A Relationship Oriented Framework for Learning in Structured Domains

Madhusudan Paul, Thamizh Selvam. D, P. Syam Kumar and Dr. R. Subramanian

Department of Computer Science, School of Engineering and Technology,
Pondicherry University, Puducherry, India
{msp.cse@gmail.com, dthamizhselvam@gmail.com, shyam.553@gmail.com and rsmanian.csc@pondiuni.edu.in}

Abstract: Most of the classical machine learning (ML) models was developed to deal with non-structured domains of learning where data in input domain is represented as fixed size vector of properties. This kind of representation cannot always capture the true nature of the data which are naturally represented in some structured form like sequences and trees. Learning in structured domains (SDs) is a new field of study which allows for a generalization of ML approaches to the treatment of complex data, offering both new impulses for theory and applications. Since relationships are the key in SD learning, edges (relationships) are more important than vertices (indivisible component of objects) in SD learning. In the existing framework for graph processing of artificial neural network models, vertices are given more importance than edges. Again geometrical information of structured data is not considered in the framework. In this paper, a new framework for graph processing is proposed in which edges are considered as key and also geometrical information is taken into account.

Keywords: Artificial Neural Networks, Cascade Correlation Learning, Feedforward Neural Networks, Learning in Structured Domains, Recursive Neural Networks**.**

1. Introduction

In traditional machine learning, an input object in the input domain is represented by a fixed-size vector of properties (or features). Though this kind of representation is quite easy to realize and process, sometimes it cannot completely capture the “true nature” of the data which naturally presents itself in a structured form, since some important contextual information may be associated with the structure of the data itself. While learning in structured domains is quite difficult to realize and process, it is the generalized machine learning approach to deal with complex data structures. A non-structured domain can be thought as a special or restricted case of structured domains. A domain of real valued vectors can be treated as a special case of a domain of sequences of real valued vectors (i.e., it is a domain of sequences, all of size one), which in turn can be considered as a special case of a domain of trees (i.e., a sequence can be treated as a tree where all the internal nodes have a single child), which in turn can be considered as a special case of a domain of directed acyclic graphs (DAGs), and so on. Therefore, focusing on structured domains, does not exclude the possibility to exploit the traditional vector-based learning models. In fact, traditional vector-based approaches are just specific instances of a more general structure-based framework.

In traditional machine learning approaches, graphs or trees are mapped into simpler representations, like vectors. However the performances of these approaches differ largely with the application at hand. In fact, the preprocessing phase is quite problem dependent and the implementation of this approach usually requires a time-consuming trial and error procedure. Moreover, the inherent topological information contained in structural representations might be partially lost.

Recently, new connectionist models, capable of directly elaborating trees and graphs without a preprocessing phase were proposed [1]. These have been extended using support vector machines [2]–[5], recursive neural networks [6] – [13] and SOMs [14] to structured data.

In the family of recursive neural networks (RNNs), constructive approach, recursive cascade correlation (RCC), has been introduced in [6]. RNN models realize an adaptive processing (encoding) of recursive (hierarchical) data structures. A recursive traversal algorithm is used to process (encode) all the graph vertices, producing state variable values for each visited vertex. In fact, all of these approaches can handle only sub-classes of graphs, not general graphs [i.e., rooted trees, directed acyclic graphs (DAG), and directed positional acyclic graphs (DPAG)]. Very recently as a first attempt, Neural Networks for Graphs (NN4G) [16] introduced the concept to the treatment of general class of graphs. It is an incremental approach where state values of vertices are updated gradually using cascade correlation learning algorithm. The principle idea behind the framework of the models is to obtain a flat description of the information associated to each vertex of graphs. In our proposed framework the idea is to obtain a flat description of the information associated to each edge instead of each vertex of graphs.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries and notations on the domains. In Section 3, existing framework for graph processing is discussed. Our new proposed framework is explained in Section 4. The proposed framework is again enhanced in Section 5 considering geometrical information. Finally, future directions and conclusion is given in Section 6.

2. Preliminaries and notations

A *labeled graph* (or *graph*) g is a quadruple (V, E, L_v, L_e), where V is the nonempty finite set of vertices/nodes, and E is the finite set of edges: $E \subseteq \{(u,v) | u, v \in V\}$. The last two items L_v and L_e associate a vector of real numbers to each vertex and edge, respectively. In fact, L_v and L_e are mappings as $L_v : V \rightarrow \mathbb{R}^{d_v}$ and $L_e : E \rightarrow \mathbb{R}^{d_e}$, where $\mathbb{R}^{d_v}$ and $\mathbb{R}^{d_e}$ are the sets of vectors of real numbers with dimension d_v and d_e, respectively. Vertex labels and edge labels are denoted by l_v, and $l_{(u,v)}$ (or l_e) respectively. The symbol l with no further specification represents the vector obtained by stacking together all the labels of the graph. The symbol | · | denotes the cardinality or the absolute value, according to whether it is applied to a set or to a number.

If g is *directed* graph, each edge (u, v) of g is an *ordered* pair of vertices, where v is the *children* or *successor* of the *parent* or *predecessor* u. For undirected graph, the ordering between u and v in (u, v) is not defined, i.e., $(u, v) = (v, u)$. Vertex v and edge e are said to be *incident* with (on or to) each other, if vertex v is an end vertex of edge e. The number of edges incident on a vertex v with self-loops counted twice is called the *degree* of vertex v and denoted by $d(v)$. A *cycle* is a finite alternating sequence of vertices and edges beginning and ending with same vertex such that each edge is incident with the vertices preceding and following it and no edge and vertex (except initial and final vertex) are repeated. A graph with no cycle is called an *acyclic* graph.

Given a set of labeled graphs G and a graph $g \epsilon G$, we denote the set of vertices of g as $V(g)$ and the set of its edges as $E(g)$. Given a vertex v, the vertices *adjacent* to v (or *neighbors* of v) are those connected to it by an edge and are represented by $N(v)$, i.e $N(v) = \{u \epsilon V(g) | (u, v) \epsilon E(g)\}$. Hence, $d(v) = |N(v)|$.

Similarly, given an edge e, the edges adjacent to e (or *neighbors* of e) are those edges having a common end vertex with the given edge and are denoted by $N(e)$, i.e., $N(e) = \{(u, v) \epsilon E(g) | u \in adj(e) or\ v \in adj(e)\}$, where $adj(e)$ is the set of two end vertices of e.

If the graph is directed, the neighbors of v (or e), either belong to the set of its *children* or *successors* $S(v)$ (or $S(e)$) or to the set of its parents or predecessors $P(v)$ (or $P(e)$).

A graph is said to be *positional* if a function π is defined for each vertex v and assigns a different position $\pi(u)$ to each neighbor $u \epsilon N(v)$, otherwise the graph is *non-positional*. Thus, combining the properties described so far it is possible to specify various graph categories: Directed Acyclic Graphs (DAGs), Directed Positional Acyclic Graphs (DPAGs) and so on.

Here we assume a class of input structured patterns as labeled graphs. Let a target function τ be defined as $\tau : g \times v \rightarrow \mathbb{R}^m$ (or $\tau : g \times e \rightarrow \mathbb{R}^m$), i.e., τ maps a graph g and one of its vertex v (or edge e) into a vector of real numbers. Our objective is to approximate the target function τ. More precisely, in classification problems codomain of τ is $\mathbb{N}^m$ (i.e., vectors of natural numbers), whereas in regression problems the codomain is $\mathbb{R}^m$. In graph classification, τ does not depend on v (or e), i.e. only one target is given for each graph. In vertex (or edge) classification problems, each vertex (or edge) in a given set has a target to be approximated.

In this paper we face the problem of devising neural network architectures and learning algorithms for the classification of structured patterns, i.e., labeled graphs. Fig. 1 reports the standard way to approach this problem using a standard neural network. Each graph is encoded as a fixed-size vector which is then given as input to a feedforward neural network for classification. This approach is motivated by the fact that neural networks only have a fixed number of input units while graphs are variable in size. The encoding process is usually defined in advance and does not depend on the classification task. It is a very expensive trial and error approach.

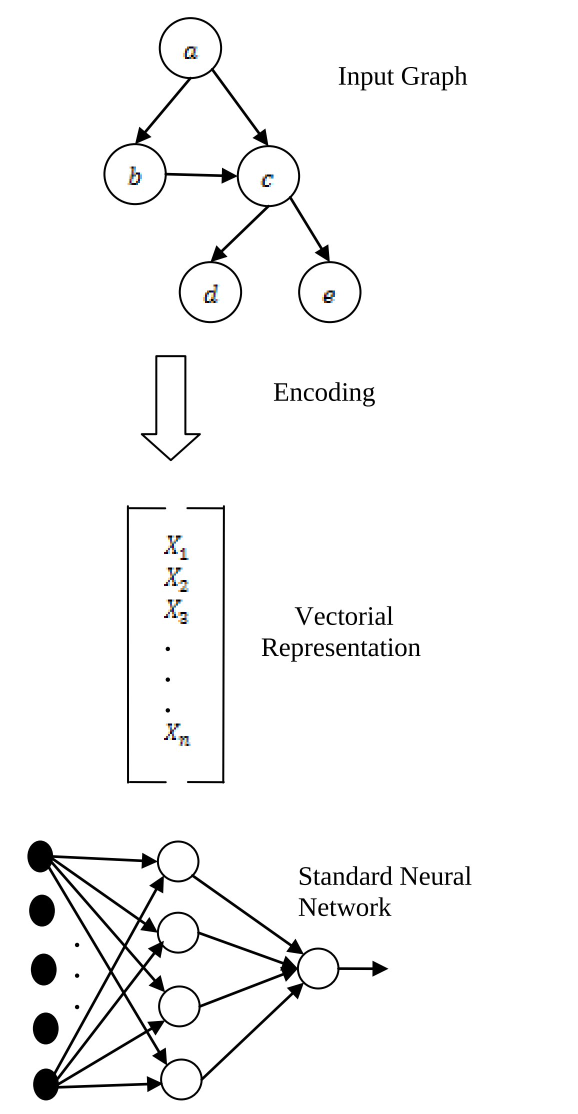


Figure 1. Classification of graphs by standard neural networks

Standard and recurrent neurons are not suitable to deal with labeled structures. In fact, neural networks using this kind of neurons can deal with approximation and classification problems in structured domains only by using a complex and very unnatural encoding scheme which maps structures onto fixed-size unstructured patterns or sequences. To solve this inadequacy of standard and recurrent neural networks the *generalized recursive neuron* was proposed in [6].

The generalized recursive neuron is an extension of the recurrent neuron where instead of just considering the output of the unit in the previous time step, we consider the outputs of the unit for all the vertices which are pointed by the current input vertex.

3. General Framework for Graph Processing

To define a general framework for graph processing, we need to implement a function $\varphi : g \times v \rightarrow \mathbb{R}^m$ to compute an output $\varphi(g, v)$ for each pair (g, v). The principle idea is to derive a flat description of the information associated to each vertex v. An object of the domain of interest can be represented by a vertex and its description is represented by a vector of real numbers called *state* denoted by $x_v \in \mathbb{R}^s$, where the *state dimension* s is a predefined parameter. In order to obtain a distributed and parallel processing framework, the *states* are computed locally at each vertex. A reasonable choice is to design x_v as the output of a parametric state transition function f_t, that depends on the vertex label l_v and on the relationships between v and its neighbors

$$x_v = f_t\left(l_v, x_{N(v)}, l_{N(v)}, l_{(v,N(v))}\right), \quad v \in V \qquad (1)$$

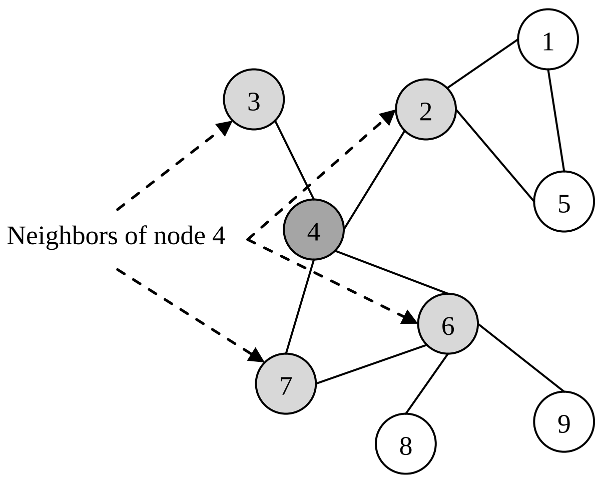
where $N(v)$ is the set of neighbors of vertex v, $x_{N(v)}$ and $l_{N(v)}$ are the sets containing the states and the labels of the vertices in $N(v)$ respectively, and $l_{(v,N(v))}$ is the set of the edge labels between v and its neighbors (Fig. 2).

Once each node has a vectorial representation, it can also be assigned an output y_v, evaluated by another parametric function g_t, called *output function*

$$y_v = g_t(x_v, l_v), \quad v \in V. \qquad (2)$$

Eqns. (1) and (2) define a method to produce an output $y_v = \varphi(g, l_v)$ for each vertex of the graph g.

Moreover, the symbolic and subsymbolic information associated with the vertices is indeed automatically encoded into a vector by the state transition function. We can show the computation graphically substituting all of the vertices with "units" that compute the function f_t. The "units" are connected according to graph topology. The resultant network is called the *encoding network* and will be the same topology as the input graph. Since the same parametric functions are applied to all the vertices, the units of the same type share the same set of parameters.



$$x_4 = f_t(l_4, x_2, x_3, x_6, x_7, l_2, l_3, l_6, l_7, l_{(4,2)}, l_{(4,3)}, l_{(4,6)}, l_{(4,7)})$$

Figure 2. The state x_v depends on the information in its neighbors

Let F_t and G_t be the vectorial functions obtained by stacking all the instances of f_t and g_t, respectively. Then Eqns. (1) and (2) can be rewritten as

$$x = F_t(x, l). \qquad (3a)$$

and

$$y = G_t(x, l). \qquad (3b)$$

where l represents the vector containing all the labels and x collects all the states. Eq. (3a) defines the global state x, while Eq. (3b) computes the output. It is relevant to mention that Eq. (3a) is recursive with respect to the state x, thus x is well defined only if Eq. (3a) has a unique solution. In conclusions, the viability of the method depends on the particular implementation of the transition function f_t.

In our supervised framework, for a subset $S \subseteq V$ of vertices, called supervised vertices, a target value t_v is defined for each $v \in S$. Thus an error signal (usual sum of squared error) can be specified as,

$$e_t = \sum_{v \in S}\left(t_v - \varphi(g, v)\right)^2. \qquad (4)$$

This signal drives an error backpropagation procedure that adapts the parameters of f_t and g_t so that the function realized by the network can approximate the targets, i.e.$\varphi(g, v) \approx t_v \in S$.

In practice, Eq. (1) is well suited to process positional graphs, since each neighbor position can be associated to a specific input argument of function f_t. In non–positional graphs, this scheme introduces an unnecessary constraint, since neighbors should be artificially ordered. A reasonable solution consists in calculating the state x_v as a sum of "contributions", one for each of its neighbors. Thus, state transition function can be rewritten as

$$x_v = \sum_{i=1}^{|N(v)|} h_t\left(l_v, x_{N_i(v)}, l_{N_i(v)}, l_{(v,N_i(v))}\right), \quad v \in V \qquad (5)$$

where $N_i(v)$ is the i-th neighbor and $|N(v)|$ is the number of neighbors of v. Several possible implementations of the functions f_t (or h_t) and g_t can be selected, e.g., Recursive Neural Networks(RNNs), Graph Neural Networks(GNNs), and recently developed NN4G [16]. RNNs, GNNs and NN4Gs differ in the implementation of the state transition function f_t and in the class of graphs that can be processed.

4. Relationship oriented framework for Graph Processing

In the previous framework for graph processing, vertices are given more importance, i.e., state value of a vertex is computed based on the information associated with it and its neighbors. In our new proposed framework, state value is computed for each edge (instead of each vertex) on the information associated with it and its neighbors.

To define edge based framework for graph processing we must implement the function $\psi: g \times e \rightarrow \mathbb{R}^s$ instead of $\varphi: g \times v \rightarrow \mathbb{R}^s$ to compute an output $\psi(g, e)$ for each pair (g, e). The principle idea behind this is to obtain a flat description of the information associated to each edge e instead of each vertex v. The description of an edge can be represented by a *state* value denoted by $x_e \in \mathbb{R}^s$, where the *state dimension* s is a predefined parameter. To obtain a distributed and parallel processing framework, the *states* are computed locally at each edge. The state value x_e can be designed as the output of a parametric state transition function f_r (instead of f_t), which depends on the edge label l_e and on the labels of vertices adjacent to the edge and its neighbors

$$x_e = f_r\left(l_e, x_{N(e)}, l_{N(e)}, l_{adj(e)}\right), \quad e \epsilon E \qquad (6)$$

where $N(e)$ is the set of neighbors of edge e, $x_{N(e)}$ and $l_{N(e)}$ are the sets containing the states and the labels of the edges in $N(e)$ respectively, and $l_{adj(e)}$ is the set containing the labels of adjacent vertices of the edge e (Fig. 3).

Again each edge has a vectorial representation, it can also be assigned an output y_e, evaluated by another parametric function g_r (instead of g_t), called *output function*

$$y_e = g_r(x_e, l_e), \quad e \epsilon E. \qquad (7)$$

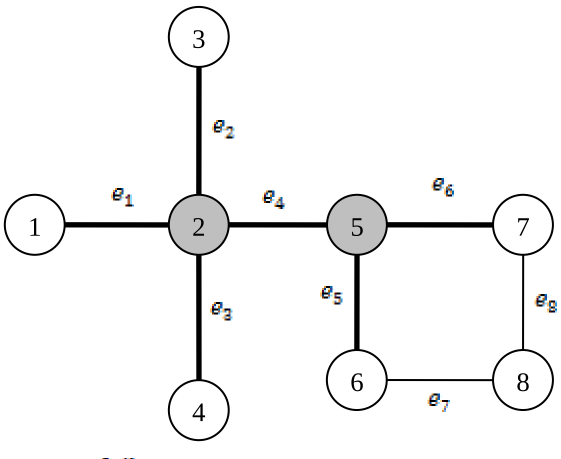
Eqns. (6) and (7) define a method to produce an output $y_e = \psi(g, l_e)$ for each edge of the graph g.

The symbolic and sub-symbolic information associated with the edges are automatically encoded into a vector by the state transition function f_r. The encoding network will be the same topology as the input graph. The units of the same type share the same set of parameters, since the same parametric functions are applied to all the edges.

Similar to vertex based framework, we can define the functions F_r and G_r by stacking all the instances of f_r and g_r, respectively and Eqns. (6) and (7) can be rewritten as

$$x = F_r(x, l) \qquad (8a)$$
$$y = G_r(x, l) \qquad (8b)$$

where l represents the vector containing all the labels and x collects all the states. Eq. (8a) defines the global state x, while Eq. (8b) computes the output. Note that Eq. (8a) is recursive with respect to the state x, thus x is well defined only if Eq. (8a) has a unique solution. The method differs from one implementation of the transition function (f_r) to another.



$$x_{(2,5)} = f_r\left(l_{(2,5)}, x_{(1,2)}, x_{(2,3)}, x_{(2,4)}, x_{(5,6)}, x_{(5,7)}, l_{(1,2)}, l_{(2,3)}, l_{(2,4)}, l_{(5,6)}, l_{(5,7)}, l_2, l_5\right)$$

where $e_1 = (1,2), e_2 = (2,3), \ldots$

Figure 3. The state $x_{(2,5)}$ depends on the information in its neighbors

In the supervised framework, for a subset $S \subseteq E$ of edges, called supervised edges, a target value t_e is defined for each $e \epsilon S$. Thus an error signal can be specified as,

$$\varepsilon_r = \sum_{e \epsilon S}\left(t_e - \psi(g, e)\right)^2. \qquad (9)$$

This signal drives an error backpropagation procedure that adapts the parameters of f_r and g_r so that the function realized by the network can approximate the targets, i.e., $(g, e) \approx t_e \epsilon S$.

A reasonable solution (like previous framework) consists in calculating the state x_e as a sum of "contributions", one for each of its neighbors. Thus, state transition function can be rewritten as

$$x_e = \sum_{i=1}^{|N(e)|} h_r\left(l_e, x_{N_i(e)}, l_{N_i(e)}, l_{adj(e)}\right), \quad e \epsilon E \qquad (10)$$

where$N_i(e)$ is the i-th neighbor and $|N(e)|$ is the number of neighbors of e. New neural network models like RNNs, GNNs can also be developed to implement the functions f_r (or h_r) and g_r.

5. Learning in Geometrical Structured Domains

There are certain application domains where the "true nature" of the data not only depends on the hierarchical relationship of the data but also on the geometrical structure/topology of the data which naturally presents itself in a geometrical structured form and some contextual information is associated with the geometrical structure of the data itself. Quantitative structure-property/activity relationships (QSPR/QSAR) are fundamental aspects in chem.-informatics, where the aim is to correlate chemical structure of molecules with their properties (or biological activity) in order to achieve prediction. Since each molecule is a 3-D geometrical structure, the 3-D geometrical structure itself should have some impact on QSPR/QSAR. If we can develop a model that can learn geometrical topology, then the accuracy of QSPR/QSAR analysis of chemical compounds may be improved more. We can find other applications also where learning in geometrical structured domains (GSD) can be incorporated to get better performance.

In Eq. (1), the label l_v of vertex v represents the information associated with the corresponding indivisible component object v of structured data. In fact, l_v is a fixed size vector of real numbers where each element of l_v represents a property or feature of the component object. Each component object itself is an unstructured (indivisible) data whereas these component objects are interconnected among them to form a structured data as a whole.

In both the frameworks of graph processing, geometrical information of structured data is not incorporated to compute the state values of vertices, which is essential to capture the true nature of geometrical structured data; in fact, only hierarchical nature of structured data is captured. We can incorporate relative coordinate information (if available) of each component object of structured data to calculate the state value of the corresponding vertex. Hence, Eq. (1) could be changed as

$$x_v = f_t\left(l_v, c_v, x_{N(v)}, l_{N(v)}, l_{(v,N(v))}\right), \quad v\epsilon V \qquad (11)$$

and Eq. (6) could be rewritten as

$$x_e = f_r\left(l_e, c_u, c_v, x_{N(e)}, l_{N(e)}, l_{adj(e)}\right), \quad e\epsilon E \qquad (12)$$

where $e = (u, v)$and c_u, c_v are the vectors of coordinate of vertices u and v respectively. Based on dimension or size of vector c_u and/or c_v we can generalize the model into n- dimensional geometrical structured data. If input domain of data is 2-D geometrical structure, the size of $c_u$and/or c_v will be 2. Similarly, the size of $c_u$and/or c_v will be 3 when the data of input domain is 3-D geometrical structure.

6. Future Directions and Conclusion

Though the new framework proposed in this paper, can capture the true nature of structured data, the success of the framework still lie on the proper implementation of the framework. Hence further research could be done on the development of new artificial network models.

Again the development of model for learning in geometrical structured domains (GSD) needs to investigate on the integration of symbolic and sub-symbolic approaches. The integration of symbolic and sub-symbolic approaches is a fundamental research topic for the development of intelligent and efficient systems capable of dealing with tasks whose nature is neither purely symbolic nor sub-symbolic. It is common opinion in the scientific community that a extensive variety of real-world problems require hybrid solutions, i.e., solutions combining techniques based on neural networks, fuzzy logic, genetic algorithms, probabilistic networks, expert systems, and other symbolic techniques. A very popular view of hybrid systems is one in which numerical data are processed by a sub-symbolic module, while structured data are processed by the symbolic counterpart of the system. Unfortunately, because of the different nature of numerical and structured representations, a tight integration of the different components seems to be very difficult.

Learning in structured domains can be used in various fields of applications such as natural language processing, image processing, speech processing, computer vision, chem.-informatics, bioinformatics, etc. Hence, a simple solution for general class of graphs is anticipated. The NN4G model is a relatively simple solution for dealing with fairly general classes of graphs by sub-symbolic approaches; similar solution for geometrical structures is also expected. We hope that the introduction of simple and general approaches (e.g., NN4G) in structured domains will be attracted to ML researchers for widespread applications.

References

[1] B. Hammer and J. Jain, “Neural methods for non-standard data,” in Proceedings of the 12th European Symposium on Artificial Neural Networks, M.Verleysen, Ed., 2004, pp. 281–292.

[2] R. Kondor and J. Lafferty, “Diffusion kernels on graphs and other discrete structures,” in Proc. 19th International Conference on Machine Learning (ICML2002), C. Sammut and A. e. Hoffmann, Eds. Morgan Kaufmann Publishers Inc, 2002, pp. 315–322.

[3] T. G¨artner, “Kernel-based learning in multi-relational data mining,” ACM SIGKDD Explorations, vol. 5, no. 1, pp. 49–58, 2003.

[4] P. Mah´e, N. Ueda, T. Akutsu, P. J.-L., and J.-P.Vert, “Extensions of marginalized graph kernels,” in Proc. 21th International Conference on Machine Learning (ICML2004). ACM Press, 2004, p. 70.

[5] J. Suzuki, H. Isozaki, and E. Maeda, “Convolution kernels with feature selection for natural language processing tasks.” in ACL, 2004, pp. 119–126.

[6] A.Sperduti and A. Starita, “Supervised neural networks for the classification of structures,” IEEE Transactions on Neural Networks, vol. 8, pp. 429–459, 1997.

[7] P. Frasconi, M. Gori, and A. Sperduti, “A general framework for adaptive processing of data structures,” IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 768–786, 1998.

[8] M. Bianchini, M. Gori, and F. Scarselli, “Processing directed acyclic graphs with recursive neural networks,” IEEE Trans. Neural Netw., vol. 12, no. 6, pp. 1464–1470, Nov. 2001

[9] A.Micheli, D. Sona, and A. Sperduti, “Contextual processing of structured data by recursive cascade correlation,” IEEE Trans. Neural Netw., vol. 15, no. 6, pp. 1396–1410, Nov. 2004.

[10] M. Gori, G. Monfardini, and F. Scarselli, “A new model for learning in graph domains,” in Proc. International Joint Conference on Neural Networks (IJCNN2005), 2005, pp. 729–734.

[11] F. Scarselli, S. Yong, M. Gori, M. Hagenbuchner, A. Tsoi, and M. Maggini, “Graph neural networks for ranking web pages,” in Proc. of the 2005 IEEE/WIC/ACM Conference on Web Intelligence (WI2005), 2005, pp. 666–672.

[12] B. Hammer, A. Micheli, and A. Sperduti, “Universal approximation capability of cascade correlation for structures,” Neural Comput., vol. 17, no. 5, pp. 1109–1159, 2005.

[13] M. Bianchini,M. Gori, L. Sarti, and F. Scarselli, “Recursive processing of cyclic graphs,” IEEE Trans. Neural Netw., vol. 17, no. 1, pp. 10–18, Jan. 2006.

[14] M. Hagenbuchner, A. Sperduti, and A. C. Tsoi, “A self-organizing map for adaptive processing of structured data,” IEEE Transactions on Neural Networks, vol. 14, no. 3, pp. 491–505, May 2003.

[15] M. Bianchini, M. Maggini, L. Sarti, and F. Scarselli, “Recursive neural networks for processing graphs with labelled edges: Theory and applications,” Neural Networks - Special Issue on Neural Networksand Kernel Methods for Structured Domains, vol. 18, pp. 1040–1050, October 2005.

[16] AlessioMicheli, “Neural Network for Graphs: A Contextual Constructive Approach,” IEEE Transactions on Neural Networks, vol. 20, no. 3, march 2009.

[17] S. Fahlman and C. Lebiere, “The cascade-correlation learning architecture,” Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-90-100, Aug. 1990.

[18] A. K¨uchler and C. Goller, “Inductive learning in symbolic domains using structure–driven recurrent neural networks,” in Advances in Artificial Intelligence, G. G¨orz and S. H¨olldobler, Eds. Berlin: Springer-Verlag, 1996, pp. 183–197.

[19] V. Di Massa, G. Monfardini, L. Sarti, F. Scarselli, M. Maggini, and M. Gori, “A comparison between recursive neural networks and graph neural networks,” in Proc. World Congr. Comput. Intell./Int. Joint Conf. Neural Netw., 2006, pp. 778–785.

[20] Ethem Alpaydin, “Introduction to Machine Learning,” Prentic-Hall of India, 2005.

[21] Martin T. Hagan, Howard B. Demuth, Mark beale, “Neural Network Design,” Cengage Learning India, 1996.

[22] Simon Haykin, “Neural Networks, A Comprehensive Foundation,” 2nd ed. Pearson Education, 2006.

[23] Mohamad H. Hassoun, “Fundamentals of Artificial Neural Networks,” Prentic-Hall of India, 2008.

[24] Duance Hanselman, Bruce Littlefield, “MASTERING MATLAB 7,” Pearson Education, 2005.

[25] Jue Wang and Qing Tao, “Machine Learning: The State of the Art,” IEEE Intelligent Systems, 2008.

Author Biographies

MADHUSUDAN PAUL He is a final year student of M.Tech. (CSE) in the Department of Computer Science, School of Engineering and Technology, Pondicherry University, Puducherry, India. He received his M.Sc. (CS) in 2008 from Visva-Bharati University, Santiniketan, West Bengal, India and B.Sc. (CS) in 2006 from University of Calcutta, Kolkata, West Bengal, India. His research interests include Machine Learning, Soft Computing, and Automata Theory.

THAMIZH SELVAM. D He received his B.Sc., Computer Science and M.Sc., Computer Sciences from Pondicherry University, Puducherry, India in 2000 and 2003 respectively. He received his M.Phil., in Computer Science from Periyar University, Tamilnadu, India in 2008. Currently, he is a pursuing his Ph.d., in Computer Science and Engineering from Department of Computer Science, School of Engineering, Pondicherry University, Puducherry. His field of research includes Distributed Algorithms, Peer-to-Peer Networks, and Overlay Network Structures.

P. SYAM KUMAR He received his B.Tech., (CSE) from JNTU, Hyderabad, India and M.Tech., (CST) from Andhra University, Visakhapatnam, India in 2003 and 2006 respectively. His research interests include Distributed Computing and Cloud Computing.

Dr. R. SUBRAMANIAN He is currently the Professor and Head of Department of Department of Computer Science, School of Engineering, Pondicherry University, Puducherry, India. He received his B.Sc., Mathematics from Madurai Kamaraj University, Madurai, India in 1982. He received his M.Sc., Mathematics and Ph.D., in Computer Science and Engineering from IIT Delhi in 1984 and 1989. He has in his credit around 20 research publications in peer-scholarly research publications in both National and Internal Journals and Conferences. His research interests include Parallel and Distributed Algorithm, Evolutionary Algorithms and Robotics