INTEGRATION AND MEMORY IN ARTIFICIAL RECURRENT NEURAL NETWORKS

Author:
Lorena Puhl

Supervisors:
Prof. Dr. Vincent Hakim
Prof. Dr. Michael Hausmann
Prof. Dr. Jürgen Hesser

A thesis submitted for the degree of M. Sc. Physics
at the
Laboratoire de Physique de l'Ecole Normale Supérieure
Department for Physics and Astronomy - Heidelberg University

13 May, 2024

Abstract

Understanding the firing-rate patterns of the brain poses a significant challenge. An increasingly popular approach involves simulating experimental observations using artificial recurrent neural networks, which are more convenient, and share significant characteristics with their biological counterparts. We train recurrent neural networks (RNNs) on biological integration- and memory-tasks. By reverse-engineering our RNNs, we aim at understanding their dynamical solutions to propose feasible hypotheses for the working-mechanisms of their biological counterparts.

We began by analysing the computational freedom of different RNNs, and investigating their dynamics using linear analysis. We proceeded by visualising firing-rate trajectories in Principal Component-space. We found, that input-signals of different durations lead to parallel firing-rate trajectories, while their separating distances were correlated to the according signal-times. We therefore hypothesized, that integration relied on measuring the denoted distances. On the other hand, we suggested that output-weights were fine-tuned to rule out the remaining firing-rate dynamics, in order to output a plateau.

Ultimately, we attempted at consolidating our findings, by proposing an analytical solution for the output-weights. We correlated our results with trained output-weights to estimate their accuracy. While we did not obtain a satisfying value, the standard deviation of our results was comparably small. We concluded, that while our analytical expression was not exact, it still captured certain systematic aspects of our RNN's integration- and memory-mechanisms.

Declaration of authenticity

I herewith declare, that this thesis is my own, and that no other sources than those cited have been used.

Lorena Puhl

Paris,
13 May, 2024

Acknowledgements

I would herewith like to thank the following persons for their valuable support, help and advice:

Theoretical neuroscience and biophysics team:

Prof. Dr. Vincent Hakim

The University of Heidelberg:

Prof. Dr Michael Hausmann
Prof. Dr. Jürgen Hesser

Contents

1 Introduction

Artificial Intelligence is a very fruitful area, which was initially inspired from nature's example of biological neural networks. The latter are the foundation of the human nervous system and are composed of interconnected neurons, that transmit and process information. These networks are responsible for various cognitive functions, including learning, memory, perception, and decision-making [1]. They "learn" to solve these most versatile tasks by inducing changes in their net structure and connectivity, a process referred to as "neural plasticity". In the same manner, artificial neural networks have been successfully modulated and trained to solve problems of increasing complexity, such as optimization problems, analysing various forms of data, categorisation, predicting model-outcomes and much more.

In the domain of neuroscience and medicine, artificial neural networks are able to replicate neuronal firing patterns linked to disorders such as Schizophrenia [2] or autism [3], and thus are becoming an increasingly relevant tool in understanding the brain. A crucial and still not solved question is that of the neural code, namely how to the neurons' electrical pulses encode and package information [4]. *What is the translation key to the Morse code of neurons?* Answering this question by observing the brain is a daunting challenge due to its sheer complexity. Computational neuroscience has been addressing this query by combining experimental neurobiology, psychophysics, and mathematical analysis. It is turning the tables and looking in the opposite direction. While in the past, we drew inspiration from biological networks to develop artificial models, we are now taking the reverse approach. The working mechanisms of artificial neural networks, propose feasible hypotheses for the functioning of their biological counterparts. We are thus *reverse-engineering* the fruit to come come back the seed.

Following this approach, we use artificial neural networks to replicate biological observations. Certain cognitive functions rely on accumulating information, and retaining the results for future processing. They are referred to as "integration of evidence" and "short-term working-memory". We simulate both features using recurrent neural networks, which are frequently used in neuroscience due to their biological likeness. This yields a tangible model, which is easier to comprehend than the brain's vast complexity. We attempt at understanding the working-principles of our artificial neural network for carrying out integration- and memory-functions. Might our insights propose a feasible hypothesis for the brain's neural code?

2 Integration and memory

Before diving into the details of this thesis, we begin by outlining significant experimental observations in the brain. A common tool to peak inside the realm of biological neural networks is made of electrodes picking up electrical currents and their emitted magnetic fields. As such, Electro- and Magnetoencephalography are described as *windows of the mind* [5]. We are interested in two types of cognitive functions, which often happen conjointly. Exemplified by observations in goldfish and monkeys, neuron populations accumulate incoming signals and retain the results for short periods of time. These patterns are known as "evidence-integration" and "short-term working-memory".

2.1 Integrator networks

Since Charles Sherrington published *The Integrative Action of the Nervous System* in 1906 [6], the notion of neurons integrating information across time has become an increasingly observed phenomenon. It is defined as the accumulation of input signals, while the resulting output reflects the running total of the inputs [7]. Examples include neurons of the oculomotor-system in goldfish, which integrate velocity-signals to output according positional signals. Furthermore, neurons in monkeys' lateral intraparietal area were observed to accumulate information for decision-making tasks.

2.1.1 Integrators in oculomotor systems

In the 1960's the first neural integrator was identified in the oculomotor system of goldfish, which regulates the eye's movement and position. In brief, it is a neural network in the brainstem and cerebellum, that receives input-signals related to head-movements. After processing them, it passes on an output to extraocular motoneurons, which encodes horizontal eye-position.

The "integration"-behaviour denotes the processing of the incoming signals, which can be interpreted as an accumulation of the inputs. Upon receiving them, the integrator-neurons exhibit a sustained increase or reduction in firing-activity. The level of their firing-activity represents the eye's position, thus changing activities result in eye-movement. Finally, the eyes are held still as their new position is stored through persistent firing at a constant level. The respective level encodes the eye's new position. [8] [9]. Figure 1 depicts a schematic concept, while figure 2 illustrates the measured activity of neurons in the integrator-network.

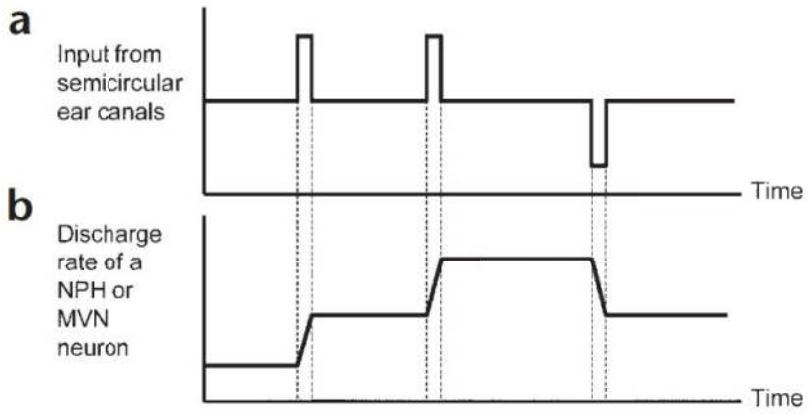


Figure 1: Schematic concept of the oculomotor system. **(a)** Head movements produce signals reflecting its velocity, while the sign depends on the direction of head movement. **(b)** As a result, brainstem-neurons show a sustained increase or decrease in firing [9].

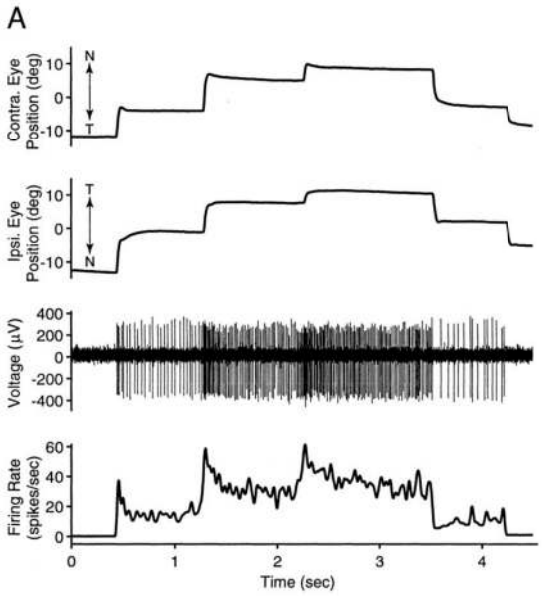


Figure 2: Activity of a downstream motoneurons encoding eye position. The top two traces illustrate the horizontal eye position. The middle trace shows extracellular recordings. The bottom trace represents the firing rates during horizontal eye movements. The neurons transitioned to different states of constant firing-activity, when eyes shifted from one fixation-point to another [10].

2.1.2 Integrators in discrimination tasks

The integrating mechanism was also observed during a discrimination task on monkeys. The animals were shown random dot movements with a small proportion coherently

moving in a certain direction. By making an eye movement to the respective target, the monkeys had to judge and indicate the denoted direction (figure 3). During the experiment, the activity of certain neurons in the brain's lateral intraparietal area was measured, which yielded the following results: While the monkeys had to observe the dot motions, the membrane-activity of certain neurons seemed do gradually increase. In other words, they accumulated negative and positive evidence for motion in each direction. At the moment of decision-making, the activity reached a certain threshold, which defined the decision-criterion. Furthermore, the accumulation rate seemed to be directly correlated to the fraction of coherently moving dots (figure 4) [11].

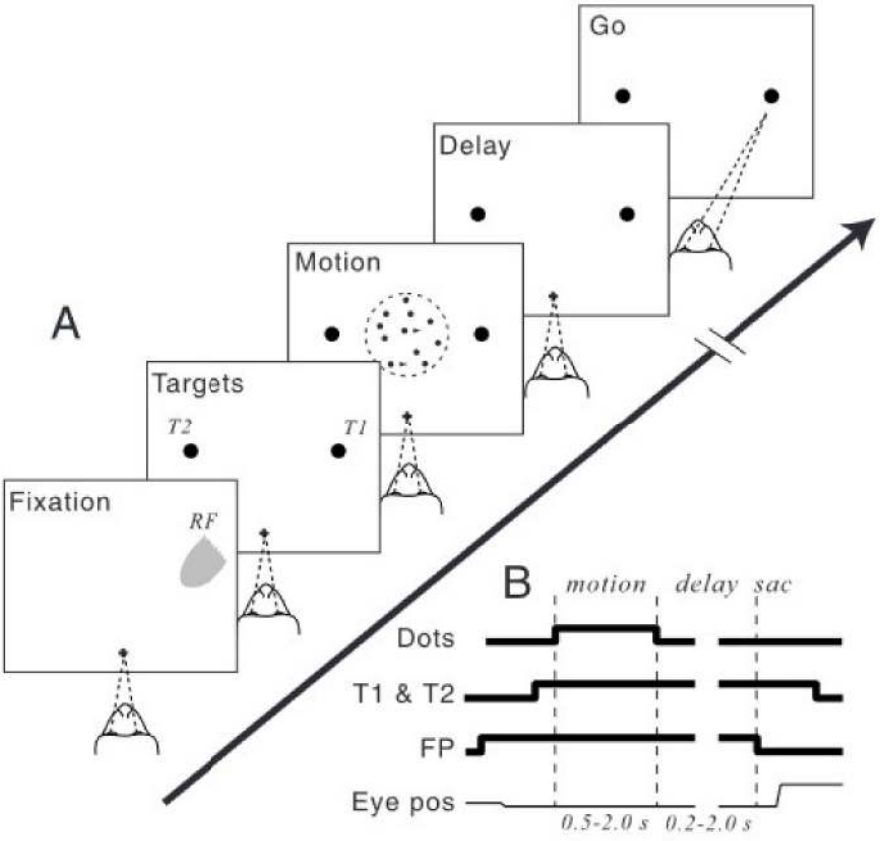


Figure 3: **(a)** Experimental set-up of the discrimination task. Monkeys are shown a set of randomly moving dots with a certain fraction coherently moving in a single direction. The animals are trained to first fixate the screen. They are then shown both areas used for indicating their decision. After a certain delay-period comes the *Go* stimulus. The monkey then indicates the direction of coherent movement by an eye-saccade to the left or right. **(b)** Time-frame of the experiment [11].

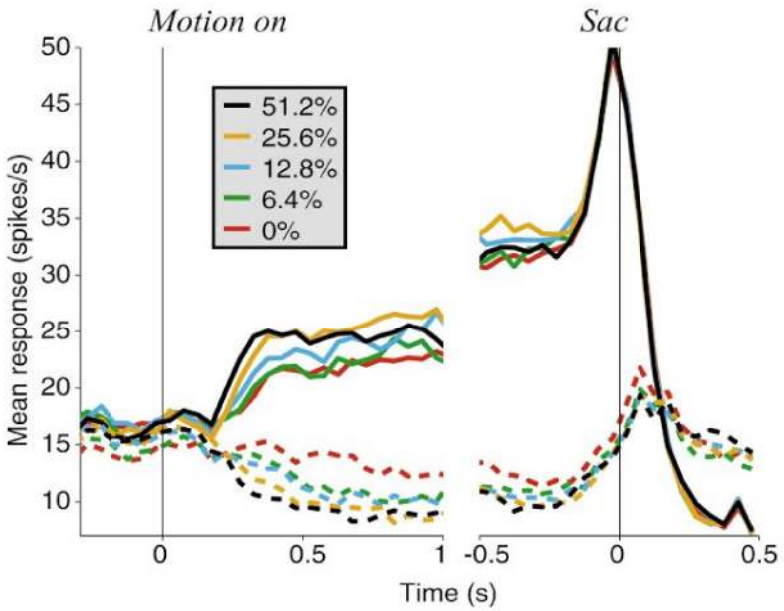


Figure 4: **(left)** Average firing rates of 104 neurons during the discrimination period. **right)** The decision-making moment is defined by an eye-saccade in the direction of choice. Solid and dashed curves are from trials during which the monkeys opted for opposite directions. The average firing rates increase at a higher rate for trials with a bigger dot-fraction moving in a certain direction [11]

2.2 Working memory

Very often, cognitive tasks require not only integrating certain stimuli , but also keeping track of the integration's final result after the input-signals are withdrawn. *Short-term working-memory* defines a plateau-like neuronal activity, which is proportional to the remembered stimulus. It hereby encodes the integration's result and "keeps it in mind". This pattern has been extensively studied in the prefrontal cortex, an area known for its important role in memory formation [9] [12].

Short-term working-memory was observed in the 1970's during an interesting experiment on monkeys. The animals were shown a brief flash of a visual target. After a $1 - 6$s delay-period, they had to demonstrate its location by an eye movement in the according direction. 288 neurons in the prefrontal cortex were recorded during the task [13]. According to the target's location, certain neurons showed a sharp activity increase and plateau-like behaviour, while each location was encoded by a separate set of neurons [14]. Figure 5 schematically demonstrates the working-memory-pattern of neurons in the prefrontal cortex. It shows the denoted plateau-like behaviour, which encodes and "remembers" the target's location. The respective neuronal recordings are illustrated in figure 6.

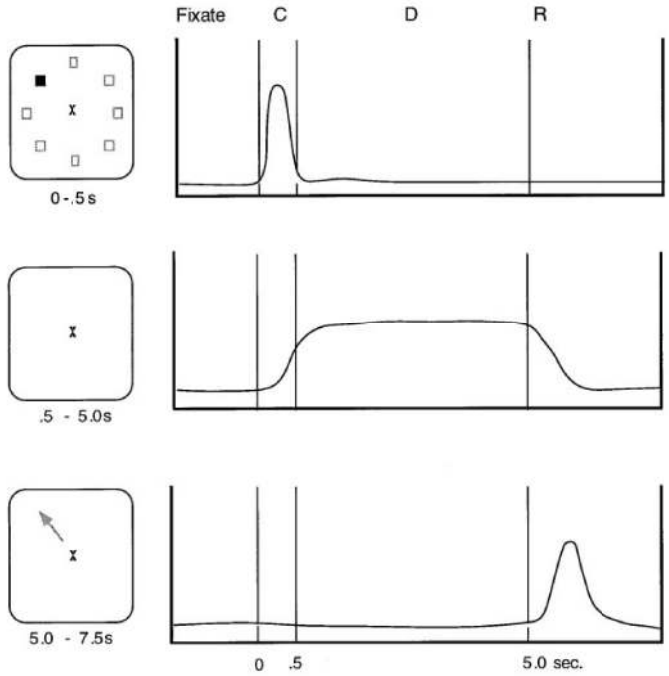


Figure 5: Conceptual time-frame of the activity of neurons in the prefrontal cortex. **(a)** Some neurons exhibit a sharp activity rise during the activation of a stimulus. **(middle)** The stimulus is remembered during the delay-period by other neurons showing a plateau-like behaviour, while specific locations are encoded by different neurons. **(bottom)** A last set of neurons fire when a reaction to the remembered stimulus is initiated [14].

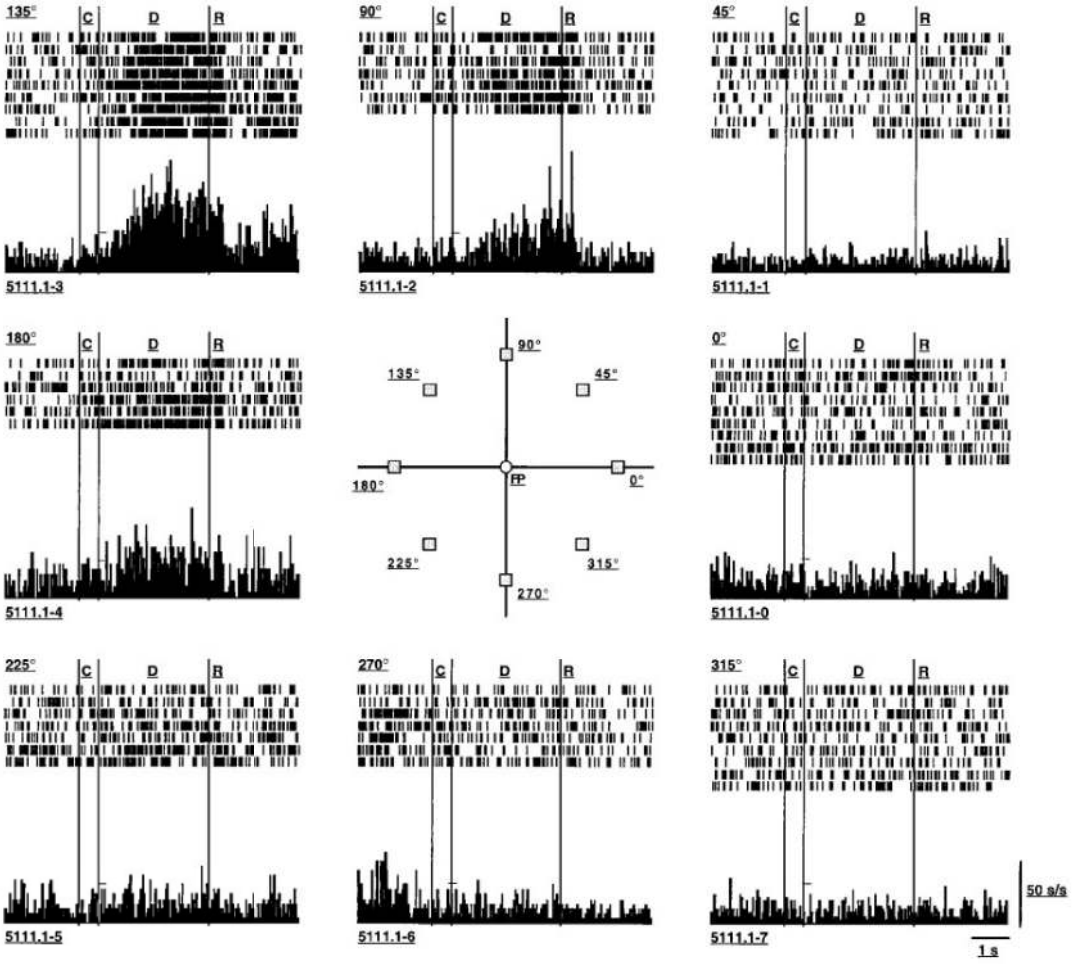


Figure 6: Repeated recordings from a single neuron over numerous trials are in a histogram of the average response to each stimulus location. The neuron's activity increases to its maximum activity after the flashing of the 1358 target and is maintained during the delay-period in a plateau-like fashion until the response. Similar activity patterns are also observed after the 908 and 1808 targets are presented, but with a lower amplitude. This indicates, that the neuron can encode different locations, but does have a "best direction" [13]).

A final experiment showcases the neuronal firing-pattern of working-memory using a vibrotactile stimulus. Monkeys were required to remember the frequency of a vibrating device, while the membrane-activities of neurons in the prefrontal cortex were recorded. Their firing-rates showed a plateau-like activity while the monkeys were remembering the respective stimulus. The according frequency was encoded by the plateau's height (figure 7) [15].

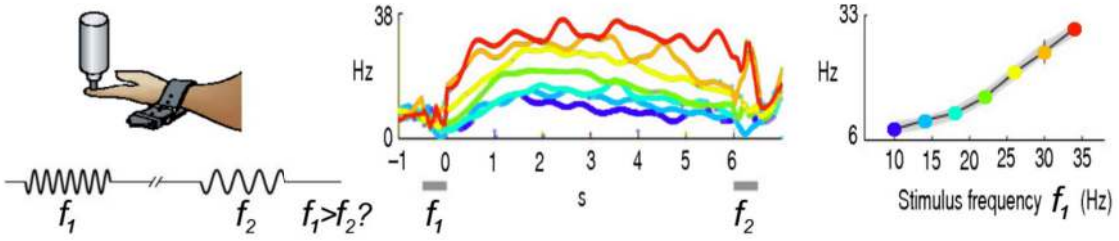


Figure 7: **(left)** Experimental set-up of the vibrotactile stimulus task on monkeys. **(middle)** After the stimulus has been turned off, neurons in the prefrontal cortex show a plateau-like persistent activity. **(right)** The plateau's height is directly correlated to the stimulus' frequency [7]

This chapter has guided us through several experimental observations, which showcase the integration and memory-behaviour of biological neural networks. We reiterate our findings, by describing "integration" as an accumulation of input-signals. It is often followed by a "memory"-mechanism, which retains the integration's result for other cognitive functions. This is achieved by a plateau-like activity, while the plateau's height encodes the according result.

3 Theoretical background

After having discussed the "integration" and "working-memory"-functions of biological neural networks, we are turning our attention to their artificial counterparts. This chapter provides the theoretical foundation of artificial neural networks and introduces the tools we will utilize to investigate their working-mechanisms.

3.1 Recurrent neural networks

The idea of artificial neural networks was born with the work of McCulloch and Pitts in 1943 [16]. Inspired by nature's example of the nervous system, they thought of imitating it by designing a cluster of interconnected computational cells processing and transmitting information, just as biological neurons do. As such, they receive certain inputs to produce and pass on according outputs. This train of thought led to the first Perceptron designed by Frank Rosenblatt in 1958 [17].

We are skipping over the history of artificial neural networks and the numerous designs and concepts they went through to directly dive into our model of interest, the Recurrent Neural Network (RNN). It refers to a specific network-architecture, which in machine learning is mainly used for sequenced data, such as handwriting, genomes, text or numerical time series. The term *recurrent* refers to its interconnected neurons, which transmit information back to themselves in cycles (figure 8a). It is the model of choice when simulating observed patterns in neuroscience, since it shares essential similarities with biological neural circuits in the brain, which we will outline in section 3.4. Furthermore, drawing our attention to a single piece of this intricate puzzle, neurons are biological devices invisible to the eye, which process and pass on electrical signals in the form of action potentials (or spikes). The non-linear and saturating properties of their membrane-potentials are approximated by the *integrate-and-fire model* [18].

To design our idea of a computational imitation of the brain, we will simplify our notion of a neuron i by its membrane potential, which we denote by $x_i(t)$. In machine-learning literature $x_i(t)$ is also referred to as the "pre-activation". We can further include an input-output saturating non-linear function $\phi(x_i(t))$ converting the membrane potential to a firing rate. Often $\phi(x_i(t)) = \tanh(x_i(t))$ is used. We follow biology's example, and let the membrane potential be governed by exponential decay

$$\tau \frac{dx_i(t)}{dt} = -x_i(t). \tag{1}$$

τ is the neuronal time constant, affecting the time-scale of the exponential decay and the $-x_i(t)$ on the RH-side of equation 1 is often referred to as the *leak term*. A neuron furthermore receives external time-dependant inputs $u(t)$, as well as inputs from the other neurons of the network [19], giving rise to the so-called *rate model* [20]:

$$\tau \frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N} G_{ij}\phi(x_j(t)) + \sum_{k=1}^{M} I_{ij}u_k(t) \tag{2}$$

Here, G is the connectivity matrix and the external inputs $u(t)$ are fed to the network through weights I. The current state of the network is defined by the N-dimensional vector (x), packaging the evolution of individual membrane potentials $x_i(t)$ inside the space of neurons. Finally, the network's output is represented by

$$z(t) = \sum_{i=1}^{N} W_i\phi(x_i(t)), \tag{3}$$

Since we are aiming at applying these concepts to questions in neuroscience, we should decide on an appropriate definition of the connectivity matrix $\mathbf{G}$. Some models follow observed biological constraints [21], while others are based on biological statistical properties [22]. In other cases, the connectivity is learned through optimisation algorithms [23] based on plasticity rules [24]. Often-times $\mathbf{G}$ is simply composed of randomly distributed components to make the least possible assumptions, while simultaneously having substantial knowledge of the network's nature using random matrix theory. An especially helpful notion in this regard is the *circular law theorem*, which will be discussed to in the following section 3.2.

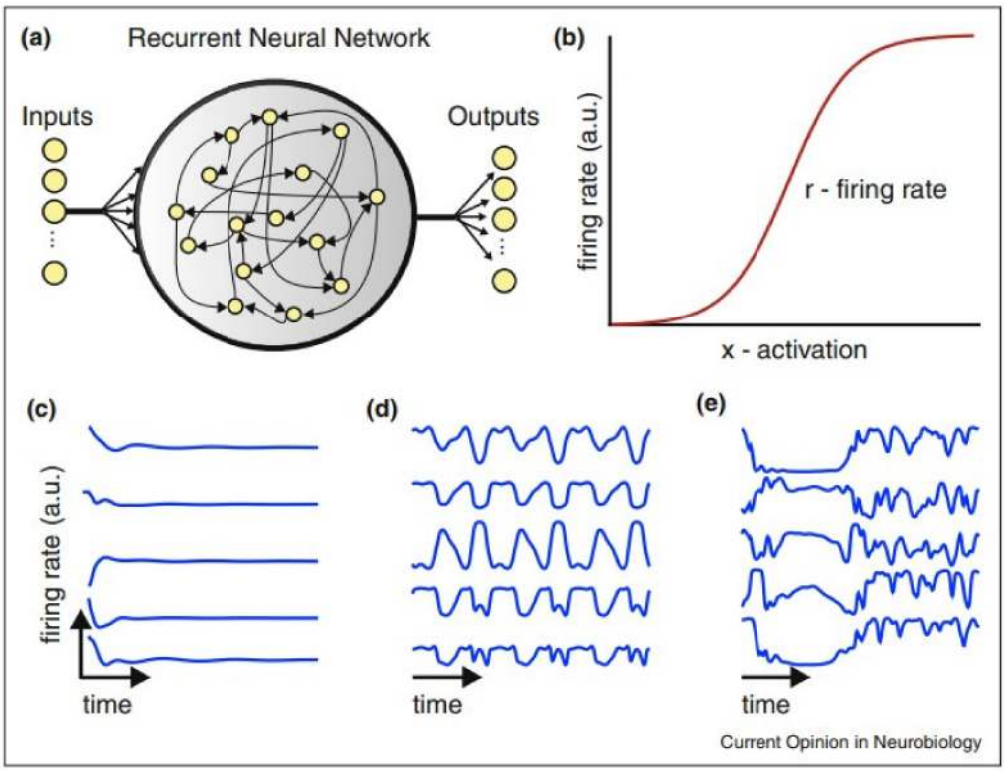


Figure 8: **a)** Architecture of a Recurrent Neural Network. **b)** Activation function $\phi(x(t))$. Units in RNNs can show **c)** attractor dynamics, **d)** limit cycles, and **e)**chaotic dynamics. Blue lines show membrane potentials through time of 5 units in the RNN [25].

3.2 Linear analysis of neural networks

We certainly know from everyday experience that every benefit most often than not also carries a drawback. As such, the non-linear activation function $\phi(x)$ on one hand enhances the processing abilities of a neural network, but on the other hand also makes its dynamics more intransparent, since it can only be gleaned at in the linear regime. Such a glance is made possible by the eigenspace of the connectivity matrix $\mathbf{G}$, linearly approximating of the network's dynamics around local fixed points [19].

We begin by simplifying equation 2 assuming $u(t) = 0$ and $\phi(x) = x$ as a linear approximation of the $\tanh(x)$ function around $x = 0$:

$$\frac{d\mathbf{x}}{dt} = \mathbf{M}\mathbf{x}, \tag{4}$$

defining the *linear stability matrix* as $\mathbf{M} = g\mathbf{G} - \mathbf{1}_N$ using the NxN identity matrix. A spectral decomposition of the stability matrix M separates the system of differential

equations in 2, giving rise to the individual eigenmode solutions

$$\frac{d\tilde{x}_i(t)}{dt} = \lambda_i \tilde{x}_i(t), \tag{5}$$

where the $\tilde{x}_i(t)$ component is the projection of the membrane potentials of all neurons $\mathbf{x}(t)$ onto the i-th eigenvector $\mathbf{v}_i$ and λ_i is the according eigenvector. We are thus converting to eigenvector-space using a change of basis:

$$\mathbf{x}(t) = \sum_{i=1}^{N} \tilde{x}_i(t)\mathbf{v}_i \tag{6}$$

using the basis transformation matrix $\mathbf{V}$ defined by $\mathbf{v}_i$ for column i. A change of basis is thus conducted by

$$\tilde{\mathbf{x}} = \mathbf{V}^{-1}\mathbf{x} \tag{7}$$

$$\tilde{\mathbf{M}} = \mathbf{V}^{-1}\mathbf{M}\mathbf{V} \tag{8}$$

We can interpret the eigenvectors $\mathbf{v}_i$ as subpopulations of interlocked neurons, so-called *modes*, while their eigenvalues λ_i determine their activity's persistence. Stability of the system is determined by whether the real parts of all eigenvalues are less than zero (or using the definition $\mathbf{M} = g\mathbf{G} - \mathbf{1}_N$ smaller than 1). If they are, the system is stable and the network's activity returns to zero. If one or more eigenvalues have real parts greater than zero, the system becomes unstable and experiences self-sustained activity. The effects of different eigenspectra on the network's activity are summarised in figure 10. Additionally, depending on the gain-factor g and the network size N, the network converges to different activity patterns including fixed points (Figure 8c), limit cycles (Figure 8d) and chaos (Figure 8e) [26].

Before concluding this chapter, we mention a final important ingredient for understanding our model's nature, namely the *circular law theorem* [27] alluded to in the previous section. It states that the spectral density of an NxN matrix consisting of identically and independently distributed random variables with $\mu = 0$ and $\sigma = 1/\sqrt{N}$ converges for $N \to \infty$ to the -uniform distribution on the unit disk (figure 9). Thus, setting a gain-factor $g > 1$ is a crucial condition for obtaining eigenvalues with positive real part and subsequently exponentially increasing and self-sustaining modes.

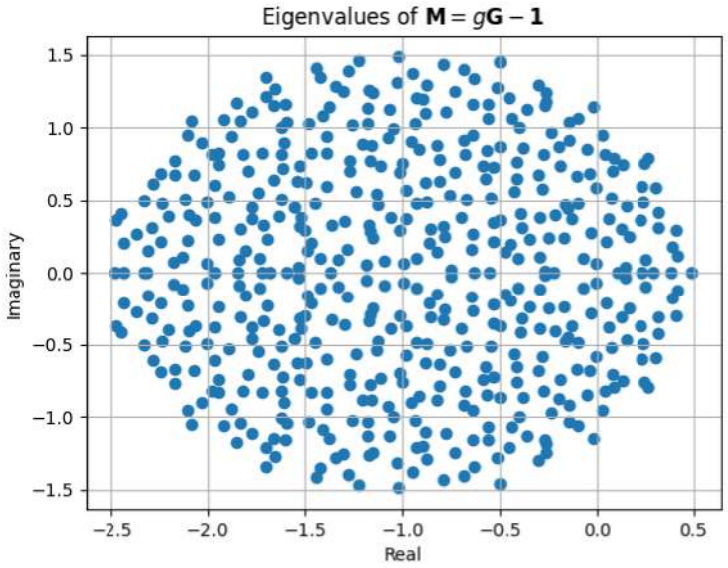


Figure 9: Eigenvalues of the linearised stability matrix $\mathbf{M}$ for $g = 1.5$, $N = 500$ and normally distributed entries of $\mathbf{G}$ with $\mu = 0$ and $\sigma = 1/\sqrt{N}$

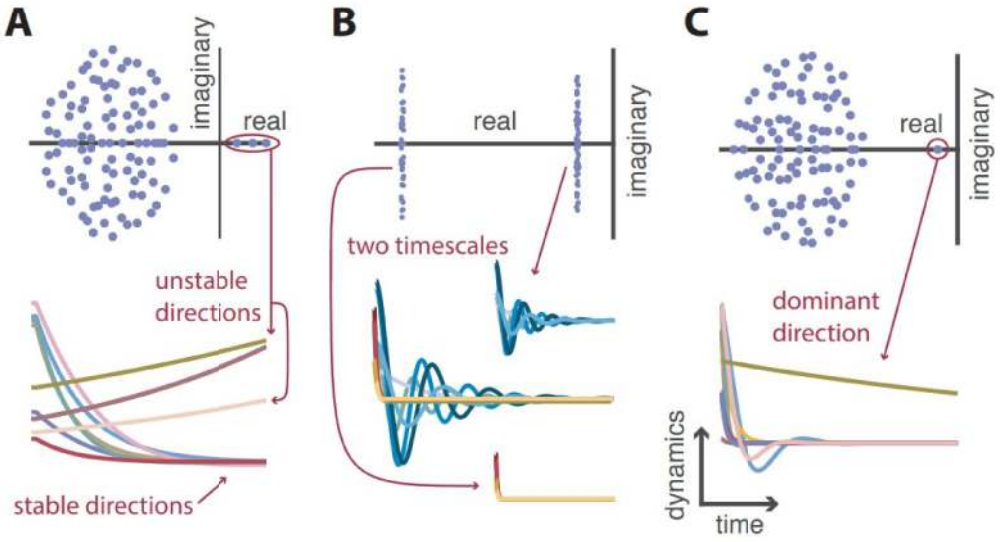


Figure 10: Interpreting $\mathbf{M}$'s eigenspectrum: **A)** Unstable directions arise from eigenvalues with real parts larger than zero. A stable dynamical system emerges when all eigenvalues have real parts less than zero. **B)** Common timescales arise from clustered eigenvalues with respect to the real axis **C)** The dominant long-term behaviour is predicted by the largest eigenvalues part much larger than the other eigenvalues [19].

3.3 Neural integrators

We move on by applying the concepts of integrating networks and working-memory in section 2 to our theoretical framework of recurrent neural networks.

Relating to the notion of memory, the leak-term in equation 2 contributes to the "forgetfulness" of neurons. Thus, their ability to temporally sum incoming synaptic potentials is restricted by the membrane's time-constant τ. When an input is fed into the network, it changes its dynamics according to equation 2. Even after the input has been turned off, this dynamical stimulus is still "echoing" through the recurrent synaptic connections. This imagery describes how a network both accumulates and retains external signals.

In more mathematical terms, perfect integration is achieved when the recurrent feedback-activity $\sum_{j=1}^{N} G_{ij}\phi(x_j(t))$ in equation 2 balances out the leak-term $-x_i$.

$$x_i = \sum_{j=1}^{N} G_{ij}\phi(x_j(t)) \tag{9}$$

$$\Rightarrow \tau\frac{dx_i(t)}{dt} = I_i u(t) \tag{10}$$

$$\Rightarrow x_i(t) = \frac{1}{\tau}\int I_i u(t)dt \tag{11}$$

Using the eigenspectrum of the stability matrix $\mathbf{M}$ as a basis $\{\mathbf{v_i}\}$, this solution translates into most eigenvectors having an associated eigenvalue with real part $<< 1$, i.e. rapidly exponentially decaying modes, and a single value ≈ 1 as the integrating component. We thus obtain equation 10 along this specific eigenvector:

$$\tau\frac{x_1(t)}{dt} = -x_1 + \lambda_1 x_1 + I_1 u(t) \Rightarrow x_1(t) = \frac{1}{\tau}\int I_1 u(t)dt \tag{12}$$

if $\lambda_1 = 1$ such that this mode leads to perfect balance between recurrent feedback and leak.

The plateau-shaped firing-pattern, acting as short-term memory then comes forth according to

$$\frac{dx_1(t)}{dt} = (\frac{\lambda_1 - 1}{\tau})x_1(t) = 0 \tag{13}$$

Consequently, this mechanism requires exact fine-tuning of the eigenvalues, which may difficult to accomplish in real biological networks. Using experimental values

$\tau_{\text{neuron}} \approx 100\text{ms}$, to ensure a network time-constant of $\tau_{\text{network}} = {}^{\tau_{\text{neuron}}}/_{|1-\lambda_1|} \approx 5\text{s}$, the network's eigenvalue λ_1 has to be tuned to an accuracy of $|1-\lambda| = {}^{\tau_{\text{neuron}}}/_{\tau_{\text{network}}} \approx 2\%$.

3.4 Reverse-engineering

After describing observations in biological neural networks, and conceptualising their artificial counterparts, we now aim at combining both notions. Recurrent Neural Networks are the model of choice, since they share essential similarities with biological neural circuits in the brain.

First of all, the brain is best modelled by a cluster of recurrently connected cells, since the output of most cortical areas is constantly fed back to its region of origin [28]. This geometry allows neurons to segregate into sub-populations, working in parallel and making use of their distributed positions in space to implement complex computations [25]. Such feedback loops in the brain have been extensively studied. To only exemplify a few, recurrent circuits have been found in the local motor, visual, and frontal cortices [29] [30] [31]. Self-feedback connectivity has also been observed in the medial prefrontal cortex, integrating inputs from basal lateral amygdala and insular cortex neurons [32].

Furthermore artificial RNNs and cortical neural networks both feature chaotic dynamics [33], which they leverage to achieve certain functions. To name an example, sensory neurons show chaotic spiking activity [34], which they synchronize, using saddle-stabilities in their dynamical landscape [35].

However, the brain's working-mechanism is difficult to investigate due to the organ's vast complexity. Thus, we use its commonalities with RNNs, which are more accessible [36]. We simulate cortical patterns of interest using artificial neural networks, and investigate the working-principles of these models instead. Our understandings of RNNs may then serve as new hypotheses for the brain's functioning. We are thus *reverse-engineering* artificial neural networks to make educated guesses on the brain.

The principle of "reverse-engineering" relies on training RNNs on *what* to compute without telling them *how* to do so. Understanding the *how* then proposes dynamical solutions for observed cortical patterns, which could be experimentally verified. A common strategy is describing the activity of neurons as time-varying trajectories in state-space, and trying to find low-dimensional structures. They might be the source of the dynamical mechanisms underlying the studied computations [37]. As such, line attractors have been proposed to enable the accumulation of evidence [25], while saddle

points have been suggested to mediate decisions [38]. Furthermore, point attractors are a possible substrate of memory formation [39].

3.5 Training algorithms

We thus conclude from the previous section, that generating suitable hypotheses for working-memory and integrating behaviours requires us to optimise a network, by training it to output these patterns with minimal instructions on how to so. This significant amount of freedom allows for a rich palette of suggestive dynamical solutions. We introduce an error function $\varepsilon(t)$ with which we compare the network's output $z(t)$ with a target output $g(t)$, and modify the network's weight by using the error function's negative gradient. Here, we use the Mean-squared-error (MSE) Loss-function $\varepsilon(t) = (z(t) - g(t))^2$. This method is known as the *gradient-descent* method [40]. When handling time-dependent signals, the error is computed for every time-point. In machine-learning, this process is illustrated by *unrolling* the network (figures 11 and 12), such that time-dependence is interpreted as a series of consecutive networks corresponding to each time-point. The total cost is thus constructed by $\varepsilon = \sum_{1 \leq t \leq N_t} \varepsilon_t$.

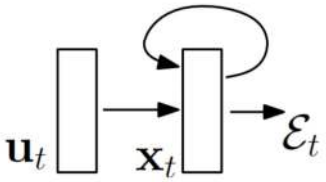


Figure 11: Schematic graph of an RNN with input $\mathbf{u}_t$ at time t. It has direct effect on the membrane potentials $\mathbf{x}_t$, whose output are evaluated and yield the error ε_t.

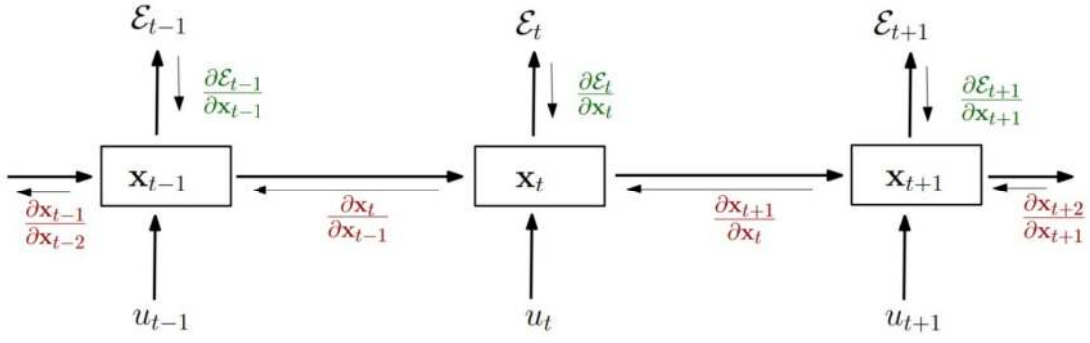


Figure 12: *Backpropagation Through Time* treats each time point as a separate RNN. Error gradients become a sum of individual gradients ε_t at each time-point t [41].

The ruling equations of this concept of *Backpropagation Through Time* (BPTT) [42] become

$$\frac{\partial \varepsilon}{\partial \theta} = \sum_{1 \leq t \leq N_t} \frac{\partial \varepsilon_t}{\partial \theta} \tag{14}$$

$$\frac{\partial \varepsilon_t}{\partial \theta} = \sum_{1 \leq k \leq t} \frac{\partial \varepsilon_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k^+}{\partial \theta} \tag{15}$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \tag{16}$$

Equation 14 depicts the error's gradient with respect to a trainable parameter θ as the sum of each time-component of the unrolled graph. Furthermore, each individual gradient is obtained through equation 15 by using the chain-rule, where $\frac{\partial \mathbf{x}_k^+}{\partial \theta}$ denotes the partial derivative of the state $\mathbf{x}_k$ with respect to θ, such that $\mathbf{x}_{k-1}$ is kept constant. The so-called individual *temporal contributions* $\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k}$ trace back how the parameter θ at previous times k affect the error at time t.

However, these long products make RNN's particularly vulnerable to the so-called *exploding* or *vanishing gradient problem*. It refers to gradient-norms uncontrollably increasing or converging towards zero due to long-term components ($\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k}$ for $k << t$) being "exponentially sensitive" to such behaviour [41].

A crucial ingredient to avoid such failure of training is a careful choice of the initial conditions for the network's weights. Zooming into a single time-point in the unrolled graph of figure 12, and approximating it as a dynamical fixed point ($\frac{d\mathbf{x}(t)}{dt} \approx 0$) yields the equation

$$x_i^t = g \sum_{j=1}^{N} G_{ij} \phi(x_j^{t-1}) + I_i u^t, \tag{17}$$

which in turn yields temporal contributions of the form

$$\frac{\partial x_i^t}{\partial x_i^{t-1}} = g \sum_{j=1}^{N} G_{ij} \phi'(x_j^{t-1}) \tag{18}$$

Avoiding exploding or vanishing gradients translates into maintaining the temporal contributions ≈ 1. This is achieved by keeping expectation values $\mathrm{E}(x_i^t) = \mathrm{E}(x_i^{t-1}) \approx 0$, such that $\phi'(x)$ does not converge towards zero (ϕ being a saturating activation function). In this linear regime, $\phi(x) \approx x$ and $\phi'(x) \approx 1$. Thus,

$$\frac{\partial x_i^t}{\partial x_i^{t-1}} \approx g \sum_{j=1}^{N} G_{ij} \tag{19}$$

which motivates a scaling of $G_{ij} \approx 1/\sqrt{N}$ for large N. This concept and a more detailed derivation are known under the name of *Xavier initialisation* [43]. We make a similar scaling argument for the output-weights $\mathbf{W}$. Knowing, that for $\mathrm{E}(x_i^t) \approx 0 \rightarrow \phi(x_i^t) \approx 1$,

$$z^t = \mathbf{W}^T \mathbf{x}^t = \sum_{i=1}^{N} w_i x_i^t \approx \sum_{i=1}^{N} w_i. \tag{20}$$

We thus scale the output-weights by $1/N$ as well. Finally, since inputs $u(t)$ do not contribute to the network dynamics through a sum, they are left as they are and scaled by 1. We finally conclude these considerations by motivating out chosen initial conditions as

$$G_{ij} \sim \mathcal{N}(\mu = 0, \sigma^2 = 1/N) \tag{21}$$

$$W_i \sim \mathcal{N}(\mu = 0, \sigma^2 = 1/N) \tag{22}$$

and

$$I_i \sim \mathcal{N}(\mu = 0, \sigma^2 = 1), \tag{23}$$

such that membrane potentials and outputs intrinsically stay in ranges of $\mathrm{E}(\mathbf{x}(t)) = \mathrm{E}(z(t)) = 0$ and $\mathrm{Var}(\mathbf{x}(t)) = \mathrm{Var}(z(t)) = 1$, and exploding or vanishing gradient problems are avoided.

After having computed the gradient $\partial\varepsilon/\partial\theta$ and avoided its convergence towards zero or infinity, it can finally be utilised to optimise the parameters θ following

$$\theta' = \theta - \alpha \frac{\partial \varepsilon}{\partial \theta}, \tag{24}$$

where the learning-rate α controls the extent to which every training step updates the parameter θ.

3.6 PCA Analysis

As previously outlined, backengineering the working mechanism of a trained RNN translates into understanding its dynamics. Since we are faced with high-dimensional coupled differential equations, insights can only be gained by reducing the dimensions of the state space incorporating the network dynamics. A popular method is the *Principal Component Analysis*, which yields an orthogonal basis determined by the directions of highest data-variance (or movement of the neuron trajectories in state-space). This is achieved by computing the eigenvectors and eigenvalues of the trajectories' $\mathbf{r}(t) = \phi(\mathbf{x}(t))$ covariance matrix Σ in equation 25. The eigenvalues define define the total variance occurring in the direction of the corresponding eigenvector.

$$\Sigma = \begin{bmatrix} \mathrm{Cov}(r_1, r_1) & \mathrm{Cov}(r_1, r_2) & \cdots & \mathrm{Cov}(r_1, r_n) \\ \mathrm{Cov}(r_2, r_1) & \mathrm{Cov}(r_2, r_2) & \cdots & \mathrm{Cov}(r_2, r_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(r_N, r_1) & \mathrm{Cov}(r_N, r_2) & \cdots & \mathrm{Cov}(r_N, r_N) \end{bmatrix} \tag{25}$$

Expression 25 makes use of the covariance between the discrete firing rate trajectories r_i^t and r_j^t of neurons i and j at times t using

$$\mathrm{Cov}(r_i, r_j) = \frac{1}{N_t - 1}(\sum_{t=1}^{N_t} r_i^t - \langle r_i \rangle_t)(\sum_{t=1}^{N_t} r_j^t - \langle r_j \rangle_t) \tag{26}$$

where we are using a sample-mean

$$\langle r_i \rangle_t = \frac{1}{N_t} \sum_{t=1}^{N_t} r_i^t \tag{27}$$

instead of a population mean μ, which requires to divide by $N_t - 1$ in equation 26.

These considerations capture the covariance between the individual neuron trajectories $\mathbf{r}(t)$ of a single experiment. However, the nature of a neural network is better revealed by conducting numerous experiments, thus feeding it different inputs and observing its varying trajectories. We are thus interested in depicting across-trial covariance and are extending our method to a *trial-concatenated PCA* [44]. We use the term"extending" in its most literal sense and concatenate the trajectories $\mathbf{r}_m^t$ for different trials $m = 1, ..., M$ to a single vector

$$\mathbf{r}_M^t = \begin{bmatrix} \mathbf{r}_1^t & \mathbf{r}_2^t & \cdots & \mathbf{r}_M^t \end{bmatrix}^T \tag{28}$$

where for the sake of completeness we repeat the definition of the firing rate vector

$\mathbf{r}_m^t = \phi(\mathbf{x}_m^t)$ of a single trial m

$$\mathbf{r}_m^t = \begin{bmatrix} (r_m^t)_1 & (r_m^t)_2 & \cdots & (r_m^t)_N \end{bmatrix}^T \tag{29}$$

for N neurons. Computing the eigenspectrum of the covariance matrix Σ for these extended firing-rate-vectors thus not only reflects the directions of highest variance within a single trial, but additionally depicts how the trial-to-trial drift occurs in space.

Finally, the degrees of freedom underlying the investigated dynamics are estimated by the number of necessary principal components to describe most of its variance. Such an approximation is given by the *participation ratio*[45]

$$D = \frac{(\sum_{i=1}^{N} \mu_i)^2}{\sum_{i=1}^{N} \mu_i^2}, \tag{30}$$

where μ_i is the i^{th} eigenvalue of the covariance matrix. If all eigenvalues contribute equally (i.e. $\frac{\mu_i}{\sum_{i=1}^{N} \mu_i} = \frac{1}{N}$ the dimension estimate is $D = N$. Conversely, if only one eigenvalue contributes then $D = 1$.

4 Training Recurrent Neural Networks

The previous chapter outlined the theory of artificial recurrent neural networks (RNNs) and the tools to understand their working-principles. These concepts will be used in the following sections to design and train RNN-models on integration- and working-memory-tasks. Successfully training an artificial neural network entails an accurate choice of parameters, an appropriate data-set, and finally an efficient algorithm. We will proceed by elaborating on these features.

4.1 Architecture

We begin by depicting the properties of our RNN. It consists of a recurrently connected set of $N = 500$ neurons, with random and independent connection-weights G_{ij} drawn from a Gaussian distribution following equation 21, and with a coupling-factor $g = 1.5$. According to the *circular law theorem* described in section 3.2, the stability matrix' eigenvalues are thus uniformly distributed in a unit disc (figure 9), featuring self-sustained membrane potentials $x_i(t)$ due to $\mathrm{Re}(\lambda_i) > 0$ for certain eigenvalues. Importantly, the recurrent connections are never trained or altered, such as to best mimic the randomness of biological networks and to make the least possible assumptions and restrictions on the model. The membrane-potentials $x_i(t)$ are converted to firing rates using the activation function $\phi(x) = \tanh(x)$, while their intrinsic time-constants are chosen as $\tau = 100\text{ms}$, a comparable value with their biological counterparts. Finally, a single input $u(t)$ is fed to every neuron i through weights I_i. Each neuron i is also connected to the output-cell through weights W_i, yielding the output $z(t)$ according to equation 3. Weights are initialised using the random distributions and 22 23.

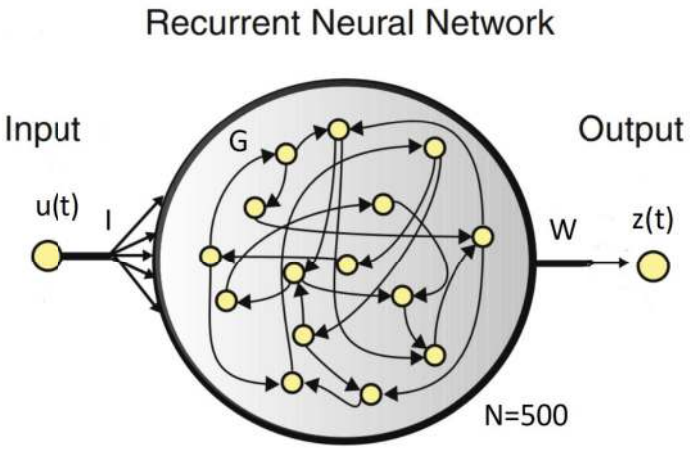


Figure 13: Design of our recurrent neural network with recurrent weights $\mathbf{G}$, input-weights $\mathbf{I}$, output-weights $\mathbf{W}$, which are connecting the network to the input-cell $u(t)$ and output-cell $z(t)$ respectively [46].

4.2 Data-set

We proceed with a description of the different inputs and according target-functions used during training. Such a pair $(\text{input}(t), \text{target}(t))$ depicts a data-tuple for a single trial, which is fed through the network to obtain its according output-function $z(t)$.

Input-signals are constant functions of amplitude $u(t) = 2$ to have a significant effect on the RNN's membrane-behaviour without fully suppressing it. As discussed in section 3.6, the chosen scaling factors for $\mathbf{G}$ and the saturating activation function yield membrane potentials $\mathbf{x}(t)$ distributed following $\mathrm{E}(x_i) = 0$ and $\mathrm{Var}(x_i) = 1$. Inputs are fed into the network for different time-spans, which are in a comparable range to the neurons' time-constants $\tau = 100\text{ms}$. We hereby stay in the network's best computational range. Input-signals are drawn from a homogeneous distribution $[50\text{ms}, 250\text{ms}]$ for each data-trial. They are fed to the network at random times drawn from the uniform distribution $t \in [2000\text{ms}, 4000\text{ms}]$.

Diving into the second component of the tuple $(\text{input}(t), \text{target}(t))$, the target-function is used to train the network on the integration- and memory-task. We implement this by using a constant function, where its value is directly correlated to the input-signal's length using $z(t) = 0.007 \cdot \text{len}(u(t))$. This yields expected output-amplitudes in the range $[0.35, 1.75]$, a suitable range for $\mathrm{E}(x_i) = 0$, $\mathrm{Var}(x_i) = 1$ and the output-weights' scaling described in equation 22. The target-function is thus defined as

$$\text{targ}(t) = \begin{cases} 0, & \text{for } t \in [0, \text{input-onset}] \\ \text{undefined}, & \text{for } t \in (\text{input-onset}, 4500\text{ms}] \\ 0.007 \cdot \text{duration of input-signal}, & \text{for } t \in (4500\text{ms}, 7000\text{ms}] \end{cases} \tag{31}$$

By using an initial membrane-state $\mathbf{h_0} = \mathbf{0}$, which is not quite realistic in its biological interpretation, but necessary for achieving a better network performance, the first line in equation 31 is trivially fulfilled. Membrane-activity is first initiated and $\neq \mathbf{0}$ with the input-signal's onset. Following our idea of *reverse-engineering*, we strive to give the least instructions on how to implement the input-signal's integration. We thus do not define the target-function after the input's turn-on and allow for unsupervised dynamics until $t = 4500\text{ms}$. Finally, we implement the working-memory behaviour for time $t \in [4500\text{ms}, 7000\text{ms}]$ by applying a plateau-function, where the height encodes the input-signal's duration and memorises it until the final time-point $N_t = 7000\text{ms}$.

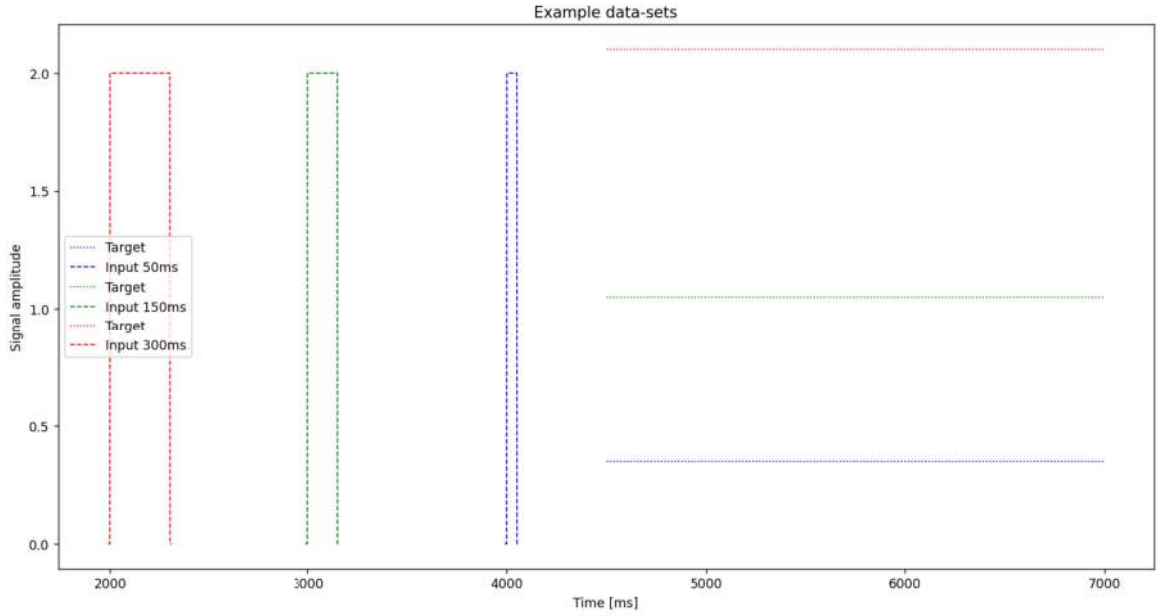


Figure 14: Three exemplifying data-sets: First, an input-signal of 50ms (blue) is fed to the network at = 4000ms. The according target function has a height of $0.007 \cdot 50 = 0.35$ and is evaluated at time $t = 4500$ms. Another example features an input signal of 150ms (green) fed into the network at $t = 3000$ms. The according target function has a plateau at an amplitude of 1.05. A last examples depicts an input signal of duration $t = 300$ms (red) fed to the network after $t = 2000$ms. The target function has a height of 2.1.

4.3 Training

After having constructed our RNN and defined input- and target functions to model integration and working-memory behaviours, we are finally ready to train the network. As described in section 3.5, we use the Backpropagation Through Time algorithm from the PyTorch-Package[47] with an MSE-Loss function and the Adam-Optimizer[48] for computing gradients and according weight-updates. We train parameters $\mathbf{I}$ and $\mathbf{W}$, while leaving the random recurrent connections G_{ij} unchanged. In the course of our backengineering-process, we will additionally feature networks with solely trained input-weights $\mathbf{I}$ and others with only trained output-weights $\mathbf{W}$.

Training is conducted over 100 epochs by using the so-called *Mini-Batch Gradient Descent*. In the framework of this method, we generate a random data-set, which contains 500 data-samples (input(t), target(t)). Each function input(t) is defined as

$$\text{input}(t) = \begin{cases} 0, & \text{for } t \in [0, \text{input-onset}] \\ 2., & \text{for } t \in (\text{input-onset}, \text{input-offset}] \\ 0 & \text{for } t \in (\text{input-offset}, 7000\text{ms}] \end{cases} \tag{32}$$

where the duration of the input is drawn from the random uniform distribution $[50\text{ms}, 250\text{ms}]$ for each data-sample. The according target-function is obtained using equation 31. To obtain a batch, 32 samples are randomly drawn from the denoted set. During the training process, each batch-element is fed through the network to compute its output, which in turn is compared to the according target-function using MSE-Loss. After looping through this process for every element of the batch, back-propagation is conducted by using the mean MSE-Loss of the entire batch, following the process described in section 3.5. Finally, we are constructing the average gradient by backpropagating the mean MSE-Loss, and updating the parameters following equation 24 and a learning rate of 0.001. Using an averaged gradient over randomly constructed batches allows us to avoid falling into local minima and and to obtain more stable convergence to the optimized solution [49].

```
1   number_epochs = 100
2   batch_size = 32
3   data_set = 500
4   learning_rate = .001
5   number_batches = data_set // batch_size
6
7   #defining training data
8   input_set, target_set = data(data_set, *parameters)
9   #defining optimizing algorithm
10  optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
11
12  #looping through epochs
13  for epoch in range(number_epochs):
14
15      #looping through batches
16      for batch in range(number_batches):
17
18          #choose random samples from the data-set to make a batch of 32 samples
19          batch_inputs = input_set[random_indexes]
20          batch_targets = target_set[random_indexes]
21
22          #feeding the inputs trough the network
23          batch_outputs = forward(batch_inputs)
24
25          #calculating MSE-Loss between ouptut and target
26          mean_loss = mse_loss(batch_outputs, batch_targets)
27
28          #calculating the mean gradient using Backpropagation Through Time
29          mean_loss.backward()
30
31          #update parameters
32          optimizer.step()
```

Figure 15: Schematic overview of the training algorithm. We first create a data-set $(\text{input}(t), \text{target}(t))$ containing 500 samples in line 8 and define the Adam optimizer-object [48] in line 10. We proceed with looping through different epochs and batches. Each batch is created by drawing random samples from the data-set in lines 19 and 20. The different inputs are fed to the network to generate according outputs in line 23. The mean MSE-Loss is computed by comparing each obtained output with the according target-function .Finally, the Adam-object's method *.backward()* proceeds with the backward-propagation to obtain the mean gradient, which is finally used to update the net-parameters $\mathbf{I}$ and $\mathbf{W}$ in line 32.

4.4 Post-training results

After the training-procedure described in section 4.3 and illustrated in figure 17, our neural network is able to process input-signals of different durations and store this information using a plateau-like output. Figure 16 depicts the output of a RNN-prototype with trained input- and output-weights. Training leads to the following features:

$$\text{norm}(\mathbf{I}) \approx 21.9 \tag{33}$$

$$\text{norm}(\mathbf{W}) \approx 1.11 \tag{34}$$

$$\text{Loss} \approx 0.00129 \tag{35}$$

The MSE-Loss in equation 35 is the mean of the three data-sets depicted in figure 16. More specifically,

$$\text{MSE-Loss} = \frac{1}{3}((z(t) - \text{targ}_{50}(t))^2 + (z(t) - \text{targ}_{150}(t))^2 + (z(t) - \text{targ}_{300}(t))^2) \tag{36}$$

We are using three data-samples characterised by input-durations of 50ms, 150ms and 300ms with their respective target-functions $\text{targ}_{50}(t)$, $\text{targ}_{150}(t)$, $\text{targ}_{300}(t)$. This choice is motivated by the range of input-durations lying within our RNN's processing capabilities and will be used throughout this thesis.

In the course of the following chapters, we will often mention sets of multiple trained networks. They denote RNNs with identical architecture and training-procedures, as outlined in sections 4.1, 4.2 and 4.3. They only differ in their initial input-weights $\mathbf{I}$ and $\mathbf{W}$ before training. In a more technical wording, weights are drawn from the same distributions 22 and 23, but have different numerical values by choosing different *seeds*.

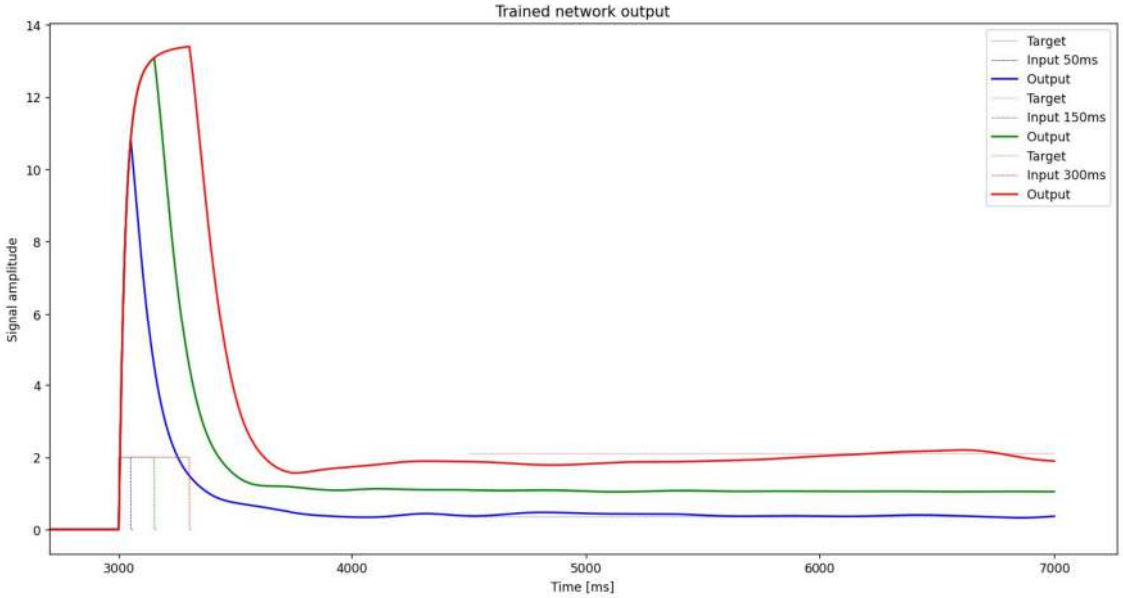
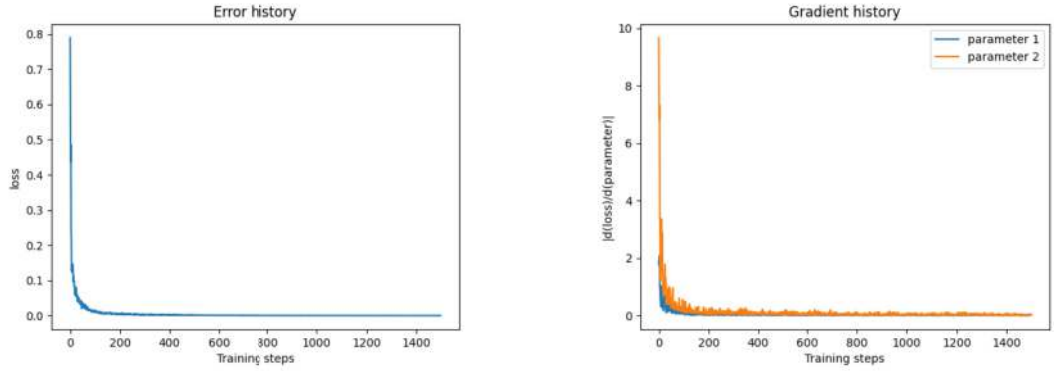


Figure 16: Trained network outputs after training input-and output-weights for 100 epochs. Three input-signals of 50ms (blue, dashed), 150ms (green, dashed) and 300ms (red, dashed) are fed to the network. The according target-functions are colour-coded in the same manner. The according network outputs $z(t) = \mathbf{W}^T\mathbf{r}(t)$ are plotted using a thick line, while each colour refers to the according input- and target-signals.



(a) Error history

(b) Parameters' gradient history

Figure 17: **(a)** Mean MSE-Loss of a batch after each weight-update, calculated as MSE$= (z(t) - \text{target}(t))^2$ for a single trial and averaged over all trials of a batch. The according target function from equation 31 is used for every trial. **(b)** Gradient-norm $|\partial\varepsilon/\partial\theta|$ (using the notation of section 3.5) after each parameter-update. Parameter 1 designates the input-weights $\mathbf{I}$ and Parameter 2 the output-weights $\mathbf{W}$. Each gradient-norm is calculated according to $|\partial\varepsilon/\partial\mathbf{I}| = \sqrt{\sum_{i=1}^{N}(\partial\varepsilon/\partial I_i)^2}$ and $|\partial\varepsilon/\partial\mathbf{w}| = \sqrt{\sum_{i=1}^{N}(\partial\varepsilon/\partial W_i)^2}$ accordingly.

5 Reverse-engineering

Following through the procedure described in the previous chapter, we replicate biological integration- and memory-patterns using artificial neural networks. While we can affirm, that our RNNs successfully carry out these tasks, we do not know their underlying working-mechanism, since it has emerged from training. As explained in section 3.4, by reverse-engineering our RNNs, we aim at understanding how they leverage their dynamics to perform on tasks of biological relevance. Our findings may propose a feasible hypothesis for integration- and working-memory-mechanisms in the brain.

5.1 Intrinsic network dynamics

We begin by considering the intrinsic dynamics of the recurrent network that can be leveraged to solve our integration- and memory-task. This inquiry is motivated by the realization, that dynamics characterized by fixed points or limit cycles lack necessary dynamical freedom to effectively address the task. Indeed, the computational power of a recurrent neural network stems from its chaotic properties [50]. Various methods exist to influence the chaotic behaviour of a recurrent network. On one hand, increasing the coupling-factor g shifts the eigenvalue-spectrum of M to the right (direction of positive x-values), leading the network to exhibit more modes with exponentially increasing behaviour ($\mathrm{Re}(\lambda) > 0$). This results in the network dynamics unfolding in a higher-dimensional space [45]. We will showcase, that increasing the number of neurons N produces a similar effect.

Figure 18 depicts a so-called "attractor". Its dynamics $\mathbf{x}(t)$ converge to a fixed point defined as ${}^{d\mathbf{x}(t)}/_{dt} = 0$ after $\approx$ 3000ms. Therefore, the output $z(t) = \mathbf{W}^T\phi(\mathbf{x}(t))$ can only feature a single value ≈ 0.9, regardless of the different inputs. Thus, the RNN's computational freedom is limited by converging dynamics. We can draw similar conclusions from figure 19, which illustrates a network with converging membrane-dynamics to a limit-cycle. After $\approx$ 8000ms, $\mathbf{x}(t)$ turns into a periodic function, which constrains the network's possible output-functions $z(t)$.

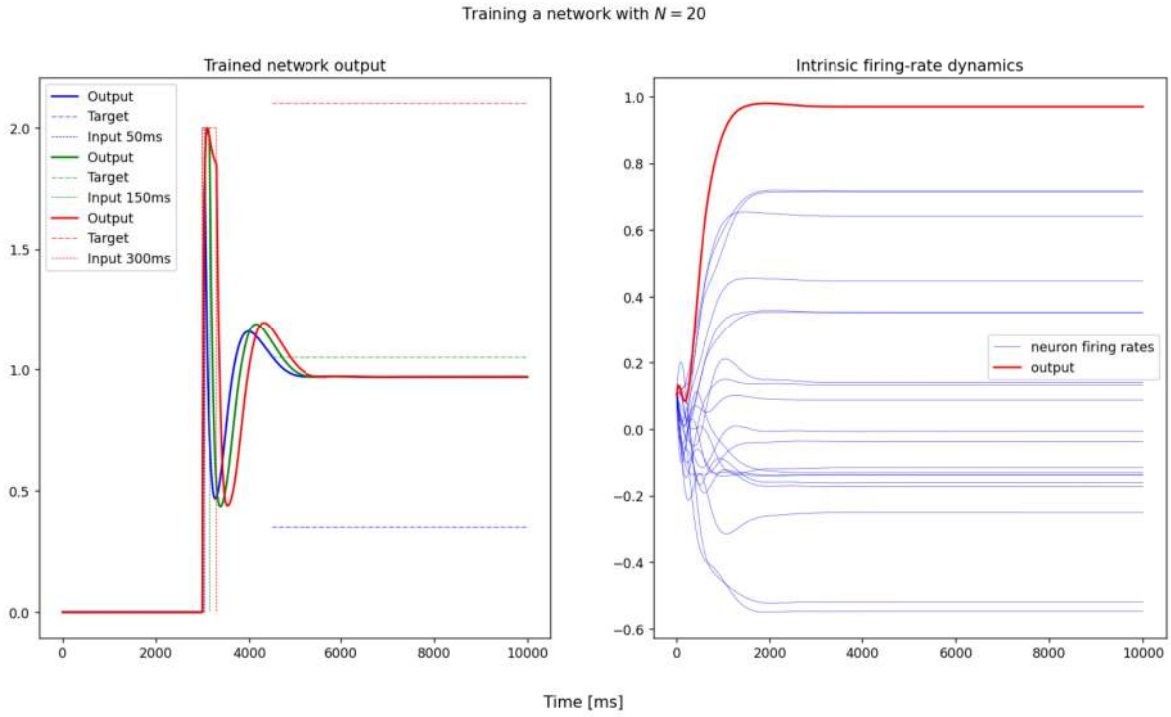


Figure 18: Training a network with $N = 20$ neurons and with intrinsic dynamics that converge to fixed points. **(left)** The network output $z(t) = \mathbf{W}^T\mathbf{r}(t)$ (thick line) is plotted for three different input-signals of 50ms (blue), 150ms (green) and 300ms (red). The dashed lines denote the according target-function and input-signals by using the same colour-code. The MSE-Loss ≈ 0.539 is computed as the average of the MSE-losses for input signals 50ms, 150ms and 300ms and by using the target function in equation 31. The y-axis is unit-less and denotes the signal amplitudes. **(right)** Neuron firing rates $\mathbf{r}(t)$ (blue) and output (red) of the intrinsic network-dynamics. No input is fed to the network and its initial condition is set to $(\mathbf{x}(t = 0))_i = 0.1$ for all neurons $i = 1, ..., N$ and the membrane potential $\mathbf{x}$. The graph demonstrates how firing rates converge to a single fixed point as a result of the chosen initial condition. The y-axis is unit-less and denotes the signal amplitudes.

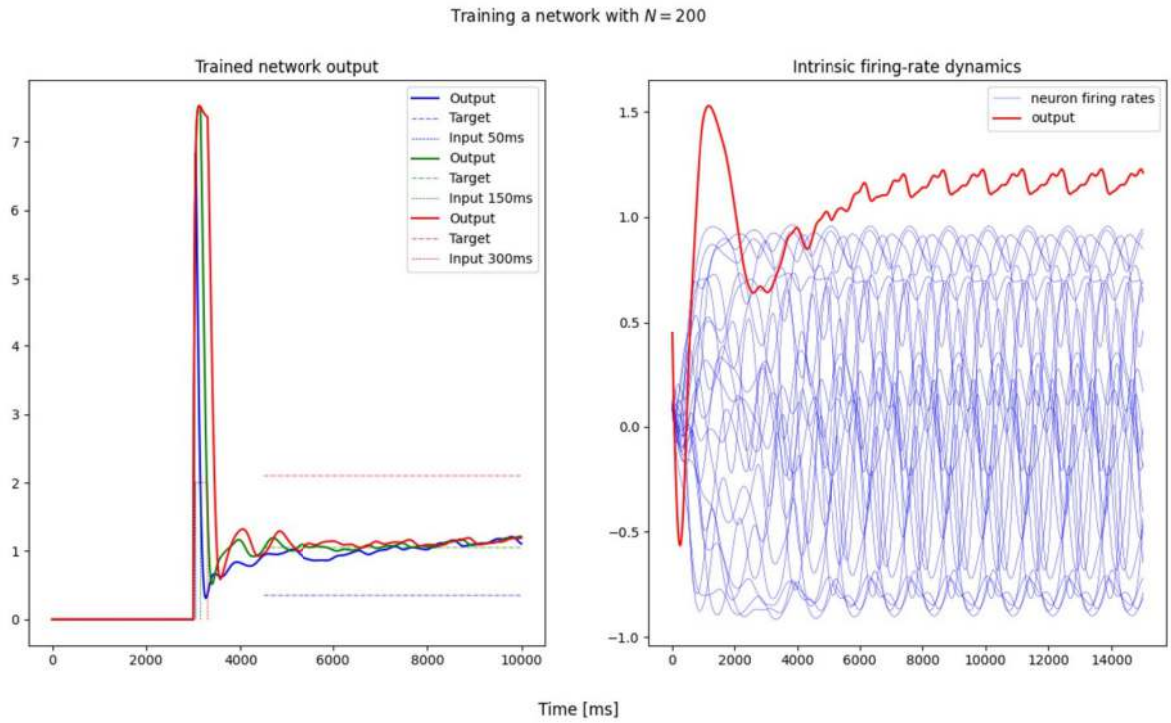


Figure 19: Training a network with $N = 200$ neurons and with intrinsic dynamics that converge to a limit cycle. **(left)** The network output $z(t) = \mathbf{W}^T\mathbf{r}(t)$ (thick line) is plotted for three different input-signals of 50ms (blue), 150ms (green) and 300ms (red). The dashed lines denote the according target-function and input-signals by using the same colour-code. The MSE-Loss ≈ 0.447 is computed as the average of the MSE-losses for input signals 50ms, 150ms and 300ms and by using the target function in equation 31. The y-axis is unit-less and denotes the signal amplitudes. **(right)** Neuron firing rates $\mathbf{r}(t)$ (blue) and output (red) of the intrinsic network-dynamics. No input is fed to the network and its initial condition is set to $(\mathbf{x}(t = 0))_i = 0.1$ for all neuron membrane potentials $i = 1, ..., N$. The y-axis is unit-less and denotes the signal amplitudes.

On the other hand, training the task on a network with chaotic dynamics yields a significantly better result, since it is not constrained by any convergences. Chaotic RNNs thus feature increased computational freedom (figure 20), which in turn improves their performance. This is showcased by figure 21, illustrating MSE-Losses of trained networks with dynamics converging to fixed points ($N = 20$), limit cycles ($N = 200$), and chaotic dynamics ($N = 500$).

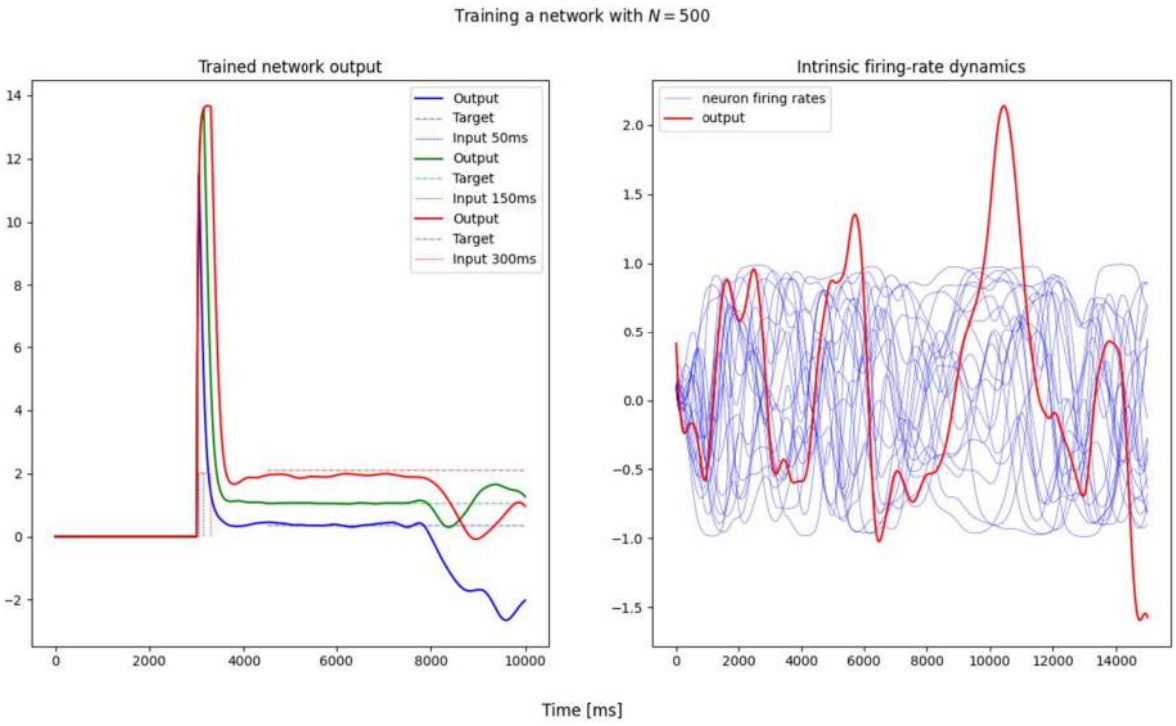


Figure 20: Training a network with $N = 500$ neurons and with chaotic dynamics. **(left)** The network output $z(t) = \mathbf{W}^T\mathbf{r}(t)$ (thick line) is plotted for three different input-signals of 50ms (blue), 150ms (green) and 300ms (red). The dashed lines denote the according target-function and input-signals by using the same colour-code. The MSE-Loss ≈ 0.0105 is computed as the average of the MSE-losses for input signals 50ms, 150ms and 300ms and by using the target function in equation 31. The y-axis is unit-less and denotes the signal amplitudes. **(right)** Neuron firing rates $\mathbf{r}(t)$ (blue) and output (red) of the intrinsic network-dynamics. No input is fed to the network and its initial condition is set to $(\mathbf{x}(t = 0))_i = 0.1$ for all membrane potentials of neurons $i = 1, ..., N$. The y-axis is unit-less and denotes the signal amplitudes.

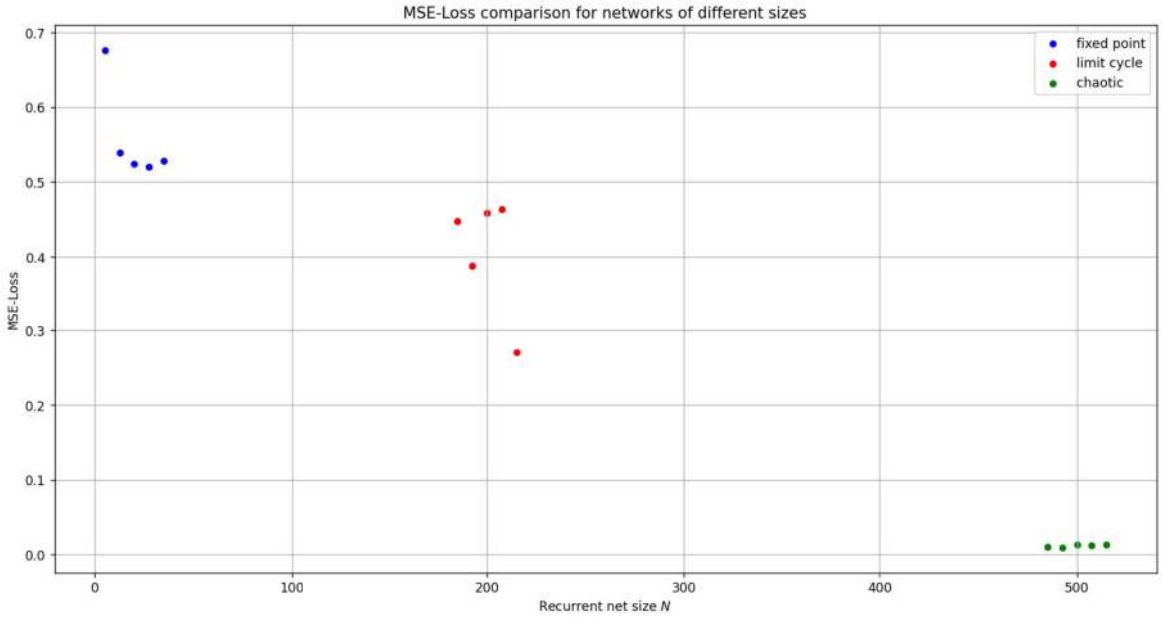


Figure 21: MSE-Loss comparison between networks featuring fixed points ($N = 20$, blue dots), limit cycles ($N = 200$, red dots) and chaotic networks ($N = 500$, green dots). The MSE-Losses are computed using equation 36 as the average over input-durations of 50ms, 150ms and 300ms. Points have been slightly shifted along the x-axis for clearer illustration.

Finally, we explore the concept of "computational freedom" by examining the dimensionality of the space in which intrinsic network dynamics occur. This dimensionality is described by the number of significant directions in which the dynamics vary, which in turn corresponds to the eigenspace of the covariance matrix described in section 3.6 and equation 25. The number of eigenvalues $>> 0$ indicates significant dimensions. Another quantitative measure is the participation ratio defined in equation 30. By conducting PCA-analyses for time-frames of different lengths, we can observe how the dimensionality of the dynamics increases (or stagnates) in time. We initiate the PCA analysis from time $t_1 = 10$s to ensure that the initial condition $\mathbf{x}(t = 0\text{ms})$ no longer impacts the RNN's dynamics.

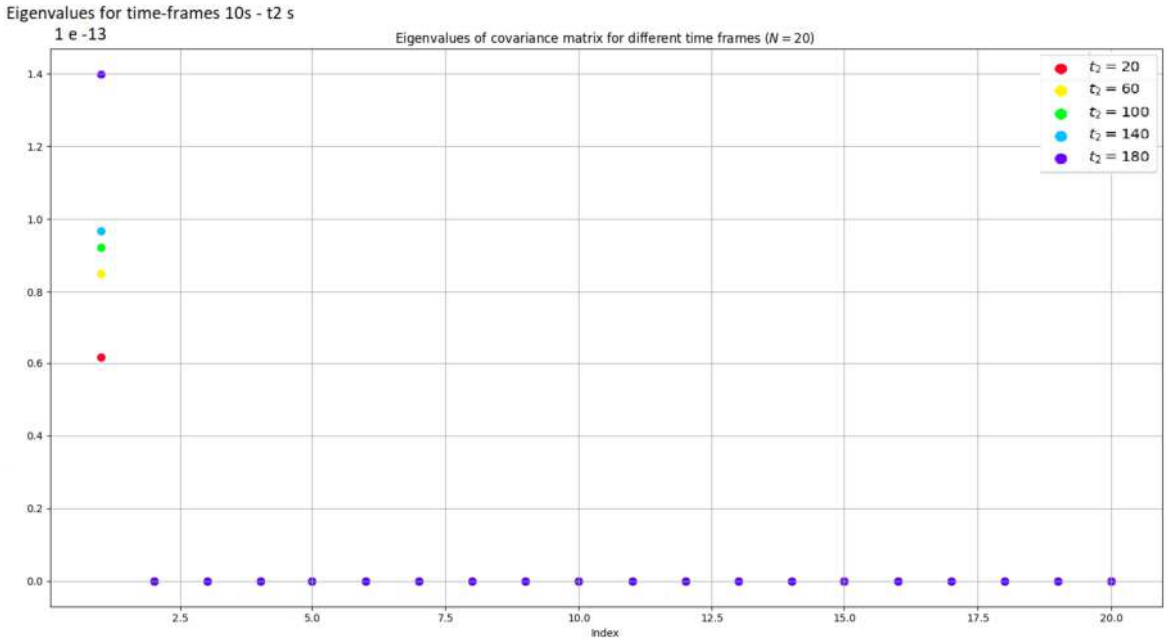


Figure 22: Eigenvalues of the covariance matrix Σ for an RNN with $N = 20$. PCAs are conducted for the $\mathbf{r}(t)$-trajectories of the RNN's intrinsic dynamics, by setting the initial condition for the membrane-potentials $(\mathbf{x_0})_i = 0.1$ for $i = 1, ..., N$ and keeping the input $u(t) = 0$. Time-frames for $t \in [t_1, t_2]$ are defined using $t_1 = 10$s, when the initial condition $\mathbf{x_0}$ has been overcast by the RNN's intrinsic dynamics. The plot shows the eigenvalue-spectra λ_i for $i = 1, ..., N$ for five different PCAs, computed according to equations 25 and 26, and with differing upper limits t_2 of their respective time-frames. Red markers denote the PCA-spectrum computed for $\mathbf{r}(t)$ with $t \in [10\text{s}, 20\text{s}]$. Yellow markers denote the time-frame $t \in [10\text{s}, 60\text{s}]$, green markers denote the time-frame $t \in [10\text{s}, 100\text{s}]$, blue markers denote $t \in [10\text{s}, 140\text{s}]$, and purple markers denote $t \in [10\text{s}, 180\text{s}]$.

Figure 22 depicts the variance of the firing-rates $\mathbf{r}(t)$ for a network with $N = 20$ neurons across different time-frames. This variance is expressed through the eigenvalues of the according covariance-matrix Σ. The dynamics of our attractor quickly converge to a fixed point, such that $\mathbf{r}$-trajectories are static in time and space. Thus, the eigenspectrum in figure 22 only feature eigenvalues $= 0$, while the first eigenvalue has a scale of $\approx 1\text{e} - 13$ expressing fluctuations due to computational precision.

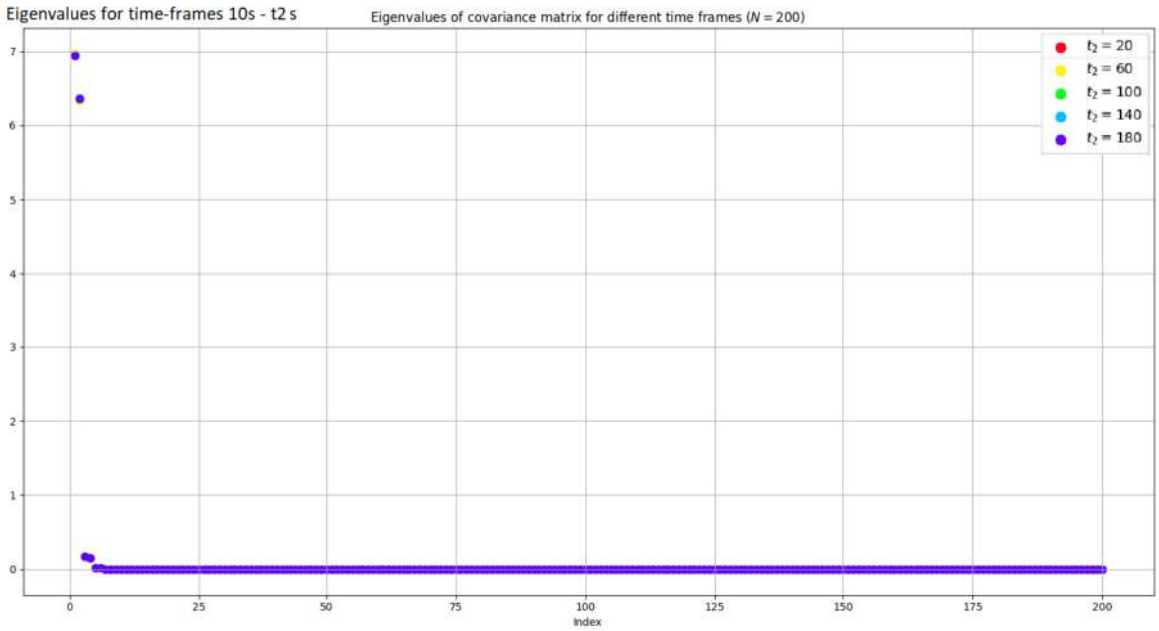
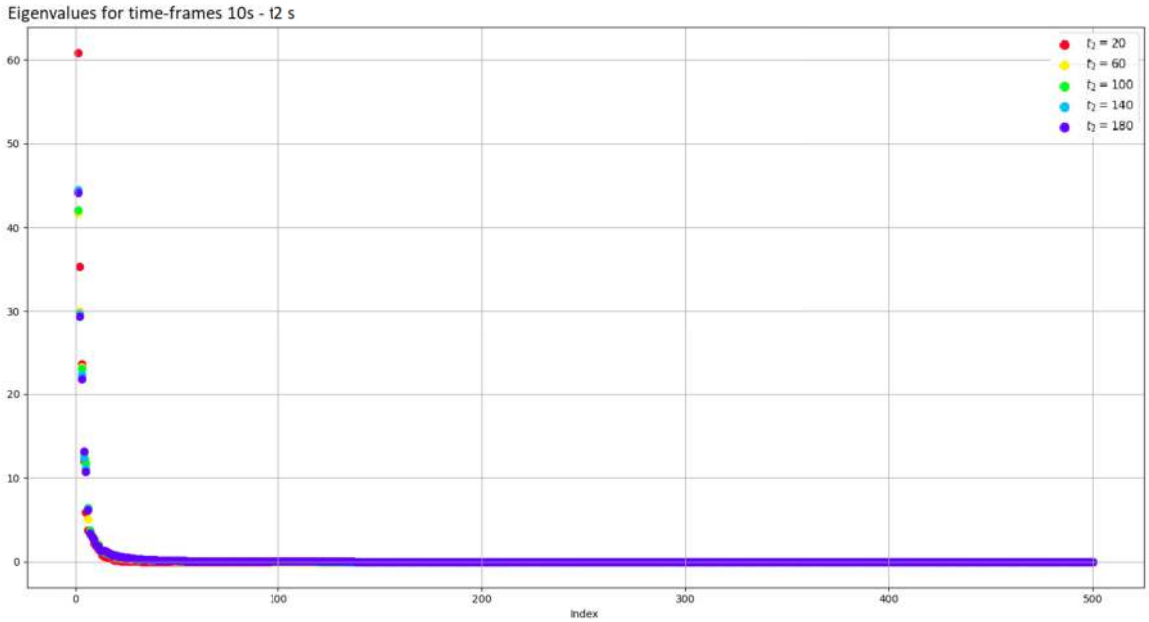
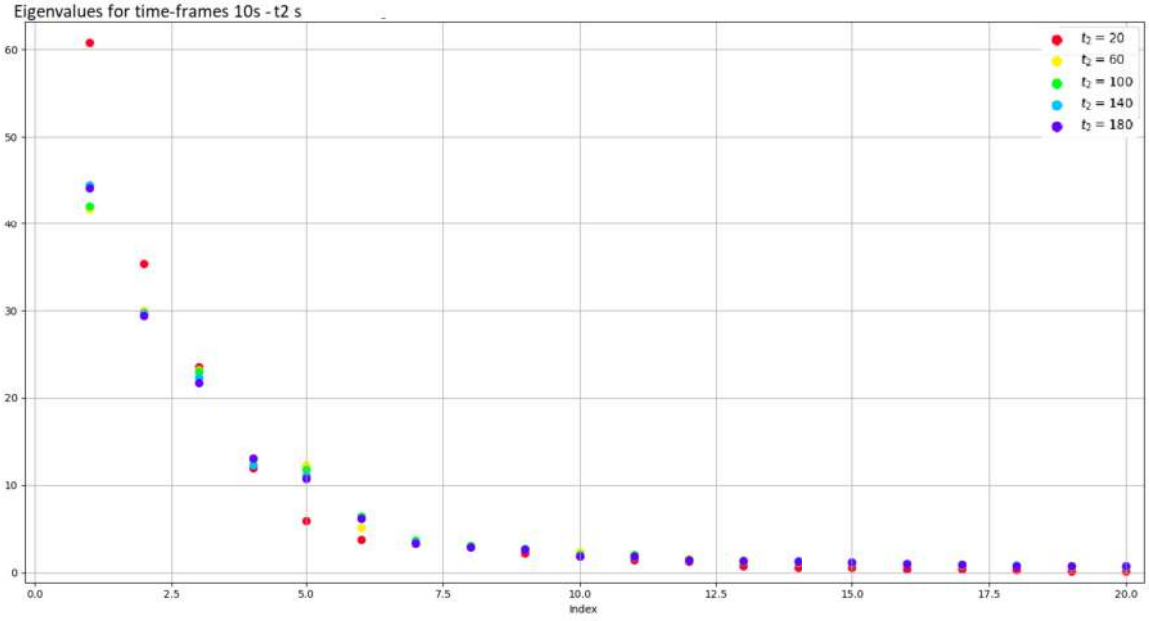


Figure 23: Eigenvalues of the covariance matrix Σ for an RNN with $N = 200$. PCAs are conducted for the $\mathbf{r}(t)$-trajectories of the RNN's intrinsic dynamics, by setting the initial condition for the membrane-potentials $(\mathbf{x_0})_i = 0.1$ for $i = 1, ..., N$ and keeping the input $u(t) = 0$. Time-frames for $t \in [t_1, t_2]$ are defined using $t_1 = 10$s, when the initial condition $\mathbf{x_0}$ has been overcast by the RNN's intrinsic dynamics. The plot shows the eigenvalue-spectra λ_i for $i = 1, ..., N$ for five different PCAs, computed according to equations 25 and 26, and with differing upper limits t_2 of their respective time-frames. Red markers denote the PCA-spectrum computed for $\mathbf{r}(t)$ with $t \in [10\text{s}, 20\text{s}]$. Yellow markers denote the time-frame $t \in [10\text{s}, 60\text{s}]$, green markers denote the time-frame $t \in [10\text{s}, 100\text{s}]$, blue markers denote $t \in [10\text{s}, 140\text{s}]$, and purple markers denote $t \in [10\text{s}, 180\text{s}]$. As all spectra yield similar results, markers overlap and thus only purple markers are distinguishable.

Moving on to a network with $N = 200$ neurons, its dynamics features a limit cycle, and therefore firing-rate trajectories with increased variance. We can infer this figure 23, which features Σ's eigenspectrum. Two significant eigenvalues $\lambda_1 \approx 6.95$ and $\lambda_2 \approx 6.5$ are distinguishable, which account for most of the dynamics' variance. Thus, the network's computational capacities is approximately restricted to a 2-dimensional space. Furthermore, the results for the PCAs over different time frames do not differ from each other. This embodies the nature of the limit cycle - regardless of how long we observe the network for, its dynamics does not change.



(a) Eigenvalue-spectrum λ_i for $i = 1, ..., N$



(b) Eigenvalue-spectrum λ_i for $i = 1, ..., 20$

Figure 24: **(a)** Eigenvalues of the covariance matrix Σ for an RNN with $N = 500$. PCAs are conducted for the $\mathbf{r}(t)$-trajectories of the RNN's intrinsic dynamics, by setting the initial condition for the membrane-potentials $(\mathbf{x_0})_i = 0.1$ for $i = 1, ..., N$ and keeping the input $u(t) = 0$. Time-frames for $t \in [t_1, t_2]$ are defined using $t_1 = 10$s, when the initial condition $\mathbf{x_0}$ has been overcast by the RNN's intrinsic dynamics. The plot shows the eigenvalue-spectra λ_i for $i = 1, ..., N$ for five different PCAs, computed according to equations 25 and 26, and with differing upper limits t_2 of their respective time-frames. Red markers denote the PCA-spectrum computed for $\mathbf{r}(t)$ with $t \in [10\text{s}, 20\text{s}]$. Yellow markers denote the time-frame $t \in [10\text{s}, 60\text{s}]$, green markers denote the time-frame $t \in [10\text{s}, 100\text{s}]$, blue markers denote $t \in [10\text{s}, 140\text{s}]$, and purple markers denote $t \in [10\text{s}, 180\text{s}]$. **(b)** Zooming into the significant eigenvalues $\lambda_i \not\approx 0$ of figure 24.

We conclude our analysis with our prototype RNN of $N = 500$ neurons, exhibiting chaotic dynamics. Figure 24 reveals a spectrum with a set of ≈ 20 significant eigenvalues embodying the number of significant dimensions in which the dynamics take place. Furthermore, we observe a "flattening" of the spectra for increasing time-frames, which depicts the membrane-trajectories gradually expanding and occupying more state-space dimensions.

The qualitative conclusions drawn above are quantified using the participation ratio from equation 30. Figure 25 demonstrates the results for the the three discussed networks featuring different long-term behaviours. As already elucidated, the trajectory featured by a $N = 20$ network is a single point in space. We therefore obtain a participation ratio of $D = 1$ regardless of the time-frame of observation, while this value rather denotes variances due to computational precision than dynamical features. A similar conclusion can be drawn from the $\mathbf{r}(t)$-trajectories of a network dominated by a limit-cycle. We calculate a participation ratio of $D \approx 2$, which accounts for the two significant eigenvalues distinguished in figure 23. The converging property is highlighted by D's independence of the considered time-frame. Finally, since chaotic dynamics does not converge to a certain pattern, its number of occupied space-dimensions steadily increases over larger periods of observation. However, dynamics does not seem to unlimitedly expand, since D saturates at ≈ 7.0. The greater "computational power" of chaotic networks might thus (partly) be explained by their higher degrees of dynamical freedom and trajectories' dimensionality. It is furthermore shown, that the saturation threshold increases with higher number of neurons N and coupling strengths g [50], which explains the increasing computational power when using larger RNNs.

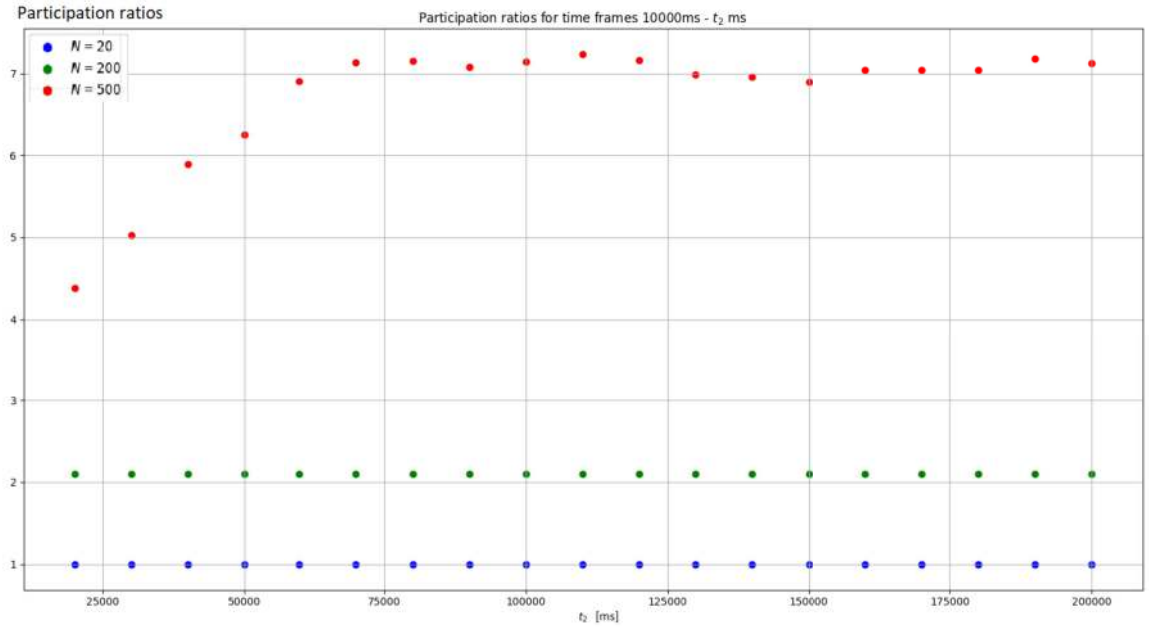


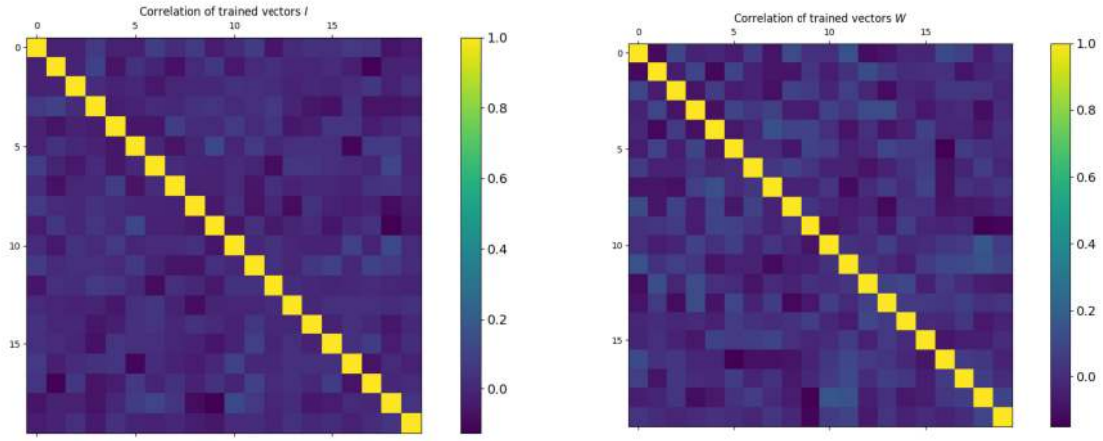
Figure 25: Participation ratios D for PCAs over increasing time-frames and for RNNs of different sizes N. PCAs are conducted for the $\mathbf{r}(t)$-trajectories of the RNN's intrinsic dynamics, by setting the initial condition for the membrane-potentials $(\mathbf{x_0})_i = 0.1$ for $i = 1, ..., N$ and keeping the input $u(t) = 0$. Time-frames for $t \in [t_1, t_2]$ are defined using $t_1 = 10.000$ms, when the initial condition $\mathbf{x_0}$ has been overcast by the RNN's intrinsic dynamics. The x-axis denotes the according upper limits. For every time-frame, the according participation-ratio D is computed according to equation 30. The blue markers denote the results of RNNs of size $N = 20$. The green markers denote a network-size of $N = 200$ and blue markers denote $N = 500$

5.2 Statistical performance analysis

Before diving into a more detailed analysis of our RNN's strategy to integrate and memories input-signals, we begin by a broad overview and rather statistical approach to understanding which features affect the network's performance.

5.2.1 Independence of solutions

We begin by asking ourselves the plainest question of all; "Are we looking for a single and specific solution, or is the RNN's working principle broader than this ?". To address this question, we train 20 prototypes with different initial conditions for the weights $\mathbf{I}$ and $\mathbf{W}$. Figure 26 showcases that solutions $(\mathbf{I}, \mathbf{W})$ are not unique, since initiating the training process from different initial conditions leads to uncorrelated results.

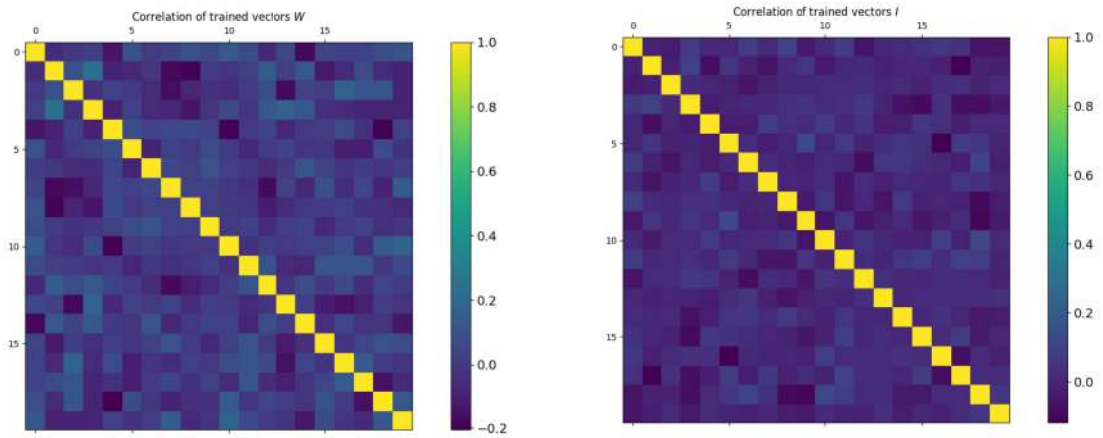


(a) Weight-correlation $\mathbf{I}$ (b) Weight-correlation $\mathbf{W}$

Figure 26: Training the input-weights $\mathbf{I}$ and output-weights $\mathbf{W}$ with different initial conditions for 20 models. The pairwise correlation between the different results found by training is computed according to the Pearson method $\mathrm{Cor}(\mathbf{I_a}, \mathbf{I_b}) = \sum_{i=1}^{N}(I_a^i - <\mathbf{I_a}>)(I_b^i - <\mathbf{I_b}>)/\sigma(\mathbf{I_a})\sigma(\mathbf{I_b})$.

5.2.2 The role of input- and output-weights

We maintain our broader approach and analyse which weights play a more important role in solving our task. We feature 20 additional prototypes by only training the input-weights $\mathbf{I}$ and keeping the output-weights $\mathbf{W}$ fixed to their initial conditions, and contrariwise. Figure 27 reveals that this scenario as well yields independent solutions.



(a) Weight-correlation for trained $\mathbf{W}$ (b) Weight-correlation for trained $\mathbf{I}$

Figure 27: (**a**) Training 20 networks with fixed input-weights $\mathbf{I}$, which are randomly distributed according to equation 23. The resulting trained output-weights $\mathbf{W}$ are pairwise correlated following the Pearson-method $\mathrm{Cor}(\mathbf{I_a}, \mathbf{I_b}) = \sum_{i=1}^{N}(I_a^i - <\mathbf{I_a}>)(I_b^i - <\mathbf{I_b}>)/\sigma(\mathbf{I_a})\sigma(\mathbf{I_b})$. (**b**) Training 20 networks with fixed output-weights $\mathbf{W}$, which are randomly distributed according to equation 22. The resulting input-weights $\mathbf{I}$ are pairwise correlated in the same manner.

We move on by displaying two examples of networks with different training-schemes. Figure 28 features the output of a RNN with trained output-weights $\mathbf{W}$, while the input-weights $\mathbf{I}$ are randomly distributed using equation 23, and not trained (thus kept fixed). On the other hand, figure 30 depicts the output of a RNN with trained $\mathbf{I}$-weights and randomly distributed $\mathbf{W}$-weights according to equation 22. This time, we only train the input-weights and do not alter output-weights. A comparison of both figures 23 and 28 suggests that output-weights play a more important role in integrating and memorising input-signals than input-weights.

Additionally, the training process encounters more difficulties to find an optimal solution when weight-modifications are restricted to inputs instead of outputs: Figure 29 (a) features MSE-Loss decreasing at a faster rate and reaching a plateau at ≈ 0.4 after ≈ 400 training steps, while the gradient-norm has a mean variance of ≈ 0.5. Conversely, figure 31 showcases the training process for fixed output-weights. In this case, the MSE-Loss plateaus at ≈ 0.9 after 800 training steps and exhibits higher fluctuations. Additionally, gradient norms also exhibit a higher mean variance of ≈ 0.2. Thus, training the output seems to be a better performing training-method.

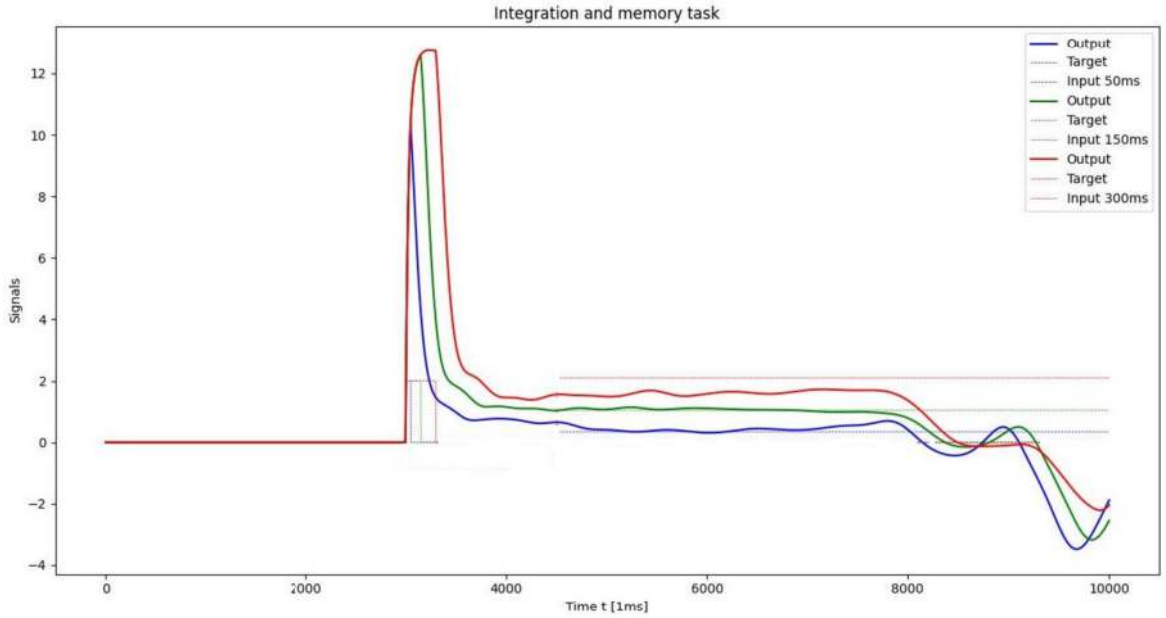
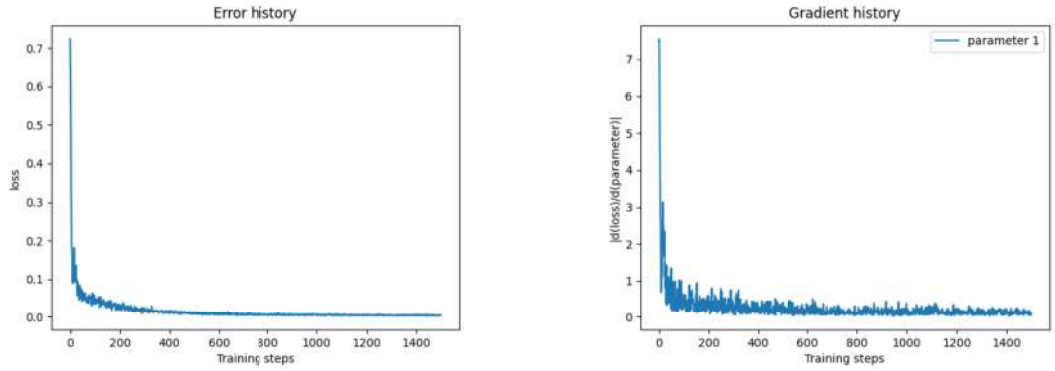


Figure 28: Example output of an RNN with trained output-weights $\mathbf{W}$ and fixed input-weights $\mathbf{I}$ randomly distributed following equation 23. The outputs are plotted for three different signals of 50ms, 150ms and 300ms in blue, green and red respectively. The according target-functions and input-signals are dashed and equally colour-coded.



(a) Error history (b) Gradient history

Figure 29: Training an RNN by optimising output-weights $\mathbf{W}$ and fixing the input weights $\mathbf{I}$ according to equation 23 for 100 epochs. The x-axis denotes the number of weight-updates during the training. (**a**) Mean MSE-Loss of a batch after each weight-update. (**b**) Gradient norm $|\partial\varepsilon/\partial\mathbf{w}| = \sqrt{\sum_{i=1}^{N}(\partial\varepsilon/\partial W_i)^2}$ after each update.

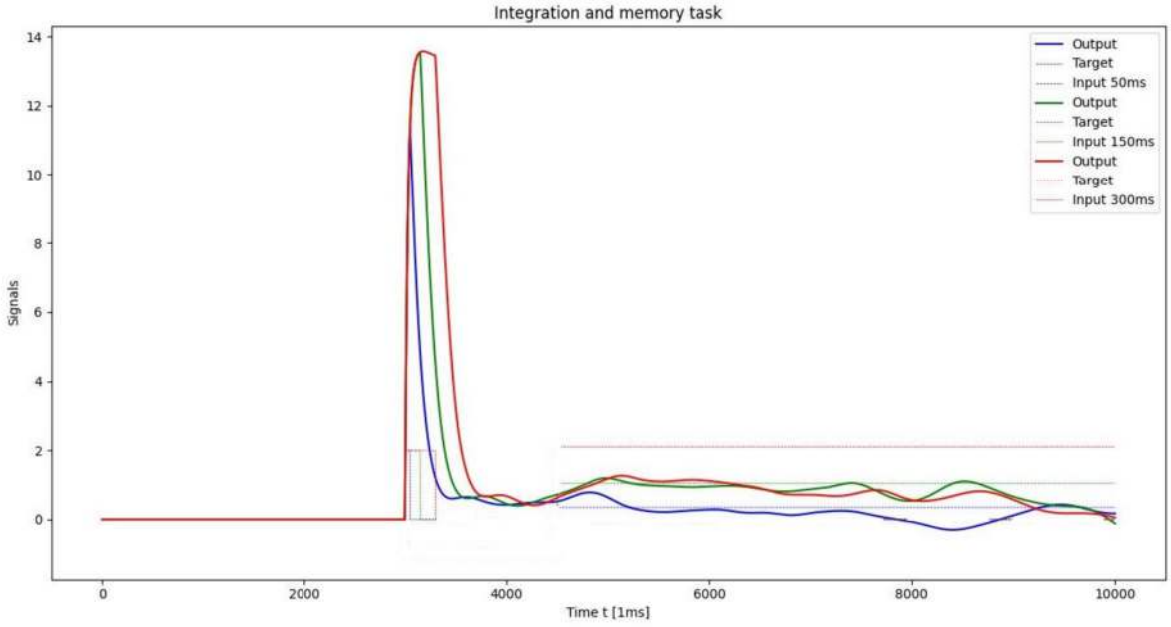
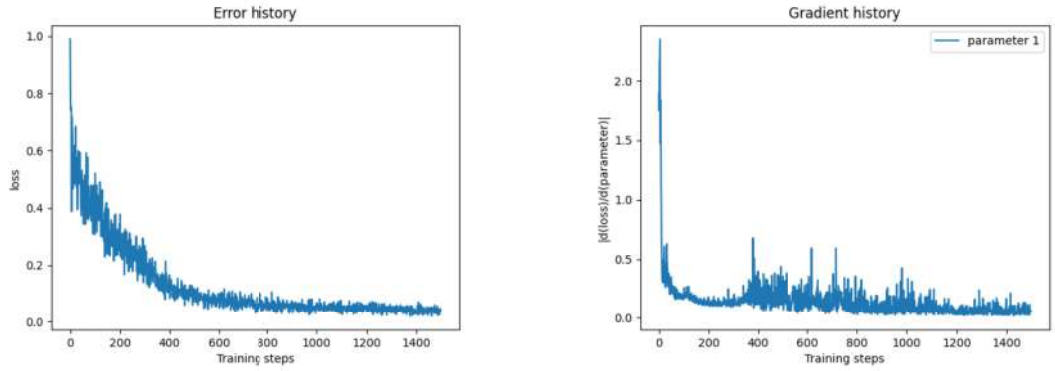


Figure 30: Example output of an RNN with trained input-weights $\mathbf{I}$ and fixed output-weights $\mathbf{W}$ randomly distributed following equation 22. The outputs are plotted for three different signals of 50ms, 150ms and 300ms in blue, green and red respectively. The according target-functions and input-signals are dashed and equally colour-coded.



(a) Error history (b) Gradient history

Figure 31: Training an RNN by optimising input-weights $\mathbf{I}$ and fixing the output-weights $\mathbf{W}$ according to equation 22 for 100 epochs. The x-axis denotes the number of weight-updates during the training. (**a**) Mean MSE-Loss of a batch after each weight-update. (**b**) Gradient norm $|\partial\varepsilon/\partial\mathbf{I}| = \sqrt{\sum_{i=1}^{N}(\partial\varepsilon/\partial I_i)^2}$ after each update.

We proceed by quantifying the previous qualitative conclusions. We train 20 RNNs by optimizing both weights $\mathbf{W}$ and $\mathbf{I}$. 20 additional prototypes only feature trained output-weights, while 20 other RNNs are trained by exclusively optimising input-weights. Finally, a batch of 20 networks with random input-and output-weights is evaluated for reference. All RNNs are trained as outlined in section 4.3. Computing the mean MSE-Loss according to equation 36 for every set yields the following results:

Trained input-and output-weights:

$$\text{Loss} = 0.0194 \pm 0.009 \tag{37}$$

Trained output-weights and random input-weights:

$$\text{Loss} = 0.08 \pm 0.06 \tag{38}$$

Trained input-weights and random output-weights:

$$\text{Loss} = 0.5 \pm 0.3 \tag{39}$$

Random input-weights and output-weights:

$$\text{Loss} = 1.9 \pm 0.7 \tag{40}$$

Figure 32 illustrates these results in a boxplot. We observation that training output-

weights is more crucial to successfully modelling integration- and memory-behaviours, than training input-weights.

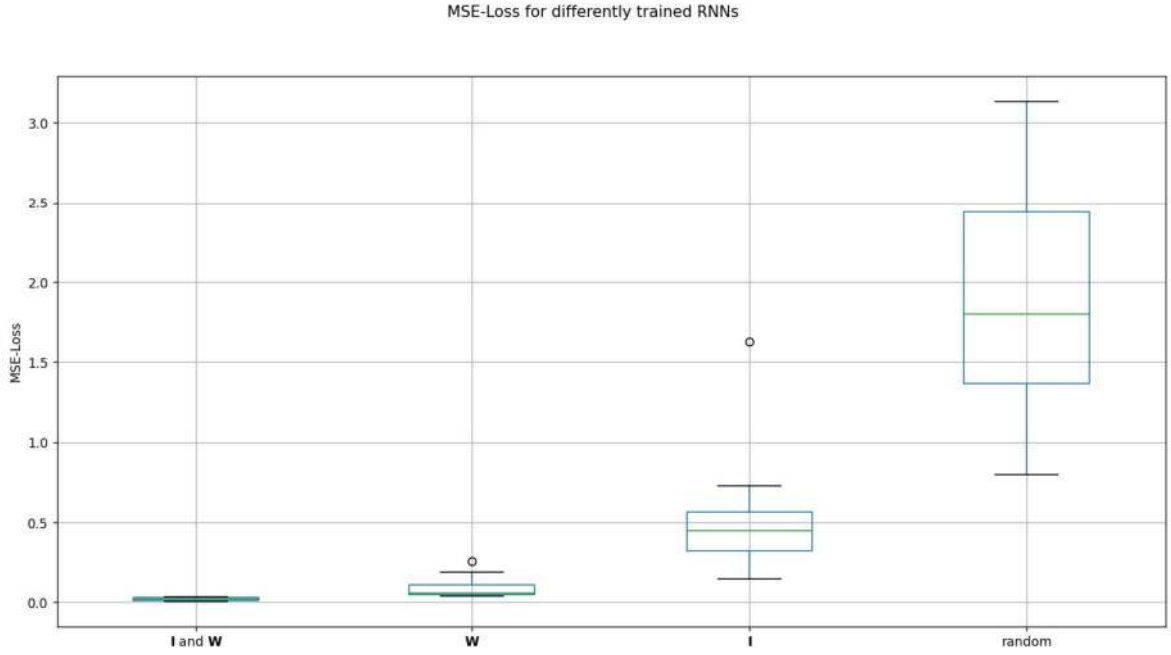


Figure 32: Boxplot featuring the MSE-Loss for four distinct series made up of 20 different models each. Median values are marked by green lines, upper and lower quartiles are denoted by blue lines and whiskers denote 1.5·Interquartile range (distance between the upper and lower quartiles). The "**I** and **W**" label denotes networks with trained input- and output-weights. Networks with trained input-weights and fixed output-weights are denoted by the label "**I**", while prototypes with trained output-weights and fixed input-weights are denoted by "**W**". Finally, RNNs with non-trained random weights drawn from distributions 22 and 23 for input- and output-weights respectively are characterised by the label "random". MSE-Loss is computed in comparison to the target-function defined in equation 31 for three input-signals of durations 50ms, 150ms and 300ms. The final value is is obtained by taking the mean over the three results.

Finally, we conclude this chapter with two important insights: First of all, solutions are not unique. According to the chosen initial conditions before training, optimization converges to different final results, which perform similarly. Second, it seems that the output-weights $\mathbf{W}$ are more important for integration- and memory-functions than input-weights $\mathbf{W}$.

5.3 Analysis in eigenspace

After the preceding introductory observations we are diving deeper into the concept of *reverse-engineering* described in section 3.4, and investigate the details of how trained RNNs leverage their chaotic dynamics to model integration and memory-behaviours.

As outlined in section 3.3, we begin by assuming that our RNN relies on the proposed concept of fine-tuning modes in eigenspace to achieve signal integration. However, an important distinction between our model and the theoretical approach lies in the fact that our RNN utilizes a completely random recurrent network that remains unchanged during training, whereas the theoretical counterpart involves modifying and refining the recurrent weights.

We hypothesize, that training the input weights $\mathbf{I}$ translates into selecting which modes to excite most and which to suppress. According to equation

$$\tau \frac{dx_i}{dt} = -x_i + g \sum_{j=1}^{N} G_{ij}\phi(x_j) + \sum_{k=1}^{M} I_{ij}u_k(t) \tag{41}$$

the network's intrinsic dynamics is dominated by the introduced input as a result of our scaling arguments in section 3.5 combined with $\mathrm{E}(\mathbf{x}(t)) = 0$ $\mathrm{Var}(\mathbf{x}(t)) = 1$.

In this case, we assume

$$g_G \sum_{j=1}^{N} G_{ij}\phi(\tilde{x_j}) << \tilde{I}_i \tag{42}$$

(writing $\mathbf{x} = \sum_{i=1}^{N} \tilde{x}_i \mathbf{v_i}$ and $\mathbf{I} = \sum_{i=1}^{N} \tilde{I}_i \mathbf{v_i}$ in M's in eigenvector-basis $\{\mathbf{v_i}\}$), such that the mode dynamics become

$$\tilde{x}_i(t) = (1 - \exp(-t(\lambda_i - 1)/\tau))\tilde{I}_i \tag{43}$$

We hypothesize that training relies on picking a $\tilde{I}_i$-component related to an eigenvalue $\lambda_i \approx 1$ while suppressing all other modes using $\tilde{I}_i \approx 0$ during the integration phase. This would infer that integration is conducted by only a single mode. To test this hypothesis, we transform the trained vector $\mathbf{I}$ into its eigenspace, using equation 7. We generate a random vector $\mathbf{I}$ using the Gaussian distribution in equation 23), and transform it to the same eigenspace.

However, figure 33 demonstrates that training does not seem to rely on picking out certain modes and suppressing others. No clear distinction between the random model and its trained counterpart can be made.

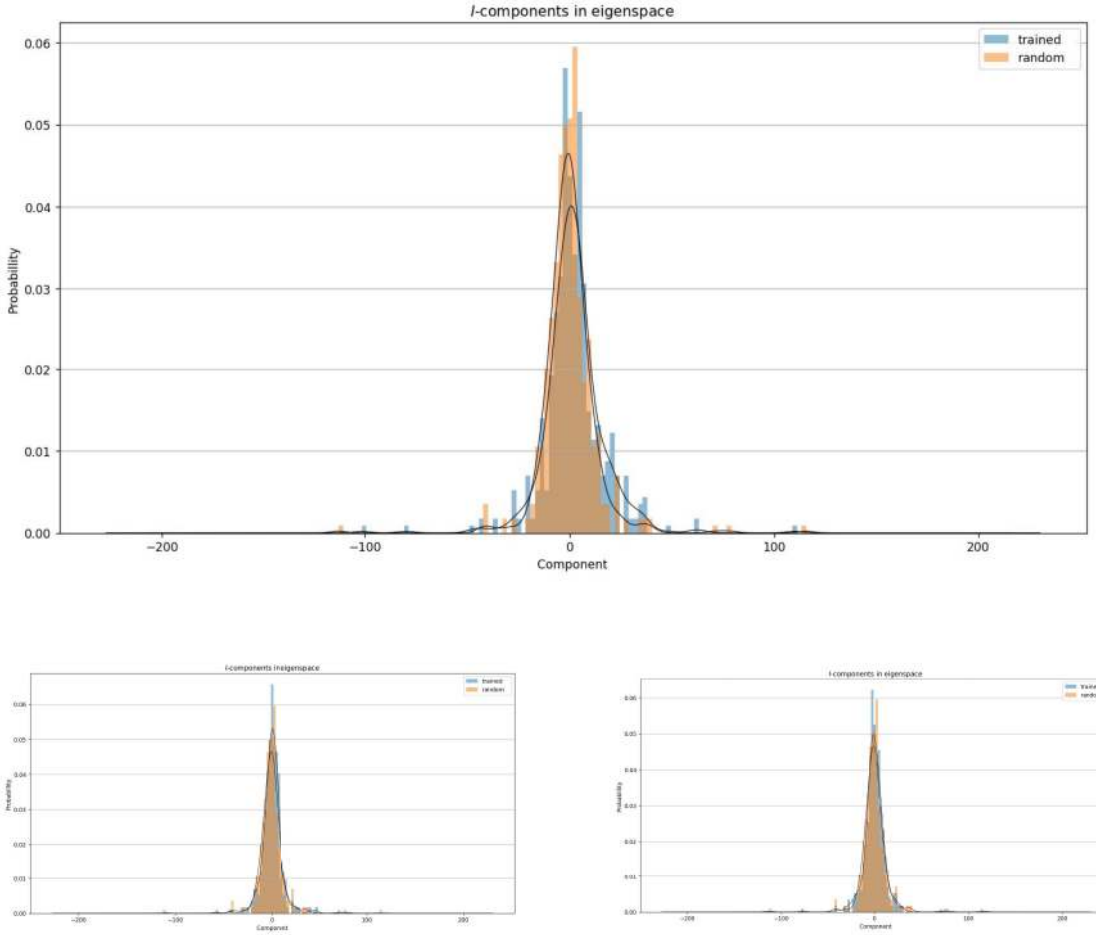


Figure 33: Three different trained models. For each one, the trained vector $\mathbf{I}$ is projected to eigenspace using equation 7. The according basis-vectors are obtained from the eigenspectrum of the stability-matrix in equation 4. A histogram of the resulting components x_i for $\mathbf{I} = \sum_{i=1}^{N} x_i \mathbf{v}_i$ is plotted. They are labelled as "'trained". For comparison, a random vector $\mathbf{I}$ is drawn from the distribution 23 and projected to the same eigenspace. Its components are labeled as "random"

In a similar fashion, we hypothesize that training might finetune the output-weights $\mathbf{W}$ to selectively read out only certain modes. We repeat the same procedure and compare trained weights to random weights drawn from the Gaussian distribution in equation 22. However, figure 34 does not reveal any significant difference between both weights.

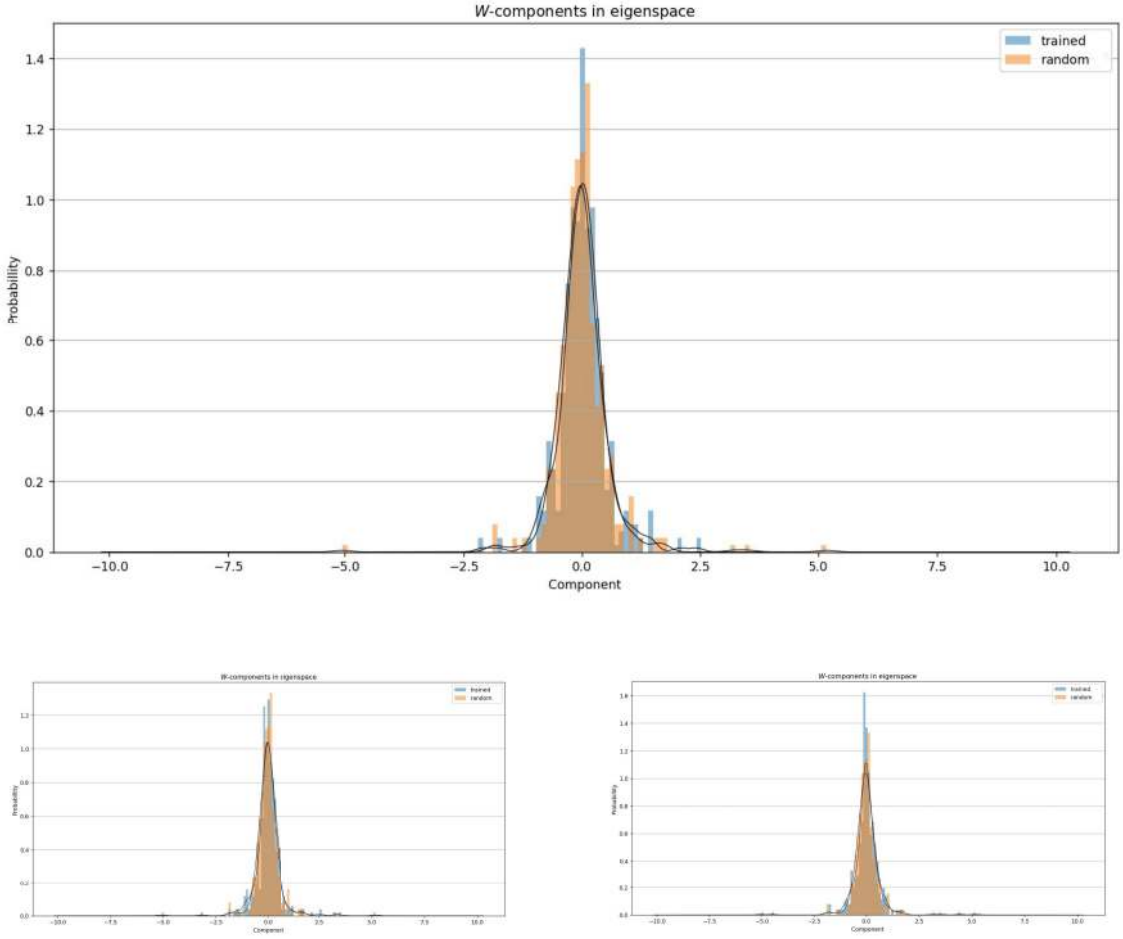


Figure 34: Three different trained models. For each one, the trained vector $\mathbf{W}$ is projected to eigenspace using equation 7. The according basis-vectors are obtained from the eigenspectrum of the stability-matrix in equation 4. A histogram of the resulting components x_i for $\mathbf{W} = \sum_{i=1}^{N} x_i \mathbf{v}_i$ is plotted. They are labelled as "'trained". For comparison, a random vector $\mathbf{W}$ is drawn from the distribution 22 and projected to the same eigenspace. Its components are labelled as "random"

In conclusion, the theoretical picture of integrator-networks described in section 3.3 does not apply to our RNNs.

5.4 Firing-rate trajectories in Principal Component-space

Based on our findings in section 5.3, it appears that linear analysis does not yield any interesting insights. In this section, we change from eigenspace to Principal Component-space. It is characterized by axes along the directions of maximum variance in the network dynamics.

We begin by setting the framework of our analyses, by dividing the integration- and memory-functions into their constituent parts. Figure 35 depicts the output of a

trained RNN, where we discern the following features: The integration-period occurs when inputs are fed to and processed by the RNN. It is followed by the memory-period when the output exhibits a plateau. Furthermore, integration can be subdivided into input-dominated dynamics (when $u(t) \neq 0$) and intrinsic dynamics (once $u(t) = 0$). According to the arguments in equations 42 and 43, $\mathbf{I} \cdot u(t)$ prevails over the remaining $-x_i + \sum_{j=1}^{N} G_{ij}\phi(x_j(t))$-term in the RNN's dynamical equation 2, such that $^{d\mathbf{x}(t)}/_{dt}$ is "input-dominated". By the same argument, the intrinsic RNN-dynamics $-x_i + \sum_{j=1}^{N} G_{ij}\phi(x_j(t))$ take over after $u(t) = 0$ during the "intrinsic phase". Furthermore, the network does not go through training during period. Firing rates are therefore not required to project a plateau-like function $z(t) = \mathbf{W}^T\mathbf{r}(t)$, and the RNN has the flexibility to process information about input-duration in any form or manner before entering the memory-phase.

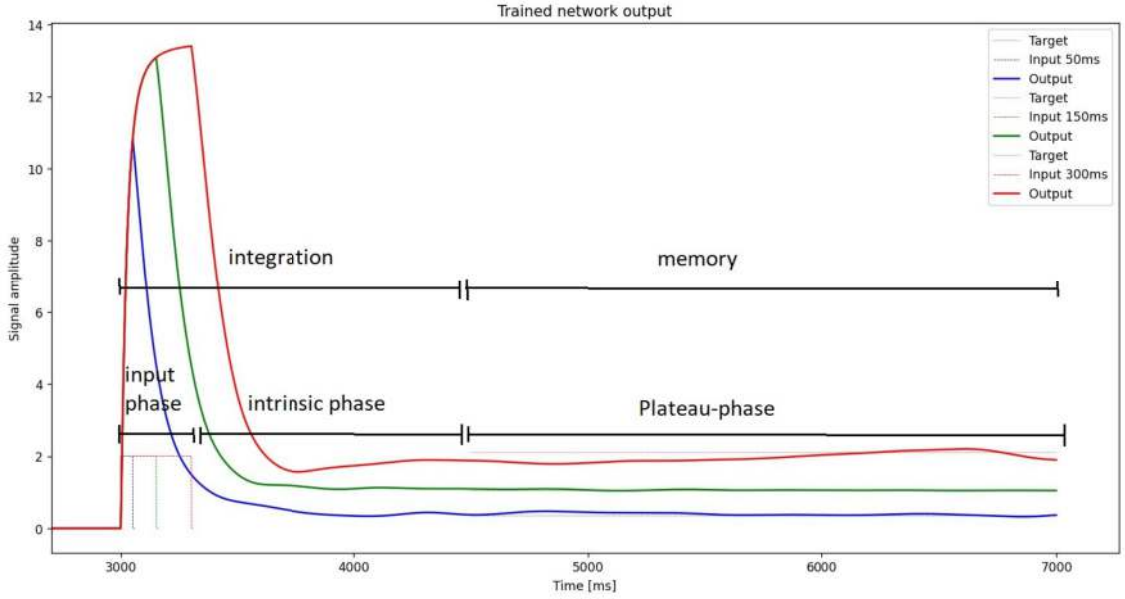


Figure 35: Network-outputs for three $u(t)$-signals of durations 50ms (blue), 150ms (green) and 300ms (red). The according input-signals and target-functions are equally colour-coded. The different phases are illustrated: The input-dominated phase is defined for $t \in (3000\text{ms}, \text{input-offset}]$, the intrinsic phase denotes the range $t \in (\text{input-offset}, 4500\text{ms}]$ and finally the plateau-phase for $t \in (4500\text{ms}, 7000\text{ms}]$. The integrating-task and memory task are implemented for $t \in (3000\text{ms}, 4500\text{ms}]$ and $t \in (4500\text{ms}, 7000\text{ms}]$ respectively.

Before understanding the working-mechanisms of integration and memory, it is useful to obtain a visual overview of the governing dynamics. We proceed by illustrating the firing-rates $\mathbf{r}(t)$ in PC-space, which allows us to capture a large fraction of their variance in a reduced number of dimensions.

We conduct a Principal Component Analysis on the RNN's intrinsic dynamics, characterised by $u(t) = 0$ and $x_i(t = 0) = 0.1$ for the membrane potentials of all neurons $i = 1, ..., N$. This entails computing the covariance-matrix Σ (equation 25) of the resulting firing-rate trajectories $\mathbf{r}(t)$. By doing so for $t \in [7000\text{ms}, 1200\text{ms}]$, we ensure that the initial condition $x_i(t = 0) = 0.1$ has lost its direct effect on the dynamics. The eigenvectors of Σ define the Principal Components, while their respective eigenvalues λ_i account for the total variance of $\mathbf{r}(t)$ along the direction $\mathbf{v_i}$. Figure 36 illustrates the resulting eigenvalue-spectrum. We proceed by projecting the $\mathbf{r}(t)$-trajectories on the three most significant Principal Components $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$ (featuring the highest eigenvalues) by using

$$r_i(t) = \mathbf{r}(t) \cdot \mathbf{v_i} \tag{44}$$

The variance-fraction captured by the three first components is

$$\text{frac} = {}^{\lambda_1+\lambda_2+\lambda_3}\!/_{\sum_{1=1}^{N} \lambda_i} \tag{45}$$

We are additionally interested in the projection of the input-weights, and project the vector $\mathbf{I}$

$$I_i = \mathbf{I} \cdot \mathbf{v_i}, \tag{46}$$

which captures a fraction

$$\text{frac} = {}^{I_1+I_2+I_3}\!/_{\text{norm}(\mathbf{I})} \tag{47}$$

of its total norm.

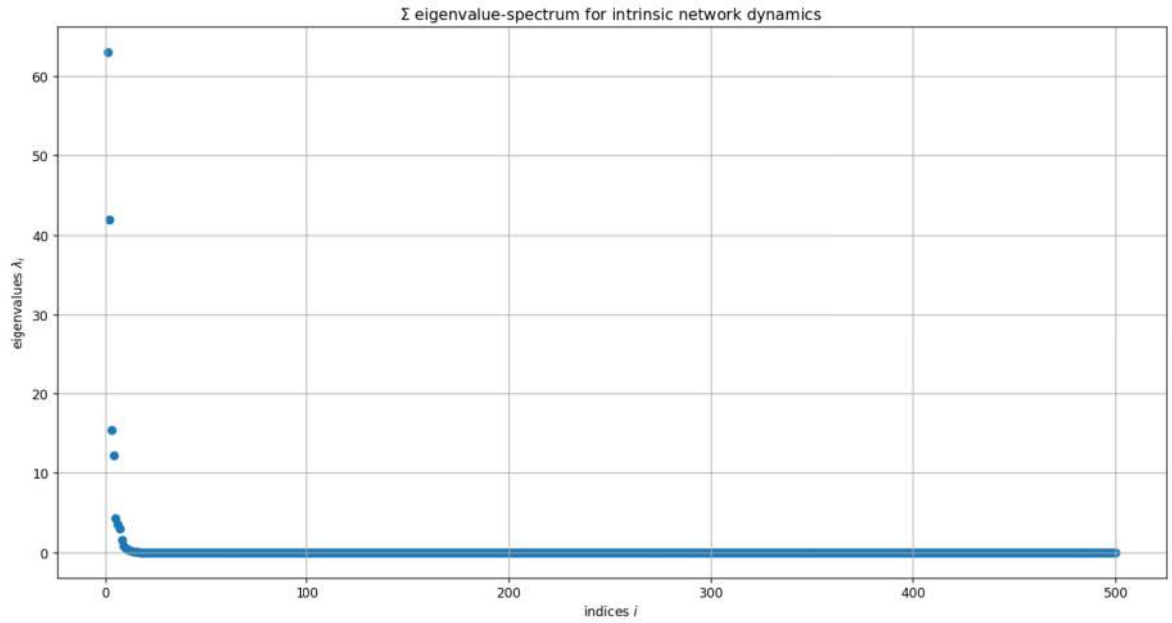
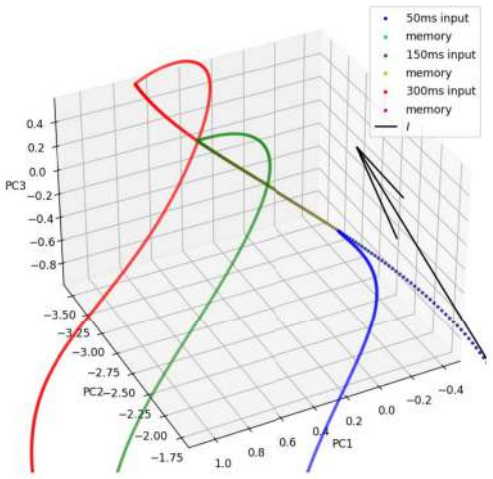
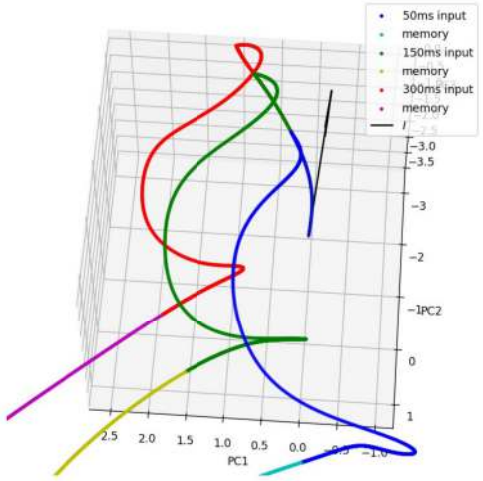


Figure 36: Eigenvalues λ_i of the covariance-matrix Σ from a PCA conducted for the RNN's intrinsic dynamics: The initial condition $(\mathbf{x_0}(t=0))_i = 0.1$ for $i \in [1, N]$ for the membrane-potentials is defined, from which the network's firing rates and output evolve according to equation 2 and 3 with $u(t) = 0$. PCA analysis is conducted in the time-frame $t \in [7000\text{ms}, 1200\text{ms}]$, where the initial condition has lost its direct effect on the network's dynamics. The eigenvalues λ_i of the resulting covariance-matrix Σ are illustrated.

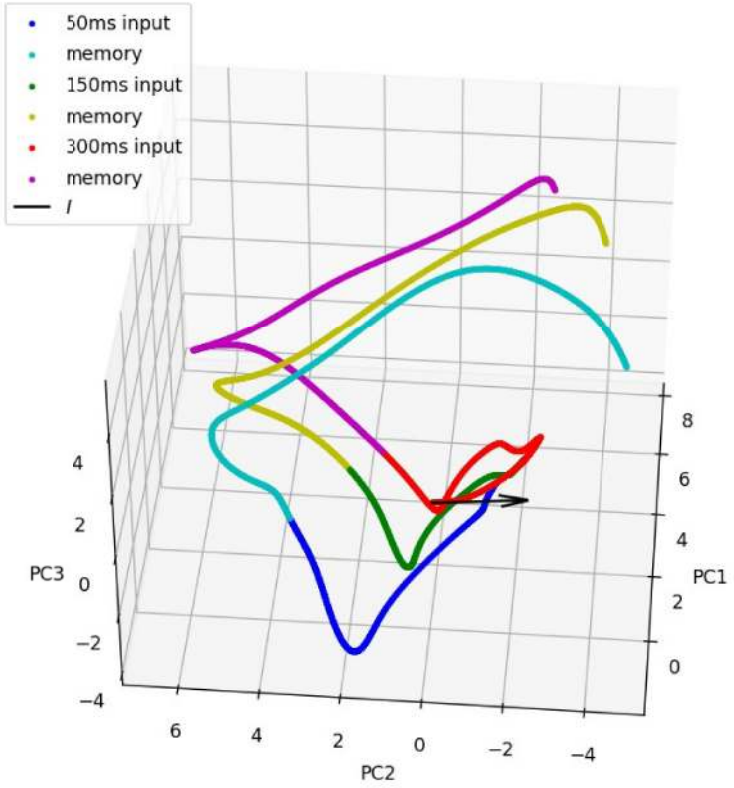
The projected dynamics are illustrated in figure 37. It depicts $\approx 79\%$ of the total $\mathbf{r}(t)$-variance (equation 45) and $\approx 7\%$ (equation 47) of the vector $\mathbf{I}$'s norm.



(a) Input-dominated phase



(b) Intrinsic phase



(c) Entire trajectory

Figure 37: Membrane trajectories $\mathbf{r}(t)$ projected onto the first three Principal Components. The axes are obtained using a PCA-analysis on the network's intrinsic dynamics: The initial condition $(\mathbf{x_0}(\mathbf{t}=\mathbf{0}))_i = 0.1$ for $i \in [1, N]$ for the membrane-potentials is defined, from which the network's firing rates and output evolve according to equation 2 and 3 with $u(t) = 0$. PCA analysis is conducted in the time-frame $t \in [7000\text{ms}, 1200\text{ms}]$, where the initial condition has lost its direct effect on the network's dynamics. The first three Principal Components are defined as the resulting covariance-matrix Σ's eigenvectors $\mathbf{v}_i$ with highest eigenvalues λ_i.

Three different trajectories $\mathbf{r}(t)$ for inputs of durations 50ms (blue and cyan), 150ms (green and olive) and 300ms (red and magenta) are projected onto these axes. $\approx$ 79% of the total $\mathbf{r}(t)$-variance (equation 45) and $\approx$ 7% (equation 47) of the vector $\mathbf{I}$'s norm are depicted. **(a)** Input-dominated phase. $\mathbf{r}(t)$-trajectories at times $t \in$ [3000ms, input-offset] where $u(t) \neq 0$. Blue trajectories denote input-durations of 50ms, green trajectories illustrate input-durations of 150ms and red trajectories depict input-durations of 300ms. Additionally, the projections of the input-weights onto PC-axes are illustrated using a vector-arrow. **(b)** Intrinsic phase. $\mathbf{r}(t)$-trajectories for times $t \in$ [3000ms $-$ 4500ms] before entering the plateau-phase. The same colour-codes denoting trajectories related to input-durations of 50ms, 150ms and 300ms are used. After the input $u(t)$ has been set to $u(t) = 0$, trajectories follow the dynamics defined by ${}^{dx_i}/_{dt} = -x_i + \sum_{j=1}^{N} G_{ij}\phi(x_j(t))$ and enter the intrinsic phase. MSE-Loss between output $z(t)$ and target is not evaluated at times $t \in$ [3000ms, 4500ms] for the gradients $\frac{\partial \varepsilon}{\partial \theta}$ for training. Again, the projections of the input-weights onto PC-axes are illustrated using a vector-arrow.**(c)** Entire trajectory. $\mathbf{r}(t)$-trajectories at times $t \in$ [3000ms, 7000ms] and input-durations 50ms (blue and cyan), 150ms (green and olive) and 300ms (red and magenta). Blue, green and red represent $\mathbf{r}(t)$ during the input-dominated and intrinsic phases. Cyan, magenta and olive are used once the curves enter the memory-period or plateau-phase for $t \in$ [4500ms, 7000ms], labelled by "memory". At this stage, MSE-Loss between output $z(t)$ and target is evaluated, and weight-gradients $\frac{\partial \varepsilon}{\partial \theta}$ are computed for training. The input-weights are denoted by an arrow.

5.5 Integration

We define "integration" as the working principle behind differencing the durations of inputs $u(t)$, and use figure 37 as a first indicator. According to the argument in expression 42, the trajectories $\mathbf{r}(t)$'s dynamics is overruled by the input $u(t)$ during the initial input-dominated phase in figure 37a. As a result, $\mathbf{r}(t)$-curves are tangential to the input vector $\mathbf{I}$ for times $t \in$ (input-onset, input-offest). The firing-rates' gradual deviation from this direction is due to the RNN's intrinsic dynamics, which is not fully suppressed according to the scaling arguments in section 3.5. Depending on the input $u(t)$'s duration, trajectories follow the common $\mathbf{I}$-dominated curve for different lengths of time. This turns our attention to figure 37b, which denotes the intrinsic phase. Once the input $u(t)$ is shut off, $\mathbf{r}(t)$ evolves according to the dynamical equation 2, and thus changes its direction to follow the path of its intrinsic dynamics. During this phase, defined for $t \in$ (input-offset, 4500ms), trajectories defined for different input-durations begin to separate.

These observations allow us to make educated guesses on the RNN's integrating-mechanism. We hypothesize that the underlying strategy to process the input's duration is as follows:

1. Since $u(t)$ is turned off at different times, the intrinsic dynamics are initiated at differing points in PC-space. This leads to a separation of trajectories, which is related to these varying input-durations. Since the governing dynamical equation 2 is the same for every curve $\mathbf{r}(t)$, we hypothesize that they are parallel.

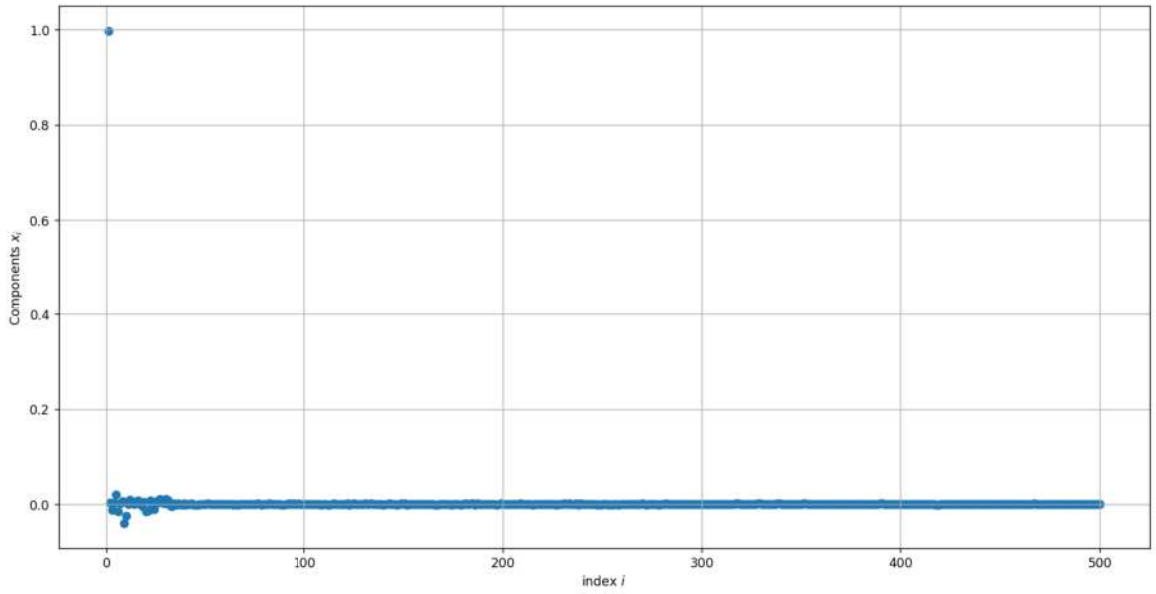
2. The distance between the individual $\mathbf{r}(t)$-curves encodes the different input-durations.

3. This separation of trajectories is an intrinsic property of the RNN's dynamics and is independent of the specifics of $\mathbf{I}$'s direction.

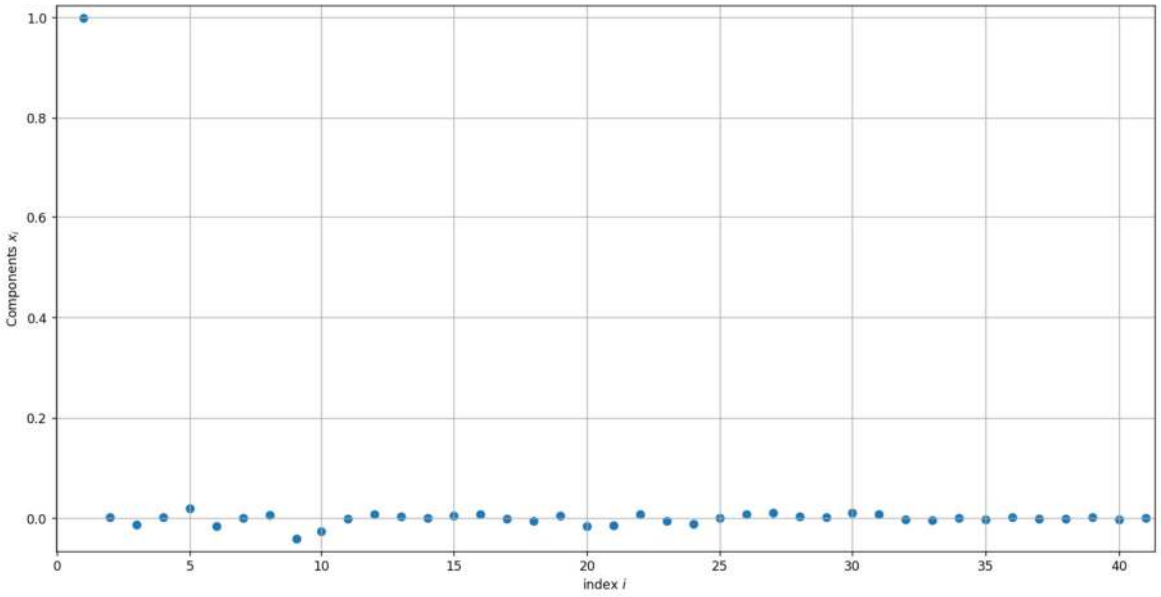
We summarise these points by hypothesizing that the RNN's integrating behaviour translates into measuring the distance between the different parallel trajectories $\mathbf{r}(t)$. We solidify this in the following sections.

5.5.1 Parallelism of trajectories

This chapter attempts at quantifying that firing trajectories are parallel (point 1 of section 5.5) to each other, and thus share common axes of highest variance. This property is crucial for letting us use the term of "distance" between trajectories of different input-durations $u(t)$ in point 2 of section 5.5. To find the denoted common axes, we conduct a trial-concatenated PCA as described in section 3.6 by using the combined firing-rate trajectories for input-durations 50ms, 150ms and 300ms. The resulting PC-axes form an orthogonal basis $\{\mathbf{a}_i\}$, where the first basis-vectors for $i = 1$, $i = 2$, $i = 3$, ... denote the directions of highest $\mathbf{r}(t)$ variance across all trials. If trajectories for different trials were to feature highest variance in the same directions, then projecting a vector $\mathbf{b}_1$ obtained by conducting a PCA for a single trial onto the basis $\{\mathbf{a}_i\}$ as $\mathbf{b}_1 = \sum_{i=1}^{N} x_i \mathbf{a}_i$ should yield $x_1 \approx 1$, while all other components should be $x_{i\neq 1} \approx 0$. If this is the case, then trajectories can be considered to be parallel. Figure 38 qualitatively demonstrates this procedure's result for a single RNN-prototype. We conclude that indeed the projection of $\mathbf{b}_1$ onto $\mathbf{a}_1$ is $x_1 \approx 1$, while the remaining components are ≈ 0. We repeat this procedure for 20 different RNNs in figure 39, which leads us to the same conclusion: The median of the resulting components x_1 is significantly higher than the remaining medians $x_{i\neq 1}$.



(a) Components x_i of $\mathbf{b}_1$ on trial-concatenated basis $\{\mathbf{a}_i\}$ for $i = 1, ..., N$ (trained $\mathbf{I}$)



(b) Components x_i of $\mathbf{b}_1$ on trial-concatenated basis $\{\mathbf{a}_i\}$ for $i = 1, ..., 40$ (trained $\mathbf{I}$)

Figure 38: Visualising the components x_i of the first Principal Component $\mathbf{b}_1$ of a single $\mathbf{r}(t)$-trajectory projected onto the Principal Components of a trial-concatenated PCA $\{\mathbf{a}_i\}$. **(a)** Conducting a trial-concatenated PCA-analysis (equation 28) for the trajectories $\mathbf{r}(t)$ obtained with input signals of 50ms, 150ms and 300ms over the time-frame of $t \in (0\text{ms}, 3 \cdot N_t)$. The resulting eigenvector-spectrum of Σ $\{\mathbf{a}_i\}$ is used as a basis. A second PCA (equation 26) for trajectories resulting from an input-signal of 200ms over the time-frame $t \in (0\text{ms}, N_t)$ is conducted. The resulting first PC is expressed in the basis $\{\mathbf{a}_i\}$ according to $\mathbf{b}_1 = \sum_{i=1}^{N} x_i \mathbf{a}_i$. The figure shows the components x_i for each index i. The analysis is done using an RNN with trained input-weights. **(b)** Zooming into figure (a) and visualising the components $i = 1, ..., 40$ of x_i.

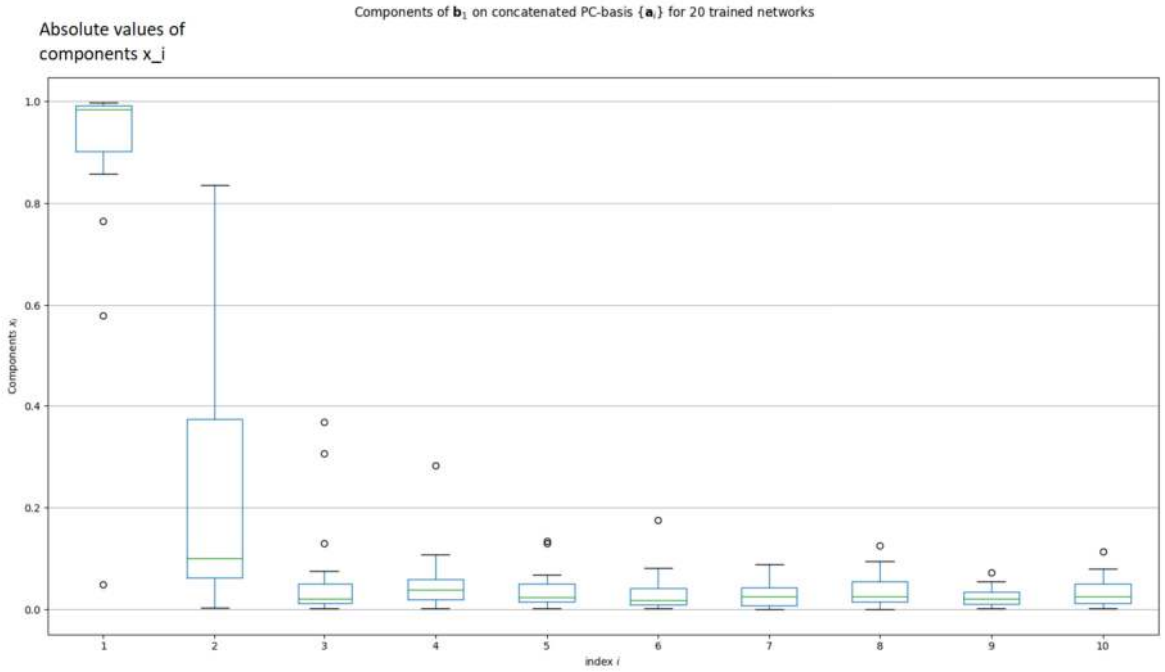


Figure 39: Visualising the absolute values of components x_i of the first Principal Component $\mathbf{b}_1$ of a single $\mathbf{r}(t)$-trajectory projected onto the Principal Components of a trial-concatenated PCA $\{\mathbf{a}_i\}$ for $i = 1, ..., 10$. The procedure explained in figure 38 is repeated for 20 different trained models and results are illustrated using a boxplot. Median values are marked by green lines, upper and lower quartiles are denoted by blue lines and whiskers indicate $1.5 \cdot$ interquartile range (distance between the upper and lower quartiles). Outliers are marked as dots.

Finally, we make an interesting observation in figures 40 and 41, which will introduce us to the next chapter. We have investigated the parallelism of the dynamics $\mathbf{r}(t)$ from trained vectors $\mathbf{I}$ by pointing out, that x_1-components are distinctly higher than the remaining $x_{i\neq 1}$-values. However, we make the same observation for the $\mathbf{r}(t)$-trajectories of random vectors $\mathbf{I}$ (distribution 23). Figure 40 demonstrates this for a single network. Analysing this feature for 20 different prototypes and depicting the resulting medians in figure 41 yields the same pattern. We conclude that trajectories $\mathbf{r}(t)$ of different-input durations are parallel for any random vector $\mathbf{I}$. We will delve deeper into this argument in the following chapter.

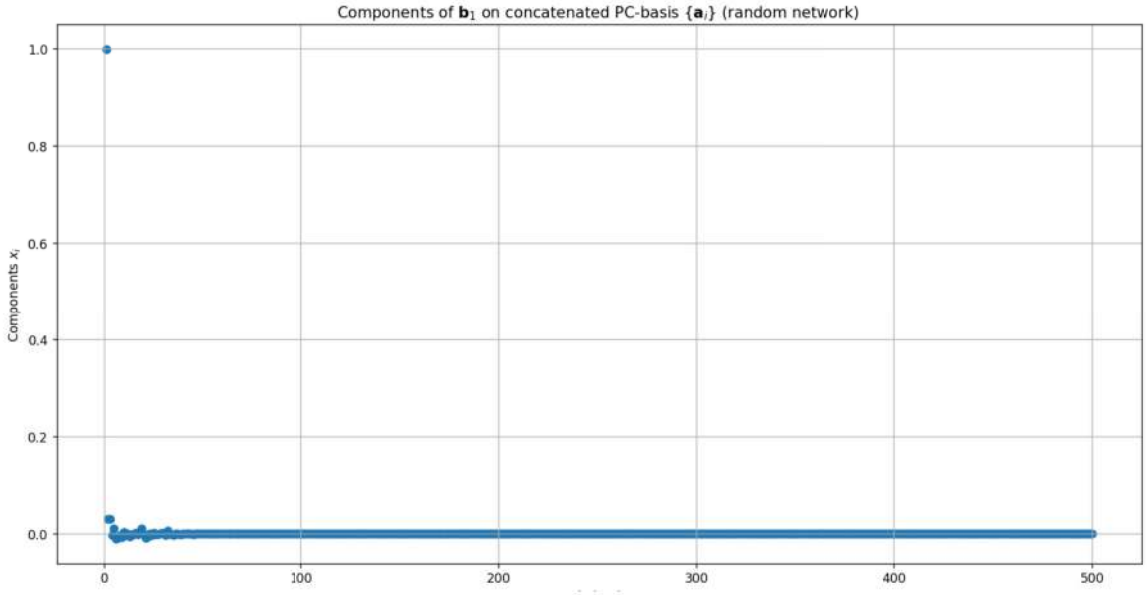


Figure 40: Visualising the components x_i of the first Principal Component $\mathbf{b}_1$ of a single $\mathbf{r}(t)$-trajectory projected onto the Principal Components of a trial-concatenated PCA $\{\mathbf{a}_i\}$. The procedure of figure 38 is applied to a network with random input-weights $\mathbf{I}$

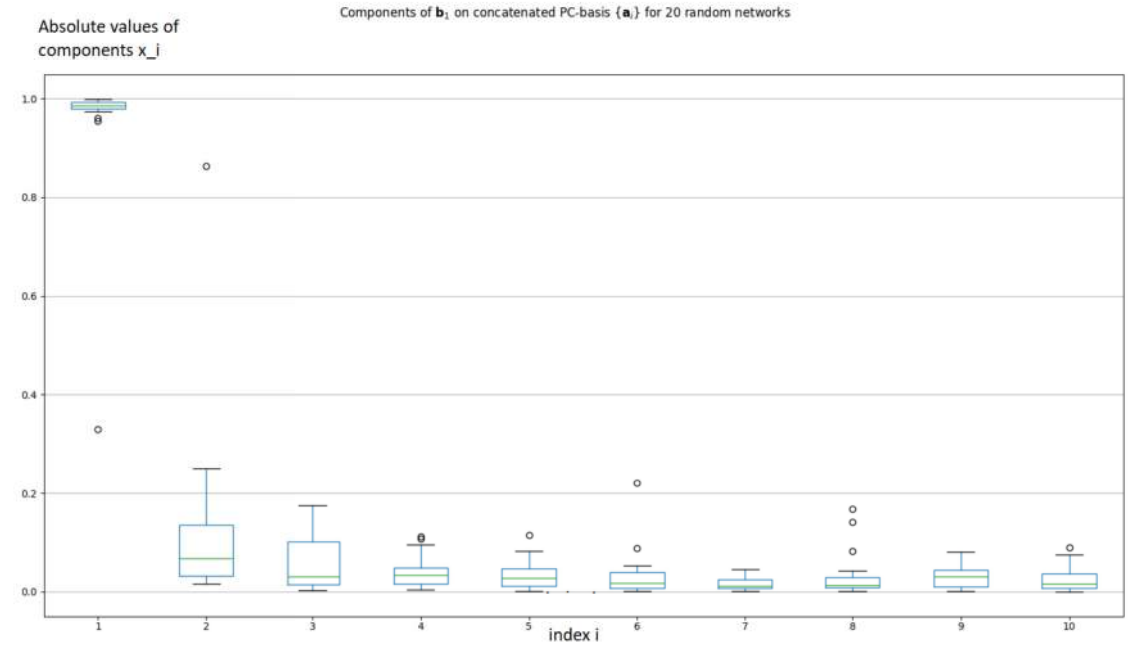


Figure 41: Visualising the absolute values of components x_i of the first Principal Component $\mathbf{b}_1$ of a single $\mathbf{r}(t)$-trajectory projected onto the Principal Components of a trial-concatenated PCA $\{\mathbf{a}_i\}$. The procedure of figure 38 is applied to a network with random input-weights $\mathbf{I}$, and is repeated for 20 different models. Results are illustrated using a boxplot for $i = 1, ..., 10$. Median values are marked by green lines, upper and lower quartiles are denoted by blue lines and whiskers denote $1.5 \cdot$ Interquartile range (distance between the upper and lower quartiles).

5.5.2 Space-unspecificity of parallel trajectory-separation

In this section, we elaborate on the point 3 made in section 5.5. It states that any vector $\mathbf{I}$ is suitable for separating trajectories from each other in a parallel fashion. It is not a phenomenon that only occurs in specific spaces.

We have already alluded to this feature in section 5.5.1 by projecting the first PC $\mathbf{b}_1$ of a single-trial-PCA to its trial-concatenated PC-basis $\{\mathbf{a}_i\}$. We observed that even for random input-vectors $\mathbf{I}$, $\mathbf{b}_1$ and $\mathbf{a}_1$ were highly correlated.

We proceed by asking ourselves if the training-process of $\mathbf{I}$ casts the $\mathbf{r}(t)$-curves to a specific subspace which would be particularly advantageous for solving the integration-task. Answering this question by "yes" would imply the following:

- If the RNN relies on utilising the distance between parallel trajectories $\mathbf{r}(t)$ to process duration-length of $u(t)$, then there exist specific subspaces where the denoted trajectory-separation is promoted. This would thus falsify point 3 of section 5.5, which states that trajectory-separation is part of the RNN's intrinsic dynamics and independent of the $\mathbf{I}$-vector's direction.
- If the RNN does not rely on utilising the distance between parallel trajectories to solve the integration-task, then casting the $\mathbf{r}(t)$-curves to a specific subspace would hint at another strategy, and thus falsify points 2 and 3 of section 5.5.

On the other hand, answering the question by "no", would lead us to the following conclusions:

- It would support the idea that trajectories $\mathbf{r}(t)$ are parallel in any subspace, regardless of the direction the vector $\mathbf{I}$ casts these curves into. As a result, $\mathbf{r}(t)$-parallelity is an intrinsic feature of the RNN's dynamics, as stated in point 3 of section 5.5.
- Furthermore, if parallel curve-separation is a natural property of the RNN's dynamics, then utilising it to measure and process input-durations would be a straight-forward strategy to solve our task. This would thus make point 2 of section 5.5 particularly plausible.

Based on the arguments below, we hypothesize that the training process of $\mathbf{I}$ does not direct the $\mathbf{r}(t)$-curves into a particular subspace.

A first and straight-forward clue reverts to figure 26a of section 5.2.1, which illustrates the Pearson-correlation between the trained input-weights $\mathbf{I}$ of 20 different trained models. They range from -0.2 to $+0.2$, which denotes, that not a single common $\mathbf{I}$-direction is found across samples. We thus conclude that there are at least more than 20 subspaces promoting parallel trajectory-separation, and more tentatively infer that $\mathbf{r}(t)$ split up independently of the space they are traced in.

We are making a second case by analysing the dynamics of the $\mathbf{r}(t)$-curves. If their separation relied on a specific subspace, then we would assume that $\mathbf{r}(t)$-trajectories for different RNNs unfold in common directions, and thus feature common axes of highest variance. Conducting a trial-concatenated PCA for the collection of all $\mathbf{r}(t)$-curves therefore should not reflect a significant dimensionality increase. We explore this reasoning in figure 42. It illustrates the participation ratios D of increasing numbers of concatenated trajectories $\mathbf{r}(t)$ for different trained models. The results are compared with the participation ratios of concatenated curves $\mathbf{r}(t)$ for random weights $\mathbf{I}$. However, figure 42 does not reveal any distinct difference between both curves. We conclude that the $\mathbf{r}(t)$-trajectories of 20 different trained RNNs do not unfold in common directions. On a side note, the concatenation of trained $\mathbf{r}(t)$-trajectories consistently showcase higher participation-rations D than their trained counterparts. We will refer to this finding in the following section.

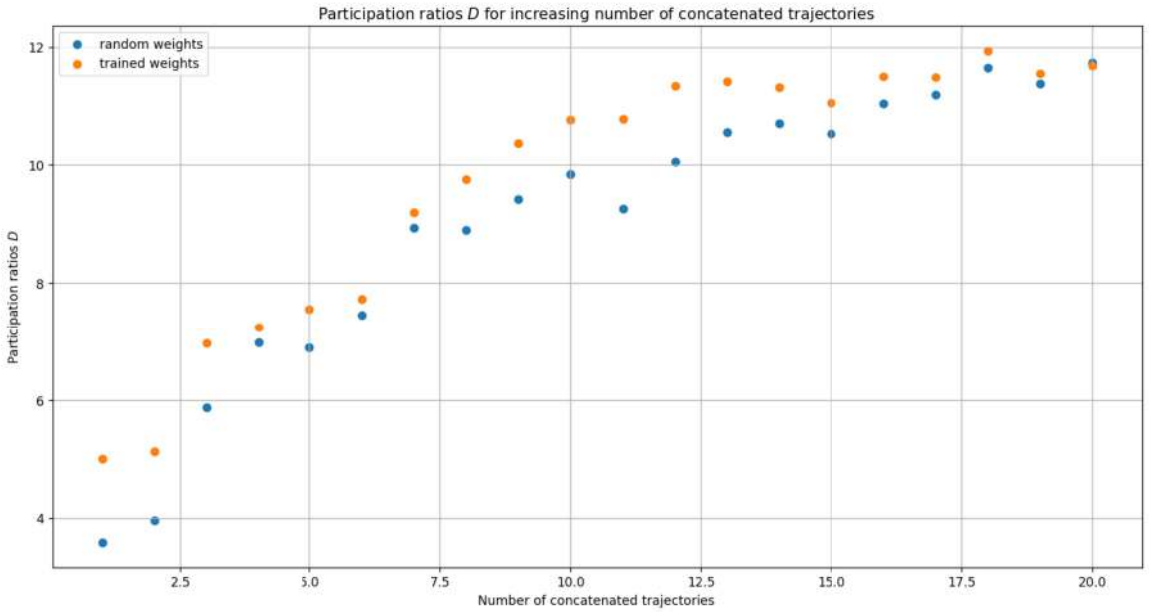


Figure 42: Participation-ratios D for increasing numbers of trajectories $\mathbf{r}(t)$. The orange curve makes use of a total of 20 different trained RNNs. The x-axis denotes how many trajectories $\mathbf{r}(t)$ from different models are concatenated to a resulting curve $\mathbf{r}(t)$ for $t \in [0, x \cdot N_t]$. It is used to conduct a trial-concatenated PCA on the entire time-frame. The resulting spectrum's dimensionality is evaluated by calculating the participation-ration D according to equation 30. The blue curve makes use of 20 random models with input-weights drawn from distribution 23 and follows the same procedure.

We conclude this chapter by reiterating our findings and their significance. We have found that trajectories $\mathbf{r}(t)$ separate in a parallel fashion when RNNs are fed input-signals $u(t)$ of different durations. Furthermore, this phenomenon is an intrinsic property of the RNN's dynamics, and does not depend on the direction of the input-vector $\mathbf{I}$. As a result, leveraging the distance between the resulting $\mathbf{r}(t)$-curves, which directly correlates with the $u(t)$-durations, is a conceivable strategy to solve our integration-task.

5.5.3 Interpreting input-durations as the distance between parallel trajectories

We proceed by investigating if the RNN utilises the separation between different $\mathbf{r}(t)$-trajectories to process varying input-durations. We are thus aiming at justifying point 2 of section 5.5.

This idea first arose while training networks with fixed and random vectors $\mathbf{I}$, such that optimisation was only implemented for the output-weights $\mathbf{W}$. From a qualitative

perspective, it resulted in networks successfully solving the integration- and memory task as illustrated in figure 28. Furthermore, these models yielded a comparable performance to fully trained networks and smaller MSE-losses compared to their fixed **W** and trained **I** counterparts (figure 32). An interesting parallel can be drawn between the network's ability to solve the task without training the weights **I** on one hand, while on the other hand, neither the separation of trajectories into parallel counterparts depends on the vector **I**. This could be an indication that both processes are related.

We dig deeper into this matter by turning our attention towards the relationship between input- and output-weights. Figures 43 and 44 showcase that they are highly correlated, such that training seems to align the vectors.

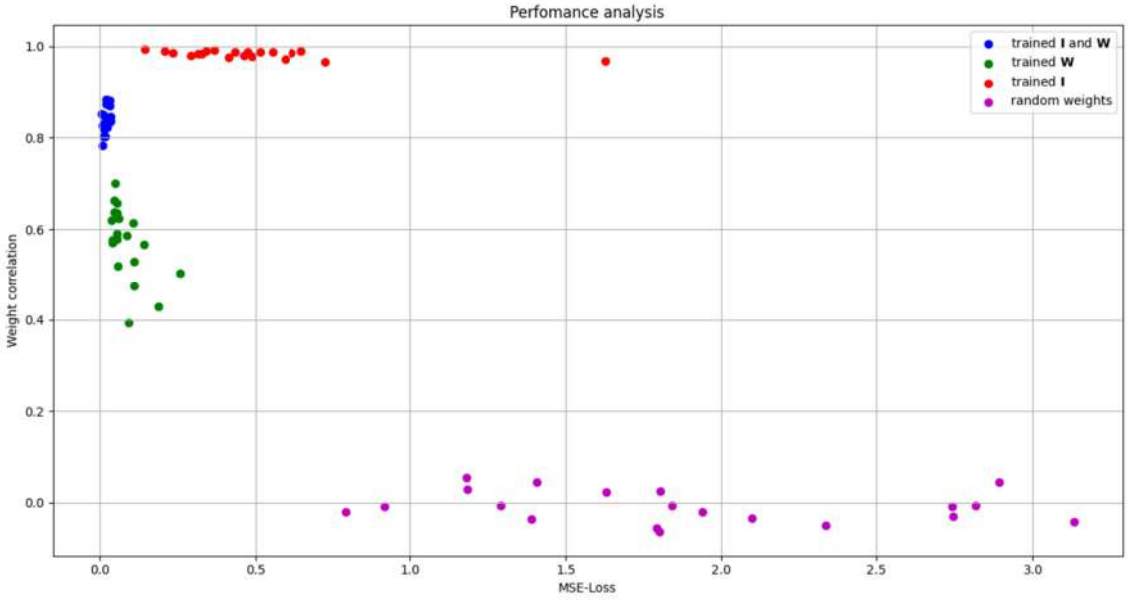


Figure 43: Comparing MSE-Loss with weight correlations $\frac{\mathbf{WI}}{|\mathbf{WI}|}$ for differently trained models. The label "trained **I** and **W**" refers to RNNs with trained input- and output-weights. Models with fixed, random **I**-weights and trained **W**-weights drawn from distribution 23 are denoted as "trained **W**". On the other hand, "trained **I**" refers to RNNs with fixed, random **W**'s drawn form distribution 22. Finally, networks with random input- and output-weights drawn from distributions 23 and 22 respectively are labelled as "random" . MSE-Loss is computed in comparison to the target-function defined in equation 31 for three input-signals of durations 50ms, 150ms and 300ms. The final value is is obtained by taking the mean over the three results.

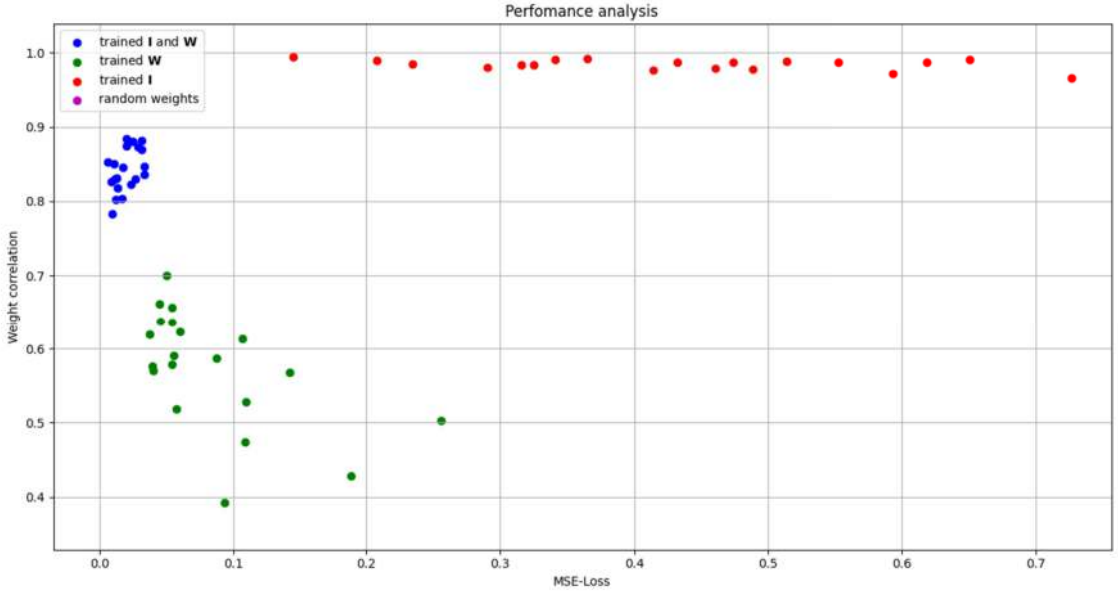


Figure 44: Zooming into figure 43

We make sense of this by highlighting the role of $\mathbf{W}$, which revolves around projecting $\mathbf{r}(t)$-dynamics onto a single scalar output $z(t) = \mathbf{W}^T\mathbf{r}(t)$. Furthermore, we have discussed in section 5.4 how the vector $\mathbf{I}$ directs firing rates into a specific direction during the input-dominated phase, and how different input-durations make $\mathbf{r}(t)$-curves deviate from this common trajectory at different points in time. We have additionally established in section 5.5.1, that this leads to parallel trajectories, with spacings correlating with the $u(t)$-durations. As a result, this renders an axis $\mathbf{D}$ (figure 45) denoting the separation between the different $\mathbf{r}(t)$-trajectories, which is parallel and correlated to $\mathbf{I}$. This holds true up to the point where the intrinsic dynamics of the RNN still result in a slight deviation from the direction of $\mathbf{I}$. To close the circle and return to the role of $\mathbf{W}$, reading out the distance between the distinct trajectories requires the vector to be aligned with the axis of separation $\mathbf{D}$. In other words, leveraging the $\mathbf{r}(t)$-spacings translates into correlating $\mathbf{W}$ and $\mathbf{I}$, which is depicted in figure 44.

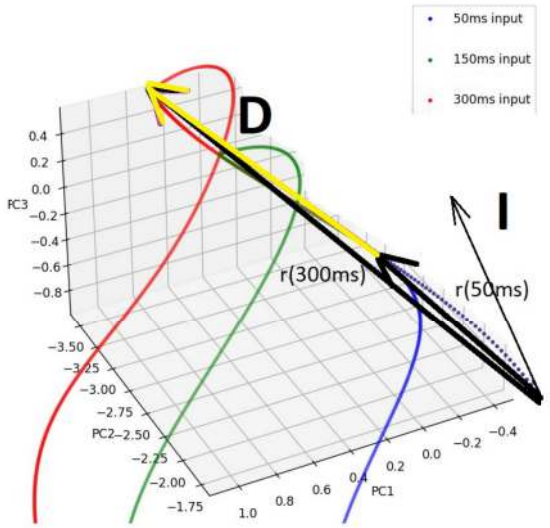


Figure 45: Depicting the separation-axis $\mathbf{D}$. Trajectories $\mathbf{r}(t)$ for three different input-durations of 50ms (blue), 150ms (green), 300ms (red) along the input-vector $\mathbf{I}$ are demonstrated. The vector, which denotes the direction separating the different curves is the separation-axis $\mathbf{D}$ marked in yellow. It is defined as $\mathbf{D} = \mathbf{r}(300\text{ms}) - \mathbf{r}(50\text{ms})$, where we use the red curve $\mathbf{r}(t)$ resulting from a 300ms input. The quantity $\mathbf{r}(300\text{ms})$ is the denoted $\mathbf{r}$-vector 300ms after input-onset and $\mathbf{r}(50\text{ms})$ the $\mathbf{r}$-vector 500ms after input-onset.

We test this hypothesis by approximating the denoted axis of separation, which we use as an output-vector $\mathbf{W}$. Figure 46 depicts the resulting output $z(t) = \mathbf{W}^T\mathbf{r}(t)$ with the redefined $\mathbf{W}$ and random input-weights $\mathbf{I}$. The three different examples qualitatively demonstrate that setting $\mathbf{W}$ along the axis separating $\mathbf{r}(t)$-curves for different input-durations is a conceivable strategy for reading out their distance in space. Figure 47 depicts the output of an RNN with random input- and output-weights for comparison.

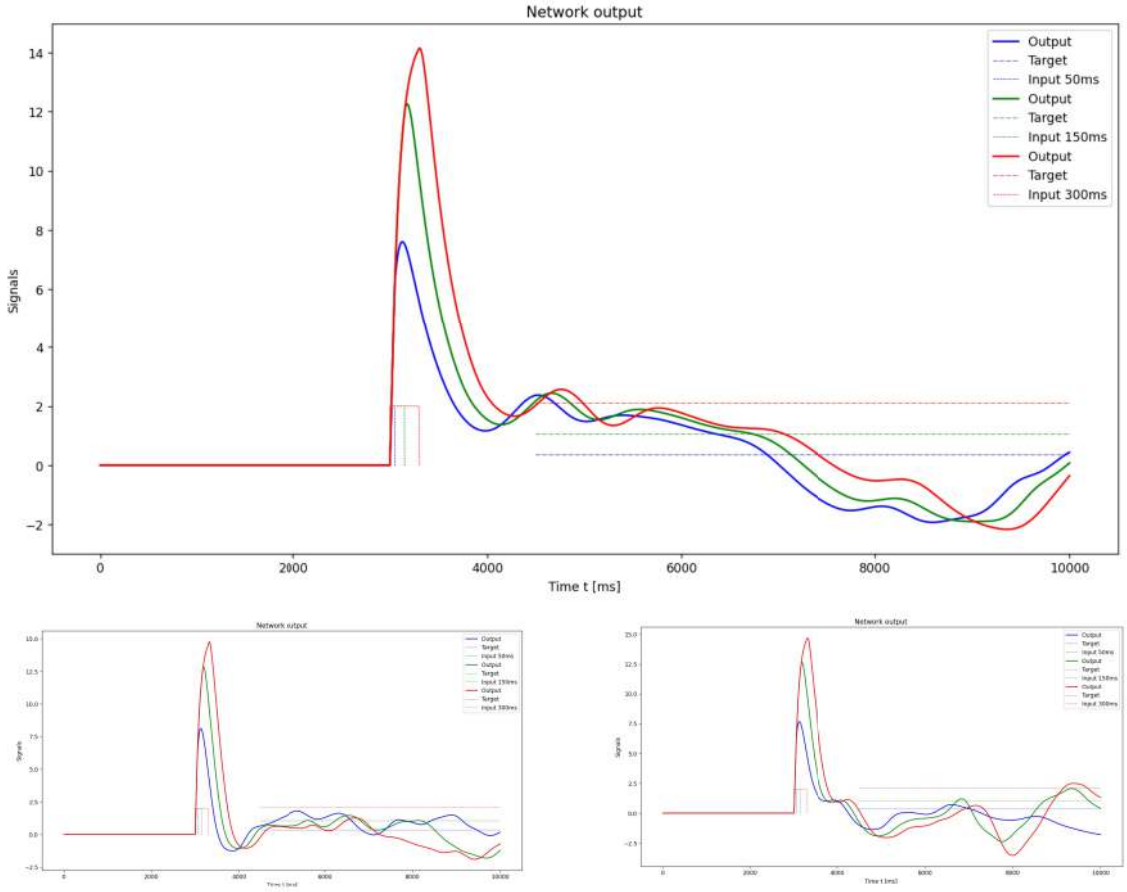


Figure 46: Three examples of RNN-outputs $z(t)$ with random input-weights $\mathbf{I}$ (distribution 23) and defined output-weights $\mathbf{W}$ along the axis of separation $\mathbf{D}$. $\mathbf{D}$ is obtained as follows: The vector $\mathbf{r}(t)$ is the resulting firing-rate trajectories from using an input-signal of 300ms duration. As illustrated by figure 45, the axis of separation can be approximated by $\mathbf{D} = \mathbf{r}(300\text{ms}) - \mathbf{r}(50\text{ms})$, where $\mathbf{r}(300\text{ms})$ is the $\mathbf{r}$-vector 300ms after input-onset and $\mathbf{r}(50\text{ms})$ the $\mathbf{r}$-vector 50ms after input-onset. The resulting vector is scaled to a $\text{norm}(\mathbf{D}) = 1$ and used to define the output-weights $\mathbf{W} = \mathbf{D}$. Each plot depicts the resulting outputs (thick lines) for three input-durations of 50ms (blue), 150ms (green), 300ms (red), while their respective input- and target-functions are equally colour-coded using dashed lines.

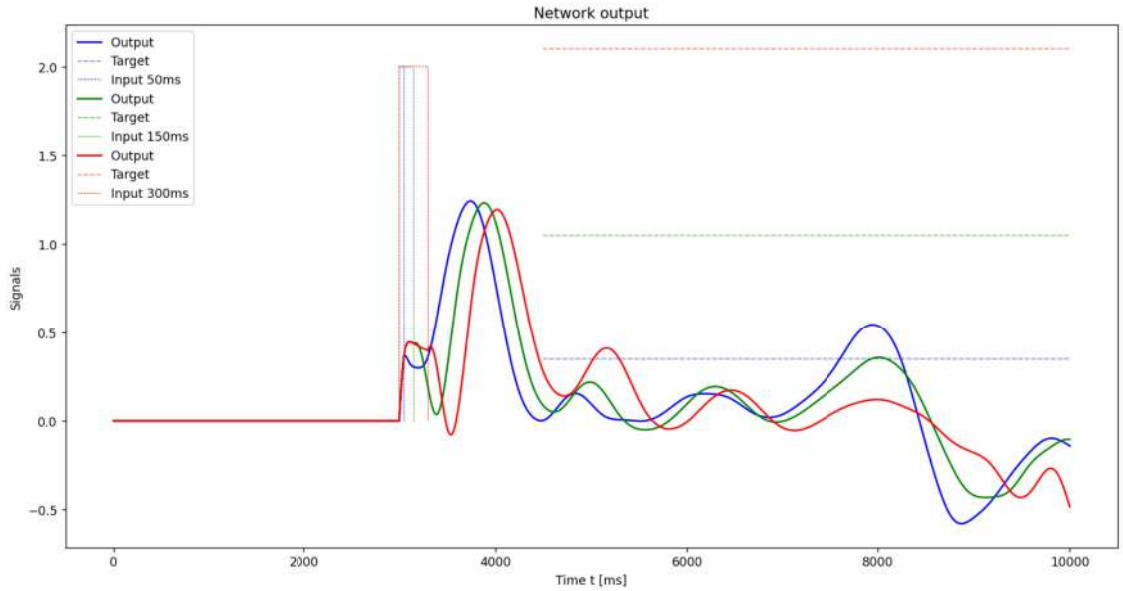


Figure 47: Three network outputs $z(t)$ (thick lines) for random input-and output-weights from the respective distributions 23 and 22. The colours denote input-durations of 50ms (blue), 150ms (green), 300ms (red), while their respective input- and target-functions are equally colour-coded using dashed lines.

To sum up, this chapter describes how we built the intuition that the RNN's strategy to distinguish $\mathbf{r}(t)$-curves for different input-durations relies on measuring their respective distance in space. We found a high correlation between input- and output-weights of trained networks, and interpreted this geometrically. Finally, we tested this interpretation by setting output-weights $\mathbf{W}$ along the separation-axis $\mathbf{D}$, which reasonably well produced outputs differencing varying input-durations. We will elaborate in the context of the following chapter 5.6 on an additional observation, which will further explain the role of $\mathbf{D}$

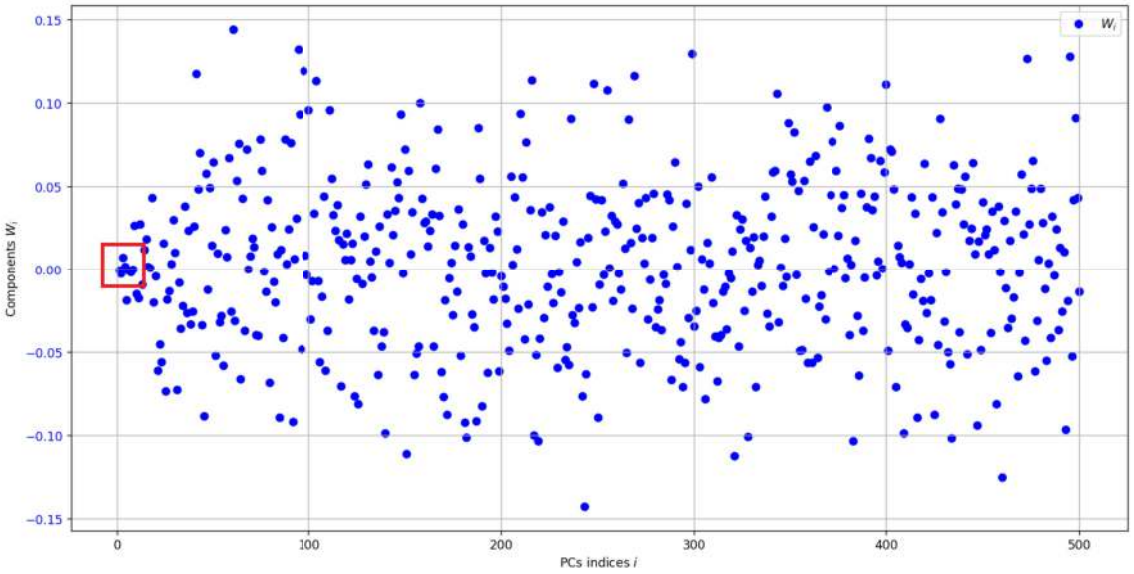
Finally, let's return to the introducing line of chapter 5.5: "We define "integration" as the working principle behind differencing the durations of inputs $u(t)$." We propose that the denoted strategy relies on reflecting the distance between $\mathbf{r}(t)$-curves in space, which correlates with the durations of inputs $u(t)$.

5.6 Working-memory

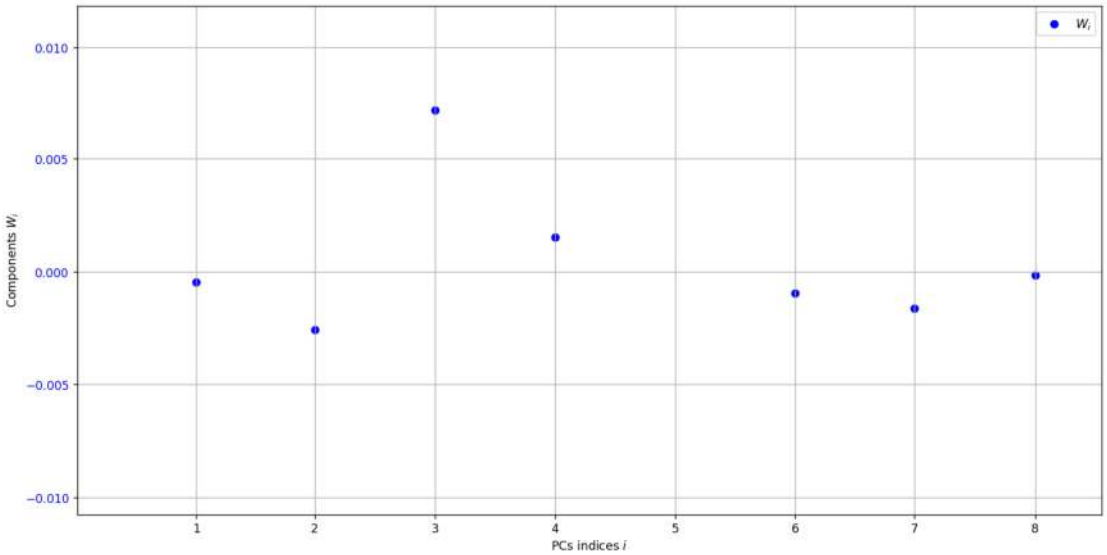
It is one thing to distinguish inputs from each other, but how useful is this if we cannot keep it in mind? In this chapter, we proceed by investigating the mechanism underlying the RNN's working-memory. More specifically, the term "working memory" refers to the network's plateau-like output, which distinguishes inputs by its according height,

and is maintained in time (and mind).

Our reasoning is as follows: Our RNNs are expected to represent input-durations by generating parallel outputs with spacings correlating with the respective input-times. We have attempted at reproducing this feature in figure 46. However, in addition to the $\mathbf{r}(t)$-spacings, the output-vector $\mathbf{W}$ seems to read out other fluctuations, which yields unstable graphs. This could be avoided, by setting the vector $\mathbf{W}$ orthogonally to the axes of highest $\mathbf{r}(t)$-variations. We investigate if this assumption is applied by the trained RNN, by conducting a PCA over the plateau-phase for $t \in (4500\text{ms}, 7000\text{ms})$. The resulting eigenvectors $\{\mathbf{v}_i\}$ of Σ yield the axes of highest $\mathbf{r}(t)$-variance. As a result, setting the vector $\mathbf{W} = \sum_{i=1}^{N} W_i \mathbf{v}_i$ orthogonally to the denoted directions would translate into finetuning the components W_i of vectors $\{\mathbf{v}_i\}$ with high eigenvalues λ_i to ≈ 0. We thus examine if this strategy is applied by the trained RNN and project a trained vector $\mathbf{W}$ onto the eigenbasis $\{\mathbf{v}_i\}$ of the plateau-period. The resulting components are illustrated in figure 48. Indeed, the W_i-results for low i's (indices of λ_i) are noticeably smaller that the remaining components. The procedure is repeated for a set of 20 trained models in figure 49 which reflects the same feature. On the other hand, it is not made out in figure 50 which makes use of a random $\mathbf{W}$ for comparison.

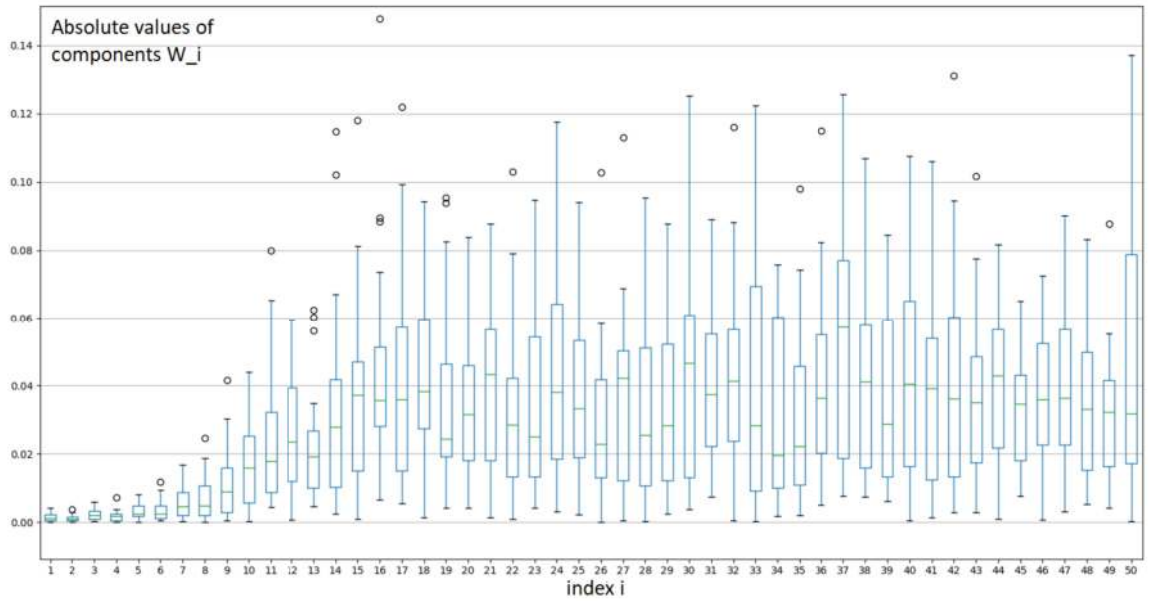


(a) Trained W_i-components for $i = 1, ..., N$

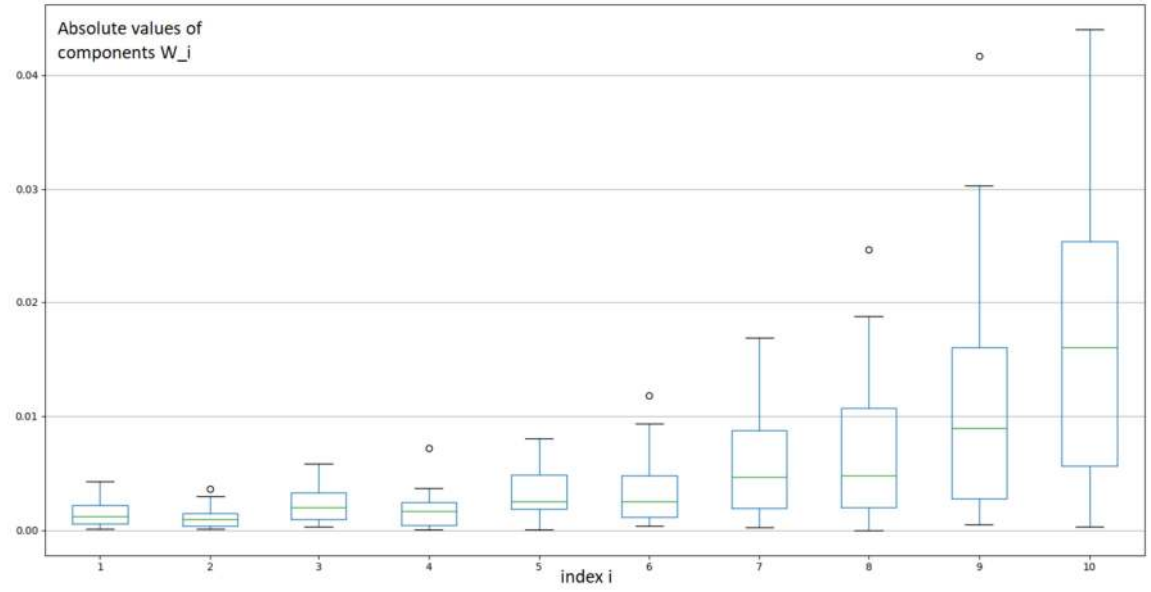


(b) Trained W_i-components for $i = 1, ..., 8$

Figure 48: **(a)** Absolute values of the components W_i for output-weights $\mathbf{W} = \sum_{i=1}^{N} W_i \mathbf{v}_i$ projected onto the PC's $\{\mathbf{v}_i\}$ of the plateau-period. A PCA-analysis is conducted for $\mathbf{r}(t)$-trajectories resulting from a 150ms input-duration. Equations 25 and 26 are applied to $\mathbf{r}(t)$ for $t \in [4500\text{ms}, 7000\text{ms}]$. The Principal Components $\{\mathbf{v}_i\}$ are the eigenvectors of the covariance-matrix Σ. Trained weights $\mathbf{W}$ are projected onto the PCs as $W_i = \mathbf{W} \cdot \mathbf{v}_i$ The x-axis denotes the according indices i. These are ordered by magnitude of λ_i, such that λ_0 denotes the highest eigenvalue and λ_N the lowest. A red square demonstrates particularly low W_i-components related to high eigenvalues. **(b)** Zooming into the red square of figure (a). The particularly low W_i-components related to high eigenvalues λ_i for $i = 1, .., 8$ are displayed.

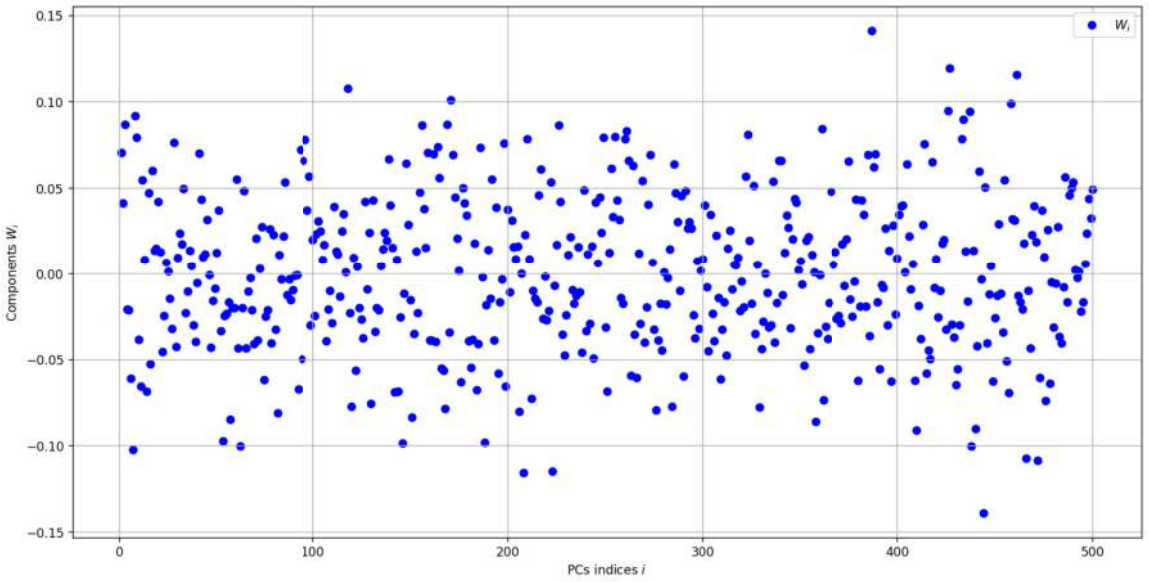


(a) Trained W_i-components of 20 different models for $i = 1, ..., 50$

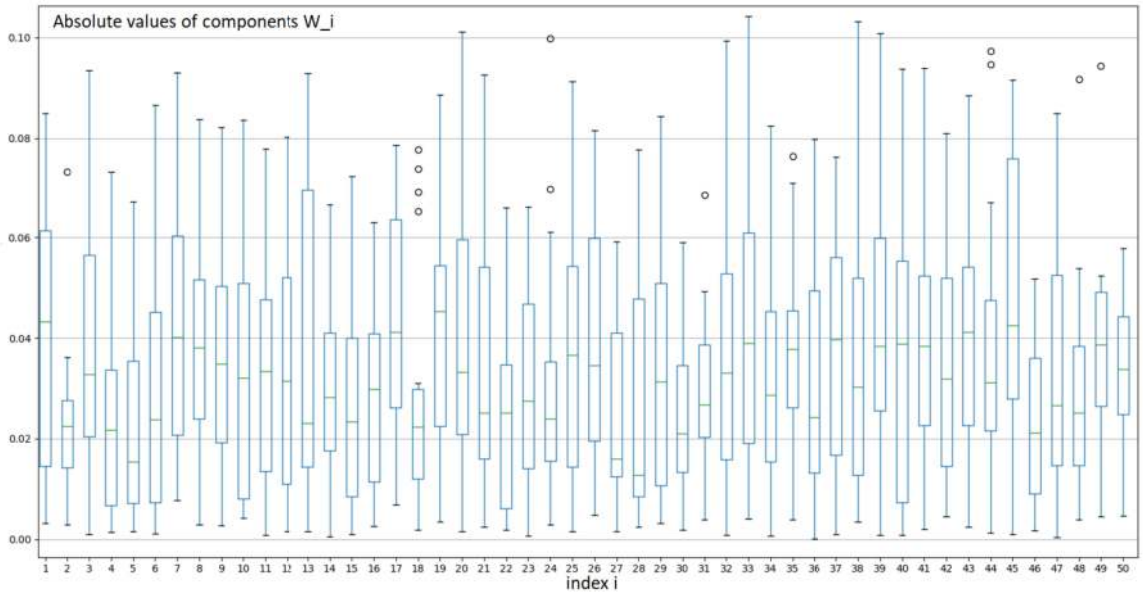


(b) Trained W_i-components of 20 different models for $i = 1, ..., 10$

Figure 49: **(a)** Repeating the process described for figure 48 for 20 different trained networks. Resulting absolute values of W_i components for indices i on the x-axis are illustrated in a boxplot. Median values are marked by green lines, upper and lower quartiles are denoted by blue lines and whiskers denote the $1.5 \cdot$ Interquartile range (distance between the upper and lower quartiles). **(b)** Zooming into the first 10 components i of the boxplot in figure (a).



(a) Random W_i-components for $i = 1, ..., N$

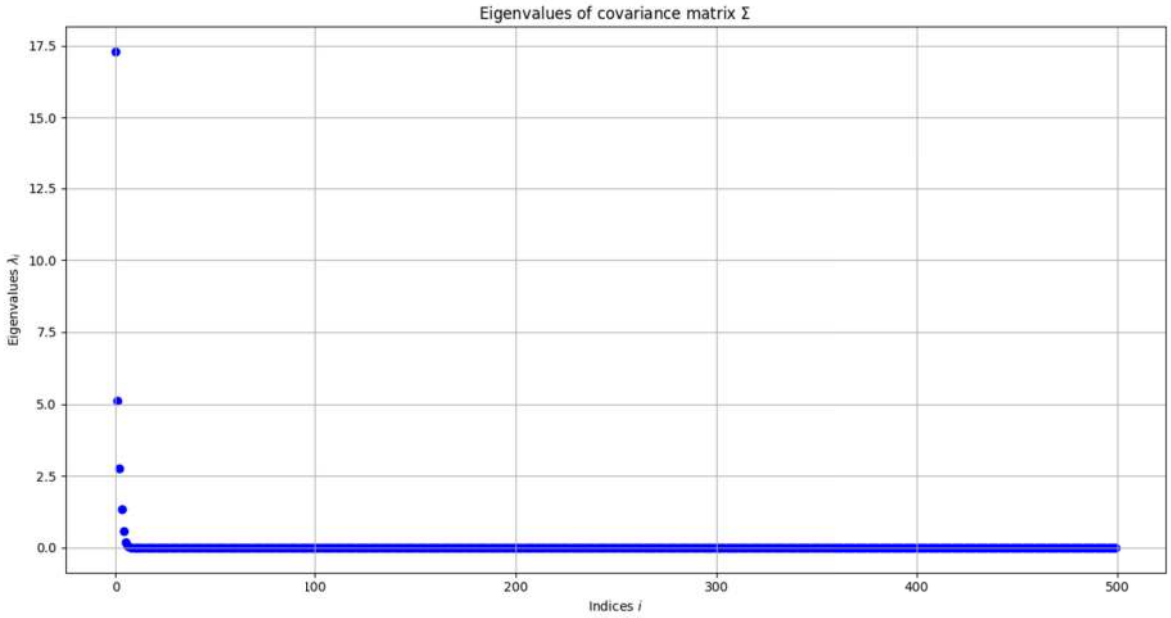


(b) Random W_i-components of 20 different models for $i = 1, ..., N$

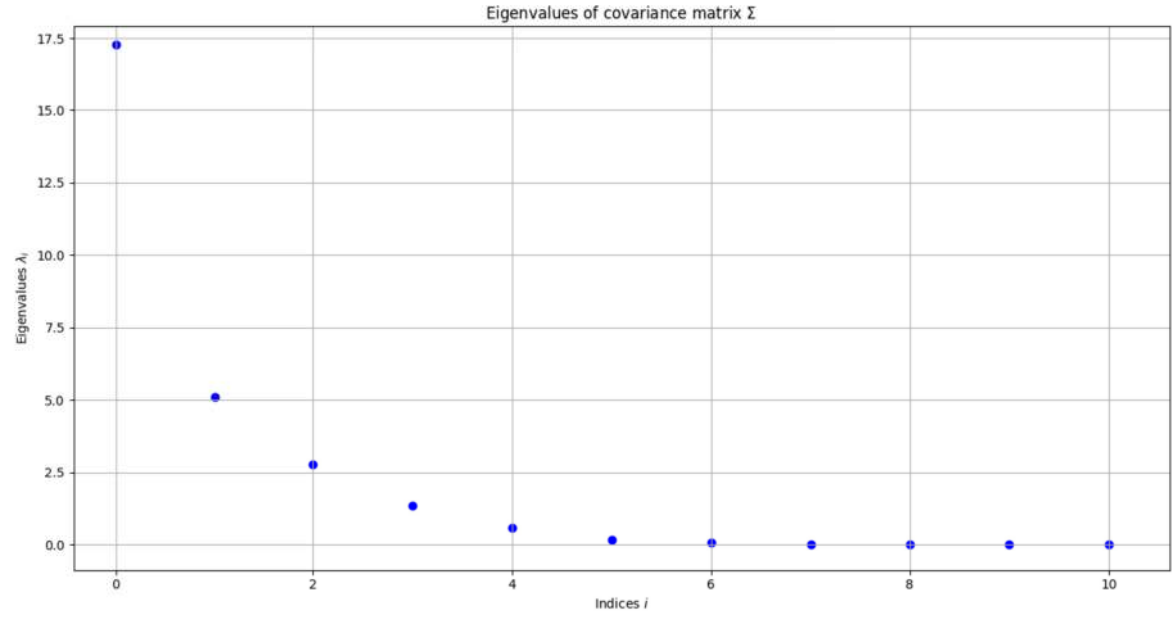
Figure 50: **(a)** Repeating the procedure outlined in figure 48 for a random vector $\mathbf{W}$. The absolute values of W_i components related to high eigenvalues λ_i for $i = 1, .., 8$ do not reveal distinctly small values. **(b)** Repeating the process described for figure 49 for 20 random vectors $\mathbf{W}$ drawn from distribution 22.

We dig deeper into this idea by manually defining weights $\mathbf{W}$ according to $\mathbf{W} = \mathbf{R} - \sum_{i=1}^{10} R_i \mathbf{v}_i$. $\mathbf{R}$ denotes a random vector drawn from distribution 22. We subtract its components $\sum_{i=1}^{10} R_i \mathbf{v}_i = \sum_{i=1}^{10} (\mathbf{R} \cdot \mathbf{v}_i) \mathbf{v}_i$ in the directions of highest variance during the plateau-period. This estimation is based on figure 51 which depicts the respective

eigenvalues λ_i for every eigenvector $\mathbf{v}_i$. We approximate that the components λ_i are $\not\approx 0$ for $i = 1, ..., 10$. The according results are depicted in figure 52, which illustrates a network's output with $\mathbf{W}$ as output-weights and random input-weights $\mathbf{I}$.



(a) Entire spectrum λ_i for $i = 1, ..., N$



(b) Significant eigenvalues λ_i for $i = 1, ..., 10$

Figure 51: **(a)** Eigenspectrum of Σ denoting the indices i with highest eigenvalues λ_i. A PCA-analysis is conducted for $\mathbf{r}(t)$-trajectories resulting from a 150ms input-duration. Equations 26 and 25 are applied to $\mathbf{r}(t)$ for $t \in [4500\text{ms}, 7000\text{ms}]$ during the pleateau-period. Input-weights $\mathbf{I}$ are randomly chosen from distribution 23. **(b)** Zooming into figure (a): The indices $i = 1, ..., 10$ denote the eigenvalues $\lambda_i \not\approx 0$.

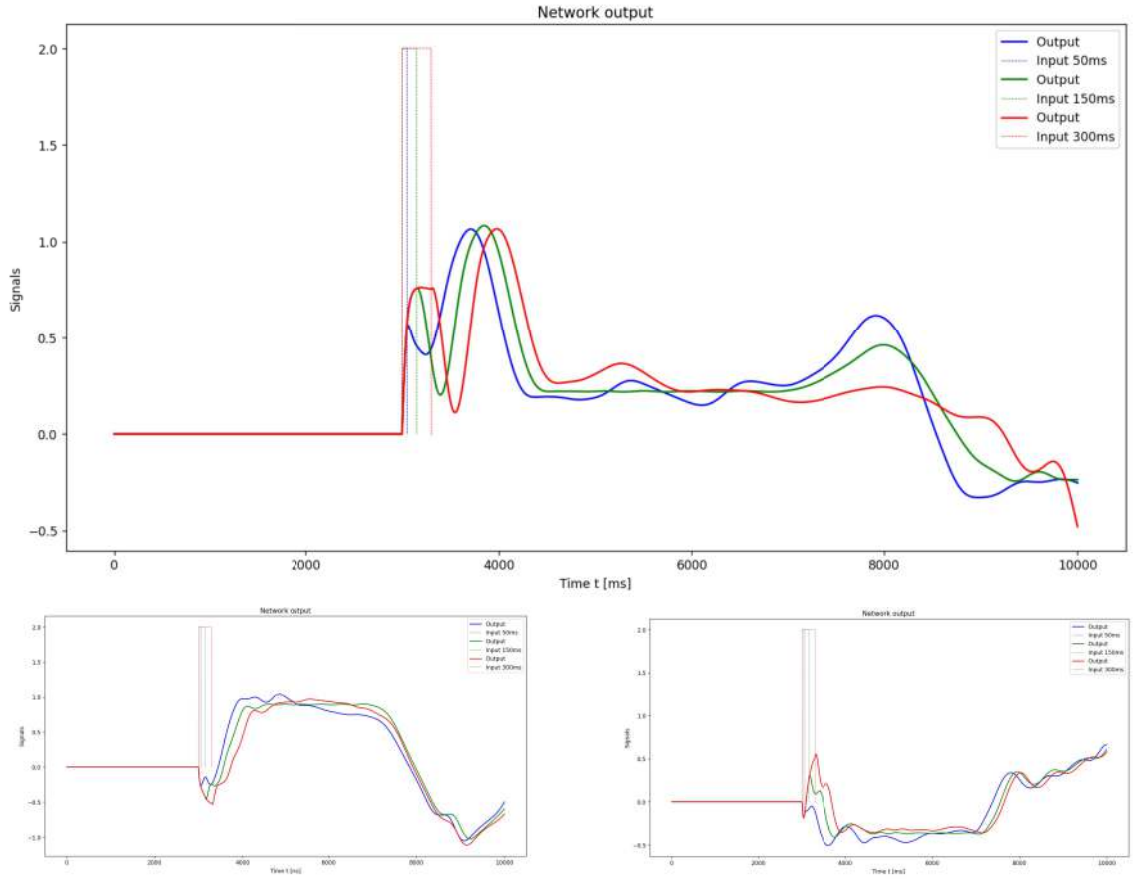
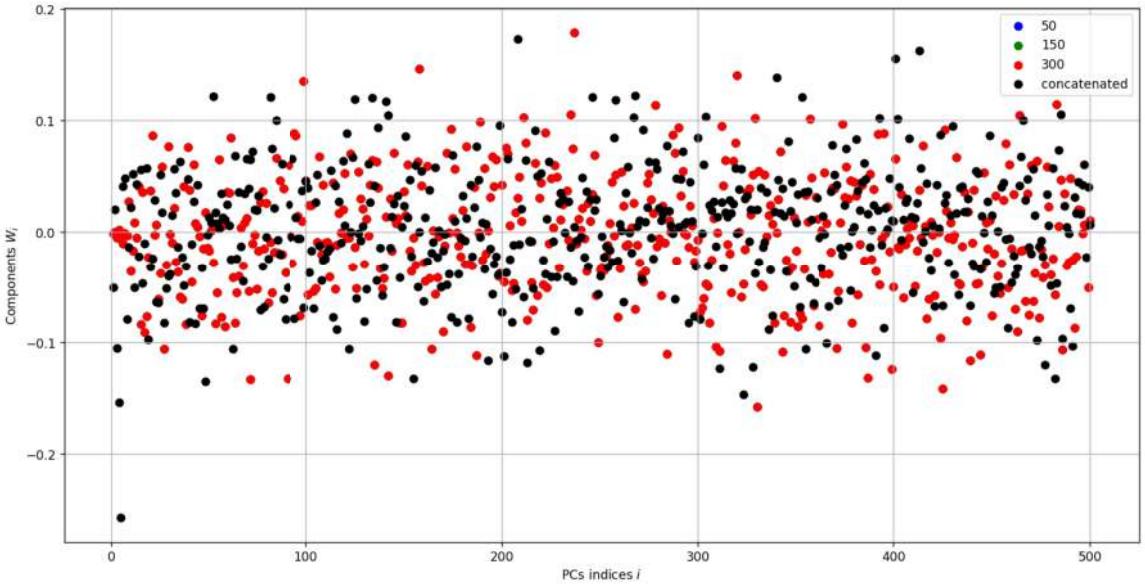


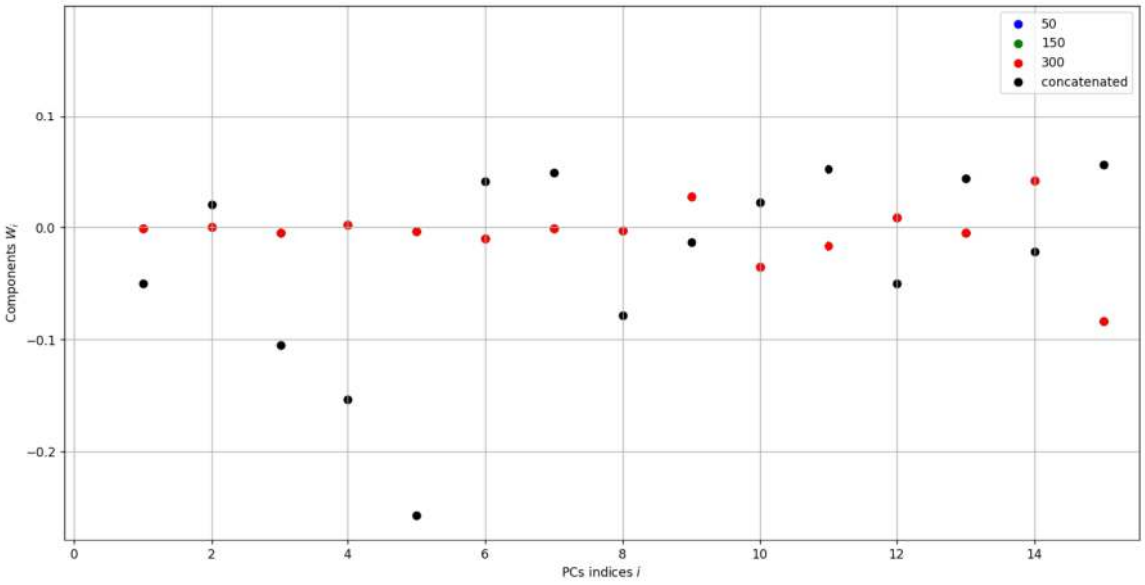
Figure 52: Three exemplary outputs for $\mathbf{W}$-weights defined orthogonally to the axes of highest variance. A PCA-analysis is conducted for $\mathbf{r}(t)$-trajectories resulting from a 150ms input-duration. Equations 26 and 25 are applied to $\mathbf{r}(t)$ for $t \in [4500\text{ms}, 7000\text{ms}]$ during the pleateau-period. The Principal Components are the obtained by the eigenvectors of the covariance-matrix Σ. Weights $\mathbf{W}$ are defined as $\mathbf{W} = \mathbf{R} - \sum_{i=1}^{10} R_i \mathbf{v}_i$ for a random vector $\mathbf{R}$ drawn from distribution 23. The indices $i = 1, ..., 10$ denote the eigenvectors $\mathbf{v}_i$ related to highest eigenvalues $\lambda_i \not\approx 0$. Input-weights $\mathbf{I}$ are randomly chosen from distribution 23. Each example features outputs (thick line) resulting from 50ms (blue), 150ms (green) and 300ms (red) input-durations, while the input-functions are illustrated by a dashed line of equal colours.

At this point, we make the detour indicated in section 5.5.3 and revert back to the meaning of the axis $\mathbf{D}$, which denotes the direction of separation between different $\mathbf{r}(t)$-trajectories. We have observed in figure 48 that training $\mathbf{W}$-weights translates into setting the vector orthogonally to individual $\mathbf{r}(t)$-curves. This is showcased by weight-components $W_i \approx 0$ along axes $\mathbf{v}_i$ of high variance within a single $\mathbf{r}(t)$-trajectory. We make the additional observation in figure 53, that while trained $\mathbf{W}$-weights are orthogonal to individual $\mathbf{r}(t)$-trajectories, this is not the case for W_i-components along axes of the trial-concatenated basis $\{\mathbf{v'}_i\}$. We interpret this finding as follows: The trial-concatenated basis $\{\mathbf{v'}_i\}$ is obtained from a PCA using the $\mathbf{r}(t)$-trajectories of

different trials simultaneously. Thus, we are combining the trajectories $\mathbf{r}(t)$ from a 50ms-input, a 150ms-input, and finally a 300ms-input. Conducting a PCA for the resulting concatenated trajectories yields the new basis $\{\mathbf{v'}_i\}$, which has interesting properties. It denotes the directions of highest variance not only within individual $\mathbf{r}(t)$-curves, but also within their composite. In other words, the $\{\mathbf{v'}_i\}$ contains the axes $\mathbf{v}_i$ from figure 48 and comprises an additional axis, which denotes the variance across all trials. Interpreted geometrically, this additional axis is the vector of separation $\mathbf{D}$ from chapter 5.5.3. Thus, we understand the trial-concatenated basis as $\{\mathbf{v'}_i\} \approx \{\mathbf{v}_i + \mathbf{D}\}$. We now turn out attention to figure 53. It showcases the W_i-components along the axes $\{\mathbf{v}_i\}$ defined for different individual trajectories $\mathbf{r}(t)$. As previously remarked, trained $\mathbf{W}$-weights are orthogonal to these curves, and thus $W_i = \mathbf{W} \cdot \mathbf{v}_i \approx 0$ for $i = 1, ..., 10$. However, we do not obtain this result when projecting trained $\mathbf{W}$-weights onto the new basis $\{\mathbf{v'}_i\}$. Using the imagery $\{\mathbf{v'}_i\} \approx \{\mathbf{v}_i + \mathbf{D}\}$, we interpret this as follows: Training $\mathbf{W}$ sets the vector orthogonally to individual $\mathbf{r}(t)$ curves, but aligns it with the separation-axis $\mathbf{D}$. Figure 54 allows us to make a more generalised statement by depicting median values of W'_i-components for 20 different trained weights $\mathbf{W}$. We observe the same feature $W'_i \not\approx 0$ for $i = 1, ..., 10$.



(a) W_i-components on different plateau-PCs for $i = 1, ..., N$



(b) W_i-components on different plateau-PCs for $i = 1, ..., 15$

Figure 53: Absolute values of components W_i for output-weights $\mathbf{W} = \sum_{i=1}^{N} W_i \mathbf{v}_i$ projected onto the PC's $\{\mathbf{v}_i\}$ of the plateau-period. **(a)** Three single-trial PCA-analyses are conducted for $\mathbf{r}(t)$-trajectories resulting from 50ms (blue), 150ms (green) and 300ms (red) input-durations. Equations 25 and 26 are applied to $\mathbf{r}(t)$ for $t \in [4500\text{ms}, 7000\text{ms}]$. The Principal Components $\{\mathbf{v}_i\}$ are the eigenvectors of the covariance-matrix Σ. Trained weights $\mathbf{W}$ are projected onto the PCs as $W_i = \mathbf{W} \cdot \mathbf{v}_i$. The x-axis denotes the according indices i. These are ordered by magnitude of λ_i, such that λ_0 denotes the highest eigenvalue and λ_N the lowest. Blue and greed data-points are topped by their red counterparts and therefore are not visible, as they all yield the same results. A fourth trial-concatenated PCA is conducted, by concatenating the $\mathbf{r}(t)$-curves for 50ms, 150ms and 300ms

input-durations according to equation 28. The Principal Components $\{\mathbf{v'}_i\}$ are the eigenvectors of the resulting covariance-matrix Σ'. Trained weights $\mathbf{W}$ are projected onto the PCs as $W_i' = \mathbf{W} \cdot \mathbf{v'}_i$ and the resulting W_i-components are marked in black. **(b)** Zooming into figure (a) for $i = 1, ..., 15$. While single-trial PCA's yield W_i-components ≈ 0, the trial-concatenated PCA displays higher W_i'-values.

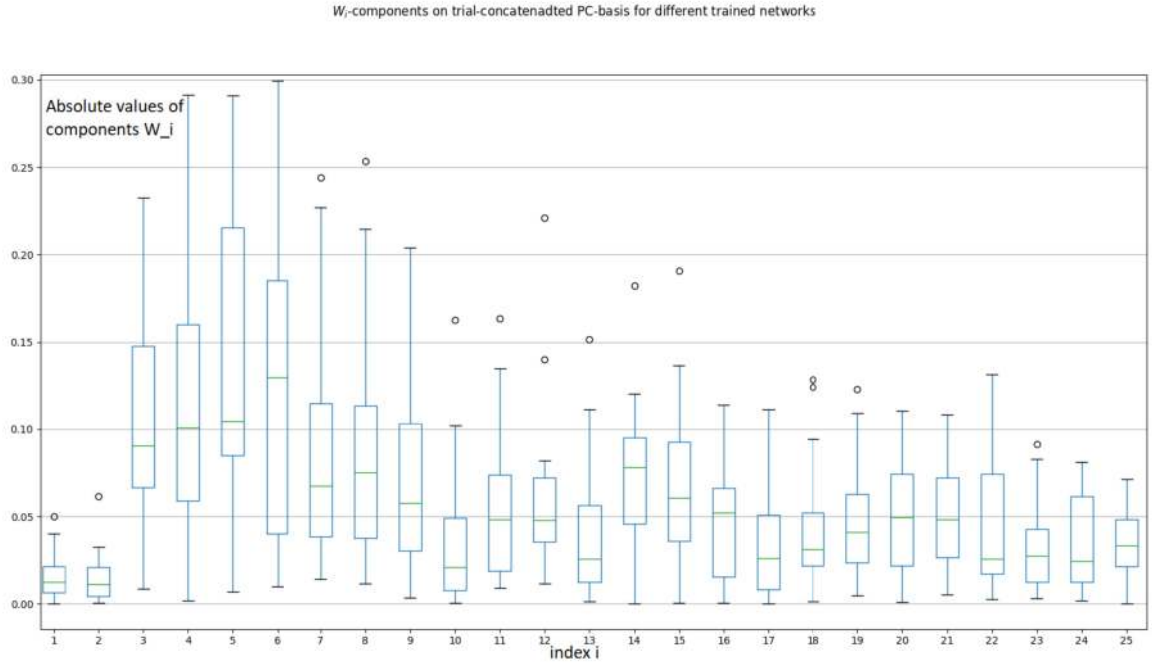


Figure 54: Absolute values of components W_i' for output-weights $\mathbf{W} = \sum_{i=1}^{N} W_i' \mathbf{v}_i$ projected onto the trial-concatenated PC's $\{\mathbf{v'}_i\}$ of the plateau-period. A trial-concatenated PCA is conducted, by concatenating the $\mathbf{r}(t)$-curves for 50ms, 150ms and 300ms at times $t \in [4500\text{ms}, 7000\text{ms}]$. Trained weights $\mathbf{W}$ are projected onto the PCs as $W_i' = \mathbf{W} \cdot \mathbf{v'}_i$. This procedure is repeated for 20 different trained RNNs an the absolute values of the resulting weights W_i' are illustrated in a boxplot. Median values are marked by green lines, upper and lower quartiles are denoted by blue lines and whiskers denote $1.5 \cdot$ Interquartile range (distance between the upper and lower quartiles).

Rounding up this chapter, the key to retaining information in mind lies in avoiding distractions from redundancies and noise. In other words, projecting $\mathbf{r}(t)$-curves to an output $z(t) = \mathbf{W}^T \mathbf{r}(t)$ with plateau-behaviour is rooted in setting the $\mathbf{W}$-vector orthogonally to the axes of highest $\mathbf{r}(t)$-variance. In this manner, we avoid reading out unnecessary fluctuations, and obtain a function $z(t)$ of roughly constant shape. In conclusion, training $\mathbf{W}$ reverts to setting the vector orthogonally to $\mathbf{W}$-trajectories and along their separating axis $\mathbf{D}$.

5.7 Proposing an analytical solution

The concluding chapter evaluates our final understanding of integration- and memory-mechanisms, that have developed through training. In doing so, we consolidate our insights to propose an analytical solution for input- and output-weights, which reproduces the features of interest, and bypasses the training-process.

We follow the structure of this thesis, and divide our concept in an integrating-mechanism to process input-durations, which is followed by a memory-function to store the resulting information. Furthermore, we ground our proposal's framework on the observation, that trained RNNs are able to find solutions for random vectors $\mathbf{I}$, as long as they are correlated with their according output-weights $\mathbf{W}$. We therefore format our solution as

$$\mathbf{W} = \mathbf{I} + \text{integration} + \text{memory}, \tag{48}$$

which finds the related weights $\mathbf{W}$ to any random vector $\mathbf{I}$. We are left with defining the according integration- and memory-terms, which leverage the RNN's dynamics to output a function $z(t) = \mathbf{W}^T\mathbf{r}(t)$ with a plateau-behaviour which is correlated with the respective input-duration.

Beginning with the integration-part, we make use of our findings in section 5.5 and rely on the intrinsic behaviour of $\mathbf{r}(t)$-curves to separate into parallel trajectories. The distance between individual curves is correlated to the according $u(t)$-durations. We leverage this property by calculating the orthogonal axis $\mathbf{D}$ to all $\mathbf{r}(t)$-trajectories, which defines the appropriate direction to read out the separation between individual curves. On a side note, we reiterate the finding that this method is applicable with any vector $\mathbf{I}$. We thus equate the integration-term with $\mathbf{D}$.

Moving on, we draw our attention to the memory-term, which we base on our findings in section 5.6. We have associated a plateau-like property with the ability to tune out the variations of the $\mathbf{r}(t)$-trajectories. Therefore, we conduct a trial-concatenated PCA over the plateau's time-frame to obtain the axes of highest fluctuations $\mathbf{v}_i$ during this period. We proceed by subtracting the respective components, and thus equating the memory term in equation 48 with $-\sum_{i=1}^{10} I_i\mathbf{v}_i = -\sum_{i=1}^{10}(\mathbf{I}\cdot\mathbf{v}_i)\mathbf{v}_i$. In doing so, invoke our previous estimation of the number of significant components $\lambda_i \not\approx 0$ for $i = 1, ..., 10$ from figure 51.

Finally, we are lead to the following proposal for $(\mathbf{I}, \mathbf{W})$-pairs:

$$\mathbf{W} = \mathbf{I} - \sum_{i=1}^{10} I_i \mathbf{v}_i + \mathbf{D} \tag{49}$$

where $\mathbf{D}$ is the separation-vector between $\mathbf{r}(t)$-curves of different input-durations. $\mathbf{v}_i$ are the eigenvectors of Σ for the trial-concatenated PCA of $\mathbf{r}(t)$ over the threshold-period. I_i are the according components $I_i = \mathbf{I} \cdot \mathbf{v}_i$. The resulting output of an RNN with the denoted weights $(\mathbf{I}, \mathbf{W})$ is depicted in figure 52.

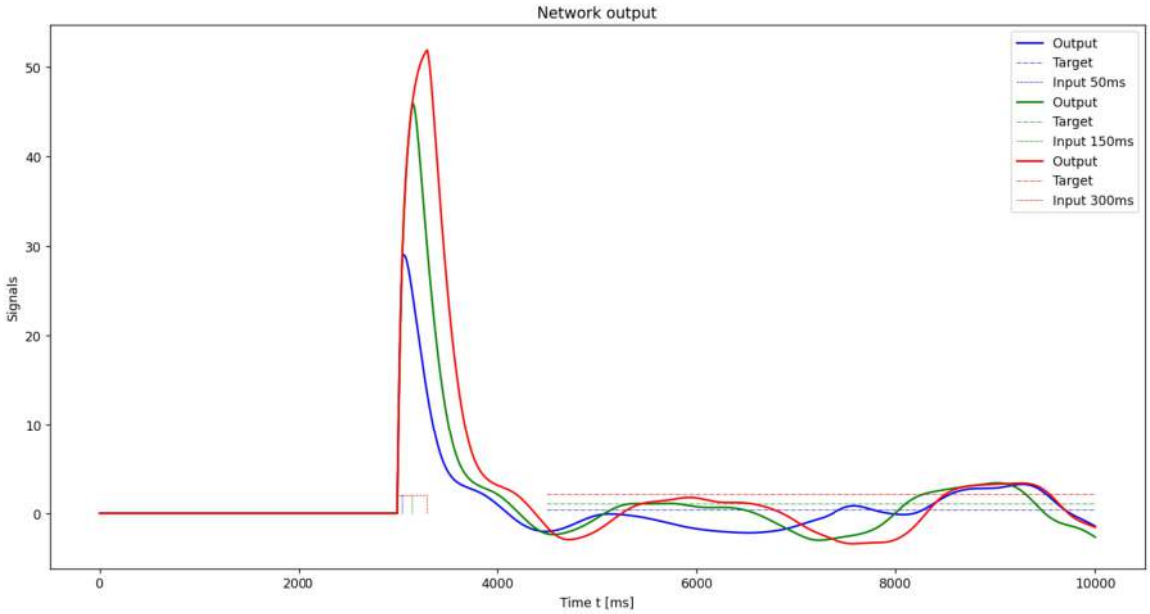


Figure 55: Three network outputs $z(t)$ (thick lines) for input-and output-weights according to equation 49. The colours denote input-durations of 50ms (blue), 150ms (green), 300ms (red), while their respective input- and target-functions are equally colour-coded using dashed lines. Random input-weights $\mathbf{I}$ are initialised using distribution 23. A trial-concatenated PCA over the plateau's time-frame for $t \in [4500\text{ms}, 7000\text{ms}]$ is conducted to obtain the axes of highest fluctuations $\mathbf{v}_i$ during this period. We proceed by subtracting the respective components, $\sum_{i=1}^{10} I_i \mathbf{v}_i = \sum_{i=1}^{10} (\mathbf{I} \cdot \mathbf{v}_i)\mathbf{v}_i$. The separation-axis $\mathbf{D}$ is obtained by using $\mathbf{D} = \mathbf{r}(300\text{ms}) - \mathbf{r}(50\text{ms})$, where we use trajectories $\mathbf{r}(t)$ for a 300ms input-duration. $\mathbf{r}(300\text{ms})$ is the $\mathbf{r}$-vector 300ms after input-onset and $\mathbf{r}(50\text{ms})$ the $\mathbf{r}$-vector 50ms after input-onset. The resulting separation-axis is scaled to $\text{norm}(\mathbf{D}) = 5$. Finally, the output-weights are defined as $\mathbf{W} = \mathbf{I} - \sum_{i=1}^{10} I_i \mathbf{v}_i + \mathbf{D}$ and scaled to a norm of $\text{norm}(\mathbf{W}) = 3$.

Judging from figure 46, it seems that our proposed solution in equation 49 does not yield an RNN able to successfully integrate and memorise input-signals of different durations. To evaluate our approach, we compare the correlation between trained weights $\mathbf{W_t}$ and calculated weights $\mathbf{W_c}$ from equation 49, by using the same random

vectors $\mathbf{I}$. Thus, $\mathbf{W_t}$-quantities result from training RNNs with fixed and random input-weights. The Results are illustrated in figure 56, and yield a mean correlation of

$$\mathbf{W_t}\cdot\mathbf{W_c}/|\mathbf{W_t}|\cdot|\mathbf{W_c}| = 0.548 \pm 0.002 \tag{50}$$

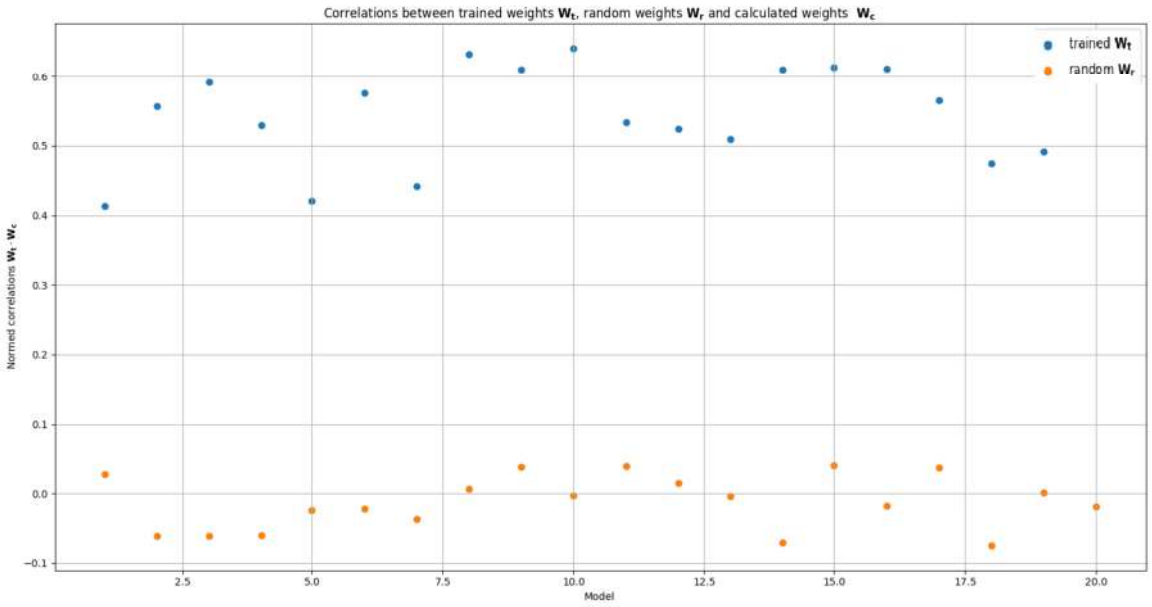


Figure 56: Correlations $\mathbf{W_t}\cdot\mathbf{W_c}/|\mathbf{W_t}|\cdot|\mathbf{W_c}|$ between trained weights $\mathbf{W_t}$ and calculated weights $\mathbf{W_c}$ (blue dots) using the same random vectors $\mathbf{I}$. Random input-weights $\mathbf{I}$ are initialised using distribution 23. We use equation 49 to calculate $\mathbf{W_c}$ as follows: A trial-concatenated PCA over the plateau's time-frame for $t \in [4500\text{ms}, 7000\text{ms}]$ is conducted to obtain the axes of highest fluctuations $\mathbf{v}_i$ during this period. We proceed by subtracting the respective components, $\sum_{i=1}^{10} I_i\mathbf{v}_i = \sum_{i=1}^{10}(\mathbf{I}\cdot\mathbf{v}_i)\mathbf{v}_i$. The separation-axis $\mathbf{D}$ is obtained by using $\mathbf{D} = \mathbf{r}(300\text{ms}) - \mathbf{r}(50\text{ms})$. We compute the trajectories $\mathbf{r}(t)$ for a 300ms input-duration. $\mathbf{r}(300\text{ms})$ is the $\mathbf{r}$-vector 300ms after input-onset and $\mathbf{r}(50\text{ms})$ the $\mathbf{r}$-vector 50ms after input-onset. On the other hand, $\mathbf{W_t}$ is obtained by training a RNN featuring the same input-weights $\mathbf{I}$ as used for the calculation of $\mathbf{W_c}$. Input-weights are fixed, while output-weights $\mathbf{W_t}$ are trained for 100 epochs as described in section 4.3. Both results $\mathbf{W_t}$ and $\mathbf{W_c}$ are compared by calculating their correlation $\mathbf{W_t}\cdot\mathbf{W_c}/|\mathbf{W_t}|\cdot|\mathbf{W_c}|$ for 20 models with different random inputs $\mathbf{I}$ respectively. For comparison, we also compute the correlations $\mathbf{W_r}\cdot\mathbf{W_c}/|\mathbf{W_r}|\cdot|\mathbf{W_c}|$ between the calculated weights $\mathbf{W_c}$ and random weights $\mathbf{W_r}$ (distribution 22) (orange dots).

Despite observing a weak correlation in equation 50, the low standard-deviation of our result suggests that we could have identified some systematics in the integration- and memory-functions of our RNN. We compare our obtained correlation with the optimal value $= 1$ and obtain $^{0.548}/_{1} = 54.8\%$. On the other hand, evaluating its accuracy yields a more satisfying result of $^{0.002}/_{0.548} = 0.365\%$.

In the following chapters, we will review and discuss our understanding of integration and memory, which have lead us to our result in equation 50.

6 Summary

Understanding the firing-rate patterns of the brain poses a significant challenge. An increasingly popular approach involves simulating experimental observations using RNNs, which are more convenient, and share significant characteristics with their biological counterparts - most importantly, their recurrence and their chaotic dynamics. By reverse-engineering our artificial model, we aimed at finding feasible hypotheses for the working-mechanisms of biological networks.

We were interested in integration- and memory-functions, which have been recorded in monkeys and goldfish. Experimental observations reveal, that biological neural networks can accumulate incoming signals, and retain them through persistent activity. We trained artificial RNNs on these tasks, while refraining from evaluating the network's output during signal-integration. In this manner, we maximized the diversity of solutions obtainable through training.

We proceeded by reverse-engineering our model's solution. As an introductory step, we analysed the relationship between intrinsic network-dynamics and computational freedom. We found, that attractors - RNNs with dynamics converging to fixed points or limit cycles - did not perform on our task. Using participation-ratios, we quantified the dynamics' dimensionality, which was directly correlated to performance: We found highest ratios for chaotic dynamics, which also yielded the best results, while networks with fixed-point-dynamics performed worst.

We delved further into the working-principles of our RNN using linear analysis. Initially, based on literature, we speculated that training input-weights might selectively excite certain modes, and similarly, training output-weights could selectively read out certain modes. However, we did not observe such patterns. We therefore changed our approach, and visualised our RNN's dynamics in Principal Component-space.

Drawing inspiration from this new point of view, we began by analysing our RNN's integrating-mechanism. We hypothesized, that inputs of different durations would result in parallel firing-rate-trajectories. To verify this, we projected the first Principal Component of a single trajectory onto the PC-basis found by concatenating various trials. We concluded, that different firing-rate trajectories featured similar orientations in space, and were thus parallel to each other. This lead us to suggesting, that processing different input-durations relied on measuring the distance between the respective

trajectories. As a result of finding a strong correlation between input- and output-weights, we suggested, that training aims to align the output-weights along the axis separating different trajectories to read out their distance.

Moreover, these considerations aligned with our observation, that RNNs with random input-weights performed comparably well on the task. Indeed, we found that firing-rate trajectories were also parallel when featuring arbitrary input-weights. Solving the task by measuring their separation therefore remained possible. Furthermore, RNNs trained exclusively on output-weights still featured high correlation between input- and output-weights, which we had previously attributed to reading out the distance between trajectories. Finally, we tried to confirm that integration relied only minimally on input-weights. We analysed, if training input-weights selected a specific subspace, which might have been relevant for the task, but did not observe such a behaviour.

On the other hand, we proposed, that working-memory was rooted in ruling out redundant firing-rate-dynamics. By setting the output-weights orthogonally to the dynamics' Principal Components, trained RNN's featured plateau-like outputs.

Finally, we attempted at consolidating our findings. We proposed an analytical solution, which featured our key aspects: The minor role of input-weights and their important correlation to output-weights, the alignment of output-weights with the axis of trajectory-separation, and finally, the orthogonal orientation to the dynamics' highest Principal Components. We correlated our results with trained output-weights to estimate their accuracy. While we did not obtain a satisfying value, the standard deviation of our results was comparably small. We concluded, that while our analytical expression was not exact, it still captured certain systematic aspects of our RNN's integration and memory mechanisms.

7 Discussion and outlook

We proceed by outlining possible reasons for our solution's lacking accuracy, and by discussing its scope of applicability.

First of all, we drew inspiration for our analysis from visualizing our RNN's dynamics in Principal Component-space. However, it does not capture the full variation of firing rates, which raises the possibility, that we may have overlooked certain features. Furthermore, our analytical solution includes the axis of separation between individual trajectories. While its concept itself may be appropriate, we may have computed it imprecisely. Our RNN's firing-rate trajectories are probably not exactly parallel. Therefore, computing the separating axis at a single point in space does not yield an accurate result. We could improve our method by redefining the axis as an average over the entire length of the firing-rate trajectories. Lastly, our analysis treats integration and working-memory as two independent mechanisms. We have not explored whether these processes are interconnected. Are we overlooking additional concepts related to the relationship between integration and memory? Mathematically speaking, are we omitting any terms describing their interplay in our proposed analytical solution?

Despite our current result not being entirely satisfactory, we may have identified systematic behaviours underlying our RNN's integration and memory-functions. We rephrase our hypothesis as follows: Our artificial RNNs achieve integration and working memory by aligning their output-weights along the axis of trajectory-separation, while setting them orthogonally to any remaining dynamics. To what extent are these findings applicable to biological networks?

We trained RNNs with random recurrent connections and chaotic dynamics, akin to those observed in their biological counterparts. Furthermore, our training-algorithm did not evaluate the network's output during signal-integration to maximize the diversity of obtainable solutions. Finally, our analysis indicated, that integration and memory was predominantly carried out by output-weights.

Consolidating these aspects, we propose a hypothesis adapted to random and recurrent neuron-populations, which predominantly relies on synaptic plasticity of their output-connections. However, there are countless features, that set artificial recurrent neural networks apart from their biological counterparts. Our model is made of a limited number of neurons with well-defined connections. On the other hand, the brain

comprises uncountable neuronal cells, which subdivide into different populations, and feature complex and ill-understood architectures. They include feedback-loops, activity synchronisation, and other communication-mechanisms between distant brain-areas. In addition, synaptic plasticity is a never ceasing process, such that the brain's architecture is constantly subject to change. It is challenging to determine the extent to which our hypothesis can be applied.

In closing, our approach can be diversified in many ways. To deepen our understanding of integration-mechanisms, varying not only input-durations but also their amplitudes could yield interesting insights. We could further examine the limits of our RNN's integration-capacities by extending the prevailing-times and magnitudes of their incoming signals.

Turning our attention to the memory function, we could explore the time-span for which our RNN can maintain its plateau-like output. What are the boundaries of its forgetfulness? Exploring different network-architectures is also insightful. What is the optimal network-size? Could feedback-loops enhance our RNN's performance? Might alternative training-algorithms lead to different dynamical solutions?

8 Final note

Closing the circle, we revert back to our initial query, namely understanding the code of biological neural networks. Observations in neuroscience showcase indistinct firing of uncountable neurons while processing and transmitting information. The question revolving around the *neural code*, denotes the relationship between firing patterns on one hand, and how they translate to specific information and achieve certain functions on the other hand.

Current research in neuroscience faces a major problem: The brain is an exceptionally complex accumulation of uncountable neurons, divided in distinct populations, interconnected, and simultaneously firing at every moment in time. Having an overview over individual processes seems a daunting challenge. To address such inquiries, we revert back to artificial neural networks. These models are easier to see through, and understanding how they leverage their dynamics may yield conceivable hypotheses for the neural code of their biological counterparts.

Title-page figures:

University of Heidelberg [51], Laboratoire de Physique de l'Ecole normale supérieure LPENS [52]

References

[1] Abdullatif Baba. Neural networks from biological to artificial and vice versa. *Biosystems*, 235:105110, 2024.

[2] Ralph E Hoffman and Thomas H McGlashan. Synaptic elimination, neurodevelopment, and the mechanism of hallucinated "voices" in schizophrenia. *American Journal of Psychiatry*, 154(12):1683–1689, 1997.

[3] Lennart Gustafsson and Andrew P Paplinski. Neural network modelling of autism. *Recent developments in autism research. Nova Science Publishers, Inc., Hauppauge, New York*, pages 100–134, 2005.

[4] Fred Rieke, David Warland, Rob de Ruyter Van Steveninck, and William Bialek. *Spikes: exploring the neural code*. MIT press, 1999.

[5] Michael D Nunez, Paul L Nunez, Ramesh Srinivasan, H Ombao, M Linquist, W Thompson, and J Aston. Electroencephalography (eeg): neurophysics, experimental methods, and signal processing. *Handbook of neuroimaging data analysis*, 1:175–197, 2016.

[6] Charles Sherrington. The integrative action of the nervous system. *The Journal of Nervous and Mental Disease*, 34(12):801–802, 1907.

[7] M Goldman, A Compte, and XJ Wang. Neural integrators: recurrent mechanisms and models. *New Encyclopedia of Neuroscience*, pages 1–26, 2007.

[8] H Sebastian Seung. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences*, 93(23):13339–13344, 1996.

[9] Alexei A Koulakov, Sridhar Raghavachari, Adam Kepecs, and John E Lisman. Model for a robust neural integrator. *Nature neuroscience*, 5(8):775–782, 2002.

[10] Emre Aksay, Robert Baker, H Sebastian Seung, and David W Tank. Anatomy and discharge properties of pre-motor neurons in the goldfish medulla that have eye-position signals during fixations. *Journal of neurophysiology*, 84(2):1035–1049, 2000.

[11] Michael N Shadlen and William T Newsome. Neural basis of a perceptual decision in the parietal cortex (area lip) of the rhesus monkey. *Journal of neurophysiology*, 86(4):1916–1936, 2001.

[12] Carlos D Brody, Ranulfo Romo, and Adam Kepecs. Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations. *Current opinion in neurobiology*, 13(2):204–211, 2003.

[13] Shintaro Funahashi, Charles J Bruce, and Patricia S Goldman-Rakic. Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. *Journal of neurophysiology*, 61(2):331–349, 1989.

[14] Patricia S Goldman-Rakic. Regional and cellular fractionation of working memory. *Proceedings of the National Academy of Sciences*, 93(24):13473–13480, 1996.

[15] VB Mountcastle, MA Steinmetz, and R Romo. Frequency discrimination in the sense of flutter: psychophysical measurements correlated with postcentral events in behaving monkeys. *Journal of Neuroscience*, 10(9):3032–3044, 1990.

[16] Tara H Abraham. (physio) logical circuits: The intellectual origins of the mcculloch–pitts neural networks. *Journal of the History of the Behavioral Sciences*, 38(1):3–25, 2002.

[17] Melody Y. Kiang. Neural networks. *Encyclopedia of Infromation Systems*, 2003.

[18] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95:1–19, 2006.

[19] Georgia Christodoulou and Tim Vogels. The eigenvalue value (in neuroscience). 2022.

[20] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

[21] Orit Shefi, Ido Golding, Ronen Segev, Eshel Ben-Jacob, and Amir Ayali. Morphological characterization of in vitro neuronal networks. *Physical Review E*, 66(2):021905, 2002.

[22] Christian Tomm, Michael Avermann, Carl Petersen, Wulfram Gerstner, and Tim P Vogels. Connection-type-specific biases make uniform random network models consistent with cortical recordings. *Journal of neurophysiology*, 112(8):1801–1814, 2014.

[23] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

[24] Everton J Agnes, Andrea I Luppi, and Tim P Vogels. Complementary inhibitory weight profiles emerge from plasticity and allow flexible switching of receptive fields. *Journal of Neuroscience*, 40(50):9634–9649, 2020.

[25] David Sussillo. Neural circuits as computational dynamical systems. *Current opinion in neurobiology*, 25:156–163, 2014.

[26] Haim Sompolinsky, Andrea Crisanti, and Hans-Jurgen Sommers. Chaos in random neural networks. *Physical review letters*, 61(3):259, 1988.

[27] Vyacheslav L Girko. Circular law. *Theory of Probability & Its Applications*, 29(4):694–706, 1985.

[28] Rodney J Douglas and KA Martin. A functional microcircuit for cat visual cortex. *The Journal of physiology*, 440(1):735–769, 1991.

[29] Solange P Brown and Shaul Hestrin. Intracortical circuits of pyramidal neurons reflect their long-range axonal targets. *Nature*, 457(7233):1133–1136, 2009.

[30] Taro Kiritani, Ian R Wickersham, H Sebastian Seung, and Gordon MG Shepherd. Hierarchical connectivity and connection-specific dynamics in the corticospinal–corticostriatal microcircuit in mouse motor cortex. *Journal of Neuroscience*, 32(14):4992–5001, 2012.

[31] Mieko Morishima and Yasuo Kawaguchi. Recurrent connection patterns of corticostriatal pyramidal cells in frontal cortex. *Journal of Neuroscience*, 26(16):4394–4405, 2006.

[32] Yihan Wang and Qian-Quan Sun. A long-range, recurrent neuronal network linking the emotion regions with the somatic motor cortex. *Cell reports*, 36(12), 2021.

[33] Hairong Lin, Chunhua Wang, Quanli Deng, Cong Xu, Zekun Deng, and Chao Zhou. Review on chaotic dynamics of memristive neuron and neural network. *Nonlinear dynamics*, 106(1):959–973, 2021.

[34] Eugene M Izhikevich. Neural excitability, spiking and bursting. *International journal of bifurcation and chaos*, 10(06):1171–1266, 2000.

[35] Fortunato Tito Arecchi. Chaotic neuron dynamics, synchronization and feature binding. *Physica A: Statistical Mechanics and its Applications*, 338(1-2):218–237, 2004.

[36] 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.

[37] Omri Barak. Recurrent neural networks as versatile tools of neuroscience research. *Current opinion in neurobiology*, 46:1–6, 2017.

[38] Xiao-Jing Wang. Decision making in recurrent neuronal circuits. *Neuron*, 60(2):215–234, 2008.

[39] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[40] Kshitij Tiwari and Nak Young Chong. *Multi-robot Exploration for Environmental Monitoring: The Resource Constrained Perspective*. Academic Press, 2019.

[41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

[42] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[43] Qipin Chen, Wenrui Hao, and Juncai He. A weight initialization based on the linear product structure for neural networks. *Applied Mathematics and Computation*, 415:126722, 2022.

[44] Alex H Williams, Tony Hyun Kim, Forea Wang, Saurabh Vyas, Stephen I Ryu, Krishna V Shenoy, Mark Schnitzer, Tamara G Kolda, and Surya Ganguli. Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis. *Neuron*, 98(6):1099–1115, 2018.

[45] R Engelken, F Wolf, and LF Abbott. Lyapunov spectra of chaotic recurrent neural networks (2020). *arXiv preprint arXiv:2006.02427*.

[46] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison,

Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[48] P Kingma Diederik. Adam: A method for stochastic optimization. *(No Title)*, 2014.

[49] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, 2014.

[50] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[51] University of Heidelberg. Title-page logo. *https://www.uni-heidelberg.de/en/university/history/heidelbergs-university-seal*, 2024.

[52] Laboratoire de Physique de l'Ecole normale supérieure LPENS. Title-page logo. *https://www.ens.psl.eu/laboratoire/laboratoire-de-physique-de-l-ecole-normale-superieure-lpens-umr8023*, 2024.