Θ (see Fig. 3c). This detailed understanding has allowed, from RNN trained on simple tasks, to engineer similarly functional networks without appealing to a learning algorithm [12; 13]. Thus this methodology, in which solutions to a task are looked for in the restricted space of minimal rank RNN, appears to provide a solution to the reverse-engineering problem that is satisfactory according to the criterion set by Richard Feynman's quote : "What I can not create, I do not understand". An application of this method is described in section 5.1.

5 RNN implementing elementary cognitive processes and brain circuits modeling

Here we review recent works that leveraged these reverse-engineering methodologies in the context of neuroscience modeling (see e.g. [23; 34; 8; 37; 12; 13; 17; 14]). We show how this has allowed to understand the network implementation of elementary cognitive processes. In the first subsection we review recent works that extracted network mechanisms for motor control and context-dependent computations. These works have been performed in parallel to experimental neuroscience studies that involve designing minimal behavioral tasks to isolate core cognitive processes such as working memory, decision making, motor-control or context-dependent computations (see Fig. 4a for an example). In the second subsection we outline how accessing explainable RNN allows to make interpretations and predictions regarding experimental measurements in neuroscience.

5.1 Network implementation of elementary cognitive processes

Motor control. Here we describe the reverse-engineering of a RNN trained on a task inspired by a study of motor control where monkeys slide their fingers on a screen to navigate from a starting point to a target point, with various configurations of obstacles in between [34]. Applying the

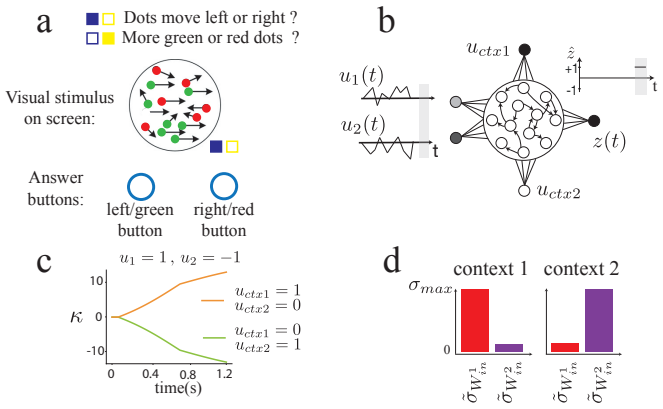

Figure 4. a. Schematic of a behavioral task used in neuroscience experiments. On a screen, a subject watches dots that are colored either green or red and that move erratically (black arrows represent velocity vectors of the dots, which are redrawn randomly at each time step). Based on some contextual cues, the subject's task is to report, by pressing one of two buttons, whether the averaged movement of the dots is to the left or to the right (context 1), or whether there are more green or red dots (context 2). b. Minimal modeling of this behavioral task for a RNN : two sensory inputs $u_1(t)$ and $u_2(t)$ fluctuate over time, the readout neuron should respond $+1$ (resp.-1) if the relevant sensory input has a positive (resp. negative) average. The relevant sensory input is determined by the activities of the two contextual neurons. c. Dynamics of the latent variable characterizing recurrent activity for two task trials. The same constant sensory inputs are shown for the two trials, but the contextual input changes. The latent variable integrates the relevant sensory input. d. In the reduced description of the RNN, the effective interactions coupling the latent variable with the two sensory inputs are modulated by context.

methodology presented in section 4.1 they performed the linear-stability analysis of the single fixed point found in the RNN. While most of the eigenvalues of the stability matrix were associated to decaying modes of activity (eigenvalues with negative real parts), three pairs of eigenvalues (complex conjugate pairs of purely imaginary numbers) were associated with three typical oscillatory modes observed in the neural trajectories for both trained RNN and experimental data.

Context-dependent computations. In [12] the authors trained low-rank RNN to implement the context-dependent task described in Fig. 4a,b whose implementation relies on a form of if... then... else... statement. They performed reverse-engineering using the methodology described in section 4.2. They found that a rank one network was sufficient to perform this task and could summarize the recurrent computation by the one dimensional non-linear dynamical system

$$\dot{\kappa} = -\kappa + \tilde{\sigma}_\kappa \kappa + \tilde{\sigma}_{W^1_{in}} u_1(t) + \tilde{\sigma}_{W^2_{in}} u_2(t) \qquad (11)$$

Within this formulation, the behavior of the network could be explained in simple terms : the contextual inputs u_{ctx1} and u_{ctx2} control the effective couplings between the latent variable and the sensory inputs $u_1(t)$ and $u_2(t)$ (Fig. 4d), allowing the latent variable to compute the temporal average $\langle u_1(t) \rangle_t$ if $u_{ctx1} = 1 \,\&\, u_{ctx2} = 0$ and $\langle u_2(t) \rangle_t$ if $u_{ctx1} = 0 \,\&\, u_{ctx2} = 1$ (Fig. 4c). This explanation is equivalent, although more concise, to the one we could give by running the methodology of section 4.1 : recurrent activity evolves on a line attractor to which is associated an input selection vector whose direction is tuned by contextual inputs to integrate the relevant sensory inputs [23; 22]. Other tasks involving context-dependent computations have been shown to rely on such context-dependent dynamical landscape for the latent variables [13].

5.2 Explainability for neuroscience experiments

The insights obtained by reverse-engineering these RNN have turned out to be quite useful for biological network modeling. A typical approach goes as follows : i) the activity of part of a brain circuit has been characterized in experiments, ii) a modeler explores various training settings (e.g. plays with regularization techniques, [34]) to obtain RNN whose neural activity matches the biological one, iii) the RNN is reverse-engineered. This leads to two types of contributions. First, by synthesizing the available data in a mechanistic framework, it allows to propose a functional role to observed properties of brain activity. For instance in [34], by using various learning procedures they obtained different RNN implementations of the task and could show that the simplest RNN (in terms of their dynamical properties) better matched the data. They could moreover show that the simplest RNN were more robust to perturbations such as neuron erasure, suggesting that features of neural activity observed in motor cortex correspond to a robust implementation of motor control. Second, it allows to make predictions for parts of the circuit, or properties of the circuit, that have not been characterized experimentally. For instance in [13] the authors propose predictions regarding the structure of neural selectivity (how the activity of neurons relate to the task variables) to be observed in animals performing specific tasks.

6 Reverse-engineering RNN performing language processing tasks

These methods have been used to reverse-engineer networks trained on language processing tasks. In [21] the authors trained various RNN architectures (Vanilla RNN, GRU, LSTM) to perform a binary sentiment analysis tasks on the Yelp review, the IMDB movie review and the Stanford Sentiment Treebank datasets. Each trial consists of a sentence where each word is mediated through an input vector to the RNN, at the end of each sentence a readout neuron is asked to provide a binary value, $+1$ if the review is positive, -1 if the review is negative. Analyzing trained networks they found that the learning algorithm builds

networks whose activity is low-dimensional (a PCA analysis shows that 90% of the RNN hidden state variance is captured with only two principal components). By visualizing neural trajectories produced throughout the presentation of a sentence in a two-dimensional space, they noticed that neural activity mainly evolves along a single dimension aligned with the network's readout vector W_{out}, with sentences associated with positive reviews driving activity in one direction and negative reviews driving activity in the other direction (see Fig. 5a). The amplitude of an activity shift along this direction was dependent on the current word being presented, with neutral words (e.g. "the", "car") eliciting no movement along this dimension and highly negative (e.g. "bad", "horrible") or positive (e.g. "amazing", "great") words eliciting large movements. By finding the fixed points of this system and analysing their linear stability (section 4.1), they could show that these trajectories are supported by a line attractor (section 3.2). They realized that a RNN implementing only a line attractor would perform as well as a bag-of-words model, although their trained RNN showed better performance. In a subsequent study [22], still using the same reverse-engineering techniques, they could reveal that trained RNN performed better than bag-of-words because they are able to process words in a context-dependent manner. They focused on the effect of modifier words (e.g. "not") that reverse the meaning of a subsequently appearing word such as in "not bad" versus "bad". They found that modifier words drive activity in a direction orthogonal to the line attractor towards points in state-space where the response to subsequent inputs is reversed (Fig. 5b), revealing a similar mechanism than the one described in section 5.1. Remarkably, using the insights gained by their reverse-engineering studies they could extend bag-of-words models to include the effect of such modifier words and reach performance similar to the one of the more complex RNN initially used.
In another recent study [2] the authors performed the same type of reverse-engineering on other NLP tasks (document classification, review star prediction, emotion tagging). With learning leading to low-dimensional activity in the RNN, they could similarly describe how the properties of the networks' dynamical landscapes govern the RNN computations.

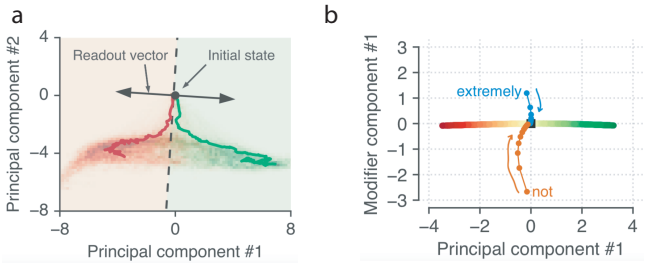

Figure 5. a. Two neural trajectories in a 2-D cut of the state-space of RNN performing a sentiment analysis task. The green trajectory corresponds to the activity of the RNN while it is presented with a positive review : positive words push neural activity towards the right. The red trajectory corresponds to a negative review. (taken from [21]). b. Modifier words push neural activity in directions that are orthogonal to the line attractor direction. Neural activity then reaches a point in state-space where the dynamical landscape is such that subsequent inputs generates a shift in state-space in a direction opposite to the one that would occur in the absence of a modifier word (taken from [22]).

7 Discussion

We have mainly reviewed two methodologies that have been developed to characterize the dynamical landscape of RNN. We have first introduced a numerical procedure that allows to find the fixed-points of trained RNN and to characterize the linearized dynamics of networks around these fixed-points (section 4.1). We have shown on examples how stable fixed-points are associated with internal network states that are important for the storage of information. We have also shown how training algorithms can build other types of dynamical structures to support other aspects of computation, such as saddle-points organizing the set of possible input-dependent transitions between internal states (Fig 3b).
We have then introduced an analytical approach that allows to reduce RNN to low-dimensional dynamical systems of latent variables corresponding to collective modes of activity of the full RNN (section 4.2). These reduced dynamical systems are parametrized by averaged quantities, or summary statistics, of the trained connectivity matrices (Fig. 3c). This allows to describe the relationship between properties of neural trajectories and the learnt structure of RNN, and thus to provide satisfactory solutions to reverse-engineering problems. This approach relies on reparametrizing the connectivity matrix by a rank D connectivity matrix and could appear rather limited. Nevertheless, it has been shown that training full rank RNN on simple tasks leads to low-dimensional implementation of the tasks that can be captured by low-rank RNNs [28], and that rank-D RNN can in principle implement any D-dimensional dynamical system [5]. In section 5.1 we have presented how this methodology has been leveraged to show how RNN implement context-dependent computations, with contexts re-configuring the effective dynamical system that governs the time evolution of latent variables.

These methodologies grant access to the inner workings of trained networks and this level of explainability appears well suited for i) applications of AI to the sciences, ii) practical applications of AI as well as for iii) theoretical investigations of neural computations.
These methodologies have been developed within the context of biological neural network modeling. As such they have allowed to tighten the link between experimental neuroscience and neural network theory by ascribing a functional role to various correlative experimental observations, and by making predictions for future experiments (section 5.2). Given the expanding use of deep-networks,

and RNN in particular, in various fields of science, like population genetics [1] or linguistic [25], this approach is expected to be useful to reveal the network mechanisms at stake in various natural phenomena.
Reverse-engineering of RNN trained on practical tasks have also been performed, confirming that the theoretical concepts presented in section 3 are also appropriate within this context. We have presented works focused on natural language processing tasks (section 6). By leveraging the insights provided by reverse-engineering RNN, researchers could a posteriori build minimal extensions of bag-of-words models that perform as well as more complicated RNN on a sentiment analysis task. This is reminiscent of the knowledge distillation approach [6] that has been proposed to reduce the complexity of trained machine learning models and make their commercial implementations more efficient. The insights gained from reverse-engineering studies can also be used to design better training algorithms by proposing new forms of regularization [17].
Finally the description of neural computations in terms of collective or latent variables, that can be thought of as encoding symbols (see e.g. section 3.1), might allow to formulate neural computations in a language closer to standard notions of computations. For instance, from the dynamical system point of view, the RNN trained on the 3-bit flip-flop task can be interpreted as a finite-state automaton, with attractors corresponding to machine states and saddle-points programming possible input-triggered transitions between states (see Fig. 3b). It would be interesting to describe what other dynamical features underlie the computing power of neural networks [30] and their ability to implement pushdown automata or Turing machines. Moreover, based on the comparison between latent variables and symbols, the dynamical system approach to neural computations might turn out relevant, as an exploratory tool, for the development of hybrid AI architectures, which combine the respective strengths of symbolic AI and artificial neural networks, and that have been identified as promising for the development of explainable IA architectures [4].

Acknowledgment

I would like to thank Michele Sebag, Thomas Bonald and Jean-Louis Dessalles for useful discussions about explainability.

Références

[1] Adrion, J. R., Galloway, J. G. Kern, A. D. Predicting the Landscape of Recombination Using Deep Learning. Molecular Biology and Evolution 37, 1790–1808 (2020).

[2] Aitken, K. et al. The geometry of integration in text classification RNNs. Proceedings of the International Conference on Learning Representations (2021).

[3] Barak, O. Recurrent neural networks as versatile tools of neuroscience research. Current Opinion in Neurobiology 46, 1–6 (2017).

[4] Beaudouin, V., Bloch, I., Bounie, D., Clémençon, S., D'Alché-Buc, F., Eagan, J., Maxwell, W., Mozharovskyi, P., Parekh, J. Flexible and Context-Specific AI Explainability : A Multidisciplinary Approach (2020)

[5] Beiran, M., Dubreuil, A., Valente, A., Mastrogiuseppe, F. Ostojic, S. Shaping dynamics with multiple populations in low-rank recurrent networks. arXiv :2007.02062 [q-bio] (2020).

[6] Bucila, C., Caruana, R., Niculescu-Mizil, A. Model Compression. Proceeding of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (2006).

[7] Ceni, A., Ashwin, P. Livi, L. Interpreting Recurrent Neural Networks Behaviour via Excitable Network Attractors. Cogn Comput 12, 330–356 (2020).

[8] Chaisangmongkon, W., Swaminathan, S. K., Freedman, D. J. Wang, X.-J. Computing by Robust Transience : How the Fronto-Parietal Network Performs Sequential, Category-Based Decisions. Neuron 93, 1504-1517.e4 (2017).

[9] Cunningham, J. P. Yu, B. M. Dimensionality reduction for large-scale neural recordings. Nature Neuroscience 17, 1500–1509 (2014).

[10] Dayan, P., Abbott, L.F. Theoretical neuroscience : computational and mathematical modeling of neural systems. Computational Neuroscience Series (2001).

[11] Derrida, B., Gardner, E. Zippelius, A. An exactly solvable asymmetric neural network model. EPL (Europhysics Letters) 4, 167 (1987).

[12] Dubreuil, A., Valente, A., Mastrogiuseppe, F. and Ostojic, S. Disentangling the roles of dimensionality and cell classes in neural computations. NeurIPS Neuro-AI Workshop (2019).

[13] Dubreuil, A., Valente, A., Beiran, M., Mastrogiuseppe, F. and Ostojic, S. Complementary roles of dimensionality and population structure in neural computations. bioRxiv (2020).

[14] Fanthomme, A., Monasson, R. Low-Dimensional manifolds support multiplexed integrations in recurrent neural networks. Neural Computation (2021).

[15] Golub, M. and Sussillo, D. FixedPointFinder : A Tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. Journal of Open Source Software, 3(31), 1003, https ://doi.org/10.21105/joss.01003 (2018).

[16] Graves, A. et al. Hybrid computing using a neural network with dynamic external memory. Nature 538, 471–476 (2016).

[17] Haviv, D., Rivkind, A. Barak, O. Understanding and controlling memory in recurrent neural networks. Proceedings of the 36th International Conference on Machine Learning (2019).

[18] Hertz, J., Krogh, A., Palmer, R. G. Introduction to the theory of neural computation. CRC Press (1991).

[19] Hopfield, J. J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proceedings of the National Academy of Sciences 79, 2554–2558 (1982).

[20] Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. Proceedings of the National Academy of Sciences 81, 3088–3092 (1984).

[21] Maheswaranathan, N., Williams, A., Golub, M., Ganguli, S. and Sussillo, D. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. Advances in neural information processing systems (2020).

[22] Maheswaranathan, N. Sussillo, D. How recurrent networks implement contextual processing in sentiment analysis. Proceedings of the 37th International Conference on Machine Learning, PMLR 119 :6608-6619, (2020).

[23] Mante, V., Sussillo, D., Shenoy, K. V. and Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. Nature 503, 78–84 (2013).

[24] Mastrogiuseppe, F. Ostojic, S. Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. Neuron, doi :10.1016/j.neuron.2018.07.003, (2018).

[25] Lakretz, Y. et al. The emergence of number and syntax units in LSTM language models. NAACL (2019).

[26] Peterson, G. E. Foundation for neural network verification and validation. In Science of Artificial Neural Networks II, volume 1966, pages 196–207. International Society for Optics and Photonics (1993).

[27] Schaeffer, R., Khona, M., Meshulam, L., Fiete Rani, I. Reverse-engineering recurrent neural network solutions to a hierarchical inference task for mice. NeurIPS (2020).

[28] Schuessler, F., Mastrogiuseppe, F., Dubreuil, A., Ostojic, S. Barak, O. The interplay between randomness and structure during learning in RNNs. Advances in neural information processing system (2020).

[29] Seung, H. S. How the brain keeps the eyes still. Proceedings of the National Academy of Sciences 93, 13339–13344 (1996).

[30] Siegelmann, H.T. and Sontag, E.D. On the computational power of neural nets. J. Comput. Syst. Sci., 50(1) :132–150 (1995).

[31] Strogatz, S.H. Nonlinear dynamics and chaos. Westview Press (1994).

[32] Sussillo, D. Barak, O. Opening the Black Box : Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. Neural Computation 25, 626–649 (2013).

[33] Sussillo, D. Neural circuits as computational dynamical systems. Current Opinion in Neurobiology 25, 156–163 (2014).

[34] Sussillo, D., Churchland, M. M., Kaufman, M. T. Shenoy, K. V. A neural network that finds a naturalistic solution for the production of muscle activity. Nature Neuroscience 18, 1025–1033 (2015).

[35] Sutskever, I., Vinyals, O. Le, Q. V. Sequence to Sequence Learning with Neural Networks. Advances in neural information processing system (2014).

[36] Tsodyks, M. and Sejnowski, T. Proceedings of the Third Workshop on Neural Networks : from Biology to High Energy Physics. Int. J. Neural Syst. (6) Suppl., 81 (1994).

[37] Wang, J., Narain, D., Hosseini, E. A. Jazayeri, M. Flexible timing by temporal scaling of cortical responses. Nature Neuroscience 21, 102–110 (2018).