university of groningen / faculty of science and engineering / artificial intelligence

Towards an iterative approach for constructing diagonal conceptors for autonomous pattern generation

Bachelor's Project Thesis

Remco Pals, S4087216, r.pals.1@student.rug.nl

Supervisor: Dr Herbert Jaeger

Abstract

Conceptors are a neuro-computational mechanism that allow for controlling the reservoir dynamics of recurrent neural networks for a variety of tasks. Diagonal conceptors are a lighter version of these conceptors that trade some of the accuracy and robustness for an increase in ease of computation and reduction in storage required. Steps were made towards an iterative approach to compute them, such that on-the-fly learning would be possible. It was found that the established training procedure is not suited for iteration due to unstable conceptor weights and consequently erratic reservoir dynamics. Additionally, diagonal conceptors were applied to a feedback network to produce temporal patterns autonomously. It was argued that this alternative setup is potentially more suitable for an iterative approach, and its limitations were discussed.

1 Introduction

As humans, we are capable of storing and remembering temporal patterns for long times. Many old people are still able to remember the lullabies that their parents used to sing them before bed, songs from primary schools or Psalms from church. We are capable of reproducing these songs without needing a constant source of input, but rather can start singing without being prompted. This ability to store temporal patterns so that they can be recalled later is a property of the brain which has been challenging to reproduce using neural networks. Traditional feed-forward neural networks have unidirectional weights connecting their layers from the input towards the output layer. Due to this architecture, any information that is fed into the network in one time-step is immediately forgotten in the next one. Regardless of this lack of memory, recent discoveries, such as transformers, have made it so that these networks can handle temporal data well and have been leading the race when it comes to text and image generation (Vaswani et al., 2017). For neuroscientists the performance of these architectures is not as interesting as for the rest of the world. Within these architectures the connects are feed-forward only, unlike the recurrent connections in our brains. That means, there is a need for another kind of architecture, which mimics our brains more closely. Therefore, recurrent neural networks (RNNs) have been the go-to tool to study human and animal brains and cognition instead (Yang & Molano-Mazón, 2021; Barak, 2017).

Within any RNN, the internal state of a network at time-step t is fed back through recurrent weights into the network at the next time-step $t+1$. Due to this, some of the information about what the network has encountered before remains and affects the state of the network in the following time-steps. This gives the network a short-term memory, making it suited for tasks that use temporal data. A disadvantage that comes as a consequence of this type of architecture is the increase in difficulty of training the recurrent weights in the networks (Lukoševičius & Jaeger, 2009), due to well-documented problems such as the vanishing gradient problem or the exploding gradient problem (Pascanu et al., 2013). Methods like backpropagation through time (Werbos, 1990) and real-time recurrent learning (Williams & Zipser, 1989) have been successfully employed to train RNNs, but none seem as simple as leaving the weights in the recurrent layer randomly initialised and only training the layer which connects this recurrent layer to the output. This architecture and training method have been discovered multiple times independently, but has been commonly adopted under the name of reservoir computing. The standard network architecture in reservoir computing is very simple, as it merely consists of an input layer, a recurrent layer known as the reservoir and an output layer. A schematic representation of this architecture is visible in Fig. 1.

Depending on the kind of input fed into the reservoir via the input weights, the reservoir neurons will attain some level of activation in each time-step. The collection of activation of all neurons in the reservoir at a certain time-step t is called the reservoir state at that time-step t. Important here is the observation that different inputs will lead to different levels and distributions of activation of the neurons in the reservoir. Thus, feeding different inputs into the network will lead to different reservoir states. Because of this, training the output weights is simple and is normally done using a linear regression algorithm that can directly map the reservoir states to the desired outputs (Lukoševičius & Jaeger, 2009). Once the output weights have been computed, the RNN is able to generate the desired output whenever it is fed with input.

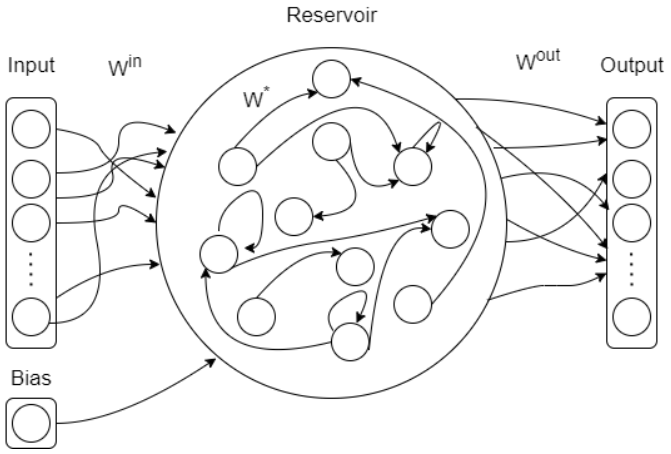

Figure 1: Architecture of a basic RNN used for reservoir computing, consisting of an input layer connected to the reservoir through input weights, a recurrently connected layer (the reservoir) and an output layer connected to the reservoir through output weights.

A neuro-computational mechanism that takes advantage of the differences in state-spaces covered by the reservoir, given different inputs is conceptors. Conceptors are neural filters, capable of restricting the state-space of a reservoir so that it can be associated with a specific behaviour. They were first introduced in a report by Jaeger (2014). In this report, conceptors were defined as very robust filters that are capable of restricting the state space of the reservoir, such that, for example, coherent output can be generated autonomously or dynamical patterns can be learned incrementally. Computationally speaking, these conceptors are full matrices that scale quadratically with the size of the reservoir, such that the state space of the reservoir can be controlled at runtime. A more efficient version of these conceptors, named 'Diagonal Conceptors' were introduced by De Jong in his 2021 Master's thesis. Diagonal conceptors trade some of the robustness and accuracy of the original conceptors for an improvement in storage and computational efficiency, as instead of full matrices they can be stored as simple vectors. De Jong (2021) developed a training method for the use of diagonal conceptors in recurrent neural networks that when correctly followed can be used to store and accurately reproduce multiple temporal patterns autonomously.

However, this training method needs periods of observing and storing before the computation of the conceptor, reservoir and output weights. The conceptors are computed first and only afterwards the reservoir and output weights are updated so that coherent output can be generated based on how the conceptors restrict the reservoir dynamics. A more biologically plausible network would be capable of iterative, on-the-fly learning without a need for storage and computation phases. This Bachelor's thesis investigated why simply iterating the training approach does not improve the performance, but rather breaks it. Additionally, a mathematical description of how diagonal conceptors can be used to autonomously generate temporal patterns was provided, and most importantly, steps towards a more iterative approach were made.

2 Mathematical background

This section provides a mathematical description of the training procedure of diagonal conceptors. It largely follows the work in De Jong (2021), which in turn almost directly follows from the original conceptor report by Jaeger (2014). For the mathematical description, certain notations will stay consistent throughout the report. A matrix is always denoted by a capital letter, where the transpose of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^T$. The element of the matrix $\mathbf{A}$ at row i and column j is denoted by $\mathbf{A}_{ij}$. A vector is denoted by a bold letter.

The goal of using diagonal conceptors in the bounds of this thesis paper was to store temporal patterns in an RNN so that they later on can be retrieved autonomously. Thus, the network should be able to generate patterns it has seen before without having to feed it any input at the time of pattern generation. That is, when putting conceptor $\mathbf{c}^j$ associated with the pattern $\mathbf{p}^j$ in place and running the RNN autonomously, the pattern $\mathbf{p}^j$ should be generated.

To store the set of discrete-time patterns $P = \{\mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^p\}$, where pattern $\mathbf{p}^j$ at time n^j is given by $\mathbf{p}^j(n) \in \mathbb{R}^M$, set up an RNN with M input neurons as well as bias neurons, a reservoir with N simple *tanh* neurons, and M output neurons. As the goal is to store patterns and generate them autonomously, the number of input and output neurons is equal. Furthermore, create random matrices $\mathbf{W}^{in} \in \mathbb{R}^{N \times M}$ and $\mathbf{W} \in \mathbb{R}^{N \times N}$ to store the input weights and reservoir weights in. Via the input neurons, the reservoir is driven with pattern $\mathbf{p}^j$. The reservoir neurons are sparsely connected via the reservoir weights. The activation of reservoir neuron i at time-step t, while driven by pattern $\mathbf{p}^j$, is given by $\mathbf{x}_i^j(t) \in (-1,1)$ and is called the *neuron state*. The state of the reservoir while driven by pattern $\mathbf{p}^j$ at time-step t is given by the vector containing all neuron states, called the *reservoir state*, and is given by $\mathbf{x}^j(t) \in (-1,1)^N$. The output produced by the network is read out via the output weights, and the output at time-step t while driven by pattern $\mathbf{p}^j$ is given by $\mathbf{y}^j(t)$. During the training procedure of the diagonal conceptors, all the reservoir states observed while driving the reservoir with the patterns, are stored to be used for re-computation of the conceptor weights, internal weights and output weights in a matrix called the *reservoir matrix*.

Storing patterns in a reservoir without conceptors

To store the patterns in a reservoir, let $\mathbf{W}^{in}$, $\mathbf{W}^*$, and $\mathbf{b}$ be random fixed matrices. These fixed matrices will not be adjusted during the training procedure. The reservoir state update equation for a normal RNN without a conceptor, driven with input is given by:

$$\mathbf{x}^j(t+1) = \tanh(\mathbf{W}^* \mathbf{x}^j(t) + \mathbf{W}^{in} \mathbf{p}^j(t) + \mathbf{b}) \tag{1}$$

The matrix $\mathbf{W}^{out}$ will be computed during the training procedure. Then, the output equation is given by:

$$\mathbf{y}^j(t) = \mathbf{W}^{out} \mathbf{x}^j(t) \tag{2}$$

When storing patterns in a reservoir, the goal is to have some superposition of the patterns stored in the reservoir that can be used to generate the patterns autonomously. For that to be possible, it is necessary to be able to observe the same reservoir dynamics that are associated with the target output, without having to provide input. Thus, it was aimed to find the reservoir weights that approximate the reservoir dynamics, that were observed when the reservoir was driven with input, without having to provide that input. Whereas usually, in reservoir computing the recurrent weights are left unchanged and the

input puts the reservoir into the desired state, in this case the reservoir weights were recomputed to achieve the desired reservoir states without needing that input. This was obtained using the following approximation:

$$\mathbf{x}^j(t+1) = \tanh(\mathbf{W}^*\mathbf{x}^j(t) + \mathbf{W}^{in}\mathbf{p}^j(t) + \mathbf{b}) \approx \tanh(\mathbf{W}\mathbf{x}^j(t) + \mathbf{b}) \tag{3}$$

Where $\mathbf{W}$ are the recomputed reservoir weights and $\mathbf{x}^j(t)$ is the reservoir state at time t. The computation of $\mathbf{W}$ was done using ridge regression and can be estimated by squaring the loss over all patterns and all time-steps, leading to the equation:

$$\mathbf{W} = \arg\min_{\tilde{\mathbf{W}}} \sum_j \sum_t ||\mathbf{W}^*\mathbf{x}^j(t) + \mathbf{W}^{in}\mathbf{p}^j(t) - \tilde{\mathbf{W}}\mathbf{x}^j(t)||^2 \tag{4}$$

Additionally, the computation of $\mathbf{W}^{out}$, such that the patterns can be estimated based on the reservoir dynamics associated with those patterns, was also done using ridge regression, to get the following estimation of the output equation:

$$\mathbf{y}^j(t) = \mathbf{W}^{out}\mathbf{x}^j(t) \approx \mathbf{p}^j(t) \tag{5}$$

Minimising the squared loss over all patterns and all time-steps lead to the following formula for the computation of $\mathbf{W}^{out}$:

$$\mathbf{W}^{out} = \arg\min_{\tilde{\mathbf{W}}_{out}} \sum_j \sum_t ||\mathbf{p}^j(t) - \tilde{\mathbf{W}}_{out}\mathbf{x}^j(t)||^2 \tag{6}$$

Now the reservoir can be autonomously updated by simply multiplying the reservoir state with the recomputed reservoir weights using the following formula:

$$\mathbf{x}^j(t+1) = \mathbf{W}\mathbf{x}^j(t) + \mathbf{b} \tag{7}$$

The output then can be computed by multiplying the reservoir state with the computed output weights:

$$\mathbf{y}^j(t) = \mathbf{W}^{out}\mathbf{x}^j \tag{8}$$

When this procedure is correctly followed and the reservoir weights and output weights are computed, there exists a superposition of the patterns in the reservoir. However, if the reservoir runs autonomously, the behaviour would be unpredictable, as the reservoir does not know which pattern to engage in. It is unable to lock its internal state to be representative of a single pattern. Instead, all observed behaviour is estimated at the same time, making the reservoir dynamics unpredictable. This is where conceptors offer a solution. They can be seen as filters, that constrain the dynamics of the reservoir into a direction that is associated with a specific pattern that is stored within the reservoir, such that the reservoir can be driven without input and the specific pattern can be produced.

Generating patterns from a reservoir with diagonal conceptors

Conceptors were first introduced by Jaeger (2014). When patterns are stored in a reservoir as described above, the reservoir could be driven autonomously and would produce unpredictable output, as it is unaware of which pattern it should engage in. Conceptor filters are $N \times N$ matrices, where N is the size of the reservoir, which, when applied in the update step of the reservoir, are capable of compressing the state cloud of the reservoir to different orthogonal directions so that it is uniquely representative of the pattern that the conceptor is associated with. Thus, for each pattern that is stored

in the reservoir, a full $N \times N$ matrix needs to be calculated and stored so that the pattern can be generated again later. Note that conceptors are capable of more than just retrieving stored patterns from a reservoir, but for the scope of this research other applications were left out. Using a conceptor, the reservoir update equation now takes two steps. First the reservoir is updated based in the input and the previous reservoir state. Then, the reservoir state is multiplied by the conceptor so it is constrained into an unique direction. For any input $\mathbf{p}$, the reservoir update equations with conceptor $\mathbf{C}$ in place would look like this:

$$\begin{aligned} \mathbf{r}(t+1) &= \tanh(\mathbf{W}\mathbf{z}(t) + \mathbf{W}^{in}\mathbf{p} + \mathbf{b}) \\ \mathbf{z}(t+1) &= \mathbf{C}\mathbf{r}\mathbf{+}\mathbf{1}(t) \end{aligned} \tag{9}$$

As traditional conceptors are positive semidefinite matrices, they are able to compress the ellipsoid covered by the reservoir state to different orthagonal axes for each different behavior. Diagonal conceptors instead only compress state to different directions along the same axes, which leads to a loss in performance in favour for an increase in spatial and computational efficiency. Regardless, as shown in De Jong (2021), diagonal conceptors can be used to successfully store multiple temporal patterns in small neural networks and reproduce them with good accuracy.

If the procedure for storing patterns in a reservoir described above is correctly followed, there exists a superposition of patterns in the reservoir. However, if the reservoir was now run without a conceptor in place, the output would become unpredictable. In the introduction it was explained that when an RNN is fed with input, that input will give some parts of the reservoir a higher level of activation than others. If the recomputed reservoir weights are used to autonomously drive the reservoir without a conceptor matrix, the reservoir will try to engage in all the activation subspaces of all the patterns stored in it at the same time. From this, no single specific pattern can be computed, and instead the behaviour of the network becomes unpredictable. By individually scaling each neuron in the reservoir during the reservoir update step, diagonal conceptors restrict which parts of the reservoir are active. The neurons that have a higher observed average activation when the reservoir is driven with input will get scaled with a value closer to 1 to leave them unchanged, while the ones that have a lower average activation will get scaled with a value closer to 0 to be suppressed. The reservoir dynamics then become representative of the dynamics that were observed in the training phase for a specific pattern. Using the output weights this pattern can be generated. This means that, for each pattern stored in the reservoir, there should be a conceptor filter that is capable of suppressing the parts of the reservoir that are not associated with that pattern, while leaving the associated parts unchanged. The diagonal conceptors scale each neuron in the reservoir individually, such that the reservoir can be seen as a two layered network, where the first layer is projected via the conceptor weights into the second one before being projected back into the first via the reservoir weights. The architecture of this setup is visualised in Fig. 2.

This setup translates to the following update equations for driving the reservoir autonomously with the diagonal conceptor in place:

$$\begin{aligned} \mathbf{r}(t+1) &= \tanh(\mathbf{W}\mathbf{z}(t) + \mathbf{W}^{in}\mathbf{p}^{j} + \mathbf{b}) \\ \mathbf{z}(t+1) &= \mathbf{C}^{j}\mathbf{r}(t+1) \end{aligned} \tag{10}$$

Where $\mathbf{C}^{j}$ is the conceptor associated with pattern $\mathbf{p}^{j}$. The output can be computed with the following formula:

$$y^{j}(t) = \mathbf{W}^{out}\mathbf{z}(t) \tag{11}$$

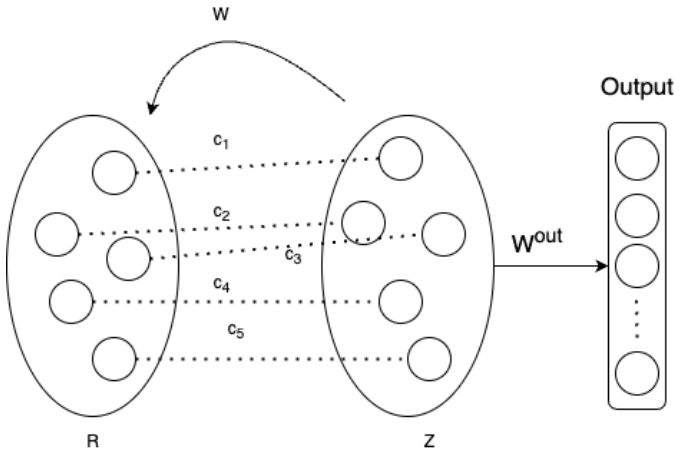

Figure 2: Architecture of an RNN that can be used to autonomously generate patterns using diagonal conceptors. Here, $\mathbf{z}$ is the reservoir state at time t, which is updated by multiplying it with the recomputed reservoir weights $\mathbf{W}$ to get $\mathbf{R}$, which is then filtered through the conceptor weights $\mathbf{c}$, such that only the relevant parts of the reservoir are kept to get $\mathbf{z}$ at a time $t+1$, from which the output can be computed.

An ideal diagonal conceptor would scale the neurons in the reservoir such that the neurons associated with the pattern of the conceptor are left unchanged, while suppressing the neurons associated with the other patterns. Therefore, in the training procedure for the conceptors, the neurons that have a higher average activation will receive a higher conceptor weight than the neurons that have a lower average activity. In order to regulate the balance between leaving states unchanged and suppressing others, the *aperture* parameter is introduced. The term aperture is derived from the aperture of a camera lens, which regulates how much light can fall through it. Similarly, the aperture of the diagonal conceptor regulates the strength of the conceptor weights by controlling how much of the reservoir is being suppressed. For pattern j, the loss function for a diagonal conceptor is given by:

$$\ell(\mathbf{C}^j) = \mathbb{E}[||\mathbf{z}^j - \mathbf{C}^j\mathbf{z}^j||^2] + (\alpha^j)^{-2}||\mathbf{C}^j||^2 \tag{12}$$

Where α is the aperture, $\mathbf{z}^j$ reservoir states associated with pattern j and $\mathbf{C}^j$ the diagonal conceptor of pattern j.

Here the balancing act that conceptors have to do can clearly be seen:
The left term of the right-hand side of the equation, $\mathbb{E}[||\mathbf{z}^j - C^j\mathbf{z}^j||]^2$, is zero when the diagonal conceptor $\mathbf{C}^j$ is equal to the identity matrix. What this means is that all the neurons of the reservoir are left unchanged, both the ones associated with pattern j and the ones not associated with it. At that point, the behaviour of the RNN would be unpredictable and exactly the same as if there were no conceptor in the loop at all.
The right term, $(\alpha^j)^{-2}||\mathbf{C}^j||^2$ is equal to zero when the diagonal conceptor is equal to the null matrix. When the diagonal conceptor is equal to the null matrix, all the neurons are scaled to 0 and all neuron states are suppressed, both the ones associated with the pattern of the conceptor and the ones that are not associated with that pattern. The output of the RNN would then be constant and become zero.

From observing that $\mathbf{C}^j$ in Equation 12 is a diagonal matrix, rewriting the equation element-wise and solving it for c_i^j yielded the following formula for the conceptor weight associated with neuron i with pattern j:

$$c_i^j = \frac{\mathbb{E}[z_i^j]^2}{\mathbb{E}[z_i^j]^2 + (\alpha^j)^{-2}} \tag{13}$$

The conceptor weights of the diagonal conceptors are the average squared activation of the reservoir over the same activation plus the aperture. This means that neurons with an average higher activation will have a higher conceptor weight and neurons will lower activation will have lower conceptor weights. That way the important parts of the reservoir are left unchanged, while the other parts are suppressed. In the same fashion, for an aperture value close to 0, the individual conceptor weights will go towards 0 as well. That way, the right term on the right-hand side of Equation 12 becomes 0, and the entire reservoir is suppressed. For a higher aperture c_i^j also increases, where its limit is 1 when the aperture moves towards infinity. The aperture is chosen for each pattern individually and heuristically.

In order to store patterns into a reservoir, so that they can be retrieved using diagonal conceptors, the following training scheme was proposed by De Jong:
Let $P = \mathbf{p}^1, \mathbf{p}^2, .., \mathbf{p}^p$ be patterns where $\mathbf{p}^j \in \mathbb{R}^m$ for all j. Set up an RNN with the following:

- input weights $\mathbf{W}^{in} \in \mathbb{R}^{M \times N}$, random values drawn from the standard normal distribution and scaled appropriately.
- reservoir weights $\mathbf{W}^* \in \mathbb{R}^{N \times N}$, a sparse matrix with random values drawn from the standard normal distribution and scaled appropriately.
- bias weights $\mathbf{b} \in \mathbb{R}^N$, random values drawn from the standard normal distribution and scaled appropriately.
- initial random diagonal conceptors for each pattern with each conceptor $\mathbf{c}^{*j} \in \mathbb{R}^N$, random values drawn from the uniform distribution $[0, 1]$.

Now, the training procedure can be divided in three steps; the washout, the adaptation and the learning phases. After these three phases, the patterns can be generated autonomously.

Washout

For pattern $\mathbf{p}^j$, drive the reservoir for $n_{washout}$ time-steps. Let the initial state of the reservoir each time be the null-vector. The reservoir states are updated using the following formula:

$$\begin{aligned} \mathbf{r}^j(n+1) &= \tanh(W^* \mathbf{z}^j(n) + W^{in} \mathbf{p}^j(n+1) + \mathbf{b}) \\ \mathbf{z}^j(n+1) &= \mathbf{c}^{*j} \circ \mathbf{r}(n+1) \end{aligned} \tag{14}$$

Where $\circ$ is the element-wise multiplication operator. During the washout period, the goal is to let the reservoir forget about the initial state of the reservoir and instead enter a state that is associated with the pattern it is being driven with. A washout phase important in this training procedure, such that the initial state is forgotten and instead the reservoir is entirely dependent on the input that it currently is being fed. Therefore, none of the reservoir states in this period are neither stored nor used for re-computation of the conceptor, reservoir and output weights.

Adaptation

After the washout period, the reservoir now covers the state-space associated with the pattern it is driven with. Then, drive the reservoir for another n_{adapt} time-steps. During this period, collect the reservoir state at each time-step such that reservoir state collection matrix $\mathbf{Z}$ can be created using:

$$\mathbf{Z}^j = [\mathbf{z}^j(n_{washout}+1), \mathbf{z}^j(n_{washout}+2), ..., \mathbf{z}^j(n_{washout}+n_{adapt})] \tag{15}$$

Using this state collection matrix, the conceptor weights c_i^j can be computed element-wise:

$$c_i^j = \frac{[z_i^j]^2}{[z_i^j]^2 + (\alpha^j)^{-2}} \tag{16}$$

Where z_i^j is the i-th element of $\mathbf{Z}^j$ and a^j is the aperture for pattern $\mathbf{p}^j$. In experiments, $\mathbf{c}^j$ is instead computed in a vectorised manner, using:

$$\mathbf{c}^j = [\mathbf{z}^j \circ \mathbf{z}^j] \oslash ([\mathbf{z}^j \circ \mathbf{z}^j] + (\alpha^j)^{-2}) \tag{17}$$

Where $\circ$ is the Hadamard product, or the element-wise multiplication operator and $\oslash$ is the element-wise division operator. The conceptor weight for each individual neuron thus is nothing more than the average squared activation of that neuron that was observed during the adaptation period, where the aperture serves as a regularization term.

Learning

Replace the random initial conceptor with the recomputed one and drive the reservoir with the associated pattern for another n_{learn} time-steps. The new reservoir state update equation now looks like this:

$$\begin{aligned} \mathbf{r}^j(n+1) &= \tanh(\mathbf{W}^* \mathbf{z}^j(n) + \mathbf{W}^{in} \mathbf{p}^j(n+1) + \mathbf{b}) \\ \mathbf{z}^j(n+1) &= \mathbf{c}^j \circ \mathbf{r}(n+1) \end{aligned} \tag{18}$$

During this period, collect the following data to recompute the reservoir and output weights:

- $\mathbf{r}^j(n)$, to create $\mathbf{X}^j$, where $\mathbf{X}^j_{*,n} = \mathbf{r}^j(n_0+n)$
- $\mathbf{z}^j(n-1)$, to create $\tilde{\mathbf{Z}}^j$, where $(\tilde{\mathbf{Z}}^j)_{*,n} = \mathbf{z}^j(n_0+n-1)$
- $\mathbf{p}^j(n)$, to create $\mathbf{P}^j$, where $\mathbf{P}^j_{*,n} = \mathbf{p}^j(n_0+n)$
- $\mathbf{W}\mathbf{z}^j(n) + \mathbf{W}^{in}\mathbf{p}^j(n)$, to create $\mathbf{W}^j_{target}$, where $(\mathbf{W}^j_{target})_{*,n} = \mathbf{W}^j\mathbf{z}^j(n_0+n) + \mathbf{W}^{in}\mathbf{p}^j(n_0+n)$

Where $n_0 = n_{washout} + n_{adapt}$ and $(.)_{*,n}$ denotes the n^{th} column of a matrix.

Computing W and $\mathbf{W}^{out}$

Once the washout, adaptation and learning phases have been performed for each pattern that needed to be stored, the output and reservoir weights were computed using ridge regression. For that, all the collected matrices for each pattern $p^1, p^2, ..., p^p$ were concatenated, such that $\mathbf{X} = [\mathbf{X}^1|\mathbf{X}^2|...|\mathbf{X}^p], \tilde{\mathbf{Z}} = [\tilde{\mathbf{Z}}^1|\tilde{\mathbf{Z}}^2|...|\tilde{\mathbf{Z}}^3], \mathbf{P} = [\mathbf{P}^1|\mathbf{P}^2|...|\mathbf{P}^p], \mathbf{W}_{target} = [\mathbf{W}^1_{target}|\mathbf{W}^2_{target}|...|\mathbf{W}^p_{target}]$.
The formulae used for ridge regression are:

$$\begin{aligned} \mathbf{W} &= (\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + \rho^{\mathbf{W}} I)^{-1}\tilde{\mathbf{Z}}\mathbf{W}^T_{target} \\ \mathbf{W}^{out} &= (\mathbf{X}\mathbf{X}^T + \rho^{\mathbf{W}^{out}} I)^{-1}\mathbf{X}\mathbf{P}^T \end{aligned} \tag{19}$$

Where $\rho^{\mathbf{W}}$ and $\rho^{\mathbf{W}^{out}}$ are the regularisation constants for $\mathbf{W}$ and $\mathbf{W}^{out}$ respectively, and I is the identity matrix. This is the final part of the training procedure as proposed by De Jong (2021). After the training procedure, pattern p^j can be generated by putting diagonal conceptor $\mathbf{c}^j$ in the update equation and updating the reservoir using:

$$\begin{aligned} \mathbf{r}(t+1) &= \tanh(\mathbf{W}\mathbf{z}(n)\mathbf{b}) \\ \mathbf{z}(t+1) &= \mathbf{c}^j \circ \mathbf{r}(n) \end{aligned} \tag{20}$$

And the output can be read using:

$$y^j(t) = \mathbf{W}^{out}\mathbf{z}(t) \tag{21}$$

The training procedure was capable of successfully training a recurrent neural network to store and generate simple patterns. However, there could be room for improvement. The conceptor weights were computed based on an initial random network that was being driven with input. Then, the reservoir and output weights were estimated based on the reservoir states of this random recurrent network with the recomputed conceptor in place. When the patterns were then generated autonomously, the observed dynamics were just an approximation of the behaviour that was seen when the random reservoir was driven with input. Thus, these dynamics might be slightly different, because of which the conceptors are not entirely correctly suppressing the right parts of the reservoir, nor might the output weights be able to compute the correct output.

One of the goals of this research was to investigate whether simply iterating the training procedure could potentially improve this mismatch described above. In case iteration does work, that also means potentially the training procedure can be altered, so that the network can show on-the-fly learning. This also entails that instead of having to feed the network entire patterns and collecting data about the reservoir state for each time-step before recomputing the reservoir and output weights, after each time-step instead the weights are slightly altered, such that the output is closer to what was desired. That hints at a great increase in memory efficiency of the training procedure, and a potential increase in computational efficiency, as well as that the training procedure could be cut off early in case the output patterns are a good enough approximation of the target patterns.

Within this research paper, three experiments are presented. The first experiment simply iterated the aforementioned training procedure and assessed why the results did not improve. The second experiment instead only repeated a part of the training procedure, and again assessed why the results did not improve. Finally, the third experiment proposed an alternative network setup with a new

training procedure. This setup provided similar approximations for the periodic patterns as the one employed in De Jong (2021), though at a lower computational cost and potentially more suited for an iterative learning procedure.

Target patterns

For the series of experiments in this thesis it was decided to use four periodic patterns, all of the same length, for simplicity. A function f is periodic if $f(t+k) = f(k)$ for some constant k. A pattern can either be *integer-periodic* or *irrational-periodic*. If a pattern $\mathbf{p}$ is integer-periodic, there exists some integer number k for which $\mathbf{p}(t+k) = \mathbf{p}(k)$. If a pattern is irrational-periodic, the pattern is sampled with a sample interval of $\mathbf{d} \in \mathbb{Z}_{>0}$ from a continuous time signal with a period of $k \in \mathbb{R}_{>0}$, with the ratio $\frac{d}{k}$ being irrational. An integer-periodic pattern with period k will occupy k points in the state-space as the same k points are visited each period. An irrational-periodic pattern will cover a more complex regio in the state-space. While analysing the reservoir dynamics of the networks during simulation and seeing the points they cover in the state-space, one can potentially learn valuable information about whether the dynamics still approximate the desired behaviour.

It was shown that storing and retrieving simple periodic patterns is trivial using regular conceptors (Jaeger, 2014). Using diagonal conceptors, a single reservoir of 100 neurons could be used to store and generate four periodic patterns, two sine waves with an irrational period of ≈ 8.93 and ≈ 9.93 and two integer-periodic patterns with a period of 5 that share the same set of points but for a single point per period (De Jong, 2021).

Due to RNNs working in discrete time-steps and using floating point numbers with limited accuracy, perfectly estimating time sequences with irrational time periods is impossible. However, they can still be estimated, and hence showing that it can be done using diagonal conceptors was a suitable indication of their strength. The time sequences or target patterns that were used for this research are shown in Fig. 3. Similar to De Jong (2021), two sine waves with irrational period lengths of ≈ 8.93 and ≈ 9.93 and two rational-period patterns with a period of 5 were used. For all for patterns 5000 points were generated and used for experimentation.

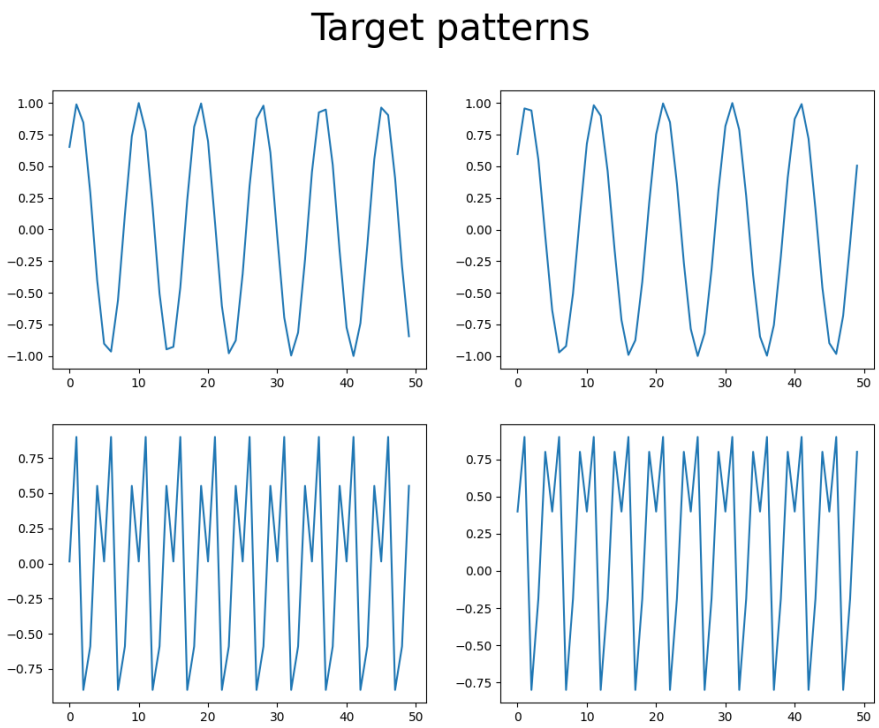

Figure 3: The four different target patterns

To determine whether the patterns were successfully stored, and can be autonomously retrieved, some sort of measurement of accuracy was needed. For this the Normalized Root Mean Square Error (NRMSE) was used. The NRMSE operates the same as the Root Mean Square Error, except it is normalised by the variance of the target pattern. Let p_{target} be a target pattern and p_{obs} be the observed pattern as generated by the network for length K, then the NRMSE can be computed using:

$$NRMSE(p_{obs}, p_{target}) = \sqrt{\frac{\frac{1}{K}\sum_{n=1}^{K}(p_{obs}(n) - p_{target}(n))^2}{\frac{1}{K}\sum_{n=1}^{K}(p_{target})^2}} \tag{22}$$

Where K is the amount of time-steps that the NRMSE is calculated over. In order to find the best NRMSE, the generated patterns were shifted such that the phases of the two patterns aligned. This was necessary, as generating the patterns was done from a random associated reservoir state, due to which the phases of the target and produced pattern were most likely not aligned.

Furthermore, for the two patterns with irrational periods, it is said that a 'good' estimation has been made if the NRMSE < 0.05. Since an irrational period is impossible to perfectly estimate using limited-accuracy floating point numbers, no matter how well the estimation was done, over time the NRMSE will increase. Hence, the accuracy for these patterns was constrained to be calculated over the first 1000 time-steps. For the other two patterns the target was to get the NRMSE < 0.0005. Since these were rational-period patterns, the reservoir should be able to perfectly estimate the patterns and hence the NRMSE was instead calculated over the available 5000 points. Only when the value for the NRMSE is that low, it is clear that the reservoir was able to distinguish between the two patterns.

3 Experiment 1

This experiment provided insight into iterating the training procedure. One iteration of the training process was defined as going through the aforementioned Washout, Adaptation and Learning periods for all four patterns needed to be stored, and the consequent recomputation of the reservoir weights, conceptor weights and output weights. Since the periodic patterns used in my thesis closely resembled the periodic patterns in De Jong (2021), my setup of the experiments was also inspired by De Jong (2021). In addition to the results presented here, experimentation was done with alternative ways of computing the conceptor weights and even alternative network structures. The first results presented in the next section were from an experiment where an RNN was setup with a conceptor as shown in Fig. 2, with parameters initialised as following:

- W^{in} scaling: 0.7
- W^{star} scaling: 2
- **b** scaling: 0.2
- ρ^{W}: 0.001
- $\rho^{W^{out}}$: 0.001
- $\mathbf{a}^{j}$: [7, 7, 6, 6]
- Reservoir size: 200

The parameter choice for this experiment was made heuristically, with small variations in individual parameters still leading to relatively good accuracy in the initial iteration. It was observed that the random initialisation of the initial reservoir played a very important role in successful storage and reproduction of the patterns, and for some initial setups it was impossible to generate accurate output. One important consequence from the parameter choice being made heuristically, was that they very likely were sub-optimal, because of which the results also were.

Since the overall goal of the experiments was to see if an improvement in performance measured by NRMSE could be achieved through iteration, during each iteration the patterns were graphed together with the observed NRMSE. Furthermore, extra attention was paid to the observed reservoir dynamics, as consistent reservoir dynamics are needed to compute accurate output. For periodic patterns, the aim was that the reservoir neurons show same-periodic behaviour as the patterns that are being produced at runtime.

Results

Running the training procedure for a single iteration should lead to results similar to what was shown by De Jong (2021). This means generating accurate approximations (NRMSE ≤ 0.05) and stable reservoir dynamics. Furthermore, to get insight into the reservoir dynamics, the activity of randomly selected neurons for each pattern were plotted alongside the singular values of the reservoir state collection matrix in Figure 4.

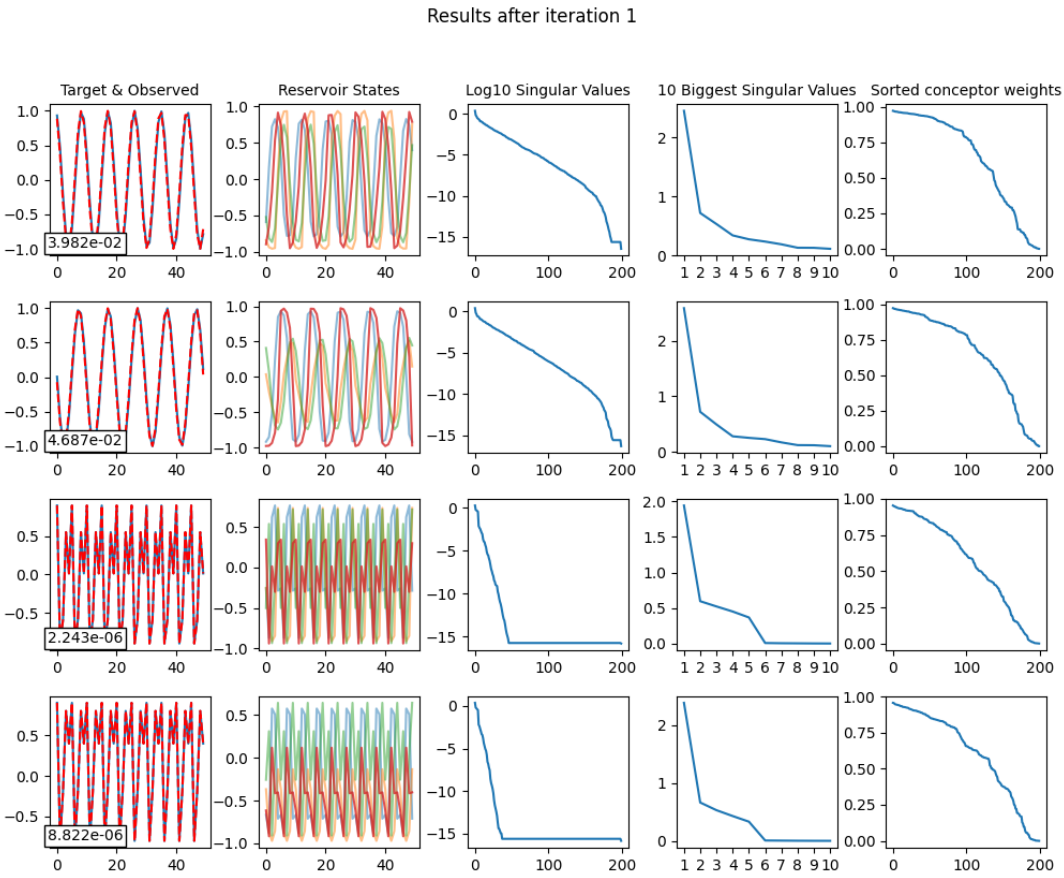

Figure 4: Each row shows the results for each of the target patterns. In the first column the target patterns are the blue lines and the generated ones are the dotted red line. The second column displays the reservoir dynamics of randomly selected neurons for each pattern. The third column shows the log_{10} of the singular values of the reservoir collection matrix (explanation below). The fourth column shows the 10 biggest singular values of the same matrix for each pattern. The fifth column shows the conceptor weights sorted in descending order. This gave information on how much of the reservoir was suppressed to produce each pattern.

From Fig. 4 it was observed that all patterns were accurately estimated, with a NRMSE smaller than 0.05 for all patterns. Furthermore, for each pattern the reservoir dynamics appeared stable and of the same periods as the target patterns.

Let X^j be the reservoir state collection matrix associated with pattern p^j. Let $R^j = \frac{X^j X^{j^T}}{n}$ be the correlation matrix of reservoir state, associated with pattern p^j, where n is the length of the pattern. Then $R^j = U\Sigma U^T$ is the singular value decomposition (SVD) of R^j. The singular values were given by the diagonal of Σ, while the principal components are given by U. In the third column of Fig. 4 it was demonstrated that for the irrational-time period patterns the principal components of R^j spanned all of R^N, whereas for the five-period patterns the principal components were non-zero in only five directions. Still, from the fourth column, it was visible that a large contribution to the singular values was concentrated on just a few principal components for all patterns, while most of the variance in the other directions was negligible, implying that for each pattern the network dynamics X are

concentrated in a low-dimensional subspace. From the fifth column it was possible to roughly estimate how much of the reservoir was suppressed. For each pattern the distribution of conceptor weights appeared similar, with values spanning all the way from 1 to 0.

These results supported that, by following the training procedure a single time, one can create a setup where, using conceptors, an RNN can be used to autonomously generate patterns. The results after completing this first iteration were taken as the baseline, and the target of this research was to see whether, by iterating the procedure, the results could be improved upon. Fig. 5 shows the results after repeating the training process a second time.

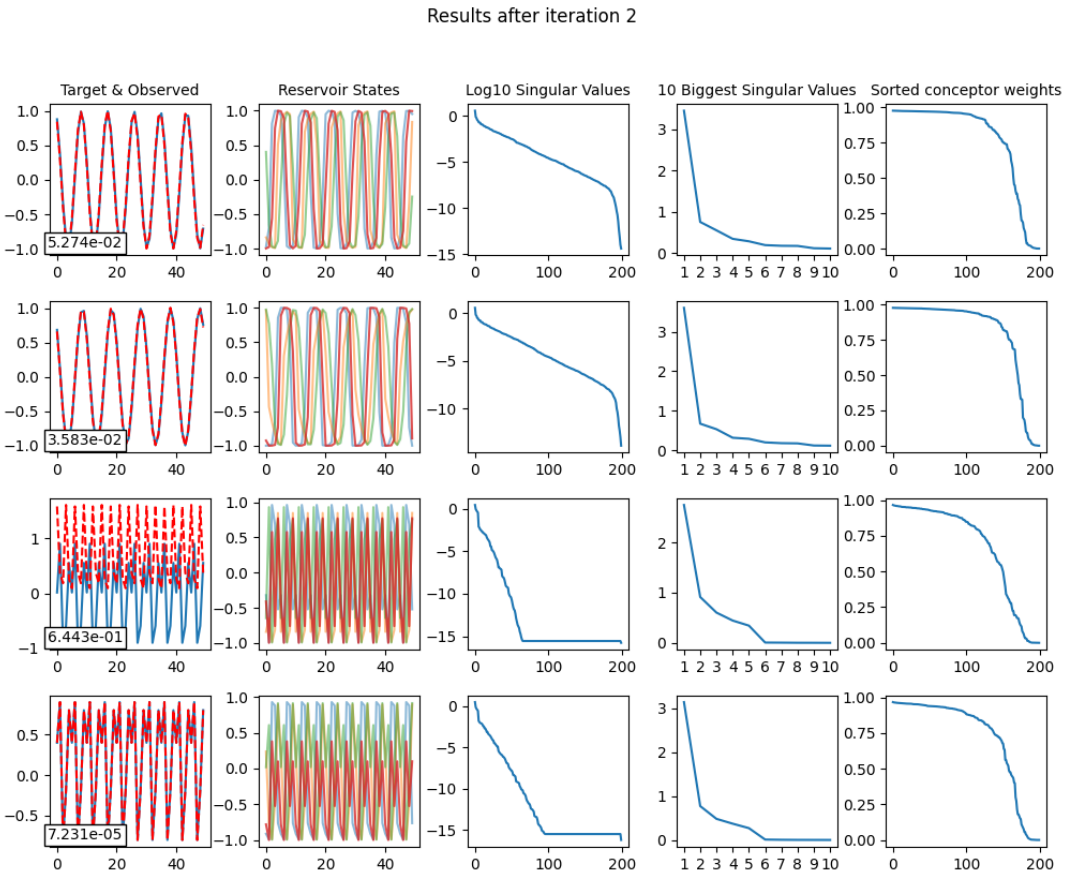

Figure 5: Results after iterating the training procedure twice. For an elaborate explanation of how to interpret this graph, see the description of Fig. 4.

In Fig. 5 the results after two iterations of the training procedure are visualised. From the first column it was observed that three of the patterns were still accurately estimated, but one of the five-period patterns' approximations was completely wrong. However, the reservoir dynamics for all periodic patterns still appeared to be of the same period as the patterns they estimated. Looking at column three and four, for the irrational-period patterns the principal components of R^j spanned all of R^N, while the principal components of the five-period patterns were only non-zero in five directions as the reservoir periodically visited the same five states. From the fifth column it was concluded that the distribution of conceptor weights has changed: the weights converged towards 1 and appear less

evenly distributed between 0 and 1. Compared to the first iteration it was clear that less of the reservoir is being suppressed and thus the reservoir dynamics were less restricted.

Repeating the iteration procedure multiple times seemed to further worsen the results.

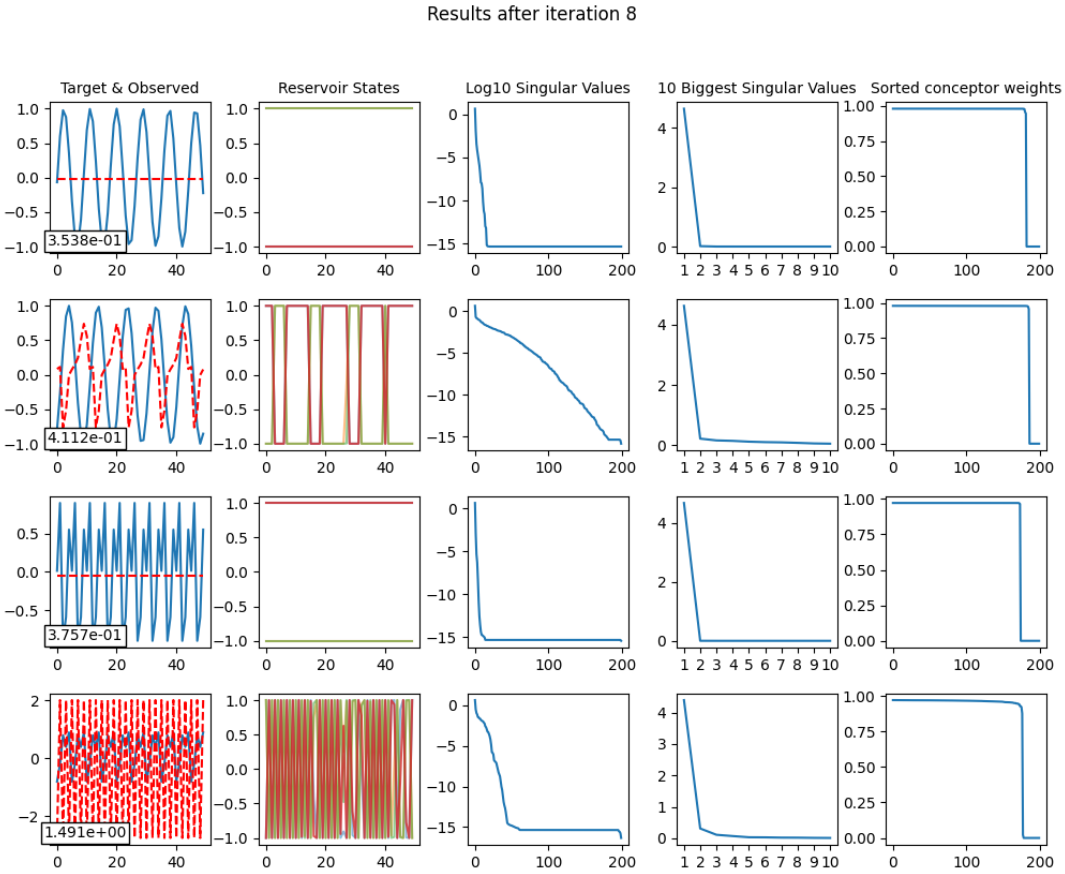

Figure 6: Results after iterating the training procedure eight times. For an elaborate explanation of how to interpret this graph, see the description of Fig. 4.

From Fig. 6 it became evident that after even more iterations the results only seemed to get worse. Based on simple visual inspection, I observed that the produced patterns did not represent the target patterns at all. Instead, the output appeared constant, binary or showing characteristics of the other patterns that were stored in the reservoir. Furthermore, the reservoir dynamics stopped showing the same periodic behaviour as the target patterns, instead they became constant or binary. The conceptor weights appeared binary: they were very close 1, or 0. For each pattern there was such an overwhelming amount of conceptor weights that were very close to 1, that hardly any part of the reservoir was suppressed.

Discussion

From the graphs in Figs 4, 5 and 6 shown in the previous section it was evident that simply iterating the training procedure did not improve the approximations. Rather than actually improving the potential mismatch between the conceptor weights and reservoir weights, the mismatch seemed

to increase, with the conceptor weights having been unable to suppress non-associated parts of the reservoir for each pattern. The reservoir dynamics lost their periodic behaviour and the majority of conceptor weights converged towards the values 1 and 0 for the aperture used in the simulation. During this testing phase, more experiments were done with different parameter settings, where similar behaviour was observed. For example, decreasing the aperture lead to larger parts of the reservoir being suppressed, however the conceptor weights remained converging to either 0 or some upper limit ≤ 1. Consequently, for higher aperture choices less of the reservoir was suppressed with each iteration. For lower aperture choices instead each iteration more and more is suppressed.

A partial explanation of the increasingly erratic reservoir dynamics that were observed in this experiment, could potentially lie in the training procedure. Within the training procedure the reservoir weights were recomputed such that the dynamics of the randomly initialised initial weights fed with the input patterns could be accurately estimated. When done correctly, the dynamics of the recomputed reservoir that can be observed when updating the reservoir without input, should be the same as the dynamics of the initial random reservoir with input. In the next iteration of the training procedure these recomputed reservoir weights were fed with input again. Without any input, the recomputed reservoir weights were capable of autonomously re-generating the dynamics of the first iteration, from which output could be computed with the computed output weights. However, as they were not run autonomously in the next iteration, but were fed with input instead, the dynamics changed and deviated from the target. The addition of the input into the recomputed reservoir made it so that the observed dynamics changed. After the entire input sequence had been fed through the recomputed network, the reservoir weights were recomputed again, so that the reservoir could autonomously re-generate the now changed reservoir dynamics that were observed, and the output weights were recomputed such that, from these dynamics, the input could be estimated. Thus, in consequent iterations, the recomputation of reservoir weights was done to estimate changing behaviour, which lead to losses in accuracy as the dynamics further deviated from the target.

To investigate whether the interplay of recomputed reservoir weights fed by input and increasing conceptor weights indeed leads to the loss of accuracy, the following experiment only recomputed the reservoir weights in the first iteration. In the following iterations, the reservoir was run autonomously and the conceptor and output weights were recomputed on the then observed dynamics.

4 Experiment 2

The network for this experiment was setup in the same way as in experiment 1. However, the reservoir weights were only recomputed during the first iteration, and then left unchanged in the following iterations of the training procedure. The conceptor and output weights were still recomputed at every step. Since, after the first iteration, the observed behaviour was expected to be correctly re-generated by the recomputed reservoir weights, potentially not further changing them, while recomputing the conceptor and output weights based on the recomputed reservoir weights, could lead to an improvement in performance. With this experiment, examining if the conceptor weights continued to converge towards some upper or lower limit, regardless of whether the reservoir weights were not continuously recomputed, could provide evidence towards that the construction of the conceptor weights, as proposed by De Jong (2021), was not suited for a more iterative approach, and instead a more stable approach would be needed.

Results

For this experiment the parameter setup was identical to Experiment 1. To avoid redundancy, the results of the first iteration were excluded from this section. After the second iteration, during which only the conceptor and output weights were recomputed, the following results were obtained:

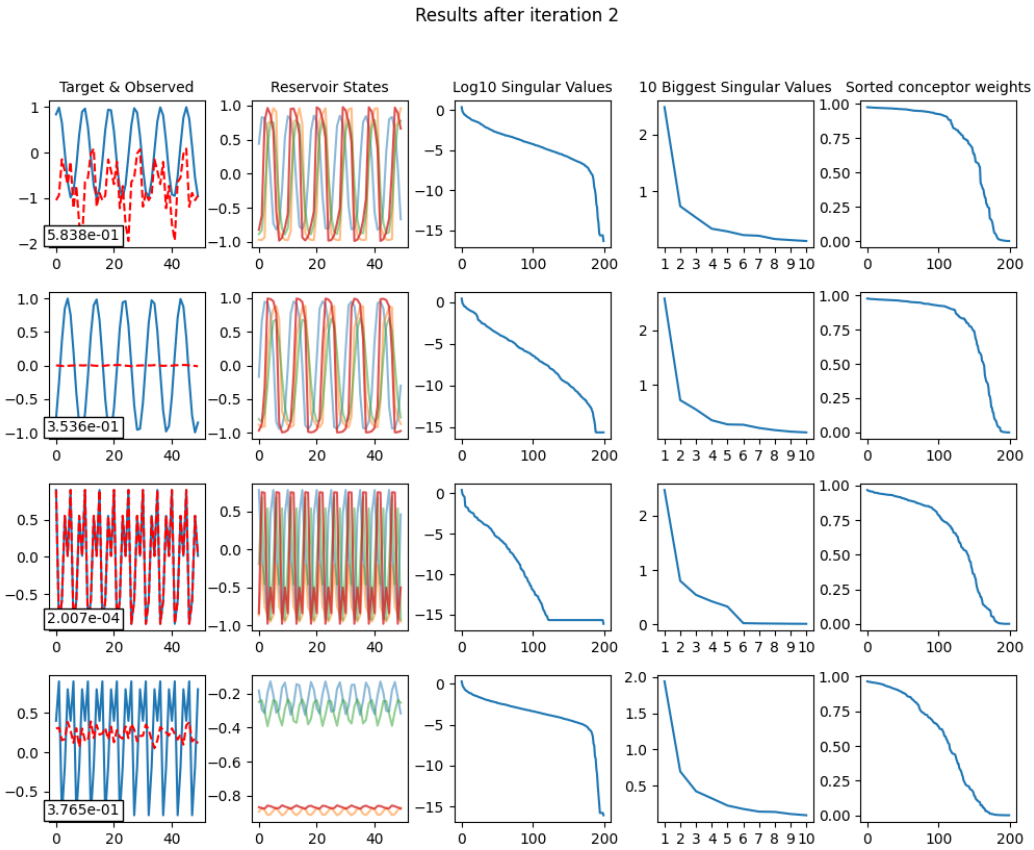

Figure 7: Results after recomputing the conceptor and output weights a second time. For an elaborate explanation of how to interpret this graph, see the description of Fig. 4.

As can be seen from Fig. 7, recomputing the conceptor and output weights an additional time, based on the reservoir dynamics observed from the recomputed reservoir weights, did not improve the approximations of the stored patterns. In fact, the approximations became less accurate than before. Even though for three of the patterns the reservoir dynamics appeared periodic with a similar period as the target pattern, for only one of them the approximation was acceptable, but still worsened compared to after the first iteration. For the fourth pattern it was observed that not only the reservoir dynamics no longer appeared to have the same period as the target patterns, they even seemed to switch between two states. Interestingly, from the fourth column I observed that the principal components were non-zero in more than five directions. This suggests that the conceptors failed to suppress the reservoir states to the point where they were limited to only five. In Fig. 8 the development of the conceptor weights over multiple iterations is shown.

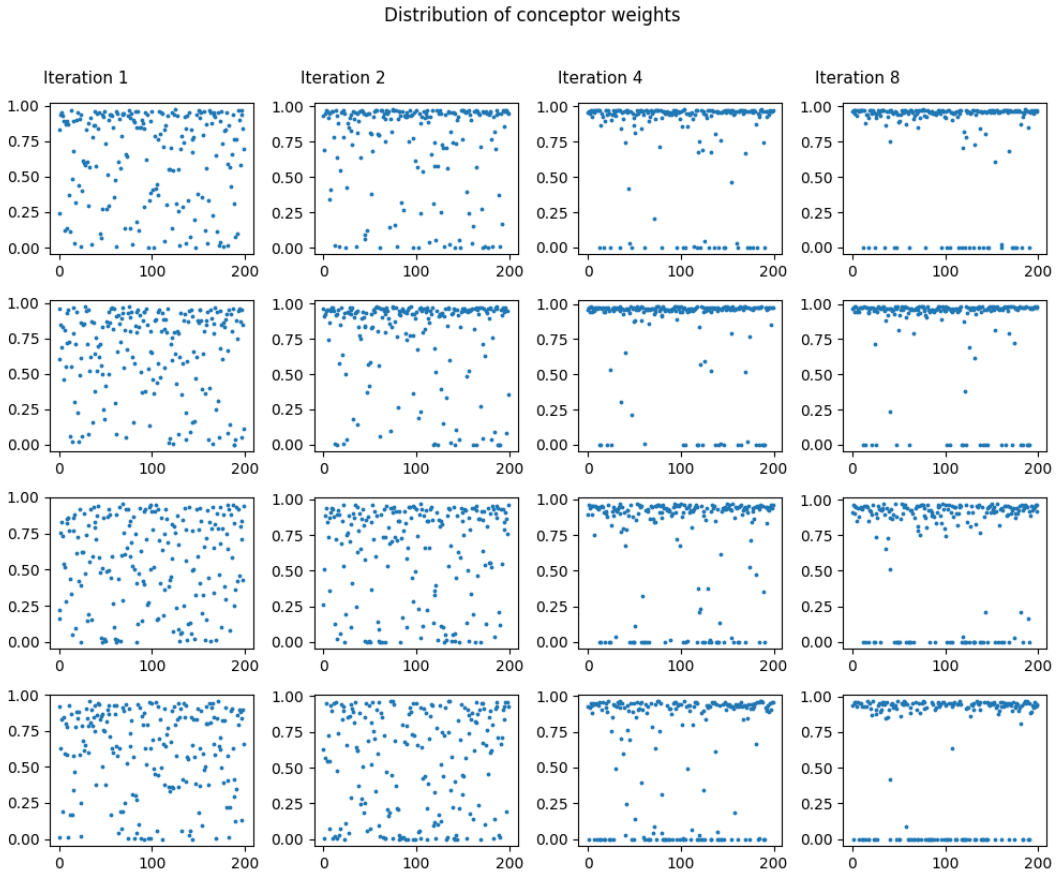

Figure 8: The progression of distribution of conceptor weights over multiple iterations. Each row represents the conceptor weights associated with a different pattern. For each pattern the 200 conceptor weights and their progression are plotted.

In Fig. 8 it was suggested that, even if the reservoir weights were only recomputed one time, recomputing the conceptor weights still caused them to converge towards binary values, from which almost no parts of the reservoir were suppressed, leading to where it could not be distinguished between the target patterns anymore. This convergence was slower than in Experiment 1, indicating that the convergence of conceptor weights shown there was caused both by the increase in reservoir activity

due to the recomputation of the reservoir weights and the recomputation of the conceptor weights.

Discussion

The results of Experiment 2 indicated that recomputing the conceptor weights and output weights multiple times did not lead to an increase in approximation accuracy compared to following the training procedure a single time. Recomputing the conceptor weights based on the recomputed reservoir weights led to a change in reservoir dynamics, after which the conceptors were not capable of correctly suppressing the irrelevant parts of the reservoir, while leaving the relevant parts unchanged, such that the output could be correctly computed.

The converging conceptor weights partially followed directly from the equation for calculating the conceptor weights:

$$c_i^j = \frac{[z_i^j]^2}{[z_i^j]^2 + (\alpha^j)^{-2}} \tag{23}$$

Where α again is the aperture. When the reservoir weights were not recomputed in each iteration, the only element changing the reservoir dynamics in each iteration was the conceptor weights themselves. One of two different behaviours could be observed for each conceptor weight. If, during the recomputation of the conceptor weight, the new weight was lower than before, the average activation of the associated neuron in the next iteration was lower too, as it was directly related to the conceptor weight. Due to this, the aperture had a relative bigger impact during recomputation of the conceptor weight as the activity of the neuron shrunk, and once again the conceptor weight was lower in the following iteration. Hence, the conceptor weight converged towards 0. If, during recomputation, the conceptor weight increased, the average activation of the related neuron also increased. The aperture then had less impact during recomputation and the recomputed conceptor weight increased again towards an upper limit which depended on the aperture. A very low aperture was observed to have a great impact, meaning that all conceptor weights eventually converged to 0 over multiple iterations. For higher apertures, the conceptor weights converged to 0 if the average activation of the related neuron was relatively low to begin with. Otherwise, the conceptor weights all converged to $\frac{1}{1+\alpha^{-2}}$. To support this finding, the experiment was also run with apertures of 2, 3, 4 and 5 for the sinus wave with a period of ≈ 8.93, the results of which are shown in Fig. 9.

From Fig. 9 it was observed that for an aperture of 2 all conceptor weights seemed to converge towards 0. For higher apertures, i.e. 3, 4 and 5, the weights converged to 0 if the weights were relatively low at the start. If they were higher to start with, they converged to $\frac{1}{1+\alpha^{-2}}$.

Using the results of Experiment 1 and 2, it was concluded that the proposed training procedure in De Jong (2021) was not suitable for a more iterative approach where on-the-fly learning was possible. How the conceptor weights were calculated caused them to become unstable and unable to control the reservoir dynamics, regardless of the aperture, and feeding the already recomputed reservoir weights with more input lead to chaotic behaviour or constant behaviour.

So far, for this network architecture and setup, no working alternative training procedure has been found that can be adapted to perform iterative computation. However, during experimentation, a promising, alternative network setup was found, that, with less computational cost than the training method presented in De Jong (2021), achieved a similar accuracy while using more robust concep-

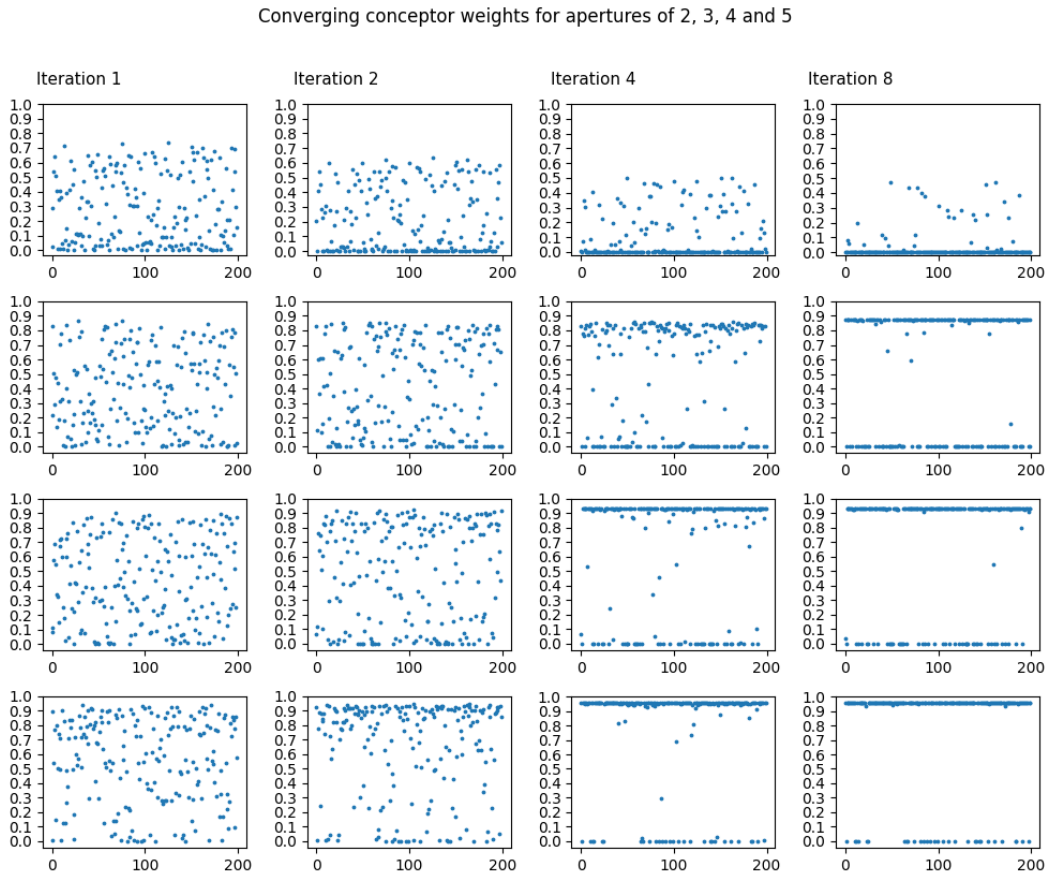

Figure 9: The converging conceptor weights after 1, 2, 4 and 8 iterations (left to right) for apertures of 2, 3, 4 and 5 (top to bottom).

tors that cannot converge. This setup showed potential suitability for an iterative approach, and was explored in Experiment 3.

5 Experiment 3

For Experiment 3, a different architecture and training procedure was used than for Experiment 1 and 2. The setup of this experiment was inspired by output feedback networks: models in which the output at one time-step is fed back into the network at the next time-step as input.

As became clear from the previous two experiments, an iterative approach to constructing conceptors was difficult due to the need to recompute the reservoir weights and the lack of a way to iteratively compute conceptor weights in a stable way. By using an output feedback network, the reservoir weights could be left unchanged and randomly initialised. A schematic overview of the output feedback network architecture with a conceptor used to autonomously generate the patterns can be found in Fig. 10.

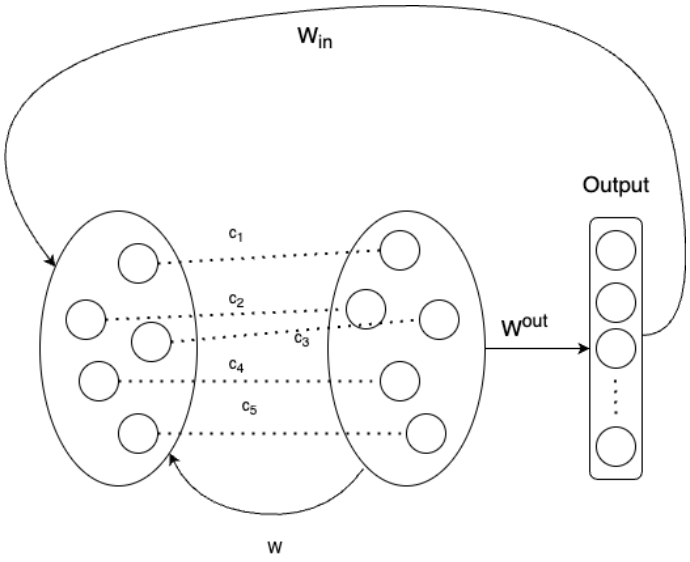

Figure 10: The setup used for autonomous generation of patterns in Experiment 3. Different from Fig. 2 is that W could be left unchanged from the start, and the output was fed back into the network through the input weights.

In the original training procedure, the randomly initialised network was fed with input, and then the reservoir weights were recomputed such that the observed behaviour was approximated without the need for input. As the task was to approximate the input, feeding the output back into the reservoir should lead to the same observed dynamics as feeding the original input into the network, for as long as the output estimated the input accurately enough. One key change needed to be made, to enable the expected behaviour, was that in the original setup at each time-step the input and the output were made to be equal. Thus, the reservoir state associated with a certain input was trained to produce output equal to that input. Now, if that same input was fed back into the reservoir, the state would hardly change, and again the same output would be recomputed and the output would become constant. Instead, the target output was the input shifted forward by one time-step.

Let $\mathbf{P} = \mathbf{p}^1, \mathbf{p}^2, \ldots, \mathbf{p}^p$ be patterns where $\mathbf{p}^j \in \mathbb{R}^M$ for all j. Set up an RNN with the following:

- N reservoir neurons.

- input weights $\mathbf{W}^{in} \in \mathbb{R}^{M*N}$, random values drawn from the standard normal distribution and scaled appropriately.

- reservoir weights $\mathbf{W}^{*} \in \mathbb{R}^{N*N}$, a sparse matrix with random values drawn from the standard normal distribution and scaled appropriately.

Furthermore, to increase stability, the diagonal conceptors were computed in a manner that allowed them to be more stable and constrain the reservoir a similar amount over multiple iterations. Rather than randomly assigning values from the normal distribution to the initial conceptors, they were generated in the following way: For each conceptor a sigmoid-shaped curve was generated from 0 to 1. Then, N values equally distributed along this curve were picked, where N is the number of neurons in the reservoir. Lastly, to each conceptor weight one of the picked values was randomly assigned. When plotted in increasing order based on the conceptor weight, the randomly assigned conceptor weights now follow a sigmoid curve:

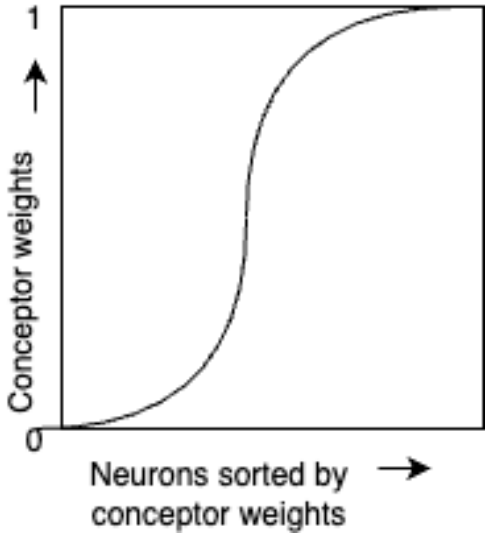

Figure 11: An example of how the conceptor weights are distributed to follow a sigmoid-shaped curve

During the training procedure, the generated values that followed the line of the sigmoid curve were only reassigned to the conceptor weights based on the activation of the respective neuron. This kept the amount of the reservoir being suppressed constant, while only the parts of the reservoir that were being suppressed changed. Where before the aperture was used to control the strength of the diagonal conceptors, now the amount of the reservoir being suppressed was controlled by changing the steepness and shifting the turning point of the sigmoid curve. The steepness and shift of the turning point where chosen heuristically during the training and testing procedure.

The diagonal conceptors as proposed in De Jong (2021) were also tested in this architecture, but no experiment was successful. It was observed that these conceptors were unstable and were unable to control the amount of suppression of the reservoir for the same reasons as discussed in Experiment 2. Therefore, the sigmoid-shaped conceptors were deemed necessary for this particular setup.

The training procedure was altered to be the following: For each pattern $\mathbf{p}^{j}$, feed the reservoir for $n_{washout}$ time-steps with the randomly initialised sigmoid conceptor in place. The reservoir state

update function was used as before, except the bias has been removed as it was found that removing it improved the NRSME of the generated patterns:

$$\begin{aligned}\mathbf{r}^j(n+1) &= \tanh(\mathbf{W} * \mathbf{z}^j(n) + \mathbf{W}^{in}\mathbf{p}^j(n+1)) \\ \mathbf{z}^j(n+1) &= \mathbf{c}_0^j \mathbf{r}(n+1)\end{aligned} \tag{24}$$

After the washout phase, drive the reservoir for another n_{adapt} time-steps. During this period, collect the reservoir state at each time-step such that state collection matrix

$$\mathbf{Z}^j = [\mathbf{z}^j(n_{washout}+1)\mathbf{z}^j(n_{washout}+2)...\mathbf{z}^j(n_{washout}+n_{adapt})] \tag{25}$$

can be created. From this matrix compute which neurons have the highest absolute average activation, by multiplying the matrix with itself element-wise and sort the neurons from highest to lowest activation. Then, assign to the neuron with the highest activation the point along the sigmoid curve with the highest value, to the neuron with the second highest absolute activation the point with the second highest value along the sigmoid curve and so forth. A schematic overview of this process is depicted in Fig. 12.

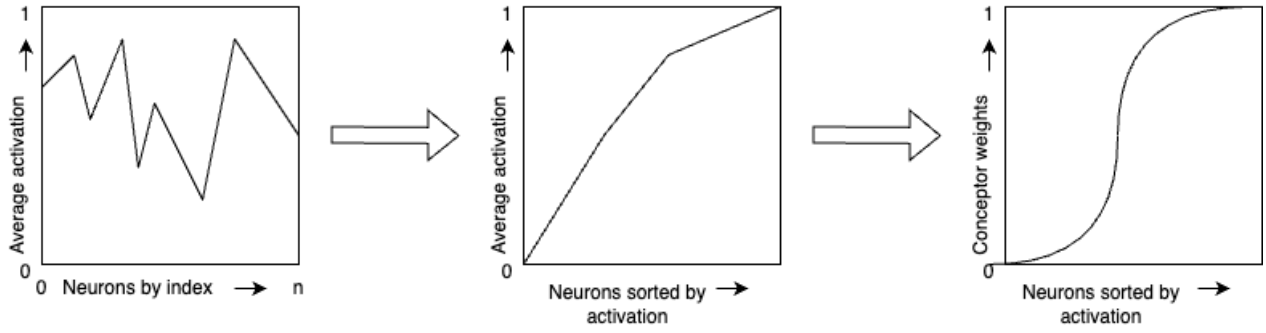

Figure 12: The left graph shows the neurons sorted by their index along the X-axis, while the Y-axis shows the average activation for each of these neurons. In the centre graph the neurons are sorted by their average activation. In the right graph the order of neurons was kept the same, but to each related conceptor weight a value along the sigmoid line is assigned. Note that the graphs are not to scale.

Now with the recomputed conceptor in place, run the reservoir for another n_{learn} time-steps. During this period, collect:

- $\mathbf{r}^j(n)$, to create $\mathbf{X}^j$, where $\mathbf{X}^j_{*,n} = \mathbf{r}^j(n_0+n)$
- $\mathbf{p}^j(n)$, to create $\mathbf{P}^j$, where $\mathbf{P}^j_{*,n} = \mathbf{p}^j(n_0+n)$

Before computing $\mathbf{W}^{out}$, delete the first entry of $\mathbf{P}^J$ and the final entry of $\mathbf{X}^j$, so that when ridge regression computes the output weights, the best mapping from the reservoir state associated with a certain input towards the next time-step is learned. The computation of $\mathbf{W}^{out}$ was done the same as before:

$$\mathbf{W}^{out} = (\mathbf{X}\mathbf{X}^T + \rho^{\mathbf{W}^{out}} I)^{-1}\mathbf{X}\mathbf{P}^T \tag{26}$$

As the reservoir weights have not been recomputed to estimate the observed behaviour during the learning phase, autonomously generating output using Eq. 20 and Eq. 21 did not work. Instead, to generate output, the following formulae were used:

$$\begin{aligned} \mathbf{r}(t+1) &= \tanh(\mathbf{W} * \mathbf{z}(n) + \mathbf{W}^{in} * y^j(t)) \\ \mathbf{z}(t+1) &= \mathbf{C}^j \mathbf{r}(n) \\ y^j(t) &= \mathbf{W}^{out} \mathbf{z}(t) \end{aligned} \tag{27}$$

The advantage of this alternative way of using conceptors to autonomously generate temporal patterns, over the method used in De Jong (2021), is that the the reservoir weights did not need recomputation, which in turn lead to less memory usage for the storage of the reservoir states needed to recompute the reservoir weights. Also, it was more computationally efficient, as the reservoir weights did not need to be recomputed.

Results

For the simulations, the following parameters were used:

- $\mathbf{W}^{in}$ scaling: 0.8
- $\mathbf{W}$ scaling: 3
- $\rho^{\mathbf{W}^{out}}$: 0.1

Additionally, all generated sigmoid curves were slightly shifted towards the right side on the x-axis, to reduce the amount of conceptor weights with a weight close to 1 and this restrict the reservoir dynamics more. This shift was found to reduce the amount the different patterns that would disturb each other during generation. The sigmoid curves that were used are depicted in the fifth column of Fig. 13. For simple patterns, such as the irrational-period patterns and the 5-period patterns, the accuracy remained similar to the training method and network setup as proposed by De Jong.

	Pattern 1	Pattern 2	Pattern 3	Pattern 4
Experiment 1	$3.98e^{-2}$	$4.69e^{-2}$	$2.24e^{-6}$	$8.82e^{-6}$
Experiment 3	$4.63e^{-2}$	$6.90e^{-2}$	$2.13e^{-4}$	$1.67e^{-5}$

Table 1: Side-by-side comparison of the NRMSE accuracy measures of Experiment 1 and 3 after one iteration. Experiment 1 followed the training procedure from De Jong (2021). Experiment 3 employed an autoregressive network for these results.

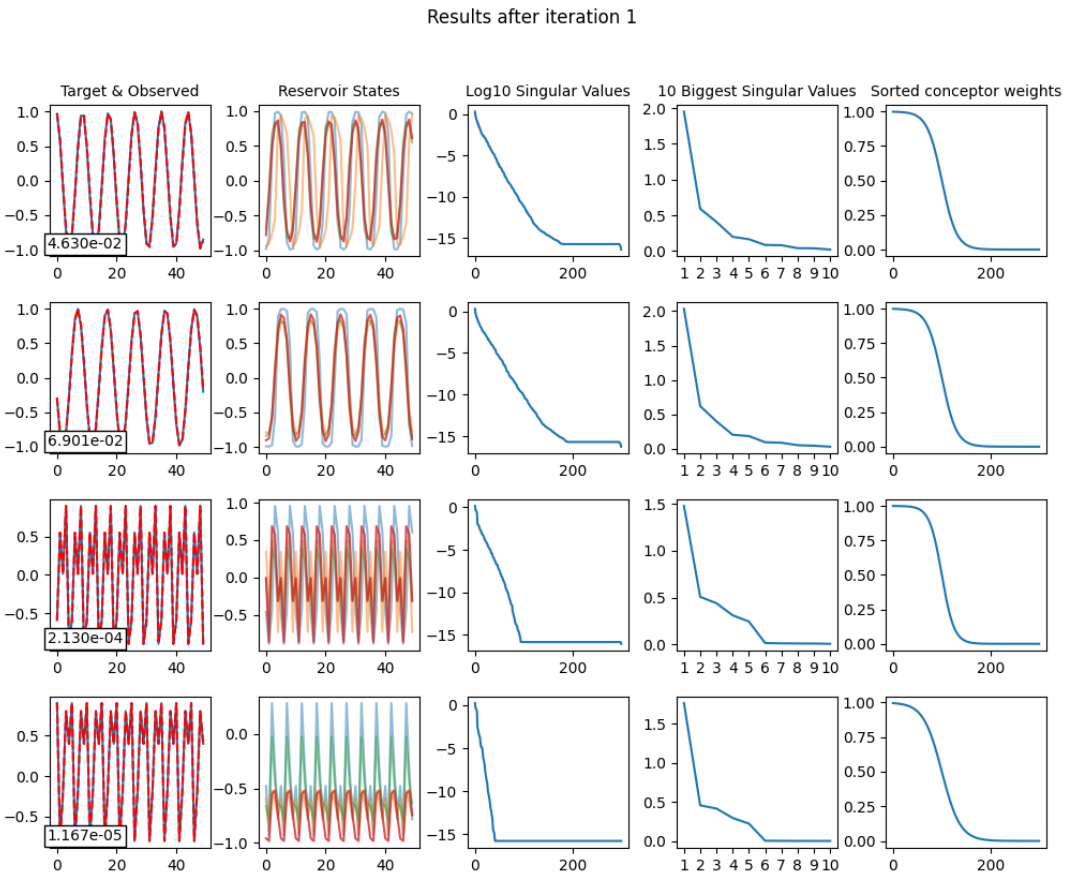

Figure 13: Results after following the training procedure for the output feedback setup. For an elaborate explanation, please see Fig. 4 to clarify.

The results presented in Fig. 13 resemble the results found in Experiment 1, depicted in Fig. 4. There was a similar accuracy. A side-by-side comparison of the NRMSE of both experiments after a single iteration can be found in Table 1.

Furthermore, the principal components of the irrational patterns spanned all of R^N, while the principal components of the 5-period patterns were non-zero in only five directions. The behaviour of the network appeared very similar as before, even though the reservoir weights of the network were left unchanged from the start. Only the output weights and conceptor weights were changed. It is trivial that the conceptor weights did look differently. They were shown sorted to highlight the sigmoid

shape that they followed. For further iterations, the distribution did not change. It was found that the generations were most accurate if more than half of the reservoir was actively suppressed.

After iterating the training procedure another three times, more results were obtained and depicted in Fig. 14.

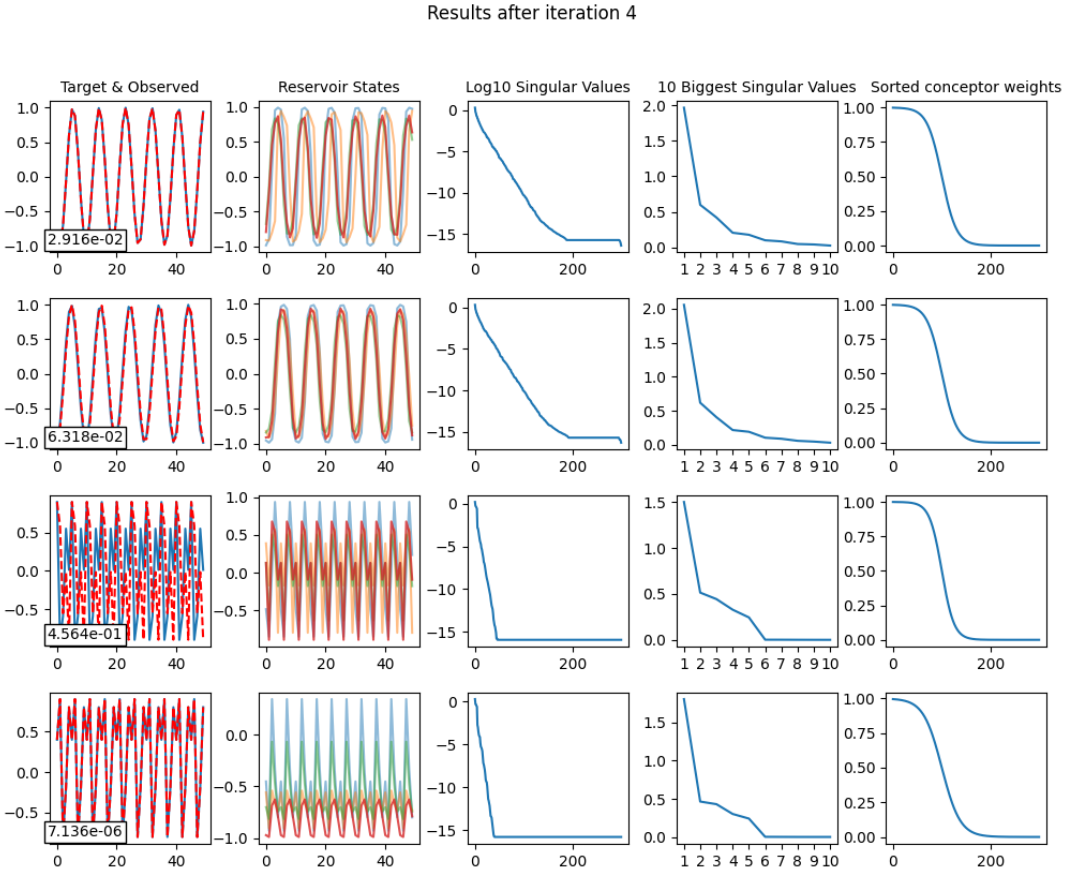

Figure 14: Results after iterating the training procedure for the output feedback setup four times. For an elaborate explanation, see Fig. 4.

From Fig. 14 it was observed that the system was unable to produce accurate output for one of the patterns. However, the reservoir dynamics during generation appeared periodic with the same period as the target pattern from the second column. The singular values also appear to still be non-zero in only five directions, implying that the network is simply revisiting the same five states at time of generation. Table 2 shows the progression of the NRMSE for each pattern over multiple iterations. Concluding from Table 2 the NRMSE did not have a trend towards improvement over multiple iterations but stays rather stable. An increase in accuracy of one pattern seemed to come at the cost of the accuracy of another pattern. After two iterations the system was only able to produce three out of the four patterns accurately.

	Pattern 1	Pattern 2	Pattern 3	Pattern 4
Iteration 1	$4.63e^{-2}$	$6.90e^{-2}$	$2.13e^{-4}$	$1.67e^{-5}$
Iteration 2	$3.88e^{-2}$	$8.47e^{-2}$	$3.23e^{-4}$	$6.28e^{-5}$
Iteration 4	$1.96e^{-2}$	$6.81e^{-2}$	$5.56e^{-1}$	$7.13e^{-6}$
Iteration 8	$3.23e^{-2}$	$6.23e^{-2}$	$4.57e^{-1}$	$1.83e^{-5}$
Iteration 10	$1.59e^{-2}$	$7.30e^{-2}$	$4.57e^{-1}$	$8.01e^{-6}$

Table 2: The progression of the NRMSE over multiple iterations for each pattern, using the output feedback setup.

Discussion

The results of Experiment 3 suggested that iterating the training procedure neither tainted nor improved the results. It appeared that iterating the approach lead to a reshuffle of the conceptor weights, which in turn changed the reservoir dynamics. However, during the iterations, the dynamics remained stable. The setup used was non-converging, meaning that the reshuffling did not stop at some point in some optimum. When comparing the results of Experiment 3 to Experiment 1, especially after multiple iterations, the reservoir dynamics and outputs appear a lot more stable in Experiment 3 than in Experiment one. Thus, potentially the feedback architecture combined with pre-shaped conceptors could be adapted for a more iterative, convergent approach.

Similar to the traditional diagonal conceptors, the sigmoid-shaped conceptors were sensitive to the initial setup of the network. For future studies, it could be investigated how changing the distribution and scaling of the initial network changes the outcome of the simulations. The sigmoid-shaped conceptors were probably not the most optimal solution, nor the most biologically plausible filters. Instead, they were thought of as a means to stop the weights from converging towards some upper or lower limit and an easy way to control the amount of the reservoir that was suppressed for each pattern.

6 Conclusions

From the presented Experiments 1, 2 and 3 it was found that the training procedure proposed in De Jong (2021) was not suited for iterative computation or on-the-fly adaptation. Unstable conceptor weights and unpredictable reservoir dynamics lead to a network that was unable to store and consequently produce autonomous output, even when restricting the input to simple patterns.

Furthermore, it was shown that diagonal conceptors could be successfully employed in a feedback network, which was a more computationally efficient system than the original one. However, as the conceptor weights followed a sigmoid curve, and the amount of the reservoir being suppressed was kept constant through the iterations, the biological plausibility of this approach was questionable.

A key difference from the training method proposed in De Jong (2021) was that, with the output feedback network, there was no actual storage of the patterns taking place within the network. Instead, the activity of the untrained network was restricted via the conceptors, and the feedback from the output weights was used in a way that coherent output from there could be produced. Alternative ways to control the amount of the reservoir being suppressed or to prevent the conceptor weights from converging emerged as an interesting study for the future.

References

Barak, O. (2017). Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology*, *46*, 1–6. doi: 10.1016/j.conb.2017.06.003

De Jong, J. P. (2021). *Controlling recurrent neural networks by diagonal conceptors* (Master's Thesis). Rijksuniversiteit Groningen.

Jaeger, H. (2014). Controlling Recurrent Neural Networks by Conceptors. *arXiv preprint arXiv:1403.3369*.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149. doi: 10.1016/j.cosrev.2009.03.005

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 1310–1318). PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. doi: 10.1109/5.58337

Williams, R. J., & Zipser, D. (1989). Experimental Analysis of the Real-time Recurrent Learning Algorithm. *Connection Science*, *1*(1), 87–111. doi: 10.1080/09540098908915631

Yang, G. R., & Molano-Mazón, M. (2021). Towards the next generation of recurrent network models for cognitive neuroscience. *Current Opinion in Neurobiology*, *70*, 182–192. doi: 10.1016/j.conb.2021.10.015