Extended Cascade-Correlation for Syntactic and Structural Pattern Recognition

Alessandro Sperduti and Darya Majidi and Antonina Starita

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 PISA, Italy

Abstract. Automatic inference is one of the main problems that syntactic and structural pattern recognition must solve for successful applications. Neural networks are artificial intelligence tools which already support automatic inference for successful applications of statistical pattern recognition. In this paper, we suggest that neural networks, and specifically *Cascade-Correlation*, can be used for automatic inference in syntactic and structural pattern recognition, as well. An extended version of a standard neuron which is able to deal with structures is presented and the Cascade-Correlation algorithm generalized to structured domains. The computational complexity of the proposed algorithm as well as experimental results obtained on problems involving logic terms are presented.

1 Introduction

Syntactic and structural pattern recognition [7, 4, 10] are based on the premise that the *structure* of an entity is very important, both for classification and description. The necessity of such point of view became clear when researchers in traditional pattern recognition realized that feature derivation is a very complex process that can hardly be automated and that features can be both numerical and symbolic. Despite the common point of view, syntactic and structural pattern recognition have a precise characterization: syntactic pattern recognition uses tools from the theory of formal languages, while structural pattern recognition is mainly based on graph matching. Unfortunately, both of them have to solve the difficulty of *automatic* inference. This difficulty has opened a new stream of research which attempt to combine *artificial intelligence* [9] with syntactic or structural pattern recognition. In statistical pattern recognition this combination has already been realized through *artificial neural networks* [5], leading to very successful applications.

The aim of this paper is to suggest that *artificial neural networks* can be applied as well to syntactic and structural pattern recognition. Specifically, we discuss the extension of a very popular neural network technique, i.e., *Cascade-Correlation* [2, 1], which allows the treatment of structured domains whose elements can be represented as *labeled acyclic directed graphs*. Both classification and approximation of continuos functions on these domains as well as induction of structural grammars, such as expansive tree grammars [4], can be performed. In this paper, we focus on classification tasks.

In Section 2, we introduce structured domains and recall notions about graphs and neural networks. An extended version of a standard neuron which is able to deal with structures is presented in Section 3. Using this extended neuron, we show in Section 4 how Cascade-Correlation can be generalized to structures, and the computational complexity of the resulting algorithm is discussed in Section 5. Section 6 introduces some problems involving logic terms and experimental results on these problems are reported. Finally, Section 7 contains the conclusions.

2 Structured Domains

In this paper, we deal with structured patterns which can be represented as labeled graphs. Some examples of these kind of patterns are shown in Figure 1, where we have reported some typical examples from the structured domain of logic terms. All these structures can also be represented by using a feature-based approach. Feature-based approaches, however, have the drawbacks of being very sensitive to the features selected for the representation and unable to represent any specific information about the relationships among components. Standard neural networks, as well as statistical methods, are usually bound to this kind of representation and thus considered not adequate for dealing with structured domains. On the contrary, we will show that neural networks can represent and classify structured patterns. Some advances on this issue have been recently presented in [13], where preliminary work on the classification of logic terms, represented as labeled graphs, through an extension of the LRAAM model [11, 12] was reported. The present paper represents a further improvement in this field.

2.1 Preliminaries

We consider finite directed node labeled graphs without multiple edges. For a set of labels Σ, a *graph* X (over Σ) is specified by a finite set V_X of *nodes*, a set E_X of ordered couples of $V_X \times V_X$ (called the set of *edges*) and a function ϕ_X from V_X to Σ (called the *labeling function*). Note that labels are not

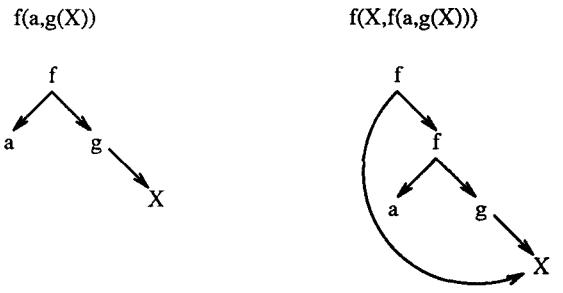


Fig. 1. Example of logic terms represented as labeled directed acyclic graphs.

restricted to be binary. Specifically, labels may also be real-valued vectors. A graph X' (over Σ) is a *subgraph of* X if $\phi_{X'} = \phi_X$, $V_{X'} \subseteq V_X$, and $E_{X'} \subseteq E_X$. For a finite set V, $\#V$ denotes its cardinality. Given a graph X and any node $x \in V_X$, the function $out\_degree_X(x)$ returns the number of edges leaving from x, i.e., $out\_degree_X(x) = \#\{(x,z) \mid (x,z) \in E_X \ \wedge \ z \in V_X\}$. Given a total order on the edges leaving from x, the node $y = out_X(x,j)$ in V_X is the node pointed by the *jth* pointer leaving from x. The *valence* of a graph X is defined as $\max_{x \in V_X}\{out\_degree_X(x)\}$. A *labeled directed acyclic graph (labeled DAG)* is a graph, as defined above, without loops. A node $s \in V_X$ is called a *supersource* for X if every node in X can be reached by a path starting from s. The root of a tree (which is a special case of directed graph) is always the (unique) *supersource* of the tree. In the following, we deal with DAGs.

We define a *structured domain* $\mathcal{D}$ (over Σ) as any (possibly infinite) set of graphs (over Σ). The *valence of a domain* $\mathcal{D}$ is defined as the maximum among the valences of the graphs belonging to $\mathcal{D}$. Since we are dealing with learning, we need to define the target function we want to learn. In approximation tasks, a *target function* $\xi()$ over $\mathcal{D}$ is defined as any function $\xi : \mathcal{D} \rightarrow \Re^k$, were k is the output dimension, while in (binary) classification tasks we have $\xi : \mathcal{D} \rightarrow \{0,1\}$ (or $\xi : \mathcal{D} \rightarrow \{-1,1\}$.) A training set T over a domain $\mathcal{D}$ is defined as a set of couples $(X, \xi(X))$, where $X \in \mathcal{U} \subseteq \mathcal{D}$ and $\xi()$ is a target function defined over $\mathcal{D}$.

The output $o^{(s)}$ of a standard neuron is given by

$$o^{(s)} = f(\sum_i w_i I_i), \tag{1}$$

where $f()$ is some non-linear squashing function applied to the weighted sum of inputs I[1]. A recurrent neuron with a single self-recurrent connection, instead, computes its output $o^{(r)}(t)$ as follows

$$o^{(r)}(t) = f(\sum_i w_i I_i(t) + w_s o^{(r)}(t-1)), \tag{2}$$

where $f()$ is applied to the weighted sum of inputs I plus the self-weight, w_s, times the previous output. The above formula can be extended both considering several interconnected recurrent neurons and delayed versions of the outputs. For the sake of presentation, we skip these extensions.

3 The First Order Generalized Recursive Neuron

Using standard and recurrent neurons, it is very difficult to deal with complex structures. This is because the former was devised for processing unstructured patterns, while the latter can only naturally process sequences. Thus, neural

[1] The threshold of the neuron is included in the weight vector by expanding the input vector with a component always to 1.

networks using these kind of neurons can face approximation and classification problems in structured domains only using a very unnatural encoding scheme which maps structures onto fixed-size unstructured patterns or sequences. We propose to solve this inadequacy of the present state of the art in neural networks by introducing the *generalized recursive neuron*. The generalized recursive neuron is an extension of the recurrent neuron where instead of just considering the output of the unit on the previous time step, we have to consider the outputs of the unit for all the nodes which are pointed by the current input node. Then the output $o^{(c)}(x)$ of the generalized recursive neuron to a node x of a graph X is defined as

$$o^{(c)}(x) = f(\sum_{i=1}^{N_L} w_i l_i + \sum_{j=1}^{out\_degree_X(x)} \hat{w}_j o^{(c)}(out_X(x,j))), \tag{3}$$

where N_L is the number of units encoding the label $\boldsymbol{l} = \phi_X(x)$ attached to the current input x, and $\hat{w}_j$ are the weights on the recursive connections. Note that, if the valence of the considered domain is n, then the generalized recursive neuron will have n recursive connections, even if not all of them will be used for computing the output of a node x with $out\_degree_X(x) < n$.

When considering k interconnected generalized recursive neurons, eq. (3) becomes

$$\boldsymbol{o}^{(c)}(x) = \boldsymbol{F}(\boldsymbol{W}\boldsymbol{l} + \sum_{j=1}^{out\_degree_X(x)} \widehat{\boldsymbol{W}}_j \boldsymbol{o}^{(c)}(out_X(x,j))), \tag{4}$$

where $\boldsymbol{F}_i(\boldsymbol{v}) = f(v_i)$, $\boldsymbol{l} \in \Re^{N_L}$, $\boldsymbol{W} \in \Re^{k \times N_L}$, $\boldsymbol{o}^{(c)}(x)$, $\boldsymbol{o}^{(c)}(out_X(x,j)) \in \Re^k$, $\widehat{\boldsymbol{W}}_j \in \Re^{k \times k}$.

In the following, we will refer to the output of a generalized neuron dropping the upper index.

3.1 Generation of Neural Representations for DAGs

To understand how generalized recursive neurons can generate representations for DAGs, let us consider a single generalized recursive neuron u and a single DAG X. The following conditions must hold:

Number of Connections: the generalized recursive neuron u must have as many recursive connections as the valence of the graph X;
Supersource: the graph X must have a reference *supersource*.

Note that, if the graph X does not have a supersource, then it is always possible to define a convention for adding to the graph X an extra node s (with a minimal number of outgoing edges) such that s is a supersource for the new graph;

If the above conditions are satisfied, we can adopt the convention that the graph X is represented by $o(s)$, i.e., the output of u to s. Consequently, due to the recursive nature of eq. (3), it follows that the neural representation for a DAG is computed by a feedforward network (*encoding network*) obtained by

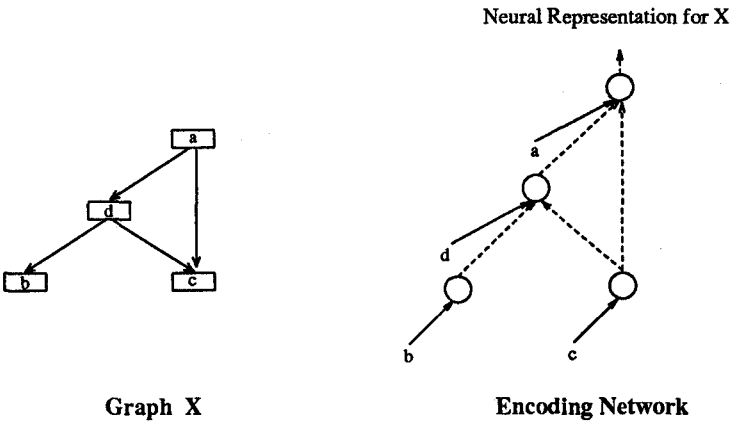


Fig. 2. A labeled graph X and the associated encoding network.

replicating the same generalized recursive neuron u and connecting these copies according to the topology of the structure (see Figure 2). The encoding network fully describes how the representation for the structure is computed.

4 Structural Cascade-Correlation

The Cascade-Correlation algorithm [2] grows a standard neural network using an incremental approach for classification of unstructured patterns. The starting network $\mathcal{N}_0$ is a network without hidden nodes trained with a Least Mean Square algorithm; if network $\mathcal{N}_0$ is not able to solve the problem, a hidden unit u_1 is added such that the *correlation* between the output of the unit and the residual error of network $\mathcal{N}_0$ is maximised[2]. The weights of u_1 are frozen and the remaining weights are retrained. If the obtained network $\mathcal{N}_1$ cannot solve the problem, the network is further grown, adding new hidden units which are connected (with frozen weights) with all the inputs and previously installed hidden units. The resulting network is a *cascade* of nodes. Fahlman extended the algorithm to classification of sequences, obtaining good results [1].

In this paper, we show that Cascade-Correlation can further be extended to DAGs by using generalized recursive neurons. The output of the kth hidden unit, in our framework, can be computed as

$$o^{(k)}(x) = f(\sum_{i=1}^{N_L} w_i^{(k)} l_i + \sum_{v=1}^{k} \sum_{j=1}^{out\_degree_X(x)} \hat{w}_{(v,j)}^{(k)} o^{(v)}(out_X(x,j)) + \sum_{q=1}^{k-1} \bar{w}_q^{(k)} o^{(q)}(x)), \tag{5}$$

[2] Since the maximization of the correlation is obtained using a gradient ascent technique on a surface with several maxima, a pool of hidden units is trained and the best one selected.

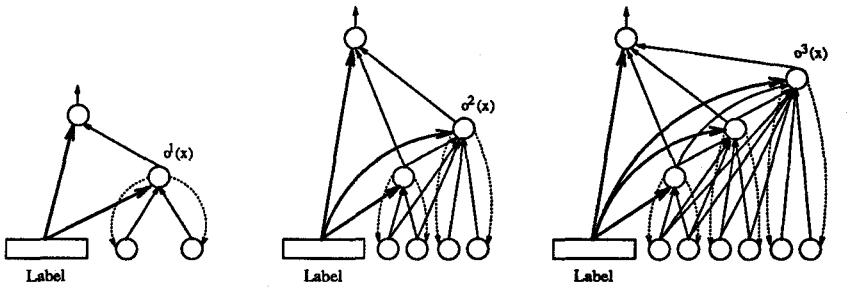


Fig. 3. The evolution of a network with two pointer fields. The units in the label are fully connected with the hidden units and the output unit.

where $w^{(k)}_{(v,j)}$ is the weight of the kth hidden unit associated to the output of the vth hidden unit computed on the jth component pointed by x, and $\bar{w}^{(k)}_q$ is the weight of the connection from the qth (frozen) hidden unit, $q < k$, and the kth hidden unit. The output of the output neuron $u^{(out)}$ is computed as

$$o^{(out)}(x) = f(\sum_{i=1}^{k} \tilde{w}_i o^{(i)}(x)), \tag{6}$$

where $\tilde{w}_i$ is the weight on the connection from the ith (frozen) hidden unit to the output unit.

Learning is performed as in standard Cascade-Correlation, with the difference that, according to equation (5), the derivatives of $o^{(k)}(x)$ with respect to the weights are now:

$$i = 1, \ldots, N_L : \frac{\partial o^{(k)}(x)}{\partial w^{(k)}_i} = (l_i + \sum_{j=1}^{\omega} \hat{w}^{(k)}_{(k,j)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial w^{(k)}_i}) f' \tag{7}$$

$$q = 1, \ldots, (k-1) : \frac{\partial o^{(k)}(x)}{\partial \bar{w}^{(k)}_q} = (o^{(q)}(x) + \sum_{j=1}^{\omega} \hat{w}^{(k)}_{(k,j)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial \bar{w}^{(k)}_q}) f' \tag{8}$$

$$\begin{matrix} v = 1,\ldots,k \\ t = 1,\ldots,\omega \end{matrix} : \frac{\partial o^{(k)}(x)}{\partial \hat{w}^{(k)}_{(v,t)}} = (o^{(v)}(out_X(x,t)) + \sum_{j=1}^{\omega} \hat{w}^{(k)}_{(v,j)} \frac{\partial o^{(k)}(out_X(x,j))}{\partial \hat{w}^{(k)}_{(v,t)}}) f' \tag{9}$$

where $\omega = out\_degree_X(x)$, f' is the derivative of $f()$. The above equations are recurrent on the structures and can be computed by observing that for a leaf node y equation (7) reduces to $\frac{\partial o^{(k)}(y)}{\partial w^{(k)}_i} = l_i$, and all the remaining derivatives are null. Consequently, we only need to store the output values of the unit and its derivatives for each component of a structure. Figure 3 shows the evolution of a network with two pointer fields. Note that, if the hidden units have self-recurrent connections only, the matrix defined by the weights $\hat{w}^{(k)}_{(v,j)}$ is not triangular, but diagonal.

5 Computational Complexity

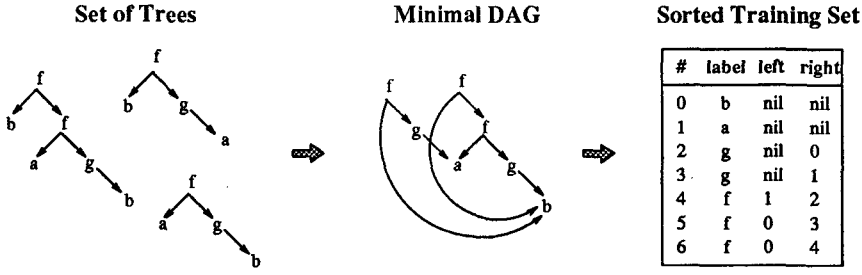
There are two different aspects which must be considered: the optimization of the training set and the complexity of the learning algorithm.

Regarding the optimization of the training set, a careful encoding of the DAGs in the training set allows an efficient implementation of the proposed algorithm (see Fig. 4). In fact, the training set T can be optimized, i.e., if there are graphs $X_1, X_2 \in T$ which share a common subgraph $\hat{X}$, then we need to explicitly represent $\hat{X}$ only once. The optimization of the training set can be performed in two stages: *(i)* all the DAGs in the training set are merged into a single minimal DAG, i.e., a DAG with minimal number of nodes; *(ii)* a topological sort on the nodes of the minimal DAG is performed to determine the updating order on the nodes for the network. Both stages can be done in linear time with respect to the size of all DAGs and the size of the minimal DAG, respectively. Specifically, stage *(i)* can be done by removing all the duplicates subgraphs through a special subgraph-indexing mechanism (which can be implemented in linear time).

Concerning learning, the derivatives for the kth hidden unit with respect to a single pattern can be computed in $O(kN_V^2)$ in time, since $out\_degree_X(x) \leq N_V$. Thus, considering the full training set it takes $O(PkN_V^2)$ in time. Finally, when building a network with k hidden units, the computation of all the derivatives takes $O(Pk^2N_V^2)$ in time. The complexity in space is dominated by the space necessary for storing the derivatives of the current trained unit, i.e., $O(PkN_V)$. When considering a diagonal connection matrix, i.e., hidden units with self-recursive connections only, the complexity of learning becames $O(PkN_V^2)$ in time and $O(PN_V)$ in space.

6 Description of the Classification Problems and Results

To test the ability of the proposed architecture to deal with several classification tasks involving terms, we have generated a set of classification problems. We have



#	label	left	right
0	b	nil	nil
1	a	nil	nil
2	g	nil	0
3	g	nil	1
4	f	1	2
5	f	0	3
6	f	0	4

Fig. 4. Optimization of the training set: the set of structures (in this case, trees) is transformed into the corresponding minimal DAG, which is then used to generate the sorted training set. The sorted training set is then transformed into a set of sorted vectors using the numeric codes for the labels and used as training set for the network.

Classification Problems

Problem	Symbols	Positive Examples.	#terms (tr.,test)	#subterms (tr.,test)	depth (pos.,neg.)
termocc1 very long	f/2 i/1 a/0 b/0 c/0	the (sub)terms i(a) or f(b,c) occur somewhere	(280,120)	(559,291)	(6,6)
inst1	f/2 a/0 b/0 c/0	instances of f(X,X)	(200,83)	(235,118)	(3,2)
inst1 long	f/2 a/0 b/0 c/0	instances of f(X,X)	(202,98)	(403,204)	(6,6)
inst4 long	f/2 a/0 b/0 c/0	instances of f(X,f(a,Y))	(290,110)	(499,245)	(7,6)

Table 1. Description of a set of classification problems involving logic terms.

summarized the characteristics of each problem in Table 1. The first column of the table reports the name of the problem, the second one the set of symbols (with associated arity) compounding the terms, the third column shows the rule(s) used to generate the positive examples of the problem[3], the fourth column reports the number of terms in the training and test set respectively, the fifth column the number of subterms in the training and test set, and the last column the maximum depth[4] of terms in the training and test set. For each problem about the same number of positive and negative examples is given. Both positive and negative examples have been generated randomly. Training and test sets are disjoint and have been generated by the same algorithm.

It must be noted that the set of proposed problems range from the detection of a particular atom (label) in a term to the satisfaction of a specific unification pattern. Specifically, in the unification patterns for the problems `inst1`, and `inst1_long` the variable X occurs twice making these problems much more difficult than `inst4_long`, because any classifier for these problems would have to compare arbitrary subterms corresponding to X.

The results shown in Table 2 are obtained for each problem using a pool of 8 units. The networks used have both triangular and diagonal recursive connections matrices and **no connection between hidden units**. We decided to remove the connections between hidden units to reduce the probability of overfitting.

We made no extended effort for optimizing the learning parameters and the number of units in the pool, thus it should be possible to significantly improve on the reported results. We are currently gathering other results to improve the statistics.

[3] Note that the terms are all ground.

[4] We define the depth of a term as the maximum number of edges between the root and leaf nodes in the term's LDAG-representation.

Results

(Networks without connections between hidden units)

	Problem	# Label Units	# Trials	# Hidden Units Mean (Min - Max)	% Test Mean (Min - Max)
Triangular	termocc1 very-long	8	4	13.25 (8 - 19)	97.70 (95 - 100)
	inst4 long	6	5	7.2 (4 - 12)	99.64 (99.09 - 100)
	inst1	6	9	11.33 (7 - 19)	90.09 (86.75 - 92.77)
	inst1 long	6	16	7.81 (6 - 11)	91.65 (88.87 - 94.89)
Diagonal	termocc1 very-long	8	3	11 (8 - 16)	96.94 (95 - 99.17)
	inst4 long	6	3	12.66 (9 - 15)	98.48 (97.27 - 100)
	inst1	6	5	12.4 (7 - 21)	90.6 (87.95 - 92.77)
	inst1 long	6	3	18.66 (17 - 21)	82.99 (75.51 - 91.84)

Table 2. Results obtained on the test sets for each classification problem using both networks with triangular and diagonal recursive connection matrices. The same number of units (8) in the pool was used for all networks.

7 Conclusion

We have proposed a generalization of the Cascade-Correlation architecture [2] for automatic inference in syntactic and structural pattern recognition. Specifically, in this paper, we demonstrated the possibility to perform classification of structures. The extended architecture and calculation of derivatives for the learning rules have been defined and its performances demonstrated on classification tasks involving logic terms. The obtained results improve the results obtained in [13].

It must be noted that automatic inference can also be obtained by using *Inductive Logic Programming* [6]. The proposed approach, however, has its own specific peculiarity, since it can approximate functions from a structured domain (possibly with real valued vectors as labels) to the reals. Specifically, we believe that the proposed approach can fruitfully be applied to molecular biology and chemistry (classification of chemical structures, quantitative structure-property relationship (QSPR), quantitative structure-activity relationship (QSAR)), where it can be used for the automatic determination of *topological indexes* [8], which are usually designed through a very expensive trial and error approach.

In conclusion, the proposed architecture extends the processing capabilities of neural networks, allowing the processing of structured patterns which can be of variable size and complexity. However, it must be pointed out that the proposed architecture has some computational limitations due to the fact that frozen hidden units cannot receive input from hidden units introduced after their insertion into the network. These limitations, in the context of standard Recurrent Cascade-Correlation (RCC), have been discussed in [3], where it is demonstrated that certain finite state automata cannot be implemented by networks built up by RCC using monotone activation functions. Since our algorithm reduces to standard RCC when considering sequences, it follows that it has limitations as well.

Acknowledgment We would like to thank Christoph Goller for the generation of the training and test sets used in this paper.

References

1. S. E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, Carnegie Mellon, 1991.
2. S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. San Mateo, CA: Morgan Kaufmann, 1990.
3. C.L. Giles, D. Chen, G.Z. Sun, H.H. Chen, Y.C. Lee, and M.W. Goudreau. Constructive learning of recurrent neural networks: Limitations of recurrent casade correlation and a simple solution. *IEEE Transactions on Neural Networks*, 6(4):829–836, 1995.
4. R. C. Gonzalez and M. G. Thomason. *Syntactical Pattern Recognition.* Addison-Wesley, 1978.
5. S. Haykin. *Neural Networks: a comprehensive Foundation.* IEEE Press, 1994.
6. S. Muggleton and L. De Raedt. Inductive login programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
7. T. Pavlidis. *Structural Pattern Recognition.* Springer-Verlag, 1977.
8. D. H. Rouvray. *Computational Chemical Graph Theory*, page 9. Nova Science Publishers: New York, 1990.
9. S. Russell and P. Norvig. *Artificial Intelligence: a comprehensive Foundation.* Prentice Hall, 1995.
10. R. J. Schalhoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley & Sons, 1992.
11. A. Sperduti. Labeling RAAM. *Connection Science*, 6(4):429–459, 1994.
12. A. Sperduti. Stability properties of labeling recursive auto-associative memory. *IEEE Transactions on Neural Networks*, 6(6):1452–1460, 1995.
13. A. Sperduti, A. Starita, and C. Goller. Learning distributed representations for the classification of terms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 509–515, 1995.