Doctoral Dissertation
Doctoral Program in Electronics Engineering (29th cycle)

Recurrent Neural Networks: Methods and Applications to Non-linear Predictions

By

Alessandro Bay

******

Supervisor:
Prof. Enico Magli

Politecnico di Torino
2017

Declaration

I hereby declare that the contents and organization of this dissertation constitute my own original work and do not compromise in any way the rights of third parties, including those relating to the security of personal data.

Alessandro Bay
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

The research presented in this thesis was carried out at the Joint Open Lab VISIBLE and was supported by a fellowship by TIM, former Telecom Italia. Besides thanking my supervisor Prof. Enrico Magli, I would also like to thank my colleagues, Skjalg Lepsøy, Gianluca Francini and Massimo Balestri for all the help they provided me with and all the good times we have shared.

Finally, this work would not have been possible without the support received from my family and my friends.

Thank you so much.

Abstract

This thesis deals with recurrent neural networks, a particular class of artificial neural networks which can learn a generative model of input sequences. The input is mapped, through a feedback loop and a non-linear activation function, into a *hidden state*, which is then projected into the output space, obtaining either a probability distribution or the new input for the next time-step.

This work consists mainly of two parts: a theoretical study for helping the understanding of recurrent neural networks framework, which is not yet deeply investigated, and their application to non-linear prediction problems, since recurrent neural networks are really powerful models suitable for solving several practical tasks in different fields.

For what concerns the theoretical part, we analyse the weaknesses of state-of-the-art models and tackle them in order to improve the performance of a recurrent neural network. Firstly, we contribute in the understanding of the dynamical properties of a recurrent neural network, highlighting the close relation between the definition of stable limit cycles and the echo state property of an *echo state network*. We provide sufficient conditions for the convergence of the hidden state to a trajectory, which is uniquely determined by the input signal, independently of the initial states. This may help extend the memory of the network and increase the design options for the network.

Moreover, we develop a novel approach to address the main problem in training recurrent neural networks, the so-called *vanishing gradient* problem. Our new method allows us to train a very simple recurrent neural network, making the gradient not to vanish even after many time-steps. Exploiting the singular value decomposition of the vanishing factors in the gradient and random matrices theory, we find that the singular values have to be confined in a narrow interval and derive conditions about their root mean square value.

Then, we also improve the efficiency of the training of a recurrent neural network, defining a new method for speeding up this process. Thanks to a least square regularization, we can initialize the parameters of the network, in order to set them closer to the minimum and running fewer *epochs* of classical training algorithms. Moreover, it is also possible to completely train the network with our initialization method, running more iterations of it without losing in performance with respect to classical training algorithms. Finally, it is also possible to use it as a real-time learning algorithm, adjusting the parameters to the new data through one iteration of our initialization.

In the last part of this thesis, we apply recurrent neural networks to non-linear prediction problems. We consider prediction of numerical sequences, estimating the following input choosing it from a probability distribution. We study an automatic text generation problem, where we need to predict the following character in order to compose words and sentences, and a path prediction of walking mobile users in the central area of a city, as a sequence of crossroads.

Then, we analyse the prediction of video frames, discovering a wide range of applications related to the prediction of movements. We study the collision problem of bouncing balls, taking into account only the sequence of video frames without any knowledge about the physical characteristics of the problem, and the distribution over days of mobile user in a city and in a whole region.

Finally, we address the state-of-the-art problem of missing data imputation, analysing the incomplete spectrogram of audio signals. We restore audio signals with missing time-frequency data, demonstrating via numerical experiments that a performance improvement can be achieved involving recurrent neural networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the latest years, machine learning has seen the rise of artificial neural networks. They are of particular interest since they can deal with a large variety of applications (*e.g.* sentiment classification [1], image retrieval [2], speech recognition [3], ...) and employing the Back-propagation algorithm [4], the weights of the network can be efficiently learned using a gradient-descent scheme.

Recurrent neural networks [5] are a particular class of artificial neural networks which learn a generative model of input sequences. These networks represent an input sequence as a *hidden state* vector, which gathers the information of many temporal steps through a feedback loop and a non-linear activation function. Then, the hidden state is mapped through a linear combination of its elements and sometimes a non-linear function into the output, which can be a probability distribution (in case of classification problems) or the input at the next time step (in case of regression problems).
Therefore, the hidden state is the key-element of a recurrent neural network. In other words, referring to the comparison with the human brain, we can say that the hidden state embodies the memory of the network itself: the more the hidden state can remember from the past input samples, the better the network will be able to predict future temporal steps.
Thus, recurrent neural networks are really powerful models, whose importance became relevant since they are suitable to solve several practical tasks, such as language modelling [6], speech recognition [7], and machine translation [8] for natural

language processing problems, and also video analysis [9] and image captioning [10] for computer vision problems.

Despite this, they have been shown to be difficult to train, especially on problems with complicated long-range temporal structure [11].
The first attempts to train a recurrent neural network were Real-Time Recurrent Learning [12], an elegant forward-pass only algorithm that computes the derivatives with respect to the parameters at each time-step, and Back-propagation Through Time algorithm [13], a modified version of the classical Back-propagation algorithm involving time. Unfortunately, the computational cost is prohibitive and gradient descent and other first-order methods completely fail on large families of problems, respectively.
Another architecture is the Long Short-Term Memory [14] that addresses the vanishing gradient problem making use of special “memory units”, which are able to store information over time. However, even if this solves some pathological problems, it increases too much the number of parameters to be learnt.
Instead, Echo State Networks [15] do not have to deal with the so-called “vanishing/exploding gradients” phenomenon [16, 11] and do not require prohibitive computational cost, but they do not have theoretical guarantees and are based only on a sufficient condition for the non-existence of echo states and a conjecture that remains to be shown [17, 18].

In this thesis, we contribute to the theoretical study of recurrent neural networks and focus our attention on their application to non-linear prediction.
For what concerns the theoretical part, we start analysing the weaknesses of state-of-the-art models and try to improve them. We contribute to the understanding of the dynamics of a recurrent neural network, highlighting the relation between stable limit cycles and the echo state property. We provide sufficient conditions for the convergence of the hidden state to a trajectory, extending the memory of the network and increasing the design options for the network.

Then, we address the vanishing gradient problem, defining a novel approach. This new method allows us to train a simple recurrent neural network, where the gradient does not vanish after many time steps. We tighten the loose power law bounds provided in literature [16], exploiting the singular value decomposition of the vanishing factors in the gradient and random matrices theory. We find that the singular values have to be confined in a narrow interval and derive conditions about their root mean square value.

Then, we also develop a new method for speeding up the training of a recurrent neural network. Thanks to a least square regularization, we can initialize the parameters of the network, in order to be closer to the minimum and running fewer epochs of classical training algorithms. Moreover, it is possible also to completely train the network with our initialization method, running more iterations of it without losing in performance with respect to classical training algorithms, and to use it as a real-time learning algorithm, adjusting the parameters to the new data through one iteration of our initialization.

Finally, we apply recurrent neural networks to a big class of problems, *i.e.* non-linear prediction problems. We consider prediction of numerical sequences, estimating the following output either in a text generation problem or in a path prediction of walking mobile users in the central area of a city. Then, we also take into account the prediction of video frames, discovering a wide range of applications: we study the physical collision problem of bouncing balls, or the distribution over days of mobile user in a city and in a whole region. Moreover, we address the state-of-the-art problem of missing data imputation, analysing the incomplete spectrogram of audio signals.

This thesis is organized as follows: we can identify two main parts, one involving theoretical results and mathematical contributions to understanding recurrent neural networks, and another one dealing with possible applications of this particular kind of network to non-linear prediction.

In Chapter 2, we start introducing the recurrent neural networks framework, providing a definition and explaining how to train them. Here we also introduce the so-called *vanishing gradient problem* and describe the attempts given by literature to address it, modifying the network architecture (long short-term memory - Section 2.3.1 and gated recurrent unit - Section 2.3.2) or resting on a conjecture, which guarantees an echo state property (echo state networks - Section 2.3.3). We also show the power of recurrent neural networks presenting a large number of practical applications they can deal with.

Inspired by the approach based on echo state network to solve the vanishing gradient problem, in Chapter 3, we enhance some mathematical properties of dynamical system theory, which is closely related to the definition of recurrent neural networks.

We establish conditions for the existence of a stable limit cycle, which guarantees the network to satisfy the echo state property.

Then, in Chapter 4, we propose an alternative solution to the vanishing gradient problem, which does not involve the modification of the network architecture nor stable limit cycles (nor the echo state property). We exploit singular values and vectors and random matrices theory, in order to provide bounds to the gradient, making it not to vanish for very deep networks.

As last chapter of the theoretical part (Chapter 5), we present a novel and fast method for training recurrent neural networks based on least squares. This method can be used both as a complete training method or only as an initialization of the parameters, speeding up the training using a classical algorithm, since the initial configuration of the parameters of the network is already close to the one that minimizes the error. Moreover, since this method is really fast, it can be used also in real-time learning, updating the parameters at each step.

Then, we move to apply recurrent neural networks to non-linear prediction problems. In Chapter 6, we perform prediction of numerical sequences, generating texts written by the computer independently, and predicting the path of mobile users, starting from an historical of their movements in a city.

In Chapter 7, we analyse video frames to predict movements. We test our network on the state-of-the-art bouncing balls problem, where the network tries to learn the physical laws of collision theory with balls that bounce off the box walls and among each other (Section 7.1). We also predict the future distribution of mobile users over a region, identifying crowded areas and traffic jams (Section 7.2), using a dataset provided by TIM.

Finally, in Chapter 8, we address the problem of missing data imputation, recovering audio signals from their incomplete spectrograms, and, in Chapter 9, we draw the concluding remarks and introduce the possible future developments of this work.

Chapter 2

Background on Recurrent Neural Networks

Brief - *This chapter is about general principles on recurrent neural networks. They represent a particular class of artificial neural networks, where an input sequence is mapped into a* hidden state *vector, through a feedback reaction and a non-linear function. Then, the hidden state is mapped through a linear combination of its elements and sometimes a non-linear function into the output, which can be a probability distribution (in case of classification problems) or the input at the next time step (in case of regression problems). These networks are really powerful models, whose importance became relevant since they are suitable to solve several practical tasks in different fields, such as natural language processing, computer vision, robotics, and many others. Despite this, recurrent neural networks are effectively hard to be trained.*

This chapter is organized as follows: in the first Section we define the general paradigm for an RNN, then, in Section 2.2 we describe the procedure to follow for training an RNN, comparing two different methods, *i.e.* Real Time Recurrent Learning and Back-Propagation Through Time, and providing details on several optimization algorithms, which can be involved. After that, in Section 2.3, we explain the main problem in training an RNN: the so-called *vanishing gradient problem* and some attempts from literature that try to address this problem. Finally, we conclude with some applications of RNNs in different research fields.

2.1 Definition of a Recurrent Neural Network

Recurrent Neural Networks (RNNs, [5]) are neural networks with feedback loops, through which past information can be stored and exploited. An RNN can be seen as a non-linear dynamical system that maps input time sequences $x(t)$ onto output time sequences $y(t)$. It is parametrized with three weight matrices and three bias vectors $[A,B,C,b,c,h_0]$, which completely describe the network: A is the input-to-hidden weight matrix, B the hidden-to-hidden one, and C the hidden-to-output one, as shown in Figure 2.1. We will generally refer to all these parameters as θ.

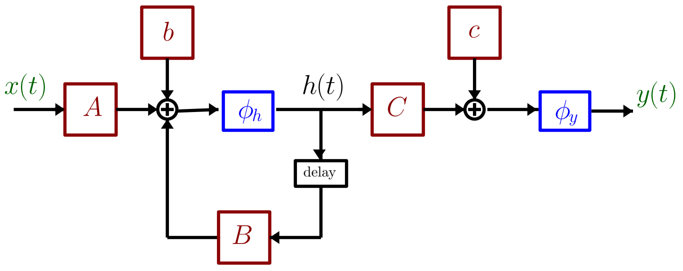


Fig. 2.1 Graphical representation of an RNN. The input/output pair $x(t)/y(t)$ is represented in green, the parameters of the network A,B,b,C,c in red, and the non-linearities ϕ_h,ϕ_y in blue.

Formally, a general RNN is modelled by the following system:

$$\begin{cases} h(t) = \phi_h\Big(Ax(t) + Bh(t-1) + b\Big) \\ y(t) = \phi_y\Big(Ch(t) + c\Big) \end{cases} \tag{2.1}$$

for $t = 1,\dots,T$, where ϕ_h and ϕ_y are non-linear element-wise functions and $h(t)$ is the hidden state, assuming $h(0) = h_0$. In this thesis we will assume h_0 as a column-vector of zeros, so practically it will not be considered as a parameter of the network.

Possible choices of the non-linear functions can be the following:

1. $\phi(z) = \tanh(z)$, the hyperbolic tangent, which shrinks the values of z, between -1 and $+1$;
2. $\phi(z) = \frac{1}{1+\exp(-z)}$, the standard logistic sigmoid function, which bounds the values of z between 0 and 1;

3. $\phi(z) = \mathrm{I}z$, the identity operator, which is often used as output non-linearity when there are no constraints on the output values;

4. $\phi(z) = \mathrm{ReLu}(z)$, the rectified linear unit, which chooses the maximum between z and 0, *i.e.*

$$\mathrm{ReLu}(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases};$$

5. $\phi(z) = \mathrm{SoftMax}(z)$, the softmax function, which computes as output a probability vector, *i.e.* a vector whose components lie in the range $(0,1)$ and sum to 1, and each element i is defined as

$$\mathrm{SoftMax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}; \tag{2.2}$$

6. $\phi(z) = \mathrm{LogSoftMax}(z)$, the log-softmax function, which computes the log of the softmax function (2.2).

Exactly like feedforward networks, for RNNs functions 1-4 are often used for regression problems, while 5-6, which are similar in spirit to probability mass functions, can be employed in classification problems.

For what concerns the input/output pairs (x,y), we should split them into three sets: a *training* set, a *validation* set, and a *testing* set. The training set is the one used to adapt the parameters θ of the network, then the validation set is used simultaneously to the training one, in order to determine if the network has learnt a specific model and if it can reproduce good results on the testing set, which is supposed to be not known to the network. For simplicity, sometimes, it is possible to consider validation and testing sets as the same one, and this is the case for this thesis, where we will generally refer to testing set for both validation and testing sets.

Thus, we can identify two phases: a *training* phase, and a *testing* phase.
During the training phase, we run several *epochs* (or iterations) for all the input/output pairs in the training set, updating the parameters of the network through the gradient

of a *cost function* $\mathscr{C}$, which depends on the *loss function* $\mathscr{L}$:

$$\mathscr{C} = \sum_{(x,y)\in\mathscr{S}_{tr}} \sum_{t=1}^{T} \mathscr{L}(\hat{y}(t), y(t)),$$

where T is the total number of inputs/outputs in the training set $\mathscr{S}_{tr}$, $\hat{y}(t) = \hat{y}(\theta, x(t))$ is the output computed starting from $x(t)$ using the current parameters θ, and $y(t)$ is the desired output.
We have different choices for the loss function $\mathscr{L}$, according to the specific application. The mostly used ones are the following:

(a) the *mean square error* (MSE), which is the most used metric in case of linear regression,

$$\mathscr{L}(\hat{y}(t), y(t)) = \frac{1}{N}\sum_{n=1}^{N} |\hat{y}_n(t) - y_n(t)|^2, \tag{2.3}$$

where N is the dimension of the output vector;

(b) the *mean absolute error* (MAE)

$$\mathscr{L}(\hat{y}(t), y(t)) = \frac{1}{N}\sum_{n=1}^{N} |\hat{y}_n(t) - y_n(t)|; \tag{2.4}$$

(c) the *cross entropy* (CE), which can be applied in case of probability distribution for classification problems, for example when the output non-linearity ϕ_y is the softmax function (2.2),

$$\mathscr{L}(\hat{y}(t), y(t)) = -\sum_{n=1}^{N} \hat{y}_n(t)\log(y_n(t)); \tag{2.5}$$

(d) the *negative log-likelihood* (NLL), which is the extension of CE for the log-softmax function

$$\mathscr{L}(\hat{y}(t), y(t)) = -\sum_{n=1}^{N} \hat{y}_n(t) y_n(t); \tag{2.6}$$

(e) the *binary cross entropy* (BCE), which is simply a two-class version of CE/NLL

$$\mathscr{L}(\hat{y}(t), y(t)) = -\sum_{n=1}^{N} \Big(\hat{y}_n(t)\log(y_n(t)) + (1-\hat{y}_n(t))\log(1-y_n(t))\Big). \tag{2.7}$$

Finally, in the testing phase, the parameters θ of the network cannot be modified, and we can evaluate whether or not the network has learnt a specific structure during the training. Thus, the training of an RNN is the most crucial part of the whole process. In the following Section, we will describe the solution found to properly train an RNN.

2.2 Training Algorithms

A first attempt to train recurrent neural networks was Real-Time Recurrent Learning (RTRL, [12]), which is an elegant forward-pass only algorithm that computes the derivatives with respect to the parameters at each time-step. Unfortunately, the computational cost is prohibitive. Thus, an extension of the classical Back-Propagation algorithm for feedforward neural networks to a modified version involving time, *i.e.* Back-Propagation Through Time algorithm (BPTT, [13]), was proposed.

Both of them, basically, are gradient descent methods that minimize the cost function $\mathscr{C}$, updating the parameters θ of the network in the opposite direction of this gradient with a step size called *learning rate*. Since the cost function is a sum of loss functions, and the derivative is a linear operation, we now focus on the computation of the gradient of the loss function $\mathscr{L}$ for these two methods, giving details in Subsections 2.2.1 and 2.2.2 respectively.

2.2.1 Real-Time Recurrent Learning

The training algorithm RTRL [12] allows to update the parameters of the network continuously. It has the advantage of not requiring a precisely defined training interval, since it operates while the network runs, but also has the disadvantage that it is computationally expensive.

The algorithm starts at $t = 1$, and computes firstly the output $\hat{y}(1)$, using an initial random configuration of the parameters θ_0, and then the gradient of the loss function

$\mathscr{L}$ with respect to each parameter A, B, b, C, c:

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial c} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(1)}\frac{\partial \hat{y}(1)}{\partial z_y(1)}\frac{\partial z_y(1)}{\partial c}\\ &= J_{\mathscr{L}}(\hat{y}(1), y(1))^\top \phi_y'\Big(Ch(1)+c\Big), \end{aligned} \tag{2.8}$$

where $J_{\mathscr{L}}(\hat{y}(1), y(1))^\top$ is the row-vector of the Jacobian of the loss function (see Appendix A for details) and $z_y(1) = Ch(1)+c$,

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial C} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(1)}\frac{\partial \hat{y}(1)}{\partial z_y(1)}\frac{\partial z_y(1)}{\partial C}\\ &= J_{\mathscr{L}}(\hat{y}(1), y(1))^\top \phi_y'\Big(Ch(1)+c\Big)\otimes h(1) \\ &= \frac{\partial \mathscr{L}}{\partial c}\otimes h(1), \end{aligned} \tag{2.9}$$

where the external product $\otimes$ between two column vector u, v is defined as $u \otimes v = uv^\top$,

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial b} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(1)}\frac{\partial \hat{y}(1)}{\partial z_y(1)}\frac{\partial z_y(1)}{\partial h(1)}\frac{\partial h(1)}{\partial z_h(1)}\frac{\partial z_h(1)}{\partial b}\\ &= J_{\mathscr{L}}(\hat{y}(1), y(1))^\top \phi_y'\Big(Ch(1)+c\Big)C\phi_h'\Big(Ax(1)+Bh_0+b\Big) \\ &= \frac{\partial \mathscr{L}}{\partial c}C\phi_h'\Big(Ax(1)+Bh_0+b\Big), \end{aligned} \tag{2.10}$$

where $z_h(1) = Ax(1)+Bh_0+b$,

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial B} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(1)}\frac{\partial \hat{y}(1)}{\partial z_y(1)}\frac{\partial z_y(1)}{\partial h(1)}\frac{\partial h(1)}{\partial z_h(1)}\frac{\partial z_h(1)}{\partial B}\\ &= J_{\mathscr{L}}(\hat{y}(1), y(1))^\top \phi_y'\Big(Ch(1)+c\Big)C\phi_h'\Big(Ax(1)+Bh_0+b\Big)\otimes h_0 \\ &= \frac{\partial \mathscr{L}}{\partial b}\otimes h(t-1), \end{aligned} \tag{2.11}$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial A} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(1)}\frac{\partial \hat{y}(1)}{\partial z_y(1)}\frac{\partial z_y(1)}{\partial h(1)}\frac{\partial h(1)}{\partial z_h(1)}\frac{\partial z_h(1)}{\partial A} \\ &= J_{\mathscr{L}}(\hat{y}(1),y(1))^\top \phi_y'\Big(Ch(1)+c\Big)C\phi_h'\Big(Ax(1)+Bh_0+b\Big)\otimes x(1) \\ &= \frac{\partial \mathscr{L}}{\partial b}\otimes x(t).\end{aligned} \tag{2.12}$$

Then, for a generic $t > 1$, the computation of the gradient becomes more complicated, since we need to consider the dependence from all previous time steps due to $h(t-1)$. As a matter of fact, equations (2.8),(2.9) have the same form, replacing 1 with t:

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial c} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(t)}\frac{\partial \hat{y}(t)}{\partial z_y(t)}\frac{\partial z_y(t)}{\partial c} \\ &= J_{\mathscr{L}}(\hat{y}(t),y(t))^\top \phi_y'\Big(Ch(t)+c\Big),\end{aligned} \tag{2.13}$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial C} &= \frac{\partial \mathscr{L}}{\partial \hat{y}(t)}\frac{\partial \hat{y}(t)}{\partial z_y(t)}\frac{\partial z_y(t)}{\partial C} \\ &= J_{\mathscr{L}}(\hat{y}(t),y(t))^\top \phi_y'\Big(Ch(t)+c\Big)\otimes h(t), \\ &= \frac{\partial \mathscr{L}}{\partial c}\otimes h(t),\end{aligned} \tag{2.14}$$

while (2.10),(2.11),(2.12) exploit the computation of the derivative of the hidden state $h(t)$ with respect to the parameters of the recurrent layer b,B,A:

$$\begin{aligned}\frac{\partial h(t)}{\partial b} &= \frac{\partial h(t)}{\partial z_h(t)}\frac{\partial z_h(t)}{\partial b} \\ &= \phi_h'(Ax(t)+Bh(t-1)+b)\left(\mathbb{I}+B\frac{\partial h(t-1)}{\partial b}\right),\end{aligned} \tag{2.15}$$

where $\mathbb{I}$ is the identity operator and $\frac{\partial h(t-1)}{\partial b}$ has been computed in the previous time step,

$$\begin{aligned}\frac{\partial h(t)}{\partial B} &= \frac{\partial h(t)}{\partial z_h(t)}\frac{\partial z_h(t)}{\partial B} \\ &= \phi_h'(Ax(t)+Bh(t-1)+b)\otimes\left(h(t-1)+B\frac{\partial h(t-1)}{\partial B}\right),\end{aligned} \tag{2.16}$$

$$\begin{aligned}\frac{\partial h(t)}{\partial A} &= \frac{\partial h(t)}{\partial z_h(t)}\frac{\partial z_h(t)}{\partial B} \\ &= \phi_h'(Ax(t)+Bh(t-1)+b)\otimes\left(x(t)+B\frac{\partial h(t-1)}{\partial A}\right).\end{aligned} \tag{2.17}$$

Thus, (2.10),(2.11),(2.12) become respectively:

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial b} =& \frac{\partial \mathscr{L}}{\partial \hat{y}(t)}\frac{\partial \hat{y}(t)}{\partial z_y(t)}\frac{\partial z_y(t)}{\partial h(t)}\frac{\partial h(t)}{\partial b} \\ &= \frac{\partial \mathscr{L}}{\partial c}C\frac{\partial h(t)}{\partial b},\end{aligned} \tag{2.18}$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial B} =& \frac{\partial \mathscr{L}}{\partial \hat{y}(t)}\frac{\partial \hat{y}(t)}{\partial z_y(t)}\frac{\partial z_y(t)}{\partial h(t)}\frac{\partial h(t)}{\partial b} \\ &= \frac{\partial \mathscr{L}}{\partial c}C\frac{\partial h(t)}{\partial B},\end{aligned} \tag{2.19}$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial A} =& \frac{\partial \mathscr{L}}{\partial \hat{y}(t)}\frac{\partial \hat{y}(t)}{\partial z_y(t)}\frac{\partial z_y(t)}{\partial h(t)}\frac{\partial h(t)}{\partial b} \\ &= \frac{\partial \mathscr{L}}{\partial c}C\frac{\partial h(t)}{\partial A}.\end{aligned} \tag{2.20}$$

Notice that starting from (2.18),(2.19),(2.20) is it possible to find (2.10),(2.11),(2.12) replacing $t = 1$ and considering $\frac{\partial h(0)}{\partial \theta} = 0$ in (2.15),(2.16),(2.17), since $h(0) = H_0$ does not depend on any of the parameters of the network.

Finally, it is now possible to update all the parameters, summing the gradients of the loss function over time t and over the input/output pairs in the training set $\mathscr{S}_{tr}$.

2.2.2 Back-Propagation Through Time

Back-Propagation [5] is an efficient and exact method for calculating all the derivatives of a target quantity with respect to a large set of parameters. BPTT [13] simply extends this method in order to apply to dynamical system such as RNNs.
As a matter of fact, RNNs can be seen as very deep feedforward neural network, sharing the weights through all the layers, which, in the case of RNNs, correspond to the time steps. Thus, we can *unfold* an RNN, in order to obtain a deep neural network with T layers, as shown in Figure 2.2, compute the gradient for each time

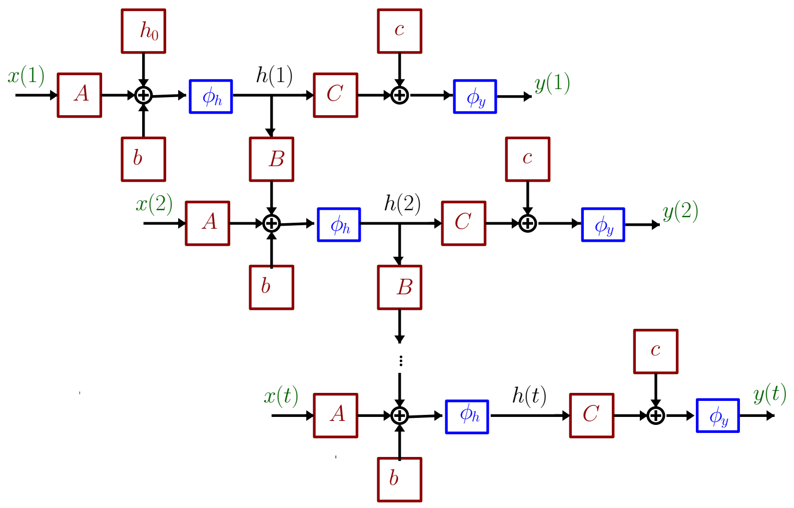


Fig. 2.2 Graphical representation of an unfolded RNN.

step, and finally sum up all the gradients

$$\frac{\partial \mathscr{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathscr{L}_t}{\partial \theta}.$$

While RTRL computes the new gradient going on in time (starting from $t = 1$), conversely, for BPTT we start from computing the error at the last step t, and then we back-propagate the error through the network for a number of steps k, for which we decide to unfold the networks. For what concerns the output layer, it is simply a feedforward network, thus, it is the same as (2.8),(2.9) (or equivalently (2.13),(2.14)) $\forall t$:

$$\frac{\partial \mathscr{L}_t}{\partial c^{(t)}} = J_{\mathscr{L}_t}(\hat{y}(t), y(t))^\top \phi_y'\left(C^{(t)}h(t) + c^{(t)}\right) \tag{2.21}$$

$$\frac{\partial \mathscr{L}_t}{\partial C^{(t)}} = \frac{\partial \mathscr{L}_t}{\partial c^{(t)}} \otimes h(t). \tag{2.22}$$

For the recurrent layer, we consider the unfolded network as if it had different parameters in each layer. Thus, the gradients of the loss function with respect to the parameters of the recurrent layer T are:

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial b^{(T)}} &= \frac{\partial \mathscr{L}_t}{\partial b^{(T)}} \\
&= J_{\mathscr{L}_t}(\hat{y}(t), y(t))^\top \frac{\partial \hat{y}(t)}{\partial z_y(t)} \frac{\partial z_y(t)}{\partial h(t)} \frac{\partial h(t)}{\partial b^{(T)}} \\
&= \frac{\partial \mathscr{L}_T}{\partial c^{(T)}} C^{(T)} \phi_h'(A^{(T)} x(T) + B^{(T)} h(T-1) + b^{(T)}),
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial B^{(T)}} &= J_{\mathscr{L}_t}(\hat{y}(t), y(t))^\top \frac{\partial \hat{y}(t)}{\partial z_y(t)} \frac{\partial z_y(t)}{\partial h(t)} \frac{\partial h(t)}{\partial B^{(T)}} \\
&= \frac{\partial \mathscr{L}_T}{\partial b^{(T)}} \otimes h(T-1),
\end{aligned} \tag{2.23}$$

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial A^{(T)}} &= J_{\mathscr{L}_t}(\hat{y}(t), y(t))^\top \frac{\partial \hat{y}(t)}{\partial z_y(t)} \frac{\partial z_y(t)}{\partial h(t)} \frac{\partial h(t)}{\partial B^{(T)}} \\
&= \frac{\partial \mathscr{L}_T}{\partial b^{(T)}} \otimes x(T).
\end{aligned} \tag{2.24}$$

Proceeding backwards, we have to compute the gradients of the loss function with respect to the bias vector of a generic layer k:

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial b^{(k)}} &= \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial b^{(k)}} \\
&= \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)} \frac{\partial h(k)}{\partial b^{(k)}} \\
&= \frac{\partial \mathscr{L}}{\partial h(k)} \frac{\partial h(k)}{\partial b^{(k)}} \\
&= \frac{\partial \mathscr{L}}{\partial h(k)} \phi_h'(A^{(k)} x(k) + B^{(k)} h(k-1) + b^{(k)}),
\end{aligned} \tag{2.25}$$

where

$$
\begin{aligned}
\frac{\partial \mathscr{L}}{\partial h(k)} &= \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)} \\
&= \sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)} + \frac{\partial \mathscr{L}_k}{\partial h(k)} \\
&= \sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k+1)} \frac{\partial h(k+1)}{\partial h(k)} + \frac{\partial \mathscr{L}_k}{\partial h(k)} \\
&= \frac{\partial \mathscr{L}}{\partial h(k+1)} \frac{\partial h(k+1)}{\partial h(k)} + \frac{\partial \mathscr{L}_k}{\partial h(k)} \\
&= \frac{\partial \mathscr{L}}{\partial h(k+1)} \frac{\partial h(k+1)}{\partial z_h(k+1)} \frac{\partial z_h(k+1)}{\partial h(k)} + \frac{\partial \mathscr{L}_k}{\partial h(k)} \\
&= \frac{\partial \mathscr{L}}{\partial h(k+1)} \phi_h'(A^{(k+1)}x(k+1) + B^{(k+1)}h(k) + b^{(k+1)})B^{(k+1)} + \frac{\partial \mathscr{L}_k}{\partial c^{(k)}} C^{(k)}.
\end{aligned}
$$

Notice that this can be efficiently computed, since $\frac{\partial \mathscr{L}}{\partial h(k+1)}$ is already available from the computation in the previous layer.
Thus, thanks to the definition of (2.25) and following the scheme used in (2.23) and (2.24), we can immediately compute the gradient with respect to the hidden-to-hidden matrix B and the input-to-hidden matrix A for every layer k, *i.e.*

$$
\begin{aligned}
\frac{\partial \mathscr{L}}{\partial B^{(k)}} &= \frac{\partial \mathscr{L}}{\partial b^{(k)}} \otimes h(k-1) \\
\frac{\partial \mathscr{L}}{\partial A^{(k)}} &= \frac{\partial \mathscr{L}}{\partial b^{(k)}} \otimes x(k).
\end{aligned}
$$

Finally, it is now possible to sum the gradients of the loss function over time t, since the parameters are the same for each layer (*i.e.* for each time step), and over the input/output pairs in the training set $\mathscr{S}_{tr}$, in order to obtain the update of the parameters and to run the next epoch with the updated parameters.

2.2.3 Optimization Algorithms

After computing the gradient with either RTRL or BPTT, then the parameters of the network can be updated employing an optimization algorithm in order to minimize the cost function.

Stochastic Gradient Descent

The most intuitive one is the Stochastic Gradient Descent (SGD) method, which performs the update of the parameters, going in the opposite direction of the gradient $\nabla_\theta \mathscr{L}$ by a step-size μ (which is the learning rate), for each input/output training pair $(x(t), y(t))$:

$$\theta = \theta - \mu \nabla_\theta \mathscr{L}(\theta; x(t), y(t)). \tag{2.26}$$

One of the problem of SGD is that the objective function heavily fluctuates, due to the frequent updates SGD performs for every time step t. However, this fluctuation can be useful to move from one local minimum to a potentially better one. As a matter of fact, a possible variant of SGD could update the parameters for the whole training dataset, but in this way the computation can be infeasible in terms of computational time and memory used and, moreover, it can converge to a local minimum, which is far from the global one.
Thus, for training a neural network, it is usually better to perform the update for *mini-batches* of n training samples. In this way, one can reach a more stable convergence, avoiding the high fluctuations and performing efficiently the computations due to optimized matrix operations for each mini-batch.

Nevertheless, SGD does not guarantee the convergence to a minimum close to global one, so it offers some challenges.
First of all, it is difficult to choose the proper learning rate μ. If too small, the convergence will be very slow. On the contrary, too large values of μ may cause fluctuations or even divergence.
Then, it is possible to adjust the learning rate during the training according to a specific schedule [19], but this schedule has to be pre-defined in advance and it should be selected according to the specific dataset considered.
Finally, as already mentioned, it is important to avoid falling in suboptimal local minima and also in saddle points, which make the gradient close to zero, making it impossible to escape [20].
In the following, we describe methods that try to address these challenges.

Momentum

Momentum [21] is an algorithm that accelerates SGD in the relevant direction, avoiding oscillations around local minima due to a steep slope. It simply adds to the current update vector the past one multiplied by a factor $\gamma < 1$:

$$\begin{cases} v_i = \gamma v_{i-1} + \mu \nabla_\theta \mathscr{L}(\theta) \\ \theta = \theta - v_i \end{cases} . \tag{2.27}$$

Thus, it increases the update term if the gradients points to the same direction, obtaining a fast convergence and reducing oscillations.

Nesterov Accelerated Gradient

Nesterov Accelerated Gradient (NAG, [22]), instead, uses the momentum term γv_{i-1} to compute the gradient according to an approximation of the next position given by $\theta - \gamma v_{i-1}$:

$$\begin{cases} v_i = \gamma v_{i-1} + \mu \nabla_\theta \mathscr{L}(\theta - \gamma v_{i-1}) \\ \theta = \theta - v_i \end{cases} . \tag{2.28}$$

Thus, while Momentum (2.27) computes the current gradient and then updates the parameters in the direction of the updated accumulated gradient, NAG updates the parameters in the direction of the previous accumulated gradient and then corrects them with the current accumulated gradient. Figure 2.3 [23] shows graphically this difference between Momentum (2.27) and NAG (2.28).

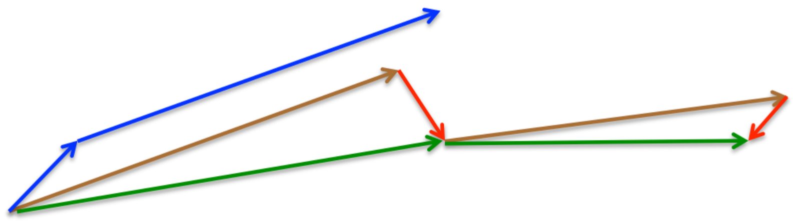


Fig. 2.3 The blue vectors represent the Momentum update, *i.e.* small blue vector is the current gradient, and big blue one is the updated accumulated gradient; the brown vectors are the previous accumulated gradient, the red ones represent the corrections, and green ones give the NAG.

Adagrad

Now, we would like to adapt our updates, without manually tune the learning rate, but performing a smaller or larger update according to each individual parameter. Adagrad [24] aims to solve this problem. It is an algorithm for gradient-based optimization that adapts the learning rate μ to the parameters θ, performing larger updates for infrequent parameters and smaller updates for frequent parameters:

$$\theta = \theta - \frac{\mu}{\sqrt{G+\varepsilon}} \nabla_\theta \mathscr{L}(\theta), \tag{2.29}$$

where G is the sum of the squares of the gradients and ε is a smoothing term that avoids division by zero.
On the one hand, Adagrad avoids the scheduled tuning of the learning rate, but, on the other hand, the accumulation of the squared gradients in G keeps growing, and the algorithm may become unable to acquire further information during the training.

Adadelta

Adadelta [25] is an extension of Adagrad (2.29) that seeks to reduce its aggressive, monotonically decreasing learning rate. It follows a new updating rule, replacing $\sqrt{G+\varepsilon}$ with the root mean squared (RMS) of the gradients, and the learning rate μ with the RMS of the previous update rule:

$$\begin{cases} \Delta\theta_i = -\frac{\mathrm{RMS}[\Delta\theta_{i-1}]}{\mathrm{RMS}[\nabla_\theta \mathscr{L}(\theta_i)]} \nabla_\theta \mathscr{L}(\theta_i) \\ \theta_{i+1} = \theta_i + \Delta\theta_i \end{cases}, \tag{2.30}$$

where

$$\mathrm{RMS}[z] = \sqrt{\frac{1}{N}\sum_{i=1}^{N} z_i^2}.$$

Notice that now there is no need to set a learning rate μ, since it has been replaced with the previous update rule.

RMSprop

RMSprop [26] is an adaptive learning rate method developed in order to solve Adagrad's diminishing learning rate problem. It exploits an exponentially decaying average of squared gradients:

$$\begin{cases} E[(\nabla_\theta \mathscr{L}(\theta_i))^2] = \gamma E[(\nabla_\theta \mathscr{L}(\theta_{i-1}))^2] + (1-\gamma)(\nabla_\theta \mathscr{L}(\theta_i))^2 \\ \theta_{i+1} = \theta_i - \frac{\mu}{\sqrt{E[(\nabla_\theta \mathscr{L}(\theta_i))^2]+\varepsilon}} \nabla_\theta \mathscr{L}(\theta_i). \end{cases} \tag{2.31}$$

Adam

Adaptive Moment Estimation (Adam, [27]) is another method that computes adaptive learning rates for each parameter. It stores an exponentially decaying average of past squared gradients v_i, in the same way as Adadelta (2.30) and RMSprop (2.31), and, moreover, it keeps an exponentially decaying average of past gradients m_i, like Momentum (2.27):

$$\begin{cases} m_i = \beta_1 m_{i-1} + (1-\beta_1)\nabla_\theta \mathscr{L}(\theta_i) \\ v_i = \beta_2 v_{i-1} + (1-\beta_2)\left(\nabla_\theta \mathscr{L}(\theta_i)\right)^2 . \\ \theta_{i+1} = \theta_i - \frac{\mu}{\sqrt{v_i}+\varepsilon} m_i \end{cases}$$

It has been shown empirically that Adam works well in practice and compares favourably to other adaptive algorithms.

2.3 Vanishing Gradient Problem

Unfortunately, gradient descent and other first-order methods completely fail on large-scale problems. For example, if the estimated gradient with respect to the coefficients in early layers approaches zero before a minimum is reached, then learning for these layers actually stops. On the other hand, if the gradient becomes enormous, then the parameter is likely to be moved to a value that is very far from the optimum. These are known as the *vanishing or exploding gradient problems*, respectively, which are considered to be the fundamental problems in training RNNs [16, 11].

In order to study these problems, it is possible to bound the norm of the gradient of the loss function with respect to the parameters of the layer k, $\theta^{(k)}$, by using the triangle inequality

$$\left\| \frac{\partial \mathscr{L}}{\partial \theta^{(k)}} \right\| = \left\| \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial \theta^{(k)}} \right\| \leq \sum_{t=k}^{T} \left\| \frac{\partial \mathscr{L}_t}{\partial \theta^{(k)}} \right\|.$$

Thus, we can analyse each single term

$$\frac{\partial \mathscr{L}_t}{\partial \theta^{(k)}} = \frac{\partial \mathscr{L}_t}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \left(\frac{\partial h(t)}{\partial h(t-1)} \cdots \frac{\partial h(t-k+1)}{\partial h(t-k)} \right) \frac{\partial h(t-k)}{\partial \theta^{(k)}}. \tag{2.32}$$

Moreover, we can define each layer-to-layer Jacobian as

$$J(i) := \frac{\partial h(t-i)}{\partial h(t-i-1)},$$

in order to simplify the notation, and rewrite equation (2.32) as

$$\frac{\partial \mathscr{L}_t}{\partial \theta^{(k)}} = \frac{\partial \mathscr{L}_t}{\partial \hat{y}(t)} \frac{\partial \hat{y}(t)}{\partial h(t)} \Big(J(0) \ldots J(k-1) \Big) \frac{\partial h(t-k)}{\partial \theta^{(k)}}. \tag{2.33}$$

Therefore, we can bound the norm of (2.33) by using the product inequality

$$\begin{aligned} \left\| \frac{\partial \mathscr{L}_t}{\partial \theta^{(k)}} \right\| &\leq \left\| \frac{\partial \mathscr{L}_t}{\partial y(t)} \frac{\partial y(t)}{\partial h(t)} \right\| \|J(0) \cdots J(k-1)\| \left\| \frac{\partial h(t-k)}{\partial \theta^{(k)}} \right\| \\ &\leq \left\| \frac{\partial \mathscr{L}_t}{\partial y(t)} \frac{\partial y(t)}{\partial h(t)} \right\| \|J(0)\| \cdots \|J(k-1)\| \left\| \frac{\partial h(t-k)}{\partial \theta^{(k)}} \right\| \\ &\leq \Big(\max_m \|J(m)\| \Big)^k \left\| \frac{\partial \mathscr{L}_t}{\partial y(t)} \frac{\partial y(t)}{\partial h(t)} \right\| \left\| \frac{\partial h(t-k)}{\partial \theta^{(k)}} \right\|, \end{aligned} \tag{2.34}$$

and by choosing the maximum among the norms of the layer-to-layer Jacobians.

Thus, we obtain a power law, which depends on the maximum of the norms of the layer-to-layer Jacobians. This means that if $\max_m \|J(m)\| < 1$, then the gradient diminishes exponentially with depth k, but, on the other hand if $\max_m \|J(m)\| > 1$ the gradient may grow exponentially. Moreover, following the same procedure used for obtaining (2.34), we can notice that $P(k) = \|J(0) \ldots J(k-1)\|$ is also a bound in the derivative of the loss with respect to the input x, which means that if $P(k)$ goes to zero, with increasing k, then the output (and the loss) is insensitive to the input, obtaining a network that does not work at all.

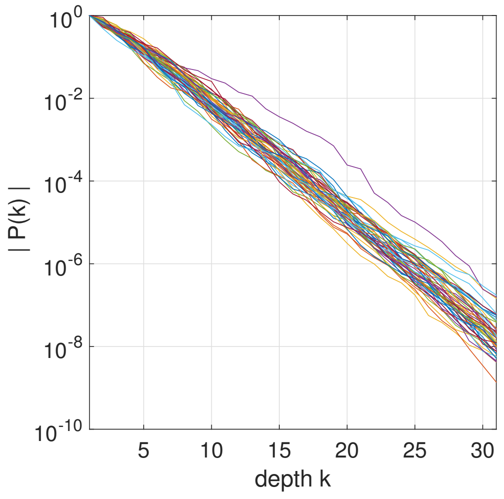


Fig. 2.4 Test showing the trend of $P(k)$ if the norm of each layer-to-layer Jacobian is exactly equal to 1.

Finally, we also analysed the particular case when $\|J(m)\| = 1, \forall m$. We found that if $J(0),\dots,J(k-1)$ are random and scaled such that $\|J(m)\| = 1$, then $P(k)$ is still extremely likely to approach zero quickly. As shown in Figure 2.4, we run several experiments (one for each line in the graphic), obtaining the same trend in every test we performed. Thus, we can say that, even if the norm of the Jacobians is exactly equal to one, the gradient is likely to vanish with depth, or time passing in our case with RNN.

Many attempts were made to face this problem. Long Short Term Memory networks (LSTMs, [14]) were a first approach and were designed through a gating mechanism. On the other hand, this problem was sidestepped also by the Echo-State Networks (ESNs, [15]), which learn neither the input-to-hidden nor the hidden-to-hidden connections, but set them randomly and only use the training data to learn the hidden-to-output connections.

In the following subsections we explain the details of these two kinds of RNNs, while we will develop our condition for overcoming this obstacle in Chapter 4.

2.3.1 Long Short Term Memory Networks

LSTMs [14] were designed to face this problem through a gating mechanism. This is composed by an internal memory cell unit $\mathfrak{c}(t)$ and four gates, an input gate $i(t)$, a "candidate" gate $j(t)$, a forget gate $f(t)$ and an output gate $o(t)$, which are involved in the computation of the hidden state $h(t)$. These gates have the same structures as the hidden state of a basic RNN (2.1), while the memory cell is a sort of convex combination of the gates. Formally, an LSTM can be modelled by the following system:

$$\begin{cases} i(t) = \phi_i\Big(A_i x(t) + B_i h(t-1) + b_i\Big) \\ j(t) = \phi_j\Big(A_j x(t) + B_j h(t-1) + b_j\Big) \\ f(t) = \phi_f\Big(A_f x(t) + B_f h(t-1) + b_f\Big) \\ o(t) = \phi_o\Big(A_o x(t) + B_o h(t-1) + b_o\Big) \\ \mathfrak{c}(t) = \mathfrak{c}(t-1) \odot f(t) + i(t) \odot j(t) \\ h(t) = \tanh\Big(\mathfrak{c}(t)\Big) \odot o(t) \\ y(t) = \phi_y\Big(C h(t) + c\Big) \end{cases}$$

for $t = 1, \dots, T$, where $\phi_\bullet$ are non-linear element-wise functions and $\odot$ represents the element-wise product.

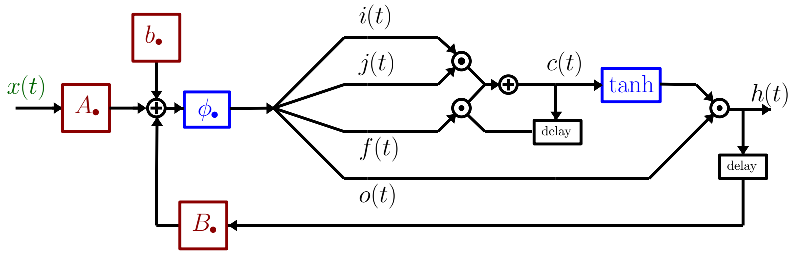


Fig. 2.5 Graphical representation of an LSTM. $A_\bullet, B_\bullet, b_\bullet, \phi_\bullet$ indicate all the input-to-hidden matrices, all the hidden-to-hidden ones, all the bias vectors, and all the non-linear element-wise functions, respectively, for the input $i(t)$, candidate $j(t)$, forget $f(t)$, and output $o(t)$ gates.

In Figure 2.5, we show how the recurrent layer of a basic RNN changes in a LSTM (the output layer that maps the hidden state $h(t)$ to the output $y(t)$ is not depicted since it is the same of an RNN, see Figure 2.1).

Thus, LSTMs address the vanishing gradient problem by reparameterizing the RNN with this gating mechanism, which makes its gradient not to vanish. However, this is not clearly evident in literature. To highlight more deeply this aspect, we can compute the gradients for BPTT in the case of an LSTM network.

Gradients for LSTM architecture

Following the principle adopted in Section 2.2.2, *i.e.* unfolding the recurrent network and considering each time-layer as if it had different parameters from other layers, we compute the derivatives of the loss function $\mathscr{L}$ with respect to all the parameters of an LSTM.

Starting from the output layer, the gradients with respect to c and C are the same as (2.21) and (2.22), respectively, $\forall t$:

$$\begin{aligned}
\frac{\partial \mathscr{L}_t}{\partial c^{(t)}} &= J_{\mathscr{L}_t}(\hat{y}(t), y(t))^\top \phi_y'\Big(C^{(t)} h(t) + c^{(t)}\Big)\\
\frac{\partial \mathscr{L}_t}{\partial C^{(t)}} &= \frac{\partial \mathscr{L}_t}{\partial c^{(t)}} \otimes h(t).
\end{aligned}$$

Back-propagating the error, we obtain, for the last layer of the unfolded network T,

$$\begin{aligned}
\frac{\partial \mathscr{L}_T}{\partial h(T)} &= \frac{\partial \mathscr{L}_T}{\partial c^{(T)}} C\\
\frac{\partial \mathscr{L}_T}{\partial o(T)} &= \frac{\partial \mathscr{L}_T}{\partial c^{(T)}} C \tanh\Big(\mathfrak{c}(T)\Big)\\
\frac{\partial \mathscr{L}_T}{\partial \mathfrak{c}(T)} &= \frac{\partial \mathscr{L}_T}{\partial c^{(T)}} C\Big(1 - \tanh^2(\mathfrak{c}(T))\Big) \odot o(T)\\
\frac{\partial \mathscr{L}_T}{\partial f(T)} &= \frac{\partial \mathscr{L}_T}{\partial \mathfrak{c}(T)} \odot \mathfrak{c}(T-1)\\
\frac{\partial \mathscr{L}_T}{\partial j(T)} &= \frac{\partial \mathscr{L}_T}{\partial \mathfrak{c}(T)} \odot i(T)\\
\frac{\partial \mathscr{L}_T}{\partial i(T)} &= \frac{\partial \mathscr{L}_T}{\partial \mathfrak{c}(T)} \odot j(T).
\end{aligned}$$

From these equations, it is easy to find the gradients with respect to the parameters of each gate $\bullet = o, f, j, i$:

$$\begin{aligned}
\frac{\partial \mathscr{L}_T}{\partial b_\bullet^{(T)}} &= \frac{\partial \mathscr{L}_T}{\partial \bullet(T)} \phi'_\bullet \left(A_\bullet^{(T)} x(T) + B_\bullet^{(T)} h(T-1) + b_\bullet^{(T)} \right) \\
\frac{\partial \mathscr{L}_T}{\partial B_\bullet^{(T)}} &= \frac{\partial \mathscr{L}_T}{\partial b_\bullet^{(T)}} \otimes h(T-1) \\
\frac{\partial \mathscr{L}_T}{\partial A_\bullet^{(T)}} &= \frac{\partial \mathscr{L}_T}{\partial b_\bullet^{(T)}} \otimes x(T).
\end{aligned}$$

Then, proceeding backwards, it becomes more complicated to compute the gradients of the loss function with respect to the bias vector of a generic layer k. We can exploit the derivative of the loss with respect to the hidden state and memory cell at previous time-step, respectively, as:

$$\frac{\partial \mathscr{L}}{\partial h(k-1)} = \frac{\partial \mathscr{L}}{\partial b_o^{(k)}} B_o^{(k)} + \frac{\partial \mathscr{L}}{\partial b_f^{(k)}} B_f^{(k)} + \frac{\partial \mathscr{L}}{\partial b_j^{(k)}} B_j^{(k)} + \frac{\partial \mathscr{L}}{\partial b_i^{(k)}} B_i^{(k)} \tag{2.35}$$

$$\frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k-1)} = \frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k)} \odot f(k). \tag{2.36}$$

Thus, for a generic layer k, we obtain

$$\frac{\partial \mathscr{L}}{\partial h(k)} = \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)} = \sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)} + \frac{\partial \mathscr{L}_k}{\partial h(k)} \tag{2.37}$$

$$\frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k)} = \sum_{t=k}^{T} \frac{\partial \mathscr{L}_t}{\partial \mathfrak{c}(k)} = \sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial \mathfrak{c}(k)} + \frac{\partial \mathscr{L}_k}{\partial \mathfrak{c}(k)}, \tag{2.38}$$

where $\sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial h(k)}$ and $\sum_{t=k+1}^{T} \frac{\partial \mathscr{L}_t}{\partial \mathfrak{c}(k)}$ have been computed in the previous layer as (2.35) and (2.36), respectively.
Now, using (2.37) and (2.38), we find the gradients

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial b_\bullet^{(k)}} &= \frac{\partial \mathscr{L}}{\partial \bullet(k)} \phi'_\bullet \left(A_\bullet^{(k)} x(k) + B_\bullet^{(k)} h(k-1) + b_\bullet^{(k)} \right) \\
\frac{\partial \mathscr{L}}{\partial B_\bullet^{(k)}} &= \frac{\partial \mathscr{L}}{\partial b_\bullet^{(k)}} \otimes h(k-1) \\
\frac{\partial \mathscr{L}}{\partial A_\bullet^{(k)}} &= \frac{\partial \mathscr{L}}{\partial b_\bullet^{(k)}} \otimes x(k),
\end{aligned}$$

where $\bullet = o, f, j, i$ and, respectively, we have:

$$\frac{\partial \mathscr{L}}{\partial o(k)} = \frac{\partial \mathscr{L}}{\partial h(k)} \tanh\Big(\mathfrak{c}(k)\Big)$$
$$\frac{\partial \mathscr{L}}{\partial f(k)} = \frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k)} \odot \mathfrak{c}(k-1)$$
$$\frac{\partial \mathscr{L}}{\partial j(k)} = \frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k)} \odot i(k)$$
$$\frac{\partial \mathscr{L}}{\partial i(k)} = \frac{\partial \mathscr{L}}{\partial \mathfrak{c}(k)} \odot j(k).$$

As shown from the computation of the gradients, in the case of an LSTM we do not have a simple multiplication of many terms such as

$$\frac{\partial h(t)}{\partial h(t-k)} = \frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial h(t-2)} \cdots \frac{\partial h(t-k+1)}{\partial h(t-k)},$$

like in an RNN, linking different layers, which makes the gradient vanishing.

Finally, a criticism of the LSTM architecture is its increasing number of components whose purpose is not evident. Moreover it is also not clear whether the LSTM is an optimal architecture [28].

2.3.2 Gated Recurrent Unit

One variant of LSTM is the Gated Recurrent Unit (GRU, [8]). It combines the forget and input gates into a single update gate z, uses a reset gate r and removes the memory cell, merging it with the hidden state. Formally, a GRU can be modelled as follows

$$\begin{cases} r(t) = \phi_r\Big(A_r x(t) + B_r h(t-1) + b_r\Big) \\ z(t) = \phi_z\Big(A_z x(t) + B_z h(t-1) + b_z\Big) \\ \tilde{h}(t) = \phi_h\Big(A_h x(t) + B_h(r(t) \odot h(t-1)) + b_h\Big) \\ h(t) = z(t) \odot h(t-1) + (1 - z(t)) \odot \tilde{h}(t) \\ y(t) = \phi_y\Big(Ch(t) + c\Big) \end{cases},$$

where $\tilde{h}$ is a candidate hidden state vector.

According to empirical evaluations [28, 29], both LSTM and GRU architectures yield comparable results in many tasks. Even if GRU may have a faster training, since it has fewer parameters than an LSTM, the greater expressive power of LSTMs may lead to better performances.

A completely different solution to training RNNs was proposed by Jaeger [15], who introduced the Echo State Networks.

2.3.3 Echo State Networks

The main idea of ESNs is to drive a random, large, fixed recurrent neural network with the input signal, thereby inducing in each neuron within this "reservoir" network a non-linear response signal, and combine a desired output signal by a trainable linear combination of all of these response signals.

The ESN structure considered in this thesis is shown in Figure 2.6 [18] and we choose the non-linear functions to be $\phi_h = \tanh$ and ϕ_y as the identity operator.

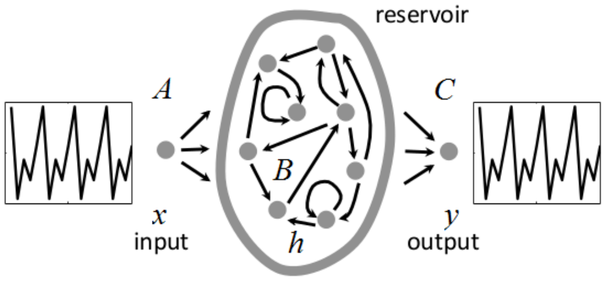


Fig. 2.6 Graphical representation of an echo state network.

Therefore, we can rewrite (2.1) as follows:

$$\begin{cases} h(t) = \tanh\Big(Ax(t) + Bh(t-1) + b\Big) \\ y(t) = Ch(t) + c \end{cases} . \tag{2.39}$$

The task of training an RNN, starting form known input/output periodic time sequences $x(t), y(t)$, is solved with the ESN approach by the following steps:

1. Create a dynamical reservoir with initial random weights.

2. Drive the dynamical reservoir with the training data to obtain the hidden units $h(t)$.

3. Compute output weights as the linear regression weights of the outputs $y(t)$ on the reservoir states $h(t)$.

In order to properly train an RNN with the ESN method, the *echo state property* [30, 31] should hold for the considered network.

Definition 2.1 (Echo state property)**.** Assume an infinite stimulus sequence $x = [x(1), x(2), \ldots]$ and two random initial internal states of the system h_0 and h'_0. To both initial states h_0 and h'_0, the sequences $h = [h_0, h(1), \ldots]$ and $h' = [h'_0, h'(1), \ldots]$ can be assigned;

$$\begin{cases} h(t+1) = F(h(t), x(t)) \\ h'(t+1) = F(h'(t), x(t)) \end{cases}.$$

Then the system is called *universally state contracting* if independently from the set $x(t)$ and for any (h_0, h'_0) and all real values $\varepsilon > 0$, given a distance metric d, there exists an iteration τ for which

$$d(h(t); h'(t)) \leq \varepsilon, \quad \forall t \geq \tau. \tag{2.40}$$

Intuitively, this property says that the effects of initial network state vanish over time: any random initial state of a reservoir is "forgotten", such that after a washout period the current network state is a function of the driver input [18]. On the other hand, the echo state property is connected to algebraic properties of the weight matrix B. Unfortunately, there is no known necessary and sufficient algebraic condition which allows one to decide, given (A, B), whether the network satisfies the echo state property [17]. There is only a sufficient condition for the non-existence of echo states, which is given by the following proposition [32].

Proposition 2.1. *Assume an untrained network* (A, B) *with state update according to* (2.39). *Let B have a spectral radius* $|\lambda_{max}| > 1$, *where* $|\lambda_{max}|$ *is the largest absolute value of an eigenvalue of B. Then, the network has no echo states with respect to any input/output interval containing the zero input/output* $(0, 0)$.

Moreover, there is also a conjecture on the existence of an echo state, which still remains to be shown [17].

Conjecture 2.1. *Let δ, ε be two small positive numbers. Then there exists a network such that, when the reservoir is randomly constructed by generating a random weight matrix B_0 by sampling the weights from a uniform distribution over $[-1,1]$, normalizing B_0 to a matrix B_1 with unit spectral radius, and scaling B_1 to $B_2 = (1-\delta)B_1$, whereby B_2 obtains a spectral radius of $(1-\delta)$, then the network (A, B_2) is an echo state network with probability $1-\varepsilon$.*

The main problem with ESNs is that they do not offer theoretical guarantees and are based only on a sufficient condition for the non-existence of echo states and a conjecture that remains to be shown [17, 18]. In general, whereas a mathematical characterization of neural networks would greatly help to understand their behaviour and design better training algorithms, this has always been an elusive goal, all the more so for recurrent networks because of the presence of cycles.

Thus, in Chapter 3, we contribute to the understanding of the dynamics of recurrent neural networks. Specifically, we establish conditions for the existence of stable limit cycles, whose existence is equivalent to the echo state property. We provide sufficient conditions for the convergence to a trajectory that is uniquely determined by the driving input signal, independently of the initial states. Under these conditions, the hidden-to-hidden matrix may have norm larger than one.
This result can help extending the memory of recurrent neural networks, since earlier work has shown that large matrix norms in the hidden layer imply longer memory duration. This would also increase the design options for recurrent neural networks.

2.4 Applications of Recurrent Neural Networks

In the last decade, machine learning has seen the rise of RNNs methods. As a matter of fact, they are of particular interest since they can deal with a large variety of applications, in many different fields. Here we present the main topics, but there are many other areas where it is possible to apply the RNNs structure.

Natural Language Processing (NLP)

- Language Modelling

    1. RNN based language model with application to speech recognition [6];

2. Neural language model that relies only on character-level inputs and is able to encode both semantic and orthographic information [33];
3. Tree Long Short-Term Memory, a neural network model based on LSTM, which is designed to predict a tree rather than a linear sequence [34].

- Speech Recognition

1. Deep LSTMs on phoneme recognition [7];
2. Attention-mechanism with features needed for speech recognition [35];
3. Accurate model, with fast decoding, for large vocabulary speech recognition [36].

- Machine Translation

1. The RNN Encoder–Decoder consists of two RNNs, one encodes a sequence of symbols into a fixed length vector representation, and the other decodes the representation into another sequence of symbols [8];
2. A multi-layered LSTM maps the input sequence to a vector of a fixed dimensionality, and then another deep LSTM decodes the target sequence from the vector on an English to French translation task [37];
3. Neural machine translation automatically soft-searches for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly [38].

- Conversation Modeling

1. Neural Responding Machine formalizes the generation of response as a decoding process based on the latent representation of the input text, while both encoding and decoding are realized with RNNs [39];
2. Straightforward model, which uses the sequence to sequence framework, predicting the next sentence given the previous sentence or sentences in a conversation [40];
3. Dialogue-based language learning, where supervision is given naturally and implicitly in the response of the dialogue partner during the conversation [41].

- Question Answering

    1. Understanding whether a system is able to answer questions via chaining facts, simple induction, deduction [42];
    2. Attention based deep neural networks that learn to read real documents and answer complex questions with minimal prior knowledge of language structure [43];
    3. Dynamic Memory Network processes input sequences and questions, forms episodic memories, and generates relevant answers [44].

Computer Vision (CV)

- Object Recognition

    1. Convolutional RNN allows to assign a class label to each pixel in an image [9];
    2. Convolutional RNN for object recognition by incorporating recurrent connections into each convolutional layer [45];
    3. RNN architectures for object-class segmentation, labelling each pixel of an image with the class of the object it belongs to [46].

- Image Generation

    1. Deep Recurrent Attentive Writer combines a novel spatial attention mechanism that mimics the fovea of the human eye, with a sequential variational auto-encoding framework that allows for the iterative construction of complex images [47];
    2. LSTM are particularly suited for image modelling due to their spatial structure, scaling to images of arbitrary size [48];
    3. RNN predicts the pixels in an image along the two spatial dimensions [49].

- Video Analysis

    1. Multilayer LSTM to learn representations of video sequences [50];

2. Spatio-temporal video autoencoder, based on a classic spatial image autoencoder and a temporal autoencoder composed of convolutional LSTM cells [51].

Multimodal (CV + NLP)

- Image Captioning
  1. Image captions are generated according to a multimodal RNN modelling the probability distribution of generating a word given previous words and an image [52];
  2. A generative model that combines recent advances in computer vision and machine translation can be used to generate natural sentences describing an image [53];
  3. Multimodal Recurrent Neural Network architecture that uses Convolutional Neural Networks over image regions, bidirectional RNNs over sentences to generate novel descriptions of image regions [10].
- Visual Question Answering
  1. Given an image and a natural language question about the image, the task is to provide an accurate natural language answer [54];
  2. Neural-Image-QA is a multi-modal problem where the language output (answer) is conditioned on visual and natural language input (image and question) [55];
  3. Combining neural networks and visual semantic embeddings, without object detection and image segmentation, it is possible to predict answers to simple questions about images [56].
- Turing Machines
  1. Neural Turing Machines can infer simple algorithms such as copying, sorting, and associative recall from input and output examples [57];
  2. Reinforcement Learning algorithm can train a neural network that interacts with discrete *interfaces* (such as memory Tape, input Tape, and output Tape) to solve simple algorithmic tasks [58].

Robotics

1. A neural sequence-to-sequence model for direction following is a task essential to realizing effective autonomous agents [59];
2. Learning policies with internal memory for high-dimensional, continuous systems, such as robotic manipulators [60].

Chapter 3

Stable Limit Cycles in Recurrent Neural Networks

Brief - *In this chapter, our goal is to contribute to the understanding of the dynamics of recurrent neural networks. Specifically, we establish conditions for the existence of stable limit cycles, whose existence is equivalent to the echo state property. We provide sufficient conditions for the convergence to a trajectory that is uniquely determined by the driving input signal, independently of the initial states. Under these conditions, the hidden-to-hidden matrix may have norm larger than one. This result paves the way for applying training algorithms for echo state networks to a larger number of classes of recurrent neural networks. Moreover, our contribution may help extending the memory of recurrent neural networks, since earlier work has shown that large matrix norms in the hidden layer imply longer memory duration. This would also increase the design options for recurrent neural networks.*

In this chapter, we address the mathematical background of echo state networks [61]. In particular, we provide conditions on the random choice of the weight matrices representing the network in order to improve the echo state-based approach. Exploiting the periodicity of the driving input signal, which is propagated to the hidden state, and the Banach fixed-point theorem [62] to guarantee the uniqueness of the solution, we establish the convergence of the hidden state to a trajectory, *i.e.* a stable limit cycle, which is uniquely determined and independent of the initial state. This means that the periodicity of the input is propagated in the hidden state [63]. Our results allow to relax the conditions on the hidden-to-hidden weight matrix, which were previously thought to require norm lower than one [15]. This relaxation widens, therefore, the cases of RNNs that may be trained as echo state networks. Moreover, our new constraints on the hidden-to-hidden matrix and on the input bias may yield RNNs that display longer memory, since earlier work has shown that large matrix norms in the hidden layer imply longer memory duration [64]. Finally, we also give a geometrical interpretation of our properties and provide examples both in one-dimensional and multidimensional cases.

3.1 Requirements for the Existence of Stable Limit Cycles

In this section, we provide the fundamentals for guaranteeing the existence of a stable limit cycle in the hidden state of a RNN. A limit cycle is a closed trajectory in phase space having the property that at least one other trajectory spirals into it as time approaches infinity.

This concept is closely related to echo states. Let us assume that the hidden state $h(t)$ converges to a stable limit cycle starting from an initial state h_0. Then, starting from another initial state h'_0, the hidden state $h'(t)$ converges approximatively to the same limit cycle, *i.e.* for every $\varepsilon \in \mathbb{R}^+$, there exists an iteration τ such that, for a distance metric d,

$$d(h(t); h'(t)) \leq \varepsilon \quad \forall t \geq \tau,$$

which is the echo state property (2.40).

3.1.1 General Properties

Let us consider (2.1), rewritten in the following way:

$$h(t+1) = \tanh\Big(Bh(t) + P(t+1)\Big), \tag{3.1}$$

where $P(t+1) = Ax(t+1) + b$.

First of all, we have this trivial result on the periodicity of $P(t)$.

Proposition 3.1 (Periodicity of A(t))**.** *If the input sequence $x(t)$ is periodic of period T_p, then $P(t)$ is also periodic of period T_p.*

Proof.

$$P(t+T_p) = Ax(t+T_p) + b = Ax(t) + b = P(t).$$

□

Thus, without loss of generality, we can consider $P(t)$ instead of $x(t)$ as input signal.

Then, we have to prove that there exists a limit cycle.

Proposition 3.2 (Existence of a Cycle of Period T_p)**.** *If $P(t)$ is periodic of period T_p and $\|B\|_2 < 1$, then $h(t)$ has a limit cycle of period T_p.*

Proof.

$$\begin{aligned}
&\lim_{t\to+\infty} \|h(t+T+1) - h(t+1)\|_2 \\
\leq &\lim_{t\to+\infty} \|B\|_2 \|h(t+T_p) - h(t)\|_2 \qquad \text{if } T = T_p \\
\leq &\lim_{t\to+\infty} \|B\|_2^2 \|h(t+T_p-1) - h(t-1)\|_2 \\
\leq &\lim_{t\to+\infty} \|B\|_2^t \|h(T_p+1) - h(1)\|_2 = \ 0 \quad \text{since } \|B\|_2 < 1.
\end{aligned}$$

□

Moreover, we need the limit cycle to be *stable* (or attractive), *i.e.* all the neighbouring trajectories approach the limit cycle as time approaches infinity. Therefore, we want to prove the existence and uniqueness of a fixed-point.

Proposition 3.3 (Existence of a Stable Limit Cycle)**.** *If $\|B\|_2 < 1$ and $P(t)$ is periodic of period T_p, then $h(t)$ has a stable limit cycle.*

Proof. Since $P(t)$ is periodic of period T_p, then we can consider T_p different equations in the form

$$h(t+1) = F(h(t)) = \tanh\Big(Bh(t) + k_i\Big),$$

where $k_i = P(t+i)$, with $i = 1, \dots, T_p$, are constant N-length vectors.
We prove that the map $F : h(t) \mapsto h(t+1)$ is a contraction, then by Banach's fixed-point theorem [62] we can say that F admits a unique fixed-point.
Let us choose two distinct hidden sequences $h(t)$ and $h'(t)$ and fix a constant k, then

$$\begin{aligned}
\|F(h(t)) - F(h'(t))\|_2 &\leq \|Bh(t) + k - Bh'(t) - k\|_2 \\
&\leq \|B\|_2 \|h(t) - h'(t)\|_2.
\end{aligned}$$

Since $\|B\|_2 < 1$, then F is a contraction map and admits a unique attractive fixed-point.
Then, since $P(t)$ is periodic of period T_p, and thus we can consider T_p equations, we have T_p attractive fixed-points, which are periodically repeated. This means that $h(t)$ has a stable limit cycle of period T_p. $\square$

It is possible to give a geometrical interpretation of the existence of a limit cycle. We provide it for $N = 1$, in order to give a graphical representation in a 2-dimensional space.

3.1.2 Geometric Interpretation

In a unidimensional case, if $|B| < 1$, the fixed-point can be found with an iterative algorithm solving the system

$$\begin{cases} z(t) = \tanh(Bh(t)+k) \\ z(t) = h(t) \end{cases}, \tag{3.2}$$

as shown in Figure 3.1, where the red line corresponds to the updating of the iterative solution.

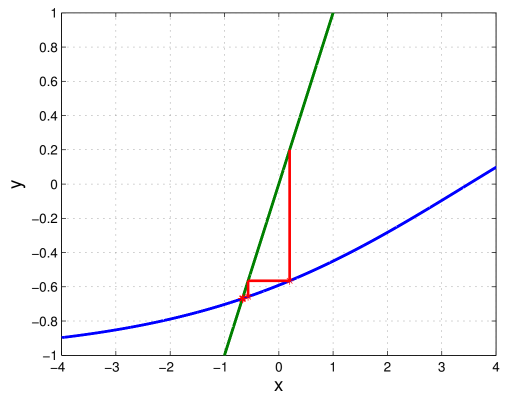


Fig. 3.1 Iterative algorithm to find the unique fixed-point of $h = \tanh(0.19h - 0.68)$.

Using $P(t)$ instead of k, this means that we need to solve T_p systems like the one in (3.2), one for every different value of $P(t)$. For example, in Figure 3.2 we consider $P(t) = \sin(\frac{2}{3}\pi t) - 0.68$, thus $T_p = 3$, which means that we have three fixed-points, one for each different value of $P(t)$, producing a limit cycle of period $T_p = 3$.

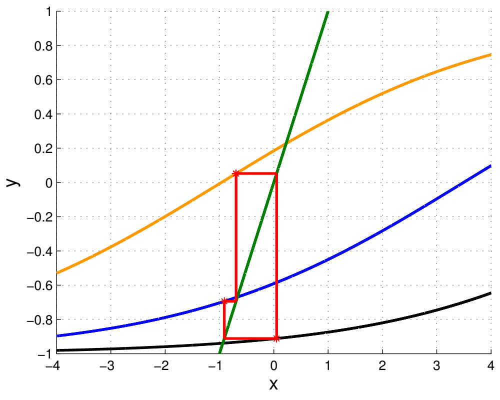


Fig. 3.2 Iterative algorithm to find the limit cycle of period $T_p = 3$ of $h = \tanh(0.19h + \sin(2/3\pi t) - 0.68)$.

Finally, the requirement $|B| < 1$ can be interpreted as the slope of the hyperbolic tangent not to be greater than the one of $z = h$: as a matter of fact, Figure 3.3 shows that, if $B = 3.78 > 1$, then there may exist three fixed-points.

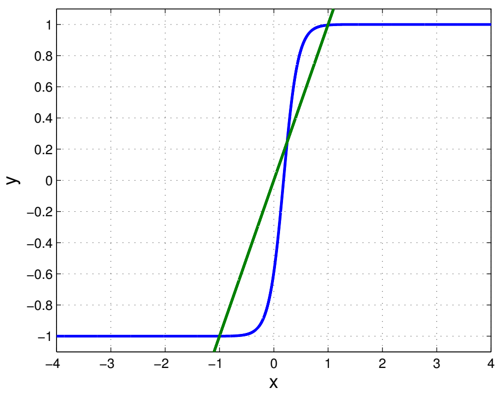


Fig. 3.3 $h = \tanh(3.78h - 0.68)$: since $|B| > 1$, then there may exist multiple fixed-points.

We now turn to the following question: Is it possible to give a condition on b that ensures the existence of a unique attractive fixed-point, even if $|B| > 1$?
Our contribution is to find a new alternative requirement on bias b, in order to relax the condition on the weights B and to allow more classes of RNNs to be trained with the echo state approach.

3.1.3 Alternative Condition on the Bias Vector

We propose a novel condition, providing bounds on b, which guarantee the existence of a stable limit cycle, even though $|B| > 1$.

Proposition 3.4 (Alternative Condition for the Existence of a Stable Limit Cycle)**.** *Let us assume* $|B| > 1$*,* $x(t)$ *periodic of period* T_p*, and*

$$\begin{cases} b < \min\{Ax\} + \hat{k} \\ b > \max\{Ax\} - \hat{k} \end{cases}, \tag{3.3}$$

with $\hat{k} = \operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - W\sqrt{1-\frac{1}{B}}$*. Then, the hidden state in* (3.1) *has a stable limit cycle.*

Proof. These bounds on b can be obtained following this procedure:

1. Find limit h_1, h_2 where $z = h$ is tangent to $h = \tanh(Bh + k)$.
2. Find bounds on c such that
   - $h_1 > \tanh(Bh_1 + k)$,
   - $h_2 < \tanh(Bh_2 + k)$.
3. Find b, recalling that $k = P(t) = Ax(t) + b$.

Below we detail these steps.

1. Find limit h_1, h_2 such that $z = h$ is tangent to $z = \tanh(Bh + k)$.

$$
\begin{aligned}
&\frac{d}{dx}\Big[\tanh(Bh+k)\Big] = \Big[1-\tanh^2(Bh+k)\Big]B = 1\\
&\tanh^2(Bh+k) = 1-\frac{1}{B} \quad \text{if } B \neq 0\\
&\tanh(Bh+k) = \sqrt{1-\frac{1}{B}} \quad \text{if } B < 0 \vee B \geq 1\\
&\begin{cases} h_1 = \frac{1}{B}\Big[\operatorname{atanh}\Big(\sqrt{1-\frac{1}{B}}\Big) - k\Big] \\ h_2 = \frac{1}{B}\Big[\operatorname{atanh}\Big(-\sqrt{1-\frac{1}{B}}\Big) - k\Big] \end{cases} \quad \text{if } B \geq 1.
\end{aligned} \tag{3.4}
$$

For example, in Figure 3.4 we can see the two limit x_1 and x_2, for which there are two fixed-points.

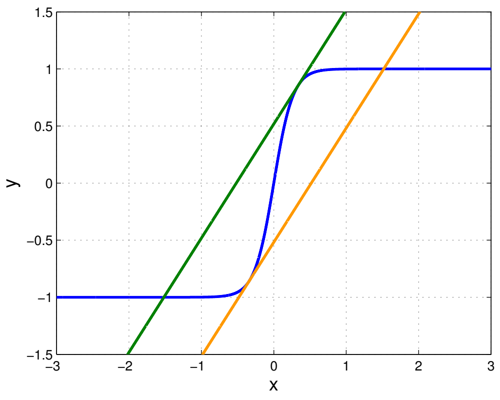


Fig. 3.4 Limit h, for which the lines are tangent to the hyperbolic tangent with $B = 3.78$ and thus, there are two fixed-points.

Thus, referring to Figure 3.4, we allow all the values of h that are on the right of the orange line (*i.e.* h_1) or on the left of the green one (*i.e.* h_2).

2. Since we require the existence of a unique fixed-point, we find the bounds for k replacing (3.4) in

$$\begin{aligned}
& h_1 > \tanh(Bh_1 + k) \\
& h_1 - \tanh(Bh_1 + k) > 0 \\
& \frac{1}{B}\left[\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - k\right] - \tanh\left(B\frac{1}{B}\left[\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - k\right] + k\right) > 0 \\
& \frac{1}{B}\left[\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - k\right] - \tanh\left(\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - k + k\right) > 0 \\
& \frac{1}{B}\left[\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - k\right] - \tanh\left(\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right)\right) > 0 \\
& \frac{1}{B}\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - \frac{k}{B} - \sqrt{1-\frac{1}{B}} > 0 \\
& k < \operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) - B\sqrt{1-\frac{1}{B}} = \hat{k}, \qquad (3.5)
\end{aligned}$$

and, analogously, starting from $h_2 < \tanh(Bh_2 + k)$, we obtain

$$k > -\operatorname{atanh}\left(\sqrt{1-\frac{1}{B}}\right) + B\sqrt{1-\frac{1}{B}} = -\hat{k}. \qquad (3.6)$$

3. Finally, recalling that $k = P(t) = Ax(t) + b$, we can find the condition on b:

$$\begin{cases} b < \min\{Ax\} + \hat{k} \\ b > \max\{Ax\} - \hat{k} \end{cases}. \qquad (3.7)$$

□

Figure 3.5 shows that, even when choosing $B = 3.78 > 1$, if we set $b = -2.97$ according to the bounds in condition (3.7), then there exists a unique fixed-point, which can be found by an iterative algorithm.

This result is obtained in a unidimensional case, but in the same way it is possible to extend Proposition 3.4 to a multidimensional case, providing a condition on b_i, which guarantees the existence of a stable limit cycle even if $||B||_2 > 1$.

Proposition 3.5 (Alternative Condition for the Existence of a Stable Limit Cycle). *Let us assume $||B||_2 > 1$, $x(t)$ periodic of period T_p, and each component i of the*

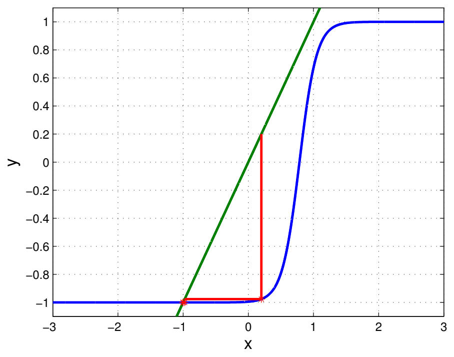


Fig. 3.5 Iterative algorithm to solve $h = \tanh(3.78h - 2.97)$; there is only one fixed-point, even if $B > 1$.

bias vector b satisfying the bounds

$$\begin{cases} b_i < \min\{Ax\} + \hat{k}_i \\ b_i > \max\{Ax\} - \hat{k}_i \end{cases}, \tag{3.8}$$

where $\hat{k}_i = \operatorname{atanh}\left(\sqrt{1 - \frac{1}{\sqrt{s_i}}}\right) - \sqrt{s_i}\sqrt{1 - \frac{1}{\sqrt{s_i}}}$ *with* $s_i = \sum_{j=1}^{N} B_{ij}^2$*. Then, the hidden state in* (3.1) *has a stable limit cycle.*

Proof. Let us assume $N = 2$. Without loss of generality, we can consider each equation of

$$h(t+1) = F(h(t)) = \tanh(Bh(t) + k)$$

separately, *i.e.*

$$\begin{cases} f_1(h_1, h_2) = \tanh(B_{11}h_1 + B_{12}h_2 + k_1) \\ f_2(h_1, h_2) = \tanh(B_{21}h_1 + B_{22}h_2 + k_2) \end{cases}.$$

Let us consider the first equation. Then, the bounds can be computed following a procedure, which is analogous to the one in proof of Proposition 3.4.

1. We look for the tangent plane to the surface
   $z = f_1(x,y)$ in (x_0,y_0):

$$\begin{aligned} z =& f_1(x_0,y_0) + \left.\frac{\partial f_1}{\partial x}\right|_{x=x_0}(x-x_0) + \left.\frac{\partial f_1}{\partial y}\right|_{y=y_0}(y-y_0) \\ =& \tanh(B_{11}x_0 + B_{12}y_0 + c_1) \\ &+ \left[1 - \tanh^2(B_{11}x_0 + B_{12}y_0 + k_1)\right] B_{11}(x-x_0) \\ &+ \left[1 - \tanh^2(B_{11}x_0 + B_{12}y_0 + k_1)\right] B_{12}(y-y_0). \end{aligned} \tag{3.9}$$

2. We want a plane such that its normal vector forms an angle of $\pi/4$ with its projection on the plane xy. Say the normal vector and its projection on the plane xy, respectively,

$$n = \begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} \qquad \text{and} \qquad n_{xy} = \begin{bmatrix} f_x \\ f_y \\ 0 \end{bmatrix},$$

   where $f_x = \frac{\partial f_1}{\partial x}$ and $f_y = \frac{\partial f_1}{\partial y}$ are the partial derivatives of f_1.
   Thus,

$$\begin{aligned} &\frac{n^\top n_{xy}}{\|n\|\|n_{xy}\|} = cos(\pi/4) = \sqrt{2}/2 \\ &\left(f_x^2 + f_y^2\right)^2 = \frac{1}{2}\left(f_x^2 + f_y^2 + 1\right)\left(f_x^2 + f_y^2\right) \\ &f_x^2 + f_y^2 = 1 \\ &\left[1 - \tanh^2(B_{11}x + B_{12}y + k_1)\right]^2 \left(B_{11}^2 + B_{12}^2\right) = 1 \\ &\tanh^2(B_{11}x + B_{12}y + k_1) = 1 \mp \frac{1}{\sqrt{s}}, \qquad \text{where } s = B_{11}^2 + B_{12}^2 \\ &B_{11}x + B_{12}y = \operatorname{atanh}\left(\pm\sqrt{1 - 1/\sqrt{s}}\right) - k_1. \end{aligned} \tag{3.10}$$

3. Evaluating (3.10) in (x_0, y_0) and substituting it in (3.9), we obtain:

$$\begin{aligned}
z &= \tanh(\operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right) - k_1 + k_1) \\
&+ \left[1-\tanh^2(\operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right))\right] B_{11}(x-x_0) \\
&+ \left[1-\tanh^2(\operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right))\right] B_{12}(y-y_0) \\
&= \pm\sqrt{1-\frac{1}{\sqrt{s}}} + \frac{1}{\sqrt{s}}\left[B_{11}x + B_{12}y - \operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right) + k_1\right]. \qquad (3.11)
\end{aligned}$$

4. Moreover our desired plane should pass through the origin (0,0), thus:

$$z = \frac{1}{\sqrt{s}}\Big(B_{11}x + B_{12}y\Big) \qquad (3.12)$$

5. Solving the system of equations (3.11) and (3.12)

$$\Big(B_{11}x + B_{12}y\Big) = \pm\sqrt{s}\sqrt{1-\frac{1}{\sqrt{s}}} + \left[B_{11}x + B_{12}y - \operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right) + k_1\right]$$

$$\begin{aligned}
k_1 &= \operatorname{atanh}\left(\pm\sqrt{1-\frac{1}{\sqrt{s}}}\right) \mp \sqrt{s}\sqrt{1-\frac{1}{\sqrt{s}}} \\
k_1 &= \pm\operatorname{atanh}\left(\sqrt{1-\frac{1}{\sqrt{s}}}\right) \mp \sqrt{s}\sqrt{1-\frac{1}{\sqrt{s}}} \quad ,
\end{aligned}$$

we obtain the bounds on k_1:

$$\begin{cases} k_1 < \operatorname{atanh}\left(\sqrt{1-\frac{1}{\sqrt{s}}}\right) - \sqrt{s}\sqrt{1-\frac{1}{\sqrt{s}}} = \hat{k} \\ k_1 > -\operatorname{atanh}\left(\sqrt{1-\frac{1}{\sqrt{s}}}\right) + \sqrt{s}\sqrt{1-\frac{1}{\sqrt{s}}} = -\hat{k} \end{cases} , \qquad (3.13)$$

which can be easily converted into conditions on b_1. The same reasoning can be applied for the second equation and thus obtaining a condition on k_2 and consequently on b_2, with $s = B_{21}^2 + B_{22}^2$.

A graphical example on how to obtain the bounds to have two tangent planes is shown in Figure 3.6.

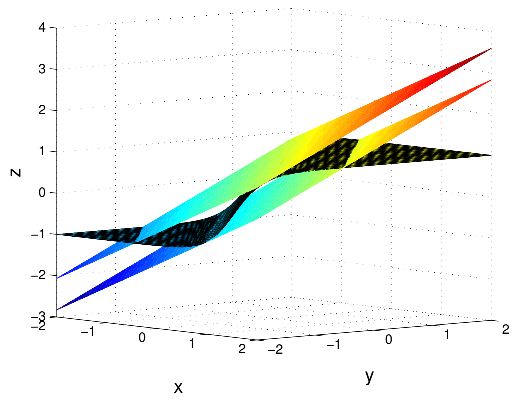


Fig. 3.6 The bounds for b are given by the tangent planes.

Notice that in the case $N = 1$, $s = B^2$ and thus (3.13) reduces obviously to (3.5)-(3.6), just substituting $\sqrt{s} = B$. This means that all our reasoning can be extended for a generic N, considering

$$s = s_i = \sum_{j=1}^{N} B_{ij}^2,$$

for each equation i, leading to a condition on b_i, which guarantees the existence of a stable limit cycle even if $||B||_2 > 1$.

$\square$

Finally, for example, let us consider $N = 2$ and scale a random weight matrix B in order to have $||W||_2 = 3.78 > 1$ and b according to our conditions (3.8). Driving a periodic input signal $x(t)$ of period $T_p = 5$, we propagate this periodicity to the hidden state h, which reaches a stable limit cycle. Figure 3.7 shows the phase plot of the hidden state. We can see that, starting from two different initial conditions (blue dots), the trajectory converges approximatively to the same limit cycle (represented in green) after some iterations (red lines).

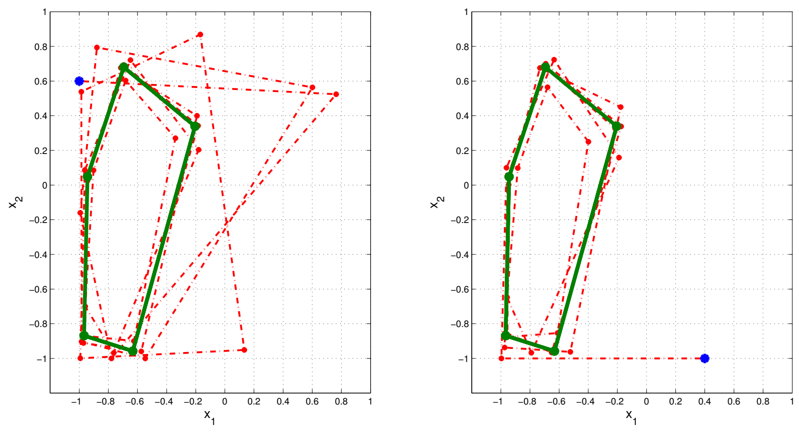


Fig. 3.7 Phase plot showing approximatively the same limit cycle (green lines) for different initial condition (blue dots).

3.2 Discussion

The purpose of this chapter is to provide some conditions to obtain a stable limit cycle for recurrent neural networks. We prove the existence and uniqueness of the limit cycle, which is independent from the initial condition thanks to the notion of attractive fixed-point and the propagation of the periodicity of the input signal to the hidden state.

Moreover, exploiting a geometrical interpretation of the considered system, we provide also a relaxation of the requirements on the weight matrix: we introduce an alternative condition on the bias vector, which guarantees the convergence of the hidden state to a stable limit cycle, even though the hidden-to-hidden matrix norm is greater than one. Thus, our contribution is not only to widen the number of classes of RNNs that can be trained with the ESN approach, but also to extend the memory of RNNs, since large matrix norms imply longer memory duration [64].

Chapter 4

Controlling the Gradient of Recurrent Neural Networks

Brief - *In this chapter, we address the main problem that makes RNNs really hard to train: the so-called vanishing gradient problem. We propose a novel approach to overcome this problem, exploiting the singular values decomposition of the specific factor in the gradient we identify to be the cause of the vanishing phenomenon. Following a probabilistic approach, based on the random matrices theory, we provide tight bounds to the singular values and provide conditions about their root mean square, making the gradient not to vanish even for very deep networks.*

In learning based on gradient descent, the parameters θ are updated iteratively by moving in a direction opposite to the (estimated) gradient of an error function (2.26). If the estimated gradient with respect to the coefficients in early layers approaches zero before a minimum is reached, then learning for these layers actually stops – this phenomenon is known as the vanishing gradient problem. If the gradient becomes enormous, then the parameter is likely to be moved to a value that is very far from the optimum. This is known as the exploding gradient problem.

The goal of this chapter is to quantify and control the norm of the gradient of the loss function

$$||\nabla_\theta \mathscr{L}|| = \left\| \sum_{t=t_0}^{T} \nabla_\theta \mathscr{L}_t \right\|,$$

such that the gradient neither vanishes nor explodes.

As is known from the literature [11, 65, 66], and as will be seen in what follows, the norm of the gradient of the loss function (with respect to the parameters of the layer k $\theta(k)$) across network depth k is exponentially bounded above

$$\|\nabla_{\theta(k)}\mathscr{L}\| \leq c_1\alpha^k. \tag{4.1}$$

With increasing depth, these bounds may either tend to zero or grow without limits, depending on the values of the positive constant c_1 and α.

By the triangle inequality we have that

$$\|\nabla_\theta\mathscr{L}\| \leq \sum_{t=t_0}^{T} \|\nabla_\theta\mathscr{L}_t\|, \tag{4.2}$$

so an upper bound to the error gradient norm may be found by summing the single terms. Unfortunately, a lower bound cannot be found in the same way, since terms that are not small may cancel each other so the sum has a norm near zero. However, we do not however expect that cancellations dominate, since the terms tend to be correlated such that most terms contribute in roughly the same direction. We will therefore still examine lower bounds to the single terms and take their sum simply as a good indication of the norm of the error gradient. By the same assumptions, we expect that the upper bound in (4.2) does not exaggerate the true value.

To control the constants α in (4.1), we will assign a variable cost $\mathscr{K}$ whenever they exceed predefined limits, in proportion to the amount of deviation from these limits. This cost will be added to the original error function, such that the update rule becomes

$$\theta \leftarrow \theta - \mu\nabla_\theta\left(\mathscr{L} + \lambda\mathscr{K}\right),$$

where $\lambda > 0$ is a regularization factor.

This chapter is organized as follows: in the first Section we explain how to bound the norm of the gradient of the loss function. However, this bound is too weak, thus, in Section 4.2, we propose new conditions to avoid the gradient to vanish in a very deep network, exploiting singular values and singular vectors and random singular matrices. Then, in Sections 4.3 and 4.4, we provide bounds for the singular values and their root mean square, respectively. Finally, we analyse some experimental results in Section 4.5 and give some concluding remarks in Section 4.6. Notice also that most of the proofs for this chapter are provided in Appendix B.

4.1 Loose power law bound

We demonstrate that the gradient of any single term of the error function indeed follows an inequality such as (4.1).

As shown in Section 2.2, the gradient of the loss function (with respect to the bias vector $b^{(k)}$) may be written as the product of three factors $r^\top$ (a row vector), P and Q (matrices),

$$\nabla_{b^{(k)}} \mathscr{L}_t = r^\top P Q,$$

where

$$\begin{aligned} r^\top &= \frac{\partial \mathscr{L}_t}{\partial \hat{y}(t)} \phi_y'(Ch(t)+c)C \\ P &= \prod_{m=0}^{k-1} \phi_h'(z_h(t-m))B^{(m)} \\ Q &= \phi_h'(z_h(t-k)). \end{aligned} \tag{4.3}$$

The factors $r^\top, P, Q$ are functions of time, but we omit this in the notation. We also use the shorthand

$$z_h(t-m) = A^{(m)}x(t-m) + B^{(m)}h(t-m-1) + b^{(m)}.$$

The gradients with respect to the matrices $A^{(k)}$ and $B^{(k)}$ have a very similar factorization. For each, a matrix is post-multiplied as

$$\begin{aligned} \nabla_{A^{(k)}} \mathscr{L}_t &= \nabla_{b^{(k)}} \mathscr{L} f(x(t-k)) \\ \nabla_{B^{(k)}} \mathscr{L}_t &= \nabla_{b^{(k)}} \mathscr{L} g(h(t-k-1)), \end{aligned}$$

where the matrices f and g are functions of the input at time $t-k$ and previous state at time $t-k-1$, respectively. The exponential factors in the bounds to the norm in Equation (4.1) are the same, so we proceed with the common factor of the gradient, $\nabla_{b^{(k)}} \mathscr{L}_t$.

In the following, we let $\sigma_1(\cdot)$ denote the largest singular value of a matrix. The largest singular value of the matrix PQ provides an upper bound to its relative amplification,

$$\|\nabla_{b^{(k)}} \mathscr{L}\| \leq \|r\| \sigma_1(PQ).$$

Furthermore, the largest singular value of the product PQ is bounded by

$$\sigma_1(PQ) \leq \sigma_1(P)\sigma_1(Q),$$

therefore

$$\|\nabla_{b^{(k)}}\mathscr{L}\| \leq \|r\|\sigma_1(P)\sigma_1(Q). \tag{4.4}$$

The number of factors of P grows with depth k (see Equation (4.3)), and it is this matrix that yields the power law seen in the inequality of Equation (4.1). Let $J(m)$ denote one factor in P of Equation (4.3)

$$J(m) = \phi_h'(z_h(t-m))\,B^{(m)}.$$

For singular values of products of matrices we have that

$$\sigma_1\left(\prod_{m=0}^{k-1} J(m)\right) \leq \prod_{m=0}^{k-1} \sigma_1(J(m)),$$

so combining this with Equation (4.4) yields

$$\|\nabla_{b^{(k)}}\mathscr{L}_t\| \leq \|r\|\sigma_1(Q)\prod_{m=0}^{k-1}\sigma_1(J(m)). \tag{4.5}$$

If we replace the products of singular values by the obvious upper bound provided by the largest factor, we obtain the power law of Equation (4.1) with

$$c_1 = \|r\|\sigma_1(Q),$$
$$\alpha = \max_m \sigma_1(J(m)).$$

Thus, if $\max_m \sigma_1(J(m)) < 1$, then the gradient diminishes exponentially with depth and, even if more computing power can help, anyway it does not solve the problem. Otherwise, if $\max_m \sigma_1(J(m)) > 1$, the gradient may grow exponentially. Moreover, the gradient could be zero also if it is stuck in a flat zone or in a saddle-point. For this reason, we shall see that it is indeed possible to control the singular values and that one may find better constraints, in a way that eliminates or al least greatly reduces the vanishing/exploding gradient problem.

4.2 Conditions for non-vanishing gradients

The bound to the gradient norm in (4.5) is very loose, and only in very special cases this bound is reached. By taking a probabilistic approach, it becomes possible to study what is expected to happen, what happens in the large majority of cases, or even what happens almost always.

As we shall see, the gradient norm usually remains in a much narrower interval. Our goal is to define a procedure that avoids that the gradient vanishes across a very deep network.

4.2.1 Singular values and vectors

The key is to consider the *singular value decomposition* (SVD) of each of the matrices $J(0),\dots,J(k-1)$

$$J(m) = U^{(m)}\Sigma^{(m)}V^{(m)^\top},$$

where

$$U^{(m)} = \begin{bmatrix} u_1^{(m)} & \cdots & u_N^{(m)} \end{bmatrix}, V^{(m)} = \begin{bmatrix} v_1^{(m)} & \cdots & v_N^{(m)} \end{bmatrix}, \Sigma^{(m)} = \begin{bmatrix} \sigma_1^{(m)} & & \\ & \ddots & \\ & & \sigma_N^{(m)} \end{bmatrix}.$$

The norm of the product P is bounded by

$$\begin{aligned} \|P\| &= \|J(0)\cdots J(k-1)\| \\ &= \|\Sigma^{(0)}V^{(0)^\top}\cdots U^{(k-1)}\Sigma^{k-1}\| \\ &\leq \max\Sigma^{(0)}\|V^{(0)^\top}\cdots U^{(k-1)}\|\max\Sigma^{k-1} \\ &\leq \max\Sigma^{(0)}\max\Sigma^{(1)}\cdots\max\Sigma^{(k-1)}. \end{aligned}$$

This upper bound is reached in the very unlikely case that a dominant principal component is projected entirely onto the following dominant principal component, *i.e.* $v_1^{(0)} = \pm u_1^{(1)}, v_1^{(1)} = \pm u_1^{(2)}, \dots, v_1^{(k-2)} = \pm u_1^{(k-1)}$.

Analysing by components the product

$$P = \prod_{m=0}^{k-1} J(m) = \left(\sum_{n_0=1}^{N} \sigma_{n_0}^{(0)} u_{n_0}^{(0)} v_{n_0}^{(0)\top} \right) \cdots \left(\sum_{n_{k-1}=1}^{N} \sigma_{n_{k-1}}^{(k-1)} u_{n_{k-1}}^{(k-1)} v_{n_{k-1}}^{(k-1)\top} \right),$$

where $\sigma_{n_m}^{(m)}$ is the n_m^{th} largest singular value of $J(m)$, and $u_{n_m}^{(m)}, v_{n_m}^{(m)}$ are the associated singular vectors, it may be rewritten as

$$\begin{aligned} P &= \sum_{n_0=1}^{N} \sigma_{n_0}^{(0)} \sum_{n_1=1}^{N} \sigma_{n_1}^{(1)} \cdots \sum_{n_{k-1}=1}^{N} \sigma_{n_{k-1}}^{(k-1)} u_{n_0}^{(0)} v_{n_0}^{(0)\top} u_{n_1}^{(1)} v_{n_1}^{(1)\top} \cdots u_{n_{k-1}}^{(k-1)} v_{n_{k-1}}^{(k-1)\top} \qquad (4.6) \\ &= \sum_{n_0=1}^{N} \sum_{n_{k-1}=1}^{N} \sigma_{n_0}^{(0)} \sigma_{n_{k-1}}^{(k-1)} \underbrace{\left(\sum_{n_1=1}^{N} \sigma_{n_1}^{(1)} \cdots \sum_{n_{k-2}=1}^{N} \sigma_{n_{k-2}}^{(k-2)} v_{n_0}^{(0)\top} u_{n_1}^{(1)} v_{n_1}^{(1)\top} \cdots u_{n_{k-1}}^{(k-1)} \right)}_{\text{scalar}} u_{n_0}^{(0)} v_{n_{k-1}}^{(k-1)\top}. \end{aligned}$$

We observe that the quantity contained within parentheses is a scalar. This is so, because all the singular vectors involved in these sums form scalar products: $(v_{n_0}^{(0)\top} u_{n_1}^{(1)})$, $(v_{n_1}^{(1)\top} u_{n_2}^{(2)})$, and so on. Equation (4.6) may, thus, be written in matrix form

$$P = U^{(0)} \Sigma^{(0)} G \Sigma^{(k-1)} V^{(k-1)\top},$$

where $U^{(0)}$ contains the left singular vectors of $J(0)$, $V^{(k-1)}$ contains the right singular vectors of $J(k-1)$, and $\Sigma^{(0)}, \Sigma^{(k-1)}$ contain the singular values of $J(0)$ and $J(k-1)$, respectively, and the $(n_0, n_{k-1})^{\text{th}}$ element of the matrix G is the scalar observed above, *i.e.*

$$G(n_0, n_{k-1}) = \left(\sum_{n_1=1}^{N} \sigma_{n_1}^{(1)} \cdots \sum_{n_{k-2}=1}^{N} \sigma_{n_{k-2}}^{(k-2)} v_{n_0}^{(0)\top} u_{n_1}^{(1)} v_{n_1}^{(1)\top} \cdots u_{n_{k-1}}^{(k-1)} \right). \qquad (4.7)$$

If we write the inner products as

$$\begin{aligned} z_{01}(n_0, n_1) &= \left(v_{n_0}^{(0)\top} u_{n_1}^{(1)} \right) \\ z_{12}(n_1, n_2) &= \left(v_{n_1}^{(1)\top} u_{n_2}^{(2)} \right) \\ &\vdots \end{aligned}$$

where each factor z is formed by singular vectors for different matrices, then Equation (4.7) becomes

$$G(n_0,n_{k-1}) = \sum_{n_1=1}^{N} \cdots \sum_{n_{k-2}=1}^{N} \sigma_{n_1}^{(1)} \cdots \sigma_{n_{k-2}}^{(k-2)} \cdot z_{01}(n_0,n_1) \cdots z_{k-2,k-1}(n_{k-2},n_{k-1}).$$

In order to characterize the norm of the product of Jacobians, we should quantify the norm of the matrix G, modelling the singular values $\sigma_{n_1}^{(1)} \ldots \sigma_{n_{k-2}}^{(k-2)}$ as constant and the inner products $z_{01}(n_0,n_1) \cdots z_{k-2,k-1}(n_{k-2},n_{k-1})$ as random. The following subsection discusses and justifies this approach.

4.2.2 Random singular matrices

In this paragraph we explain why the singular values can be considered as constants and why the internal products can be considered stochastic and even independent.

If we consider the input $x(t-m), m = 0,1,\ldots$ as random vectors, then

$$q(m) = Ax(t-m) + Bh(t-m-1) + b$$

is a random vector, so

$$J(m) = \Phi'(q(m))B$$

is also random. Under a certain condition, the singular matrices are sensitive to changes in the elements of $J(m)$, caused by the random variation in $q(m)$. In that case, the inner products $\left(v_{n_i}^{(i)\top} u_{n_{i+1}}^{(i+1)} \right)$ vary by chance.

A motivation to this can be found in [67], where it is proved that the gradient of the singular values with respect to the Jacobians are

$$\frac{\partial \sigma_n}{\partial J_{ij}} = U_{in}V_{jn} \in [-1,1],$$

but it is mostly close to 0. Moreover the derivative of U and V with respect to the Jacobians are

$$\frac{\partial U}{\partial J_{ij}} = \sqrt{\sum_{k=1}^{N} \sum_{l=1}^{N} \upsilon_{kl}^{ij^2}} \qquad \text{and} \qquad \frac{\partial V}{\partial J_{ij}} = \sqrt{\sum_{k=1}^{N} \sum_{l=1}^{N} \omega_{kl}^{ij^2}},$$

where υ_{kl}^{ij} and ω_{kl}^{ij} are solutions of the linear system

$$\underbrace{\begin{bmatrix} \sigma_k & \sigma_l \\ \sigma_l & \sigma_k \end{bmatrix}}_{S_{kl}} \begin{bmatrix} \upsilon_{kl}^{ij} \\ \omega_{kl}^{ij} \end{bmatrix} = \begin{bmatrix} U_{ik}V_{jl} \\ -U_{il}V_{jk} \end{bmatrix}.$$

However, the condition number of this system is

$$\operatorname{cond}(S_{kl}) = \frac{\sigma_k + \sigma_l}{|\sigma_k - \sigma_l|}, \tag{4.8}$$

which means that if $\sigma_k \approx \sigma_l$, then $\upsilon_{kl}^{ij}, \omega_{kl}^{ij}$ are very sensitive to changes in J_{ij}, and therefore also $\left\|\frac{\partial U}{\partial J_{ij}}\right\|, \left\|\frac{\partial V}{\partial J_{ij}}\right\|$. For this reason, many singular values should be close and thus we keep them in a narrow interval. Moreover, as shown in Appendix B.1, this also provides independence of the inner products $z_i(n_i, n_{i+1}) = \left(v_{n_i}^{(i)^T} u_{n_{i+1}}^{(i+1)}\right)$.

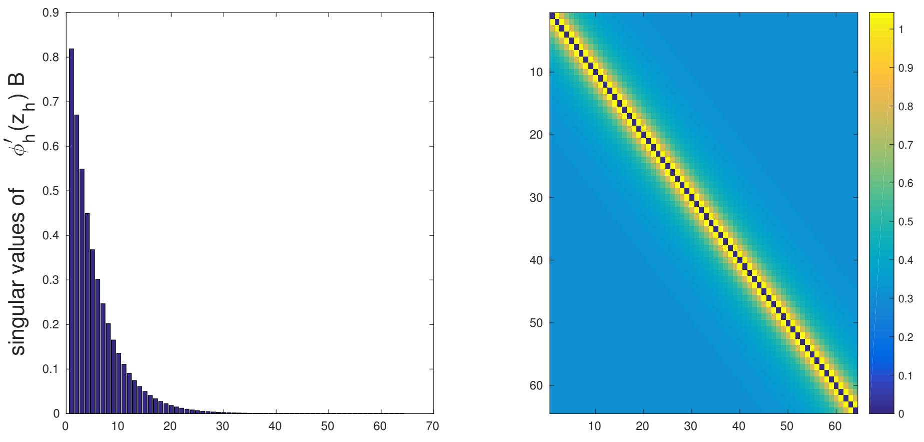


Fig. 4.1 Singular values exponentially decaying (on the left side) and respective logarithm of the condition number for every k, l (on the right side).

In Figures 4.1-4.2, we test two different choices for the singular values of the Jacobians. We first set the singular values to decay exponentially (left side of Fig. 4.1) and we observe the condition number (4.8) for $k, l = 1, \ldots, 64$ which is in the order of magnitude of 10. Otherwise, if we set the singular values to be bounded in a narrow interval, such as $[0.9, 1.1]$ (left side of Fig. 4.2), then the condition number is much greater (order of magnitude of $10^5 - 10^6$), which implies that $\upsilon^{ij}_{kl}, \omega^{ij}_{kl}$ might be numerically large, amplifying the changes in J_{ij}.

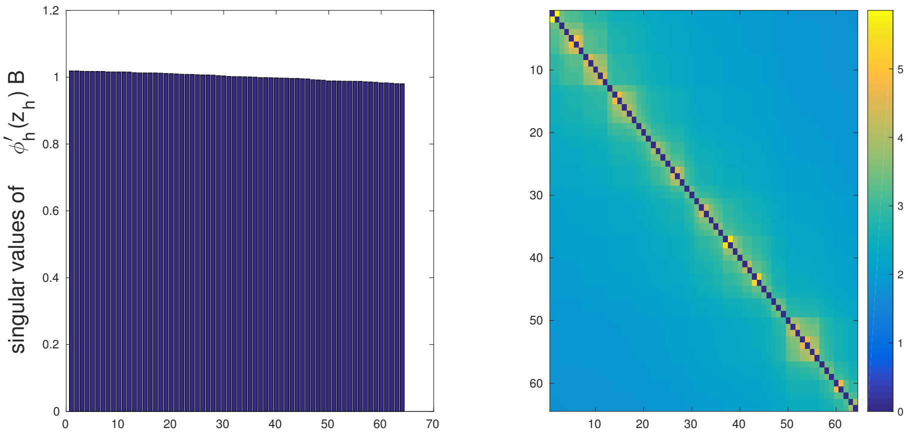


Fig. 4.2 Singular values chosen in a narrow interval (on the left side) and respective logarithm of the condition number for every k,l (on the right side).

4.2.3 The expected norm of the product of Jacobians

Having established that the matrix G is random, we need to show that its elements all have the same mean and variance. This will be sufficient to establish bounds for its norm, as given by the following proposition.

Proposition 4.1. *The expected norm of a random matrix M with $N \times N$ elements of s^2 variance and zero mean lies between the extremes:*

$$s\sqrt{N} \leq \mathbb{E}\left[\|M\|_2\right] \leq sN. \tag{4.9}$$

In particular, if the elements are i.i.d. $\mathcal{N}(0, s^2)$, then

$$\mathbb{E}\left[\|M\|_2\right] \leq 2s\sqrt{N}. \tag{4.10}$$

Proof. The proof is provided in Appendix B.2. □

Therefore, in order to estimate the norm of the matrix G, we need to find the two first statistical moments of its elements. In order to find the variance, we note that

$$\mathrm{Var}(G(n_0,n_{k-1})) = \mathbb{E}\left[G(n_0,n_{k-1})^2\right] - \mathbb{E}\left[G(n_0,n_{k-1})\right]^2. \tag{4.11}$$

First of all we evaluate the expectation of the square of the matrix G

$$\begin{aligned}
&\mathbb{E}\left[G(n_0,n_{k-1})^2\right] \\
&= \sum_{n_1,\dots,n_{k-2},m_1,\dots,m_{k-2}=1}^{N} \sigma_{n_1}^{(1)}\cdots\sigma_{m_{k-2}}^{(k-2)}\mathbb{E}\left[z_0(n_0,n_1)z_0(m_0,m_1)z_1(n_1,n_2)z_1(m_1,m_2)\cdots\right] \\
&= \sum_{n_1,\dots,n_{k-2},m_1,\dots,m_{k-2}=1}^{N} \sigma_{n_1}^{(1)}\cdots\sigma_{m_{k-2}}^{(k-2)}\mathbb{E}\left[z_0(n_0,n_1)z_0(m_0,m_1)\right]\mathbb{E}\left[z_1(n_1,n_2)z_1(m_1,m_2)\right]\cdots,
\end{aligned}$$

by linearity of the expectation and independence of z_i and z_j (see Appendix B.1). Moreover, as shown in Appendix B.5

$$\mathbb{E}[z_i(n_i,n_{i+1})z_i(m_i,m_{i+1})] = \delta_{n_i,m_i}\delta_{n_{i+1},m_{i+1}}\left(\frac{1}{N}\right),$$

where δ is the Kronecker delta function, it follows that

$$\begin{aligned}
\mathbb{E}\left[G(n_0,n_{k-1})^2\right] &= \sum_{n_1,\dots,n_{k-2}=1}^{N} \sigma_{n_1}^{(1)^2}\cdots\sigma_{n_{k-2}}^{(k-2)^2}\left(\frac{1}{N}\right)^{k-1} \\
&= \left(\sum_{n_1=1}^{N}\sigma_{n_1}^{(1)^2}\right)\cdots\left(\sum_{n_{k-2}=1}^{N}\sigma_{n_{k-2}}^{(k-2)^2}\right)\left(\frac{1}{N}\right)^{k-1} \\
&= \left(\frac{1}{N}\sum_{n_1=1}^{N}\sigma_{n_1}^{(1)^2}\right)\cdots\left(\frac{1}{N}\sum_{n_{k-2}=1}^{N}\sigma_{n_{k-2}}^{(k-2)^2}\right)\frac{1}{N}.
\end{aligned} \tag{4.12}$$

Then, we compute the expectation of the elements of the matrix G:

$$\begin{aligned}
\mathbb{E}\left[G(n_0,n_{k-1})\right] &= \sum_{n_1,\dots,n_{k-2}=1}^{N} \sigma_{n_1}^{(1)}\cdots\sigma_{n_{k-2}}^{(k-2)}\mathbb{E}\left[z_0(n_0,n_1)\cdots z_{k-2}(n_{k-2},n_{k-1})\right] \\
&= \sum_{n_1,\dots,n_{k-2}=1}^{N} \sigma_{n_1}^{(1)}\cdots\sigma_{n_{k-2}}^{(k-2)}\mathbb{E}\left[z_0(n_0,n_1)\right]\cdots\mathbb{E}\left[z_{k-2}(n_{k-2},n_{k-1})\right] \\
&= 0,
\end{aligned} \tag{4.13}$$

by linearity of the expectation, independence of z_i and z_j (see Appendix B.1), and since $\mathbb{E}[z_i(n_i, n_{i+1})] = 0$, as shown in Appendix B.3.

Thus, combining (4.12) and (4.13), we can rewrite (4.11) as

$$\mathrm{Var}(G(n_0, n_{k-1})) = \left(\frac{1}{N}\sum_{n_1=1}^{N} \sigma_{n_1}^{(1)^2}\right) \cdots \left(\frac{1}{N}\sum_{n_{k-2}=1}^{N} \sigma_{n_{k-2}}^{(k-2)^2}\right)\frac{1}{N}.$$

If the singular values have the same mean square μ, then

$$\mathrm{Var}(G(n_0, n_{k-1})) = \mu^{k-2}\frac{1}{N}. \tag{4.14}$$

We can notice that, if $\mu = 1$, then $\mathrm{Var}(G(n_0, n_{k-1})) = \frac{1}{N}$, independently from the depth k of the network.

Moreover, if we replace the variance of (4.14) in our Proposition 4.1, for (4.9) we have

$$\mu^{\frac{k-2}{2}} \leq \mathbb{E}[\|G\|_2] \leq \sqrt{N}\mu^{\frac{k-2}{2}} \tag{4.15}$$

and, in case of independence, for (4.10)

$$\mathbb{E}[\|G\|_2] \leq 2\mu^{\frac{k-2}{2}}. \tag{4.16}$$

Thus, the norm of the product of the Jacobians is

$$\|P\| = \left\|\prod_{m=0}^{k-1} J(m)\right\| = \|U^{(0)}\Sigma^{(0)}G\Sigma^{(k-1)}V^{(k-1)^\top}\| = \|\Sigma^{(0)}G\Sigma^{(k-1)}\|,$$

which can be bound using (4.15) or (4.16) respectively

$$\begin{aligned} \mu^{\frac{k-2}{2}}\sigma_1^{(0)}\sigma_1^{(k-1)} \leq &\mathbb{E}[\|P\|_2] \leq \sqrt{N}\mu^{\frac{k-2}{2}}\sigma_1^{(0)}\sigma_1^{(k-1)} \\ &\mathbb{E}[\|P\|_2] \leq 2\mu^{\frac{k-2}{2}}\sigma_1^{(0)} \qquad \text{in case of independence.} \end{aligned}$$

The ideal case is obtained when the root mean square (RMS) of the singular values is one,

$$RMS(\sigma_{n_i}^{(i)}) = \sqrt{\frac{1}{N}\sum_{n_i=1}^{N} \sigma_{n_i}^{(i)^2}} = \mu^{1/2} = 1, \forall i, \tag{4.17}$$

and the biggest singular values are the same for all layers,

$$\sigma_1^{(0)} = \ldots = \sigma_1^{(k-1)} = \sigma_1 .$$

Then the norm of the product of the Jacobians can be bounded

$$\begin{aligned} \sigma_1^2 \leq & \mathbb{E}\left[\|P\|_2\right] \leq \sqrt{N}\sigma_1^2 \\ & \mathbb{E}\left[\|P\|_2\right] \leq 2\sigma_1^2 \qquad \text{in case of independence,} \end{aligned}$$

independently from the depth k of the network. This means that regardless of depth of the network, the product of Jacobians will never cause the gradient to approach zero.

The only possible remaining causes for a zero gradient must be found in the factors r and Q of Equation (4.3). The factor Q is the Jacobian of the non-linearity of the k^{th} layer; if the non-linearity is not completely saturated, this factor should never become zero. This leaves the factor r, which is proportional to the gradient of the loss (i.e. the error committed by the network); it is zero when a lower loss cannot be found at any nearby point in the output space.

4.3 Bounding the singular values to an interval

The previous paragraph presented a new and additional optimality condition for RNNs. This condition alone does not imply how such a network should be trained. However, it is possible to incorporate the condition in a regularization term that may be added to the cost function, such that gradient descent can be applied as before.

In this paragraph, therefore, we propose a technique that will push the singular values of J into some admissible interval. This will be done by adding a regularization term to the error function. The term assigns a cost to those singular values that are outside the admissible interval. The more a singular value exceeds a bound, the higher the cost is going to be, and a good choice for a cost term is the sum of squared excesses. For the lower bound ρ, the cost term is the sum with contributions from all σ_n below it

$$\mathscr{K}_{lo}(\sigma_1, \ldots, \sigma_N) = \frac{1}{2} \sum_{\sigma_n < \rho} (\sigma_n - \rho)^2 . \tag{4.18}$$

For the upper bound τ, the cost term is the sum with contributions from all σ_n above it

$$\mathscr{K}_{up}(\sigma_1,\ldots,\sigma_N) = \frac{1}{2}\sum_{\sigma_n>\tau}(\sigma_n-\tau)^2.$$

A cost zero is reached when all singular values are within the bounds

$$\begin{gathered}\rho \leq \sigma_n \leq \tau, \qquad n=1,\ldots,N \\ \Downarrow \\ \mathscr{K}_{lo}+\mathscr{K}_{up}=0.\end{gathered}$$

Finding the gradient requires the following lemma.

Lemma 4.1. *Suppose that a matrix M has the SVD $M = U\Sigma V^\top$, and that its singular values are $\sigma_1 \geq \ldots \geq \sigma_N$. Let C be a cost function as in Equation* (4.18)*, with a lower bound of ρ. The gradient of C with respect to the elements in M is then*

$$\frac{\partial\mathscr{K}}{\partial M} = \sum_{\sigma_n<\rho}(\sigma_n-\rho)\,U_{:,n}V_{:,n}^\top,$$

where $U_{:,n}$ and $V_{:,n}$ are the n^{th} columns of U and V. In matrix form, the gradient is

$$\frac{\partial\mathscr{K}}{\partial M} = U\nu\left(\Sigma-\rho I\right)V^\top, \tag{4.19}$$

where $\nu(\cdot)$ sets the positive elements to zero and leaves the negative ones unchanged:

$$\nu(x) = \begin{cases} x & \text{if } x<0 \\ 0 & \text{if } x\geq 0 \end{cases}.$$

Proof. A proof can found in Appendix B.6. □

To enforce the upper bound $\sigma \leq \tau$, Equation (4.19) is replaced by

$$\frac{\partial\mathscr{K}_{up}}{\partial M} = U\kappa\left(\Sigma-\tau I\right)V^\top$$

where $\kappa(\cdot)$ sets the negative elements to zero and leaves the positive ones unchanged

$$\kappa(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}.$$

The gradient of the cost function with respect to the elements in B can be written as in the following proposition.

Proposition 4.2. *Let the matrix J be a product*

$$J = \Phi'(Ax + Bh + b)B$$

where $\Phi'(Ax+Bh+b)$ is a diagonal matrix with the vector $\phi'(Ax+Bh+b)$ on the diagonal, and B is a square matrix. Let $J = U\Sigma V^\top$ be the SVD of J, and let $\mathscr{K}(J)$ be the cost function of Equation (4.18) *where ρ is a lower bound for the singular values. The derivatives of $\mathscr{K}(J)$ with respect to a single element in A, B, b are*

$$\begin{aligned} \frac{\partial \mathscr{K}}{\partial A_{ij}} &= U_i \nu(\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \frac{\partial J_{in}}{\partial A_{ij}} \\ \frac{\partial \mathscr{K}}{\partial B_{ij}} &= U_i \nu(\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \frac{\partial J_{in}}{\partial B_{ij}} \\ \frac{\partial \mathscr{K}}{\partial b_i} &= U_i \nu(\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \frac{\partial J_{in}}{\partial b_i}, \end{aligned} \tag{4.20}$$

where U_i and V_n are respectively the i^{th} and n^{th} rows of U and V.

Proof. A proof can be found in Appendix B.7 □

The partial derivatives of J are as follows. If the two elements J_{in}, B_{ij} are in different columns ($n \neq j$), then

$$\frac{\partial J_{in}}{\partial B_{ij}} = \frac{\partial}{\partial B_{ij}} \phi'(A_i x + B_i h + b_i) B_{in} = \phi''(A_i x + B_i h + b_i) h_j B_{in}, \tag{4.21}$$

by the chain rule. If the two elements J_{in}, B_{ij} are in the same column ($n = j$), then

$$\begin{aligned}
\frac{\partial J_{ij}}{\partial B_{ij}} &= \frac{\partial}{\partial B_{ij}} \phi'(A_i x + B_i h + b_i) B_{ij} \\
&= \frac{\partial}{\partial B_{ij}} \phi'(A_i x + B_i h + b_i) B_{ij} + \phi'(A_i x + B_i h + b_i) \frac{\partial B_{ij}}{\partial B_{ij}} \\
&= \phi''(A_i x + B_i h + b_i) h_j B_{ij} + \phi'(A_i x + B_i h + b_i), \qquad (4.22)
\end{aligned}$$

which is similar to Equation (4.21) but with an additional term. Equations (4.21) and (4.22) may be combined as

$$\frac{\partial J_{in}}{\partial B_{ij}} = \phi''(A_i x + B_i h + b_i) h_j B_{in} + \delta_{nj} \phi'(A_i x + B_i h + b_i), \qquad (4.23)$$

for $j = 1, \ldots, N$. By the same reasoning,

$$\begin{aligned}
\frac{\partial J_{in}}{\partial A_{ij}} &= \phi''(A_i x + B_i h + b_i) x_j B_{in}, \\
\frac{\partial J_{in}}{\partial b_i} &= \phi''(A_i x + B_i h + b_i) B_{in}.
\end{aligned}$$

The full expressions of the derivative of the cost function are now

$$\begin{aligned}
\frac{\partial C}{\partial A_{ij}} &= U_i \nu (\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \phi''(A_i x + B_i h + b_i) x_j B_{in}, \\
\frac{\partial C}{\partial B_{ij}} &= U_i \nu (\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \left(\phi''(A_i x + B_i h + b_i) h_j B_{in} + \delta_{nj} \phi'(A_i x + B_i h + b_i) \right), \\
\frac{\partial C}{\partial b_i} &= U_i \nu (\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \phi''(A_i x + B_i h + b_i) B_{in}.
\end{aligned}$$

4.4 Setting the RMS of the singular values

A fundamental requirement from the previous sections is that the root mean square of the singular values of any layer-to-layer Jacobian has to be constrained to 1 or just below (see Equation (4.17)).

We will use the following error function to quantify how far the RMS is from a chosen constant h

$$\begin{aligned} f(\sigma_1,\ldots,\sigma_N) &= \left(\sqrt{\frac{1}{N}\sum_{n=1}^{N}\sigma_n^2} - h\right)^2 \\ &= \left(\frac{\|\Sigma\|_F}{\sqrt{N}} - h\right)^2. \end{aligned}$$

The derivative of this error with respect to a singular value is

$$\frac{\partial f}{\partial \sigma_n} = 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}\sigma_n.$$

Now, we recall that the Jacobian for a layer is

$$J(m) = U^{(m)}\Sigma^{(m)}V^{(m)^\top} = \Phi'\left(z(t-m)\right)B,$$

where $z(t-m) = Ax(t-m) + Bh(t-m-1) + b$.

The derivative of the error f with respect to an element in the Jacobian $J(m)$ can be expanded by the chain rule in terms of the singular values

$$\frac{\partial f}{\partial J_{kl}} = \sum_{n=1}^{N}\frac{\partial f}{\partial \sigma_n}\frac{\partial \sigma_n}{\partial J_{kl}} \tag{4.24}$$

where we write J for $J(m)$. By Equation (B.8) in Appendix B.6,

$$\frac{\partial \sigma_n}{\partial J_{kl}} = U_{kn}V_{ln} \tag{4.25}$$

By Equations (4.24) and (4.25), the derivative becomes

$$\begin{aligned} \frac{\partial f}{\partial J_{kl}} &= 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}\sum_{n=1}^{N}\sigma_n U_{kn}V_{ln} \\ &= 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}J_{kl} \end{aligned}$$

We repeat Equation (B.10)

$$\frac{\partial J_{kl}}{\partial B_{ij}} = \delta_{ki}\frac{\partial J_{kl}}{\partial B_{ij}}$$

and combine these to obtain the derivative of the RMS error f with respect to an element in the matrix B

$$\begin{aligned}\frac{\partial f}{\partial B_{ij}} &= \sum_{k=1}^{N}\sum_{l=1}^{N}\frac{\partial f}{\partial J_{kl}}\frac{\partial J_{kl}}{\partial B_{ij}}\\ &= \sum_{k=1}^{N}\delta_{ki}\sum_{l=1}^{N}\frac{\partial f}{\partial J_{kl}}\frac{\partial J_{kl}}{\partial B_{ij}}\\ &= \sum_{l=1}^{N}\frac{\partial f}{\partial J_{il}}\frac{\partial J_{il}}{\partial B_{ij}}.\end{aligned}$$

Now, inserting Equation (4.23) we finally obtain

$$\frac{\partial f}{\partial B_{ij}} = 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}\sum_{l=1}^{N} J_{il}\left(\phi''(A_i x + B_i h + b_i)h_j B_{il} + \delta_{lj}\phi'(A_i x + B_i h + b_i)\right).$$

For the derivatives with respect to elements in A and b, we have that

$$\begin{aligned}\frac{\partial J_{kl}}{\partial A_{ij}} &= \delta_{ki}\Phi''\left(A_i x + B_i h + b_i\right)B_{il}x_j\\ \frac{\partial J_{kl}}{\partial b_i} &= \delta_{ki}\Phi''\left(A_i x + B_i h + b_i\right)B_{il}.\end{aligned}$$

Accordingly, the derivative of the RMS error f with respect to coefficients in A and b are

$$\frac{\partial f}{\partial A_{ij}} = 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}\phi''(A_i x + B_i h + b_i)\sum_{l=1}^{N} J_{il}B_{il}x_j$$

and

$$\frac{\partial f}{\partial b_i} = 2\frac{\|\Sigma\|_F - h\sqrt{N}}{N\|\Sigma\|_F}\phi''(A_i x + B_i h + b_i)\sum_{l=1}^{N} J_{il}B_{il}.$$

4.5 Experimental results

In this section, we propose two different experiments: a so-called *teacher-student*, where we have to train a *student* RNN in order to make it replicate the *teacher* one, and a subsequence detection problem, where the network should give as output an alarm signal when it recognize a specific pattern in the input data.

Teacher-student problem

In order to test our technique, we employ a *teacher-student* experiment. We define a *teacher* RNN such that the lower and upper bounds of the singular values are set to $\rho = 0.9$ and $\tau = 1.1$, respectively. We consider a hidden state of dimension $\texttt{stateDim} = 64$ with $\phi_h = \phi_y = \tanh$ as non-linear functions. Then, using this teacher network we generate 64-length output vectors $y(t)$ for $t = 1, \ldots, T$, with $T = 2000$, starting from random input $x(t)$ of the same length as the output:

$$\begin{cases} h(t) = \tanh\left(A_{\text{teacher}} x(t) + B_{\text{teacher}} h(t-1) + b_{\text{teacher}}\right) \\ y(t) = \tanh\left(C_{\text{teacher}} h(t) + c_{\text{teacher}}\right) \end{cases} .$$

Thus, we can use these pairs for both training and testing sets. As we can see in Figure 4.3, the teacher network we built (blue line) has good properties, since the product of Jacobians does not vanish even after a thousand of time-steps, which correspond to the depth of the unfolded network.

Then, our aim is to train a new network, the *student* one, which resembles the teacher starting from a random initialization of the parameters and using the training data $x(t), y(t)$:

$$\begin{cases} h(t) = \tanh\left(A_{\text{student}} x(t) + B_{\text{student}} h(t-1) + b_{\text{student}}\right) \\ y(t) = \tanh\left(C_{\text{student}} h(t) + c_{\text{student}}\right) \end{cases} .$$

We compare our regularized BPTT algorithm (REG, violet line) with the classical BPTT scheme (red line), minimizing the mean square error (MSE, see Equation (2.3)) as loss function $\mathscr{L}_t$ for every time step t. Analysing the norm of the product of Jacobians, we can see that the network trained using our method follows better the teacher's trend, since the gradient is not exploding and vanishes only after 600 time-steps, while the classical BPTT algorithm produces a network with a gradient that vanishes in less than 100 iterations.

Then, we perform some tests for different number of neurons in the hidden state, *i.e.* we choose $\texttt{stateDim} = \{16, 32, 64, 128\}$. The trends of the norm of the product of Jacobians, which is the reason of the vanishing gradient phenomenon, are shown

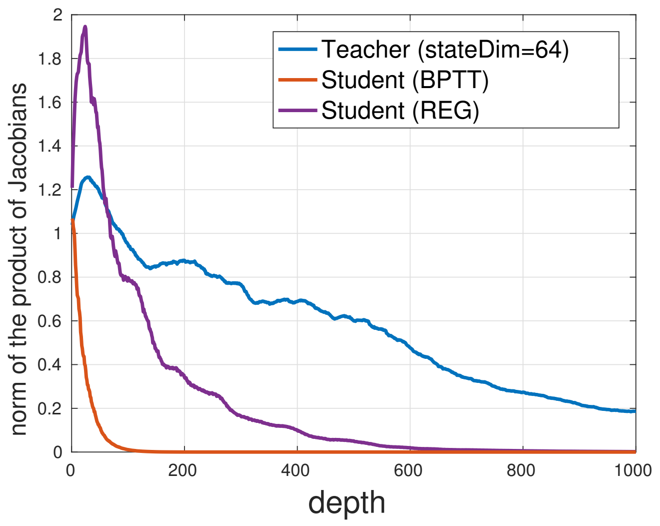


Fig. 4.3 Trend of the norm of the product of the Jacobians $J(m)$ for $\texttt{stateDim} = 64$.

in Figures 4.4-4.6. Our method (REG, violet line) manage to replicate the teacher (blue line) with an higher accuracy than classical BPTT (red line), producing student networks with a memory of 400 iterations for $\texttt{stateDim} = 16, 32$ and even a norm that does not go to zero after a thousand iteration for $\texttt{stateDim} = 128$.

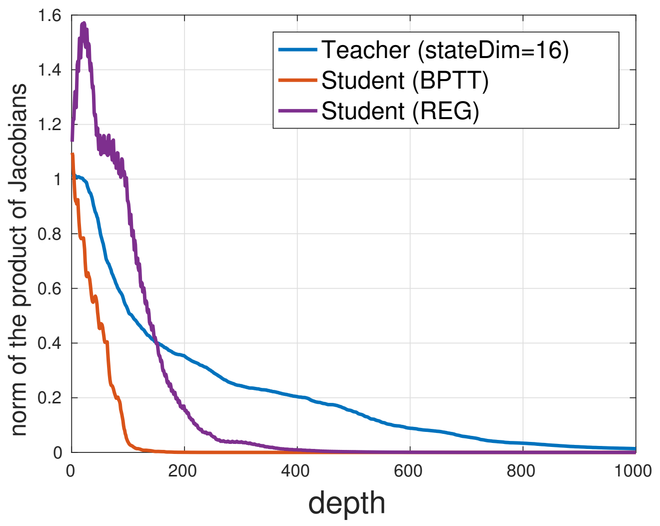


Fig. 4.4 Trend of the norm of the product of the Jacobians $J(m)$ for $\texttt{stateDim} = 16$.

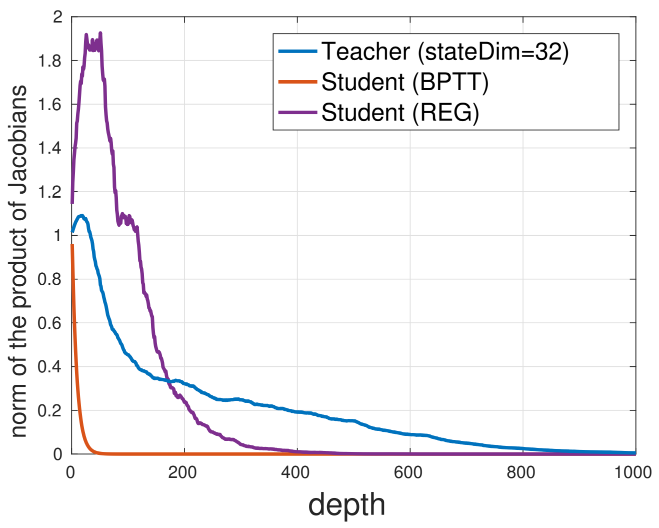


Fig. 4.5 Trend of the norm of the product of the Jacobians $J(m)$ for `stateDim` $= 32$.

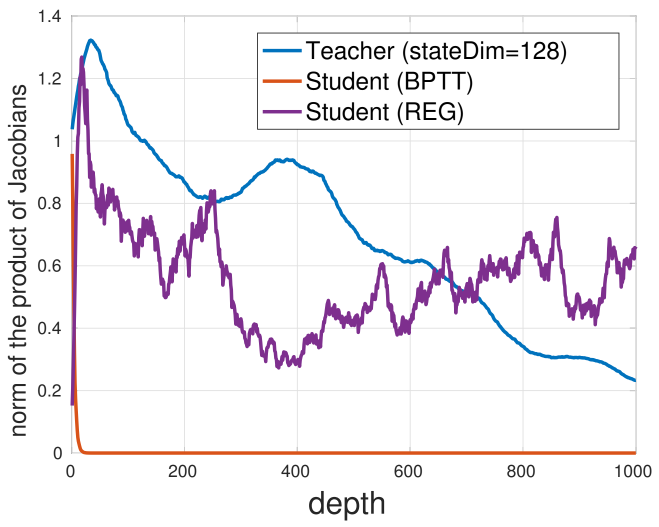


Fig. 4.6 Trend of the norm of the product of the Jacobians $J(m)$ for `stateDim` $= 128$.

Moreover, we report the results in terms of MSE in Table 4.1, where we can notice an improvement of the performance with our new method on both training and testing set for the different values of `stateDim`.

Table 4.1 MSE between the estimated output and the desired one for different dimensions of the hidden state.

	BPTT		REG	
`stateDim`	Training	Testing	Training	Testing
16	0.0899	0.1243	**0.0632**	**0.0956**
32	0.2499	0.3271	**0.0867**	**0.2742**
64	0.7588	1.4484	**0.4568**	**1.1162**
128	1.4519	1.8083	**1.4294**	**1.7226**

Finally, for each value of `stateDim`, we set a teacher RNN and run 50 experiments training student networks, starting from different random initialization of the parameters and using different input/output data. Figure 4.7 and Table 4.2 represent the results with average MSEs and their standard deviations for both REG and BPTT methods on the testing set. As we can notice, for every value of `stateDim`, our REG method trains better student networks, since the average MSEs are lower and also their variation, computed in terms of standard deviation, are less relevant.

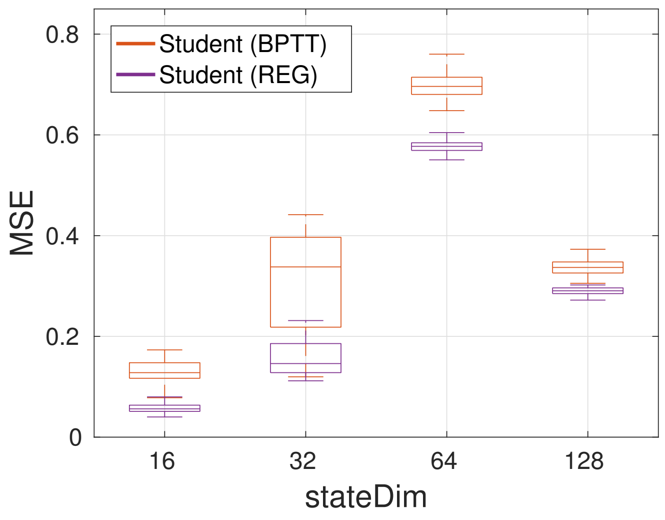


Fig. 4.7 Box-plot of MSEs for 50 experiments on the testing set.

Table 4.2 Mean and standard deviation of the MSE between the estimated output and the desired one on the testing set over 50 experiments for different dimensions of the hidden state.

	BPTT		REG	
`stateDim`	Mean	Std	Mean	Std
16	0.1319	0.0297	**0.0596**	**0.0143**
32	0.3237	0.1819	**0.2665**	**0.1302**
64	0.7008	0.0333	**0.5776**	**0.0116**
128	0.3382	0.0166	**0.2903**	**0.0074**

Thus, it is useful to regularize the cost function bounding the singular values, to a narrow interval and with RMS equal (or really close) to 1, since in this way it is possible to extend the memory of the network, which can remember a large number of previous time steps.

Subsequence detection problem

We propose another experiment in order to test the memory of a network trained with our regularization technique. We want the network to detect a specific behaviour of an input signal and to give as output an alarm signal that switches its components from 0 to 1 after the detected sequence. In order to generate these data, we build a unidimensional input signal of length $T = 2e4$ and a 10-element-length subsequence, extracting their components randomly from a Gaussian distribution with zero mean and variance equal to one. Then, we insert the subsequence in the input many times randomly with an average interval of `meanSpacing`$= 40$. Finally, the output is a zero signal, except for the 10 components after the subsequence, which are equal to one, meaning that the subsequence has been detected and an alarm should be sent. A graphical representation of input and output signals is provided in Figure 4.8, where we can notice that the output (*i.e* the second row) goes from 0 to 1 after the repeated subsequence.

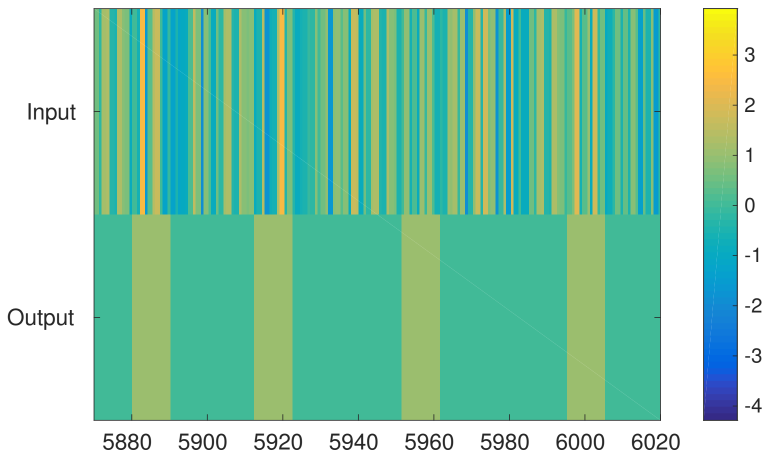


Fig. 4.8 Graphical representation of input (first row) and output (second row) signals.

We compare our regularization (REG), imposing the singular values to be bounded in $[0.9, 1.1]$ and their RMS to be close to one, to a simple RNN and an LSTM, choosing an hidden state of 20 neurons and a one-hot representation of the output in order to use a LogSoftMax output function and an NLL minimization criterion. Moreover, we use the RMSprop (2.31) optimization scheme to update the parameters for 50 epochs.

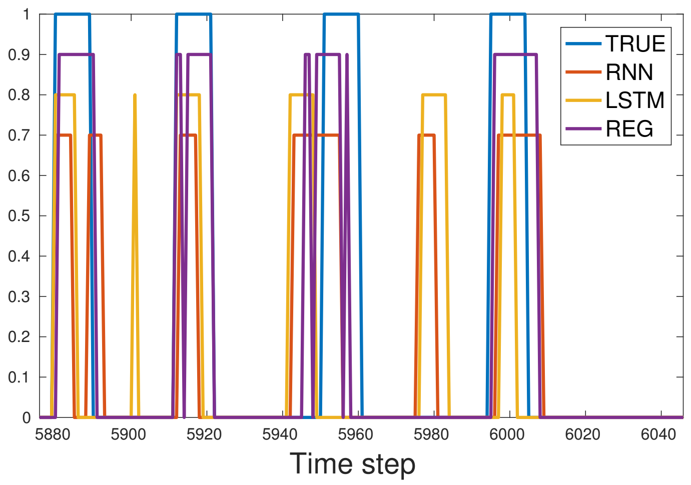


Fig. 4.9 Comparison of alarm signals: the expected output in blue, a simple RNN in red (multiplied by 0.7), an LSTM in yellow (multiplied by 0.8), and an RNN trained with our regularization (REG, multiplied by 0.9).

In Figure 4.9 we represent the output alarm signals for the three methods (RNN in red, LSTM in yellow, and REG in violet), which have been multiplied by 0.7, 0.8, and 0.9 factors respectively in order to help the visualization, and the expected one (blue line) with `meanSpacing`$= 40$. As we can see, the REG matches better the true output in correspondence of the alarm, providing also the right duration of the alarm, with an NLL error of 0.1851 for the training set and 0.2186 for the testing set, while for RNN (which is the worst one) and for LSTM we have 0.3645, 0.3647 and 0.3615, 0.3625, respectively. We also compute precision and recall values, as the sum of the true positives detected over the number of retrieved alarm (*i.e.* the sum of true positive and false positives) and over the number of true alarms (*i.e.* the sum of true positives and false negatives): we obtain a precision and a recall of 0.7755 and 0.7748, respectively, for our regularized network, while 0.4852 and 0.4849 for the simple RNN and 0.5448 and 0.5442 for the LSTM.

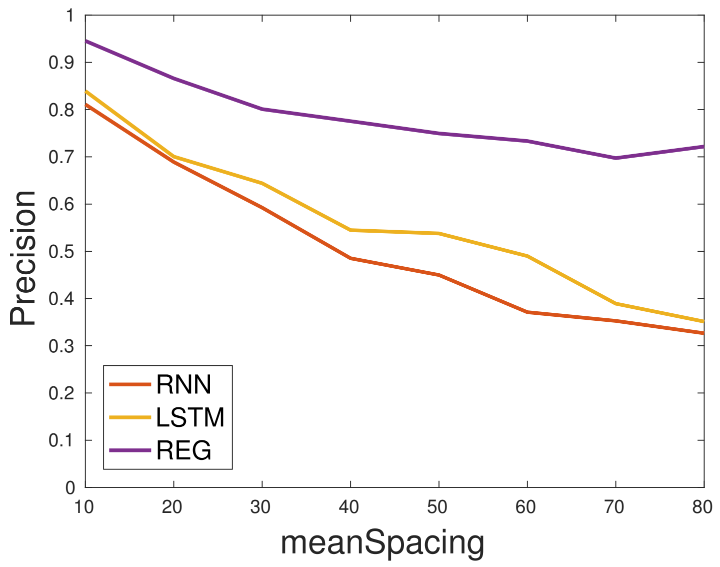


Fig. 4.10 Precision of the compared methods for different values of `meanSpacing`.

Finally, we vary the average interval between two consecutive subsequences `meanSpacing`$= \{10,20,30,40,50,60,70,80\}$. As we can see in Figure 4.10, where we represent the precision of the three methods, and in Table 4.3, where we reported all the values of precision and recall for different values of `meanSpacing`, our technique outperforms both the simple RNN and the LSTM architecture.

Table 4.3 Comparison of precision and recall for different values of `meanSpacing`.

	RNN		LSTM		REG	
`meanSpacing`	Precision	Recall	Precision	Recall	Precision	Recall
10	0.8110	0.8108	0.8393	0.8390	**0.9455**	**0.9453**
20	0.6890	0.6888	0.7005	0.7003	**0.8660**	**0.8657**
30	0.5924	0.5919	0.6440	0.6435	**0.8010**	**0.8004**
40	0.4852	0.4849	0.5448	0.5442	**0.7755**	**0.7748**
50	0.4498	0.4493	0.5378	0.5375	**0.7495**	**0.7491**
60	0.3711	0.3706	0.4899	0.4895	**0.7334**	**0.7329**
70	0.3527	0.3524	0.3892	0.3889	**0.6973**	**0.6967**
80	0.3265	0.3259	0.3512	0.3509	**0.7218**	**0.7211**

4.6 Discussion

Summing up, in this chapter we developed a new method to extend the memory of a simple RNNs, avoiding the so-called vanishing gradient problem.

We addressed this problem from a theoretical point of view and thanks to our study we found conditions not to make the gradient going fast to zero when increasing the depth (or the number of time steps) of the network. We exploit the random matrices theory and a singular values analysis, imposing constraints on the vanishing factors in the gradients. We showed that if the singular values of the product of Jacobians are bounded in a narrow interval, then independence hypothesis of the inner products is satisfied.

Moreover, we also showed that if the RMS of the singular values is equal to one, the upper bound to the norm of the gradient is independent on the depth of the network, which implies that the gradient does not vanish. Practically, it is not feasible to constrain the RMS of the singular values exactly to one, so we require it to be *close* to one, which means that the gradient will go to zero, but only after a large number of time-steps.

In the near future, we will carry on this study on the vanishing gradient and we will apply our conditions to solve problems, which require a lot of memory (such as the subsequence detection explained in previous section) and thus they are hard to be solved with a simple RNN.

Chapter 5

Speed-up in the Training using Least Squares Initialization

Brief - *In this chapter, we introduce a new method to modify the parameters of a recurrent neural network. We develop an initialization of the parameters based on a least square regularization, in order to start the training closer to the solution and, therefore, to accelerate the training process, running few epochs of classical training algorithms. Moreover, we notice that our method gets really close to the solution, reaching almost the same performance as classical training algorithms, and thus it can be used a fast training algorithm itself. Finally, thanks to its speed, it can be used also as a real-time training algorithm, updating the parameters every time new data are available.*

Deep neural networks can represent in a compact way non-linear and highly varying functions. Until around 2007, it was not clear how to train such deep networks, since gradient-based optimization starting from random initialization appeared to often get stuck in poor solutions, see Bengio *et al.* [68] and Bengio *et al.* [69]. Then, a novel training strategy was introduced by Hinton *et al.* [70], exploiting the pre-training of one layer at a time in a greedy way. This algorithm helps the optimization, if the weights are initialized in a region near a good local minimum, see Bengio *et al.* [71].

Thus, starting from this idea, we propose a new method to accelerate the training of RNNs, which can be represented as very deep networks, initializing the parameters according to a least square approximation, similar to the one used for training multi-layer neural networks as a cascading structure, see Li and Rad [72], or multi-layer perceptrons in blocks, as presented by Navia-Vasquez and Figueiras-Vidal [73].

This chapter shows how an adaptation of layer-wise least squares training to recurrent networks may speed up training and even improve the performance of the resulting networks.

5.1 Initialization of the Network's Parameters

We start considering the general form of a simple RNN (2.1):

$$\begin{cases} h(t) = \phi_h\Big(Ax(t) + Bh(t-1) + b\Big) \\ y(t) = \phi_y\Big(Ch(t) + c\Big) \end{cases} .$$

Since during the training phase the input/output pairs $x(t), y(t)$ are available, using a initial random configuration of the parameters A, B, b, C, c, we can compute the hidden state $h(t)$ normally, using the first equation of (2.1), or proceeding backward inverting the second one. We will refer to $h_{\text{forw}}(t)$ and $h_{\text{back}}(t)$, respectively, as follows:

$$\begin{cases} h_{\text{forw}}(t) = \phi_h\Big(Ax(t) + Bh(t-1) + b\Big) \\ h_{\text{back}}(t) = C^{\dagger}\Big(\phi_y^{-1}(y(t)) - c\Big) \end{cases} , \qquad \forall t = 1, \dots, T \tag{5.1}$$

where † represents the pseudo-inverse of a matrix and ϕ_y^{-1} is the inverse of the non-linear output function.

Let us start with the first equation of (2.1). We can write it in matrix form considering the time-steps all together and inverting the hidden non-linear function ϕ_h:

$$\begin{bmatrix} A & B & b \end{bmatrix} \begin{bmatrix} x(1) & x(2) & \dots & x(T) \\ h(0) & h(1) & \dots & h(T-1) \\ 1 & 1 & \dots & 1 \end{bmatrix} = \phi_h^{-1}\begin{bmatrix} h(1) & h(2) & \dots & h(T) \end{bmatrix} . \tag{5.2}$$

Then, we compute the transpose of (5.2)

$$\begin{bmatrix} x(1) & x(2) & \dots & x(T) \\ h(0) & h(1) & \dots & h(T-1) \\ 1 & 1 & \dots & 1 \end{bmatrix}^\top \begin{bmatrix} A^\top \\ B^\top \\ b^\top \end{bmatrix} = \phi_h^{-1} \begin{bmatrix} h(1)^\top \\ h(2)^\top \\ \vdots \\ h(T)^\top \end{bmatrix}, \tag{5.3}$$

recalling that the transpose of the product of two matrices M_1, M_2 is

$$(M_1 M_2)^\top = M_2^\top M_1^\top .$$

Now, we can solve the linear system (5.3) through a least square method to find the estimated parameters $\hat{A}, \hat{B}, \hat{b}$, using, instead of h, its backward definition h_{back} from (5.1), and finally update the parameters of the recurrent layer as

$$\begin{cases} A = A - \lambda(A - \hat{A}) \\ B = B - \lambda(B - \hat{B}) \\ b = b - \lambda(b - \hat{b}) \end{cases}, \tag{5.4}$$

where λ is a step parameter, which plays the role of the learning rate.

Similarly, we can write the equation of the output layer (*i.e.* the second one of (2.1)) in matrix form considering all time-steps together and inverting the output non-linear function ϕ_y:

$$\begin{bmatrix} C & c \end{bmatrix} \begin{bmatrix} h(1) & h(2) & \dots & h(T) \\ 1 & 1 & \dots & 1 \end{bmatrix} = \phi_y^{-1} \begin{bmatrix} y(1) & y(2) & \dots & y(T) \end{bmatrix}. \tag{5.5}$$

Then, we compute the transpose of (5.5)

$$\begin{bmatrix} h(1)^\top & 1 \\ h(2)^\top & 1 \\ \vdots & \\ h(T)^\top & 1 \end{bmatrix} \begin{bmatrix} C^\top \\ c^\top \end{bmatrix} = \phi_y^{-1} \begin{bmatrix} y(1)^\top \\ y(2)^\top \\ \vdots \\ y(T)^\top \end{bmatrix}, \tag{5.6}$$

obtaining a linear system with C, c unknowns. We can solve (5.6), using h_{forw} from the first equation of (5.1), in order to obtain the estimated parameters $\hat{C}$ and $\hat{c}$, and update also the parameters of the output layer as

$$\begin{cases} C = C - \lambda(C - \hat{C}) \\ c = c - \lambda(c - \hat{c}) \end{cases} . \tag{5.7}$$

Thus, we can sum up our initialization method following this iterative procedure:

1. using the current configuration of parameters, compute $h_{\text{forw}}(t)$ and $h_{\text{back}}(t)$, for every time-step t, according to (5.1);
2. solve the linear systems (5.3) and (5.6), in order to obtain the estimated parameters $\hat{A}, \hat{B}, \hat{b}$ and $\hat{C}, \hat{c}$, respectively;
3. update the parameters of recurrent and output layers, as in (5.4) and (5.7), respectively;
4. repeat this procedure from point 1. using the updated parameters.

5.2 Advantages in using our Initialization

The initialization technique presented here may be advantageous in three aspects: before training, during training and after training.

It can be employed as a fast method to reach a configuration of parameters closer to the best one before running classical training algorithm, such as BPTT. Moreover, by running more iterations of our algorithm it is possible to obtain a performance which is comparable to the one reached by BPTT. Finally, it can be also used during the testing phase, exploiting its speed to adapt the parameters to the latest data, exactly as a real-time learning algorithm.

In order to test our method, we set up a teacher-student experiment, where the teacher network is defined with all the parameters extracted from a Normal distribution and an hidden state with 128 neurons. Then, we generate a random input sequence $x(t)$ of dimension 64, for $t = \{1, \dots, T = 2000\}$, and compute the corresponding output $y(t)$ driving the input signal through the network with non-linear functions $\phi_h(\cdot) = \phi_y(\cdot) = \tanh(\cdot)$. The student network has to resemble the teacher one, starting from another random initialization of the parameters and using the generated input/output pairs during the training.

The first result is shown in Figure 5.1.

Running 7 iterations of our initialization (violet line), we can provide an initial configuration of parameters, leading to a solution that is closer to the expected output (blue line) than starting from a random initialization of the parameters (red line). Moreover, running 10 epochs of BPTT, the solution obtained starting with the initialized parameters (green line) gets almost overlapped to the expected solution, while the one obtained with a random initialization (yellow line) is even worse than initialization itself.

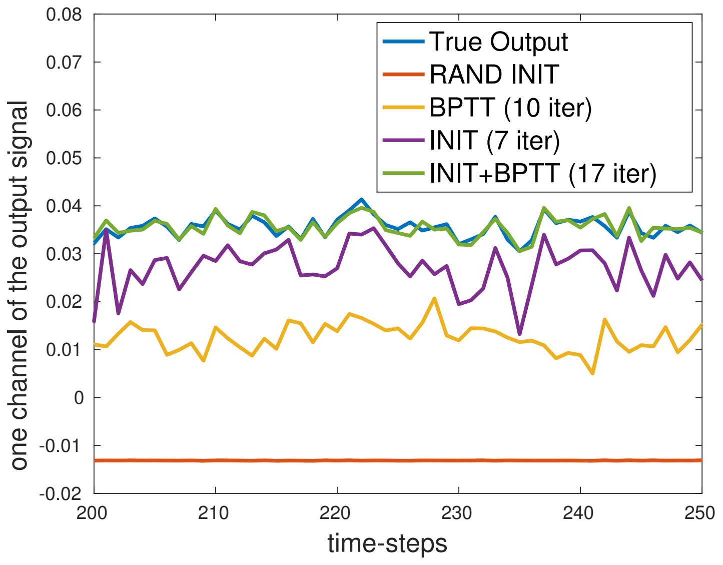


Fig. 5.1 Estimated output using a random initialization or our least square method, followed by BPTT: starting from a good initialization of the parameters, it is possible to reach a more accurate solution.

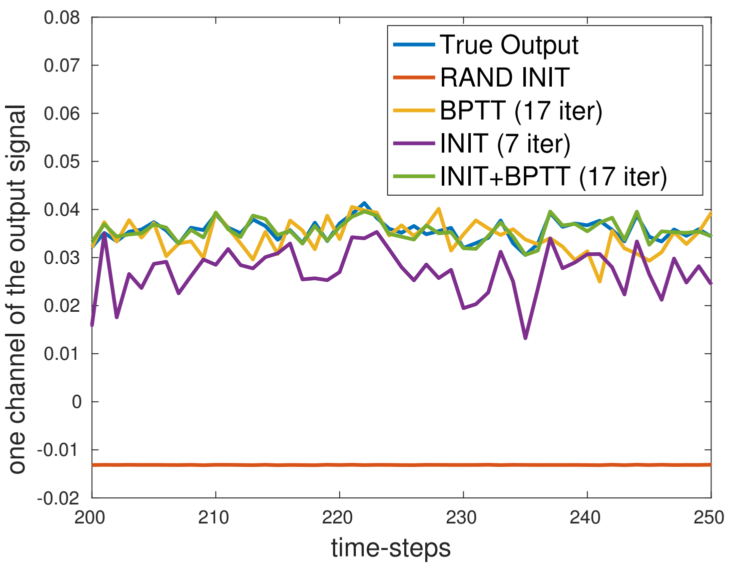


Fig. 5.2 Estimated output running more epochs of BPTT.

However, it is possible to obtain a good solution also applying BPTT scheme to a random parameters initialization. As a matter of fact, Figure 5.2 shows that running BPTT for 17 epochs, which is the sum of the 7 iterations of initialization and the 10 epochs of BPTT, the estimated output (yellow line) is close to expected one (blue line).

Anyway, this helps us thinking to increase the number of iterations for our initialization method, in order to use it as a full training algorithm. The results reported in Figure 5.3 are very promising, since if we run only 10 iterations of our scheme, the estimated output (violet line) really approaches the exact one.

Finally, we represented the results in Figures 5.1-5.3 for one channel of the 64-length output signal, in order to be able to represent it as a 2D graph. Nevertheless, the same trend can be noticed for all the other channels, and to provide a complete analysis we collect the MSEs in Table 5.1, where we compare our initialization (INIT) followed or not by some epochs of BPTT and classical BPTT with a random initialization of parameters, both on the training and the testing set.

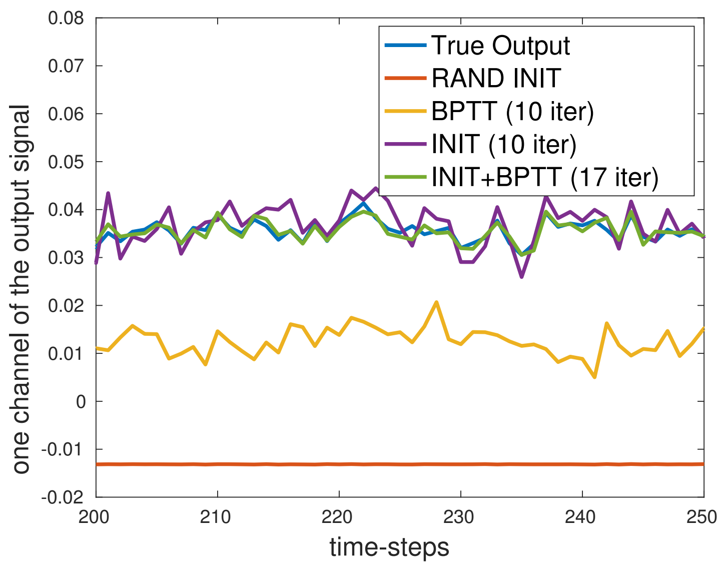


Fig. 5.3 Estimated output running more iterations of the initialization technique: this result shows that it can be employed as a training algorithm itself.

Table 5.1 Comparison of MSEs in a teacher-student experiment.

	Training Set	Testing Set
BPTT (10 iter)	22.5125e-3	23.5832e-3
INIT (7 iter)	9.0098e-3	9.1434e-3
INIT + BPTT (17 iter)	**0.9934e-3**	**1.0027e-3**
BPTT (17 iter)	4.0690e-3	4.5098e-3
INIT (10 iter)	3.3425e-3	3.4286e-3

Thus, classical training algorithms such as BPTT can benefit a lot from our least square initialization.

5.3 Update of the Parameters in Real-Time Training

Another key feature of our initialization technique is the efficiency in terms of time speed. As a matter of fact, our method (INIT) is more than 20 times faster than classical BPTT, for the dimensions considered in the teacher-student experiment of previous section. The time elapsed[1] to run one epoch (or iteration) is reported in Table 5.2, which means that for 10 iteration of our method (violet line in Figure 5.3) we need 6.5 seconds, while for 17 epochs of BPTT starting from a random initialization (yellow line in Figure 5.2) we have to wait almost 4 minutes, and the accuracy is better with our method as well (see Table 5.1).

Table 5.2 Comparison between BPTT and our initialization in terms of time speed.

	Time Elapsed per Iteration [s/iter]
BPTT	13.97
INIT	**0.65**

This suggests us that our method can give results almost in real-time. In particular, there are many problems where it is necessary to provide information as soon as possible. Moreover, real-time situations are usually more influenced by latest time-steps. Thus, the long-time dependence is not so relevant, while the latest time-steps are the most significant. Thus, in these situations, we can apply our initialization method as a real-time algorithm, which modifies the parameters as soon as new data are available running a couple of iterations in few seconds. This technique provides a model that fits better the latest data, adapting the parameters during the testing phase. We will show that this can be effectively useful in Chapter 7.2, where we will apply our initialization after the classical BPTT scheme, in order to predict the flow of distribution of mobile users in a region during days.

[1]The execution time is computed with MATLAB R2016a, on a HP Z230 Tower Workstation with Intel Xeon processor E3-1245 v3, 4 cores, 3.4 GHz and 32 GB of RAM.

Chapter 6

Prediction of Numerical Sequences with Recurrent Neural Networks

Brief - *In this chapter, we test recurrent neural networks performing non-linear prediction of numerical sequences. Given the beginning values of the sequence as inputs, then the network is left to continue the sequence using as input the output predicted at previous time-step. We apply this to the generation of written text, where every character is associated to a number, noticing that the network is able to learn the style of a specific author. Moreover, we also predict the paths of mobile users moving in a city center as a sequence of numbers corresponding to road intersections crossed by the walking users, which brings several applications in the smart cities framework.*

With Chapter 6, we begin the second part of this thesis, dedicated to the application of RNNs to non-linear prediction problems.

Since a RNN is a generative model, we expect it can be successfully used for prediction. We propose different kind of RNN, opportunely tailored to the specific problem, finding the best alternative architecture. In Section 6.1, we train a network implementing a character-level generation of written text comparing a simple RNN, an LSTM and our regularized RNN using the method presented in Chapter 4. On the other hand, in Section 6.2, we predict the movement of mobile users as a sequence of crossroads, which is a problem that does not require a lot of memory, but needs to communicate quickly the information to the user. Therefore, for this specific problem, we prefer a faster RNN rather than an LSTM.

Then, in Chapter 7, we extend the prediction of numerical sequences to higher dimensions, *i.e.* video frames sequences. In Section 7.1 we train the network to learn the physical laws of colliding balls, comparing classical RNN and LSTM structure to our regularization approach of Chapter 4 to make the gradient not to vanish and extend the memory of the network. This method turns to be very useful to improve the prediction of the movement of the bouncing balls. Then, in Section 7.2, we want the network to predict the distribution of mobile user over a region hourly. In this situation, the latest data are the more relevant than the past ones. Therefore, we do not need to extend the memory of the network and, on the contrary, we need to communicate the prediction as soon as possible. Thus, we apply our fast least square initialization of Chapter 5 and use it as an on-line adaptative method, obtaining impressive results.

Finally, in Chapter 8, we design both RNN and LSTM for restoring audio signal with incomplete spectrograms, which is a well-known problem in literature. We demonstrate via numerical experiment that a performance improvement can be achieved using neural networks with respect to the state-of-the-art methods.

6.1 Text Generation

In order to perform prediction of characters in the generation of written text, we map the list of all possible characters, *e.g.* the whole alphabet in capitol and small letters, all the numbers and many other symbols (including the blank and the new line), into a vector of indices, where each element represent a character. Then, we use a *one-hot* representation [74] of the input text, building a matrix as follows: each column, representing a letter of the text sequence, is filled with zeros, except for the position corresponding to the current character, where we have 1. For example, as shown in Figure 6.1, if our alphabet consists of `[h,e,l,o]`, a one-hot representation of the word `hello` is $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$.

Then, if we want our recurrent network to write `hello`, we have to split it into input x and output y sequences. Since we want the output at time t to be the following

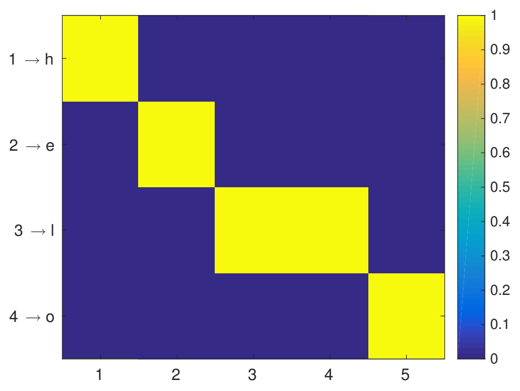


Fig. 6.1 Visual matrix form of `hello` using the one-hot representation.

input, *i.e.* at time $t+1$, we set

$$\begin{cases} x = [\texttt{h},\ \texttt{e},\ \texttt{l},\ \texttt{l}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \\ y = [\texttt{e},\ \texttt{l},\ \texttt{l},\ \texttt{o}] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{cases} .$$

We choose three neurons in the hidden state, due to the small dimensionality of this problem, and train a small network, where the non-linear function of the output layer ϕ_y is the LogSoftMax function, using the NLL criterion (2.6). Since we want to find a probability distribution function for the output, which tells us the following most probable character, we compute the exponential of the output, obtaining the probability distribution shown in Figure 6.2. Taking the maximum, we obtain $\hat{y} = [\texttt{e}, \texttt{l}, \texttt{l}, \texttt{o}] = y$, which means that our RNN has learnt to write the word `hello`.

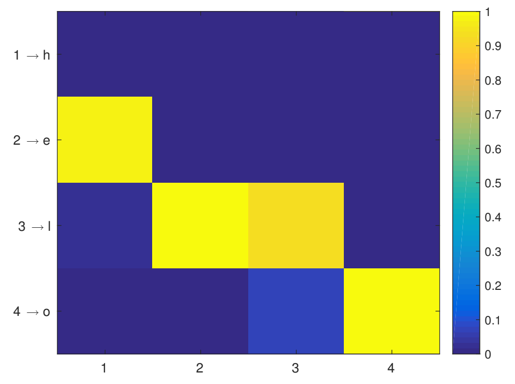


Fig. 6.2 Probability distribution for the output of a network generating the word `hello`.

We obtain the probability distribution represented in Figure 6.2 training a RNN with the regularization conditions of Chapter 4 (REG): we bound the singular values in the interval $[0.9, 1.1]$ and impose the constraint that the RMS of the singular values is close to 1. We also choose Adagrad optimization method (2.29) for the update step.

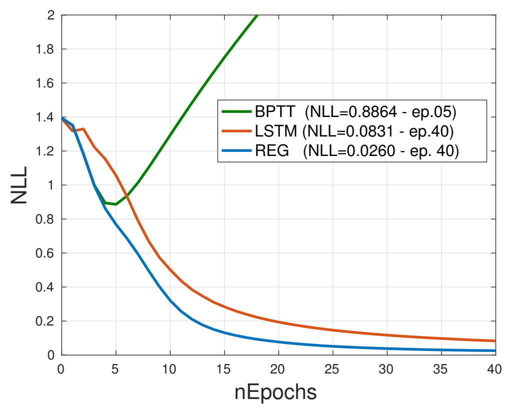


Fig. 6.3 NLL errors generating the word `hello`.

Thanks to our conditions, as shown in Figure 6.3, we outperform both the classical BPTT on a simple RNN and the LSTM architecture. At the end of the training, after running 40 epochs, we find an NLL error of 0.026 (blue line), which is slightly better not only than the best performance obtained by BPTT at epoch 5 (green line), *i.e.* NLL=0.8864, because then it explodes (NLL=3.42 after the last epoch) and does not arrive to convergence, but also than LSTM (red line), which arrives to NLL=0.0831, and it is always above (higher error) our method. On the other hand, the classical BPTT algorithm applied to a simple RNN produces the word `hllll`, which means that the network has learnt only that `l` is the most frequent letter and thus it repeats always that one.

This was only a simple example of how to generate text, as sequence of characters using an RNN. The same principle can be applied widening the number of symbols used in the alphabet. Here, we present the results of a bigger training performed on a collection of Sir William Shakespeare plays. We train a network using input/output vectors with 65 different characters, for a total of $T = 1e6$ characters, and an hidden state of 128 neurons. Then, we let the network generate its own sequence:

ROMEO:
To dew I leave thee can thou shall little man
Is sore-pairs of own be prevail a quamiofh
Crunk of and then must do everfillly you;
And fore your grace.
Kills you were in beither so thee,
Bray to bears this crown that five married.
O, Green, that's no lesterally,–

WARWICK:
Do youth; I have spoke his valeity,
Whom it swillget plinks of all heably
Direct my destail to victory.

GREMIO:
In a mother, nurse, have you bud, here in
the men, and hardly wind more frial'd eyes,
And leave him? Now she perspace.

KING EDWARD IV:
Roun your mistress
Interdails their save and honests to be.

Of course the neural network produces a meaningless play, but it is amazing to notice that it has learnt some words typical of Shakespeare's ages, such as "thee" or 'thou", and also the structure of a Shakespearian play, with the name of the character (all in capitol letters) before its speech. We want to highlight also the fact that many names are really names of characters of Shakespeare's works, which successively alternate in the performance, creating dialogues between, for example, Warwick and Gremio, which appear in "Henry VI" and "The Taming of the Shrew", respectively. Moreover, even if some generated words does not exist, at least it is possible to define them as "plausible", and also their length is not too long, which means that the network has learnt where to insert properly a blank space.

Finally, using the same settings as in the previous example, we also "gave the network to read" the cookbook by the famous Pellegrino Artusi, who wrote *La scienza in cucina e l'arte di mangiar bene* ("The Science of Cooking and the Art of Fine Dining"), and asked our network to produce a recipe on its own. The result is impressive:

> 750. Crostate per ridotto staccarsi.
> Farina d'Ungheria, grammi 300.
> Burro, grammi 40.
> Parmigiano grattato, grammi 30.
> Parmigiano, cucchiaiate n. 5.
> Peperato tra convitella di vainiglia.
> L'ora.
> Prendete o tritate il sale al composto una pobe può bastire per dieci moditi. Tuco bianco come versatelo in Italia in genemmo, un po' di sugo sufficientelo nel burro in padella della vedrete tratti sul granelmo a questo.
> Cucchiaiate dal fuoco con qualche cucchiaiata, due cucchiaiate di reso cerdo con sale e pepe, dopo ossersto, perché sarabbario.
> Farina di farma di magro grossi, grammi 30.
> Prosciutto grasso e uno spicchio d'aglio amalito. La trigle, nettate unite il tagiame onde non sono dirante lunghe rimuscono e per compresa di un piattoccino 'osprazione

This recipe has a sort of title at the beginning, then the list of the ingredients and finally the description of the preparation process, and a part from some meaningless

words, in which anyway can notice an Italian-style sequence of vowels and consonants, there are many right ones, even very long such as “cucchiaiate” or groups of words that are used together, such as “Farina d’Ungheria”, “Parmigiano grattato”, or “spicchio d’aglio”.
Finally, just for fun, we propose another recipe if an interested reader wants to try it:

> Pezzettini di filettina n. 31.
> Principii. Sformato pinetta col brodo.
> Un pizzico di farina, grammi 200.
> Dette, foglie di pollo. Fate bollire vannia colla lunetta sulla carne mezza e pasta di latta in cui neranti e dopo col mestolo ove fare la due persone.
> Contonuta fanno famallo fatelo calde.
> 387. Pavoli frutta ci’nosca, grammi 500.
> Latte, spacciate in mandorle, l’olio, grammi 40.
> Panna di zucchino, grammi 45.
> Uova, n. 1.
> Diecime, cucchiaiate e mettete il riso e intinto allunga comune passato in una spino per farà il cucino con la Gulattera da marsa per tantella mezza noccuccuo restano aggiandoli, fatelo soltanto, asciatto e alquanto più succo la loro bianco, il coltello e fategli una teglia e sciogliendo la conserva e quanto basta.

6.2 Path Prediction of Mobile Users

Another interesting problem that can be addressed making predictions with RNNs is the study of the paths covered by walking mobile users. These data were provided us by the Joint Open Lab SWARM of TIM. In order to preserve the privacy of the users, TIM refers to the users with an ID, which changes every day, but for our purpose it is not important to know who covered a specific path. These data were given us as a sequence of indices, each of them corresponding to a crossing in the center of Turin (we will refer to them as *nodes*). We have a total of 531 possible crossings, plus one that refers to the end of a path and the beginning of a new one, and so we build the input/output vectors using a one-hot representation. Again, we set the number of hidden neurons to 128 and choose the LogSoftMax as output non-linearity, coupled with a NLL criterion to minimize the cost function.

After using these data to train the network, we perform some tests giving the initial nodes to the network and then letting the network predict the next ones. We represented the results on the map of Turin center: we compare Figures 6.4 and 6.5, where we show the actually covered paths and the ones predicted by our RNN.

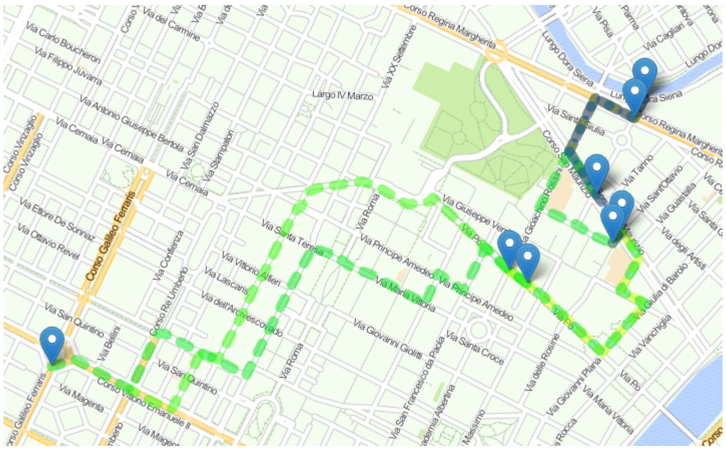


Fig. 6.4 Ground truth of the test paths.

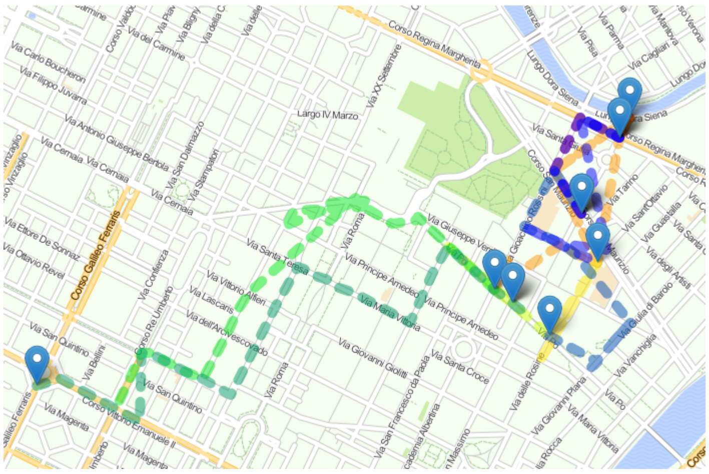


Fig. 6.5 Predicted paths with RNN.

As we can see, the paths in the two figures can be mainly overlapped, except of a couple of nodes that the RNN failed to predict, such as the one in *via San Quintino* for the darkgreen path or the one in *via Santa Teresa* for the lightgreen one, but even in this case the network recovers the true path after the wrong node.

Since when the RNN fails to predict the right node it is not a node near to the previous one, we decided to "help" the RNN giving the adjacency matrix of the graph built on the map, so that the network has to choose only between the near crossings. This increases the performance, and we report the results obtained exploiting the adjacency matrix in Table 6.1, where we also study how much the number of initial nodes affects the results. As we can notice, with few initial nodes the network hesitates and repeats some node, but then it finds the right path, which is only shifted in time, while with 6 or more nodes the path is exactly the same as the ground truth one. On the other hand, if we use the adjacency matrix to choose randomly the following node, but avoiding loops in the graph, we obtain a path that initially can look like the true one, but then it diverges on wrong roads.

Table 6.1 Path prediction varying the number of nodes used as initial seed.

Time step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ground Truth	509	526	525	524	527	4	73	76	77	86	87	93	94	170	215	217	223	224	229
2 nodes for seed	509	526	509	526	527	526	527	45	73	76	77	86	87	93	94	170	215	217	223
3 nodes for seed	509	526	525	526	527	526	527	45	73	76	77	86	87	93	94	170	215	217	223
4 nodes for seed	509	526	525	524	527	45	73	76	77	86	87	93	94	170	215	217	223	224	229
5 nodes for seed	509	526	525	524	527	45	73	76	77	86	87	93	94	170	215	217	223	224	229
6 nodes for seed	509	526	525	524	527	4	73	76	77	86	87	93	94	170	215	217	223	224	229
Random choice	509	526	42	524	527	45	458	46	44	43	32	38	48	50	51	54	56	80	65

This kind of problem is important because it has a large variety of applications, especially in a smart-city framework. As a matter of fact, it is useful to foresee the path of a mobile user in order to suggest or ask him/her to change the way in case of necessity. For example, if some garbage cans or street lights are not covered by any wireless connection and may need to communicate to their corresponding operations center that they are stuffed with rubbish or a breakdown, we can ask the user to go on a specific street, close to the one he/she is, in order to let the garbage cans or street lights to use the mobile connection in exchange of some special offer for the mobile user. Moreover, it can be possible to advise the user to move to the next street if there is some kind of accident, such as traffic jam or men at work.

Thus, an important aspect of the prediction is to communicate promptly with the users, which implies the use of faster RNN rather than the LSTM architecture, and moreover we do not need a large memory of previous time-steps to predict the following one, so we used a simple RNN for the prediction of mobile users' paths.

Chapter 7

Prediction of Video Sequences using Recurrent Neural Networks

Brief - *In this chapter, we present how to predict video frames with recurrent neural networks. We show performance on three datasets. We start from the literature bouncing balls problem, where the network has to learn the physical laws of colliding balls, without having knowledge about the physical characteristics of problem such as the velocity and the direction of the moving balls. On the other hand, the network is able to predict correctly the movement of the balls only exploiting the visual information from the video frames. Then, we train our network on bigger scale problems: thanks to the dataset provided by TIM, we study the distribution of mobile phones in the city of Turin and over a wider area, such as the whole Piedmont region, which is a problem with many applications in different fields.*

7.1 Bouncing Balls Problem

Since in previous chapter we explained how to predict numerical sequences with RNNs, here we introduce the natural extension to two-dimensional sequences prediction, corresponding to video predictions, where each frame is an image and thus it can be represented as a matrix.

Since RNNs can be associated to dynamical systems, they are suitable to face physical problems. One problem faced in literature is the bouncing balls problem [75, 76], where a RNN is trained to learn the physical laws of colliding balls.
We generate a 18×18 video with two or three balls, whose position is defined by the coordinates of the centres. The balls bounce among each other and off the walls,

considering their radius and their velocity. Then, we give as input to the RNN each video frame, which can be represented as a matrix with ones, where there are the balls, and zeros elsewhere. The network has to predict the following frame, so, exactly as in the text generation problem, the output at time t is the input at time $t+1$, *i.e.* $x(t+1) = y(t)$, as shown in Figure 7.1.

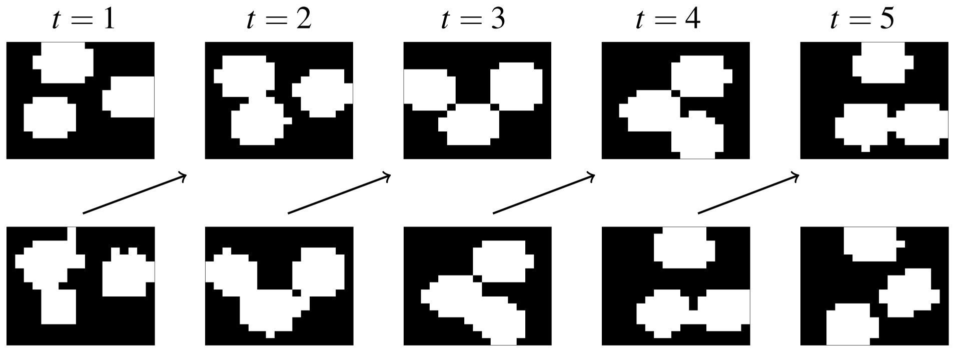


Fig. 7.1 A three bouncing balls sequence: input sequence in the first line, prediction of the following input frame with RNN in the second one.

Unlike the generation of text, where we need a probability distribution as output in order to predict the most probable character, here we need the image representing the balls position at each time-step. For this reason, we change the output non-linear function ϕ_y to the logistic sigmoid function

$$\phi_y(z) = \frac{1}{1+\exp^{-z}},$$

which bounds the values of z between 0 and 1, so we have to apply a threshold to obtain the next $0/1$ input matrix. Therefore, we need also to change the criterion for computing the error to the MSE (2.3), because now we do not have a classification problem but a regression problem. We trained both a simple RNN, an LSTM and an RNN with our regularization to extend its memory (REG) for 20 epochs, using the Adam optimization scheme to update the parameters. The results are reported in Table 7.1. We can see that our method, where we bound the singular values between 0.9 and 1.1 with the additional condition on their RMS to be close to 1, outperforms the others, both on the training set and on the testing set, in all the cases of two and three bouncing balls.

Table 7.1 MSE for two and three bouncing balls on training and testing sets.

	Two balls		Three balls	
	Training	Testing	Training	Testing
BPTT	5.8002	6.7410	6.8348	7.4344
LSTM	6.9431	8.0639	6.1513	6.5897
REG	**5.5308**	**6.5234**	**5.9903**	**6.5842**

Moreover, if we analyse the trend of the error for all the epochs, we can notice from Figure 7.2 that, especially in the case of three bouncing balls (right image), our method (blue line) reaches the convergence value much earlier than the others (BPTT with a simple RNN in green, and BPTT for LSTM in red).

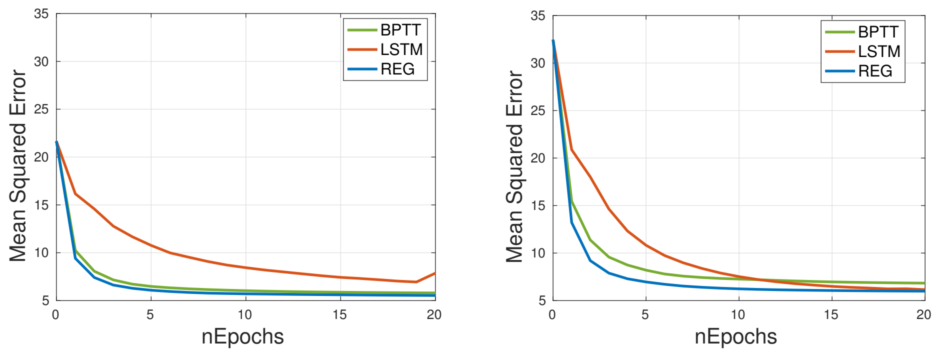


Fig. 7.2 MSE trends of different methods for two (left) and three (right) bouncing balls.

7.2 Mobile Users Distribution over a Region

In this Section, we deal with bigger images than the one considered for the bouncing balls problem, using data provided TIM about the distribution of mobile users over a region. Each image of the video sequence is represented with a 1927×1556 presence matrix, where in each pixel, referring to an area of about 17e3 m^2, we have the number of active mobile phones. Each image is generated every 15 minutes, but, since the variation are not so relevant in that time lapse, we sample the data every hour. In Figure 7.3, we represent a video frame of the Piedmont region (on the left side) and a zoom in the area of Turin (on the right side) through a colormap, where the red pixels refers to a more crowded area and the blue ones to an empty zone, and, since the Piedmont has not a rectangular shape, the value of the pixels outside the region are set to -1.

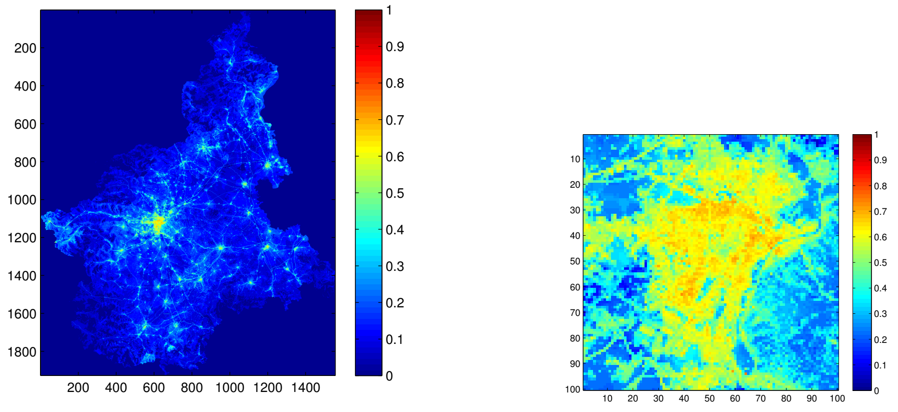


Fig. 7.3 Example of distribution of mobile users in Piedmont region (left) and a zoom in the Turin area (right), represented in a logarithmic scale.

Moreover, in order to enhance the values, we display the natural logarithm of the presence matrix, and thus we can notice that the image highlights the residential areas and the main streets, as shown in Figure 7.4, where we overlapped our data to a map of the Piedmont region.

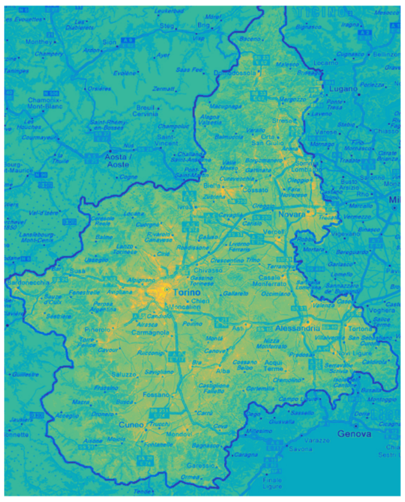


Fig. 7.4 Overlap between analysed data and a map of Piedmont.

Since the dimensionality of the data is really high, *i.e.* we have about 3e6 pixels for $T = 2880$ time-steps, which corresponds to the period between April and July 2016, we exploit the SVD of the input images, building a low-rank approximation of them . Let us consider the collection of input images in a big matrix X of dimension 2880×2998412, where each row corresponds to one time-step vectorized image. We compute an low-rank incremental SVD [77, 78] for this matrix while loading the data. At the end of this process, we obtain the matrices U, S, and V such that

$$USV^\top \approx X, \qquad \text{where} \begin{cases} U \text{ is } 2880 \times k \\ S \text{ is } k \times k \\ V \text{ is } 2998412 \times k \end{cases}, \tag{7.1}$$

with k rank of the approximation.

We test the accuracy of this approximation for $k = 100, 200$ in three specific zones: a pixel in the center of Turin, which is subject to high fluctuation and might become very crowded, a pixel in the suburb of Turin, where there are fewer people, and one in a Alessandria, a smaller town of Piedmont. As shown in Figure 7.5, we

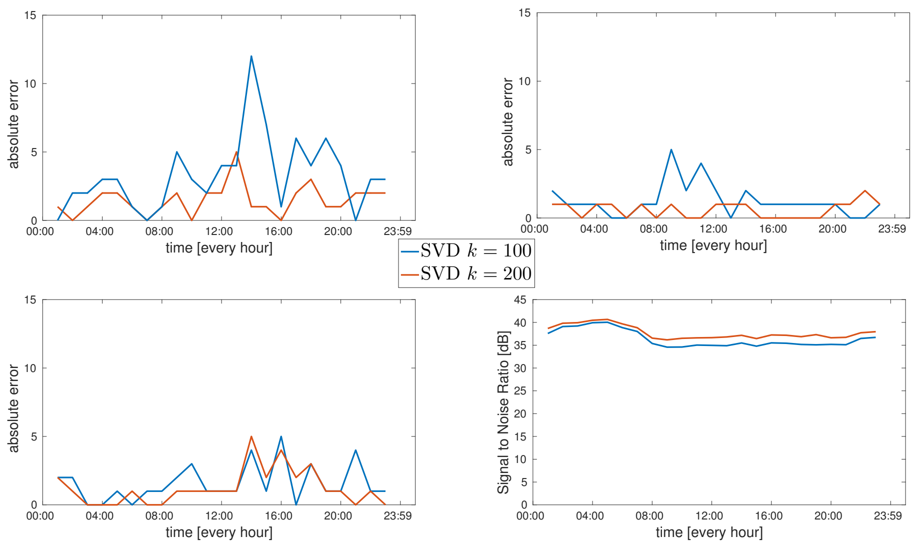


Fig. 7.5 Absolute error of the SVD reduction in three different zones, *i.e.* Turin center (top-left), Turin suburb (top-right) and Alessandria (bottom-left), and SNR considering the whole Piedmont region (bottom-right).

compare the absolute error committed each hour during a day in these three areas, along with the signal to noise ratio (SNR) on the whole Piedmont. As expected, the approximation with rank $k = 200$ gives a better performance, which means lower absolute error and higher SNR, than the one with $k = 100$. Anyway, both of them provides interesting results, since, if we compute the mean of the absolute error, we obtain at most, as reported in Table 7.2, an error of two people for $k = 100$ in the center of Turin and of 1.3 for $k = 200$. Moreover, we obtain also high values of the SNR: 36.4927 dB and 37.5608 dB for $k = 100, 200$, respectively, over the whole Piedmont region.

Table 7.2 Absolute mean error of the SVD approximation.

	Turin Center	Turin Suburb	Alessandria
SVD $k = 100$	2.1304	0.5652	1.3043
SVD $k = 200$	1.3043	0.5217	0.7826

These results encourage us to exploit the SVD low-rank approximation in order to reduce the dimensionality of the data. Therefore, we use as input $x(t)$ each column of the matrix obtained by multiplying $(US)^\top$ of (7.1), which means that our inputs are k-length vectors. We choose an hidden state of 128 neurons and an output non-linear function ϕ_y as the identity function, since we do not have any requirements on the output values. Then, once computed the outputs $y(t)$, $\forall t$, it is used as next input $x(t+1)$, or we multiply it by $V^\top$ of (7.1), in order to obtain back again the image representing the distribution of mobile phones. In the following, for the experiments, we choose $k = 200$.

The prediction of mobile users in future time-steps is a particularly interesting problem for its application fields. The security force could benefit from this knowledge to predict the movement of the population in the near future, for example in case of events such as a football match or a concert or a festival. Public transport agency could enhance their supply in case of sudden increasing flow of people. On the other hand, also TIM could have a tool to dispense further services, such as applications to provide users information about crowded places or traffic jams on streets. Moreover, these predictions could furnish details on the usage of the mobile network, increasing the signal where needed.

Therefore, as the promptness of the prediction is essential, we do not use LSTM architecture. Moreover, latest pieces of information are more relevant than other in the past, so we do not use our regularization to extend the memory of the network either, but we can apply our least square initialization of Chapter 5 (INIT) and use it also as on-line adaptative method to update the parameters (UPDATE) as soon as new data are available.

The results are reported in the following. Figure 7.6 shows three matrices, one next to the other, with the true distribution at a specific instant of time in the left side, the estimated one using an RNN trained starting with initialized parameters and using our update method, and the absolute error between them in the right side. As we can see, in the right side the matrix is totally coloured in blue, which corresponds

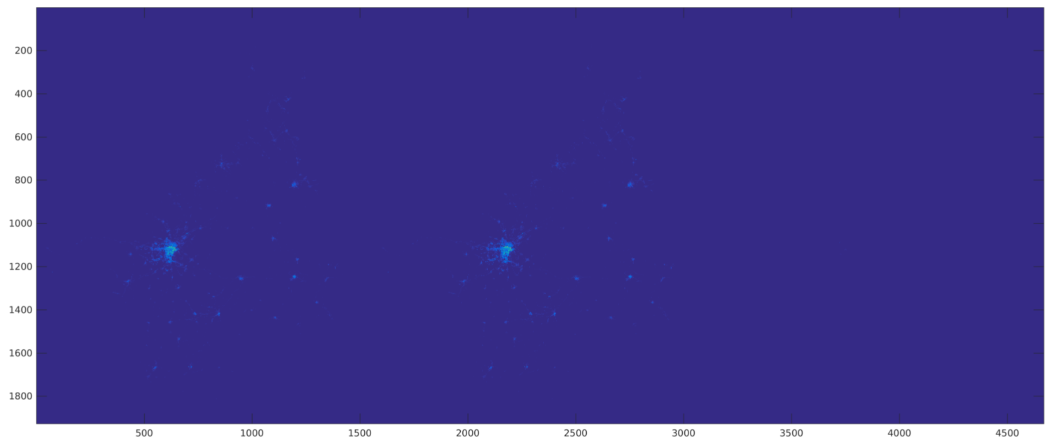


Fig. 7.6 From left: real distribution, predicted one and absolute error between them.

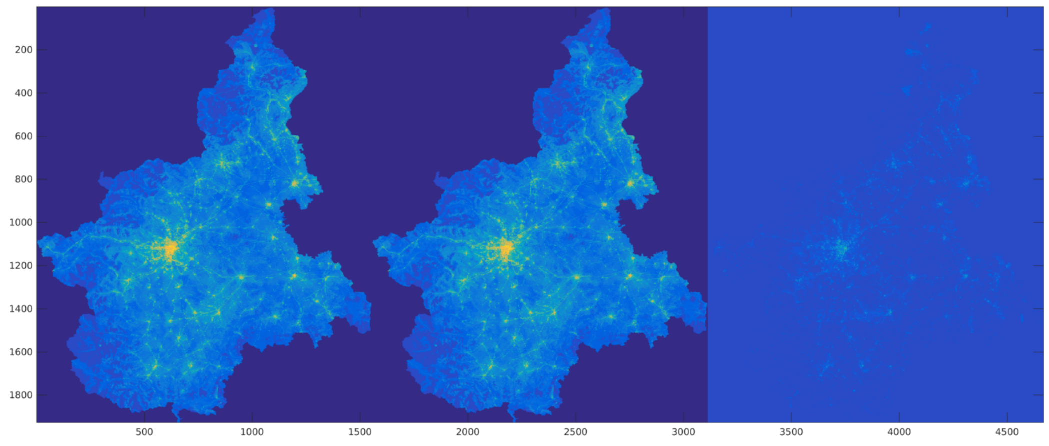


Fig. 7.7 From left: real distribution, predicted one and absolute error between them, all filtered with a logarithm.

to a small error and an high SNR (32.3355 dB, for $k = 200$). In Figure 7.7, we repeat the same images as the one in Figure 7.6 filtered with a natural logarithm in order to enhance the values, recognizing the shape of Piedmont and the main cities and streets, and also to better view the absolute error on the right side. As shown in Table 7.3, using a neural network approach to predict the future time-steps outperforms a bare prediction such as the replication (REPL) of previous time-step. Moreover, our initialization is helpful in this experiment, since it improves the results of classical BPTT, especially using it also in an adaptative way, running 2 iterations in order to update the parameters according to the latest time steps.

Table 7.3 Signal to noise ratio of the prediction of mobile users in Piedmont region.

	Piedmont (SNR [dB])
REPL	27.9337
BPTT	30.3370
INIT	32.2833
INIT+UPDATE	**32.3355**

Then, we focus our attention on three characteristic pixels, one in the center of Turin, one in the suburb of Turin, and one in a Alessandria, showing the trend of the distribution of mobile users and the absolute error during a day, in Figures 7.8-7.9-7.10, respectively. Even in these pixels starting with initialized parameters and updating them continuously with new data produces a relative absolute error around $2-3.5\%$, which translate into an prediction error of 2-6 people, while classical BPTT is between 3.5% and 7% and the replica even between 5% and 9%, *i.e.* it can fail by 20 people in average. The mean absolute errors are reported in Table 7.4.

Table 7.4 Mean absolute error of the prediction of mobile users.

	Turin Center	Turin Suburb	Alessandria
REPL	13.5909	8.6818	5.8182
BPTT	10.3043	5.6522	3.9565
INIT	6.6957	3.5217	2.4783
INIT+UPDATE	**6.2174**	**3.3043**	**2.0870**

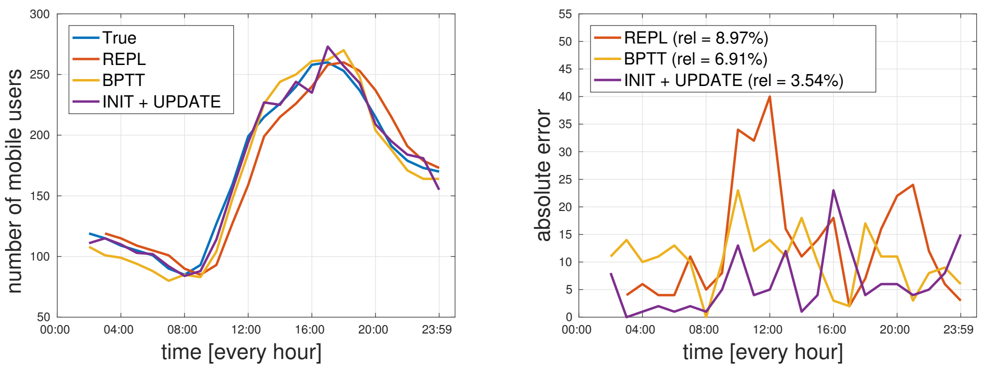


Fig. 7.8 Number of mobile users and absolute error in the center of Turin during a day.

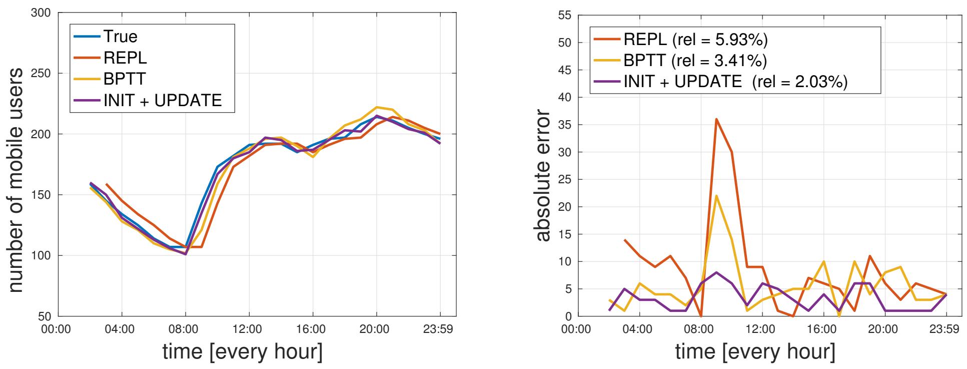


Fig. 7.9 Number of mobile users and absolute error in the suburb of Turin during a day.

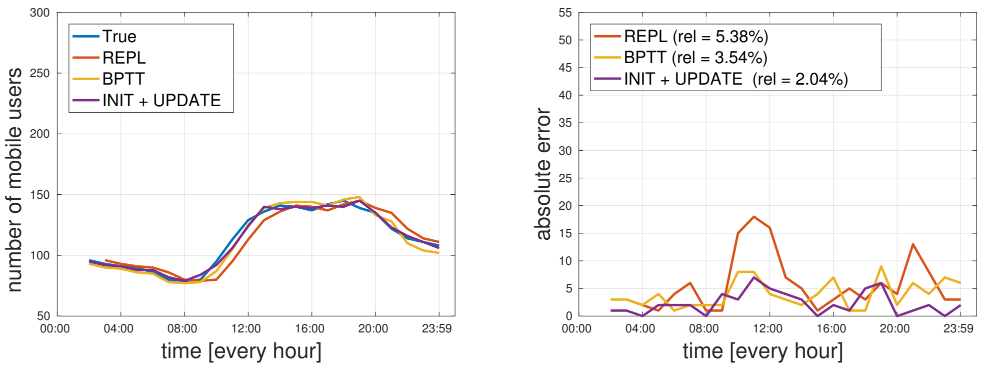


Fig. 7.10 Number of mobile users and absolute error in Alessandria during a day.

7.3 Discussion

In this chapter we analysed the prediction of video sequences using RNNs.

We started with the literature problem of bouncing balls, where the network had to learn the physical laws of two or three colliding balls only from the visual information of the video. We compared our regularization method explained in Chapter 4 with a simple RNN and an LSTM architecture. The results showed an improvement in the performance using our method, both in case of two and three balls and both on training and testing sets (see Table 7.1).

Then, we considered data provided by TIM about the distribution of mobile users over a region. We addressed the huge amount of data in this problem exploiting an incremental SVD of the data as soon as new ones are available. This is an very interesting problem with a great number of different application fields: in security scenarios, in public transport for increasing the buses supply, or for the mobile company to provide a tool to its users with information about crowded places and traffic jams. Since the promptness of the prediction is fundamental in this problem, we did not use LSTM networks neither the regularization to extend the memory because the latest data are more relevant. Therefore we apply our least square initialization (see Chapter 5) using it also as an real-time learning method. The results, reported in Table 7.4, show that we can predict the distribution of mobile users with an error from 2 to 6 people in a less or more crowded area respectively.

Chapter 8

Missing Data Imputation in Audio Spectrograms via Recurrent Neural Networks

Brief - *This chapter proposes the use of neural networks to restore audio signals with incomplete spectrograms, that is, with missing time-frequency data. An approximate recovery of the missing data is often possible, since redundancies and fixed structures repetitions are very common in music and speech. For this long-standing problem, different deterministic and probabilistic models and algorithms have been proposed in the literature, which generally assume some prior knowledge on the structure of the spectrogram, namely on its rank or some probabilistic distribution on its columns. Neural networks, instead, can learn the structure of the data with no prior assumption and hence provide a better adaptation for missing data imputation. We design recurrent and long short-term memory neural networks specifically tailored to audio signals, and we demonstrate via numerical experiments that a performance improvement can be achieved with respect to the state-of-the-art methods.*

Missing data imputation refers to the problem of recovering portions of signals that have been lost or occluded by noise or by competing signals. This is a pervasive problem in image and audio processing [79, 80], in particular in speech recognition [81, 82]. Audio signals are usually represented as *spectrograms*, that is, time-frequency distributions of acoustic power. Visually, a spectrogram is a two-dimensional matrix in which each entry represents the signal power at a given time and frequency. It is often the case that the spectrogram is incomplete, for example due to noise, interferences, filtering operation in signal transmission, aggressive compression (such as MP3) or audio editing. This may remove some time-frequency data or make them unreliable, which makes the audio signal unusable. On the other hand, spectrograms generally have significant time-frequency correlations which can be exploited to assign reasonable values to the missing data. In the literature, several approaches have been proposed that process the available data to discover fixed structures and redundancies that allow to impute the missing values. On one hand, deterministic models like k nearest neighbours or singular value decomposition have been explored for interpolation; on the other hand, probabilistic models have been proposed to build a statistics, based on available data, and then estimate the missing values [80].

In this chapter, we propose a new approach to missing data imputation for time-frequency representation of audio signals using neural networks. As in [80], we consider a spectrogram with gaps, which we try to recover by learning the spectral structure of the signal from the available spectrogram cells. We take into account a particular kind on neural network, known as recurrent neural network (RNN, [83, Chapter 10]), which exploits a temporal dependency of the data. Moreover, we consider the extension to long short-term memory networks (LSTMs, [14]), which can store information of long time sequences and have been applied successfully in various domains, *e.g.* handwriting recognition [84], speech recognition [7], and image captioning [10].

In the last years, neural networks have been largely explored as methods to patch data sequences with gaps, as they naturally provide an interpolation strategy that is able to take into account the data structure with no prior information or predetermined model. In audio signal processing, neural networks have been proposed to predict and recover music [85, 86], but more classical probabilistic schemes are still preferred in the recent literature [80, 87–90]. Probabilistic methods typically assume some prior structure. For example, in [80], the columns of the spectrogram are considered as histograms drawn from a mixture multinomial process, similarly to the model used in probabilistic latent semantic analysis (PLAS, [91]), while in [90] wider compositional models have been analysed, based on which a spectrogram is a linear combinations of *atoms*, and both atoms and coefficients of the linear combination have to be estimated. Well-known mathematical methods like Expectation-Maximization (EM) and convex optimization are used to perform the estimation. Experiments show that such methods achieve satisfactory performance. However, different parameters need to be priorly set, *e.g.*, the number of atoms and the number of bases of the mixture.

Our rationale is that neural networks may overcome such a tricky presetting and modelling by learning the specific features directly from data, employing a powerful non-linear model. A suitably designed neural network, in fact, has the capability to learn and adapt promptly to the considered signals, which reveals a significant potential for audio signal processing [12]. The aim of this chapter is to start to draw from such potential, by designing neural networks able to be trained over a small dataset and to outperform the state-of-the-art methods for incomplete audio spectrograms recovery.

8.1 Background on State-of-the-art Methods

As already mentioned, deterministic ad probabilistic schemes have been evaluated in the literature for audio imputation. The most popular deterministic schemes for matrices (*e.g.*, spectrograms) with missing entries are k nearest neighbours (k-NN) and singular value decomposition (SVD) [92]. Given an incomplete matrix, k-NN search for the k columns closer to the incomplete columns and uses their mean for imputation. The SVD method, instead, iteratively computes a least squares SVD approximation of the incomplete matrix, having priorly set a certain rank. SVD is generally more accurate than k-NN, but its complexity is higher.

Deterministic methods have been recently outperformed by probabilistic methods. In [80], a model similar to PLSA model has been exploited. Each spectral vector S_t is considered as a scaled histogram: given a random variable f over the frequencies $\{1,\dots,F\}$, at time t, S_t is generated according to the mixture multinomial distribution $P_t(f) = \sum_{z=1}^{Z} P_t(z)P(f|z)$, where $P(f|z)$ are the "bases" of the mixture (not dependent on time), and $P_t(z)$ the weights. The number Z of bases has to be set a priori. The EM algorithm is then used to learn the parameters of the distribution from available data and impute the missing data.

In [88] a variant has been proposed that recasts the model into a non-negative hidden Markov model (N-HMM). PLSA in fact considers each time frame independently, while N-HMM envisages also the dynamics in time. More precisely, while PLSA build a unique dictionary of spectral vectors, N-HMM builds different dictionaries for different frequency sub-bands. As shown in [88], this is the right choice in case of audio clipping, that is, when some some frequencies are cut off and then totally missing.

8.2 Proposed Neural Network Design

RNNs, and in particular LSTMs, have been largely explored for speech and music learning purposes. Above all, pop music is often a combination of few melodic and harmonic patterns, repeated in time. Such simple structures, which can be easily learned and remembered by a human brain, are good candidates to be learned by a neural network. Repeated patterns require to store some past information. This suggests the feed-forward neural networks, which exploit only the present information to predict the future, are not enough for this purpose. This is why researchers addressed much effort towards RNNs, and in particular LSTMs, which are able to store long-time patterns.

Experiments on music learning, reproduction, generation, and event detection via RNN/LSTMs have attracted a lot of attention in the last decades (see [93] and the references therein). In particular, RNNs/LSTMs have been designed to reveal acoustic novelties (*e.g.*, for surveillance applications) [94], and to learn structures from a particular music genre to predict melodies [95]. Besides, a huge number of more audacious experiments have been documented, which attempt to compose music after training on a specific genre or composer discography.

Our contribution differs from these previous works at least for two aspects. First, we do not perform a training over a large dataset of correlated music, but only on the part of song which is available, which dramatically reduces the training time. Second, we dot not process the song itself, but its spectrogram, which represents the evolution of the frequencies in time. The audio signal evolution and the spectrogram are clearly closely correlated, but the analysis on the spectrogram is more sophisticated as both correlation in time and in frequency can be used. This could be suitable for example to recover lost sub-bands, *e.g.*, after aggressive MP3 compression.

In order to apply a neural network approach to tackle the audio imputation problem, we set the inputs $x(t)$ to be the columns of the spectrogram for each time step t, while the expected output is the following time step, *i.e.*, $y(t) = x(t+1)$. Thus, each input/output pair is a 129-column vector.

For what concerns the non-linearities, on the one hand, in the RNN we set $\phi_h = \tanh$, which bounds the values of the hidden state between -1 and 1 [83, Section 10.2], and ϕ_y as the identity function, *i.e.*, we do not impose constraints on the output values. On the other hand, for the LSTM we set $\phi_i = \sigma$, $\phi_j = \tanh$, $\phi_f = \sigma$, $\phi_o = \sigma$, where $\sigma(z) = 1/(1+e^{-z})$ is the standard logistic sigmoid function, which shrinks the values between 0 and 1 [14]. As for the RNN, ϕ_y is the identity function.

We train both our RNNs and LSTMs on the available data (*i.e.*, the columns of the spectrogram before and after the gap) with the back-propagation algorithm, employing the Adam optimization method [27] to update the parameters of the network. We choose an hidden state $h(t)$ of 129 units, which is equal to the number of the frequencies in the spectrogram, and, during the training, we unfold the network for 45 time steps and using 20 mini-batches to be trained in parallel. We run the training phase for 10 epochs, starting with a learning rate of 1e-3, which decays by a factor of 0.97 each epoch starting after the 5th one. All these hyper-parameters have been tuned on the basis of several experiments. Finally, once the network has been trained, we fill the gaps running the network with input $y(t-1)$ at time t.

8.3 Experiments

In this section, we present some tests of reconstruction of audio signals with incomplete spectrograms. We consider the setting where spectrogram values are totally or partially missing in a given interval of time, and we compare our RNN and LSTM recovery to EM [80] and SVD methods [92]. We point out that EM can be applied only if some frequencies are still available in the incomplete interval. Moreover, we do not compare to N-HMM [88] which, as we said, is oriented to clipping problems.

8.3.1 Setup

As in [80], we formulate the problem in terms of minimization of the mean-square error on the spectrogram, which is the most common metric. In [90], other metrics have been studied (*i.e.*, Kullback-Leibler and Itakur-Saito divergences), which according to some authors are more suitable for audio performance evaluation: this is a point that we will investigate further in our future work. The spectrogram that we consider is the magnitude of the short-time Fourier transform (STFT) of the audio signal. Finally we assume that the “spectrographic mask” that determines which spectrogram cells are unreliable or missing is known.

We show two performance metrics. The first one is the spectrogram relative error (RE), defined as

$$\mathrm{RE} := \frac{\left\| \widehat{S} - S \right\|_F}{\|S\|_F},$$

where S and $\widehat{S}$ respectively represent the true and the estimated spectrograms, while $\|\cdot\|_F$ is the Frobenius norm. The second one is the signal-to-noise ratio (SNR) on the audio signal, that is:

$$\mathrm{SNR} := \frac{\|z\|_2}{\|z - \widehat{z}\|_2},$$

where z and $\widehat{z}$ respectively represent the true and the estimated audio signals.

For EM and SVD, we respectively set the number of mixture atoms $Z = 10$ and the rank $= 10$ on the basis of preliminary tests: beyond these values, accuracy does not increase significantly, while complexity becomes critical. The EM distribution parameters are learned from the complete data (with no gaps), which slightly favours EM.

8.3.2 Results

For our experiments, we consider six music pieces of different genres: (I) Black Night, by Deep Purple; (II) If, by Pink Floyd; (III) Walk on the Wild Side, by Lou Reed; (IV) Satellite of Love, by Lou Reed; (V) 16 Shades of Blue, by Tori Amos; (VI) Polonaise A flat Op. 53, by Chopin.

For all of them, we consider a central gap of width equal to $\frac{1}{4}$ of the spectrogram time range. In this interval, we assume that only 2, 4, or 16 frequencies over 129 (starting from the lowest ones) are available, or that all the frequencies are missing (in which case, EM cannot be applied).

The results are collected in the Tables I-IV, respectively considering the cases with 0, 2, 4, or 16 available frequencies. The best performance for each experiment is highlighted in bold. As expected, by increasing the number of available frequencies the performance is generally improved. Notice that, in almost all our experiments, our neural network-based approach outperforms SVD or EM methods. We notice that with 16 available frequencies the RE is always below the 10%, and the corresponding SNRs are high, for all the methods. With fewer available frequencies, instead, LSTM and RNN have the best performance, except for (VI); this can be due to sophisticated structure of this piece if compared to the others. Finally, comparing only RNN and LSTM performances, LSTM has a general better performance (except for a couple of instances) since it characterized by a longer memory thanks to the overcoming of the "vanishing gradient" problem [14].

Finally, we graphically illustrate the recovery test on (IV) by depicting the spectrograms in Figure 8.1. In the first row, we show the original and the incomplete spectrograms, while in the second and third row we show the recovered spectrograms, respectively SVD, EM, RNN, and LSTM. While the SVD and the EM methods tends to fill the gaps with a combination of the other columns, our approaches predict the following columns providing an horizontal evolution in time. This horizontal correlation, which is significant in the images, helps improving the performance in the audio recovery.

8.4 Discussion

In this chapter, we showed how to address the missing data problem for audio spectrograms using recurrent neural networks. We experimentally proved that it is possible to fill the gaps with RNNs and LSTM networks improving the performance of state-of-the-art methods. Moreover, unlike EM, RNNs can recover the spectrogram even in case of time intervals in which all frequencies are missing. Next step would be to improve RNN/LSTMs performance exploiting the two-directional correlations in the spectrogram.

Table 8.1 Complete gap (no available frequencies)

Song	RE			SNR (dB)		
	SVD	RNN	LSTM	SVD	RNN	LSTM
(I)	0.32	**0.28**	0.29	10.62	**11.52**	11.33
(II)	0.31	0.37	**0.28**	10.98	8.83	**11.77**
(III)	0.95	0.49	**0.37**	2.11	6.10	**9.08**
(IV)	0.43	0.39	**0.28**	7.11	8.68	**9.89**
(V)	0.53	**0.31**	0.40	6.09	**10.84**	8.10
(VI)	**0.35**	0.41	0.37	**9.69**	7.99	8.48

Table 8.2 2 available frequencies

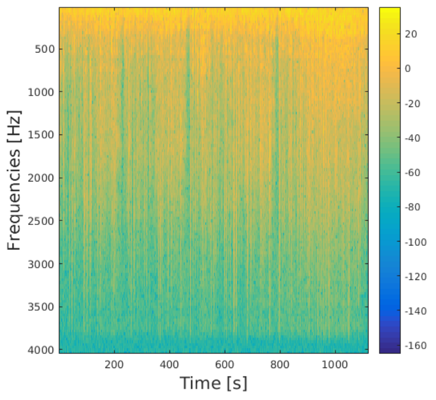
Song	RE				SNR (dB)			
	SVD	EM	RNN	LSTM	SVD	EM	RNN	LSTM
(I)	0.22	0.24	0.19	**0.18**	13.17	12.71	14.28	**14.57**
(II)	0.17	0.14	0.15	**0.14**	16.38	17.72	17.59	**17.84**
(III)	0.16	0.17	0.28	**0.16**	15.08	14.96	10.81	**15.21**
(IV)	0.33	0.28	0.41	**0.21**	9.26	11.73	7.97	**13.27**
(V)	0.28	0.31	0.28	**0.28**	9.45	9.11	9.48	**10.31**
(VI)	0.34	**0.32**	0.36	0.35	9.85	**10.19**	8.57	8.89

Table 8.3 4 available frequencies

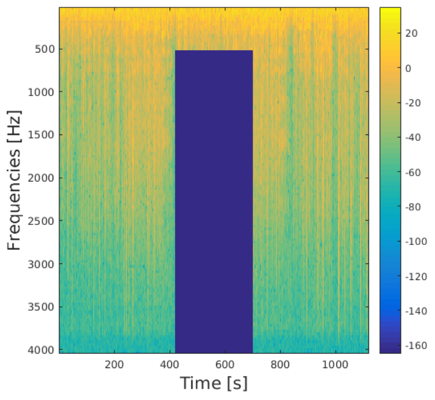
Song	RE				SNR (dB)			
	SVD	EM	RNN	LSTM	SVD	EM	RNN	LSTM
(I)	0.16	0.15	0.15	**0.13**	15.59	16.64	16.61	**17.39**
(II)	0.05	0.05	**0.04**	0.04	26.64	26.64	**27.57**	27.50
(III)	0.08	0.08	0.11	**0.07**	21.05	21.50	18.44	**21.86**
(IV)	0.21	0.17	0.38	**0.16**	12.98	15.22	8.40	**15.79**
(V)	0.25	0.23	0.24	**0.22**	10.78	11.66	11.23	**11.93**
(VI)	0.24	0.22	0.25	**0.21**	13.12	13.58	12.18	**13.62**

Table 8.4 16 available frequencies

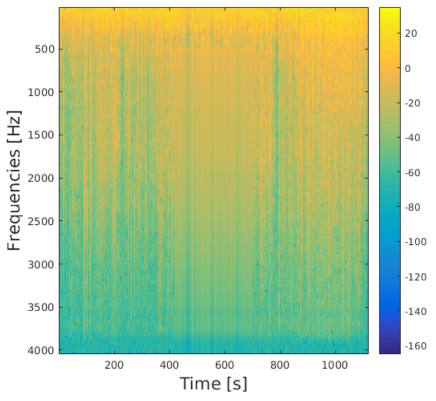
Song	RE				SNR (dB)			
	SVD	EM	RNN	LSTM	SVD	EM	RNN	LSTM
(I)	0.09	0.07	0.07	**0.06**	20.61	21.18	23.40	**24.18**
(II)	0.02	0.02	**0.02**	0.02	33.45	33.35	**33.52**	33.17
(III)	0.04	0.04	0.04	**0.04**	27.76	27.80	26.42	**27.90**
(IV)	0.08	0.07	0.09	**0.07**	21.34	22.41	19.92	**22.58**
(V)	0.06	0.04	0.06	**0.04**	23.90	26.97	24.17	**27.12**
(VI)	0.03	0.03	0.03	**0.03**	30.01	29.34	28.99	**30.67**



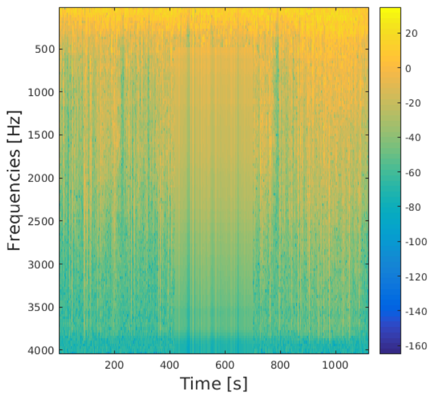
(a) original



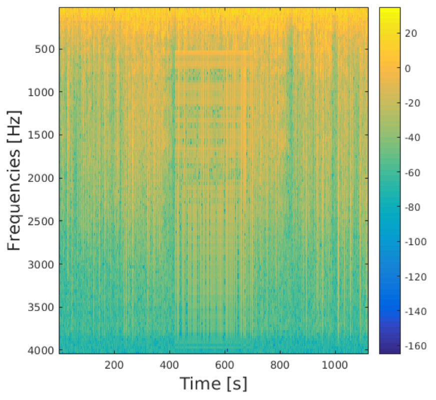
(b) incomplete



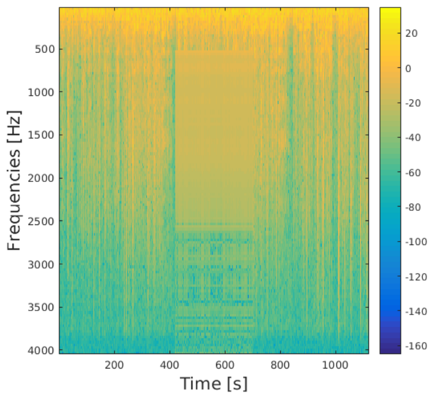
(c) SVD



(d) EM



(e) RNN



(f) LSTM

Fig. 8.1 Recovery of spectrogram of (IV). From top to bottom, and left to right: (a) original spectrogram; (b) incomplete spectrogram; recovered spectrograms by (c) SVD, (d) EM, (e) RNN, (f) LSTM.

Chapter 9

Conclusions and Future Developments

In this thesis, we studied a particular class of artificial neural networks, which is specific for analysing time sequences and is characterized by a feedback loop of the *hidden state* to link the current time step to the previous one. It is known as *recurrent neural network* (RNN, [5]).

Despite RNNs are known to be suitable to solve many tasks in several fields (*e.g.* language modelling [6], speech recognition [7], video analysis [9], image captioning [10], and so on), they are difficult to train, especially on long-range temporal structure problems, due to the so-called *vanishing gradient* problem [16, 11]. Therefore, our work focussed both on a theoretical understanding of RNNs and on their application to a wide range of non-linear prediction problems.

As regards the theoretical part, we addressed the weaknesses of state-of-the-art models and contributed to improve the performance of RNNs. First of all, we analysed echo state networks (ESN, [15]), which do not have to deal with vanishing gradients and do not require prohibitive computational costs, but they do not have any theoretical guarantees, since they are based on a conjecture that remains to be shown [17, 18].
Specifically, we established conditions for the existence of stable limit cycles, whose existence is equivalent to the *echo state* property. We provided sufficient conditions for the convergence to a trajectory that is uniquely determined by the driving input

signal, independently on the initial states, thanks to the notion of attractive fixed-point and the propagation of the periodicity of the input signal to the hidden state.

Then, we proposed an alternative solution to the vanishing gradient problem, which does not involve a modification in the architecture, such as long short-term memory (LSTM, [14]) networks, nor ESNs and stable limit cycles. Our new method allows us to train a very simple RNN, exploiting the singular value decomposition of the vanishing factors in the gradient and the random matrices theory. We found that the singular values have to be confined in a narrow interval and derive conditions about their root mean square value, which should be equal (or really close) to one.

Moreover, we developed a novel approach for accelerating the training process of an RNN. Thanks to a least square regularization, we can initialize the parameters of the network, in order to start the training closer to the solution and, therefore, to speed up the training process, running fewer epochs of classical training algorithms. We also noticed that our method gets really close to the solution, so we can run more iterations and use it as a fast training algorithm. Finally, thanks to its speed, it can be also used as a real-time training algorithm, in particular in situations where the promptness of the information is a critical requirement and the latest data are the most relevant one.

In the last part of this thesis, we applied RNNs to non-linear prediction problems. We considered prediction of numerical sequences, estimating the following output choosing from a probability distribution the next input of the sequence. We tested the generation of written text, where we need to predict the following character, which is associated to a number, in order to compose words and sentences. We also predicted the path of walking mobile users in the city center as a sequence of crossroads, which brings several applications in terms of a smart cities framework.

Then, we presented how to predict video frames, starting from the literature bouncing balls problem [75, 76], where the network had to learn the physical laws of colliding balls only analysing the visual information. We showed that our regularization method can improve the results of both a simple RNN and an LSTM both in case of two and three bouncing balls. We also considered data provided by TIM about the distribution of mobile users over a region and, for this specific problem, we compared our initialization method to a classical RNN, since the latest data are the most relevant and so we did not need long memory. As a matter of fact, we used it as a real-time learning algorithm, obtaining an absolute error of 2 or at most 6 people in very crowded areas.

Finally, we showed how to address the problem of missing data for audio spectrograms [80] with RNNs. We restored audio signals with missing time-frequency data, designing RNNs and LSTMs specifically tailored to audio signals, and demonstrated via numerical experiments that a performance improvement can be achieved with respect to the state-of-the-art methods. Moreover, unlike state-of-the-art methods [80], RNNs can recover the spectrogram even in case of time intervals in which all frequencies are missing.

As future work, we are going to carry on our study on the vanishing gradient phenomenon, solving hard problems which require a lot of memory. We are also going to find stronger conditions on the parameters involved in our regularization method and to develop new approaches to address the problem.

Moreover, we are going to apply our methods to other prediction problems. For example, we are going to test them in the challenging dataset from NASA (https://eosweb.larc.nasa.gov), where meteorological data are available all over the Earth. At each coordinate, for a period of time from the 1st of July in 1983 to the 30th of June in 2005 (*i.e.* 22 years, for a total of $T = 8036$ time-steps), the following 19 meteorological parameters are sampled daily:

1-3. Year
Month
Day
4. Average insolation incident [kWh/m^2/day] (Figure 9.1)
5. Average insolation clearness index (from 0 to 1.0)
6. Average clear sky insolation incident [kWh/m^2/day]
7. Average clear sky diffuse insolation [kWh/m^2/day]
8. Average clear sky direct normal radiation [kWh/m^2/day]
9. Average clear sky insolation clearness index (from 0 to 1.0)
10. Average downward long-wave radiative flux [kWh/m^2/day]
11. Average top-of-atmosphere insolation [kWh/m^2/day]
12. Average atmospheric pressure [kPa]
13. Average Earth skin temperature [°C]
14. Average air temperature (Figure 9.2)
15-16. Min/Max air temperature [°C]
17-18. Average specific/relative humidity (Figure 9.3)
19. Dew/frost point temperature [°C]

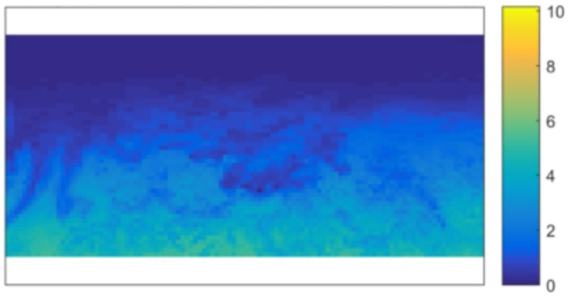


Fig. 9.1 Average insolation incident on a horizontal surface $[kWh/m^2/day]$.

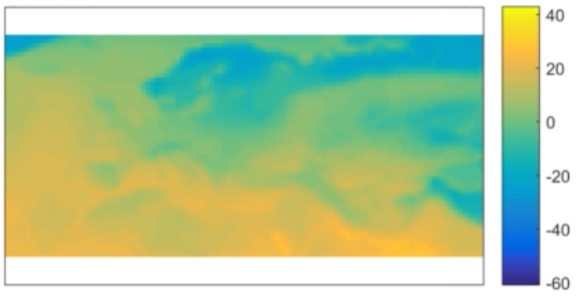


Fig. 9.2 Average air temperature at 10m above the surface on the Earth [°C].

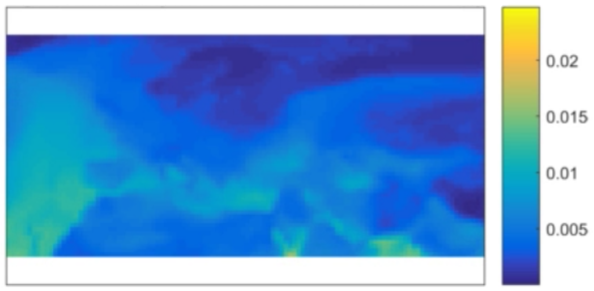


Fig. 9.3 Average specific humidity at 10m above the surface on the Earth $[kg/kg]$

We can use them as numerical sequence, analysing one single point on the map for all the meteorological parameters, exploiting the intra-correlation between them. On the other hand, we can also use them in a video prediction problem, taking into account each meteorological parameter separately over the whole Earth, or a smaller area.

Finally, we are going to improve RNN/LSTMs performance in missing data imputation problems, exploiting the two-directional correlation in the spectrogram, which mean we will not only consider the spectrogram as a time sequence of frequencies, but also as a frequency sequence of time steps.

References

[1] X. Glorot, A. Bordes, and Y. Bengio, “Domain adaptation for large-scale sentiment classification: A deep learning approach,” in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 513–520.

[2] A. Krizhevsky and G. E. Hinton, “Using very deep autoencoders for content-based image retrieval.” in *European Symposium on Artificial Neural Networks*. Citeseer, 2011.

[3] A.-r. Mohamed, G. E. Dahl, and G. Hinton, “Acoustic modeling using deep belief networks,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 14–22, 2012.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[5] ——, “Learning internal representations by error propagation,” *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, 1986.

[6] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, “Recurrent neural network based language model.” in *Interspeech*, vol. 2, 2010, p. 3.

[7] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[9] P. H. Pinheiro and R. Collobert, “Recurrent convolutional neural networks for scene labeling.” in *ICML*, 2014, pp. 82–90.

[10] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[11] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[12] R. J. Williams and D. Zipser, “A learning algorithm for continually running fully recurrent neural networks,” *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[13] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] H. Jaeger and H. Haas, “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication,” *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[16] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen netzen,” *Diploma, Technische Universität München*, p. 91, 1991.

[17] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the “echo state network" approach.* GMD-Forschungszentrum Informationstechnik, 2002.

[18] ——, “Controlling recurrent neural networks by conceptors,” *arXiv preprint arXiv:1403.3369*, 2014.

[19] C. Darken, J. Chang, and J. Moody, “Learning rate schedules for faster stochastic gradient search,” in *Neural networks for signal processing*, vol. 2. Citeseer, 1992.

[20] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization,” in *Advances in neural information processing systems*, 2014, pp. 2933–2941.

[21] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[22] Y. Nesterov, “A method of solving a convex programming problem with convergence rate o(1/k2),” in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[23] G. Hinton, N. Srivastava, and K. Swersky, “Lecture 6a overview of mini–batch gradient descent,” *COURSERA: Neural Networks for Machine Learning*.

[24] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[25] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.

[26] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, 2012.

[27] J. Ba and D. Kingma, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representation*, 2015.

[28] W. Zaremba, “An empirical exploration of recurrent network architectures,” 2015.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[30] N. M. Mayer, “Input-anticipating critical reservoirs show power law forgetting of unexpected input events,” *Neural computation*, 2015.

[31] H. Jaeger, *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik, 2001.

[32] ——, “The “echo state” approach to analysing and training recurrent neural networks-with an erratum note,” *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[33] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, “Character-aware neural language models,” *arXiv preprint arXiv:1508.06615*, 2015.

[34] X. Zhang, L. Lu, and M. Lapata, “Top-down tree long short-term memory networks,” *arXiv preprint arXiv:1511.00060*, pp. 0–5, 2016.

[35] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” in *Advances in Neural Information Processing Systems*, 2015, pp. 577–585.

[36] H. Sak, A. Senior, K. Rao, and F. Beaufays, “Fast and accurate recurrent neural network acoustic models for speech recognition,” *arXiv preprint arXiv:1507.06947*, 2015.

[37] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[38] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[39] L. Shang, Z. Lu, and H. Li, “Neural responding machine for short-text conversation,” *arXiv preprint arXiv:1503.02364*, 2015.

[40] O. Vinyals and Q. Le, "A neural conversational model," *arXiv preprint arXiv:1506.05869*, 2015.

[41] J. Weston, "Dialog-based language learning," *arXiv preprint arXiv:1604.06045*, 2016.

[42] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[43] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom, "Teaching machines to read and comprehend," in *Advances in Neural Information Processing Systems*, 2015, pp. 1693–1701.

[44] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[45] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3367–3375.

[46] M. S. Pavel, H. Schulz, and S. Behnke, "Recurrent convolutional neural networks for object-class segmentation of rgb-d video," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[47] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[48] L. Theis and M. Bethge, "Generative image modeling using spatial lstms," in *Advances in Neural Information Processing Systems*, 2015, pp. 1927–1935.

[49] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[50] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using lstms," *CoRR, abs/1502.04681*, vol. 2, 2015.

[51] V. Patraucean, A. Handa, and R. Cipolla, "Spatio-temporal video autoencoder with differentiable memory," *arXiv preprint arXiv:1511.06309*, 2015.

[52] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," *arXiv preprint arXiv:1412.6632*, 2014.

[53] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3156–3164.

[54] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2425–2433.

[55] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1–9.

[56] M. Ren, R. Kiros, and R. Zemel, "Exploring models and data for image question answering," in *Advances in Neural Information Processing Systems*, 2015, pp. 2953–2961.

[57] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[58] W. Zaremba and I. Sutskever, "Reinforcement learning neural turing machines," *arXiv preprint arXiv:1505.00521*, vol. 362, 2015.

[59] H. Mei, M. Bansal, and M. R. Walter, "Listen, attend, and walk: Neural mapping of navigational instructions to action sequences," *arXiv preprint arXiv:1506.04089*, 2015.

[60] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel, "Learning deep neural network policies with continuous memory states," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 520–527.

[61] A. Bay, S. Lepsoy, and E. Magli, "Stable limit cycles in recurrent neural networks," in *Communications (COMM), 2016 International Conference on*. IEEE, 2016, pp. 89–92.

[62] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fund. Math*, vol. 3, no. 1, pp. 133–181, 1922.

[63] H. Jaeger, "Long short-term memory in echo state networks: Details of a simulation study," Technical report, Jacobs University Bremen, Tech. Rep., 2012.

[64] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[65] S. Hochreiter, "Recurrent neural net learning and vanishing gradient," *International Journal of Uncertainity, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[66] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1310–1318.

[67] T. Papadopoulo and M. I. Lourakis, “Estimating the jacobian of the singular value decomposition: Theory and applications,” in *European Conference on Computer Vision*. Springer, 2000, pp. 554–570.

[68] Y. Bengio, O. Delalleau, and N. Le Roux, “The curse of highly variable functions for local kernel machines,” *Advances in neural information processing systems*, vol. 18, p. 107, 2006.

[69] Y. Bengio, Y. LeCun *et al.*, “Scaling learning algorithms towards ai,” *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[70] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[71] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, “Greedy layer-wise training of deep networks,” *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[72] Y. Li and A. B. Rad, “A cascading structure and training method for multilayer neural networks,” *International journal of neural systems*, vol. 8, no. 05n06, pp. 509–515, 1997.

[73] A. Navia-Vazquez and A. R. Figueiras-Vidal, “Efficient block training of multilayer perceptrons,” *Neural computation*, vol. 12, no. 6, pp. 1429–1447, 2000.

[74] D. Harris and S. Harris, *Digital design and computer architecture*. Elsevier, 2012.

[75] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 1033–1040.

[76] I. Sutskever, “Training recurrent neural networks,” Ph.D. dissertation, University of Toronto, 2013.

[77] H. Zha and H. D. Simon, “On updating problems in latent semantic indexing,” *SIAM Journal on Scientific Computing*, vol. 21, no. 2, pp. 782–791, 1999.

[78] M. Brand, “Fast low-rank modifications of the thin singular value decomposition,” *Linear algebra and its applications*, vol. 415, no. 1, pp. 20–30, 2006.

[79] J. F. Gemmeke, H. V. Hamme, B. Cranen, and L. Boves, “Compressive sensing for missing data imputation in noise robust speech recognition,” *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 2, pp. 272–287, 2010.

[80] P. Smaragdis, B. Raj, and M. Shashanka, “Missing data imputation for time-frequency representations of audio signals,” *J. Signal Process. Syst.*, vol. 65, no. 3, pp. 361–370, 2011.

[81] B. Raj, M. L. Seltzer, and R. M. Stern, "Reconstruction of missing features for robust speech recognition," *Speech Communication*, vol. 43, no. 4, pp. 275–296, 2004.

[82] B. Raj and R. M. Stern, "Missing-feature approaches in speech recognition," *IEEE Signal Process. Mag.*, vol. 22, no. 5, pp. 101–116, 2005.

[83] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016, book in preparation for MIT Press. [Online]. Available: http://www.deeplearningbook.org

[84] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, 2009.

[85] G. Cocchi and A. Uncini, "Subband neural networks prediction for on-line audio signal recovery," *IEEE Trans. Neural Netw.*, vol. 13, no. 4, pp. 867–876, 2002.

[86] A. Uncini, "Audio signal processing by neural networks," *Neurocomputing*, vol. 55, no. 3, pp. 593–625, 2003.

[87] B. J. Borgstrom and A. Alwan, "HMM-based reconstruction of unreliable spectrographic data for noise robust speech recognition," *IEEE Trans. Audio, Speech, Language Process.*, vol. 18, no. 6, pp. 1612–1623, 2010.

[88] J. Han, G. J. Mysore, and B. Pardo, "Audio imputation using the non-negative hidden markov model," in *Proc. International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2012, pp. 347–355.

[89] R. E. Turner and M. Sahani, "Time-frequency analysis as probabilistic inference," *IEEE Trans. S*, vol. 62, no. 23, pp. 6171–6183, 2014.

[90] T. Virtanen, J. F. Gemmeke, B. Raj, and P. Smaragdis, "Compositional models for audio processing: Uncovering the structure of sound mixtures," *IEEE Signal P*, vol. 32, no. 2, pp. 125–144, 2015.

[91] T. Hofmann, "Probabilistic latent semantic analysis," in *Proc. 15th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 289–296.

[92] T. Hastie, R. Tibshirani, G. Sherlock, M. Eisen, P. Brown, and D. Botstein, "Imputing missing data for gene expression arrays," 1999.

[93] J. A. Franklin, "Recurrent neural networks for music computation," *INFORMS J. Comput.*, vol. 18, no. 3, pp. 321–338, 2006.

[94] E. Marchi, F. Vesperini, F. Weninger, F. Eyben, S. Squartini, and B. Schuller, "Non-linear prediction with lstm recurrent neural networks for acoustic novelty detection," in *Proc. IEEE International Joint Conference on Neural Networks*, 2015, pp. 1–7.

[95] D. Eck and J. Schmidhuber, “Learning the long-term structure of the blues,” in *Proc. International Conference on Artificial Neural Networks*. Springer, 2002, pp. 284–289.

[96] A. Gretton and L. Györfi, “Nonparametric independence tests: Space partitioning and kernel approaches,” in *International Conference on Algorithmic Learning Theory*. Springer, 2008, pp. 183–198.

[97] M. Rudelson and R. Vershynin, “Non-asymptotic theory of random matrices: extreme singular values,” *arXiv preprint arXiv:1003.2990*, 2010.

[98] S. Goldstein, J. Lebowitz, R. Tumulka, and N. Zanghì, “Any orthonormal basis in high dimension is uniformly distributed over the sphere,” *arXiv:1406.2576*, 2015, to appear in *Annales de l’Institut Henri Poincaré*, 2016.

[99] A. Papoulis, *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Book Company, 1965.

Appendix A

Computation of the Jacobian of the loss function

In order to update the parameters of the network, we have to compute the gradient of the loss function w.r.t. these parameters. Thanks to chain rule, as seen in Section 2.2, it is possible to split the gradient into a product of different terms. The first one is always the derivative of the loss function w.r.t. the estimated output. Since in Section 2.1 we showed different choices for the loss function $\mathscr{L}$, here we present its possible derivatives: if we choose

(a) the MSE (2.3), then

$$J_{\mathscr{L}}(\hat{y}(t), y(t)) = \frac{\partial \mathscr{L}_t}{\partial \hat{y}(t)} = \frac{2}{N}\left(\hat{y}(t) - y(t)\right); \tag{A.1}$$

(b) the ABS (2.4), then

$$J_{\mathscr{L}}(\hat{y}(t), y(t)) = \frac{1}{N}\text{ones}(N, 1),$$

where $\text{ones}(N, 1)$ is a unitary column vector;

(c) the CE (2.5), then

$$J_{\mathscr{L}}(\hat{y}(t), y(t)) = \frac{1}{N}\left(\hat{y}(t) - y(t)\right),$$

which is proportional to (A.1), but with a slightly different meaning;

(d) the NLL (2.6), then

$$J_{\mathscr{L}}(\hat{y}(t), y(t)) = -y(t) + \exp(\hat{y}(t));$$

(e) the BCE (2.7)

$$J_{\mathscr{L}}(\hat{y}(t), y(t)) = \frac{1}{N}\left(\hat{y}(t) - y_n(t)\right).$$

Appendix B

Proofs for Chapter 4

B.1 Hypothesis test for the independece of inner products

Generate

$$B = \tilde{U} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_N \end{bmatrix} \tilde{V}^\top,$$

with different choices for σ_m.

Generate $U^{(m)}, V^{(m)}, m = 1, 2, 3$ as

$$\begin{aligned} Y(m) &= \Phi'(q_m) B \\ U^{(m)} \Sigma^{(m)} V^{(m)\top} &= Y(m), \end{aligned}$$

where q_m has i.i.d. elements $\mathcal{N}(0, \beta)$.

Generate for all n_1, n_2, n_3

$$z_{12}(n_1, n_2) = \left(v_{n_1}^{(1)\top} u_{n_2}^{(2)} \right) \text{ and } z_{23}(n_2, n_3) = \left(v_{n_2}^{(2)\top} u_{n_3}^{(3)} \right)$$

Estimate marginal and joint probabilities (by 1D and 2D histograms)

$$\hat{p}_{z_{12}}, \hat{p}_{z_{23}}, \hat{p}_{z_{12}, z_{23}}.$$

z_{12}, z_{23} are independent if

$$\hat{p}_{z_{12},z_{23}} = \hat{p}_{z_{12}} \cdot \hat{p}_{z_{23}}.$$

As we can see in Figure B.1, z_{12} and z_{23} are likely to be independent, since the product of the marginal probabilities (on the left side) is almost equal to the joint probability (in the middle) and their difference is close to zero (right side).

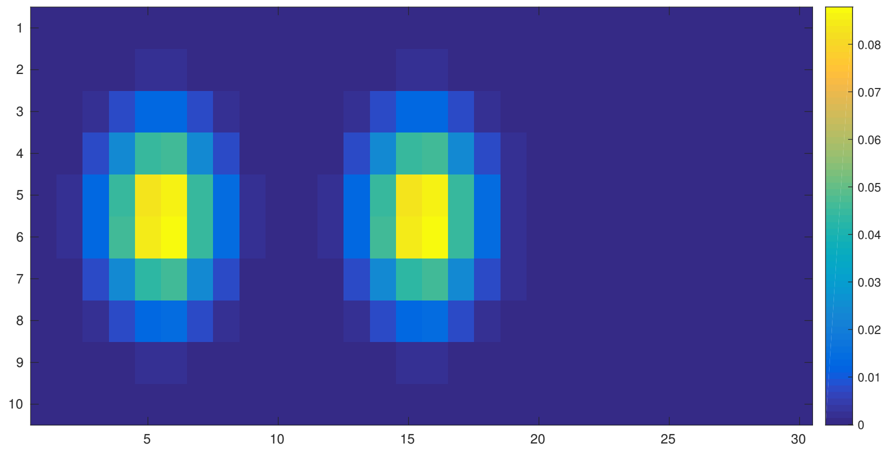


Fig. B.1 Product of the marginal probabilities $\hat{p}_{z_{12}}, \hat{p}_{z_{23}}$ text to the joint one $\hat{p}_{z_{12},z_{23}}$ and the difference between them.

Thus, we would like to test $\begin{array}{ll} H_0 & \text{‘dependence’} \\ H_1 & \text{‘independence’} \end{array}$.

We perform a Gretton and Gyorfi L1 Test [96] in order to validate this hypothesis test. If the test statistic is greater than the test threshold for level alpha test, then we reject hypothesis H_0 holds, which means the inner products are independent. On the other hand, if the test statistic is below the test threshold, we do not have enough evidence to abandon hypothesis H_0, which implies the dependence of inner products.

The results are reported in Table B.1: if we bound the singular values to a narrow interval, such as in $[0.9, 1.1]$, then the inner products are independent, while if we impose an exponential decreasing of the singular values, the dependence assumption passes the test.

Table B.1 Results of the Gretton and Gyorfi hypothesis test.

Singular values	Test Statistic	Threshold	Result
Bounded in $[0.9, 1.1]$	0.0019	0.0016	Independence
Exponential decay	0.0012	0.0016	Dependence

□

B.2 Proof of Proposition 4.1

Let M be a random $N \times N$ matrix with zero mean and s^2 variance.
Then, its expected Frobenius norm is

$$\begin{aligned} \mathbb{E}\left[\|M\|_F^2\right] &= \mathbb{E}\left[\left\|\sum_{i,j=1}^{N} M_{ij}^2\right\|_F\right] \\ &= \sum_{i,j=1}^{N} \mathbb{E}\left[\|M_{ij}^2\|_F\right] \\ &= N^2 s^2. \end{aligned} \tag{B.1}$$

On the other hand it is possible to compute expected Frobenius norm also considering the singular values of M, *i.e.* $\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_N$, as

$$\mathbb{E}\left[\|M\|_F^2\right] = \mathbb{E}\left[\sigma_1^2 + \ldots + \sigma_N^2\right]. \tag{B.2}$$

We can distinguish two cases: when the expected norm of the matrix M, *i.e.* $\mathbb{E}[\|M\|_2] = \mathbb{E}[\sigma_1]$, is minimum or when it is maximum.

When the expected norm of the matrix M is minimum, it follows that

$$\sigma_1 = \sigma_2 = \ldots = \sigma_N = \sigma.$$

This means that we can write (B.2) as

$$\begin{aligned}\mathbb{E}\left[||M||_F^2\right] &= \mathbb{E}\left[\underbrace{\sigma^2+\ldots+\sigma^2}_{N \text{ times}}\right] \\ &= N\mathbb{E}\left[\sigma^2\right] = N\mathbb{E}\left[||M||_2^2\right].\end{aligned}$$

Exploiting (B.1), we obtain

$$\mathbb{E}\left[||M||_2^2\right] = \frac{1}{N}\mathbb{E}\left[||M||_F^2\right] = \frac{1}{N}N^2s^2 = Ns^2.$$

This is equivalent to

$$\mathbb{E}\left[||M||_2\right] = s\sqrt{N}, \tag{B.3}$$

which is the lower bound of (4.9) in Proposition 4.1.

On the other end, the expected norm of the matrix M is maximum when

$$\sigma_1 = \sigma, \quad \sigma_2 = \ldots = \sigma_N = 0.$$

This means that we can write (B.2) as

$$\mathbb{E}\left[||M||_F^2\right] = \mathbb{E}\left[\sigma^2\right] = \mathbb{E}\left[||M||_2^2\right]$$

and, exploiting (B.1), we obtain

$$\mathbb{E}\left[||M||_2^2\right] = \frac{1}{N}\mathbb{E}\left[||M||_F^2\right] = N^2s^2.$$

This is equivalent to

$$\mathbb{E}\left[||M||_2\right] = sN, \tag{B.4}$$

which is the upper bound of (4.9) in Proposition 4.1.

In Figure B.2 we represent the expected norm of a random matrix M, whose elements have zero mean and $s^2 = 1$ variance (blue line), and which is bounded above by (B.4) and below by (B.3) (red and yellow lines, respectively).

Moreover, in the case that the elements of M are i.i.d. $\mathcal{N}(0,s^2)$, then following Theorem 2.6 in [97], it is possible to find a tighter upper bound for the expected norm of M

$$\mathbb{E}\left[\|M\|_2^2\right] \le 2s\sqrt{N},$$

as shown in Figure B.3, where this upper bound is represented in red, while the expected norm of M in blue and the lower bound in yellow as in Figure B.2.

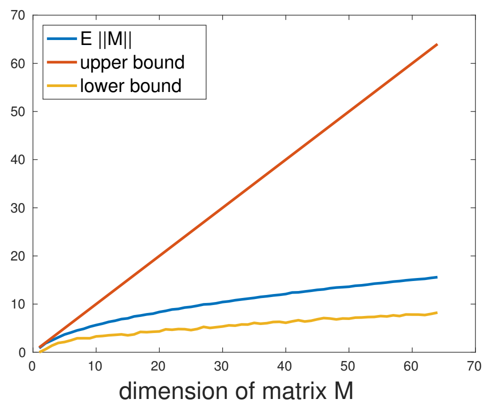


Fig. B.2 The expected norm a random matrix is bounded.

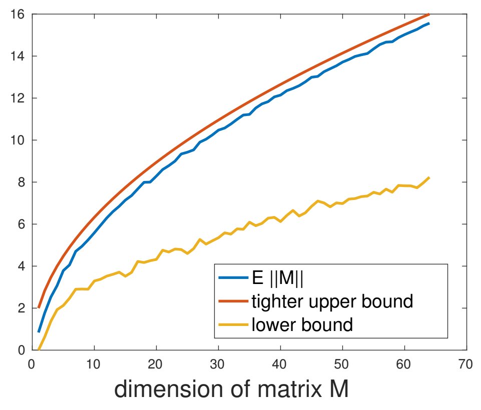


Fig. B.3 Tighter upper bound for the expected norm of a random matrix with elements i.i.d. $\mathcal{N}(0,s^2)$.

$\square$

B.3 Expectation of $v^\top u$

Lemma B.1. *Let U and V be the right and left singular matrices of two independent random matrices of $N \times N$ elements. Let u and v be columns in U and V, respectively. Let Z be their inner product,*

$$Z = u^\top v.$$

Then,

$$\mathbb{E}[Z] = 0.$$

Proof. Since ${V^{(m)}}^\top U^{(m+1)}$ has orthonormal rows and columns, then it is uniformly distributed over the sphere [98], which means that

$$\begin{aligned}
\mathbb{E}\left[v^\top u\right] &= \lim_{M\to\infty} \frac{1}{M} \sum_{m=1}^{M} \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} {v_i^{(m)}}^\top u_j^{(m+1)} \\
&= \frac{1}{N^2} \lim_{M\to\infty} \frac{1}{M} \sum_{m=1}^{M} \mathbf{1}^\top {V^{(m)}}^\top U^{(m+1)} \mathbf{1} \\
&= 0.
\end{aligned}$$

□

B.4 Variance of $v^\top u$

Lemma B.2. *Let U and V be the right and left singular matrices of two independent random matrices of $N \times N$ elements. Let u and v be columns in U and V, respectively. Let Z be their inner product,*

$$Z = u^\top v.$$

Then,

$$Var(Z) = \frac{1}{N}.$$

Proof.

$$\begin{aligned}
\mathrm{Var}(Z) &= \mathbb{E}\left[(v^\top u)^2\right] - \mathbb{E}\left[(v^\top u)\right]^2 \\
&= \mathbb{E}\left[(v^\top u)^2\right],
\end{aligned}$$

since for Lemma B.1 $\mathbb{E}[v^\top u] = 0$.

Ergodicity, which means that averages over column indices ('time') in the singular matrices equal expected values [99], in the variance of inner products implies that

$$\mathbb{E}\left[\left(\mathbf{v}^{(m)^\top}\mathbf{u}^{(m+1)}\right)^2\right] = \lim_{M\to\infty}\frac{1}{MN^2}\sum_{m=1}^{M}\sum_{k=1}^{N}\sum_{l=1}^{N}\left(v_k^{(m)^\top}u_l^{(m+1)}\right)^2.$$

In this case, the cross-correlation can be estimated by considering only two consecutive matrices (*i.e.* one index m), since

$$\frac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}\left(v_k^{(m)^\top}u_l^{(m+1)}\right)^2 = \frac{1}{N^2}\|V^{(m)^\top}U^{(m+1)}\|_F^2 = \frac{1}{N},$$

since the squared Frobenius norm of any product $V^\top U$ is equal to the sum of squared singular values, which are all 1. We may therefore state that the covariance of of inner products is

$$\mathbb{E}\left[\left(\mathbf{v}^\top\mathbf{u}\right)^2\right] = \frac{1}{N},$$

where we have omitted the indices that denote position in the sequence of matrices. Thus, the variance of the inner products is

$$\begin{aligned}\mathrm{Var}(v^\top u) = \mathbb{E}\left[(v^\top u)^2\right] &= \lim_{M\to\infty}\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(v_i^{(m)^\top}u_j^{(m+1)}\right)^2\\ &= \lim_{M\to\infty}\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N^2}\left\|V^{(m)^\top}U^{(m+1)}\right\|_F^2\\ &= \lim_{M\to\infty}\frac{1}{M}M\frac{1}{N^2}N\\ &= \frac{1}{N}.\end{aligned}$$

□

B.5 Expectation of $v^\top u v^\top u$

Lemma B.3. *Let U and V be the right and left singular matrices of two independent random matrices of $N\times N$ elements. Let $u_{n_{i+1}}^{(i+1)}, u_{m_{i+1}}^{(i+1)}$ and $v_{n_i}^{(i)}, v_{m_i}^{(i)}$ be columns in U*

and V, respectively. Let $z_i(n_i,n_{i+1}), z_i(m_i,m_{i+1})$ be their inner products,

$$z_i(n_i,n_{i+1}) = {v_{n_i}^{(i)}}^\top u_{n_{i+1}}^{(i+1)}$$
$$z_i(m_i,m_{i+1}) = {v_{m_i}^{(i)}}^\top u_{m_{i+1}}^{(i+1)}.$$

Then,

$$\mathbb{E}[z_i(n_i,n_{i+1})z_i(m_i,m_{i+1})] = \delta_{n_i,m_i}\delta_{n_{i+1},m_{i+1}}\left(\frac{1}{N}\right),$$

where δ is the Kronecker delta function.

Proof. In order to simplify the notation, let us write

$$\mathbb{E}\left[{v_{n_i}^{(i)}}^\top u_{n_{i+1}}^{(i+1)} {v_{m_i}^{(i)}}^\top u_{m_{i+1}}^{(i+1)}\right] = \mathbb{E}\left[v_i^\top u_j v_k^\top u_l\right],$$

so that v_i, v_k are columns in V, while u_j, u_l are columns in U. We can define four cases:

(A) if $i = k$ and $j = l$;

(B) if $i \neq k$ and $j \neq l$;

(C) if $i = k$ and $j \neq l$;

(D) if $i \neq k$ and $j = l$, which actually is equivalent to (C).

For case (A), *i.e.* $i = k, j = l$, as shown in Lemma B.2,

$$E\left[v_i^\top u_j v_i^\top u_j\right] = E\left[\left(v_i^\top u_j\right)^2\right] = \frac{1}{N}. \tag{B.5}$$

For case (B), *i.e.* $i \neq k, j \neq l$, we will write permutations $k = q(i)$, $l = p(j)$ (matrix form Q, P) and, thanks to [98], we obtain

$$\begin{aligned}
\mathbb{E}\left[v_i^\top u_j v_k^\top u_l\right] &= \frac{1}{N^2}\mathbb{E}_{q,p}\left[\sum_i \sum_j v_i^\top u_j u_{p(j)}{}^\top v_{q(i)}\right] \\
&= \frac{1}{N^2}\mathbb{E}_{q,p}\left[\sum_i v_i^\top \left(\sum_j u_j u_{p(j)}{}^\top\right) v_{q(i)}\right] \\
&= \frac{1}{N^2}\mathbb{E}_{q,P}\left[\sum_i v_i^\top U (UP)^\top v_{q(i)}\right] \\
&= \frac{1}{N^2}\mathbb{E}_{Q,P}\left[\mathbf{1}^\top \left(V^\top U P^\top U^\top V Q\right)\mathbf{1}\right] \\
&= 0. \qquad \text{(B.6)}
\end{aligned}$$

For case (C), *i.e.* $i = k, j \neq l$, or equivalently for case (D), we find

$$\begin{aligned}
\mathbb{E}\left[v_i^\top u_j v_i^\top u_l\right] &= \frac{1}{(N-1)N^2}\mathbb{E}\left[\sum_i \sum_k \sum_{j \neq l} v_i^\top u_j v_i^\top u_l\right] \\
&= \frac{1}{(N-1)N^2}\mathbb{E}\left[\sum_j \sum_{j \neq l} u_j^\top \left(\sum_i v_i v_i^\top\right) u_l\right] \\
&= \frac{1}{(N-1)N^2}\mathbb{E}\left[\sum_j \sum_{j \neq l} u_j^\top V V^\top u_l\right] \\
&= \frac{1}{(N-1)N^2}\mathbb{E}\left[\sum_j \sum_{j \neq l} u_j^\top u_l\right] \\
&= 0. \qquad \text{(B.7)}
\end{aligned}$$

Combining Equations (B.5),(B.6) and (B.7), and exploiting the Kronecker function δ, we obtain

$$\mathbb{E}\left[v_i^\top u_j v_k^\top u_l\right] = \delta_{i,k}\delta_{j,l}\left(\frac{1}{N}\right).$$

□

B.6 Proof of Lemma 4.1

The cost function is $\mathscr{K}$ in Equation 4.18. Its gradient is

$$\frac{\partial \mathscr{K}(M)}{\partial M} = \sum_{\sigma_n < \rho} (\sigma_n - \rho) \frac{\partial \sigma_n}{\partial M}.$$

Following Papadopoulo and Lourakis [67], the derivative of the nth singular value with respect to the (i, j)th element in B is

$$\frac{\partial \sigma_n}{\partial M_{ij}} = U_{in} V_{jn}. \tag{B.8}$$

The derivative of σ_n with respect to all elements in B can therefore be written as the outer product of the nth column of U and the nth column of V, such that Equation B.6 becomes

$$\frac{\partial \mathscr{K}(M)}{\partial M} = \sum_{\sigma_n < \rho} (\sigma_n - \rho) U_{:,n} V_{:,n}^{\top}.$$

□

B.7 Proof of Proposition 4.2

Essentially, the proof is about finding the derivative of a function like $g(f(z)z)$ with respect to z, which is quite straightforward. The proof is somewhat complicated by the fact that the involved quantities are matrices and vectors.

The derivative of the cost function with respect to the element B_{ij} is

$$\frac{\partial \mathscr{K}}{\partial B_{ij}} = \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{\partial \mathscr{K}}{\partial J_{mn}} \frac{\partial J_{mn}}{\partial B_{ij}}. \tag{B.9}$$

This double sum is reduced to a single sum, as will be seen by considering the derivatives of J. We will write the mth rows of A and B as A_m, B_m such that the mth row of J is

$$J_m = \phi_h'(A_m x + B_m h + b_m) B_m,$$

where ϕ_h is the scalar non-linearity. The derivative of an element in J with respect to an element in B is then

$$\frac{\partial J_{mn}}{\partial B_{ij}} = \frac{\partial}{\partial B_{ij}} \phi'(A_m x + B_m h + b_m) B_{mn}.$$

If the two elements are in different rows ($m \neq i$), then

$$\frac{\partial}{\partial B_{ij}} \phi'(A_m x + B_m h + b_m) B_{mn} = 0.$$

We may therefore write

$$\frac{\partial J_{mn}}{\partial B_{ij}} = \delta_{mi} \frac{\partial J_{mn}}{\partial B_{ij}} \tag{B.10}$$

and the double sum in Equation (B.9) becomes a single sum

$$\frac{\partial \mathscr{K}}{\partial B_{ij}} = \sum_{m=1}^{N} \delta_{mi} \sum_{n=1}^{N} \frac{\partial \mathscr{K}}{\partial J_{mn}} \frac{\partial J_{mn}}{\partial B_{ij}} = \sum_{n=1}^{N} \frac{\partial \mathscr{K}}{\partial J_{in}} \frac{\partial J_{in}}{\partial B_{ij}}. \tag{B.11}$$

Combining Equation (4.19) (the cost as a function of the SVD factors of J) and Equation (B.11),

$$\frac{\partial \mathscr{K}}{\partial B_{ij}} = U_i \nu (\Sigma - \rho I) \sum_{n=1}^{N} V_n^\top \frac{\partial J_{in}}{\partial B_{ij}},$$

which is Equation (4.20). The partial derivatives w.r.t. A_{ij} and b_i follow the same scheme.

□