Parametric properties in chaotic neural networks

Lino Vliex, s4347471

Artificial Intelligence

Radboud University Nijmegen

Supervised by

Silvan Quax & dr. Marcel van Gerven,
Donders centre for cognitive science, Donders Institute, Radboud University

Bachelor's Thesis in Artificial Intelligence

July 25, 2016

Abstract

In the framework of reservoir computing a randomly constructed recurrent neural network (RNN) is used to function as a reservoir containing input history and its many random transformations. Output units transform whatever is inside this reservoir into a meaningful output. However, it is difficult to appropriately construct a random reservoir such that the complete model is able to learn and perform a certain task. A model's performance depends on two things: the parameters of the RNN and the task's complexity. The used network parameters need to be fine-tuned to the task at hand for achieving the best results. First we propose and investigate several measures of quantifying the difficulty of the chosen to-be-trained pattern. The used patters are visual. Hereafter we investigate RNN performance after FORCE learning on tasks of different complexities, selected with the apparently most appropriate of these measures, and we investigate how an RNN's parameters determine its eventual capability to learn. Under investigation are reservoir size, connectivity and weight scaling. All findings were done through simulation studies, with additional theoretical explanations and references to relevant literature.

Acknowledgements

I would like to express my gratitude to my supervisors Silvan Quax and dr. Marcel van Gerven for their invaluable advice, suggestions, patience and enthusiasm during this project. I would also like to thank Jaap Buurman for the very informative discussions, which provided me with very helpful insights.

Contents

Chapter 1

Introduction

A recurrent neural network (RNN) is a network of artificial neurons containing feedback projections, such that connections between units form loops. Such RNNs can be structured, for example, in layers, but they can also be semi-randomly constructed, as in the reservoir computing approach. In this approach a randomly constructed RNN called the “reservoir” is used to perform arbitrarily complex computations. It projects to one or several readout units, which, in a way, translate the reservoir’s activity into a meaningful output. The reservoir may receive input from an input pathway, which may also directly project to the readouts. The reservoir may additionally or exclusively receive input from feedback projections from the output units. From now on such a model in its entirety will be referred to as the RNN. Learning typically only happens for the output weights, as was done in this project, but it is possible to train all connections of a model.

These neural networks are considered general purpose systems that are taught to perform a specific task through synaptic modification. they are inspired by the architecture of cerebral microcircuits of neurons, as it is believed that seemingly random groups of neurons work together to perform a certain complex task. High recurrence in artificial neural networks allow them to perform complex computational tasks and exhibit dynamic temporal behaviour [22, 16]. Besides biologically plausible modelling, these models are also used for more technical applications of signal processing, such as filtering, detecting certain events and classifying time series, where they excel over other neural models because of their capability of learning long term dependencies [11].

1.1 Issues in reservoir computing

Reservoir computing approaches are often applied in such a way that the reservoir is treated as if it were a black box, as it is not obvious what goes on inside (although methods exist by which can be understood what happens inside the reservoir. See for example [23]). However, these RNNs are easy to train and apply, as only the output weights need to be trained and the reservoir needs not be understood as long as it is constructed in an appropriate fashion. How to construct an RNN appropriately, i.e. in a way to make it able to learn, is described in much of the relevant literature.

A known issue in neural networks research is catastrophic interference, also known as catastrophic forgetting [3]: a neural network is not able to learn infinitely many patterns, and at some point in learning newer patterns it will forget previously learnt ones. It may even happen that teaching one pattern too many will cause the network to forget all previously learnt patterns. In reservoir computing this problem is easily overcome by adding new readout units and training these on the new items instead of the previously established output units trained on the previous items, while using the same reservoir. So instead of instantiating an entire new RNN only the required amount of inputs and outputs have to be added, together with some appropriate global parameter settings. This is not biologically implausible, as it is believed that the same brain areas participate in multiple tasks, while possibly being modulated by a global signal for the task specifically at hand, in a similar fashion [8, 15].

Despite these advantages in initial usage of these RNNs, the question arises what aspects and to what degree certain aspects of a neural network determine its potential. Reverse engineering an RNN to be most appropriate for a certain task is difficult. Although these models are considered general purpose systems, the network parameters chosen determine their eventual capability to learn a specific task.

These RNNs outperform many other approaches, especially in tasks containing strong temporal dependencies as these RNNs already have implicit short-term fading memory, and other forms of memory may emerge through training if it is required by the learnt task. The RNN's fading memory capacity determines its ability to "remember" its state and input history, and it is bound by the amount of neurons in the reservoir, such that the number of "memorized" previous states available to the readouts can not exceed the amount of reservoir units [9]. The reservoir also implements computation of many non-linear combinations of input components and their projection onto a very high-dimensional space through its many neurons and their random connectivity [17]. This high dimensionality of the reservoir's state usually facilitates the readouts, but, although these beneficial properties are bound by the amount of neurons put in the reservoir, Koryakin et al. [12] found that the RNN's eventual performance degenerates with the use of too many neurons, after training with the echo state approach. They concluded that there is an optimal reservoir size for a certain task. In this project we investigated if this also holds in chaotic RNNs trained with the FORCE learning procedure [22]. The FORCE training procedure was used for all RNN training in this project, and it is described in section 2.1.1.

Furthermore, we investigated the effect of the amount of connections used in the RNN, as this determines the overlap of features in the high-dimensional projection of previous inputs and states in the reservoir. Sparse connectivity in the reservoir lets it decompose into many loosely coupled sub-networks. This way a reservoir is created with rich dynamics, that allows multiple independent internal representations, which should benefits performance [16, 8, 11]. However, the presence of many non-linear combinations of input components and their projections onto a very high-dimensional space do not disappear with greater inter-connectivity of neurons [17]. Therefore it remains questionable whether sparser connectivity benefits RNN performance, especially since Koryakin et al. [12] did not find any such effect. Moreover, they found connectivity sparseness not to have any effect at all.

Finally, we investigated the benefits of using larger weights for the connections in the reservoir, resulting in reservoirs transitioning from being ordered to being chaotic. More accurately the presence of chaos in the reservoir also depends on the magnitude of input and external feedback signals. Put simply, activity in chaotic reservoirs never dies out, as it would in ordered reservoirs. This means that the state of the reservoir will always depend on its initialisation, while in ordered reservoirs it will only depend on a certain amount of previous states. In many approaches ordered dynamics are favourable, e.g. in the echo state approach, because chaotic activity tends to enlarge small errors made by the network over time, making the RNN's output increasingly erroneous over time [8]. In the FORCE learning procedure, however, chaotic spontaneous activity in the RNN before learning is beneficial: learning converges faster, the trained RNNs are more accurate and they are more stable [22]. Furthermore, chaotic RNNs have a greater computational capacity in general than non-chaotic RNNs [2]. On the other hand, if the chaos becomes too large also FORCE learning will fail. This depends on the ability to suppress the reservoir's internal chaos through the external feedback pathway. Therefore we investigated the effect of both reservoir and external feedback weight strength on the speed of learning and the resulting RNN's accuracy.

Chapter 2

Methods

2.1 Network architecture & state updates

The recurrent neural network model used in this project is designed as a generic recurrent circuit not built for any specific purpose, following the mentality of liquid-state machines [16] and echo state networks [8]. Such circuits are molded to fit a certain task through synaptic modification. Such RNNs consist of what is conventionally called a "Reservoir" of sparsely interconnected neurons with synapses being either inhibitory or excitatory and of varying strengths. The outputs of neurons model their firing rates, which, for the reservoir, are collected into a column vector **r**. To this reservoir readout units are connected (all-to-all). The network output at time t is defined as

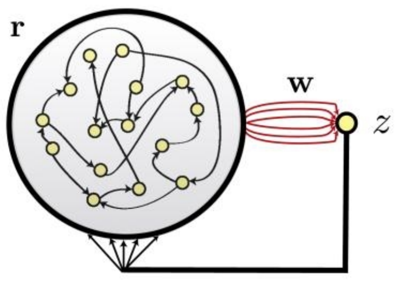


Figure 2.1: A schematic view of the RNN architecture used. In this model only the output weights **w**are trained. Figure taken from [22].

$$z(t) = \mathbf{w}^\top \mathbf{r}(t) \tag{2.1}$$

where **w** is a $N \times B$ matrix of readout weights from N reservoir units to B readouts. Equation 2.1 corresponds with the identity function as activation function for the readout units. In our models the activation function used for the reservoir neurons is the sigmoid $tanh(\cdot)$, but other appropriate functions exist such as other sigmoid functions or even just the identity function [19, 9].

Linear units are shown to work best for improving an echo state network's (ESN) short term memory capacity or even general computational capacity in the echo state approach of RNN training. However, this is biologically less realistic and not shown to improve performance for models trained using FORCE learning, so we have followed [22] in activation function choice. Finally, through an external feedback loop the read out values are fed back into the reservoir, typically all-to-all. See figure 2.1 for a schematic depiction of our used RNN architecture.

Changes in reservoir activations $x(t)$ at a time t result from feedback projections from the outputs and from previous reservoir firing rates $\mathbf{r}(t-dt) = tanh(x(t-dt))$ through recurrence. So the reservoir state update equation is defined by

$$x(t+dt) = \big(1 - \frac{dt}{\tau}\big)x(t) + \frac{dt}{\tau}\big(g\mathbf{J}\mathbf{r}(t) + w_{fb}z(t)\big) \tag{2.2}$$

where $x(0)$ is randomly drawn from a standard normal distribution and multiplied by 0.1, τ is the time constant of all neurons, $\mathbf{J}$ the $N \times N$ internal weights matrix, g the spectral scaling of J as its spectral radius is approximately one, and w_{fb} the $N \times B$ feedback weights matrix. The network is chaotic for $g > 1$ [20].

The feedback weights matrix is randomly constructed with each weight drawn from the uniform distribution on the open interval $(0, 1)$. The internal connectivity matrix $\mathbf{J}$ is first constructed as a random sparse matrix with approximately $p \cdot N^2$ normally distributed nonzero entries (with zero mean and a variance of one).For example if $p = 0.1$, then 10% of all weights in $\mathbf{J}$ are nonzero. thereafter $\mathbf{J}$ is scaled by dividing it by $\sqrt{p \cdot N}$. This ensures that the spectral radius of J is approximately 1, such that the networks operate on the boundary between ordered and chaotic dynamics if $g = 1$ [4, 26, 2]. Simultaneously this scaling by the inverse of $\sqrt{p \cdot N}$ calibrates the variance in the input to each neuron from other reservoir neurons, so that this variance is independent of the amount of recurrent projections each neuron receives. This ensures that a change in the chosen connectivity sparseness of the reservoir does not influence the RNN's behaviour due to simply changed input variance of the neurons. See the appendix for how this input calibration can be derived.

2.1.1 Training

Several training methods exist to train recurrent neural networks, such as backpropagation through time or other gradient methods [25], evolutionary algorithms (e.g. [24]), and reservoir computing approaches [15]. Reservoir computing approaches are seldom outperformed, and especially excel in producing networks with a relatively great short-term memory capacity compared to gradient-descent methods. A particularly powerful exception to this are dedicated long short-term memory circuits specifically designed for memory [5]. An additional advantage to reservoir computing is the ease of training, as merely the weights from the reservoir to the output units need to be trained instead of all synapses. Several training methods exist, such as through regression as in the echo state approach [8], reward-modulated Hebbian learning [7] and the FORCE-learning procedure [22]. In the echo state approach RNNs are trained by initialising with some random reservoir state, feeding a teacher forced output back into the network (i.e. clamping the output feedback to the target), ignoring some initial period of network activity until the reservoir state no longer depends on the initial random state, and performing a linear regression to determine the output weights. This method often results in stability issues in the resulting RNN, because of the clamping of output feedback. One can imagine that the RNN will seldom generate perfect output in the test phase, and will feed erroneous feedback to the reservoir. This will cause a different trajectory of the reservoir's internal state from the training phase, making the output drift away from the target pattern.

In [8, 10] this is resolved by inserting noise into the reservoir during training. With reward-modulated Hebbian learning the output feedback is not clamped to the target, as it is an unsupervised training method, but noise is actually a requirement for learning to happen, as learning happens through a sort of trial and error [7]. In the FORCE learning procedure instability is resolved differently. Sussillo and Abbott [22] do not clamp output feedback to the target function, but instead use rapid updating of the output weights during training to make the fed back signal close to what it would be if the output perfectly matched the target. They named this procedure therefore the First-Order Reduced and Controlled Error, or FORCE, learning procedure. A weight modification algorithm that suits the requirement of rapidly and effectively updating weights to keep the fed back error small is the recursive least-squares (RLS) algorithm [see 6, chapter 10].

Our RNNs are also trained by only updating the output weights $\mathbf{w}$, but it is possible to train all weights, using the FORCE learning procedure. We use the RLS weight update algorithm, following [22]. For a more elaborate description of RLS, see [6]. In FORCE learning, there are two errors to consider at each time step t of weight modification, where dt is the time interval between weight updates, with respect to a target function f and reservoir activations $\mathbf{r}$. The considered errors are the error $e_-(t)$ before weight modification and the error $e_+(t)$ after weight modification:

$$e_-(t) = \mathbf{w}^\top(t - dt)\mathbf{r}(t) - f(t) \tag{2.3}$$

$$e_+(t) = \mathbf{w}^\top(t)\mathbf{r}(t) - f(t) \tag{2.4}$$

This $e_+(t)$ is also the error fed back into the reservoir. When FORCE learning implements RLS, the output weights vector $\mathbf{w}$ at a time step t is updated according to

$$\mathbf{w}(t) = \mathbf{w}(t - dt) - e_-(t)\mathbf{P}(t)\mathbf{r}(t), \tag{2.5}$$

where $\mathbf{P}$ is a running estimate of the inverse correlation matrix of network activations $\mathbf{r}$ and is computed with

$$\mathbf{P}(t) = \mathbf{P}(t - dt) - \frac{\mathbf{P}(t - dt)\mathbf{r}(t)\mathbf{r}^\top(t)\mathbf{P}(t - dt)}{1 + \mathbf{r}^\top(t)\mathbf{P}(t - dt)\mathbf{r}(t)}, \tag{2.6}$$

which requires an initial $\mathbf{P}(0) = \frac{\mathbf{I}}{\alpha}$, where $\mathbf{I}$ is the identity matrix and α the regularisation constant of the RLS algorithm. Equation 2.5 is very similar to a delta-rule, but instead of using a single learning rate multiple learning rates provided by $\mathbf{P}$ are used. In FORCE learning α has to be much smaller than the amount of neurons used. If α is too big FORCE learning may not keep the output close to the target function and learning may fail, but if α is too small weight changes may be so rapid that FORCE learning becomes unstable. Often $\alpha \geq 1$ is appropriate, as long as $\alpha \ll N$.

2.1.2 Testing & performance evaluation

Some researchers directly follow up the training phase with the test phase without altering the network's state after training [12, 22], which is mostly done if the reservoir's activity is only driven by an output feedback loop and not by any input signal. Others use a "washout" phase before the actual test phase: the network is reset and left running shortly between the training and test phase while feeding it a certain input or feedback signal. For example, Jaeger [8] resets the used ESN's internal state to 0 and clamps the output feedback loop to the target function for a short while, after which the network is left running independently (that is without teacher forcing) in what is the actual test phase. A washout phase is effective in ordered RNNs as its internal state only depends on a certain number of previous states, so at some point the internal state no longer depends on its resetting. We follow the former approach, i.e. we do not reset the reservoir before the testing phase, as a washout phase is not effective in chaotic RNNs, since the reset state, whether to 0 or anything else, can never be washed out. Instead the RNN is left running independently after the training phase and its outputs z are collected. However, chaotic RNNs are able to recover if their internal state is reset, which is impressive given the chaotic trajectory of its internal state. Nonetheless we chose to leave out a washout phase to reduce variance in our experimental outcomes, as the mentioned recovery is not guaranteed.

After the testing phase the RNN's performance is measured by its Mean Euclidean error (MEE) from the target pattern, which is defined as the average 2-norm of the error vector $e_f = y_f - z_f$ of each target output f, that is

$$MEE = \frac{1}{F}\sum_{f=1}^{F}\sqrt{e_f^\top e_f}. \tag{2.7}$$

In many figures we include an indication of the so-called chance level performance on a particular target. This chance level performance is defined as the MEE obtained with uniform random guessing of each pixel value.

All data processing was performed off-line using MATLAB R2016a (The MathWorks Inc., Natick, MA, 2000).

2.2 Data set

For all experiments video clips were used from the *HOLLYWOOD2 Human Actions and Scenes Data set* [18] as target functions for the RNNs. The data set contains 3669 video clip samples picked from 69 movies. The data set was used by Marszalek et al. [18] to build support vector machine classifiers able to classify scenes and actions in visual input. The data set is split up in a scenes and an actions data set, but in this project all videos were used with equal probability when examining their complexity, but three particular clips (mentioned below) were selected for all subsequent experiments. Each movie was re-sized to be 48×48 pixels, as this kept objects and people recognisable in the video clip and allowed for an acceptable computational load of the simulations. Of all movies the frame rate was about 25 frames per second. Movie lengths of 300 frames were used, which comes down to circa 12 seconds of video material, unless explicitly mentioned. All pixel values were converted to grey scale values, which were scaled to lie between 0 and 1, and gave the RNNs an amount of outputs equal to the amount of pixels in each frame.

Pixels in a video clip can be considered separately, but as each pixel in the RNN output contributes to the reservoir's internal state through the feedback pathway, it is more sensible to consider all pixels in a frame together. This is possible by looking at each pixel intensity value as if it were a coordinate in a very highly dimensional space, in which a frame is represented by a single point. In this way a movie clip is simply a trajectory through this space. While it is impossible to imagine a space of $48 \times 48 = 2304$ dimensions, we can get an impression of what such a trajectory would look like by means of a principal component analysis (PCA), for which we use the singular value decomposition (SVD) of all frames in a clip. For this we used the `pca` function in MATLAB, by which we obtain the principal components (PC's) of a clip in descending order of component variance. We projected the three mainly used clips onto their first 3 PC's in figure 2.2. These projections give an indication of the complexity of these clips, but, although they were selected by their complexity, they were selected using different criteria.

2.2.1 Video clip complexity

To obtain video clips of appropriate difficulty, i.e. clips for which not only the difficulty of learning but also visual characteristics varied considerably, we searched for a data complexity measure that captures visual characteristics and simultaneously correlates strongly with RNN performance. An RNN is very able to learn to produce very similar frames, but as subsequent data points begin

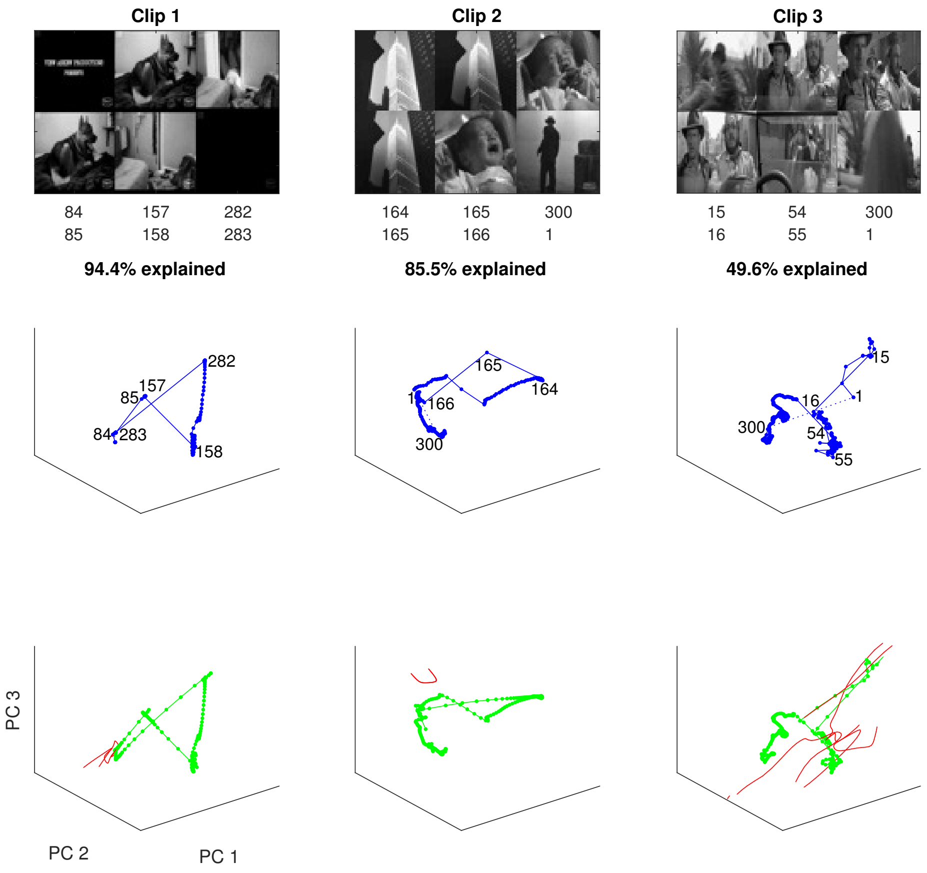


Figure 2.2: A principal component (PC) analysis of the 3 mainly used clips.
The images in the top row display frames from the used clip. Each lower frame is the direct successor of the frame above. Between each displayed pair of subsequent frames there is a large distance in the PC space. The corresponding frame numbers are written below.
The blue plots in the middle row show the clip trajectories projected onto their first three PC's. Some points in these plots are labelled with the corresponding frame numbers. Above each plot the proportion of variance explained by these 3 PC's is displayed.
The plots in the bottom row show RNN performances on the same clips projected onto the same 3 PC's. Green is of one of the best performances, red is of one of the worst performances.

to show an increasing difference, e.g. a scene transition or a lot of motion in a movie clip, the test error increases rapidly. Output errors range from mixing characteristics of the target patterns to producing something indistinguishable from random output. So the clip difficulty measures evaluated were designed to quantify to some degree the increase of difficulty due to the presence of scene transitions, camera angle transitions, and large motions in the used clip. A desirable data complexity measure is one such that an RNN's test error increases as the used clip's complexity score increases. We investigated multiple measures to investigate which one is strongest correlated with, or best predicts, network error.

As complexity measures we considered total cosine distance, total city block distance and total Euclidean distance between all subsequent frames, and we considered information contained in data point transitions. These measures are meant to capture the intuition that two subsequent frames of a scene transition, or subsequent frames containing much motion of the pictured objects or people, are very dissimilar and thus harder to learn. The main difference between the cosine and entropy measures, and the euclidean and city block measures, is that the former are more sensitive to changes in separate regions of the clip, while the latter are more sensitive to changes in the entire frame. For example, the cosine distance between a white and a black frame, i.e. from an all ones to an all zeros vector, is zero, as is the entropy in such a transition, but the Euclidean and city block distance between such frames is maximal. Another such transition is a change in brightness, as all pixel values move in the same direction. The city block measure is most sensitive to such transitions, while Euclidean distance is less sensitive to this due to its realisation of the triangle inequality.

Now we will describe how each measure is computed for a video clip. Let F be the amount of frames in the video clip and let B be the amount of pixels in each frame (and so the amount of output units for an RNN). Now let $y_{f,b} \in [0, 1]$ be the target value of one pixel $b \in [1..B]$ of B pixels in a frame $f \in [1..F]$. A video clip with F frames has $F - 1$ frame transitions, but as each RNN was trained to repeat the trained video clip, there is also a frame transition from the last frame to the first. Therefore we define the next frame f^+ that comes after a frame f as

$$f^+ = (f \bmod F) + 1$$

Now we define the city block complexity of a video clip as the sum of city block distances between all subsequent frames. The city block complexity measure can be described as the sum of all absolute pixel luminance changes. So

$$C_{cityblock} = \sum_{f=1}^{F} \sum_{b=1}^{B} y_{f^+,b} - y_{f,b}$$

If all frames are considered points in a highly-dimensional Euclidean space, then the Euclidean complexity measure can be described as the total distance travelled when moving through all points in the data set. We define the Euclidean complexity as the sum of Euclidean distances between all subsequent frames, that is

$$C_{Euclidean} = \sum_{f=1}^{F} \sqrt{\sum_{b=1}^{B} (y_{f+,b} - y_{f,b})^2}$$

Our cosine complexity is defined as the sum of cosine distances between all subsequent frames,

$$C_{cosine} = \sum_{f=1}^{F} 1 - \frac{\sum_{b=1}^{B} y_{f+,b} \cdot y_{f,b}}{\sqrt{\sum_{b=1}^{B} y_{f+,b}^2 \cdot \sum_{b=1}^{B} y_{f,b}^2}}$$

Finally, the entropy of data point transitions is a quantification of the complexity of such a transition in terms of information contained in the transition. The more information is needed to describe the transition, the higher the entropy. The entropy is computed by allocating of one transition all pixel transition values $(y_{f+,b} - y_{f,b})$ to one of 256 bins. Then the probability p_i that a pixel transition value corresponds to a bin value is computed for each bin i. The transition entropy E of a frame f is computed using $E_f = -\sum_i p_i \cdot log_2(p_i)$. This is implemented by the `entropy` function in MATLAB we used. We define the entropy complexity as the sum of all transition entropies:

$$C_{entropy} = \sum_{f=1}^{F} E_f$$

In figure 2.3 these complexity measures are displayed against frame number for several clips. It can be seen that each measure behaves differently. Note that in this figure each measure was scaled to lie within the same range; otherwise their values would lie very far apart.

Figure 2.3 shows that the entropy measure does not behave as expected. It was expected to be similar to the cosine measure, but it appears to be close to a straight line. This is likely due to the noise present in the video clips, making each transition contain more information than would be initially expected. In figure 2.4 the entropy of each frame transition in `sceneclipautoau-totrain00001` is plotted. Above the plot some sample frames are depicted to give an indication of the transitions in the clip. Although these transitions do not seem to be very sharp, the entropy in each transition fluctuates heavily. Due to much noise in the video clip material, the used entropy computation was not discriminative for dissimilarity between subsequent frames.

2.3 Experimental design

In each experiment regarding the effect of a parameter's setting, performance of RNNs on learning to generate video clips from the Hollywood2 data set

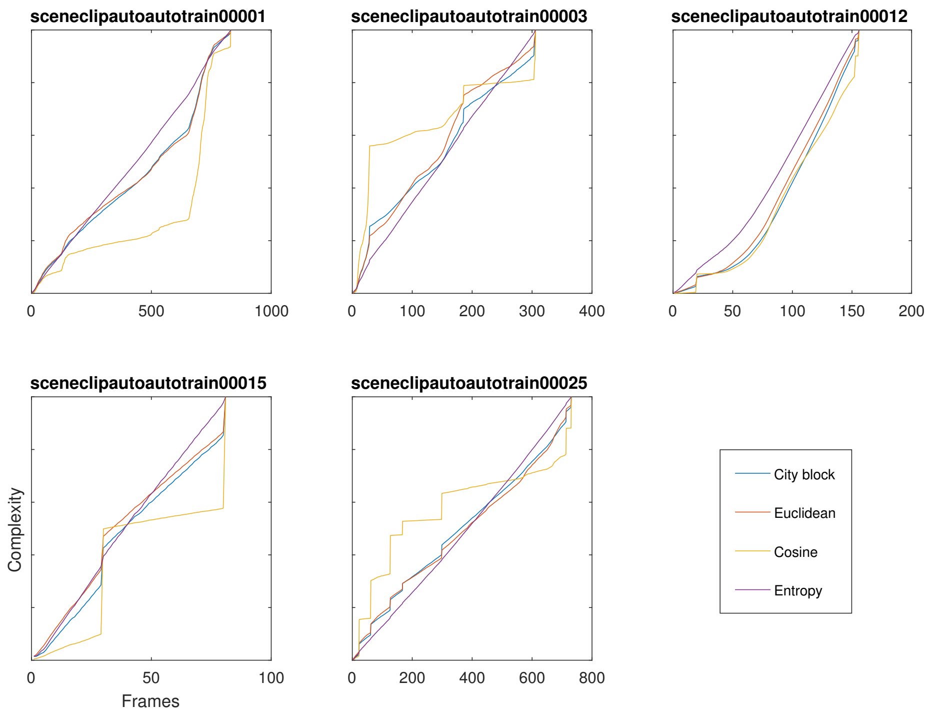


Figure 2.3: Complexity measure values against amount of frames taken for several clips, scaled to lie within the same range.

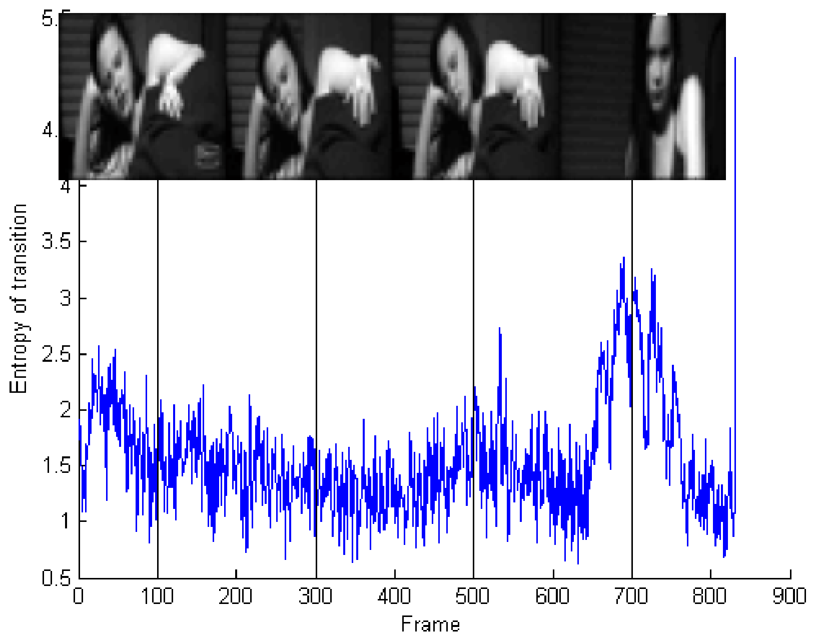


Figure 2.4: A plot of the entropy of each frame transition for sceneclipautoautotrain00001.avi. At the top some sample frames are displayed. The vertical black lines indicate at which time point each sample frame was picked. Note that the y-axis starts at 0.5. The peak entropy starting around 650 frames is when the woman in the video clip sits up straight.

[18] is investigated. Each experiment was repeated for three particular clips, namely the `sceneclipautoautotrain00232`, `actionclipautoautotrain00684` and `sceneCliptest00292` AVI clips, also referred to as clip 1, clip 2 and clip 3 respectively. These clips contain desired scene transitions, camera angle transitions, camera panning and movements of entities. The clips were resized to 48×48 pixels and only the first 300 frames were taken. We used three particular clips instead of all random clips to eliminate variance caused by random clip selection, as this proved an issue in prior experimentations. How these clips were exactly selected is described in section 3.1. Unless explicitly mentioned, RNNs were constructed randomly with the corresponding parameters

$$\begin{array}{lll} N = 300 & g = 1.5 & \tau = 1 \\ B = 48^2 = 2304 & dt = \frac{1}{framerate} & \alpha = 1 \\ p = 0.1 & & \end{array}$$

Learning happened every third frame, and each RNN trained for $n = 5$ training passes over a video clip.

Using this design we first investigated how to appropriately select clips of differing difficulty, i.e. which complexity measure best describes a clip's difficulty, so we could use this to select easy and hard clips. After having selected the three clips mentioned above using the most appropriate complexity measure, we investigated the effect of changing several parameters individually on the resulting RNN's performance. This was done for reservoir size N, reservoir connectivity sparseness p and the reservoir's spectral radius g. The effect of some parameters may have consequences for another parameter's effect due to their implications for the RNN's behaviour. For example, one parameter's optimal value may change as another one is chosen differently. Therefor we investigated whether there are such interaction effects between p and p_{fb}, g and g_{fb} and between g and n. More on how or why specifically these pairs were selected, is provided in the corresponding results sections. In our results we do not elaborate on the effect of g individually, as its effect is strongly related to g_{fb}. Therefor these parameters are discussed together.

Chapter 3

Results

3.1 Data complexity measures

In our search for the most appropriate data complexity measure, we trained 1000 RNNs on 1000 randomly selected video clips and collected the test MEEs. Each clip's length was picked randomly between 200 and 500 frames. The complexity of each used clip was determined with the four complexity measures, such that we could investigate which complexity measure correlates strongest with network error. The strongest correlation $r = 0.684$ was found for the Euclidean complexity measure. For the city block complexity measure it was $r = 0.656$, for the entropy measure $r = 0.647$, for the cosine measure $r = 0.338$, and notably the correlation between the amount of frames and the MEE was $r = 0.275$. See figure 3.1 for a scatter plot of the MEE against Euclidean complexity.

With the Euclidean complexity measure we selected the three clips we used for our other experiments: `sceneclipautoautotrain00232`, `actionclipautoautotrain00684` and `sceneclipteest00292`. These clips were found to have very different complexities with similar amounts of frames. The euclidean complexity scores of the 300 frames used of each clip are respectively 238.89, 344.01 and 752.53. These clips do not only vary in difficulty, but they also vary in contained visual characteristics. The first clip mostly contains rapid transitions of what is viewed and little actual motion by the camera or by depicted entities. The second clip is similar, but contains fading instead of instant scene transitions and contains more motion of depicted entities, which is mostly due to the entities being depicted larger. The third clip contains many instant changes in camera direction and contains a lot of motion.

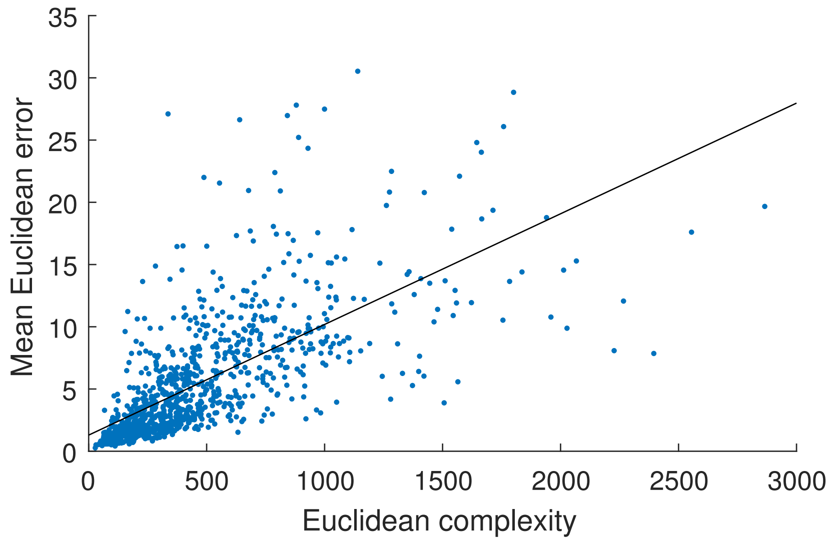


Figure 3.1: A scatter plot of mean Euclidean error against euclidean complexity for 1000 clips. A black least-squares line is added.

3.2 Reservoir size

In many investigations it was found that a larger reservoir size improves performance of an RNN and especially as the modelled problem's complexity increases (for example [14]). While using the echo state training approach Koryakin et al. [12] found there is an optimal reservoir size beyond which enlarging again reduces performance. Therefore we investigated the effect of reservoir size on video clip generation performance when using the FORCE learning procedure. Note that we do not provide a comparison between the two training methods.

For each video clip used as target pattern we trained 400 RNNs with $N \in [1..3000]$, taken with logarithmic increments. The resulting performances are scattered in figure 3.2 paired with plots of the likelihood of achieving at least a certain performance.

In general we found that increasing reservoir size always benefits performance, and that here is no reservoir size beyond which enlarging becomes counterproductive, but instead that the probability that FORCE learning successfully trains an RNN increases with N, or to put it differently, that the expected test MEE drops with N. So if some reservoir size already allows a performance close to the best performance possible, then enlarging beyond that N increases the likelihood of achieving such a good performance. However, obviously it is more efficient to use smaller RNNs that achieve the same results, in terms of computational load.

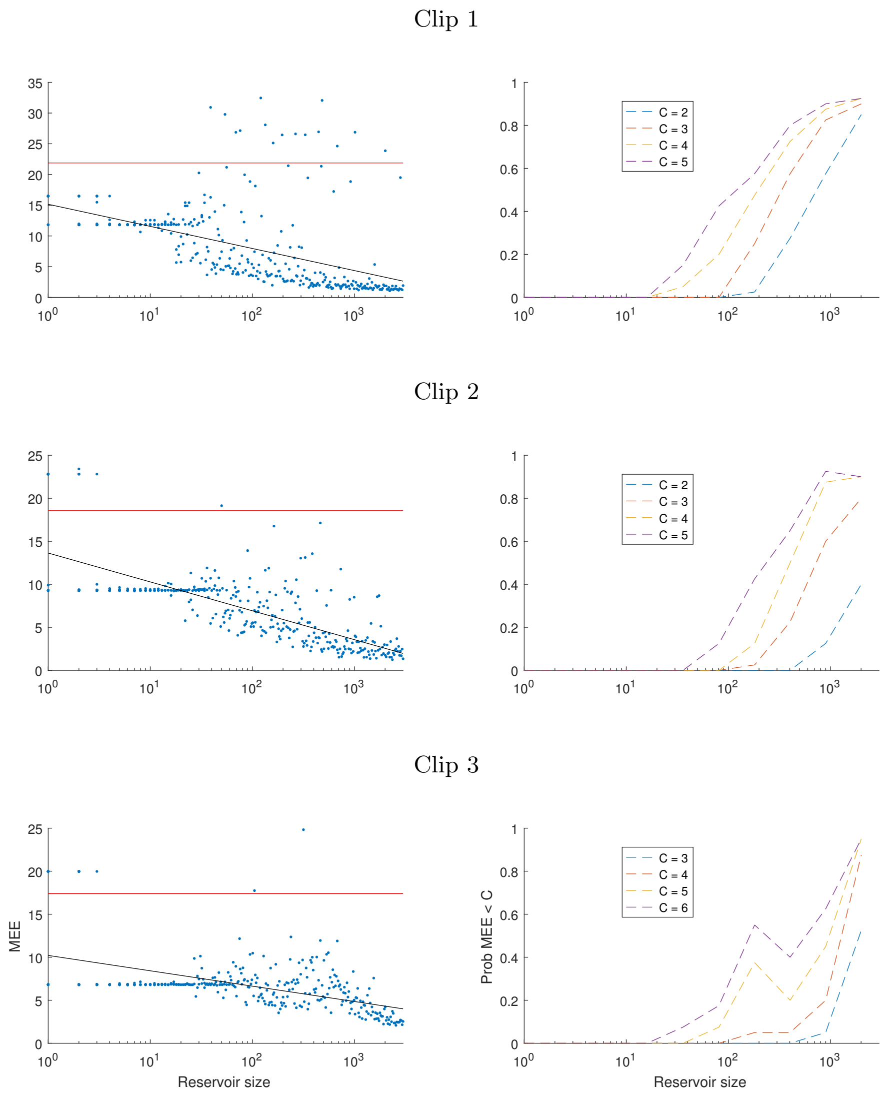


Figure 3.2: To the left, scatter plots of RNN performances against reservoir size. To the right there are plots of the likelihood of achieving at least a certain performance against reservoir size. The smallest C was chosen to be an integer below which at least one RNN performed. The red horizontal line indicates chance level performance.

The test MEE appears to decrease virtually linear with $log(N)$. For clips 1, 2 and 3 respectively we find correlations between $log(N)$ and the test MEE of $r = 0.567$, $r = 0.732$ and $r = 0.496$. Least-squares lines are included in figure 3.2, and their respective slopes are $b = -3.597$, $b = -3.356$ and $b = -1.785$. Assuming these lines represent the correct relation, these coefficients indicate an expected decrease of the mean Euclidean error for a proportional increase of reservoir size: If the reservoir size is changed from some N_1 to N_2, then the expected change in MEE is $b \cdot log(\frac{N_2}{N_1})$. For example, if the new size is ten times the old size, then the MEE would be expected to change by b. This can be explained by the fact that more reservoir neurons enable a higher dimensional projection of the fed back signal and of recent states, and compute more non-linear combinations of input components and features, which boost the performance of linear readouts [17]. That performance increases with proportional growth instead of linear growth of the reservoir size can be explained by the occurring overlap of representations in the reservoir, almost surely inevitable due to its random construction.

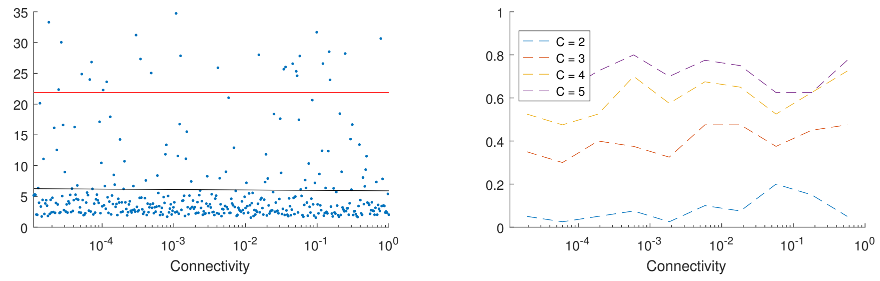
3.3 Connectivity

Some researchers have found connectivity in terms of reservoir structure to be of great influence on RNN performance [19, 21]. But even without any structure others argue that connectivity sparseness in the reservoir is crucial. Sparse connectivity in the reservoir lets it decompose into many loosely coupled sub-networks. This way a reservoir is created with rich dynamics, that allows multiple independent internal representations and therefor benefits performance [16, 8, 11]. This decoupling can even be pushed to an extreme of using multiple small reservoirs in parallel instead of just one, an approach developed by Xue et al. [27] called decoupled echo state networks (DESN).

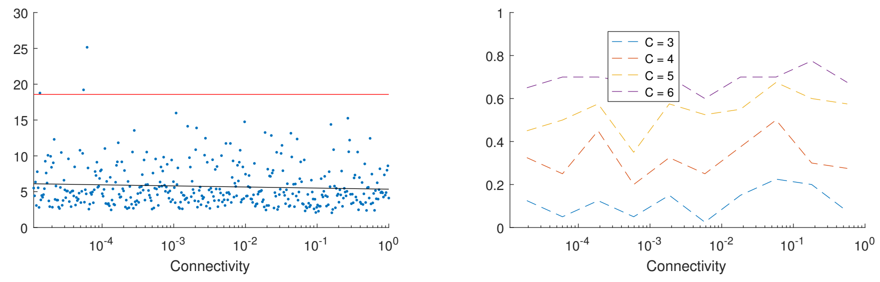
In approaches using only one sparsely connected reservoir appropriate connectivity sparseness is said to be between 1% and 20%, i.e. each reservoir neuron is connected by chance to approximately 1% to 20% of the other reservoir neurons. However, some findings implicate connectivity in randomly generated reservoirs not to influence performance at all [12]. We investigated how connectivity sparseness influences RNN performance on visual sensations.

In our initial experiments, for each clip we trained 400 randomly constructed RNNs while increasing the reservoir connectivity p from very sparse $p = (\frac{1}{300})^2$, to full $p = 1$ in logarithmic steps. This comes down to going from approximately 1 recurrent connection in total in the reservoir to all-to-all reservoir connectivity. The results are scattered in figure 3.3 together with plots of the probabilities of achieving a certain performance. Least-squares lines are added to the scatter plots to indicate the global trend. For clips 1, 2 and 3 the found correlations between $log(p)$ and the test MEE are respectively $r = -0.015$, $r = -0.072$ and $r = 0.025$. The corresponding slopes of the least-squares lines are respectively

Clip 1



Clip 2



Clip 3

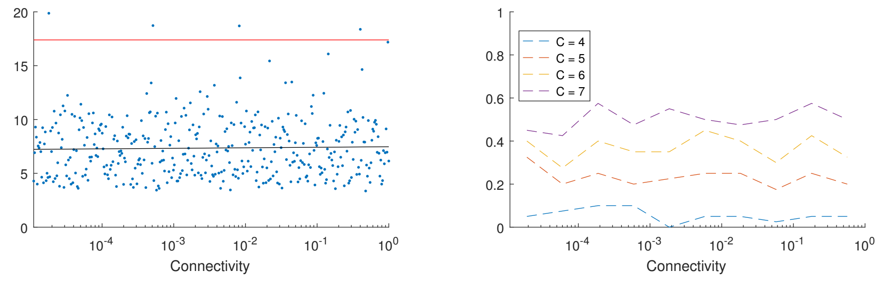


Figure 3.3: Scatter plots of RNN performances against reservoir connectivity paired with plots of the likelihood of achieving at least a certain performance against reservoir connectivity. The smallest C was chosen to be an integer below which at least one RNN performed. The red horizontal line indicates chance level performance.

$b = -0.07$, $b = -0.152$ and $b = 0.049$.

So it appears that p has no effect on performance. This could be due to p not having an effect, which is in accordance with [12] (but note the difference in used training procedure; Koryakin et al. [12] used a simple regression to determine the output weights, whereas we used the FORCE learning procedure). An alternative explanation could be that visual data per se does not require multiple independent representations in the reservoir. Compare for example the MSO problem[1], for which it is shown by Wierstra et al. [24] that decoupling of internal representations is a requirement, although [12] used this same problem. It may also be so that independent problem representations emerge in the reservoir regardless of connectivity sparseness, although this seems counter-intuitive, but the reservoir keeps containing a high-dimensional projection of its inputs and prior states. Because the reservoir weights are chosen randomly, the reservoir computes many non-linear combinations of input components and features. This so-called kernel property determines the performance of the linear readouts and is not influenced by connectivity sparseness [17] (recall that all weights are scaled to nullify any change in a neuron's input variance).

With our used target patterns, it may also be the case that the immense amount of feedback projections, i.e. 2304×300, prevents the emergence of independent representations inside the reservoir. Therefor we investigated the influence of feedback connectivity sparseness p_{fb} combined with differing reservoir connectivity. To this end we trained 100 randomly constructed RNNs for each of thirty logarithmically spaced values of p_{fb} from $\frac{1}{300}$ to 1, while varying p logarithmically from $(\frac{1}{300})^2$ through 1. so p_{fb} was varied from on average one feedback connection per output unit to full feedback connectivity, and the reservoir was changed from on average one recurrent projection in total to full connectivity. This was done for each target pattern video clip. The resulting probabilities at low errors in the testing phase against p and p_{fb} are depicted in figure 3.4. In this experiment the magnitude of the feedback weights was not scaled according to their multiplicity. We used the same feedback weight magnitude as e.g. [22] with only one output, as the RNNs were still able to learn. However, using B outputs instead of just one entails that the variance of the received feedback signal of each reservoir neuron is B times greater. Similarly, using $p_{fb} \cdot B$ feedback connections instead of all B possible feedback connections entails that the variance of the input to each reservoir neuron from external feedback is multiplied by a factor p_{fb} (assuming all outputs $z_i \in z$ are identically distributed. See the appendix for a derivation). Therefor we also repeated the experiment with scaling of the feedback weights, of which the results are shown in figure 3.5. Note that the reservoir weights were always scaled according to their multiplicity (see section 2.1), but that we now scaled the feedback weights according to their deficiency: they were scaled with a factor $\frac{1}{\sqrt{p_{fb}}}$, which is the inverse of the

[1]The Multiple Superimposed Oscillation (MSO) problem consists of multiple sine waves superimposed on each other while the individual frequencies of these sine waves are not multiples of each other. Thus the resulting signal's wavelength can become extremely long, making it harder to predict.

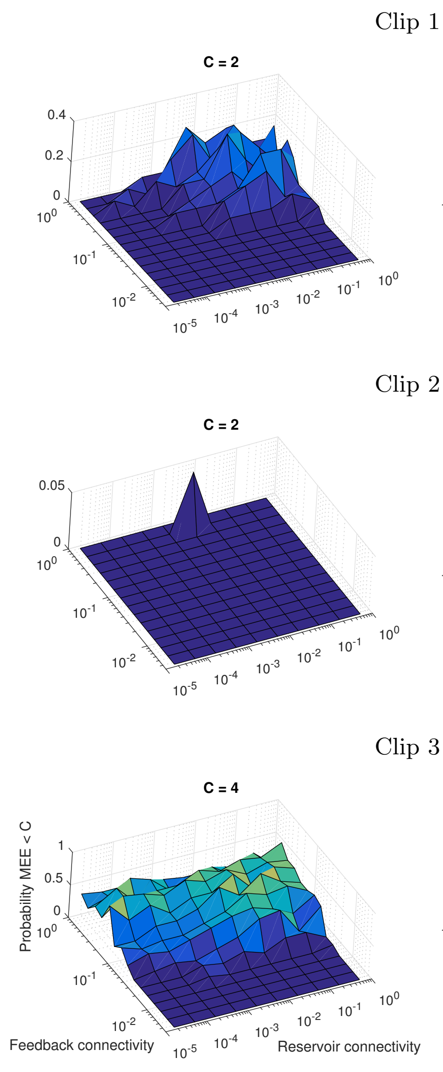


Figure 3.4: The probability of achieving a very low test MEE against different reservoir and feedback connectivity settings with unscaled weights. The used C's where chosen to be integers below which at least 1 test MEE was obtained.

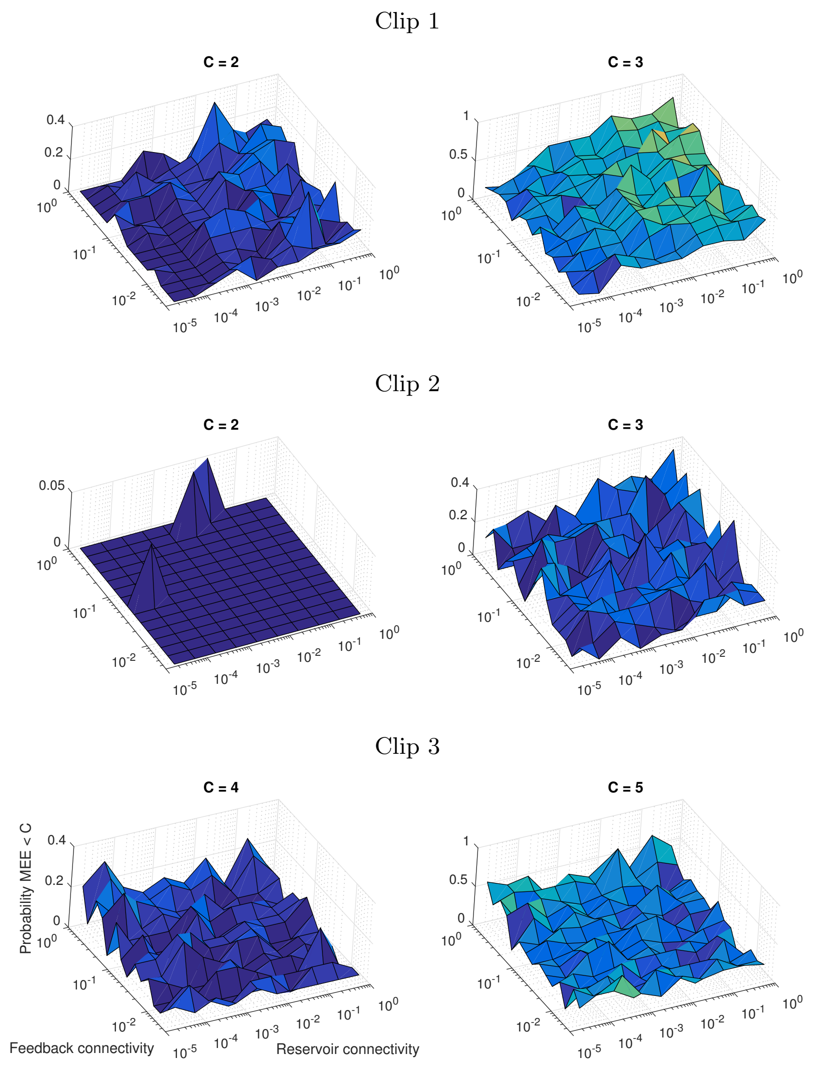


Figure 3.5: The probability of achieving a very low test MEE against different reservoir and feedback connectivity settings with not only reservoir weights but also feedback weights scaled according to their numbers. The used C's where chosen to be integers below which at least 1 test MEE was obtained.

square root of the approximate factor with which the variance in the feedback input to a reservoir neuron is reduced as the amount of feedback connections is reduced to $p_{fb} \cdot B$. So this scaling ensures that the variance of the feedback input to each reservoir neuron remains approximately the same. See the appendix for how this is derived.

The effects of feedback and reservoir connectivity sparseness is different for scaled versus unscaled feedback weights. With unscaled feedback weights a too low feedback connectivity results in a decreasing probability of a very low resulting MEE. This can be seen in figure 3.4, as the shown probability of achieving a test $MEE < 2$ quickly reaches zero for a feedback connectivity of $p_{fb} < 10^{-1}$. On the third clip no test MEE below 3 was found, but the probability of achieving a test $MEE < 4$ drops similarly as $p_{fb} < 10^{-1}$. When tolerating a slightly bigger test MEE, the effect of a drop in the probability of achieving such an MEE begins to disappear and remains only present for very low $p_{fb} < 10^{-2}$. In addition to performance loss with a too low feedback connectivity, there appears to be no advantage in using full connectivity from the results obtained with unscaled feedback. Figure 3.4 shows that the probability of achieving the depicted low test MEE's peaks before the feedback reaches full connectivity.

However, these differences in likelihood of achieving a certain performance with different connectivity sparsities are mostly due to the strength of the feedback connections, because when the feedback weights are scaled according to connectivity sparseness, these differences almost completely disappear. Now compare the previous results with the new results depicted in figure 3.5 using scaled sparse feedback. The previously seeming effect of the feedback sparseness on the likelihood of achieving a good performance is no longer visible. On the other hand, the previously found peaks where this likelihood was maximal, were higher than all of the newly observed probabilities. This suggests that the effect found with unscaled feedback weights was actually due to the magnitude of these weights being more appropriate when using fewer feedback connections. Such results were also found by Koryakin et al. [12], who investigated the effect of feedback weight scaling on the capabilities of echo state networks. In short summary, they found that there is an optimal output feedback weight range for an ESN of a particular size, and now we found our feedback weight scaling to be more adequate for a certain range of feedback connectivity. However, the interaction between the feedback weights and the amount of used feedback connections deserves further investigation before conclusions similar or opposite to [12] can be drawn.

3.4 Weight scaling

Using a different spectral scaling factor for the reservoir weights has several consequences for the RNN's behaviour and learning. As the largest absolute

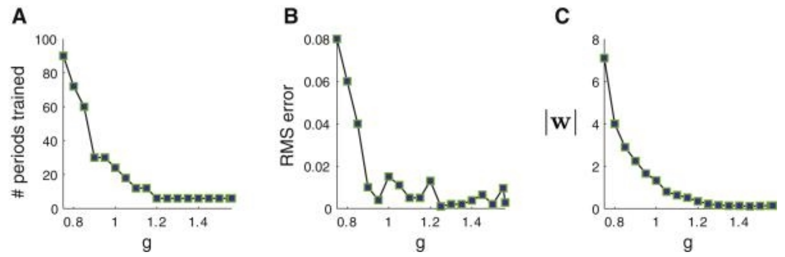


Figure 3.6: Networks with $0.75 < g < 1.56$ were trained to produce the sum of four sinusoids. Outside this range of g learning did not converge. **A**. Number of repetitions of one period of the target function required for training. **B**. RNN performance after training. **C**. Readout weight vector length $|\mathbf{w}|$ after training, which is an indication of network stability. A larger $|\mathbf{w}|$ usually corresponds to a less stable solution. Figure taken from [22]

eigenvalue, also called the spectral radius, of the reservoir weights matrix $\mathbf{J}$ is initially 1, the spectral scaling factor g directly scales the spectral radius of $\mathbf{J}$. First of all, using a spectral scaling $g \leq 1$ makes the reservoir behave with ordered dynamics. Put simply, this means that the network's internal state at a certain time point only depends on the previous so many states and inputs. In the absence of input the activity in ordered networks dies out. On the other hand, using a spectral scaling $g > 1$ enables chaotic dynamics in the reservoir. This means that activity in the reservoir will never die out by itself, and therefor the RNN's state always depends on all its previous states [20].

For some learning approaches ordered dynamics are desirable, such as in the echo state approach [8]. In other approaches chaotic dynamics are found to have benefits and spectral scaling is preferably chosen above 1. One such approach is the FORCE learning procedure. However, spectral scale must not be chosen too high, as the internal chaos will become impossible to control and make the RNN incapable of learning any task. Therefor $g \approx 1.56$ is typically chosen. Benefits of this g compared to lower gs include a greater computational capability of the RNN [2], faster convergence of FORCE learning, and the resulting RNNs are more accurate and robust to noise [22]. See figure 3.6 for experimental outcomes from [22] regarding the latter benefits. moreover, in their work they found that training an RNN to produce the sum of four sinusoids failed if not $0.75 < g < 1.56$.

First we investigated if our typical RNNs are bound by the same limits $0.75 < g < 1.56$ for learning to succeed. But, in contrast to Sussillo and Abbott [22], we did not encounter decreasing RNN performance for any $0 < g \leq 4$. We suspected this was due to the difference in output feedback. Where RNNs in [22] contained only one output unit with a feedback connection to each reservoir neuron, our targets require more outputs and thus add more feedback connections to the RNN (compare our $N \times B$ versus their $N \times 1$ feedback weights matrix). If these many feedback connections are not scaled according to their multiplicity, then the feedback pathway will exert more control on the reservoir's activity as more feedback connections are added. Therefor we also investigated the interaction between different reservoir weight scalings and different feedback weight scalings, in section 3.4.1

Next we investigated if we could reproduce the findings of faster learning and greater accuracy in chaotic RNNs with video clips as targets, in section 3.4.2.

3.4.1 Reservoir & feedback weight scaling

To investigate the interaction between feedback strength and chaotic activity in the reservoir we varied the scaling g_{fb} of the feedback weights of each RNN logarithmically from $\frac{1}{2300}$ to 1 in 30 steps. At each step we trained and evaluated 100 RNNs. Of each newly constructed RNN the scaling g of the internal reservoir weights was linearly incremented with $g \in [0, 4]$. We repeated this for each target pattern video clip. The resulting performances are displayed in figure 3.7. We split the results for RNNs with ordered and chaotic activity in the reservoir.

In ordered RNNs no good performances emerge with too small feedback weights. Learning typically fails below a value of $g_{fb} \approx 10^{-3}$. However, with larger reservoir weights this lower limit of g_{fb} becomes even slightly lower than with smaller reservoir weights. This indicates that RNNs with a smaller g need to be driven by the feedback, so they require a larger g_{fb}. RNNs with a larger g allow a smaller g_{fb}, as the reservoir activity does not die out too quickly for the weaker feedback to compensate. This interaction is best visible for clips 1 and 2, but is harder to find with clip 3, indicating that the mutual compensation of g and g_{fb} is harder to find for (too) difficult target patterns. Moreover, performance increases even further with larger feedback weights, but intuitively there should be an upper limit to feedback weight strength. This, however, requires further investigation. Additionally larger reservoir weights improve performance, which can be explained by the fact that larger reservoir weights allow a longer fading memory capacity [9, 17].

In chaotic RNNs also no good performances emerge with too small feedback weights, but in contrast to ordered RNNs the chaotic ones achieve far greater errors, reflecting the importance of the feedback loop in the chaotic RNNs to be able to suppress the chaos in the reservoir. The scaling of g_{fb} determines the limit of g above which learning starts to fail. This can be seen in figure 3.7. The limit of g becomes higher with stronger feedback. If the output feedback can suppress the chaos in the reservoir, the RNN is able to learn a task, and larger feedback weights benefit this chaos suppression [22]. Additionally, the test errors appear to go up very close to $g_{fb} = 1$ compared to values slightly below, suggesting that a too large value for g_{fb} is counterproductive, but this requires further investigation. However, [12] did show that g_{fb} must not be too large and has an optimal range regarding achievable performance in echo state

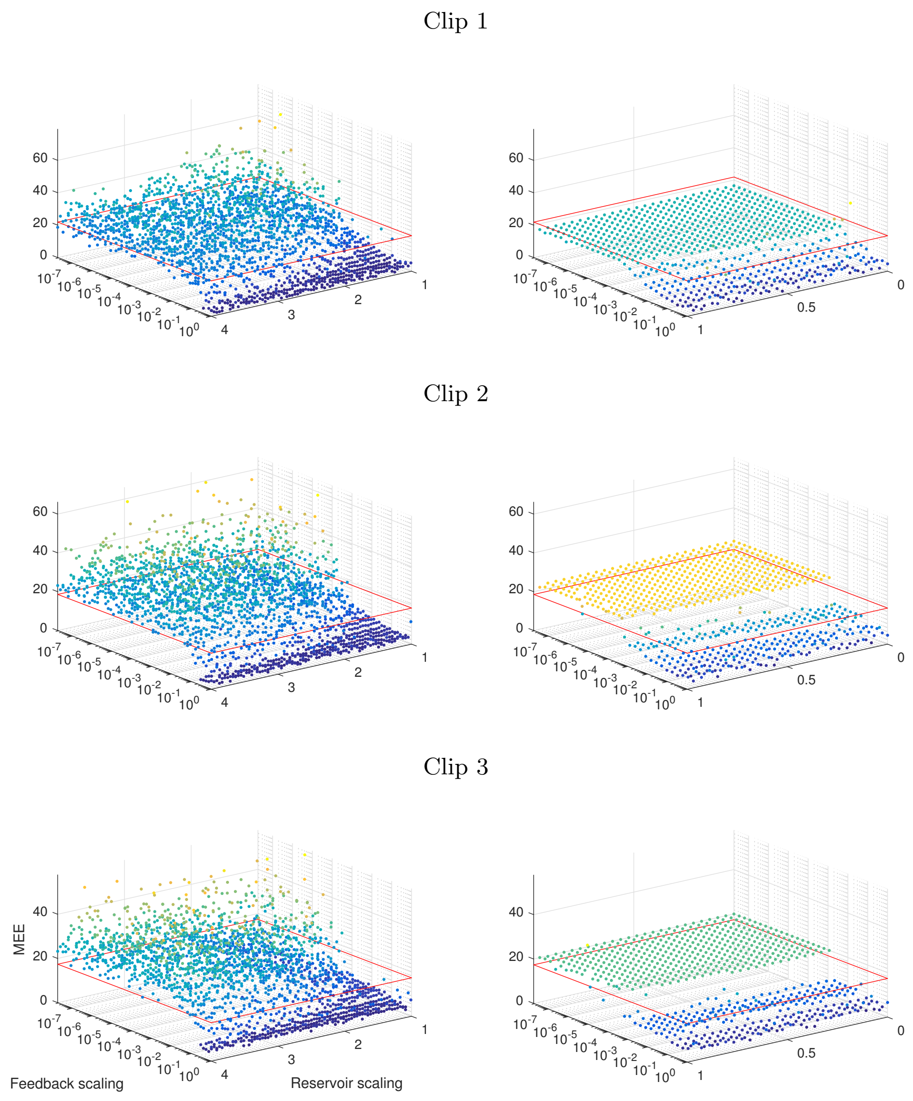


Figure 3.7: Test MEE against the reservoir's spectral scaling and the feedback weight scaling. The red bands indicate chance level performance on the particular clips. The individual dots are coloured according to their relative height. The left plots show test performances for chaotic RNNs ($g > 1$), and the right plots show performances for ordered RNNs ($g \leq 1$).

networks.

So in short summary, for $g \leq 1$ our experimental outcomes show how too small feedback weights disable learning, as the ordered dynamics in the reservoir are not driven by a sufficiently strong output feedback signal. As the internal reservoir weights become stronger, weaker feedback weights are tolerable, because the reservoir becomes more driven by itself. Also do larger reservoir weights allow a longer fading memory capacity [9, 17]. For $g > 1$ this is reversed: as g grows, stronger feedback weights are required to keep the RNN capable of learning. As long as the output feedback is able to suppress the chaos in the reservoir, the RNN is capable of learning a task [22].

3.4.2 Reservoir weight scaling & amount of training passes

To investigate the effect of reservoir weight scaling g on the performance achieved after a certain amount of training passes n, we trained 200 RNNs on each clip while linearly incrementing g in each newly constructed RNN from 0 through 4. We did this for each value of n from 1 through 15. The resulting probabilities for a certain good performance in the testing phase are shown against g and n in figure 3.8. So we did not count the amount of training passes required to achieve such a performance, but instead determined the chance of achieving below a certain test MEE to deduce whether we would require less training passes if we use a greater g. The plots show that the best performances are more likely to be achieved with the same amount of training passes if the RNNs have a greater g. With lower gs the same success rate is achieved with many training passes as with higher gs and just a few training passes. This points in the direction of the same conclusions as [22], that fewer training passes are required if a greater g is used, but we can see by the plots that this only holds for the lowest tolerance for MEE. On the third clip no RNN performed with $MEE < 3$, and in the corresponding plots an effect of g is absent, just like the effect becomes vaguer on the other clips for the increased tolerance of $MEE < 3$.

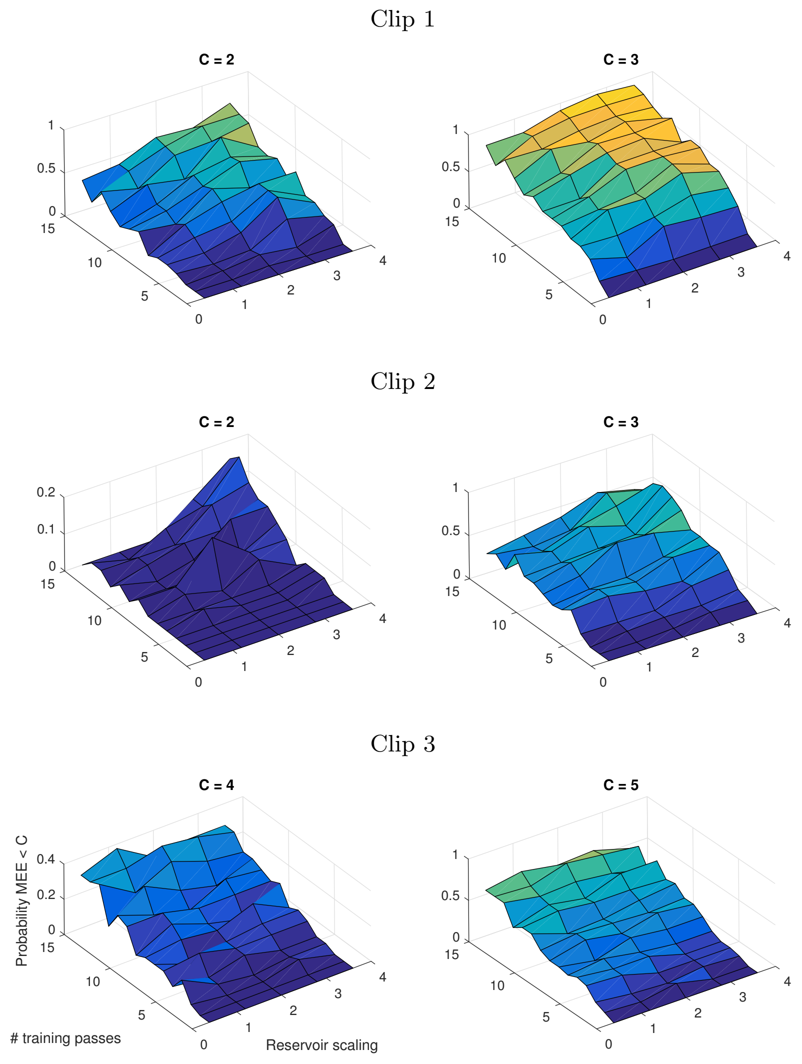


Figure 3.8: Probability that an MEE below a value C is obtained against the reservoir's spectral scaling and the amount of training passes used on a clip. The surfaces are coloured according to their height.

Chapter 4

Conclusions & discussion

In summary we found that the RNNs capability increases with its reservoir size. There is no disadvantage to using a bigger reservoir size, except the additional computational load in simulations and the additional required space and energy in real brains. This contrasts the findings of [12], who found there is an optimal reservoir size beyond which error will again increase, but this difference is likely due to the difference in used training procedure. Also, RNN performance is not affected by the used connectivity sparseness, as long as there is a certain degree of recurrence to allow internal dynamics to emerge and as long as the weights are scaled according to the amount of connections used, which is also found by [12].

The weight scaling itself, however, was found to be rather important. In ordered RNNs the scaling of the reservoir and feedback weights determines which set of weights drives the RNN's activity. If the reservoir weights are chosen to be small, then the feedback weights should be chosen to be relatively larger to stimulate activity in the reservoir. If the reservoir weight matrix's spectral radius approaches 1, the feedback weights may be chosen to be smaller. However, using a g and a g_{fb} close to 1 seems to be optimal for ordered RNNs. In chaotic RNNs the scaling of the feedback weights determines the allowable chaos in the reservoir, as larger feedback weights are better able to suppress the chaos in the reservoir and thus allow a greater spectral radius of the reservoir weights matrix. A larger spectral radius is also beneficial if it is not too large. If it is too large, the chaos in the reservoir cannot be suppressed by the feedback pathway, not even during learning, and learning will not converge. However, below its limit greater reservoir weight scaling speeds up learning and improves RNN accuracy and stability. This was also found by [22]. These benefits are mostly visible when aiming for the best performances. If the tolerated error is higher, then these benefits of spectral scaling disappear.

4.1 Shortcomings

As performance measure we chose to use the Mean Euclidean error of network output z, equation 2.7. A more popular error measure is the normalised root mean squared error (NRMSE), which is defined as

$$NRMSE = \sqrt{\frac{\sum_{f=1}^{F} e_f^\top e_f}{F \cdot \sigma^2}} \tag{4.1}$$

where F is the target pattern length, $e_f = y_f - z_f$ and σ the variance of y. However, there are several advantages to using MEE over $NRMSE$, such as a more natural interpretation and allowing better balancing of large errors by small errors [13].

To determine how hard it would be to teach a video clip to an RNN we used the total euclidean length of the clip's trajectory in its highly dimensional pixel space. However, different complexity measures are also possible. For example, the amount of variance explained by a clip's first n principal components could, in retrospect, be a promising predictor of RNN performance and deserves investigation. The reservoir projects past outputs onto a highly dimensional space to be translated to a pixel intensity by the readouts [9], and the more dimensions in this new space, which grows with reservoir size, the more variance in the fed back signal can be captured in this space. This will aid the output units in learning to compute the appropriate pixel intensity.

A different possible complexity measure might be permutation entropy as described by Bandt and Pompe [1]. This measure simplifies subsequent transitions of n steps in a time series by labelling each point in the transition with their relative magnitude (e.g. 11-3-7 becomes 3-1-2, with $n = 3$). Then the entropy is computed of these labels among labels generated using different values for n. For a complete description of permutation entropy, see [1]. Additionally, this measure behaves similar to Lyapunov exponents. A great advantage over the used measures is its robustness to the presence of noise, which is virtually always present in the used data set. Like the used measures the permutation entropy is easy and fast to compute.

4.2 Topics for future research

Video clips were selected based on containing the desired characteristics of scene transitions, camera angle transitions, camera panning and movements of entities. Even more characteristics of visual data can be conceived, such as changes in lighting, zooming in or the camera rotating about its longitudinal axis. Such characteristics were in this project meant to increase a clip's complexity to make it harder to be learnt. An interesting research topic would also be how well RNN's can represent each of the actual characteristics, e.g. could one represent the zooming itself? A proposed procedure is to train RNNs on the chosen visual change and test whether the same change is generated by the RNN on different visual scenes. For example an RNN can be trained on zooming in on a picture of a dinosaur, or even a more general simple grid, after which it would be interesting to see what happens if it is displayed a picture of something else.

Jaeger [9] and Millea [19] have shown that a linear activation function for all neurons works best for improving an RNN's short term memory capacity or even general computational capacity, at least when trained using the ESN approach (i.e. determining the output weights through regression). For a sigmoid activation function such as the $tanh(\cdot)$ an ESN works best if the network units operate on the almost linear part of the $tanh(\cdot)$. This is, however, biologically less realistic and not shown to improve performance for models trained using repetitive synaptic modification methods such as FORCE [22] or reward-modulated Hebbian learning [7]. These would be interesting research topics, i.e. to compare computational capacity of RNNs with different activation functions, when trained with repetitive synaptic modification methods. In addition to this, we have used the $tanh(\cdot)$ as activation function for its greater biological plausibility for the reservoir units, but not for the readout units due to how they desirably function. This has as a consequence that the signal fed back into the reservoir does not come from a biologically plausible source. [22] propose several network architectures that do allow more realistic feedback to the reservoir. These architectures do not feed readout unit activity back into the reservoir and separate the feedback pathway from the readout pathway. This is done by either using an added feedback reservoir, or by not using an external feedback pathway at all but instead relying on the reservoir's internal recurrence for providing internal feedback. It would be interesting to compare the computational capabilities and learning efficacy of such different architectures.

We used output feedback weights uniformly randomly drawn from the open interval $(-1, 1)$. [12] found that performance increases for an RNN with one output unit if the output feedback weight is chosen between 10^{-2} and 10^{-8}, that performance remains approximately the same for an output feedback weight between 10^{-8} and 10^{-14} and that performance again decreases below 10^{-14}. More importantly, they showed that for a larger reservoir size a larger output feedback weight is most suitable. We did not investigate this for the used network and output sizes, but our findings suggest the possibility that there is an optimal weight scaling. This was found in our investigations regarding feedback connectivity sparseness where the feedback weights were not scaled according to their numbers, as well as in our investigations regarding feedback weight scaling itself, where test error seemed to increase close to our upper bound for the feedback weight scaling. However, it remains a topic for future research to investigate how different choices of feedback weight scale outside our used range affect performance with the FORCE training procedure. In such research their effect on the suppression of chaos in the reservoir should be kept in mind, such that any potential effects can not be due to the present chaos. For example, in [12] reservoirs with ordered dynamics ($g \leq 1$) were used. Also should be kept in mind the amount of used feedback connections, such that a reservoir neuron's input variance from feedback does not depend on the amount of received input connections.

Lastly, we used the same value for $\alpha = 1$, the regularisation constant for RLS. However, [22] point out that α should be adjusted to the particular pattern being learnt, keeping in mind the constraint $\alpha \ll N$. We did not investigate what might be the proper choice for α for the used target patterns, and it remains a topic of future research to evaluate the influence of α on learning success on the used data set of video clips.

Bibliography

[1] Bandt, C. and Pompe, B. (2002). Permutation entropy: A natural complexity measure for time series. *Phys. Rev. Lett.*, 88:174102.

[2] Bertschinger, N. and Natschläger, T. (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–1436.

[3] French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128 – 135.

[4] Geman, S. (1986). The Spectral Radius of Large Random Matrices. *The Annals of Probability*, 14(4):1318–1328.

[5] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

[6] Haykin, S. (2014). *Adaptive Filter Theory Fifth Edition.*

[7] Hoerzer, G. M., Legenstein, R., and Maass, W. (2014). Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cerebral Cortex*, 24(3):677–690.

[8] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34.

[9] Jaeger, H. (2002a). Short term memory in echo state networks. *GMD Report 152.*

[10] Jaeger, H. (2002b). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach.* GMD-Forschungszentrum Informationstechnik.

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

[12] Koryakin, D., Lohmann, J., and Butz, M. V. (2012). Balanced echo state

networks. *Neural networks : the official journal of the International Neural Network Society*, 36:35–45.

[13] Li, X. R. and Zhao, Z. (2001). Measures of performance for evaluation of estimators and filters. In *International Symposium on Optical Science and Technology*, pages 530–541. International Society for Optics and Photonics.

[14] Lukoševičius, M. (2012). A practical guide to applying echo state networks. *Neural Networks: Tricks of the Trade, Reloaded*, pages 659–686.

[15] Lukoševičius, M. and Jaeger, H. (2009). Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3(3):127–149.

[16] Maass, W., Natschläger, T., and Markram, H. (2002). Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560.

[17] Maass, W., Natschläger, T., and Markram, H. (2004). Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology Paris*, 98(4-6 SPEC. ISS.):315–330.

[18] Marszalek, M., Laptev, I., and Schmid, C. (2009). Actions in context. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2929–2936. IEEE.

[19] Millea, A. (2014). *Explorations in Echo State Networks*. PhD thesis, University of Groningen.

[20] Sompolinsky, H., Crisanti, A., and Sommers, H. (1988). Chaos in random neural networks. *Physical Review Letters*, 61(3):259.

[21] Song, Q. and Feng, Z. (2010). Effects of connectivity structure of complex echo state network on its prediction performance for nonlinear time series. *Neurocomputing*, 73(10-12):2177–2185.

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

[23] Sussillo, D. and Barak, O. (2013). Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–49.

[24] Wierstra, D., Gomez, F. J., and Schmidhuber, J. (2005). Modeling systems with internal state using evolino. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1795–1802. ACM.

[25] Williams, R. J. and Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Back-propagation: Theory, architectures and applications*, pages 433–486.

[26] Wood, P. M. (2012). Universality and the circular law for sparse random matrices. *Annals of Applied Probability*, 22(3):1266–1300.

[27] Xue, Y., Yang, L., and Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376.

Appendix

Calibrating variance in neural input from recurrence

A problem with a changing amount of inputs to a neuron, is that the total input's variance increases with the amount of inputs. A greater input variance results in the neurons activation to be easier saturated, i.e. the neuron's activation x is mostly of a value so distant from zero that its activation function $tanh(x)$ has almost no more slope. However, the RNN functions best if its neurons operate within the region where their activation function are approximately linear [9]. Therefore the variance in a neuron's input is usually normalised to be independent of the amount of inputs to the neuron by scaling the weights accordingly. Consider any neuron's input as $w^\top f(x)$ where $w = (w_1, \ldots, w_N)$ is the input weights vector, $x = (x_1, \ldots, x_N)$ the activations of the sending neurons and f the activation function. Now we can derive from the variance of a neuron's input $Var(w^\top f(x))$

$$\begin{aligned}
Var(w^\top f(x)) &= Var(\sum_i^N w_i f(x_i)) \\
&= \sum_i^N Var(w_i f(x_i)) \\
&= \sum_i^N [E(w_i)]^2 Var(f(x_i)) + [E(f(x_i))]^2 Var(w_i) + Var(w_i)Var(f(x_i)) \\
&= \sum_i^N Var(w_i)Var(f(x_i)) \\
&= N \cdot Var(w)Var(f(x))
\end{aligned} \tag{1}$$

We assumed that $w_i f(x_i)$ and $w_j f(x_j)$ $(i \neq j)$ are uncorrelated in the first step. In the second step we used the fact that w_i and $f(x_i)$ are independent. In step 3 we used $E(f(x_i)) = E(w_i) = 0$, as we draw all weights from distributions with zero mean, and we assumed for simplicity $E(f(x_i)) = 0$, which is valid for our activation function $f = tanh$, but this does not necessarily hold. In the last step we assumed all w_i and $f(x_i)$ are identically distributed. Now if we pick w such that $Var(w) = \frac{1}{N}$, then the variance in the input to a neuron will no longer depend on the amount of inputs, but will only depend on the variance in the pre-synaptic output, according to the derivation above. Since $Var(\sigma X) = \sigma^2 Var(X)$ we can simply draw w from a distribution with unit variance and scale it with $\frac{1}{\sqrt{N}}$.

Calibrating variance in neural input from output feedback

In section 3.3 we discuss the scaling of the feedback weights according to feedback connectivity sparseness, i.e. the feedback weights w'_{fb} were scaled by a factor $\frac{1}{\sqrt{p_{fb}}}$ where p_{fb} is the proportion of feedback connections that are non-zero. For this derivation we used the derivation in the previous section of $Var(w^\top f(x)) = N \cdot Var(w)Var(f(x))$, but now we do not consider the input to a reservoir neuron from N other neurons, but from $B \cdot p_{fb}$ output units, using the weights vector w'_{fb} and pre-synaptic activations $z = (z_1, \dots, z_B)$. As the activation function of the output units is the identity function $f(z) = z$. For the variance in the output feedback to correspond to the other experiments, i.e. to be independent of the feedback sparseness, we require that

$$B \cdot Var(w_{fb})Var(z) = B \cdot p_{fb} \cdot Var(w'_{fb})Var(z), \tag{2}$$

where w_{fb} is the output feedback weights vector with only non-zero entries (so p_{fb} would be 1), as used in the other experiments. From this equation we can derive

$$\begin{aligned} Var(w_{fb}) &= p_{fb} \cdot Var(w'_{fb}) \\ \frac{1}{p_{fb}} Var(w_{fb}) &= Var(w'_{fb}) \end{aligned} \tag{3}$$

So for the variance in the reservoir input from the output feedback loop to be independent of feedback sparseness, the variance in the sparse w'_{fb} should be $\frac{1}{p_{fb}}$ times the variance of the full w_{fb}. In other words, w'_{fb} can be drawn from the same distribution as w_{fb}, but has to be scaled by $\frac{1}{\sqrt{p_{fb}}}$ since $\sigma^2 Var(X) = Var(\sigma X)$.

In the previous section we mentioned that the expected output of a neuron $E(f(x_i))$ is not necessarily zero. This is especially true for our output units, assuming the outputs are very close to the target function which was scaled to lie between zero and one. So $0 \leq E(z) \leq 1$. Here we assume for simplicity all z_i

are identically distributed. With this the derivation from the previous section for the feedback loop becomes (we write w instead of w_{fb} for readability)

$$\begin{aligned} Var(w^\top z) &= Var(\sum_{i}^{B \cdot p_{fb}} w_i z_i) \\ &= \sum_{i}^{B \cdot p_{fb}} Var(w_i z_i) \\ &= \sum_{i}^{B \cdot p_{fb}} [E(w_i)]^2 Var(z_i) + [E(z_i)]^2 Var(w_i) + Var(w_i) Var(z_i) \\ &= \sum_{i}^{B \cdot p_{fb}} [E(z_i)]^2 Var(w_i) + Var(w_i) Var(z_i) \\ &= B \cdot p_{fb} \cdot Var(w) \cdot ([E(z)]^2 + Var(z)). \end{aligned} \tag{4}$$

From this derivation it can be seen that scaling the feedback weights vector as aforementioned will make the variance of $w_{fb}^\top z$ still only depend on z, since the corresponding adaptation of equation 2

$$B \cdot Var(w_{fb}) \cdot ([E(z)]^2 + Var(z)) = B \cdot p_{fb} \cdot Var(w'_{fb}) \cdot ([E(z)]^2 + Var(z)), \tag{5}$$

can also be derived to derivation 3.