How to Process Structured Data by Neural Networks?

Vladimír Kvasnička

Department of Mathematics
Slovak Technical University
Bratislava, Slovakia

November 9-12, 1998
SCANN '98, Smolenice Castle, Slovakia

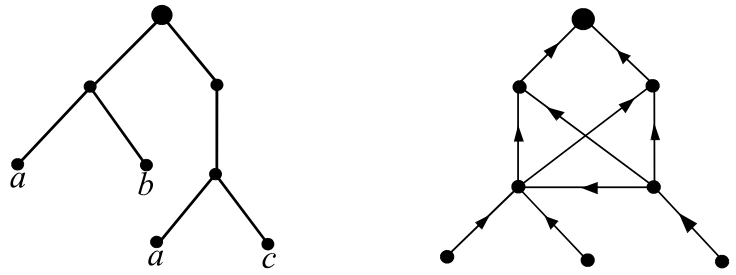
Contents

1. Structured and unstructured data

* **Unstructured data** are real vectors of **fixed dimension**,

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in R^n$$

* **Structured data** are
  - token strings of variable lengths, e.g. $\alpha \in \{a, b, c, \ldots\}^*$
  - acyclic oriented rooted graphs (with labeled vertices)



rooted tree oriented acyclic graph

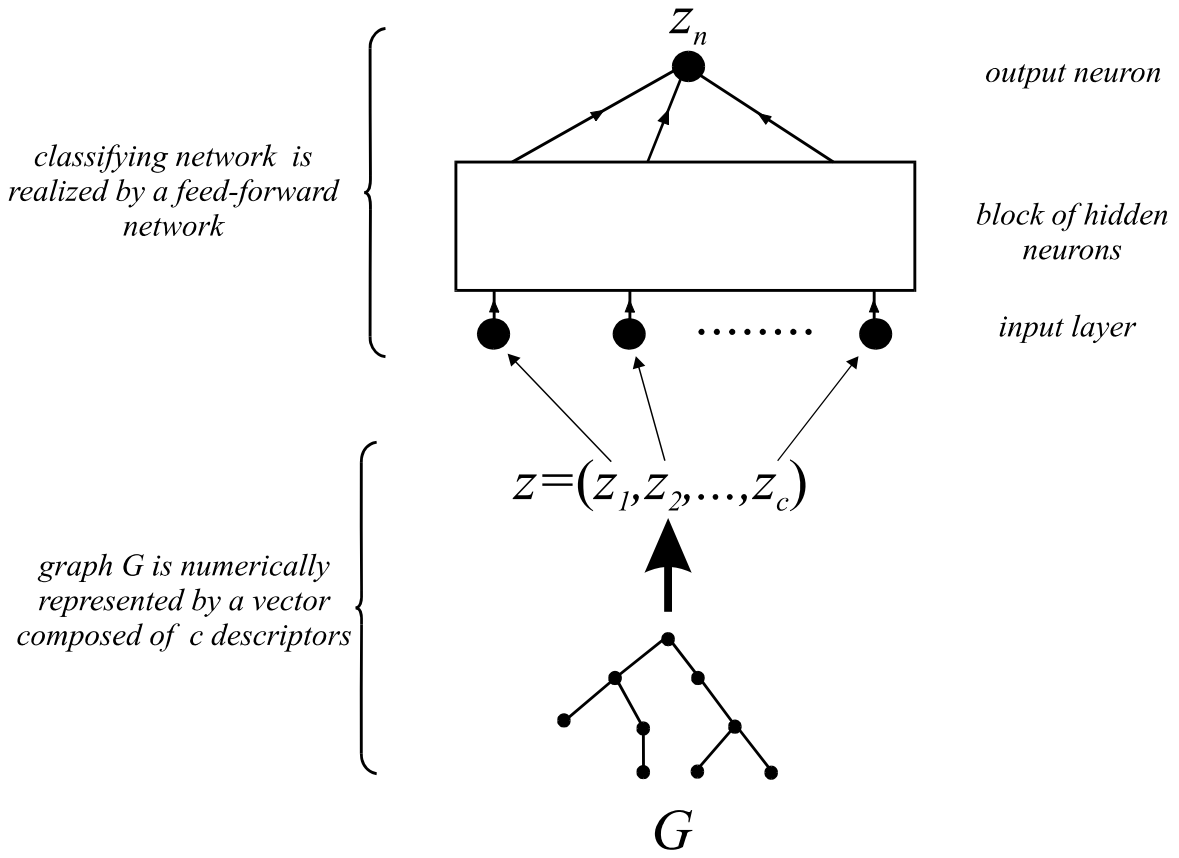
Where are the structured data important?

The ability to recognize and classify structured data - patterns is fundamental to many applications, such as

- medical or technical diagnosis
- molecular biology
- chemistry (structural formulae)
- automated reasoning
- linguistics (formal structure of sentences)
- software engineering (flow-chart diagrams)
- ……

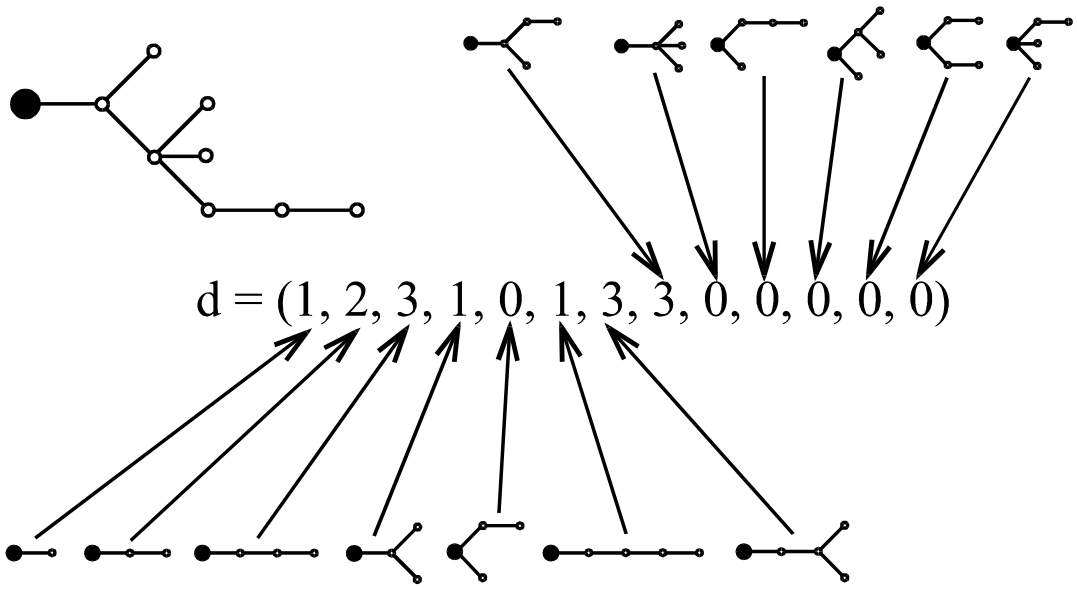
Standard approach to the classification of graphs by neural networks

The input graph G is numerically represented by a vector composed of c descriptors.



Illustrative example of coding of rooted trees

Any rooted tree is numerically represented by a vector composed of 13 entries, each vector entry corresponds to a number of appearance of a given rooted subtree (fragment) in the rooted tree.



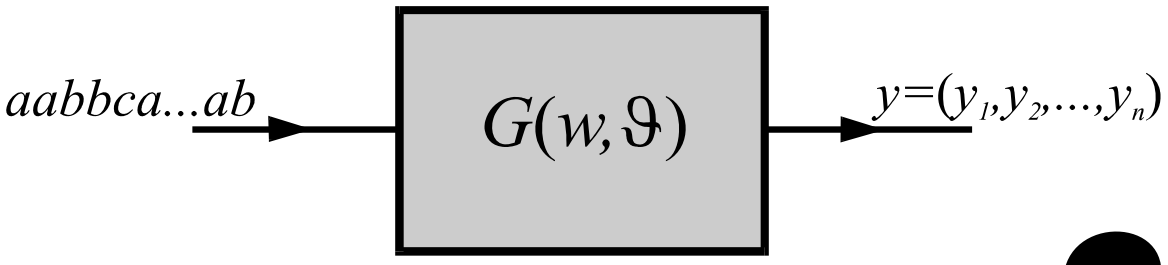
2. Elman's simple recurrent neural networks for classification of token strings of variable lengths

A process of transformation of strings composed of variable number of tokens onto a real vector of fixed dimension may be formally considered as a parametric mapping

$$G(w,\vartheta):\{a,b,c,...\}^{*}\rightarrow[0,1]^{n}$$

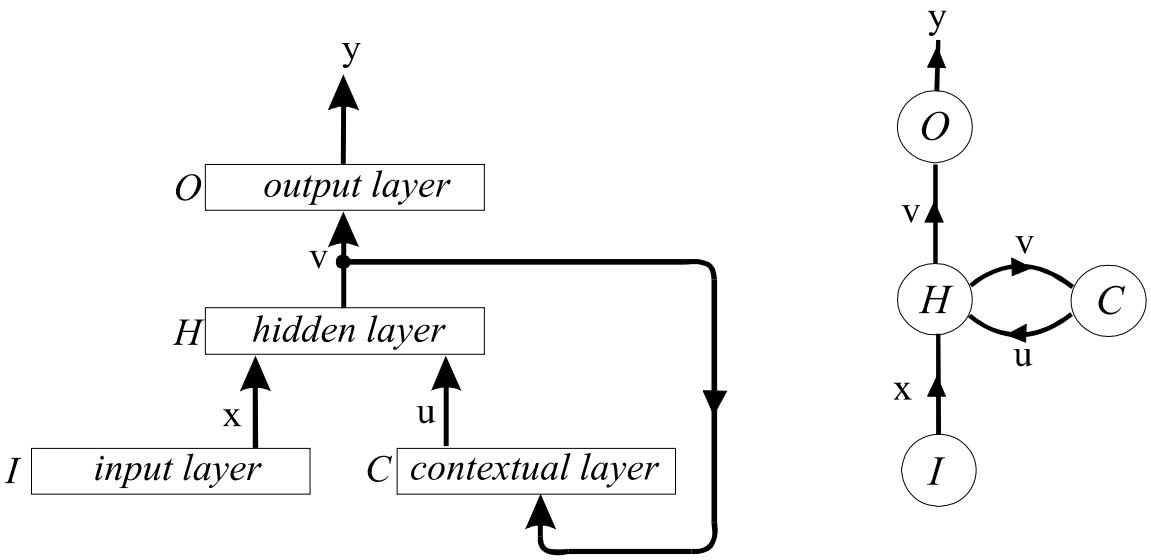
$$\boldsymbol{y}=G(\alpha;w,\vartheta)$$

This mapping may be diagrammatically expressed



The parametric mapping $G(w,\vartheta)$ will be realized by recurrent neural networks

Elman's recurrent neural network



$$\left.\begin{aligned} \boldsymbol{x} &= \text{input} \in \{a,b,c,...\}^* \\ \boldsymbol{u} &= C(\boldsymbol{v}) \\ \boldsymbol{v} &= H(\boldsymbol{x},\boldsymbol{u}) \\ \boldsymbol{y} &= O(\boldsymbol{v}) \in R^n \end{aligned}\right\} \Rightarrow \left\{\begin{aligned} \boldsymbol{x} &= \text{input} \in \{a,b,c,...\}^* \\ \boldsymbol{v} &= H\big(\boldsymbol{x},C(\boldsymbol{v})\big) \\ \boldsymbol{y} &= O(\boldsymbol{v}) \in R^n \end{aligned}\right.$$

a nonlinearity, this equation is solved by simple iterative scheme

$$\boldsymbol{v}_{k+1} = H\big(\boldsymbol{x},C(\boldsymbol{v}_k)\big)$$

$$\left.\begin{aligned} \boldsymbol{x}^{(t)} &= \text{input} \\ \boldsymbol{u}^{(t)} &= \begin{cases} 0 & (t=1) \\ C\left(\boldsymbol{v}^{(t-1)}\right) & (t \geq 2) \end{cases} \\ \boldsymbol{v}^{(t)} &= H\left(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)}\right) \\ \boldsymbol{y}^{(t)} &= O\left(\boldsymbol{v}^{(t)}\right) \end{aligned}\right\} \quad \text{for } t = 1,2,\ldots,t_{max}$$

Activities are explicitly determined by

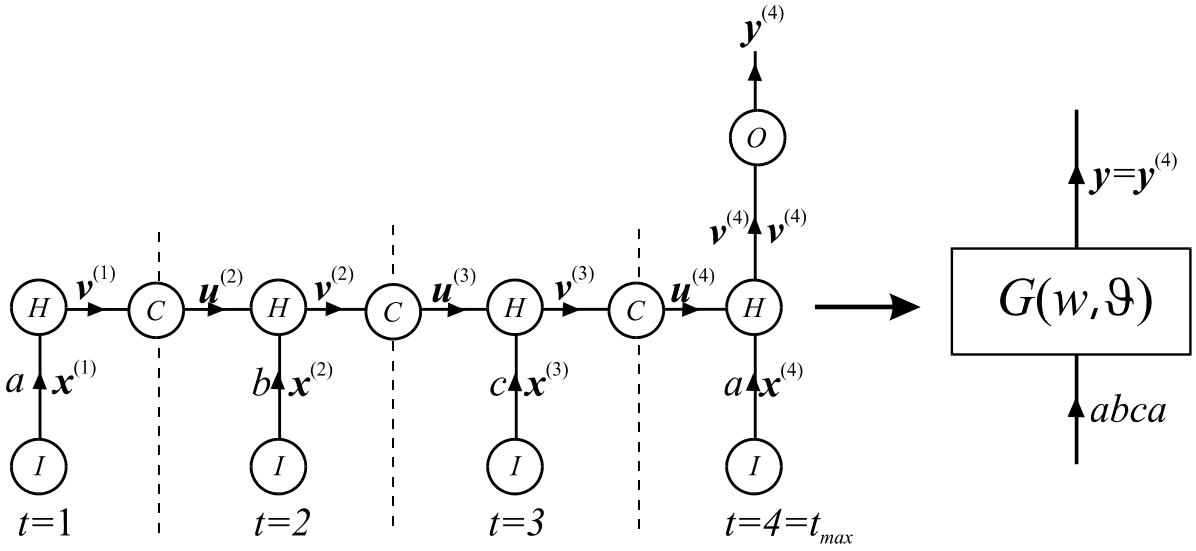
$$x_i^{(t)} = \text{input}$$

$$u_i^{(t)} = \begin{cases} 0 & (\text{for } t=1) \\ t\left(\vartheta_i^{(C)} + \sum_{j=1}^{N_H} w_{ij}^{(HC)} v_j^{(t-1)} \right) & (\text{otherwise}) \end{cases}$$

$$v_i^{(t)} = t\left(\vartheta_i^{(H)} + \sum_{j=1}^{N_I} w_{ij}^{(HI)} x_j^{(t)} + \sum_{j=1}^{N_C} w_{ij}^{(HC)} u_j^{(t)} \right)$$

$$y_i^{(t)} = t\left(\vartheta_i^{(O)} + \sum_{j=1}^{N_H} w_{ij}^{(OH)} v_j^{(t)} \right)$$

Unfolded Elman's recurrent neural network



Unfolded Elman's network is of the standard ”feed forward” type that can be simply adapted by the ”back propagation” method.

Model calculations performed by the Elman's recurrent neural network

Training set

No.	Token string	y_{req}
1	abb	0
2	bab	1
3	aabb	0
4	babaa	1
5	aba	0
6	bbbaba	1
7	aaabab	1
8	bbbaaa	0
9	aaabba	0
10	bbabb	1
11	ababaa	1
12	aabbaa	0
13	abaaaa	0
14	bbaabb	0
15	aabab	1
16	bba	0
17	bbaa	0
18	aabab	1
19	ababaa	1
20	bbaab	0

Testing set

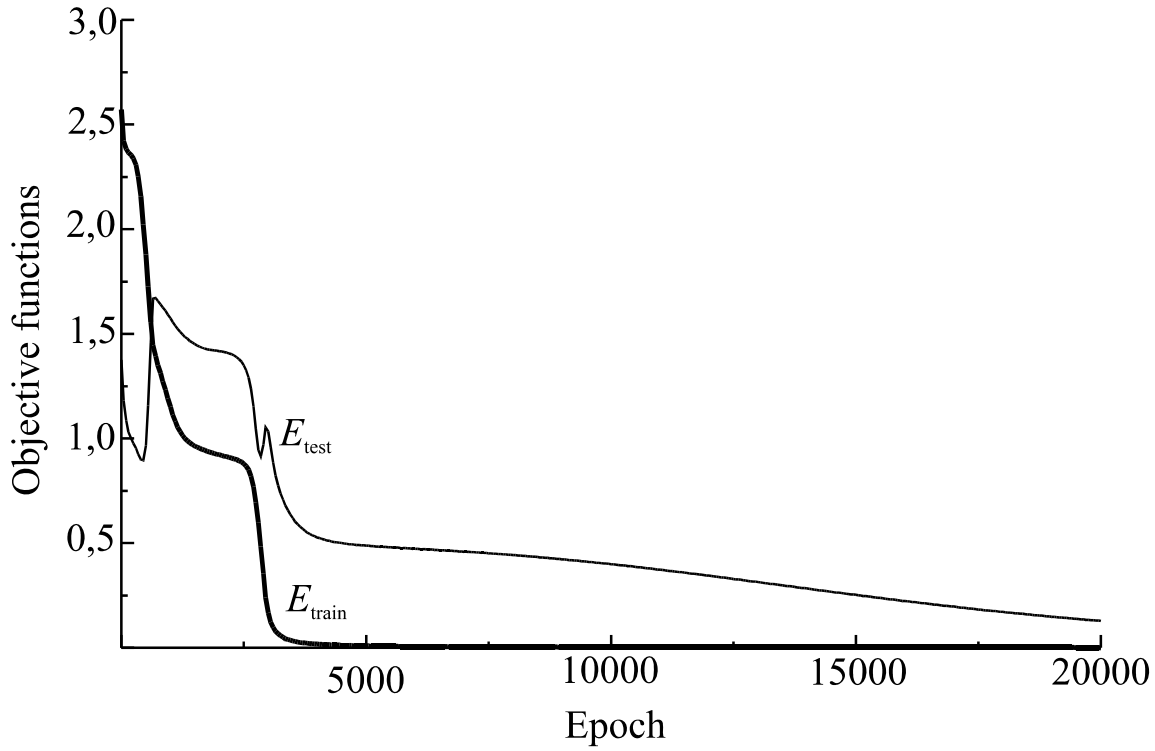
No.	Token string	y_{req}
1	babbab	1
2	bbbbab	1
3	aaabbb	0
4	abaabba	0
5	aaababaaa	1
6	ababaab	1
7	abbbaa	0
8	abbaa	0
9	bbbaabba	0
10	aababbbb	1

$$y_{req}(\alpha) = \begin{cases} 0 & (\text{if } \boldsymbol{bab} \notin \alpha) \\ 1 & (\text{if } \boldsymbol{bab} \in \alpha) \end{cases}$$

Elman's recurrent neural network is adapted with respect to the training set, then its adapted form is **verified** by patterns (strings) from the testing set.

$$E_{train} = \frac{1}{2} \sum_{\alpha \in A_{train}} \left(y_{calc} - y_{req} \right)^2$$

$$E_{test} = \frac{1}{2} \sum_{\alpha \in A_{test}} \left(y_{calc} - y_{req} \right)^2$$



Classification of testing patterns by adapted Elman's recurrent neural network

pattern	y_{calc}	y_{req}
babbab	**0.60**	1
bbbbab	0.94	1
aaabbb	0.01	0
abaabba	0.01	0
aaababaaa	0.99	1
ababaab	0.99	1
abbbaa	0.01	0
abbaa	0.01	0
bbbaabba	0.01	0
aababbbb	0.99	1

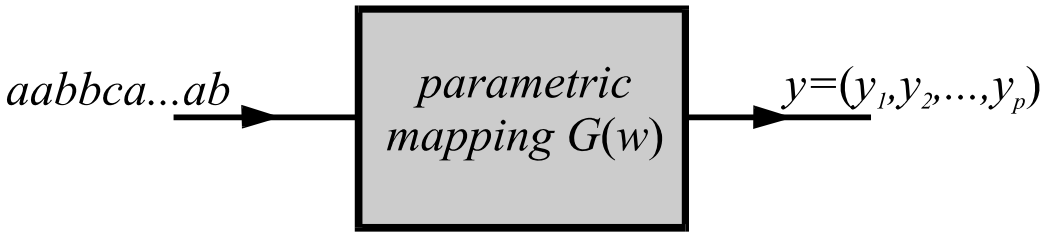
Conclusion: Elman's recurrent neural network is able to classify (almost correctly) token strings of variable lengths whether these strings contain a substring ***bab***

$$y_{calc}(\alpha) \cong \begin{cases} 0 & (\text{if } \textbf{\textit{bab}} \notin \alpha) \\ 1 & (\text{if } \textbf{\textit{bab}} \in \alpha) \end{cases}$$

3. Pollack's recurrent auto-associative memory (RAAM) for classification of rooted trees

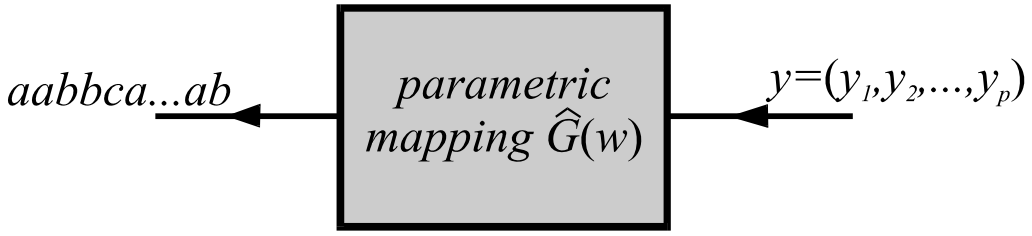
Two different parametric mappings are introduced

Direct mapping



$$G(w): A^* = \{a,b,c,...\}^* \rightarrow (0,1)^p$$

Inverse mapping



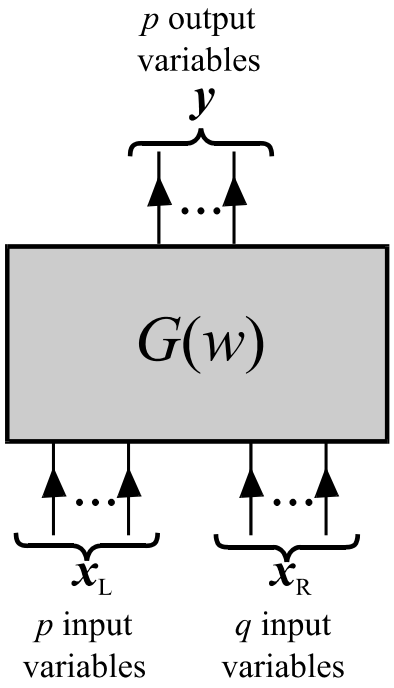
$$\hat{G}(\hat{w}): (0,1)^p \rightarrow A^* = \{a,b,c,...\}^*$$

A way of realization of the direct

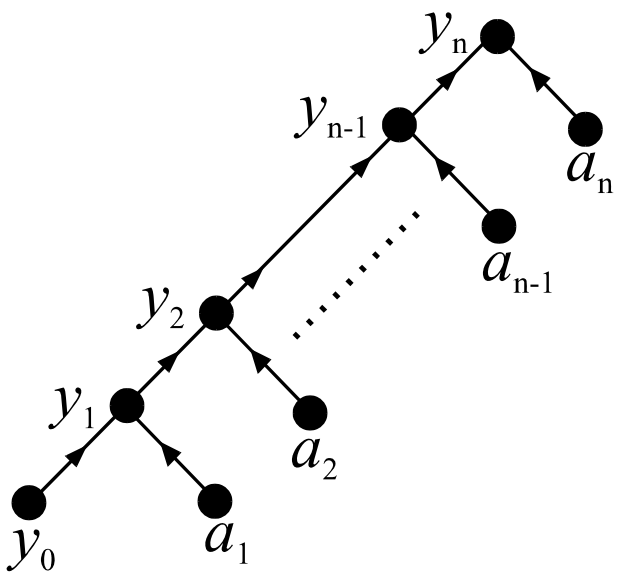
mapping

Let us postulate that string tokens are numerically coded by q-dimensional binary vectors.

The parametric mapping $G(w)$ is specified as follows



$$\boldsymbol{y} = G\left(\boldsymbol{x}_L, \boldsymbol{x}_R; w\right)$$



$$\boldsymbol{y}_1 = G(\boldsymbol{\varepsilon}, \boldsymbol{a}_1; w)$$

$$\boldsymbol{y}_2 = G(\boldsymbol{y}_1, \boldsymbol{a}_2; w)$$

$$\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\boldsymbol{y}_n = G(\boldsymbol{y}_{n-1}, \boldsymbol{a}_n; w)$$

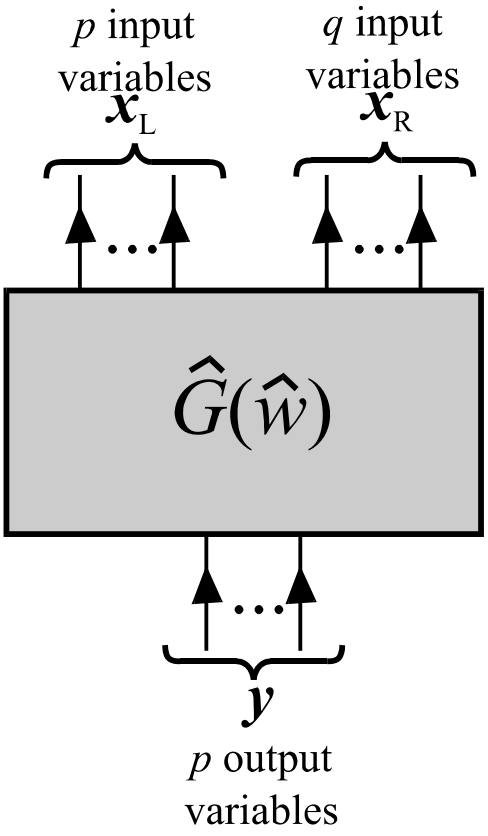
A final vector $\boldsymbol{y}_n$ represents a mapping of the string $\alpha = (a_1 a_2 ... a_n) \in A^*$ onto a p-dimensional real vector $\boldsymbol{y} = (y_1, y_2, ..., y_p) \in (0,1)^p$

Empty token is coded by p-dimensional vector composed entirely of 0-entries

$$\boldsymbol{y}_0 = (0, 0, ..., 0) \in (1, 0)^p$$

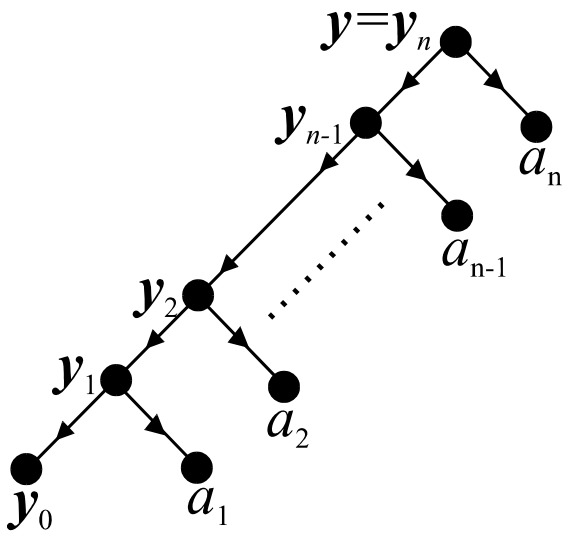
A way of realization of the inverse mapping

The inverse parametric mapping $\hat{G}(\hat{w})$ is specified as follows



$$(x_L, x_R) = \hat{G}(y; \hat{w})$$

A code $\boldsymbol{y}=\boldsymbol{y}_n$ is recurrently decoded as follows



$$(\boldsymbol{y}_{n-1}, a_n) = \hat{G}(\boldsymbol{y}_n; \hat{w})$$

$$(\boldsymbol{y}_{n-2}, a_{n-1}) = \hat{G}(\boldsymbol{y}_{n-1}; \hat{w})$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$(\boldsymbol{y}_1, a_2) = \hat{G}(\boldsymbol{y}_2; \hat{w})$$

$$(\boldsymbol{y}_0, a_1) = \hat{G}(\boldsymbol{y}_1, \hat{\boldsymbol{w}})$$

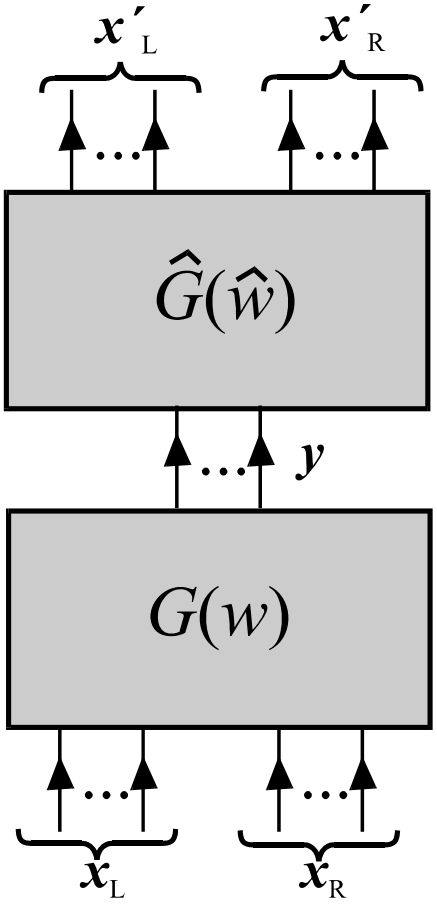
Note: Applying this recurrent calculation we obtain a token sequence a_n, a_{n-1}, …, a_2, a_1 (which constitutes the string α) from the vector $\boldsymbol{y}$ by making use of the inverse mapping $\hat{G}(\hat{w})$.

Both mappings $G(w)$ and $\hat{G}(\hat{w})$ are unified to auto-associative mapping transforming strings onto strings

$$y = G(x_L, x_R; w) \qquad (x_L, x_R) = \hat{G}(y; \hat{w})$$

$$\downarrow$$

$$(x'_L, x'_R) = \hat{G}(G(x_L, x_R; w); \hat{w})$$



Composite mapping is auto-associative

$$\downarrow$$

$$(\boldsymbol{x}_L, \boldsymbol{x}_R) = \hat{G}(G(\boldsymbol{x}_L, \boldsymbol{x}_R; w); \hat{w})$$

A fulfillment of this condition depends on the parameters of both mappings G and $\hat{G}$. Let us define an ***objective function***

$$E(w, \hat{w}) = \frac{1}{2}\Big((\boldsymbol{x}'_L(w, \hat{w}) - \boldsymbol{x}_L)^2 + (\boldsymbol{x}'_R(w, \hat{w}) - \boldsymbol{x}_R)^2\Big)$$

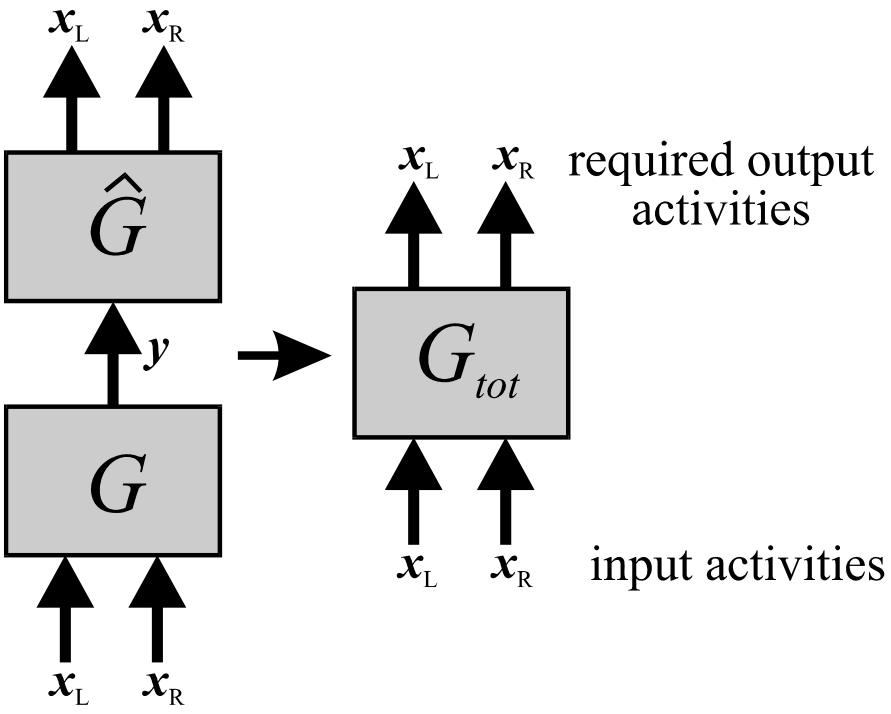
If composite mapping $\hat{G} \circ G$ is auto-associative, then this objective function should be vanishing

How to learn the auto-associative

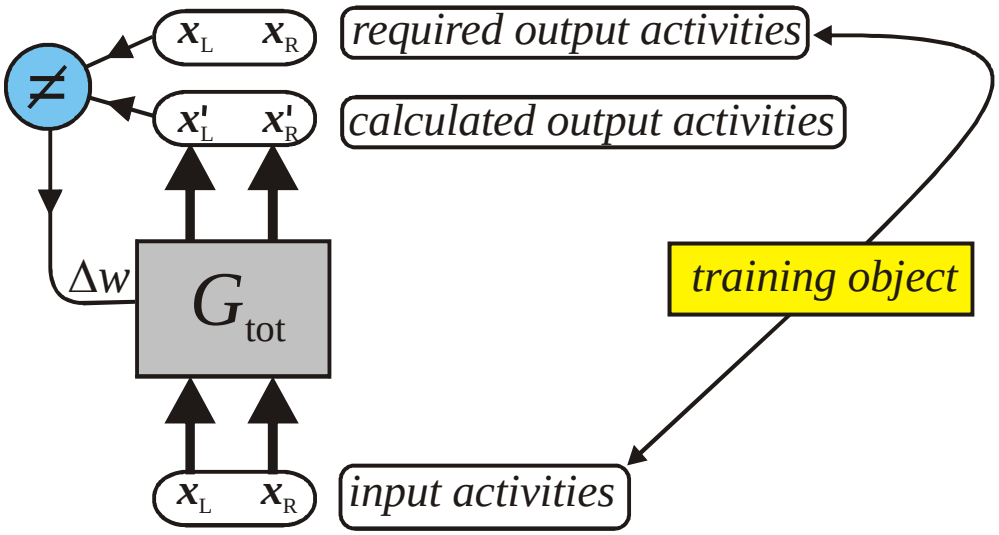
composite mapping $\hat{G} \circ G$?

Let us introduce the notation

$$(\boldsymbol{x}_L, \boldsymbol{x}_R) = \hat{G}(G(\boldsymbol{x}_L, \boldsymbol{x}_R; w); \hat{w})$$
$$= G_{tot}(\boldsymbol{x}_L, \boldsymbol{x}_R; w, \hat{w})$$

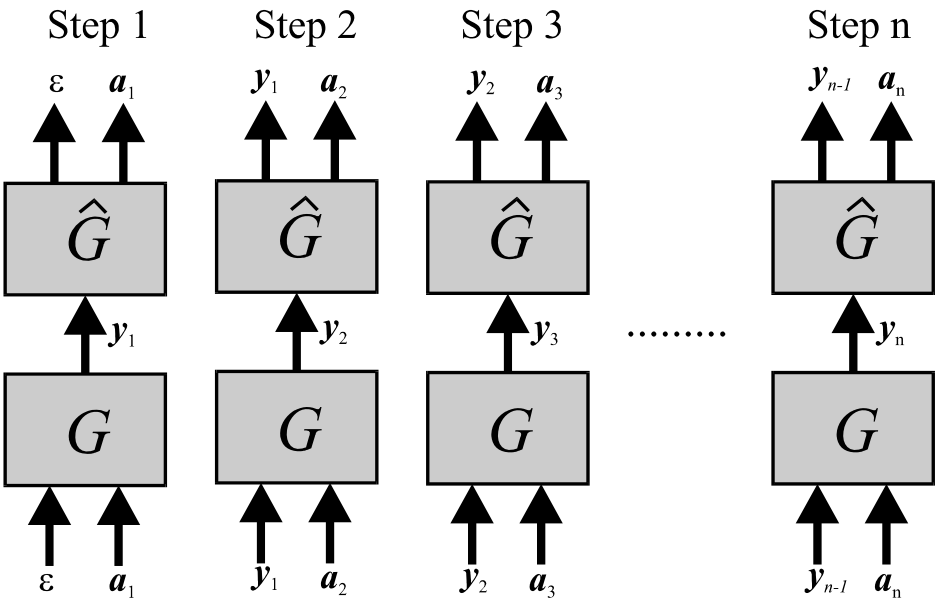


An outline of the learning process



The adaptation process corresponds to the so-called ”**moving target problem**”

A string $\alpha = (a_1 a_2 ... a_n) \in A^*$ from the training set corresponds to *n* learning tasks



$$\boldsymbol{y}_1, \boldsymbol{y}_2, ... \boldsymbol{y}_n$$

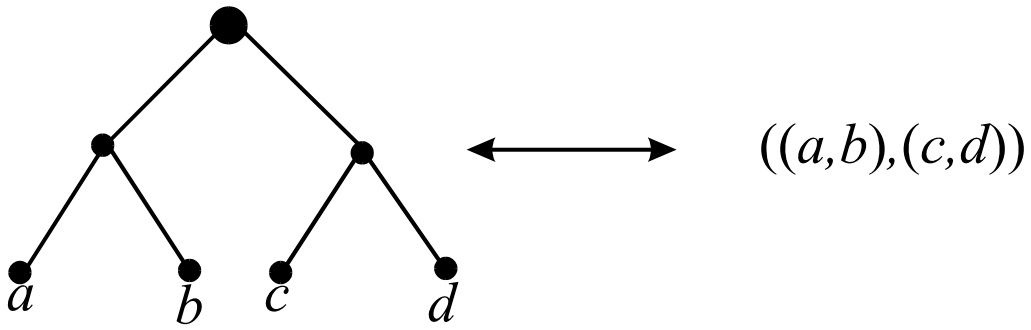
The last vector $\boldsymbol{y}_n$ is a final output from the RAAM mapping G_{tot}. If this mapping is auto-associative (i.e. input and output are equal), then $\boldsymbol{y}_n$ is a vector code of the string α. Formally

$$\boldsymbol{y} = \boldsymbol{y}_n = G_{tot}(\alpha; w, \hat{w})$$

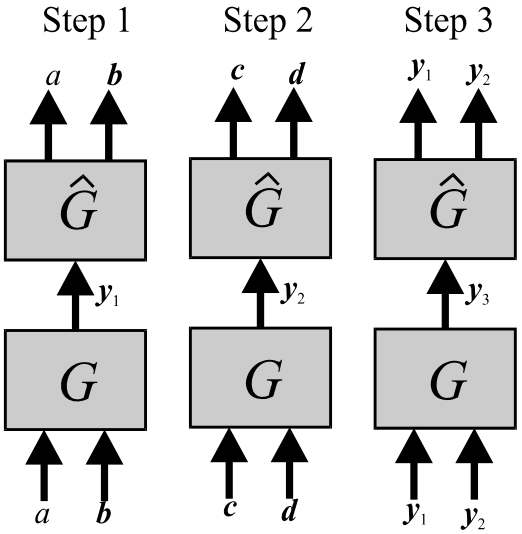
$$G_{tot}(w, \hat{w}): A^* \to (0,1)^p$$

How to process binary trees by Pollack's RAAM ?

RAAM networks are applicable also for the processing of **binary trees** (with terminal vertices evaluated by tokens).



General RAAM is modified so that $p=q$, then

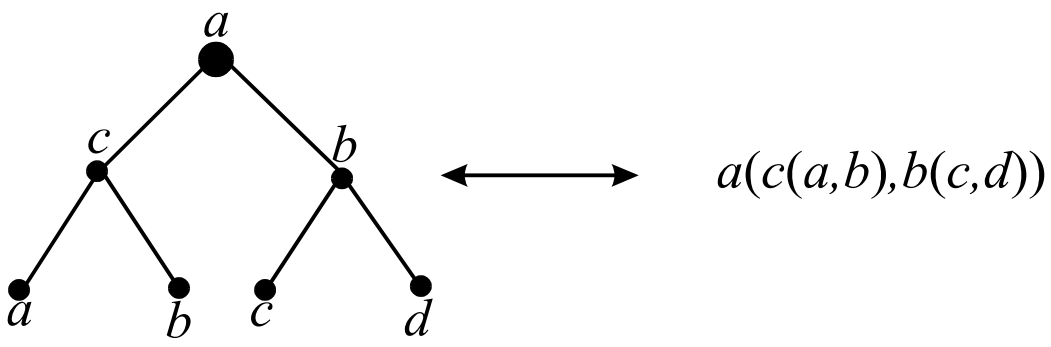


$$\boldsymbol{y}_1 = G_{tot}\left(\boldsymbol{a},\boldsymbol{b};\boldsymbol{w},\hat{\boldsymbol{w}}\right)$$
$$\boldsymbol{y}_2 = G_{tot}\left(\boldsymbol{c},\boldsymbol{d};\boldsymbol{w},\hat{\boldsymbol{w}}\right)$$
$$\boldsymbol{y}_3 = G_{tot}\left(\boldsymbol{y}_1,\boldsymbol{y}_2;\boldsymbol{w},\hat{\boldsymbol{w}}\right)$$

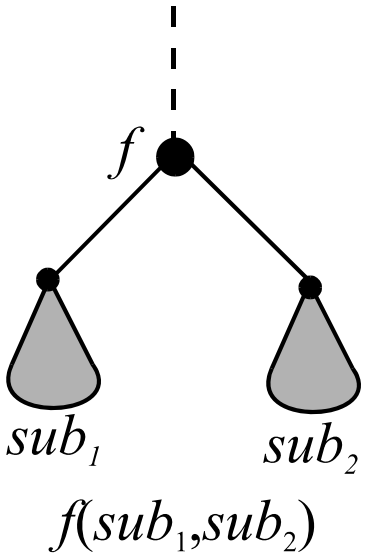
Illustrative example of RAAM application to binary strings

No.	String	*y*	Diagrammatic representation
1	0	(0.51,0.00,0.28)	
2	1	(0.45,1.00,0.67)	
3	00	(0.82,0.01,0.39)	
4	01	(0.77,1.00,0.77)	
5	10	(0.20,0.00,0.22)	
6	11	(0.17,0.99,0.60)	
7	000	(0.87,0.01,0.23)	
8	001	(0.84,1.00,0.61)	
9	010	(0.30,0.00,0.12)	
10	011	(0.25,0.99,0.41)	
11	100	(0.70,0.00,0.54)	
12	101	(0.65,1.00,0.86)	
13	110	(0.13,0.00,0.34)	
14	111	(0.11,0.99,0.73)	

4. Sperduti's labeled RAAM (LRAAM) for classification of rooted trees with vertices evaluated by symbols



Each nonterminal vertex has two predecessors.

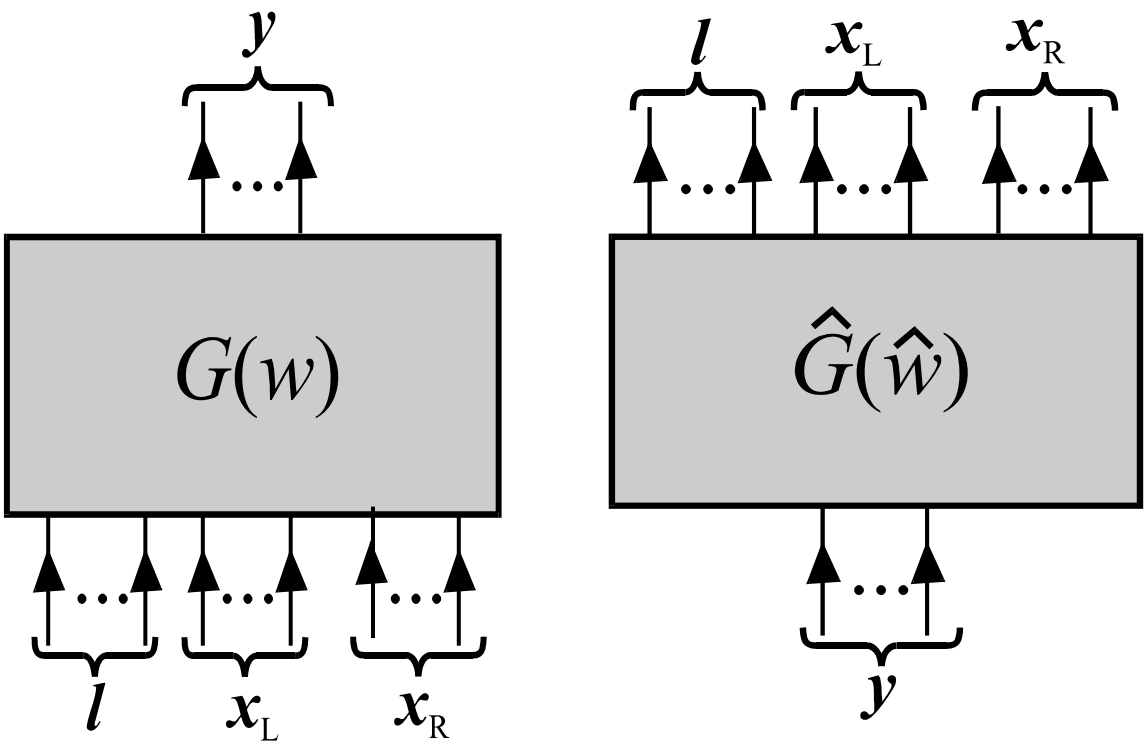


$f = a$

$sub_1 = c(a,b)$

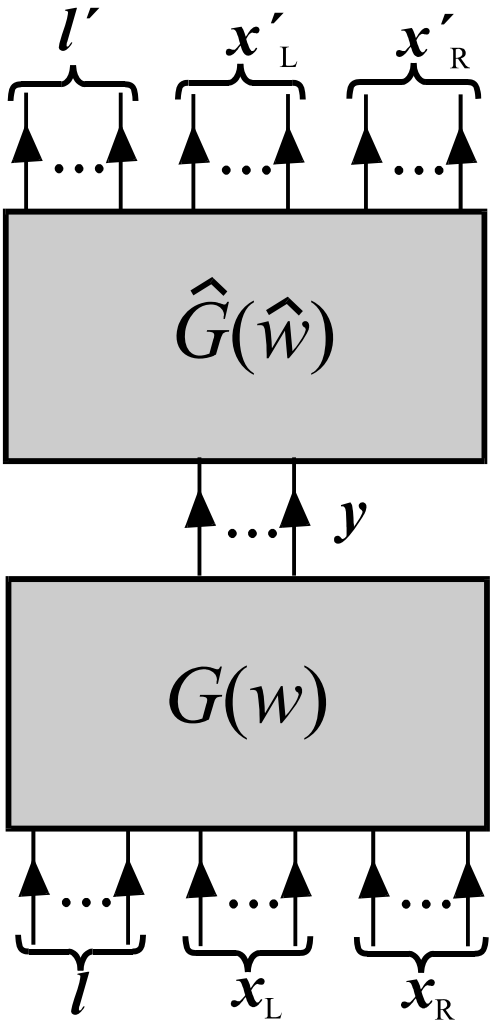
$sub_2 = b(c,d)$

Two parametric mappings $\boldsymbol{G}(\boldsymbol{w})$ and $\hat{G}(\hat{\boldsymbol{w}})$ are introduced

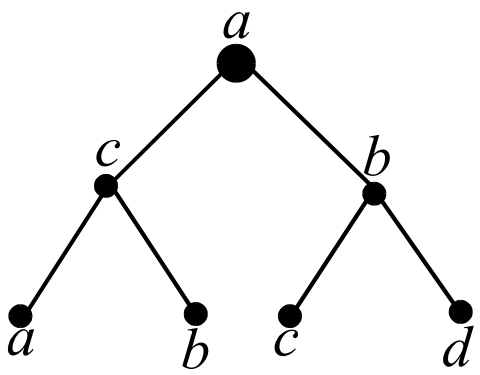


A composition of these mappings gives an autoassociative mapping

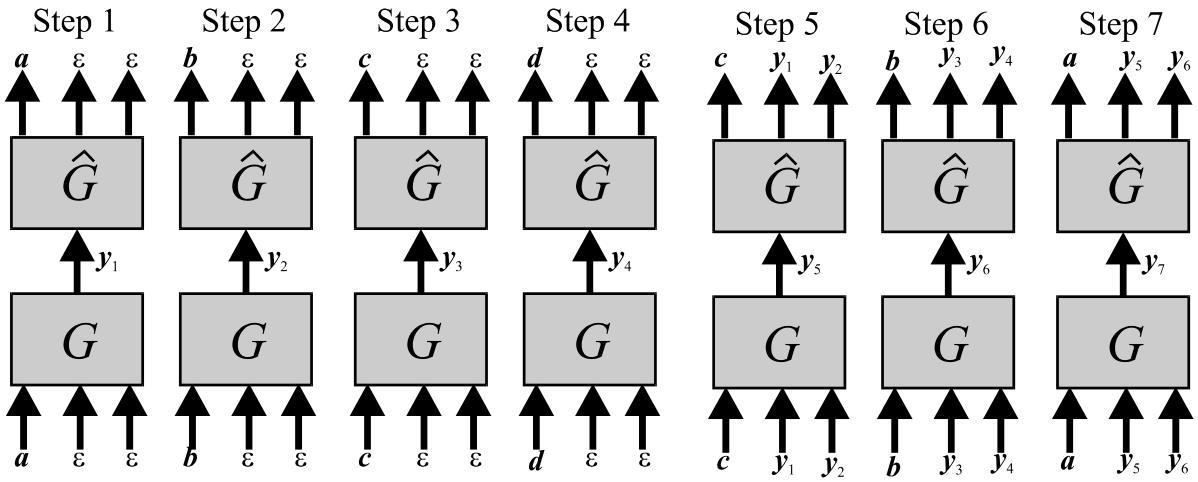
$$(\boldsymbol{l}', \boldsymbol{x}_L', \boldsymbol{x}_R') = G_{tot}(\boldsymbol{l}, \boldsymbol{x}_L, \boldsymbol{x}_R; \boldsymbol{w}, \hat{\boldsymbol{w}})$$
$$= \hat{G}\big(G(\boldsymbol{l}, \boldsymbol{x}_L, \boldsymbol{x}_R; \boldsymbol{w}); \hat{\boldsymbol{w}}\big)$$



We apply the present approach to a binary tree with all vertices evaluated by tokens

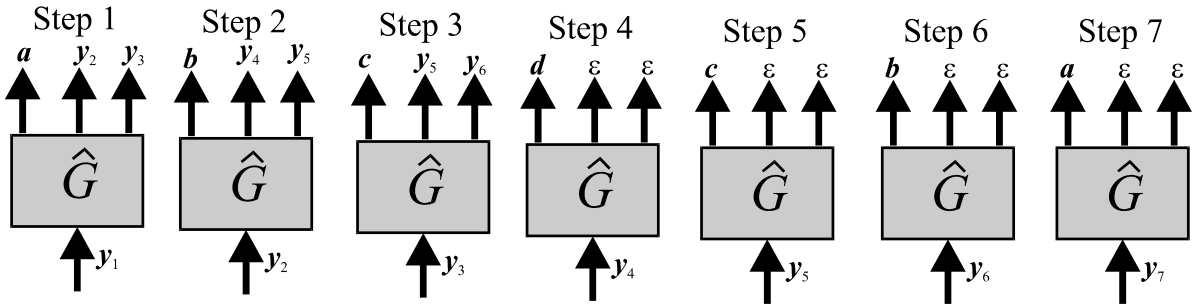


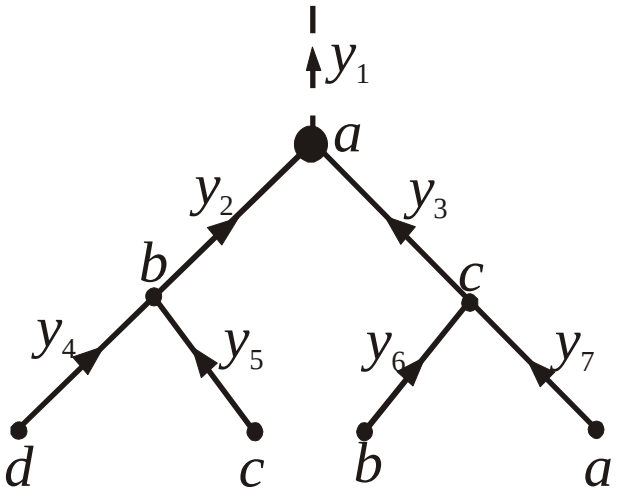
$a(c(a,b),b(c,d))$



Labeled tree is coded by $\boldsymbol{y}_7$

How to decode the real vector *y* that codes a labeled binary tree?





$$a(b(d,a),c(b,a))$$

Illustrative example - learning syntactic trees

Simple context free grammar

$$S \rightarrow S(NP,VP) \mid S(NP,V)$$

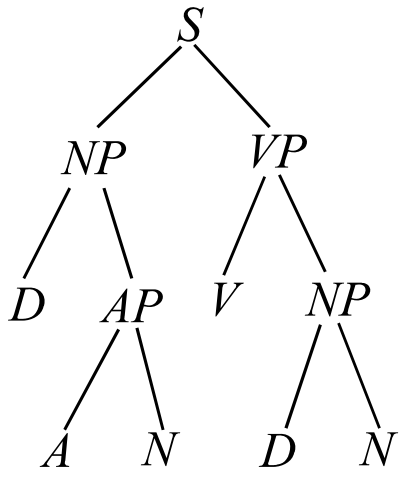
$$NP \rightarrow NP(D,AP) \mid NP(D,N) \mid NP(NP,PP)$$

$$PP \rightarrow PP(P,NP)$$

$$VP \rightarrow VP(V,NP) \mid VP(V,PP)$$

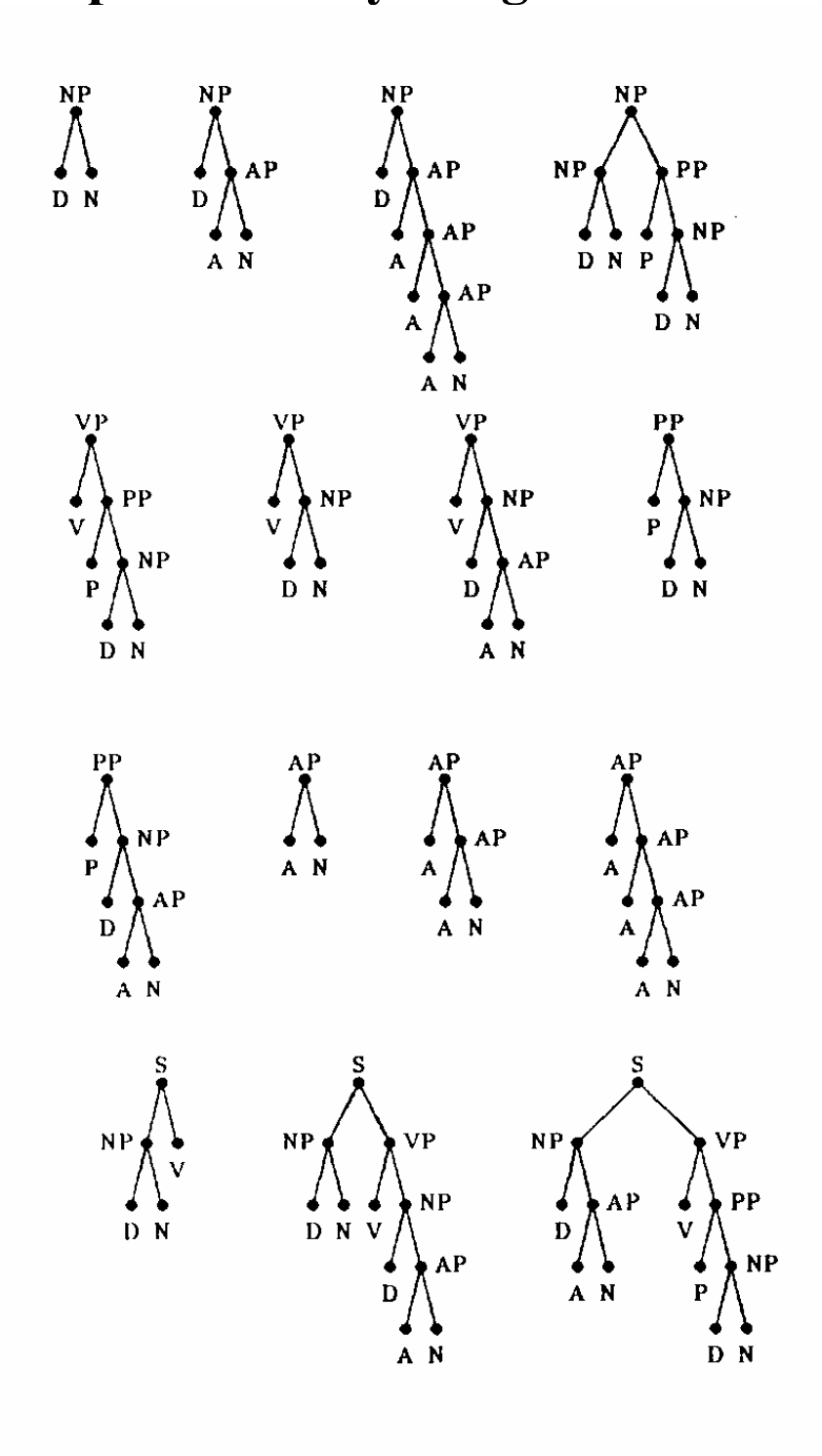
$$AP \rightarrow AP(A,AP) \mid AP(A,N)$$

Nonterminal symbols: S, NP, VP, PP, AP
Terminal symbols : D, P, A



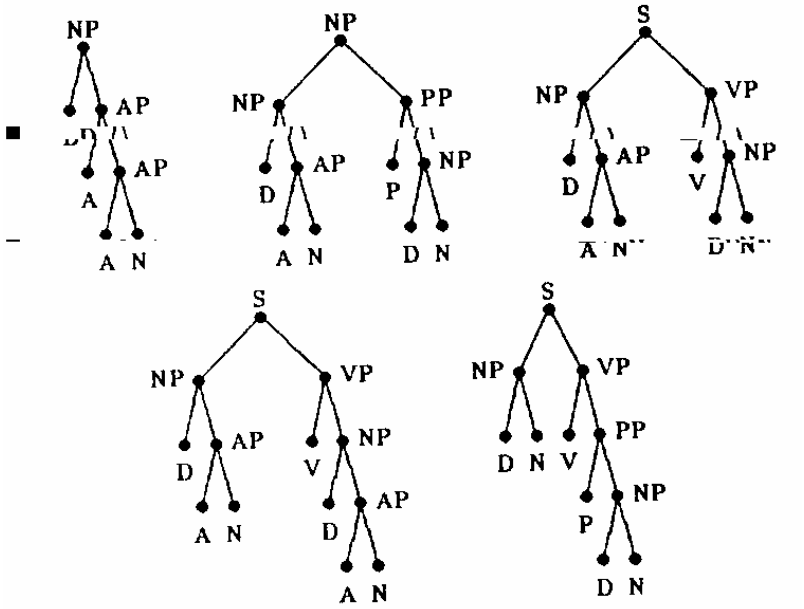
$$S(NP(D,AP(A,N))),VP(V,NP(D,N)))$$

Training set composed of 15 syntactic trees produced by the grammar

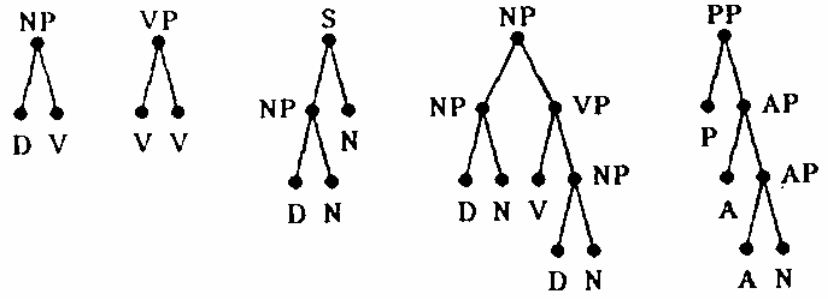


Syntactic trees produced by the adapted LRAAM from randomly generated vectors *y*

Trees belonging to the grammar

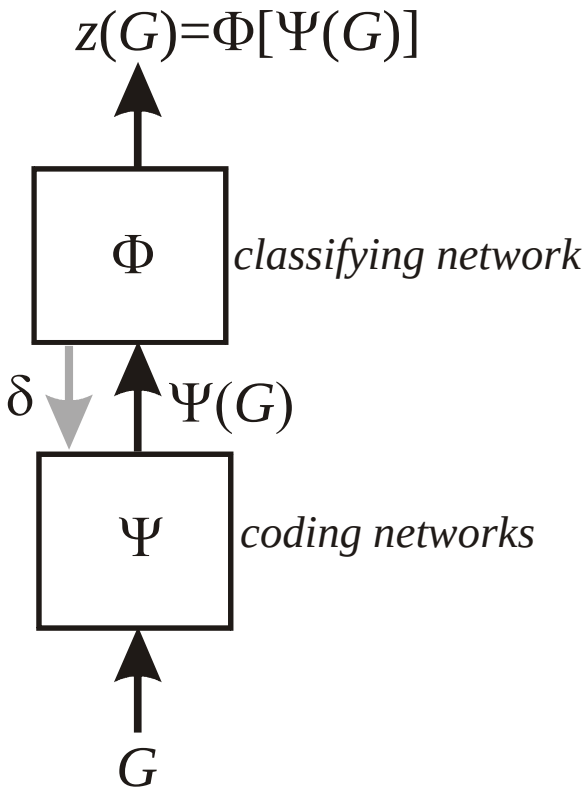


Trees not belonging to the grammar

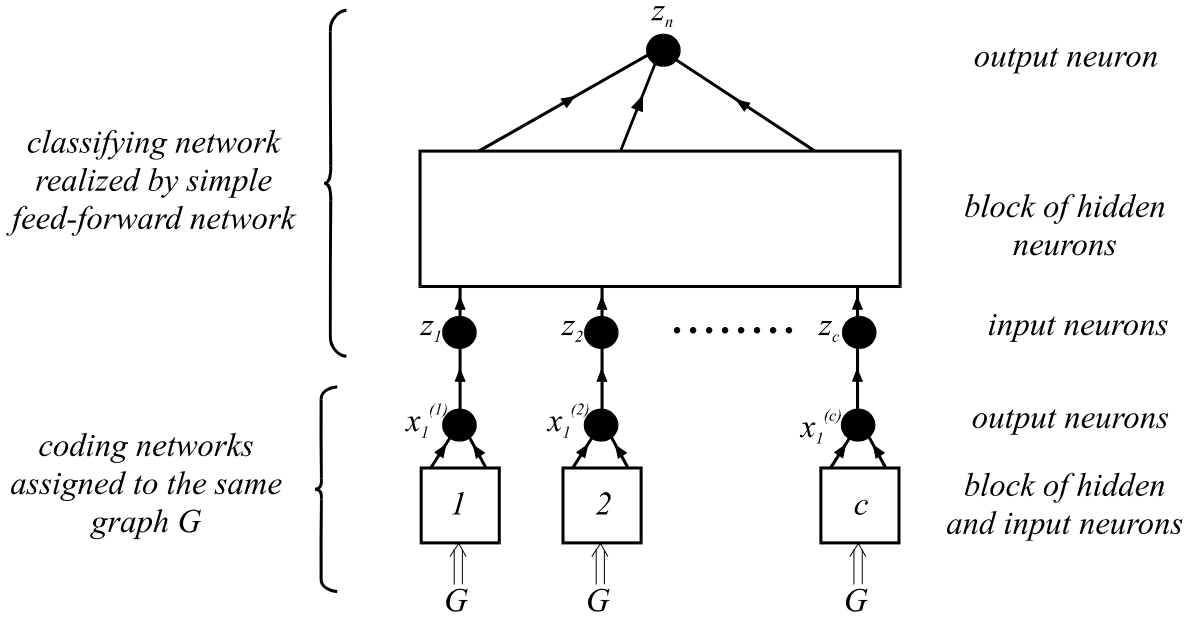


5. Sperduti's feed-forward neural networks for classification of cyclic oriented graphs with evaluated vertices

A new architecture designed by Alessandro Sperduti (1997)



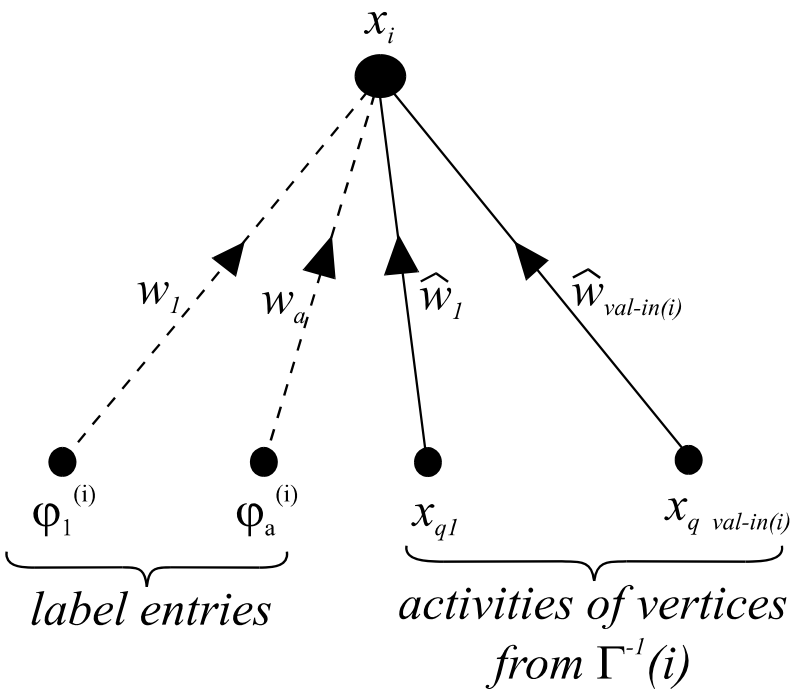
A detailed view of the proposed architecure



A total network for the classification of graphs is a composition of a feed-forward neural network and neural networks assigned to the graph *G*. The bottom part (called the **coding networks**) is composed of *c* neural networks with identical architecture, which are assigned to the same graph *G*. Output activities of these nets are used as input activities of (top part of diagram) a feed-forward neural network (called the **classifying network,** it is composed of *c* input neurons, one output neuron, and $N_h=n-c$ hidden neurons), its output activity is assigned to a classification of the graph *G*.

How to code acyclic oriented graphs?

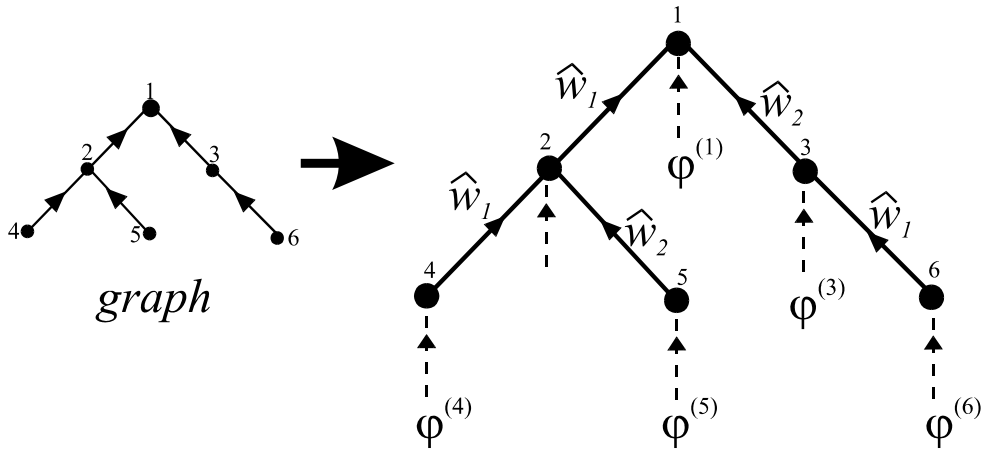
Activity of a single neuron assigned to the i-th graph vertex is determined as a weight summation of label entries and activities of the "previous" graph vertices from $\Gamma^{-1}(i)$



$$x_i = t(\xi_i) = \frac{1}{1+e^{-\xi_i}}$$

$$\xi_i = \sum_{k=1}^{a} w_k \varphi_k^{(i)} + \sum_{q \in \Gamma^{-1}(i)} \hat{w}_{\mu(q,i)} x_q$$

Illustrative example of a neural network assigned to an acyclic oriented graph



Activities of single neurons are calculated recurrently in a bottom-up manner

$$x_6 = t\left(w_1\varphi_1^{(6)}\right), \quad x_5 = t\left(w_1\varphi_1^{(5)}\right), \quad x_4 = t\left(w_1\varphi_1^{(4)}\right)$$

$$x_3 = t\left(w_1\varphi_1^{(3)} + \hat{w}_1 x_6\right)$$

$$x_2 = t\left(w_1\varphi_1^{(2)} + \hat{w}_1 x_4 + \hat{w}_2 x_5\right)$$

$$x_1 = t\left(w_1\varphi_1^{(1)} + \hat{w}_1 x_2 + \hat{w}_2 x_3\right)$$

Output activity x_1 (assigned to the root of graph) is a numerical representation of the graph used as an input to the top classifying network.

Adaptation process

Neural network may be considered as a parametric mapping that assigns to each graph G an output activity

$$F\left(w,\vartheta\right):G\rightarrow\left(0,1\right)$$

A goal of adaptation process is to determine parameters of the neural network so that "functional value" $F(G;w,\vartheta)$ is closely related to the required property $\chi(G)$, for all objects of the training set.

$$E(w,\vartheta)=\frac{1}{2}\sum_{G/\chi(G)\in A_{train}}\left[F(G;w,\vartheta)-\chi(G)\right]^2$$
$$=\sum_{G/\chi(G)\in A_{train}}E^{(G)}$$

Adaptation process consists in a minimization of the above objective (error) function

$$\left(w_{opt}, \vartheta_{opt}\right) = \arg \min_{(w,\vartheta)} E(w,\vartheta)$$

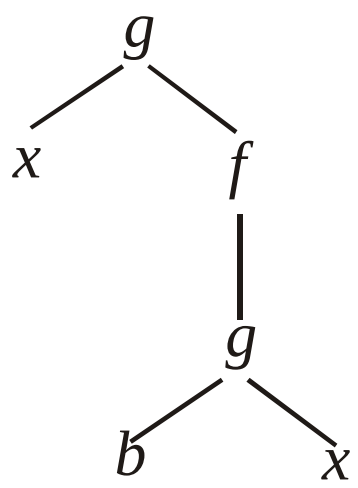
It is believed that a parametric function $F(w_{opt},\vartheta_{opt})$, induced by the adapted neural network well aproximates the property function χ

$$\chi(G) \approx F\left(G; w_{opt}, \vartheta_{opt}\right)$$

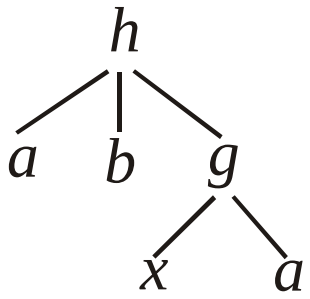
Illustrative example – labeled trees generated by a context free grammar

$$r \rightarrow f(x) \,|\, g(x,x) \,|\, h(x,x,x)$$
$$x \rightarrow a \,|\, b \,|\, f(x) \,|\, g(b,x) \,|\, g(x,a) \,|\, h(a,b,x) \,|$$
$$h(b,x,a) \,|\, h(x,a,b)$$

nonterminal symbols: r,f,g,h
terminal symbols: a,b,x



$g(x,f(g(b,x)))$



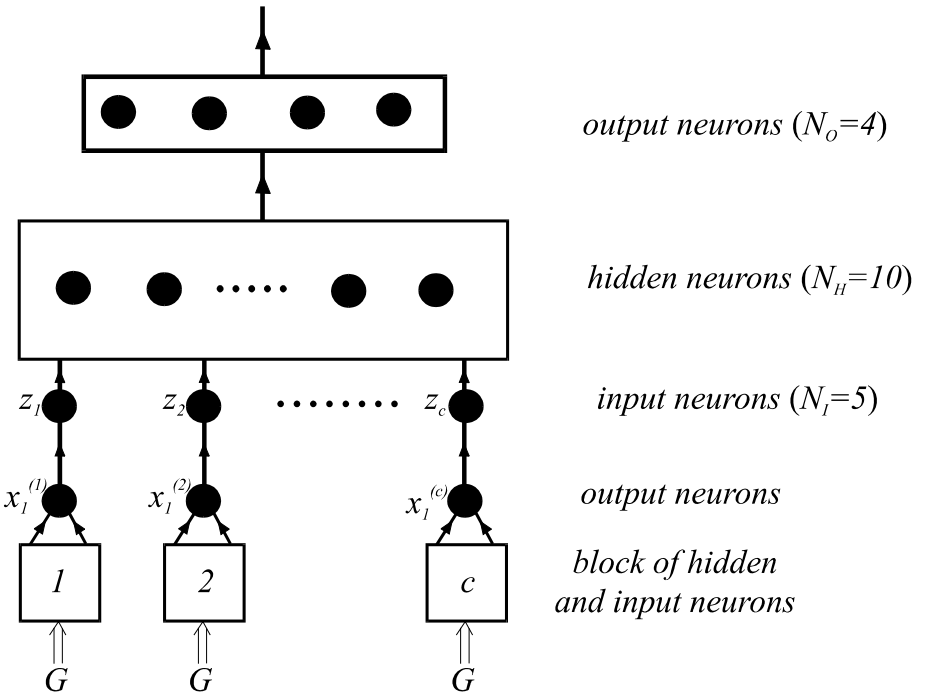
$h(a,b,g(x,a))$

Training set (composed of 12 vertices, at most)

No.	Classification	Property	Number
1	(1,1,0,0)	Trees are generated by the grammar and contain a subtree $f(g(\ ,\ ))$	35
2	(1,0,1,0)	Trees are generated by the grammar and contain a subtree g(b,a)	25
3	(1,0,0,1)	Trees are generated by the grammar and contain a subtree g(b,g(,))	40
4	(0,0,0,0)	Trees do not belong to the grammar	50

Σ=150

Architecture of the used neural network



epochs = 10.000, λ=0.1

E_{train}=0.02 (all training patterns are correctly classified)

Testing set

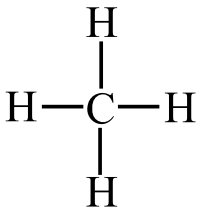
No.	Classification	Number	Correct.
1	(1,1,0,0)	25	23 (92%)
2	(1,0,1,0)	30	29 (97%)
3	(1,0,0,1)	20	18 (90%)
4	(0,0,0,0)	25	25 (100%)

Σ=100

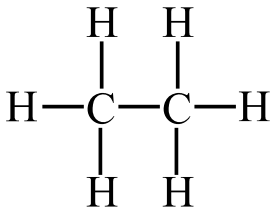
Chemical application - prediction of C^{13} NMR chemical shifts of alkanes

C^{13} NMR chemical shift is a physico-chemical property of atoms in molecules, it is important experimental parameter corresponding to an electron environment of the given atom in molecule.

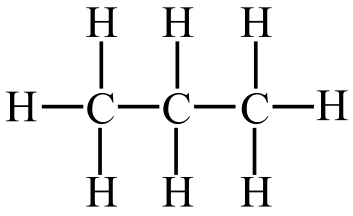
Alkanes are saturated acyclic hydrocarbons



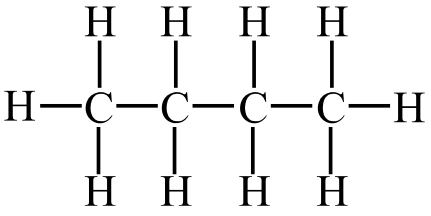
methan



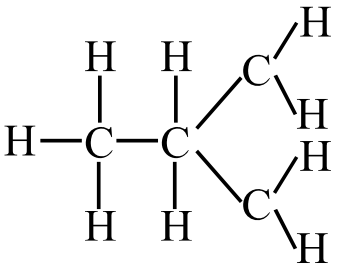
ethan



propan



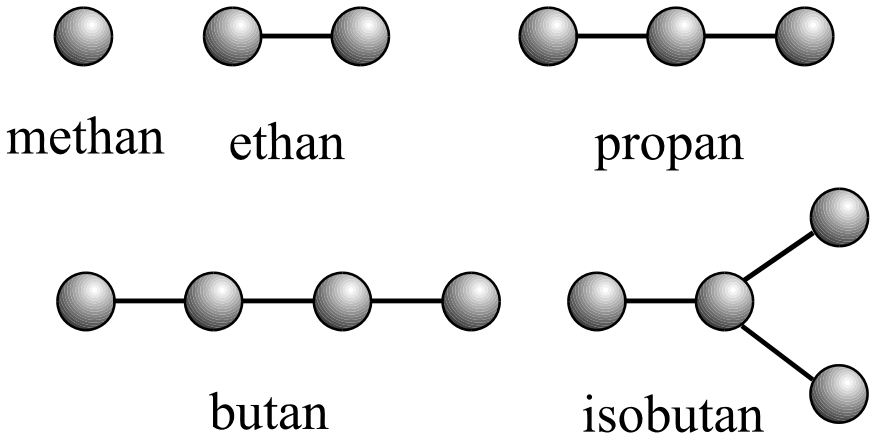
butan



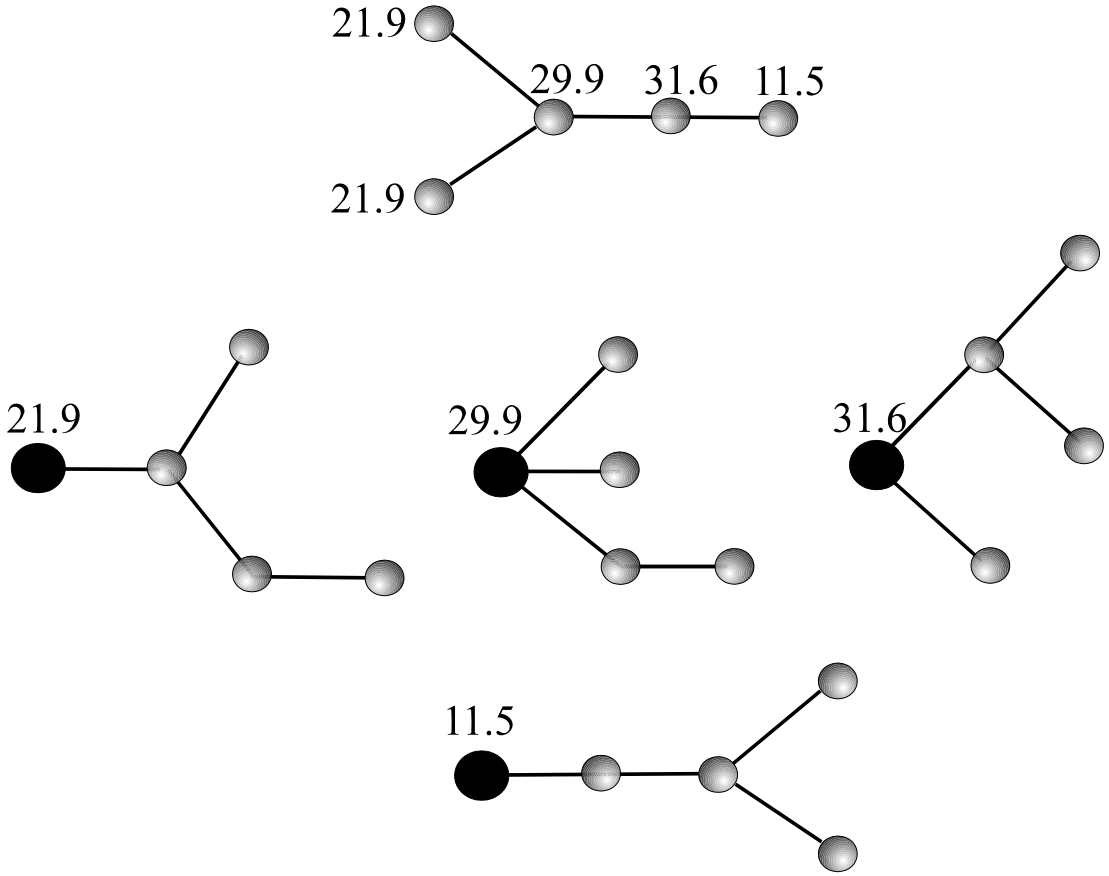
isobutan

Note: The fuel of cigarette lighters is propan and butan

Graph-theoretical representation
of alkanes



Experimental C13 NMR chemical shifts



Each rooted tree in the upper row is evaluated by the chemical shift assigned to the root, whereas in the bottom row vertices of rooted trees are evaluated by labels that correspond to the so-called in-valences.

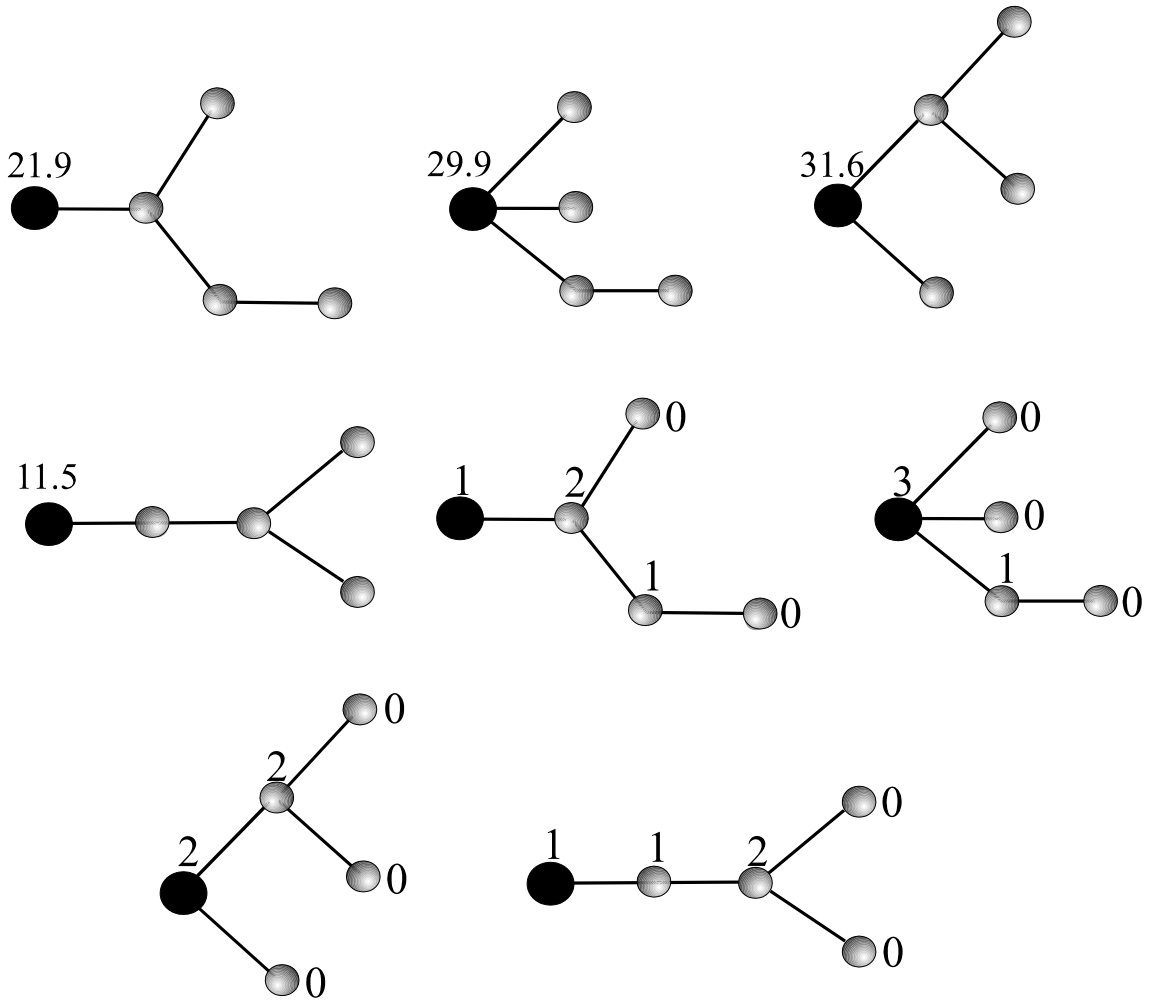


Table. Values of training and testing objective functions for different values of hidden and input neurons

N_H	N_I	E_{train}	E_{test}	N_H	N_I	E_{train}	E_{test}
2	2	5.250	5.720	4	2	2.423	2.440
2	3	4.490	4.550	4	3	2.421	2.450
2	4	4.421	4.338	4	4	2.466	2.501
2	5	2.612	2.950	4	5	2.601	2.609
3	2	1.860	1.856	5	2	4.251	4.260
3	3	1.222	1.260	5	3	4.101	4.180
3	4	1.227	1.314	5	4	3.992	4.001
3	5	1.860	1.856	5	5	3.910	4.012

Summary: The present approach allows to use acyclic oriented graphs as a direct input of feed-forward neural networks. There is not required a coding of graphs into a form of feature vectors (vectors of real entries with fixed dimension). Presented chemical application shows that the theory offers very promising results for a classification of acyclic molecular structures.

6. Conclusions

- **We have described a few types of neural networks that are able to process structured data.**
- **Linearly structured data (token strings) are simply processed by recurrent neural network (we use the Elman's recurrent network).**
- **Labeled trees are processed by Pollack's RAAM (or its generalized version LRAAM).**
- **Recently, Sperduti's feed-forwad neural network represents most general approach for processing of labeled acyclic oriented graphs. This type of neural network opens completely new possibilities for chemical applications.**
- **Summarizing, current state of the art of neural networks allows their application to direct processing of structured data.**