Generalized Theory of Recurrent Auto-Associative Memory (RAAM) for Linear Strings of Symbols

Vladimír Kvasnička

Institute of Applied Informatics
Faculty if Informatics and Information Technologies STU
Bratislava

email: **kvasnicka@fiit.stuba.sk**

November 2007

Basic concepts of strings

A vocabulary of symbols-tokens

$$A = \{a, b, c, ...\}$$

A set composed of all possible strings that can be constructed from this vocabulary is denoted by

$$A^* = \{a, b, c, ...\}^* = \{\varepsilon, a, b, c, ..., aa, ab, ...aaa, aab, ...\}$$

where ε is the so-called empty string (or empty token). A length (number of nonempty tokens) of the string $\alpha \in A^*$ is denoted by $|\alpha|$, then $|\varepsilon| = 0$.

Example

$$A = \{a, b\}$$

$$A^* = \{a, b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, ...aaa, aab, ...\}$$

$$\alpha = (aabbaa) \in A^* \Rightarrow |\alpha| = 6$$

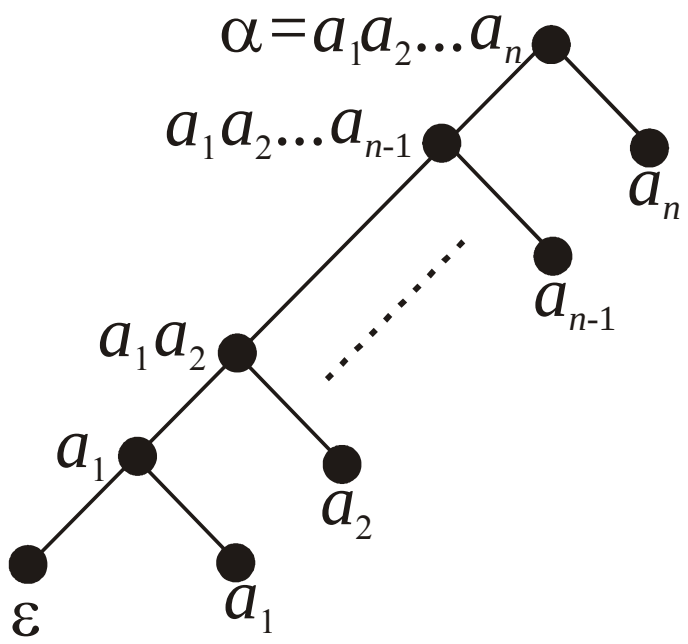
$$\beta = (baba) \in A^* \Rightarrow |\beta| = 4$$

A binary-tree representation of strings

Let us consider a string

$$\alpha = (a_1 a_2 \ldots a_n) \in A^*$$

its length is $|\alpha| = n$. This string may be represented as a binary tree as follows



Its leafs are assigned to single tokens of the string.

How to code strings?

Symbols from a vocabulary $A=\{a,b,c,...\}$ are coded by q-dimensional binary vectors, e.g.

$$a=(1,0,0,...)\in\{0,1\}^q$$
$$b=(0,1,0,...)$$
$$c=(0,0,1,...)$$
$$.....$$

A string from $\{a,b,c,...\}^*$ is then considered as a sequence of binary vectors

$$abca\rightarrow((100...),(010...),(001...),(100...))$$

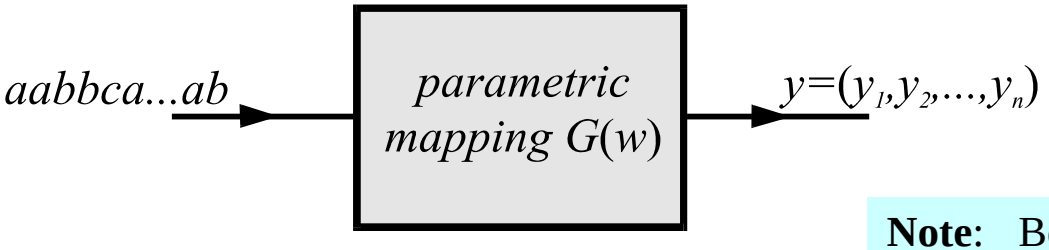
In general

$$(\alpha_1\alpha_2...\alpha_n)\in\{a,b,c,...\}^*$$
$$\downarrow$$
$$\boldsymbol{x}=(x_1x_2...x_n)\in\left(\{0,1\}^q\right)^n$$

Note: There exists many other ways how to code tokens from the vocabulary A by binary vectors of the fixed width.

A parametric mapping transforms strings of variable lengths onto real vectors of the fixed dimension and vice versa

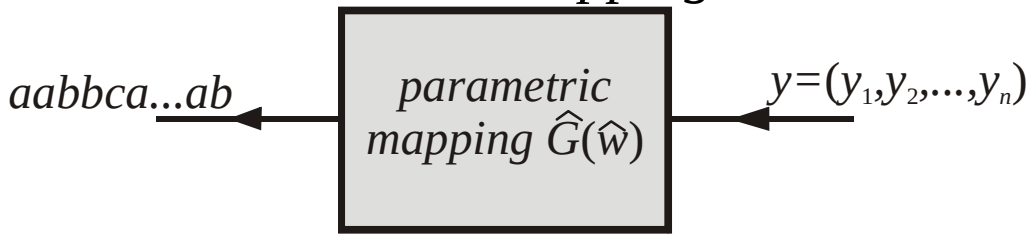
Direct mapping



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

Note: Both these mappings depends on the parameter *w*, a change of this parameter causes a change of mappings.

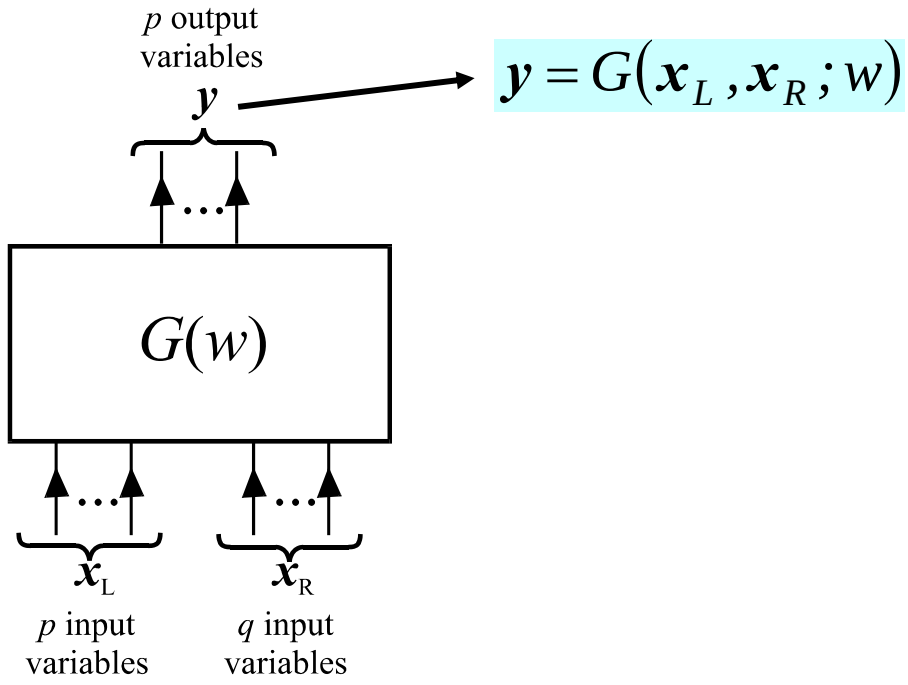
Inverse mapping



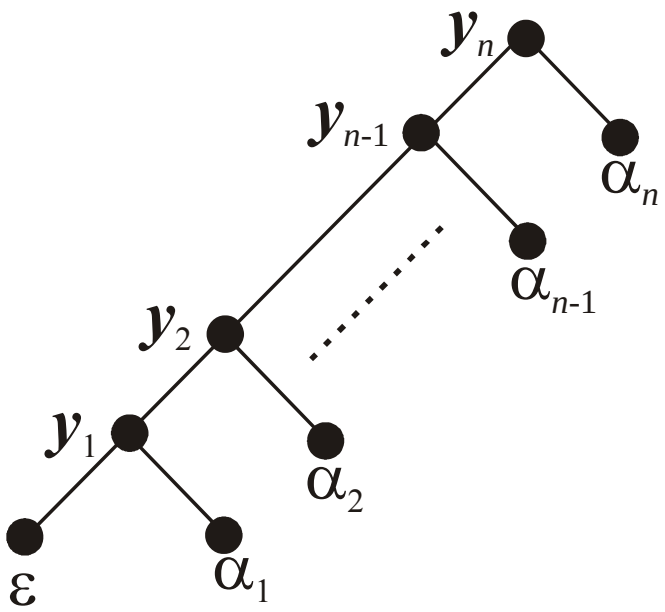
$$\hat{G}(\hat{w}): (0,1)^p \rightarrow A^* = \{a,b,c,...\}^*$$

A way of realization of the direct mapping (strings onto real vectors of fixed width)

Let us postulate of ring tokens are numerically coded by k-dimensional binary vectors. The parametric mapping $G(w)$ is specified by the 3-layer feed-forward neural network



A string $\alpha = (\alpha_1 \alpha_2 ... \alpha_n) \in A^*$ is recurrently coded as follows



$$y_1 = G(\varepsilon, \alpha_1; w)$$

$$y_2 = G(y_1, \alpha_2; w)$$

.....

$$y_n = G(y_{n-1}, \alpha_n; w)$$

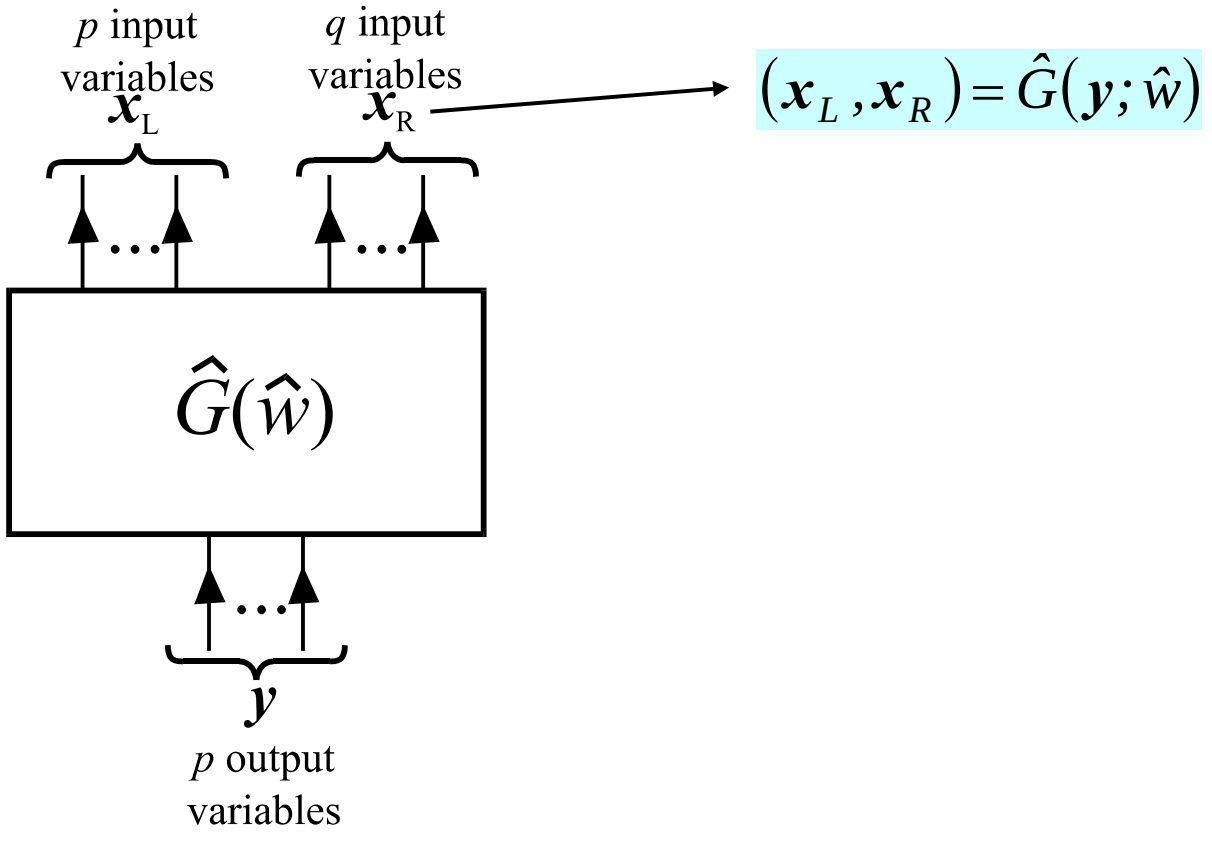
A final vector y_n represents a mapping of the string $\alpha = (\alpha_1 \alpha_2 ... \alpha_n) \in A^*$ by a p-dimensional real vector $y = (y_1, y_2, ..., y_p) \in (0,1)^p$

Empty token ε is coded by p-dimensional vector composed entirely of 0.5-entries

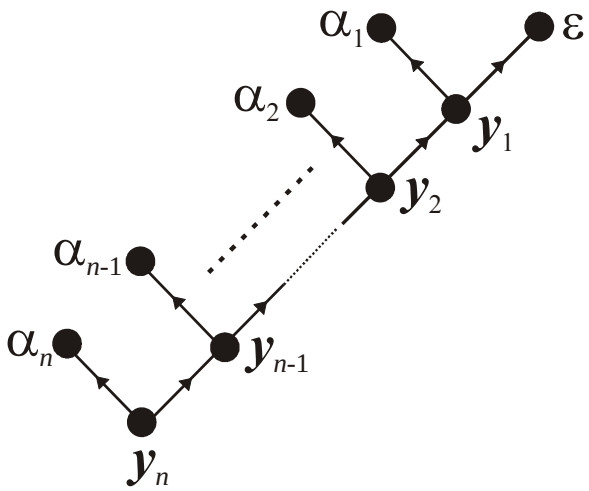
$$y_\varepsilon = (0.5, 0.5, ..., 0.5) \in (0,1)^p$$

A way of realization of the inverse mapping (real vectors of fixed width onto string)

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



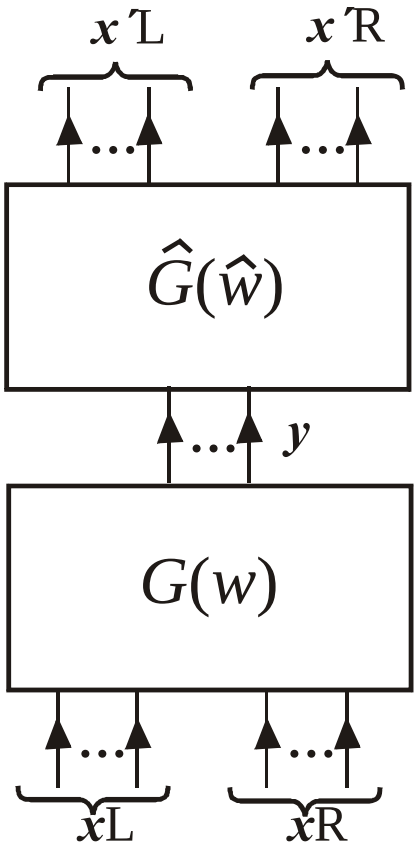
A code $\boldsymbol{y}=\boldsymbol{y}_n$ is recurrently decoded as follows



$$(\boldsymbol{y}_{n-1},\alpha_n)=\hat{G}(\boldsymbol{y}_n;\hat{w})$$
$$(\boldsymbol{y}_{n-2},\alpha_{n-1})=\hat{G}(\boldsymbol{y}_{n-1};\hat{w})$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$(\boldsymbol{y}_1,\alpha_2)=\hat{G}(\boldsymbol{y}_2;\hat{w})$$
$$(\varepsilon,\alpha_1)=\hat{G}(\boldsymbol{y}_1;\hat{w})$$

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

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


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

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

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

$$\downarrow$$

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

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*

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

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

Conclusion: A condition that a composite mapping $\hat{G} \circ G$ is auto-associative can be achieved by minimizing the objective function E with respect to its parameters.

This minimization problem is realized by the steepest-descent method (the simplest gradient method)

$$w^{(t+1)} = w^{(t)} - \lambda \left(\frac{\partial E}{\partial w} \right)^{(t)}$$

$$\hat{w}^{(t+1)} = \hat{w}^{(t)} - \lambda \left(\frac{\partial E}{\partial \hat{w}} \right)^{(t)}$$

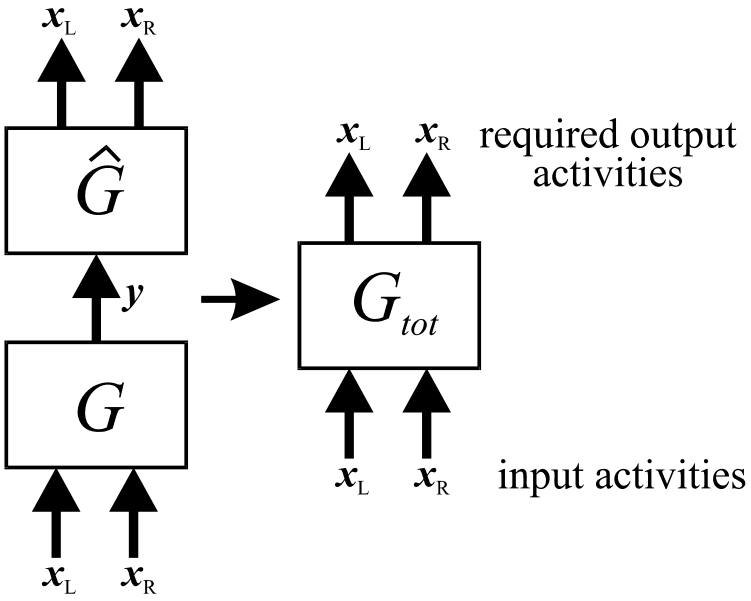
where λ is a small positive coefficient called the *learning rate*.

How to learn the auto-associative composite mapping $\hat{G} \circ G$?

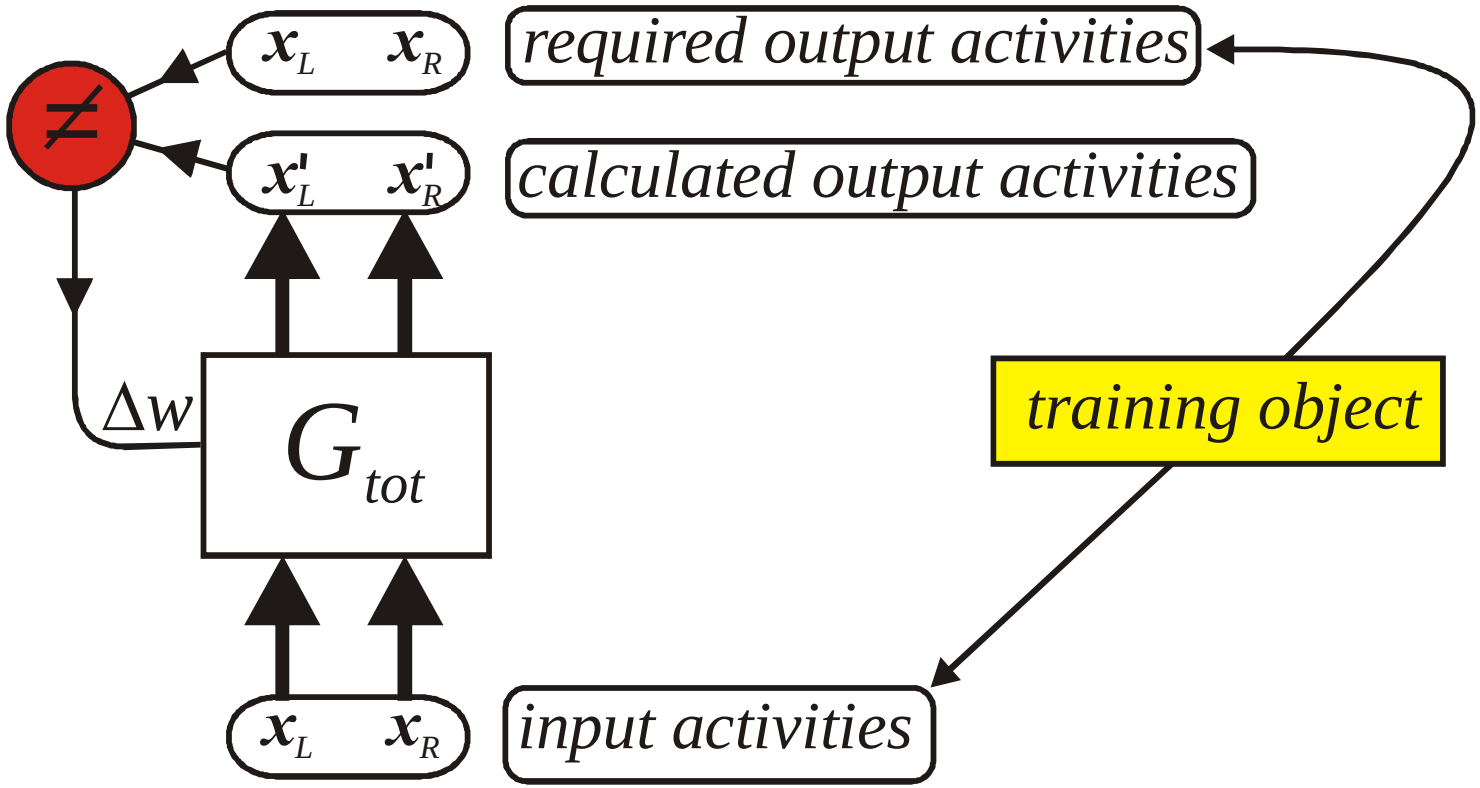
Let us introduce the notation

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

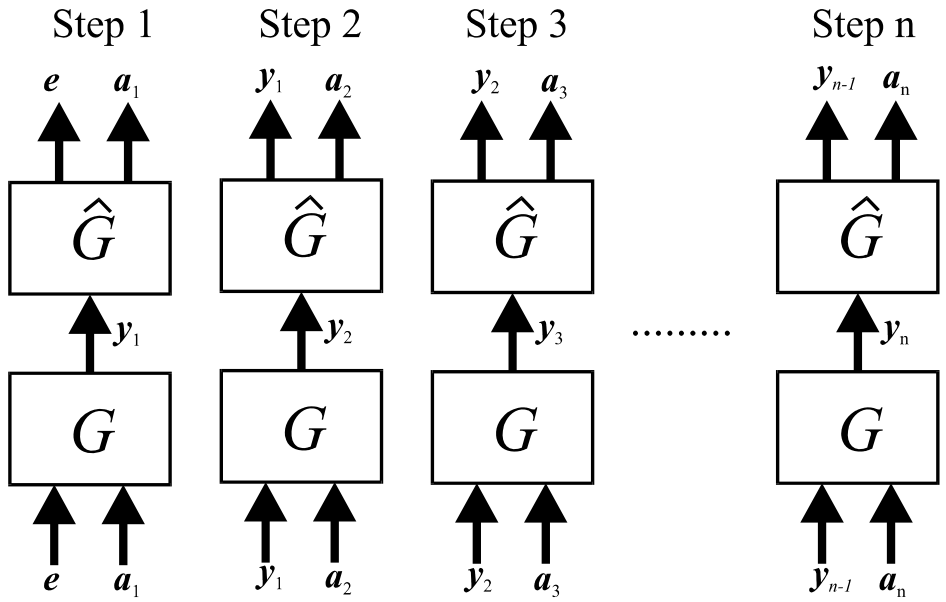
A string $\alpha = \left(a_1 a_2 ... a_n\right) \in A^*$ is used as a *training object* of the composite mapping $\hat{G} \circ G$



An outline of adaptation process



A string $\alpha = (a_1 a_2 ... a_n) \in A^*$ from the training set corresponds to *n* learning tasks (that are often characterized as a “moving target” problems)

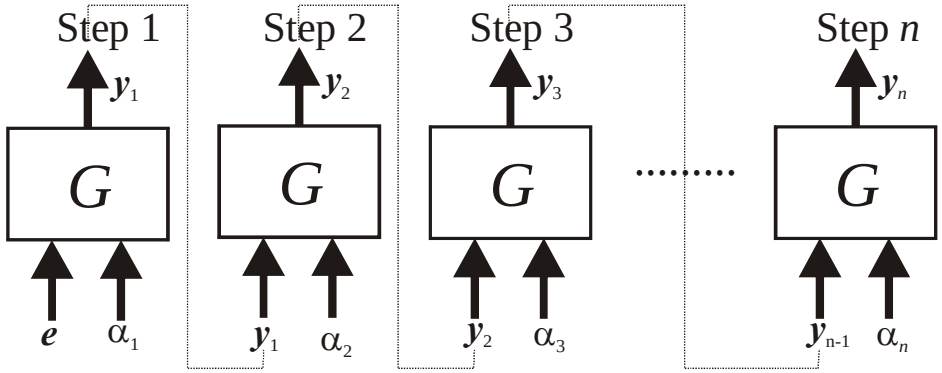


The last vector $\boldsymbol{y}_n$ is as a final output from the RAAM mapping G_{tot}. If this mapping is auto-associative (i.e. input and output are equal), then $\boldsymbol{y}_n$ is a vector code of the string α. Formally

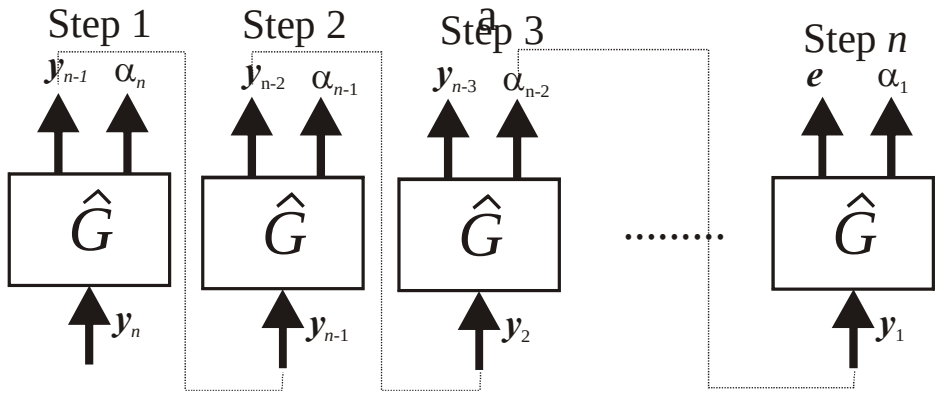
$$\boldsymbol{y} = \boldsymbol{y}_n = G_{tot}(\alpha; w, \hat{w})$$

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

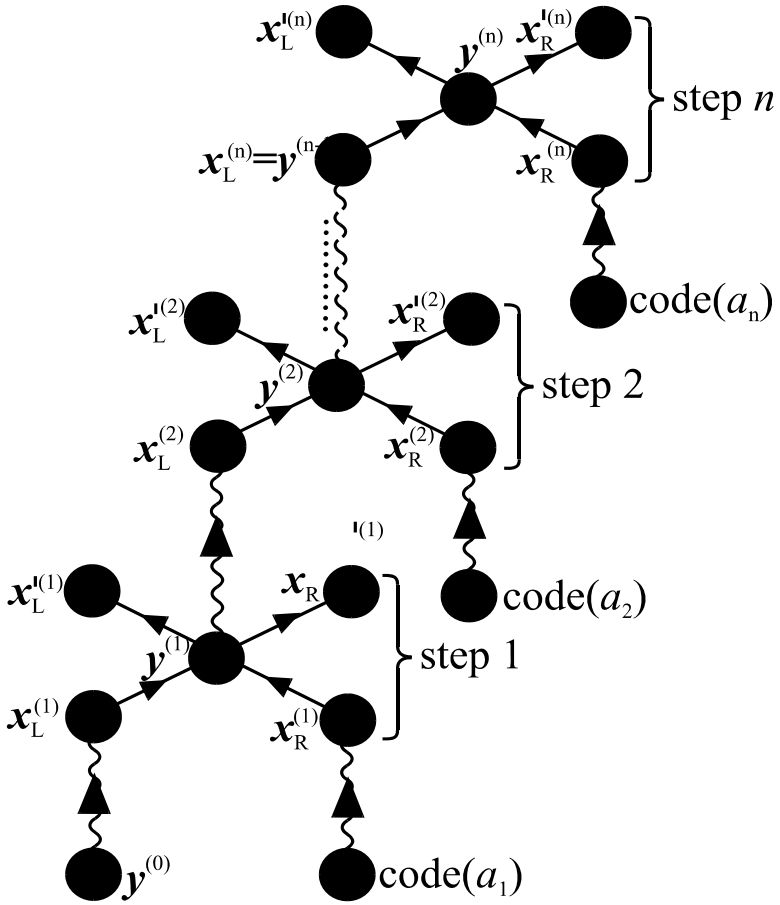
Coding of a string $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)\in A^*$ into a vector $\boldsymbol{y}=(y_1,y_2,\ldots,y_p)\in(0,1)^p$



Decoding of vector $\boldsymbol{y}=(y_1,y_2,\ldots,y_p)\in(0,1)^p$ into a string $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)\in A^*$



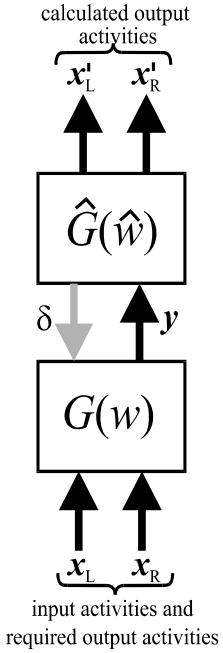
Recurrent construction of numerical code represented by an n-dimensional real vector $y \in (0,1)^p$



The resulting code is assigned to a string $\alpha=(a_1 a_2 \ldots a_n)$. This construction is composed of n successive steps, where the weight and threshold coefficients are updated when input activities $(y^{(i-1)}, \text{code}(a_i))$ are different from output activities $\left(x_L'^{(i)}, x_R'^{(i)}\right)$.

Waved vertical lines mean that activities of top vertices are set equal to activities of bottom vertices. The numerical code y assigned to the string α is determined by the resulting n-th vector of hidden activities, $\text{code}(\alpha)= y^{(n)}$.

How to calculate gradient of the objective function ?



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

Partial derivatives of E with respect to the parameters w and $\partial\hat{w}$ are determined as follows

$$\frac{\partial E}{\partial w} = \left(x'_L - x_L\right)\frac{\partial x'_L}{\partial w} + \left(x'_R - x_R\right)\frac{\partial x'_R}{\partial w}$$

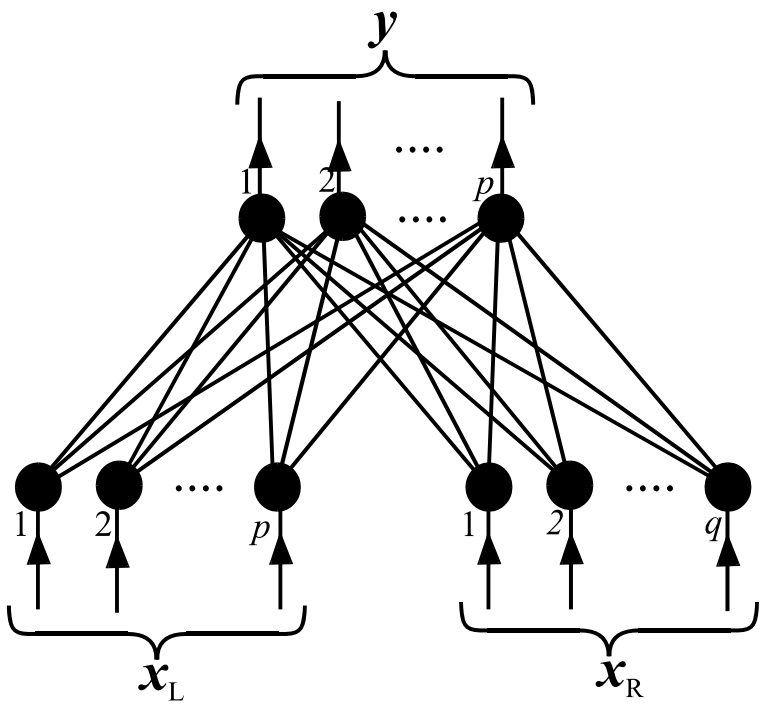
$$\frac{\partial E}{\partial \hat{w}} = \left(x'_L - x_L\right)\frac{\partial x'_L}{\partial y}\frac{\partial y}{\partial \hat{w}} + \left(x'_R - x_R\right)\frac{\partial x'_R}{\partial y}\frac{\partial y}{\partial \hat{w}}$$

$$= \left(x'_L - x_L\right)\delta_L\frac{\partial y}{\partial \hat{w}} + \left(x'_R - x_R\right)\delta_R\frac{\partial y}{\partial \hat{w}}$$

- Partial derivatives $\partial x'_L/\partial w$, $\partial x'_R/\partial w$, and $\partial y/\partial\hat{w}$ can be calculated *if we know functional forms* of dependencies of variables on parameters w and $\hat{w}$.
- The term δ corresponds to an *error injection* to the bottom mapping block $G(w)$ from the top mapping block $\hat{G}(\hat{w})$
- Partial derivatives are calculated in such a way that in the first stage are calculated those ones that correspond to the top mapping block and then in the second state those ones that correspond to the bottom mapping block – an analogue of the so-called *back propagation method*.

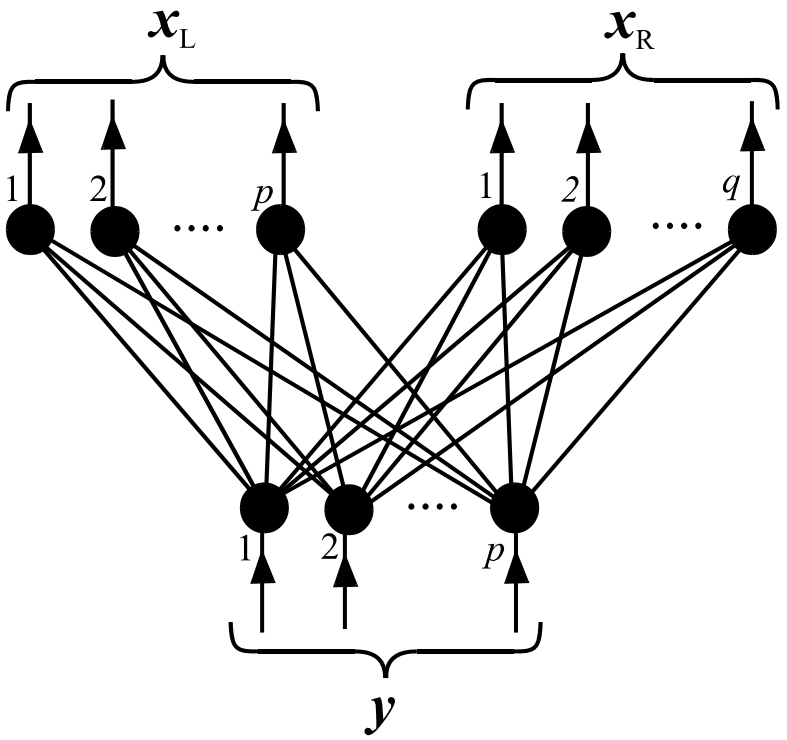
An actual realization of mappings $G(w)$ and $\hat{G}(\hat{w})$ by feed-forward neural networks composed of two layers

$G(w)$:



$$y_i = t\left(\vartheta_i + \sum_{j=1}^{p} w_{ij}^{(L)} x_j^{(L)} + \sum_{j=1}^{q} w_{ij}^{(R)} x_j^{(R)} \right), \text{ where } \quad t(x) = \frac{1}{1+e^{-x}}$$

$\hat{G}(\hat{w})$:

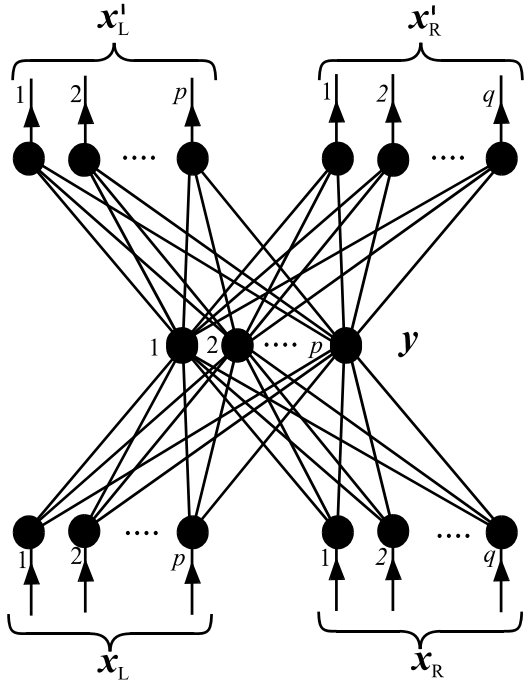


$$x_i^{(L)} = t\left(\vartheta_i^{(L)} + \sum_{j=1}^{p} \hat{w}_{ij}^{(L)} y_j \right)$$

$$x_i^{(R)} = t\left(\vartheta_i^{(R)} + \sum_{j=1}^{p} \hat{w}_{ij}^{(R)} y_j \right)$$

Unified network corresponding to composed mappings $\hat{G} \circ G$ is

$G_{tot}\left(w, \hat{w}\right)$:



feed-forward network: (p+q) - p - (p+q)(p+q)

$$x_i'^{(L)} = t\left(\vartheta_i^{(L)} + \sum_{j=1}^{p} \hat{w}_{ij}^{(L)} y_j \right)$$

$$x_i'^{(R)} = t\left(\vartheta_i^{(R)} + \sum_{j=1}^{p} \hat{w}_{ij}^{(R)} y_j \right)$$

$$y_i = t\left(\vartheta_i + \sum_{j=1}^{p} w_{ij}^{(L)} x_j^{(L)} + \sum_{j=1}^{q} w_{ij}^{(R)} x_j^{(R)} \right)$$

A back-propagation approach for construction of partial derivatives of the objective function

(1) Output neurons

$$\frac{\partial E}{\partial \vartheta_i^{(L)}} = \left(x_i'^{(L)} - x_i^{(L)}\right) \cdot x_i'^{(L)} \left(1 - x_i'^{(L)}\right)$$

$$\frac{\partial E}{\partial \vartheta_i^{(R)}} = \left(x_i'^{(R)} - x_i^{(R)}\right) \cdot x_i'^{(R)} \left(1 - x_i'^{(R)}\right)$$

$$\frac{\partial E}{\partial \hat{w}_{ij}^{(L)}} = \frac{\partial E}{\partial \vartheta_i^{(L)}} y_j \ , \ \frac{\partial E}{\partial \hat{w}_{ij}^{(R)}} = \frac{\partial E}{\partial \vartheta_i^{(R)}} y_j$$

(2) Hidden neurons

$$\frac{\partial E}{\partial \vartheta_i} = y_i\left(1-y_i\right)\left(\sum_{j=1}^{p}\frac{\partial E}{\partial \vartheta_j^{(L)}}\hat{w}_{ji}^{(L)} + \sum_{j=1}^{q}\frac{\partial E}{\partial \vartheta_j^{(R)}}\hat{w}_{ji}^{(R)}\right)$$

$$\frac{\partial E}{\partial w_{ij}^{(L)}} = \frac{\partial E}{\partial \vartheta_i}x_j^{(L)}\ ,\ \frac{\partial E}{\partial w_{ij}^{(R)}} = \frac{\partial E}{\partial \vartheta_i}x_j^{(R)}$$

How to update the neural-network coefficients ?

We use standard gradient *steepest-descent* optimization technique accelerated by the so-called *momentum method*

$$\vartheta_i^{(L)(t+1)} = \vartheta_i^{(L)(t)} - \lambda \left(\frac{\partial E}{\partial \vartheta_i^{(L)}} \right)^{(t)} + \mu \Delta \vartheta_i^{(L)(t)}$$

where $\Delta \vartheta_i^{(L)(t)} = \vartheta_i^{(L)(t)} - \vartheta_i^{(L)(t-1)}$ is a difference of $\vartheta_i^{(L)}$ from the last two iterations.

Other network coefficients are updated by the similar way

$$\vartheta_i^{(R)(t+1)} = \vartheta_i^{(R)(t)} - \lambda\left(\frac{\partial E}{\partial \vartheta_i^{(R)}}\right)^{(t)} + \mu\Delta\vartheta_i^{(R)(t)}$$

$$\hat{w}_{ij}^{(L)(t+1)} = \hat{w}_{ij}^{(L)(t)} - \lambda\left(\frac{\partial E}{\partial \hat{w}_{ij}^{(L)}}\right)^{(t)} + \mu\Delta\hat{w}_{ij}^{(L)(t)}$$

$$\hat{w}_{ij}^{(R)(t+1)} = \hat{w}_{ij}^{(R)(t)} - \lambda\left(\frac{\partial E}{\partial \hat{w}_{ij}^{(R)}}\right)^{(t)} + \mu\Delta\hat{w}_{ij}^{(R)(t)}$$

Similarly, network parameters for hidden neurons are updated by

$$\vartheta_i^{(t+1)} = \vartheta_i^{(t)} - \lambda \left(\frac{\partial E}{\partial \vartheta_i} \right)^{(t)} + \mu \Delta \vartheta_i^{(t)}$$

$$w_{ij}^{(L)(t+1)} = w_{ij}^{(L)(t)} - \lambda \left(\frac{\partial E}{\partial w_{ij}^{(L)}} \right)^{(t)} + \mu \Delta w_{ij}^{(L)(t)}$$

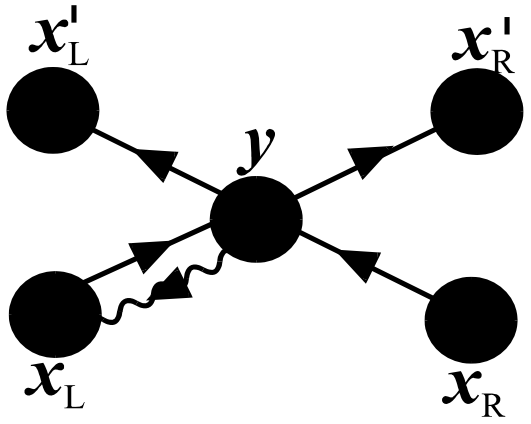
$$w_{ij}^{(R)(t+1)} = w_{ij}^{(R)(t)} - \lambda \left(\frac{\partial E}{\partial w_{ij}^{(R)}} \right)^{(t)} + \mu \Delta w_{ij}^{(R)(t)}$$

Note: The momentum term in above update formulae helps to overcome local minima in the initial stage of optimization, in general, it integrates the effects of previous update steps. The momentum term μ is usually taken from

$$0.3 \leq \mu < 1.0$$

The optimization process is started by randomly generated network coefficients from the interval [-1,1].

RAAM networks applied to coding of linear token strings may be interpreted as a special type of recurrent neural networks.



RAAM network treated as a recurrent neural network. A loop between left input neurons and hidden neurons is created. The waved line means that activities of left input neurons are set equal to activities of hidden neurons.

Activities of single neurons are formally determined by

$$\boldsymbol{x}'_L = O_L\left(\boldsymbol{y}\right),\ \boldsymbol{x}'_R = O_R\left(\boldsymbol{y}\right)$$

$$\boldsymbol{y} = H\left(\boldsymbol{x}_L, \boldsymbol{x}_R\right),\ \boldsymbol{x}_L = \boldsymbol{y}$$

Activities $\boldsymbol{y}$ of hidden neurons are determined by a nonlinear relation

$$\boldsymbol{y} = H\left(\boldsymbol{y}, \boldsymbol{x}_R\right)$$

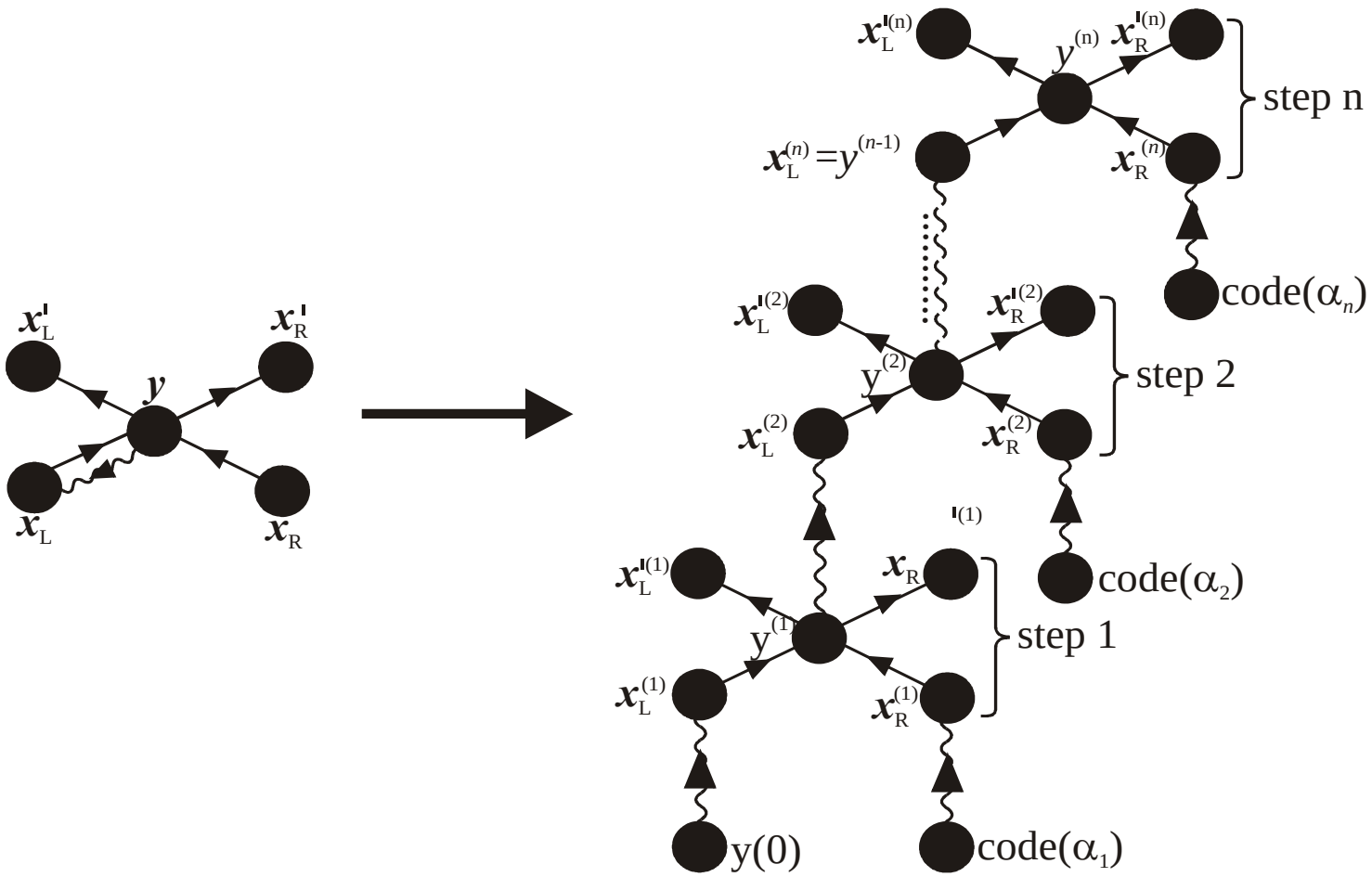
If is is solved iteratively, stating from solution $\boldsymbol{y}^{(0)}$ we get the following recurrent scheme

$$\boldsymbol{x}'^{(t)}_L = O_L\left(\boldsymbol{y}^{(t)}\right),\ \boldsymbol{x}'^{(t)}_R = O_R\left(\boldsymbol{y}^{(t)}\right)$$

$$\boldsymbol{y}^{(t)} = H\left(\boldsymbol{y}^{(t-1)}, \boldsymbol{x}^{(t)}_R\right),$$

for t=1,2,… . If this recurrent is diagrammatically visualized, then we immediatelyget a scheme already presented in transparency 18

Unfolded RAAM neural network



The End