Linear Autoencoder Networks for Structured Data

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

Abstract

We study linear autoencoder networks for structured inputs, such as sequences, trees. We show that the problem of training an autoencoder has a closed form solution which can be obtained via the definition of linear dynamical systems modelling the structural information present in the dataset of structures. Relationship with principal directions is discussed. We also briefly discuss how autoencoder networks can be used in relation to classification tasks.

1 Introduction

With the recent development of Deep Learning (e.g., [Hinton and Salakhutdinov, 2006; Hinton *et al.*, 2006; Bengio, 2009; di Lena *et al.*, 2012]), nonlinear autoencoder networks have witnessed a resurge of attention as a tool for developing rich internal representations of input data. In fact it seems more and more evident that effective learning should be based on relevant and robust internal representations developed in autonomy by the learning system.

Since in many cases neural-symbolic integration involves structured data, such as sequences, trees, and graphs, in this paper we investigate on autoencoder networks for structured data. Specifically, we discuss the most basic version of autoencoder networks, i.e. linear systems. We show that, for linear systems, it is actually possible to devise a closed form solution for learning which relates to Principal Component Analysis of the extended vectorial representations of the structured data. Principal Component Analysis (PCA) ([Jolliffe, 2002]) constitutes one of the oldest and best known tools in Pattern Recognition and Machine Learning. It is a powerful technique for dimensionality reduction, while preserving much of the relevant information conveyed by a set of variables. It is theoretically well founded and reduces to the solution of an eigenvalue problem involving the covariance (or correlation) matrix of the available data.

Exploiting the well known kernel trick, Kernel PCA ([Schölkopf *et al.*, 1997]), which is a nonlinear form of PCA, has been proposed. Through the kernel trick, it is possible to implicitly project data into high-dimensional feature spaces. PCA is then performed in feature space, discovering principal directions that correspond to principal curves in the original data space. By defining a kernel on structured data, such as sequences, trees, and graphs, it is also possible to apply PCA to application domains where it is natural to represent data in a structured form; just to name a few, Chemistry, Bioinformatics, and Natural Language Processing.

In this paper, we are interested in understanding how a PCA-like tool can be extended to structured data without using an a priori defined kernel for structures, especially when the information attached to each vertex of the structure is a real-valued vector.

We show that linear autoencoder networks are actually closely linked to PCA not only in the case of vectorial input (see [Bourlard and Kamp, 1988; Baldi and Hornik, 1989]), but also in the case of structured data modeled via a suitable linear dynamical system. From this perspective, performing PCA of a set of structures is equivalent to discover the most compact and informative dynamical system able to map each structure into a state space vector representing as much as possible of the information conveyed by the structure itself. The state space of such dynamical system would then define a quite compact data dependent feature space, directly amenable to be used by any of the standard Machine Learning tools for tasks such as classification, regression, and clustering. This would constitute a complementary approach to kernel-based methods for structured input (see [Gartner, 2003] for a survey), where the input structures are mapped by an a priori defined nonlinear function into a very large feature space. The definition of a data dependent and compact feature space is reminiscent of the hidden activation space of Recursive Neural Networks (e.g. see [Sperduti and Starita, 1997; Frasconi *et al.*, 1998; Hammer, 2000; Baldi and Pollastri, 2003; Micheli *et al.*, 2001]), which have been successfully applied to learning tasks in Chemistry and Bioinformatics. Recursive Neural Networks, however, suffer the local minima problem, and often are not so easy to train because of the use of sigmoidal functions that introduce plateaus into the objective function.

We show that it is actually possible to define PCA-like representations for structured data via autoencoder networks, mainly for sequences and trees, since graphs can be treated indirectly as a sequence of nodes. We start by recalling PCA for vectors (Section 2.1). Then, we introduce a linear dynamical system as basic component of a linear autoencoder, and then the main result used for the definition of PCA-like rep-

resentations for sequences (Section 2.2). The application of the same ideas to trees is discussed in Section 2.3, where an extended dynamical system is introduced. In Section 3 we briefly discuss how autoencoders can be used for classification tasks. Conclusions are drawn in Section 4.

Preliminary work described in this paper and results obtained on empirical data have been presented in ([Sperduti, 2006; Micheli and Sperduti, 2007; Sperduti, 2007]).

2 Linear Autoencoder Networks and PCA for Vectors and Structures

In the following we study the relationship between solutions to linear autoencoder networks and the computation of PCA for vectors and structured data. We briefly recall the standard PCA with a perspective that will allow us to readily understand the connection with solutions to linear autoencoder networks for sequences. The suggested approach is then further extended to cover the direct treatment of trees.

2.1 Vectors

One of the aims of standard PCA ([Jolliffe, 2002]) is to reduce the dimensionality of a data set, while preserving as much as possible the information present in it. This is achieved by looking for orthogonal directions of maximum variance within the data set. The principal components are sorted according to the amount of variance they explain, so that the first few retain most of the variation present in all of the original variables. It turns out that the qth principal component is given by the projection of the data onto the eigenvector of the (sample) covariance matrix $\mathbf{C}$ of the data corresponding to the qth largest eigenvalue.

From a mathematical point of view, PCA can be understood as given by an orthogonal linear transformation of the given set of variables (i.e., the coordinates of the vectorial space in which data is embedded):

$$\mathbf{y}_i = \mathbf{A}\mathbf{x}_i$$

where $\mathbf{x}_i \in \mathbb{R}^k$ are the vectors belonging to the data set, and $\mathbf{A} \in \mathbb{R}^{k\times k}$ is the orthogonal matrix whose qth row is the qth eigenvector of the covariance matrix. Typically, larger variances are associated with the first $p < k$ principal components. Thus one can conclude that most relevant information occur only in the first p dimensions. The process of retaining only the first p principal components is known as *dimensional reduction*. Given a fixed value for p, principal components allow also to minimize the reconstruction error, i.e. the square error of the difference between the original vector $\mathbf{x}_i$ and the vector obtained by projecting its principal components $\mathbf{y}_i$ back into the original space by the linear transformation ${\mathbf{A}^{(p)}}^{\mathsf{T}}\mathbf{y}_i$:

$$\mathbf{A}^{(p)} = \arg\min_{\mathbf{M}\in\mathbb{R}^{p\times k}} \sum_i \|\mathbf{x}_i - \mathbf{M}^{\mathsf{T}}\mathbf{M}\mathbf{x}_i\|^2$$

where the rows of $\mathbf{A}^{(p)} \in \mathbb{R}^{p\times k}$ corresponds to the first p eigenvectors of $\mathbf{C}$. This can be shown by resorting to a Rayleigh quotient. In fact, let

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \mathbf{x}_3^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{bmatrix},$$

then the direction of maximum variance can be computed as

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w}}{\mathbf{w}^{\mathsf{T}}\mathbf{w}} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^{\mathsf{T}}\mathbf{C}\mathbf{w}}{\mathbf{w}^{\mathsf{T}}\mathbf{w}}.$$

By imposing with no loss in generality that $\|\mathbf{w}\| = 1$, an equivalent problem is

$$\mathbf{w}^* = \arg\max_{\|\mathbf{w}\|=1} \mathbf{w}^{\mathsf{T}}\mathbf{C}\mathbf{w}.$$

This is a constrained optimization problem that can be solved by optimizing the Lagrangian

$$\mathcal{L}(\mathbf{w},\lambda) = \mathbf{w}^{\mathsf{T}}\mathbf{C}\mathbf{w} - \lambda(\mathbf{w}^{\mathsf{T}}\mathbf{w} - 1).$$

By differentiating the Lagrangian with respect to $\mathbf{w}$ and equating to zero leads to

$$\mathbf{C}\mathbf{w} - \lambda\mathbf{w} = 0,$$

which corresponds to the following symmetric eigenvalue problem

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}.$$

Thus the first principal direction corresponds to the eigenvector with maximum eigenvalue, while the other principal directions correspond to the other eigenvectors (pairwise orthogonal by definition), sorted according to the corresponding eigenvalues.

In [Bourlard and Kamp, 1988; Baldi and Hornik, 1989], principal directions have been related to solutions obtained by training linear autoencoder networks

$$\mathbf{o}_i = \mathbf{W}_{hidden}\mathbf{W}_{input}\mathbf{x}_i,\ i = 1,\ldots,n, \tag{1}$$

where $\mathbf{W}_{input} \in \mathbb{R}^{p\times k}$, $\mathbf{W}_{hidden} \in \mathbb{R}^{k\times p}$, $p \ll k$, and the network is trained so to get $\mathbf{o}_i = \mathbf{x}_i,\ \forall i$.

2.2 Sequences

When a temporal sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots$ of input vectors, where t is a discrete time index, is considered, the autoencoder defined in eq. (1) is extended by considering the coupled linear dynamical systems

$$\mathbf{y}_t = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{y}_{t-1} \tag{2}$$

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

It should be noticed that eq. (2) extends the linear transformation defined in eq. (2.1) by introducing a *memory term* involving the matrix $\mathbf{B} \in \mathbb{R}^{p\times p}$. This approach has been proposed, for example, in ([Voegtlin, 2005]) where an iterative procedure, based on Oja's rule, is presented. No proof of convergence for the proposed procedure is given.

Here we give a formal treatment which allows us to reach a sound solution to the above problem while returning, as a by-product, a closed form solution to the problem of training the autoencoder defined by eqs. (2) and (3).

The basic idea is to look for directions of high variance into the *state space* of the dynamical linear system (2).

Let start by considering a single sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_n$ and the state vectors of the corresponding induced state sequence collected as rows of a matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^\mathsf{T} \\ \mathbf{y}_2^\mathsf{T} \\ \mathbf{y}_3^\mathsf{T} \\ \vdots \\ \mathbf{y}_n^\mathsf{T} \end{bmatrix}.$$

By using the initial condition $\mathbf{y}_0 = \mathbf{0}$ (the null vector), and the dynamical linear system (2), we can rewrite the matrix as

$$\mathbf{Y} = \begin{bmatrix} (\mathbf{A}\mathbf{x}_1)^\mathsf{T} \\ (\mathbf{A}\mathbf{x}_2 + \mathbf{B}\mathbf{A}\mathbf{x}_1))^\mathsf{T} \\ (\mathbf{A}\mathbf{x}_3 + \mathbf{B}\mathbf{A}\mathbf{x}_2 + \mathbf{B}^2\mathbf{A}\mathbf{x}_1)^\mathsf{T} \\ \vdots \\ (\mathbf{A}\mathbf{x}_n + \cdots + \mathbf{B}^{n-2}\mathbf{A}\mathbf{x}_2 + \mathbf{B}^{n-1}\mathbf{A}\mathbf{x}_1)^\mathsf{T} \end{bmatrix},$$

which can be factorized as

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

where, given $s = kn$, $\boldsymbol{\Xi} \in \mathbb{R}^{n \times s}$ is a data matrix collecting all the (inverted) input subsequences (including the whole sequence) as rows, and $\boldsymbol{\Omega}$ is the parameter matrix of the dynamical system.

Now, we are interested in using a state space of smallest dimension p, i.e. $\mathbf{y}_t \in \mathbb{R}^p$, such that all information contained in $\boldsymbol{\Omega}$ is preserved. We start by factorizing $\boldsymbol{\Xi}$ using SVD, obtaining

$$\boldsymbol{\Xi} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{U}^\mathsf{T}$$

where $\mathbf{V} \in \mathbb{R}^{n \times n}$ is an unitary matrix, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times s}$ is a rectangular diagonal matrix with nonnegative real numbers on the diagonal with $\lambda_{1,1} \geq \lambda_{2,2} \geq \cdots \geq \lambda_{n,n}$ (the singular values), and $\mathbf{U}^\mathsf{T} \in \mathbb{R}^{s \times n}$ is a unitary matrix.

It is important to notice that columns of $\mathbf{U}^\mathsf{T}$ which correspond to nonzero singular values, apart some mathematical technicalities, basically correspond to the principal directions of data, i.e. PCA.

If the rank of $\boldsymbol{\Xi}$ is p, then only the first p elements of the diagonal of $\boldsymbol{\Lambda}$ are not null, and the above decomposition can be reduced to

$$\boldsymbol{\Xi} = \mathbf{V}^{(p)}\boldsymbol{\Lambda}^{(p)}{\mathbf{U}^{(p)}}^\mathsf{T} \tag{4}$$

where $\mathbf{V}^{(p)} \in \mathbb{R}^{n \times p}$, $\boldsymbol{\Lambda}^{(p)} \in \mathbb{R}^{p \times p}$, and ${\mathbf{U}^{(p)}}^\mathsf{T} \in \mathbb{R}^{p \times n}$. Now we can observe that ${\mathbf{U}^{(p)}}^\mathsf{T}\mathbf{U}^{(p)} = \mathbf{I}$ (where $\mathbf{I}$ is the identity matrix of dimension p), since by definition the columns of $\mathbf{U}^{(p)}$ are orthogonal, and by imposing $\boldsymbol{\Omega} = \mathbf{U}^{(p)}$, we can derive "optimal" matrices $\mathbf{A} \in \mathbb{R}^{p \times k}$ and $\mathbf{B} \in \mathbb{R}^{p \times p}$ for our dynamical system, which will have corresponding state space matrix

$$\mathbf{Y}^{(p)} = \boldsymbol{\Xi}\boldsymbol{\Omega} = \boldsymbol{\Xi}\mathbf{U}^{(p)} = \mathbf{V}^{(p)}\boldsymbol{\Lambda}^{(p)}{\mathbf{U}^{(p)}}^\mathsf{T}\mathbf{U}^{(p)} = \mathbf{V}^{(p)}\boldsymbol{\Lambda}^{(p)}.$$

Thus, if we represent $\mathbf{U}^{(p)}$ as composed of n submatrices $\mathbf{U}_i^{(p)}$, each of size $k \times p$, the problem reduces to find matrices $\mathbf{A}$ and $\mathbf{B}$ such that

$$\boldsymbol{\Omega} = \begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{A}^\mathsf{T}\mathbf{B}^\mathsf{T} \\ \mathbf{A}^\mathsf{T}{\mathbf{B}^2}^\mathsf{T} \\ \vdots \\ \mathbf{A}^\mathsf{T}{\mathbf{B}^{n-1}}^\mathsf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1^{(p)} \\ \mathbf{U}_2^{(p)} \\ \mathbf{U}_3^{(p)} \\ \vdots \\ \mathbf{U}_n^{(p)} \end{bmatrix} = \mathbf{U}^{(p)}. \tag{5}$$

In the following, we demonstrate that there exists a solution to the above equation. We start by observing that $\boldsymbol{\Xi}$ owns a special structure, i.e. given $\boldsymbol{\Xi} = [\boldsymbol{\Xi}_1 \ \boldsymbol{\Xi}_2 \ \cdots \ \boldsymbol{\Xi}_n]$, where $\boldsymbol{\Xi}_i \in \mathbb{R}^{n \times k}$, then for $i = 1, \ldots, n-1$

$$\boldsymbol{\Xi}_{i+1} = \mathbf{R}_{k,s}\boldsymbol{\Xi}_i = \begin{bmatrix} \mathbf{0}_{1\times(n-1)} & \mathbf{0}_{1\times(s-n+1)} \\ \mathbf{I}_{(n-1)\times(n-1)} & \mathbf{0}_{(n-1)\times(s-n+1)} \end{bmatrix} \boldsymbol{\Xi}_i,$$

and

$$\mathbf{R}_{k,s}\boldsymbol{\Xi}_n = \mathbf{0}, \text{ i.e. the null matrix of size } n \times k.$$

Moreover, by eq. (4), we have

$$\boldsymbol{\Xi}_i = \mathbf{V}^{(p)}\boldsymbol{\Lambda}^{(p)}{\mathbf{U}_i^{(p)}}^\mathsf{T}, \text{ for } i = 1, \ldots, n.$$

Using the fact that ${\mathbf{V}^{(p)}}^\mathsf{T}\mathbf{V}^{(p)} = \mathbf{I}$, and combining the above equations, we get

$$\mathbf{U}_{i+t}^{(p)} = \mathbf{U}_i^{(p)}\mathbf{Q}^t, \text{ for } i = 1, \ldots, n-1, \text{ and } t = 1, \ldots, n-i$$

where $\mathbf{Q} = \boldsymbol{\Lambda}^{(p)}{\mathbf{V}^{(p)}}^\mathsf{T}\mathbf{R}_{k,s}^\mathsf{T}\mathbf{V}^{(p)}{\boldsymbol{\Lambda}^{(p)}}^{-1}$. Moreover, we have that $\mathbf{U}_n^{(p)}\mathbf{Q} = \mathbf{0}$ since

$$\begin{aligned} \mathbf{U}_n^{(p)}\mathbf{Q} &= \mathbf{U}_n^{(p)}\boldsymbol{\Lambda}^{(p)}{\mathbf{V}^{(p)}}^\mathsf{T}\mathbf{R}_{k,s}^\mathsf{T}\mathbf{V}^{(p)}{\boldsymbol{\Lambda}^{(p)}}^{-1} \\ &= (\mathbf{R}_{k,s}\boldsymbol{\Xi}_n)^\mathsf{T}\mathbf{V}^{(p)}{\boldsymbol{\Lambda}^{(p)}}^{-1}. \end{aligned} \tag{6}$$

Thus, eq. (5) is satisfied by $\mathbf{A} = {\mathbf{U}_1^{(p)}}^\mathsf{T}$ and $\mathbf{B} = \mathbf{Q}^\mathsf{T}$.

It is interesting to note that the original data $\boldsymbol{\Xi}$ can be recovered by computing

$$\mathbf{Y}^{(p)}{\mathbf{U}^{(p)}}^\mathsf{T} = \mathbf{V}^{(p)}\boldsymbol{\Lambda}^{(p)}{\mathbf{U}^{(p)}}^\mathsf{T} = \boldsymbol{\Xi},$$

which can be achieved by running the dynamical system

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

starting from $\mathbf{y}_n$, i.e. $\begin{bmatrix} \mathbf{A}^\mathsf{T} \\ \mathbf{B}^\mathsf{T} \end{bmatrix}$ is the matrix $\mathbf{C}$ defined in eq. (3).

Finally, it is important to remark that the above construction works not only for a single sequence, but also for a set of sequences of different length. For example, let consider the two sequences $(\mathbf{x}_1^a, \mathbf{x}_2^a, \mathbf{x}_3^a, \mathbf{x}_4^a)$ and $(\mathbf{x}^b{}_1, \mathbf{x}^b{}_2)$. Then, we have

$$\mathbf{\Xi_a} = \begin{bmatrix} \mathbf{x}_1^{a\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_2^{a\mathsf{T}} & \mathbf{x}_1^{a\mathsf{T}} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_3^{a\mathsf{T}} & \mathbf{x}_2^{a\mathsf{T}} & \mathbf{x}_1^{a\mathsf{T}} & \mathbf{0} \\ \mathbf{x}_4^{a\mathsf{T}} & \mathbf{x}_3^{a\mathsf{T}} & \mathbf{x}_2^{a\mathsf{T}} & \mathbf{x_1}^{a\mathsf{T}} \end{bmatrix}$$

and

$$\mathbf{\Xi_b} = \begin{bmatrix} \mathbf{x}_1^{b\mathsf{T}} & \mathbf{0} \\ \mathbf{x}_2^{b\mathsf{T}} & \mathbf{x}_1^{b\mathsf{T}} \end{bmatrix}$$

which can be collected together into the matrix

$$\mathbf{\Xi} = \left[\begin{array}{cc} \mathbf{\Xi_a} & \\ \hline \mathbf{\Xi_b} & \mathbf{0}_{2\times 2} \end{array}\right]$$

and matrix

$$\mathbf{R} = \left[\begin{array}{cc} \mathbf{R}_{k,4k} & \\ \hline \mathbf{R}_{k,2k} & \mathbf{0}_{2\times 2} \end{array}\right]$$

to define matrix $\mathbf{Q}$.

The following lemma shows the link between matrix $\mathbf{Q}$ and the principal components corresponding to matrix $\mathbf{U}^{(p)}$

Lemma *(Relationship with Principal Directions)*

$$\mathbf{Q} = \mathbf{U}^{(p)\mathsf{T}}\mathbf{R}_{k,s}^{\mathsf{T}}\mathbf{U}^{(p)} = \sum_{i=1}^{n-1} \mathbf{U}_i^{(p)\mathsf{T}}\mathbf{U}_{i+1}^{(p)}$$

Proof: By definition $\sum_{i=1}^{n} \mathbf{U}_i^{(p)\mathsf{T}}\mathbf{U}_i^{(p)} = \mathbf{I}$ and

$$\begin{aligned} \mathbf{Q} &= \left(\sum_{i=1}^{n} \mathbf{U}_i^{(p)\mathsf{T}}\mathbf{U}_i^{(p)}\right)\mathbf{Q} = \sum_{i=1}^{n} \mathbf{U}_i^{(p)\mathsf{T}}\left(\mathbf{U}_i^{(p)}\mathbf{Q}\right) \\ &= \sum_{i=1}^{n-1} \mathbf{U}_i^{(p)\mathsf{T}}\mathbf{U}_{i+1}^{(p)} \end{aligned}$$

where we have used $\mathbf{U}_{i+1}^{(p)} = \mathbf{U}_i^{(p)}\mathbf{Q}$ and $\mathbf{U}_n^{(p)}\mathbf{Q} = \mathbf{0}$. ■

As a final remark, it should be stressed that the above construction *only* works if p is equal to the rank of $\mathbf{\Xi}$. If p is smaller, then there is no formal proof that the proposed solution is optimal, although empirical experimental results seem to be quite good (see [Sperduti, 2006; Micheli and Sperduti, 2007; Sperduti, 2007]).

2.3 Trees

When considering trees, an extension of the approach used for sequences can be used. First of all, let us illustrate what happens for an example given by a complete binary tree. Then, we will generalize the construction to (in)complete b-ary trees. For $b = 2$, we consider the following linear dynamical system (the decoding is obtained by the transposed dynamical system, as we have seen for sequences)

$$\mathbf{y}_u = \mathbf{A}\mathbf{x}_u + \mathbf{B_l}\mathbf{y}_{ch_l[u]} + \mathbf{B_r}\mathbf{y}_{ch_r[u]} \tag{7}$$

where u is a vertex of the tree, $ch_l[u]$ is the left child of u, $ch_r[u]$ is the right child of u, $\mathbf{B_l}, \mathbf{B_r} \in \mathbb{R}^{p\times p}$.

Let consider the following complete binary tree

$$\mathcal{T} \equiv \mathbf{x}_7(\mathbf{x}_5(\mathbf{x}_1, \mathbf{x}_2), \mathbf{x}_6(\mathbf{x}_3, \mathbf{x}_4)),$$

where we have used parentheses to represent the tree structure. Then we can consider the corresponding induced state elements collected as rows of a matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^{\mathsf{T}} \\ \mathbf{y}_2^{\mathsf{T}} \\ \mathbf{y}_3^{\mathsf{T}} \\ \vdots \\ \mathbf{y}_7^{\mathsf{T}} \end{bmatrix}.$$

By using the initial condition $\mathbf{y}_{nil} = \mathbf{0}$ (the null vector), and the dynamical linear system (7), we have

$$\mathbf{Y} = \begin{bmatrix} (\mathbf{A}\mathbf{x}_1)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_2)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_3)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_4)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_5 + \mathbf{B_l}\mathbf{A}\mathbf{x}_1 + \mathbf{B_r}\mathbf{A}\mathbf{x}_2)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_6 + \mathbf{B_l}\mathbf{A}\mathbf{x}_3 + \mathbf{B_r}\mathbf{A}\mathbf{x}_4)^{\mathsf{T}} \\ (\mathbf{A}\mathbf{x}_7 + \mathbf{B_l}\mathbf{A}\mathbf{x}_5 + \mathbf{B_r}\mathbf{A}\mathbf{x}_6 + \mathbf{B_l}^2\mathbf{A}\mathbf{x}_1 + \mathbf{B_l}\mathbf{B_r}\mathbf{A}\mathbf{x}_2 \\ \quad +\mathbf{B_r}\mathbf{B_l}\mathbf{A}\mathbf{x}_3 + \mathbf{B_r}^2\mathbf{A}\mathbf{x}_4)^{\mathsf{T}} \end{bmatrix},$$

which can be factorized as

$$\mathbf{Y} = \underbrace{\begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_2^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_3^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_4^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_5^{\mathsf{T}} & \mathbf{x}_1^{\mathsf{T}} & \mathbf{x}_2^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_6^{\mathsf{T}} & \mathbf{x}_3^{\mathsf{T}} & \mathbf{x}_4^{\mathsf{T}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{x}_7^{\mathsf{T}} & \mathbf{x}_5^{\mathsf{T}} & \mathbf{x}_6^{\mathsf{T}} & \mathbf{x}_1^{\mathsf{T}} & \mathbf{x}_2^{\mathsf{T}} & \mathbf{x}_3^{\mathsf{T}} & \mathbf{x}_4^{\mathsf{T}} \end{bmatrix}}_{\mathbf{\Xi}} \underbrace{\begin{bmatrix} \mathbf{A}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{2\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{2\mathsf{T}} \end{bmatrix}}_{\mathbf{\Omega}}.$$

It is not difficult to recognize that each macro component of $\mathbf{\Omega}$ corresponds to a path into a generic binary tree: $\mathbf{A}^{\mathsf{T}}$ is associated to the root, $\mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}}$ is associated to the left child of the root, and so on. As for sequences, we have now to solve the following matricial equation

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{A}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{2\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_l}^{\mathsf{T}}\mathbf{B_r}^{\mathsf{T}} \\ \mathbf{A}^{\mathsf{T}}\mathbf{B_r}^{2\mathsf{T}} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_1^{(p)} \\ \mathbf{U}_2^{(p)} \\ \mathbf{U}_3^{(p)} \\ \mathbf{U}_4^{(p)} \\ \mathbf{U}_5^{(p)} \\ \mathbf{U}_6^{(p)} \\ \mathbf{U}_7^{(p)} \end{bmatrix} = \mathbf{U}^{?},$$

with respect to the three matrices $\mathbf{A}$, $\mathbf{B_l}$, and $\mathbf{B_r}$.

As for the case of sequences, we can observe that $\mathbf{\Xi}$ owns a special structure, i.e. given $\mathbf{\Xi} = [\mathbf{\Xi}_1\ \mathbf{\Xi}_2\ \cdots\ \mathbf{\Xi}_7]$ we can state that matrix $\mathbf{\Xi}_{left} = [\mathbf{\Xi}_2\ \mathbf{\Xi}_4\ \mathbf{\Xi}_5\ \mathbf{0}_{7\times 4k}]$ is equal to

$$\mathbf{\Xi}_{left} = \mathbf{R}_{k,left}\mathbf{\Xi} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{\Xi},$$

and matrix $\mathbf{\Xi}_{right} = [\mathbf{\Xi}_3\ \mathbf{\Xi}_6\ \mathbf{\Xi}_7\ \mathbf{0}_{7\times 4k}]$ is equal to

$$\mathbf{\Xi}_{right} = \mathbf{R}_{k,right}\mathbf{\Xi} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{\Xi},$$

which combined with eq. (4) gives matrices

$$\mathbf{Q}_{left} = \mathbf{\Lambda}^{(p)}{\mathbf{V}^{(p)}}^{\mathsf{T}}\mathbf{R}_{k,left}^{\mathsf{T}}\mathbf{V}^{(p)}{\mathbf{\Lambda}^{(p)}}^{-1}$$

and

$$\mathbf{Q}_{right} = \mathbf{\Lambda}^{(p)}{\mathbf{V}^{(p)}}^{\mathsf{T}}\mathbf{R}_{k,right}^{\mathsf{T}}\mathbf{V}^{(p)}{\mathbf{\Lambda}^{(p)}}^{-1},$$

leading to the solution $\mathbf{A} = {\mathbf{U}_1^{(p)}}^{\mathsf{T}}$, $\mathbf{B_l} = \mathbf{Q}_{left}^{\mathsf{T}}$, and $\mathbf{B_r} = \mathbf{Q}_{right}^{\mathsf{T}}$.

Also in this case, we have *push* operators, i.e., $\mathbf{R}_{k,left}$ is a *push left* operator, while $\mathbf{R}_{k,right}$ is a *push right* operator. In the following, we describe how these push operators can be defined in general for complete binary trees.

Each vertex u of the binary tree is associated to a binary string $id(u)$ obtained as follows: the binary string "1" is associated to the root of the tree. Any other vertex has associated the string obtained by concatenating the string of its parent with the string "0" if it is a left child, "1" otherwise. Then, all the dimensions of the state space $\mathbb{S}$ are partitioned in s/k groups of k dimensions. The label associated to vertex v is stored into the j-th group, where j is the integer represented by the binary string $id(u)$. E.g. the label of the root is stored into group 1, since $id(root)$ ="1", the label of the vertex which can be reached by the path ll starting from the root is stored into group 4, since $id(u)$ ="100", while the label of the vertex reachable through the path rlr is stored into group 13, since $id(u)$ ="1101". Notice that, if the input tree is not complete, the components corresponding to missing vertexes are set to be equal to 0.

Matrices $\mathbf{B_l}$ and $\mathbf{B_r}$ are defined as follows. Both matrices are composed of two types of blocks, i.e. $\mathbf{I}_{k\times k}$ and $\mathbf{0}_{k\times k}$. Matrix $\mathbf{B_l}$ has to implement a *push left* operator, i.e. the tree $\mathcal{T}$ encoded by a vector $\mathbf{y}_{root(\mathcal{T})}$ has to become the left child of a new node u whose label is the current input $\mathbf{x}_u$. Thus $root(\mathcal{T})$ has to become the left child of u and also all the other vertexes in $\mathcal{T}$ have their position redefined accordingly. From a mathematical point of view, the new position of any vertex a in $\mathcal{T}$ is obtained by redefining $id(a)$ as follows: *i)* the most significative bit of $id(a)$ is set to "0", obtaining the string $id_0(a)$; *ii)* the new string $id_{new}(a)$ ="1"+$id_0(a)$ is defined, where + is the string concatenation operator. If $id_{new}(a)$ represents a number greater than s/k then this means that the vertex has been pushed outside the available memory, i.e. the vertex a is *lost*. Consequently, groups which correspond to *lost* vertexes have to be annihilated. Thus, $\mathbf{B_l}$ is composed of $(q+1)\times(q+1)$ blocks, all of type $\mathbf{0}_{k\times k}$, except for the blocks in row $id_{new}(a)$ and column $id(a)$, with $id_{new}(a) \leq s/k$, where a block $\mathbf{I}_{k\times k}$ is placed. Matrix $\mathbf{B_r}$ is defined similarly: it has to implement a *push right* operator, i.e.: *i)* the most significative bit of $id(a)$ is set to "1", obtaining the string $id_1(a)$; *ii)* the new string $id_{new}(a)$ ="1"+$id_1(a)$ is defined.

It should be noticed that the definition of the operators described above is not dependent on the number of the processed trees, but only depends on the size of the largest tree (memory size).

Generalization of the above scheme for complete b-ary trees is not difficult. The dynamical linear system becomes

$$\mathbf{y}_u = \mathbf{A}\mathbf{x}_u + \sum_{c=0}^{b-1} \mathbf{B_c}\mathbf{y}_{ch_c[u]} \tag{8}$$

where $ch_c[u]$ is the $c+1$-th child of u, and a matrix $\mathbf{B_c}$ is defined for each child. The string associated to each vertex is defined on the alphabet {"0","1",...,"b-1"}, since there are b children. The symbol $b-1$ is associated with the root and b push operations have to be defined. The new string associated to any vertex a in $\mathcal{T}$, after a c-push operation, is obtained by redefining $id(a)$ as follows: *i)* the most significative symbol of $id(a)$ is set to c, obtaining the string $id_c(a)$; *ii)* the new string $id_{new}(a)$ ="b-1"+$id_c(a)$ is defined. E.g., if $b = 5$ and c ="3", then *i)* the most significative symbol of $id(a)$ is set to "3", obtaining the string $id_3(a)$; *ii)* the new string $id_{new}(a)$ ="b-1"+$id_3(a)$ is defined. Matrix $\mathbf{B_c}$ is defined by placing blocks $\mathbf{I}_{k\times k}$ in positions $(id_{new}(a), id(a))$ only if $id_{new}(a) \leq s/k$, where $id_{new}(a)$ is interpreted as a number represented in base b. Finally, the optimal solution for a state space of dimension p is given by matrices

$$\mathbf{A} = {\mathbf{U}_1^{(p)}}^{\mathsf{T}},\ \mathbf{B}_c = \mathbf{Q}_{push_c}.$$

A problem in dealing with complete trees is that very soon there is a combinatorial explosion of the number of paths to consider, i.e. in order for the machine to deal with moderately deep trees, a huge value for s needs to be used. In practical applications, however, the observed trees tend to follow a specific generative model, and thus there may be many topologies which are never, or very seldomly, generated.

For this reason we suggest to use the following approach. Given a set of trees $\mathbf{T}$, the optimized graph $G_{\mathbf{T}}$ ([Sperduti and Starita, 1997]) is obtained by joining all the trees in such a way that any (sub)tree in $\mathbf{T}$ is represented only once. The optimized graph $G_{\mathbf{T}}$, which is a DAG, is then visited bottom-up, generating for each visited vertex v the set of id strings associated to the tree rooted in v, thus simulating all the different push operations which should be performed when presenting the trees in $\mathbf{T}$ to the machine. Repeated id strings are removed. The obtained set P is then used to define the state space of the machine: each string is associated to one group of k coordinates. In this way, only paths which appear in the set $\mathbf{T}$ (including all subtrees) are represented, thus drastically reducing the size of s, which will be equal to $|P| \times k$.

3 Classification of Structures

In this section, we briefly discuss how an autoencoder network for structures can be related to a linear dynamical system for classification. For the sake of presentation we restrict our discussion to binary classification problems involving trees.

We are interested to understand how a linear dynamical system of the form shown in eq. (8) can be used for classification. Specifically, we assume that the state $\mathbf{y}_u$, where u is the root node of tree T, is used as input to a classifier, e.g. an SVM (see [Cardin *et al.*, 2009]). Let $\{(T_i, d_i)\}_{i=1}^n$ be a training set of trees T_i, each belonging to class $d_i \in \{+1, -1\}$.

Let $\mathbf{r}(T_i)$ be any vectorial explicit representation of T_i compliant with the above dynamical system, i.e. the row of $\boldsymbol{\Xi}$ corresponding to the state obtained after presenting *all* the nodes of the tree to the above dynamical system. Let assume that the vectorial training set $\{(\mathbf{r}(T_i), d_i)\}_{i=1}^n$ is linearly separable and $\mathbf{w}^* = \sum_{j=1}^q \alpha_{i_j}^* \mathbf{r}(T_{i_j})$ be the optimal weight vector returned by an SVM, where $\alpha_{i_j}^* > 0$ are the optimal dual variables corresponding to support vectors $\mathbf{r}(T_{i_j})$.

We observe that, since $\mathbf{w}^*$ only depends on support vectors, then any linear dynamical system which only encodes the subspace spanned by the corresponding trees T_{i_j} is able to define an equivalent classifier. Specifically, let $\mathbf{T}_{sup} = \{T_{i_j}\}_{j=1}^q$ be the set of support trees and $\mathbf{T}_\perp$ the set of trees that are not support trees, i.e. all the trees in the training set are given by $\mathbf{T}_{sup} \cup \mathbf{T}_\perp$. Let $\boldsymbol{\Xi}(\mathbf{T}_{sup})$ be the data matrix generated by *all* nodes in trees belonging to $\mathbf{T}_{sup}$ and $\boldsymbol{\Xi}(\mathbf{T}_\perp)$ the corresponding matrix for $\mathbf{T}_\perp$. Notice that any $\mathbf{r}(T_{i_j})$ will be one row of $\boldsymbol{\Xi}(\mathbf{T}_{sup})$, however $\boldsymbol{\Xi}(\mathbf{T}_{sup})$ will also contain rows that correspond to subtrees belonging to any T_{i_j}. We also notice that matrix $\boldsymbol{\Xi}(\mathbf{T}_\perp)$ contains useful information for classification only with respect to the components of its rows that belong to the subspace spanned by the rows of $\boldsymbol{\Xi}(\mathbf{T}_{sup})$. We can compute this contribution by projecting the rows of $\boldsymbol{\Xi}(\mathbf{T}_\perp)$ over the rows of $\boldsymbol{\Xi}(\mathbf{T}_{sup})$. Such projection is obtained by finding the matrix solving the following optimization problem

$$\arg\min_{\mathbf{M}} \|\mathbf{M}\boldsymbol{\Xi}(\mathbf{T}_{sup}) - \boldsymbol{\Xi}(\mathbf{T}_\perp)\|^2.$$

The solution to the above problem is given by $\mathbf{M} = \boldsymbol{\Xi}(\mathbf{T}_\perp)\boldsymbol{\Xi}(\mathbf{T}_{sup})^+$, where $\boldsymbol{\Xi}(\mathbf{T}_{sup})^+$ is the pseudo-inverse of $\boldsymbol{\Xi}(\mathbf{T}_{sup})$ (easily obtainable by using its SVD decomposition). We now compute the "optimal" dynamical system with matrices $\mathbf{A}^*$, $\mathbf{B}_\mathbf{c}^*$, for matrix $\begin{bmatrix} \boldsymbol{\Xi}(\mathbf{T}_{sup}) \\ \boldsymbol{\Xi}(\mathbf{T}_\perp)\boldsymbol{\Xi}(\mathbf{T}_{sup})\boldsymbol{\Xi}(\mathbf{T}_{sup})^+ \end{bmatrix}$. Let $\mathbf{y}^*_{root(T_{i_j})}$ be the state vectors of such system corresponding to the roots of support trees T_{i_j}. Then, the reduced weight vector $\mathbf{w}_r^* = \sum_{j=1}^q \alpha_{i_j}^* \mathbf{y}^*_{root(T_{i_j})}$ returns the same classification of $\mathbf{w}^*$ over the extended representations $\mathbf{r}(T_i)$. The reason for that can be understood by recalling that $\mathbf{r}(T_{i_j}) = \mathbf{y}^*_{root(T_{i_j})} {\mathbf{U}^{(p)}}^\mathsf{T}$ and so $\mathbf{r}(T_{i_j})\mathbf{r}(T_{i_j})^\mathsf{T} = \mathbf{y}^*_{root(T_{i_j})} {\mathbf{U}^{(p)}}^\mathsf{T} \mathbf{U}^{(p)} {\mathbf{y}^*}^\mathsf{T}_{root(T_{i_j})} = \mathbf{y}^*_{root(T_{i_j})} {\mathbf{y}^*}^\mathsf{T}_{root(T_{i_j})}$ (please, remember that we are managing row vectors, so $\mathbf{r}(T_{i_j})\mathbf{r}(T_{i_j})^\mathsf{T}$ is a dot product). Moreover, for trees that are not of support, only the projection over the vectorial representation of the support trees matters, which is exactly reconstructed by the dynamical system.

Of course, the above construction can also be used when data is not linearly separable and a kernel is used. In this case, the linear dynamical system is used to generate reduced representations which are then used in substitution of the full explicit representations used by the SVM.

4 Conclusion

In this paper, we have shown that solutions to linear autoencoder networks for structured data is closely related to principal directions of explicit vectorial representations of structured data which are compliant to suitable linear dynamical systems describing the structural information. Through this approach, we showed it is possible to reach a theoretically well founded solution for linear autoencoders. We briefly also discussed how linear autoencoders can be used to define linear dynamical systems for classification.

Here we did not discuss that it is actually possible to compute such solution when a kernel map is used in the label space, although the computational burden can increase significantly when the total number of components is larger than the dimension of the label space times the depth of the memory used to represent structural information.

Experimental results are not discussed here, however in other papers ([Sperduti, 2006; Micheli and Sperduti, 2007; Sperduti, 2007]) experiments involving sequences, trees, and graphs have confirmed the feasibility of the approach and the effectiveness of the proposed methods for reducing the computational burden. From the computational point of view, further improvements, not explored in this paper, can be obtained by considering the sparsity of the $\boldsymbol{\Xi}$ matrix, and the adoption of more sophisticated numerical algorithms for computing the SVD factorization.

More work is needed to precisely relate the proposed approach with KPCA using a kernel for structures. Also a more convincing formulation for the processing of graphs is needed. In fact, while the proposed formulation is fully satisfactory for sequences and trees, it is not easy to directly address graphs, mainly because of the presence of confluent edges and cycles. It may also be interesting to investigate if it is possible to extend the proposed approach to include nonlinearities in analogy to the work proposed by Hinton & Salakhutdinov for vectorial data ([Hinton *et al.*, 2006]).

References

[Baldi and Hornik, 1989] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[Baldi and Pollastri, 2003] Pierre Baldi and Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *J. Mach. Learn. Res.*, 4:575–602, 2003.

[Bengio, 2009] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[Bourlard and Kamp, 1988] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988.

[Callan and Palmer-Brown, 1997] Robert E. Callan and Dominic Palmer-Brown. (s)raam: An analytical technique for fast and reliable derivation of connectionist symbol structure representations. *Connect. Sci.*, 9(2):139–160, 1997.

[Cardin *et al.*, 2009] Riccardo Cardin, Lisa Michielan, Stefano Moro, and Alessandro Sperduti. Pca-based representations of graphs for prediction in qsar studies. In *ICANN (2)*, pages 105–114, 2009.

[di Lena *et al.*, 2012] Pietro di Lena, Ken Nagata, and Pierre Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28(19):2449–2457, 2012.

[Frasconi *et al.*, 1998] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing in data structures. *IEEE Transactions on Neural Networks*, Vol 9(5):768–785, 1998.

[Gartner, 2003] Thomas Gartner. A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):49–58, 2003.

[Hammer, 2000] Barbara Hammer. *Learning with Recurrent Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.

[Hasselmann, 1988] K. Hasselmann. Pips and pops: The reduction of complex dynamical systems using principal interaction and oscillation patters. *Geophys. Res.*, 93:11015–11021, 1988.

[Hinton and Salakhutdinov, 2006] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[Hinton *et al.*, 2006] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[Jolliffe, 2002] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag New York, Inc., 2002.

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

[Micheli *et al.*, 2001] Alessio Micheli, Alessandro Sperduti, Antonina Starita, and Anna Maria Bianucci. Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines. *Journal of Chemical Information and Computer Sciences*, 41(2):202–218, 2001.

[N.E. *et al.*, 2001] Goljandina N.E., Nekrutkin V.V., and Zhigljavsky A.A. *Analysis of Time Series Structure: SSA and related techniques*. Chapman & Hall / CRS, 2001.

[Paccanaro and Hinton, 2001] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244, 2001.

[Pollack, 1990] Jordan B. Pollack. Recursive distributed representations. *Artif. Intell.*, 46(1-2):77–105, 1990.

[Pollack, 1991] Jordan B. Pollack. The induction of dynamical recognizers. *Mach. Learn.*, 7(2-3):227–252, 1991.

[Schölkopf *et al.*, 1997] Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 583–588, London, UK, 1997. Springer-Verlag.

[Sperduti and Starita, 1997] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

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

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

[Voegtlin and Dominey, 2005] Thomas Voegtlin and Peter F. Dominey. Linear recursive distributed representations. *Neural Netw.*, 18(7):878–895, 2005.

[Voegtlin, 2005] Thomas Voegtlin. Recursive principal components analysis. *Neural Netw.*, 18(8):1051–1063, 2005.