Graceful degradation of recurrent neural networks as a function of network size, memory length, and connectivity damage

Cecilia Jarne\textsuperscript{a,b}, Rodrigo Laje\textsuperscript{a,b}

\textsuperscript{a}Universidad Nacional de Quilmes - Departamento de Ciencia y Tecnología
\textsuperscript{b}CONICET, Buenos Aires, Argentina

Abstract

Recurrent Neural Networks (RNNs) are frequently used to model aspects of brain function and structure. In this work, we trained small fully-connected RNNs to perform temporal and flow control tasks with time-varying stimuli. Our results show that different RNNs can solve the same task by converging to different underlying dynamics and that the performance gracefully degrades either as network size is decreased, interval duration is increased, or connectivity is damaged. Our results are useful to quantify different aspects of the models, which are normally used as black boxes and need to be understood in advance to modeling the biological response of cerebral cortex areas.

Keywords: Recurrent Neural Networks, connectivity, degradation

1. Introduction:

Recurrent Neural Networks (RNN) emerged a few decades ago (see for example the founding works of Hopfield [1], Elman [2] and Funahashi and Nakamura [3, 4]) to model the mechanisms of a variety of systems, such as brain processes [5, 6, 7, 8, 9, 10, 11], stability and control [12, 13, 14, 15], and in Machine Learning [16, 17, 18, 19].

Within the Machine Learning (ML) strategy, the main objective is to produce efficient network topologies and training methods to solve computational problems. On the other hand, the aim when implementing these models in Computational Neurosciences is to model the recurrent connectivity of the brain to understand the mechanisms that underlie different processes.
This paper takes relatively new training algorithms from the Machine Learning community and applies them to analyze computational problems related to temporal tasks relevant to neurosciences. In particular, we want to study the problem of how cortical subpopulation processes underlie various temporal tasks inspired by aspects of cognition.

Recurrent neural networks are powerful tools since it has been proven that, given enough units, they can be trained to approximate any dynamical system [20, 21, 22, 23]. It has been studied that RNN can display complex dynamics including attractors, limit cycles, and chaos [24, 25].

It is well established that RNN constitutes a versatile model in neuroscience research. They can be trained to process temporal information and perform different tasks such as flow control and many kinds of operations that roughly represent computation in different brain areas. A simple RNN model could perform tasks that are similar to stimuli selection, gain modulation, and temporal pattern generation in the cortex [26].

Trained networks serve as a source of mechanistic hypotheses and also as a testing ground for data analyses that could link neural activity and behavior. RNNs are a valuable platform for theoretical investigation, and some aspects of RNN models are used to describe a great variety of experimental results observed in different studies, for example working memory, motor control, temporal processing, and decision making [27, 28, 29, 30]. For instance, it has recently been studied that recurrent circuits in the brain may play a role in object identification [31].

Another aspect considered is the computational principles that allow decisions and action regarding flexibility in time. In a recent review [32], a dynamical system perspective is used to study such flexibility and shows how it can be achieved through manipulations of inputs and initial conditions.

Recent records of neurons in the cerebral cortex also show complex temporal dynamics [24, 33, 34, 9], where different mechanisms of information flow control could be present and coexist. Temporal aspects of RNN constraint the parameters, topologies, and different parts of the computation. Those aspects deserve to be studied and will allow improving current neuronal models. Given the complexity of these systems, it is not surprising that there are still so many fundamental gaps in the theory of RNNs [35], such as how RNNs control the flow of information [36].

A recurrent neural network can be trained to reproduce a task considering two paradigms: On the one hand, we can have an intuition about the behavior of the system. Here the network represents an abstract variable that obeys
an equation of a low-dimensional dynamical system, and the dynamics can be
translated into the connectivity of the network. The underlying mechanism is
modeled using a low-dimensional dynamic system that is then implemented
in a high-dimensional RNN [35].

The other paradigm is to build a functional RNN, which is used in ML
for training and frequently also in Computational Neuroscience. In this
case, one presents to the network the information relevant to the task to be
accomplished. This is done in terms of input-output data but without any
direct prescription on how to accomplish it. If the mechanisms implemented
by the network can be conceptualized, the network analysis can become a
method for generating hypotheses for future experiments and data analysis
[24]. The present work is based on this approach.

A cognitive task is typically composed of elementary sensory, cognitive,
and motor processes. We examined a set of neuroscience-inspired tasks: time
reproduction [37], Oscillatory response [8, 33], a flip flop [33] and a set of
decision making boolean-like tasks. These simple tasks have been used in
Cognitive and Computational Neuroscience in the context of RNN studies
[6, 24, 38], among others with the aim to understand how different brain
areas process information. In particular, we have taken a set of temporal
and decision-making tasks, parametrized them and trained recurrent neural
networks to reproduce them. We use in the process a new open-source
framework that we developed that could be useful for others to create their
other tasks of interest, based on the ones that we parametrized to develop
their models.

In this framework, we studied the properties of a simple model representing
a cortical subpopulation of neurons that, after training, can perform tasks
that are relevant to processing information and flow control. We will show
the results obtained using numerical simulation of our studies.

On the trained networks that we obtained, we observe the different con-
figurations that can emerge. We carry out also a set of studies on how the
performance of the trained network degrades in the frame of different scenarios
to show the scope and limitations of the model. These studies are related
to the performance degradation either as network size is decreased or the
time duration is increased, and what happens when connectivity of trained
networks is damaged.

The rest of the paper is organized as follows. In Section 2, we describe the
network model, training method, the details of the code implementation and
task parametrization. In section 3, we explain each task and describe how
those are implemented. Then, we present the results of the different studies that we performed. Section 4 we discuss the results obtained and finally in Section 5 we present the conclusions.

2. Methods

2.1. Model

Equation 1 rules the dynamics of the interconnected $n$ units (or neurons) with firing-rate activities $h_i(t)$ ($i = 1, 2, ..., n$):

$$\frac{dh_i(t)}{dt} = -\frac{h_i(t)}{\tau} + \sigma \left( \sum_j w_{ij}^{\text{Rec}} h_j(t) + \sum_j w_{ij}^{\text{in}} x_j \right)$$ (1)

In this equation, $\tau$ represents the time constant of the system, and $\sigma$ is a saturating nonlinear activation function. $x_j$ are the components of the vector $\mathbf{X}$ representing the input signal. The synaptic connection strengths $w_{ij}^{\text{Rec}}$ are the elements of the matrix $\mathbf{W}^{\text{Rec}}$, and $w_{ij}^{\text{in}}$ are the matrix elements of $\mathbf{W}^{\text{in}}$, the synaptic weights for connections with the input units. In order to read out the network’s activity it is common to include a readout $z(t)$ that computes a weighted average of all units in the recurrent network:

$$z(t) = \sum_j w_{ij}^{\text{out}} h_j(t)$$ (2)

Thus the recurrent neural network describes firing rates and does not explicitly model spiking units with action potentials [24]. Originally Equation 1 was used to model the state of the potential membrane. This equation has been used in numerous published works with different variants since Hopfield [1]. We also considered $\tau = 1$ ms.

In vector form, the equations 1 and 2 can be written as:

$$\frac{d\mathbf{H}(t)}{dt} = -\mathbf{H}(t) + \sigma(\mathbf{W}^{\text{Rec}} \mathbf{H}(t) + \mathbf{W}^{\text{in}} \mathbf{X}(t))$$ (3)

$$\mathbf{z}(t) = \mathbf{W}^{\text{out}} \mathbf{H}(t)$$ (4)

The units imitate biological neurons in a fully connected network in which a simplified set of important computational properties is retained.
2.2. From continuous to discrete time walking with small steps

When the model is implemented for its computation, it is always necessary to make a passage from the system in continuous time to a system with discrete time steps, with as much precision as necessary for modeling the considered problem.

Traditionally the system represented by Equation 3 is numerically integrated using Euler’s method with a time step $\delta t$, which is small compared to the timescale of the system following [39, 40, 41]. Then the dynamics of the RNN can be approximated by:

$H(t + \delta t) = H(t) + (-H(t) + \sigma(W^{\text{Rec}}H(t) + W^{\text{in}}X(t)))\delta t,$  \hspace{1cm} (5)

for small enough time step with no further discussion regarding the discrete or continuous nature of the system. Very early works have proved that it is possible to use discrete-time RRN to uniformly approximate a discrete-time state-space trajectory which is produced by either a dynamical system or a continuous-time function to any degree of precision [42, 3].

Modern scientific open-source libraries, such as TensorFlow for high-performance numerical computation and particularly Keras, allows implementing architectures such as equation 5 directly with a high-level neural networks API, written in Python and capable of running on top of TensorFlow. Keras framework has a large set of architectures and training algorithms that have already been tested by the ML community with a detailed documentation [43].

2.3. On training methods

There are great variety of algorithms to train recurrent neural networks. In a recent work from the ML field, a very detailed survey on RNNs with new advances for training algorithms and modern recurrent architectures was presented in [44].

In general, the training methods for neural networks can be unsupervised or supervised. In this work, we will focus on applying a supervised method.

The studies where some form of gradient descent is applied stand out in the literature of supervised methods. An example is the Reservoir computing paradigm with liquid- or echo-state networks [45], where the modifications of the network weights are made in the output layer weights $W^{\text{out}}$. 

5
One of the most outstanding methods was the one developed by Sussillo and Abbot. They have developed a method called FORCE that allows them to reproduce complex output patterns, including human motion-captured data [46]. Modifications to the algorithm have also been applied successfully in various applications [47, 48].

A frequently used alternative is gradient descent with backpropagation for its calculation and then some optimization method for minimizing it. Given the recent advances in the implementation of this method with Open Source libraries, this is the method we explored.

We were inspired in the algorithm selection for training by [44] and [49]. In [49], authors use the Adam method to train networks and obtain numerical examples. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions [50]. It is based on adaptive estimates of lower-order moments. This method is straightforward to implement and computationally efficient.

Adam’s method has small memory requirements for implementation and is invariant to a diagonal rescaling of the gradients. This algorithm is well suited for problems with many parameters or involving large datasets. It is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients [50]. Adam’s algorithm began to be used in neuroscience only recently [51].

Keras has a software implementation for this algorithm, and it is the one used in the present work.

2.4. Network implementation and training protocol

We use a simple RNN model with three layers. One is the input, the second is the recurrent hidden layer, and the last is the output layer. The input layer has \( N \) input units. The inputs to this layer are a sequence of vectors through time \( t \), whose components are \( x_j \). Every input neuron is connected to every neuron of the hidden layer. The connectivity weight between input neuron \( i \) and hidden neuron \( j \) is \( w_{ij}^{in} \) in \( \mathbf{W}^{in} \). The hidden layer has \( M \) hidden units \( h_t = (h_1, h_2, ..., h_M) \), that are also connected to each other with recurrent connections weights \( w_{ij}^{Rec} \) in \( \mathbf{W}^{Rec} \). The initialization of the connections \( w_{ij}^{Rec} \) of hidden units using small non-zero elements can improve the overall performance of the training. The output layer is represented by the \( \mathbf{W}^{out} \) matrix, also connected to the hidden layer with weights \( w_{ij}^{out} \).

In the present work, we implemented a recurrent neural network with \( M = 50 \) hidden units, or indicated otherwise. We used as activation function
the hyperbolic tangent.

The weight matrix $\mathbf{W}^{\text{in}}$ was initialised randomly from the uniform distribution with the Glorot uniform initializer from Keras, which is the default selection. It draws samples from a uniform distribution within $[-\text{limit}, \text{limit}]$, where $\text{limit} = \sqrt{6}/(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})$, ($\text{fan}_{\text{in}}$ is the number of input units in the weight tensor and $\text{fan}_{\text{out}}$ is the number of output units [52]). Recurrent weights matrix $\mathbf{W}^{\text{Rec}}$ is initialized in two different ways during the study: either as a general random matrix or as an orthogonal random matrix. The general random matrix has elements drawn from a normal distribution with $\sigma = 1/\sqrt{N}$ and $\mu = 0$, where $N$ is the number of recurrent units. [53]. The orthogonal random matrices are created with Orthogonal initializer from Keras, where an orthogonal matrix is obtained by QR decomposition of a matrix of random numbers drawn from a normal distribution.

This selection allows us to implement Adam as a training method with low computational cost and diminish the impact of the vanishing gradient problem for this simple implementation.

The upper panel of Figure 1 shows a scheme for the neural network model studied in this work. The input signal is a two-component vector with arbitrary time evolution in each input unit. The output is the readout unit that provide the output decision. The lower panel shows and example for a Neural Network connectivity matrices $\mathbf{W}^{\text{in}}, \mathbf{W}^{\text{Rec}},$ and $\mathbf{W}^{\text{out}}$.

For instance, let us consider a network with 50 units that are further used in the analysis presented in Section 3. The units in the hidden layer are fully connected to each other, in the sense that the connectivity matrix $\mathbf{W}^{\text{Rec}}$ is not sparse. Let us consider a two-input task, meaning that the input layer has two input units. The input layer $\mathbf{W}^{\text{in}}$ will have $50 \times 2$ connections to process a $2 \times [\text{length}]$ vectors at the input, in general we considered time series of 200 ms. The $\mathbf{W}^{\text{Rec}}$ matrix has $50 \times 50$ connections that produce the temporal response of the 50 units with a vector of $50 \times [200\,\text{ms}]$. At the output we have one unit representing the output decision that combines the 50 activity responses of the hidden units, using the $\mathbf{W}^{\text{out}}$ matrix of $1 \times 50$. It produces a vector $\mathbf{Z}(t)$ of $1 \times [200\,\text{ms}]$.

The loss function used to train the model is the mean square error between the target function and the output of the network. It is defined as:

$$E(w) = \frac{1}{2} \sum_{t=1}^{M} \sum_{j=1}^{L} |Z_j(t) - Z^{\text{target}}_j(t)|^2,$$

(6)
Figure 1: Upper panel: neural network schema of our model with two units at the input to process the time series with arbitrary value and one output. Lower panel: example for a Neural Network connectivity matrices $W^{\text{in}}$, $W^{\text{Rec}}$, and $W^{\text{out}}$ for a 100 unit recurrent network (arbitrary size) with two units at the input and one at the output. Rows represent the output connection (post-synaptic) of each unit or column (pre-synaptic).

where $Z^\text{target}_j(t)$ is the desired target function and $Z_j(t)$ is the actual output.

We trained the model with no less than 15,000 input signal samples for all tasks described in the following section. We generated each sample of the training set as a noisy time series that may contain a square pulse with noise or only noise. Noise is an additive random variable drawn from a Gaussian distribution with zero mean and amplitude of 10% the height of the pulse.

We also generated an additional set of time series that we use to test after training. We check if the output response against the testing time series corresponds to the task for which the network has been trained.

In each experiment, we saved the initial pre-training configuration and the final instance of the network weights to study how the weight matrix changes.
During training. The aim of the training is to adjust all the parameters and obtain a network that can reproduce the task for which it was trained.

The model was implemented in Python using Keras and TensorFlow as indicated in Section 2.2. That allows us to make use of all the algorithms and optimizations developed by that ML community. Framework and Code to train the networks and produce the Figures in this paper is open and available in the following repository:

https://github.com/katejarne/RNN_study_with_keras

2.5. Time scale and general aspects of the taks

We considered that every input signal may have a square pulse of fixed duration, and the network responds according to the rule that it was trained for a given data set of the training sample.

The time scales considered are of the order of tens of milliseconds in each case, part of the range of interest of signals processed by the cerebral cortex [54]. The value of the time step for the time evolution is $\delta t = 1 \text{ mS}$. Then, from considering equation 5 with such time step, the activity of the recurrent units at the next time step is:

$$
\mathbf{H}(t+1) = \sigma(\mathbf{W}^{\text{Rec}} \mathbf{H}(t) + \mathbf{W}^{\text{in}} \mathbf{X}(t))
$$

Equation 7 is the one that we implemented. We considered the training data set as the set of inputs signal and target outputs with a low edge-triggered response to the input signals with a delay time of 20 mS. In each task presented in the following section, we show a trained network responding to different testing samples. Each stimulus at the input is presented in green (input signal 1) and pink line (input signal 2) for those tasks with two inputs. The target is in a grey solid line, and the network response is in red. We considered a time series of between 200 ms and 500 ms length, but the length and the position of the stimulus are arbitrary.

2.6. Description of the tasks considered in the study

The networks learn to decide which state the output signal should take in the face of the different stimuli presented at the input. In this sense, one task is defined as the different possible ways in which it can decide the output state based on input stimuli and the respective regimes that the output should take.
The motivation for the selection of the considered tasks is to simulate flow control processes that can occur in the cortex when receiving stimuli from subcortical areas. In [36] the notion of gating was discussed as a mechanism capable of controlling the flow of information from one set of neurons to another. In the present work, the gating mechanisms are model using networks with a relatively small set of units.

It has been proposed that some sets of neurons in the brain could roughly function as gates [36]. It also is interesting the dynamics of trained networks for the Flip Flop task, which is generally related to the concept of working memory. It has been previously studied in [33, 24], but in this case, with a more complex task referring to a 3-bit register called in the paper a 3-bit Flip Flop.

We focus on the study of networks trained for the following list of tasks related to the processing of stimuli as temporal inputs:

1. Time reproduction.
2. Basic logic gate operation: AND, OR, NOT, XOR.
3. Flip-Flop (1-bit memory storage).
4. Finite-duration oscillation.

It should be noted that the tasks described in item 2 are not related to those made by static feedforward networks like [55] but to a network solving logic gates with time-varying inputs. We want to point out that, in item 3, we are not referring to the concept of “Flip Flop neurons” such as the one proposed in [56] but to a network learning the “Flip-Flop rule” as in [33] with two inputs. The focus is on the process of temporal signals similar to the Xor temporal task implemented in [2]. In every task, the focus is on the processing of pulsed (i.e., non-stationary) signals.

The RNN model emulates a “cognitive-type” cortical circuit such as the prefrontal cortex, which receives converging inputs from multiple sensory pathways and projects downstream to other areas. The chosen network architecture and size showed to be sufficient to learn all the tasks mentioned above. We used dimensional reduction methods to study the inner state of the network during and after training and discuss, specifically, the results and observations regarding each task [57]. In particular, PCA is the method that we chose because it has been widely used in the study of simulations, as well as experimental high dimensional neural space states [58].

For the network implementation and training, we use the Keras libraries from [43] and TensorFlow from [52] as frameworks, instead of more traditional
choices like Matlab from [59] or Theano from [60], implemented in some works such as [61]. The reason for our selection is that these new scientific libraries are open source, and their use is rapidly growing. In the case of Keras, it is the first time that it is used for such kind of study. In the case of TensorFlow, there are a few recent works that use it (for instance, see Ref [62]).

3. Results

The results of the numerical studies are shown in this section. First, we present in Section 3.1 a description of each task and the parametrization used in the framework that we developed. This framework could be used to modify any of the seven tasks that we considered, but also it could it is possible to use it for developing different tasks of interest. In Section 3.2 we discuss how different initialization schemas could improve the network training.

Section 3.3 shows different aspects that we observed of the dynamics of the trained networks. Finally, Sections 3.4 and 3.5 show the result of different studies performed on the memory capacity, scale and damage of the trained networks.

To follow the examples shown in Section 3.2 and Section 3.3 (Figures 6 to 11), the trained networks have been labelled. The labels include a number to identify the simulation number of the corresponding task and initial condition. These also allow identifying the data in the Supplementary Information to view the examples and the repository on Github.

3.1. Tasks

3.1.1. Time reproduction

Let’s begin by describing a simple temporal task considered in Section 2.6. In this task, when the network has a stimulus at the input (a Gaussian pulse with noise), it has to respond with a pulse at the output matching the input signal, and no response otherwise.

We trained the network to produce an output pulse at a fixed time delay after an input pulse occurs, and output close to zero if there is no input pulse. In Figure 2, we show six successful runs for different input samples. The same neural network is considered, trained to memorize and reproduce a Gaussian pulse with noise at the input, after a 20 ms delay. If there is no pulse, the network must output a zero signal. This task was used in the scaling studies presented in Section 3.4.
3.1.2. Binary basic operations between input stimuli with AND, OR, XOR and NOT

In these sets of tasks, the network has to perform different binary inspired operations with temporal stimuli at the input (or inputs). The result of the output will be switching to a HIGH value or keeping a LOW value depending on the task and the presented input(s). The input stimuli are square signals with a duration of 20 ms and Gaussian noise of 10% of the total amplitude. We considered our input data set random time series of 200 ms length with or without a pulse. For two-input tasks, the corresponding input pulses (if present) are simultaneous in time. The network has to decide the state of the output, which matches the training set rule for each considered task. We used 50-unit networks, and all networks were able to successfully reproduce all tasks after training.

For each of the tasks, we created the target output time series according
to table 1. The truth tables for all logic operations are displayed in Table 1.

| Input 1 | Input 2 | And Output | Or Output | Xor Output |
|---------|---------|------------|-----------|------------|
| 0       | 0       | 0          | 0         | 0          |
| 0       | 1       | 0          | 1         | 1          |
| 1       | 0       | 0          | 1         | 1          |
| 1       | 1       | 1          | 1         | 0          |

Table 1: AND, OR and XOR states of the output with respect to the inputs states.

3.1.3. NOT task

The boolean NOT task consists of turning the output to a “High level” state when input is in a “Low Level” state and vice versa. In Figure panel d of Figure 3, we show the state of the output compared with the input.

Figure 3 shows the temporal response for four neuronal networks, each trained to perform the Boolean operations indicated for each panel when receiving the stimuli at the inputs: And, Or, Not, and Xor. The green and pink lines show, in each case, the time series of each input, the thick black line the target output, and the red line the status of the output.

3.1.4. Flip-Flop (1-bit memory storage)

In this study, we trained a network with two inputs with different functions. One works as a “Set” signal, and the other is a ‘Reset” signal. If the network receives a pulse in the S-input, then the output turns to HIGH. If the network receives a stimulus in the R-input, then the output turns to LOW. Two consecutive pulses do not change the output state. Table 2 summarizes the rule learned by the network. Time series are 400 ms in length to show different changes in the inputs during the same time-lapse.

In Figure 4, the temporal response of a neural network trained to perform the “Flip Flip” task with its Set and Reset inputs is shown. Each panel shows six possible random time series from the testing data set. The green and pink lines show the time series of each input, the thick black line the target output, and the red line the status of the output. The training data set consists of trains of pulses at the S-input and R-input with noise and a target output according to Table 2. We successfully obtained a set of neural networks capable of performing the Flip-Flop task.
Figure 3: Four trained neural networks responses for 6 testing samples for: a) AND operation between two stimuli, b) OR operation, c) XOR operation and d) NOT operation applied to the input. Time is in ms and amplitude in arbitrary units.

3.1.5. Finite-duration oscillator

In this task, we trained a network to obtain in the output an oscillation with a frequency of 30 Hz, 20 ms after a pulse in the input. When the network receives the stimulus, the output behaves as it is shown in Figure 5. If the network has no stimulus, the output remains at the “LOW” state. Once again, we successfully trained a set of neural networks that perform this
Table 2: Flip Flop task table. $Q_N$ means that the output remains at the previous state. $X$ means that the state is forbidden for the data set.

| Set | Reset | Output state |
|-----|-------|--------------|
| 0   | 0     | $Q_N$        |
| 0   | 1     | 0            |
| 1   | 0     | 1            |
| 1   | 1     | $X$          |

Figure 4: The response of a trained neural network for six testing samples for the Flip Flop task. S-Signal in pink, R-Signal in green. Grey thick line is the Target output, and red is the Network response. The output state depends on the states of the set and reset signals, regardless of the length of the time series considered.

Figure 5 shows the temporal response of a neural network trained to oscillate when receiving a stimulus at the input. Possible time series of the test data set is shown in Figure 6. Upon receiving a pulse at the input, the output must respond with oscillation for a certain time. In all Figures (from 2 to 4), the networks have 50 units and 20 ms for the response time.
Figure 5: A trained neural network response of the output to 6 testing samples for the “finite-duration oscillator” task. The network does not spontaneously oscillate unless there is a pulse in the input.

3.2. Network initialization studies

We will show different properties of the network activity and connectivity based on population analysis employing PCA and the estimation of the eigenvalues of the recurrent weight matrix.

We started by training a set of networks to perform each of the considered tasks described in the previous section. Each network was created by randomly choosing the connectivity strengths $W_{\text{Rec}}$ from a normal distribution with zero mean and variance $1/N$, as described in Section 2.1. We considered two cases: orthogonal matrices and non-orthogonal matrices (20 matrices each case). We studied the eigenvalue spectrum of the recurrent matrix $W_{\text{Rec}}$. The eigenvalues of two sample matrices are shown in the upper and lower left panels of Figure 6.

The non-orthogonal matrix, previous to the training (upper left panel), shows a distribution consistent with the random matrix theory of Girko’s circle law [63], which states that, for large N, the majority of eigenvalues of an
$N \times N$ asymmetric random matrix lie uniformly within the unit circle in the complex plane and the fraction of eigenvalues lying outside the circle vanishes in the limit $N_{\text{units}} \to \infty$, when the elements are chosen from a distribution with zero mean and variance $\frac{1}{N}$.

In the orthogonal matrix, the eigenvalues lie at the border of the circle (bottom left panel of Figure 6) As a result of the training, some eigenvalues are pushed out of the circle. In the case of the networks shown in Figure 6, both final configurations correspond to fixed-point configurations with one eigenvalue with real part greater than one and zero imaginary part, and the rest of the eigenvalues scattered within the unit circle. Here we show a comparison between the initial state (left panels) and post-training (right panels) of Figure 6.

We found in our simulations that this behavior is consistent with estimates made previously in [64, 65], even when our networks are trained (i.e. non-random) and input-driven. The system will either show nontrivial stationary solutions or oscillations depending on the nature of the eigenvalue with the largest real part of the connectivity matrix as in [65]. A more profound explanation will be given in a further work in process which is beyond the scope of the present manuscript.

For each of the tasks and the two conditions (orthogonal and non-orthogonal), we estimated the rate of networks that successfully passed the training. The results are shown in Table 3. The orthogonal condition slightly improves the success rate for each task. This is consistent with studies previously conducted by [66]. A possible explanation for the success rate differences between the two possible initial conditions is that at the training stage is “easier” to pull out the eigenvalues when they are placed in the edge of the circle (orthogonal condition) than when they are scattered within the unit circle (non-orthogonal condition).

The time reproduction task shows a perfect training rate (100%) with both initializations. We think that this is because this is the simplest task to be learned for the network. This is also why we chose this task for our scaling studies in the following sections.

From this Figure, it is interesting to note that there are a few eigenvalues that dominate the behavior of the states that result in the task for which the network was trained.

The correlation between eigenvalue spectra and the dynamics of neural networks has also been studied in [67], relating the design of networks with memory face with the eigenvalues outside the circle in the complex plane.
| Task                          | Initial orthogonal | Initial Rand Normal |
|-------------------------------|--------------------|---------------------|
| And                           | 85%                | 65%                 |
| Or                            | 90%                | 80%                 |
| Xor                           | 90%                | 55%                 |
| Not                           | 90%                | 45%                 |
| Flip Flop (1-bit memory storage) | 95%           | 65%                 |
| Oscillatory                   | 90%                | 65%                 |
| Time reproduction             | 100%               | 100%                |

Table 3: The success rate for the training of 20 networks for orthogonal initial condition compared with the random normal initial condition.

3.3. Network dynamics

Now we consider networks trained in the AND task. We want to show the behavior of the recurrent units. We studied the response to the stimuli corresponding to the four different input configurations (Table 1). We consider the response to noiseless stimuli to show the undisturbed trajectory and thus build a geometrical description of the phase space. All networks were trained with noisy input signals, as previously described.

We plot the components $h_i(t)$ from the $H(t)$ from Equation 5, this is the temporal evolution of all recurrent units. We applied Principal Component Analysis (PCA) to this set in each case.

Figure 7 and Figure 8 show the behavior of a trained network (labeled as #ID14 in Supplementary Materials) when the stimulus is applied. The left side of the Figure shows the output and inputs signals vs. time in the upper panel and some recurrent units $h_i(t)$. The right panel shows the time evolution of the first three PCs. Figure 7 corresponds to “LOW-LOW” and “HIGH-HIGH” and Figure 8 for the other combinations “LOW-HIGH” and “HIGH-LOW”. All four cases are represented by stable fixed points. When no pulse is applied to the inputs, the activity of every recurrent unit is zero (Figure 7 (a)). When both inputs are pulsed, the recurrent units are perturbed, and then the system migrates to a fixed point that is different from zero (Figure 7 (b)). When any single input is pulsed (Figure 8 (a) and (b)), the network dynamics converge to fixed points that are different from the previous two and are also different from each other. It can be seen that the recurrent units change their activity to reproduce the learned behavior (HIGH output) with a different final internal state that depends on which
input was activated.

We repeated this analysis for all trained networks and found that the way to achieve the desired trained rule is not unique, which is consistent with [68]. We identified different dynamical regimes for learning the same rule by different networks. We show in Figure 9 and 10 the results for a different network trained in the And task. The HIGH output state (corresponding to “LOW-LOW” and “HIGH-HIGH” stimuli for the inputs, Figure 9) is solved with two different stable fixed points as before, yet the LOW output state (“LOW-HIGH” and “HIGH-LOW” stimuli for the inputs, Figure 10)
is represented by two different limit cycles (i.e., an oscillation in the activity of the recurrent units). In this case, the $W_{\text{Rec}}$ matrix has one real and two complex conjugated leading eigenvalues.

From these results, it is clear that the same task can be performed with different internal configurations. In the case of the first network (#ID14, Figures 7 and 8), the task is solved by converging to stable fixed point recurrent states. In this case, the distribution of eigenvalues of the trained matrix has pure real dominant eigenvalues.

On the other hand, in the network identified as #ID04 (Figures 8 and 10) the input configurations “01” and “10” produce oscillatory recurrent states, while “00” and “11” produce stable fixed point recurrent states. In this case, the leading eigenvalues are complex conjugates (i.e., nonzero imaginary part).

The same situation for the other configurations occur when considering the different tasks studied, except for the case of oscillation caused by a stimulus presented in Section 3.1.5, where there the internal state is always oscillatory.

It is worth noting that we have found the same behavior in larger networks (500 units), i.e., behavior driven by a small set of eigenvalues and a variety of dynamical solutions underlying the same learned task (See Supplementary Materials). On the other hand, and perhaps not surprisingly, the finite-duration oscillation task was learned by all networks in the same way—a limit cycle [33].

Next, we show a network that learned the Flip-Flop Task in Figure 11. The input A represents the “Set” Signal, and the input B represents “Reset”. The upper panel shows an example of “Set” followed by “Reset”, and the lower panel “Reset” followed by “Set”. The HIGH output solution is a stable fixed point, while the LOW output is a stable limit cycle. It is interesting to note that a reset signal will take the system to a state different from the one that started with before stimulation, even if the output must go back to the same value (zero). This behavior is also ruled by the three leading eigenvalues: one real and a set of complex conjugated, as it is shown in the bottom panel of Figure 11.

Other examples of dynamics in the trained networks are available in the repository, for all studied tasks and different initial conditions, showing the different possible internal states achieved with different pre-training connectivity weights (Supplementary materials).

In this subsection, we showed that a small, fully connected nonlinear RNN
Figure 7: Left: Neural network #ID14 response in the AND task (“LOW-LOW” input and “HIGH-HIGH” input). Both configurations are represented by stable fixed points. Right: the first three PCs for the same dataset.

Trained with ADAM and backpropagation through time can successfully learn and reproduce many different tasks. We also showed that the post-training phase space is not uniquely determined by the learned task, as different dynamical solutions (for the recurrent units) are compatible with a single learned behavior (output unit), which is consistent with [68].
3.4. Network memory capacity and size scaling

Another interesting aspect of the trained networks is that they are translationally invariant in time, even though they were always trained with the stimulus occurring at the same moment. This situation is shown in Figure 12. The network correctly responds to the input pulse no matter when it arrives. See the Supplementary Information for translational time invariance of the
“And” task.

We asked ourselves how much time between the stimulus and the answer is possible to learn for particular network size. This problem is related to the well-known vanishing gradient problem concerning long-time dependencies. When estimating the gradient for further minimization, by applying the chain rule the value of the gradient will show short- and long-time dependencies on

"And" task.

Figure 9: Left. Neural network ID04 response of the output in the AND task (“Low -Low” input and “High - High” input). Right: PCA analysis for the same dataset. Top left panel shows inputs, target, and output. Bottom left panel shows 25 individual $h_i(t)$ states.
past values. Long-time dependencies suffer from the shrinkage of the gradient (see Supplementary Material). In the context of ML, this problem was solved by using other recurrent network architectures such as LTSTM and GRU [69, 70].

To show the temporal limitations of the model, we performed a study where we trained a set of networks on the time reproduction task, and then
measure the rate of success in terms of Euclidean distance between target and output. The top panel of Figure 13 shows the result of our study. Each point in the plot is obtained as the average of the distance obtained for the considered set of 20 networks trained for that time interval of response. For this task and a particular duration of the time series, a distance equal to 1 means that all networks reproduce the desired output for the task for the sample test set with good performance. When a distance is equal to 1.8, it means that almost none of the networks could reproduce the task given the worst-case distance between target and output, meaning the maximum possible distance between signal and target.

Our results presented in the upper panel of Figure 13 shows that the mean distance increases with the time duration until it reaches the worst performance at around 120 mS.

Next, we considered a fixed 150 ms delay between input signal and response, where the success rate is low. We then increased the number of recurrent units. The results are shown in the bottom panel of Figure 13. For a fixed time interval, the memory capacity improves with the size of the network as expected, reaching the best performance when the number of units approaches 200. Larger networks do not give any additional advantage.

3.5. Network response to damage

We induced post-training “damage” to a network that was previously successfully trained by removing connections. We then measure the performance of the network as a function of the degree of damage.

We considered a set of 10 successfully trained networks (50 recurrent units) that perform the AND task. Since we are using fully connected networks, the total number of connections is 2500, including positive and negative connections. We removed the connections gradually in a symmetric way by zeroing a growing number of the connections from the smallest (in absolute value) up in the connectivity distribution. For each percentage removed from each network, we calculated the distance between the target and the output in the four possible combinations of input values, and then the average.

The result of this study is shown in Figure 14, trained and damaged connectivity distributions in the insets. Colored lines represent the results for every input configuration of the “And” task (Table 1). In this plot, all connections up to the given percentage are removed. It is clear that, when we removed all the connections (positive and negative) with strength in the lowest 14%, the output of the networks deteriorates (the distance between
output and target is larger than 1 for any input configuration). At 20% of connections removed, the distances are larger than 1.5, meaning the networks stop working correctly (output very different from the target). In panels b and c of Figure 14, we show the result of removing either positive or negative only connections, respectively, up to the given percentage. Here we show that the networks are disrupted, if we remove either only positive or only negative connections. Both types are equally necessary to perform the considered task, and there is no apparent difference for either sign, which is consistent with our generic networks (no distinct excitatory and inhibitory subpopulations).

Next we study what happens if we remove a single percentile (Figure 14 d and e; for instance, 14% means that all connections between 13% and 14% are zeroed). The output deteriorates at values a little higher than before. These results show that both the connectivity strength and the number of removed connections are important.

The effect of removing connections on the network causes the learned task to deteriorate or destroy, whether one removes bands of connections of a certain intensity above a value, or if one accumulates the removal of many connections. The effect does not depend on whether we remove negative or positive connections. The cumulative effect of removing the lower connection portion produces deterioration at a lower percentage intensity value.
Figure 11: Left. Neural network #ID06 in the Flip-Flop task. Right PCA analysis for the same dataset. Top left panel shows inputs signals, target, and output. Bottom left panel shows 25 individual $h_i(t)$ states. Bottom panel: Eigenvalue spectrum of the $W^{Rec}$ matrix.
Figure 12: Time transnational invariance for the stimulus for an “Time reproduction” task. The trained network is stimulated with a time series where the stimulus pulse occurs in different moments. The pink line represents the state of the input signal. The Grey line represents the output response and the red thick line the output target.
Figure 13: Target-output distance in the time reproduction task and scaling properties. Upper panel: Mean distance between target and output vs. time between stimulus and response of the learned task. Bottom panel: Mean distance between target and output vs. size between stimulus and response of the learned task. A distance equal to 1 means that all networks reproduce the desired output for the task. A distance is equal to 1.8, it means that almost none of the networks could reproduce the task, given the worst-case distance between target and output.
Figure 14: Damage analysis: results of removing connections in RNN. (a) Removal of the smallest (in absolute value) connections up to the corresponding percentile. (b) and (c)) Same as (a), but only positive or negative connections, respectively. (d) and (e) Removal of connections within the corresponding percentile only (positive and negative connections, respectively). Mean ± standard error across 10 networks. Within each of the five figures is represented an example of how a band of weights is removed from the distribution of weight connections.
4. Discussion

Decision-making in a time processing task involves the perception, production, comparison, and maintenance of time intervals in working memory [41]. These processes are crucial for animals to anticipate or act correctly at the right time. Neural networks models are useful to understand computation in this context.

Training recurrent networks to imitate different bio-inspired or cognitive tasks is not new. However, in the literature, there are not many examples of open-source frameworks to use. Some of them are very recent. Two examples of useful code that can be used are: [33, 6]. In [33] the main focus is the dynamics and fixed points and in [6] RNNs with a particular structure are considered.

Our work also is a new open-source example that can be used and modified to include different aspects. Also, within the decision-making tasks, a subset of those that we proposed had not been implemented and characterized before. We obtained different realizations for the same task with different dynamical behaviours, which is consistent with [68] as indicated in Section 3.3.

To our knowledge, this the first time where initialization differences in trained RNNs for bio-inspired tasks are studied by comparing Orthogonal Initial condition and Random Normal initial condition, but also the first detailed study on the network scale in terms of temporal response and size concerning the capacity to learn such tasks.

Also, this is the first study performed on the trained RNN with damage to provide insights into how robust is a trained network in terms of its connectivity.

A better understanding of model constraints for simple tasks, such the ones studied in the present work, could help to develop better models in computational neuroscience.

5. Conclusions

We have presented the results of a set of studies performed on RNNs trained to perform various temporal and flow control tasks. We showed that small-sized networks with simple rate models for the individual units are adequate to learn and perform tasks in response to temporally dynamic inputs. We also showed that recurrent networks can learn a given task by developing different internal dynamics—for instance, a constant value in the output can
be produced by the recurrent network either converging to a fixed point or entering a limit cycle [25].

With this study, we were able to characterize the memory limits for a given trained network, showing a trade-off between network size and target duration for a simple task. We explicitly showed how the problem of the vanishing gradients arises as the target duration is increased, which would help when selecting a specific model, network size, and target time scale. Finally, we showed how much damage can sustain a trained network before collapsing. In our model, it is the cumulative effect of removing connections that have a greater effect, rather than the value of the largest connections removed. In other words, we observed, given this training scheme and topology, in which way the task is broken when deactivating certain parts of the network and which part of the weights is significant.

The three analyses proposed, are useful when building neural network models used in computational neuroscience and that frequently their details are not presented, or if they are, they are limited to a small paragraph of supplementary information. In this work, we analyzed a simple model of a small network, making available a numerical framework for the use of an open-source tool in neuroscience studies. One must be cautious when interpreting model results, particularly in complex systems where a mechanistic interpretation of the model behavior is difficult to develop, thus obscuring its relationship with reality. We showed that certain characteristics can emerge naturally when applying network training methods and that the responses obtained due to them are varied and do not necessarily reflect phenomena typical of the cortex, but rather typical of the model used. So the appropriate hypotheses must be considered to make a truly descriptive model.

Further steps in our studies will include a description of networks with excitatory and inhibitory units.

Acknowledgments

Present work was supported by CONICET and UNQ. We want to thank the anonymous reviewers for their careful reading of the manuscript and their insightful comments and suggestions.

References

[1] J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, Proceedings of
the National Academy of Sciences 81 (10) (1984) 3088–3092. doi:10.1073/pnas.81.10.3088.

[2] J. L. Elman, Finding structure in time, Cognitive Science 14 (2) (1990) 179 – 211. doi:https://doi.org/10.1016/0364-0213(90)90002-E. URL http://www.sciencedirect.com/science/article/pii/036402139090002E

[3] K. Funahashi, Y. Nakamura, Approximation of dynamical systems by continuous time recurrent neural networks, Neural Networks 6 (6) (1993) 801–806. doi:10.1016/S0893-6080(05)80125-X.

[4] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (3) (1989) 183–192. doi:10.1016/0893-6080(89)90003-8.

[5] W. Gerstner, H. Sprekeler, G. Deco, Theory and simulation in neuroscience, Science 338 (6103) (2012) 60–65. doi:10.1126/science.1227356.

[6] H. F. Song, G. R. Yang, X.-J. Wang, Training excitatory-inhibitory recurrent neural networks for cognitive tasks: A simple and flexible framework, PLOS Computational Biology 12 (2) (2016) 1–30. doi:10.1371/journal.pcbi.1004792.

[7] J. A. Michaels, B. Dann, H. Scherberger, Neural population dynamics during reaching are better explained by a dynamical system than representational tuning, PLOS Computational Biology 12 (11) (2016) 1–22. doi:10.1371/journal.pcbi.1005175. URL https://doi.org/10.1371/journal.pcbi.1005175

[8] T. Hoellinger, M. Petieau, M. Duvinage, T. Castermans, K. Seetharaman, A.-M. Cebolla, A. Bengoetxea, Y. Ivanenko, B. Dan, G. Cheron, Biological oscillations for learning walking coordination: dynamic recurrent neural network functionally models physiological central pattern generator, Frontiers in Computational Neuroscience 7 (2013) 70. doi:10.3389/fncom.2013.00070.

[9] C. Pehlevan, F. Ali, B. P. Ölveczky, Flexibility in motor timing constrains the topology and dynamics of pattern generator circuits, Nature Communications 9 (2018). doi:10.1038/s41467-018-03261-5.
[10] D. Sussillo, M. M. Churchland, M. T. Kaufman, K. V. Shenoy, A neural network that finds a naturalistic solution for the production of muscle activity, Nature Neuroscience (2014). doi:https://doi.org/10.1038/nn.4042.

[11] D. Remington, Evan D.and Narain, M. Hosseini, Eghbal A.and Jazayeri, Flexible sensorimotor computations through rapid reconfiguration of cortical dynamics, Neuron (2018) 0896–6273doi:doi:10.1016/j.neuron.2018.05.020.

[12] J. Deng, Dynamic neural networks with hybrid structures for nonlinear system identification, Engineering Applications of Artificial Intelligence 26 (1) (2013) 281 – 292. doi:https://doi.org/10.1016/j.engappai.2012.05.003.

[13] H. Dinh, R. Kamalapurkar, S. Bhasin, W. Dixon, Dynamic neural network-based robust observers for uncertain nonlinear systems, Neural Networks 60 (2014) 44 – 52. doi:https://doi.org/10.1016/j.neunet.2014.07.009. URL http://www.sciencedirect.com/science/article/pii/S089360801400166X

[14] N. Mohajerin, S. L. Waslander, State initialization for recurrent neural network modeling of time-series data, in: 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2330–2337. doi: 10.1109/IJCNN.2017.7966138.

[15] A. Rivkind, O. Barak, Local dynamics in trained recurrent neural networks, Phys. Rev. Lett. 118 (2017) 258101. doi:10.1103/PhysRevLett.118.258101. URL https://link.aps.org/doi/10.1103/PhysRevLett.118.258101

[16] C. Gallicchio, A. Micheli, L. Pedrelli, Deep reservoir computing: A critical experimental analysis, Neurocomputing 268 (2017) 87–99. doi: 10.1016/j.neucom.2016.12.089. URL https://doi.org/10.1016/j.neucom.2016.12.089

[17] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King,
C. Summerfield, P. Blunsom, K. Kavukcuoglu, D. Hassabis, Hybrid computing using a neural network with dynamic external memory, Nature 538 (2016). doi:10.1038/nature20101.

[18] A. Gulli, S. Pal, Deep learning with Keras, Packt Publishing, Mumbai, c2017.

[19] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015). doi:https://doi.org/10.1038/nature14539.

[20] N. M. Nakamura Y., Approximation capability of continuous time recurrent neural networks for non-autonomous dynamical systems, in: Alippi C., Polycarpou M., Panayiotou C., Ellinas G. (eds) Artificial Neural Networks – ICANN 2009. Lecture Notes in Computer Science, vol 5769. Springer, Berlin, Heidelberg, 2009. doi:https://doi.org/10.1007/978-3-642-04277-5_60.

[21] R. N. M. Kimura, Learning dynamical systems from trajectories by continuous time recurrent neural networks, in: Proceedings of ICNN’95 - International Conference on Neural Networks, 1995. doi:DOI:10.1109/ICNN.1995.487258.

[22] J. F. J.C. Gallacher, Continuous time recurrent neural networks: a paradigm for evolvable analog controller circuits, in: Proceedings of the IEEE 2000 National Aerospace and Electronics Conference. NAECON 2000. Engineering Tomorrow (Cat. No.00CH37093), 2000. doi:DOI: 10.1109/NAECON.2000.894924.

[23] T. W. S. Chow, X.-D. Li, Modeling of continuous time dynamical systems with input by recurrent neural networks, Ieee transactions on circuits and systems—i: fundamental theory and applications 47 (4) (2000). doi:DOI:10.1109/81.841860.

[24] D. Sussillo, Neural circuits as computational dynamical systems, Current Opinion in Neurobiology 25 (2014) 156 – 163, theoretical and computational neuroscience. doi:https://doi.org/10.1016/j.conb.2014.01.008.

[25] S. Vyas, M. D. Golub, D. Sussillo, K. V. Shenoy, Computation through neural population dynamics, Annual Review of Neuroscience
[26] H. Richard, S. Rahul, M. Misha, A. Douglas, R. J. Seung, H. Sebastian, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit, Nature 405 (2000). doi:10.1038/35016072.

[27] V. Mante, D. Sussillo, K. V. Shenoy, W. T. Newsome, Context-dependent computation by recurrent dynamics in prefrontal cortex, Nature doi: 10.1038/nature12742.

[28] F. Carnevale, V. de Lafuente, R. Romo, O. Barak, N. Parga, Dynamic control of response criterion in premotor cortex during perceptual detection under temporal uncertainty, Neuron 86 (2015). doi: 10.1016/j.neuron.2015.04.014.

[29] M. van Gerven, Computational foundations of natural intelligence, Frontiers in Computational Neuroscience 11 (2017) 112. doi:10.3389/fncom.2017.00112. URL https://www.frontiersin.org/article/10.3389/fncom.2017.00112

[30] J. Wang, D. H. Narain, E. A, M. Jazayeri, Flexible timing by temporal scaling of cortical responses, Nature Neuroscience 21 (2018). doi:10.1038/s41593-017-0028-6.

[31] K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, J. J. DiCarlo, Evidence that recurrent circuits are critical to the ventral stream’s execution of core object recognition behavior, Nature neuroscience 22 (6) (2019) 974—983. doi:10.1038/s41593-019-0392-5. URL https://doi.org/10.1038/s41593-019-0392-5

[32] E. D. Remington, S. W. Egger, D. Narain, J. Wang, M. Jazayeri, A dynamical systems perspective on flexible motor timing, Trends in Cognitive Sciences (2018). doi:doi:10.1016/j.tics.2018.07.010.

[33] D. Sussillo, O. Barak, Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks, Neural Computation 25 (3) (2013) 626–649. doi:10.1162/NECO_a_00409.
[34] M. Siegel, T. J. Buschman, E. K. Miller, Cortical information flow during flexible sensorimotor decisions, Nature Reviews Neuroscience 16 (2015). doi:10.1126/science.aab0551.

[35] O. Barak, Recurrent neural networks as versatile tools of neuroscience research, Current Opinion in Neurobiology 46 (2017) 1 – 6, computational Neuroscience. doi:https://doi.org/10.1016/j.conb.2017.06.003. URL http://www.sciencedirect.com/science/article/pii/S0959438817300429

[36] T. Gisiger, M. Boukadoum, Mechanisms gating the flow of information in the cortex: What they might look like and what their uses may be, Frontiers in Computational Neuroscience 5 (2011) 1. doi:10.3389/fncom.2011.00001. URL https://www.frontiersin.org/article/10.3389/fncom.2011.00001

[37] M. Jazayeri, M. N. Shadlen, Temporal context calibrates interval timing, Nature Neuroscience 13 (8) (2010) 1020–1026. doi:10.1038/nn.2590. URL https://doi.org/10.1038/nn.2590

[38] G. R. Yang, M. R. Joglekar, H. F. Song, W. T. Newsome, X.-J. Wang, Task representations in neural networks trained to perform many cognitive tasks, Nature Neuroscience 22 (2) (2019) 297–306. doi:10.1038/s41593-018-0310-2. URL https://doi.org/10.1038/s41593-018-0310-2

[39] G. Bondanelli, S. Ostojic, Coding with transient trajectories in recurrent neural networks, PLOS Computational Biology 16 (2) (2020) 1–36. doi:10.1371/journal.pcbi.1007655. URL https://doi.org/10.1371/journal.pcbi.1007655

[40] A. Ingrosso, L. F. Abbott, Training dynamically balanced excitatory-inhibitory networks, PLOS ONE 14 (8) (2019) 1–18. doi:10.1371/journal.pone.0220547. URL https://doi.org/10.1371/journal.pone.0220547

[41] Z. Bi, C. Zhou, Understanding the computation of time using neural network models, Proceedings of the National Academy of Sciences 117 (19) (2020) 10530–10540. arXiv:https://www.pnas.org/content/117/19/37
[42] L. Jin, M. M. Gupta, P. N. Nikiforuk, Universal approximation using dynamic recurrent neural networks: discrete-time version, in: Proceedings of ICNN’95 - International Conference on Neural Networks, Vol. 1, 1995, pp. 403–408 vol.1. doi:10.1109/ICNN.1995.488134.

[43] F. Chollet, et al., Keras, https://keras.io (2015).

[44] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, S. Valaee, Recent advances in recurrent neural networks, CoRR abs/1801.01078 (2018). arXiv:1801.01078. URL http://arxiv.org/abs/1801.01078

[45] W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural Computation 14 (11) (2002) 2531–2560. arXiv:https://doi.org/10.1162/089976602760407955. doi:10.1162/089976602760407955. URL https://doi.org/10.1162/089976602760407955

[46] D. Sussillo, L. Abbott, Generating coherent patterns of activity from chaotic neural networks, Neuron 63 (2009). doi:doi:10.1016/j.neuron.2009.07.018.

[47] R. Laje, D. V. Buonomano, Robust timing and motor patterns by taming chaos in recurrent neural networks, Nature Neuroscience 16 (2013) 925–933. doi:10.1038/nn.3405. URL https://doi.org/10.1038/nn.3405

[48] B. DePasquale, C. J. Cuevas, K. Rajan, G. S. Escola, L. F. Abbott, full-force: A target-based method for training recurrent networks, PLOS ONE 13 (2) (2018) 1–18. doi:10.1371/journal.pone.0191527. URL https://doi.org/10.1371/journal.pone.0191527

[49] A. P. Trischler, G. M. D’Eleuterio, Synthesis of recurrent neural networks for dynamical system simulation, Neural Networks 80 (2016) 67 – 78. doi:https://doi.org/10.1016/j.neunet.2016.04.001.
[50] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2014). \texttt{arXiv:1412.6980}.
URL http://arxiv.org/abs/1412.6980

[51] A. A. Russo, S. R. Bittner, S. M. Perkins, J. S. Seely, B. M. London, A. H. Lara, A. Miri, N. J. Marshall, A. Kohn, T. M. Jessell, L. F. Abbott, J. P. Cunningham, M. M. Churchland, Motor cortex embeds muscle-like commands in an untangled population response, Neuron 97 (2018). doi:doi:10.1016/j.neuron.2018.01.004.

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[53] A. M. Saxe, J. L. McClelland, S. Ganguli, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, CoRR abs/1312.6120 (2013). \texttt{arXiv:1312.6120}.
URL http://arxiv.org/abs/1312.6120

[54] A. Goel, D. V. Buonomano, Timing as an intrinsic property of neural networks: evidence from in vivo and in vitro experiments, Philosophical Transactions of the Royal Society of London B: Biological Sciences 369 (1637) (2014). \texttt{arXiv:http://rstb.royalsocietypublishing.org/content/369/1637/20120460.full.pdf}, doi:10.1098/rstb.2012.0460.
URL http://rstb.royalsocietypublishing.org/content/369/1637/20120460

[55] R. Rojas, Springer-Verlag, Berlin, New-York, 1996. [link].
URL https://page.mi.fu-berlin.de/rojas/neural/

[56] P. Holla, S. Chakravarthy, Decision making with long delays using networks of flip-flop neurons, in: 2016 International Joint Conference on
Neural Networks (IJCNN), 2016, pp. 2767–2773. doi:10.1109/IJCNN.2016.7727548.

[57] J. P. Cunningham, B. M. Yu, Dimensionality reduction for large-scale neural recordings, Nature Neuroscience 17 (2014). doi:http://dx.doi.org/10.1038/nn.3776.

[58] E. Balaguer-Ballester, C. C. Lapish, J. K. Seamans, D. Durstewitz, Attracting dynamics of frontal cortex ensembles during memory-guided decision-making, PLOS Computational Biology 7 (5) (2011) 1–19. doi:10.1371/journal.pcbi.1002057. URL https://doi.org/10.1371/journal.pcbi.1002057

[59] C. Thompson, L. Shure, Image Processing Toolbox: For Use with MATLAB;[user’s Guide], MathWorks, 1995.

[60] Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints abs/1605.02688 (May 2016). URL http://arxiv.org/abs/1605.02688

[61] S. Kuroki, T. Isomura, Task-related synaptic changes localized to small neuronal population in recurrent neural network cortical models, Frontiers in Computational Neuroscience 12 (2018) 83. doi:10.3389/fncom.2018.00083. URL https://www.frontiersin.org/article/10.3389/fncom.2018.00083

[62] A. H. Williams, F. Kim, Tony Hyun ans Wang, S. Vyas, S. I. Ryu, K. V. Shenoy, M. Schnitzer, T. G. Kolda, S. Ganguli, Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis, Neuron 98 (2018). doi:10.1016/j.neuron.2018.05.015.

[63] V. Girko, Circular law, Theory of Probability & Its Applications 29 (4) (1985) 694–706. arXiv:https://doi.org/10.1137/1129095, doi:10.1137/1129095. URL https://doi.org/10.1137/1129095

[64] I. D. Landau, H. Sompolinsky, Coherent chaos in a recurrent neural network with structured connectivity, PLOS Computational Biology
[65] L. C. García del Molino, K. Pakdaman, J. Touboul, G. Wainrib, Synchronization in random balanced networks, Phys. Rev. E 88 (2013) 042824. doi:10.1103/PhysRevE.88.042824. URL https://link.aps.org/doi/10.1103/PhysRevE.88.042824

[66] E. Vorontsov, C. Trabelsi, S. Kadoury, C. J. Pal, On orthogonality and learning recurrent networks with long term dependencies, in: ICML, 2017.

[67] Q. Zhou, T. Jin, H. Zhao, Correlation between eigenvalue spectra and dynamics of neural networks, Neural Computation 21 (10) (2009) 2931–2941, pMID: 19635013. arXiv:https://doi.org/10.1162/neco.2009.12-07-671, doi:10.1162/neco.2009.12-07-671. URL https://doi.org/10.1162/neco.2009.12-07-671

[68] N. Maheswaranathan, A. H. Williams, M. D. Golub, S. Ganguli, D. Susillo, Universality and individuality in neural dynamics across large populations of recurrent networks (2019). arXiv:1907.08549.

[69] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation (2014). arXiv:1406.1078.

[70] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735