Interpretation of Deep Neural Networks as Dynamic Systems

A dissertation submitted by:

Eduardo Sánchez Karhunen

Supervisors:

Dr. Miguel Angel Gutiérrez Naranjo, University of Seville

Dr. José Francisco Quesada Moreno, University of Seville

in partial fulfilment of the requirements for the degree of Máster Universitario en Matemáticas.

University of Seville

December, 2020

Abstract

During last years the application of deep learning techniques has changed our lives from banking operations to medical diagnosis techniques. Unfortunately the ancient stereotype of neural networks as black boxes remains almost unaltered. We need models able to shed light on the endless list of unanswered questions about deep learning but our society has clearly tipped the balanced in favour of gaining accuracy of models. With everyday passing, the lack of a solid mathematical framework is becoming a greater problem.

In this thesis a research line in the opposite direction is presented: the interpretation of Recurrent Neural Networks as discrete time dynamical systems. This idea offers a completely new point of view for an understanding of neural networks. This "simple fact" equips us with the huge machinery of this branch of mathematics. Concepts such as state spaces, orbits, limit sets and attractors take on meaning in a new scenario.

Instead of a simple enumeration of the implications of this new perspective we have selected a dual approach. First, highlighting the connections between recurrent networks and dynamical systems. Secondly, these ideas have been applied to a simple but illustrative toy case. A sequence of type $a^n b^n$ is used as input to a RNN that tries to predict the next symbol of the sequence. First of all, different dimensionality reduction techniques are explored in order to obtain interpretable 2D/3D plots. Using these graphs the state space of the network after training is analysed. The transitions between states are made explicit and the orbits deformations during network training can be observed. Using different initial seeds the existence of line attractors becomes evident. Hidden layer dimensionality parameter space is studied obtaining that an increment in the distance between attractors is observed as dimension increases. Finally, it is shown the impact of noise on the attractors structure.

Resumen

Durante los últimos años hemos asistido a cambios en nuestra vida cotidiana asociados al deep learning: desde las operaciones bancarias hasta el diagnóstico médico. Desgraciadamente, el estereotipo de caja negra que desde sus inicios ha acompañado a las redes neuronales apenas se ha visto alterado. Necesitamos modelos que arrojen luz a la lista interminable de preguntas sin respuesta relacionadas con el deep learning pero, como sociedad, hemos preferido centrar los esfuerzos en arañar décimas a la precisión de los modelos. De hecho, cada día que pasa, el no disponer de un marco matemático sólido es un problema mayor.

En este trabajo se muestra una de las líneas de investigación en la dirección opuesta: la interpretación de las redes neuronales recurrentes como sistemas dinámicos en tiempo discreto. Este "simple hecho" nos permite acceder a la potente maquinaria de esta rama de las matemáticas. Conceptos como espacios de estados, órbitas, conjuntos límite o atractores cobran sentido en este nuevo escenario.

En lugar de una mera exposición de las implicaciones de esta nueva perspectiva, hemos optado por una visión dual. Presentando, en primer lugar, las conexiones entre las redes recurrentes y los sistemas dinámicos. Para posteriormente ilustrar estas ideas mediante un caso de uso muy simple. Para ello, alimentamos una RNN mediante una secuencia de entrada del tipo $a^n b^n$ y la red debe predecir el siguiente símbolo en la secuencia. En primer lugar, hemos explorado diferentes técnicas de reducción de la dimensionalidad para obtener representaciones en 2D/3D del espacio de estados de la red tras el entrenamiento. Las transiciones entre estados se muestran explícitamente y se pueden observar las deformaciones de las órbitas durante el entrenamiento. Mediante la utilización de diferentes semillas podemos ver claramente la existencia de attractores unidimensionales. Asimismo, analizamos el espacio de parámetros de la dimensión de la capa oculta, mostrando como una reducción de la dimensionalidad, reduce la distancia entre los atractores. Finalmente, añadimos ruido a la señal de entrada observando cambios en la estructura de los atractores.

Contents

Chapter 1

Introduction

Last decade has witnessed huge advances in the application of deep neural networks into different fields of Artificial Intelligence (AI). First practical advances were obtained thanks to the application of convolutional architectures (CNN) to image problems (Zou et al. 2019). After that, the development of different types of recurrent networks (RNN) has successfully tackled problems related to natural language processing (NLP) (Baktha and Tripathy 2017). In parallel the field of non-supervised learning has been addressed using, among others, adversarial networks (GAN).

The above problems were mainly related to the so called non-structured data types (i.e. images, audio waves or video) while other classical machine learning techniques were considered more suitable to deal with structured-data problems (i.e. classical tabulated datasets). As a result, neural networks were initially discarded as an useful approach for this type of problems. In the last years a complete reversal of this former view has happened, brought about among other things by excellent results obtained in the analysis, treatment and forecasting of time series (Hewamalage et al. 2020). These success has been possible thanks to the reuse and combination of techniques initially design for structured-problems.

All these achievements have been made possible by deep learning (DL) and its complex architectures. Associated to these advances, a great variety of techniques have been developed for network training: drop-out, weight-decay, batch-normalization, different strategies for weight initialization or even activation functions with desirable properties. All these techniques have made possible training deep networks and augmented their generalization capabilities.

In parallel new fresh ideas as Transfer Learning have arisen, where the knowledge captured by the network during the problem resolution, can be reused as a starting point for solving problems of a different nature than the original one.

Current memory storage price is nearly free allowing a huge data availability in almost every field. On the other hand, the generalized use of GPUs and TPUs are providing computational capabilities never known before. These technical advances are all fostering the development of real life applications based on DL techniques with impact in our society.

Imagine a deep learning model used for medical diagnosis. One day the model throws a completely different treatment to the proposed by human experts. Which one should be selected? Nowadays, there is probably no doubt: we would rely on the human criteria. But what makes humans to be scared about certain machines decisions? As a good black box, neural networks are not able to argue their decisions, becoming a matter of faith based upon their excellent accumulated results. Sooner rather than later, the argument of the Minority Report movie will become true, where the foreknowledge of precogs will be substituted by deep learning models. On that day, our lives will be in the hands of machines forecasting possible crimes. Do you think is not worth investing time in mathematical foundational models of deep learning?

Hence, before any neuronal network is deployed it seems reasonable to understand the internal mechanisms used by the network to solve a problem, why the results are the ones offered by the model and not another ones or to identify if the model presents some kind of bias not obvious for the designer.

Once in this wave, deeper questions arise: why some architectures are more suitable for a problem than another, why a concrete number of layers and neurons per layer is suitable for a problem or if there exist critical parameters in the net that should not be altered to ensure the stability of the results. What if a trade-off between different parameters of the network could be identified. Perhaps modifying some elements an equivalent simplified network could be found or, even further, which high-level network parameters impact in the training speed. What if there exists a theoretical limit to the minimum amount of training data necessary to solve a problem. It could be interesting to determine if there exists a critical point where it is not worth obtaining more training data because the resulting increase of accuracy would be negligible. A countless amount of questions can be formulated.

Unfortunately, this successful history of DL techniques solving real life problems has not been accompanied by a deep comprehension of its internal mechanisms and there is a lack of solid mathematical foundations. On the contrary, practical solutions are evolving towards increasingly complex structures with dozens of layers and hundred of nodes per layer. Such a complexity level makes even more challenging to understand what is happening inside the net, how it has been able to generalise and which are the core ideas its behaviour is relying on. It is the famous idea of neural networks as black-boxes. As a result, all the previous questions remain already unanswered.

As in any other science or engineering field, there is a moment when although practical successes are accumulating, rigorous mathematical models are needed to push further the frontiers of knowledge in the field. It would allow to base conclusions obtained from empirical results, identify wrong conclusions and understand what has lead to those errors. On

the other hand new mathematical models allow to formulate questions in a new framework, to derive unexpected conclusions, to study known problems from a fresh point of view or even to study unexplored related problems.

This Master Thesis is an alligate against a narrow vision of neural networks and deep learning: many times based on hypes that distract the investigation from core ideas. We have lost a primitive idea that impregnated neural networks that year after year has been buried under tons of excellent practical results. It is the perspective of neural networks as dynamical systems.

This work is structured as follows, Chapter 1 presents a general justification of the interest of the ideas included in this work. In Chapter 2 it is shown that since their origins connectionist models have a "dynamical flavour" around them. Also a brief description of the main milestones in the history of neural networks are presented and how they have led to the classical feed-forward networks. Chapter 3 highlights the limitation of these feed-forward networks to tackle problems when time is involved. The evolution of RNNs until its current situation is presented and the most common conventions for representing RNNs are described. The mathematical formulation of RNNs obtained in Chapter 4 allows us to interpret this type of networks as discrete time dynamical systems enabling us to analyse them under a completely new perspective. The ideas exposed in the previous chapter are only valid for autonomous systems but real-life systems are input driven non-autonomous systems much more difficult to analyse. Consequences of this issue in RNN training are exposed in Chapter 5 and some recent works trying to avoid this non-autonomous consequences are also presented. In Chapter 6 these ideas are applied to a simple but illustrative toy problem where a RNN is fed with a sequence of type $a^n b^n$ and the net is trained to predict the next symbol in the sequence. The dimensionality of the network makes the visual interpretation of the phenomenon difficult. Thus, in Chapter 7 dimensionality reduction techniques are analysed and Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) are selected as suitable for our problem. These projections in a 2D/3D space are used in Chapter 8 to plot the orbits in the state space of the trained network. This idea is extended and the intermediate state spaces captured during network training are composed to show how orbits suffer deformations while the optimization process is acting on the network weights. A core idea in dynamical systems is the existence of different kind of attractors. In Chapter 9 orbits obtained using different seeds are plotted and two clear regions appear acting as line attractors. The hidden dimension role in attractors topology is analysed, obtaining that it controls the distance between line attractors. The higher the network dimensionality, the greater the distance between the attractors. Noise addition to the input sequence is considered in Chapter 10 giving rise to spreader attractors and making network predictions more difficult. Finally in Chapter 11 some conclusions and further lines

of investigation are presented.

All the results presented in this work have been performed using a custom library developed from scratch using Python3 as programming language. As the main purpose of this thesis is to present a general overview of the dynamical system theory applied to RNNs, no source code descriptions have been included. However, some footnotes are given during the dissertation to make easier the association between library variables and the presented ideas. [1]

[1] github: https://github.com/esanchek/State_Space_Analysis

Chapter 2

Back to the roots: dynamics everywhere

2.1 Parallel Distributed Processing Framework

In the 80's the MIT PDP Research Group (Parallel Distributed Processing) proposed a new framework for modelling human cognition (perception, memory, learning and, in general, any intelligent information processing). This model was called connectionist because its central idea was that our brain is composed by a great number of elementary units (neurons) connected in a network. In this context, mental processes consist of interactions between neurons in a parallel manner rather than as mere sequential operations. Following this idea, knowledge is no longer stored in localized structures. Instead, it consists of connections between pairs of neurons distributed throughout the network. The most impressive idea that underlies behind this model is that intelligence would emerge from the interaction of large number of simple processing units. With these works, the sadly period known as AI Winter came to an end (David E. Rumelhart and James L. McClelland 1986).

These units, organized into sets called *pools*, and the weighted connections between pools, known as *projections*, constitute the network architecture within which excitatory and inhibitory signals propagate. Units receive inputs from other neurons and also may receive external inputs. Meanwhile, outputs may be propagated out of the network. Processing takes place in a single step of time and consists on propagating activation signals from units to others, weighted by the strength of the connection between each pair of units. All these weighted signals are added at the receiving input and passed through an specific activation function that provides the activation for the next processing step. Those units that can receive direct input from outside the network are called *visible* units and those that cannot are called *hidden* units. The connection weights could be adjusted as a result of some training processing.

According to this description, PDP could be thought as is a mere proto-description of a nowadays deep learning network, assimilating pools to layers and projections to the connections between layers. But PDP is a framework that generalizes them where any kind of connections are allowed: between units of different pools, neurons of the same pool and even

bidirectional connections. Therefore, it is not feasible to prefix a global learning mechanism of projection layer weights.

Applying constraints to the architecture elements, a broad range of different network operating schemas can be obtained. Two of them are analysed in the following sections:

- The competitive learning architecture implements an unsupervised learning schema where interactions among its neurons give rise to a dynamics in the system associated to changes in connection weights until an equilibrium point is reached. Kohonen maps are an special case of this structures.
- Pattern associators architectures work under a supervised learning paradigm. In this case, there exists a reference of the correct output of the network for each input pattern. The difference between the expected output and the obtained is used to adjust weights between neurons. The well known feed-forward networks are a type of pattern associators.

As we will see, in both cases there exists a dynamical behaviour associated to the network. In competitive learning the dynamics is "explicit", consisting on an adjustment of its weights until a new equilibrium is reached each time a new pattern is presented to the network. In pattern associators there exists an "implicit" dynamics associated to the learning process via backpropagation until an optimal set of weights is reached.

2.2 Competitive learning and Self-Organizing Maps (SOM)

In this structure, as shown in Fig.2.1, processing units are organized in a set of hierarchically layered competitive pools where each unit of a layer receives an input of each unit of the other layer and also can project to each unit of the next layer. The number of layers is arbitrary and units of a layer are divided into a set of non-overlapping clusters. All elements within a cluster inhibit all other neurons in the same cluster such that the more strongly any particular unit responds to an incoming stimulus, the more it shuts down the other members of its cluster. Thus, only one unit per cluster may be active achieving its maximum value while all other units are pushed to their minimum value. In this schema units are all the same except for some randomly distributed parameter which makes each of them respond slightly differently to a set of input patterns.

The learning rule distributes a fixed amount of weight associated to each unit among its input connections. If an unit responds to a particular pattern and wins the competition, each of its input lines gives a proportion of its weight that is distributed equally among the

active input lines. With this strategy, a random initialization of network weights, after some cycles the system reaches a stable equilibrium configuration i.e. the weights on average are not changing any more. Hence, configurations of active units in a layer can be thought as representations of the input signal provided by a layer to the next one. Following this idea, it can be seen as an unsupervised learning algorithm implementation on a network.

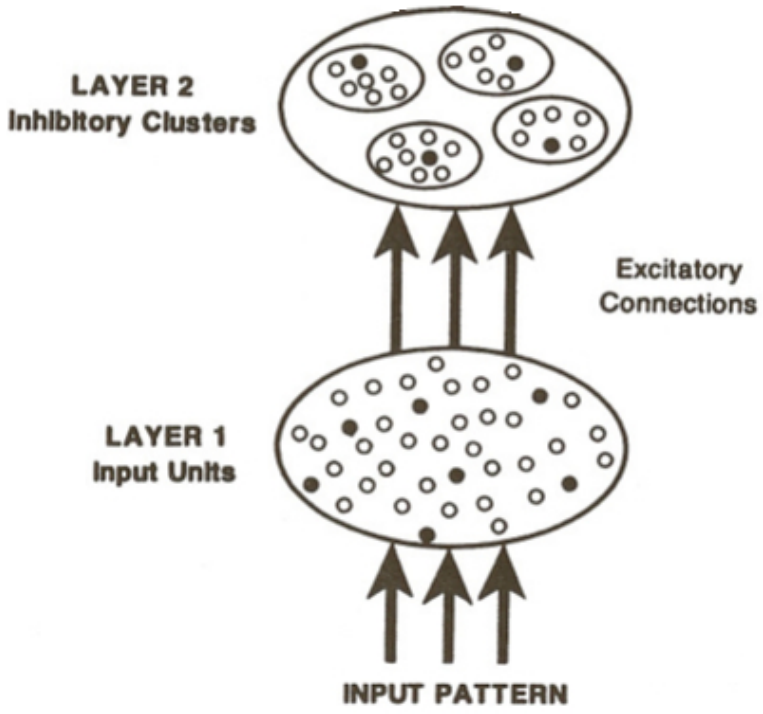


Figure 2.1: The architecture of the competitive learning mechanism. Source: David E. Rumelhart and Zipser 1985.

Different variants of this basic architecture have been proposed (Grossberg 1976; David E. Rumelhart and Zipser 1985). Among them, Kohonen Maps are a class of models born from a modification of competitive learning models when neighbourhood constraints are applied on the output units. More concretely, output units are arranged in a spatial grid (Kohonen 1982). However, instead of just moving the winning unit weights, the winning unit and its neighbours are adjusted. The amount of adjustment is determined by the distance in the grid of a given unit from the winning one. As a result, output units tend to respond to similar input pattern producing a "topology preserving" map from input space to output space.

2.3 Pattern associators

Let's consider another basic network architecture: the pattern associator. It has a set of input units connected to a set of outputs by a single layer of modifiable connections that are suitable for training with quite a simple rules, e.g. the Hebb's rule and the delta rule.

One-layer pattern associators have several suggestive properties that have made them attractive. They can generalize their responses to novel inputs that are similar to the inputs that they have been trained on. Another interesting property is that in the course of processing they can learn incrementally. The first two classical learning procedures were:

- Hebb's rule or correlational learning rule (Hebb 1949). It can be resumed in the phrase: "cells that fire together, wire together", i.e. when a neuron participates in firing another, the strength of the connection between them should be increased. That is, particular units, when active, will tend to excite other units whose activations have been correlated with them in the past. These correlations sometimes produce useful associative learning. However, often they are not sufficient to allow a network to learn even very simple associations between patterns of activation.
- Delta rule (Rosenblatt 1958). A solution for the Hebb's learning rule drawbacks is the idea of adjusting weights using some error measure of the difference between target activations and outputs obtained through learning. Because the rule is driven by differences it was called the delta rule. The training procedure is continued for several cycles through the whole set of patterns and each of these cycles is called a training epoch.

If the output units of the pattern associator, are equipped with linear threshold units we have the perceptron of Rosenblatt (Rosenblatt 1958). In this case, if the sum of all the weighted inputs is greater than a threshold the unit is turned on, otherwise it is turned off. The perceptron learning rule, i.e. the delta rule, compares this output to the desired one. If the input vector is correctly categorized, no change is made to the weights.

The learning limitations of this basic perceptron structure are well known and they are evidenced by the classical XOR problem (Minsky and Papert 1972). The proposed solution to this problem consists in the addition of more dimensions or features to the problem allowing a multilayered perceptron. The units of these multilayered networks are classified in three classes: input units, which receive the input patterns, output units associated to the targets and hidden units, not directly connected to the "environment". Nowadays, these multilayered pattern associators with non-linear activation functions are called feed-forward networks.

The next question is to know which new features ought to be learn by intermediate layers to solve each concrete problem. For that purpose an error measure must be defined (typically summed squared error for regression and cross-entropy for classification problems) and the delta rule must be readapted. Now, the weights are modified proportionally to the negative of the derivative of this error with respect to each weight. Following this mechanism each weight is moved towards its own minimum. The selected algorithm to find their minimum is the gradient descent. Hence, when all the weights have reached their minimum, the system has achieved an equilibrium. Possibly, due to the existence local minima in the error curve the problem probably is not solved exactly but it will find at least a set of weights that produces an error as small as possible.

This idea gives rise to new difficulties: how to compute the derivative of the error function with respect to any weight in the network, using gradient descent in a least squared sum function in non-linear multilayer perceptrons. The backpropagation rule came to the rescue of the problem (David E Rumelhart and James L McClelland 1987). First of all, a differentiable output function is needed not the threshold function used in the perceptron. A common choice are sigmoid family functions, i.e. logistic function and hyperbolic tangent functions. Now, the chain rule can be applied to compute the partial derivatives of the error function with respect to any particular weight.

As any gradient descent procedure, backpropagation follows the contour of an error surface, always moving downhill. In multilayer networks these surfaces can be quite complex with multiple local minima. Some of them can be global minima, in the sense that a completely errorless state is reached. Other possibility is that some residual error remains and the gradient descent can not find the best solution to he problem. For this reason, great efforts have been done to refine this gradient descent algorithm to avoid the local minima problem, three of the main proposal are the use of adjustable learning rates, momentum, weight decay and symmetry breaking.

The constant of proportionality that controls the changes in the weights is called the learning rate. To make the optimization process faster it would be desirable to set high values of this learning rate. Unfortunately, this can lead to steps that overshoot the minimum, resulting possibly in a large increase in error. To maintain learning rate in high values without leading to oscillation, a momentum term is included that determines the effect of past weight changes on the current direction of movement. This technique filters out high-frequency variations of the error surface in the weight-space.

On the other hand, the weight decay technique includes a tendency for weights to be reduced very slightly every time they are updated. It can be seen as a procedure for minimizing the total magnitude of the weights, minimizing the sum of the squares of the weights.

Finally, if all weights start with equal values and the solution is such that weights must be unequal, the network will never learns. This is because error is back-propagated through the network in proportion to the values of the weights. Hence, all hidden units change exactly in the same amount. As a result, after each update the weights maintain equal values. To avoid this problem weights are initialized with small random values to break any initial symmetry.

In this minima finding process there exists a dynamic behaviour while moving through the error function. But this dynamic is usually hidden by the use of the chain rule and the facilities given by the different deep learning libraries to perform these painful computations without suffering for users.

Chapter 3

Recurrent Neural Networks

3.1 A new approach is needed when time is involved

Feed-forward neural network are suited specially well for task related to machine perception, where the raw underlying features are not individually interpretable. This ability is attributed to their inherent capacity to learn hierarchical representations, unlike traditional machine learning methods that rely on hand-engineered features (Farabet et al. 2013). But they have two main drawbacks:

a) Despite this ability, feed-forward networks make a limited analysis of inputs relying upon the assumption of independence among the samples. After each data point is processed, the output of the system is lost when the following input sample completes its travelling throw the network. When data points are generated independently, this is not a problem but in many situations input samples are related in time or spatially. Hence, in these cases, feed-forward networks are clearly wasting valuable information.

b) The second limitation is that feed-forward networks rely on input data being vectors of fixed length. Thus, information must be chopped into chunks of a concrete size. Specially in problems with dependency between samples the selection of the most appropriate length can be problematic and even any choice may lead to loss of information e.g. in audio waves analysis.

The independence assumption fails in several tasks involved in cognition as a temporal component is present e.g. Natural Language Processing problems that commonly are studied as word sequences in the context of a whole phrase. Hence, time representation problem arises associated to the identification of temporal patterns in data. A classical solution was to represent time as an explicit variable in the same manner as any other feature could be considered in tabulated data, giving rise to an extra dimension. The main drawback of this approach is an inherent difficulty to distinguish relative temporal structures in data. Even a simple shift produces completely dissimilar representations. Consider the 3-D unitary vectors $u_1 = [1, 0, 0]$, $u_2 = [0, 1, 0]$ and $u_3 = [0, 0, 1]$. They can be interpreted as one position

shifted versions one of each other but their geometrical representation cannot be worst to capture the underlying idea of temporal correlation as they form an orthogonal base.

To overcome these problems it seems clear that a new representation paradigm for temporal data was needed. A new one where time is represented by its effect on processing. Under this point of view time is implicitly embedded in the processing procedure represented by its effect on processing, but not as an additional dimension of input data. This objective can be accomplished giving the processing system some kind of dynamic properties responsive to time sequences. In other words, the system needs some kind of memory to be able to learn temporal information contained in the sequences. One option for introducing this notion of time in the model is by the augmentation of feed-forward neural networks with the inclusion of feedback or recurrent connections. This approach gives rise to Recurrent Neural networks.

3.2 Early Recurrent Networks

Recurrent networks have roots in both cognitive modelling and supervised machine learning. Not surprisingly three of the main foundational papers were published in cognitive science and computational neuroscience journals in the 1980s.

In a famous paper, Hopfield (Hopfield 1982) wondered if the stability of memories, the construction of categories of generalization, or time-sequential memory could be emergent properties of a system composed by simple interacting neurons. The author also identifies the classes of physical systems whose spontaneous behaviour can be used as a form of general content-addressable memory. When the time evolution of a system can be described by a set of general coordinates, the instantaneous condition of the system is represented as a point in a state space and its evolution can be seen as a flow in that space. If the system flow is attracted to locally stable points, then the system is potentially useful to be used as a memory.

A set of system that fulfils the previous requirements is proposed: recurrent neural networks or feed-forward networks with strong back-couplings as he called them. They are simple neurons with step activation functions with a concrete threshold. The state of the system is identified as the vector whose components are each neuron output. These nets run according to its update rules flowing from one attractor (representing a learnt pattern) to another attractor (representing another pattern). In the training procedure an energy function associated to the net is involved and the system moves towards states of minimum energy. These Hopfield networks have been useful for recovering a stored pattern from a corrupted version and are the forerunners of Boltzmann machines and auto-encoders.

Jordan (Jordan 1986) introduced an architecture consisting of a feed-forward network with a hidden layer which is extended by new units called state units. This structure allows the network's output to be seen by the network hidden units. Hence, the network's behaviour can be influenced by its previous outputs obtaining the main goal: give the network memory. This initial idea had some structural constraints. Firstly, there was a one-to-one connection between output and hidden states. Secondly, the weights associated to these connections had fixed and unitary values.

This Jordan architecture was simplified by Elman (Elman 1990). The network input was augmented by additional inputs called *Context Units* and these context units were a one-to-one identity map of the traditional hidden units. In this way the context units remember the previous state of the network, providing it with memory. Each context unit takes as input the state of the corresponding hidden node at the previous time step. At each time cycle both the input units and the context units activate the hidden units that feed forward to activate the output units. This architecture is equivalent a RNN where each hidden node has a self-connected recurrent edge. Elman trained this network using backpropagation and showed that it can learn time dependencies.

In these early stages there was an absence of a common nomenclature or even graphical representation of the proposed recurrence as shown in Fig. 3.1:

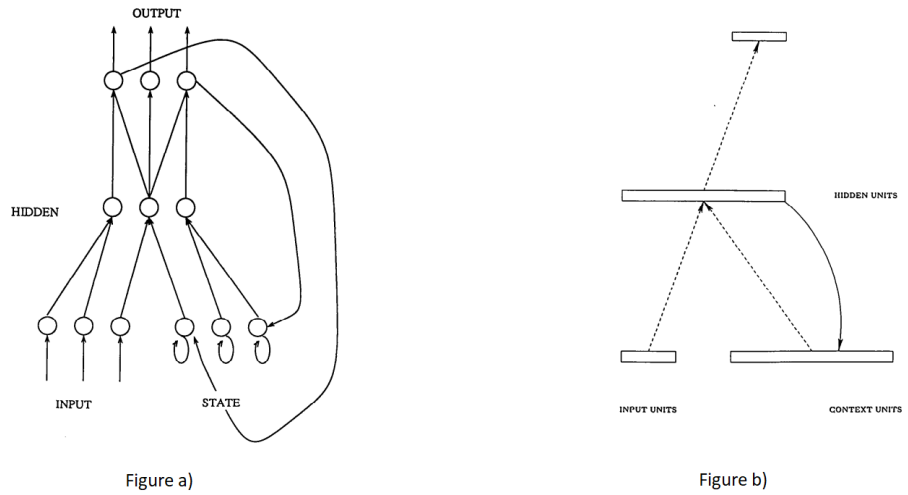


Figure 3.1: a) Original simple recurrent architecture proposed in (Jordan 1986). b) Enhanced structure suggested in (Elman 1990).

A last seminal paper proposed the application of backpropagation to recurrent networks (David E. Rumelhart and James L. McClelland 1986). The key idea is that, as Minsky

and Papert pointed out, every recurrent network can be transformed into a feed-forward network with identical behaviour (over a finite period of time). This transformation has a cost: "hardware" is duplicated many times over for the feed-forward version of the network as many times as time periods are considered. Additionally, the weights at each level of the feed-forward networks were constrained to be the same.

3.3 Representation of RNNs

In Fig.3.2 a) is shown a modern schematic representation of an RNNs with input x, hidden unit h, resulting prediction $\hat{y}$ and the recurrence from hidden layer into input is remarked explicitly with a feedback arrow. In its most basic operation mode, data runs through this loop a fixed number of time steps. Only after a certain amount of steps the output $\hat{y}$ is inspected using it as the prediction for the problem under analysis.

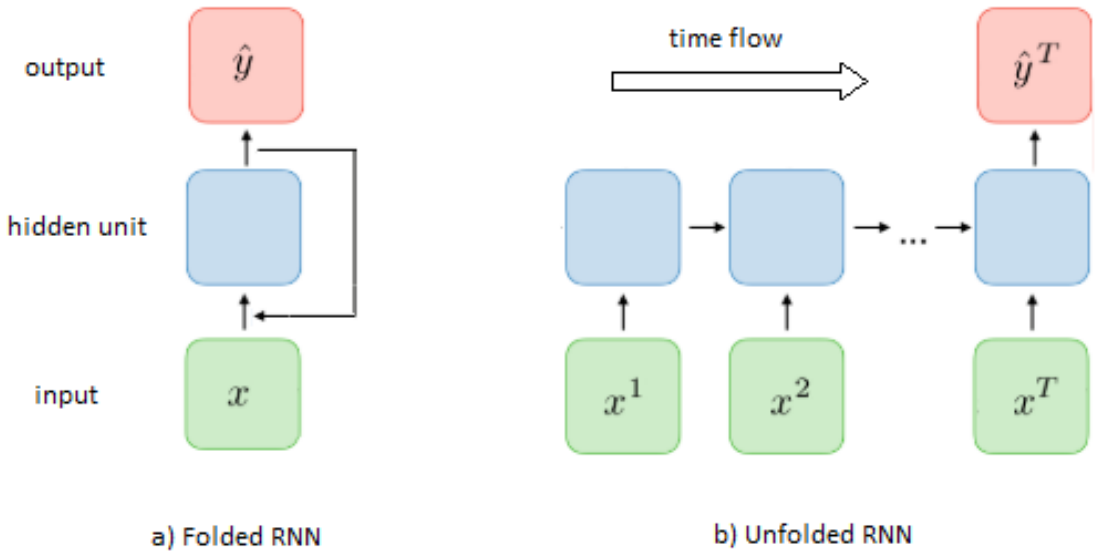


Figure 3.2: Folded and unfolded representation of a RNN

Thus, there is an important parameter in RNNs, typically called n_{steps} indicating how many past samples of the sequence are considered for the prediction. The representation used in Fig.3.2 a) is called the folded representation of the RNN because time evolution is not made explicit and n_{steps} cannot be deduced from it.

On the other hand, as seen in the previous section, a natural approximation to RNNs training is to extend the backpropagation algorithm used for feed-forward networks usually called

BPTT (BackPropagation Through Time) (David E. Rumelhart and James L. McClelland 1986) and (Werbos 1990). In RNNs the forward propagation step is straightforward but backpropagation computations of gradients of the error function with regard to weights is much trickier. To compute these partial derivatives the explicit relations between model output variables and weights are needed. But an additional element increases the technical complexity: recurrence loop is computed n_{steps} times.

To derive the explicit relations requires us to expand or unfold the computational graph of the RNN one time step at a time to obtain the dependencies. Then, based on the chain rule, backpropagation is applied to compute gradients and adjust weights to solve the problem. The result of the expansion process is called the unfolded version of the RNN as shown in Fig.3.2 b). In this representation the computational flow moves from left to right along with time course, n_{steps} is highlighted making clear that the prediction is the output value after n_{steps}.

The increasing application of RNN to concrete problems have expanded the possible operating modes of an RNN. Using the unfolded representation these modes can be classified based on the number of past samples of the sequence (number of inputs) considered to perform the operations and how many intermediate snapshots are taken from the output (number of outputs).

The operation mode in Fig.3.3 a) is called "many-to-one", multiple samples of the sequence (multiple inputs) are used to obtain a single prediction after completing T recurrent processing steps (single output). A typical application field for this mode is sentiment analysis, where all words of the sentence or paragraph are available and a score must be assigned after analysing them as a whole.

As shown in Fig. 3.3 b) nothing prevents us from inspecting what is happening to the output not only in the very final step but at intermediate computation steps from $t = 1, \ldots, T$. This mode is called "many-to-many" as multiple inputs are considered to obtain multiple outputs. This structure is useful in sequence to sequence learning problems and many different variants can be found. This type of structures is applied to machine translation where a sentence with multiple words must be translated into a second language, obtaining a new sequence of multiple words. Another classical application is the NLP problem of Name Entity Recognition.

Some applications can be even more surprising, using a simple input to generate a sequence with multiple outputs. This "one-to-many" mode is shown in Fig.3.3 c). In the context of NLP, automatic text generation is an application of this operating mode, letting the user select a word as input to the model that responds generating a sentence or even news about the selected theme.

In Fig.3.3 d) the number of time steps is reduced up to a single one obtaining the "one-to-one" mode corresponding to a traditional feed-forward network. Hence, the simplest RNN mode to be analysed is the many-to-one configuration and it has been selected in posterior chapters for a practical demonstration of the dynamical system ideas presented in this work.

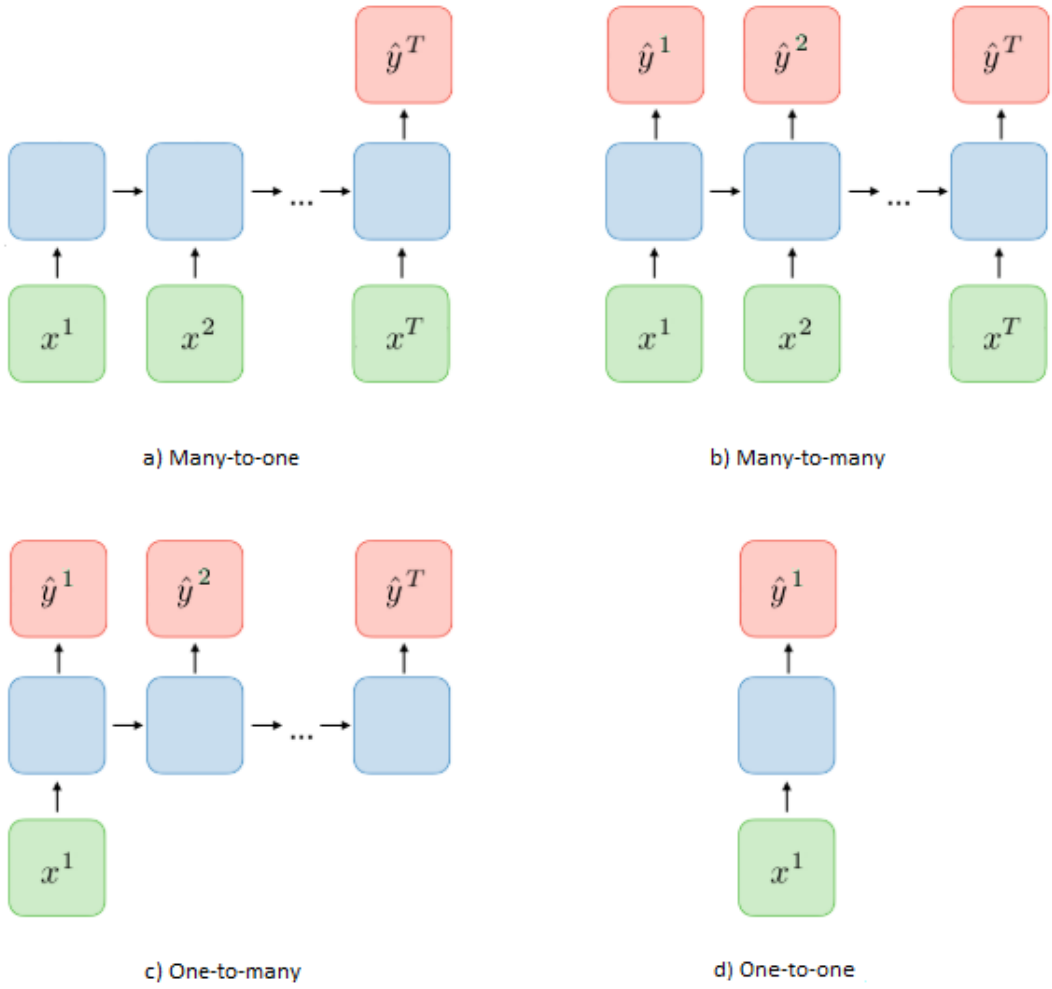


Figure 3.3: Different RNN operating modes.[1]

In the previous figures, boxes in green represent inputs or past samples of the sequence, red ones are the system output at each time step but nothing has yet been said about the blue boxes or hidden layer. These boxes are simply a feed-forward layer that has been trapped in the recurrent loop. Obviously, hidden layers in an unfolded RNN are all identical.

[1]Source: figures 3.2, 3.3 and 3.5 are modified versions of figures obtained from Stanford University webpage https://stanford.edu/ shervine/teaching/cs-230/

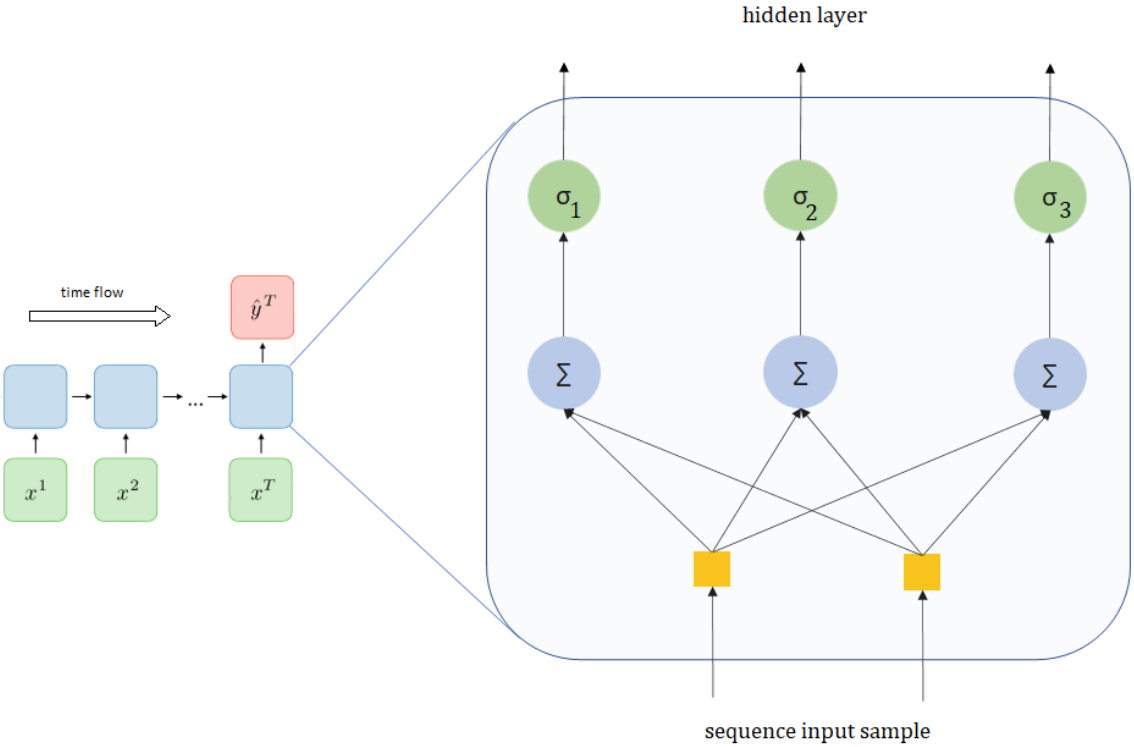


Figure 3.4: Basic RNN hidden layer

In Fig.3.4 a RNN's typical hidden layer is represented with a 2D-input sequence. The input to the layer is a sample x^i of the sequence whose dimensionality n_{input} is given by the number of dimensions of the sequence. Unidimensional case is the simplest one, e.g. when sequence under study represents the unemployment rate in a country. However, for modelling many problems multidimensional sequences are a must, e.g if GPB and unemployment rate are considered jointly.

Hence, x^i input travels through a dense layer. The number of neurons of this layer (commonly denoted as n_{hidden}) is known as the hidden layer dimensionality. Thus, the input sample is transformed from its original dimensionality n_{input} into a n_{hidden} dimensional representation. In later chapters formal relations are deduced but roughly speaking if input is called $x(t)$, and W_{in} is the input weight matrix, the output $h(t) = W_{in}x(t)$ is received by the activation functions σ_i of each neuron i. Typically, all these activation functions are identical $\sigma_1 = \ldots = \sigma_n$. Therefore, the transformed signal $h(t) = \sigma(W_{in}x(t))$ is the output of the hidden layer.

The RNN structure exposed in this section is commonly called a traditional RNN as many improvements have been proposed in the last years for the connections between cells and the hidden layer internal structure.

3.4 Training difficulties

The problem of learning with classical recurrent networks can be specially challenging mainly due to the difficulties to capture long-range dependencies (Y. Bengio, Frasconi et al. 1993). The two main problems are vanishing gradients and exploding gradients (i.e. derivative of the error function with respect to the network parameters) that appears when the network is trained using backpropagation across many time steps. In practice, this means that after a certain amount of steps, some weights of the network tend to become zero while others are given an excessive relevance.

To obtain an intuition of the problem, consider a linear simplification of an RNN where non-linearities (generated by the activation functions σ) are removed, obtaining the following equation:

$$h(t) = W_{rec}h(t-1) \tag{3.1}$$

where W_{rec} is the weight matrix of the recurrence loop and h is the hidden state of the system.

Taking into account that a RNN can be seen as the composition of a map multiple times (in this case, a linear map) Eq.3.1 can be rewritten considering an initial state $h(0)$:

$$h(t) = W_{rec}^t h(0) \tag{3.2}$$

Suppose that W_{rec} admits an spectral decomposition: $W_{rec} = Q\Lambda Q^T$ where Λ is the diagonal eigenvalues matrix and Q is the eigenvectors matrix. Hence, based on the properties of the obtained decomposition:

$$h(t) = Q\Lambda^t Q^t \; h(0) \tag{3.3}$$

The p-power of the eigenvalues diagonal matrix is again a diagonal matrix whose diagonal elements are the original eigenvalues to power p:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix} \qquad \Lambda^p = \begin{bmatrix} \lambda_1^p & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_n^p \end{bmatrix}$$

Hence, after p time steps those eigenvalues whose absolute value is larger than one will explode exponentially while those smaller than one will decrease asymptotically to zero when $p \to \infty$.

To avoid gradients explosion, truncation techniques have been applied successfully as clipping and Truncated Backpropagation Through Time (TBPTT) while fighting against gradients vanishing is much trickier and more advances were needed.

3.5 Improving structure: Modern Recurrent Networks

In the late 1990s important improvements were proposed to overcome this problem of gradients vanishing.

A first important enhancement was the proposal of networks called LSTM (Long Short-Term Memory) (Hochreiter and Schmidhuber 1997) where traditional nodes in the hidden layer are replaced by memory cells acting as units of computation. These new cells are composite units built from simpler nodes (input node, input gate, internal state, forget gate and output gate) following a specific connectivity pattern. The use of gates is a novel distinctive feature of LSTM where input gates have a multiplicative function while forget gates provide the capacity to flush the contents of the internal state, i.e. the capacity to forget.

In traditional RNNs the output of an unit is always replaced with a new value computed from the current input and the previous hidden state. On the contrary, LSTM keeps the existing content and add the new content on top of it. This additive behaviour has two advantages. First, it is easier for each unit to remember the existence of a specific feature in the input stream for a long series of steps. Secondly, this addition operation creates shortcut paths that bypass multiple temporal steps. These shortcuts allow the error to be back-propagated easily without vanishing too quickly. Thus, in practice, LSTM networks outperforms the results obtained by traditional RNN learning long-range dependencies.

A second important proposal was the introduction of Bidirectional Recurrent Networks (BRNN) (Schuster and Paliwal 1997). In this architecture information from both the future and the past are used to determine the output. In contrast with the previous networks where only past inputs are considered. For that purpose, the unique hidden layer of the previous RNNs architectures is replaced by two layers of hidden nodes.

The first layer has recurrent connections from the past time steps while the second layer receives a flipped version of the input, making use of the future time steps. Of course, if the model is running online it is not possible to have future samples and this limitation is its main drawback. However, there are many problems suitable for BRNNs e.g. part-of-speech tagging in NLP where the information about all the words in the sentence is available and can be used to increment the accuracy of word labelling.

A third proposal are GRUs (Gated Recurrent Units) (Cho et al. 2014). It is also a gated structure that can be considered a particular case of LSTM where its internal gate structure is simpler. As in LSTM, the gating units modulate the flow of information inside the unit. However, in GRUs the separated memory cells are eliminated making it easier to train the

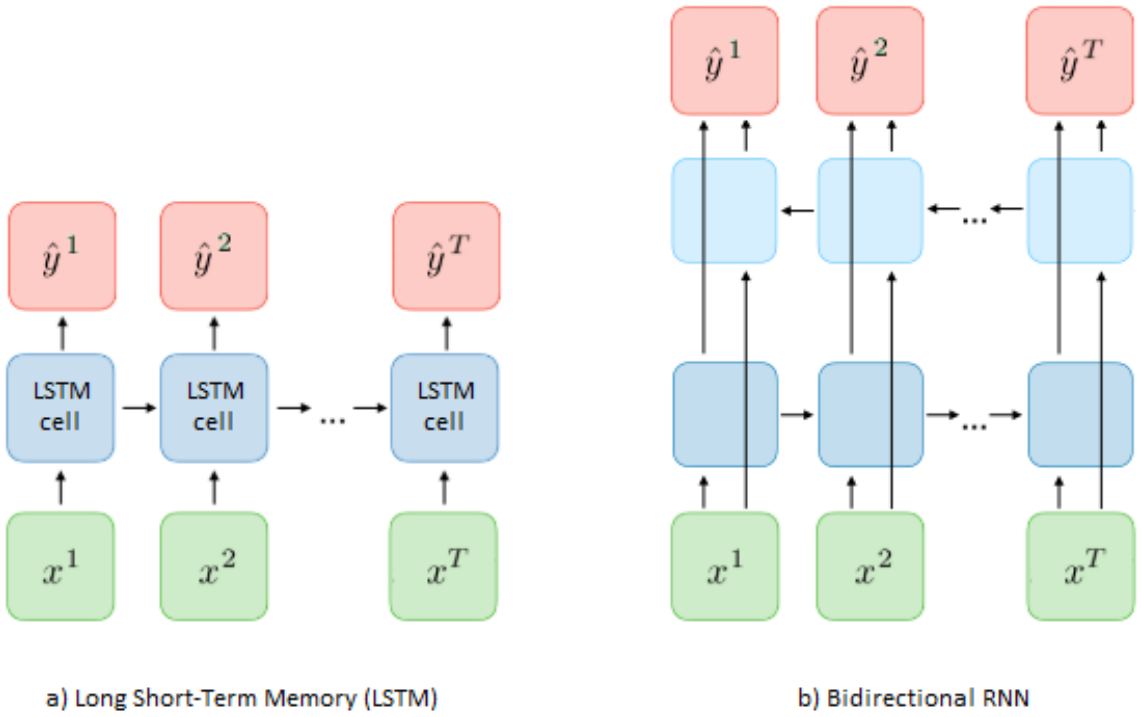


Figure 3.5: Unfolded representation of LSTM-RNN and Bidirectional RNN.

network. Performance comparisons between LSTM and GRUs are not conclusive throwing highly dependent results of the working dataset.

These techniques are represented Fig.3.5 using the same conventions as in previous figures. The key innovations introduced by each technique can be clearly seen: LSTM (and GRU) introduces modifications on the hidden units while BRNN modifies the connections regardless of what type of hidden units it is made from.

As a final remark the early RNNs where neither BRNN nor LSTM / GRU techniques are used, are usually called "vanilla RNN". In the following chapters only this type of RNN is considered.

Chapter 4

RNNs and discrete-time dynamical systems

4.1 RNNs are iterative processes

The defining feature of a RNN is its recurrent feedback. In this chapter an interesting consequence of this issue will be analysed.

Consider a typical vanilla RNN with n_{hidden} neurons and an input signal $x(t)$ of n_{input} dimensionality. The current output of each hidden layer neuron $h_i(t)$, $i = 1, \ldots, n_{hidden}$ is generated from the previous hidden state of all neurons in the layer $h_j(t-1)$, $j = 1, \ldots, n_{hidden}$ and the current external input to the system $x(t)$. A typical implementation with $n_{hidden} = 3$ and $n_{input} = 2$ is shown in Fig.4.1.

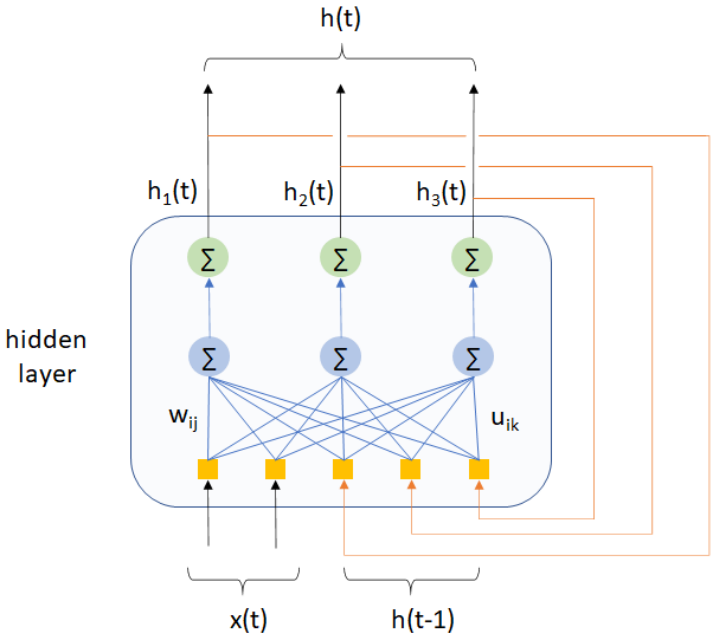


Figure 4.1: Detailed hidden layer structure of a vanilla RNN with $n_{hidden} = 3$ and $n_{input} = 2$

.

Recurrent feedback is made through weights $w_{ij},\ i = 1,\dots,n_{hidden},\ j = 1,\dots,n_{hidden}$ while the external input enters the system through weights $u_{ik},\ i = 1,\dots,n_{hidden},\ k = 1,\dots,n_{input}$ obtaining:

$$h_i(t) = \sum_{j=1}^{n_{hidden}} w_{ij} h_j(t-1) + \sum_{k=1}^{n_{input}} u_{ik} x_k(t) \tag{4.1}$$

Finally, the aggregated signal in Eq. 4.1 feeds a non-linear activation function $\sigma_i : \mathbb{R} \to I \subseteq \mathbb{R}$ that takes values in a real interval I. Usually $\sigma_i(\cdot)$ takes a sigmoidal form, i.e. $tanh(x)$ with $I = [-1, 1]$ or a logistic function $1/(1 + e^{-x})$ with $I = [0, 1]$. Thus, the hidden states only take values in the hypercube $I^n \subseteq \mathbb{R}^n$.

Usually a compact formulation is considered where:

- $H = I^{n_{hidden}}$, the state space of the network.
- $h(t) = (h_1(t), \dots, h_{n_{hidden}}(t))$ with $h(t) \in I$, the state at time t.
- $x(t) = (x_1(t), \dots, x_{n_{input}}(t))$ with $x(t) \in \mathbb{R}^{n_{input}}$, the external input to the system.
- $W = \{w_{ij}\}$ with $W \in \mathbb{R}^{n_{hidden} \times n_{hidden}}$, the connection weight matrix.
- $U = \{u_{ik}\}$ with $U \in \mathbb{R}^{n_{hidden} \times n_{input}}$, the input weight matrix.
- $\sigma = (\sigma_1(\cdot), \dots, \sigma_{n_{hidden}}(\cdot))$ the network activation function. Usually $\sigma_1 = \dots = \sigma_{n_{hidden}} = \sigma$. In this case, it is said that $\sigma(\cdot)$ is applied component-wise.

obtaining:

$$h(t) = \sigma(Wh(t-1) + Ux(t)) \tag{4.2}$$

Hence, the evolution of the hidden layer activations is determined by the map: $h \mapsto \sigma(Wh + Ux)$.

4.2 RNNs as dynamical systems

Mathematically a dynamical system formalizes the concept of a deterministic process based on the evolution of some state variables in time following a deterministic evolution law. Hence, two elements are needed: a set X of all possibles values of these state variables and the law F that describes their evolution through time $t \in T$.

Classical literature considers:

$$x(t+1) = F(x(t), t) \tag{4.3}$$

as a *discrete-time dynamical system* (DT-DS). More formally a DT-DS is defined by a triple $D = (X, T, \phi)$, consisting of a *state space* X, a discrete temporal domain $T \subseteq \mathbb{Z}$ and a function $F : X \times T \to X$ that describes how the system changes in time $t \in T$.

Because dynamical systems theory is focused on the asymptotic behaviour of solutions, this definition of DT-DS has two major drawbacks.

- First, the existence of solutions for $t \to \infty$ is not guaranteed for all $x_0 \in X$. In particular, solutions may escape to infinity within finite time. Hence, the solutions have to stay in $X, \forall t \geq 0$.
- Second, it is usually demanded that F does not depend on time t explicitly and the dynamical system is said to be *autonomous*. Otherwise, a non-constant evolution law over time implies that the dynamical system and its properties could change with t. In this case the system is called *non-autonomous*.

Alternatively, modern authors define dynamical systems as a family of maps $\{\varphi^t\}_{t \in T}$ with $\varphi^t : X \to X$ where $t \in T$ and $T = Z$ which obey the semigroups equations:

- $\varphi^0 = id$
- $\varphi^{t+s} = \varphi^t \circ \varphi^s, \quad \forall s, t \in T$

The map φ^t is called *evolution operator* and maps initial states $x_0 \in X$ into same state $x(t) \in X$ at time t. With two remarks:

- The reduction of $T = \mathbb{Z}$ ensures the existence of solutions $x(t), \forall x_0 \in X$ and $t \to \infty$.
- The existence of solutions is ensured if the image of X under F is a subset of X, i.e. $F : X \to X$.

Therefore it holds $\varphi^t = F^t, \forall t \in \mathbb{N}$ and $\varphi^0 = id$. As a consequence, a DT-DS is completely defined by its time-one-map $\varphi^1 \equiv F$.

In Eq.4.2 it was shown that a RNN changes following a map of the type: $h \mapsto \sigma(Wh + Ux)$ where h is a bounded map $h : H \to H$. As seen previously, $H \subset I^n \subseteq \mathbb{R}^n$, i.e. the hidden states only take values in the hypercube I^n (n = number of neurons in the hidden layer) given by the output interval I of the activation function.

Thus, a RNN is a discrete-time dynamic system with $\varphi^1 \equiv F$ given by:

$$h(t) = F(h(t-1), t) = \sigma(Wh(t-1) + Ux(t)) \tag{4.4}$$

where $h(t) \in H$ is the state of the RNN at time t and $u(t)$ is the respective input vector. W and U are their respective weight matrices and σ denotes an element-wise application of the node output function.

In general, Eq.4.4 is non-autonomous, because input $x(t)$ depend on time. As said before, this can produce a variation of the dynamics over time. Classical dynamical system theory has been developed only for autonomous systems while the analysis of non-autonomous systems is extremely complex. Hence, authors avoid input impact making it constant, obtaining an autonomous system. Thus, in this chapter it is also assumed constant input.

4.3 Orbits, limit sets and attractors

In dynamical system theory there are two distinct but related goals: dynamics and bifurcations.

- Dynamics is concerned with the asymptotic behaviour of the network, which includes the identification and analysis of limit sets: fixed points, periodic orbits, ... and their asymptotic stability (stable, unstable and saddle). Usually, this is also called as state space analysis of the system.
- On the other hand, bifurcations is concerned with how the dynamics of the system changes as system parameters are varied. It is also commonly called as *parameter space analysis* of the system.

A first important set in state space are orbits and its intimately bounded concept of trajectory. The evolution of an initial state x_0 with time t is called trajectory of x_0 starting at time t_0. More formally, trajectory is the map $t \mapsto x(t; x_0; t_0)$ with $x(t_0) = x_0$. In continuous-time dynamical systems (CT-DS) trajectories are curves in the state space X parametrised by t. On the contrary, in discrete-time trajectories are sequences of points.

If the time idea associated to the trajectory is neglected, the concept of orbit is obtained. The set of points defined by the trajectory starting in x_0 is called the orbit of x_0 and is denoted by $\gamma(x_0)$. The collection of all possible orbits of the system is called phase portrait or phase space of the system. As in discrete-time orbits are sequences of separated points, their visualisation in the phase portrait requires an additional effort to highlight which points belong to each orbit.

Interestingly, an orbit $\gamma(x_0)$ has infinitely many points that may not be all distinct. Consider an autonomous dynamical system given by $\{\varphi^t\}_{t \in T}$:

- Points in the orbit can be all identical. Then, a point $\bar{x} \in X$ is called a fixed point, stationary state or equilibrium, if $\varphi^t \bar{x} = \bar{x}, \ \forall t \in T$. Hence, if a system reaches a fixed point it will remain in it onwards.
- On the other hand, there can be only p-distinct points in the orbit. The trajectory of x_0 is called periodic or cycle, if $\exists p > 0 : \varphi^p x_0 = x_0$. The smallest integer p such that holds the condition is called the period of the cycle.

Both previous concepts and more complex ideas as chaotic attractors are called invariant sets, i.e. sets that do not change under evolution operator. Formally, a subset $S \subset X$ is said to be positive invariant under φ^t if $\varphi^t S \subseteq S, \ \forall t \geq 0$. Similarly, S is said to be negative invariant under φ^t if $\varphi^t S \supseteq S, \ \forall t \geq 0$. If S is both positive and negative invariant, then S is said to be invariant under φ^t and it holds that $\varphi^t S \equiv S$.

As the basic objective of dynamical systems theory is to describe the asymptotic behaviour of the states as $t \to \infty$, the first main question is about the existence of the simplest type of invariant sets in the system: fixed points or periodic orbits. Unfortunately, the analytical computation of fixed points for difference equations is in general not possible. However, the Brouwer's Fixed Point Theorem guarantees the existence of a fixed point in a continuous map $F : B \to \mathbb{R}^n$ if B is a compact convex subset of $\mathbb{R}^n$, such that $F(B) \subseteq B$. The map F in Eq.4.4 satisfies this theorem conditions assuring the existence of at least a fixed point in RNNs.

Another important question is which initial states lead to these fixed points or periodic orbits, giving rise to the idea of attractors and basins of attraction. The intuitive idea is that basins partition the state space into sets of different asymptotic behaviour. Hence, if a complete description of all possible attractors and their basins of attraction would be available, the long term behaviour of each initial state could be predicted.

The concept of *limit set* captures the idea of all possible asymptotic states of a dynamical system. The ω-limit set $\omega(B)$ of a set $B \subset X$ is defined as

$$\omega(B) = \{x \in X \mid \exists t_n \to \infty, \ x_n \in B : \ \varphi^{t_n} x_n \to x \text{ for } n \to \infty\}$$

Roughly speaking, $\omega(B)$ is the set that contains the limit points of orbits. If φ^t is continuous and $\bigcup_{t \geq 0} \varphi^t B$ is compact it can be shown that $\omega(B)$ is a compact invariant set. As the RNN model obtained in Eq.4.4 is continuous and bounded, it can be proven that all its ω-limit sets are compact invariant sets (Sell and You 2002).

The idea of basin of attraction is quite vague and admits multiple definitions that highlight different properties of the idea of attraction. We consider a definition based on ω-limits.

The basin of attraction $\mathcal{B}(S)$ of a compact invariant set S is defined as:

$$\mathcal{B}(S) = \{x \in X \mid \omega(\{x\}) \subseteq S\}$$

where $\mathcal{B}(S)$ captures the idea of all states in X which asymptotically approach S, i.e. whose ω-limit set $\omega(B)$ is in S. Hence, some sort of attraction to S is defined.

Finally, an *attractor* is defined as a compact invariant set $\mathcal{A} \subset X$ that attracts a neighbourhood of itself. More formally, there exists a neighbourhood U of $\mathcal{A}$ such that $\forall x \in U$ $dist(\varphi^t x, \mathcal{A}) \to 0$ for $t \to \infty$. That is, the orbit approaches A as t tends to infinity. The attractor is said to be minimal if it cannot be decomposed into two disjoint attractors. The attractor is called global if it attracts every bounded set $B \in X$.

If B is an absorbing bounded set, i.e. $\varphi^t \bar{B} \subset B, \forall t \geq 0$. Then $\omega(B)$ is an attractor which attracts B. Again, boundedness of RNN in Eq.4.4 ensures the existence of an attractor. Particularly, $\omega(X)$ which is not necessarily minimal and often can be decomposed into smaller attractors.

4.4 Stability of fixed points

Another interesting question in system dynamics is the stability of fixed points. It is usually analysed considering the local phase portrait near each fixed point. Consider the local phase portrait of a dynamical system around a fixed point $\bar{x}$ given by:

$$x = \bar{x} + J(\bar{x})(x - \bar{x}) + O(2) \tag{4.5}$$

where $O(2)$ represents the second order terms of the expansion. From Eq.4.5 it can be seen that locally this phase portrait depends on the eigenvalues of the Jacobian matrix evaluated at $\bar{x}$:

$$J(\bar{x}) = D_x\varphi(\bar{x}) = \left(\frac{\partial \varphi_i}{\partial x_j}\right)_{i,j}(\bar{x}) \tag{4.6}$$

The eigenvalues $\lambda_1, \ldots, \lambda_n$ of $J(\bar{x})$ determine the stability of the fixed point $\bar{x}$, such that:

- If none of the multipliers of $\bar{x}$ (or eigenvalues) lies on the complex unit circle, i.e. $\nexists \lambda_i \in \mathbb{C}$ such that $|\lambda| = 1$. $\bar{x}$ is said to be hyperbolic.
- Otherwise, $\bar{x}$ is said to be non-hyperbolic.

As a generic matrix has no eigenvalues lying on the unit circle, then hyperbolicity is a typical property of a generic fixed point on an arbitrary dynamic system.

Determining the hyperbolicity of a fixed point is crucial because according to the Grobman-Hartman theorem for DT-DS if $\bar{x}$ is a hyperbolic fixed point of $\varphi \in C^1$ then the dynamical system is, in a neighbourhood of $\bar{x}$, locally topologically conjugate to its linearisation.

In plane words, two dynamical systems are locally topologically equivalent if their local phase portraits are similar in a qualitative sense, i.e. they can be transformed into each other through a continuous transformation. More technically, if there exists a homeomorphism between orbits of both systems preserving the direction of time. If the homeomorphism also preserves that trajectories evolve with the same speed the systems are called topologically conjugate.

Hence, applying the Grobman-Hartman theorem the phase portraits of hyperbolic fixed points $\bar{x}$ can be classified studying the eigenvalues of the Jacobian $J(\bar{x})$:

- A hyperbolic fixed point $\bar{x}$ is called a stable node if all its multipliers are inside the unit circle.
- $\bar{x}$ is called an unstable node if all its eigenvalues are outside the unit circle .
- Finally, if there exists multipliers both inside and outside the unit circle $\bar{x}$ is called a saddle point.

As a final summary of the previous ideas of this chapter, the dynamical system approach explicitly describes RNN hidden states as time-varying trajectories in a high-dimensional state space.

From the continuity and boundedness of the difference equations of an RNN in Eq.4.4 derives that:

- By the Brouwer Fixed Theorem, there exists at least a fixed point in RNNs.
- Its ω-limits are compact invariant sets and the existence of attractors (of their neighbourhood) is assured.

Thus, depending of the initial state x_0, the convergence of the state x_t of the system to an attractor $\mathcal{A}$ is guaranteed, under the repeated application of map $\varphi^1 \equiv F$. Possibly (but not assured), there are several different attractors in the system that describe the asymptotic behaviour of the model. The state space X is divided into basins of attraction $\mathcal{B}$, one for each attractor. If the network is started in one basin of attraction, the model will converge to the corresponding attractor as t grows.

From this point of view, the fundamental problem in RNNs is to design the number, position and stability of their invariant sets (attractors) and their basins of attraction such that the dynamics of the network is able to solve a specific task.

4.5 First Wave: theoretical dynamical analysis of RNNs

In 1984 Hopfield suggested an associative memory model based on neurons, which was able to memorize several desired patterns as fixed points of the underlying dynamics. Starting from an arbitrary initial state, the dynamics approaches one of the attractors, i.e. iterating until a fixed point is reached, recalling the corresponding pattern (Hopfield 1984). This associative memories are examples of multiple attractor networks where representational content is assigned to its attractors.

From this initial work a fundamental question arose: how the results of a computation are represented in a neural network model. Even a more general issue can be raised about the underlying computational objects of RNNs. Two families of answers were given from the dynamical system point of view:

a) On one side attractors are considered as the fundamental elements. It was found that even small recurrent networks exhibit complex dynamical behaviour: from traditional fixed points to the existence of cycles or even chaotic attractors. This fact was proved in specific discrete-time networks of two neurons in (Wang 1991).

b) Sometimes emphasis is placed on the evolving trajectory rather than in the final attractor itself. Its interaction with the attractors structure and their basin of attraction when the trajectory is captured by different attractors. e.g. when the system interacts with an environment the space is subject to changes due to the input signals. These RNN structures are typically associated to neuroscience and information processing in human and animal brains where input signal is the sensorial information that alters the topological structure of the state space.

Therefore, the interest focus on given a rich variety of dynamical behaviours how they can be controlled and which parameter sets cause a specific dynamics. The knowledge of these parameter sets would allow to control switching between different dynamical regimes and to improve RNN learning algorithms.

To gain a deeper understanding of the underlying processes techniques from dynamical systems theory are applied, specially bifurcation analysis in the parameter space of the system. Basically, these analysis try to identify how the topological structure of attractors

changes when some system parameter is modified in a working range. The typical output of these analysis are bifurcation manifolds that separate regions in parameter space which exhibit qualitatively different dynamical behaviour.

Unfortunately, analytical computation of bifurcation manifolds become infeasible for networks with more than three neurons. Hence, a reductionist approach was adopted by the different authors: two and three-neuron systems were analysed as an starting point to build more complex networks as a composition of these basic building blocks. Initial steps in this directions obtained these bifurcation curves using numerical methods and often restricted to simplified connection matrices.

Given that attractors are basics elements, first analytical works focused on the question of how to constrain the weights of the RNN so that they exhibit only fixed points and the conditions under they have an attractive behaviour. Obtaining for a rather general class of recurrent neural networks conditions under which all fixed points of the network are attractive determined by the weight matrix of the network. Even more, with bounded derivatives of the non-linear activation function, these conditions, were easy to check using simple algebraic manipulation of matrix weights (Casey 1996; Jin et al. 1994).

Two-neuron recurrent networks fixed point number, position and stability were analysed by (Tiňo et al. 2001) for sigmoid-shaped activation functions. Also the saddle node bifurcation mechanism was identified as the usual responsible for the creation of a new fixed point. It creates a pair of new fixed points, one attractive and one saddle-point. Similarly, fixed points disappear in a reverse manner: an attractor collides with a saddle and they get annihilated. Additionally, the author exposes a partition of the state space into several regions corresponding to different stability types of the fixed points.

Two and Three-neuron networks were deeply analysed by Haschke obtaining analytical expressions for the bifurcation curves considering the external input as the main bifurcation parameter. Depending on the eigenvalues of two-neuron RNN fixed points three different types of bifurcations are possible: saddle nodes (associated to the birth of two new fixed points), period doubling (appearance of 2-cycle) and Neimark-Sacker bifurcation (giving rise to a limit cycle) (Haschke, Steil and Ritter 2001). The three-neuron case was studied in (Haschke 2003) and (Haschke and Steil 2005).

Unfortunately, this first wave in dynamical system approximation to RNNs had three main drawbacks:

- A very complex underlying mathematical analysis even for the simplest two and three-neuron cases. The increasing complexity of the obtained results with the number of neurons makes it almost unfeasible to extend the analysis to more complex networks.

- As a consequence of the previous problem, the real main drawback of this approach is its inherent difficulty to translate the obtained results to RNNs solving real-life tasks.
- Finally, these analysis are performed for autonomous systems. In order to handle non-autonomous systems, the alternative way of think is to consider piecewise constant inputs, i.e. inputs changing on a time scale much slower than the networks dynamics. Hence, the dynamical system must be analysed as a sequence of slowly changing autonomous systems. However, the natural field of application of RNNs is the analysis of sequence that are inherently associated to time-varying inputs.

Chapter 5

Non-autonomous Dynamical Systems

5.1 Consequences of Non-autonomous

All dynamical systems concepts described in the previous chapter: fixed points, limit sets, attractors and, in general, the qualitative description of the space portrait topology and bifurcation analysis were developed under the assumption of constant input signal. Thus, system evolving law remains unchanged upon time, i.e. autonomous dynamical systems. On the contrary, real-life problems consider time-varying input sequences that feeds RNNs. Hence, the systems is suffering the influence of external forces and their update equations exhibit a temporal change, i.e. non-autonomous dynamical systems.

The mathematical analysis of non-autonomous systems is by far more complex and it has begun to be developed in the last decade (Kloeden and Rasmussen 2011). Extending the theory of autonomous dynamical systems to non-autonomous ones present many difficulties. Mainly, trivial attractors in fixed-input situations can be transformed into complex state spaces in the non-autonomous case.

Recall that in the classical notation, a Discrete Time autonomous system on a state space X was given by a map $F : X \to X$ and the state of the system at time n is given by $x_n = F(x_{n-1})$. Alternatively, a non-autonomous systems on a space X is defined as a family of maps $\{F_n\}$, where each $F_n : X \to X$ is a continuous map. In this case, the state of the system at time n is given by $x_n = F_{n-1}(x_{n-1})$.

RNNs are a particular case of non-autonomous systems, called input-driven systems (IDS). An IDS is given by a map $F : U \times X \to X$, F_n and an input sequence $\{u_n\}$, with $u_n \in U$. In this case, the state of the system at time n is given by $x_n = F(x_{n-1}, u_n)$. Thus, $F_n(\cdot) = F(u_n, \cdot)$.

The key idea is that while the input signal is constant, the topology of the phase portrait remains unchanged. Each time the input signal changes, the attractors topology can change dramatically. Moreover, attractors are defined as sequence of sets, where the position of the set in the sequence determines the time index. Nowadays this is an active research field.

This time-varying input signal has a direct impact on RNNs training, making them hard to

be trained properly. As described in section 3.4 two of the main reasons are the vanishing gradient and the exploding gradient problems described by Bengio (Y. Bengio, Simard et al. 1994). These phenomena can be interpreted under the perspective of non-autonomous dynamical systems.

5.1.1 Vanishing gradients: contractive basins of attraction

Vanishing gradients refers to the problem: when long-term components go exponentially to zero making impossible to learn correlation between temporally distant events. In a classical paper (Y. Bengio, Frasconi et al. 1993) presented interesting ideas relating gradient vanishing problems in RNNs trained using backpropagation and dynamical systems theory. Even more, he stated that any gradient descent algorithm in the output error is inappropriate to train a non-autonomous recurrent network.

To illustrate these concepts the author considers a two-class noisy sequence classification problem. During the training process to learn a state information (e.g. the class of the incoming sequence) our hope is to be able to restrict the values of the system to a subset S of the state space. Following this idea, the state can be interpreted as being located inside S or outside S. If this region is designed as a basin of attraction it is assured the system remains in S.

As seen in the previous section if the input sequence changes we are under the scope of non-autonomous systems theory. Therefore, if input changes the system can be pushed out of the current attractor possibly to the basin of attraction of another attractor indicating that the incoming sequence is associated to a different class.

The key idea in vanishing is the so called the reduced attracting set of an hyperbolic attractor. It is a subset of the basin of attraction whose states holds some interesting constraints on their eigenvalues. Formally, given a DT-DS defined by $\varphi^1 \equiv F$ the reduced attracting set $\Gamma(X)$ of an hyperbolic attractor X is defined as:

$$\Gamma(X) = \{x \in \mathcal{B}(X) \mid \forall p \geq 1, \text{all eigenvalues of } D\varphi^p(x) < 1\} \tag{5.1}$$

Intuitively, an initial state $x \in \Gamma(X)$ moving through the state space [1] is accompanied by some kind of "contraction" during its travelling. This "contraction" will be given by all its Jacobian eigenvalues located inside the unit circle during its journey ($\forall p \geq 1$) until the

[1] Future states of x are given by $\varphi^p(x)$

attractor is reached (forced by $x \in \mathcal{B}(X)$). By definition, for a hyperbolic attractor X, $X \subset \Gamma(X) \subset \mathcal{B}(X)$.

Now consider a state $x(0)$ and a ball of uncertainty around it. It can be proved that for an hyperbolic attractor X if $x(0) \in \mathcal{B}(X)$ but $x(0) \notin \Gamma(X)$ then the size of the ball of uncertainty will grow exponentially as t increases. Hence, small perturbations in the inputs could push the trajectory towards another basin of attraction implying that the system is not resistant to input noise.

On the other hand, a system is said to be robustly latched at time t_0 to an attractor X if $x(t_0) \in \Gamma(X)$, because it is guaranteed that $x(t)$ remains within certain distance of X when the input noise is bounded. In the case of non-autonomous systems, the ball of uncertainty is generated by the input u_t. The system remains robustly latched to X as long as u_t is such that $x(t) \in \Gamma(X)$ for $t > t_0$. The smaller the eigenvalues are the more robust to noise is the system.

A price has to be paid for storing info in a way resistant to noise. If the input sequence is such that the system remains robustly latched on attractor X, by definition of $\Gamma(X)$ and considering $p = 1$ it holds:

$$|D\varphi^1 x(t-1)| = |\frac{\partial x(t)}{\partial x(t-1)}| < 1 \tag{5.2}$$

Hence:

$$|\frac{\partial x(t)}{\partial x(0)}| \to 0 \quad \text{as } t \to \infty. \tag{5.3}$$

This constraint has a direct impact on the gradient descent algorithms of backpropagation, in the first step the forward propagated cost function ε is computed and the key step is to compute the gradients of ε with respect to the network weights W. The gradient of the error in a RNN at time t is:

$$\frac{\partial \varepsilon}{\partial W} = \sum_{1 \le k \le t} \frac{\partial \varepsilon_t}{\partial W} = \sum_{1 \le k \le t} \frac{\partial \varepsilon_t}{\partial x(k)} \frac{\partial x(k)}{\partial W} = \sum_{1 \le k \le t} \frac{\partial \varepsilon_t}{\partial x(t)} \frac{\partial x(t)}{\partial x(k)} \frac{\partial x(k)}{\partial W} \tag{5.4}$$

The impact of Eq.5.3 is clear, the sum terms tend to zero with $k << t$. This means that long-range dependencies terms tend to zero and this relations are not captured by the network.

5.1.2 Exploding gradients: trespassing basins of attraction frontiers

While vanishing gradients refers to components going to zero, exploding gradients problem refers to the large increase in the norm of the gradients during training due to the explosion of the long term components.

In dynamical systems theory, the parameters of the model are typically denoted as $\theta \in \Theta$ where Θ is the parameter space. To make explicit this analysis F is usually denoted as $F(x;\theta)$. Given any parameter assignment θ the autonomous system evolves from an initial state converging to one of the possible several attractors of the RNN.

When θ changes continuously the asymptotic behaviour of the system varies smoothly. This smoothness is broken when certain critical value of the parameter is reached. This point is called bifurcation point and qualitative changes occur in the phase portrait and topology of invariant sets can be altered, i.e. attractors can change their stability, emerge, collide or even disappear, and of course their basin of attraction shapes can be altered.

Doya hypothesizes that gradients explode when a bifurcation is crossed (Doya 1993). This situation is illustrated in the bifurcation diagram presented in Fig.5.1 for a single unit RNN where x represents the value of the fixed point and b is the bias of the unique neuron.

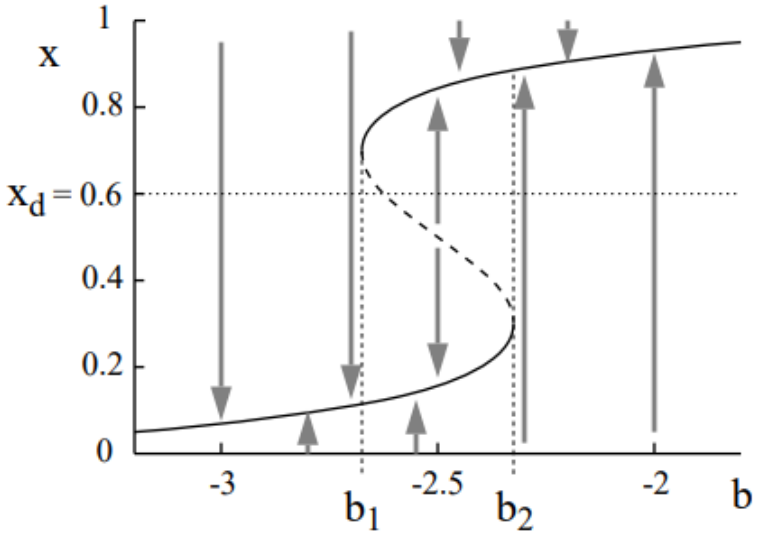


Figure 5.1: Bifurcation diagram of a 1 neuron RNN. Source: Doya 1993

Suppose the desired output of the system is $x = x_d$. Starting from e.g. $b = 0$ and given an initial state x_0 the system converges to the unique fixed point given by the point $x(b = 0)$ located in the upper branch of the curve. As $x(b = 0) \neq x_d$ the application of a gradient

descent algorithm changes b making the fixed point moves toward x_d. b moves continuously until a critical point $b = b_1$ is reached. In this moment the stable point of the upper branch disappears and a new one appears given by the lower branch of x. There is a jump in the convergence point for b_1^+ and b_1^- resulting in an abrupt change in the error for a tiny variation of b. Hence, gradient with respect to b goes to infinity at the bifurcation $b = b_1$. An important remark is that this huge error changes can occur with very small values of learning rate as it happens in the step from b_1^+ to b_1^-.

To reach x_d the learning procedure flips the changing direction of b towards higher values. This increase reaches a new bifurcation point at $b = b_2$ giving rise to a new abrupt change in the error. It can be observed the learning procedure remains in an infinite loop between b_1 and b_2 never reaching x_d.

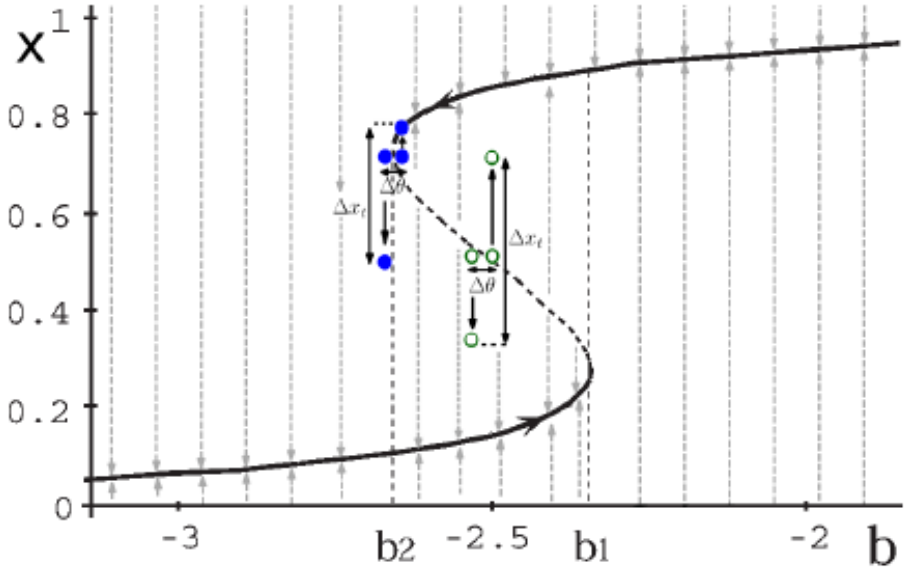


Figure 5.2: Bifurcation diagram of a 1 neuron RNN. Source: (Pascanu et al. 2013)

This analysis has been enriched by Pascanu (Pascanu et al. 2013) arguing that:

- Crossing a bifurcation is a global event that can have no locally effects. To understand this idea, consider a two attractors network with a certain parameter changing until a bifurcation point is reached. Suppose that this bifurcation alters the topology of attractors, disappearing one attractor. If in that precise moment the system's state is not located in the basin of attraction of that attractor, learning process will not be affected by it. Hence, crossing a bifurcation is neither necessary nor sufficient for gradients to explode.

- Basin of attraction boundary crossing is a local event with effects. Consider a change in the state of the system such that the state falls in a different basin of attraction from the current one. Thus, the state will be attracted towards a new direction giving rise to the gradients explosion. This effect also appears if the basin of attractions boundaries are altered in such a way that the state of the system changes of basin. Hence, this boundaries crossing is sufficient for gradients to explode.

In Fig.5.2 the original Doya's Fig.5.1 is reinterpreted by Pascanu. When b decreases from ∞ there is a bifurcation point at b_1 emerging a second attractor. At b_2 there is a another bifurcation point that collapses the recently created new attractor. In the interval (b_1, b_2) coexist two attractors in the system and the boundary of their basins of attraction is plotted by a dashed line. There are only two points where bifurcation appear but their a whole region where transition between boundaries are possible.

Given that the collapse of an attractor or the appearance of a new one, implies changes in basin of attraction boundaries, the cases considered by Doya are a particular case of Pascanu's proposal.

5.2 Last advances in RNN training

The training difficulties described in the previous sections have hindered the widespread use of RNNs. It has made them to fall out of favour during almost a decade, approximately from 2005 to 2015.

On the one hand, exploding gradients problems were successfully alleviated considering gradient clipping of large gradients and the introduction of L2 or L1 weight norm penalties (Pascanu et al. 2013).

On the contrary, vanishing gradients are much trickier and multiple research lines have tried to tackle it from completely different points of view:

i) Introducing gating mechanisms in the hidden layers. As seen previously, the original hidden units are replaced by new ones with a more complex internal architecture incorporating gating elements. This architectures create multiple flowing paths inside the hidden layer, the most important is an additive path that acts as a memory retaining long-term dependencies. The most common proposals are GRU (Cho et al. 2014) and LSTM (Hochreiter and Schmidhuber 1997).

ii) Considering non-saturating activation functions. These authors proposed that traditional bounded activation functions (logistic and tanh) are the responsible of vanish-

ing gradients due to their non-linear saturating behaviour, i.e. contractive functions that saturate at the bounds. In that sense, Rectified Linear Units (ReLU) are non-saturating functions and prevents gradients from vanishing (Nair and G. E. Hinton 2010). As expected, Non-saturating Recurrent Units (NRU) architectures have been proposed as hidden layer for RNNs (Chandar et al. 2019).

iii) Improving optimization algorithms manipulating the propagation path of gradients. This technique hypothesizes that the gradient components through the linear path in the LSTM computational graph carries information about long term dependencies. When LSTM weights are large, these components would be suppressed. An algorithm has been proposed for preventing gradients flowing through this path from getting suppressed, allowing LSTM to capture such dependencies better. (Kanuparthi et al. 2019)

iv) Introducing new optimization algorithms that substitute gradient descent techniques considered in back propagation through time (BPTT). Gradient descent relies on a smoothness assumption while 2nd-order optimization techniques as Hessian free can take into account abrupt changes in error function curvature (Sutskever et al. 2011), (Sutskever 2013).

v) Considering spectral constraints, i.e. constraints on weight matrices. The key idea is that vanishing and exploding gradients can be modulated controlling the maximum and minimum gain of weight matrices. One of the main ideas in this direction is that keeping weight matrix close to orthogonality, gradient norm is preserved during backpropagation. There are different variants of these techniques, even some authors propose loosen the constraints to increase the rate of gradient descend convergence (Vorontsov et al. 2017).

vi) Limiting the scope of the optimization problem to the readout weights. This gives rise to a new paradigm known as "Reservoir Computing" where the recurrent layer (called a reservoir) connections are randomly initialized from uniform or Gaussian distributions and these weights are not fined-tuned via gradient-descent optimization mechanisms. The main idea is to exploit the rich dynamics generated by this reservoir with an output layer (the read-out) that is optimised to solve an specific task. Hence, gradients explosion and vanishing is avoided by not learning the recurrence and input matrices. One example of this paradigm are Echo State Networks (Jaeger and Hass 2004). As these networks are trained introducing perturbations in the reservoir layer and learning the output matrix, some of the previous techniques have been combined to improve learning process, e.g. weight matrix orthogonality.

5.3 Renaissance: practical dynamical analysis of RNNs

The good results obtained by the previous developments in RNN training have brought a rebirth of the use of these networks. This renaissance comes hand in hand with a new interest in their dynamical system perspective.

As seen in section 4.5, the first wave of dynamical system theory applied to RNNs was characterized by extremely theoretical analysis of very simple two or three neuron RNNs. On the contrary, in this second wave a practical perspective permeates all trying to capture the essential dynamical behaviour of RNNs from a high-level perspective neglecting "microscopic" details characteristics of the first wave, e.g. deriving attractor equations based on network weights.

A seminal paper of this new point of view is (Sussillo and Barak 2013) were the classical approach is refreshed. The basic line of attack is maintained, as the qualitative behaviour of non-linear systems can vary greatly between different regions of phase space, a divide and conquer procedure is followed. The different regions in space portrait are studied separately and, finally, the interactions between different regions are considered.

The classical first step is identifying the topological structure of the fixed points and their behaviour is analysed locally linearising in their neighbourhood. Classical fixed points are zero speed points such that if the system reaches a fixed point it evolves remaining in it. However, the authors propose other areas of the phase space to linearise the system that are not true fixed points but merely points of "very slow movement" or slow points. To find all these points of interest a simple optimization technique is proposed based of the minimization of an auxiliary function that takes into account some kind of "kinetic" energy.

In order to understand the interactions of different regions, the stability of the fixed points must be analysed. Stable fixed points are important because states in their vicinity will converge to them. Based on its eigenvalues there are different behaviour of unstable fixed points: from completely unstable to saddle points. These unstable fixed points with a few unstable modes are very useful to understand the interaction between regions. They can funnel a large number of orbits through stable modes and them send them to different attractors depending on which unstable mode is taken.

As an important novelty, the authors apply these ideas to different quite simple problems revealing an unexpected but powerful key idea. Despite their theoretical capacity to implement complex, high-dimensional computations, trained RNN networks converge to highly interpretable, low-dimensional representations, i.e. the movements of the states of the system are restricted to a low-dimensional manifold. Hence, once trained a network to solve a

concrete task, the states in this manifold could be projected in 3D plots with the hope of obtaining a qualitative understanding about how the RNN solves the task.

As an example of these ideas, the authors apply them to the 3-bit flip-flop problem. In this case, an RNN implemented as a Echo State Network, tries to predict the next state of 3 flows of bit pulses given certain transient rules that determine the output. After training the states of the system are projected using the first three principal components in a 3D plot shown in Fig.5.3.

In order to determine the state of three 0/1 pulses the RNN needs eight possible memory states, i.e. stable fixed points, represented in black crosses. In green crosses are saddle points separating fixed points. If the input signal remained unchanged it would be an autonomous dynamical system and the system would be trapped in one of the fixed points. But as input changes, the system is non-autonomous with all its consequences. The authors interpret that when input changes the state of the system is perturbed and came closer to the saddle point and then finally is send to another fixed points. Hence, saddle points have an important role in this problem, mediating the transitions between attractors.

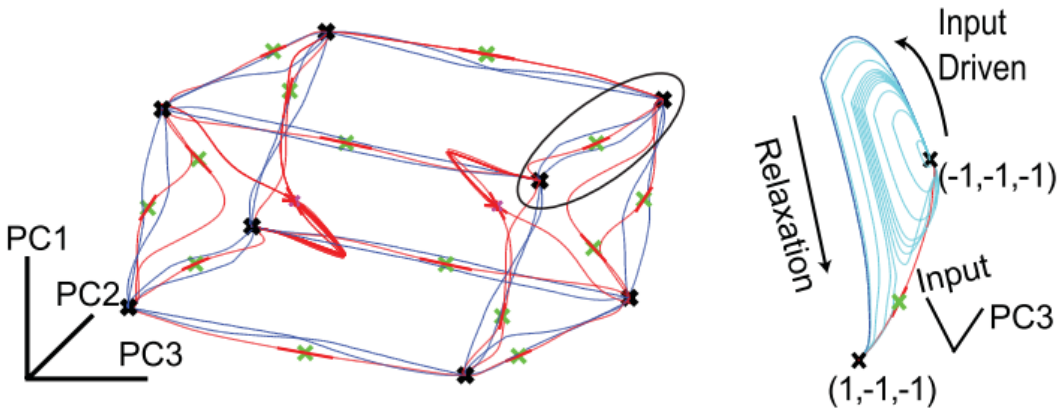


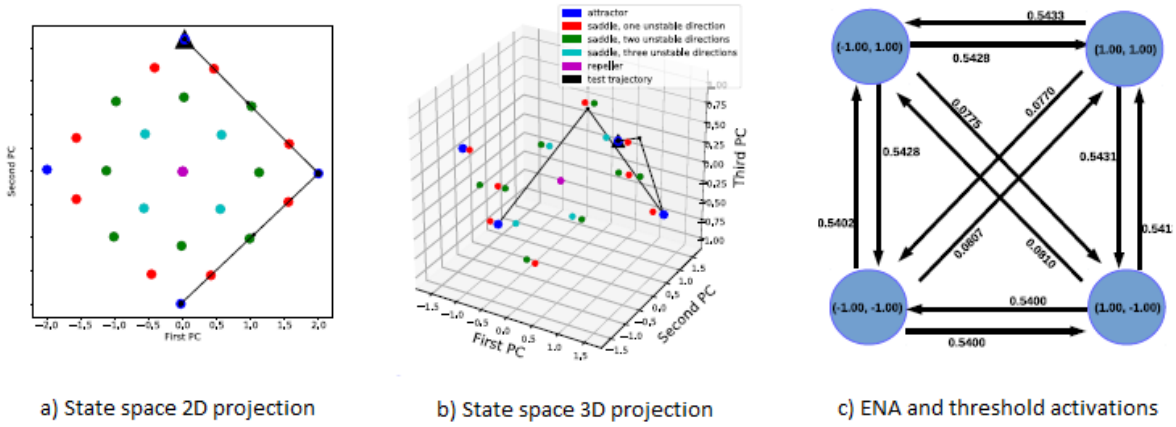
Figure 5.3: State space of trained ESN for the 3-bit flip-flop problem Source: Sussillo and Barak 2013

Recently this same investigation group has published new results applying these techniques to more complex sentiment analysis problems (Maheswaranathan et al. 2019). The solutions mechanism found by RNN to solve the problem corresponds to phase portraits that live in 1D-attractors or "line-attractors". Even more, they show that this mechanism is present across different RNN architectures (LSTMs, GRUs, and vanilla RNNs) trained on multiple datasets.

Finally, an interesting paper has been publish (Ceni et al. 2019) that goes a step further in the interpretation of RNN behaviour when solving a computational task. A more advanced mathematical structure is considered: Excitable Network Attractors (ENA), which are invariant sets in phase space composed of stable attractors (of any kind: fixed points, limit cycle, limit tori or strange attractors) and excitable connections between them. This underlying ENA is represented as a directed graph on which the dynamics takes place. The nodes of the graph are the fixed points of the system and directed edges describe excitable connections between them.

The idea is quite simple, there exists an excitable connection of amplitude δ from an attractors p_i to the attractor p_j if a ball of radius δ around p_i intersects the basin of attraction of p_j. In plain words, if an input perturbation applied in p_i can reach p_j's basin of attraction. The authors introduce the concept of excitability threshold as the minimum δ for which the ball intersects the basin.

This technique is applied to the same flip-flop problem as Sussillo but in a 2-bit version. The outputs of this technique are shown in Fig.5.4. They confirm Sussillo's ideas that dynamics of high-dimensional RNNs take place in a much lower dimensional phase space region and additionally, a transition graph between attractors is obtained. This model can be exploited to provide a mechanistic interpretation of errors occurring during the computation and also the excitability thresholds can be used to assess the robustness of RNNs to noise-induced perturbations.



a) State space 2D projection b) State space 3D projection c) ENA and threshold activations

Figure 5.4: State space 2D and 3D projections and ENA of a trained RNN for the 3-bit flip-flop problem. Source: Ceni et al. 2019

Chapter 6

Practical case selection

In order to illustrate the previous dynamical system ideas applied to RNNs we have selected a toy problem extremely intuitive whose computational solution achieved by the RNN is easily viewable. In the following chapters the concepts of state space, orbits, attractors and parameter space are exposed based on this example. Hopefully, the utmost importance of dynamics for the interpretation of RNN behaviour solving tasks will be even clearer after these final chapters. As any problem on sequences the selected one is composed by an specific task to solve over an input sequence and a machine learning model that tries to complete the task.

6.1 Input: sequence a^4b^4

Let's consider an uni-dimensional discrete bi-valued sequence $\{X_i\}_{i>0}$, with $X_i = (x_{i1})$ and $x_{i1} \in \{0, 1\}$ i.e. only one component varies over discrete steps of a parameter i taking only two possible values represented by 0 and 1.

Uni-dimensionality appears when an isolated variable is the case of study (e.g. daily evolution of the number of infected persons by a disease in a region). In most cases the discrete step i at which the sequence take values represents time $(i = t)$ but it also admits other interpretations as spatial steps (e.g. number of infected persons at distance steps from an suspicious initial focus point) or even more sophisticated ideas associate parameter steps with jumps through graphs nodes.

In real cases, input sequences are usually more complex, composed by multiple variables varying over time (e.g. daily evolution of the number of infected persons by a disease in a region and the daily average temperature in that region). In this situations input is represented by a vector with as many components as input variables are considered $X_i = (x_{i1}, \ldots, x_{in})$. Obviously, in these cases, the interpretation of the parameter i must be common for all variables.

Time sequences are applied to machine learning models to solve a problem with the hope that it conveys some kind of internal structure, i.e. long-term vs short-term dependencies.

This internal structure in the sequence can be possibly deeply hidden in the input signal and the model must work hardly to unveil it and make use of it to complete successfully the task.

In our practical case, we have imposed one of the simplest patterns to the sequence, an evident one for a human observer. In formal language theory it is the classical grammar $a^{n_1}b^{n_2}$. The resultant sequence consists in the infinite repetition of a chunk composed by n_1 a's followed by n_2 b's. For the sake of simplicity we have considered that $a = 0$ and $b = 1$. Thus, our sequence is represented by the pattern $0^{n_1}1^{n_2}$.

This pattern is simplified even more considering an equality constraint on the length of both chunks, such that $n_1 = n_2 = n$ [1] and a total sequence length long enough to be considered infinite for our purposes even if the sequence is divided into different datasets for training, validation and testing. [2] As a concrete value for n is needed, a length of 4 is enough to illustrate the problem. Hence, from now on the working sequence follows a grammar with pattern 0^41^4, i.e. $\ldots 00001111 \ldots$.

6.2 Task: Next symbol estimation

Given our arriving sequence $\{X_t\}_{t>0}$ of 0s and 1s, the simplest machine learning problem on this sequence is selected: the next symbol in sequence prediction. More formally, at every time step t' predict the next arriving element in the sequence $X_{t'+1}$ taking into account the previously received history $\{X_t\}_{0<t<t'}$ as shown in Fig.6.1. As the selected sequence can take only two possible values, this prediction task can be interpreted as a classification problem among two different classes: 0 and 1.

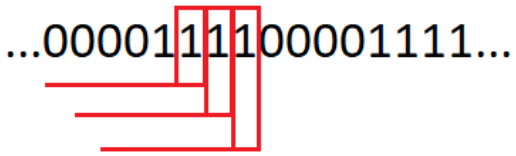


Figure 6.1: Next symbol prediction of a 0^41^4 flow.

[1] In the python code, n_1 and n_2 are called *len_zeros* and *len_ones* respectively

[2] The sequence length is controlled by the parameter *multiplicity* representing the number of times the pattern is replicated. In our case *multiplicity* is set to 1.000

6.3 Model: Binary classifier

This problem can be tackled using different machine learning models. One of the very common approaches consists in the use of solutions based on RNN models. The simplest one is composed by two stages as shown in Fig. 6.2:

- A first vanilla-RNN layer that processes the input sequence in a recurrent manner. As seen in previous chapters, there are multiple possible operating modes of a RNN. In this case our input is the previous history of the sequence (multiple inputs) and the expected output is a single value for the next element in the sequence, the most suitable mode for this input-output configuration is the many-to-one configuration.
- A final dense layer that simply takes the output of the previous layer as input information to estimate the class of the next element in the sequence.

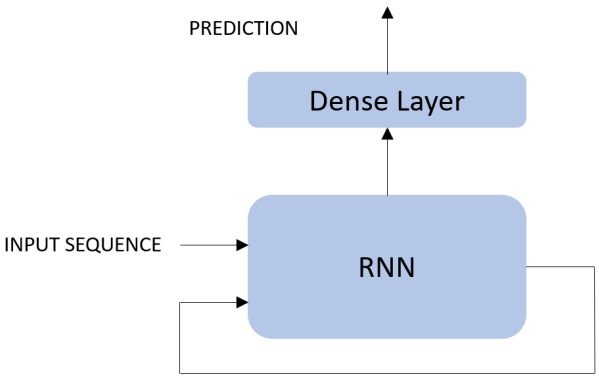


Figure 6.2: Block structure of solutions based on RNN models.

Any vanilla-RNN stage is characterized by three main parameters:[3]

a) *n_steps*: the number of time steps considered backward to estimate the next element in the sequence. It is also known as the window width or window size used in the prediction. To capture an 0^n1^n input pattern, it seems reasonable to consider a window size higher or equal to the length n of the chunk of zeros or ones in the input sequence. Therefore, $n\_steps \geq n+1$. Applying this assumption for our a^4b^4 sequence results a value for the parameter $n\_steps = 4+1 = 5$.

[3]In the code, these parameters are called: *n_steps*, *neurons* and *activation_RNNCell*.

b) *n_hidden*: the number of neurons of the hidden layer. One of our goals is to test the Sussillo's claim that trained state spaces are restricted to low-dimensional manifolds much lower than the original high-dimensionality of the RNN. Therefore, a high-dimensional recurrent layer is selected with $n\_hidden = 8$. In later chapters *n_hidden* will be considered a network parameter and its impact in the phase portrait is studied.

c) *activation_function*: of each unit in the hidden layer. For our task we have selected a classical tanh function.

In Fig.6.3 the obtained RNN is represented with its fixed parameters: n_{hidden} and n_{steps} and dimensions of the input data and hidden layer tensors are also shown. Input data tensor dimensions are given by the number of variables in the sequence (sequence dimensionality), the selected window size and the number of batches considered. Usually, the number of batches is left open indicated as $None$:

$$(batches, window\_size, n_{input}) = (batches, n_{steps}, n_{input}) = (None, 5, 1)$$

Similarly, the shape of the hidden layer is:

$$(batches, n_{hidden}) = (None, 8)$$

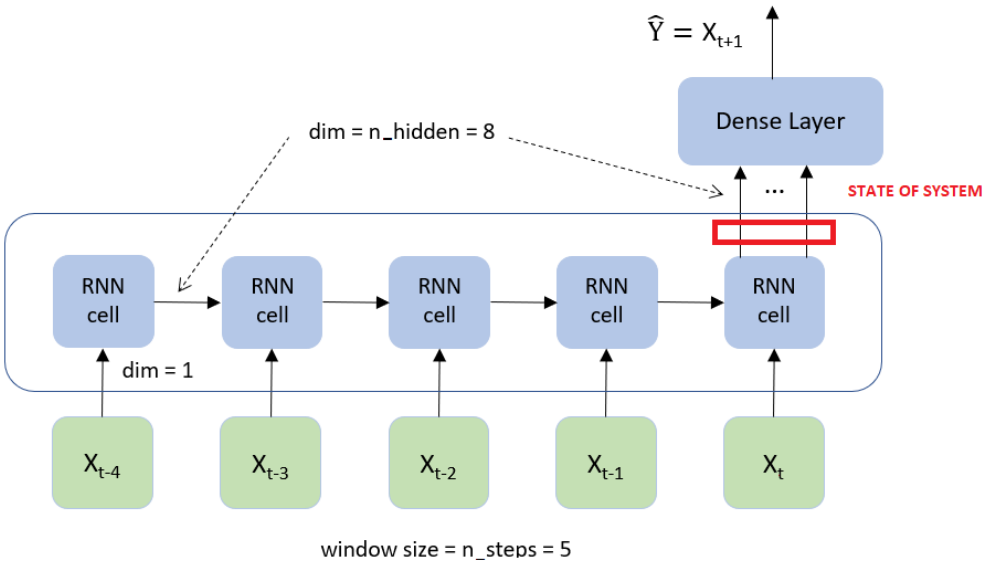


Figure 6.3: Selected RNN for solving the next element of the sequence problem.

In the many-to-one operating mode the output of the hidden layer is inspected only in one point of the unfolded representation. This hidden layer output is the state of the system at

each time step considered in the previous chapters. This output values can be interpreted as a 8D-representation of the input sequence built by the network in order to solve the problem and it is offered as input to a very simple final dense layer that predicts the class of the next element of the sequence. The number of neurons of this layer is fixed by the dimensionality of the output of the previous stage. Therefore, only one parameter is needed to fix this dense layer:

a) $activation\_function$:[4] of the output layer of this dense block. As the task on hands is a classification problem among two classes 0/1, the typical activation function for this purposes is a sigmoid.

The input tensor to this dense layer is the output of the previous stage. The output shape of this layer is simply the estimated class of the next element in the sequence $(None, sequence\_dim) = (None, 1)$. Hence, its shape is:

$$(batches, n_{hidden}) = (None, 8)$$

A summary of layers, their dimensionality and the number of parameters involved in the network is shown in Fig.6.4:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 8)                 80
_________________________________________________________________
dense (Dense)                (None, 1)                 9
=================================================================
Total params: 89
Trainable params: 89
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.4: Layer summary of the selected RNN

The number of internal parameters can be easily deduced. The RNN layer receives two inputs: the sequence and the previous time step feedback. Each one is transformed using a weight matrix of the adequate dimension. As $dim(x_t) + dim(hidden) = 1 + neurons$ and taking into account the bias associated to each neuron $1 \times neurons$, the total number of parameters is: $(1 + neurons) \times neurons + 1 \times neurons = 80$.

[4] In the code is named *activation_dense*

On the other hand, the dense layer has an unique output neuron which receives inputs from n_hidden neurons. The associated weight matrix has $n\_hidden \times 1$ elements and a bias associated to the output neuron giving a total number of $n\_hidden \times 1 + 1 \times 1 = 8 \times 1 + 1 = 9$. Hence, the total number of parameters of the model is $80 + 9 = 89$. Any of these 89 network parameters could be chosen for a parametric analysis of the system.

In the following chapters we will explore the structure of the phase portrait of the system, the behaviour of the network under variations of sequence and model parameters. Up to this point we can consider that our problem has the following parameters:

- Sequence parameters:
  - *len_ones*: number of elements of the chunk of zeros pattern.
  - *len_eros*: number of elements of the chunk of ones pattern.
  - *multiplicity*: number of repetitions of the pattern.
- RNN parameters:
  - *n_steps*: window of input signal considered to predict the next element.
  - *n_hidden*: number of neurons of the output.

Chapter 7

State Space after network training: dimensionality reduction and visualization

In this chapter we seek to confirm Sussillo's hypothesis about the dimensionality of the state space of trained RNNs. Let's recall that this author suggests that after training a high dimensional RNNs to solve a concrete task, the state space obtained by the network is such that the movements of the states are restricted to low-dimensional manifolds.

To check if this idea holds in our problem we must perform the following steps:

1) Train the network as usual to solve the problem.

2) Once trained, obtain the state space, i.e. the hidden layer output in the original 8D-space.

3) Plot 2D and 3D projections of the state space. If these projections throw interpretable results it means that the system actually performs its computations restricted to a 2D-3D space instead of the original 8D-state space.

To obtain the results presented in the following chapters a python3 library has been developed from scratch. It has been designed following a Object Oriented philosophy with a technology agnostic flavour. Networks definition, manipulation and training has been performed using Tensorflow package.

7.1 Model Training

As explained in the previous chapter the input sequence was built as a concatenation of the pattern $0^4 1^4$. Given the simplicity of the task, there is no need for a long sequence. Hence, 1000 repetitions of the pattern have been taken obtaining a total symbol length of 8000[1].

[1] To reduce memory consumption an unsigned 8-bit integer datatype was selected.

To complete our next symbol estimation task, the input sequence needs to be transformed into two sets: feature (X) and label (Y) datasets. The feature dataset is built chopping the original sequence into chunks of length n_steps = 5 while the label dataset consists on picking the immediately following element in the sequence for each of input pattern in X. Obviously, the degenerated or incomplete final samples are discarded.

As usual, these feature and label datasets are splitted into three subsets for training, validation and test following a classical schema 70-20-10 proportion. Hence, the lengths of the different datasets are: 5596, 1599 and 800 respectively. These datasets must be reshaped before being feeded into the network according to the tensor's input shape restrictions as shown in Fig. 7.1. Finally batches of one hundred samples are created using the idea of TensorFlow DataSets [2].

```
Sequence summary:
------------------------------------------------------------------
sequence snapshot: [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1]
sequence length: 8000
data type: dtype('uint8')
noise: No
seed: 123
------------------------------------------------------------------
Datasets summary: [0.7, 0.2, 0.1]
------------------------------------------------------------------
Dim X, Y:             (7995, 5, 1),(7995, 1)
Dim X_train, Y_train: (5596, 5, 1),(5596, 1)
Dim X_valid, Y_valid: (1599, 5, 1),(1599, 1)
Dim X_test, Y_test:   (800, 5, 1),(800, 1)
------------------------------------------------------------------
```

Figure 7.1: Summary of the input sequence and its splitted datasets.

For the network training process, some elements need to be fixed [3].

- As usual in classification problem with only two classes, we have chosen the binary cross-entropy as the loss function to be minimized during training.
- To perform the stochastic gradient descent on the loss function curve the selected algorithm is the classical Adam with a $learning\_rate$ = 0.01 with default values for *beta* and *epsilon*.
- Commonly, accuracy is selected as metric to be tracked during the training of this

[2] In the python implementation, these variables are called: proportions and batchsize.

[3] In the code, their names are: loss, optimizer, metrics, and epochs.

kind of classification problems.

- Finally, the number of epochs for the training procedure must be selected. Due to the simplicity of the problem, 5 epochs must be more than enough to complete the task.

Given a training dataset length of 5596 and a batch size of 100, the number of batches considered in each epoch is 5596/100 = 56 (for training) and 1599/100 = 16 (for validation data). In Fig.7.2 the training log is shown , as expected, the network captures soon the idea behind the input sequence, after the very first epoch a 100% accuracy is achieved. This fact is undoubtedly due to the easy structure of our sequence.

```
Train for 56 steps, validate for 16 steps
Epoch 1/5
56/56 [==============] - 2s 32ms/step - loss: 0.0870 - accuracy: 0.9846 - val_loss: 3.6267e-04 - val_accuracy: 1.0000
Epoch 2/5
56/56 [==============] - 0s 4ms/step - loss: 2.4257e-04 - accuracy: 1.0000 - val_loss: 1.7659e-04 - val_accuracy: 1.0000
Epoch 3/5
56/56 [==============] - 0s 4ms/step - loss: 1.4318e-04 - accuracy: 1.0000 - val_loss: 1.1539e-04 - val_accuracy: 1.0000
Epoch 4/5
56/56 [==============] - 0s 4ms/step - loss: 9.7524e-05 - accuracy: 1.0000 - val_loss: 8.2145e-05 - val_accuracy: 1.0000
Epoch 5/5
56/56 [==============] - 0s 4ms/step - loss: 7.1570e-05 - accuracy: 1.0000 - val_loss: 6.2182e-05 - val_accuracy: 1.0000
```

Figure 7.2: Summary of the training progress of our model.

7.2 State space generation

Once completed the model training, a set of weights that solves the problem has been obtained. On the other hand, from a dynamical system view, a phase space has been generated in which the state of the system moves following orbits while the network performs computations to solve the problem in response to input sequence samples.

This state evolves at each time step in response to the previous state of the system, the structural characteristics of the network and the excitatory inputs to the system. Hence, it is an input driven non-autonomous dynamic system. Thus, the trained model can be excited with the test dataset as input signal giving raise to a state space. Each sample of the input test brings about a state, i.e. a point in the phase space. Hence, as the test dataset contains 800 samples, 800 states are induced in the network hidden layer.

7.3 Dimensionality reduction technique selection

Usually, high dimensionality hidden states are employed to solve problems. As a counterpart, a new drawback appears making it difficult to visualize the states evolution through the portrait space. In our case, the hidden state is a 8D-vector. To gain insight into the dynamics of the state space some kind of dimensionality reduction must be applied with the hope of obtaining meaningful representation of reduced dimensionality.

To perform this task there is a whole battery of techniques but our analysis is limited to the most popular ones that are also implemented in the scikit-learn library:

- Principal Component Analysis (PCA) constructs a new set of variables, called Principal Components (PC), such that they capture the most variance. The goal is to obtain a reduced number of PC that explains the bulk in the variance in he original variables. Operationally it performs a matrix decomposition using Singular Value Decomposition (SVD) (Jolliffe 2002).

- Multidimensional Scaling (MDS) is based on the notion of distance between observation points in a multi-attribute space. The objective of MDS is to find a set of coordinates in k dimensions such that the distances between the resulting pairs of points are as close as possible to their pair-wise distances in multi-attribute space. Note that different measures of distance could be considered although scikit-learn only implements euclidean distance (T. F. Cox and M. A. Cox 1994).

- t-distributed Stochastic Neighbour Embedding (t-SNE) models the probability distribution of neighbours around each point. In the original, space using a Gaussian distribution, in the 2D space with a t-distribution. The goal is to find a mapping that minimizes the differences between these two distributions over all points. Hence, it is design to preserve local structure, points which are close to one another in the high-dimensional data set will tend to be close to one another in the new space. This technique was mainly design for visualization purposes not for dimensionality reduction (van der Maaten and G. Hinton 2008).

- Linear Locally Embeddings (LLE) is design to discover non-linear structure in high-dimensional data. It is based on the idea that in well-sampled manifolds a data point and its neighbours lie close to a locally linear patch of the manifold and exploits local symmetries of linear reconstructions of these patches (Roweis 2000).

  LLE appears in different flavours: standard, modified, Hessian eigenmapping, and Local Tangent Space Alignment (LTSA).

- Spectral Embedding is also a non-linear technique. An affinity graph is built where data points are the vertices and edges indicate the k-neighbours of each point in the euclidean sense, producing highly connected matrices. The low dimensional embedding is found by applying spectral decomposition to the corresponding graph Laplacian (Belkin and Niyogi 2003).

- Isometric Mapping (Isomap) is also a method based on spectral decomposition that tries to preserve the geodesic distances during the projection. Again an affinity weighted matrix is built considering the k-nearest neighbours in the euclidean sense where weights represent the geodesic distance defined as the sum of edge weights along the shortest path between two nodes. The low dimensional embedding is found by applying spectral decomposition to a derived matrix from the previous one (Tenenbaum et al. 2000).

Given such a wide variety of dimensionality reduction techniques an analysis of their suitability to our problem is needed. To achieve this goal the following steps are considered:

1) Each of the latter dimensionality reduction techniques is applied to this new state space obtaining a bunch of 3D projected spaces. The first two dimensions of each projection are represented in an 2D-scatter plot. Each point in the graph represents a state which is accompanied with a colour mark (red / blue) which represents the next symbol of the sequence which the network must try to predict. Hence, a red point represents a state of the RNN such that the next element in the sequence is a zero. On the contrary, points are coloured in blue if the next element is a one. Although the generated state spaces contains 800 states, in the plot they can appear overlapping each other as shown in Fig.7.3.

2) A second stage consisting in a visual and qualitative exploration of these plotted projections to determine which technique better suits our problem.

   Let us recall the state of the system is a representation of the input sequence built by the network to be provided to the dense layer trying to predict the next symbol in the sequence. Thus, appropriate projected states must capture, at least, part of this representation and the loss of information due to the reduction of dimensionality should be enough to generate a meaningful result.

   For a 100% score of correct predictions, the final layer must be able to separate all the points in red from the ones in blue. This perfect accuracy is achieved by our model as pointed out in Fig.7.2. Thus, we will consider that a reduction dimensionality is suitable for our problem if the plotted projections capture clearly for an human observer this fully separability property of colour groups.

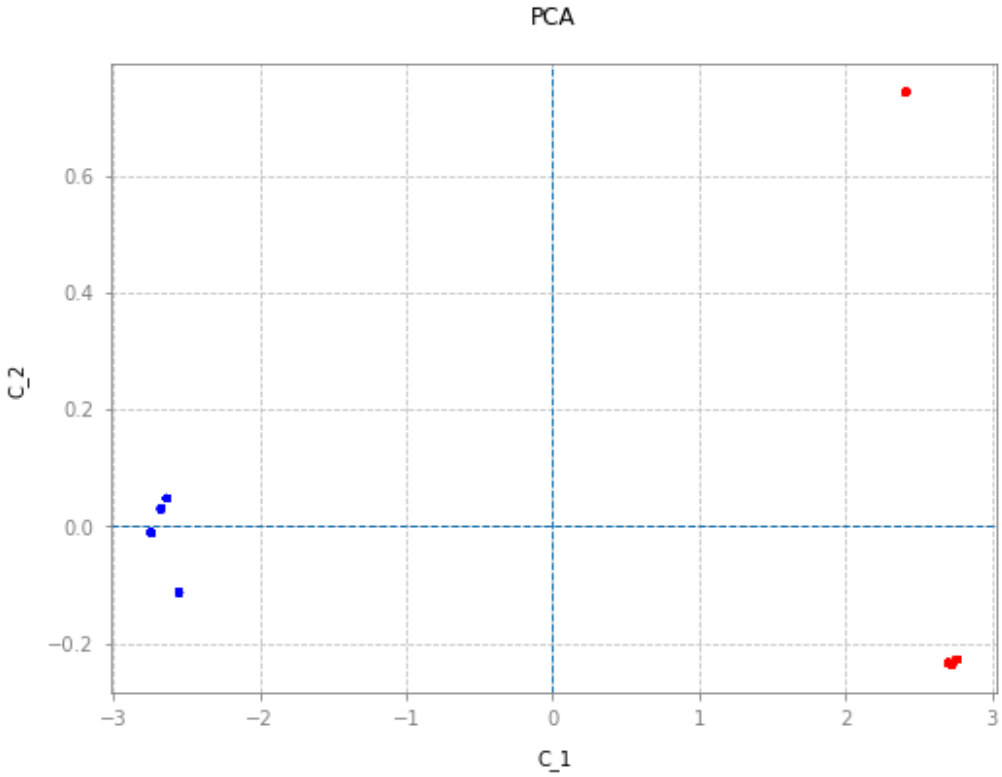


Figure 7.3: First two components of state space projected using PCA

In Fig.7.3 are represented the first two principal components of the state space of the system projected using a PCA technique. On the other hand, the first two dimensions of a MDS projection of the same state space are plotted in Fig.7.4. Undoubtedly, both techniques meet the suitability requirement providing easily interpretable plot. A simply 2D-projection is enough to separate both classes of points: the ones whose next element to be predicted is a one from those whose next symbol in the sequence is a zero.

Meanwhile, Fig.7.5 shows the first two dimensions of t-SNE and LLE Hessian projection methods. Both techniques present a pretty similar behaviour to each other. States do not appear overlapped in an small amount of points but they scattered over a delimited area in a non-separable distribution. Although, the following remarks must be done:

a) t-SNE has a tuning parameter called perplexity. Higher values (Scikit-learn default = 50) implies considering the whole dataset in the calculus while smaller values (e.g. perplexity = 5) forces the algorithm to limit the probability distribution to the nearest neighbours of each point. Unfortunately, in our case, the default value generates a messy cloud of points while lower values do not generated projections much easier to interpret.

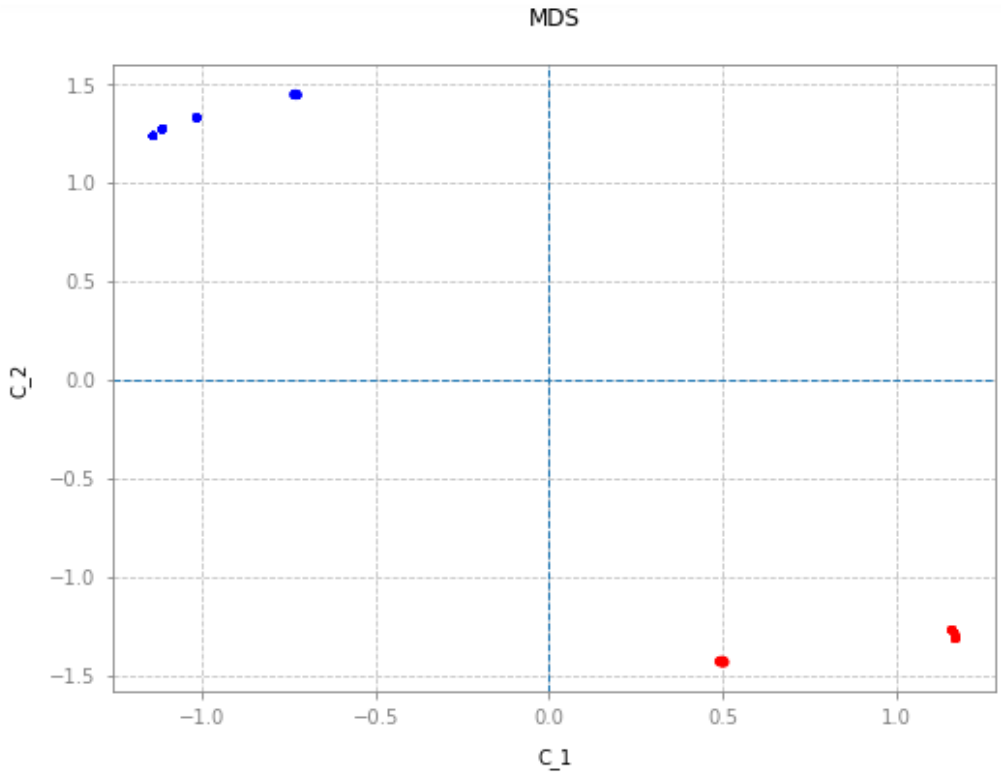


Figure 7.4: First two dimensions of state space projected using MDS.

b) For a 2D projection using Hessian LLE the minimum number of neighbours to consider is 6. Due to the reduced number of states in our problem, the technique seems to be non suitable for our problem.

In Fig.7.6 the 2D-projections obtained using the standard and LTSA flavours of LLE are shown. In both cases a similar behaviour can be observed: the states concentrate and overlap in a small number of points but in a non-separable spatial distribution. In both cases the number of neighbours selected is 4, an unit higher than the number of components of the projected space.

The following conclusions can be obtained from the qualitative analysis of the previous plots:

- Only PCA and MDS techniques provide easy and meaningful interpretable 2D projections of the state space. A key property of MDS is that close points in the original high-dimensional space remain close in the projected space. In Fig.7.4 can be seen all states have been clustered into two different areas, suggesting clearly that state space has two differentiated areas.

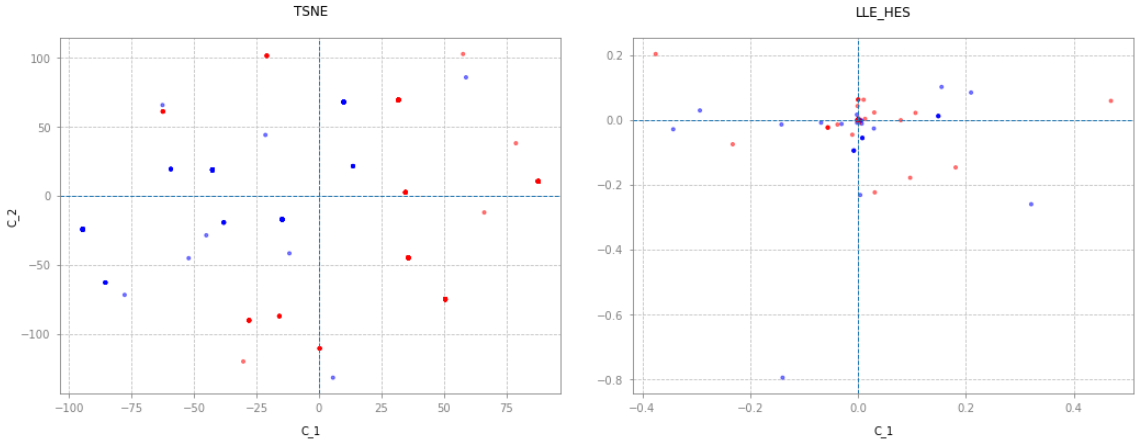


Figure 7.5: First two dimensions of the state space projected using t-SNE and LLE Hessian.

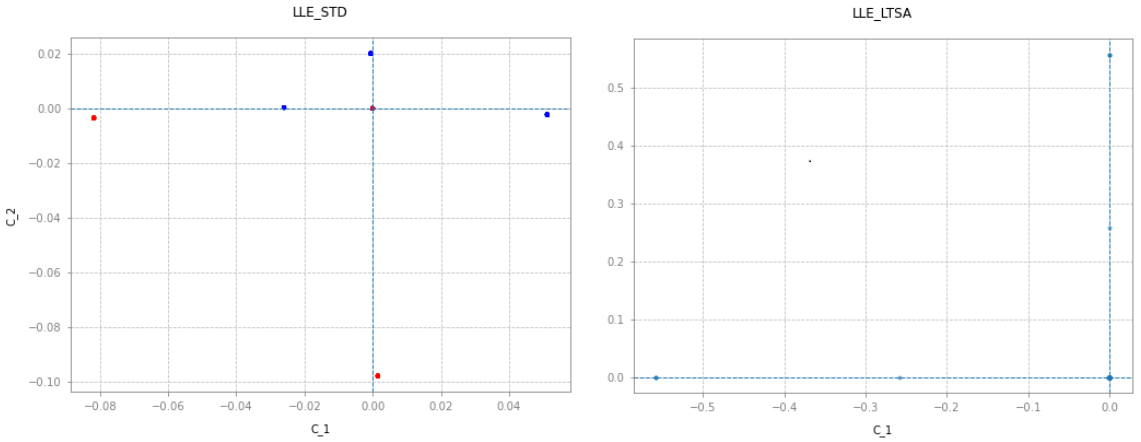


Figure 7.6: First two dimensions of the state space projected using LLE Standard and LTSA.

- In general, non-linear techniques based on local embeddings and spectral decomposition are not adequate to our concrete problem. Probably due to the reduced actual number of different states in our phase space. In future lines of investigation these techniques could be of interest as the number of states in non-toy problems probably exploit.

- Of course, these conclusions about projection techniques suitability are only of application for this concrete problem under analysis. For this reason, all these projection techniques have been implemented in our library for future potential uses. The Object Oriented approach in the library design makes it easy to implement new projection techniques using polymorphism.

7.4 Working minimum dimensionality

Once selected PCA and MDS as the useful projection techniques for our problem, the next natural question is to determine how many dimensions or components are really needed to work with.

In the PCA case, the most common technique to determine the best dimension number is plotting the accumulated explained variance associated to the eigenvalues obtained in the singular decomposition ordered decreasingly by their value as shown in Fig.7.7. A common convention is to take a threshold of 95%. The application of this criteria to our problem suggests that 3 components explain almost 100% of the overall variance.

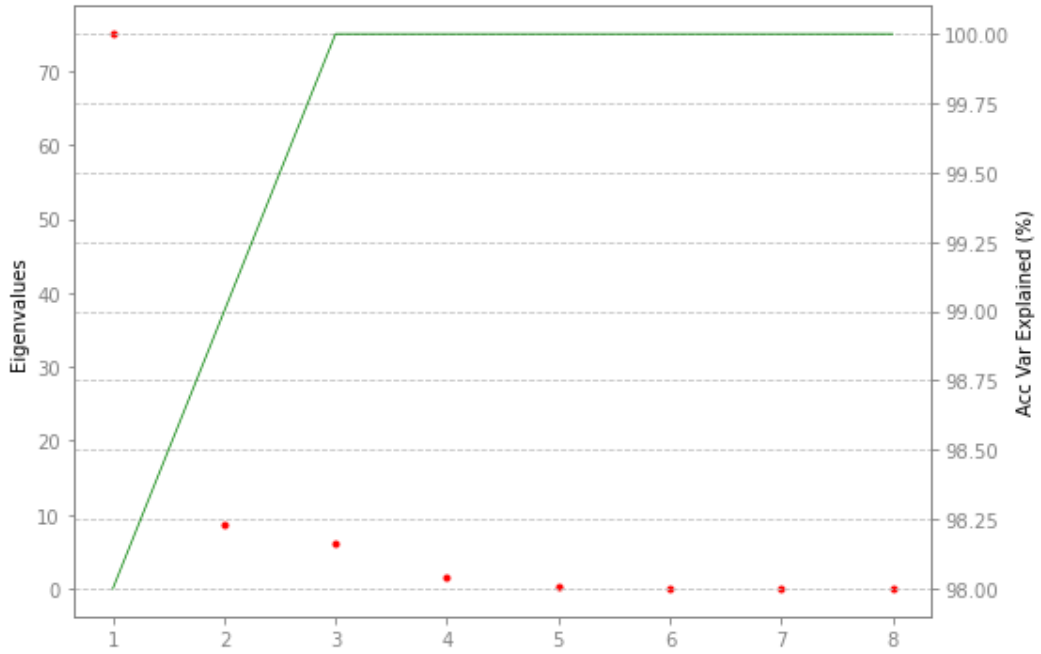


Figure 7.7: Accumulated explained variance of each eigenvalue for the selection of the adequate number of PCA components

An alternative criterion to select the number of components is the so-called Kaiser criterion, which suggests to take the components for which the eigenvalue exceeds 1. For this purpose, the absolute value of each eigenvalue has been added in Fig.7.7. Again, this technique suggests to select three or four components. Hence, in the following chapters a 3D-PCA will be considered.

For the MDS case, there are no simple methods to select an adequate dimension number

but calculating the so called reconstruction error. Thus, taking into account the existing parallelism between PCA and MDS techniques, also 3D-MDS will be considered.

These results confirms Sussillo's hypotheses: after training the original 8D-RNN, the obtained space state has a movement constrained to a 3D sub-manifold (i.e. the states yield in a curve) that can be analysed using 3D-PCA and 3D-MDS.

Chapter 8

State space orbits

Up to this point we have obtained the following advances in our analysis:

1) A vanilla-RNN has been trained to solve the next symbol in sequence estimation for a $0^4 1^4$ pattern input sequence. Different parameters of the network and the input sequence have been fixed. Among them, the most relevant for our analysis is the number of neurons in the hidden layer that determines the dimensionality of the state space. For our problem, a 8D-space.

2) The resultant trained network can be analysed from a dynamical system theory perspective. In this context, the hidden layer activations can be interpreted as the state of an input driven non-autonomous dynamical system that evolves as a response to an input sequence to the system.

3) To obtain the orbits of the non-autonomous system, the trained model is exposed to an excitatory signal. In our case, the test dataset plays the role of input excitation. Given the repetitive structure of the input the whole state space is composed by an unique orbit with eight different states.

4) Due to the high dimensionality of the 8D-state space, different dimensionality reduction techniques have been analysed. Linear Local Embeddings and t-SNE have been discarded due to an incompatibility between their dimension reduction approach based on neighbours and the reduced number of states in our problem. As a result 2D/3D-PCA and 2D/3D-MDS have been selected as suitable for our task obtaining separable representations of the states and capturing more accurately the state space manifold structure of this concrete problem.

5) The low dimensionality hypothesis proposed by Sussillo's for the state space of high dimensional trained models in real problem has been confirmed for our toy example. The orbits followed by the model while performing the computations are constrained to surfaces in a 3D space.

8.1 Trained network: orbits in PCA space

The next logical step is to closely analyse the state space taking advantage of the visualization possibilities of the selected 2D and 3D projection techniques. From Eq. 4.2 each incoming input sample jointly with the previous system state contributes to generate a new state, i.e. a new point in the phase portrait. Hence, when the next input sample is received, the system suffers a transition from its current state to the next state in the orbit.

As shown in the previous chapter, the state space of the trained model is generated injecting in the system the test dataset and obtaining its reaction. Therefore, it is necessary to understand the nature of this input signal.

Its samples are the ones associated to a $0^4 1^4$ grammar. Thus, it consists in a 8-sample cycle that repeats indefinitely:

$$..., 01111, 11110, 11100, 11000, 10000, 00001, 00011, 00111, ... \tag{8.1}$$

In Fig. 8.1, 8.2 and 8.3, the 2D and 3D-PCA projections of the states in the orbit travelling through the state space are represented. Each pair of subplots is the reaction of the network to a new input sample received following the pattern in Eq. 8.1.

To make the visualization of these states and their transitions clearer some additional elements have been incorporated to the plots:

- As each subplot in the figures must be understood as the rise of a state in response to a new input data exciting the system, the resultant state is labelled with the related input sample value.
- States are also marked in a colour indicating the correct next symbol in the sequence that must be predicted by the network. If a point is in blue the next element in the sequence will be a zero. On the other hand, if the point is in red, the next element will be a one.
- Finally, let's also recall that in discrete time dynamical systems, the orbits are sequences of points instead of continuous curves. Given a discrete state space, this issue makes it difficult to visualize the order of appearance of the states in the orbits. Therefore, transitions from one state to another one are indicated with a green dotted line joining consecutive states. This provides us with a geometrical intuition of the pattern followed by the transitions.

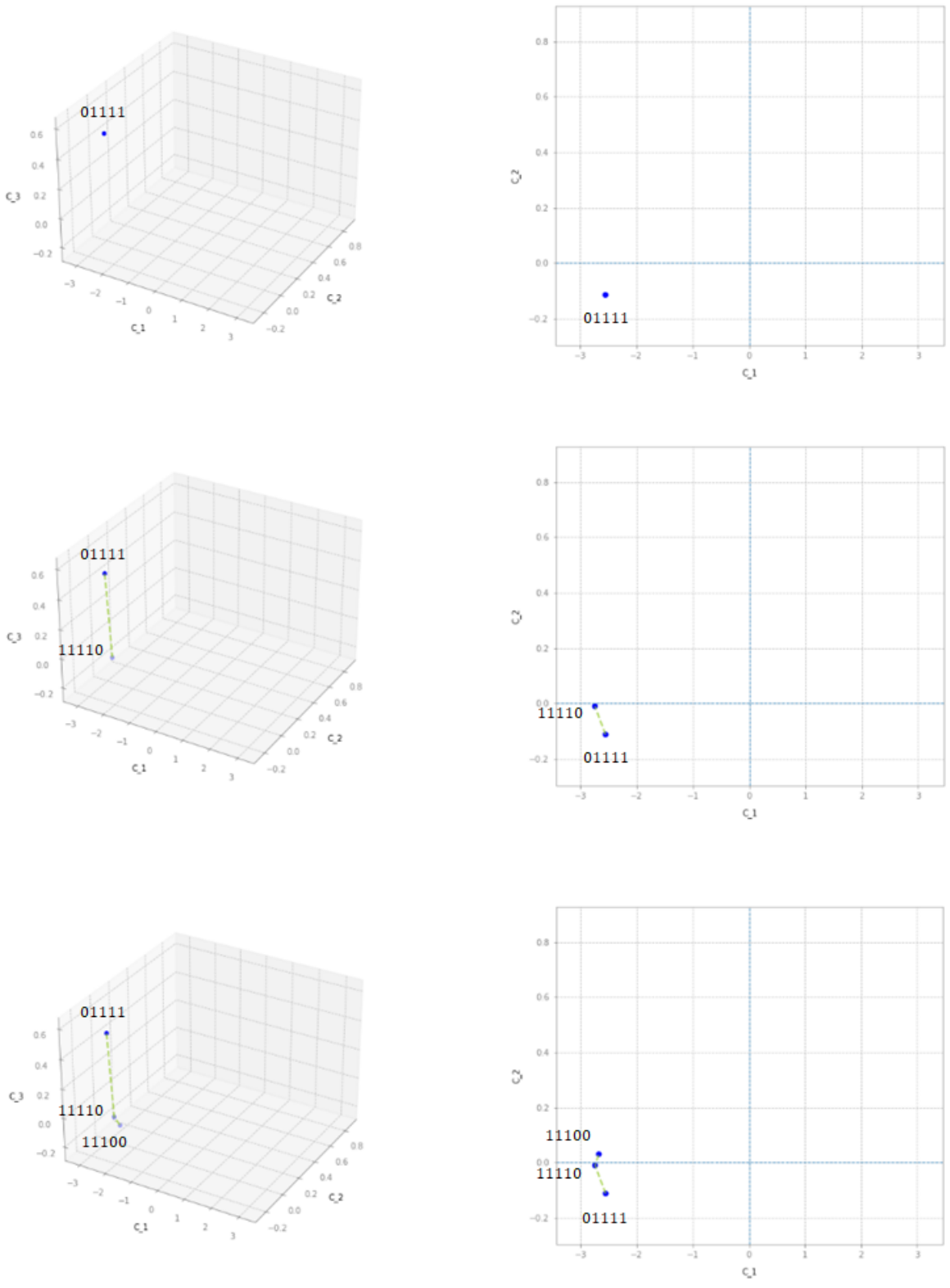


Figure 8.1: 2D/3D PCA projections of states associated to the first three input samples. Top: 01111, Center: 11110, Bottom: 11100.

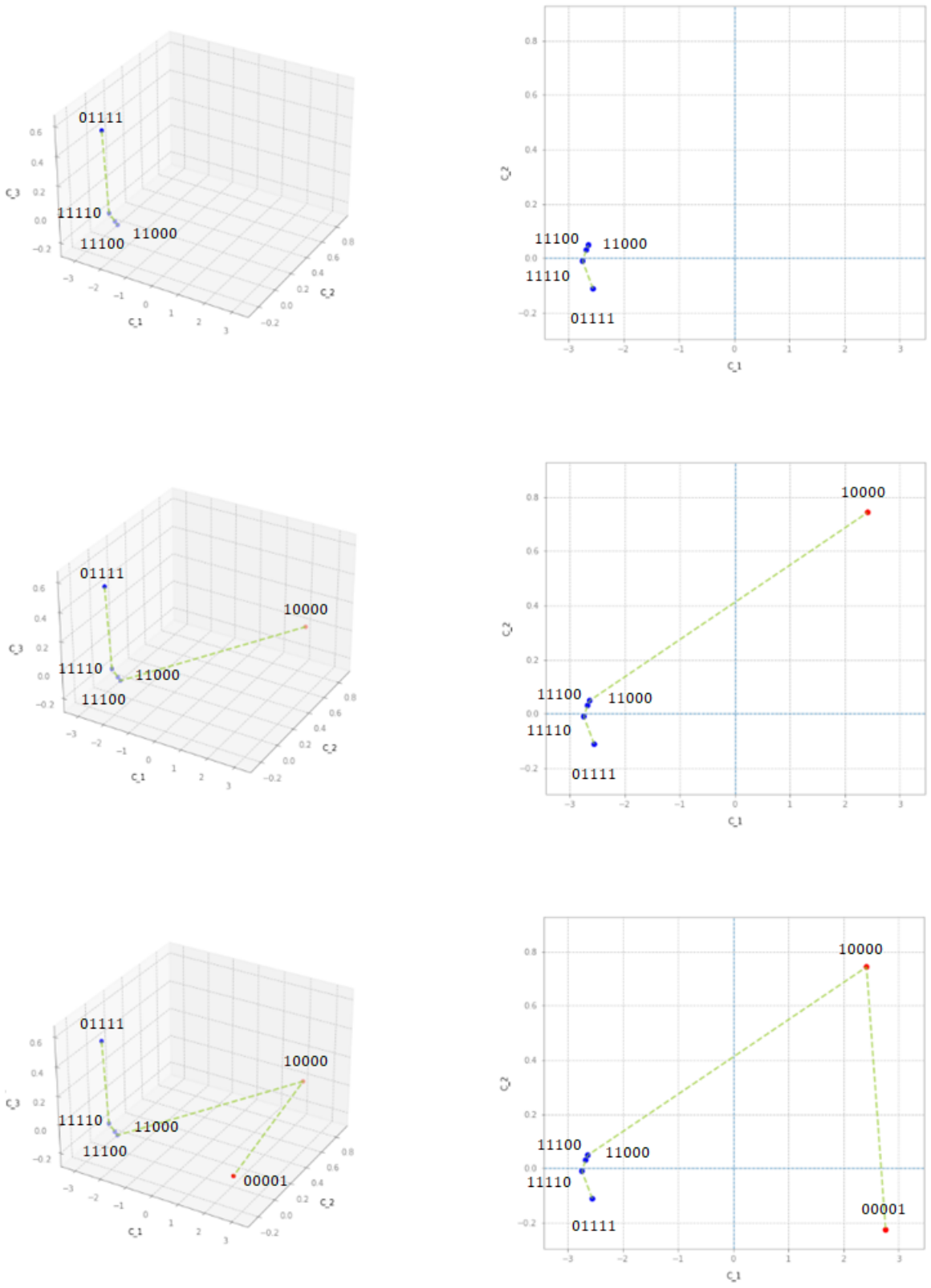


Figure 8.2: 2D/3D PCA projections of states associated to the following three input samples. Top: 11000, Center: 10000, Bottom: 00001.

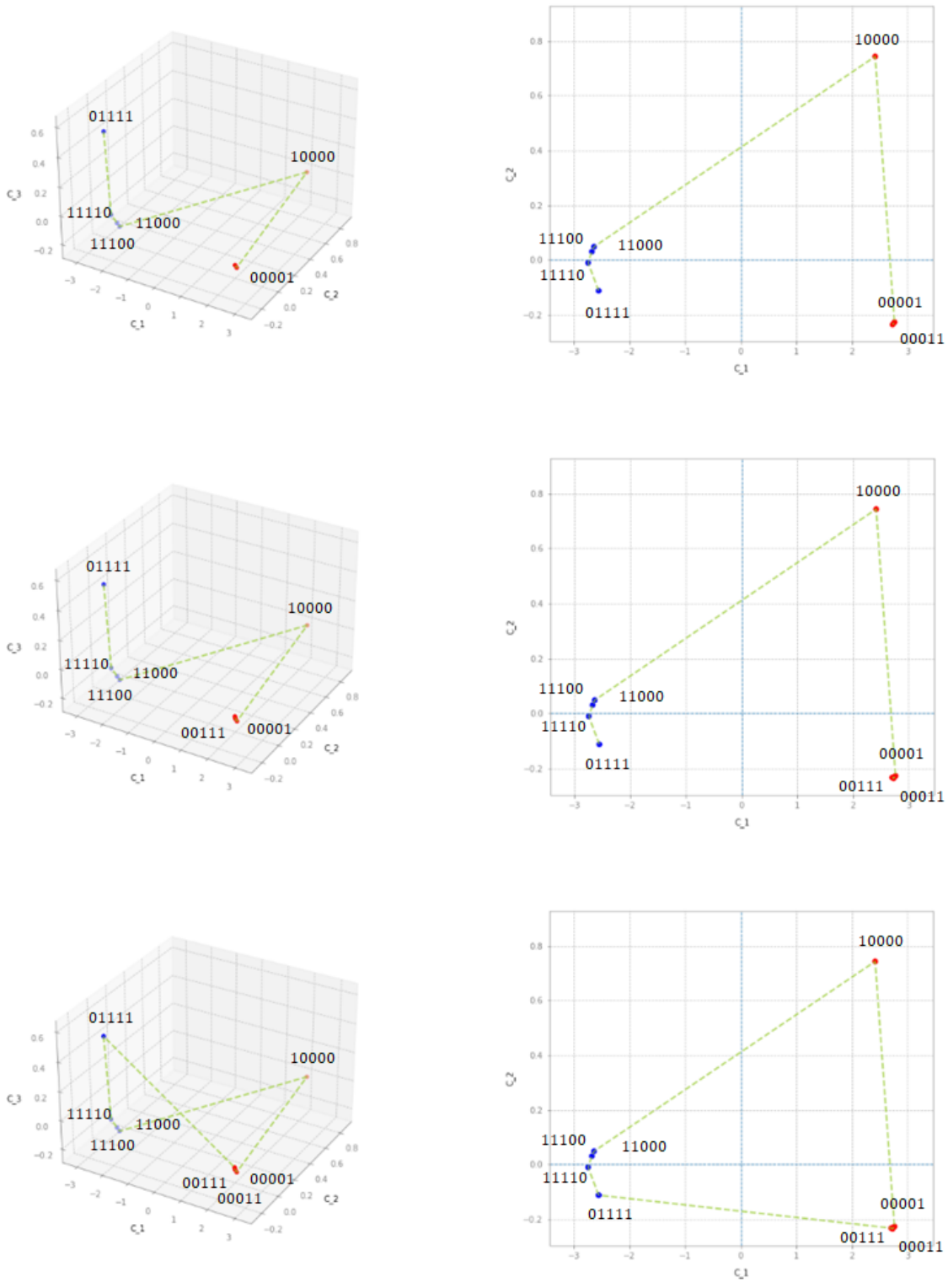


Figure 8.3: 2D/3D PCA projections of states associated to the last input samples. Top: 00011, Center: 00111, Bottom: 01111.

8.2 Interpretation of the principal components

From a closer inspection of Fig. 8.1, 8.2 and 8.3 a nice interpretation can be given to the obtained principal components. For that purpose, consider the complete orbit represented in Fig. 8.4:

i) Firstly, from the 2D-representation it is clear that the first principal component is enough to separate all the points in blue from the ones in red. Therefore, it can be directly interpreted as an indicator of the next element in the sequence. The positive direction of the first component indicates the next element in the sequence will be a one and the opposite direction indicates the next element to be predicted is a zero. Lonely this first component would be enough to solve our classification problem. This result is coherent with the high variance (approx 98%) captured by this first principal component in the projection as shown in Fig. 7.7.

ii) Second and third principal components interpretation is much trickier and must be done jointly. In Fig. 8.4 four areas (named RI, RII, RIII and RIV) with different behaviours can be identified in the state space.

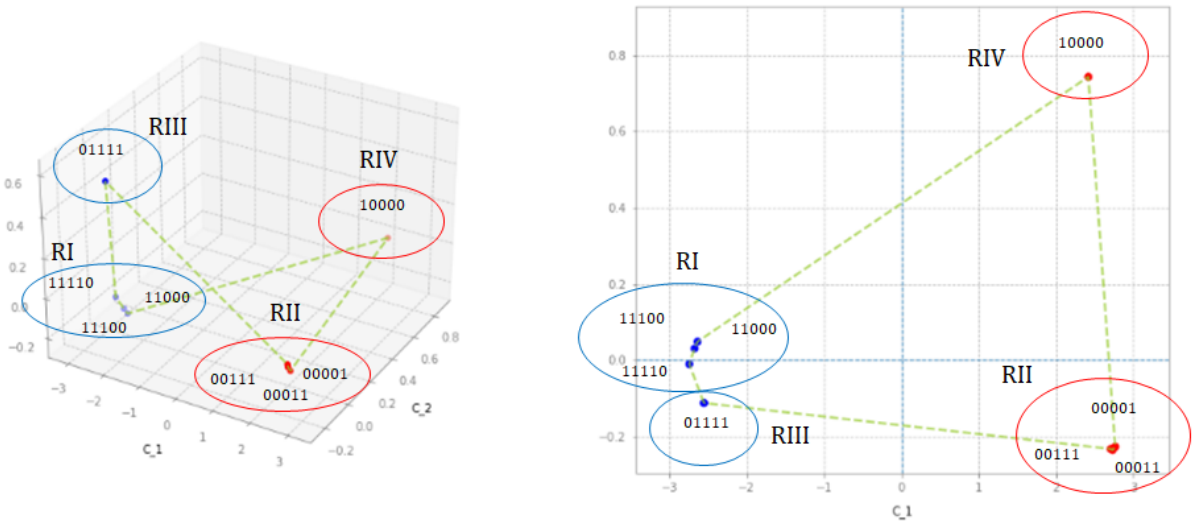


Figure 8.4: Topological structure of the trained network state space projected using PCA. Left: 2D projection, Right: 3D projection. Four regions with different behaviour are identified.

While the received input signals are 11110, 11100,11000 the system has no doubt the next element to be predicted is a zero and the states in 2D-3D projections remain grouped in a cluster labelled as RI, i.e. all three component values suffer only tiny changes. However, when the input signal received is 10000, of course, the first component changes indicating a one for the prediction but the second component also varies abruptly to RII giving signals that a switch in the status of the sequence has happened.

From this point on, the next element in the sequence to be predicted is a one. After this transition and while the input signal is one of the following 00001, 00011, 00111 all three components relax again to a new cluster of points RIII where they suffer minimal changes.

Finally, when the last input signal 10000 arrives, the first component changes indicating a zero for the prediction but now the third component is excited and suddenly changes to RIV indicating again the status of the sequence has changed.

Hence, second and third principal components capture the idea of a change in the status of the sequence has happened in the very near past, i.e. exactly from the previous state to the current one. In concrete, second component is an indicator of a recent switch in the sequence from zero to one while third component is the complementary one, indicating a transition in the prediction from ones to zeros.

8.3 Trained network: orbits in MDS space

In the previous plots we have analysed the behaviour of orbits in the state space of the trained network with the help of 2D/3D PCA projections. In Fig. 8.5 is shown a comparison between the same 8D-orbits considering both reduction techniques: 2D/3D-PCA and 2D/3D-MDS.

As mentioned earlier there exists a parallelism between PCA and MDS projections. This behaviour can be appreciated in the previous figure where the same 8D orbit is projected considering both projection systems. A qualitative inspection of the plots gives us the following intuitions:

- The first dimension of MDS plays a similar role to the first principal component of PCA. Allowing us to separate both families of states, the ones in red from the blue ones.
- The second and third components are more difficult to interpret but it seems there is an underlying similar but softened behaviour between both projections methods.

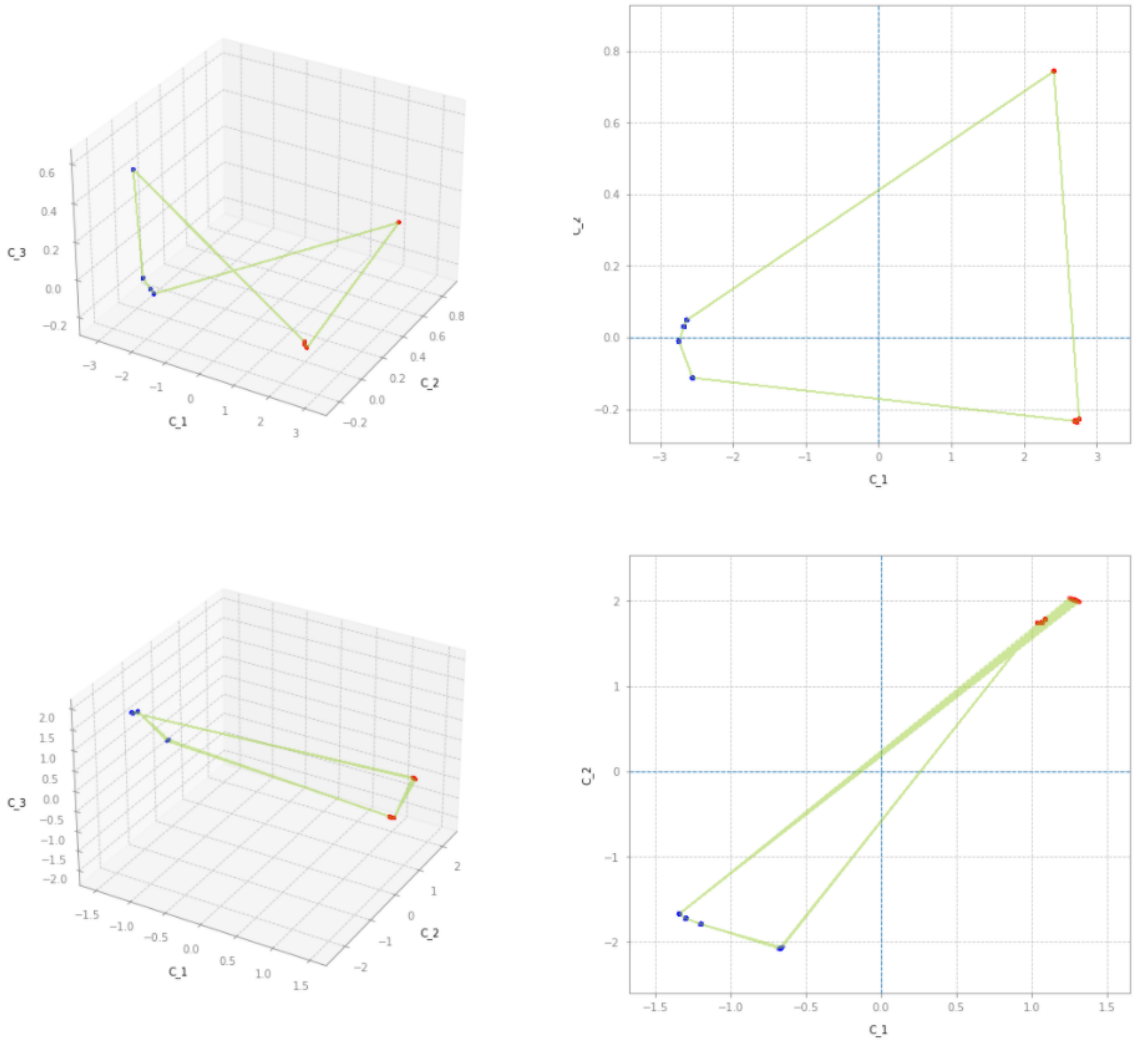


Figure 8.5: Comparison of 2D-3D PCA and MDS projections of trained network state space. Top: PCA projections. Bottom: MDS projections.

8.4 Orbits deformation during network training

In the previous section the orbits of the trained network have been analysed. The following natural question is what has happened to the orbits during the training process.

Roughly speaking the network training process consists is a progressive adjustment of its weights following some learning process. Although significant advances has been made in recent years, the traditional default technique is BackPropagation Through Time (BPTT) with some incorporations. Nevertheless, any training mechanism makes successive transformations of the different weight matrices trying to optimize some loss function.

Typically, weights are not updated after each input sample but training data is divided into blocks called batches. This batch size is an user adjustable parameter before training in any deep learning library. Once completed the forward step for the entire batch, the error in the loss function is computed and weights are updated following the backpropagation idea. Thus, during training there will be as many weight updates as batches are involved.

Each weight arrangement can be interpreted as a state space in progress towards the final solution. These state spaces can be inspected to obtain its orbits injecting an excitation input following the same procedure used in the previous sections. Following this idea, the training process can be understood as an orbits deformation procedure until the training process is completed and a final space space is reached.

To visualize this process for our problem on hands a batch size of 100 was selected for a training dataset containing 5.596 samples. Hence, the number of steps in the training procedure is $5596/100 \approx 56$ as indicated in Fig.7.2. These 56 steps are performed for each of the 5 epoch selected for training. Hence, the total number of weight updates is $56 \times 5 = 280$.

Using Tensorflow callbacks an snapshot of model weights after each batch has been stored. Thus, 280 models are registered during training. The 8D-state space of each of these models is obtained applying the same excitation input sequence. Finally, these high dimensional state spaces are projected using the 3D-PCA technique. In the associate notebook all the results of this thesis are made reproducible and an animation of the orbits deformation during training is included. In Fig.8.6 are shown the orbits for some steps during training:

- The initial step corresponds to the random weights assigned to the weights during net initialization. In Tensorflow the default method is called Glorot uniform which draws samples from a uniform distribution based on the number of input units and output units in the weight tensor. In this initial step states are randomly mixed in the plane.
- Even after some initial steps the first component has captured the idea of separating ones and zeros. It is the first principal component birth.
- In the next steps the learning mechanism shrinks and expands the orbit until the first component captures the maximum variance creating the clusters named as RI and RII. To create these regions the orbit has suffered major deformations.
- In the last steps, RI and RII clusters are compressed as much as possible and second and third component try to capture their vicinity change idea creating RIII and RIV. Finally, the orbit structure stabilizes as no more improvement in the information capturing process can be obtained. This is coherent with the classical information returned by Tensorflow during training shown in Fig.7.2 where after the first epoch, the problem is solved by the network.

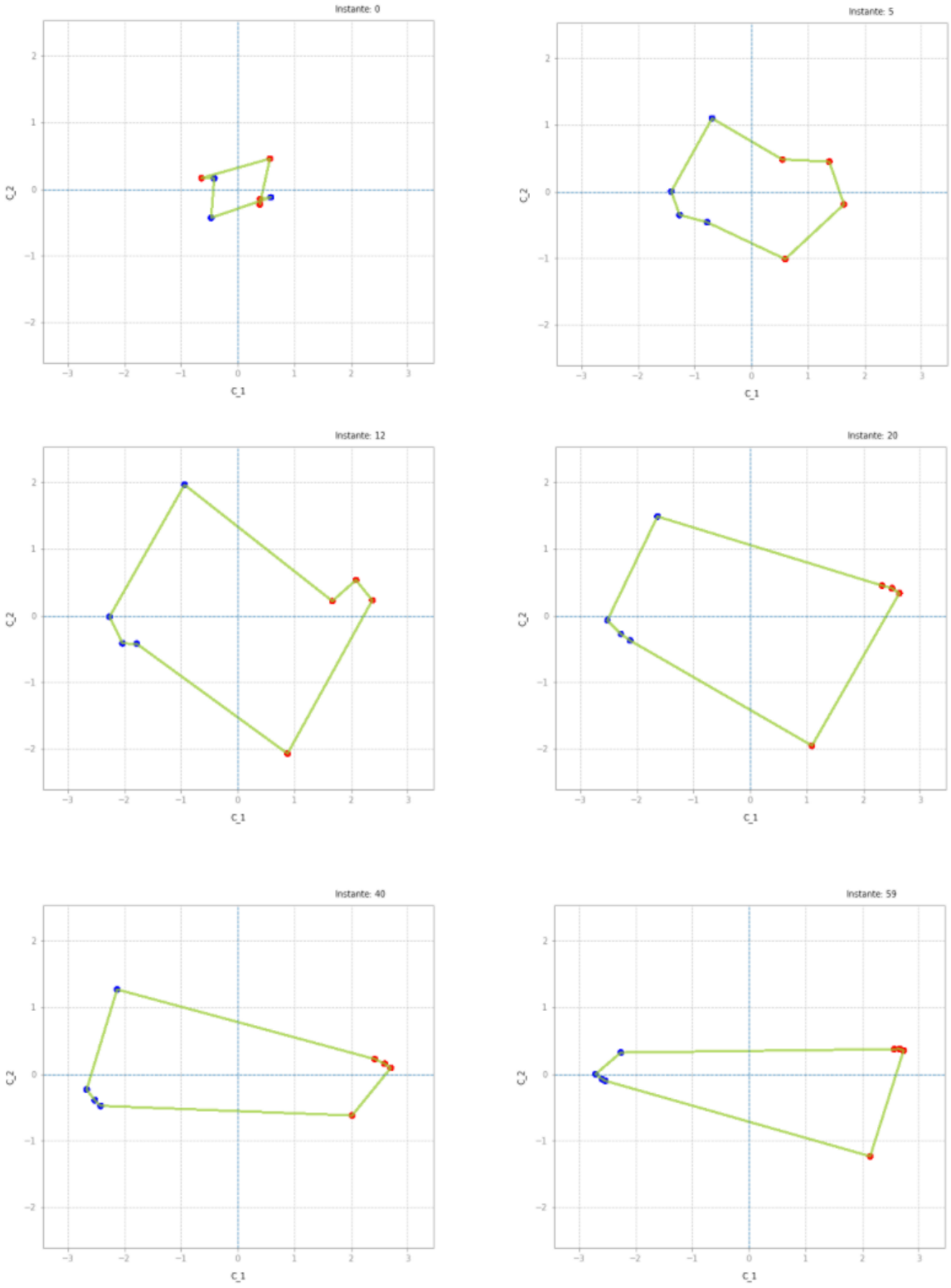


Figure 8.6: Orbits deformation during network training. Top left: random weight initialization, Top: right: first PC can be appreciated (5 steps), Center left: to create clusters orbit needs to deform (12 steps), Center right: clusters created (20 steps), Bottom left: Information transferred between RIII and RIV (40 steps), Bottom right: Near orbit stability (59 steps $\approx$ 1 epoch)

Chapter 9

Line attractors

9.1 Attractors visualization

In the previous chapters we have analysed the phase space for trained networks and the orbits deformation during weights learning process until they get stabilized when training ends.

However, we have tiptoed around an important issue. The initial step of this training process implies a random initialization of model weights. There exists different methods to assign values to the initial weights but they usually consist in taking samples from a statistical distribution (typically Gaussian or uniform) of weights (Glorot and Yoshua Bengio 2010). The main objective of this technique is to ensure a similar variance in the gradients of the different layers.

To obtain these random samples from the distribution a seed is considered in the Python, Numpy and Tensorflow internal random number generators. Hence, an interesting question to tackle is, which is the behaviour of our model when different seeds are considered as starting points for training.

For this purpose we have obtained the orbits for a range of 20 different seeds taken from a uniform distribution. The PCA projections are obtained for each state space individually[1] and the superposition of the first two components is represented in Fig.9.1.

Two clearly differentiated regions can be identified. One region that concentrates or attract states in blue and another one where the remaining states in red are clustered. The obtained distribution of states is quite close to a line, When an attractor behaves in this manner is called a line attractor. This behaviour is similar to the described in Maheswaranathan et al. 2019 for the existence of a line attractor in the classification problem associated to

[1]Each state space has been projected individually and the first two components are superposed and plotted. Another approach could be chosen, adding all the state spaces of the different seeds and obtaining an unique projection

sentiment analysis.

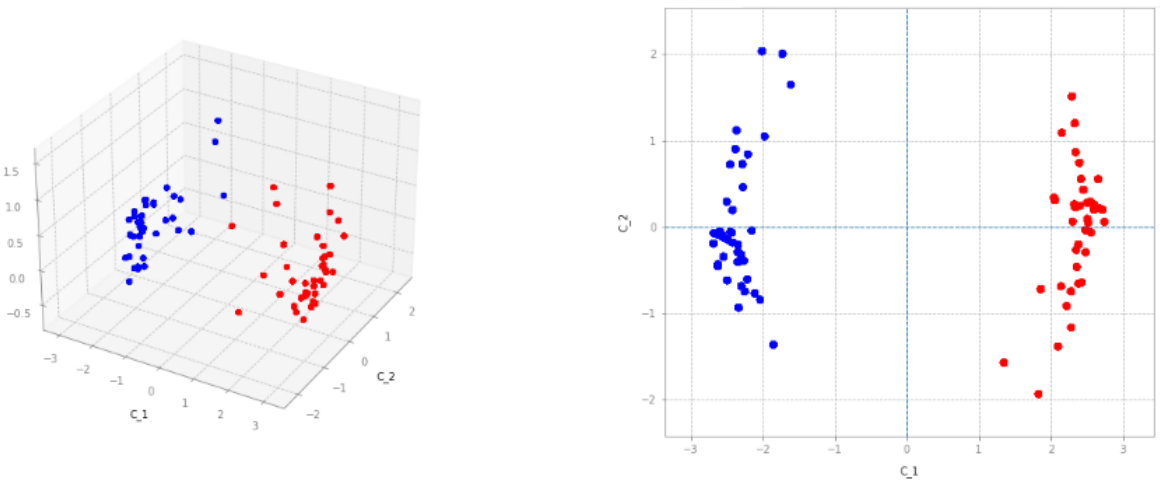


Figure 9.1: Superposition of state space PCA projections of trained networks obtained for 20 different seeds in the weight initialization process.

9.2 Parametric analysis: hidden layer dimensionality impact

As indicated in section 6.3 one of the main parameters of a RNN is the hidden layer dimensionality. However, all practical results obtained in the previous chapters considered a fixed 8D hidden layer. Thus, it seems reasonable to analyse the behaviour of orbits and attractor lines when this dimensionality takes different values.

For this purpose, the orbits and line attractors are obtained for different values in the range $n_{hidden} \in \{3, \ldots, 16\}$. In the accompanying notebook to the thesis [2] is presented an animated dynamic evolution to appreciate clearly its impact. As the hidden layer dimensionality takes higher values, the intuitive distance between both attractor lines increases.

In Fig.9.2 the line attractors for eight different values of n_{hidden} are represented. Consider, simply as an intuitive notion of distance, the difference between the intersection points of the first principal component axis and each line attractor. For $n_{hidden} = 3$ this distance is roughly 4 units while for $n_{hidden} = 16$ the distance increases up to 8. Hence, clearly there exists a relationship between the hidden dimensionality and some kind of distance metric between the attractors.

[2] In the github repository all the results presented in this thesis can be reproduced using a notebook.

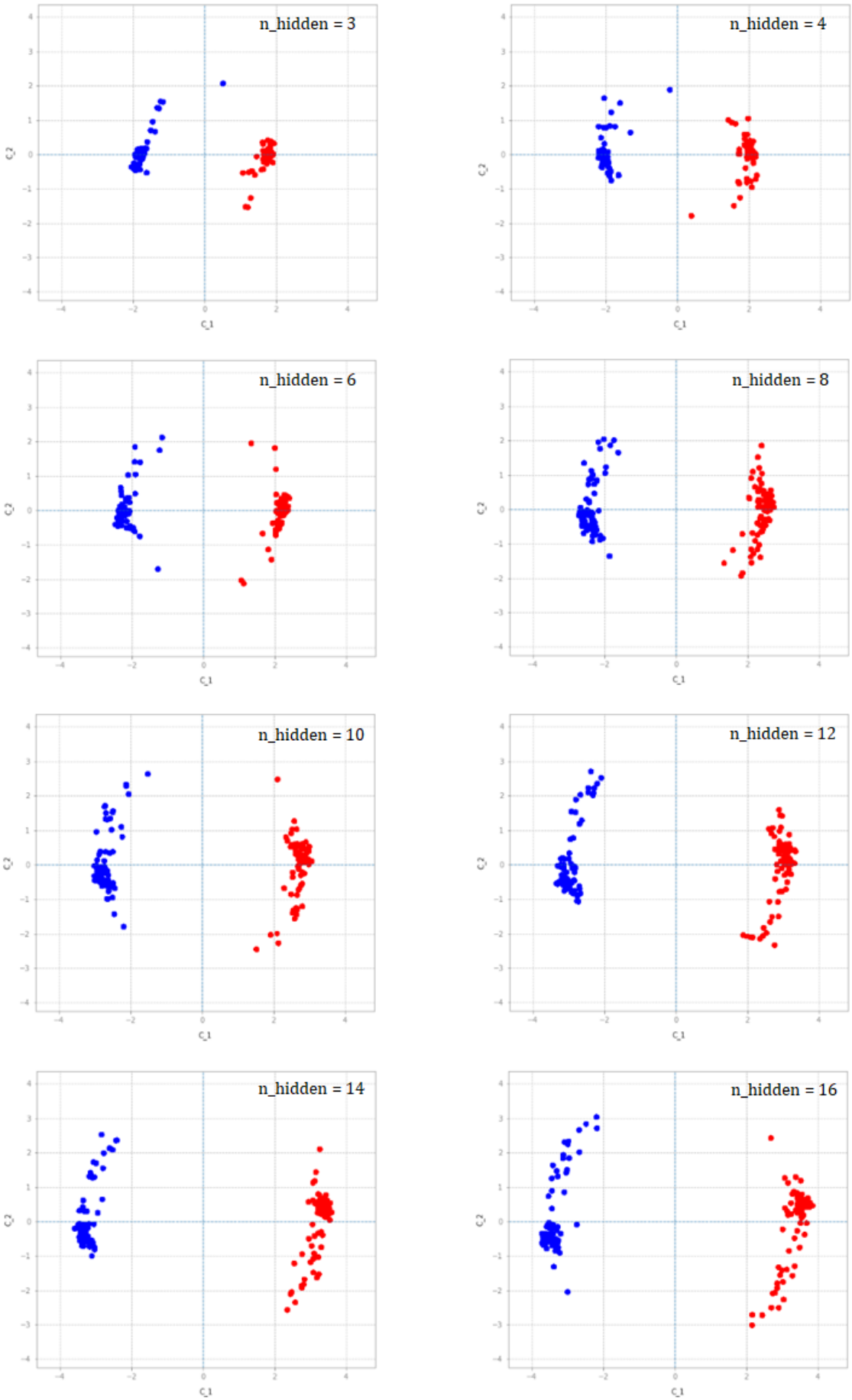


Figure 9.2: Line attractors obtained for different values of hidden dimension

In some sense, although PCA explained variance analysis assigns nearly null variance to dimensions higher than 3, during the training process the network makes use of these extra dimensions to increases its confidence in the results obtained creating more separated line attractors.

Chapter 10

Noise impact on line attractors

In this final chapter noise in the input sequence is considered and its impact on the state space topology is analysed.

Let's consider a noisy input signal obtained adding a random perturbation to the original sequence. For the sake of simplicity, given our original sequence $0^4 1^4$, only noise composed by 0's and 1's is considered. Each symbol of this noise is obtained sampling a uniform discrete distribution with parameter $p = error\_prob$. The higher the $error\_prob$, more noise is acting. The perturbed input is obtained with a modulo-2 addition between the original signal and the random noise, i.e. when noise takes value 1 error is acting on the input, if noise is 0, there is no noise.

To observe the impact of this noise in the attractors topology, for each value of p ranging from 0 to 0.5 ($step = 0.05$), a noisy input sequence is generated, i.e. from a no-noise situation, until a high noise impact scenario. These 9 perturbed input signal are used as input for training our RNN model. As in the previous chapter, to observe the line attractors, the training process is performed considering 20 different weight initialization seeds. As usual, the model and the learned weights can be interpreted as a state space whose orbits can be visualized injecting an excitatory signal in the network. Finally, these state spaces are projected considering a 2D-PCA technique.

In Fig.10.1 the obtained line attractors for each error probability are represented.[1] When there is no noise presence, the situation is similar to Fig.9.1. Immediately when noise appears, the attractor structure gets curved due to a greater sparsity of the states. As error probability is increased, the distance between the attractors diminishes. Finally, when error reaches 0.4 both attractors are clustered near the origin and beyond this point, the original structure of attractors reaches a completely unrecognizable situation.

[1] As in previous chapters, in the github notebook an animated representation of the attractors evolution is provided

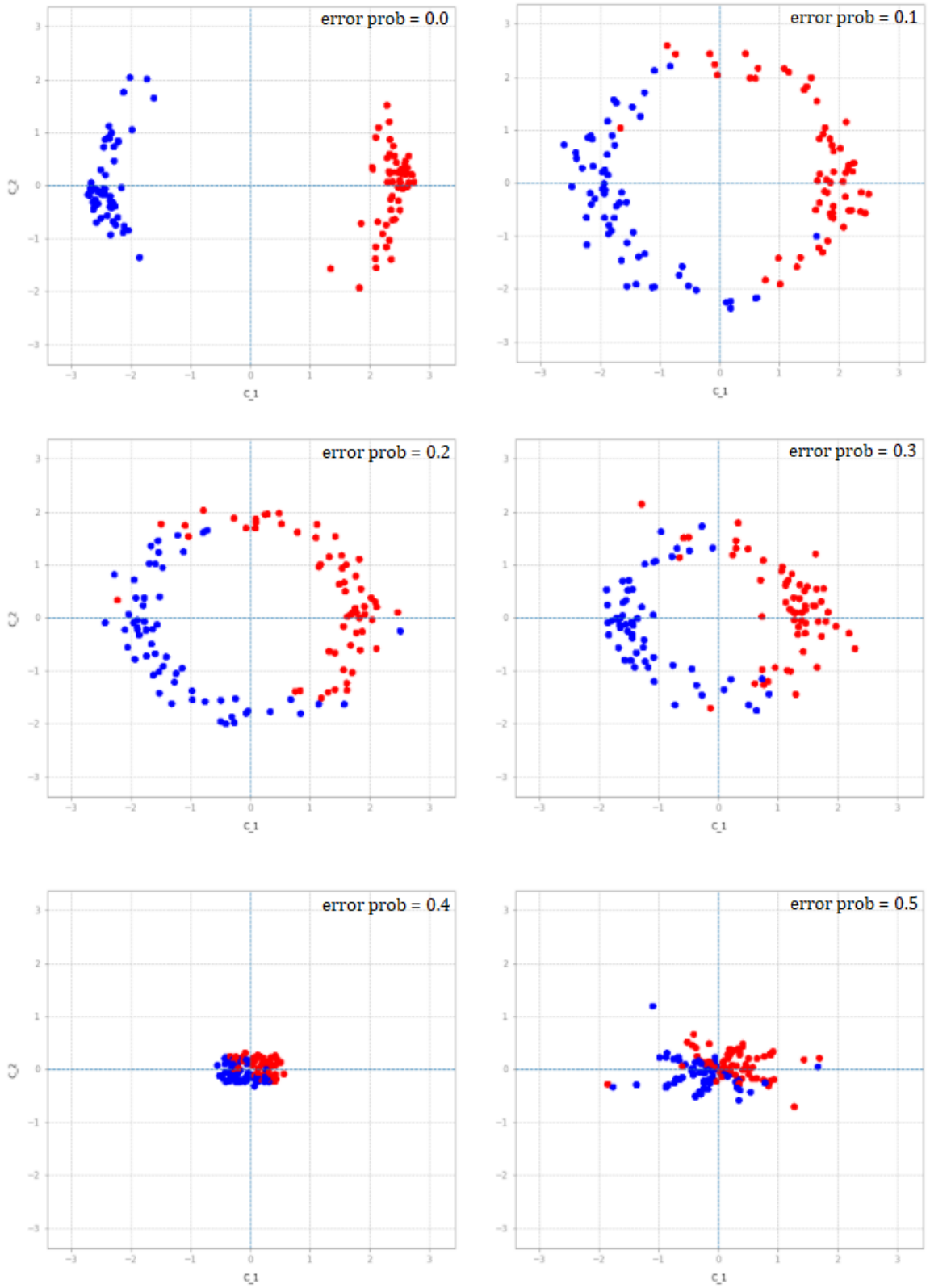


Figure 10.1: Linear attractors evolution for network trained with noisy dataset. Noise ranges from error probability = 0 to 0.5

Chapter 11

Conclusions and future investigation lines

Despite the success history of deep learning application to real-life problems, the black-box idea associated to neural networks remains left on hold. It is becoming a must to understand the internal mechanisms behind the results provided by these models. Moreover, when the design and implementation of neural networks is almost an art, based on heuristic rules not based on a solid mathematical framework.

In this work a complementary approach to RNNs analysis is presented: the interpretation of Recurrent Neural Networks as discrete time non-autonomous dynamical systems. This idea allows us to apply powerful tools of this branch of mathematics. State spaces, orbits and attractors can be used to get a deeper knowledge of how information is represented during network internal computations.

To illustrate these ideas a toy example is considered. A high dimensional (8D) vanilla-RNN has been trained to solve the next symbol in sequence estimation for a $0^4 1^4$ pattern input sequence. The hidden layer of the networks is interpreted as the state of the system that evolves as a response to an input sequence and several dimensionality reduction techniques are analysed to obtain useful 2D and 3D representation of the phase space. Using PCA and MDS can be observed that the state space of the trained model is constrained to low dimensional spaces. The progressive deformation of the orbits during the training process until the final state space is reached is also studied. taking into account different seed for the weight initialization process, two clearly separated regions can be identified in the space portrait, interpreted as line attractors. The distance between these two attractors depends on the hidden layer dimensionality. If noise is introduced in the training process, the attractors topology is altered spreading the attractors.

Future lines of investigation must consider increasingly complexity problems or alternatively reproduce the results of state-of-art papers in this field. For this purpose the developed library must be completed with more functionalities. In general, more analysis with toy examples can be done as the impact of the training method on the state space or an increase in the number of classes.

Bibliography

Baktha, K. & Tripathy, B. K. (2017). Investigation of recurrent neural networks in the field of sentiment analysis. *2017 International Conference on Communication and Signal Processing (ICCSP)*, 2047–2050. https://doi.org/10.1109/ICCSP.2017.8286763

Belkin, M. & Niyogi, P. (2003). Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, *15*(6), 1373–1396. https://doi.org/10.1162/089976603321780317

Bengio, Y. [Y.], Frasconi, P. & Simard, P. (1993). Problem of learning long-term dependencies in recurrent networks. *Proceedings IEEE International Conference on Neural Networks*, *3*, 1183–1188. https://doi.org/10.1109/ICNN.1993.298725

Bengio, Y. [Y.], Simard, P. & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Casey, M. (1996). The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction. *Neural Computation*, *8*(6), 1135–1178. https://doi.org/10.1162/neco.1996.8.6.1135

Ceni, A., Ashwin, P. & Livi, L. (2019). Interpreting Recurrent Neural Networks Behaviour via Excitable Network Attractors. *Cognitive Computation*, 330–356. https://doi.org/10.1007/s12559-019-09634-2

Chandar, S., Sankar, C., Vorontsov, E., Kahou, S. E. & Bengio, Y. (2019). Towards Non-Saturating Recurrent Units for Modelling Long-Term Dependencies. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 3280–3287. https://doi.org/10.1609/aaai.v33i01.33013280

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. https://doi.org/10.3115/v1/D14-1179

Cox, T. F. & Cox, M. A. (1994). *Multidimensional scaling* (1. ed). Chapman & Hall. https://doi.org/10.1007/978-3-540-33037-0_14

Doya, K. (1993). Bifurcations of Recurrent Neural Networks in Gradient Descent Learning. *IEEE Transactions on Neural Networks*, *1*, 75–80.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/10.1207/s15516709cog1402_1

Farabet, C., Couprie, C., Najman, L. & LeCun, Y. (2013). Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1915–1929. https://doi.org/10.1109/TPAMI.2012.231

Glorot, X. & Bengio, Y. [Yoshua]. (2010). Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS'10).*, *9*, 249–256.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biological Cybernetics*, *23*(3), 121–134. https://doi.org/10.1007/BF00344744

Haschke, R. (2003). *Bifurcations in discrete-time neural networks : Controlling complex network behaviour with inputs* (Doctoral dissertation). Bielefeld University. Germany. https://pub.uni-bielefeld.de/record/2302580

Haschke, R. & Steil, J. J. (2005). Input space bifurcation manifolds of recurrent neural networks. *Neurocomputing*, *64*, 25–38. https://doi.org/10.1016/j.neucom.2004.11.030

Haschke, R., Steil, J. J. & Ritter, H. (2001). Controlling Oscillatory Behaviour of a Two Neuron Recurrent Neural Network Using Inputs. *ICANN 2001 - International Conference in Artificial Neural Networks*, 1109–1114. https://doi.org/10.1007/3-540-44668-0_154

Hebb, D. (1949). The organization of behavior: A neuropsychological theory. *The Journal of Comparative Neurology*, *93*(3), 459–460. https://doi.org/10.1002/cne.900930310

Hewamalage, H., Bergmeir, C. & Bandara, K. (2020). Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*. https://doi.org/10.1016/j.ijforecast.2020.06.008

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558. https://doi.org/10.1073/pnas.79.8.2554

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, *81*(10), 3088–3092. https://doi.org/10.1073/pnas.81.10.3088

Jaeger, H. & Hass, H. (2004). Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, *304*(5667), 78–80. https://doi.org/10.1126/science.1091277

Jin, L., Nikiforuk, P. & Gupta, M. (1994). Absolute stability conditions for discrete-time recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(6), 954–964. https://doi.org/10.1109/72.329693

Jolliffe, I. T. (2002). *Principal Component Analysis* (2nd ed.). Springer-Verlag. https://doi.org/10.1007/b98835

Jordan, M. I. (1986). *Serial order: A parallel distributed processing approach. Technical Report 8604* (tech. rep.). Institute for Cognitive Science. California University, San Diego (USA). https://doi.org/10.1016/S0166-4115(97)80111-2

Kanuparthi, B., Arpit, D., Kerg, G., Ke, N. R., Mitliagkas, I. & Bengio, Y. (2019). H-detach: Modifying the LSTM Gradient Towards Better Optimization. *International Conference on Learning Representations*. http://arxiv.org/abs/1810.03023

Kloeden, P. E. & Rasmussen, M. (2011). *Nonautonomous dynamical systems*. American Mathematical Society. https://doi.org/10.1090/surv/176

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*(1), 59–69. https://doi.org/10.1007/BF00337288

Maheswaranathan, N., Williams, A., Golub, M. D., Ganguli, S. & Sussillo, D. (2019). Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics. *Advances in Neural Information Processing Systems*, *32*, 15696–15705. http://arxiv.org/abs/1906.10720

Minsky, M. & Papert, S. A. (1972). *Perceptrons: An introduction to computational geometry*. The MIT Press. https://doi.org/10.7551/mitpress/11301.001.0001

Nair, V. & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *ICML'10: Proceedings of the 27th International Conference on International Conference on Machine Learning*, 807–814. https://dl.acm.org/doi/10.5555/3104322.3104425

Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training Recurrent Neural Networks. *ICML'13: Proceedings of the 30th International Conference on Machine Learning*, *28*, 1310–1318. http://arxiv.org/abs/1211.5063

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Roweis, S. T. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, *290*(5500), 2323–2326. https://doi.org/10.1126/science.290.5500.2323

Rumelhart, D. E. [David E.] & McClelland, J. L. [James L.]. (1986). *Parallel Distributed Processing. Volume 1. Foundations. Explorations in the Microstructure of Cognition.* MIT Press. https://doi.org/10.7551/mitpress/5236.001.0001

Rumelhart, D. E. [David E] & McClelland, J. L. [James L]. (1987). Learning Internal Representations by Error Propagation. *Parallel distributed processing. explorations in the microstructure of cognition* (pp. 318–362). https://doi.org/10.1016/B978-1-4832-1446-7.50035-2

Rumelhart, D. E. [David E.] & Zipser, D. (1985). Feature Discovery by Competitive Learning. *Cognitive Science*, *9*(1), 75–112. https://doi.org/10.1207/s15516709cog0901_5

Schuster, M. & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Sell, G. R. & You, Y. (2002). *Dynamics of Evolutionary Equations* (Vol. 143). Springer New York. https://doi.org/10.1007/978-1-4757-5037-9

Sussillo, D. & Barak, O. (2013). Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. *Neural Computation*, *25*(3), 626–649. https://doi.org/10.1162/NECO_a_00409

Sutskever, I. (2013). *Training Recurrent Neural Networks* (Doctoral dissertation) [ISBN: 9780499220660 http://hdl.handle.net/1807/36012]. University of Toronto. Canada. https://dl.acm.org/doi/book/10.5555/2604780

Sutskever, I., Martens, J. & Hinton, G. (2011). Generating Text with Recurrent Neural Networks [https://icml.cc/Conferences/2011/papers/524_icmlpaper.pdf]. *Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11*, 1017–1024. https://dl.acm.org/doi/10.5555/3104482.3104610

Tenenbaum, J. B., Silva, V. d. & Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, *290*(5500), 2319–2323. https://doi.org/10.1126/science.290.5500.2319

Tiňo, P., Horne, B. G. & Giles, C. L. (2001). Attractive Periodic Sets in Discrete-Time Recurrent Networks (with Emphasis on Fixed-Point Stability and Bifurcations in Two-Neuron Networks). *Neural Computation*, *13*(6), 1379–1414. https://doi.org/10.1162/08997660152002898

van der Maaten, L. & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605. https://jmlr.org/papers/v9/vandermaaten08a.html

Vorontsov, E., Trabelsi, C., Kadoury, S. & Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. *ICML'17. Proceedings of the 34th International Conference on Machine Learning*, *70*, 3570–3578. http://arxiv.org/abs/1702.00071

Wang, X. (1991). *Discrete-time neural networks as dynamical systems* (Doctoral dissertation). Mathematics Department. University of Southern California. Los Angeles, CA, United States. https://dl.acm.org/doi/book/10.5555/921548

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

Zou, L., Yu, S., Meng, T., Zhang, Z., Liang, X. & Xie, Y. (2019). A Technical Review of Convolutional Neural Network-Based Mammographic Breast Cancer Diagnosis. *Computational and Mathematical Methods in Medicine*, *2019*, 1–16. https://doi.org/10.1155/2019/6509357