Equivalence Results between Feedforward and Recurrent Neural Networks for Sequences*

Alessandro Sperduti
Department of Mathematics
University of Padova, Italy
Email: sperduti@math.unipd.it

Abstract

In the context of sequence processing, we study the relationship between single-layer feedforward neural networks, that have simultaneous access to all items composing a sequence, and single-layer recurrent neural networks which access information one step at a time. We treat both linear and nonlinear networks, describing a constructive procedure, based on linear autoencoders for sequences, that given a feedforward neural network shows how to define a recurrent neural network that implements the same function in time. Upper bounds on the required number of hidden units for the recurrent network as a function of some features of the feedforward network are given. By separating the functional from the memory component, the proposed procedure suggests new efficient learning as well as interpretation procedures for recurrent neural networks.

1 Introduction

Learning on sequential data has always been a hot topic since many are the application domains where this skill could be applied, e.g. natural language processing, bioinformatics, video surveillance, time series prediction, robotics, etc. There are many different approaches that can be used to perform learning on sequential data, ranging from deterministic to probabilistic ones. In this paper, we focus on Recurrent Neural Networks (RNN) (see for example [Kremer, 2001]), which constitute a powerful computational tool for sequences modelling and prediction, as recently demonstrated in [Boulanger-Lewandowski *et al.*, 2012], where many different approaches for sequence learning have been compared on a prediction task involving polyphonic music. Training a RNN, however, is hard, the main problem being the well known vanishing gradient problem which makes difficult to learn long-term dependencies [Bengio *et al.*, 1994]. Partial solutions to this problem have been suggested in literature, such as the introduction of LSTM networks [Hochreiter and Schmidhuber, 1996], more efficient training procedures, such as Hessian Free Optimization [Martens and Sutskever, 2011], and smart weight initialisation procedures, as stressed, for example, in [Sutskever *et al.*, 2013], and demonstrated for RNN in [Pasa and Sperduti, 2014]. Notwithstanding these progresses, efficient and effective training of RNNs is still an open problem.

Our position about this state of the art is that current learning algorithms for general RNN architectures are not based on a good understanding of how information is processed by a RNN. Specifically, we believe that efficient and effective learning algorithms should address in different ways the functional and the memory components of a RNN. Thus a better understanding of how these two components interact each other inside a RNN is needed. In this paper, we report on some results that we believe, in part, elucidate the interplay between these two components. The approach we have pursued can be explained as follows. We consider a set of bounded sequences for which a prediction task at each time step should be carried on. We pretend to have enough computational resources to be able to fully unroll these sequences and to exploit a feedforward neural network with as many input units as the number of components belonging to the longest sequence. We consider the set of functions that such network can compute on the given set of sequences and show how it is possibile to design a RNN able to reproduce the same set of functions. The feedforward network does not need to have a memory component since the (sub)sequences presented in input are fully unrolled. Thus, the feedforward network can be understood as a system with a pure functional component. However, when we construct the "equivalent" RNN, we need to introduce a memory component, since the RNN can only read one item of the sequence at a time. The memory component we introduce is based on linear autoencoders for sequences, i.e., a special case of the autoencoders for structured data introduced in [Sperduti, 2006; Micheli and Sperduti, 2007; Sperduti, 2007]. Specifically, an autoencoder is used to keep memory of the temporary results computed by the functional component of the feedforward network. Additional considerations of properties of sigmoidal functions will also allow the substitution of linear units with sigmoidal units, incurring into an arbitrarily small error in reproducing the functions computed by the original feedforward network. Overall, we get a practical procedure that, given a feedforward neural network processing unrolled

*This work was supported by the University of Padova under the strategic project BIOINFOGEN.

sequences, is able to return an "equivalent" RNN processing the sequences one item at a time. Since we use autoencoders, the procedure also returns RNNs with a number of hidden units which is not extremely large. In fact, we try to give upper bounds on the number of hidden units of the resulting RNN.

We believe that the main contribution of the paper is to start a research path towards a better understanding of the weight space of RNNs, so to allow, in a hopefully near future, the design of more efficient and effective learning algorithms for RNNs.

2 Background

In the following, we introduce background knowledge on linear autoencoders for sequences, and feedforward and recurrent neural networks with a single layer.

2.1 Linear Autoencoders for sequences

In [Sperduti, 2006; Micheli and Sperduti, 2007; Sperduti, 2007], a closed form solution for linear autoencoders for structured data has been proposed. Specifically, given a set of temporal sequences of input vectors $\{\mathbf{s}^q \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \ldots, \mathbf{x}_{l_q}^q) | q = 1, \ldots, n,\ \mathbf{x}_j^q \in \mathbb{R}^a\}$, a linear autoencoder can be defined by considering the coupled linear dynamical systems

$$\mathbf{y}_t = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{y}_{t-1} \tag{1}$$

$$\begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_{t-1} \end{bmatrix} = \mathbf{C}\mathbf{y}_t \tag{2}$$

and looking for the smallest possibile matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ able to satisfy eq. (1) and eq. (2).

The solution proposed in [Sperduti, 2006] hinges on the factorisation of the matrix $\mathbf{Y}$ collecting as rows the state vectors of the linear system described by eq. (1). In fact, assuming as initial state $\mathbf{y}_0 = \mathbf{0}$ (the null vector), and a single sequence for the sake of presentation, $\mathbf{Y}$ can be factorized as

$$\underbrace{\begin{bmatrix} \mathbf{y}_1^\mathsf{T} \\ \mathbf{y}_2^\mathsf{T} \\ \mathbf{y}_3^\mathsf{T} \\ \vdots \\ \mathbf{y}_l^\mathsf{T} \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} \mathbf{x}_1^\mathsf{T} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{x}_2^\mathsf{T} & \mathbf{x}_1^\mathsf{T} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{x}_3^\mathsf{T} & \mathbf{x}_2^\mathsf{T} & \mathbf{x}_1^\mathsf{T} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_l^\mathsf{T} & \mathbf{x}_{l-1}^\mathsf{T} & \cdots & \mathbf{x}_2^\mathsf{T} & \mathbf{x}_1^\mathsf{T} \end{bmatrix}}_{\boldsymbol{\Xi}} \underbrace{\begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{A}^\mathsf{T}\mathbf{B}^\mathsf{T} \\ \mathbf{A}^\mathsf{T}{\mathbf{B}^2}^\mathsf{T} \\ \vdots \\ \mathbf{A}^\mathsf{T}{\mathbf{B}^{l-1}}^\mathsf{T} \end{bmatrix}}_{\boldsymbol{\Omega}}$$

where, given $s = al$, $\boldsymbol{\Xi} \in \mathbb{R}^{l \times s}$ is the data matrix collecting all the (inverted) input subsequences (including the whole sequence) as rows, and $\boldsymbol{\Omega}$ is the parameter matrix of the dynamical system. By considering the thin svd decomposition of $\boldsymbol{\Xi} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{U}^\mathsf{T}$, the state space can be confined into a subspace of dimension $\rho = rank(\boldsymbol{\Xi})$ if matrices $\mathbf{A}$ and $\mathbf{B}$ are chosen so to have $\mathbf{U}^\mathsf{T}\boldsymbol{\Omega} = \mathbf{I}$, i.e. the identity matrix. By exploiting the fact that $\mathbf{U}\mathbf{U}^\mathsf{T} = \mathbf{I}$, this condition turns into $\boldsymbol{\Omega} = \mathbf{U}$. Using matrices

$$\mathbf{P}_{a,s} \equiv \begin{bmatrix} \mathbf{I}_{a\times a} \\ \mathbf{0}_{(s-a)\times a} \end{bmatrix}, \quad \mathbf{R}_{a,s} \equiv \begin{bmatrix} \mathbf{0}_{a\times(s-a)} & \mathbf{0}_{a\times a} \\ \mathbf{I}_{(s-a)\times(s-a)} & \mathbf{0}_{(s-a)\times a} \end{bmatrix},$$

where $\mathbf{P}_{a,s}$ is an operator that annihilates all except the first a components of vectors multiplied to its left and $\mathbf{R}_{a,s}$ is an operator that shifts down of a positions the components of vectors multiplied to its right replacing the first a components with zeros, it is not difficult to verify that $\mathbf{A} \equiv \mathbf{U}^\mathsf{T}\mathbf{P}_{a,s} \in \mathbb{R}^{\rho\times a}$ and $\mathbf{B} \equiv \mathbf{U}^\mathsf{T}\mathbf{R}_{a,s}\mathbf{U} \in \mathbb{R}^{\rho\times\rho}$ satisfy equation $\boldsymbol{\Omega} = \mathbf{U}$. It is also not difficult to recognise that, since $\mathbf{Y} = \mathbf{V}\boldsymbol{\Lambda}$, it is possibile to fully reconstruct the original data $\boldsymbol{\Xi}$ by computing $\mathbf{Y}\mathbf{U}^\mathsf{T} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{U}^\mathsf{T} = \boldsymbol{\Xi}$, which can be achieved by running the dynamical system

$$\begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_{t-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{B}^\mathsf{T} \end{bmatrix} \mathbf{y}_t$$

starting from $\mathbf{y}_l$, i.e. $\begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{B}^\mathsf{T} \end{bmatrix}$ is the matrix $\mathbf{C}$ defined in eq. (2).

Finally, when considering a set of sequences, the same result can be obtained by stacking the data matrix of each sequence and filling up with zeros where needed. E.g., given a sequence $\mathbf{s}_1$ of length 3 and a sequence $\mathbf{s}_2$ of length 2, the full data matrix can be defined as $\boldsymbol{\Xi} = \left[\begin{array}{cc} \boldsymbol{\Xi}_{\mathbf{s}_1} & \\ \hline \boldsymbol{\Xi}_{\mathbf{s}_2} & \mathbf{0}_{a\times a} \end{array}\right]$.

2.2 Feedforward and Recurrent Neural Networks

In this paper, we mainly work with feedforward and recurrent neural networks with a single hidden layer. Specifically, with no loss in generalisation, we consider feedforward neural networks $\mathcal{N}_f$ with a single output described by the following equation

$$o_{\mathcal{N}_f}(\mathbf{x}) = \sigma\left(\sum_{h=0}^{H} w_h^o \underbrace{\sigma(\mathbf{w}_h^\mathsf{T}\mathbf{x})}_{\mathbf{h}_h(\mathbf{x})}\right),$$

where $\mathbf{x}$ is the input vector with a component set to 1 to account for the bias term, $\sigma(\cdot)$ is the hyperbolic tangent function, H is the number of hidden units of the network, $\mathbf{w}_h$ are the weight vectors of the hidden units, and w_h^o are the components of the output weight vector, where w_0^o is associated to the constant 1 (i.e., here we assume that $\mathbf{h}_0(\mathbf{x}) = 1$) to account for the bias term.

Here we also consider a linear version of the feedforward network (which, because of linearity, reduces to a single linear unit), i.e. $\mathcal{N}_L$ whose output is described by

$$o_{\mathcal{N}_L}(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x}. \tag{3}$$

The output of the recurrent neural network $\mathcal{N}_{rec}$ considered here is a nonlinear version of eq. (1) and eq. (2), i.e.

$$o_{\mathcal{N}_{rec}}(\mathbf{x}_t) = \sigma(\mathbf{c}^\mathsf{T}\mathbf{y}_t) \tag{4}$$

$$\mathbf{y}_t = \boldsymbol{\sigma}(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{y}_{t-1}) \tag{5}$$

where we assume $\mathbf{y}_0 = \mathbf{0}$, $\boldsymbol{\sigma}(\cdot)$ is the function which applies component-wise the hyperbolic tangent to the input vector, $\mathbf{A} \in \mathbb{R}^{H\times(a+1)}$ is the input-to-hidden weight matrix (we assume a constant input to 1 to account for the bias), $\mathbf{B} \in \mathbb{R}^{H\times H}$ is the hidden-to-hidden weight matrix, $\mathbf{c} \in \mathbb{R}^{H+1}$ is the hidden-to-output weight vector (again, we assume a constant input to 1 to account for the bias).

As a final remark, we observe that the hyperbolic tangent function $\sigma(z)$ is almost linear around 0. In fact, if we consider the function $\beta\sigma(\frac{z}{\beta})$, given $z \in [-b, b]$ and an error value ϵ, there exists a value $\beta^\star$ such that $|\beta^\star\sigma(\frac{z}{\beta^\star}) - z| < \epsilon$. This fact can be understood recalling that the first terms of the Taylor expansion of the hyperbolic tangent function are $z - \frac{z^3}{3} + \frac{2z^5}{15} - \cdots$, which for $\beta\sigma(\frac{z}{\beta})$ becomes $z - \frac{z^3}{3\beta^2} + \frac{2z^5}{15\beta^4} - \cdots$.

We will use this fact for some of our results on nonlinear networks.

3 Theoretical Results

We start by considering the task to learn a function $\mathcal{F}(\cdot)$ from multivariate input sequences of bounded length to desired output values, in the following sense: given a training set $\mathcal{T} = \{(\mathbf{s}^q, \mathbf{d}^q) | q = 1, \ldots, n,\ \mathbf{s}^q \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \ldots, \mathbf{x}_{l_q}^q),\ \mathbf{d}^q \equiv (d_1^q, d_2^q, \ldots, d_{l_q}^q),\ \mathbf{x}_t^q \in \mathbb{R}^a,\ d_t^q \in \mathbb{R}\}$, we expect to learn a function $\mathcal{F}(\cdot)$ such that $\forall q, t\ \mathcal{F}(\mathbf{s}^q[1,t]) = d_t^q$, where $\mathbf{s}^q[1,t] \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \ldots, \mathbf{x}_t^q)$. Actually, here we are not interested into generalisation issues, but into computational issues, i.e., we would like to seek an answer to the following question: what is the relationship between an implementation of $\mathcal{F}(\cdot)$ by a feedforward neural network able to directly access the sequence items read up to time t and an implementation of $\mathcal{F}(\cdot)$ by a recurrent neural network that can directly access only the current item at time t (after having read all the previous items of the sequence). An answer to this question would allow to gain a better understanding of how a recurrent neural network is able to manage the memory component which is needed to store the (relevant features of the) sequence items which are read one at a time, jointly with the functional component needed to reproduce the desired output value. This improved understanding could be a good starting point to device more efficient and effective learning algorithms for recurrent neural networks.

We start our investigation by first considering linear functions. After that, we discuss how the findings obtained with linear functions can be extended to nonlinear functions.

3.1 Linear Functions

Here we consider linear functions $\mathcal{F}_{\mathbf{w}}(\cdot)$ which can be implemented by a linear network $\mathcal{N}_L$ with weight vector $\mathbf{w}$, as described by eq. (3). Starting from $\mathcal{N}_L$ we show how to build a linear dynamical system (a special case of the one described by eq. (1) and eq. (2)) which computes the very same function. Specifically, let $l = \max_{i\in\{1,\ldots,q\}} l_i$, $m = \arg\max_{i\in\{1,\ldots,q\}} l_i$. We consider $\mathcal{N}_L$ to have an input of size al, i.e. $\mathbf{w} \in \mathbb{R}^{al}$. Given an input sequence $\mathbf{s}^q \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \cdots, \mathbf{x}_{l_q-1}^q, \mathbf{x}_{l_q}^q)$, the linear network $\mathcal{N}_L$ takes as input the vectors obtained by concatenating (in reverse order) the first i items of the sequence, padding the obtained vector with zeros, so to obtain a vector in $\mathbb{R}^{al}$. By collecting these vectors as rows of a matrix we obtain

$$\Xi_q = \begin{bmatrix} \mathbf{x}_1^{q^\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{x}_2^{q^\mathsf{T}} & \mathbf{x}_1^{q^\mathsf{T}} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{x}_3^{q^\mathsf{T}} & \mathbf{x}_2^{q^\mathsf{T}} & \mathbf{x}_1^{q^\mathsf{T}} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_{l_q}^{q^\mathsf{T}} & \mathbf{x}_{l_q-1}^{q^\mathsf{T}} & \mathbf{x}_{l_q-2}^{q^\mathsf{T}} & \cdots & \mathbf{x}_2^{q^\mathsf{T}} & \mathbf{x}_1^{q^\mathsf{T}} & \underbrace{\mathbf{0} \cdots \mathbf{0}}_{l-l_q} \end{bmatrix}$$

where $\mathbf{0} \in \mathbb{R}^a$ is a row vector of zeros. The reason to have the sequence items in reverse order is just to maintain the same convention adopted for the autoencoder described at the beginning of the paper. It is like assuming that new information is always arriving from the left side. If we consider all the sequences in $\mathcal{T}$, we can describe the output vector $\mathbf{d}$ collecting all the outputs by $\mathcal{N}_L$ as

$$\underbrace{\begin{bmatrix} \Xi_1 \\ \Xi_2 \\ \vdots \\ \Xi_l \\ \vdots \\ \Xi_n \end{bmatrix}}_{\Xi} \underbrace{\begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_m \\ \vdots \\ \mathbf{w}_l \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} d_1^1 \\ d_2^1 \\ \vdots \\ d_1^2 \\ d_2^2 \\ \vdots \\ d_{l_{n-1}}^n \\ d_{l_n}^n \end{bmatrix}}_{\mathbf{d}}$$

where we have made explicit that the weight vector $\mathbf{w}$ can be decomposed into l subvectors $\mathbf{w}_i$ of size a, i.e. $\forall i\ \mathbf{w}_i \in \mathbb{R}^a$: $\mathbf{w}_1$ is used to process the last read item of the sequence, $\mathbf{w}_2$ to process the second-last read item, and so on. Thus, the tth read item of the jth sequence can be computed as $\mathcal{F}_{\mathbf{w}}(\mathbf{s}^j[1,t]) = d_t^j = \sum_{k=1}^t \mathbf{x}_k^{j^\mathsf{T}} \mathbf{w}_{t-k+1}$. We start from this equation to show how to define a linear dynamical system able to reproduce the same output. In addition, we require the linear dynamical system to have the smallest state space as possibile.

Let consider the vectors belonging to the canonical basis of $\mathbb{R}^l$, i.e. $\boldsymbol{e}^i \in \mathbb{R}^l$, where $\boldsymbol{e}_j^i = \delta_{i,j}$ and $\delta_{i,j}$ is the Kronecker delta function. We can use these vectors to rewrite d_t^j as follows

$$\begin{aligned} d_t^j &= \sum_{k=1}^t \mathbf{x}_k^{j^\mathsf{T}} \underbrace{[\mathbf{w}_1 \mathbf{w}_2 \cdots \mathbf{w}_l]}_{\mathbf{W}} \boldsymbol{e}^{t-k+1} = \sum_{k=1}^t \mathbf{x}_k^{j^\mathsf{T}} \mathbf{W} \boldsymbol{e}^{t-k+1} \\ &= \sum_{k=1}^t \underbrace{\mathbf{x}_k^{j^\mathsf{T}} \mathfrak{B}}_{\boldsymbol{\xi}_k^{j^\mathsf{T}}} \boldsymbol{\Gamma} \boldsymbol{e}^{t-k+1} = \sum_{k=1}^t \boldsymbol{\xi}_k^{j^\mathsf{T}} \boldsymbol{\Gamma}_{t-k+1}, \end{aligned}$$

where $\mathfrak{B} \in \mathbb{R}^{a\times r}$, $r = rank(\mathbf{W}) \leq a$, is a matrix whose columns constitute a basis for the subspace spanned by the columns of $\mathbf{W}$, and $\boldsymbol{\Gamma} \in \mathbb{R}^{r\times l}$ is such that $\mathbf{W} = \mathfrak{B}\boldsymbol{\Gamma}$. By defining $\boldsymbol{z}_t^j = [\boldsymbol{\xi}_t^{j^\mathsf{T}}, \boldsymbol{\xi}_{t-1}^{j^\mathsf{T}}, \cdots, \boldsymbol{\xi}_1^{j^\mathsf{T}}]$, we can introduce, for

$j \in \{1,..,n\}$, matrices

$$\mathbf{Z}_j = \begin{bmatrix} \boldsymbol{z}_1^j & \mathbf{0}_{1\times(l-1)a} \\ \boldsymbol{z}_2^j & \mathbf{0}_{1\times(l-2)a} \\ \vdots & \vdots \\ \boldsymbol{z}_{l_j}^j & \mathbf{0}_{1\times(l-l_j)a} \end{bmatrix}, \text{ and } \mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \\ \vdots \\ \mathbf{Z}_n \end{bmatrix},$$

such that $\mathbf{d} = \mathbf{Z}\ vec(\mathbf{\Gamma})$, where $vec(\cdot)$ is the vectorization function which, given a matrix in input, returns a column vector obtained by stacking the columns of the matrix on top of one another. We can now observe that $\mathbf{Z}$ has the same structure of $\mathbf{\Xi}$, so given the thin svd decomposition $\mathbf{Z} = \mathbf{V}\mathbf{\Lambda}\mathbf{U}^\mathsf{T}$ we can exploit the encoding part of the linear autoencoder given by $\mathbf{A} \equiv \mathbf{U}^\mathsf{T}\mathbf{P}_{r,rl} \in \mathbb{R}^{\rho\times r}$ and $\mathbf{B} \equiv \mathbf{U}^\mathsf{T}\mathbf{R}_{r,rl}\mathbf{U} \in \mathbb{R}^{\rho\times\rho}$, where $\rho = rank(\mathbf{Z})$, to define the following equivalent (over $\mathcal{T}$) linear dynamical system $\mathcal{N}_{L_{rec}}$:

$$\mathbf{y}_t = \mathbf{A}\mathfrak{B}^\mathsf{T}\mathbf{x}_t + \mathbf{B}\mathbf{y}_{t-1}$$
$$o_{\mathcal{N}_{L_{rec}}}(\mathbf{x}_t) = vec(\mathbf{\Gamma})^\mathsf{T}\mathbf{U}\mathbf{y}_t$$

where $\mathbf{y}_0 = \mathbf{0}$, i.e. the zero vector of dimension ρ. It is not difficult to verify that, by construction, $\forall j, t$

$$o_{\mathcal{N}_{L_{rec}}}(\mathbf{x}_t^j) = d_t^j = \mathcal{F}_\mathbf{w}(\mathbf{s}^j[1,t]) = o_{\mathcal{N}_L}([\mathbf{x}_t^j,\ldots,\mathbf{x}_1^j,\ldots,\mathbf{0}]).$$

Thus, we have demonstrated that, given a specific $\mathcal{T}$ and $\mathbf{w}$, it is possibile to build an *equivalent* (over $\mathcal{T}$) linear dynamical system with state space dimension equal to $\rho = rank(\mathbf{Z})$. Notice that, since $vec(\mathbf{\Gamma})^\mathsf{T}\mathbf{U} \in \mathbb{R}^\rho$, the total number of parameters of the dynamical system is $\rho(r+\rho+1)$, instead of the al parameters compounding $\mathbf{w}$. Of course, it is trivial to state that $\rho \leq \min\{\sum_{i=1}^n l_i, rl\}$, i.e. the minimum between the number of rows and the number of columns of $\mathbf{Z}$. To ease future discussion, let $N = \sum_{i=1}^n l_i$.

In the following, we argue that, looking at properties of $\mathbf{w}$ and/or to structural features of the proposed construction, it may be possible to define an equivalent linear dynamical system with state space dimension smaller than ρ. In order to discuss this issue, we need to define two specific properties of $\mathbf{w}$:

Definition 3.1 (rank and timespan of $\mathbf{w}$ over $\mathcal{T}$). *We say that* $\mathbf{w}$ *has rank* $r^\star$ *if* $r^\star = rank(\mathbf{W})$ *and timespan* $t^\star$ *if* $\mathbf{w}_{t^\star} \neq \mathbf{0}$ *and* $\forall t > t^\star, \mathbf{w}_t = \mathbf{0}$.

We start by observing that if $t^\star < l$, then

$$d_t^j = \sum_{k=\max\{1,t-t^\star+1\}}^{t} \boldsymbol{\xi}_k^{j\mathsf{T}}\mathbf{\Gamma}_{t-k+1}.$$

Thus we can define new vectors

$$\boldsymbol{z}_{t,t^\star}^j = [\boldsymbol{\xi}_t^{j\mathsf{T}}, \boldsymbol{\xi}_{t-1}^{j\mathsf{T}}, \cdots, \boldsymbol{\xi}_{\max\{1,t-t^\star+1\}}^{j\mathsf{T}}]$$

to use for the construction of matrices $\mathbf{Z}_j^{t^\star}$ and $\mathbf{Z}^{t^\star}$, with the feature that the number of nonzero columns of $\mathbf{Z}^{t^\star}$ is less or equal to $r^\star t^\star$, and in general $\rho^{t^\star} = rank(\mathbf{Z}^{t^\star}) \leq \rho$, since by construction the columns of $\mathbf{Z}^{t^\star}$ are also in $\mathbf{Z}$. In this case, the trivial bound to $\rho^{t^\star}$ is $\min\{N, r^\star t^\star\}$.

Another important observation is that it is not difficult to define a dynamical system that can selectively forget some of the information stored into the state space. In fact, we can exploit the same idea underpinning a linear autoencoder and make a small modification. We can notice that matrix $\mathbf{R}_{a,s}$ corresponds to a linear function that pushes down the components of the input vector of a positions, i.e. given in input a vector $\mathbf{x} \in \mathbb{R}^s$, it returns a vector $\mathbf{x}^\downarrow \in \mathbb{R}^s$ with $x_i^\downarrow = 0$ for $i = 1,\ldots,a$, and $x_i^\downarrow = x_{i-a+1}$ for $i = a+1,\ldots,s$. If we now consider the matrix $\mathbf{R}_{a,s/p} = \mathbf{R}_{a,s} - \mathbf{1}_s^{p+a,p}$, where $\mathbf{1}_s^{i,j}$ is a $s \times s$ matrix of all zeros except for entry i,j, which is equal to 1, then $\mathbf{R}_{a,s/p}$, in addition to shifting down of a positions the components of $\mathbf{x}$, also annihilates the $(p+a)$th component of the resulting vector. Thus, if we consider the following linear dynamical system

$$\mathbf{y}_t = \mathbf{P}_{a,s}\mathbf{x}_t + \mathbf{R}_{a,s/p}\mathbf{y}_{t-1}$$

and feed it with all the sequences in $\mathcal{T}$, the matrix $\mathbf{\Xi}_{/p}$ obtained by collecting all the state vectors $\mathbf{y}_1^1, \mathbf{y}_2^1, \ldots, \mathbf{y}_{l_n}^n$ as rows, will be equal to $\mathbf{\Xi}$, except for columns $p+a, p+2a, p+3a, \ldots$ which will have zero entries. If we now consider the thin svd decomposition of $\mathbf{\Xi}_{/p} = \mathbf{V}_{/p}\mathbf{\Lambda}_{/p}\mathbf{U}_{/p}^\mathsf{T}$, we can project the state space vectors into the subspace spanned by the columns of $\mathbf{\Xi}_{/p}$, obtaining $\tilde{\mathbf{y}}_t = \mathbf{U}_{/p}{}^\mathsf{T}\mathbf{y}_t$. Exploiting the fact that $\mathbf{U}_{/p}\tilde{\mathbf{y}}_t = \mathbf{y}_t$, we can write

$$\begin{aligned} \tilde{\mathbf{y}}_t &= \mathbf{U}_{/p}{}^\mathsf{T}(\mathbf{P}_{a,s}\mathbf{x}_t + \mathbf{R}_{a,s/p}\mathbf{y}_{t-1}) \\ &= \underbrace{\mathbf{U}_{/p}{}^\mathsf{T}\mathbf{P}_{a,s}}_{\mathbf{A}_{/\mathbf{p}}}\mathbf{x}_t + \underbrace{\mathbf{U}_{/p}{}^\mathsf{T}\mathbf{R}_{a,s/p}\mathbf{U}_{/p}}_{\mathbf{B}_{/\mathbf{p}}}\tilde{\mathbf{y}}_{t-1} \end{aligned}$$

i.e., we get a very similar solution as in the case of the linear autoencoder where instead of using the thin svd decomposition of $\mathbf{\Xi}$ and $\mathbf{R}_{a,s}$, we use the thin svd decomposition of $\mathbf{\Xi}_{/p}$ and $\mathbf{R}_{a,s/p}$. Notice that $\rho_{/p} = rank(\mathbf{\Xi}_{/p}) \leq \rho$. Exploiting this observation, we can state the following theorem

Theorem 3.1. Given a vector $\mathbf{w}$ with rank $r^\star$ and timespan $t^\star$ over $\mathcal{T}$, then there exists a linear dynamical system with space dimension $\rho_{/\pi} \leq \min\{N, \frac{1}{2}[(1+2t^\star)r^\star - r^{\star 2}]\}$ implementing $\mathcal{F}_\mathbf{w}(\cdot)$ over $\mathcal{T}$.

Proof. From the above discussion it is clear that there exists a linear dynamical system with space dimension less or equal to $r^\star t^\star$ able to implement $\mathcal{F}_\mathbf{w}(\cdot)$. However, it is possibile to further reduce such dimension by an appropriate choice of the basis $\mathfrak{B}$. In fact, if we create $\mathfrak{B}$ incrementally by setting $\mathfrak{B}_1 = \mathbf{w}_{t^\star}$ and then trying to add new columns to $\mathfrak{B}$ by proceeding backward to vectors $\mathbf{w}_{t^\star-1}$, $\mathbf{w}_{t^\star-2}$, ..., the memory of the dynamical system can discard one by one the contribution of the columns of $\mathfrak{B}$ while approaching $t^\star$. The worst case is given by the scenario where $\mathfrak{B}$ is constituted by the linearly independent vectors $\mathbf{w}_{t^\star-r^\star+1},\ldots,\mathbf{w}_{t^\star}$, since in that case the dynamical system must keep in memory the full contributions of $\mathfrak{B}$ for the first $t^\star - r^\star$ time steps; after that, the contributions of columns of $\mathfrak{B}$ can be discarded one by one at each time step, starting form the last column backward to the first column. This can be obtained by using $\mathbf{R}_{a,s/\pi}$, where π is the sequence of indexes that need to be annihilated. Thus, in the worst case the dimension of the state vector is given by $(t^\star - r^\star)r^\star$ coordinates for storing information

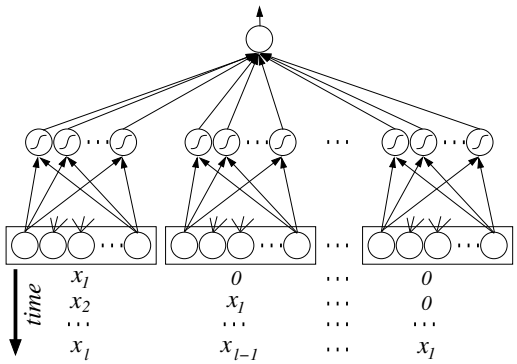

Figure 1: Feedforward network with the 1-time-step property.

for the first $t^\star - r^\star$ time steps, plus $\frac{1}{2}(r^\star+1)r^\star$ coordinates for storing information for the last time steps before reaching $t^\star$. □

It should be remarked that, by definition, $r^\star \leq a$. Moreover, the rank of the matrix collecting all the state vectors of the resulting dynamical system can in practice be far less than the bound given above.

3.2 Nonlinear Functions

In this section, we discuss how to exploit the results obtained for linear functions in the case in which $\mathcal{F}(\cdot)$ can be implemented by feedforward neural networks. Specifically, we show how to build a recurrent neural network able to implement $\mathcal{F}(\cdot)$ over $\mathcal{T}$ with a number of hidden units which is bounded by a function of two quantities that are generalisations of the rank and timespan already discussed for the case of linear functions.

We start by considering a specific class of feedforward neural networks, and then we move to the general class of feedforward neural networks.

Definition 3.2. *A single hidden layer feedforward neural network $\mathcal{N}_f$ is said to have the 1-time-step property if any hidden unit only connects to input units belonging to the same time step.*

An example of feedforward network with this property is shown in Figure 1. If a network has the 1-time-step property, then it is possibile to write the output $o_{\mathcal{N}_f}$ of $\mathcal{N}_f$ to the (unrolled) input (sub)sequence $\mathbf{s} \equiv (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l)$ as

$$o_{\mathcal{N}_f}(\mathbf{s}) = \sigma\left(\sum_{t=1}^{l}\sum_{h=1}^{H_t} w^o_{ht}\underbrace{\sigma(\mathbf{w}^\mathsf{T}_{ht}\mathbf{x}_t)}_{h_{ht}(\mathbf{x}_t)}\right),$$

where H_t is the number of hidden units associated with the input at time step t. Notice that H_t can be 0 for some values of t. Moreover, w^o_{ht} are the weights from the hidden units to the output unit, thus they are the components of a vector that we denote as $\mathbf{w}^o$. Let $H = \sum_{t=1}^{l} H_t$, i.e. the number of hidden units of $\mathcal{N}_f$, and

$$\mathfrak{h}(\mathbf{x}_i) = [h_{11}(\mathbf{x}_i), h_{21}(\mathbf{x}_i), .., h_{H_1 1}(\mathbf{x}_i), h_{12}(\mathbf{x}_i), .., h_{H_l l}(\mathbf{x}_i)]$$

be the row vector that collects the output of all hidden units to the same input $\mathbf{x}_i$. This vector can be understood as the set of contributions that $\mathbf{x}_i$ can give at time steps from 1 to l. The idea is to replace the original sequence $\mathbf{s} \equiv (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l)$ with sequence $(\mathfrak{h}(\mathbf{x}_1), \mathfrak{h}(\mathbf{x}_2), \cdots, \mathfrak{h}(\mathbf{x}_l))$ and then apply a construction similar to the one described for linear functions. Specifically, we can define $\mathbf{H}$ as

$$\mathbf{H}_q = \begin{bmatrix} \mathfrak{h}(\mathbf{x}^q_1) & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathfrak{h}(\mathbf{x}^q_2) & \mathfrak{h}(\mathbf{x}^q_1) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathfrak{h}(\mathbf{x}^q_3) & \mathfrak{h}(\mathbf{x}^q_2) & \mathfrak{h}(\mathbf{x}^q_1) & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathfrak{h}(\mathbf{x}^q_{l_q}) & \mathfrak{h}(\mathbf{x}^q_{l_q-1}) & \mathfrak{h}(\mathbf{x}^q_{l_q-2}) & \cdots & \mathfrak{h}(\mathbf{x}^q_1) \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_n \end{bmatrix}$$

Notice that $\mathbf{H}$ contains all the information needed to compute the desired function over $\mathcal{T}$. In fact, by defining the weight row vector $\tilde{\mathbf{w}} \in \mathbb{R}^{Hl}$ as

$$\begin{aligned}\tilde{\mathbf{w}} \;=\; & [w^o_{1l}, .., w^o_{H_l l}, \underbrace{0, .., 0}_{H}, w^o_{1(l-1)}, .., w^o_{H_{l-1}(l-1)}, \underbrace{0, .., 0}_{H}, \\ & .., 0, w^o_{11}, .., w^o_{H_1 1}]\end{aligned}$$

we can write, $\forall j, t$

$$o_{\mathcal{N}_f}(\mathbf{s}^j[1,t]) = \sigma\left(\tilde{\mathbf{w}}\mathfrak{h}^\mathsf{T}_{j,t}\right),$$

where $\mathfrak{h}_{j,t} = row(\iota(j,t), \mathbf{H})$, i.e. the row of matrix $\mathbf{H}$ of index $\iota(j,t) = \sum_{k=1}^{j-1} H_k + t$.

A recurrent neural network $\mathcal{N}_{rec}$ with vector states given by rows of $\mathbf{H}$ can be obtained as follows. Let $\mathbf{H}_{/[1,H]}$ be $\mathbf{H}$ without the first H columns and $\mathbf{H}_{[1,H]}$ be the first H columns of $\mathbf{H}$. $\mathcal{N}_{rec}$ will have a input units, i.e. what is needed to read input vector $\mathbf{x}_t \in \mathbb{R}^a$ at time t. Moreover, $\mathcal{N}_{rec}$ will have a hidden layer composed of H hidden units with hyperbolic tangent function, which are the same as the hidden units of $\mathcal{N}_f$, all connected to the input units. In addition to these hidden units, the hidden layer of $\mathcal{N}_{rec}$ will have $\rho^{\mathbf{H}_{/[1,H]}} = rank(\mathbf{H}_{/[1,H]})$ linear hidden units implementing the encoding part of an autoencoder defined on the sequences generated by the first H hidden units. Notice that items of these sequences are rows of $\mathbf{H}_{[1,H]}$ and that $\mathbf{H}_{/[1,H]}$ corresponds to the $\boldsymbol{\Xi}$ matrix of the state space matrix factorisation. In summary, while the first H sigmoidal hidden units take as input only the input vector $\mathbf{x}_t$ at time t, the linear hidden units at time t take as input the output of the full hidden layer at time $t-1$.

Mathematically, the obtained network, with $H + \rho^{\mathbf{H}_{/[1,H]}}$ hidden units, can be described as

$$\begin{aligned} o_{\mathcal{N}_{rec}}(\mathbf{x}_t) &= \sigma(\tilde{\mathbf{w}} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{U} \end{bmatrix} \mathbf{y}_t) \\ \mathbf{y}_t &= \begin{bmatrix} \mathfrak{h}(\mathbf{x}_t) \\ \tilde{\mathbf{y}}_{t-1} \end{bmatrix} \\ \tilde{\mathbf{y}}_t &= [\ \mathbf{A} \quad \mathbf{B}\ ]\mathbf{y}_{t-1} \end{aligned}$$

where matrices $\mathbf{U}$, $\mathbf{A}$, and $\mathbf{B}$ are obtained by $\mathbf{H}_{/[1,H]}$. Moreover, we assume $\mathbf{y}_0 = \mathbf{0}$ and $\tilde{\mathbf{y}}_0 = \mathbf{0}$. A pictorial representation of the network is shown in Figure 2.

Notice that linear hidden units can be substituted by sigmoidal units by incurring into a predefined error ϵ. In fact, we can observe that, by construction, the linear hidden units get

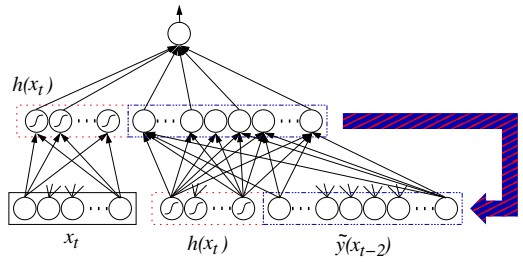

Figure 2: Topology of the recurrent network derived from the feedforward network holding the 1-time-step property.

a bounded input, since they process the output of sigmoidal units, which is bounded between -1 and 1. Moreover, we are considering bounded sequences. Thus, by using sigmoidal units $\beta\sigma(\frac{z}{\beta})$ in place of linear units, we can always find suitable values of βs so to keep an approximation error for the output within the desired tolerance ϵ.

As in the linear case, we can further reduce the size of the hidden space. In fact, we can define the timespan of $\mathcal{N}_f$ over $\mathcal{T}$ as follows

Definition 3.3 (timespan of $\mathcal{N}_f$ over $\mathcal{T}$). *We say that $\mathcal{N}_f$ has timespan $t^\star$ if $H_{t^\star} \neq \mathbf{0}$ and $\forall t > t^\star, H_t = 0$.*

We can observe that the dynamical system derived from $\mathbf{H}_{/[1,H]}$ can forget the first tH state components after t time steps. This leads to a bound on the number of hidden units which is $\min\{N, \frac{1}{2}H_{t^\star}(H_{t^\star}+1)\}$, where $H_{t^\star} = \sum_{i=1}^{t^\star} H_i$.

Let now consider a feedforward neural network with no restriction, i.e. a feedforward neural network where hidden units take input from all time slots. In this case, for each hidden node, we can apply what we have seen for linear functions when considering its net input. Thus, each hidden unit h generates a corresponding matrix of states $\mathbf{Y}(h)$. All these matrices can be concatenated to form a single large matrix $\tilde{\mathbf{Y}} \equiv [\mathbf{Y}(1), \ldots, \mathbf{Y}(H)]$ that can again be decomposed by thin svd $\tilde{\mathbf{Y}} = \tilde{\mathbf{V}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^\mathsf{T}$. $\tilde{\mathbf{U}}$ can now be used to compress this component of the state vector and $\tilde{\mathbf{U}}$ to expand it back when it is needed to use it for the net input of the hidden units. Since all operations are linear, the size of the resulting component of the state vector will be $\tilde{\rho} = rank(\tilde{\mathbf{Y}})$. Upper bounds on $\tilde{\rho}$ can trivially be derived by using the upper bound we already derived for a single linear unit when considering its rank and timespan. Thus, overall, the resulting $\mathcal{N}_{rec}$ will have $H + \tilde{\rho}$ hidden units. Again, linear units can be substituted by sigmoidal units incurring into a predefined error.

4 Empirical assesment

In this section, we empirically evaluate on a real dataset the size of the state space of a recurrent neural network derived by a feed-forward network with hidden units. This constitutes the worst case scenario among the ones described in the previous sections. As dataset we have selected 71 DNA sequences[1] of up to 159 amino-acids. Each of the 20 occurring amino-acid is encoded by a vector of size 12 with ± 1 components. For these sequences we have defined a binary classification task at each position (amino-acid). The task consists in predicting whether the considered amino-acid is part of a specific secondary structure, i.e. a *helix*, or not. In the first case, the target for that amino-acid is set to $+1$ (2,282 cases), otherwise to -1 (4,925 cases). The length distribution of the sequences is reported in Figure 3. Overall, 7207 predictions have to be performed.

A feed-forward network with 10 hidden units and 12×159 inputs, able to access all the amino-acids read till a given position, has been trained on the classification task, obtaining an accuracy of almost 0.99 (73 errors over 7,207 predictions). The resulting timespan of the hidden weight vectors is maximum (159), as can be verified in Figure 4 for one hidden unit. This feed-forward network has then been transformed into a recurrent neural network, as explained in Subsection 3.2. Exact reconstruction of the output of the feed-forward network is obtained with a state space of size 1,655. However, by

[1]The dataset can be retrieved at the following URL: http://www.math.unipd.it/~sperduti/DNAdataset.zip.

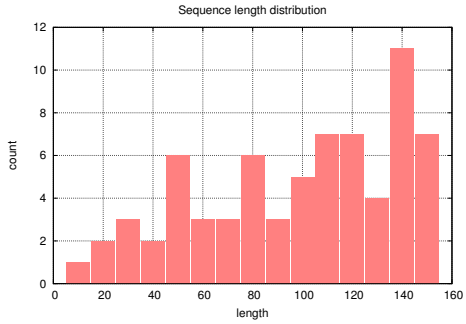

Figure 3: Distribution of the length of the sequences comprising the DNA dataset.

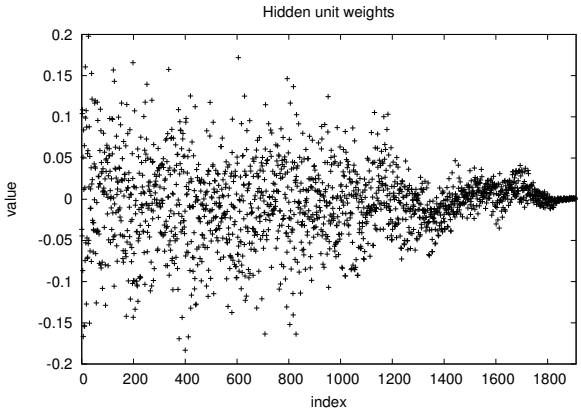

Figure 4: Weight values for hidden unit 4. Weights with index from 1 to 12 correspond to the last presented input. From 13 to 24 to the input presented 1 time step before, and so on. Thus, moving to higher indices corresponds to moving back in time. All other hidden units have similar weight distribution. It is clear that the timespan of the weight vectors is maximum, i.e. 159.

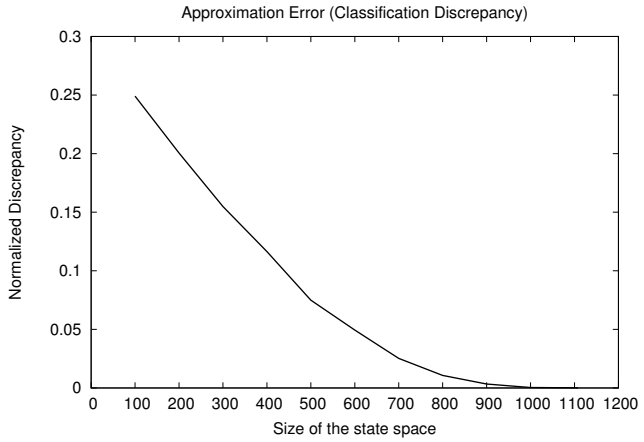

Figure 5: Discrepancy between the output of the feed-forward network with 10 hidden units and the output of the reconstructed recurrent neural network as a function of the size of its state space. Normalised discrepancy is computed as the fraction of the number of predictions where the feed-forward network and the recurrent network disagree, divided by the total number of predictions (7207).

only considering the first k singular values of the SVD decomposition (thin SVD), it is also possibile to construct recurrent neural networks with different degrees of approximation. The very same prediction of the feed-forward network is preserved for $k \geq 1,108$. For smaller values of k the predictions of the obtained recurrent neural network disagree with the predictions of the feed-forward network at different levels. In Figure 5 we have reported the normalised discrepancy $nd_{\mathcal{D}}(\mathcal{N}_f, \mathcal{N}_{rec_k})$ between the recurrent network $\mathcal{N}_{rec_k}$ with state space of size k and the feed-forward network $\mathcal{N}_f$, over the dataset $\mathcal{D}$, as a function of k. Normalised discrepancy is computed as

$$nd_{\mathcal{D}}(\mathcal{N}_f, \mathcal{N}_{rec_k}) = \frac{\sum_{x \in \mathcal{D}} |\mathcal{N}_f(x) - \mathcal{N}_{rec_k}(x)|}{2|\mathcal{D}|}$$

where with x we refer to the input for a single prediction, i.e. in our case the amino-acid (and amino-acid already read for the considered sequence) for which a prediction should be performed. The factor 2 takes into account that predictions are ± 1 values, i.e. in case of disagreement $|\mathcal{N}_f(x) - \mathcal{N}_{rec_k}(x)| = 2$.

For this dataset and task, it is clear that the need to implement a memory (which is external in the case of the feedforward network) within the recurrent neural network costs very many additional parameters (weights).

5 Conclusions

The results we have described in this paper should be considered into the right perspective. They do not constitute a complete basis for the design of new efficient and effective learning algorithms for RNN. However, they suggest many research directions, both concerning learning and interpretation of a RNN. When considering learning, one trivial suggestion is to design a learning algorithm that learns feedforward components in separation and then merges them as shown in the paper, or an approach that starts by learning shorter sequences using feedforward networks which are then turned into RNNs to be further combined with additional feedforward networks. Another line of research could be to make an explicit separation, inside a RNN, of the functional component from the memory component, and to interleave learning on the functional component with learning on the memory component. When considering interpretation of a RNN, a trivial observation seems to be that hidden units with high gain are probably mainly involved into the functional component of the RNN, while hidden units with low gain are probably mainly involved into the memory component of the RNN. More sophisticated interpretation approaches could try to devise a decoding function for the hidden units in analogy to what happens with linear autoencoders.

References

[Bengio *et al.*, 1994] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[Boulanger-Lewandowski *et al.*, 2012] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

[Hochreiter and Schmidhuber, 1996] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *NIPS*, pages 473–479, 1996.

[Kremer, 2001] Stefan C. Kremer. *Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 1st edition, 2001.

[Martens and Sutskever, 2011] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *ICML*, pages 1033–1040, 2011.

[Micheli and Sperduti, 2007] Alessio Micheli and Alessandro Sperduti. Recursive principal component analysis of graphs. In *ICANN (2)*, pages 826–835, 2007.

[Pasa and Sperduti, 2014] Luca Pasa and Alessandro Sperduti. Pre-training of recurrent neural networks via linear autoencoders. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3572–3580, 2014.

[Sperduti, 2006] Alessandro Sperduti. Exact solutions for recursive principal components analysis of sequences and trees. In *ICANN (1)*, pages 349–356, 2006.

[Sperduti, 2007] Alessandro Sperduti. Efficient computation of recursive principal component analysis for structured input. In *ECML*, pages 335–346, 2007.

[Sutskever *et al.*, 2013] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML (3)*, pages 1139–1147, 2013.