university of groningen / faculty of science and engineering

Controlling Recurrent Neural Networks with Improved Feature Conceptors

Graduation Project

A.G.B. (Otto) Bervoets (S3417522)

March 6, 2025

Abstract

Conceptors are used to control various neural network architectures. Matrix Conceptors can be used to control recurrent neural networks and feed forward neural networks. Although Conceptor Control yields good performance, each Matrix Conceptor's size is equal to the number of neurons squared. Diagonal Conceptor matrices were introduced to address the high storage cost. However, they lack performance and flexibility of use. Random Feature Conceptors (RFCs) are an expansion of these diagonal conceptors. Using Principal Component Analysis (PCA) this research further improves the RFC architecture. Here we show that with this newly proposed PCA based method, stability and prediction accuracy improve compared to RFCs. These results are broadly applicable in most scenarios where matrix Conceptors and RFCs are used.

Main Supervisor: Prof. Dr. Herbert Jaeger (Artificial Intelligence, University of Groningen)
Second Supervisor: Guillaume Pourcel (Artificial Intelligence, University of Groningen)

Artificial Intelligence
University of Groningen, The Netherlands

1 Introduction

Research in artificial intelligence can be viewed in two ways. On the one hand, one may adopt a top-down approach by beginning at higher levels of cognitive performance, such as reasoning, conceptual knowledge and emotion, and then attempt to model these phenomena using mathematical logic, computer science, and linguistics. However, this research focuses on a bottom-up approach. More specifically, we will research how one can control recurrent neural networks using conceptors as introduced by Jaeger (2017).

Within the field of reservoir computing, a reservoir — also known as a randomly generated recurrent neural network — can be loaded with multiple patterns. Loading refers to the process of adjusting the weights of the reservoir such that a pattern can be regenerated autonomously. Subsequently, the reservoir can be allowed to 'hallucinate' or regenerate one of the stored patterns. Without the use of conceptors, it would be impossible to control which of the loaded patterns is recalled.

Matrix conceptors are the first conceptors introduced by Jaeger (2017). These conceptors are full $N \times N$ matrices for a network size N. Although they yield good performance, matrix conceptors are expensive to store, especially for larger networks as the storage cost increases quadratically with the network size.

Diagonal conceptors were introduced to reduce the storage costs associated with full matrix conceptors. Diagonal conceptors are diagonal $N \times N$ matrices, having only N non-zero elements. As shown by De Jong (2021), diagonal conceptors can yield good performance. However, this requires a less flexible approach than full matrix conceptors. The methods described by De Jong (2021) do yield good performance but cannot consecutively load patterns. This consecutively loading is, however, one of the important upsides of conceptor as this makes continual learning possible, which is an active research field within AI.

Random Feature Conceptors (RFCs) further expand on the idea of diagonal conceptors. First introduced in Jaeger (2017), RFCs linearly map the reservoir state to a higher dimensional space. In this higher dimensional space, the conceptor is represented as a vector of size M, where M is larger than N but, for large reservoirs, is substantially smaller than N^2. In Jaeger (2017) it is shown that for $M = 5N$ performance is already reasonable good. Also, RFCs *do* support consecutively loading patterns and with that continual learning.

The word *random* in Random Feature Conceptors refers to the randomly drawn features used to map the reservoir state to the higher dimensional space. These features can also be viewed as directions in the reservoir space. As RFCs use random directions of the reservoir space, we investigated whether these features could be constructed systematically.

The objective of this research is to find a systematic approach to selecting the features for the Feature Conceptor (FC) that improves on the RFC architecture.

The significance of this research can be seen at different levels. First of all, RFCs can be used similarly as matrix conceptors, without the high storage costs. However, this comes at the cost of a lower task accuracy. This research aims to improve the quality of RFCs while maintaining the storage benefits. For a machine learner who aims to take advantage of the nice properties that conceptors offer, the potential similar performance for a fraction of the storage cost is attractive. Hence, further optimizing and exploring the RFC architecture is important.

On a more fundamental level, the RFC architecture is not biologically implausible (Jaeger, 2017). Therefore, if one wishes to build truly Artificial Intelligence from the ground up, this research is highly significant.

There are also some limitations within this study. First, although the biological plausibility of this research is an important motivator, this research will not cover this, and we refer to Jaeger (2017) for more information about biological plausibility. Also, although conceptors can be applied to a variety of neural networks, this

research is limited to the setting of reservoir computing. However, the findings naturally transfer to other forms of neural networks.

The remaining sections of this thesis will be structured as follows. The literature review, Section 2, will present some background information. Then Section 3 will define the mathematical landscape and the ins and outs of the conceptor architecture. Section 4 will present the methods and experimental settings. Section 5 will present the results of the experiments and Section 6 will interpret these results. Finally Section 7 will conclude this research and give directions for further research.

2 Literature Review

Random Feature Conceptors are introduced in Jaeger (2017), together with the more broadly used matrix conceptor architecture. Since then, matrix conceptors have been used and researched widely. Wherever one can use a matrix conceptor, one could also use a random feature conceptor. Hence, the use of matrix conceptors is also relevant to RFC research.

Conceptors are used in several applications. Let us highlight some of them. Firstly, conceptors can be used to prevent catastrophic forgetting. Catastrophic forgetting refers to the phenomenon where neural networks tend to forget acquired knowledge during the learning of new knowledge. Jaeger (2017) introduces how conceptors may be used to prevent catastrophic forgetting. Building on this, He & Jaeger (2018) describe a conceptor backpropagation algorithm to prevent this effect. The conceptors were used as a "shield" to prevent important knowledge from being overwritten. The authors were able to outperform two other methods on the disjoint MNIST task. The disjoint MNIST task, a common benchmark for continual learning at that time, contains two data sets, one with the numbers 0-4, and a second one with the numbers 5-9. The network then learns the two sets one after the other. Although acieving good performance, the authors mention the relatively large computational cost of the conceptors. The creation of the conceptor has a time complexity of $O(nN^2 + N^3)$, with n observations and N features (in this case$N = 784$ pixels).

Another application of (matrix) conceptors is in that of correcting for (racial/gender/etc.) bias in word representations. Karve et al. (2019) use matrix conceptors to adjust for biases in word embeddings. Embedding bias is the phenomenon that some gender-neutral words (receptionist) lay closer to gender-based words (men/women). Using conceptor matrices, the authors were able to mitigate these issues successfully. The work of Postmus (2024) further extends the line of research. They show that using conceptors they outperform traditional point-based steering methods on several benchmarks.

Next, literature that uses random feature conceptors is discussed. Gast et al. (2017) use Hierarchical Feature Conceptors (HFC) based on RFC to classify bird songs, also proposed by Jaeger (2017). They use a two-step approach, first extracting syllables from songs and then classifying songs based on these syllables. The classification uses HFC. Here the RFCs are used as a sort of representation for a song. Those representations can then be compared to the reservoir states emerging from driving the reservoir with test data and thus be used for song classification.

Meyer zu Driehausen et al. (2019) use RFC architecture in combination with the Hierarchical Feature Conceptors to accurately model bistable perception, the phenomenon of perception alternating between stable states when a subject is presented with two incompatible stimuli. They conclude that the HFC architecture is a promising model for general human perception.

Finally, both De Jong (2021) and Pals (2024) present research concerning diagonal conceptors. Diagonal conceptors are matrix conceptors where only the diagonal is non-zero. One could also say that these are RFC conceptors with identity mapping of size N.

De Jong (2021) explores the mechanics of diagonal conceptors. They show that diagonal conceptors can be used on the same tasks as matrix conceptors. Just like matrix conceptors, diagonal conceptors can be morphed into each other. De Jong (2021) also shows that diagonal conceptors are able to control periodic patterns, chaotic attractors, and human motion patterns. However, by design, the methods that are described by Pals (2024) are unfit for continual learning. De Jong (2021) starts from random conceptors before the loading of the patterns, making the methods useless for continual learning. De Jong (2021) also finds that controlling patterns using diagonal conceptors requires more precision parameter tuning. The range of parameters that leads to a model that is able to learn a pattern is narrower.

The literature review concludes that a conceptor, matrix or vector, is a useful tool. In the RFC architecture, conceptors are cheaper to store than a full matrix conceptor. Research towards diagonal conceptors, which can be defined as an RFC, shows that they lack stability. However, there is no research addressing these issues. The goal of this research is to fill this gap. Hence, how can we improve the prediction accuracy and stability of RFCs?

3 Theoretical Framework

In the following section, the theory will be presented. This follows the theory described in Jaeger (2017). This will be a self-contained description. This research is centred around storing patterns in a reservoir to retrieve them as accurately as possible later. This section will start by describing the theoretical landscape. Following this, the matrix conceptor architecture will be defined. Then, the RFC Architecture will be defined, together with the reasoning behind the RFC architecture. Next, the computation of the weights and the loading of the patterns will be explained. Finally, the adaptations of RFCs will be discussed and the novel way to construct expansion mapping F' will be presented.

3.1 Conceptor Architecture

In this subsection, we will formally define the core model used in this research, the random feature architecture. But first, let us draw up the mathematical landscape, starting with some mathematical notations. In this report, matrices are denoted by a capital letter, where then a_{ij} denotes the element on the i-th row and j-th column of a matrix A. The transpose of matrix A is denoted as A'. Vectors are denoted by lowercase letters, and v_i denotes the i-th element of vector v. To denote a diagonal matrix with the diagonal the elements of vector v $\text{diag}(v)$ is used. Furthermore, when the norm $||\cdot||$ is used, the Frobenius norm is meant unless specified otherwise. The Frobenius norm of a real matrix A is defined as $||A|| = \sqrt{\sum_i \sum_j a_{ij}^2}$. $E[x(n)]$ denotes the expectation (temporal average) of a stationary signal $x(n)$, assuming it is well-defined.

Let $\mathcal{P} = \{p^j | j \in \mathbb{N}\}$ be a set of p patterns. Patterns are signals that are generated by some generator. For example, a random pattern can be generated by sampling from a random number generator. This research only considers discrete patterns and $p^j(n) \in \mathbb{R}^K$ denotes step $n \in \mathbb{N}$ of the sequence j generated by the generator of pattern j with $K \in \mathbb{N}$, the dimension of the signal.

Now let us define the reservoir state vector $r(n) \in (-1,1)^N$, with $N \in \mathbb{N}$, which can be represented as a collection of N neurons. These neurons are connected, and these connections are represented by a weight matrix $W \in \mathbb{R}^{N \times N}$. The reservoir can be autonomously updated by applying the weight matrix on the reservoir state vector leading to $r(n+1) = \tanh(Wr(n))$, with $\tanh(\cdot)$ the tanh activation function. The reservoir can be driven by a pattern p^j. Let us define $W^{\text{in}} \in \mathbb{R}^{N \times K}$ as the input weight matrix. This can be represented by K input neurons connected to the reservoir neurons with weights W^{in}. The state update function of a reservoir driven by a pattern p is defined as $r(n+1) = \tanh(W^{\text{in}} p(n) + Wr(n))$.

For our network to be able to learn, it needs to adhere to the Echo State Property (ESP). The ESP states that for a resevoir that is beeing driven by a pattern p, the effect of initial conditions $r(0)$ should vanish as time passes (Yildiz et al., 2012). To ensure that the ESP is satisfied the spectral radius of the reservoir weight matrix W will be adjusted. Intuitively, a too-high spectral radius will lead neurons to have on-off plipping behaviour, whereas a too-low spectral radius can lead to a too small effect of input on the reservoir state, or too short memory spans.

This research is interested in the ability to regenerate patterns. Hence, a way to construct a pattern from the reservoir states is needed. To do so, the states of the reservoir will be read out using an output matrix $W^{\text{out}} \in \mathbb{R}^{K \times N}$, where W^{out} connects K readout neurons to the reservoir neurons. The K read-out neurons *should* then follow the loaded pattern. The complete mechanics of a reservoir driven by a pattern are described

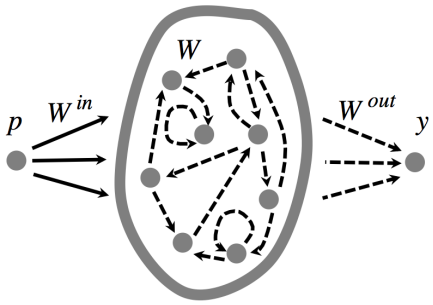

Figure 1: Driving of a reservoir. p is the one-dimensional input pattern that drives the network using W^{in} as the input weight. y the observer or output of the reservoir that uses W^{out} the output matrix. W the internal weight matrix. Taken from Jaeger (2017).

by

$$r(n+1) = \tanh(W^{\text{in}} p^j(n) + W r(n) + b), \tag{1}$$
$$y(n) = W^{\text{out}} r(n), \tag{2}$$

where $b \in \mathbb{R}^N$ is a bias vector. Figure 1 presents a schematic overview of these reservoir architecture. When the reservoir is loaded and runs autonomously, the reservoir is updated according to the following equation

$$r(n+1) = \tanh(Dr(n) + W r(n) + b), \tag{3}$$

where D is the input simulation matrix. Hence, $Dr(n)$ replaces any of the driving patterns, $Dr(n)$ tries to mimic the effect on the rservoir of $W^{\text{in}} p^j(n)$ for *all* patterns. This implies that we have $\tanh(Wr(n) + Dr(n) + b) \approx \tanh(Wr(n) + W^{\text{in}} p^j(n) + b)$ and $W^{\text{out}} r(n) \approx p^j(n)$ for every $p^j \in \mathcal{P}$. Note that Jaeger (2017) also proposes a second method for adapting the weight matrix W. In this case, the unadjusted raw weight matrix W^*, is adjusted to make sure that $\tanh(Wr(n) + b) \approx \tanh(W^* r(n) + W^{\text{in}} p^j(n) + b)$.

Now that the basic reservoir is defined. Let us describe the matrix conceptor architecture. When a pattern p^j drives the reservoir, the neurons get excited. Recall that $r^j(n)$, the reservoir states, represents the excitement of these neurons. The neurons vary in their excitement, leading to a reservoir state sequence $r^j(n), r^j(n+1), \ldots$ which can be represented as a cloud of points in the N-dimensional reservoir state space, referred to as the point cloud related to pattern j. The consecutive states $r^j(n), r^j(n+1) \ldots$ will be collected in X^j. A point cloud related to pattern j can be described using the correlation matrix $R^j = E[r^j (r^j)']$.

Let us now assume that there is a D computed such that we have that $Dr(n) \approx W^{\text{in}} p(n)$ for *multiple* patterns, i.e. multiple patterns are *loaded* into the network. If one wishes to retrieve a certain pattern and starts driving the network autonomously, as described in Equation 3, the behaviour will be unpredictable. The network does not "know" which pattern to retrieve. Ideally, the point cloud of a pattern j is confined to a proper linear subspace $\mathcal{S}^j \subset (-1,1)^N$, such that the state vector can be projected on $\mathcal{S}^j$ using projection matrix $P_{\mathcal{S}^j}$. However, the state $r^j(n)$ vectors usually span the entire state space $(-1,1)^N$ so $P_{\mathcal{S}^j}$ then would be the identity matrix, which would not help to single out a certain pattern.

Using Principal Component Analysis (PCA), the shape of a point cloud can be captured by an N-dimensional ellipsoid whose main axes point in the main directions of the point cloud. This ellipsoid is a geometrical representation of the correlation matrix R of the state points. The singular values $\sigma_1, \ldots, \sigma_N$ represent the lengths

of the axes of the ellipsoid, indicating the importance of the principal components. The directions of the axis are described by the eigenvectors of R. The eigenvector relating to the largest singular value describes the most important direction of the point cloud. Figure 2 gives an illustration of the reservoir activations and their ellipsoids.

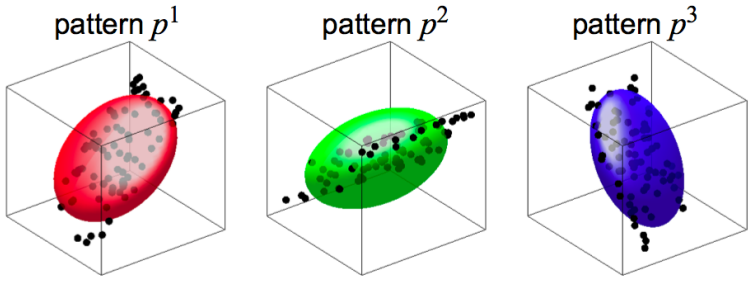

Figure 2: Conceptor geometry (schematic, here network size N = 3). Three patterns p^1, p^2, p^3 excite neural state clouds (black dots) whose shapes can be characterized by ellipsoids (red, green, blue) corresponding to conceptors C^1, C^2, C^3. Taken from Jaeger (2014).

The conceptor is the result of a minimization of a cost function $\mathcal{L}(C^j|R^j,\alpha)$. This cost function has two components. The first component reflects the objective that C^j should behave as an identity matrix for the states that occur in the pattern-driven run of the reservoir. This component is $E[||r^j(n) - C^j r^j(n)||^2]$, the time-averaged difference between the product $C^j r^j(n)$ and the state vector r^j. The second component of $\mathcal{L}$ tries to pull C^j toward the zero matrix. This component is $\alpha^{-2}||C^j||^2$, smaller values for alpha diminishes all singular values of C^j. A conceptor is the solution of the following minimization,

$$C^j(R,\alpha) = \text{argmin}_{C^j} E[||r^j(n) - C^j r(n)||^2] + \alpha^{-2}||C^j||^2,$$

which has the following solution,

$$C^j(R,\alpha) = R^j(R^j + \alpha^{-2} I)^{-1}, \tag{4}$$

with $R^j = E[r^j(n) r^{j\prime}(n)]$ the correlation matrix. If one now wishes to use a conceptor, Equation 3 is adjusted to be

$$r(n+1) = C^j \tanh(Dr(n) + Wr(n) + b),$$

with C^j the conceptor related to pattern j and D the input simulation matrix. Where $Dr(n)$, the input simulation matrix, mimics $W^{\text{in}} p(n)$ for all loaded patterns p^j. How to find D will be discussed later on. Note that applying a matrix conceptor takes N^2 multiplications and N^2 additions. As $C \in \mathbb{R}^{N \times N}$ the size of a matrix conceptor grows quadratically with the number of neurons in the reservoir.

Let us expand this to the random feature conceptor architecture. The random feature conceptor still tries to map the reservoir state space towards an ellipsoid that describes the important directions associated with a pattern. Just as with the matrix conceptor, the dominant directions of the ellipsoid are let go free, whereas the unimportant directions are dampened. Matrix conceptors achieve this by applying a conceptor matrix $C \in \mathbb{R}^{N \times N}$ onto the reservoir states r. However, within the RFC architecture, the storage cost of an Conceptor are significantly lower.

In the RFC structure, the N-dimensional reservoir state is first mapped to a higher M-dimensional feature space or z-state. The components of the higher dimensional z state are $z_i = c_i f_i' r$. The conception weights c_i

take over the role of the matrix conceptors with $c = (c_1, \dots c_M)$ denoting the conceptor vector. Note that the conceptor multiplications now are M scalar multiplications. In more detail, the following steps are taken:

1. Expand the N-dimensional reservoir state r in to the M-dimensional random feature space by a random feature map $F' \in \mathbb{R}^{M \times N}$ with $F' = (f_1, \dots, f_M)'$ by computing the M-dimensional feature vector $F'r$.
2. Multiply each of the M feature projections $f_i' r$ with an conception weight c_i^j to get a conceptor-weighted feature state $z = \text{diag}(c)F'r$, where the conception vector $c^j \in \mathbb{R}^M$ with $c^j = (c_1^j, \dots, c_M^j)'$ is made of the conception weights and c^j is the conceptor associated with pattern j.
3. Project z back to the reservoir state by a random $N \times M$ back projection matrix $\tilde{G} \in \mathbb{R}^{N \times M}$, closing the loop.

The matrices W and potentially $\tilde{G}$ are initially random and recomputed later on. They can be joined in a single map $G^* = W\tilde{G}$. This leads to the following update cycle for the RFC architecture:

$$r(n) = \tanh(G^* z(n) + W^{\text{in}} p(n) + b), \tag{5}$$
$$z(n+1) = \text{diag}(c)F'r(n), \tag{6}$$
$$y(n) = W^{\text{out}} r(n),$$

with $r(n) \in \mathbb{R}^N$ and $z(n), c \in \mathbb{R}^M$. G^* will later be replaced by G, a regularization of G^*. A schematic overview of the RFC architecture is presented in Figure 3.

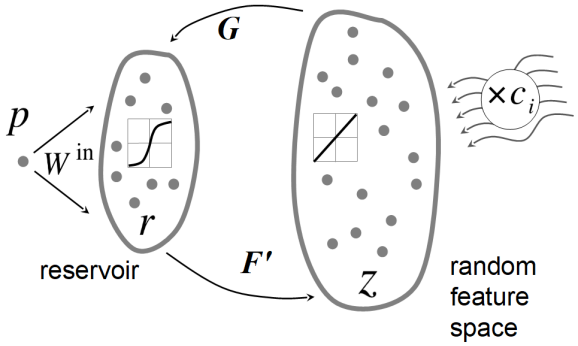

Figure 3: Schematic overview of the RFC structure. The small dots represent neurons, the map F' maps the r-state to the higher dimensional z-state where the conceptor multiplications are done. After this, G maps the z-state back to the reservoir. Using W^{in}, the network is driven by p. Note that a readout y is not displayed here but could be done by observing r using a matrix referred to as W^{out}. Figure retrieved from Jaeger (2017).

Now that the RFC structure is defined, let us describe how to find the conceptor vector c^j. The individual conceptor weights c_i of vector conceptor $c = (c_1, \dots, c_M)$ can be computed using

$$c_i^j = E[z_i^j(n)^2](E[z_i^j(n)^2] + \alpha^{-2})^{-1}, \tag{7}$$

which is the solution of minimizing the following objective function

$$E[(z_i^j(n) - c_i z_i^j(n)^2] + \alpha^{-2} c_i^2$$

with α the aperture. The aperture, an important parameter, controls the amount of damping. A small aperture forces the conceptor weights to zero, limiting all directions of the feature space. A large aperture forces the conceptor weights towards one, limiting no direction of the feature space.

To compute c_i^j, $z_i^j(n)$ needs to be collected. This will be done in the following manner. First, the basic network without expansion to higher dimensional space as described in Equation 1 will be driven. After an initial washout phase, n_c reservoir states will be collected. Next, using map F the states $r^j(1),\ldots,r^j(n_c)$ will be mapped to the higher dimensional z-space, resulting in states $z^j(1),\ldots,z^j(n_c)$. These will then be used to compute the conceptors as described in Equation 7.

Let us now highlight the intuition behind the map F'. The elements f_i of F can be viewed as *directions* in the reservoir space. Hence, $f_i' r$ measures the magnitude of the state space in the direction of f_i. The conceptor weight c_i can now manipulate the magnitude of this specific direction. By doing this for many directions of the state space, some directions of the reservoir state can still be suppressed depending on the intended pattern. Note that in the *random* feature conceptor architecture, the directions of the state space f_i are sampled randomly.

3.2 Computing weights

In the following section, the process of loading the reservoir will be described. First, loading the RFC architecture will be described. This will follow the order that is used in practice. Afterwards, the loading of the matrix conceptor architecture will be described. Although matrix conceptors are not the main focus, of this research, the RFC architecture will be compared to the matrix conceptor architecture and hence, for completeness, it will be described.

As a first step, F', b, W^{in} and G^* are sampled from a centred normal distribution with $\sigma_{F'} = \sigma_{G^*} = 1$ and $\sigma_b, \sigma_{W^{\text{in}}}$ chosen optimally. Note that G^* is the raw version of G. Since the $N \times N$ map G^*F' replaces the weight matrix W of the basic echo state network, the map G^*F' needs to be scaled such that the network attains the ESP as discussed in Section 3.1. Hence F' and G^* are scaled such that the combined map G^*F' attains a prescribed spectral radius.

Next, D, W^{out} and G are (re)computed. The matrixix $D \in \mathbb{R}^{N\times M}$ is called the input simulation matrix. The goal of this matrix is to imitate the effects of the driving pattern on the reservoir. Hence, D needs to be chosen such that we have $Dr(n) \approx W^{\text{in}} p(n)$.

To obtain these weight matrices, the states $r^j(n)$ and $z^j(n)$ need to be collected. This is done by driving the system with each pattern p^j as defined in Equation 5 and Equation 6 for n_{harvest} steps, collecting the states of $r^j(n)$ and $z^j(n)$ in matrix $X \in (-1,1)^{N\times n_{\text{harvest}}|\mathcal{P}|}$ and $Z \in \mathbb{R}^{M\times n_{\text{harvest}}|\mathcal{P}|}$, where $|\mathcal{P}|$ denotes the number of patterns in $\mathcal{P}$. Hence, all the reservoir states and the z-states are concatenated in a matrix. Similarly, the patterns are collected in $P \in \mathbb{R}^{K\times n_{\text{harvest}}|\mathcal{P}|}$. To make sure the initial reservoir state is of no influence, the system is first washed out for n_{washout} steps before collection.

Using the matrices X, Z and P, the weight matrices can be computed. The input simulation matrix $D \in \mathbb{R}^{N\times M}$ can be computed by solving the regularized linear regression

$$\begin{aligned} D &= \text{argmin}_{\tilde{D}} \sum_{n,j} ||W^{\text{in}} p^j(n) - \tilde{D} z^j(n-1)||^2 + \beta_D ||\tilde{D}||^2, \\ &= ((ZZ' + \beta_D I_M)^{-1} Z (W^{\text{in}} P)')', \end{aligned} \tag{8}$$

where $\tilde{D}$ is the raw version of D, β_D a suitably chosen Tychonov regularizer, and I_M the $M \times M$ identity matrix. The intuition behind Equation 8 is that autonomous system update $z^j(n+1) = \text{diag}(c^j)F' \tanh(G^* z^j(n) + Dz^j(n) + b)$ should be able to simulate input-driven updates as defined in Equation 5 and Equation 6.

Next, W^{out} is computed by solving

$$W^{\text{out}} = \text{argmin}_{\tilde{W}} \sum_{n,j} ||p^j(n) - \tilde{W} r^j(n)||^2 + \beta_{W^{\text{out}}} ||\tilde{W}||^2, \tag{9}$$

$$= ((XX' + \beta_{W^{\text{out}}} I_N)^{-1} XP')', \tag{10}$$

where $\tilde{W}$ is the raw version of W^{out}, $\beta_{W^{\text{out}}}$ a suitably chosen Tychonov regulizer, and I_N the N-dimentional identity matrix. The intuition behind Equation 9 is to retrieve $p^j(n)$ from $r(n)$ as well as possible for all j patterns at all timesteps n. Equation 10 shows the closed form solution of Equation 9.

Optionally, which is the way how it is done in this research, one can also recompute G^* by solving the trivial regularized linear regression:

$$G = \text{argmin}_{\tilde{G}} \sum_{n,j} ||G^* z^j(n) - \tilde{G} z^j(n)||^2 + \beta_G ||\tilde{G}||^2,$$
$$= ((ZZ' + \beta_G I_M))^{-1} ZZ')'$$

where G^* the raw weights, for a suitable chosen Tychonov regularizer β_G, and I_M the $M \times M$ identity matrix. While G^* and G should behave virtually identically on the training inputs, the average absolute size of entries in G will be (typically much) smaller than the original weights in G^* as a result of the regularization. Such regularized auto-adaptations have been found to be beneficial in pattern-generating recurrent neural networks (Reinhart & Steil, 2010), and Jaeger (2017) took advantage of this re-computation of G.

When one now wants to re-generate a pattern p^j one runs the following system:

$$r(n) = \tanh(Gz(n) + Dz(n) + b),$$
$$y(n) = W^{\text{out}} r(n),$$
$$z(n+1) = \text{diag}(c) F' r(n),$$

where the stating state $z(n)$ can be initialized randomly by sampling from the normal $\mathcal{N}(0, \frac{1}{2})$. Another option is to use the state vector z^j at the end of the training sequence for pattern j. The latter case is equal to forecasting the next timesteps of pattern j.

Let us now describe the loading of matrix conceptors. This method is exactly equal to the one described in Jaeger (2017), and is based on recalculating W^*.

First, W^{in}, b and W^* are sampled from a centred normal distribution with $\sigma_{W^*} = 1$ and $\sigma_{W^{\text{in}}}, \sigma_b$ chosen based on an optimization process discussed later. The internal weight matrix W^* is adjusted to have a sparsity of $\frac{10}{N}$, i.e. on average neurons are connected with 10 other neurons. Next, W^* is scaled to a prescribed spectral radius.

To load the pattern, W^* needs to be recalculated. To read out from the loaded reservoir, W^{out} needs to be computed. To do this, the reservoir states $r^j(n)$ for each pattern need to be collected. This is done by driving the reservoir with each pattern as defined in Equation 1. After an initial washout period of length n_{washout}, the reservoir states are collected for n_{harvest} steps. These recorded reservoir states $r^j(n)$ are collected and combined in matrix $X \in (-1,1)^{N \times n_{\text{harvest}}|\mathcal{P}|}$. Defining $\tilde{X}$ as the one-timestep-delayed version of X, hence recording $r(n-1)$. The driving patterns are also collected in $\P \in \mathbb{R}^{K \times n_{\text{harvest}}|P|}$.

The conceptors C^j can be computed according to Equation 4. The internal weight matrix W will be computed using ridge regression. Writing B for the $N \times (|P| \cdot n_{\text{harvest}})$ matrix whose columns are all identical equal to b. The ridge regression solution is defined as

$$W = ((\tilde{X}\tilde{X}' + \beta_W)^{-1}\tilde{X}(\tanh^{-1}(R) - B)')',$$

whit β_W a suitably chosen Tychonov regularizer. To compute W^{out}, the readout matrix, compute

$$W^{\text{out}} = ((RR' + \beta_{W^{\text{out}}} I_N)^{-1} RP')',$$

with $\beta_{W^{\text{out}}}$ a suitably chosen Tychonov regularizer.

3.3 Construction of F' and G

As stated previously, in the random feature conceptor architecture described by Jaeger (2017) F' and G are constructed randomly. This research aims to explore if there exist methods to construct these maps to improve task accuracy. For every case in this research, G^* will be constructed using $G^* = WF$, with W the randomly sampled weight matrix and F the expansion map.

This research aims to improve task accuracy by changing the construction of F. The goal is to choose the directional vectors f_i that combined make up F in such a manner that they are better able to control the reservoir than random vectors. Hence, we want to find directional vectors f_i that point in directions of the reservoir space that are relevant to a certain pattern. A very simple way to do this would be by driving the reservoir with a certain pattern p^j and sampling the reservoir states that occur. In this case, the reservoir *without* expansion would be driven. Hence, the reservoir update follows

$$r^j(n+1) = \tanh(W^{\text{in}} p^j(n) + W^* r(n)), \tag{11}$$

where $r(n)$ would be sampled at some fixed interval and stored for all patterns p^j. These sampled reservoir states $r(n)$ are scaled to have unit norm and used as directional vectors f_i to construct F.

In the original research Jaeger (2017), G^* is sampled from a random normal distribution. However, if this were done, the dynamics of the original weight matrix W^* would be lost. To ensure the reservoir dynamics captured by F are still present in the (autonomous) update sequence, one would define $G^* = WF$.

However, during initial testing, sampling directly from the reservoir states yields poor performance. The average inner product of the sampled reservoir vectors f_i equals around 0.8 as was observed during initial experimentation. This implies that the directional vectors f_i all point in very similar directions, making distinguishing the different patterns with limited features hard. At this point, this research direction is not further investigated.

A second, more involved method is also proposed. First, the reservoir is driven by the patterns p^j as described above in Equation 11. For each pattern j, all reservoir states $r^j(n)$ are stored in a matrix X^j. Next, construct the reservoir state correlation matrix $R^j = X^j(X^j)'$ for each pattern. Now, a Principal Component Analysis (PCA) is conducted on each R^j. This results in a set of N unit vectors for each pattern j that point in the leading directions of the reservoir state when it is driven by pattern j. For each pattern, the first k_{PCA} components are selected as vectors f_j to form F.

Note that X^j forms a point cloud in the reservoir space. Also note that by applying PCA on the reservoir correlation matrix $R^j = X^j(X^j)'$, the main axes of the scattering directions of this are found. The idea here is that using these main scattering directions as vectors helps to control the network in choosing which pattern

to regenerate. The inspiration for the PCA method originates from the idea that if one wants to control the reservoir activations in directions that are important for a certain pattern, it may help to know these directions.

Also note that the maximum number of PCA vectors per pattern is N, the size of the reservoir. Hence, in both of the above-described methods, it is possible and sometimes necessary to add randomly sampled unit vectors to the map F. Again, compute $G^* = WF$ to ensure that the reservoir dynamics are preserved.

4 Experiments

The following section will describe the experiments investigating the performance of the newly proposed feature conceptors relative to the original feature conceptors. The performance of the proposed architectures will be assessed on prediction accuracy and stability. For completeness, the performance of matrix conceptors will be reported as well. First, the chaotic patterns will be presented, next, the assessment criteria will be explained, and lastly, the hyperparameter tuning will be explained.

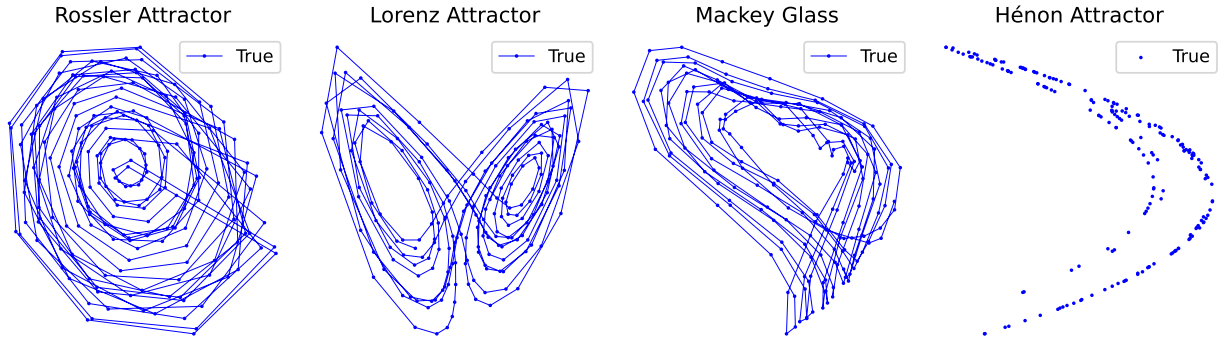

Figure 4: 200 timesteps from the true patterns that are stored in the reservoirs.

All models are loaded with four chaotic attractors to assess the performance of the different models. These chaotic attractors, taken from Jaeger (2017), are defined in Appendix A. Figure 4 displays a 200-timestep sample from the true patterns. Each pattern is a two-dimensional representation of a chaotic attractor.

The reservoir was loaded with a training sequence of 3000 timesteps for each pattern. The prediction accuracy was asses by computing the NRMSE, as defined in Figure 4, on the collection of the next 84 timesteps. The NRMSE of network-generated signal $y(n)$ matched against the target pattern $p(n)$ is defined as

$$\text{NRMSE} = \sqrt{\frac{[(y(n)-p(n))^2]}{[p(n)^2]}},$$

where $[\cdot]$ is the mean operator over data points n. After loading the patterns, the last z-state or r-state (in the case of matrix conceptors) is stored, and the model predicts from that point on.

Next, the stability of the architecture needs to be assessed. Stability will be tested by adding some (normally distributed) noise to the saved last training reservoir z-state or r-state. Then, by using visual inspection, the stability will be assessed. A model is considered stable for a certain amount of noise if it recovers within 5000-timesteps after applying the perturbation five times in a row. Each time, a new model was initialized, trained and then perturbed. The more noise a model can recover from, the more stable the model.

First the model is run for 5000 timesteps without perturbation. If the model does "derail" within these 5000 timesteps, it is considered not stable, as even without noise the model is unable to behave like the loaded pattern. After this, the model is perturbed with decreasing amounts of noise. The noise levels that are tested are $\sigma = 1, 0.5, 0.1, 0.05, \ldots, 0.00005, 0.00001$. If, by assessment of visual inspection, the model "derails" for at least one of the chaotic attractors, the noise will be decreased. The highest amount of noise that can be added without the model derailing will be reported. Let us now turn to the optimization of the hyperparameters.

Table 2 displays all parameters that are relevant for the feature conceptors. Similarly, Table 3 displays the

parameters that are optimized for the matrix conceptor. As can be seen, next to the starting value a range, step type and step size are reported. When a relative step size is used, the step size is relative to the current value. For relative stepsize , the actual step size is calculated by the current value $\times$ relative step size. For example, if the current value is 10, and the relative step size is 0.1, the actual step size = $10 \times 0.1 = 1$, if the current value is 100, the same relative step size would imply an actual stepsize of 10. Relative step sizes make sure that for parameters with a wide spread, the step sizes are still of significant size. If a step type is absolute, the step size is equal to the step size that is reported.

The number of parameters is large, and their tuning is delicate. Hence, a full grid search is not feasible. Therefore, the parameters are optimized by looping over the parameters and finding the optimal parameter for each setting. Hence, one-by-one each parameter is optimized keeping the others fixed. This looping over the parameters is done three times.

When optimizing one parameter, the average NRMSE for 5 runs is calculated. Each run, the models are re-initialized. The NRMSE is calculated by comparing the nest 84 predicted timeseries when continuing from the last stored reservoir state and with the true signal for these time steps. Next, the direction for the parameter is decided (bigger or smaller) by computing the NRMSE for the two surrounding parameter values. The direction with the lowest NRMSE is chosen and as long as the NRMSE improves, steps in this direction will be taken. When the parameter is at its (local) optimal value, proceed to the next parameter. This is then looped three times. After optimizing the parameters, the hyperparameters with the lowest average NRMSE will be used to conduct 30 experiments, the average NRMSE on these 30 experiments will be reported.

For the Feature Conceptors, the reservoir size will be set to $N = 250$. The models will be optimized for different sizes of the expansion map M. The values of M are presented in Table 1. The matrix conceptor architecture will be optimized for both a reservoir size of $N = 250$ and $N = 500$. Note that only the prediction error is used as a metric to optimize the parameters.

M	100	125	187	250	312	375	500	750	1000
Multiple of N	$\frac{2}{5}$	$\frac{1}{2}$	$\frac{3}{4}$	1	$\frac{5}{4}$	$\frac{3}{2}$	2	3	4

Table 1: Values of M and their multiple of N

Parameter	Starting Value	Range	Step type	Step size
$\sigma_{\text{reservoir}}$	$5 \cdot 10^{-4}$	$[0, 0.5]$	Relative	0.1
σ_{signal}	$7 \cdot 10^{-3}$	$[0, 0.5]$	Relative	0.1
k_{PCA}	$\min(N/4, M/4)$	$[0, N]$	Relative	0.1
α_{Rossler}	48	$(0, 10^4]$	Relative	0.1
α_{Lorenz}	40	$(0, 10^4]$	Relative	0.1
$\alpha_{\text{Mackey Glass}}$	40	$(0, 10^4]$	Relative	0.1
α_{Henon}	40	$(0, 10^4]$	Relative	0.1
$\beta_{W^{\text{out}}}$	0.02	$[0, 10]$	Relative	0.25
β_D	$6 \cdot 10^{-5}$	$[0, 10]$	Relative	0.1
β_G	0.4	$[0, 10]$	Relative	0.1
$\sigma_{W_{\text{in}}}$	1	[0,2]	Absolute	0.05
σ_b	1	[0,2]	Absolute	0.05
$\rho(F'G)$	1.4	$(0, 2]$	Absolute	0.1
$\rho(W)$	1.2	$(0, 2]$	Absolute	0.1

Table 2: Feature Conceptor parameters to optimize, starting value and range. $\sigma_{\text{reservoir}}$ and σ_{signal} denote the standard deviation of the noise added to respectively the reservoir and the signal, k_{PCA} denotes the maximum number of components to take from the PCA. β_i denotes the Tikhonov regularizor for matrix i, α_i denotes the aperture for pattern i and σ_i denotes the standard deviation when sampling matrix i from the normal distribution, $\rho(\cdot)$ denotes the spectral radius. Note that k_{PCA} is only relevant for PCA FC.

Parameter	Starting Value	Range	Step type	Step size
$\sigma_{\text{reservoir}}$	$5 \cdot 10^{-4}$	$[0, 0.5]$	Relative	0.1
σ_{signal}	$7 \cdot 10^{-3}$	$[0, 0.5]$	Relative	0.1
α_{Rossler}	150	$(0, 10^4]$	Relative	0.1
α_{Lorenz}	50	$(0, 10^4]$	Relative	0.1
$\alpha_{\text{Mackey Glass}}$	100	$(0, 10^4]$	Relative	0.1
α_{Henon}	60	$(0, 10^4]$	Relative	0.1
β_W	0.4	$[0, 10]$	Relative	0.1
$\beta_{W^{\text{out}}}$	0.02	$[0, 10]$	Relative	0.25
$\sigma_{W_{\text{in}}}$	1	[0,2]	Absolute	0.05
σ_b	1	[0,2]	Absolute	0.05
$\rho(W)$	1.2	$(0, 2]$	Absolute	0.1

Table 3: Feature Conceptor parameters to optimize, starting value and range. $\sigma_{\text{reservoir}}$ and σ_{signal} denote the standard deviation of the noise added to respectively the reservoir and the signal, k_{PCA} denotes the maximum number of components to take from the PCA. β_i denotes the Tikhonov regularizor for matrix i, α_i denotes the aperture for pattern i and σ_i denotes the standard deviation when sampling matrix i from the normal distribution, $\rho(\cdot)$ denotes the spectral radius.

All experiments where implemented using python 3.12.0 All matrix calculations where done using NumPy 2.1.1 and Scikit Learn 1.5.2 was used to solve the ridge regressions. The code can be found at `https://github.com/ottobervoets/RandomFeatureConceptors`.

5 Results

The following section presents the results of the experiments as described in Section 4. First, the results of the hyperparameter tuning will be presented, and with that the prediction accuracies of the models. Next, the stability results will be presented. Finally, some figures will be presented to gain further insights into the obtained results.

The performance of the PCA Feature Conceptors and Random Feature Conceptors are presented in Figure 5. For every M, the NRMSE of the PCA FC is on average lower than the one of the RFC. Also, with the increasing size of M, the NRMSE decreases. For an expansion size of 312, both the RFC and the PCA FC obtain a lower NRMSE than a matrix conceptor with $N = 250$. Note that the performance of a matrix conceptor with a reservoir size of $N = 500$ is similar to a PCA FC with $N = 250$ and $M = 500$. For large M the mean NRMSE for the PCA FC and RFC are lower than a matrix conceptor with $N = 500$.

To gain insight into the spread, symmetric error bars of one standard deviation are plotted. Note that the NRMSE is skewed towards 0, the minimum value for the NRMSE. Table 3 and Table 2 in Appendix B present the optimal hyperparameters that are used to generate Figure 5.

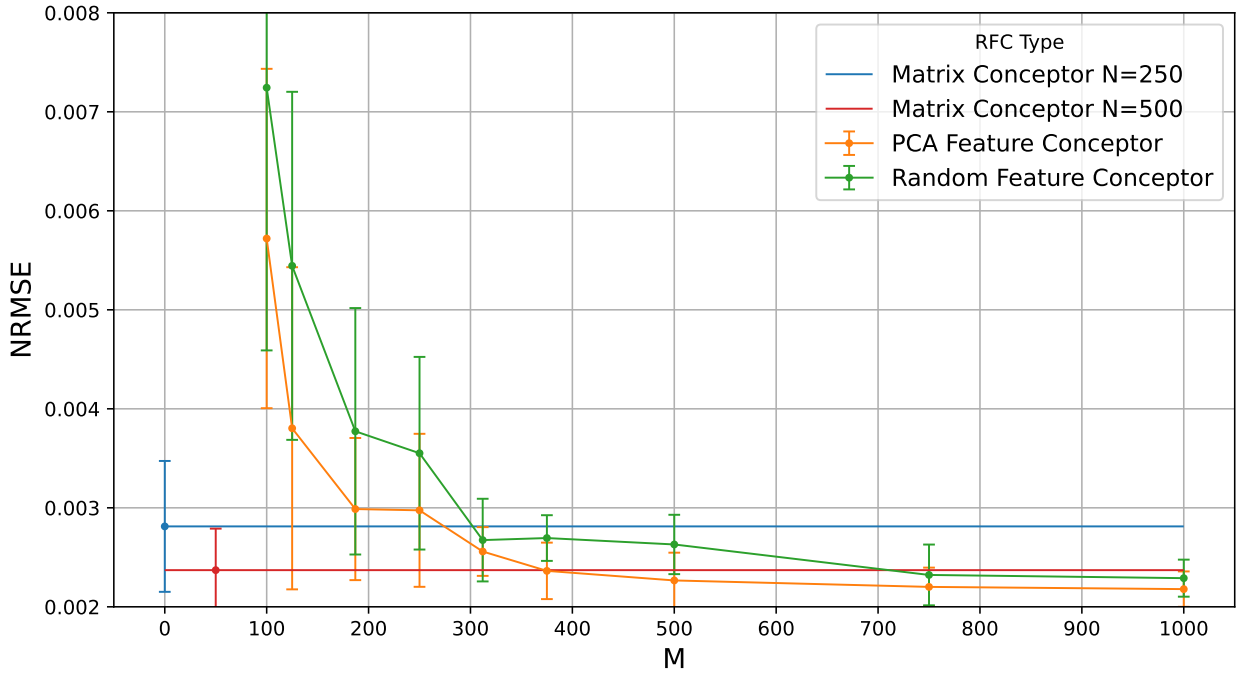

Figure 5: Average NRMSE over 30 experiments for all the architectures with optimal hyperparameters. Error bars indicate one standard deviation. Note that the matrix conceptors have no expansion map and their error bar is plotted at $M = 0$ and $M = 50$.

To give some intuition, Figure 6 and Figure 7 present some prediction results of a PCA RF with an expansion mapping size of $M = 100$ and $M = 1000$. As can be observed, for $N = 100$ to Lorenz Attractor is not recalled properly. Also note that the Mackey Glass is recalled as a simple path, without much detail. For $N = 1000$ the characteristics of the are captured a lot better.

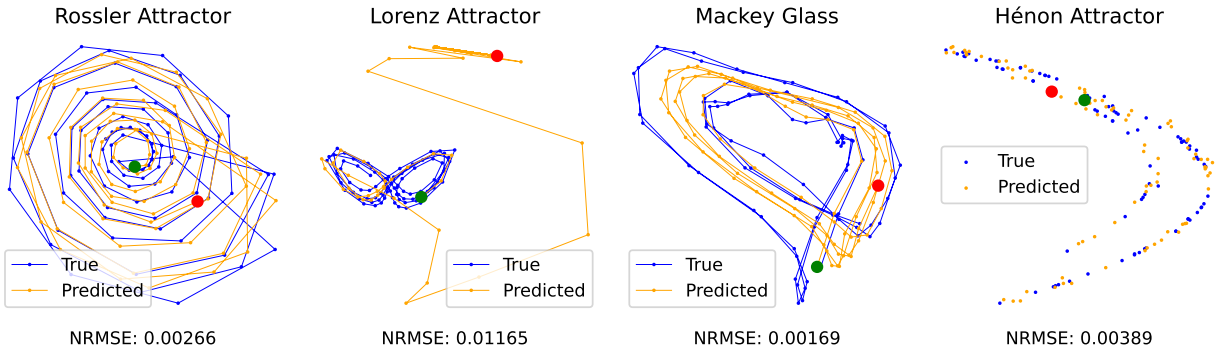

Figure 6: Predictions PCA feature conceptor z-space size $M = 100$. Green and red dot indicating the starting and ending point of the prediction.

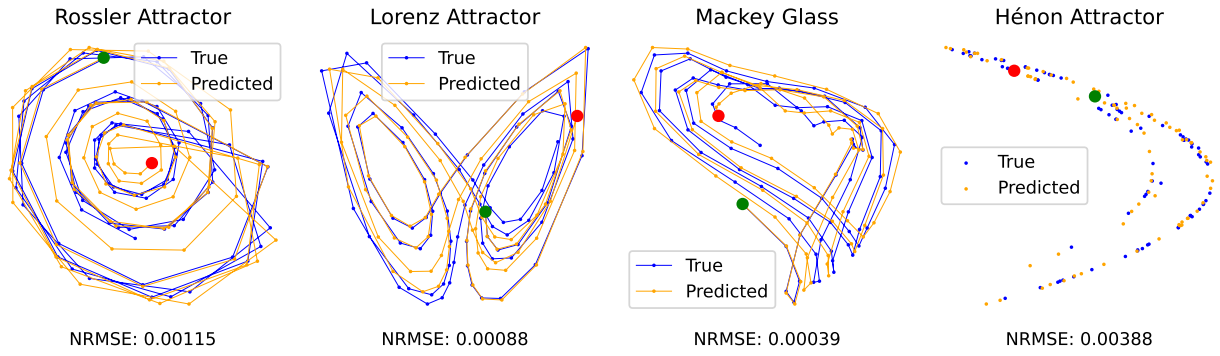

Figure 7: Predictions PCA feature conceptor z-space size $M = 1000$. The green and red dot indicate the starting and ending points of the prediction.

Let us now turn to the stability. Table 4 presents the results of the stability. As can be observed, for small values of M, stability is not guaranteed. Hence, without adding noise, the models would (sometimes) derail and never return to the loaded attractor. For $M = 375$ the RFC is stable. For values of $M \geq 500$ both PCA FC and RFC are stable. For $M = 750$ and $M = 1000$ the amount of noise that can be added to a PCA FC is similar to that of the matrix conceptors and higher than the amount that can be added to the RFC architecture.

M	100	125	187	250	312	375	500	750	1000
Random Feature Conceptor	ns	ns	ns	ns	ns	0.00005	0.001	0.005	0.005
PCA Feature Conceptor	ns	ns	ns	ns	ns	ns	0.001	0.01	0.01
Matrix (250)	0.01								
Matrix (500)	0.01								

Table 4: Maximum amount of noise to be added. Where 'ns' indicates that without adding noise, this setting was not stable.

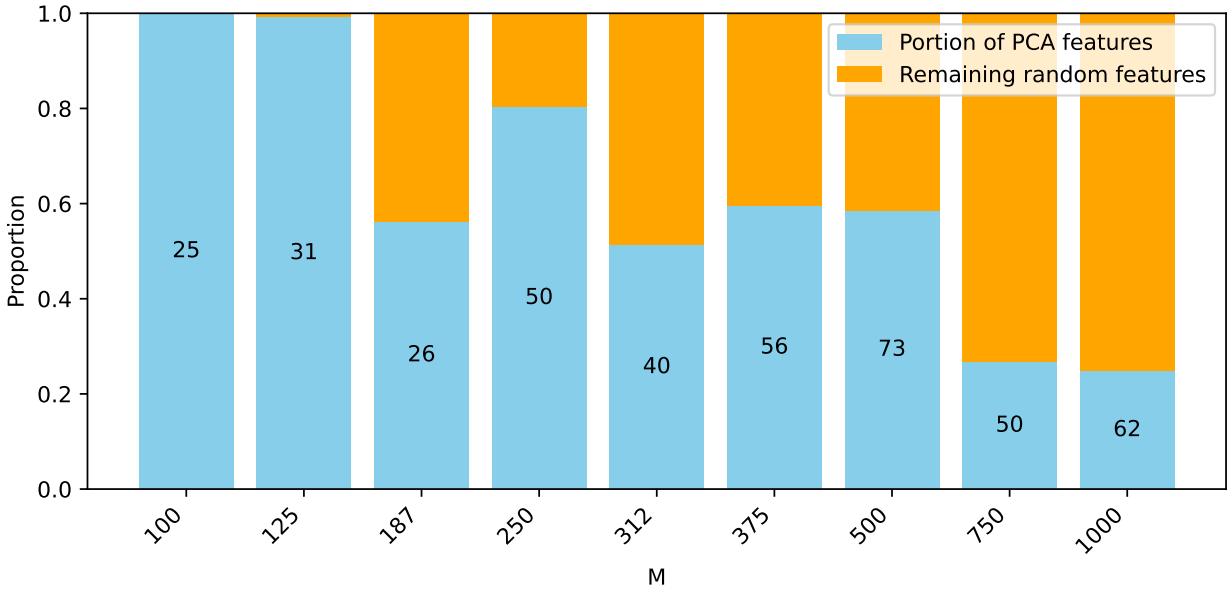

Figure 8: The optimal number of PCA features (blue) in relation to the number of random features (orange). In text the optimal number of PCA features *per* pattern is displayed. Hence, since this research deals with loading four patterns, if the number of PCA features *per* pattern equals 25, the total amount of PCA features equals 100.

The optimal number of PCA features is optimized. Figure 8 presents the number of features as a portion of the total number of features. Note that the remaining features are drawn from a N-dimensional symmetric normal distribution and scaled to have a norm equal to one. Also, note that the number displayed inside the bar is the number of features taken from *one* pattern. Hence in this research, the total amount of PCA features is this number multiplied with four. The *total* number of PCA features is represented by the *hight* of the blue bar.

To get a better understanding of the size of the conceptor weights, Figure 9 displays the conceptor weights for the four patterns. Note that in this example, the number of PCA components was set equal to 250. The first 250 conceptor weights are related to the Rossler Attractor, the second 250 to Lorenz Attractor, 500-749 are based on the Mackey Glass and the remaining 250 are connected to the Hénon attractor.

It can be observed that the weights of the conceptor vector of the pattern that corresponds to the PCA features are only visibly above zero for up until about the first 50 of the PCA features. The total size of the PCA weights also decreases as the PCA features become less important and for the last PCA feature, the weights are close to zero.

Figure 10 presents the conceptor weights for randomly drawn features. If Figure 10 is compared with Figure 9 weights are generally close to one. If the average sum of all weights c_i equals 0.93.

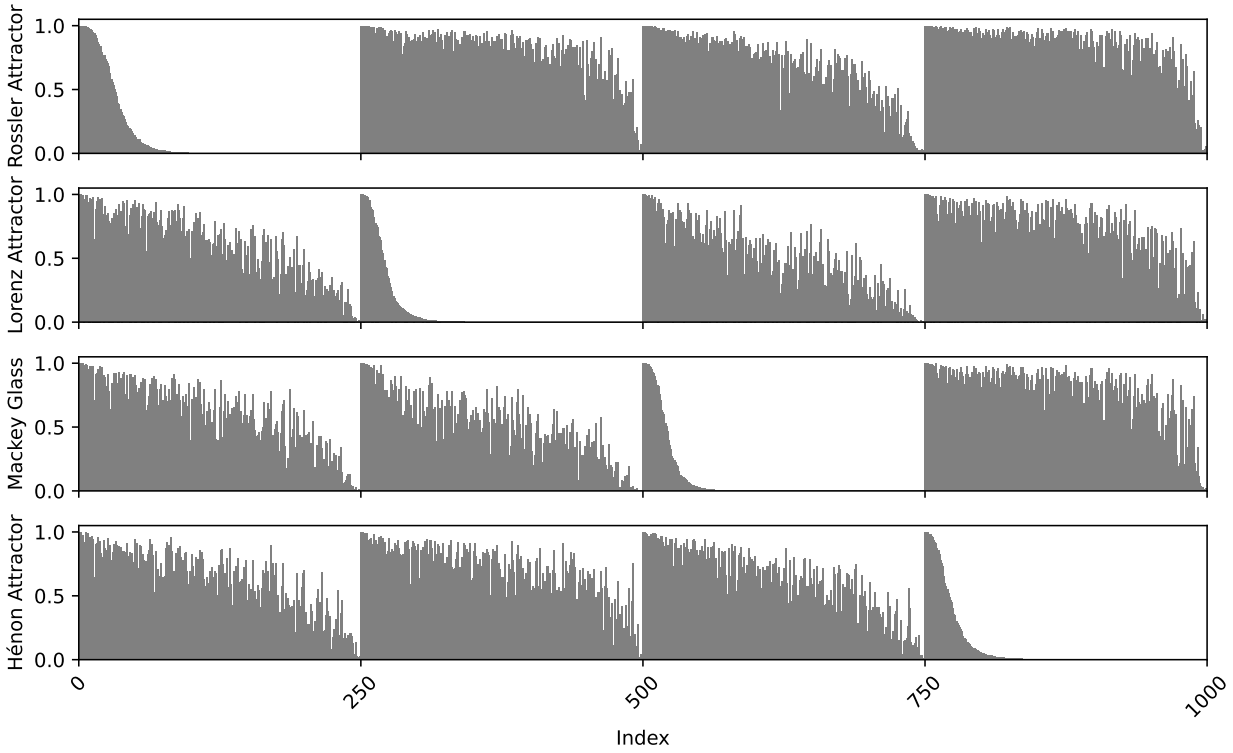

Figure 9: Conceptor weights per pattern. The expansion map size is $M = 1000$, and the number of PCA components equals the reservoir size of 250. Hence, all feature vectors f_i are PCA components. The first 250 components are taken from the Rossler Attractor, the second 250 from the Lorenz Attractor, the next from Mackey Glass and the last 250 from the Hénon Attractor.

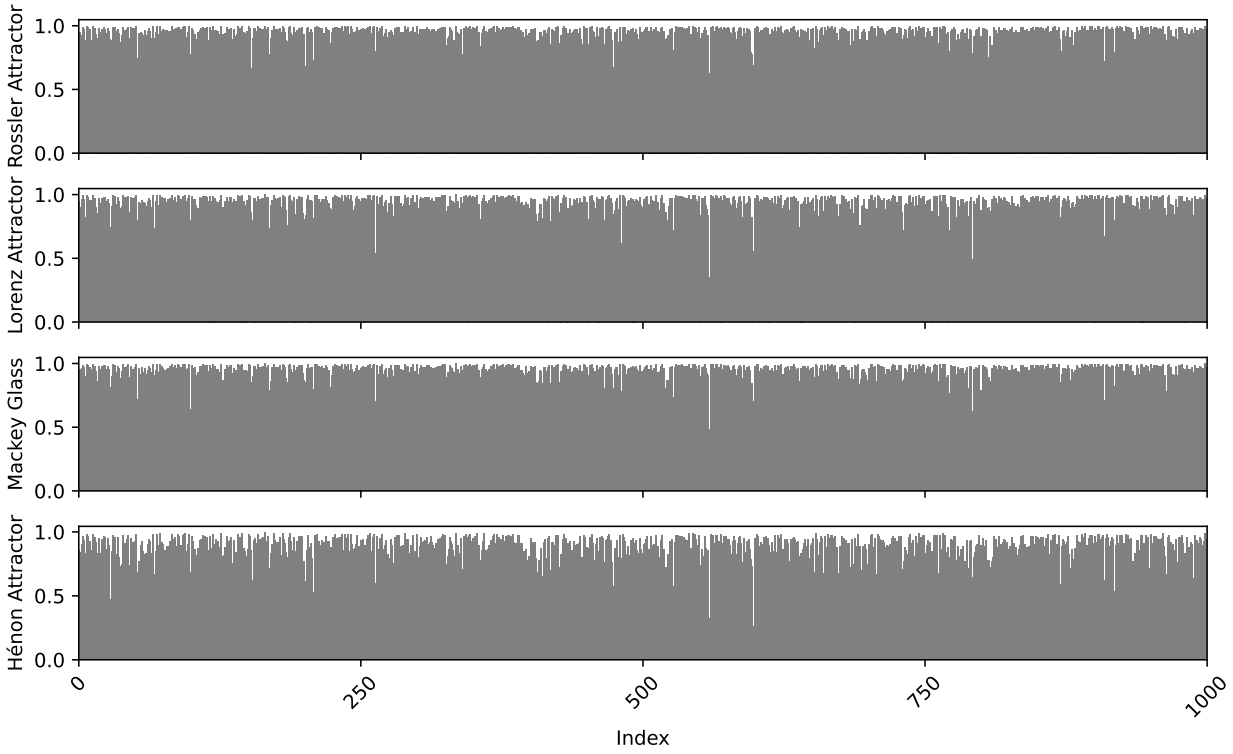

Figure 10: Conceptor weights per pattern. The expansion map size is $M = 1000$, All features are drawn from a random normal distribution and scaled to a uniform size.

6 Discussion

This section will interpret the results presented in Section 5. First, the results presented in Figure 5 will be discussed, together with the optimizer. Secondly, the stability of the models will be discussed and lastly, the optimal number of PCA components will be discussed.

Let us first turn our attention to the prediction accuracy, as presented in Figure 5. Observe that PCA FC obtains lower NRMSE than RFCs in every setting. For $M = 500$ the PCA FC very clearly outperforms a matrix conceptor with a reservoir size of 250 on prediction accuracy and is similar to the matrix conceptor with $N = 500$. The addition of PCA features seems to have the highest effect for small values of M. This might be explained by the fact that for smaller values of M, there are fewer features to work with. Hence, features that point into the dominant directions of the reservoir state space associated with a pattern will help control the reservoir more effectively.

The stability of PCA feature conceptors is generally similar to or better than that of RFCs. However, larger expansion sizes ($M = 3N$) are necessary to obtain the level of stability that a matrix conceptor offers. Notice that with increasing M, the stability increases as well. For larger M there are more directions in which the reservoir state space can be controlled, hence more stability is as expected.

However, stability is also closely related to the aperture α. Small values of aperture result in more stable models, as for smaller apertures the conceptor approaches the all zero-map. Hence, the level of stability is adjustable and there is a trade-off to be made with prediction accuracy. Since this research optimized for prediction error and only tested for stability afterwards, it is hard to conclude which model is the most stable. What can be concluded is that for large M the RFC FC has a better accuracy with similar stability.

Next to the aperture, signal and reservoir noise ($\sigma_{\text{signal}}, \sigma_{\text{reservoir}}$) influence both the stability and the prediction accuracy. Adding very little noise might lead to very accurate models that are not stable. We have to note that stability is also not uniformly defined. By using 5 repeated experiments and visual inspection we do give insight into the stability of the different models.

The optimal number of PCA features seems to lie around 50, a fifth of the reservoir size. If one inspects Figure 9, it is clear to see that each pattern only uses about 50 of its "own" PCA features. Hence, both the parameter optimization and the plot in Figure 9 suggest that, for large enough M the optimal number of PCA features lies around 50. The maximum number of PCA features needs to be optimized, but a good starting point would probably be $\frac{N}{5}$.

The optimization of parameters is very important to asses the performance of these methods. For smaller M, it may happen that *sometimes* a pattern is not captured, see Figure 6. This "random" behaviour can make it challenging to optimize parameters, as the lowest optimization NRMSE may be (and will be) based on chance. It may happen that with sub-optimal parameters, all four patterns fit fairly well 5 times in a row, resulting in a low average NRMSE.

To coop with this, Figure 5 presents the mean of 30 *new* experiments using the hyperparameters that were found during optimization. This always leads to a higher average NRMSE than the one found during optimization. It might happen that the optimizer was better able to optimize the parameters for PCA FC, than for RFC. To check for this the optimal hyperparameters for the RFC were used by a PCA FC with (at maximum) 50 PCA features. This always resulted in (far) better results than without the PCA features.

7 Conclusion

This thesis aimed to expand the knowledge concerning Random Feature Conceptors and improve task performance. We focused on finding an alternative way to construct expansion map F', as introduced by Jaeger (2017). We found a better-performing method of constructing map F'. In this method, some of the features f_i of the map F' are based on the principal components of the reservoir state space related to a given pattern. This systematic PCA-based method improved all performance measures.

The performance assessment was based on the architecture's ability to control a recurrent neural network, also known as a reservoir. Four 2-dimensional representations of chaotic patterns were loaded into the reservoir using a training sequence of 3000 timesteps. The first performance measure is the prediction accuracy of the network on the next 84 steps. Secondly, the stability of the network is of interest. To assess the stability, some random noise was added to the network. The maximum amount of noise that can be added while the network can still trace back to behave as the original network indicates its degree of stability.

PCA feature conceptors outperformed random feature conceptors in every way. The optimal number of PCA features seems to be around one-fifth of the total number of neurons in the reservoir. However, this parameter needs to be optimized for each application.

The improvement of the Feature Conceptor architecture allows future users of this architecture to either use smaller expansion mappings F without losing performance or improve performance by including some PCA features within their map. As PCA is a very cheap operation for modern computers, and since no real changes need to be done to the architecture, even systems that are already in place can be improved easily.

These findings also open up new research directions. In Jaeger (2017) it is shown that the matrix conceptor architecture can be loaded with patterns in an incremental manner. It has yet to be shown how to do this with feature conceptors. An especially interesting idea would be to also expand the map F' with only a *few* features. However, this might not be easily done due to the input simulation matrix.

Feature conceptors can be applied in every situation where matrix conceptors can be applied, with a potential fraction of the storage cost. Hence, in each setting where matrix conceptors are used, it would be interesting to research how PCA feature-based conceptors would perform.

Lastly, Conceptor control does require both storage and computational effort. It might be interesting to research what would happen when one applies conceptor control not every timestep. This has obvious computational advantages, but what would happen performance-wise?

References

De Jong, J. (2021). *Controlling recurrent neural networks by diagonal conceptors* (Master's thesis, University of Groningen). Retrieved from https://fse.studenttheses.ub.rug.nl/24863/

Gast, R., Faion, P., Standvoss, K., Suckro, A., Lewis, B., & Pipa, G. (2017). Encoding and decoding dynamic sensory signals with recurrent neural networks: An application of conceptors to birdsongs. *bioRxiv*. Retrieved from https://www.biorxiv.org/content/early/2017/04/28/131052 doi: 10.1101/131052

He, X., & Jaeger, H. (2018). Overcoming catastrophic interference using conceptor-aided backpropagation. *International Conference on Learning Representations (ICLR 2018)*. Retrieved from https://openreview.net/forum?id=B1al7jg0b

Jaeger, H. (2014). *Conceptors: an easy introduction.* Retrieved from https://arxiv.org/abs/1406.2671

Jaeger, H. (2017). *Controlling recurrent neural networks by conceptors.* Retrieved from https://arxiv.org/abs/1403.3369

Karve, S., Ungar, L., & Sedoc, J. (2019). *Conceptor debiasing of word representations evaluated on WEAT.* Retrieved from https://arxiv.org/abs/1906.05993

Meyer zu Driehausen, F., Busche, R., Leugering, J., & Pipa, G. (2019). Bistable perception in conceptor networks. In I. V. Tetko, V. Kůrková, P. Karpov, & F. Theis (Eds.), *Artificial neural networks and machine learning – ICANN 2019: Workshop and special sessions* (pp. 24–34). Cham: Springer International Publishing.

Pals, R. (2024). *Towards an iterative approach for constructing diagonal conceptors for autonomous pattern generation* (Bachelor's thesis, University of Groningen). Retrieved from https://fse.studenttheses.ub.rug.nl/33677/

Postmus, J. (2024). *Steering large language models using conceptors: An alternative to point-based activation engineering* (Bachelor's thesis, University of Groningen). Retrieved from https://fse.studenttheses.ub.rug.nl/33973/

Reinhart, F., & Steil, J. (2010). A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, *19*, 27-46. doi: 10.1007/s12591-010-0067-x

Yildiz, I. B., Jaeger, H., & Kiebel, S. J. (2012). Re-visiting the echo state property. *Neural Networks*, *35*, 1-9. Retrieved from https://www.sciencedirect.com/science/article/pii/S0893608012001852 doi: https://doi.org/10.1016/j.neunet.2012.07.005

A Chaotic attractors

For the Rössler attractor, training time series were obtained from running simple Euler approximations of the following ODEs:

$$\begin{aligned}\dot{x} &= -(y+z)\\ \dot{y} &= x+ay\\ \dot{z} &= b+xz-cz,\end{aligned}$$

using parameters $a=b=0.2, c=8$. The evolution of this system was Euler approximated with stepsize $1/200$ and the resulting discrete time series was then subsampled by 150. The x and y coordinates were assembled in a 2-dimensional driving sequence, where each of the two channels was shifted/scaled to a range of $[0,1]$.

For the Lorenz attractor, the ODE

$$\begin{aligned}\dot{x} &= \sigma(y-x)\\ \dot{y} &= rx-y-xz\\ \dot{z} &= xy-bz\end{aligned}$$

with $\sigma=10, r=28, b=8/3$ was Euler-approximated with stepsize $1/200$ and subsequent subsampling by 15. The x and z coordinates were collected in a 2-dimensional driving sequence, again each channel normalized to a range of $[0,1]$.

The Mackey-Glass time series was obtained from the delay differential equation

$$\dot{x}(t) = \frac{\beta x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t)$$

with $\beta=0.2, n=10, \tau=17, \gamma=0.1$, a customary setting when this attractor is used in neural network demonstrations. An Euler approximation with stepsize $1/10$ was used. To obtain a 2-dimensional time series that could be fed to the reservoir through the same two input channels as the other attractor data, pairs $x(t), x(t-\tau)$ were combined into 2-dimensional vectors. Again, these two signals were normalized to the $[0,1]$ range.

The Hénon attractor is governed by the iterated map

$$\begin{aligned}x(n+1) &= y(n)+1-ax(n)\\ y(n+1) &= bx(n),\end{aligned}$$

where I used $a=1.4, b=0.3$. The two components were filed into a 2-dimensional time series $(x(n), y(n))'$ with no further subsampling, and again normalization to a range of $[0,1]$ in each component.

B Optimal Parameters

M	100	125	187	250	312	375	500	750	1000
$\sigma_{\text{reservoir}}$	$2.39 \cdot 10^{-4}$	$2.39 \cdot 10^{-4}$	$2.39 \cdot 10^{-4}$	$4.95 \cdot 10^{-4}$	$1.74 \cdot 10^{-4}$	$3.85 \cdot 10^{-4}$	$3.85 \cdot 10^{-4}$	$3.12 \cdot 10^{-4}$	$4.05 \cdot 10^{-4}$
σ_{signal}	$4.55 \cdot 10^{-3}$	$5.67 \cdot 10^{-3}$	$8.22 \cdot 10^{-3}$	$4.13 \cdot 10^{-3}$	$5.5 \cdot 10^{-3}$	$5.67 \cdot 10^{-3}$	$6.79 \cdot 10^{-3}$	$5.1 \cdot 10^{-3}$	$4.13 \cdot 10^{-3}$
k_{PCA}	25	31	26	50	40	56	73	50	62
α_{Rossler}	28.1	34.3	34.0	31.5	19.7	31.5	36.3	31.5	36.6
α_{Lorenz}	28.0	24.7	17.9	18.4	32.3	22.7	26.1	22.7	22.7
$\alpha_{\text{Mackey Glass}}$	15.9	17.5	5.03	21.7	12.3	14.3	9.18	14.3	14.3
α_{Henon}	14.6	13.0	11.5	11.8	7.37	14.3	7.32	14.3	14.3
$\beta_{W^{\text{out}}}$	0.0105	0.0165	0.0165	$8.44 \cdot 10^{-3}$	0.0112	$8.44 \cdot 10^{-3}$	$4.75 \cdot 10^{-3}$	$8.44 \cdot 10^{-3}$	0.0112
β_D	$8.01 \cdot 10^{-6}$	$8.79 \cdot 10^{-5}$	$2.22 \cdot 10^{-5}$	$5.63 \cdot 10^{-5}$	$1.42 \cdot 10^{-5}$	$1.67 \cdot 10^{-5}$	$9.39 \cdot 10^{-6}$	$1.67 \cdot 10^{-5}$	$6 \cdot 10^{-5}$
β_G	0.324	0.213	0.474	0.324	0.257	0.260	0.342	0.260	0.360
$\sigma_{W_{\text{in}}}$	1.25	1.25	1.40	1.50	1.40	1.25	1.30	1.25	1.50
$\rho(F'G)$	0.600	0.500	1.00	1.10	1.05	1.10	1.20	1.10	1.10
$\rho(W)$	1.10	1.15	1.35	1.10	0.900	1.25	1.20	1.25	1.20
b	0.2	0.45	0.35	0.3	0.45	0.4	0.4	0.35	0.45

Table 5: The optimal parameters for the PCA FC with various M and a fix reservoir size of $N = 250$

	100	125	187	250	312	375	500	750	1000
$\sigma_{\text{reservoir}}$	$2.66 \cdot 10^{-4}$	$3.22 \cdot 10^{-4}$	$2.39 \cdot 10^{-4}$	$3.64 \cdot 10^{-4}$	$1.74 \cdot 10^{-4}$	$4.37 \cdot 10^{-4}$	$2.15 \cdot 10^{-4}$	$2.39 \cdot 10^{-4}$	$2.95 \cdot 10^{-4}$
σ_{signal}	$6.86 \cdot 10^{-3}$	$5.61 \cdot 10^{-3}$	$8.22 \cdot 10^{-3}$	$5.61 \cdot 10^{-3}$	$5.5 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	$4.01 \cdot 10^{-3}$	$5.45 \cdot 10^{-3}$	$4.95 \cdot 10^{-3}$
α_{Rossler}	48.0	10.1	34.0	24.6	19.7	48.0	12.6	12.6	19.7
α_{Lorenz}	20.6	11.5	17.9	28.0	32.3	9.18	13.2	17.9	17.9
$\alpha_{\text{Mackey Glass}}$	17.7	39.8	5.03	12.3	12.3	30.0	11.8	18.4	18.4
α_{Henon}	9.22	3.77	11.5	18.0	7.37	11.5	4.72	11.5	11.5
$\beta_{W^{\text{out}}}$	0.0200	0.0206	0.0165	0.0112	0.0112	0.0112	$4.75 \cdot 10^{-3}$	$6.33 \cdot 10^{-3}$	0.0112
β_D	$3.71 \cdot 10^{-5}$	$6 \cdot 10^{-5}$	$2.22 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$	$1.42 \cdot 10^{-5}$	$6 \cdot 10^{-5}$	$6.01 \cdot 10^{-6}$	$1.9 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$
β_G	0.423	0.257	0.474	0.353	0.257	0.436	0.152	0.250	0.208
$\sigma_{W_{\text{in}}}$	1.45	1.50	1.40	1.50	1.40	1.50	1.30	1.40	1.30
$\rho(F'G)$	1.00	0.950	1.00	1.05	1.05	1.00	1.05	1.15	1.15
$\rho(W)$	1.20	1.05	1.35	1.20	0.900	1.20	1.10	1.05	1.20
b	0.25	0.3	0.5	0.3	0.4	0.35	0.4	0.45	0.4

Table 6: The optimal parameters for the RFC with various M and a fix reservoir size of $N = 250$

N	250	500
$\sigma_{\text{reservoir}}$	$8.65 \cdot 10^{-5}$	$3.95 \cdot 10^{-4}$
σ_{signal}	$4.05 \cdot 10^{-4}$	$4.9 \cdot 10^{-3}$
α_{Rossler}	150	1050
α_{Lorenz}	25.6	255
$\alpha_{\text{Mackey Glass}}$	41.0	475
α_{Henon}	60.0	410
$\beta_{W^{\text{out}}}$	$7.29 \cdot 10^{-3}$	$7.14 \cdot 10^{-3}$
β_W	$1 \cdot 10^{-4}$	$5.31 \cdot 10^{-5}$
$\sigma_{W_{\text{in}}}$	1.15	1.25
$\rho(W)$	0.75	0.55
b	0.3	0.55

Table 7: Optimal parameters found for the matrix Conceptors with reservoir size $N = 250$ and $N = 500$