UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Corso di Laurea in Fisica

Tesi di Laurea

Attrattori dinamici nelle reti neurali ricorrenti

Attractors of recurrent neural networks

Relatore

Dott. Michele Allegra

Laureanda

Ginevra Beltrame

Anno accademico 2022/2023

Abstract

In recent neuroscientific literature, task representation has received a lot of attention. A major open question is how the architecture of clusters of neurons relates to the cognitive processes they execute and to the neurons' functional specialization. Artificial neural networks are a powerful, widely employed tool to model and therefore comprehend the activity of these neural clusters. By training artificial networks to perform the same tasks of biological ones and looking at their structure and dynamics, significant insight can be gained into task representation by biological brains. This thesis focuses specifically on recurrent neural networks (RNNs), characterized by connections between layers which are shared over time and trained with the backpropagation-through-time method. The RNN at hand is trained to perform a simple cognitive task ("Go"), and the resulting response is analyzed both by examining the input and output signals, as well as the network parameters, and through the lens of a dynamical systems approach.

Contents

Chapter 1

Introduction

The understanding of how neural activity is related to behavior is a key, highly challenging endeavour of modern-day brain studies. Clusters of neurons first encode incoming sensory stimuli in terms of electrical activity. They later transform this information into perceptual decision or motor action by weighing evidence for choice alternatives, thus implementing computations that generate behavior (Wang 2002). To reach a simple, mechanistic grasp of such processes is therefore a central research goal in neuroscience.

A powerful way to address this problem is to rely on simplified, computational models of neural networks to provide a testing ground for theoretical hypotheses on cognitive task performance. Moreover, mathematical, physical and information theory techniques can be employed to examine artificial brain-like networks and elucidate the underlying neural mechanisms (Mastrogiuseppe et al. 2018, Yang et al. 2019).

One promising approach in this direction is to analyze "vanilla" recurrent neural networks (RNNs) that have been optimized to perform the same tasks as behaving animals by means of the backpropagation-through-time (BPTT) training algorithm (Yang et al. 2020, Song et al. 2016, Werbos 1990). Among artificial neural network architectures, only RNNs can adequately represent time, as they are provided with a dynamic memory. Including this feature in neural models is particularly important as time is inextricably bounded with many behaviors (such as natural language understanding) which unfold as sequential actions (Elman 1990, LeCun et al. 2015). In the context of handling large data sets, e.g. long time series, the effectiveness of the machine learning paradigm resides in not needing to explicitly instruct the networks on how to infer predictions from the given input. Employing automatic learning algorithms though results in a significant drawback: as RNNs are intrinsically constructed as "black boxes", the exact nature of how neural computation mechanisms are implemented is not easily recognizable (Sussillo et al. 2013).

The opening of the aforementioned "black boxes" is aided by multiple analysis techniques, among which is viewing the RNN as a nonlinear dynamical system. Specifically, the equation describing network dynamics can be obtained and studied both with computational and classical mathematical tools (Sussillo et al. 2013, Golub et al. 2018). Moreover, an RNN trained to perform several tasks can also be examined from the point of view of task representation, that is the way the network encodes the cognitive processes which transform the sensory input signal into a motor output signal. It was hypothesized (Yang et al. 2019) that such representations strongly depend on the motion and spatial distribution of high-dimensional neural activity vectors, visualized in an appropriate state space as a dynamical landscape.

The aim of this thesis is to simulate the dynamics of an RNN trained to perform a simple cognitive task (indicated as “Go”) and to investigate the claim made by Yang et al.

In Chapter 2, a more detailed overview of the role played in neuroscientific studies by artificial, and specifically recurrent, neural networks is presented along with a general mathematical description of RNN structure. A thorough presentation of the backpropagation training algorithm is included, both for its classical version and for the BPTT variant.

Chapter 3 instead illustrates the architecture of the specific RNN which is the object of the thesis. The input and output signal encoding is also described, along with Go and other related tasks.

The results of the computational implementation of RNN dynamics, are presented in Chapter 4. The obtained output is compared to the input and to the expected “target” output signal. This Chapter also reports the connectivity parameter matrix resulting from training, as well as a qualitative description of neural activity.

Lastly, task representation and activity dynamics are investigated through dynamical systems analysis techniques in Chapter 5. In this context, the notion of task vector and the *compositionality* property are also presented.

Chapter 2

General use of RNNs

2.1 ANNs and RNNs in neuroscientific research

From the early 2010s on, there has been a significant rise in interest regarding machine learning methods, such as artificial neural networks (ANNs) and deep learning, paralleled by an explosive development of techniques and applications. A major field in which artificial intelligence finds application and sources of inspiration is neuroscience, with a continuous flow of ideas between the two disciplines, and cross-validation of key hypotheses about cognitive processes. Nowadays machine learning algorithms are indeed one of the main computational tools to study brain circuits and other similar biological systems: according to Yang and Wang 2020 they can be successfully employed in the analysis of neuroscientific data, providing efficient tracking and processing of behavioural and neurophysiological patterns.

However, while applying generic artificial intelligence methods in the analysis of neuroscientific problems can offer relevant interpretations of existing biological processes, it often fails to provide simplified models of such processes, that can be easily modified and adapted to different real-life situations (i.e. by the introduction of appropriate physical or mathematical constraints) without the need to conduct experiments *in vivo*.

The required modelling capacity is instead allowed by the employment of ANNs specifically designed to reproduce neural circuit structures, as in Figure 2.1. Biological justification and adherence to available data is not the primary concern when developing these networks, which are rather aimed at offering a wide range of fresh, diverse approaches to meet new challenges in this rapidly evolving field of study.

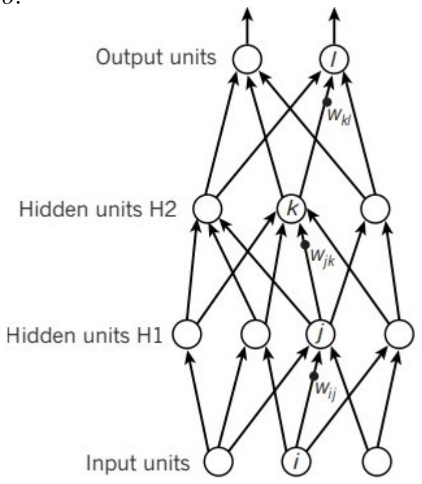


Figure 2.1: Example of a simple 3-layer (or 2-hidden-layer) neural network, from LeCun et al. 2015. Hidden layers interact exclusively with other internal nodes and not with the outside. Each connection is characterized by the weight w_{ij}.

As living organisms learn from and adapt to their surroundings with a series of trial-and-error steps, so must artificial brains behave with the aid of "deep learning" methods. This term specifically refers to a class of representation learning methods, which allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification (LeCun et al. 2015). Using "deep", multilayered architectures enables the acquisition of functions which can be very complex, thanks to the fact that each layer implements a different, higher level of abstraction with respect to the original data. Moreover, the functions themselves are learned directly from the input-output mapping, as no explicit instruction is given to the machine. In fact, this procedure only relies on a general-purpose sort of training algorithm, typically backpropagation which will be addressed in depth in Section 2.2.

Historically the backpropagation technique was first employed to train a specific type of ANNs, the recurrent neural networks, or RNNs. The key feature of these models is that they process information in time by feeding hidden unit patterns back to themselves (Figure 2.2). One of the first thorough descriptions of this kind of networks in the literature is found in Elman 1990, where RNNs are presented as an efficient answer to the problem of time representation in connectionist brain models. As time clearly emerged as a fundamental aspect of cognition, the article proposed to weave it into network-like neural models by the use of recurrent links. The latter endow the networks with dynamic properties that are responsive to temporal sequences, i.e., with a dynamic memory. Each unit of the network can be interpreted as the firing rate of a single neuron or the spatial average of a group of similarly tuned neurons.

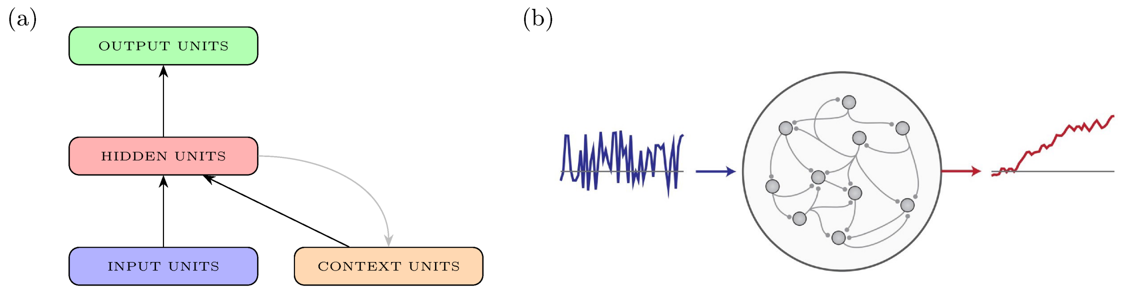


Figure 2.2: (a) Historical example of an Elman RNN structure showing recurrent connections between hidden layers, adapted from Elman 1990. (b) An RNN receives an input stream (temporal series), which encodes task-relevant sensory information or internal rules, and converts it to output, which instead indicates a decision in the form of an abstract decision variable, probability distribution, or direct motor signal (Song et al. 2016).

The starting point of Elman's discussion is the endeavour to avoid a spatial representation of time in the pattern vector of the dynamical system, which could be provided with an additional, explicit variable denoting the serial order of the pattern itself. An implicit, internal representation is instead to be preferred, as networks built in this way are capable of incorporating task processing with a dependence from memorized prior internal states. This is achieved precisely with recurrent connections that link hidden units to their own previous output, stored in "context" units (as in Figure 2.2a).

From a mathematical viewpoint, RNNs can be described in terms of their n-dimensional neural activity vector.

$$\mathbf{r}_t = f(\mathbf{c}_t) = f(W^{rec}\mathbf{r}_{t-1} + W^{in}\mathbf{u}_t + \mathbf{b}_{rec}) \qquad (2.1)$$

This equation, where an appropriate time discretization is employed, shows that $\mathbf{r}_t$ is affected by the activity of the preceding time step $\mathbf{r}_{t-1}$. The latter can be though of as input coming from the context units of the network, which are linked to the hidden units through a recurrent $n \times n$ connection matrix W^{rec} and keep in memory the activity at the preceding time step. As in Figure 2.2a, input $\mathbf{u}_t$ also acts on the units of the hidden layer through the weights contained in W^{in}. Furthermore, f is the activation function of the model neurons, the argument $\mathbf{c}_t$ of which represents the *cell state*, analogous to membrane potential or input current (Yang and Wang 2020). Such curve is usually nonlinear and typically takes the form of a Rectified Linear Unit (ReLU, $f(x) = max(x, 0)$), a hyperbolic tangent (tanh) or variants of these two. Lastly, $\mathbf{b}_{rec}$ is the bias or background input term, that introduces an n-dimensional shift degree of freedom to the learning algorithm (Yang et al. 2019).

The output of the network is instead

$$\mathbf{z}_t = g(W^{out}\mathbf{r}_t + \mathbf{b}_{out}) \tag{2.2}$$

This equation introduces a third connection matrix W^{out} and an additional activation function g, which strongly depends on the expected output. A typical choice is the sigmoid logistic function $g(x) = \frac{1}{1+e^{-x}}$ that returns values between 0 and 1 and is therefore suitable for a digital readout. This term is also provided with a bias $\mathbf{b}_{out}$.

2.2 RNN learning method: backpropagation through time

In order to efficiently predict the desired output, ANNs must learn from data itself the parameters of the network, commonly denoted collectively as a vector $\boldsymbol{\theta}$. In the case of Equations 2.1 and 2.2 these are the entries of the three connection matrices as well as the biases. The standard way to do this is by implementing a supervised learning protocol such as the Stochastic Gradient Descent (SGD), paired with the employment of the *backpropagation* technique, an effective and exact method for calculating all the derivatives of a single target quantity (in this case, the pattern classification error) with respect to a large set of parameters (Werbos 1990).

2.2.1 The classic backpropagation algorithm

The most basic example of a multi-layer ANN is the multi-layer perceptron (MLP), a feedforward system that doesn't involve processing information in time (Figure 2.3).

The following equations describe a generic N-layer MLP

$$\begin{aligned} \mathbf{r}_{(1)} &= \mathbf{u} \\ \mathbf{r}_{(l)} &= f(\mathbf{W}^{(l)}\mathbf{r}_{(l-1)} + \mathbf{b}_{(l)}), \quad 1 < l < N \\ \mathbf{z} &= g(\mathbf{W}^{(N)}\mathbf{r}_{(N-1)} + \mathbf{b}_{(N)}) \end{aligned}$$

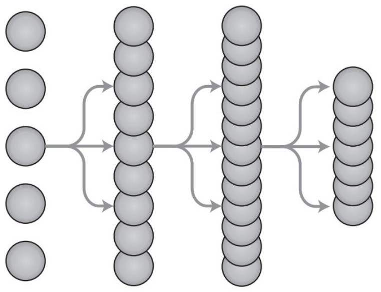


Figure 2.3: Example of an MLP from Yang et al. 2020.

where the same symbols as Equations 2.1 and 2.2 are used and the time variable is substituted with the layer counter l. $W^{(l)}$ is then the connection matrix between the $(l-1)$-th and the l-th layer (Yang and Wang 2020). To this the classic version of the backpropagation algorithm is typically applied, the understanding of which is crucial to comprehend how recurrent neural networks are trained.

The first step is to compute the loss (or cost) function of the network. This is defined as the sum of distances d (in whichever metric of the output vector space is most convenient) between the obtained and desired outputs, with respect to the training data examples. Because it is usually very expensive to evaluate the entire training set, a *minibatch* $\mathcal{B}$ (a small, randomly created subset with M items) is used for the summation instead.

$$L(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i\in\mathcal{B}} d(\mathbf{z}_i(\boldsymbol{\theta}), \mathbf{z}_i^{train}) \tag{2.3}$$

To calculate such a function, a multilayered network needs to be “run in a forward pass” (Yang and Wang 2020), that is the output has to be computed with the existing parameters. These are typically initialized at random and are possibly subjected to appropriate, context-dependent mathematical constraints.

The random weights are then updated to minimize the (non-negative) “error” L: the gradient of the loss function $\frac{\partial L}{\partial \boldsymbol{\theta}}$ is therefore estimated. Because it represents the direction in parameter space in which a small (approximately infinitesimal) parameter change leads to the maximum increase in L itself, the algorithm slightly shifts the vector $\boldsymbol{\theta}$ in the opposite direction

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \eta \frac{\partial L}{\partial \boldsymbol{\theta}} \tag{2.4}$$

where $\eta \ll 1$ is the learning rate.

A key phase of the SGD is then to efficiently compute this gradient, which means computing as many partial derivatives of the loss function as the dimension of $\boldsymbol{\theta}$, equal to the combined number of weights of the connection matrices plus the entries of the bias vector. For multilayered ANNs this can be achieved through the *backpropagation method*, which starts by calculating the derivatives of L with respect to the output layer. This task is a relatively simple one, as the vector $\mathbf{z}_t$ is already known thanks to the forward pass.

Moving to the generic l-th layer, a calculation procedure can be derived

$$\frac{\partial L}{\partial \mathbf{r}_{(l)}} = [W^{(l+1)}]^T \frac{\partial L}{\partial \mathbf{r}_{(l+1)}} \tag{2.5}$$

where $[W^{(l+1)}]^T$ is the transpose of the connection matrix between the l-th and the $(l+1)$-th layer. It is therefore possible to recursively compute each gradient, starting from $\frac{\partial L}{\partial \mathbf{z}}$ and moving backwards across the network. Equation 2.5 is valid under the simplifying assumption that all activation functions $f(x) = g(x) = x$ are linear: otherwise, an analogous equation including the partial derivatives of f should be computed at each iteration. For the bias term, an similar equation can be derived.

Once the $\frac{\partial L}{\partial \mathbf{r}_{(l)}}$ are known, it is finally possible to determine the derivatives with respect to the weights by applying the chain rule.

$$\frac{\partial L}{\partial W^{(l+1)}} = \frac{\partial L}{\partial \mathbf{r}_{(l+1)}} \mathbf{r}_{(l)}^T \tag{2.6}$$

After the appropriate update of the weights, the loss function is diminished. The entire algorithm is run multiple times until a (relative) minimum of L is reached. At this point the network is considered trained: it is possible to evaluate its performance in different tasks and to study its dynamical response to any input signal.

2.2.2 Backpropagation through time (BPTT)

With respect to RNNs, the backpropagation algorithm can still be applied, with a significant modification that takes into account the dependency of their dynamics on previous states. Indeed, computing the gradient for an RNN involves propagating information backwards in time, as it can be “unrolled” and viewed as an MLP where each layer actually corresponds to a different time step (Figure 2.4).

In the forward pass phase, the loss function is calculated from the outputs at the last time point t_{last} and a linear activation function is again assumed for simplicity.

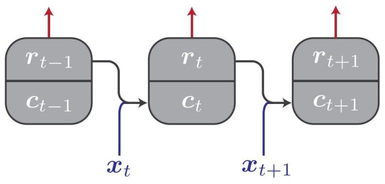


Figure 2.4: Example of an RNN unrolled in time as a feedforward system with each layer corresponding to the network state at one time step. Figure from Yang and Wang 2020, where $\mathbf{x}_t$ is the input and $\mathbf{c}_t$ is the cell state.

To estimate the gradients at time t with respect to the entries of W^{out}, that are comprised in $\boldsymbol{\theta}$, the chain rule is applied on time steps, resulting in the backpropagating equation

$$\frac{\partial L}{\partial W^{out}} = \sum_{t=1}^{t_{last}} \frac{\partial L}{\partial \mathbf{z}_t} \mathbf{r}_t^T \tag{2.7}$$

Considering now the recurrent connection matrix W^{rec}, an iterative computation similar to that of Equation 2.5 can be implemented (Yang and Wang 2020)

$$\frac{\partial L}{\partial \mathbf{r}_t} = [W^{rec}]^T \frac{\partial L}{\partial \mathbf{r}_{t+1}} = [W^{rec}]^2 \frac{\partial L}{\partial \mathbf{r}_{t+2}} = ... \tag{2.8}$$

from which the derivative $\frac{\partial L}{\partial W^{rec}}$ is estimated as in Equation 2.7. It must however be taken into account that the hidden state at time step $t+1$ is dependent on the hidden state at time step t, a second temporal summation is required within the first (Werbos 1990). $\frac{\partial L}{\partial W^{in}}$ is also then estimated in the same way.

Backpropagation through time is a very powerful tool, but not devoid of criticalities. Specifically, particular care should be employed when dealing with large matrix products (as in subsequent steps of the Equation 2.8), as they can either explode or vanish if the involved eigenvalues are respectively big or very small. Modern techniques that tend to preserve the norm of the backpropagated gradients (Yang and Wang 2020) can be exploited to alleviate these problems.

Chapter 3

Network architecture and tasks

3.1 Architecture of the RNN

This thesis studies a trained, given RNN which replicates the ones employed in Yang et al. 2019. Before time discretization, the activity vector $\mathbf{r}$ follows the continuous dynamical equation

$$\tau \frac{d\mathbf{r}}{dt} = -\mathbf{r} + f(W^{rec}\mathbf{r} + W^{in}\mathbf{u} + \mathbf{b}_{rec} + \sqrt{2\tau\sigma_{rec}^2}\,\boldsymbol{\xi}) \tag{3.1}$$

where the number of hidden units, corresponding to the dimension of $\mathbf{r}$, is $N^{rec} = 256$. The input layer, modelled by the vector $\mathbf{u}$, is instead comprised of $N^{in} = 85$ units.

In the equation above, $\tau = 100$ms is the neuronal time constant, slightly longer than the typical 20ms of real nerve cells. This choice was originally made to mimic the slower synaptic dynamics on the basis of N-methyl-D-aspartate (NMDA) receptors, crucial for activity-dependent synaptic plasticity, which in turn underpins many higher functions including learning and memory (Wang 2002, Furukawa et al. 2005).

Regarding the utilized symbols, the neuronal nonlinearity is a re-parameterized standard softplus activation function

$$f(x) = log(1 + e^x) \tag{3.2}$$

the non negativity and non-saturation properties of which are crucial to effectively mimic the corresponding biological system. The vector $\boldsymbol{\xi}$ contains N^{rec} independent Gaussian white noise processes with zero mean and unit variance, while $\sigma_{rec} = 0.05$ is the strength of the noise. As for the $N^{out} = 33$ output units, they read out nonlinearly from the network

$$\mathbf{z} = g(W^{out}\mathbf{r} + \mathbf{b}_{out}) \tag{3.3}$$

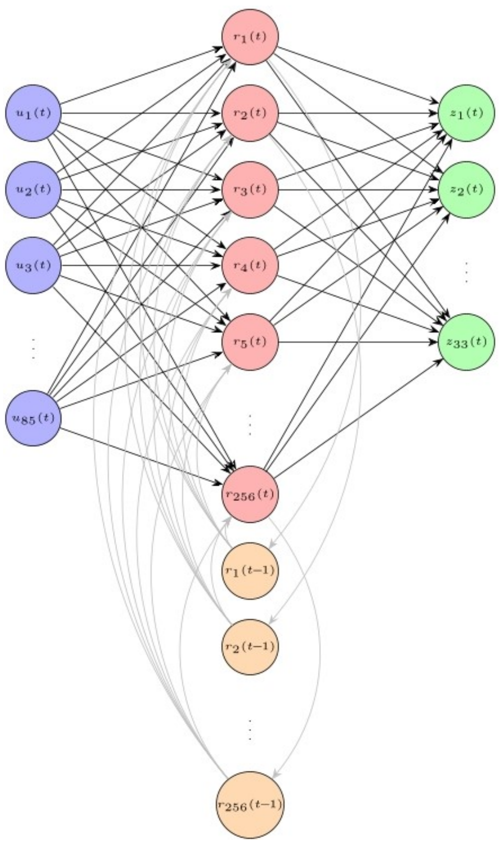


Figure 3.1: Graph representing the described RNN. Recurrent connections are shown in grey.

where $g(x) = \frac{1}{1+e^{-x}}$ is the logistic function, bounding output activities between 0 and 1. No constraint on the sign or the structure of the weight matrices is imposed.

After using a first-order Euler approximation with a time-discretization step $\Delta t = 20$ms, Equation 3.1 becomes

$$\mathbf{r}_t = (1-\alpha)\mathbf{r}_{t-1} + \alpha f(W^{rec}\mathbf{r}_{t-1} + W^{in}\mathbf{u}_t + \mathbf{b}_{rec} + \sqrt{2\alpha^{-1}\sigma_{rec}^2}\,\mathbf{N}(0,1)) \tag{3.4}$$

where $\alpha = \frac{\Delta t}{\tau} = 0.2$ and $\mathbf{N}(0,1)$ stands for the standard normal distribution.

3.2 The input and output signals

This RNN architecture is a toy mathematical model of complex neural processes, the input signal of which represents the combination of sensory stimuli and task-dependent cues. The latter are internal rules that instruct the hidden decision layer, composing the "heart" of the network, to perform 20 different cognitive tasks, some of which will be described in more detail in the following paragraphs. The output indicates the decision resulting from recurrent computations in the form of a motor signal, e.g. an eye saccade[1].

For each task the network learns, it receives four types of input signals: fixation, two stimuli and rule

$$\mathbf{u} = (u_{fix}, \mathbf{u}_{mod1}, \mathbf{u}_{mod2}, \mathbf{u}_{rule}) + \mathbf{u}_{noise} \tag{3.5}$$

supplied with random Gaussian noise $\mathbf{u}_{noise} = \sqrt{\frac{2}{\alpha}\sigma_{in}^2}\,\mathbf{N}(0,1)$, with $\sigma_{in} = 0.01$.

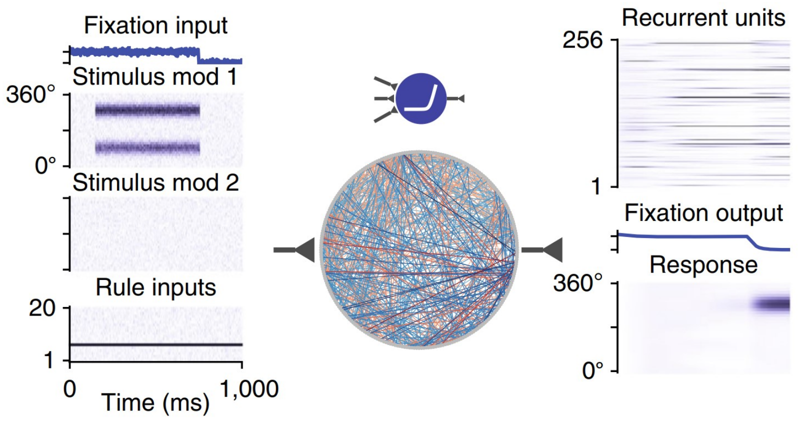


Figure 3.2: Example of a fully connected, 256-recurrent units RNN (middle, 1% of connections shown, with the softplus activation function depicted above) from Yang et al. 2019. The activities of input, recurrent and output neurons are shown for a 1s trial time series. The fixation is depicted as a continuous line, with activity on the vertical axis, while the time-discretized 32-dim ring units are cells the colour of which becomes darker as the correspondent activity increases.

The 1-dimensional fixation input indicates whether the network state should be fixed ($u_{fix} = 1$) or it should respond ($u_{fix} = 0$). Indeed, the decrease in this input provides a "go signal" to the RNN (Yang et al. 2019). The stimulus input consists instead in two modalities $\mathbf{u}_{mod1}$ and $\mathbf{u}_{mod2}$, each represented by a "ring" of units that encodes a one-dimensional angular variable. The rings contain 32 units i each, whose preferred directions θ_i are uniformly spaced from 0 to 2π, and whose neural activities u_i are calculated as

$$u_i(\psi) = 0.8\gamma\ e^{-\frac{1}{2}\left(\frac{|\psi-\theta_i|}{\pi/8}\right)^2} \tag{3.6}$$

[1]Saccades are a class of eye movements that involve a rapid shift of gaze generated to a location of interest and allow individuals to explore the visual environment (from Pierce, Clementz, and McDowell 2019). Correspondingly, the RNN at hand orients the direction of the 32-unit output ring to mimic eye orientation.

for a stimulus presented at direction ψ with strength γ.

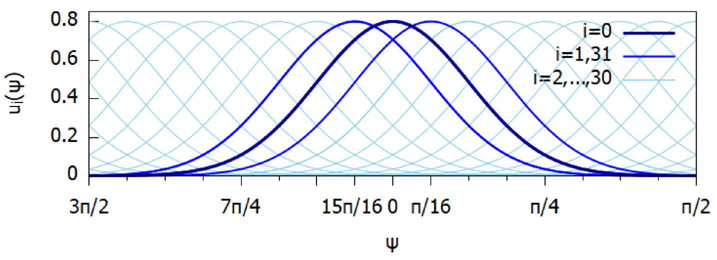


Figure 3.3: Gaussian shape of the functions $u_i(\psi)$. The horizontal axis presents an angular variable, so the coordinates 0 and 2π are equivalent.

It can be observed that this Equation is in the form of a (non-normalized) Gaussian distribution with standard deviation $\sigma = \frac{\pi}{8}$, which amounts to twice the difference between subsequent θ_i. Figure 3.3 highlights that the shape of the u_i distributions mimics that of the *receptive field*[2], where each unit responds in an allowed direction which can differ slightly from the given ψ.

Lastly, a 20-dimensional rule input one-hot vector $\mathbf{u}_{rule}$ encodes which one of the available tasks the network is supposed to perform in each trial. The unit corresponding to the current task is activated at 1, while the others remain at 0, making all different rule unit activation patterns orthogonal to one another. Each of the $N^{in}=85$ input neurons is thus accounted for.

A similar description is valid for the $N^{out}=33$ output units as well, as the network projects a fixation output unit z_{fix} (the value of which should be 1 before and 0 after the response) and a 32-unit output ring $\mathbf{z}_{ring}$ that encodes the response directions using similar tuning curves to the ones of $\mathbf{u}_{mod1}$ and $\mathbf{u}_{mod2}$ (Yang et al. 2019).

The time trials corresponding to each task can be divided into four periods, that is segments of sequential time steps with constant input signals (except for the noise), as in Figure 3.4. At the beginning, throughout the *context* epoch, $u_{fix} = 1$ as no response is required, there is no stimulus input and the rule input provides the network with information about task context, remaining subsequently activated for the entire trial. This last feature changes as the *stimulus* period starts and one or both of $\mathbf{u}_{mod1}$ and $\mathbf{u}_{mod2}$ is activated. Thirdly, in the *memory* period the stimuli cease (the input is therefore identical to that of the context) and the RNN prepares for the *response* or *go* epoch, during which $u_{fix} = 0$ and the output signal is generated.

As depicted in Figure 3.4, when u_{fix} is on the fixation output also has a high activity ($z_{fix}=0.8$), which subsequently decreases to the target $z_{fix} = 0.05$ in the response period. In fact, during this epoch the network needs to report the response direction $\mathbf{z}_{ring}$ through activities of the output ring: with respect to the target response direction ψ, the expected output for unit i can be calculated as in Equation 3.6

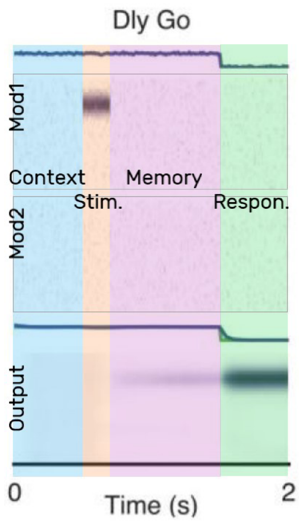


Figure 3.4: Trial periods in a sample "Delayed Go" task from Yang et al. 2019. 65 of the input units (above, without the rule) and the 33 output units (below) are piled vertically.

$$\hat{z}_i(\psi) = 0.8\, e^{-\frac{1}{2}\left(\frac{|\psi-\theta_i|}{\pi/8}\right)^2} + 0.05 \tag{3.7}$$

[2]The receptive field is a region encompassing receptors that feed into sensory neurons, within which stimuli can influence the electrical activity of sensory cells. It provides a description of the location at which a stimulus must be presented in order to elicit a response from a sensory cell (from "Encyclopedia Britannica").

where the additive term +0.05 is the target output activity when no response is required. A trial is considered correct only if fixation is correctly maintained over 0.5 and response is in the correct direction (within 36^o of ψ).

According to Driscoll et al. 2022, during each task period with unique inputs the network could be treated as a separate, autonomous dynamical system with a distinct set of fixed points. This type of analysis will be addressed in Chapter 5.

3.3 The Go task

The given RNN is trained to perform the Go task as presented in Yang et al. 2019, which can be grouped within a "Go task family" characterized by a single stimulus randomly shown in either modality 1 or 2, and a target response in the direction of the given input. Separate tasks within this group differ in the stimulus onset and offset times: in Go (Figure 3.5) it appears before the fixation cue goes off and remains stationary throughout the whole task, while in Dly Go (Figure 3.2) it appears briefly and is followed by a delay (memory) period.

The Go task can therefore be instructed as "saccade to the direction of the stimulus after the fixation cue goes off", while for Dly Go the network is required to remember the direction of the stimulus throughout the memory epoch, then saccade. Another noteworthy task family is "Anti", including the Anti and Dly Anti tasks. These are the same as their Go-family counterparts, except that the response should be made in the opposite direction with respect to the stimulus.

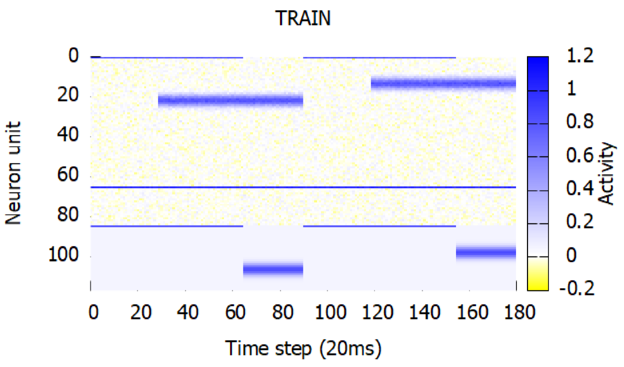


Figure 3.5: Input and output neural activities for the Go task during training, for two time trials. Figure obtained from data made available by Yang et al. 2019 (see Chapter 4).

As regards to the learning of the aforementioned tasks, the network is trained employing a variant of the SGD, Adam, which was first introduced in Kingma and Ba 2015. With this method the value of a parameter update is magnified if the gradient of its cost function has low variance, i.e. has been consistent across steps (Yang and Wang 2020). The loss is computed by time-averaging the squared errors between the target output and the network output $\mathbf{z}$, as

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N^{out}} \sum_{t=1}^{T} m_{i,t} \left(z_{i,t}(\boldsymbol{\theta}) - z_{i,t}^{target}\right)^2 \tag{3.8}$$

where T is the length of each trial and $m_{i,t}$ is a non-negative "mask matrix" that has the function to weight the different squared errors according to the specific time points and output units. In the present case, from the beginning of the trial to the end of the memory period $m_{i,t}$ is 1 for the output ring units, while the first 100ms of the response epoch constitute a "grace period" with $m_{i,t} = 0$. For the rest of the response epoch $m_{i,t} = 5$ and for the fixation output unit, the mask is two times stronger during all epochs. As for the initialization of the weight matrices, at the beginning of the training W^{rec} is a scaled 256×256 identity matrix with entries 0.5 on the diagonal. Moreover, the employed minibatches are composed of 64 trials each.

The following Chapter 4 will specifically focus on how the described RNN responds to an input signal encoding for the Go task.

Chapter 4

RNN output

This Chapter is dedicated the presentation of results concerning the computational implementation of Equation 3.4, with respect to given input time series for training and test trials of the Go task.

Specifically, the training time series is composed of $N_{trials} = 200$ trials, each including $N_{points} = 90$ time points. Since the discretization step is $\Delta t = 20$ms, such trials last 1.8s. On the other hand, the test time series comprises 20 trials with 98 points apiece, therefore lasting 1.96s. As shown below in Figure 4.1, the rule for the Go task is encoded by the first rule neuron (the 66th overall).

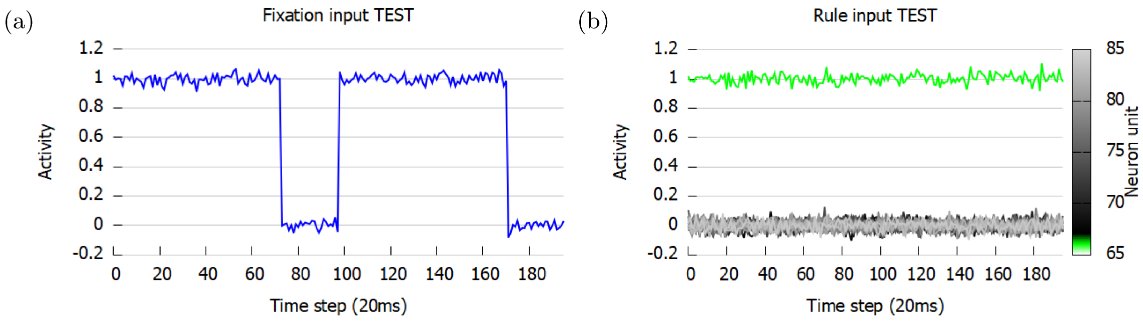


Figure 4.1: Activity of the fixation and rule input units during two trials of the TEST time series, it replicates the features of the Go task described in Section 3.3.

As regards to the network parameters, the three weight matrices and two bias terms (for the recurrent and output units) resulting from the training are provided for this study.

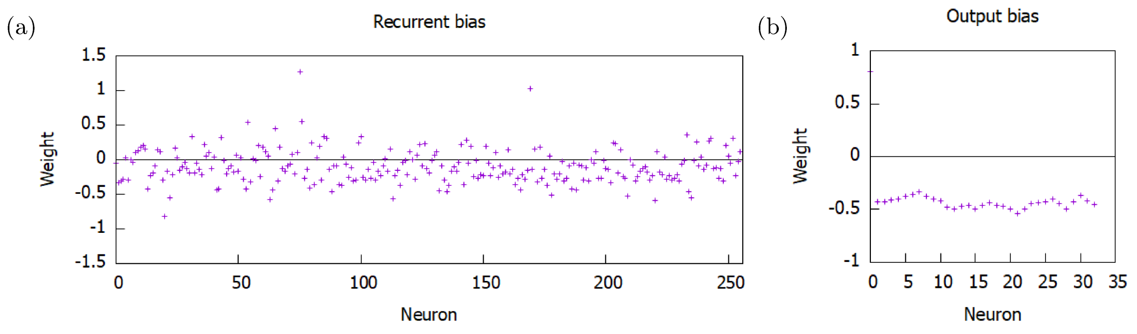


Figure 4.2: Bias terms for the recurrent and output neural units. It can be noted that, while $\mathbf{b}_{rec}$ follows an approximately Gaussian random distribution with mean 0.10 and variance 0.06, the elements of $\mathbf{b}_{out}$ instead are evenly scattered around the value -0.45 (variance 0.002), suggesting that the function learned by the RNN has a corresponding offset.

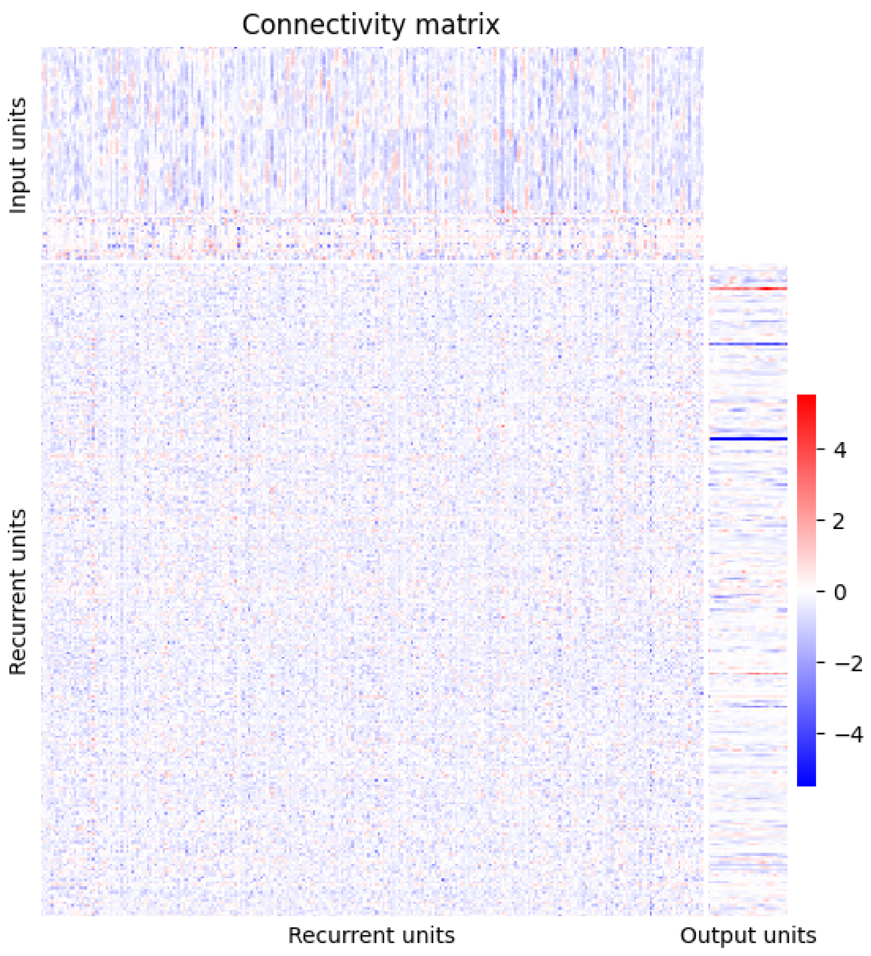


Figure 4.3: Depiction of the full connectivity matrix for the RNN, where the colour indicates the weights w_{ij}.

The previous Figure visualizes the weights of the input, recurrent and output connections, and it can be observed that a significant part of the entries has a negative value (blue). From the point of view of neural connectivity, this translates into *inhibitory* connections which tend to dampen the activity of the target neurons, as opposed to positive, *excitatory* links (red).

Moreover, one red and two blue horizontal lines in the upper part of W^{out} can be observed, indicating that three of the recurrent units are connected to all output ring units (z_{fix} doesn't fit in this pattern) with very strong weights of the same sign. This feature aligns with observations made on biological brains: indeed, it was established (Dale's law, see Song et al. 2016) that neurons in the mammalian cortex have a purely excitatory or inhibitory effect on all other neurons. Conversely, as typically emerges from the training of unconstrained RNNs, the sign varies for the vast majority of the illustrated weights, suggesting that the respective units are unrealistically able to emit connections which boost the activity of some neurons and suppress that of the others. If these model networks were to be made more adherent to biological ones, it would therefore be advisable to impose an appropriate mathematical constraint that bounds the sign of all real-valued weights related to the same recurrent unit.

The activity of recurrent units can also be represented by a color variable in a full-network visualization that reproduces the one in Figure 3.2. The following Figure 4.4 indeed depicts a graph, the nodes of which are the recurrent units of the RNN while the edges depict only the recurrent links with $|w_{ij}| > 1.75$ for a better visualization. It can be observed that, while most units are nearly completely deactivated when averaged across all 200 trials, for some of them activity strongly depends on the task period. As an example, the unit at the bottom right corner of the graphs which is yellow-coloured for the full trial, appears slightly less activated at the beginning and in the middle of the trial and very strongly activated during the response period. By contrast, the yellow-coloured node at the top of the "Beginning" graph is only activated during this task period (see also Figure 4.5a).

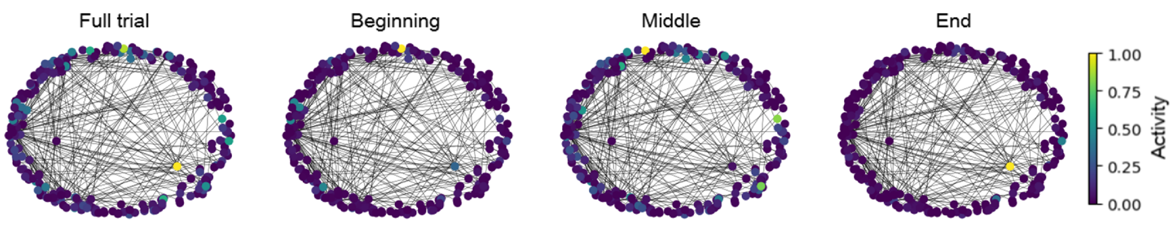
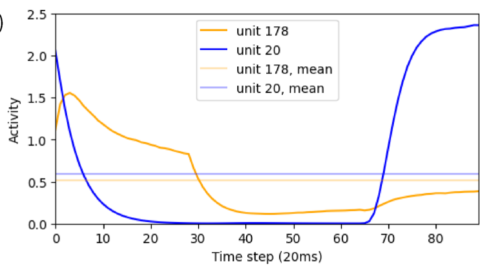


Figure 4.4: Graph depicting the activated recurrent units and all connections with weight greater than 1. Activity is calculated by averaging across all 200 trials of the TRAIN time series, then three ranges of time points at the beginning (0-30), middle (30-70) and end (70-90) of each trial are isolated.

(a)


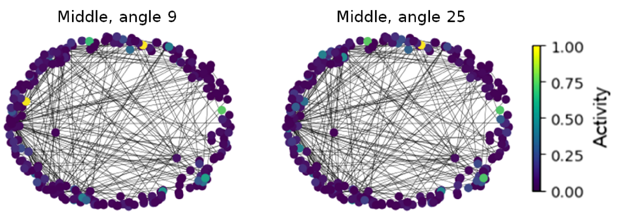
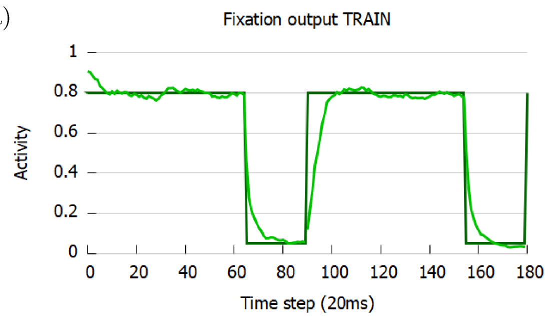
(b)


Figure 4.5: (a) Plot of the activity of the aforementioned example unit, yellow-coloured in the full-trial and end-of trial graphs of Figure 4.4 (blue). Plot of the yellow-coloured unit at the top of the beginning-of-trial graph (orange), which only activates during the first 30 time points. Both activities are compared with their mean across trials. (b) Comparison between the mid-trial time-averaged recurrent neural activity for output angular directions number 9 and 25.

A further analysis was conducted to determine whether recurrent activity is dependent on the encoded angle (which reflects in the response direction and, hence, in what is the most activated output ring unit) in addition to being task period-dependent. The result of time-averaging subsets of trials, all characterized by the same response angle, was that while some changes in the activity pattern were indeed recognizable, these do not modify the aforementioned essential properties of the graph: most units remained inactive and the others underwent only slight colour modifications, as shown in Figure 4.5b.

The output of the network reproduces well the given target signal, as shown in Figure 4.6.

(a)


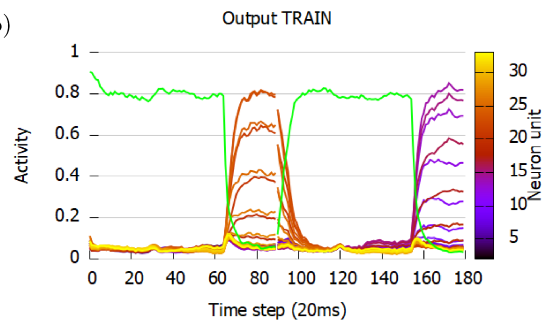
(b)


Figure 4.6: Output of the RNN for two example trials (the first two) in the training time series. (a) focuses on the fixation unit and compares the network (light green) and target (dark green) signals for the first two trials, while (b) represents all 33 units of $\mathbf{z}_t$ (z_{fix} again in green).

As in Figure 3.5, the output ring units corresponding to input $\mathbf{u}_{mod1}$ activate with intensity scaled by 0.8. Indeed, apart from a transient at the beginning of each distinct task period during which the RNN adjusts to the input signal, the activity is able to replicate $\mathbf{z}_t^{target}$ in shape and intensity. To highlight this equivalence, the difference between the two signals is plotted below for the training time series (the same behaviour was observed for the test): the expected and obtained behaviours are well aligned, consistently with the Gaussian noise of the hidden units $\sigma_{rec}=0.05$.

Such error is moreover characterized by a distribution (Figure 4.7b) which on the logarithmic scale appears to be approximately bell-shaped, with an average value of 10^{-2}. It presents a slightly more consistent tail in the direction of higher values, interrupting abruptly at 1 because neural response activity is a bounded quantity and so must be its error. Lastly, the errors were averaged across all trials to get rid of noise and dependency from the response angle, resulting in smoother plots (Figures 4.7c and 4.7d).

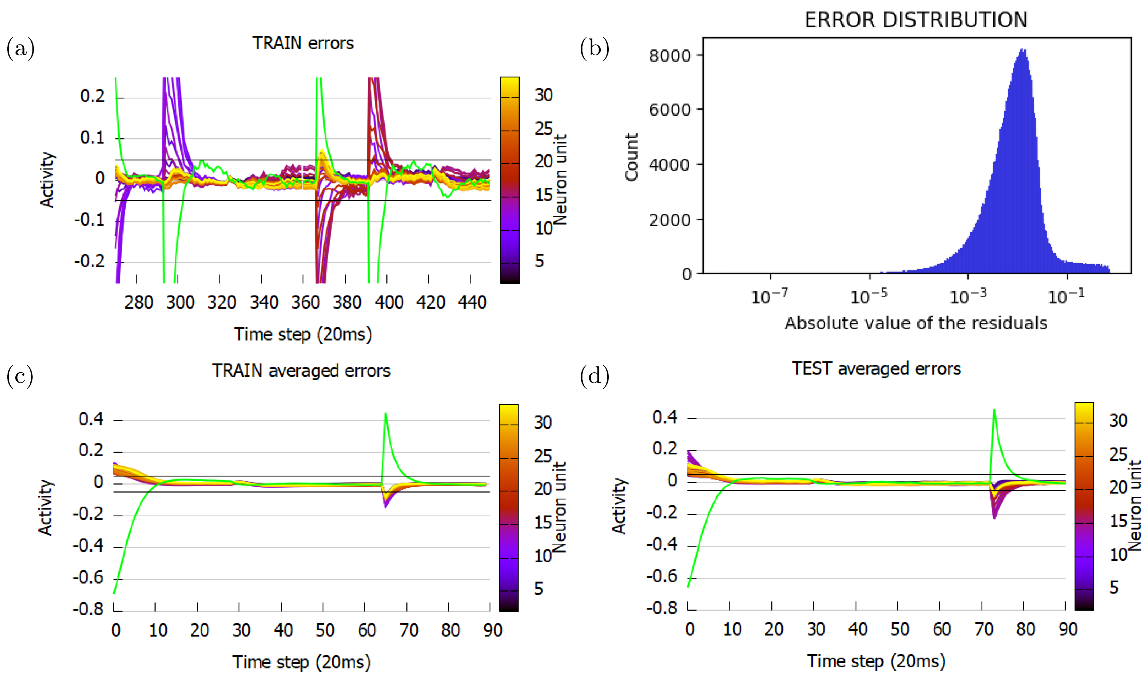


Figure 4.7: (a) Difference between network and target signals for the 33 output units (fixation in green) for the training time series. The strength of the recurrent noise ±0.05 is highlighted. (b) Distribution of all the residuals plotted with a logarithmic scale. (c),(d) Errors averaged on all trials (200 for the training set, 20 for the test set).

Chapter 5

Analysis of RNN dynamics

5.1 Task representation and compositionality

One of the main contributions in Yang et al. 2019 is the analysis of how RNNs represent learned tasks as single high-dimensional vectors in an appropriate state space.

To compute one of these “task vectors” $\tilde{\mathbf{r}}$, network activities across all trials are averaged, in order to examine all possible stimulus conditions at once. Considering (for simplicity) only the neural population steady-state response during the stimulus epoch, at time $t_{mod1,last}$ corresponding to the end of stimulus presentation, a representation of how the RNN processes a particular task input is obtained:

$$\tilde{\mathbf{r}} = \sum_{j=1}^{N_{trials}} \mathbf{r}(j, t_{mod1,last}) \qquad (5.1)$$

where $\mathbf{r}(j,t)$ is the vector of recurrent network activities at trial j and time t.

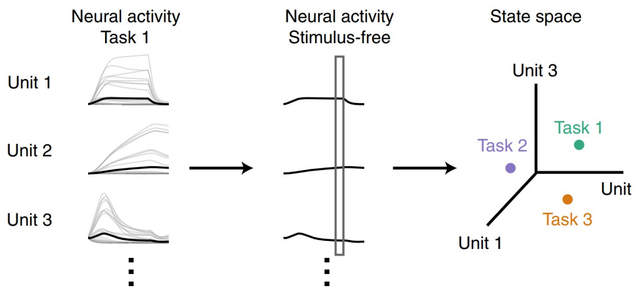


Figure 5.1: Obtaining task vectors from neural output activities (Yang et al. 2019)

Task vectors change in different epochs, as the network responds in a unique way for each given task. Indeed, this idea is confirmed using principal component analysis (or PCA)[1], which reveals that task vectors are distinct for all 20 different tasks when visualized in a state space that spans the top two principal components (Yang et al. 2019).

Task vector representation makes it possible to analyze the *compositionality* of the executed tasks. This property consists in the ability of the network to correctly perform new tasks with modest amounts of training, specifically composing pre-learned elementary processes. At the level of neural representations, the compositions could be implemented in several ways, for instance algebraically by summing task vectors in state space to yield representations for complex actions. Analyzing the degree of correspondence between these algebraic sum vectors and task vectors obtained by training the RNN to perform the respective composite tasks could thus yield insight into the way artificial and biological neural networks implement compositionality.

[1]Principal component (PC) analysis is a dimensionality reduction technique used to extract information from a high-dimensional space by projecting it into a lower-dimensional subspace, easier to visualize. It strives to preserve data features with more variation and remove non-essential ones.

As an example, Figure 5.2 presents the relative positions in PC space of task vectors referring to the Go and Anti task families as in Yang et al. 2019. It can be observed that the vector pointing from Go to Dly Go is very similar to the one pointing from Anti to Dly Anti, thus suggesting that they are likely to represent the cognitive process of working memory (abbreviated as "Dly"), the ability to maintain and manipulate information for several seconds during the planning and execution of a task (Song et al. 2016). The presence of many task vector points for the same task indicates that the analysis described above was repeated for many different RNNs, showing that results are consistent across networks.

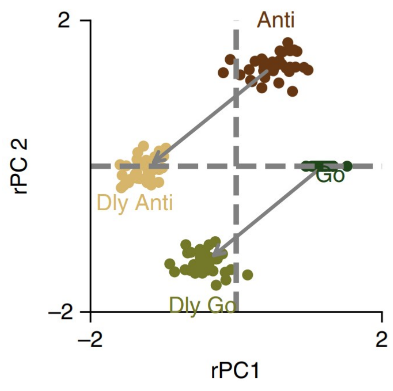


Figure 5.2: Task vectors represented in a rotated and reflected PC space, suitable for visualization (from Yang et al. 2019).

This finding suggests that networks trained to perform complex cognitive processes involving "Dly" as a key subprocess do exploit a form of compositionality, at least at the level of *task representation*. Further studies are needed to investigate whether compositionality extends to the level of *task execution*, i.e., whether jointly activating several rule units corresponding to subprocesses is sufficient to have the network perform the complex task.

5.2 The dynamical systems approach

The present paragraph concentrates on transcending a static view of neural representation to focus instead on a dynamical approach, particularly suitable as RNNs are computational models for which temporal development is crucial. As their dynamics is that of a complex, high-dimensional nonlinear system, it is useful to analyze it around *fixed points*, locations in state space where the motion of the system is approximately nullified and the dynamics can therefore be greatly simplified with a linearization. It was indeed argued (Sussillo et al. 2013) that this kind of study can aid the "opening of the black box" of RNN mechanisms of operation, that is it can shed light on how the networks implement their computations.

Let the following be the continuous dynamical equation describing a generic RNN

$$\frac{d\mathbf{r}}{dt} = \frac{1}{\tau}\big(-\mathbf{r} + f(W^{rec}\mathbf{r} + W^{in}\mathbf{u} + \mathbf{b}_{rec} + \sqrt{2\tau\sigma_{rec}^2}\ \boldsymbol{\xi})\big) \equiv F(\mathbf{r},\mathbf{u}) \tag{5.2}$$

where $\mathbf{r}$ is the activity of the recurrent units and the notation $\mathbf{u}$ refers to the input. A fixed point $(\mathbf{r}^*, \mathbf{u}^*)$ is a state such that $F(\mathbf{r}^*, \mathbf{u}^*) = 0$, around which the network dynamics can be linearized through Taylor first-order expansion (Strogatz 2001).

$$F(\mathbf{r}^* + \delta\mathbf{r}, \mathbf{u}^* + \delta\mathbf{u}) \ \approx\ F(\mathbf{r}^*, \mathbf{u}^*) + \frac{\partial F}{\partial \mathbf{r}}(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{r} + \frac{\partial F}{\partial \mathbf{u}}(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{u} \ \approx\ \mathbf{J}(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{r} \tag{5.3}$$

where $\delta\mathbf{r} \equiv \mathbf{r} - \mathbf{r}^*$ and $\delta\mathbf{u} \equiv \mathbf{u} - \mathbf{u}^*$. Indeed, the second order terms and $\delta\mathbf{u}$ are negligible since during all task periods the input is kept constant (apart from the noise). Moreover, the term $\frac{\partial F}{\partial \mathbf{r}}(\mathbf{r}^*, \mathbf{u}^*)$ is the Jacobian matrix evaluated at the fixed point, which can be written element-wise as

$$J_{ij} = \frac{\partial F_i}{\partial r_j}(\mathbf{r}^*, \mathbf{u}^*) = \frac{1}{\tau}\big(-\delta_{ij} + f'(\mathbf{r}^*, \mathbf{u}^*)\ W_{ij}^{rec}\big) \tag{5.4}$$

where δ_{ij} indicates the Kronecker delta function and the activation function is derived with respect to its argument, the cell state.

By studying the eigendecomposition of $\mathbf{J}(\mathbf{r}^*, \mathbf{u}^*)$, useful information on network dynamics can be inferred. Specifically, if λ is defined as the real part of an eigenvalue, the associated eigenvector corresponds to a contracting dimension of the state space if $\lambda > 1$, to an expanding dimension if $\lambda < 1$ and to a marginally stable one if $\lambda \approx 1$. A fixed point that is contracting in every dimension is a *basin of attraction*, a state particularly useful to prepare an optimal initial condition for the next task period (Driscoll et al. 2022).

The key step of this analysis is then the research of fixed points in the dynamical landscape referring to each task period of interest. In ANNs, they are found through gradient-based optimization of the auxiliary "kinetic energy" scalar function

$$q(\mathbf{r}, \mathbf{u}) = \frac{1}{2} \left|F(\mathbf{r}, \mathbf{u})\right|^2 \tag{5.5}$$

which leads to minimizing $|F(\mathbf{r}, \mathbf{u})|$.

From an analytical viewpoint the following inequalities must be verified around fixed points.

$$|F'(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{r}| > |F(\mathbf{r}^*, \mathbf{u}^*)| \tag{5.6}$$

$$|F'(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{r}| > \left|\frac{1}{2}\delta\mathbf{r}F''(\mathbf{r}^*, \mathbf{u}^*)\delta\mathbf{r}\right| \tag{5.7}$$

This kind of approach relies on hard-coded analytic derivatives of the function f and can therefore become cumbersome as the complexity of the RNN models increases. The problem could be partially alleviated by employing numerical derivation methods, although these include additional computational costs which typically make them unfeasible (Golub and Sussillo 2018), so semi-analytical algorithms or specifically designed computational tools are instead to be preferred.

A way to understand how fixed points fit into a wider description of the computations implemented by RNNs is to infer *dynamical motifs*[2] from locally linearized dynamical equations. This type of analysis is aided by graphically visualizing how neural activity vectors move in state space, typically in a 2-dimensional PCA space as shown in Figure 5.3. Here, it can be observed that activity tends to converge to a line in the Cartesian plane for all stimulus conditions, thus indicating the presence of a line attractor.

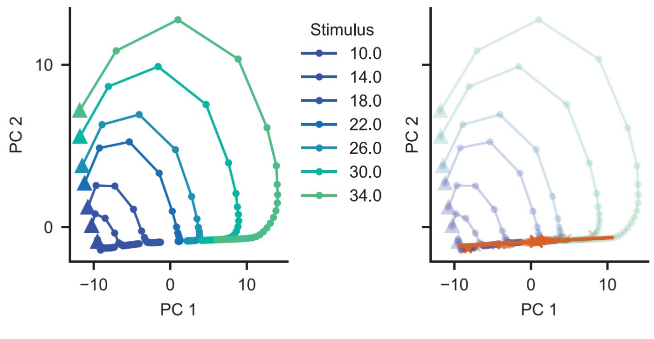


Figure 5.3: From Yang and Wang 2020. (left) Trial-averaged activity during the delay period (triangles indicate the start) for different stimulus values. (right) Fixed points found through optimization, in orange. A line attractor was estimated around them by finding the eigenvector with a corresponding eigenvalue close to 0.

[2]According to Driscoll et al. 2022, a dynamical motif is the high-dimensional nonlinear dynamics around a fixed point, that implements computation for a specified input.

Along the lines of the above illustration, a PCA was performed, separately on the activity of the input, recurrent and output layer of the network. For each layer, the first two components were kept, explaining respectively $XX\%$ and $YY\%$ of the total variance.

The neural activity of the RNN at hand is plotted in PC space for all trials in the following Figure 5.4. Each subplot represents either the input, recurrent or output activity at different time points.

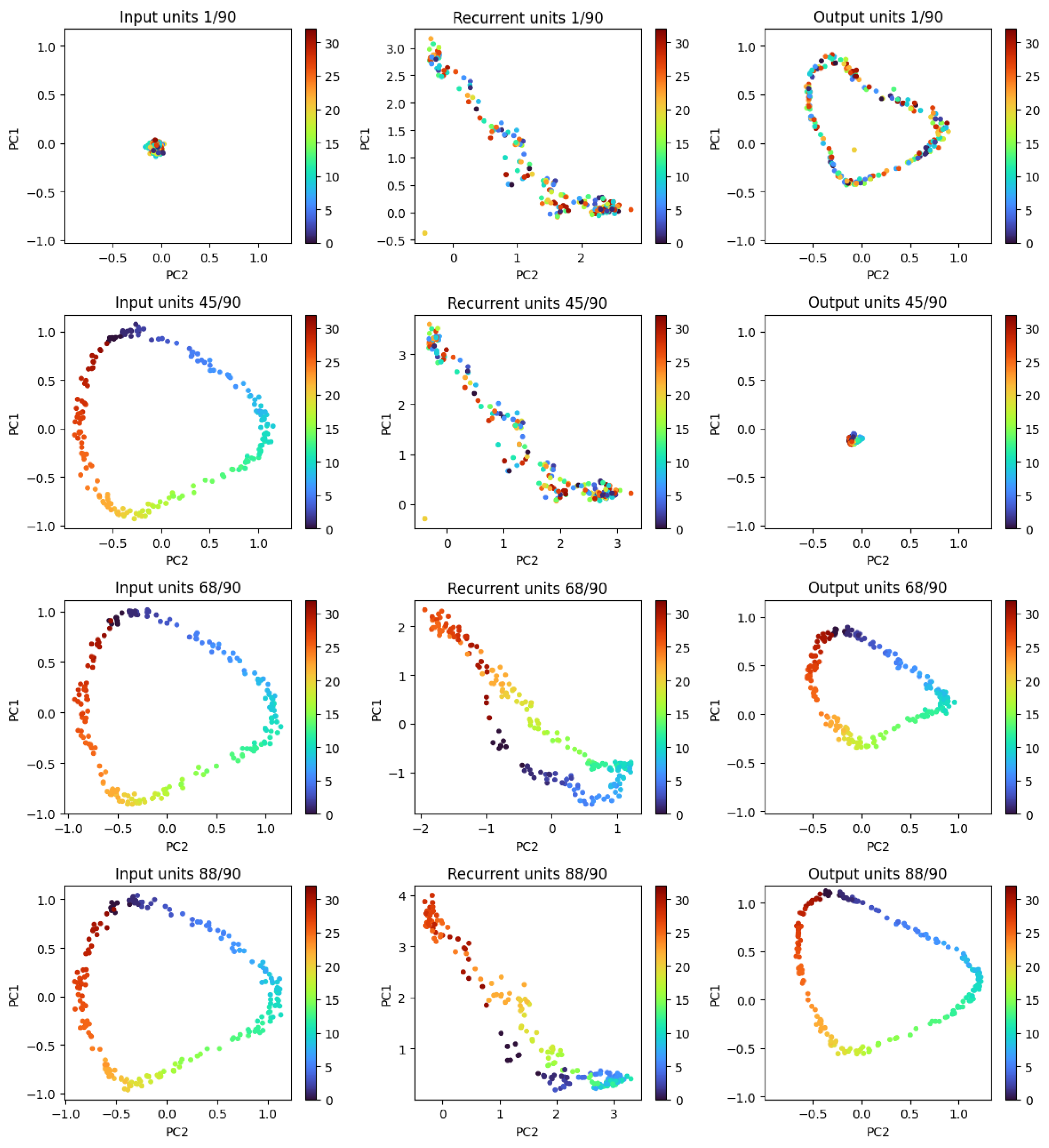


Figure 5.4: Neural activity for the training data set, subdivided according to the neural unit group (input, recurrent, output) and to the time point. Each dot in the 2-dim PC space scatterplot corresponds to one trial: the colour scale indicates the trial-specific input angle, which corresponds to the output angle as the task at hand is Go.

From the above Figure, many features of the implemented Go task can be inferred. The input is initially off (first panel, input activity is 0 for all trials) and it switches on at time point 30 to remain constant until the end of the trial. It can indeed be observed that in the last three panels of the first column the dynamical landscape takes the same shape, topologically equivalent to a circle where the points are ordered angle-wise with the corresponding colour. This also indicated that all angles within $[0, 2\pi)$ are spanned during the 200 trials.

The output is dormant until the fixation cue goes off at time point 65, after which it expands smoothly to reproduce the input (last panels of the third column). The angle dependency is maintained, showing that apparently the response of the network is indeed in the same direction of the input, as required from the Go task. The first output panel, though, shows a disorderly colour distribution, as well as a distorted circular shape. This illustrates the transient period (circa 30 time points long), where the output is still that of the previous trial and adjusting to the new one by retracting in a single point near the origin. This interpretation is confirmed by the presence of a yellow central dot: it corresponds to the first trial of the time series, the only one for which the output is initially off as there is no previous input.

As for the recurrent part (second column), the first two plots reveal that the neural activity of the hidden units is also shaped in PC space as the topological equivalent of a circle. The reason why this is not as evident as in the input and output plots is presumably that a third principal component is needed to visualize it. Before the response period (first two panels), the angular dependency is not that of the current trial but of the precedent one. After the fixation cue goes off instead, the described relation between the angular variable around the plotted circle and the most activated ring units is reinstated, as in the output.

For the test set the same results can be observed, consistently with the fact that the time trials are 20 instead of 200, so data is more sparse and sensitive to noise.

The second part of this analysis focuses on the time evolution of neural activity vectors in PC space. With the aim of identifying time as a direction in this space, accounting for a significant part of the data set variance, the first 3 principal components were considered, accounting respectively for $XX\%$, $YY\%$ and $ZZ\%$ of the total variance. The results are plotted below.

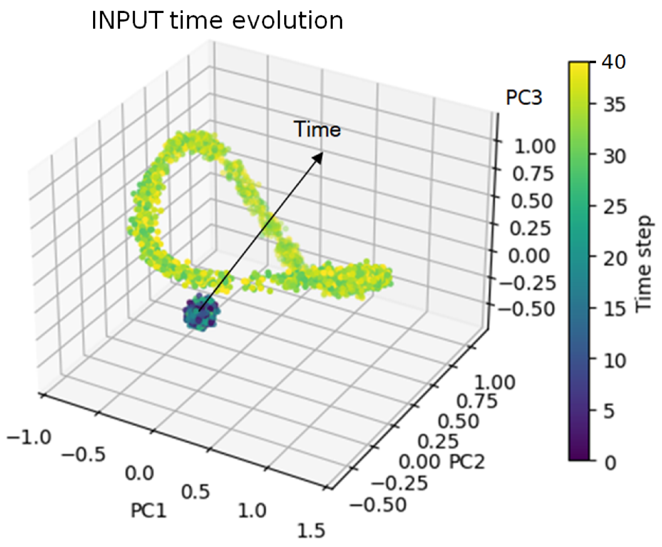


Figure 5.5: Input neural activity at the beginning of the training time trials, plotted in a 3-dimensional PC space. The time coordinate is inferred from the respective positions of the two data clusters.

Firstly, Figure 5.5 provides a visualization of the input signal in time for time points 0-40. It can be noted that the plot is neatly split in two clusters, corresponding to neural activity before and after the input signal is switched on. Indeed, for the first 30 time steps all dots are bundled together near the origin, while later they are arranged in the circular structure also visible in Figure 5.4. As the input units activate quite abruptly and time discretization was introduced, there are no points between the two clusters, modeling an instantaneous change of the dynamical landscape. It can therefore be inferred that, in this 3-dimensional state space, the time coordinate evolves in the direction of the black arrow. For all further time points input activity remains unaltered, with vectors arranging on the circular structure until the end of the trial.

The next step of this discussion concerns the output, the dynamics of which can be divided in three different phases (Figure 5.6). At the beginning of the trial (0-10) the circular structure resulting from the previous trial collapses towards the origin of PC space (first panel), where it remains stationary until the start of the response period (10-64). It then expands once again (second panel) to reach the yellow ring that reproduces the given input (65-90). The resulting figure in the 3-dimensional space is then topologically equivalent to a cone. The third panel uses the colour scale to visualize the angle as in Figure 5.4, highlighting that the ring expands smoothly.

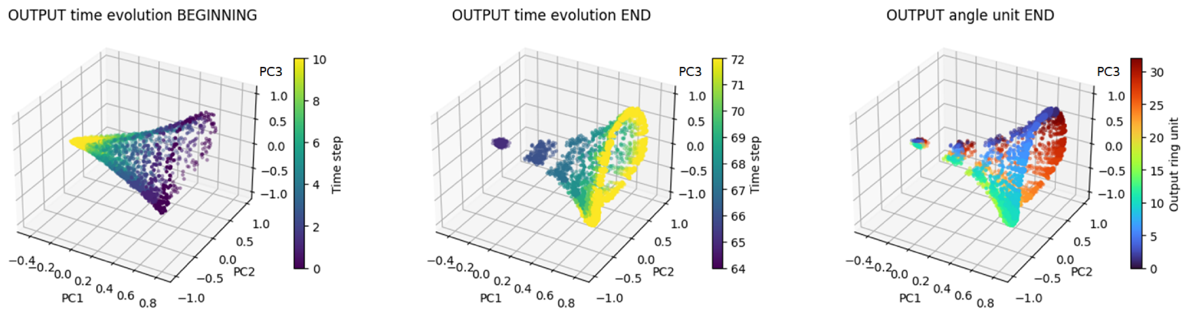


Figure 5.6: Output neural activity at the beginning and end of the training time trials, plotted in a 3-dimensional PC space.

Moving on to the recurrent units activity, once again the time trials can be divided in two separate parts, one encompassing the delay and stimulus task periods and the other the response of the network. In this case, activity evolving in time takes a shape which is more complex (Figures 5.7 and 5.8): the first 65 time points are plotted in the second panel and the last 25 in the third one. The dots start on a triangle (topologically, a ring), and converge to a point in the interval 0-10, hence describing a figure that is topologically equivalent to a cone. This is very similar to what is observed for the output unit. In the interval 10-30, the dots remain confined around the origin, while in 30-65 they expand on a plane that is approximately perpendicular to the axis of the cone, where they describe a triangle (topologically, a ring) with increasing radius. Finally, in the interval 65-90, the dots describe a triangular prism (topologically, a cylinder), going back to the triangle found at time 0.

The following Figures 5.7 and 5.8 illustrate the described dynamics in a 3-dimensional PC space. While the former depicts the side of the (topologically) cylindric structure on which the dots move in time, the latter is rotated by 45 degrees in order to highlight the triangular shape at the base of the prism and cones.

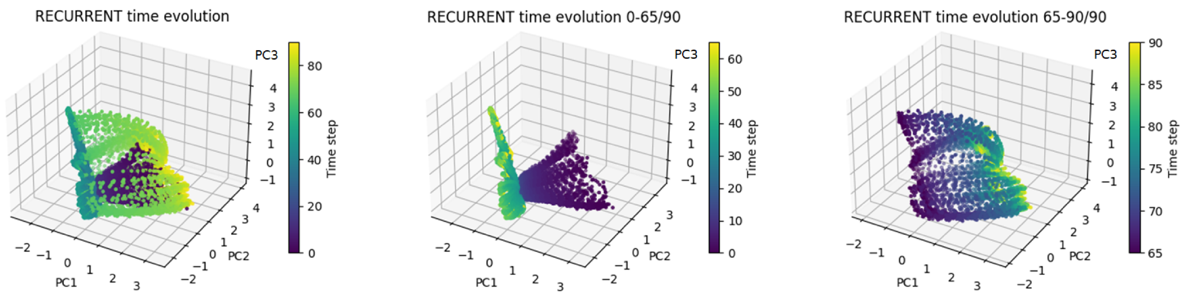


Figure 5.7: Recurrent neural activity for the whole training trial length, at the beginning and at the end, plotted in a 3-dimensional PC space.

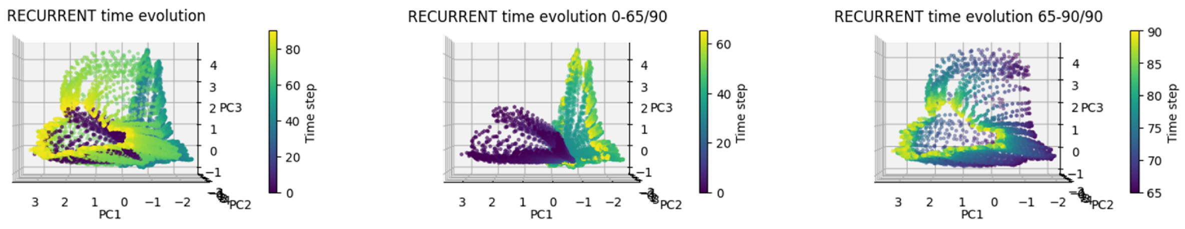


Figure 5.8: Recurrent neural activity for the whole training trial length, at the beginning and at the end, plotted in a 3-dimensional PC space and rotated to best visualize the dynamics.

Using the colour scale to plot the angular stimulus variable, it can be noted that for the recurrent part of the network as well, the angle varies smoothly around the (topological) rings for all time points.

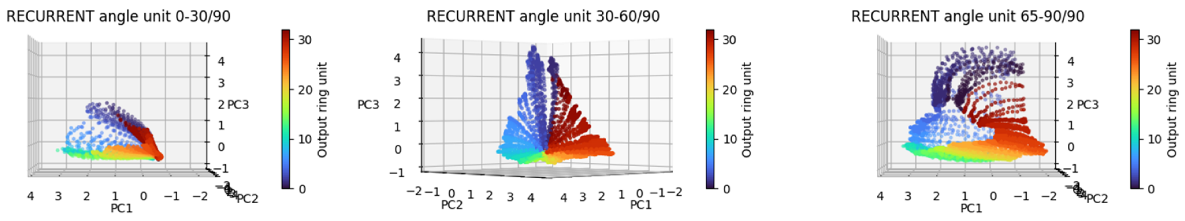


Figure 5.9: Recurrent neural activity for the whole training trial length, at the beginning and at the end, plotted in a 3-dimensional PC space.

The last part of the analyses based on a dynamical systems approach is devoted to the search for fixed points in the dynamics. In particular, according to the general hypotheses advanced by Driscoll et al. 2022, the trajectories were expected to reach fixed points encoding for the stimulus and the output motion angles. More precisely, taking into account the above analysis of trajectories, fixed points were expected to be found in three distinct locations: in correspondence of the dot to which trajectories converge at time 10-30, around the points reached by trajectories at time 60, encoding for the stimulus angle, and thirdly around the points reached by the trajectories at time 90, encoding for the motion angle.

The analysis of fixed points revolves around the definition of the multivariate function F introduced in Equation 5.2,

$$F(\mathbf{r},\mathbf{u}) = \frac{1}{\tau}(-\mathbf{r} + f(W^{in}\mathbf{u} + W^{rec}\mathbf{r} + \mathbf{b}_{rec} + \boldsymbol{\xi}))$$

F can be considered the system's "velocity", and its modulus the system's "speed". Therefore, fixed points can be determined by finding its roots: by imposing that $F(\mathbf{r},\mathbf{u}) = 0$ for generic input and recurrent activity vectors, the condition below follows.

$$-\mathbf{r} + f(W^{in}\mathbf{u} + W^{rec}\mathbf{r} + \mathbf{b}_{rec} + \boldsymbol{\xi}) = 0 \tag{5.8}$$

On this basis, the root-finding algorithm implemented in Python was employed to determine its zeros in search of possible fixed points. This procedure did return some approximate zeros, but two issues arose. The first one is that the function did not exactly nullify at these points and secondly, the points were far from the initial guesses. In fact, the dynamical speed never reached values close to zero in the analyzed trajectories, except around time point 30 (not shown).

What emerges is therefore a partially negative result: on the one hand, it can be established that low-dimensional trajectories in state space accurately encode the angular dynamics of ring units in all different task periods, and that the top three principal components are sufficient to get a clear view of such trajectories and infer their key properties. On the other hand, the presence of fixed points at times 65 and 90 can be ruled out, and the presence of a fixed point at time 30 must remain hypothetical, as the 3-dimensional trajectories seem to converge but the employed numerical root-finding algorithm is not able to directly detect it.

This discrepancy with respect to the hypothesis put forward by Driscoll et al. 2022 may be due to the fact that the RNN was trained to perform only one task, which may be insufficient for a full-scale fixed point structure to arise. To further investigate these assumptions, more comprehensive analyses should be carried out on networks able to execute multiple cognitive tasks, as in Yang et al. 2019.

Chapter 6

Conclusions

The general goal of this thesis was to investigate how the computational properties of a recurrent neural network relate to its dynamics, specifically to phase space trajectories and attractors.

In order to accomplish this objective, two major hypotheses advanced by previous literature were tested. The first one is that task variables are collectively represented by all units, by means of neural activity trajectories unfolding in a low dimensional space. Moreover, according to the second claim, such trajectories cross the fixed points of the dynamical landscape during one or more phases in each trial.

This study results can confirm the first hypothesis, but not the second one. However, it only focused on an RNN trained for one simple task ("Go"). Whether networks trained for multitasking might display a clearer fixed points structure remains an open question.

Further advances in this research area will require improving the biological plausibility of RNN brain models. Directions for progress are suggested in Yang et al. 2020 as well as in Miconi 2017, and they specifically involve modifying the network training algorithm. Indeed, the backpropagation-trough-time mechanism is known to be unrealistic, but RNNs can be trained with other methods, much more adherent to biological neuroplasticity, such as a reward-modulated "Hebbian" learning rule (Miconi 2017). This type of training has been successfully used to effectively reproduce the neural dynamics observed in mammalian brains during cognitive tasks.

Bibliography

Wang, X.-J. (2002). "Probabilistic Decision Making by Slow Reverberation in Cortical Circuits". In: *Neuron* 36, pp. 955–968. DOI: https://doi.org/10.1016/S0896-6273(02)01092-9.

Mastrogiuseppe, F. and S. Ostojic (2018). "Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks". In: *Neuron* 99.3, 609–623.e29. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2018.07.003.

Yang, G.R., M.R. Joglekar, et al. (2019). "Task representations in neural networks trained to perform many cognitive tasks". In: *Nature Neuroscience* 22, pp. 297–306. DOI: https://doi.org/10.1038/s41593-018-0310-2.

Yang, G.R. and X.-J. Wang (2020). "Artificial Neural Networks for Neuroscientists: A Primer". In: *Neuron* 107, pp. 1048–1070. DOI: https://doi.org/10.1016/j.neuron.2021.01.022.

Song, H.F., G.R. Yang, and X.-J. Wang (2016). "Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework." In: *PLoS Computational Biology* 12. DOI: https://doi.org/10.1371/journal.pcbi.1004792.

Werbos, P.J. (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560. DOI: https://doi.org/10.1109/5.58337.

Elman, J.L. (1990). "Finding structure in time". In: *Cognitive Science* 14, pp. 179–211. DOI: https://doi.org/10.1016/0364-0213(90)90002-E.

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning". In: *Nature* 521, pp. 436–444. DOI: https://doi.org/10.1038/nature14539.

Sussillo, D. and O. Barak (2013). "Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks." In: *Neural Comput.* 25, pp. 626–649. DOI: https://doi.org/10.1162/neco_a_00409.

Golub, M. and D. Sussillo (Nov. 2018). "FixedPointFinder: A Tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks". In: *The Journal of Open Source Software* 3, p. 1003. DOI: https://doi.org/10.21105/joss.01003.

Furukawa, H. et al. (2005). "Subunit arrangement and function in NMDA receptors". In: *Nature* 438, pp. 185–192. DOI: https://doi.org/10.1038/nature04089.

Pierce, J.E., B.A. Clementz, and J.E. McDowell (2019). In: *Eye Movement Research: An Introduction to its Scientific Foundations and Applications.* Springer International Publishing. Chap. Saccades: Fundamentals and Neural Mechanisms, pp. 11–71. ISBN: 978-3-030-20085-5. DOI: https://doi.org/10.1007/978-3-030-20085-5_2.

Levitt, J. B. (2018). "Encyclopedia Britannica". In: chap. "Receptive field". URL: https://www.britannica.com/science/receptive-field.

Driscoll, L., K. Shenoy, and D. Sussillo (2022). "Flexible multitask computation in recurrent networks utilizes shared dynamical motifs". In: *bioRxiv.* DOI: https://doi.org/10.1101/2022.08.15.503870.

Kingma, D.P. and J.L. Ba (2015). "Adam: A method for stochastic optimization". In: *ICLR.* DOI: https://doi.org/10.48550/arXiv.1412.6980.

Strogatz, S.H. (2001). *Nonlinear Dynamics And Chaos With Applications to Physics, Biology, Chemistry and Engineering*. Westview Press. ISBN: 0-7382-0453-6. DOI: `https://doi.org/10.1201/9780429492563`.

Miconi, T. (2017). "Biologically plausible learning in recurrent neural networks reproduces neural dynamics observed during cognitive tasks". In: *eLife* 6, e20899. DOI: `https://doi.org/10.7554/eLife.20899`.