UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

Studio della dimensionalità in reti neurali ricorrenti

Study of dimensionality in recurrent neural networks

Relatore

Dott. Michele Allegra

Laureando

Giacomo Gasparotto

Anno Accademico 2023/2024

Abstract

In computational neuroscience, recurrent neural networks (RNNs) are commonly used to model the structure and the dynamics of neuronal circuits performing simple cognitive operations. One of the most important dynamic features of RNNs is the intrinsic dimension of their activity (in configuration space). While most studies agree on indicating a low dimension, the specific value of the dimension seems to vary depending on details of the implementation. In this thesis, the graduate student will become familiar with RNNs and intrinsic dimension estimators and will analyze the dependence of dimension on the algorithm used to train the RNN.

Nelle neuroscienze computazionali, le reti neurali ricorrenti (RNN) sono comunemente usate per modellizzare la struttura e la meccanica di circuiti neuronali che svolgono semplici operazioni cognitive. Una delle caratteristiche dinamiche salienti delle RNN è la dimensionalità intrinseca della loro attività (nello spazio delle configurazioni). Se la maggior parte degli studi è concorde nell'indicare una bassa dimensionalità, il valore specifico della dimensione sembra variare in base a dettagli della realizzazione. In questa tesi, il laureando prenderà dimestichezza con le RNN e gli stimatori di dimensione intrinseca, e analizzerà la dipendenza della dimensione dall'algoritmo con cui la RNN è addestrata.

Contents

Chapter 1

Introduction

Large volumes of high-dimensional data are of fundamental importance in neuroscientific studies. However, their analysis could be complex, due to the large amount of data to examine. There are several approaches working on the assumption that the relevant information of a given dataset can be successfully compressed in a low number of variables. Geometrically, this possibility depends on the fact that data points typically belong to a manifold whose intrinsic dimension (ID) is much lower than the actual, larger, number of coordinates in data space. [2]

In this thesis project we aim to understand the dynamics of recurrent neural networks (RNNs) trained to perform cognitive tasks. In neuroscientific literature there are many hypothesis about the fact that a task's computational complexity and the training algorithm applied to learn the task would influence the intrinsic dimension of the network's activity during the given task.

In order to test the truthfulness of these hypothesis we have considered 20 different cognitive tasks, taken from [8], that depend on working memory, decision making, categorization, and inhibitory control. We trained RNNs to perform each of these tasks using two different algorithms: *Back-propagation through time* (BPTT) and *Reservoir computing* (RC). These two architectures for RNNs have some differences:

- BPTT updates every weight, including recurrent ones, by propagating errors through each temporal step;
- RC keeps the weights of the recurrent part unchanged and only trains the output weights.

We want to investigate if these differences result in different intrinsic dimensions, as estimated by a geometric method.

Among all intrinsic dimension estimators we have implemented the TWO-NN algorithm [2]. We opted to use this method because the manifold in which the data lies is generally twisted and curved and points on it will be non-uniformly distributed. Considering only the first and second shortest distances of each point, TWO-NN reduces the effects of curvature, of density variation, and the resulting computational cost.

In Chapter 2, all the theoretical methods applied in the analysis are described. First of all we will introduce what a RNN is and how it works in general, focusing on the dynamics for BPTT and RC. Subsequently, the algorithm for estimating intrinsic dimension will be explained, including the case of scale dependence.

In Chapter 3, we will present an overview of the chosen tasks and network architecture, along with the application of the intrinsic dimension algorithm to two randomly chosen tasks (*Anti-response* and *Context-dependent delayed decision making 1*), including a comparison of the estimated dimensions with the PCA analysis. Finally, the chapter will conclude with the final results, featuring a comparison of the two training algorithms in terms of their efficiency in performing each task [3–5, 7].

Chapter 2

Methods

In this chapter, all the methods applied in the analysis will be reviewed. First of all, there will be a brief introduction to recurrent neural networks, including their dynamics and network architecture. Then, we will present two training methods: back-propagation through time (BPTT) and reservoir computing (RC). Finally, we will discuss an algorithm used for the estimation of intrinsic dimension.

2.1 Recurrent neural networks

Recurrent neural networks (RNNs) are computer models inspired by the brain, designed by more or less detailed analogy with biological brain modules. In an RNN, many basic units (like brain cells), the neurons, are connected together by abstract synaptic connections. These links allow information to move through the network. Unlike feedforward neural networks, RNNs display cycles, with important impacts:

- An RNN can generate its own ongoing activity patterns through its internal connections, even without any input. This means that, mathematically, an RNN behaves like a dynamical system, whereas feedforward networks simply process input and produce output;
- If provided with an input signal, an RNN retains a nonlinear transformation of its input history within its internal state. This means that it possesses a dynamic memory and can handle temporal context information.

These properties lead to use RNNs for many different scientific purposes, from modeling biological brains (in computational neuroscience) or implementing machine learning algorithms (in artificial intelligence) [6].

Any application of RNNs involves three ingredients:

- Network architecture;
- Learning problem;
- Training algorithm.

Network architecture Among all artificial neural networks (ANNs), RNNs have the specific ability to process information in time.

To describe the network dynamics in general, is useful to present the cell state at time t, $\boldsymbol{c}_t$.

$$\boldsymbol{c}_t = \boldsymbol{W}_{rec}\boldsymbol{r}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{u}_t + \boldsymbol{b}_{rec} \tag{2.1}$$

The cell state represent the drivers of neuronal activity at time t, $\boldsymbol{r}_t$, which include recurrent connectivity $\boldsymbol{W}_{rec}$ and inputs $\boldsymbol{u}_t$ (through connectivity $\boldsymbol{W}_{in}$), $\boldsymbol{b}_{rec}$ being a bias term (Fig. 2.1).

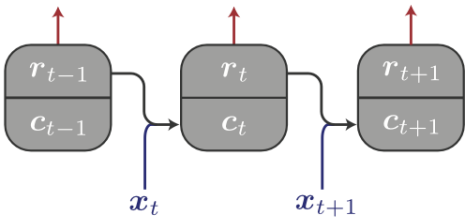

Figure 2.1: Picture of a recurrent neural network in which is shown that each layer corresponds to the network state at one time step [9].

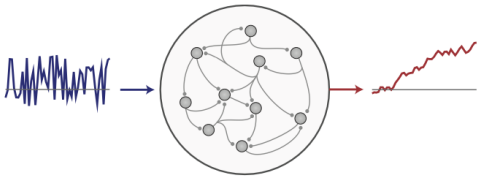

Figure 2.2: Scheme of a recurrent neural network. It receives an input signal from the left which is processed in the middle, yielding the desired output on the right [9].

The neuronal activity at time t, $\boldsymbol{r}_t$, is a function of the cell state:

$$\boldsymbol{r}_t = f(\boldsymbol{c}_t) \tag{2.2}$$

where $f(\cdot)$ is a chosen nonlinear activation function.

The output vector $\boldsymbol{z}_t$ is computed as the product between $\boldsymbol{W}_{out}$ and $\boldsymbol{r}_t$ with the addiction of a bias $\boldsymbol{b}_{out}$.

$$\boldsymbol{z}_t = \boldsymbol{W}_{out}\boldsymbol{r}_t + \boldsymbol{b}_{out} \tag{2.3}$$

In practice, the RNN can be seen as a (infinitely deep) feedforward network, where each time point t corresponds to a layer. The neurons in the t-th layer $\boldsymbol{r}_t$ receive inputs form the $(t-1)$-th layer $\boldsymbol{r}_{t-1}$ and additional inputs from outside of the recurrent network $\boldsymbol{u}_t$. The connections form each layer to the next are shared across time. What has just been described is outlined in the figure (Fig. 2.3). As you can see there are 85 input units, 256 recurrent units and 33 output units. These numbers have been chosen because they represent the number of units used in the application that will be discussed in the next chapter.

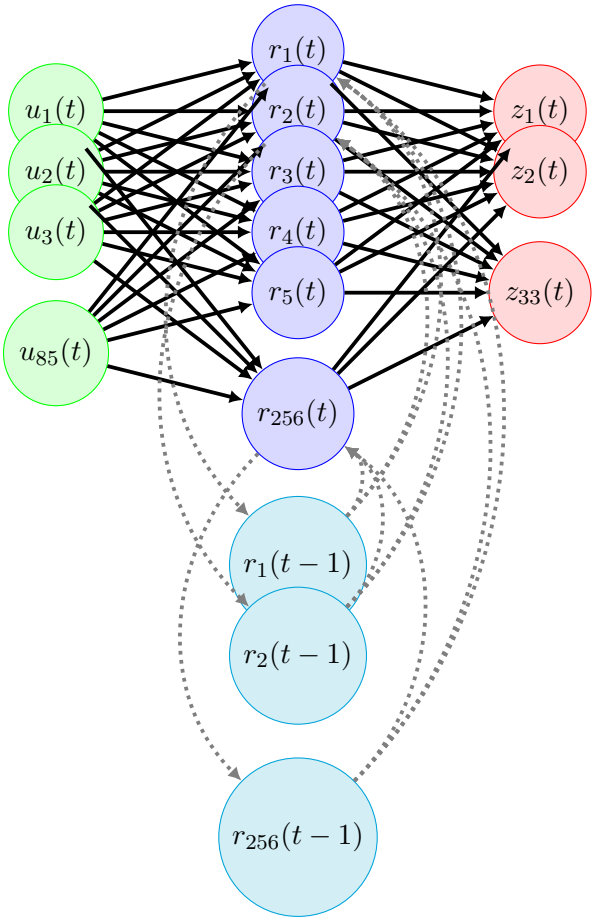

Figure 2.3: Schematic of the RNN used. The input units are connected to the recurrent units that reproduce the output. Dotted lines represent recurrent connections.

Learning problem The network architecture constrains the structure of connections between neurons in the neural network. However, it does not fix the values of the weights and biases, collectively denoted by $\boldsymbol{\theta}$. These values are randomly assigned at initialization, and then trained using an algorithm. This training algorithm specifies how connections weights change to better perform a specific task [9].

The task consists in learning to fit a dataset containing a set of inputs $\boldsymbol{x}^{(i)}$, with $i = 1, ..., N$, paired with target outputs $\boldsymbol{y}^{(i)}_{target}$. The neural network output $\boldsymbol{y}^{(i)}$ can be seen as a function $F(\boldsymbol{x}^{(i)}, \boldsymbol{\theta})$ of the input and network parameters. The goal of the learning problem is to estimate the values of the

parameters $\boldsymbol{\theta}$ such that a neural network function $F(\cdot, \boldsymbol{\theta})$ predicts the target output given the inputs, $\boldsymbol{y}^{(i)} = F(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \approx \boldsymbol{y}^{(i)}_{target}$ (Fig. 2.2). In other words, the system is trained to minimize the so-called loss function $L = \frac{1}{N}\sum_i L(\boldsymbol{y}^{(i)}, \boldsymbol{y}^{(i)}_{target})$, where $L(\boldsymbol{y}^{(i)}, \boldsymbol{y}^{(i)}_{target})$ quantifies the difference between the target output and the actual output. Doing so, the value of an objective function is optimized [9].

Training algorithm In most cases the training algorithm used is stochastic gradient descent (SGD). Trainable parameters $\boldsymbol{\theta}$ are updated in the opposite direction of the gradient loss $\frac{\partial L}{\partial \boldsymbol{\theta}}$. Usually, evaluating the loss using the entire training dataset is computationally too expensive, so it is preferred to use a small number M of randomly selected training examples (minibatch), indexed by $\mathcal{B} = \{k_1, ..., k_M\}$,

$$L_{batch} = \frac{1}{M}\sum_{k\in\mathcal{B}} L(\boldsymbol{y}^{(k)}, \boldsymbol{y}^{(k)}_{target}) \tag{2.4}$$

The gradient represents the direction of parameters change that would lead to the maximum increase in the loss function when the change is small enough. In order to decrease the loss, the parameters has to be updated in the opposite direction of the gradient proportionally to the learning rate η:

$$\Delta\boldsymbol{\theta} = -\eta\frac{\partial L}{\partial \boldsymbol{\theta}} \tag{2.5}$$

The computation of the gradient is commonly performed with the back-propagation algorithm, which computes gradient components recursively starting from the last layer. However, in RNNs, the presence of cycles (or recurrent connections) introduces a temporal dependency in the network's computations, requiring consideration of the temporal dimension during the error backpropagation process. BPTT handles this aspect by unrolling the network over time and applying error backpropagation at each time step. [9].

An alternative to BPTT is *Reservoir computing* (RC). In 2001, Wolfgang Maass with his *Liquid State Machines* and Herbert Jager with *Echo State Networks*, independently proposed a new approach to RNN design and training, involving only training of the recurrent-to-output connections. In this method a RNN, which is the so called reservoir, is randomly created and remains unchanged during training. It is passively excited by the input signal and maintains in its state a nonlinear transformation of the input history. The output signal is generated as a linear combination of the neuron's signals from the input-excited reservoir. Using the teacher signal as a target, the linear weights are optimized (via linear regression) [6].

In the following figure (Fig. 2.4) we show the different approaches of classic gradient-based RNNs and a RC.

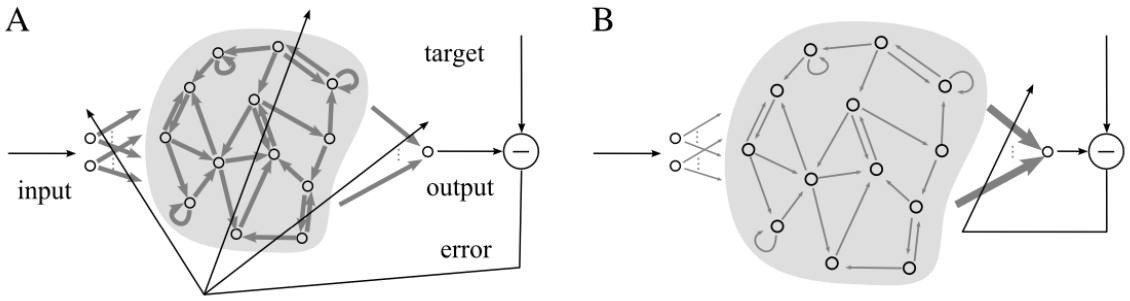

Figure 2.4: A: Classic BPTT network. All weights are adapted: input to recurrent, recurrent to internal and recurrent to output.
B: RC network. Only recurrent to output weights are adapted [6].

According to the proponents of the method, RC has more advantages compared to classic RNN methods, including modeling accuracy, modeling capacity, biological plausibility, extensibility and

parsimony. However, substantial enhancements and expansions are needed to make RC generally effective. In particular, the mere random creation of a reservoir is inadequate: to address a particular modeling task, a tailored reservoir design customized to the task will yield superior outcomes compared to a simplistic random approach [6].

2.2 Network dynamics

Now, we will describe the dynamics of the two networks that will be applied in the following analyses. We used slightly different architectures depending on the training method adopted (BPTT or RC).

Network dynamics for BPTT. We used a time-discretized network with positive activity. Before time discretization, the network dynamics is described by the following dynamical equation:

$$\tau \frac{d}{dt}\boldsymbol{r} = -\boldsymbol{r} + f(W_{rec}\boldsymbol{r} + W_{in}\boldsymbol{u} + \boldsymbol{b} + \sqrt{2\tau\sigma_{rec}^2}\boldsymbol{\xi}) \tag{2.6}$$

where $\tau = 100$ ms represent the time constant which mimics the slower synaptic dynamics on the basis of NMDA receptors, even if real neurons typically have time constants around 20 ms. $\boldsymbol{u}$ is the input vector, $\boldsymbol{r}$ is the network activity, $\boldsymbol{b}$ is the bias, $\boldsymbol{\xi}$ are N_{rec} independent Gaussian white noise with 0 mean and unit variance and $\sigma_{rec} = 0.05$ is the strength of the noise. Neuronal non-linearity is implemented using a standard Softplus function as activation function:

$$f(x) = \log(1 + e^x) \tag{2.7}$$

The dynamical equation (Eq. 2.6) has been discretized using a first order Euler approximation with a time discretization step $\Delta t = 20$ms:

$$\boldsymbol{r}_t = (1-\alpha)\boldsymbol{r}_{t-1} + \alpha f(\boldsymbol{W}_{rec}\boldsymbol{r}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{u}_t + \boldsymbol{b} + \sqrt{2\alpha^{-1}\sigma_{rec}^2}\boldsymbol{N(0,1)}) \tag{2.8}$$

In this equation $\alpha = \frac{\Delta t}{\tau}$ and $N(0,1)$ is the standard normal distribution [8].

The network dynamics produce a set of output units $\boldsymbol{z}$ which read out non-linearity from the network,

$$\boldsymbol{z}_t = g(\boldsymbol{W}_{out}\boldsymbol{r}_t) + \boldsymbol{b}_{out} \tag{2.9}$$

where the activation function chosen is the logistic function $g(x) = \frac{1}{1+e^{-x}}$, returning values between 0 and 1. $\boldsymbol{W}_{in}$, $\boldsymbol{W}_{rec}$ and $\boldsymbol{W}_{out}$ are the three matrices representing input, recurrent and output connections, respectively.

Network dynamics for RC. Here is presented the dynamic for the reservoir computing used in the following analysis:

$$\boldsymbol{r}_t = (1-\alpha)\boldsymbol{r}_{t-1} + \alpha f(\boldsymbol{W}_{rec}\boldsymbol{r}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{u}_t) \tag{2.10}$$

$$\boldsymbol{z}_t = \boldsymbol{W}_{out}\boldsymbol{r}_t \tag{2.11}$$

where the activation function is $f(\cdot) = \tanh(\cdot)$.

2.3 Intrinsic dimension estimation

A dataset is often composed of a large amount of data represented by high-dimensional feature vectors. However, in many cases they can be embedded in lower dimensional spaces without information loss and this dimensionality reduction is often necessary for algorithmic purposes. Dimensionality reduction leads to important improvements in many situations and for many aims:

- Simplifying complexity;
- Eliminating information redundancy;
- Preventing overfitting;
- Allowing for data visualization
- Discovering hidden structures.

In real datasets, not all variables have the same importance; some are more significant than others. The minimum number of variables that can be used to represent the dataset is called intrinsic dimension (ID).

It is typically assumed that points in the dataset constitute and independent and identically distributed (I.I.D.) sample of a probability density $\rho(x)$. In principle, the ID could be easily estimated from the distances of nearest-neighbors in the dataset (the nearest points to a given point of the dataset), provided that $\rho(x)$ is uniform and sufficiently well-sampled. However, these two requirements are rarely met, making the task difficult in practice. Typical density distributions are inhomogeneous and, due to the so-called curse of dimensionality, undersampled whenever the ID is large. While the second problem is unavoidable, many different methods have been proposed to cope with the first. The one chosen for this thesis is the *Two nearest neighbours* (TWO-NN) [2]. This technique only considers the distance to the two nearest neighbors, minimizing the influence of dataset inhomogeneities in the estimation process.

If the density remains roughly consistent across the scale defined by the typical distance to the second neighbor, we can calculate the distribution and cumulative distribution of the ratio between the second distance and the first one. Surprisingly, these distributions are determined by the intrinsic dimension d rather than the density itself. At this juncture, we derive an equation that relates the theoretical cumulative distribution function F to d. By approximating F using the empirical cumulative distribution obtained from the dataset, we can then estimate the intrinsic dimension [2].

2.3.1 Two Nearest Neighbors estimator algorithm

Here are presented all the steps used to implement the TWO-NN algorithm.

For every point i of the dataset we call r_1 and r_2 the two shortest distances from i with other points, i.e., the distances of the first two nearest neighbours. Conventionally we set $r_0 = 0$. The volume of the hyperspherical shell between two successive neighbours $l-1$ and l is:

$$\Delta v_l = \omega_d (r_l^d - r_{l-1}^d) \tag{2.12}$$

where ω_d is the sphere with unitary radius and d is the dimension in which the points are embedded.

If the density ρ is uniformly constant around point i, all the Δv_l follow an exponential distribution with rate equal to ρ.

$$P(\Delta v_l \in [v, v+dv]) = \rho e^{-\rho v} dv \tag{2.13}$$

Consider now two shells Δv_1 and Δv_2 and define $R = \frac{\Delta v_i}{\Delta v_j}$. Applying the previous formula (Eq. 2.13) allows us to compute the probability density function (pdf) of R:

$$P(R \in [\bar{R}, \bar{R} + d\bar{R}]) = \int_0^\infty dv_i \int_0^\infty dv_j \rho^2 e^{-\rho(v_i+v_j)} \mathbf{1}_{\{\frac{v_i}{v_j} \in [\bar{R}, \bar{R}+d\bar{R}]\}} = d\bar{R} \frac{1}{(1+\bar{R})^2} \tag{2.14}$$

where **1** represent the indicator function. Dividing by $d\bar{R}$ we obtain the pdf for R:

$$g(R) = \frac{1}{(1+R)^2} \tag{2.15}$$

As we can see the pdf does not depend explicitly on d which is only a parameter in the definition of R, so in order to obtain a dependency on d we define $\mu \doteq \frac{r_2}{r_1} \in [1, +\infty)$. The relation between R and μ is the following:

$$R = \mu^d - 1 \tag{2.16}$$

and this equation allow us to find an explicit formula for the distribution of μ:

$$f(\mu) = d\mu^{-d-1} \mathbf{1}_{[1,+\infty]}(\mu) \tag{2.17}$$

from which we can obtain the cumulative distribution (cdf) by integration:

$$F(\mu) = (1 - \mu^{-d}) \mathbf{1}_{[1,+\infty]}(\mu) \tag{2.18}$$

These last two equations are independent of the local density but depend explicitly by the intrinsic dimension d.

The value of the intrinsic dimension d can be computed thanks to the following equation:

$$d = \frac{\log(1 - F(\mu))}{\log(\mu)} \tag{2.19}$$

As said before, since the cdf $F(\mu)$ is independent of ρ, the dimension estimation is not susceptible to density variations. Now we consider the set $S \subset \mathbb{R}^2, S \doteq \{(log(\mu), -log(1 - F(\mu)))\}$. S is contained in a straight line $l \doteq \{(x, y) | y = d \cdot x\}$ passing through the origin, whose slope is the intrinsic dimension d. We can thus obtain information about the intrinsic dimension by looking at the angular coefficient of this line.

Here is an algorithm to estimate the intrinsic dimension of a given dataset:

1. Compute the pairwise distance for each point in the dataset $i = 1, ..., N$;
2. For each point i find the two shortest distances r_1 and r_2;
3. For each point compute $\mu_i = \frac{r_2}{r_1}$;
4. Compute the empirical cumulative $F^{emp}(\mu)$ by sorting the values of μ in an ascending order through a permutation σ, then define $F^{emp}(\mu_{\sigma(i)}) \doteq \frac{i}{N}$;
5. Fit the points of the plane by coordinates $\{(log(\mu), -log(1 - F^{emp}(\mu))) \, | i = 1, ..., N\}$ with a straight line passing through the origin. [2]

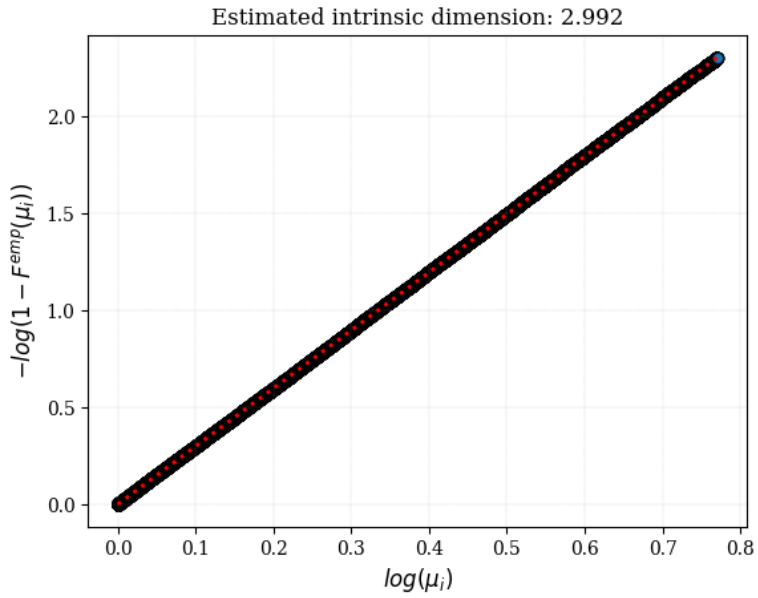

Figure 2.5: Example of an intrinsic dimension estimated from a multivariate normal distribution dataset.

2.3.2 Scale-dependent intrinsic dimension

The ID is, to some extent, a scale-dependent property. In (Figs. 2.6 and 2.7) there is a representation of a $3D$ multivariate normal distribution dataset with high variance ($\sigma^2 = 3000$) on the first and second direction and small variance ($\sigma^2 = 1$) on the third. When observed at a very small spatial scale this data are approximately two-dimensional; when observed at a larger scale, they are effectively three-dimensional. The locality of the TWO-NN estimator, obtained by considering only the two nearest neighbours of each point, can be very useful to understand how the ID varies with the scale, and distinguish in this way the number of "soft" directions.

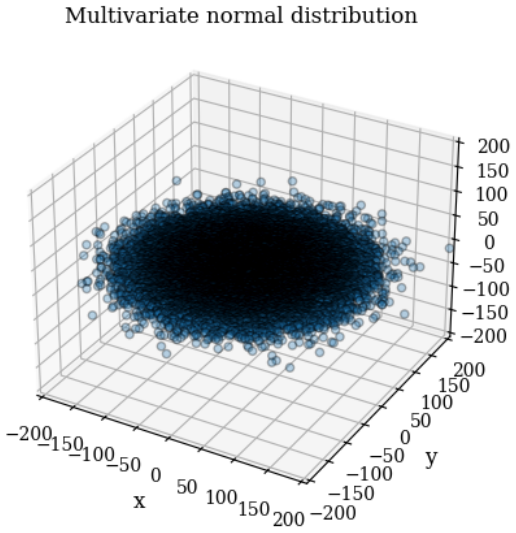

Figure 2.6: Representation of the multivariate normal distribution

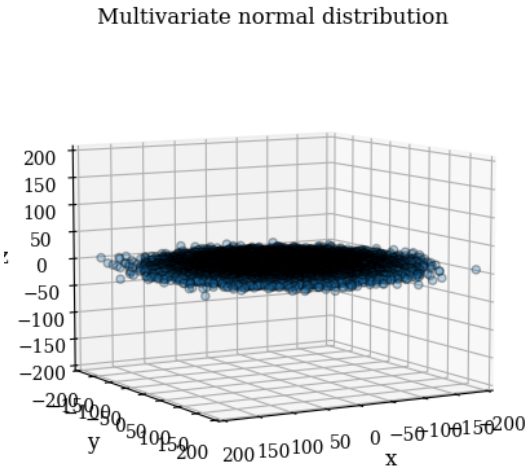

Figure 2.7: Representation of the multivariate normal distribution

To this aim, one considers random subsamples of the dataset and computes the ID on them. As the number of data points N is reduced, the average distance to the second neighbor increases, leading to a smaller intrinsic dimension. We can study the stability of the ID estimates in relation to changes in neighborhood size. The significant intrinsic dimension of the dataset can be determined by identifying a range of N values where N remains constant, indicating a plateau in the graph of $d(N)$. The value

of d at the plateau is the number of "soft", or relevant, directions in the dataset [2].

In (Fig. 2.8) we show this analysis for the aforementioned example of a $3D$ multivariate normal distribution dataset with high variance ($3000\sigma^2$) on the first and second direction. As said before, increasing the subsample size, there is a convergence on the value of $d = 3$. On the y-axis, the ID is represented with its error bar, computed as the standard deviation of the points in each sub sample. As we can see, the error is larger when the number of points is smaller because with fewer points there is greater variability in the distances between points. The last point, computed for the whole dataset, as we can expect, has the same dimension as the classic TWO-NN estimator (Fig. 2.5).

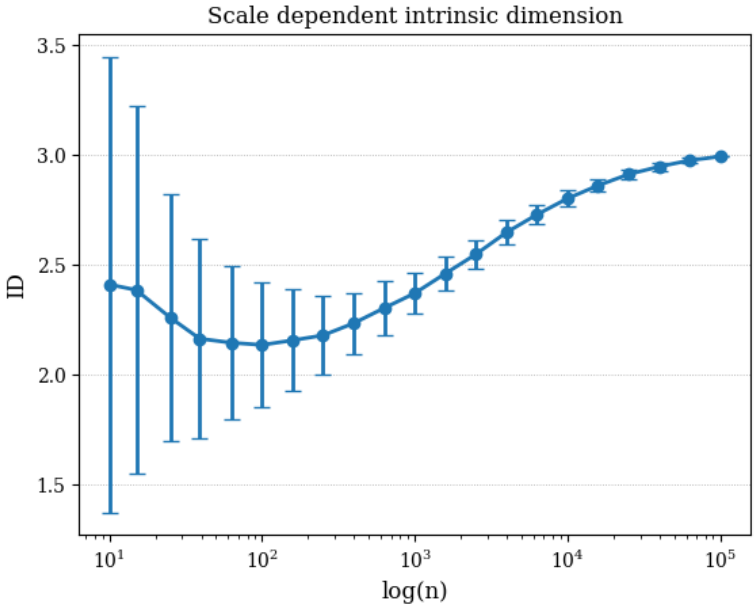

Figure 2.8: Scale dependent intrinsic dimension

Chapter 3

Results

The aim of this chapter is to analyze the network dynamics obtained by training the RNN with both BPTT and RC methods on 20 tasks. First of all, we will present a brief description of each task and how every input signal is mapped into an output one through the dynamics of recurrent networks presented in the previous chapter. After that, we will present the analysis for the ID estimation, focusing on two tasks.

3.1 Tasks and network architecture

Tasks. The tasks that will be considered in the following analysis represent idealized versions of tasks commonly used in neurophysiological studies of animals and crucial to our understanding of the neural mechanisms of cognition. All of them could be grouped into five principal families. The set of chosen task includes variants of memory-guided response, simple perceptual DM (decision making), context-dependent DM, multi-sensory integration, parametric working memory, inhibitory control (for example, in anti-saccade), delayed match-to-sample, and delayed match-to-category tasks [8].

The cognitive processes that are potentially involved in each task are shown in the following table (Tab. 3.1) with their names, an abbreviations used to indicate them and a brief description of their potential cognitive processes involved [8].

Task name	Abbreviation	Task family	Potential cognitive process involved	Reference
Go	Go	Go	Pro-response	N/A
Reaction-time Go	RT Go	Go	Pro-response	N/A
Delayed Go	Dly Go	Go	Pro-response, Working memory	Funahashi 89
Anti-response	Anti	Anti	Anti-response	Munoz 04
Reaction-time Anti-response	RT Anti	Anti	Anti-response	Munoz 04
Delayed anti-response	Dly Anti	Anti	Anti-response, Working memory	Munoz 04
Decision making 1	DM 1	DM	Decision making	Gold 07
Decision making 2	DM 2	DM	Decision making	Gold 07
Context-dependent decision making 1	Ctx DM 1	DM	Decision making, Gating mod 1	Mante 13
Context-dependent decision making 2	Ctx DM 2	DM	Decision making, Gating mod 2	Mante 13
Multi-sensory decision making	MultSen DM	DM	Decision making, Integrating mod 1 and 2	Raposo 14
Delayed decision making 1	Dly DM 1	Dly DM	Working memory	Romo 99
Delayed decision making 2	Dly DM 2	Dly DM	Working memory	Romo 99
Context-dependent delayed decision making 1	Ctx Dly DM 1	Dly DM	Working memory, Gating mod 1	N/A
Context-dependent delayed decision making 2	Ctx Dly DM 2	Dly DM	Working memory, Gating mod 2	N/A
Multi-sensory delayed decision making	MultSen Dly DM	Dly DM	Working memory, Integrating mod 1 and mod 2	N/A
Delayed match-to-sample	DMS	Matching	Working memory, Comparison	Miller 96
Delayed non-match-to-sample	DNMS	Matching	Working memory, Comparison	Miller 96
Delayed match-to-category	DMC	Matching	Categorization, Working memory, Comparison	Freedman 16
Delayed non-match-to-category	DNMC	Matching	Categorization, Working memory, Comparison	Freedman 16

Table 3.1: Names, abbreviations and potential congnitive processes for each task used in the analysis. [8] Tasks are grouped in their families.

Each task involves the presentation of one or two stimuli and a "Go" signal, and it can be divided into specific time periods. The *Fixation* (fix) epoch occurs before any stimulus is presented. This is followed by the *Stimulus Epoch 1* (stim1). If two stimuli are presented sequentially, the time between

them is called the *Delay Epoch*, and the second stimulus is shown during the *Stimulus Epoch 2* (stim2). The period during which the network is expected to respond is called the *"Go"* epoch. The durations of the fixation, stim1, delay1, stim2, and go epochs are denoted as T_{fix}, T_{stim1}, T_{delay1}, T_{stim2}, and T_{go}, respectively.

Network architecture. The network architecture was designed to be as general as possible to be suitable for every task. As shown in (Fig. 3.1), the network is composed by an input layer, a recurrent layer and an output layer.

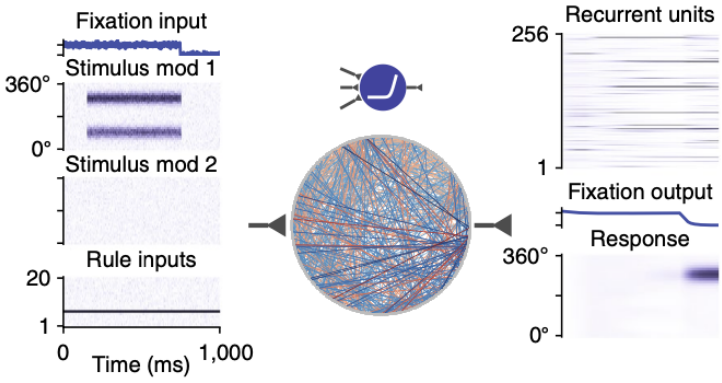

Figure 3.1: Example of a fully-connected RNN (picture taken from [9]) composed by input, recurrent and output parts. In the middle (recurrent) only 1% of connections are shown.

The input layer consists of 85 units. For every task the network receives in input three different types of signals: *fixation*, *stimulus* (including two stimulus modalities) and *rule*. Each input signal is affected by Gaussian noise:

$$\mathbf{u} = (\mathbf{u}_{fix}, \mathbf{u}_{mod1}, \mathbf{u}_{mod2}, \mathbf{u}_{rule}) + \mathbf{u}_{noise} \tag{3.1}$$

In which, $\mathbf{u}_{noise} = \sqrt{\frac{2}{\alpha}}\sigma_{in}\mathbf{N}(0,1)$, where $\sigma_{in} = 0.01$ is the input noise strength and $\mathbf{N}(0,1)$ is the standard normal distribution.

The *fixation* input $\mathbf{u}_{fix}$ indicates whether the network should "fixate" or respond. Its value is typically at the highest value of 1 when the network should fixate and decreases to 0 when the network has to respond. A decrease of that fixation signal gives a "Go" signal to the network.

The *stimulus* inputs $\mathbf{u}_{mod1}$ and $\mathbf{u}_{mod2}$, involves two different stimulus modalities (for example, a visual and an auditory stimulus), each represented by a ring of input units that encodes a one-dimensional circular variable that could be a motion direction or a color on a color wheel, representing a one-dimensional circular variable described by the degrees around a circle. Each ring contains 32 units, each responding preferentially to stimulus directions uniformly spaced from 0 to 2π. When a stimulus in direction ψ is presented, the i-th unit (with a preferred direction ψ_i) has an activity

$$u_i = \gamma \cdot 0.8e^{\left[-\frac{1}{2}\left(\frac{8|\psi-\psi_i|}{\pi}\right)^2\right]} \tag{3.2}$$

In this equation, γ is the strength of the stimulus.

For every task, the *rule* input $\mathbf{u}_{rule}$ is activated to instruct the network on which task it is supposed to perform (among a set of 20 possible tasks). Normally is a one-hot vector, which means that the rule input unit corresponding to the current task is activated at 1, while other rule input units remain at 0.

Overall, the number of input units used are:

$$N_{input} = \underbrace{1}_{fixation} + \underbrace{32 \times 2}_{mod1,2} + \underbrace{20}_{rule} = 85 \tag{3.3}$$

The recurrent layer corresponds to $N_{rec} = 256$ recurrent units, with incoming connections from the input layer and outgoing connections to the output one.

The output layer consists of 33 units. This layer encodes a fixation output unit and a group of motor units encoding the response direction as a one-dimensional variable on a ring of possible outputs (for example, saccade direction[1], reach direction[2]). Similarly as described before, the output ring units encode the response directions using similar tuning curves to the ones used for the input rings. The response direction ψ_i is encoded as

$$\hat{z}_i = 0.8e^{\left[-\frac{1}{2}\left(\frac{8|\psi-\psi_i|}{\pi}\right)^2\right]} + 0.05 \tag{3.4}$$

In addition, there is a fixation output unit z_{fix}, which has his maximum value of 1 before the response and his minimum value of 0 once the response is generated. Overall, the number of output units is:

$$N_{output} = \underbrace{1}_{fixation} + \underbrace{32}_{output} = 33 \tag{3.5}$$

[1]Saccade direction refers to the rapid and involuntary movement that an eye can do. [8]

[2]Reach direction refers to the movement an arm or limb extends to reach or interact with an object. [8]

3.2 Intrinsic dimension estimation

We will focus analysis on two example tasks (a simple and a complex one), but all the considerations that will be made can be extended to the others. These two chosen tasks are the *Anti-response* and the *Context-dependent delayed decision making 1*. A brief description of them can be found in the table (Tab. 3.1). In particular, the second task is computationally more complex than the first, and this choice was made to enable a first and rapid comparison for verifying the hypothesis that a more computationally complex task induced a dynamics with larger ID.

3.2.1 Anti-response task

Anti-response is part of the *Anti task Family* which also includes the *RT Anti* and *Dly Anti* tasks. In this task, the stimulus can be randomly shown in either modality 1 or 2 and the response should be made in the opposite direction of the stimulus. In this case:

$$T_{stim1} \sim U(500, 1500) \tag{3.6}$$

where $U(t_1, t_2)$ is a uniform distribution between t_1 and t_2 and the unit is millisecond (ms).

In (Fig. 3.2) we show the input signals and the output ones for the network trained with BPTT. Only one trial is shown with a duration of 2 s. The first 66 rows correspond to the input. The first row corresponds to the fixation input, which is turned off only after the activation of the "Go" signal, as mentioned before. Rows 2-65 represent the stimulus input. For the first trial, the stimulus is in mod2. The last row (66) corresponds to the rule signal, which selects which task to activate (we do not show the remaining 19 rule input units, which are held constant at 0 during the whole task). The last 33 rows correspond to the output units. Row 67 is the fixation output, while rows 68-99 are the response outputs. In (Fig. 3.3) we show the input and output plot as well for the network trained with RC. The description is almost the same, the only difference is that in this case the stimulus is in mod1 and in the input part there are 85 rows. In fact rows 66-85 represent the rule signals and only the row that correspond to the *Anti* task is activated.

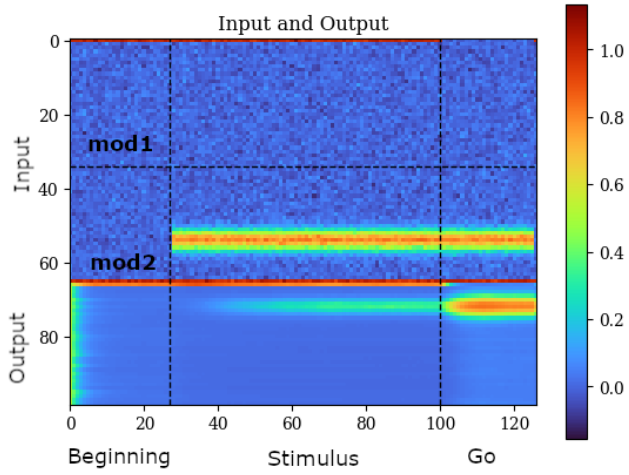

Figure 3.2: Input and output signals for the network trained with BPTT.

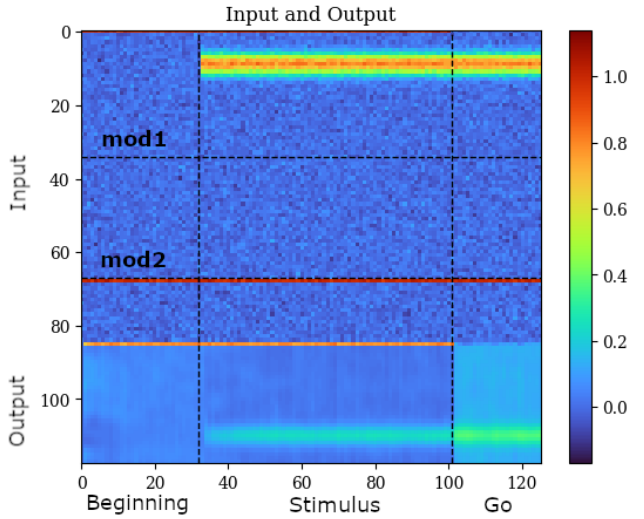

Figure 3.3: Input and output signals for the network trained with RC.

The outputs for both BPTT and RC are shown in (Figs. 3.4 and 3.5) respectively. In the first and second panels of these pictures there is a comparison of the output of the trained network and the ideal one. Visually, for BPTT, the two are similar so the network dynamics have worked correctly. Looking at the third panel of (Fig. 3.4) an error plot is shown as the difference between the two simulations. It can be easily noticed that for the most part of the trials the error is near to zero except at the end

of each trial (and beginning of the next one), and at the start of the go signal. This trend occurs because at the beginning of each trial, the network needs a finite time to relax to the ideal output (0). Analogously, a finite amount of time is needed to respond to the "Go" signal.

Otherwise, the real output obtained with RC (Fig. 3.5) is different respect to the one obtained with BPTT. In fact, RC performs well on the "Go" signal (which is blue in the picture) but only fairly well on the angle: the width of the angle is reduced, but the positions are correct. This is why, looking at the third panel of the plot, the error is never almost close to zero; there is the effect of the relaxation time discussed before and the difference in width for the angles in the ideal and real cases.

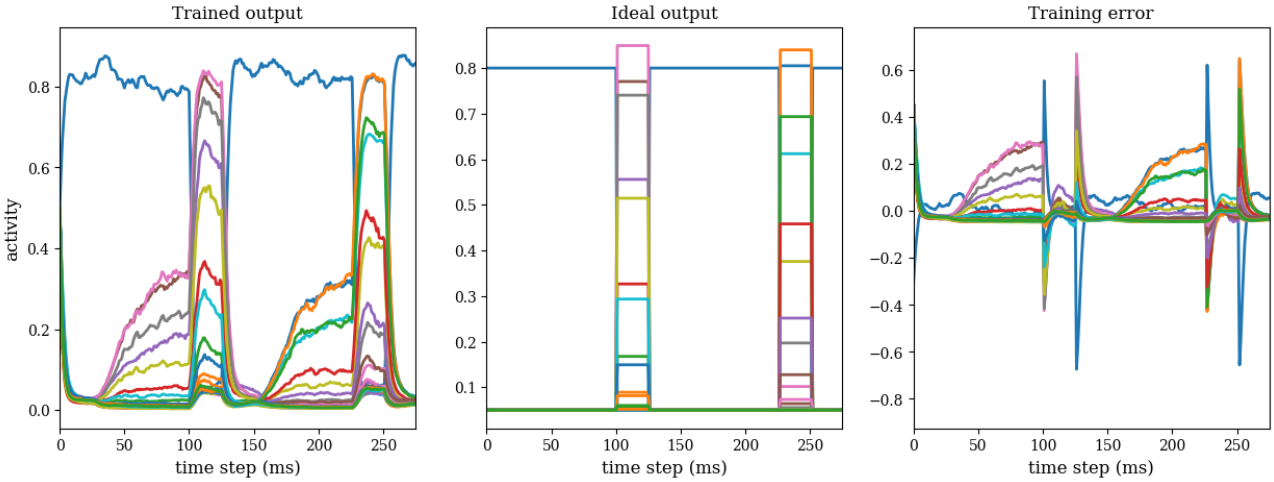

Figure 3.4: In the first and second panels there is a comparison between trained and ideal output for *Anti-response* task trained with Back-propagation through time. Only two trials are shown. In the third one an error plot.

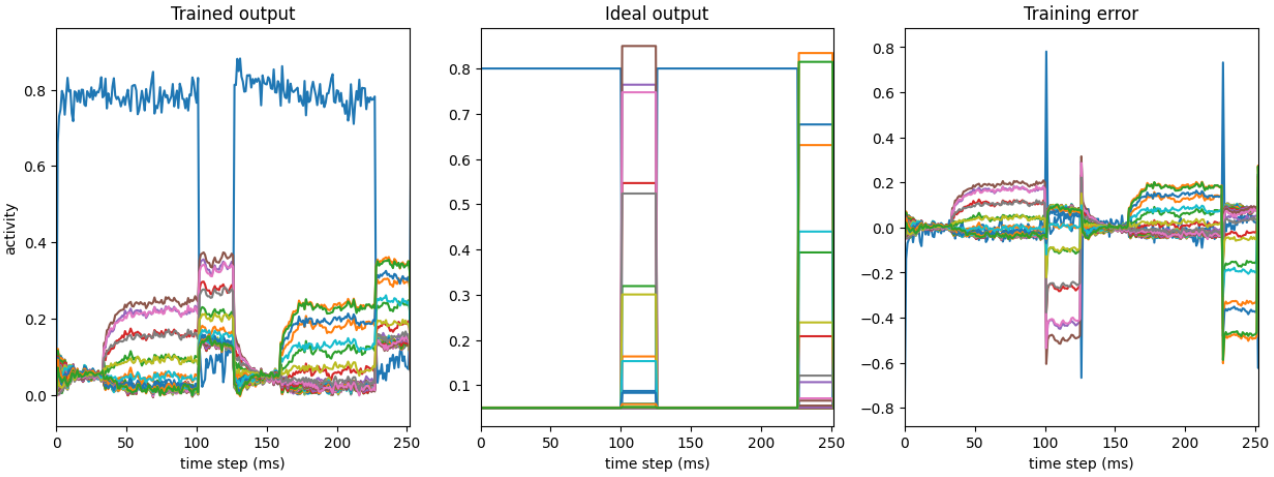

Figure 3.5: In the first and second panels there is a comparison between trained and ideal output for *Anti-response* task trained with Reservoir Computing. Only two trials are shown. In the third one an error plot.

Looking at (Figs. 3.2 and 3.3) it is possible to divide every trial in four parts:

- **Beginning**: from the beginning of the fixation input until the start of the stimulus signal;
- **Middle**: from the start of the stimulus signal until the start of the Go signal;
- **Final**: from the start of the Go signal until the end of the trial;
- **Total**: the whole trial.

For each of these four parts, the intrinsic dimension (ID) of trajectories of the recurrent part has been computed in order to investigate how it changes along the evolution of the signal. The estimation was done for every trials of the network dynamics. An overview for the BPTT is presented in (Fig. 3.6).

The estimated dimension are, for the four regions of the trial: 12.08, 12.92, 11.34, 12.44, respectively. The intrinsic dimension has been computed even for the task trained with RC and the results for the four regions of the trial are: 37.45, 41.20, 35.97, 43.58. These dimensions are much higher than the others, but this is not unexpected. The reason must be found in the network architecture itself. In fact, as discussed in the previous chapter, in RC all recurrent weights remain constant during training and encode a lot of variables, not just the few related to the given task. This does not happen with BPTT because the weights of the network are trained, and it is the recurrent dynamics itself that efficiently encodes the task variables.

Considering the fact that the tasks involve simple stimuli and responses, such as saccade direction and reach direction, the input and output are expected to encode a one-dimensional circular variable (such as motion direction or color on a color wheel) [8]. So, the estimated dimensions seems to be higher than what is expected. For that reason, using again the TWO-NN ID estimator, we have calculated the scale dependent intrinsic dimension for every part of the trial which is shown in (Fig. 3.7).

In those plots, it is clear that the trend for the dimension of each part is lower than before. For BPTT the results are: 4.04, 3.80, 3.44 and 3.83, respectively. For RC the results are: 23.52, 26.37, 22.52, 28.40. These values were computed as the arithmetic average of the points that fall on the plateau. The scale-dependent estimation worked well for the BPTT method because it gives low dimensions, close to the expected value of two. As for the RC method, the scale-dependent estimation gives values higher than BPTT, consistent with the fact that the RC dynamics is not low-dimensional.

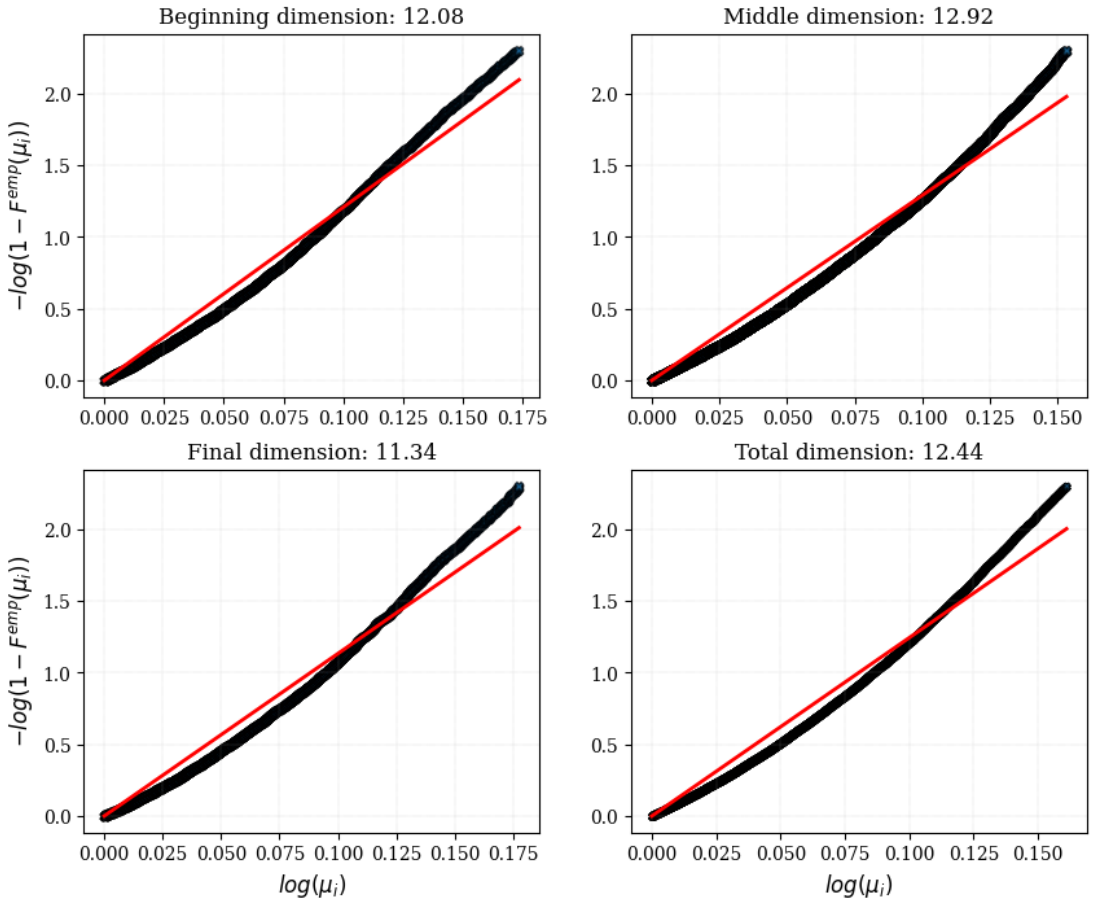

Figure 3.6: Intrinsic dimension estimated with TWO-NN estimator for the ranges described before. The plots refer to BPTT network.

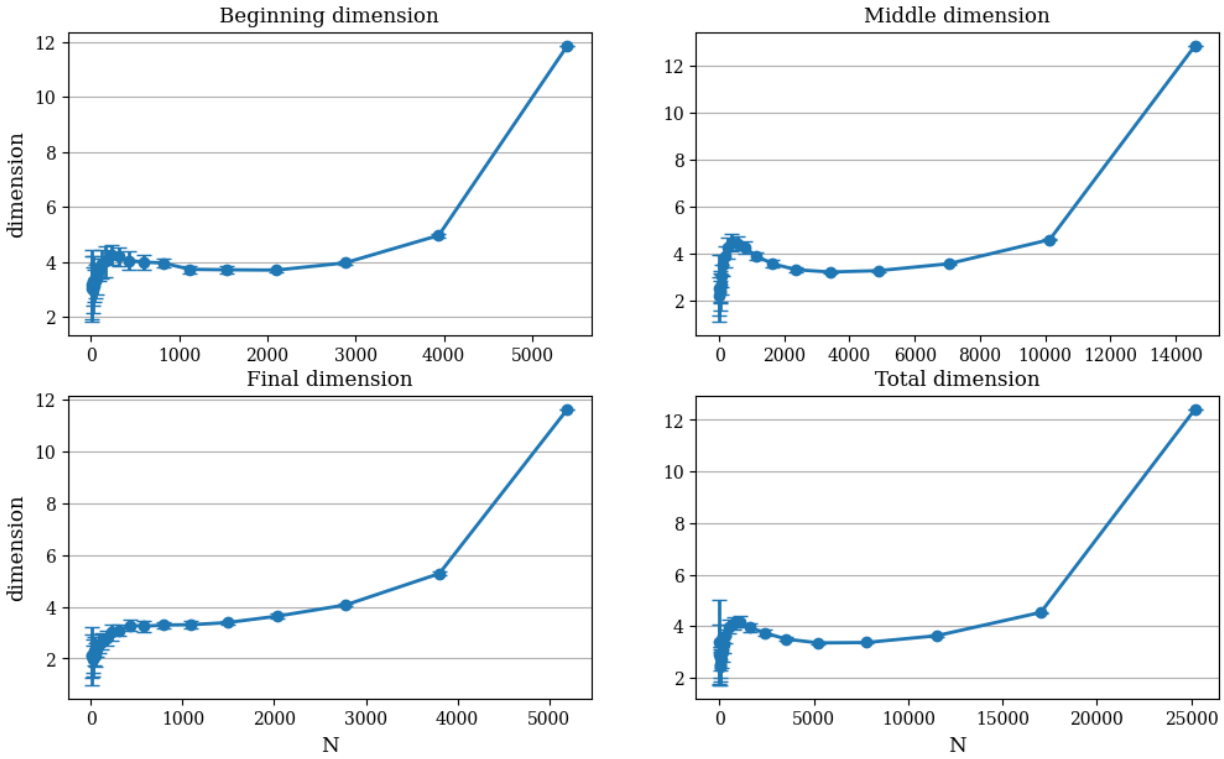

Figure 3.7: Scale dependent intrinsic dimension. The plots refer to BPTT network.

Focusing now on BPTT, another way to support the hypothesis that the real dimension is the one obtained with the scale dependent analysis is by using Principal Component Analysis (PCA) [3]. With that analysis it is possible to reduce the number of dimensions to make sense of the estimates provided by TWO-NN estimator. In figures (Figs. 3.9 and 3.10) we can see how PCA dimensions was set to 2 or 3 respectively. Already in 2D, and even more in 3D, it is clear that the trajectories lie in a well-structured geometric object which seems to be a two-dimensional manifold. The interesting thing is that we observe a 2D "twisted" manifold, as we can see in the 3D plot. The actual dimension around 3.5 is probably due to non-zero variance in directions orthogonal to the manifold. This is consistent with the fact that the TWO-NN ID estimator has predicted a dimension higher than three.

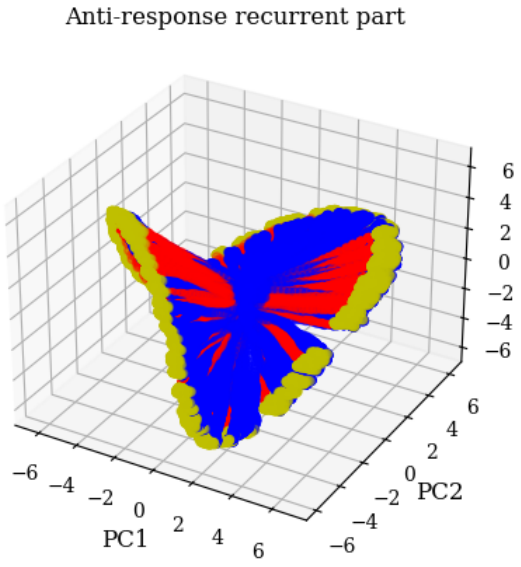

Figure 3.8: Principal component analysis on the whole recurrent part. In red it is represented the *beginning* part, in blue the *middle* part and in yellow the *final* one.

In figure (Fig. 3.11) we show the so-called "explained variance ratio". The explained variance ratio quantifies how much of the total variance in the original dataset is accounted for by each principal component. It is computed as the ratio of the eigenvalue of a principal component to the sum of the eigenvalues of all principal components. We

[3]Principal Component Analysis (PCA) is a statistical technique used to simplify and interpret complex data sets by reducing their dimensionality. It identifies the directions of maximal covariance in the data and represents them in a smaller number of dimensions called principal components, allowing for easier visualization and analysis. PCA is commonly used in various fields such as data analysis, image processing, and pattern recognition.

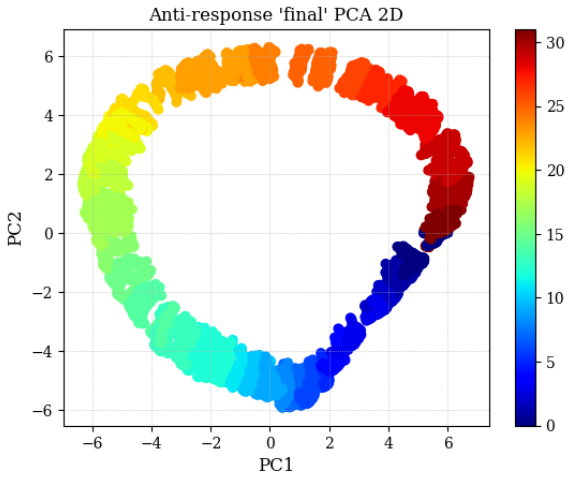

Figure 3.9: PCA 2D referred to the final part of the trial.

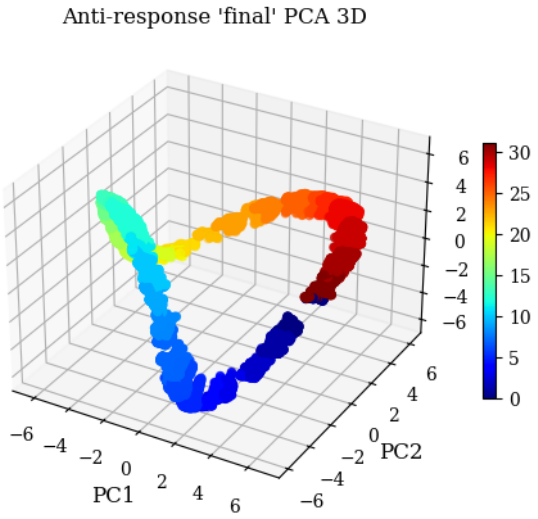

Figure 3.10: PCA 3d referred to the final part of the trial.

computed the variance for each component of the PCA analysis, showing in (Fig. 3.11). In the right panel of the plot there is a zoom on the first 20 principal components. We can see that there is a high variance for the first 5 PCs that fell off to zero from almost the 7-th PC until the end. That means that the majority of the information on the dataset is contained in the first principal components. Yet, the fact that at least ≈ 5 PCs are needed suggests that the two-dimensional manifold is highly twisted, so that the dimension of an approximate linear embedding (≈ 5) differs from the intrinsic dimension.

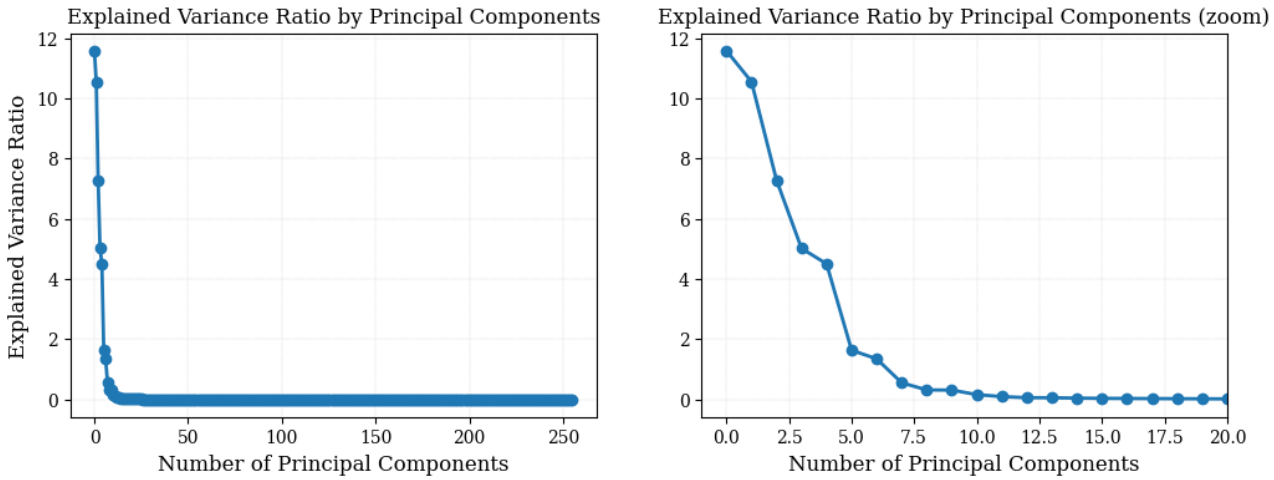

Figure 3.11: *Anti* task explained variance ratio for principal components.

3.2.2 Context-dependent delayed decision making 1 task

This task is part of the *Dly DM Family* which includes *Dly DM 1*, *Dly DM 2*, *Ctx Dly DM 1* and *Ctx Dly DM 2*.

The task is inspired from classical perceptual DM tasks based on random-dot stimuli with two input stimuli to model momentary motion evidence toward the two target directions. The two stimuli are shown briefly and are separated by a short delay period. Another short delay period follows the offset of the second stimulus. For this task:

$$T_{delay1} \sim U(\{200, 400, 800, 1600\}) \tag{3.7}$$

$$c \sim U(\{-0.32, -0.16, -0.008, 0.008, 0.16, 0.32\}) \tag{3.8}$$

and $T_{stim1} = T_{stim2} = 300$.
In (Eq. 3.8), c represents the coherence, which determines the overall strength of the net evidence toward the direction represented by stimulus 1 in each trial.

In (Fig. 3.12) we show the input signals and the output ones. Similarly to before, only the first trial is shown with a duration of 2s. The first 66 rows are related to the input signal, while the last 33 are for the output one. The first and the last rows of the input part (1, 66) corresponds to the fixation input and rule signal respectively while in the middle of them there is the stimulus input which is both in mod1 and mod2. In (Fig. 3.13) we show the same plot but for the task trained with RC. The description is the same but we can see that in the rows of the rule signal (66-85) only the one representing the *Context-dependent delayed decision making 1* task is activated.

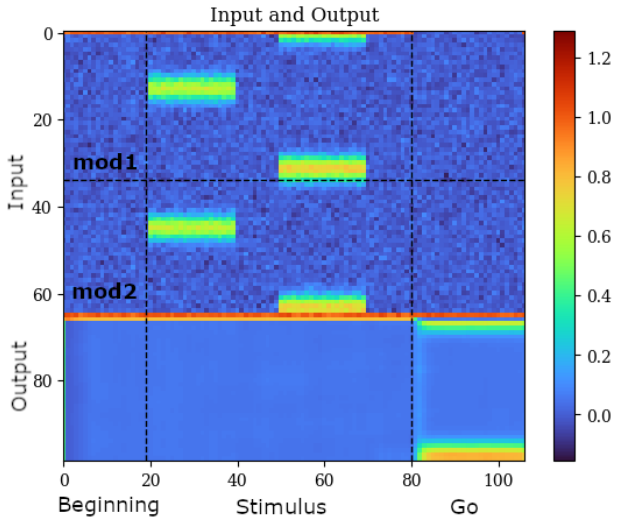

Figure 3.12: Input and output signal for the network trained with BPTT.

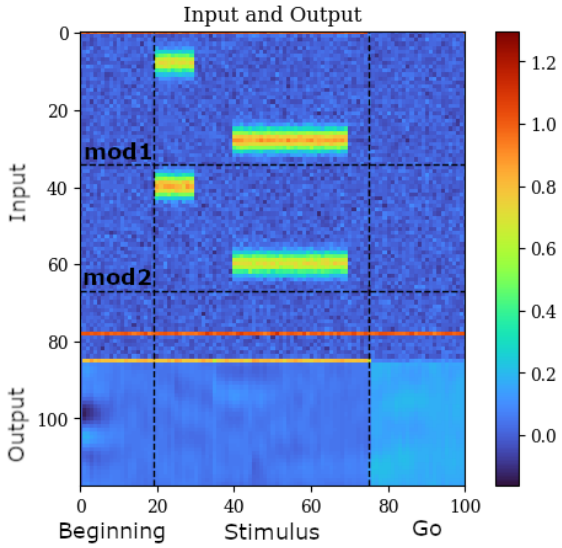

Figure 3.13: Input and output signal for the network trained with RC.

The outputs for both BPTT and RC networks are shown in (Figs. 3.14 and 3.15). In the first and second panels of both pictures there is a visual comparison between real and ideal output. In the case of BPTT network they are very similar, so the network has worked correctly. This is clear even looking at the third panel (Fig. 3.14) in which an error plot is represented. In fact, we can see that the difference between real and ideal data is almost around zero in every point, except for the end of each trial (and beginning of the next one) and at the start of the go signal. This is because the network needs a finite time to relax to the ideal output (0), as discussed before.

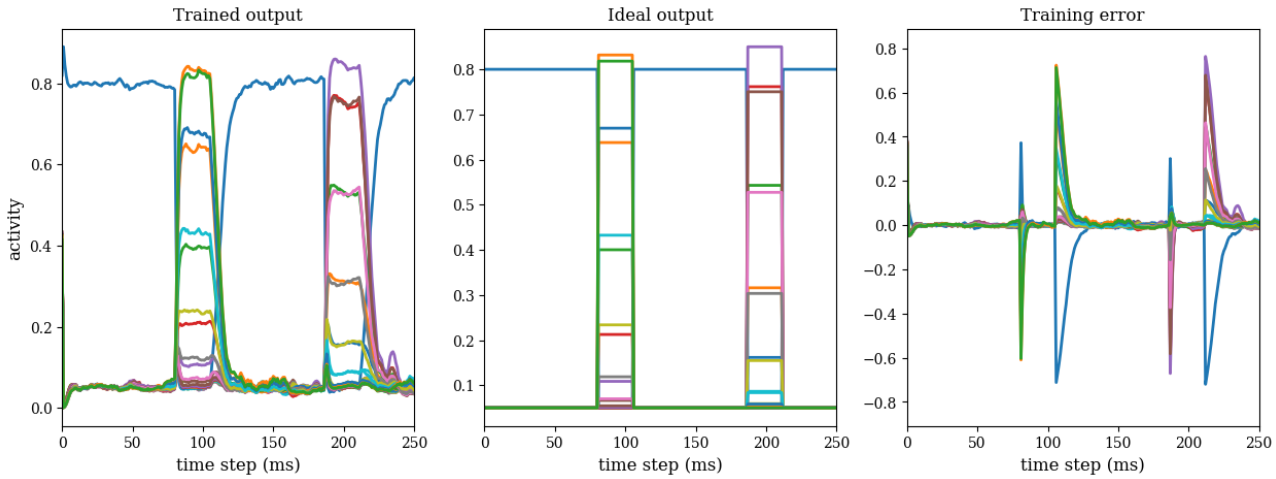

Figure 3.14: Comparison between trained and ideal output for *Context-dependent delayed decision making 1* task trained with Back-propagation through time. Only two trials are shown.

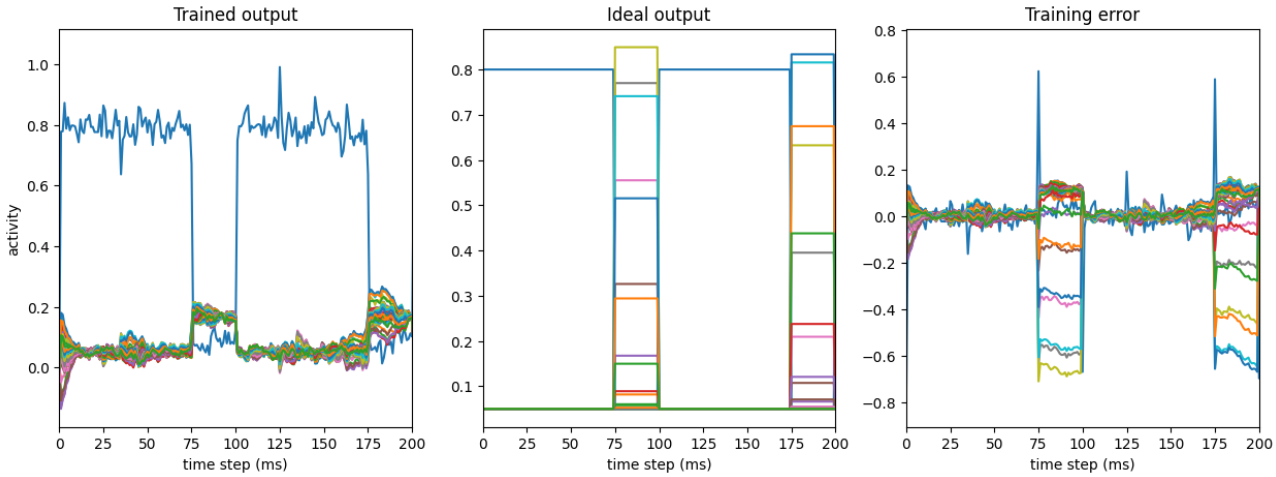

Figure 3.15: Comparison between trained and ideal output for *Context-dependent delayed decision making 1* task trained with Reservoir computing. Only two trials are shown.

Regarding the RC network, we observe that the "Go" signal (depicted in blue in the first and second panels) performs well. However, concerning the angles, it is difficult to distinguish their positions. Consequently, in the comparison between the real and ideal cases (third panel), the error never approaches zero in the intervals between the end of one trial and the beginning of the next.

For this task, and generally for more complex ones, RC is not as accurate as BPTT in predicting the value of the angles. This is due to its dynamics; the fact that recurrent weights remain constant throughout training can lead to a loss of accuracy, resulting in poorer performance.

As done before, referring to (Figs. 3.12 and 3.13) it is possible to divide each trial into the **beginning**, **middle**, **final** and **total** parts. In order to investigate how the intrinsic dimension changes along the evolution of the signal, we used the TWO-NN estimator to compute the dimensions. The results for BPTT, as we can even see in (Fig. 3.16) are: 4.74, 5.00, 4.62 and 5.06. Instead, the results for RC are: 47.67, 21.91, 36.59 and 28.10.

Even for this task we expect lower dimensions (the network have to encode at least two angles), so we have implemented the scale dependent intrinsic dimension, for each of the four parts. An overview is presented in (Fig. 3.17). With this method, for BPTT, the estimated dimensions are: 1.82, 2.02, 1.96, 2.06, all around the expected value of 2. Otherwise, for RC: 38.45, 12.22, 24.29 and 15.98. These estimates are higher than the ones obtained with BPTT due to the RC architecture discussed for the

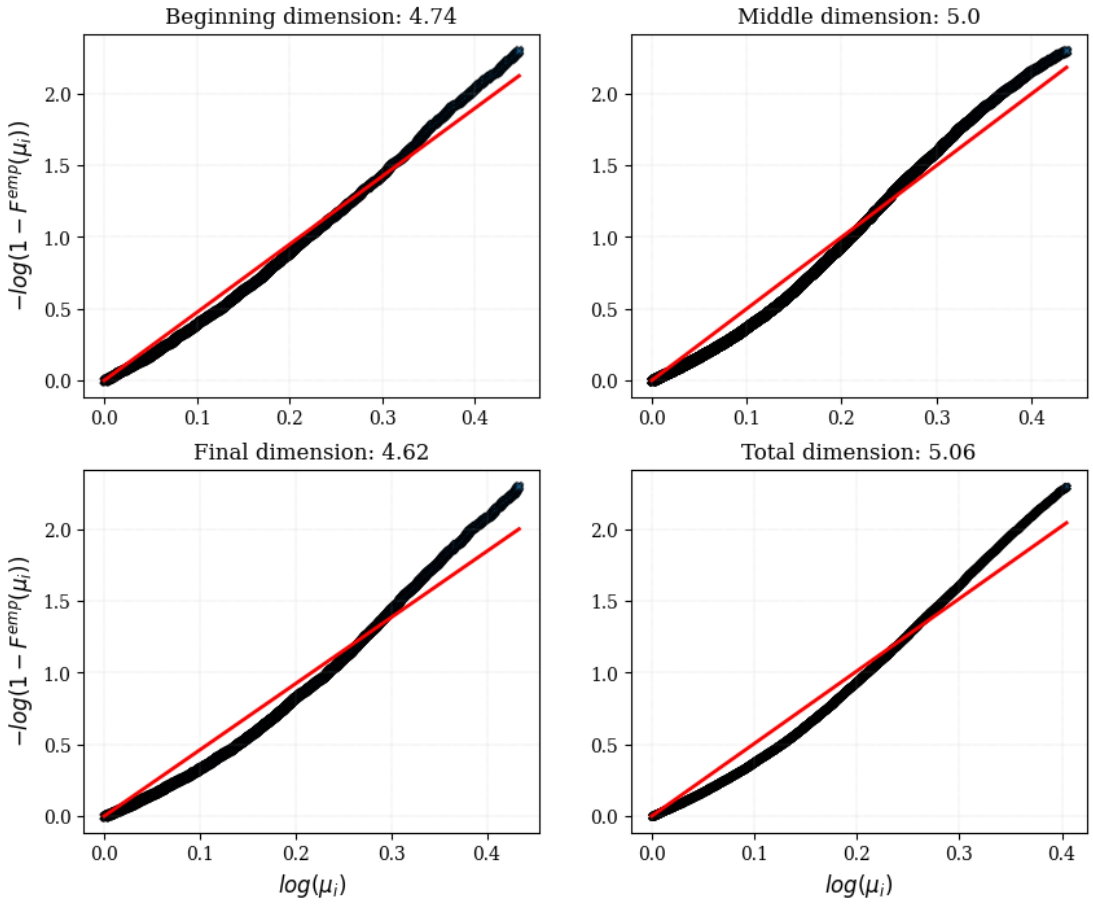

Figure 3.16: Intrinsic dimension estimated with TWO-NN estimator for the ranges described before. The plots refer to BPTT network.

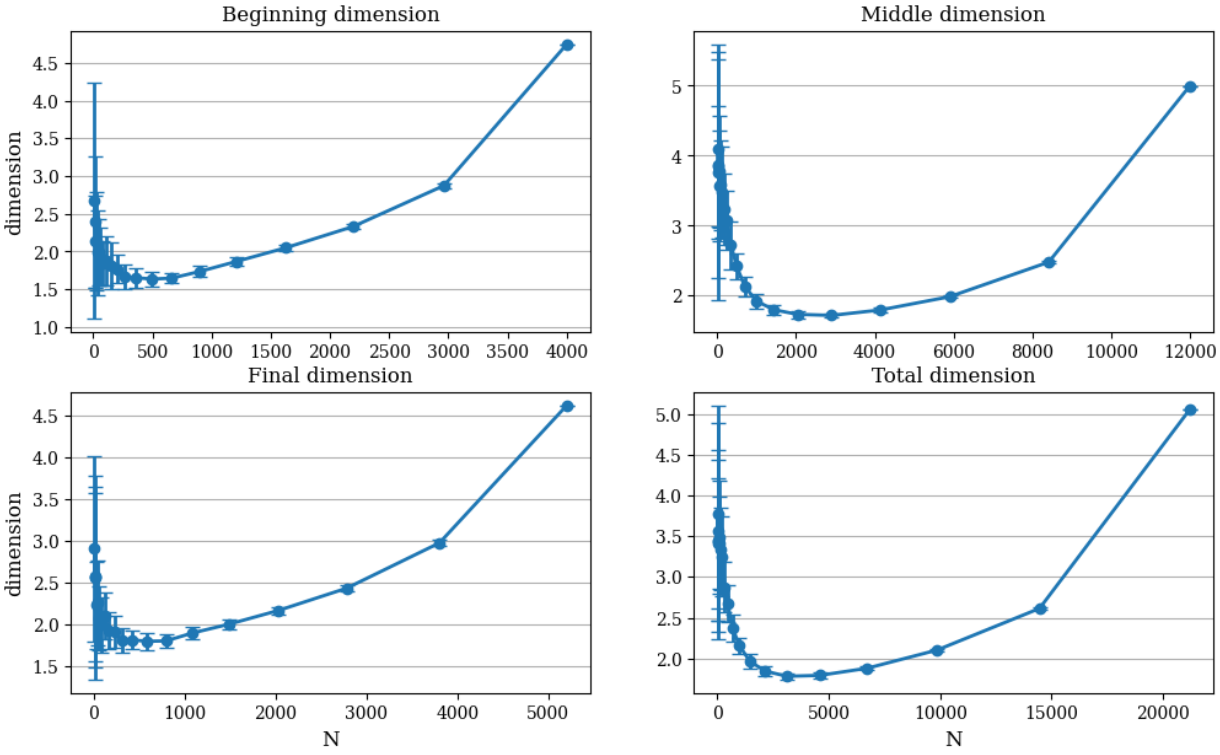

Figure 3.17: Scale dependent intrinsic dimension. The plots refer to BPTT network.

previous task.

Looking at the PCA plots for BPTT (Fig. 3.18), we can see the arrangement of the points in the PC space. As the previous task we have imposed the PCA dimension at 2 and 3 (Figs. 3.19 and 3.20). Even in that case we can see that both in 2D and 3D the trajectories lie in a structured geometric object that is a two-dimensional manifold that seems to be curved and twisted when passing in a 3D representation. Locally, the trajectories have a dimension of approximately 1.5 because the network tends to "orthogonalize" the representation of different outputs and this causes the dimensions to be greater than two.

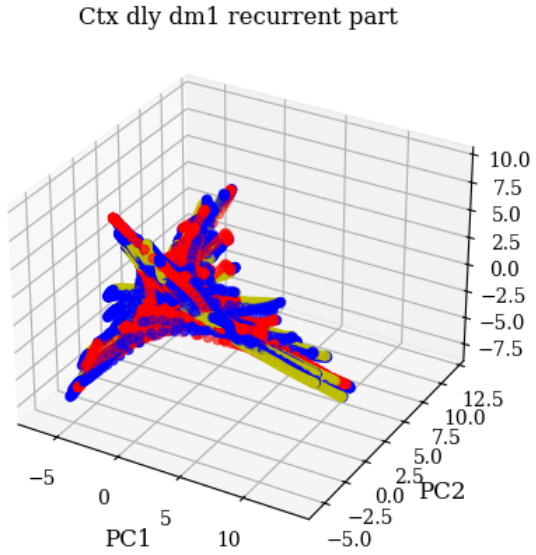

Figure 3.18: Principal component analysis on the whole recurrent part. In red it is represented the *beginning* part, in blue the *middle* part and in yellow the *final* one.

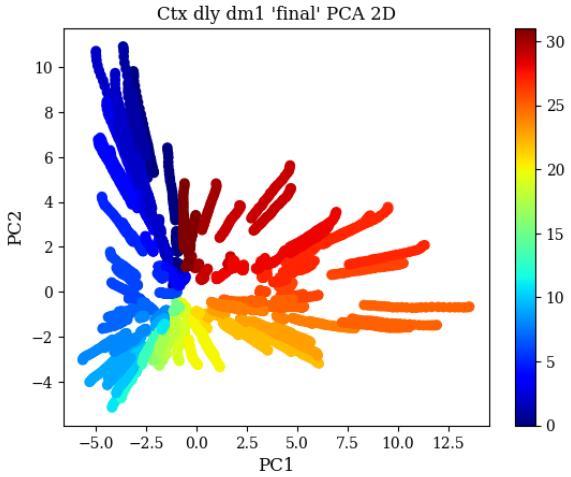

Figure 3.19: PCA 2D referred to the final part of the trial.

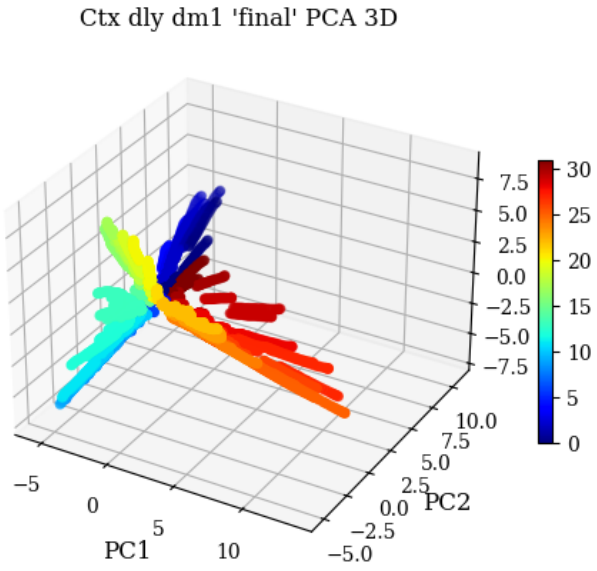

Figure 3.20: PCA 3D referred to the final part of the trial.

Similarly to the previous task, the "explained variance ratio" plot (Fig. 3.21) was produced to understand how much of the variance is accounted for by each principal component. In the second panel of the figure, we can see that the majority of the variance is captured by the first 8 or 9 PCs, dropping to near zero from the 10-th onward. This suggests that for this task, which is more complex than the previous one, the two-dimensional manifold is much more highly twisted. Consequently, the dimension

of an approximate linear embedding differs from the intrinsic dimension estimated.

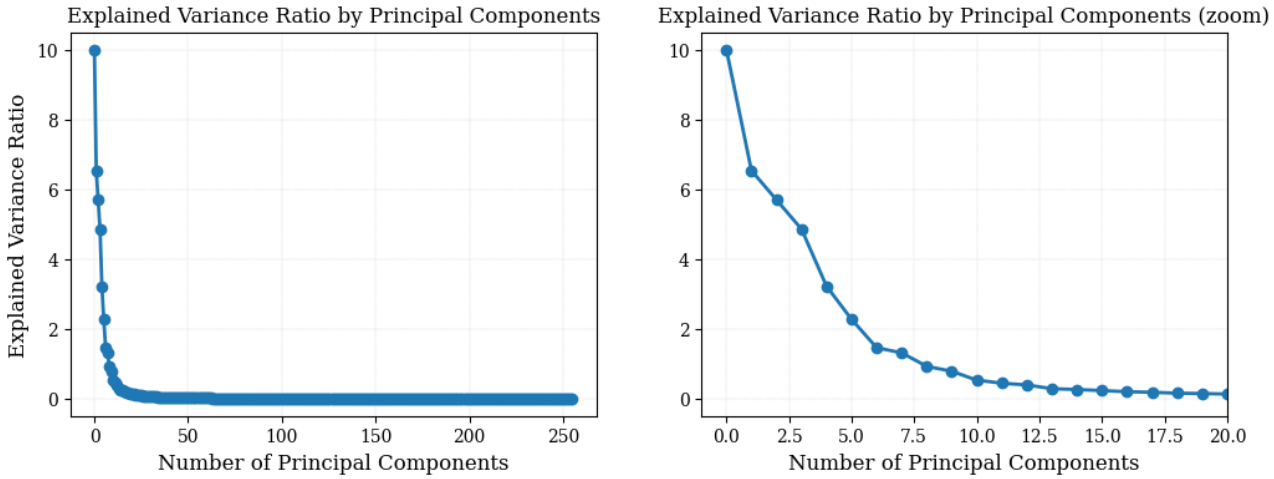

Figure 3.21: *Context-dependent delayed decision making 1* task explained variance ratio for principal components.

3.3 Final results

Looking at the two tasks described as examples, it is clear that BPTT is more effective in training the networks for the kind of tasks we wanted to study. In fact, for our purposes, the RC network still has some limits. It seems obvious that, when addressing a specific modeling task, a specific reservoir design that is adapted to the task will lead to better results than a naive random creation [6]. This could be observed in the comparison between the shapes of the output signals in both networks (Figs. 3.4 and 3.5), (Figs. 3.14 and 3.15). The following results are all referred to the tasks trained with BPTT.

The analysis made for the two tasks has been repeated for all of the 20 tasks. An overview is presented in the following plot (Fig. 3.22). In this picture we have reported four dimensions for each task, corresponding to the four trial phases (beginning, middle, final, total).

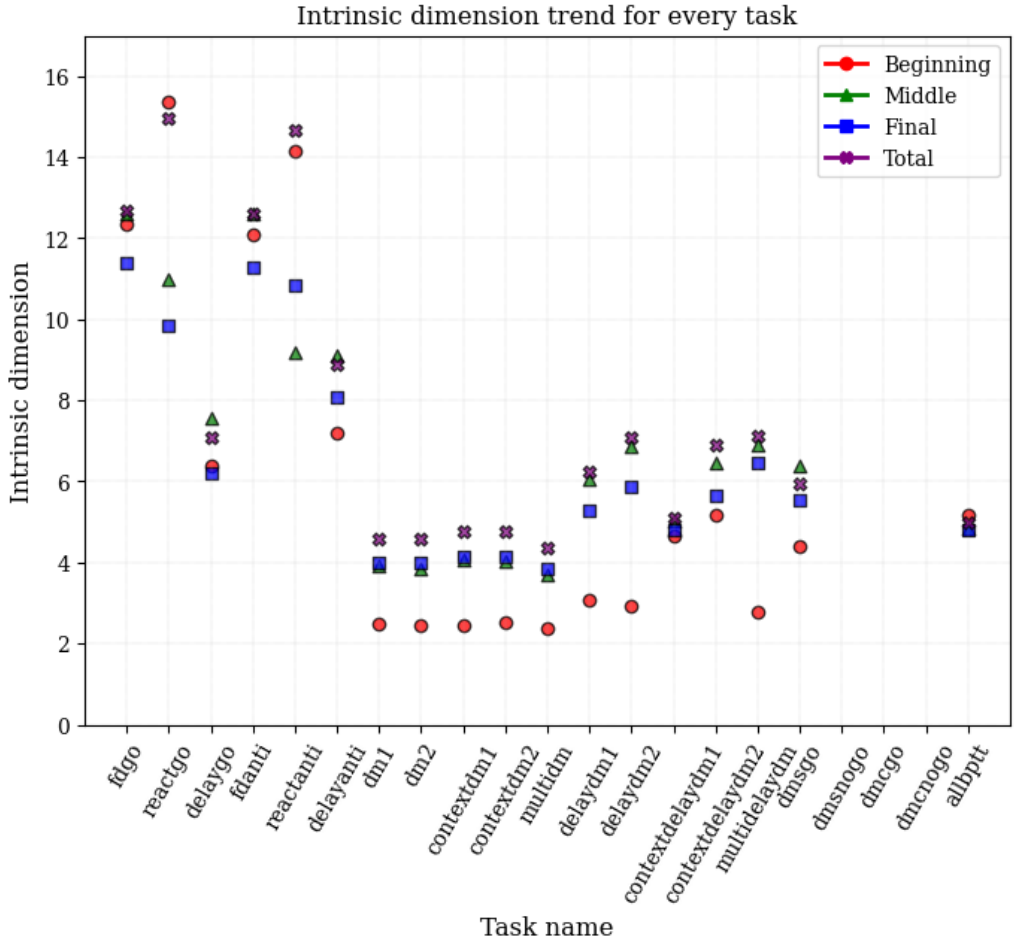

Figure 3.22: BPTT ID recap.

As we can see, for three tasks (*dmsnogo*, *dmcgo*, *dmcnogo*) there are no points because the network did not perform well for them, so we decided to exclude them from the analysis. In the last column of both (Figs. 3.22 and 3.23) there is the so called *All tasks*, which was trained to simultaneously solve all tasks.

All dimensions are in contrast with the hypothesis that they should be around a value of two, in fact they are all more than two. This fact was noticed even before and here is proved for every task. For that reason it is better to study the scale dependent intrinsic dimension whose results are shown in (Fig. 3.23).

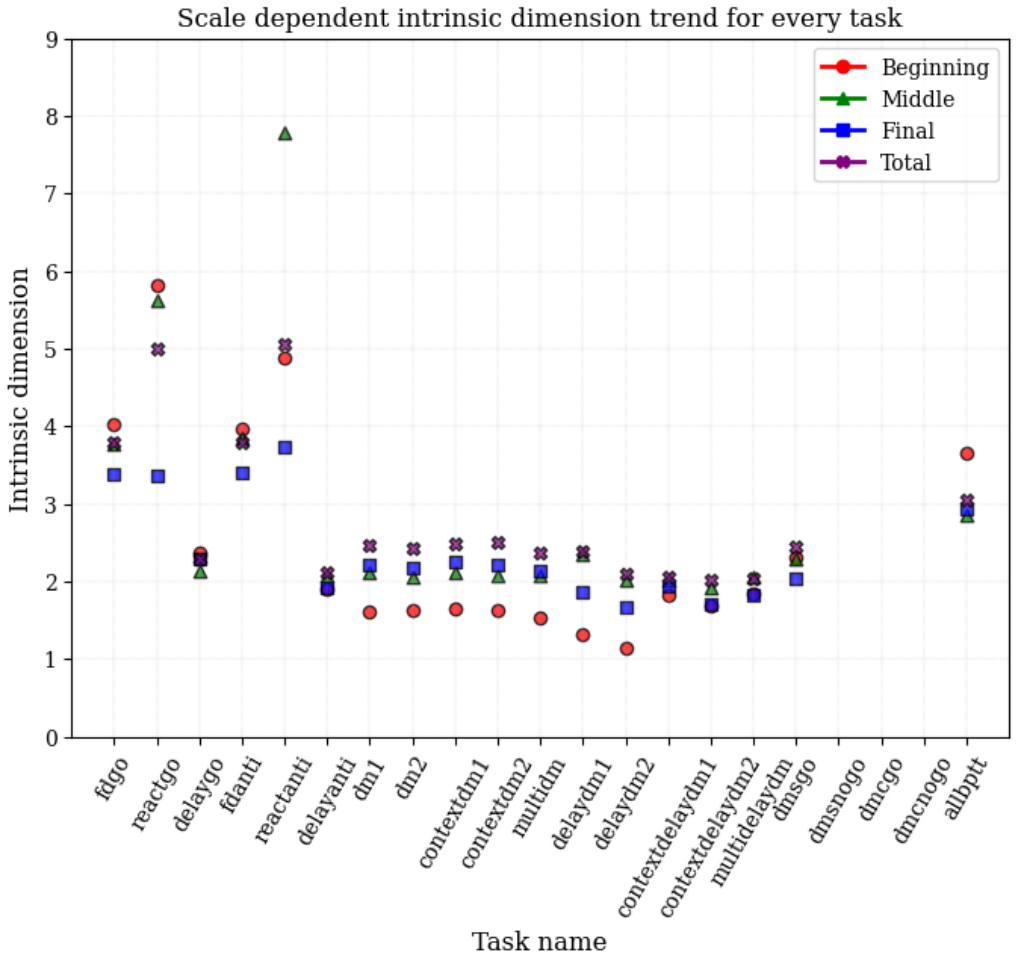

Figure 3.23: BPTT scale ID recap.

Focusing on this picture, we can see a general decrease of every dimension which tends to stabilize around a value of two for the most of the tasks. What we can see in this picture and in the previous one is a trend for trial phases. In fact the *beginning* dimensions are lower than the others because the time from the beginning of the fixation input until the start of the stimulus signal is generally short. For the same reasons the *middle* and *final* dimensions are higher because the stimulus and "Go" ranges are much longer (middle more than final, in fact for the most of the tasks the middle part has a slightly higher dimension).

What is strange is the trend for the first six tasks (belonging to the *Go* and *Anti* families) whose dimensions differ from the other tasks by having values greater than two.

This fact is in contrast with the initial hypothesis that the more computationally complex a task is the more dimensions are needed to describe it. What we see is that, not considering the first six, every task needs approximately the same number of dimensions to be described, which is two, as we expect. This suggests that for the tasks considered in this thesis, the computational complexity is almost the same and for that reason the intrinsic dimensions are very similar.

Chapter 4

Conclusions

The aim of this thesis was to verify two main hypothesis:

- RC has a higher dimension than a network trained with BPTT;
- The intrinsic dimension depends on the computational complexity of the task.

The first result we have obtained is that the networks trained with BPTT are lower-dimensional than the ones trained with RC, confirming the first of our main hypotheses. This effect can be attributed to the respective structures. As discussed in Chapter 2, in RC-trained networks the recurrent weights are constant along the training, which means that the same recurrent part is always used for every task. Thus, we cannot expect the recurrent part of the network to exhibit a simple encoding of the relevant variables of each specific task. In BPTT-trained networks, instead, each recurrent weight is adapted and the recurrent part is thus specifically trained for each task. As a result, these networks could be expected to simply encode task-relevant variables. This main difference implies that we can expect a higher intrinsic dimension in RC-trained compared BPTT-trained networks.

As for the relation between dimension and task complexity, what we actually found is that the tasks related to the *Go* and *Anti* families (which are supposed to be the less computational complex) correspond to a higher dimension. Vice versa, tasks belonging to the other families (more complex) need a lower ID. This observation is at odds with our second main hypothesis. We speculate that this discrepancy might actually depend on a limitation of the ID estimator we used. In fact, the TWO-NN estimator, due to the algorithm's design, effectively captures only the local dimension of trajectory bundles in the neighborhood of each data point, not the global dimension of the manifold. While the two coincide for smooth geometrical structures, they might considerably differ in the case of non-smooth structures. Inspection of the data in the low-dimensional space spanned by the first principal components suggests that activity manifolds corresponding to complex tasks may not always be smooth. Comparing the explained variance ratio plot in principal component analysis (Figs. 3.11 and 3.21), we hypothesised that the global dimension can be significantly larger than the local one in the case of the more complex tasks. In fact, plots of the data in the low-dimensional space spanned by the first principal components suggest that in the more complex tasks more independent axes are used to encode different values of the task-relevant variable. In this case, while the local dimension remains low, the global one would not.

In future work, we might devote more attention to fixed points, testing whether they actually lie in a high-dimensional space. In addition, we might test other ID estimators that are expected to be more accurate in the estimation of the local ID, such as [1].

Bibliography

[1] Vittorio Erba, Marco Gherardi, and Pietro Rotondo. Intrinsic dimension estimation for locally undersampled data. *Scientific reports*, 9(1):17133, 2019.

[2] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, 7(1), September 2017.

[3] Stefano Fusi, Earl K Miller, and Mattia Rigotti. Why neurons mix: high dimensionality for higher cognition. *Current opinion in neurobiology*, 37:66–74, 2016.

[4] Peiran Gao and Surya Ganguli. On simplicity and complexity in the brave new world of large-scale neuroscience. *Current opinion in neurobiology*, 32:148–155, 2015.

[5] Mehrdad Jazayeri and Srdjan Ostojic. Interpreting neural computations by examining intrinsic and embedding dimensionality of neural activity. *Current opinion in neurobiology*, 70:113–120, 2021.

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

[7] Francesca Mastrogiuseppe and Srdjan Ostojic. Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron*, 99(3):609–623, 2018.

[8] Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature neuroscience*, 22(2):297–306, 2019.

[9] Guangyu Robert Yang and Xiao-Jing Wang. Artificial neural networks for neuroscientists: a primer. *Neuron*, 107(6):1048–1070, 2020.