Recursive Principal Component Analysis of Graphs

Alessio Micheli[1] and Alessandro Sperduti[2]

[1] Department of Computer Science, University of Pisa,Italy
[2] Department of Pure and Applied Mathematics, University of Padova, Italy

Abstract. Treatment of general structured information by neural networks is an emerging research topic. Here we show how representations for graphs preserving all the information can be devised by Recursive Principal Components Analysis learning. These representations are derived from eigenanalysis of extended vectorial representations of the input graphs. Experimental results performed on a set of chemical compounds represented as undirected graphs show the feasibility and effectiveness of the proposed approach.

1 Introduction

The representation of graphs via numerical vectors is the first necessary step to the application of numerical methods for clustering, classification, and regression. Traditional approaches select apriori a set of structural features of interest and represent each graph via a vector where each component reports how much the associated structural feature is matched. An example of this approach can be found in QSPR/QSAR studies in Chemistry, where topological indexes are used as features.

A different approach has been proposed in the last 10 years in the neural networks field, where Recursive Neural Networks have been proposed and successfully applied in applications involving structured patterns (e.g. see [8, 2, 4, 1, 6]). The underpinning idea at the basis of this type of neural networks is the dynamic generation of feedforward networks (encoding networks) whose topology matches the topology of the input and which exploit shared weights to cope with structures of different sizes. The output of these encoding networks is a numerical representation of the input structure. The advantage of this approach with respect to the former approach is that learning procedures can be exploited to adapt the numerical representations to the classification or regression task at hand. For example, if another neural network (output network) is used to post-process (either for producing a classification or a numerical prediction) the output of the encoding networks, the error obtained by the output network can be back-propagated to the encoding networks and the (shared) weights of the encoding networks adapted to contribute to the minimization of the loss function of interest. Almost all the proposed models in the family of recursive neural networks are only able to directly deal with directed acyclic graphs (and other derivable structures, such as trees and sequences).

An additional approach has been pursued by kernel methods for structured patterns (see [3] for a survey). These methods avoid to explicitly generate numerical vectors representing the input graphs, but directly compute the similarity between two graphs through a kernel function that implicitly projects each graph into a numerical feature space and that returns the dot product between corresponding vectors into the feature space. One problem with this approach is that almost all the proposed kernels are defined for structures with vertexes annotated by discrete variables, and very often the calculation of the kernel is computationally very demanding. Finally, as in the case of topological indexes, kernels are usually defined apriori, regardless of the specific task of interest and regardless of the available dataset.

In this paper, we address the problem to devise vectorial representations of graphs from a dataset preserving all the information needed to discriminate among them. Specifically, we show how a recently proposed approach to the calculation of Recursive PCA [7] for sequences and trees can be adapted to graphs, either with directed or undirected arcs. 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. The applicability and effectiveness of the proposed method is evaluated on a dataset of chemical compounds of significant diversity and sizes, involving thousands of atoms and bonds.

2 Principal Components Analysis for Sequences and Trees

In [7] it is shown how Principal Component Analysis can be extended to the direct treatment of sequences and trees. More specifically, given a temporal sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$ of input vectors $\mathbf{x}_i \in \mathbb{R}^k$, where t is a discrete time index, we are interested in modeling the sequence through the following linear dynamical system:

$$\mathbf{y}_t = \mathbf{W}_\mathbf{x}\mathbf{x}_t + \sqrt{\alpha}\mathbf{W}_\mathbf{y}\mathbf{y}_{t-1} \tag{1}$$

where $\mathbf{W}_\mathbf{x} \in \mathbb{R}^{p\times k}$ and $\mathbf{W}_\mathbf{y} \in \mathbb{R}^{p\times p}$ are the matrices of synaptic efficiencies, which correspond to feed-forward and recurrent connections, respectively, $\mathbf{y}_t \in \mathbb{R}^p$ is an output vector, $\alpha \in [0,1]$ is a gain[3] parameter which modulates the importance of the past history, i.e. $\mathbf{y}_{t-1}$, with respect to the current input $\mathbf{x}_t$. The aim is to define proper synaptic matrices, with *dimension p as small as possible*, 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. (1), and then, starting from the obtained encoding $\mathbf{y}_t$, it should be possible to reconstruct backwards the original sequence using the transposes of $\mathbf{W}_\mathbf{x}$ and $\mathbf{W}_\mathbf{y}$. This requirement implies that the following equations

$$\mathbf{x}_t = \mathbf{W}_\mathbf{x}^\mathsf{T}\mathbf{y}_t \tag{2}$$

$$\mathbf{y}_{t-1} = \mathbf{W}_\mathbf{x}\mathbf{x}_{t-1} + \mathbf{W}_\mathbf{y}\mathbf{y}_{t-2} = \mathbf{W}_\mathbf{y}^\mathsf{T}\mathbf{y}_t \tag{3}$$

should hold. In fact, the aim of 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 (2) and the original input vectors for different values of t

$$error(t) = \sum_{i=1}^{t} ||\mathbf{x}_i - \mathbf{W}_\mathbf{x}^\mathsf{T}(\mathbf{W}_\mathbf{y}^\mathsf{T})^{t-i}\mathbf{y}_t||^2 \tag{4}$$

[3] Here, without loss of generality, we focus on the case where $\alpha = 1$ and $\mathbf{y}_0$ is the null vector.

is as small as possible, i.e. we look for the smallest value of p such that $error(t)$ is minimized.

In [7] it has been shown that, when considering several sequences but the same synaptic matrices, zero error, i.e. an exact solution to the above minimization error, can be obtained by performing eigenanalysis of extended vectorial representations of the input sequences, where a sequence at time t is represented by the vector

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

where T is 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 (for all sequences at any time step). If the input vectors $\mathbf{x}_i \in \mathbb{R}^k$ have zero mean, $s = T \cdot k$, $\mathbf{U\Lambda U}^\mathsf{T}$ is the eigenvalue decomposition of $\mathbf{XX}^\mathsf{T}$ 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, then p^* is the smallest value for which the synaptic matrices defined as:

$$\widetilde{\mathbf{W}}_\mathbf{x} \equiv \widetilde{\mathbf{U}}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{I}_{k \times k} \\ \mathbf{0}_{(s-k) \times k} \end{bmatrix}}_{\text{adding to the first } k \text{ positions}} \in \mathbb{R}^{p^* \times k}$$

and

$$\widetilde{\mathbf{W}}_\mathbf{y} \equiv \widetilde{\mathbf{U}}^\mathsf{T} \underbrace{\begin{bmatrix} \mathbf{0}_{k \times (s-k)} & \mathbf{0}_{k \times k} \\ \mathbf{I}_{(s-k) \times (s-k)} & \mathbf{0}_{(s-k) \times k} \end{bmatrix}}_{\text{shifting to the right of } k \text{ positions}} \widetilde{\mathbf{U}} \in \mathbb{R}^{p^* \times p^*},$$

have $error(T) = 0$. Please, note that smaller synaptic matrices can be obtained by removing from $\widetilde{\mathbf{U}}$ eigenvectors corresponding to smallest eigenvalues, i.e. $p < p^*$. Doing that, however, it is not clear whether the optimal value of $error(T)$ given p is obtained.

A similar, but a bit more elaborated result can be obtained for trees (with maximum outdegree b), where the linear dynamical system considered is

$$\mathbf{y}_u = \mathbf{W}_\mathbf{x}\mathbf{x}_u + \sum_{c=0}^{b-1} \mathbf{W}_\mathbf{c}\mathbf{y}_{ch_c[u]} \qquad (5)$$

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.

3 Graphs and Recursive Principal Component Analysis

The basic idea of standard Principal Component Analysis is to discover orthogonal directions (i.e. principal components) of maximum variance of the data. These directions allow to define the subspace of smallest dimensionality where data is embedded. A nice feature of principal components is that they define a (linear) projection from the original space to the embedding space which can be "inverted" so to reconstruct the "original" vector from its projection into the embedding space. The projection into the embedding space (encoding) and the reconstruction from the embedding space (decoding) are operations which are preserved also into the recursive version of PCA for sequences and trees. 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.

Cycles may be present in directed graphs and are present in undirected graphs by definition[4]. Their presence is problematic when considering the encoding function since it introduces mutual functional dependences among vertexes. In fact, the encoding function is usually defined by induction: the *basis* is applied to vertexes with no out-coming edges, for which there is no functional dependency, and the *induction step* is applied to the remaining vertexes. For example, in the case of rooted trees, the encoding for leaves (*basis*) is given only as function of the label attached to them, while the encoding for internal vertexes is given as function of both the attached label and the encoding of the children. The encoding of a whole tree is obtained by considering the encoding for the root of the tree. If a cycle is present, the above scheme suffers the problem of mutual recursion, which from a mathematical point of view can be translated as the definition of a (linear) dynamical system whom state vector (i.e., the output of our encoding function) may eventually diverge or converge to a single or few attractors. In the first case, no stable encoding can be obtained; in the second case, the same encoding (i.e. attractor) is obtained for different input graphs, which consequently cannot be discriminated.

The second main problem, i.e. the identification of the origin and destination of an edge during decoding, is not present in the case of sequences (which can be seen as linked lists) and trees, since each vertex in this type of data structures is reachable by a single path from the beginning of the list or from the root, respectively. This property implies that it is possible to define the decoding function again by an inductive process: the *basis* is applied to vectors in the embedding space which lie in a designed subspace (e.g. around the origin, or which satisfy a specific "termination" property), i.e. when a vector in the embedding space belongs to the designed subspace or it satisfies the specific termination property, the decoding process is terminated for that vector; the *induction step* is applied when the basis does not apply, i.e. the information about the label is generated as a function of the vector, as well as one (for lists) or several (for trees) further vectors in the embedding space to which the inductive process is recursively applied. This decoding scheme generates trajectories into the origin space which can unambiguously be assigned to paths in a tree (or a single path for a list). When con-

[4] In fact, edges into undirected graphs can be traveled in both directions, and thus any graph with at least one edge generates a cycle of length 2 if the connected vertexes are different.

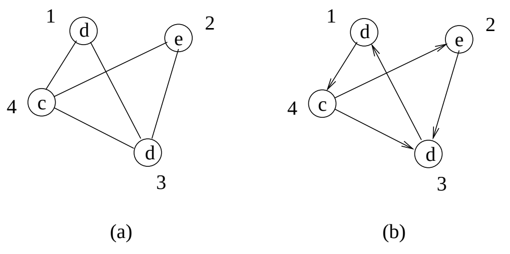


Fig. 1. Examples of undirected (a) and directed (b) graphs with labeled vertexes. The integer number associated to each vertex constitutes the enumeration of the vertex within the same graph.

sidering graphs the above scheme cannot work, since in general a vertex can be reached by several paths, and it is not obvious how to assign a "vertex" semantic to each step of the generated trajectories, i.e. if the same label is generated by different trajectories or from the same trajectory at different decoding steps, how can we be sure that its interpretation is that the same vertex of the graph has been generated via different paths or we are facing the generation of different vertexes with the same label ?

Here we propose to solve the above problems with 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) neural network during encoding to read one by one the information about each vertex and associated edges, pushing the read information into an internal stack (the encoding space). Decoding is obtained by popping from the internal stack, one by one, the information about vertexes and associated edges. Just to give a concrete example, let consider the two graphs in Figure 1. The enumeration for each vertex is reported as an integer besides each vertex. Assuming that the maximum number of vertexes in the input graph domain is 4, the representation for the undirected graph shown in Figure 1(a) is as follows

$$[(c,[0]),(d,[0,1]),(e,[0,1,1]),(d,[0,0,1,1])]. \tag{6}$$

The representation is used as follows during the encoding process: the first element of the list tells the neural network to push into the internal stack a vertex with associated label c and no edge with itself. Then the neural network reads the second element of the list: a vertex with label d is pushed into the internal stack together with the information that the current vertex has no edge with itself, but shares an edge with the vertex previously pushed into the stack. Subsequently the third element of the list is read and the neural network pushes into the internal stack a vertex with label e and the following information about edges: no edge with itself, shared edge with the vertex previously inserted into the internal stack, and shared edge with the vertex inserted two time steps before, and so on. Please, note that, since the edges are undirected it is sufficient to represent only the upper (or lower) part of the incidence matrix describing the connectivity of the graph.

For the directed graph shown in Figure 1(b) the representation is as follows

$$[(c, [0, 0, 0, 1]), (d, [1, 0, 1, 0]), (e, [1, 0, 0, 0]), (d, [0, 1, 0, 0])].$$

The use of the representation during the encoding is similar to the one described above, with the difference that now the full incidence matrix should be represented in order to retain the information about the direction of the edges. Thus, when considering the first element of the list, the interpretation of the information about the edges, i.e. the list $[0, 0, 0, 1]$, should be understood as follows: the first element of the associated edge list is 0, which means that there is no edge arriving from the vertex pushed as first into the internal stack; the same for the second element and for the third one; the last element of the list is 1, which means that there is an edge arriving from the vertex which will be pushed as fourth into the internal stack.

A linear dynamical system supporting the above idea may be the following

$$\mathbf{y}_i = \mathbf{W}_{\mathbf{v}}[\mathbf{v}_{label}^{\mathsf{T}}, \mathbf{v}_{edges}^{\mathsf{T}}]^{\mathsf{T}} + \mathbf{W}_{\mathbf{y}}\mathbf{y}_{i-1} \tag{7}$$

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 $\mathbf{v}^{\mathsf{T}} = [\mathbf{v}_{label}^{\mathsf{T}}, \mathbf{v}_{edges}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{k+N}$ and the space embedding the explicit representation of the stack is $s = N(k+N)$ 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 (1) for sequences.

However, if undirected graphs are considered, a specific 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. Just to exemplify this point, let consider the encoding of graph (6). Recall that we assumed that the maximum number of vertexes per graph was 4 and let assume that input symbols are coded via a 10 bits code, so we have for each vertex a coding vector $\mathbf{v}$ of dimension $d = k + N = 14$. Now let consider the organization of the internal stack when all the vertexes of the graph have been read. It can be readily understood that the stack only needs $14 + 13 + 12 + 11$ bits. In fact, the first vertex inserted into the stack has a single edge bit which is non null, the second vertex only 2 bits, and so on. Thus, all the codes for the inserted vertexes can loose the current last bit of the code every time they are shifted to the right because of a push into the stack. Since the first inserted vertex (code) is shifted to the right 3 times, it will "forget" the last three bits of the code, which however are 0s since the first inserted vertex can just have coded an edge which is a self-connection. The second inserted vertex (code) will be shifted to the right 2 times, so it will loose the last 2 bits of the code, which however are 0s since the second inserted vertex can just have coded one edge as self-connection and a second one as a connection with the first inserted vertex, and so on for the other inserted vertexes.

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 optimal 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 $\widetilde{\mathbf{W}}_{\mathbf{y}} \equiv \widetilde{\mathbf{U}}^{\mathsf{T}}\mathbf{S}\widetilde{\mathbf{U}}$.

4 Experimental Evaluation

The data used for testing our approach 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. Figure 2 shows four compounds of the data set using the typical chemical graphical visualization where the vertexes without symbols are carbon atoms (C) and the hydrogens (H) and their bonds (completing the carbon valence) are not shown (hydrogen suppressed graphs). As shown in Figure 2 the data include 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 use for each compound undirected vertex labeled and edge labeled graphs (i.e. a graph 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

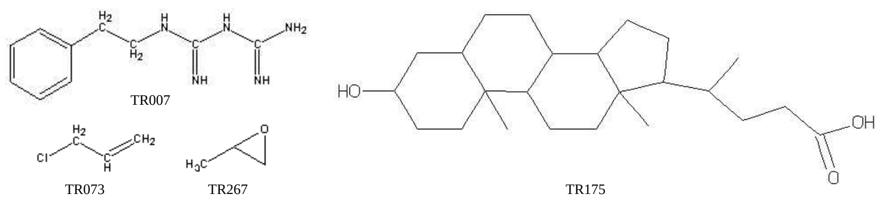


Fig. 2. Four chemical compounds belonging to the used data set.

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 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. Among the 394 compounds, 235 graphs are selected for training, and the remaining 159 graphs are used for testing the generalization ability of the system, i.e. the ability to successfully decode the components of the input chemical compound starting from the vector encoding the whole compound. In Table 1 we have summarized some general statistics about each split.

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 2 times in the training set and 3 times in the test set. Double bonds occur 910 times in the training set and 599 times in the test set. 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. Thus, because the maximum number of vertexes in the dataset is $N = 70$, we have an input dimension for each vertex which is $d = k + (N-1) = 10 + 69 = 79$ (recall that we do not consider self-connections) which leads to a stack size of $s = \sum_{i=0}^{N-1}(d-i) = 3115$, since the graphs are undirected. We used the dummy state $\mathbf{y}_{dummy}$ described in [7] to get zero-mean vectors.

The spectral analysis required around 27 cpu/min on an Athlon 1900+ based computer using Scilab. Values for the main eigenvalues are plotted in Figure 3.

In Figure 4 we have reported the training (top) and test (bottom) decoding errors for both label atoms and edges. The error in decoding is computed as follows. Each graph is first fed into the system, so to get the final encoding $\mathbf{y}$ for the graph. Then the final encoding is decoded so to regenerate all the items (atom and bond labels) of the graph. A decoded atom label is considered to be correct if the position of the highest value in the decoded label matches the position of the 1 in the correct label, otherwise

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

Dataset Split	# examples	Max. number atoms	Max. number bonds	Avg. number atoms (bonds)	Tot. number items (atoms+bonds)
Training	235	70	73	24.42 (24.76)	11,557
Test	159	67	66	23.03 (23.38)	7,379
Total	394	70	73	23.86 (24.20)	18,936

Table 1. Occurrences of atoms symbols in the data set and statistical properties of the dataset and of each split.

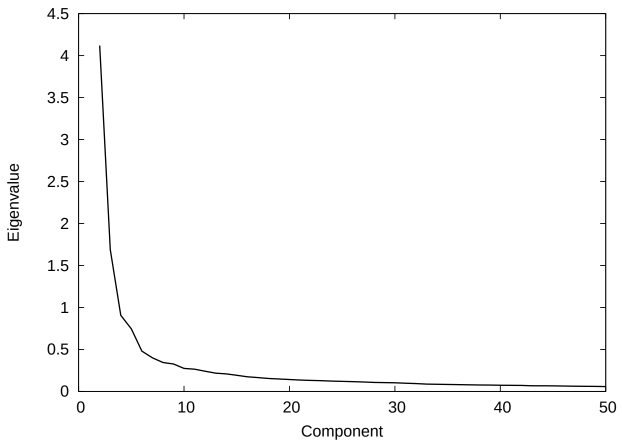


Fig. 3. Eigenvalues from rank 2 to 50 are shown. The most significant eigenvalue, caused by the introduction of $\mathbf{y}_{dummy}$, is not shown since it is very high (32189.69), as well as eigenvalues beyond rank 50. Only 949 eigenvalues over 3115 are non-null, i.e. $p^* = 949$.

a loss of 1 is suffered. A decoded bond entry is considered to be correct if its rounding to the nearest integer matches the target bond entry. If there is a mismatch a loss of 1 is suffered.

The final error is computed as the ratio between the total loss suffered and the total number of items (atom labels and total number of bond entries) in the dataset. For the bond entries we have normalized with respect to the number of bits that have been explicitly decoded by the system.

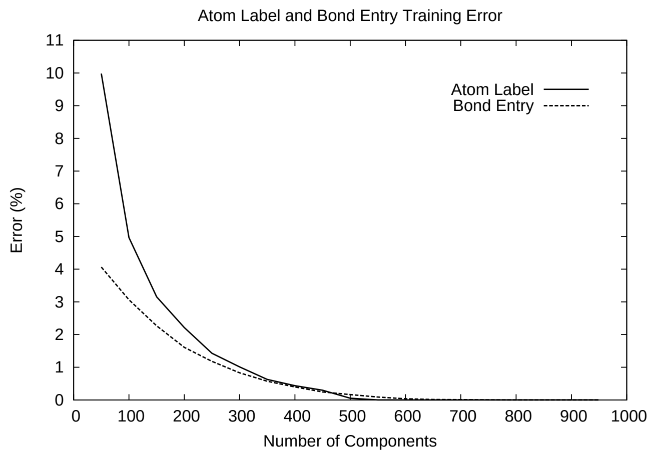
From the experimental results it is clear that learning is quite successful. In fact, with as few as 350 components it is possible to get a training error below 1% and a test error below 2% for both atoms and edges labels.

5 Conclusion

We have suggested a way to compute recursive principal components for both directed and undirected graphs with labeled vertexes and edges. Feasibility and efficacy of the proposed approach has been demonstrated on a dataset of chemical compound of significant variety and size. The obtained representations are quite informative and can be used as input vectors for any type of classification or regression method, such as Neural Networks and Support Vector Machines.

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.



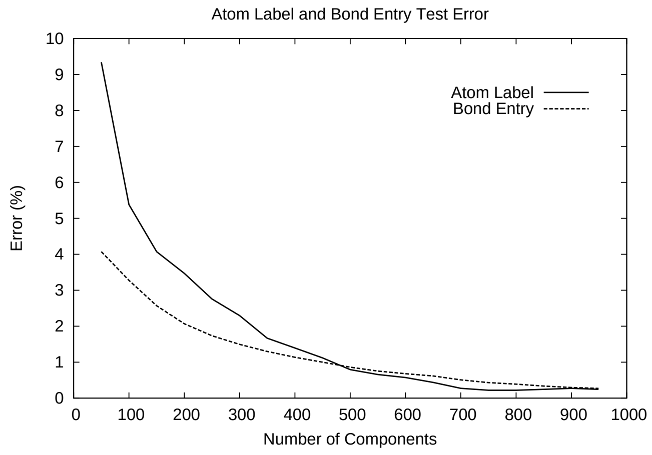


Fig. 4. Plots of the experimental results obtained for the training (top) and test (down) sets.

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. New.*, 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. A. Micheli, A. Sperduti, A. Starita, A. M. Bianucci. Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines. *J. of Chem. Inf. and Comp. Sci.*, 41(1):202–218, 2001.
7. Sperduti, A.: Exact Solutions for Recursive Principal Components Analysis of Sequences and Trees. ICANN 2006 (2006)349–356 .
8. Sperduti, A., Starita, A.: Supervised neural networks for the classifi cation of structures. IEEE Transactions on Neural Networks **8** (1997) 714–735.