Learning Sequential Data with the Help of Linear Systems

Luca Pasa and Alessandro Sperduti(✉)

Department of Mathematics, University of Padova, Padua, Italy
{lpasa,sperduti}@math.unipd.it

Abstract. The aim of the paper is to show that linear dynamical systems can be quite useful when dealing with sequence learning tasks. According to the complexity of the problem to face, linear dynamical systems may directly contribute to provide a good solution at a reduced computational cost, or indirectly provide support at a pre-training stage for nonlinear models. We present and discuss several approaches, both linear and nonlinear, where linear dynamical systems play an important role. These approaches are empirically assessed on two nontrivial datasets of sequences on a prediction task. Experimental results show that indeed linear dynamical systems can either directly provide a satisfactory solution, as well as they may be crucial for the success of more sophisticated nonlinear approaches.

Keywords: Linear dynamical systems · Autoencoders · Learning in sequential domains

1 Introduction

With the diffusion of cheap sensors, sensor-equipped devices (e.g., drones), and sensor networks (such as *Internet of Things* [1]), as well as the development of inexpensive human-machine interaction interfaces, the ability to quickly and effectively process sequential data is becoming more and more important. Many are the tasks that may benefit from advancement in this field, ranging from monitoring and classification of human behaviour to prediction of future events. Most of the above tasks require pattern recognition and machine learning capabilities.

Many are the approaches that have been proposed in the past to learn in sequential domains (e.g., [2]). A special mention goes to recent advancements involving Deep Learning [3–5]. Deep Learning is based on very non-linear systems, which reach quite good classification/prediction performances, very often at the expenses of a very high computational burden. Actually, it is common practice, when facing learning in a sequential, or more in general structured, domain to readily resort to non-linear systems. Not always, however, the task really requires a non-linear system. So the risk is to run into difficult and computational expensive training procedures to eventually get a solution that improves

F. Schwenker et al. (Eds.): ANNPR 2016, LNAI 9896, pp. 3–17, 2016.
DOI: 10.1007/978-3-319-46182-3_1

of an epsilon (if not at all) the performances that can be reached by a simple linear dynamical system involving simpler training procedures and a much lower computational effort.

The aim of this paper is to open a discussion about the role that linear dynamical systems may have in learning in sequential domains. On one hand, we like to point out that a linear dynamical system (LDS) is able, in many cases, to already provide good performances at a relatively low computational cost. On the other hand, when a linear dynamical system is not enough to provide a reasonable solution, we show how to resort to it to design quite effective pre-training techniques for non-linear dynamical systems, such as Echo State Networks (ESNs) [6] and simple Recurrent Neural Networks (RNNs) [7].

Specifically, here we consider the task of predicting the next event into a sequence of events. Two datasets involving polyphonic music and quite long sequences are used as practical exemplification of this task. We start by introducing a simple state space LDS. Three different approaches to train the LDS are then considered. The first one is based on random projections and it is particularly efficient from a computational point of view. The second, computationally more demanding approach, projects the input sequences onto an approximation of their spanned sub-space obtained via a linear autoencoder naturally associated to the LDS. For both approaches the output weights are obtained by computing the pseudo-inverse of the hidden states matrix. Finally, we consider a refinement via stochastic gradient descent of the solution obtained by the autoencoder-based training scheme. Of course, this last approach requires additional computational resources.

We then move to the introduction of non-linearities. From this point of view, ESNs can be considered a natural extension of the first linear approach, since non-linear random projections are used to define a coding of input sequences, and pseudo-inverse exploited to estimate output weights. In addition, these are the less computationally demanding models in the non-linear models arena. The second considered family of non-linear models is given by simple RNNs, which computationally are significantly more demanding. Here we experimentally show that, at least for the addressed prediction task and the considered datasets, the introduction of pre-training approaches involving linear systems leads to quite large improvements in prediction performances. Specifically, we review pre-training via linear autoencoder previously introduced in [8], and an alternative based on Hidden Markov Models (HMMs) [9].

Finally, it is worth to notice that linear autoencoders have been exploited in a recent theoretical work [10] to provide equivalence results between feed-forward networks, that have simultaneous access to all items composing a sequence, and single-layer RNNs which access information one step at a time.

2 Computational Models

In this section, we introduce the addressed learning task as well as the studied linear and non-linear models. Moreover, we present the different training approaches for these models that we have experimentally assessed.

2.1 Learning Task

In this paper, we focus on a prediction task over sequences that can be formalized as described in the following. We would like to learn a function $\mathcal{F}(\cdot)$ from multivariate bounded length input sequences to desired output values. Specifically, given a training set $\mathcal{T} = \{(\mathbf{s}^q, \mathbf{d}^q) | q = 1, \ldots, N,\ \mathbf{s}^q \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \ldots, \mathbf{x}_{l_q}^q), \mathbf{d}^q \equiv (\mathbf{d}_1^q, \mathbf{d}_2^q, \ldots, \mathbf{d}_{l_q}^q),\ \mathbf{x}_t^q \in \mathbb{R}^n,\ \mathbf{d}_t^q \in \mathbb{R}^s\}$, we wish to learn a function $\mathcal{F}(\cdot)$ such that $\forall q, t\ \mathcal{F}(\mathbf{s}^q[1,t]) = \mathbf{d}_t^q$, where $\mathbf{s}^q[1,t] \equiv (\mathbf{x}_1^q, \mathbf{x}_2^q, \ldots, \mathbf{x}_t^q)$. Experimental assessment has been performed in the special case in which $\mathbf{d}_k^q = \mathbf{x}_{k+1}^q$. Different learning approaches have been considered for both linear and non-linear dynamical systems, as described in the following.

2.2 Linear and Non-linear Models

The linear model we use is a discrete-state dynamical system defined as:

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

$$\mathbf{o}_t = \mathbf{C}\,\mathbf{h}_t, \tag{2}$$

where $\mathbf{h}_t \in \mathbb{R}^m$ is the state of the system at time t, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{B} \in \mathbb{R}^{m \times m}$, $\mathbf{C}^{s \times m}$ are respectively the input matrix, the state matrix and the output matrix. In addition, we assume $\mathbf{h}_0 = \mathbf{0}$, i.e. the null vector.

Associated with this dynamical system, we consider a linear autoencoder obtained by substituting Eq. (2) with

$$\begin{bmatrix} \mathbf{x}_t \\ \mathbf{h}_{t-1} \end{bmatrix} = \mathbf{C}\,\mathbf{h}_t, \tag{3}$$

where $\mathbf{C} \in \mathbb{R}^{(n+m) \times m}$, and m takes the smallest value satisfying Eqs. (1) and (3). Specifically, the smallest value of m can be found as proposed in [11], i.e. by factorisation of the state matrix $\mathbf{H}$ collecting as rows the state vectors of the linear system described by Eq. (1). For the sake of presentation, let illustrate such factorisation for a single sequence $\mathbf{s} \equiv (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l)$

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

where, $\mathbf{\Xi} \in \mathbb{R}^{l \times n \cdot l}$ is the data matrix which collects the (inverted) subsequences $\mathbf{s}[1,i]$, $\forall i = 1, \ldots, l$, as rows, and $\mathbf{\Omega}$ is the parameter matrix of the dynamical system.

The smallest value of m preserving all information about $\mathbf{s}$ is obtained by thin svd decomposition of $\mathbf{\Xi} = \mathbf{V}\mathbf{\Lambda}\mathbf{U}^\mathsf{T}$. In fact, by imposing that $\mathbf{U}^\mathsf{T}\mathbf{\Omega} = \mathbf{I}$, i.e.

the identity matrix, the resulting state space preserves all information and it is of the smallest possibile dimension $m = rank(\mathbf{\Xi})$, i.e. the number of nonzero singular values of $\mathbf{\Lambda}$. Matrices $\mathbf{A}$ and $\mathbf{B}$ satisfying such condition can be obtained by exploiting the fact that $\mathbf{U}\mathbf{U}^\mathsf{T} = \mathbf{I}$, which entails $\mathbf{\Omega} = \mathbf{U}$. This equality can be met using matrices

$$\mathbf{P}_{n,n\cdot l} \equiv \begin{bmatrix} \mathbf{I}_{n\times n} \\ \mathbf{0}_{n(l-1)\times n} \end{bmatrix}, \text{ and } \mathbf{R}_{n,n\cdot l} \equiv \begin{bmatrix} \mathbf{0}_{n\times n(l-1)} & \mathbf{0}_{n\times n} \\ \mathbf{I}_{n(l-1)\times n(l-1)} & \mathbf{0}_{n(l-1)\times n} \end{bmatrix},$$

to define $\mathbf{A} \equiv \mathbf{U}^\mathsf{T}\mathbf{P}_{n,n\cdot l}$ and $\mathbf{B} \equiv \mathbf{U}^\mathsf{T}\mathbf{R}_{n,n\cdot l}\mathbf{U}$. Moreover, since $\mathbf{H} = \mathbf{V}\mathbf{\Lambda}$, the original data $\mathbf{\Xi}$ can be fully reconstructed by computing $\mathbf{H}\mathbf{U}^\mathsf{T} = \mathbf{V}\mathbf{\Lambda}\mathbf{U}^\mathsf{T} = \mathbf{\Xi}$, which can be achieved by running the dynamical system

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

starting from $\mathbf{h}_l$, i.e. $\mathbf{C} = \begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{B}^\mathsf{T} \end{bmatrix}$. The very same result can be obtained when considering a set of sequences. It is enough to stack all the data matrices corresponding to sequences in the set, and padding with zeros when needed. For example, given the set $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2\}$ with $length(\mathbf{s}_1) = 3$ and $length(\mathbf{s}_2) = 2$, the data matrix corresponding to the set is defined as $\mathbf{\Xi}_\mathcal{S} = \left[\begin{array}{c} \mathbf{\Xi}_{\mathbf{s_1}} \\ \hline \mathbf{\Xi}_{\mathbf{s_2}} \mathbf{0}_{n\times n} \end{array}\right]$.

It's crucial to notice that the above method works exactly only in the case where m is equal to the rank of $\mathbf{\Xi}$. Due to this fact, applying this method in real world scenarios could be difficult due to the fact that the rank of matrix $\mathbf{\Xi}$ is very large. For this reason, in [8] a procedure for approximate computation of the truncated thin svd decomposition of $\mathbf{\Xi}$ to a preset value of $m \ll rank(\mathbf{\Xi})$ is proposed.

The considered non-linear model is obtained by adding nonlinear functions $f()$ and $g()$ to Eqs. (1) and (2), i.e.:

$$\tilde{\mathbf{h}}_t = f(\mathbf{A}\,\mathbf{x}_t + \mathbf{B}\,\tilde{\mathbf{h}}_{t-1}), \tag{4}$$

$$\tilde{\mathbf{o}}_t = g(\mathbf{C}\,\tilde{\mathbf{h}}_t). \tag{5}$$

We have considered two specific instances of the above model: *(i)* when using it as an ESN, $f()$ as been instantiated to $tanh()$ and $g()$ to the identity function; *(ii)* when using it as a simple RNN, both $f()$ and $g()$ have been instantiated to $tanh()$.

2.3 Training Approaches

For the linear model described by Eqs. (1) and (2), we consider three different training approaches:

$\mathcal{L}_1$: Randomly generate matrices $\mathbf{A}$ and $\mathbf{B}$; compute the corresponding state matrix $\mathbf{H}$ for the full set of training sequences; define $\mathbf{C} = \mathbf{D}\mathbf{H}^{\mathsf{T}^+}$, where

$\mathbf{D} = [\mathbf{d}_1^1, \mathbf{d}_2^1, \dots, \mathbf{d}_{l_N}^N]$ is the matrix collecting all the target vectors, and $\mathbf{H}^{\mathsf{T}+}$ is the pseudo-inverse of $\mathbf{H}^{\mathsf{T}}$. The use of pseudo-inverse leads to the minimization of the Mean Squared Error between target and actual output. This approach corresponds to a ESN-like training for linear systems.

$\mathcal{L}_2$: Compute $\mathbf{A}$ and $\mathbf{B}$ according to the procedure proposed in [8] for the linear auto-encoder; compute the corresponding state matrix $\mathbf{H}$ for the full set of training sequences; define $\mathbf{C} = \mathbf{D}\mathbf{H}^{\mathsf{T}+}$.

$\mathcal{L}_3$: Perform stochastic gradient descent (SGD) with respect to the regularized error function

$$E_{\mathcal{T}} = \frac{1}{NL}\sum_{q=1}^{N}\sum_{j=1}^{l_q}(\mathbf{d}_j^q - \mathbf{o}_j^q)^2 + R_1 + R_2,$$

where $L = \sum_{q=1}^{N} l_q$, and

$$R_1 = |\sum_{i}^{m}\sum_{j}^{n}\mathbf{A}_{ij}| + |\sum_{i}^{m}\sum_{j}^{m}\mathbf{B}_{ij}| + |\sum_{i}^{s}\sum_{j}^{m}\mathbf{C}_{ij}|,$$
$$R_2 = \sum_{i}^{m}\sum_{j}^{n}\mathbf{A}_{ij}^2 + \sum_{i}^{m}\sum_{j}^{m}\mathbf{B}_{ij}^2 + \sum_{i}^{s}\sum_{j}^{m}\mathbf{C}_{ij}^2,$$

with starting point given by the result of approach $\mathcal{L}_2$ (pre-training).

Concerning approach $\mathcal{L}_1$, a relevant issue is how to generate matrices $\mathbf{A}$ and $\mathbf{B}$. This issue has already be addressed in ESN. Indeed, in order to avoid problems in computing the system state and ensure good results, a set of rules to follow for random matrix initialization has been proposed. This set of rules is called Echo State Property [12], and in particular they prescribe to ensure that the random initialized matrices have *spectral radius* ρ less than or equal to 1. Unfortunately computing the spectral radius of large matrices is computationally demanding, so we use a much faster approach where we require $\mathbf{A}$ and $\mathbf{B}$ to have norm $\|\cdot\|$ (either L1-norm or L2-norm) less than or equal to 1. Since for any matrix $\mathbf{M}$, $\rho(\mathbf{M}) \leq \|\mathbf{M}\|$, in this way the Echo State Property is preserved.

For the non-linear model described by Eqs. (4) and (5), we consider four different training approaches:

$\mathcal{N}_1$: Adopt the Echo State Network training procedure that randomly initializes matrices $\mathbf{A}$ and $\mathbf{B}$ according to the Echo State Property and only trains the output weights using pseudo-inverse of the hidden representations, i.e. compute $\mathbf{C} = \mathbf{D}\tilde{\mathbf{H}}^{+}$, where $\tilde{\mathbf{H}} = [\tilde{\mathbf{h}}_1^1, \tilde{\mathbf{h}}_2^1, \dots, \tilde{\mathbf{h}}_{l_N}^N]$ is the matrix that collects all the hidden representations obtained by running the system, with random matrices $\mathbf{A}$ and $\mathbf{B}$, over all sequences in the training set.

$\mathcal{N}_2$: Perform SGD with respect to the regularized error function $E_{\mathcal{T}}$, with standard random initialization for matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. This corresponds to a standard RNN training procedure.

$\mathcal{N}_3$: Perform SGD with respect to the regularized error function $E_{\mathcal{T}}$, initializing matrices $\mathbf{A}$ and $\mathbf{B}$ according to the procedure proposed in [8] for the linear auto-encoder, and $\mathbf{C}$ with $\mathbf{D}\tilde{\mathbf{H}}^{+}$, where $\tilde{\mathbf{H}}$ is obtained by first running Eq. (4) with initialized matrices $\mathbf{A}$ and $\mathbf{B}$. This approach corresponds to the pre-training one proposed in [8].

$\mathcal{N}_4$: Perform SGD with respect to the regularized error function $E_{\mathcal{T}}$, starting from matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ obtained by the following pre-training approach: *(i)* train a linear HMM over the training set $\mathcal{T}$; *(ii)* using the obtained HMM, generate N_{pt} sequences that will constitute the pre-training dataset $\mathcal{T}_{pr}$; *(iii)* perform SGD with respect to the regularized error function $E_{\mathcal{T}_{pr}}$ using $\mathcal{T}_{pr}$ and standard random initialization for matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. This approach corresponds to the pre-training one proposed in [9].

3 Experimental Assessment

In this section we are going to compare the capability of the different systems. The task chosen for testing the various models is the prediction of polyphonic music sequences. The learning task consists in predicting the notes played at time t given the sequence of notes played till time $t-1$. This task turns out to be really interesting because of the nature of the data. Indeed the music sequences are complex, and follow a multi-modal complex distribution that makes difficult to perform the training on them. Moreover prediction of these sequences requires a good capability by the network in managing long-term temporal dependencies. This task has been already tested on different models [8,9,13]. The models and associated training approaches have been tested on two different datasets:

- Nottingham dataset, that contains over 1000 Folk tunes stored in a special text format. These tunes have a simple structure and the set of notes and chords used is quite small.
- Piano-midi.de dataset that contains classic music songs that present a complex structure and that use a wide range of different cords and notes. The songs are longer than the tunes contained in the Nottingham dataset.

We have decided to use these two datasets because they contain very different type of data. Moreover, they allow to stress the models in order to understand the strengths and weaknesses of them. The sequences contained in the dataset are the midi representation of music songs. Each song in the dataset is represented by using a sequence of binary vectors. Each binary vector is composed of 88 values that represent each single note in the piano-roll that span the whole range from A0 to C8. The i-th bit of j-th vector is set to one only if the i-th note is played at j-th time step of the considered song. The average number of bits set to 1 (maximum polyphony) is 15 (average 3.9). The music sequences contained in the datasets have variable length. In particular, in Piano-midi.de many sequences are composed of thousands of time steps. Statistics on the dataset (largest sequence length, number of sequences, etc.) are given in [9].

As performance measure we adopted the accuracy measure used in [13] and described in [14]. Each dataset is split in training set, validation set, and test set.

We first discuss the experimental results obtained for approaches where unsupervised linear or non-linear projections are used to define the current state, i.e. there is no supervised learning of the hidden state mapping, while supervision is used for the hidden to output mapping via pseudo-inverse. These approaches are: LDS-$\mathcal{L}_1$, LDS-$\mathcal{L}_2$, and RNN-$\mathcal{N}_1$. Because of the unsupervised projections, a larger state space is supposedly needed for these approaches to get good performances. Subsequently, we present experimental results obtained for SGD-based approaches, i.e. LDS-$\mathcal{L}_3$, RNN-$\mathcal{N}_2$, RNN-$\mathcal{L}_3$, and RNN-$\mathcal{N}_4$, where a smaller state space is required.

3.1 Results of Approaches Using Unsupervised Projections

As discussed before, approaches that use random or unsupervised projections, in principle, require larger state spaces in order to get good performances. A profitable size for the state space also depends on the complexity of the dataset. We have explored this issue by performing experimental tests using 500, 1000, 1500 hidden units for the Nottingham dataset, and 500, 1000, 2000 hidden units for the Piano-midi.de dataset. The use of 2000 units takes into account the higher complexity of the Piano-midi.de dataset. The considered approaches are LDS-$\mathcal{L}_1$, LDS-$\mathcal{L}_2$, and RNN-$\mathcal{N}_1$. Experimental results for the Notthingam dataset are shown in Fig. 1, while the results for the Piano-midi.de dataset are shown in Fig. 2.

In general, it seems that random projection-based approaches (i.e. LDS-$\mathcal{L}_1$ and RNN-$\mathcal{N}_1$) are insensitive to the size of the state space, while this seems not to be the case for LDS-$\mathcal{L}_2$. In fact, the autoencoder-based training approach seems to be sensitive to the state space size. This fact can be explained by considering the nature of the training technique. Indeed, the autoncoder-based method exploits the SVD decomposition in order to extract the most relevant information from input and previous states. What happens by increasing the state space size is that those features that have lower variance will be added to the state representation. This allows to collect more relevant information in the state. In other words, by using this approach with increasing size of state space, it very likely that some new relevant features enter the state representation. Therefore, since the size of state space is $\ll rank(\Xi)$, the approach will improve its performance (at least on the training set). For both datasets it seems that a further increase of the state space size would keep improving performances. Finally, it can be noticed that on the Piano-midi.de dataset the RNN-$\mathcal{N}_1$ approach seems to show some overfitting with the increase of state space size.

3.2 Results of Approaches Using Supervision and Pre-training

Here we present the results obtained for approaches exploiting supervision and pre-training, i.e. LDS-$\mathcal{L}_3$, RNN-$\mathcal{L}_2$, and RNN-$\mathcal{N}_3$, and RNN-$\mathcal{N}_4$. Actually, all

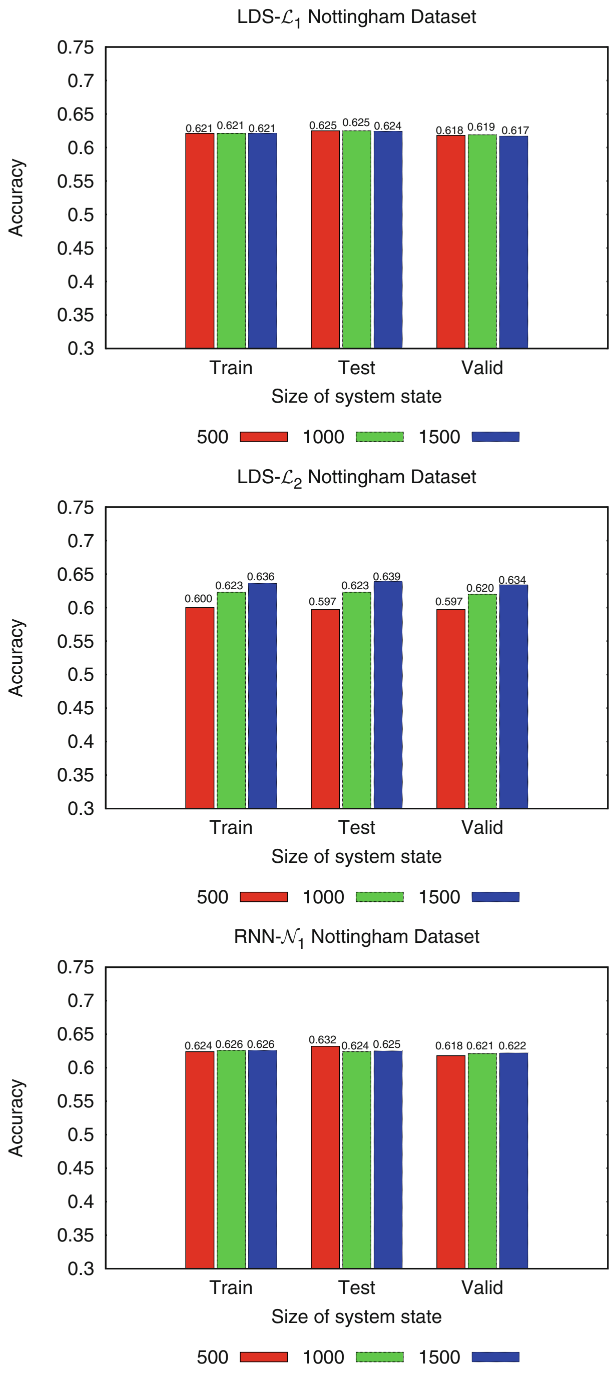


Fig. 1. Results achieved for the Nottingham dataset. Each chart shows the accuracy achieved by training LDS-$\mathcal{L}_1$, LDS-$\mathcal{L}_2$, and RNN-$\mathcal{N}_1$ with different state space sizes.

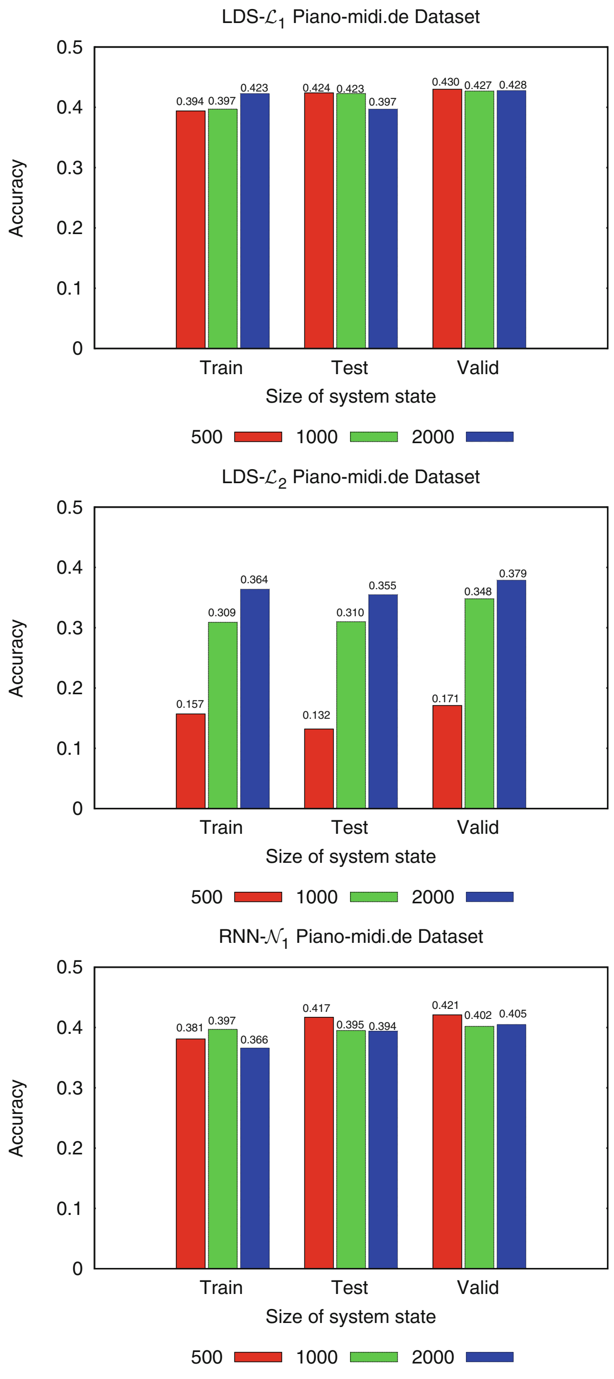


Fig. 2. Results achieved by LDS Piano-midi.de dataset. Each chart shows the accuracy achieved by training LDS-$\mathcal{L}_1$, LDS-$\mathcal{L}_2$, and RNN-$\mathcal{N}_1$ with different state space sizes.

approaches use a form of pre-training except for RNN-$\mathcal{L}_2$, which uses random inizialization for the weights. Since full supervision is used for all approaches, much smaller state spaces are used. Figure 3 reports results, for both datasets, obtained for approaches LDS-$\mathcal{L}_3$, RNN-$\mathcal{L}_2$, and RNN-$\mathcal{N}_3$ using the same settings and model selection procedure described in [8], while Fig. 4 reports results, for both datasets, obtained for approaches RNN-$\mathcal{N}_4$, and RNN-$\mathcal{N}_2$, where the last one is considered for the sake of comparison, using the same settings and model selection procedure described in [9].

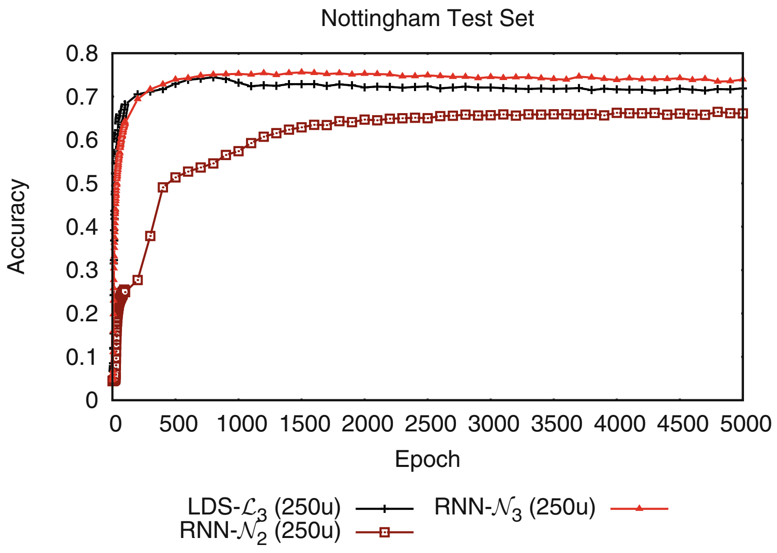
From Fig. 3 it is clear that the use of pre-training exploiting the linear autoencoder allows to achieve better results in less epochs. Indeed, in both datasets the pre-trained versions of the RNN obtain an higher accuracy regardless the number of hidden units. Moreover a very good level of accuracy can be reached in a lower number of epochs. The LDS-$\mathcal{L}_3$ approach (i.e., LDS pre-trained via linear autoncoder initialization and then fine-tuned via SGD) on Notthingham dataset achieved results similar to a non-linear RNN model. However, the same approach has a totally different behavior on Piano-midi.de. In that case, after a few training epochs, the accuracy quickly decreases under the accuracy achieved by randomly-initialized systems. This behavior is due to the high complexity of sequences in Piano-midi.de.

Figure 4 presents the results obtained when a Hidden Markov Model [15] is used to pre-train the RNN. The curves, in this case, also consider the pre-training time needed to create the HMM, to generate the artificial dataset, and to train the RNN on this dataset. This is why curves for RNN-$\mathcal{N}_4$ do not start from the origin. Obtained results are quite good, especially for the Notthingham dataset. Even in this case the use of a linear (probabilistic) model to pre-train the network allows to obtain significantly better results on both datasets, in less time with respect to a standard RNN, i.e. RNN-$\mathcal{N}_2$. Unlike the previous case, by using the HMM-based approach the pre-trained RNN, after the pre-training phase, starts the fine-tuning phase from a significantly better level of accuracy than RNN-$\mathcal{N}_2$. This behavior is due to the fact that the HMM-based pre-training phase exploits a training phase on a different dataset, that allows to start the optimization of the network parameters already during pre-training.

3.3 Discussion

In this section, we try to summarize the experimental results obtained for all the different approaches. In Fig. 5 we report the performances, after model selection via the validation sets, of all the studied approaches on the two considered datasets. The reported performances for RNN-$\mathcal{N}_2$ is taken from [13], since those values are better than the ones we where able to achieve.

The results obtained on the Nottingham dataset seem to show that better performances can be obtained thanks to pre-training. In fact, both linear and non-linear approaches with pre-training return good results, while all the remaining approaches get more or less the same lower performance, regardless of the linearity or non-linearity of the considered model. Among approaches that exploit pre-training, non-linear models get a slighter better performance. It must



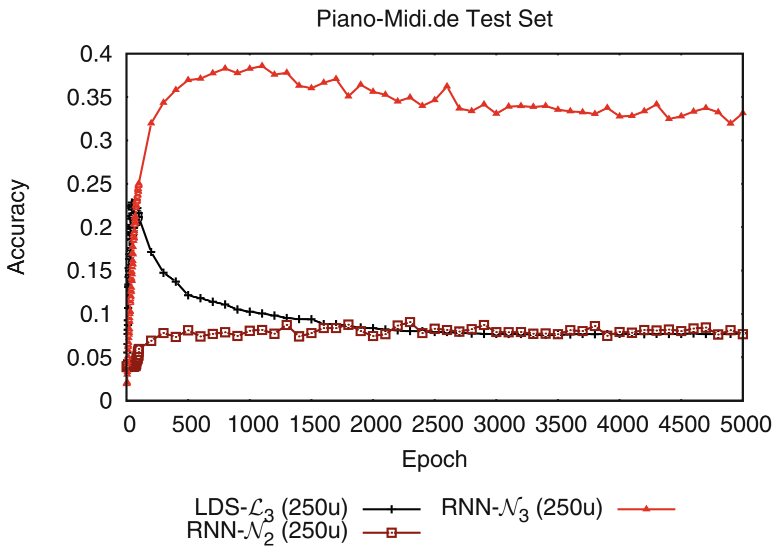
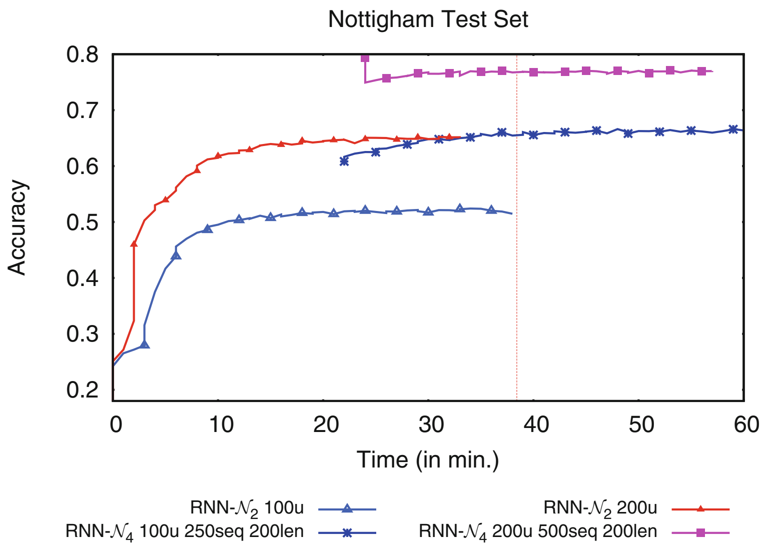


Fig. 3. Test curves on the two datasets by models LDS-$\mathcal{L}_3$, RNN-$\mathcal{N}_2$ and RNN-$\mathcal{N}_3$. Curves are sampled at each epoch till epoch 100, and at steps of 100 epochs afterwards.

however be noticed that the computationally less demanding linear model with pre-training already returns a quite good performance.

For the Piano-midi.de, the situation seems to be quite different, since the best performers are based on random projections, independently from the linearity or non-linearity of the model. Pre-training based approaches seem, for this dataset, to be less effective. This may be due to the fact that the Piano-midi.de owns a



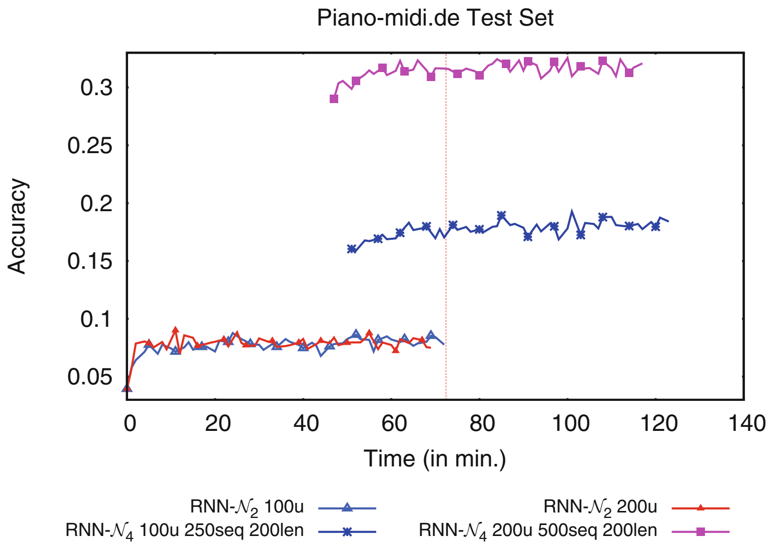
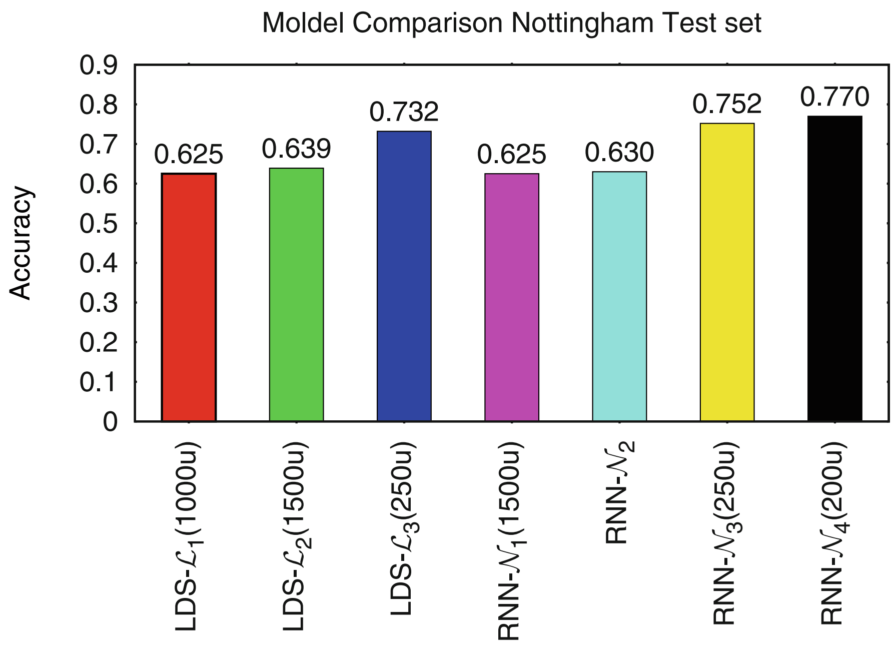


Fig. 4. Test curves obtained for the two datasets by using RNN-$\mathcal{N}_2$ and RNN-$\mathcal{N}_4$ with 250 hidden units. Each curve starts at the time when the pre-training ends. The labels associated with RNN-$\mathcal{N}_4$ curves are composed by three identifiers $\mathtt{n_1u\ n_2\ n3}$, where $\mathtt{n_1}$ is the number of used hidden units, $\mathtt{n_2}$ is the number of sequences generated by the HMM with 10 states, and $\mathtt{n_3}$ is the length of such sequences. Curves that refer to RNN-$\mathcal{N}_2$ are identified by the number of used hidden units. The dotted vertical line is used to mark the end of training of RNN-$\mathcal{N}_2$ after 5000 epochs.



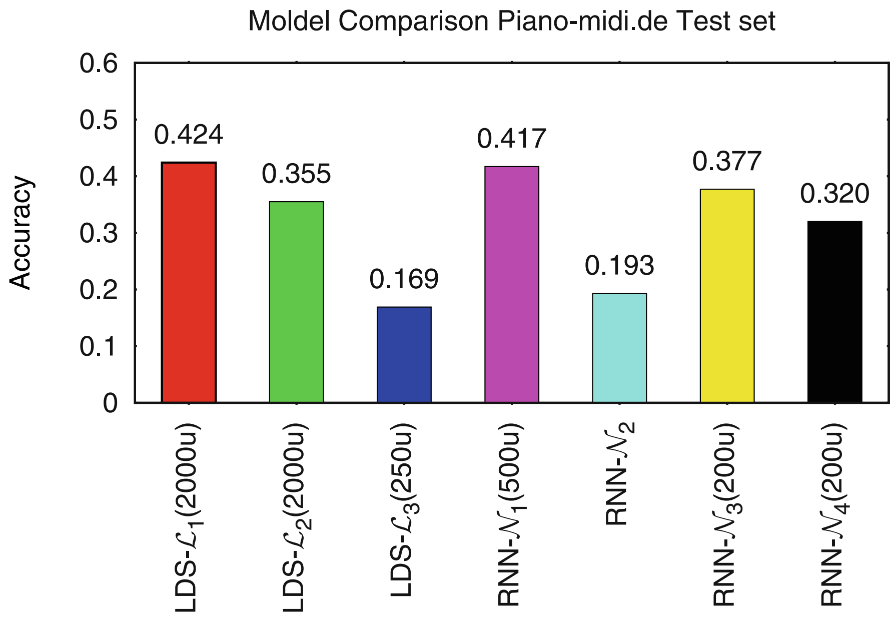


Fig. 5. Comparison of the performances obtained by all the studied approaches. For each model the charts report the accuracy achieved on test set. The number reported in brackets is the number of hidden units selected by model selection using the validation set. The results for RNN-$\mathcal{N}_2$ are taken from [13].

higher complexity, and it is likely that far larger state spaces are needed to reach better performances. In fact, it can be noted that approaches based on random projections, either linear or non-linear, exploit a state space that is tenfold larger than the ones used by SGD-based approaches.

Overall, it is clear that the adoption of a pre-training approach allows to systematically improve over the standard RNN approach, i.e. RNN-$\mathcal{N}_2$.

From a computational point of view linear approaches are far more efficient than non-linear ones, with the only exception of RNN-$\mathcal{N}_1$, that does not require training of the hidden state mapping.

4 Conclusion

In this paper, we have addressed a issue that is often disregarded when considering prediction tasks in sequential domains: the usefulness of linear dynamical systems. We empirically studied the performances of three linear approaches and four non-linear approaches for sequence learning on two significantly complex datasets. Experimental results seem to show that directly or indirectly (as basis for pre-training approaches) linear dynamical systems may play an important role. In fact, when used directly they may by themselves return state-of-the-art performance (see for example LDS-$\mathcal{L}_3$ for the Nottingham dataset) while requiring a much lower computational effort with respect to their non-linear counterpart. Moreover, even when linear models do not perform well, it is always possible to successfully exploit them within pre-training approaches for non-linear systems. Thus, it is important, when facing a prediction task for sequences, to take in full consideration the contribution that linear models can give.

References

1. Ashton, K.: That internet of things thing. RFiD J. **22**(7), 97–114 (2009)
2. Sun, R., Giles, C.L. (eds.): Sequence Learning - Paradigms, Algorithms, and Applications. Springer, London (2001)
3. Graves, A., Mohamed, A., Hinton, G.: Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. IEEE (2013)
4. Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y.: How to construct deep recurrent neural networks (2013). arXiv preprint arXiv:1312.6026
5. Gregor, K., Danihelka, I., Graves, A., Rezende, D.J., Wierstra, D.: Draw: A recurrent neural network for image generation (2015). arXiv preprint arXiv:1502.04623
6. Jaeger, H.: Echo state network. Scholarpedia **2**(9), 2330 (2007)
7. Jerome, T., Connor, R., Martin, D.: Recurrent neural networks and robust time series prediction. IEEE Trans. Neural Netw. **5**(2), 240–254 (1994)
8. Pasa, L., Sperduti, A.: Pre-training of recurrent neural networks via linear autoencoders. In: Advances in Neural Information Processing Systems, pp. 3572–3580 (2014)
9. Pasa, L., Testolin, A., Sperduti, A.: Neural networks for sequential data: a pre-training approach based on hidden Markov models. Neurocomputing **169**, 323–333 (2015)

10. Sperduti, A.: Equivalence results between feedforward and recurrent neural networks for sequences. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 3827–3833. AAAI Press (2015)
11. Sperduti, A.: Exact solutions for recursive principal components analysis of sequences and trees. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 349–356. Springer, Heidelberg (2006)
12. Herbert, J.: The echo state approach to analysing and training recurrent neural networks-with an erratum note. Technical report, German National Research Center for Information Technology GMD, Bonn, Germany, 148:34 (2001)
13. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In: ICML (2012)
14. Bay, M., Ehmann, AF., Downie, J.S.: Evaluation of multiple-F0 estimation and tracking systems. In: ISMIR, pp. 315–320 (2009)
15. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE **77**(2), 257–286 (1989)