Efficient Computation of Recursive Principal Component Analysis for Structured Input

Alessandro Sperduti

Department of Pure and Applied Mathematics, University of Padova, Italy,
sperduti@math.unipd.it

Abstract. Recently, a successful extension of Principal Component Analysis for structured input, such as sequences, trees, and graphs, has been proposed. This allows the embedding of discrete structures into vectorial spaces, where all the classical pattern recognition and machine learning methods can be applied. The proposed approach is based on eigenanalysis of extended vectorial representations of the input structures and substructures. One problem with the approach is that eigenanalysis can be computationally quite demanding when considering large datasets of structured objects. In this paper we propose a general approach for reducing the computational burden. Experimental results show a significant speed-up of the computation.

1 Introduction

In many real-world applications it is natural to represent data in a structured form. Just to name a few, in Chemistry chemical compounds can be represented as undirected annotated graphs; in Natural Language Processing, the semantics of a sentence is described in terms of a parse tree. In addition, many problems in these application domains are characterized by the presence of noise and/or uncertainty in the data. Moreover, these problems can naturally be formulated as clustering, or classification, or regression tasks, which are well suited to be treated by machine learning approaches. Many standard machine learning approaches, however, can deal only with numerical vectors. Thus, a first necessary step to their application to structured objects is the apriori selection of a set of structural features of interest which will constitute the dimensions of a vectorial space where each structure can be represented according to its own degree of matching.

Recently, different an more direct approaches have been proposed and successfully applied to structured domains, such as Recursive Neural Networks (e.g. see [11, 2, 4, 1, 9]), and Kernel Methods for structured patterns (see [3] for a survey). Both these approaches, however, have their problems, such has local minima for Neural Networks, and the a priori definition of the kernel for Kernel Methods.

More recently, an alternative approach has been proposed. The idea is to devise vectorial representations of structures, belonging to a data set, which preserve all the information needed to discriminate among each other. This approach hinges on the calculation of Principal Component Analysis (PCA) for structured objects, such as sequences, trees, and graphs [10, 8]. The aim is to provide a method to generate informative representations which are amenable to be used into already well known unsupervised and supervised techniques for clustering, classification, and regression.

A problem with this approach, however, is that it is computationally quite demanding. In this paper, we address this problem by proposing some techniques to reduce the computational burden. After presenting in Section 2 the basic concepts about PCA for vectors and structures, we discuss in Section 3 three different approaches that, if applied simultaneously, can significantly reduce the computational burden in the case of structures. This is experimentally demonstrated in two datasets of relevant size and complexity (Section 4).

2 Principal Components Analysis for Vectors and Structures

In the following we present the main ideas underpinning the computation of PCA for vectors and structured inputs. We briefly recall the standard PCA with a perspective that will allow us to readily introduce its extension to the case of sequences. The suggested approach is then further extended to cover the direct treatment of trees, and finally we discuss our proposal to deal with directed or undirected graphs.

2.1 Vectors

The aim of standard PCA [6] 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):

$$\mathbf{y}_i = \mathbf{W_x} \mathbf{x}_i \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^k$ are the vectors belonging to the data set, and $\mathbf{W_x}$ 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 $\mathbf{W}_\mathbf{x}^\mathsf{T} \mathbf{y}_i$:

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

where the rows of $\mathbf{W_x}$ corresponds to the first p eigenvectors of $\mathbf{C}$.

2.2 Sequences

In [10] it is shown how PCA can be extended to the direct treatment of sequences. More specifically, given a temporal sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots$ of input vectors, where t is a discrete time index, we are interested in modeling the sequence through the following linear dynamical system:

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

which extends the linear transformation defined in eq. (1) by introducing a *memory term* involving the matrix $\mathbf{W_y}$ and the principal components $\mathbf{y}_{t-1}$ computed up to time step $t-1$, i.e. the principal components describing the input sequence up to time step $t-1$. The aim is to define proper matrices $\mathbf{W_x}$ and $\mathbf{W_y}$ such that $\mathbf{y}_t$ can be considered a good "encoding" of the input sequence read till time step t, i.e., the sequence is first encoded using eq. (2), and then, starting from the obtained encoding $\mathbf{y}_t$, it should be possible to reconstruct backwards the original sequence using the transposes of $\mathbf{W_x}$ and $\mathbf{W_y}$. This requirement implies that the following equations

$$\mathbf{x}_t = \mathbf{W}_\mathbf{x}^\mathsf{T}\mathbf{y}_t \tag{3}$$

$$\mathbf{y}_{t-1} = \mathbf{W_x}\mathbf{x}_{t-1} + \mathbf{W_y}\mathbf{y}_{t-2} = \mathbf{W}_\mathbf{y}^\mathsf{T}\mathbf{y}_t \tag{4}$$

should hold. In fact, the perspective of this proposal for recursive principal component analysis is to find a low-dimensional representation of the input sequence such that the expected reconstruction error, i.e. the sum of the (squared) differences between the vectors generated by equation (3) and the original input vectors for different values of t

$$error(t) = \sum_{i=1}^{t} ||\mathbf{x}_i - \mathbf{W}_\mathbf{x}^\mathsf{T}(\mathbf{W}_\mathbf{y}^\mathsf{T})^{t-i} \underbrace{\sum_{j=1}^{t}(\mathbf{W_y})^{t-j}\mathbf{W_x}\mathbf{x}_i}_{\mathbf{y}_t} ||^2 \tag{5}$$

is as small as possible, i.e. given a fixed value of p, where $\mathbf{y}_t \in \mathbb{R}^p$, we look for

$$(\mathbf{W_x}, \mathbf{W_y}) = \underset{\substack{\mathbf{M} \in \mathbb{R}^{p \times k} \\ \mathbf{N} \in \mathbb{R}^{p \times p}}}{\arg\min} \sum_{i=1}^{t} ||\mathbf{x}_i - \mathbf{M}^\mathsf{T}(\mathbf{N}^\mathsf{T})^{t-i} \sum_{j=1}^{t} \mathbf{N}^{t-j}\mathbf{M}\mathbf{x}_i||^2.$$

In [10] it has been shown that, when considering several sequences for the same linear system, it is possible to find a value of p where the reconstruction error is zero by performing eigenanalysis of extended vectorial representations (belonging to the so called state space) of the input sequences, where a sequence at time t is represented by the vector

$$[\mathbf{x}_t^\mathsf{T}, \ldots, \mathbf{x}_1^\mathsf{T}, \underbrace{\mathbf{0}^\mathsf{T}, \ldots, \mathbf{0}^\mathsf{T}}_{(T-t)}] \tag{6}$$

being T the maximum length for any input sequence. This representation can be understood as an explicit representation of a stack where a new input vector, e.g. $\mathbf{x}_{t+1}$, is pushed into the stack by shifting to the right the current content by k positions, and inserting (adding) $\mathbf{x}_{t+1}$ into the freed positions:

$$[\mathbf{0}^{\mathsf{T}}, \mathbf{x}_t^{\mathsf{T}}, \ldots, \mathbf{x}_1^{\mathsf{T}}, \underbrace{\mathbf{0}^{\mathsf{T}}, \ldots, \mathbf{0}^{\mathsf{T}}}_{(T-t-1)}] + [\mathbf{x}_{t+1}^{\mathsf{T}}, \underbrace{\mathbf{0}^{\mathsf{T}}, \ldots, \mathbf{0}^{\mathsf{T}}}_{(T-1)}] = [\mathbf{x}_{t+1}^{\mathsf{T}}, \mathbf{x}_t^{\mathsf{T}}, \ldots, \mathbf{x}_1^{\mathsf{T}}, \underbrace{\mathbf{0}^{\mathsf{T}}, \ldots, \mathbf{0}^{\mathsf{T}}}_{(T-t-1)}]$$

More precisely, let $\mathbf{X}$ be the matrix which collects all the vectors of the above form by columns (for all sequences at any time step), if the input vectors $\mathbf{x}_i \in \mathbb{R}^k$ have zero mean, $s = T \cdot k$, $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}$ is the eigenvalue decomposition of $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ and $\widetilde{\mathbf{U}} \in \mathbb{R}^{s \times p}$ is the matrix obtained by $\mathbf{U}$ removing all the eigenvectors corresponding to null eigenvalues λ_i, the "optimal" matrices for an encoding space of dimension p can be defined as:

$$\widetilde{\mathbf{W}}_{\mathbf{x}} \equiv \widetilde{\mathbf{U}}^{\mathsf{T}} \underbrace{\begin{bmatrix} \mathbf{I}_{k \times k} \\ \mathbf{0}_{(s-k) \times k} \end{bmatrix}}_{\substack{\text{adding to the} \\ \text{first } k \text{ positions}}} \quad \text{and} \quad \widetilde{\mathbf{W}}_{\mathbf{y}} \equiv \widetilde{\mathbf{U}}^{\mathsf{T}} \underbrace{\begin{bmatrix} \mathbf{0}_{k \times (s-k)} & \mathbf{0}_{k \times k} \\ \mathbf{I}_{(s-k) \times (s-k)} & \mathbf{0}_{(s-k) \times k} \end{bmatrix}}_{\text{shifting to the right of } k \text{ positions}} \widetilde{\mathbf{U}}.$$

2.3 Trees

In [10] a similar, but a bit more elaborated result than the one presented for sequences, has been obtained for trees (with maximum outdegree b). Specifically, for trees the linear dynamical system to be considered is

$$\mathbf{y}_u = \mathbf{W}_{\mathbf{x}} \mathbf{x}_u + \sum_{c=0}^{b-1} \mathbf{W}_{\mathbf{c}} \mathbf{y}_{ch_c[u]} \tag{7}$$

where u is a node of the tree, $ch_c[u]$ is the $c + 1$-th child of u, and a different matrix $\mathbf{W}_{\mathbf{c}}$ is defined for each child.

Let us illustrate what happens for binary complete trees. For $b = 2$, we have the following linear model

$$\mathbf{y}_u = \mathbf{W}_{\mathbf{x}} \mathbf{x}_u + \mathbf{W}_{\mathbf{l}} \mathbf{y}_{ch_l[u]} + \mathbf{W}_{\mathbf{r}} \mathbf{y}_{ch_r[u]} \tag{8}$$

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, $\mathbf{W}_{\mathbf{l}}, \mathbf{W}_{\mathbf{r}} \in \mathbb{R}^{s \times s}$, where s is the dimension of the state space. In this case, the basic idea is to partition the state space $\mathbb{S}$ according to a perfectly balanced binary tree. More precisely, 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 $\mathbb{S}$ 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. Using this convention, extended state space vectors maintain the definition of eq. (6), where the first k components are used to store the current input label, i.e. the label associated to the root of the (sub)tree presented up to now as input, while the remaining components are defined according to the scheme described above.

Matrices $\mathbf{W_l}$ and $\mathbf{W_r}$ 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 $\mathbf{W_l}$ has to implement a push-left operation, 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 annilated. Thus, $\mathbf{W_l}$ 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 $\mathbf{W_r}$ is defined similarly: it has to implement a push-right operation, 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. Matrix $\mathbf{W_x}$ is defined as in the case of sequences. Performing the eigenspace analysis, we obtain the solution matrices $\widetilde{\mathbf{W}}_\mathbf{x} \equiv \widetilde{\mathbf{U}}^\top \mathbf{W_x}$, $\widetilde{\mathbf{W}}_\mathbf{l} \equiv \widetilde{\mathbf{U}}^\top \mathbf{W_l} \widetilde{\mathbf{U}}$, and $\widetilde{\mathbf{W}}_\mathbf{r} \equiv \widetilde{\mathbf{U}}^\top \mathbf{W_r} \widetilde{\mathbf{U}}$. A description of the construction for the general case, i.e. when $b > 2$, can be found in [10].

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. Thus, in practice, instead of considering each possible path in the complete tree, only paths that are present into the dataset are considered.

2.4 Graphs

When considering the possibility to extend Recursive PCA to graphs either with directed or undirected edges we have to face two problems: *i)* how to deal with cycles during the encoding; *ii)* how to identify the origin and destination of an edge during decoding.

In [8], these two problems are solved through a coding trick. The basic idea is to enumerate the set of vertexes following a given convention and representing a (directed or undirected) graph as an (inverted) ordered list of vertex’s labels associated with a list of edges for which the vertex is origin and where the position in the associated list is referring to the destination vertex. The idea is that the list is used by the linear dynamical system during encoding to read one by one the information about each vertex and associated edges, pushing the read information into the internal stack. Decoding is obtained by popping from the internal stack, one by one, the information about vertexes and associated edges.

The proposed linear dynamical system supporting the above idea is defined as

$$\mathbf{y}_i = \mathbf{W}_{\mathbf{v}}[\mathbf{v}_{label}^{\mathsf{T}}, \mathbf{v}_{edges}^{\mathsf{T}}]^{\mathsf{T}} + \mathbf{W}_{\mathbf{y}}\mathbf{y}_{i-1} \tag{9}$$

where i ranges over the enumeration of the vertexes, i.e. positions in the list representing the graph, $\mathbf{v}_{label} \in \mathbb{R}^k$ is the numerical encoding of the current label, $\mathbf{v}_{edges} \in \mathbb{R}^N$ is the vector representing the information about the edges entering the current vertex where N is the maximum number of vertexes that the system can manage for a single input graph, and $\mathbf{y}_0$ is the null vector. Thus $[\mathbf{v}_{label}^{\mathsf{T}}, \mathbf{v}_{edges}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{k+N}$ and the space embedding the explicit representation of the stack is $N(k+N)$ since no more than N vertexes can be inserted. It should be noted that this size of the stack is needed only if the input graphs are directed, and the above system is basically equivalent to system (2) for sequences.

However, if undirected graphs are considered, a specific state space optimization can be performed. In fact, when inserting the first vertex into the internal stack only the first entry of the vector $\mathbf{v}_{edges}$ may be non null (the one encoding the self-connection), since no other vertex has already been presented to the system. In general, if vertex i is being inserted, only the first i components of $\mathbf{v}_{edges}$ may be non null. Because of that, the shift operator embedded into matrix $\mathbf{W}_{\mathbf{y}}$ may "forget" the last component of each field into which the internal stack is organized. Formally, the shift operator described above can be implemented by the following matrix

$$\mathbf{S} \equiv \begin{bmatrix} \mathbf{0}_{d\times s} & & \\ \mathbf{I}_{(d-1)\times(d-1)} & \mathbf{0}_{(d-1)\times(s-d+1)} & \\ \mathbf{0}_{(d-2)\times d} & \mathbf{I}_{(d-2)\times(d-2)} & \mathbf{0}_{(d-2)\times(s-2(d-1))} \\ \mathbf{0}_{(d-3)\times(2d-1)} & \mathbf{I}_{(d-3)\times(d-3)} & \mathbf{0}_{(d-3)\times(s-3(d-1))} \\ & \cdots & \\ & \mathbf{0}_{(k+1)\times(s-k-1)} & \mathbf{I}_{(k+1)\times(k+1)} \end{bmatrix}$$

and the solution matrices defined as $\widetilde{\mathbf{W}}_{\mathbf{v}} \equiv \widetilde{\mathbf{U}}^{\mathsf{T}} \begin{bmatrix} \mathbf{I}_{d\times d} \\ \mathbf{0}_{(s-d)\times d} \end{bmatrix}$ and $\widetilde{\mathbf{W}}_{\mathbf{y}} \equiv \widetilde{\mathbf{U}}^{\mathsf{T}}\mathbf{S}\widetilde{\mathbf{U}}$.

3 Improving the Computation

In the previous section we have presented the theoretical basis for the definition of PCA for structured inputs. In this section we discuss some practical problems which are encountered when trying to apply the theory "as is". Then we make some observations about some specific features of structured domains. Finally, on the basis of these observations we suggest some techniques that help in reducing the computational burden, thus allowing the application of PCA for structures to larger and more complex datasets.

3.1 Practical problems

The definition of PCA for structures, as outlined above, hinges on the explicit definition of the state space in order to get the "compressed" solution of the problem, i.e. the weights matrices. From a practical point of view this implies that when considering datasets of significant size and complexity (in terms of number of components for single structure) a quite large matrix $\mathbf{X}$ describing the state space should be explicitly defined. Specifically, $\mathbf{X}$ will have a column for each single component of the dataset and a given number[1] of rows for each element of the encoding scheme. More precisely, in the case of sequences, k rows for each time step; in the case of trees, k rows for each path explicitly defined in the training data; finally, in the case of graphs, $k + N - i$ rows for each item in the internal stack at distance i from the top. It is not difficult to understand that when considering a large number of components and sufficiently deep structures, the global size of $\mathbf{X}$ can soon become unmanageable because of storage requirements. In addition to that, computing the eigenvalues and eigenvectors of $\mathbf{X}$ can become problematic even for small dimensions since additional storage is required for the internal data structures used for the computation. Least, but not last, even if storage requirements are satisfied, the time needed to perform the eigenanalysis is more than quadratic with respect to the size of $\mathbf{X}$. Thus, it is clear that strategies which try to keep $\mathbf{X}$ as small as possible and to reduce the computational time for its eigenanalysis should be defined in order to allow the treatment of datasets of significative size and complexity.

In the next subsection we observe that, even before resorting to more or less sophisticated numerical analysis techniques and algorithms, the structured nature of the data can be exploited to get a significative reduction of the size of $\mathbf{X}$, as well as a (fractional) reduction of the time needed to perform its eigenanalysis.

3.2 Some basic observations and their exploitation

Let make some observations about the structure of $\mathbf{X}$: *i)* the number of rows is determined by both the size of the representation of each possible component and the adopted structural encoding scheme; *ii)* if the components are finite and discrete, then it is quite probable that different structures will have one or more components in common. This implies that columns in $\mathbf{X}$ that correspond to these common components will be identical; this can be exploited by redefining a more compact version of $\mathbf{X}$ where each distinct component is represented only once, but considering its multiplicity; *iii)* a quicker eigenanalysis of $\mathbf{X}$ can be computed by precomputing a QR decomposition of either $\mathbf{X}$ or $\mathbf{X}^{\mathsf{T}}$.

Observation *i)* leads to a general scheme for defining a "minimal" state space. Observation *ii)* may lead to a significative reduction in size of $\mathbf{X}$ when considering structures compounded of discrete components, while observation *iii)* can be exploited in general.

In the following three subsections we discuss the above points, according to the order of presentation given above.

3.3 Defining a "minimal" state space

The state space defined in Section 2 is designed to be able to potentially represent all possible structures up to a given predefined limit. For example, when considering

[1] We previously assumed that each component could be described by a vector of dimension k.

sequences, all the possible sequences up to length T can be represented in the state space. The same is true for graphs. Only for trees, a strategy which tries not to represent all possible paths has been suggested: only paths defined in the dataset are represented in the state space. However, any input vector can be associated to any position within the defined paths. In conclusion, it is clear that, since PCA is computed on a specific dataset, there is no point in having such a general encoding scheme for the state space. It is better to devise from the beginning an encoding scheme for the state space which takes into account the dataset.

Here we suggest to define a state space where for each structural component only items which occur associated to it in the dataset are explicitly represented. For example, if we consider the set of sequences $Tr = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ up to length T, then given position i, the state space is designed to be able to represent only those input vectors $\mathbf{x}_i^{(s_1)}, \ldots, \mathbf{x}_i^{(s_i)}$ (more precisely, the subspace $Subs(Tr, i) = spanned(\mathbf{x}_i^{(s_1)}, \ldots, \mathbf{x}_i^{(s_i)})$ spanned by the input vectors) which occur at position i when they are processed. Please, note that: *i)* an input vector $\mathbf{x}_p^{(j)} \in \mathbb{R}^k$ that occurs at position p into the sequence j of length l_j, occurs into the state space as a sub-vector from position $k \cdot p$ to position $k(l_j - p)$, i.e. into the state space vectors

$$[\mathbf{x}_p^{(j)^\top}, \ldots, \mathbf{x}_1^{(j)^\top}, \underbrace{\mathbf{0}^\top, \ldots, \mathbf{0}^\top}_{(T-i)}], \;\; \cdots \;\; , [\mathbf{x}_{l_j}^{(j)^\top}, \ldots, \mathbf{x}_p^{(j)^\top}, \ldots, \mathbf{x}_1^{(j)^\top}, \underbrace{\mathbf{0}^\top, \ldots, \mathbf{0}^\top}_{(T-l_j)}];$$

ii) the matrix obtained by using the input vectors $\mathbf{x}_i^{(s_1)}, \ldots, \mathbf{x}_i^{(s_i)}$ as columns may have rank $r < k$ (full rank is k), which implies that if with B_i we refer to a basis of $Subs(Tr, i)$, then a "minimal" state space can be obtained by using as basis of the space the union of these bases, i.e. $B_{ss} = \bigcup_{i=1}^{T} B_i$. A suitable shift operator performing a change of basis at each position should correspondingly be defined, which is always possible.

If we consider the special case of structures where each component may be one among k different labels represented by vectors of the canonical basis, B_i will correspond to the subset of canonical vectors associated to the labels occurring at position i into the "state space stack". In this case the shift operator will just "forget" the components that are not present from that point up to the last components of the state space.

3.4 Representing unique substructures in the case of discrete components

When each component is described by a discrete entity, e.g. a symbol or label, the likelihood that different input structures share a common substructure is often not marginal. This is particularly true if the structures are created by a generative source, such as a generative model, e.g. a grammar. Repeated components correspond to repeated identical columns in $\mathbf{X}$. Without loss of generality, let assume that the columns of $\mathbf{X}$ are sorted so to have identical columns in contiguous positions. Let $\mathbf{r}_1, \ldots, \mathbf{r}_z$ be the all different columns occurring in $\mathbf{X}$, and $\mu_1, \ldots, \mu_z$ their multiplicity in $\mathbf{X}$. Then it is not difficult to verify that the matrix $\mathbf{X}^{\mathbf{red}} \equiv [\sqrt{\mu_1}\mathbf{r}_1, \ldots, \sqrt{\mu_z}\mathbf{r}_z]$ has the same covariance matrix as $\mathbf{X}$, and thus it shares the same column eigenvectors with $\mathbf{X}$. However, $\mathbf{X}^{\mathbf{red}}$ has only z columns versus $\sum_{i=1}^{z} \mu_i \geq z$ columns of $\mathbf{X}$. The problem, then, is

```
                              MinimalDAG

Input: A tree forest $F = \{T_1, \ldots, T_N\}$
  /* $l \equiv label, f \equiv frequency, tree \equiv tree\_rooted\_at$ */
  Initialize: $\mu D \leftarrow$ void DAG;
   for $j \leftarrow 1$ to $N$ do
   $vertex\_list \leftarrow$ InvTopologOrder$(T_j)$;
   while $vertex\_list \neq \emptyset$ do
       $v = pop(vertex\_list)$;
       if $\exists u \in \mu D$ s.t. $tree(u) \equiv tree(v)$ then $f(u) \leftarrow f(u) + 1$
           else add to $\mu D$ a node $w$ where $l(w) = l(v)$ and $f(w) = 1$
            forall children $ch_i[v]$ of $v$
               add arc $(w, c_i)$ to $\mu D$ where $c_i \in Nodes(\mu D)$ and $tree(c_i) \equiv tree(ch_i[v])$
   return $\mu D$
```

Fig. 1. The algorithm to transform a tree-forest into a minimal DAG, where all the subtrees are represented only once.

to discover these repeated components, and their multiplicity, even before generating columns representing them. Since graphs are basically treated as a special type of lists, here we focus on trees, which constitute the most general case since a list can be considered as a tree with outdegree 1. The problem is how to efficiently remove from the dataset repeated occurrences of (sub)trees. A dataset consisting of trees can be represented as a tree forest F. We define a procedure that merges all the trees in F into a single *minimal* DAG, i.e., a DAG with a *minimal* number of vertices. We will refer to this DAG as $\mu D = \mu DAG(F)$.

In Figure 1, we give an algorithm to efficiently compute shared subtrees, and to efficiently represent a forest as an annotated DAG (ADAG). More formally, with annotated DAG, we refer to a DAG where each node is annotated with a pair (*label*,*frequency*). The *label* field represents information associated with the node, while the *frequency* field is used to count how many repetitions of the same subtree rooted in that node are present in the tree forest. The frequency can then be used to define $\mathbf{X^{red}}$ The procedure `InvTopologOrder`(T_j) used in the algorithm returns a total order of vertexes of T_j which is compatible with the (inverted) partial order defined by the arcs of T_j. Thus, the first vertexes of the list will be vertexes with zero outdegree, followed by vertexes which have only children with zero outdegree, and so on. Using this order guarantees the (unique) existence of vertexes $c_i \in \mu D$ s.t. $tree\_rooted\_at(c_i) \equiv tree\_rooted\_at(ch_i[v])$. In fact, for each i, the vertex $ch_i[v]$ is processed before vertex v and is either inserted in μD or recognized as a duplicated of a vertex already present in μD.

It should be noted that the function $tree\_rooted\_at(\cdot)$ can be implemented quite efficiently by an indexing mechanism, where a unique code is defined for a void child, and a unique code for the root of each different (sub)tree is generated by recursively considering the label of the root and the (unique) codes computed for its children.

Exploiting the indexing mechanism described above, the overall time complexity of the algorithm is $O(n\log(n))$, where n is the total number of vertexes of the forest F.

Please, notice that both the encoding scheme (i.e. the different paths defined by the forest) and the columns of $\mathbf{X}^{\mathbf{red}}$ can be generated by a visit of $\mu DAG(F)$.

3.5 Exploiting QR decomposition

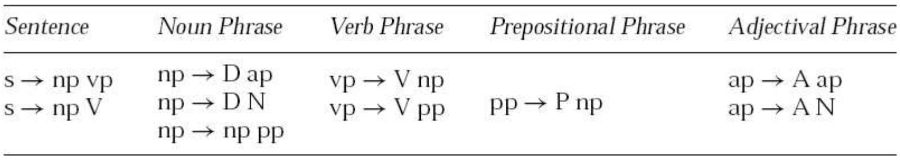
Eigenanalysis of the correlation matrix $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ can be performed in a robust and "economic" way by performing an SVD decomposition of either $\mathbf{X}$, in the case the number of rows is higher than the number of columns, or $\mathbf{X}^{\mathsf{T}}$ in the case the number of columns is higher than the number of rows. In fact, if we consider the SVD decomposition of $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}}$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\mathbf{S}$ is a diagonal matrix containing the singular values, the eigenvectors of $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ are the columns of matrix $\mathbf{U}$ and the corresponding eigenvalues are the square of the singular values stored in $\mathbf{S}$.

The SVD of $\mathbf{X}$ can be performed by first performing a QR decomposition of $\mathbf{X} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q}$ is an orthogonal matrix, and $\mathbf{R}$ is an upper triangular matrix, and then performing an SVD decomposition of $\mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}}$. In this way the eigenvectors are obtained by computing $\mathbf{Q}\mathbf{U}$. The decomposition described above is quicker than the direct SVD decomposition of $\mathbf{X}$ or $\mathbf{X}^{\mathsf{T}}$ if we choose to start from the matrix that produces a smaller $\mathbf{R}$, so that the successive SVD decomposition is quicker. Moreover, we do not need to compute the full product between $\mathbf{Q}$ and $\mathbf{U}$, since, already knowing the singular values returned in $\mathbf{S}$, only the relevant columns of the product need to be computed, savings additional computations.

4 Experimental Evaluation

Two datasets were used to test our approach. The first one is derived from the data set of the PTC (Predictive Toxicology Challenge, [5]) originally provided by the U.S. National Institute for Environmental Health Sciences - US National Toxicology Program (NTP) in the context of carcinogenicity studies. The publicly available dataset (see http://www.predictive-toxicology.org/data/ntp/) is a collection of about four hundred chemical compounds. The dataset includes a range of molecular classes and molecular dimension spanning from small and simple cases to medium size with multi-cycles. In order to represent these chemical structures and their components, we used undirected graphs with labels associated to vertexes and edges. The vertexes of these graphs correspond to the various atoms and the vertexes labels correspond to the type of atoms. The edges correspond to the bonds between the atoms and the edges labels correspond to the type of bonds. This explicit graph modeling can be obtained through the information directly extracted by standard formats based on connection table representation, limited, in our case, to the information on atoms type (including C and H), bond type (single, double or triple) and their 2D-topology, as implicit in the set of vertexes connections. Here, we do not assume any specific canonical ordering of such information, assuming directly the form provided in the original PTC data set.

For testing our approach, we have considered molecules with atoms occurring at least more than 3 times in the original data set and with a maximum dimension (number



Sentence	*Noun Phrase*	*Verb Phrase*	*Prepositional Phrase*	*Adjectival Phrase*
s → np vp s → np V	np → D ap np → D N np → np pp	vp → V np vp → V pp	pp → P np	ap → A ap ap → A N

Fig. 2. Context-Free Grammar used in the experiments.

of vertexes) of 70. In all, 394 distinct chemical compounds are considered, with the smallest having 4 atoms. 10 distinct atoms occur in the used data set, corresponding to the following chemical symbols: C, N, O, P, S, F, Cl, Br, H, Na. In Table 1 we report the frequencies of such atoms through the compounds as well as some general statistics.

Symbols are represented by 10-dimensional vectors (i.e. $k = 10$) following a "one-hot" coding scheme. Bond's type is coded by integers in the set $\{0, 1, 2, 3\}$, where 0 represents the absence of a bond and the other numbers are for single, double and triple bonds, respectively. Triple bonds occur only 5 times in the dataset. Double bonds occur 1509 times in the dataset. The remaining bonds are single. Since the graphs do not have self-connections for vertexes, we can avoid to represent the information about self-connections. Since the maximum number of vertexes in the dataset is $N = 70$, a standard full representation of the state space (stack size) would require $s = \sum_{i=0}^{N-1}(d-i) = 3115$ different components, since the graphs are undirected. We used the dummy state $\mathbf{y}_{dummy}$ described in [10] to get zero-mean vectors.

The second dataset is given by parse trees derived by the context-free grammar shown in Figure 2, and already used by Pollack [7]. In all 421 distinct parse trees have been randomly generated for a total of $14,815$ nodes. Terminal and nonterminal symbols are represented by 6-dimensional vectors (i.e. $k = 6$), where the first component is 0 for terminal symbols and 3 for nonterminal symbols, while the remaining 5 components follow a "one-hot" coding scheme. Since in the dataset there were 1193 distinct paths when processing the trees, a standard full representation of the state space (stack size) would require $s = 1193 \times 6 = 7158$ components.

In Table 2 we have reported the results obtained by using an Intel Xeon E5345 based computer using Scilab.

Chemical Symbol	C	N	O	P	S	F	Cl	Br	H	Na
Frequency	3608	417	766	25	76	11	326	46	4103	22

# examples	Max. number atoms	Max. number bonds	Avg. number atoms (bonds)	Tot. number items (atoms+bonds)
394	70	73	23.86 (24.20)	18,936

Table 1. Occurrences of atoms symbols in the chemical dataset and some of its statistical properties.

5 Conclusion

We have suggested a way to speed-up the computation of principal components for structured input. The validity of the proposed approach has been experimentally evaluated with quite good results. Additional improvements, not explored in this paper, can be obtained by considering the sparsity of the $\mathbf{X}$ matrix, and the adoption of more sophisticated numerical algorithms for its eigenanalysis.

References

1. P. Baldi and G. Pollastri. The principled design of large-scale recursive neural network architectures-DAG-RNNs and the protein structure prediction problem. *Journal of Machine Learning Research*, 4:575–602, 2003.
2. P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Trans. Neural Networks*, 9(5):768–786, 1998.
3. T. Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, 2003.
4. B. Hammer. *Learning with Recurrent Neural Networks*, volume 254 of *Springer Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2000.
5. Helma, C., King,R.D., Kramer,S., Srinivasan,A.: The predictive toxicology challenge 2000-2001. Bioinformatics, **17**(1) (2001), 107–108.
6. Jolliffe, IT. (2002). Principal Component Analysis. Springer-Verlag.
7. Pollack, J.B.: Recursive distributed representations. Artificial Intelligence **46** (1990) 77–105.
8. A. Micheli, A. Sperduti. Recursive Principal Component Analysis of Graphs. *ICANN 2007*. To appear.
9. A. Micheli, A. Sperduti, A. Starita, and A. M. Bianucci. Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines. *Journal of Chem. Inf. and Comp. Sci.*, 41(1):202–218, January 2001.
10. Sperduti, A.: Exact Solutions for Recursive Principal Components Analysis of Sequences and Trees. ICANN 2006 (2006)349–356 .
11. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks **8** (1997) 714–735.

Dataset	# components for "minimal" state space (size full space)	Time in sec. direct SVD full matrix	Time in sec. QR+SVD full matrix	# col. in μDAG matrix	Time in sec. direct SVD μDAG matrix	Time in sec. QR+SVD μDAG matrix
Graphs	1587 (3115)	313.25	222.87	2020	182.36	138.91
Trees	2795 (7158)	1556.95	1063.41	1217	77.3	63.08

Table 2. Experimental results.