Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Neural Networks for Sequential Data: a Pre-training Approach based on Hidden Markov Models

Luca Pasa [a,*], Alberto Testolin [b], Alessandro Sperduti [a]

[a] *Department of Mathematics, University of Padova, Italy*
[b] *Department of Developmental Psychology and Socialisation, University of Padova, Italy*

ARTICLE INFO

Article history:
Received 8 July 2014
Received in revised form
22 October 2014
Accepted 11 November 2014

Keywords:
Sequential Data
Hidden Markov Model
Recurrent Neural Networks
Pre-Training
Curriculum learning
RNNRBM

ABSTRACT

In the last few years, research highlighted the critical role of unsupervised pre-training strategies to improve the performance of artificial neural networks. However, the scope of existing pre-training methods is limited to static data, whereas many learning tasks require to deal with temporal information. We propose a novel approach to pre-training sequential neural networks that exploits a simpler, first-order Hidden Markov Model to generate an approximate distribution of the original dataset. The learned distribution is used to generate a smoothed dataset that is used for pre-training. In this way, it is possible to drive the connection weights in a better region of the parameter space, where subsequent fine-tuning on the original dataset can be more effective. This novel pre-training approach is model-independent and can be readily applied to different network architectures. The benefits of the proposed method, both in terms of accuracy and training times, are demonstrated on a prediction task using four datasets of polyphonic music. The flexibility of the proposed strategy is shown by applying it to two different recurrent neural network architectures, and we also empirically investigate the impact of different hyperparameters on the performance of the proposed pre-training strategy.

1. Introduction

A broad range of real-world applications involve learning over sequential data, e.g. classifying time series of heart rates (ECG) to decide if the data comes from a patient with heart disease, predicting the future value of a company stock traded on an exchange, interpreting a sequence of utterances in speech understanding, and predicting the secondary or tertiary protein structure from its DNA sequence. Sequence learning is a hard task and for this reason different approaches, tailored to specific data and task features (e.g., discrete vs. continuous valued sequences and classification vs. prediction tasks), have been developed. All these approaches can be grouped into three main categories: (i) feature-based approaches, which transform a sequence into a feature vector and then apply conventional vectorial-based methods (e.g., [1]); (ii) distance-based approaches, which employ a distance function measuring the similarity between sequences, e.g. Euclidean distance (e.g., [2]), edit-distance (e.g., [3]), dynamic time warping distance (e.g., [4]), or a kernel function (e.g., [5,6]); (iii) model-based approaches, such as using Hidden Markov Models (e.g., [7,8]), or Recurrent Neural Networks (e.g., [9,10]), to process sequences. Methods falling into the first category are successful only if *a priori* knowledge on the application domain can be used to select the most relevant sequence features for the task at hand. A notable example of these approaches, in the case of discrete valued sequences, is the use of short sequence segments of k consecutive symbols (k-grams) as features; a sequence is represented as a vector of the presence/absence (or frequencies) of the k-grams. The obtained vectors can then be fed into conventional learning machines, such as decision trees [11], Support Vector Machines [12], feed-forward neural networks [13], for any kind of learning task (classification, prediction, ranking, etc.). The drawback of these approaches is that the number of features to consider easily grows exponentially with that amount of *history/memory* the feature has to store (e.g., the size of k in k-grams). If *a priori* knowledge is not available for pruning the feature space, feature selection strategies (see [14]) need to be used. Moreover, ad-hoc strategies, such as discretization, are needed to deal with continuous valued sequences (e.g., [15]). Distance-based approaches treat each sequence as a single entity and exploit a sequence similarity function to determine how similar two sequences are. This information can then be used within an instance-based approach for learning (e.g., k-Nearest Neighbor [16]), or directly inside a kernel method if the used similarity function is a proper kernel. These approaches tend to be expensive from a computational point of view since computing the sequence similarity function usually involves a relevant computational burden (e.g., edit-distance [17]). Moreover, these approaches usually have problems to extrapolate the learned function to sequences that are longer than the ones used for training. Finally,

* Corresponding author.
E-mail address: lpasa@math.unipd.it (L. Pasa).

http://dx.doi.org/10.1016/j.neucom.2014.11.081
0925-2312/© 2015 Elsevier B.V. All rights reserved.

model-based approaches assume that the observed sequences, as well as the function to learn, have been generated by a law (or model). Because of that, they aim at reconstructing such model, with the goal of successfully extrapolating the learned function to the full sequence domain. Model-based approaches are typically computationally demanding, however, if a good approximation of the target model is learned, very good performances on the whole sequence domain can be obtained. Graphical models [18], and in particular Hidden Markov Models (HMMs), are often used as learning models. HMMs assume that each sequence item has been generated by hidden variables that are not directly observable. The way each sequence item observed at time t is generated is described by a (parametric) probability distribution which depends on the *state* at time t of the HMM, i.e. the values taken by the hidden variables at time t; moreover, another (parametric) probability distribution drives the way the values assigned to hidden variables change through time. Learning aims at tuning these probability distributions in order to make the observed sequences more likely to be generated when sampling from the model. A deterministic alternative to HMMs is given by Recurrent Neural Networks (RNNs), which can be understood as non-linear dynamical systems where learning is performed by using gradient-based approaches [13]. From an abstract computational point of view, given a graphical model for sequences, it is possible to specify a RNN which constitutes a specific deterministic implementation of that graphical model [19]. Due to their non-linearity, RNNs are potentially very expressive and powerful. Because of that, however, they are also difficult to train, mainly because temporal dependencies introduce constraints that limit the efficacy of gradient-based learning algorithms [20] as well as the parallelization of computation. Despite old [21] and recent developments [22,23], the computational burden to train RNNs still remains very high.

It is worth to note that models can also be used to define kernels (e.g., [24,25]). One very general way to exploit a generative model for defining a kernel is given by the Fisher kernel approach, originally proposed by [24]. The underpinning idea of Fisher kernel is to use the training data to create a generative model, e.g. a HMM, and then to define a kernel on sequences from the Fisher score vectors extracted from the generative model. Besides being computationally very expensive, Fisher kernel may suffer from quite bad feature representation due to maximum likelihood training which leads to develop a large number of very small gradients (data with high probability under the model) and a few very large ones (data with low probability under the model) [26]. Because of that, Fisher kernel may suffer when used for discriminative tasks. A technique that tries to correct this problem has been proposed in [26]. The basic idea is to learn the generative model parameters in such a way that the resulting embedding has a low nearest-neighbor error. This approach improves the discriminative performance at the expenses of an increased computational cost, which makes Fisher kernel very computational demanding when considering long sequences.

All the above methods have difficulties in learning over long sequences: feature-based approaches typically use features associated to the occurrence of short temporal/positional subsequences, i.e., *local* features, which fail to capture long-term dependencies. Distance-based approaches typically select a subset of the training sequences as reference to perform the desired computation[1]; because of that, the learned function typically has difficulties to deal with sequences that are longer than the ones used for training. Model-based approaches also tend to have problems capturing long-term dependencies, either because they have discrete finite memory (i.e., the number of different states in which a HMM can be), such as in the most commonly used versions of HMMs, or because learning algorithms fail to find the "right" setting for the parameters, such as in RNNs. Notwithstanding the difficulties in training RNNs, their computational power is so high (see, for example [28]) that it is worth to study new approaches to improve learning.

If we turn our attention to static data, recent advances in training *deep* neural networks, i.e. networks composed by many layers of non-linear processing units, now allow to reach state-of-the-art performance in complex machine learning tasks, such as image classification [29], speech recognition [30] and natural language processing [31]. One reason for this progress is due to the possibility of learning algorithms to enlarge the exploration of the parameter space thanks to the advent of new, high-performance parallel computing architectures, which exploit powerful graphic processors to significantly speed-up learning [32]. However, the breakthrough that allowed to effectively train large-scale "deep" networks has been the introduction of an unsupervised pre-training phase [33], in which the network is trained to build a generative model of the data, which can be subsequently refined using a supervised criterion (fine-tuning phase). Pre-training initializes the weights of the network in a region where optimization is somehow easier, thus helping the fine-tuning phase to reach better local optima. It might also perform some form of regularization, by introducing a bias towards good configurations of the parameter space [34].

Going back to sequences, an interesting research question is whether the benefits of pre-training could also be extended to the temporal domain. Up to now, the most popular approaches to pre-train sequential models do not take into account temporal dependencies (e.g., [30,35]) and only pre-train input-to-hidden connections by considering each item of the sequence as independent of the others. This pre-training strategy is clearly unsatisfactory, because by definition the items belonging to the same sequence are dependent on each other, and this information should be exploited also during pre-training.

In this paper, we propose an alternative pre-training method. Instead of using the same dataset for both the pre-training and the fine-tuning phases, we propose to use an HMM with a limited number of states to generate a new dataset, which represents an approximation of the target probability distribution. This simpler, smoothed dataset is then used to pre-train a more powerful non-linear model, which is subsequently fine-tuned using the original sequences. Importantly, this method does not require to develop any ad-hoc pre-training algorithm: we can adopt standard gradient descent learning, and apply it first on the approximate distribution and then on the original dataset.

We tested our method on a complex learning task, which requires to extract the structure of polyphonic music encoded using symbolic sequences (piano-roll MIDI format). We first applied the HMM pre-training on a recently proposed recurrent model [35] that has been shown to obtain state-of-the-art performance on a prediction task for the considered dataset. We then assessed the robustness and the generality of the method by applying it also to a classic recurrent neural network. As a final part of the study, we investigated the influence of various hyperparameters on the training performance. In particular, we performed extensive tests in order to understand how the number and the length of the sequences that populate the smoothed dataset affect both training times and the final quality of the non-linear model. Our results confirm the value of the proposed pre-training strategy, which allows us to learn an accurate model of the data in a significantly shorter time, sometimes also leading to improvements in prediction accuracy. A preliminary version of this work appeared in [36].

[1] For example, k-Nearest Neighbor exploits all training sequences or a subset obtained by editing the training set (e.g., [27]); also Support Vector Machines use a subset of the training set, i.e. the support sequences.

1.1. Related work

As mentioned before, in the last few years the benefits of pre-training strategies have been extensively investigated in the static domain, where the task requires, for example, to learn the spatial structure of images encoded as fixed-size vectors. In particular, it has been shown that the model performance can be significantly improved by first performing an unsupervised pre-training phase, where the goal is to learn an accurate generative model of the data by discovering its latent structure [37]. Notably, this approach allows us to train large-scale, hierarchical neural networks by using a large amount of training patterns that usually come without any attached label. After pre-training, the generative model can be further improved to perform classification tasks by using labeled information during a fine-tuning, supervised phase [33], although it has been shown that unsupervised pre-training on deep architectures might already allow us to build high-level, abstract representations of the input patterns [38,39] that can be easily read-out even by a linear classifier [40].

However, it is not straightforward to extend these pre-training techniques to the temporal domain, where the information is provided to the system in a sequential way and the latent structure must be gradually extracted over time [9]. Moreover, temporal dependencies introduce processing constraints that increase the computational complexity of learning algorithms. Although some interesting extensions of generative models have been proposed to deal with sequential data [41,42], they usually rely on approximate inference and learning algorithms. Research on pre-training strategies for sequential models therefore appears to be still very limited, and the most commonly used approaches usually ignore temporal dependencies during pre-training. For example, deep learning methods have been recently combined with HMMs to obtain state-of-the-art performance in acoustic modeling [30], but the pre-training phase did not consider the temporal structure of the data and only focused on modeling vectors of spectral features independently. A similar approach has been adopted to learn music sequences [35], where input-to-hidden connections of the generative model were pre-trained by considering each vector of notes as independent of the other time-steps. Temporal connections were instead initialized using the weights learned by a recurrent network trained with a second-order optimization method [22], but according to the authors this initialization had a minor impact on the final performance compared to the input-to-hidden initialization.

A final approach to pre-train sequential models, from which we take inspiration in our work, is related to the idea of curriculum learning [43]. The rationale behind this proposal is that human learning is facilitated if we start to approach simpler concepts before dealing with more complex ones [44]. In line with this idea, previous studies on simple recurrent networks [45] found that learning convergence can be improved if we first expose the system to simpler examples, and then gradually introduce more complex examples as learning proceeds. As explained in detail in Section 3, we propose a similar approach, on which a simplified set of patterns is effectively used to pre-train a sequential model. The more complex, original patterns are then used during fine-tuning, with the aim of refining and improving the acquired knowledge.

2. Background

In this section, we briefly review the formal characterization of the models considered in this paper. In the mathematical notation we adopted, scalar values are represented with lower case letters (l), vectors with lower case letters in bold ($\mathbf{v}$), matrices with upper case letters in bold ($\mathbf{M}$, and M_{ij} denotes its element in position ij), and variables with upper case letters (V). In each model discussed, $\mathbf{v}$ constitutes the input to the model. Since the models process sequential data, each time step in a time series is identified by an index represented using an apex in round brackets (e.g., $\mathbf{v}^{(t)}$ refers to the input values at time step t).

2.1. Hidden Markov Models

Most real-world information sources emit, at each time step t, observable events which are correlated with the internal state of the generating process. In our case the observable events are a vector of binary values. More importantly, the only available information is the outcome of the stochastic process at each time step t, i.e. the event $\mathbf{v}^{(t)}$, while the state of the system is unobservable (*hidden*). Hidden Markov Models (HMMs) allow modeling general stochastic processes where the state transition dynamics is disentangled from the observable information generated by the process. The state-transition dynamics, which is non-observable, is modelled by a Markov chain of discrete and finite latent variables referred to as the *hidden states*.

The dependency relationships among the different variables involved are typically represented by a graphical model, as exemplified for the HMM in Fig. 1: the hidden states are latent variables $H^{(t)}$, while the sequence elements $V^{(t)}$ are observed. The conditional dependencies represented by the arrows $H^{(t)} \to V^{(t)}$ indicate that the observed element at time t of the sequence is generated by the corresponding hidden state H_t through the *emission distribution*:

$$\mathbf{e}_{h^{(t)}}(\mathbf{v}^{(t)}) = P(V^{(t)} = \mathbf{v}^{(t)} | H^{(t)} = \mathbf{h}^{(t)}).$$

The joint distribution of the observed sequence $\mathbf{v} = \mathbf{v}^{(1)}, \dots, \mathbf{v}^{(T)}$ and associated hidden states $\mathbf{h} = \mathbf{h}^{(1)}, \dots, \mathbf{h}^{(T)}$ can be written as

$$P(V = \mathbf{v}, H = \mathbf{h}) = P(\mathbf{h}^{(1)})P(\mathbf{v}^{(1)} | \mathbf{h}^{(1)}) \prod_{t=2}^{T} P(\mathbf{h}^{(t)} | \mathbf{h}^{(t-1)})P(\mathbf{v}^{(t)} | \mathbf{h}^{(t)}). \quad (1)$$

The actual parametrization of the probabilities in Eq. (1) depends on the form of the observation and hidden states variables. A stationary hidden states chain, with n states, is regulated by an $n \times n$ matrix of *state-transitions* $A_{ij} = P(H^{(t)} = i | H^{(t-1)} = j)$ and by an n-dimensional vector of *initial state* probabilities $\boldsymbol{\pi}_i = P(H^{(t)} = i)$, where i, j are drawn from $\{1, \dots, n\}$. Moreover, for discrete sequence observations $\mathbf{v}^{(t)} \in \{1, \dots, m\}$ (which is the case we are interested in here), the emission distribution is an $m \times n$ emission matrix $\mathbf{E}$ with elements

$$\mathbf{e}_i(k) = E_{ki} = P(V^{(t)} = k | H^{(t)} = i).$$

The most common tasks performed when using an HMM are (i) to compute the most likely sequence of states given an observed sequence; (ii) to train a model, which consists in finding the parameters (emission probabilities, transition probabilities, and initial state probability) that maximize the probability of the observed sequences contained in a training set, given the model. The first task is achieved using the Viterbi algorithm [7], while training an HMM is usually performed using the Baum–Welch algorithm [46].

Probabilistic models of sequential data with hidden variables, which act as an internal state, can capture the temporal

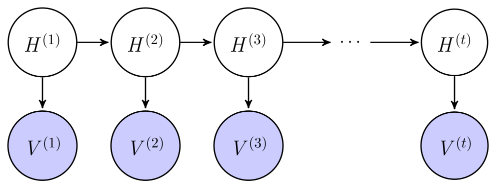


Fig. 1. Graphical model corresponding to a first-order Hidden Markov Model, where the observable variables $V^{(t)}$ are driven by the hidden states $H^{(t)}$.

dependencies between elements of a sequence by exploiting different types of internal representations. HMMs use a *localist* state representation, where the history of a time series is encoded using a discrete k-state multinomial distribution, where k is the number of hidden states in the model. This implies that in order to model n bits of information about the past history, 2^n hidden states are required. To avoid this exponential explosion, more powerful and expressive internal representations can be exploited. In particular, the neural network models presented in the following sections rely on *distributed* representations, where the history of a time series is encoded by a pattern of activity distributed over many hidden variables, and each variable is involved in representing many different entities [47]. This componential representation allows us to greatly increment the representational capacity of the model, which can efficiently encode history information using a linear number of components.

2.2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a particular type of neural networks designed to model sequential data. They are composed of three separate layers of units: one input layer, one hidden layer and one output layer. Each layer consists of several units (neurons) that are not connected to each other, but that are connected to all units in the next layer. Connections are weighted and the weights represent the model parameters. At each time step the corresponding elements of the sequence are given as input to the network through the input layer. One or more hidden layers are used to encode the latent features of the data, and the last hidden layer is then connected to the output layer, which is used to encode the desired network response. In our case, the task consists in predicting, at time t, the element of the sequence that should be generated at the next time step $t+1$.

Let us consider a network with just one hidden layer. If we denote with $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(T)} \in \mathbb{R}^n$ the sequence of input vectors, $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \ldots, \mathbf{h}^{(T)} \in \mathbb{R}^m$ the sequence of hidden states and $\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \ldots, \mathbf{o}^{(T)} \in \mathbb{R}^k$ the sequence of output vectors, the computation performed by an RNN at time t can be described by the following equations:

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_{hv}\mathbf{v}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h),$$
$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_{oh}\mathbf{h}^{(t)} + \mathbf{b}_o),$$

where σ is the component-wise logistic function,[2] $\mathbf{b}_h \in \mathbb{R}^m$ and $\mathbf{b}_o \in \mathbb{R}^k$ are the biases of hidden and output units, respectively, and $\mathbf{W}_{hv} \in \mathbb{R}^{m \times n}$, $\mathbf{W}_{oh} \in \mathbb{R}^{k \times m}$, and $\mathbf{W}_{hh} \in \mathbb{R}^{m \times m}$ are the input-to-hidden, hidden-to-output and hidden-to-hidden weight matrices, respectively. As mentioned above, we are interested in using the RNN in a prediction task, where a subsequence $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(\tau)}$, with $0 \le \tau < T$, is given as input to the network and the goal is to predict the element observed at the $(\tau+1)$th time step, which corresponds to the activation of the output units: $\mathbf{o}_{pred} = o^{(\tau+1)}$. In our case, $\forall t$, $\mathbf{v}^{(t)} \in \{0,1\}^n$, hence the input and output values will be sequences of 0 and 1.

The standard way to train RNNs is via the back-propagation through time (BPTT) algorithm [48], either in a batch mode or using a stochastic gradient descent (SGD) approach. Unfortunately, the performance of BPTT decreases in the presence of long-term temporal dependencies due to the vanishing gradient phenomenon [20]. Two main possible remedies to this problem have been proposed: the first one is called Long-Short Terms memory (LSTM) [21] and it consists in extending the model by using special linear memory units, while a more recent proposal relies on an Hessian-Free (HF) optimization algorithm [22]. In our experiments, we trained the networks using the BPTT algorithm, where the gradient at each time step t was computed according to the cross entropy cost function:

$$crossEntropy(\mathbf{o}^{(t)}, \mathbf{d}^{(t)}) = \sum_{j=1}^{k} -o_j^{(t)} \log(d_j^{(t)}) - (1-o_j^{(t)}) \log(1-d_j^{(t)}),$$

where $\mathbf{o}^{(t)}$ is the prediction of the network and $\mathbf{d}^{(t)}$ is the desired target at time t. Moreover, two different regularization functions are added to the cost function to improve generalization:

$$l_1(t) = ||\mathbf{W}_{hh}^{(t)}||_1 + ||\mathbf{W}_{hv}^{(t)}||_1 + ||\mathbf{W}_{oh}^{(t)}||_1,$$
$$l_2(t) = ||\mathbf{W}_{hh}^{(t)}||_2^2 + ||\mathbf{W}_{hw}^{(t)}||_2^2 + ||\mathbf{W}_{oh}^{(t)}||_2^2.$$

where $\mathbf{W}_{xy}^{(t)}$ denotes the weight matrix $\mathbf{W}_{xy}$ at time step t. The final cost computed at time t is

$$cost(t) = crossEntropy(\mathbf{o}^{(t)}, \mathbf{d}^{(t)}) + l_1(t) + l_2(t).$$

2.3. Recurrent Neural Networks with Restricted Boltzmann Machines

The Recurrent Neural Network-Restricted Boltzmann Machine (RNN-RBM) [35] is a sequential neural network that combines the best features of RNNs, which are particularly effective in learning temporal dependencies, within an RBM, which can model complex and multi-modal distributions. The RNN-RBM network is similar to the RTRBM (Recurrent Temporal Recurrent Temporal Restricted Boltzmann Machine) [41]. However, instead of exploiting a simple connection between the RBM hidden units of two contiguous time-steps, it hinges on the hidden units of an RNN to keep track of the relevant temporal information, thus allowing the encoding of long-term temporal dependencies. RNN-RBMs are nonlinear stochastic models, for which the joint probability distribution of hidden and input units is defined as

$$P(\mathbf{v}^{(t)}, \mathbf{h}^{(t)}) = \prod_{t=1}^{T} P(\mathbf{v}^{(t)}, \mathbf{h}^{(t)} | \mathbf{v}^{(t-1)}, \mathbf{v}^{(t-2)}, \ldots, \mathbf{v}^{(1)}, \hat{\mathbf{h}}^{(t-1)}, \hat{\mathbf{h}}^{(t-2)}, \ldots, \hat{\mathbf{h}}^{(1)})$$

where $\hat{\mathbf{h}}^{(t)} = \sigma(\mathbf{W}_2\mathbf{v}^{(t)} + \mathbf{W}_3\hat{\mathbf{h}}^{(t-1)} + \mathbf{b}_{\hat{\mathbf{h}}})$ and $\mathbf{v}^{(t)}$, $\mathbf{h}^{(t)}$ and $\hat{\mathbf{h}}^{(t)}$ represent, respectively, the input units, the RBM-hidden units and the RNN-hidden units, whereas $\mathbf{b}_{\hat{\mathbf{h}}}$ represents the RNN-hidden unit biases (for a graphical representation, see Fig. 2). This type of network is harder to train compared to RNNs and RTRBMs, so it requires an ad-hoc learning algorithm. The idea is to propagate the value of hidden units $\hat{\mathbf{h}}^{(t)}$ in the RNN-part of the network and then to use it to dynamically adjust some of the parameters of the RBM-part. Specifically, time-variant biases for RBM are derived by the hidden units of the RNN according to the following equations:

$$\mathbf{b}_h^{(t)} = \mathbf{b}_{\hat{\mathbf{h}}} + \mathbf{W}'\hat{\mathbf{h}}^{(t-1)},$$
$$\mathbf{b}_v^{(t)} = \mathbf{b}_{\hat{\mathbf{h}}} + \mathbf{W}''\hat{\mathbf{h}}^{(t-1)}.$$

The RBM-part of the network can then be trained by using Gibbs sampling and the contrastive divergence algorithm [49]. In particular, the model expectations on visible units $\mathbf{v}^{(t)*}$ can be estimated using block Gibbs sampling, and the log-likelihood gradient with respect to the RBM weights $\mathbf{W}$ and biases can be computed by contrasting the model's expectations with the observed data distribution. The log-likelihood gradient of the RBM-part of the network can then be propagated backward through time by using the BPTT algorithm [48], in order to estimate the gradient with respect to the RNN-part parameters (for details on the learning procedure, see [35]).

To obtain a good model of the data, a pre-training phase is usually performed. In particular, the authors of the RNN-RBM separately pre-trained the RBM-part and the RNN-part. Specifically, pre-training of the RBM-part of the model is performed by using contrastive divergence on information associated to single

[2] For our experiments we used the hyperbolic tangent.

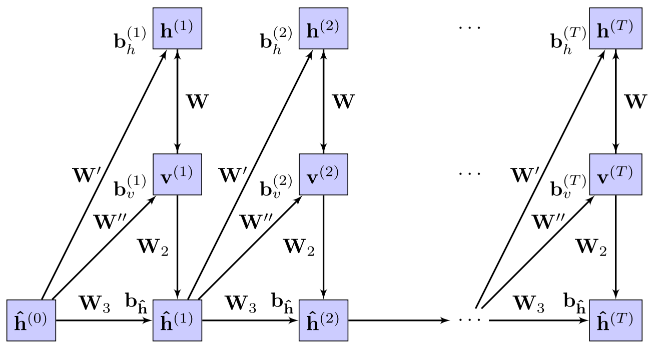


Fig. 2. Schematic representation of the RNN-RBM (see [35] for details).

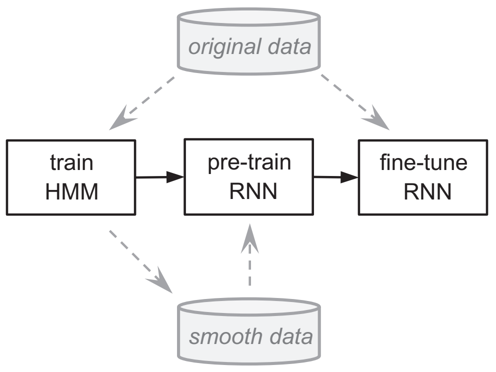


Fig. 3. Flow chart of the proposed HMM-based pre-training method for RNN. The flow chart is the same if an RNN-RTRBM model is used in place of an RNN: it is sufficient to replace the label RNN with the label RNN-RBM in the picture (in fact, any sequential model could in principle be used as an alternative to the RNN).

time steps, while the RNN-part of the model can be pre-trained either by using SGD or Hessian-Free optimization, with the aim to better capture temporal dependencies.

3. Our pre-training approach

The pre-training method we propose relies on an approximation of the actual data distribution in order to drive the network weights in a better region of the parameter space. To this aim, a linear HMM is first trained on the real sequences (original data). We chose to use this type of probabilistic models because they are particularly efficient to train, and they have shown to be effective on many sequence learning problems [7], including music modeling [35]. After training, the HMM is used to generate a fixed number of sequences that will populate a new dataset (smooth data). The intuition is that the sequences generated by the linear model will constitute a smoothed, approximated version of the original sequences, but will nevertheless retain the main structure of the data. In order to generate a set of sequences from the HMM, we first select the starting hidden state according to the learned initial state probability distribution. The first element of the sequence is then sampled according to the emission distribution associated with that hidden state, and the same process is iteratively repeated by selecting the next target state according to the learned state transition probabilities. A straightforward, naive implementation of the random sampling process can be obtained by first computing the cumulative probability distribution from the target distribution and then selecting the element corresponding to a random number drawn from the interval [0,1]. Notably, our pre-training procedure does not need any form of bootstrapping from the original sequences, because the smooth dataset is generated by sampling the HMM in a completely unconstrained fashion. The simplified, HMM-generated dataset is then used to pre-train the more powerful nonlinear model, with the aim of transferring the knowledge acquired by the HMM to the recurrent neural network. The recurrent network pre-training phase uses the same algorithm that is used for the normal training phase. After completing the pre-training phase on the smooth dataset, the neural network is then fine-tuned using the original music sequences, in order to allow the nonlinear model to extract more complex structure from the data distribution. The pseudocode for the proposed HMM-based pre-training method is given in Algorithm 1, and a flow chart of the procedure is illustrated in Fig. 3.

Algorithm 1. Pseudocode for the proposed HMM-based pre-training. At the beginning, several parameters need to be initialized: n and l represent, respectively, the number and length of the sequences generated by the HMM, θ_{hmm} represents the training hyperparameters for the HMM (e.g., number of hidden states) and θ_{rnn} represents the training hyperparameters for the recurrent neural network (e.g., the number of hidden units and the learning rate).

```
1: begin
2:     set $n, l, \theta_{hmm}, \theta_{rnn}$;
3:     $hmm \leftarrow train\_hmm(originalData, \theta_{hmm})$;
4:     $smoothData \leftarrow sample(hmm, n, l)$;
5:     $rnn \leftarrow random\_initialization$;
6:     $rnn \leftarrow train\_rnn(smoothData, \theta_{rnn}, rnn)$;
7:     $rnn \leftarrow train\_rnn(originalData, \theta_{rnn}, rnn)$;
8:     $return(rnn)$;
9: end.
```

Notably, the introduction of the pre-training phase before fine-tuning does not significantly affect the computational cost of the whole learning procedure, because both learning and sampling in HMMs can be performed in an efficient way. In particular, our method performs three main steps during pre-training: train the HMM, generate the smooth dataset and pre-train the nonlinear network. The training phase for the HMM (step 3) is performed using the Baum–Welch algorithm, which has a complexity of order $O(N^2T)$ for each iteration and observation, where T is the length of the observation used to train the model, and N is the number of states in the HMM [50]. The smooth sequences generation (step 4) is performed using the Viterbi algorithm. For each generated sequence, this algorithm has a computational complexity of order $O((NF)^2T)$, where F is the size of the input at a single time step, T is the length of the generated sequence and N is the number of states in the HMM. Finally, step 6 consists in pre-training the recurrent neural network. In order to perform the pre-training phase we exploit the standard training algorithm, therefore the complexity of this step depends on the type of network that we aim to use. Moreover, it should be noted that the improved initialization of the network weights could allow us to speed-up convergence during the fine-tuning phase. The number and length of the sequences generated by the HMM are important parameters for which it is difficult to make an operational choice. A rule of thumb is to choose them in accord with the training set statistics. In Section 4.4 we experimentally explore some of these issues.

4. Experimental assessment

We tested our pre-training method on both RNNs and RNN-RBMs on a next-step prediction task over four datasets containing polyphonic music sequences. Due to the very high computational demand of RNN-RBMs, the assessment of the proposed method for this type of network is performed using only two out of the four available datasets.

4.1. Datasets

The polyphonic music datasets considered in our study contain different musical genres, which involve varying degrees of complexity in the temporal dependencies that are relevant for the prediction task. Specifically, the Nottingham dataset contains folk songs, characterized by a small number of different chords and a redundant structure; the Piano-midi.de dataset is an archive of classic piano music, containing more complex songs with many different chords; the Muse Data and JSB Chorales datasets contain, respectively, piano and orchestral classical music; moreover, the JSB chorales are redundant and all composed by a single author, so the songs style is largely shared by different patterns.

In Table 1 we report the main datasets statistics, including the size of the training and test sets, and the maximum, minimum, and average length of the contained sequences. All data that we used for the experiments was in MIDI format. Each sequence was converted into a binary format, where each time step was encoded using a vector of 88 binary values that represented the 88 notes spanning the whole piano range (from A0 to C8). In particular, each binary value was set to 1 if the note was played at the current time step, and 0 otherwise. The number of notes played simultaneously varied from 0 to 15. The output prediction was represented using the same format of the input.

4.2. Experimental setup

We tested the generality of our approach by validating it on the two different types of sequential networks described above, i.e., RNNs and RNN-RBMs.

Due to the large number of hyperparameters involved in the computation (e.g., the number of hidden variables for both HMMs and recurrent networks, and the number and length of the sequences generated by the HMM), a systematic exploration of the parameters space would be unfeasible. We therefore adopted a “probe” approach, where the model-dependent parameters that were already known to give good results for the used datasets were kept fixed, while the remaining parameters were probed for few different values. Specifically, the fixed parameters for both the sequential networks are the number of pre-training and training epochs (which are set to 100 and 200, respectively, for the RNN-RBM and to 2500 and 5000 for the RNN). Moreover, for both the RNN and the RNN-RBM we used a learning rate of 0.001. Only for the RNN-RBM, the number of hidden units was fixed to 150 for the RBM-part, and to 100 for the RNN-part. For the pre-training of both networks we used an HMM that was trained for 10,000 iterations using the Baum–Welch algorithm. This setting might not be ideal, since a large portion of the parameter space is left unexplored. Nevertheless, the few parameter configurations we have probed were enough to provide evidence that the proposed approach is robust with respect to the learning parameters, since for all the considered datasets we obtained significant improvements, either in terms of accuracy or in terms of computation time.

Following the procedure outlined above, we started our investigation with the RNN-RBM network. Specifically, we explored the effect of using different numbers of states for the HMM, while keeping the number of HMM generated sequences fixed at 500, all of length 200. As mentioned above, since training of RNN-RBMs is very time consuming, we restricted our experiments to the Nottingham and Piano-midi.de datasets. In order to evaluate the experimental results, we made a comparison of our method against the pre-existing pre-training approaches, i.e. SGD and HF, in terms of accuracy (calculated according to the method proposed in [51]) and computation time. For RNNs, we investigated the impact of our pre-training approach on the learning process by varying the number of hidden units and the number (and length) of the sequences generated by the HMM, while keeping the number of hidden states of the HMM fixed to 10 (i.e. the number of hidden states that constituted to the best trade-off between speed of training and quality of the final result in the RNN-RBM experiments).

All the experiments were run using the Theano software [52], on an Intel© Xeon© CPU E5-2670 @2.60 GHz with 128 GB of RAM, equipped with an NVidia© K20 GPU.

4.3. Experimental results

Learning the structure of polyphonic music with HMMs was challenging due to the potential exponential number of possible configurations of notes that can be produced at each time step, which would cause the alphabet of the model to have an intractable size. We fixed this issue by only considering the configurations that were actually present in each dataset, which reduced the complexity of the alphabet but at the same time maintained enough variability to produce realistic samples. We assessed the accuracy of the models on the prediction task defined in [35], i.e. prediction at time $t-1$ of the next sequence input at time t, using the same evaluation metric and model parameters. We also collected the total training times, which are composed of both the pre-training time and the fine-tuning time.

For the RNN-RBM, we compared our pre-training method with that used by the authors of the model. Pre-training was performed for 100 epochs, and fine-tuning for 200 epochs. Total training

Table 1
Datasets statistics, including the number of sequences contained in each dataset.

Dataset	Subset	# Samples	Max length	Min length	Avg length
Nottingham	**Training**	195	641	54	200.8
	Test	170	1495	54	219
	Validation	173	1229	81	220.3
Piano-midi.de	**Training**	87	4405	78	812.3
	Test	25	2305	134	694.1
	Validation	12	1740	312	882.4
MuseData	**Training**	524	2434	9	474.2
	Test	25	3402	70	554.5
	Validation	135	2523	94	583
JSB Chorales	**Training**	229	259	50	120.8
	Test	77	320	64	123
	Validation	76	289	64	121.4

times and prediction accuracies for the HMM, SGD and HF pre-trainings are reported in Fig. 4 for the Nottingham dataset, and in Fig. 5 for the Piano-midi.de dataset. In both figures, information about pre-training time is reported in the figure legend, after each curve label. Specifically, each label is followed by a couple of values in parenthesis that represent, for our pre-training approach, the time required for training the HMM and the time required for the pre-training phase, while for the other pre-training approaches the values represent the time required for pre-training the RBM-part and the RNN-part of the network by using HF or SGD.

In general, different pre-training methods led to similar accuracies (both for training and test sets) at the end of the fine-tuning phase. However, in the more complex Piano-midi.de dataset our pre-training obtained slightly better results. Regarding convergence speed, the HMM method always significantly outperformed the others (e.g., it saved more than 8 h of computing in the Nottingham dataset). We also assessed the change in performance as the number of HMM states varies. As expected, using a smaller number of hidden states (≤ 25) reduced pre-training times. Interestingly, this did not affect the quality of the models after the fine-tuning phase, which still converged to good solutions. Using a HMM with 50 states, instead, was detrimental due to the slow convergence speed of the HMM training. Thus, the HMM pre-training seems to perform a better initialization of the network, which allowed us to improve convergence speed also during the fine-tuning phase. For example, the network pre-trained with the HMM reached the highest accuracy after only 110/120 epochs, compared to 200 epochs required by the other methods. It is worth noting that the accuracies measured directly on the HMMs were always fairly low, at best approaching 53% in the Nottingham dataset and 10.1% in the Piano-midi.de dataset.

Concerning the experiments involving RNNs, we recall that a single HMM with 10 hidden states for each dataset was used to generate all the pre-training sequences of the corresponding dataset. Three different network architectures were used containing 50, 100, and 200 hidden units. Pre-training for the network with 50 hidden units was performed by using both 25 generated sequences of length 25 and 500 generated sequences of length

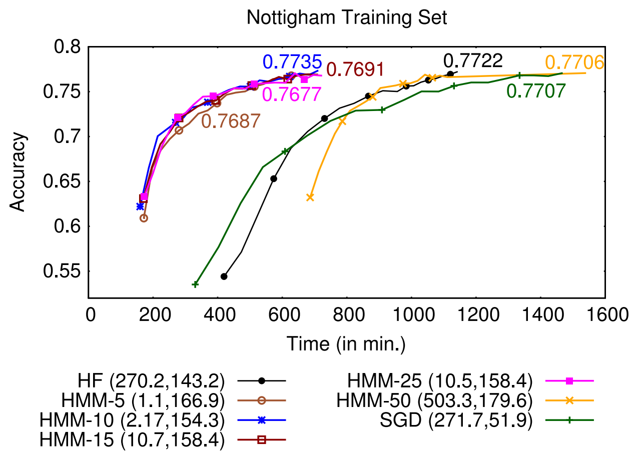


Fig. 4. Accuracy and running times of the tested pre-training methods, measured on the Nottingham dataset. Each curve is identified by a label followed by a couple of execution times in parenthesis: the pattern `HMM-n (time`$_1$`,time`$_2$`)` refers to our approach, where `n` is the number of hidden states used for the HMM, `time`$_1$ is the training time for the HMM, `time`$_2$ is the pre-training time; with the label `HF` (or `SGD`) we represent the Hessian Free (or Stochastic Gradient Descent) pre-training performed in `time`$_1$ time for the RBM-part, and `time`$_2$ time for the RNN-part. The final test set performance for each method is reported at the end of each corresponding curve.

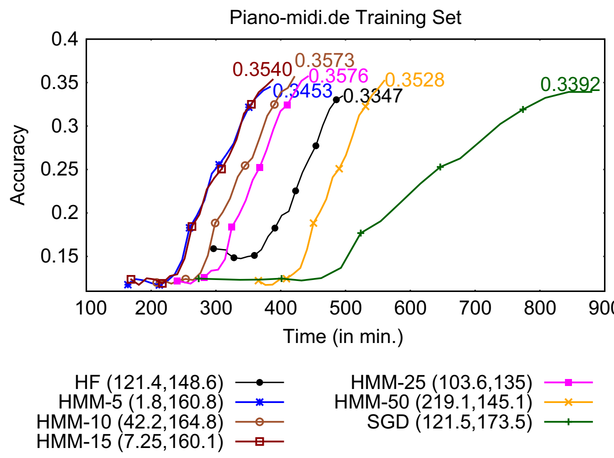


Fig. 5. Accuracy and running times of the tested pre-training methods, measured on the Piano-midi.de dataset. Each curve is identified by a label followed by a couple of execution times in parenthesis: the pattern `HMM-n (time`$_1$`,time`$_2$`)` refers to our approach, where `n` is the number of hidden states used for the HMM, `time`$_1$ is the training time for the HMM, `time`$_2$ is the pre-training time; with the label `HF` (or `SGD`) we represent the Hessian Free (or Stochastic Gradient Descent) pre-training performed in `time`$_1$ time for the RBM-part, and `time`$_2$ time for the RNN-part. The final test set performance for each method is reported at the end of each corresponding curve.

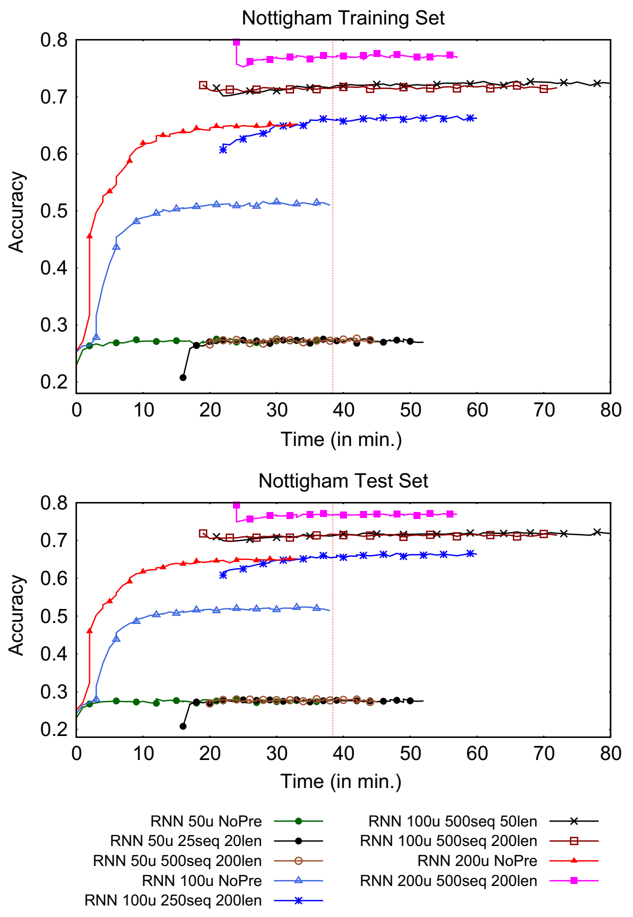


Fig. 6. Training (top) and test (bottom) accuracy and running times for RNNs on the Nottingham dataset. Each curve is identified by the label `RNN` followed by three or two identifiers: the three identifiers pattern `n`$_1$`U n`$_2$` n`$_3$ refers to our approach, where `n`$_1$ is the number of used hidden units for RNN, `n`$_2$ is the number of sequences generated by an HMM with 10 states, and `n`$_3$ is the length of such sequences; the two identifiers pattern `nU NoPre` refers to a RNN with standard random initialization and `n` hidden units. A dotted vertical line is used to mark the end of training of RNNs, with no pre-training, after 5000 epochs. The same number of epochs is used to train RNNs with pre-training. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

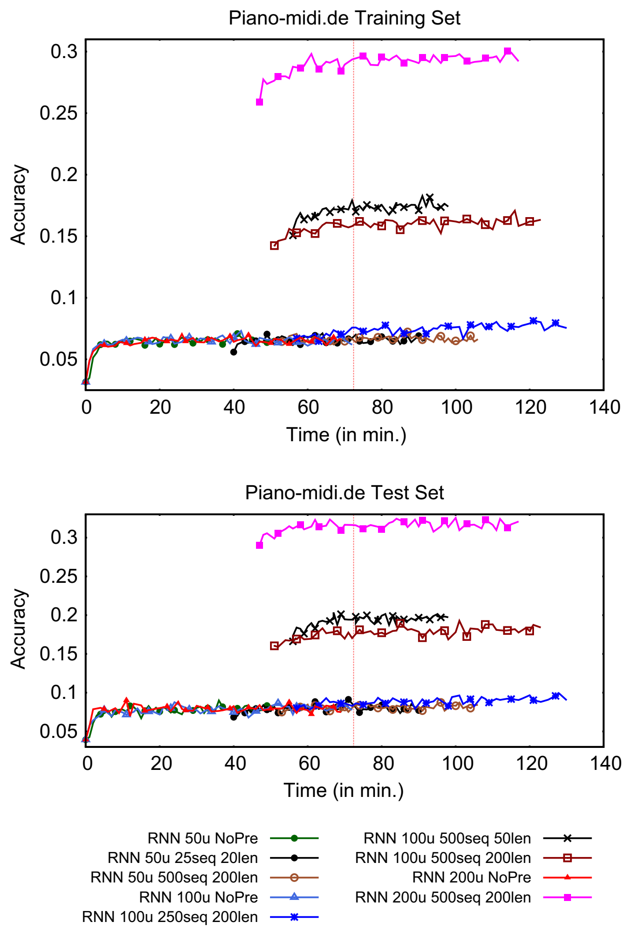


Fig. 7. Training (top) and test (bottom) accuracy and running times for RNNs on the Piano-midi.de dataset. Each curve is identified by the label RNN followed by three or two identifiers: the three identifiers pattern n_1U n_2 n_3 refers to our approach, where n_1 is the number of used hidden units for RNN, n_2 is the number of sequences generated by an HMM with 10 states, and n_3 is the length of such sequences; the two identifiers pattern nU NoPre refers to a RNN with standard random initialization and n hidden units. A dotted vertical line is used to mark the end of training of RNNs, with no pre-training, after 5000 epochs. The same number of epochs is used to train RNNs with pre-training. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

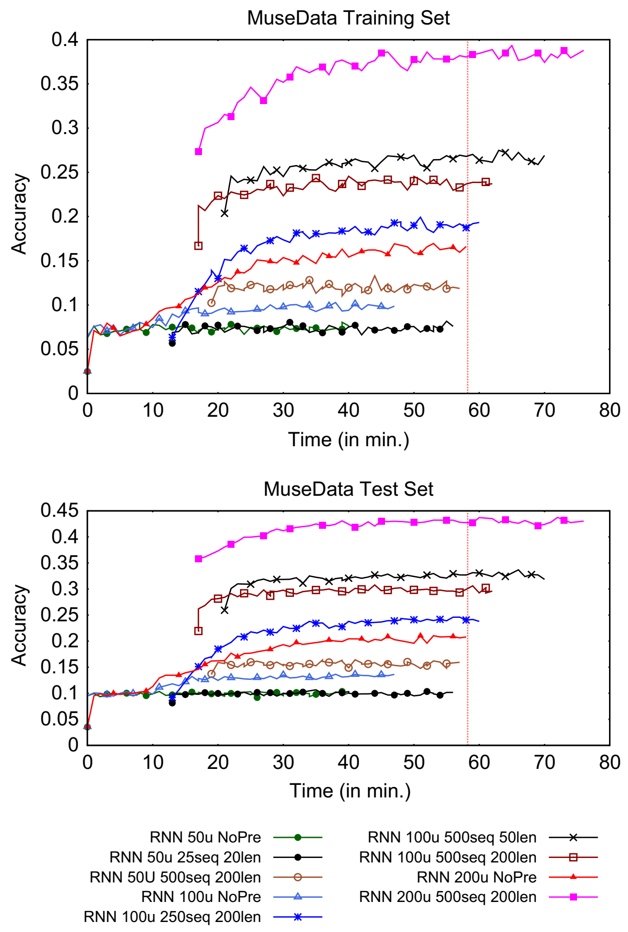


Fig. 8. Training (top) and test (bottom) accuracy and running times for RNNs on the Muse dataset. Each curve is identified by the label RNN followed by three or two identifiers: the three identifiers pattern n_1U n_2 n_3 refers to our approach, where n_1 is the number of used hidden units for RNN, n_2 is the number of sequences generated by an HMM with 10 states, and n_3 is the length of such sequences; the two identifiers pattern nU NoPre refers to a RNN with standard random initialization and n hidden units. A dotted vertical line is used to mark the end of training of RNNs, with no pre-training, after 5000 epochs. The same number of epochs is used to train RNNs with pre-training. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

200. For the architecture with 100 hidden units, three different pre-training configurations were used: 250 generated sequences of length 200, 500 generated sequences of length 50, 500 generated sequences of length 50. Finally, for the architecture with 200 hidden units, 500 generated sequences of length 200 were used.

In Figs. 6–9 we compare, for all the four datasets, the learning curves obtained for the training and test sets by our pre-training method versus the learning curves obtained without pre-training. The starting point of the curves involving pre-training takes into consideration the pre-training time. Since the number of training epochs for the RNNs is fixed to 5000 and it does not depend on the presence of pre-training, in the plots we highlighted, via a vertical dotted line, the point in time where the slowest RNN without pre-training finished training. From the plots it can be observed that some runs of RNNs using the same number of hidden units have a significant difference in execution time. We believe that this is mainly due to the Theano dynamic C code generation feature, which can, under favorable conditions, speed-up computation in a significant way.

From the learning curves we can notice that the performance on the test sequences is very similar to the behavior on the training sequences (i.e., the models did not overfit). Concerning the effectiveness of pre-training, it is clear that using just 50 hidden units does not lead to any benefit, while pre-training turns to be quite effective for networks with 100 and 200 hidden units, allowing the RNN to reach very good generalization performances. Moreover, using many short generated sequences seems to be the best choice for all datasets, as clearly demonstrated by the networks using 100 hidden units and 3 different configurations for the generated sequences. For the Nottingham and JSB databases, pre-training seems to reach a very good starting point that is subsequently *lost* by training with the original dataset. This is an interesting behavior that needs to be more carefully investigated in future studies. At the time marked by the vertical line (i.e. when the slowest RNN without pre-training finished training), all the curves associated to RNNs adopting pre-training reached a performance that was very close to the final one. This suggests that our pre-training can reach significantly better solutions using the same amount of time used by RNNs with no pre-training.

Finally, it is interesting to compare the test performances reached by the RNNs with the results obtained in [35]. In Table 2 we have reported, for each dataset, the test performances of their Hidden Markov Models (HMM) using Gaussian Mixture Models

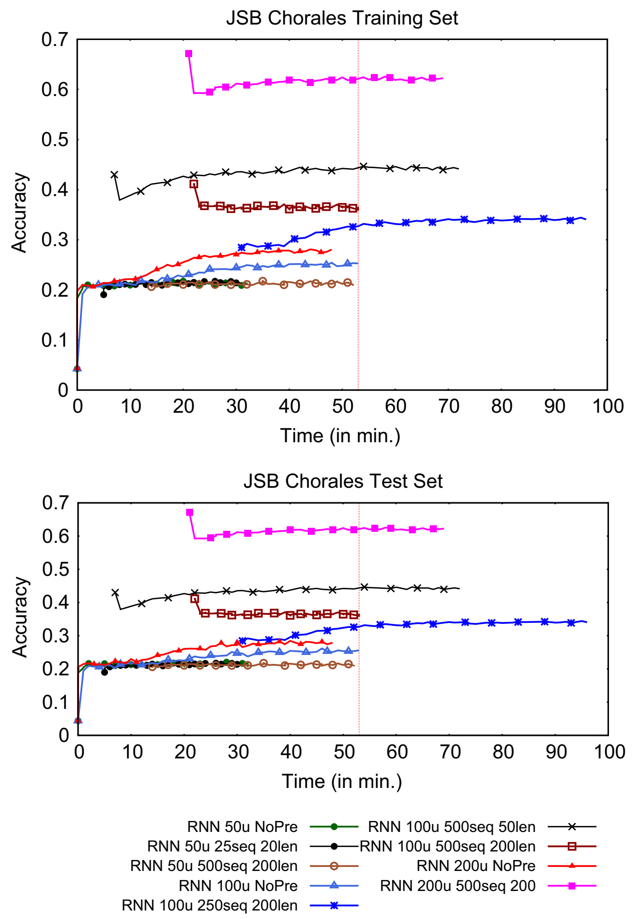


Fig. 9. Training (top) and test (bottom) accuracy and running times for RNNs on the JSB dataset. Each curve is identified by the label RNN followed by three or two identifiers: the three identifiers pattern n_1U n_2 n_3 refers to our approach, where n_1 is the number of used hidden units for RNN, n_2 is the number of sequences generated by an HMM with 10 states, and n_3 is the length of such sequences; the two identifiers pattern nU NoPre refers to a RNN with standard random initialization and n hidden units. A dotted vertical line is used to mark the end of training of RNNs, with no pre-training, after 5000 epochs. The same number of epochs is used to train RNNs with pre-training. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

Table 2
Accuracy results for state-of-the-art models [35] vs our pre-training approach. The acronym GMM+HMM is used to identify Hidden Markov Models (HMM) using Gaussian Mixture Models (GMM) indices as their state. The acronym (w. HF) is used to identify an RNN trained by Hessian Free optimization.

Dataset	Model	ACC%
Nottingham	GMM + HMM	59.27
	RNN (w. HF)	62.93 (66.64)
	RNN-RBM	75.40
	PreT-RNN (200U 500 200)	**80.47**
Piano-midi.de	GMM + HMM	7.91
	RNN (w. HF)	19.33 (23.34)
	RNN-RBM	28.92
	PreT-RNN (200U 500 200)	**36.51**
MuseData	GMM + HMM	13.93
	RNN (w. HF)	23.25 (30.49)
	RNN-RBM	34.02
	PreT-RNN (200U 500 200)	**44.96**
JSB Chorales	GMM + HMM	19.24
	RNN (w. HF)	28.46 (29.41)
	RNN-RBM	33.12
	PreT-RNN (200U 500 200)	**67.36**

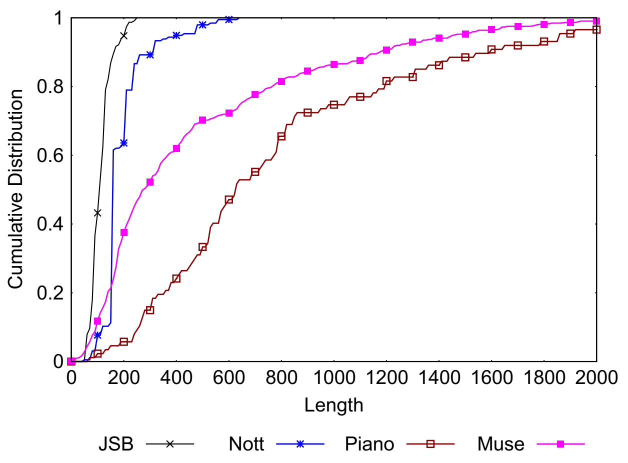


Fig. 10. Cumulative probability distribution for the lengths of sequences contained in the four considered datasets. Note that, in order to have a clear separation among the four curves, only lengths up to 2000 are considered in the plot. (For interpretation of the references to color in the text, the reader is referred to the web version of this paper.)

(GMM) indices as their state, their RNN (also with HF training) and their RNN-RBM networks, jointly with the test performances reached by our RNNs with pre-training (PreT-RNN) as selected by using the associated validation set. For all datasets we obtain significantly better results, especially when considering the HMM-based approach. The improvement is particularly large for the JSB dataset.

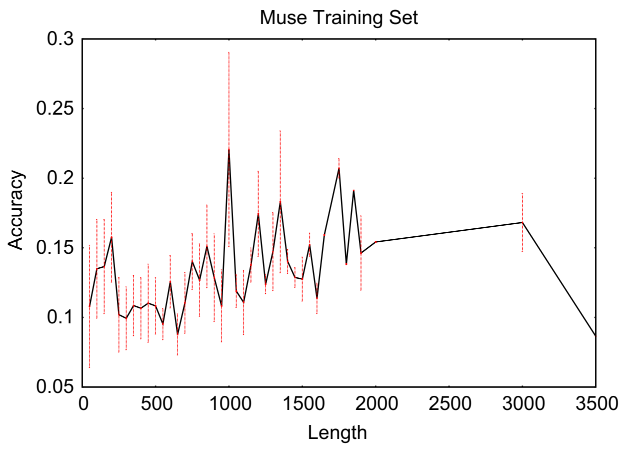
It can be also observed that our pre-training on the RNN-RBM architecture did not seem to improve on test performances. However, we recall that we used a much smaller number of hidden units with respect to the network used in [35], which may explain why no significantly better results are obtained for the Nottingham and Piano-midi.de datasets, although there was a significant improvement in training times.

4.4. Parameters setting

Even if our experimental investigation confirms the appeal of the proposed pre-training strategy, the optimal choice of the parameters involved in our method is still partially unexplored. In particular, in our experiments we fixed many learning parameters (e.g., learning rates and number of learning epochs) and only coarsely explored the best values for the other parameters. In this final section we briefly make some considerations that could help us to better understand what is the impact of some settings that directly affect our pre-training method, in particular, the length of the sequences generated by the HMM.

As shown in Figs. 6–9, the best final performance is clearly achieved when using more hidden units in the RNN (i.e., 200 units instead of 100). However, the results do not clearly characterize how the number and the length of the generated sequences affect the final RNN performance. Specifically, it seems that by sampling more sequences we usually obtain better results (compare, for example, the blue – 250 – and the red – 500 – lines in all the above mentioned figures). A possible reason for this phenomenon is that by sampling more sequences we add more variability to the pre-training sequences, which results in a better generalization. At the same time, we face a trade-off because adding more sequences to the smooth dataset also causes an increase in pre-training times.

The parameter representing the length of the sequences sampled from the HMM, instead, appears to be more subtle to optimize. In particular, it seems that we do not need to generate very long sequences from the HMM, because good performances



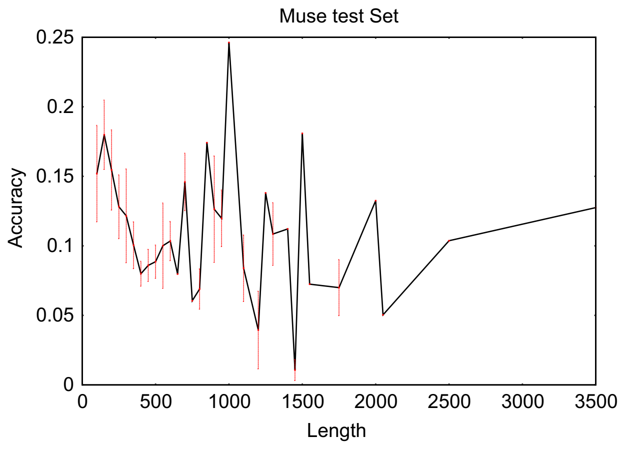


Fig. 11. Average accuracy of HMM on the sequences contained on Muse training and test sets, grouped in bins of size 50 over the length. The standard deviation for each bin is reported as well.

can already be obtained by using a length of only 50 (see black, crossed lines in the above mentioned figures). This result might be due to the statistics of the considered datasets, because most of the structure to be learned could be encoded in short sequences. The statistics reported in Table 1 only report maximum, minimum and mean values for the sequences contained in the four datasets, but such measures do not fully characterize the distribution of sequences length in the different sets. Indeed, as shown in Fig. 10 which reports the datasets cumulative distributions of sequence length, it appears that in both JSB and Nottingham datasets most of the sequences are fairly short.

The experimental results seem to suggest that a good choice for the length of the HMM generated sequences is to choose a value that covers about the 50% of the cumulative distribution over the lengths in the training set. However, the trade-off between the length of the sequences and the required time for pre-training must also be taken into account. Indeed, with longer sequences, the computational burden of pre-training in terms of time increases significantly. Besides, using long HMM generated sequences does not guarantee that the accuracy of the pre-trained network is going to improve. In fact, from the experimental results, it can be observed that networks pre-trained with 500 sequences of length 50 (black line with crosses) get a better accuracy with respect to networks pre-trained with 500 sequences of length 200 (brown line with square).

A possible explanation for the worse performance obtained when generating long sequences might be found in the short memory of HMMs, which would cause longer sequences to "drift away" from the correct structure as the generation proceeds. We tested this hypothesis by plotting in Fig. 11 the HMM[3] accuracy versus the length of sequences belonging to the Muse training and test datasets. The Muse datasets were chosen because they contain sequences with complex structures as well as a sufficient number of sequences to get reliable statistics. The plot is actually reporting the average accuracy of bins of size 50 over the length. The standard deviation for each bin is reported as well in the plot.

The plots do not seem to show the "drift away" effect described above, although it must be recognized that we do not know the memory size needed to cover all (or most) of the long-term dependencies which actually occur into the Muse datasets, so it is difficult to evaluate how many of the long-term dependencies have been identified by the HMM. However, the fact that many long sequences have better accuracy values than the average (0.1265 for the training set, and 0.129 for the test set) seems to be a good indication that there is not a memory issue with HMM.

5. Conclusions and future directions

In this paper, we proposed a novel method for pre-training recurrent neural networks, which consists in generating a smoothed, approximated dataset using a first-order HMM trained on the original data. When applied to a recently proposed recurrent neural network architecture on a prediction task involving polyphonic music, our HMM-based pre-training led to prediction accuracies comparable (and sometimes slightly better) than those obtained with currently available pre-training strategies, but requiring a significantly lower computational time. We also tested the method on a classic recurrent neural network, and also in this case the effectiveness of our approach was confirmed, obtaining a large improvement in all datasets.

It should be stressed that the proposed method does not need any ad-hoc adjustments of existing learning algorithms, because it consists in generating a simplified dataset that is used to initialize the parameters of the learner. Our pre-training strategy is therefore very general, and its benefits could be readily extended to pre-train many other types of sequential models.

Although in this paper we tried to explore the joint parameter space which includes both the pre-training parameters as well the recurrent neural network parameters, further research is needed to better understand how to reach the optimal setting for the number and length of sequences generated by the HMM. The empirical evidence we have collected in our experiments seems to suggest that using too long sequences is detrimental for the fine-tuning phase. It seems much better to use many short sequences. This may be understood as a way to avoid overfitting by the pre-training phase: the main mode of the data can be captured by short sequences, while long-term dependencies are left to the fine-tuning phase. As a consequence of that, the empirical evidence seems to suggest that it is not important to use a Hidden Markov Model with many hidden states. In fact, having many hidden states makes pre-training too slow and prone to introduce overfitting. A mathematical explanation of these empirical observations is needed, and it will constitute the main effort for future research in the topic.

References

[1] Nadia A. Chuzhanova, Antonia J. Jones, Steve Margetts, Feature selection for genetic sequence classification, Bioinformatics 14 (2) (1998) 139–143.
[2] Li Wei, Eamonn J. Keogh, Semi-supervised time series classification, in: Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20–23, 2006, 2006, pp. 748–753.
[3] Andrés Marzal, Enrique Vidal, Computation of normalized edit distance and applications, IEEE Trans. Pattern Anal. Mach. Intell. 15 (9) (1993) 926–932.

[3] We recall that the HMM is using 10 states.

[4] Xiaopeng Xi, Eamonn J. Keogh, Christian R. Shelton, Li Wei, Chotirat Ann Ratanamahatana, Fast time series classification using numerosity reduction, in: Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25–29, 2006, 2006, pp. 1033–1040.
[5] Christina S. Leslie, Rui Kuang, Fast string kernels using inexact matching for protein sequences, J. Mach. Learn. Res. 5 (2004) 1435–1455.
[6] Corinna Cortes, Patrick Haffner, Mehryar Mohri, Rational kernels: theory and algorithms, J. Mach. Learn. Res. 5 (2004) 1035–1062.
[7] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[8] Oksana Yakhnenko, Adrian Silvescu, Vasant Honavar, Discriminatively trained Markov model for sequence classification, in: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), Houston, Texas, USA, November 27–30, 2005,2005, pp. 498–505.
[9] Jeffrey L. Elman, Finding structure in time, Cognit. Sci. 14 (2) (1990) 179–211.
[10] Paolo Frasconi, Marco Gori, Andreas Kuechler, Alessandro Sperduti, From sequences to data structures: theory and applications, in: J. Kolen, S. Kremer (Eds.), A Field Guide to Dynamic Recurrent Networks, 2001, pp. 351–374.
[11] J. Ross Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Kluwer Academic Publisher, Boston, 1993.
[12] Corinna Cortes, Vladimir Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[13] Yves Chauvin, David E. Rumelhart (Eds.), Backpropagation: Theory, Architectures, and Applications, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.
[14] Isabelle Guyon, André Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[15] Lexiang Ye, Eamonn J. Keogh, Time series shapelets: a new primitive for data mining, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28–July 1, 2009, 2009, pp. 947–956.
[16] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theor. 13 (September (1)) (2006) 21–27.
[17] Eric Sven Ristad, Peter N. Yianilos, Learning string-edit distance, IEEE Trans. Pattern Anal. Mach. Intell. 20 (5) (1998) 522–532.
[18] Daphne Koller, Nir Friedman, Probabilistic Graphical Models—Principles and Techniques, MIT Press, Cambridge, MA, USA, 2009.
[19] Paolo Frasconi, Marco Gori, Alessandro Sperduti, A general framework for adaptive processing of data structures, IEEE Trans. Neural Netw. 9 (5) (1998) 768–786.
[20] Yoshua Bengio, Patrice Y. Simard, Paolo Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.
[21] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[22] J. Martens, I. Sutskever, Learning recurrent neural networks with Hessian-free optimization, in: International Conference on Machine Learning, 2011, pp. 1033–1040.
[23] Yoshua Bengio, Nicolas Boulanger-Lewandowski, Razvan Pascanu, Advances in optimizing recurrent networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Washington, DC, USA, 2013, pp. 8624–8628.
[24] T.S. Jaakkola, D. Haussler, Exploiting generative models in discriminative classifiers, in: Advances in Neural Information Processing Systems, 1999, pp. 487–493.
[25] Fabio Aiolli, Giovanni Da San Martino, Markus Hagenbuchner, Alessandro Sperduti, Learning nonsparse kernels by self-organizing maps for structured data, IEEE Trans. Neural Netw. 20 (12) (2009) 1938–1949.
[26] Laurens van der Maaten, Learning discriminative Fisher kernels, in: Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28–July 2, 2011, 2011, pp. 217–224.
[27] Peter E. Hart, The condensed nearest neighbor rule (corresp.), IEEE Trans. Inf. Theory 14 (3) (1968) 515–516.
[28] Jérémie Cabessa, Hava T. Siegelmann, The computational power of interactive recurrent neural networks, Neural Comput. 24 (4) (2012) 996–1019.
[29] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[30] G.E.DahlA. Mohamed, G.E. Hinton, Acoustic modeling using deep belief networks, IEEE Trans. Audio Speech Lang. Process. 20 (2012) 14–22.
[31] Ronan Collobert, Jason Weston, A unified architecture for natural language processing: Deep neural networks with multitask learning, in: Proceedings of the 25th International Conference on Machine learning, ACM, New York, NY, USA, 2008, pp. 160–167.
[32] R. Raina, A. Madhavan, A.Y. Ng, Large-scale deep unsupervised learning using graphics processors, in: International Conference on Machine Learning, 2009, pp. 110–880.
[33] G.E. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[34] D. Erhan, Y. Bengio, A. Courville, Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11 (2010) 625–660.
[35] Nicolas Boulanger-Lewandowski, Yoshua Bengio, Pascal Vincent, Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription, in: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26–July 1, 2012, 2012.
[36] L. Pasa, A. Testolin, A. Sperduti, A HMM-based pre-training approach for sequential data, in: M. Verleysen (Ed.), Proceedings of the 2014 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), 2014.
[37] Geoffrey E. Hinton, Terrence Joseph Sejnowski, Unsupervised Learning: Foundations of Neural Computation, MIT press, Cambridge, MA, USA, 1999.
[38] Geoffrey E. Hinton, Learning to represent visual input, Philos. Trans. R. Soc. B: Biol. Sci. 365 (1537) (2010) 177–184.
[39] Quoc V. Le, Marc Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, Andrew Y. Ng, Building high-level features using large scale unsupervised learning, in: ICML, vol. 11, 2012, pp. 873–880.
[40] Alberto Testolin, Ivilin Stoianov, Michele De Filippo De Grazia, Marco Zorzi, Deep unsupervised learning on a desktop pc: a primer for cognitive scientists, Front. Psychol. 4 (2013).
[41] Ilya Sutskever, Geoffrey E. Hinton, Graham W. Taylor, The recurrent temporal restricted Boltzmann machine, in: Advances in Neural Information Processing Systems, 2008, pp. 1601–1608.
[42] Graham W. Taylor, Geoffrey E. Hinton, Factored conditional restricted Boltzmann machines for modeling motion style, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, New York, NY, USA, 2009, pp. 1025–1032.
[43] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, International Conference on Machine Learning, 2009, pp. 1–8.
[44] Faisal Khan, Bilge Mutlu, Xiaojin Zhu, How do humans teach: on curriculum learning and teaching dimension, in: Advances in Neural Information Processing Systems, 2011, pp. 1449–1457.
[45] Jeffrey L. Elman, Learning and development in neural networks: the importance of starting small, Cognition 48 (1) (1993) 71–99.
[46] Lloyd R. Welch, Hidden Markov models and the Baum–Welch algorithm, IEEE Inf. Theory Soc. Newslett. 53 (4) (2003) 10–13.
[47] James L. McClelland, David E. Rumelhart, PDP Research Group, et al., Parallel Distributed Processing. Explorations in the Microstructure of Cognition, vol. 1, 1986.
[48] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 70 (10) (1990) 1550–1560.
[49] Geoffrey E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. 14 (8) (2002) 1771–1800.
[50] Luis Javier Rodríguez, Inés Torres, Comparative study of the Baum–Welch and Viterbi training algorithms applied to read and spontaneous speech recognition, in: Pattern Recognition and Image Analysis, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 847–857.
[51] Mert Bay, A.F. Ehmann, J.S. Downie, Evaluation of multiple-F0 estimation and tracking systems, in: ISMIR, 2009, pp. 315–320.
[52] ⟨https://github.com/gwtaylor/theano-rnn⟩.

Luca Pasa received the Laurea degree in computer science in 2013, at University of Padova, Italy, with thesis about the application of RBM-based model on sequential data. In January 2014 he started Ph.D. in computer science at Department of Mathematics at the same university. His main research interest is correlated to machine learning and in particular neural networks and computational neuroscience. Most of the research that he carried out up to now is about the application of neural network on complex structured data.

Alberto Testolin received an M.Sc. in Computer Science (2011) and a Ph.D. in Cognitive Science (2015) from the University of Padova, where he is currently working as a post-doc researcher focusing on computational modeling of cognitive processes. His main interests include deep learning, recurrent neural networks and probabilistic generative models, which are applied to investigate visual processing and attentional mechanisms.

Alessandro Sperduti has a Ph.D. in computer science from the University of Pisa and since 2002 is a Professor in computer science at the University of Padova. He has been chair of the Data Mining and Neural Networks Technical Committees of IEEE CIS, and Associate Editor of IEEE TNNLS. He is currently an Action Editor for the journals AI Communications, Neural Networks, Theoretical Computer Science (Section C). His research interests include machine learning, neural networks, learning in structured domains, data and process mining.