WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS
AND INFORMATION SCIENCE

Neural Networks

Lecture 6

Neural network for Classification

Neural Network for Classification

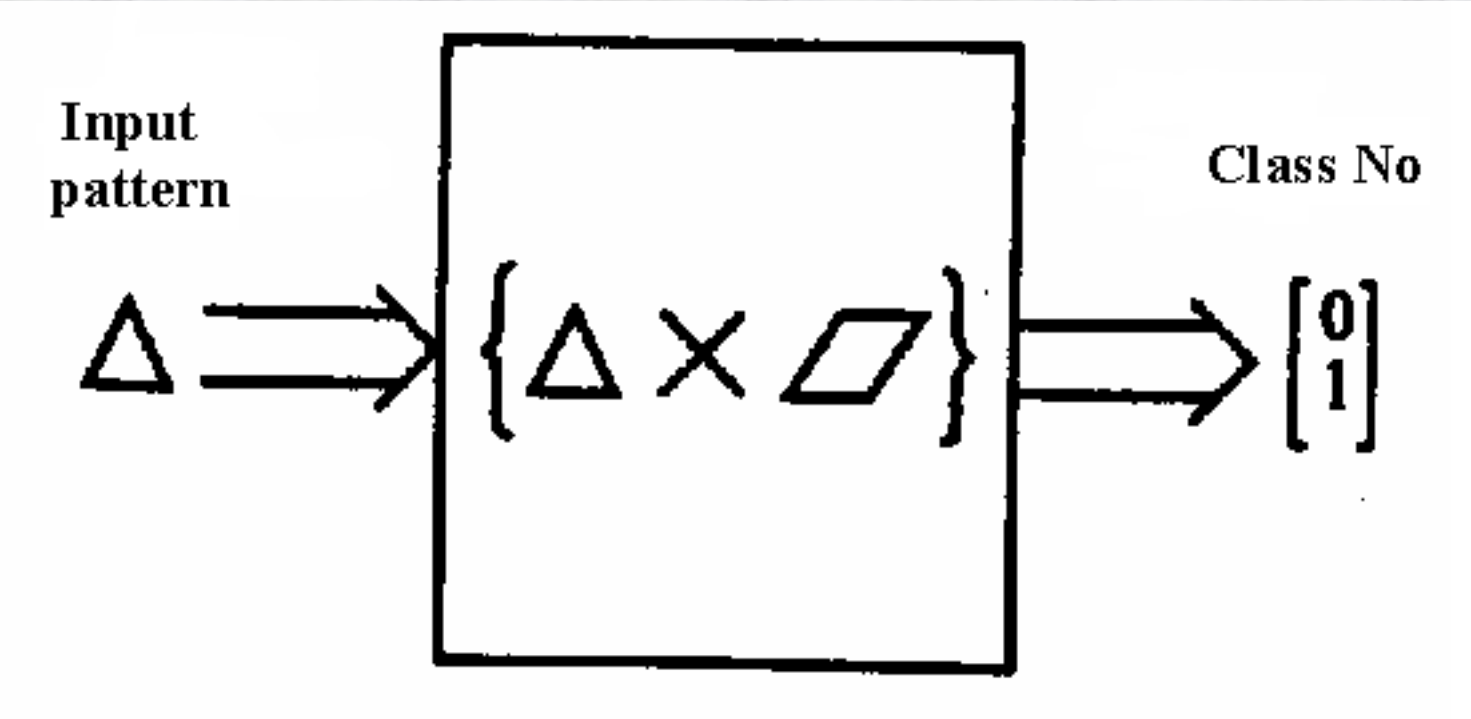
The fundamental objective for pattern recognition is ***classification*** and it is the most typical form of neural transformation. A pattern recognition system can be treated as a two stage device: **feature extraction** and **classification**. A feature is defined as a measurement taken on the input pattern that is to be classified but the classifier has to map the input features onto a classification state. The classifier must to decide which type of class category they match most closely.

Neural Network for Classification

Any given input pattern must belong to one of the classes that are in consideration. So, the mapping from the input to the required class must exists.

This mapping is a function that transforms the input pattern into the correct output class, and we will consider that our network has learnt to perform correctly, if it can carry out this mapping.

Neural Network for Classification

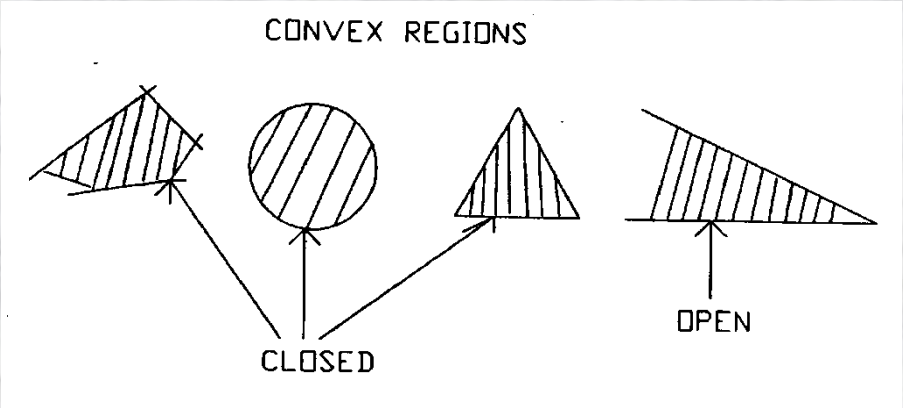


Neural Network for Classification

We spoke about the multilayer perceptron limitations. When the problem of linear separability was well understood it was also found that the problem of single-layer network can be overcome by adding more layers. For example, the two layer network may be formed by cascading two single-layer networks. These can perform more general classification, separating those points that are contained in convex open or closed regions.

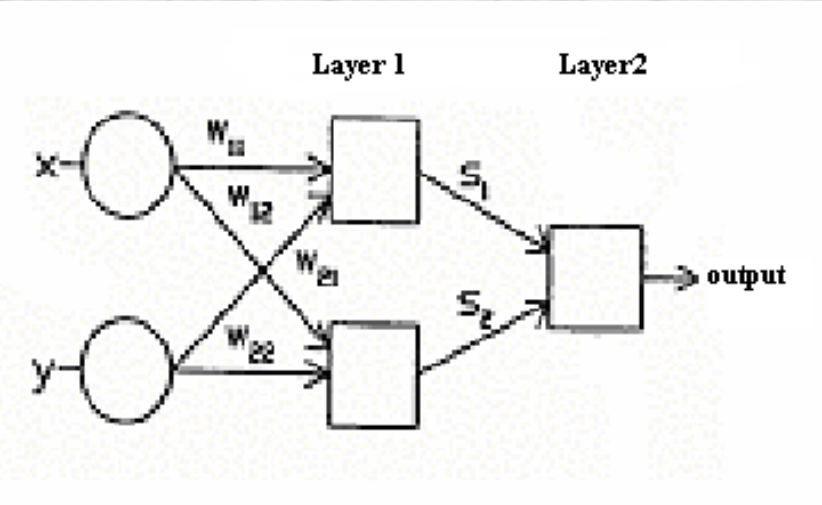
A convex regions

A convex region – is one in which any two points in the region can be joined by a straight line that does not leave the region.



Neural Network for Classification

To understand the convexity limitation, consider a simple two-layer network with two inputs going to two neurons in the first layer, both feeding a single neuron in the layer 2. The output neuron threshold is set to 0.75 and weights s_i are both set to 0.5



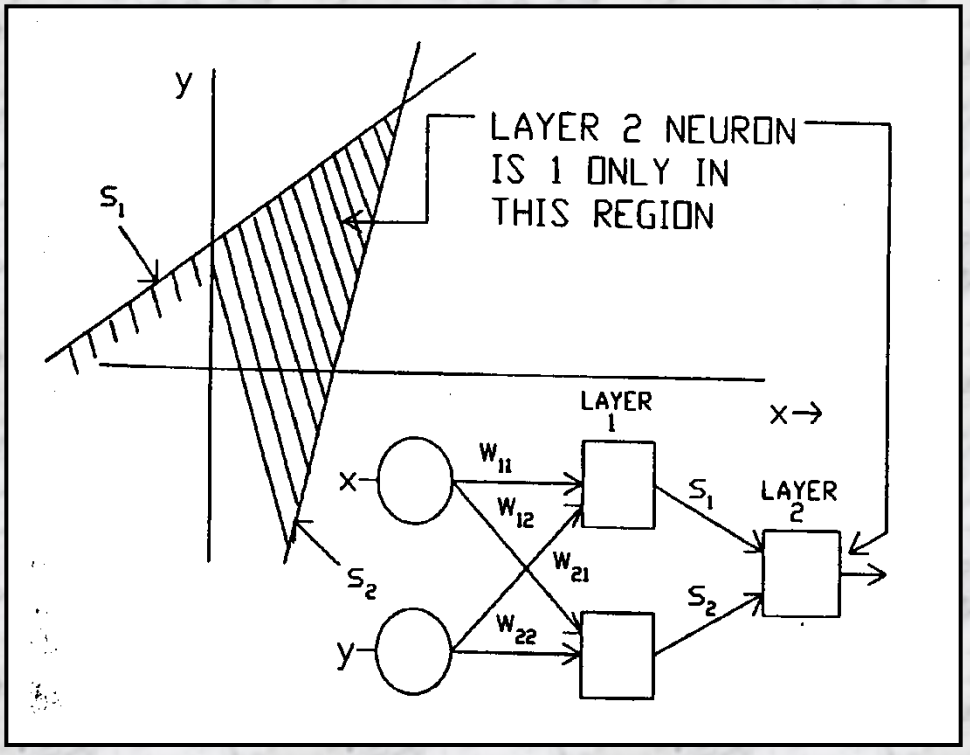
Neural Network for Classification

In this case the output of one (**1**) is required from both layer 1 neurons to exceed the threshold and to produce a one (**1**) on the output. Thus the output neuron performs a logical **AND** function.

Each neuron in layer 1 subdivides the **OXY** plane , producing tan output of one (**1**) on one side of the line.

Neural Network for Classification

The result of double subdivision when the output of one (**1**) of the layer 2 neuron is only over V-shaped region.



Neural Network for Classification

Similarly, three neurons used in the layer 1 further subdivide the plane, creating for example a triangle-shaped region. By including enough neurons in the layer 1, a convex region of any desired shape can be formed.

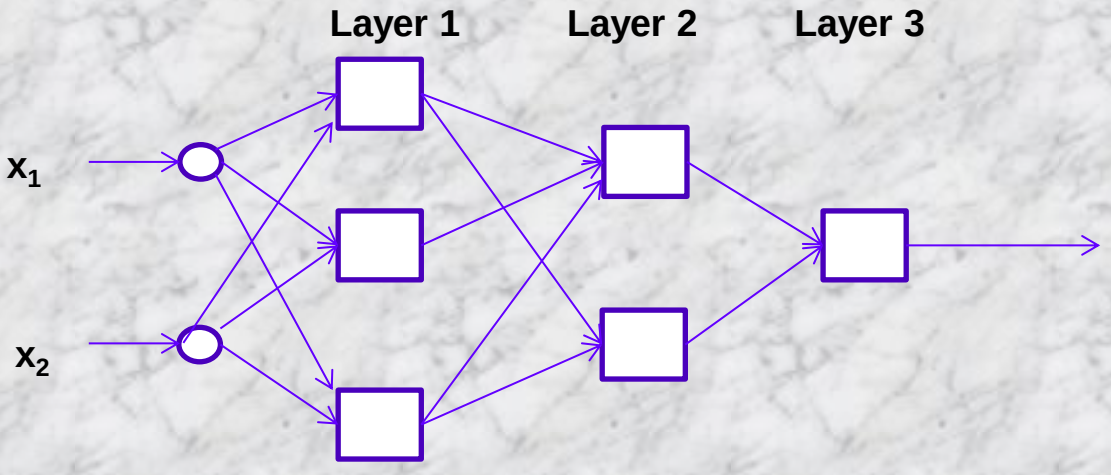
The layer 2 of course is not limited to the AND function, it can produce many other functions if the weights and threshold are suitably chosen.

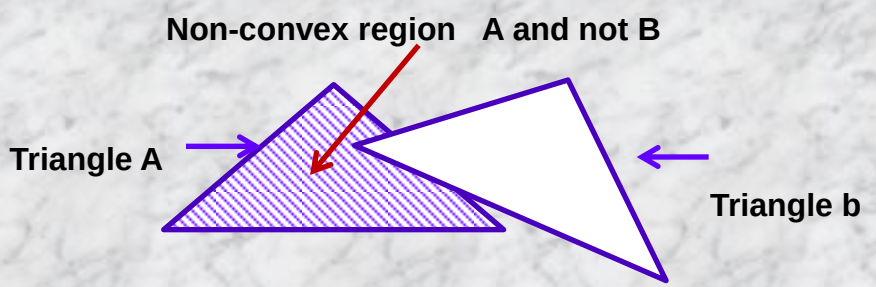
Neural Network for Classification

A three-layer network is still more general, its limitation capability is limited only by the number of neurons and weights. There are no convexity constrains; the layer 3 neuron receives as input a group of convex polygons, and the logical combination of that need not to be convex.

Neural Network for Classification

A concave decision region formed by intersection of two convex regions. Logical operation **A and not B**





Neural Network for Classification

Existence (Kolmogorov) Theorem

Any continuous function of n variables can be computed using only linear summations and nonlinear but continuously increasing functions of only one variable. It effectively states that a three layer perceptron with n(2n+I) nodes using continuously increasing non-linearities can compute any continuous function of n variables. A three layer perceptron could then be used to create any continuos likelihood function required in a classifier.

Neural Network for Classification

Conclusion:

To create any arbitrary complex shape (decision region), we never need more that three layers in the network.

It gives the limitation on layers but does not define:

- how many elements is necessary to create a network (in general and in particular layers),
- how these elements should be connected,
- which weights value should be.

Network structure

Inconsistency in nomenclature

What is layer???

- some authors refer to the number of layers of variable weights
- some authors describe the number of layers of nodes

Usually, the nodes in the first layer, the input layer, merely distribute the inputs to subsequent layers, and do not perform and operations (summation or thresholding) n.b. some authors miss out these nodes.

Network structure

What is a network layer?

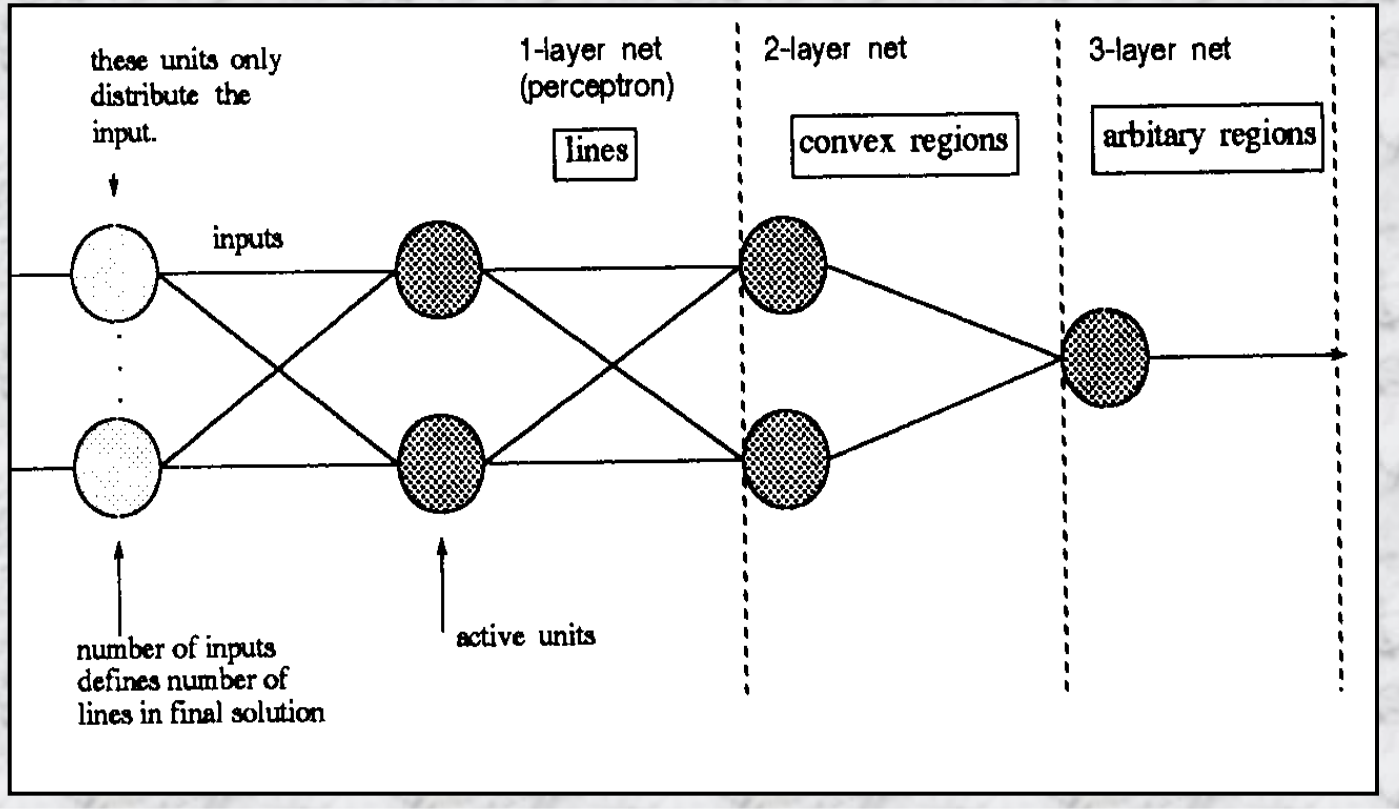
A layer - it is the part of network structure which contains active elements performing some operation. A multilayer network receives a number of inputs. These are distributed by a layer of input nodes that do not perform any operation – these inputs are then passed along the first layer of adaptive weights to a layer of perceptron-like units, which do sum and threshold their inputs. This layer is able to produce classification lines in pattern space.

Neural Network for Classification

The output from this layer in then passed to another layer, and the output of this layer forms convex regions in pattern space. A further layer is able to define any arbitrary shape in pattern space.

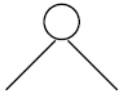
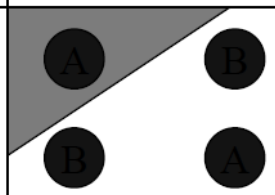
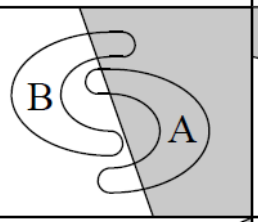
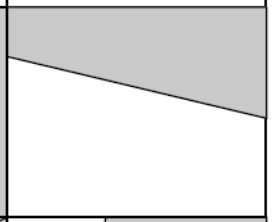
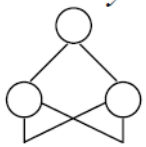
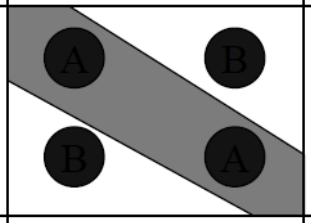
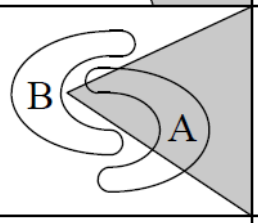
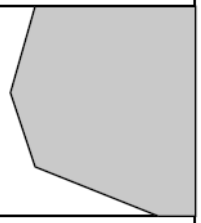
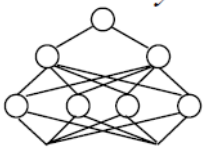
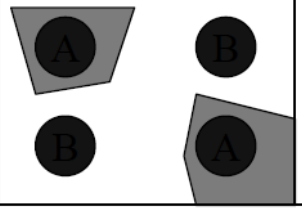
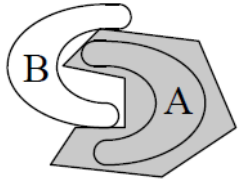
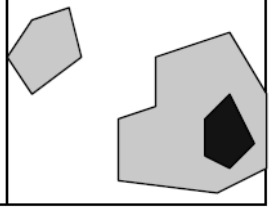
Neural Network for Classification

Neural networks and their corresponding decision regions



Neural Network for Classification

Different non linearly separable problems and number of layers

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Abitrary (Complexity Limited by No. of Nodes)			

Neural Networks – An Introduction Dr. Andrew Hunter

Neural Network for Classification

A multilayer perceptron is **fault-tolerant**, since it is distributed parallel processing element, with each node contributing to the final output.

If a node or its weights are lost or damaged. recall is impaired in quality, but the distributed nature of the information means that the damage has to be extensive before the network's response degrades badly. The network demonstrate graceful degradation rather that catastrophic failure.

Backpropagation Algorithm

Backpropagation

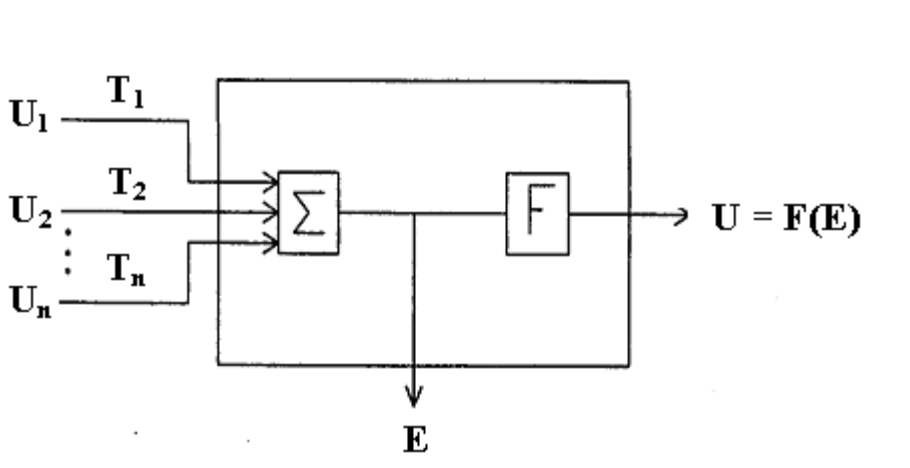
For many years there was no theoretical sound algorithm for training multilayer artificial neural networks. Since single-layer networks proved severely limited in what they could represent, the entire field went into virtual eclipse.

In 1986 Rumelhart and McClelland suggested a new learning rule known as ***a backpropagation rule*** which is today used in many practical applications like for example in solving the optimization problems.

Backpropagation

The invention of the backpropagation algorithm has played a large part in the resurgence of interest in artificial neural networks.

Backpropagation is a systematic method for training multilayer artificial neural networks.



Backpropagation

It is the rule how to change the weights $\mathbf{T_{ij}}$ between network elements.

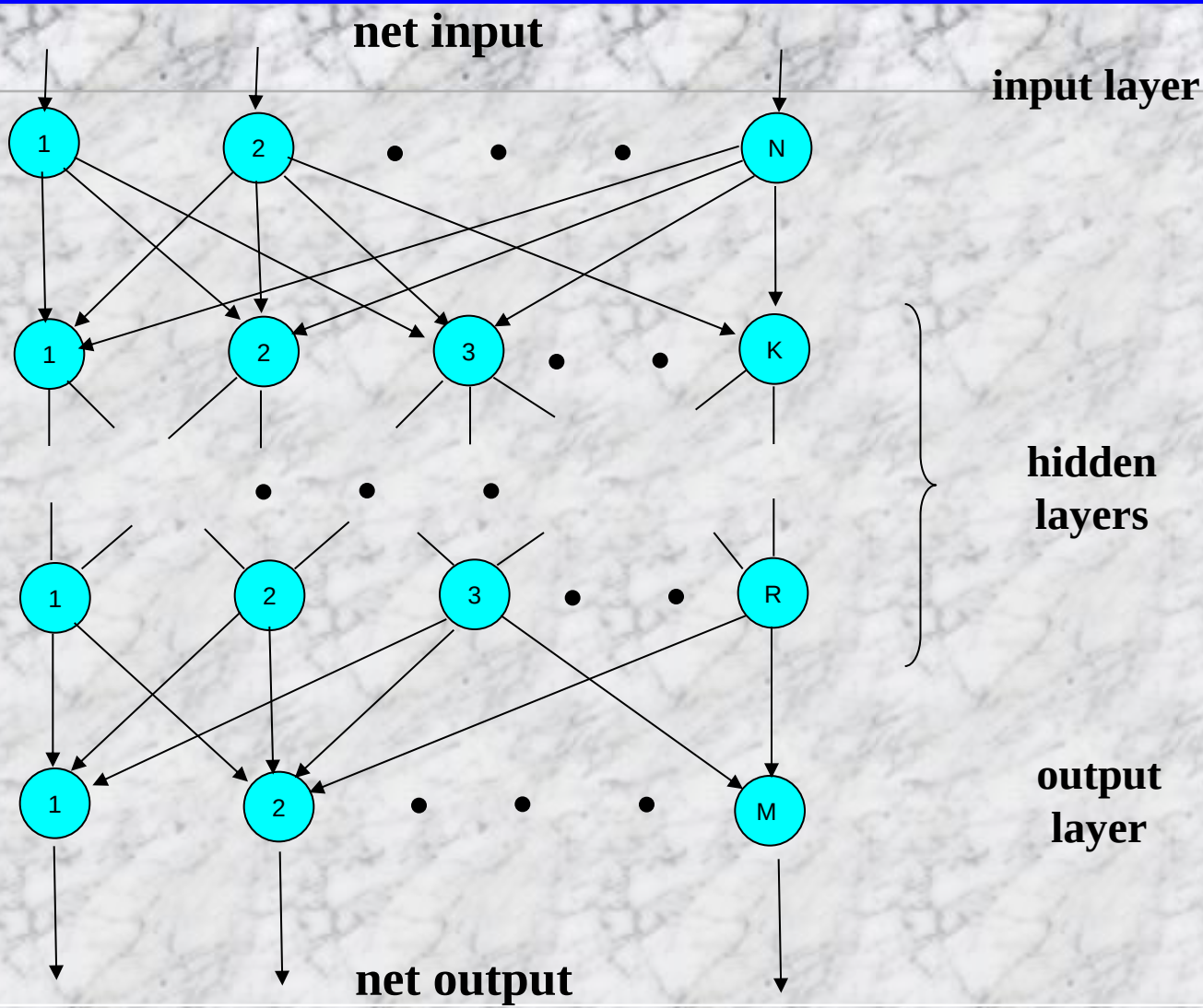
The algorithm is based on the idea to minimize the square root of errors by use of the gradient descent method.

Backpropagation

Assumptions:

- the net is the regular, multilayer structure
- the first layer – input layer
- the last layer – output layer
- layers between – hidden layers
- feed forward propagation only

Backpropagation



Backpropagation

To define a state of *j*-th neuron in layer *n* we calculate the weighted sum of its **M** inputs

$$\mathbf{E}_{\mathbf{j}}^{\mathbf{n}} = \sum_{\mathbf{i=1}}^{\mathbf{M}} \mathbf{T}_{\mathbf{j,i}}^{\mathbf{n}} \mathbf{U}_{\mathbf{i}}^{\mathbf{n-1}} \qquad \textbf{(1)}$$

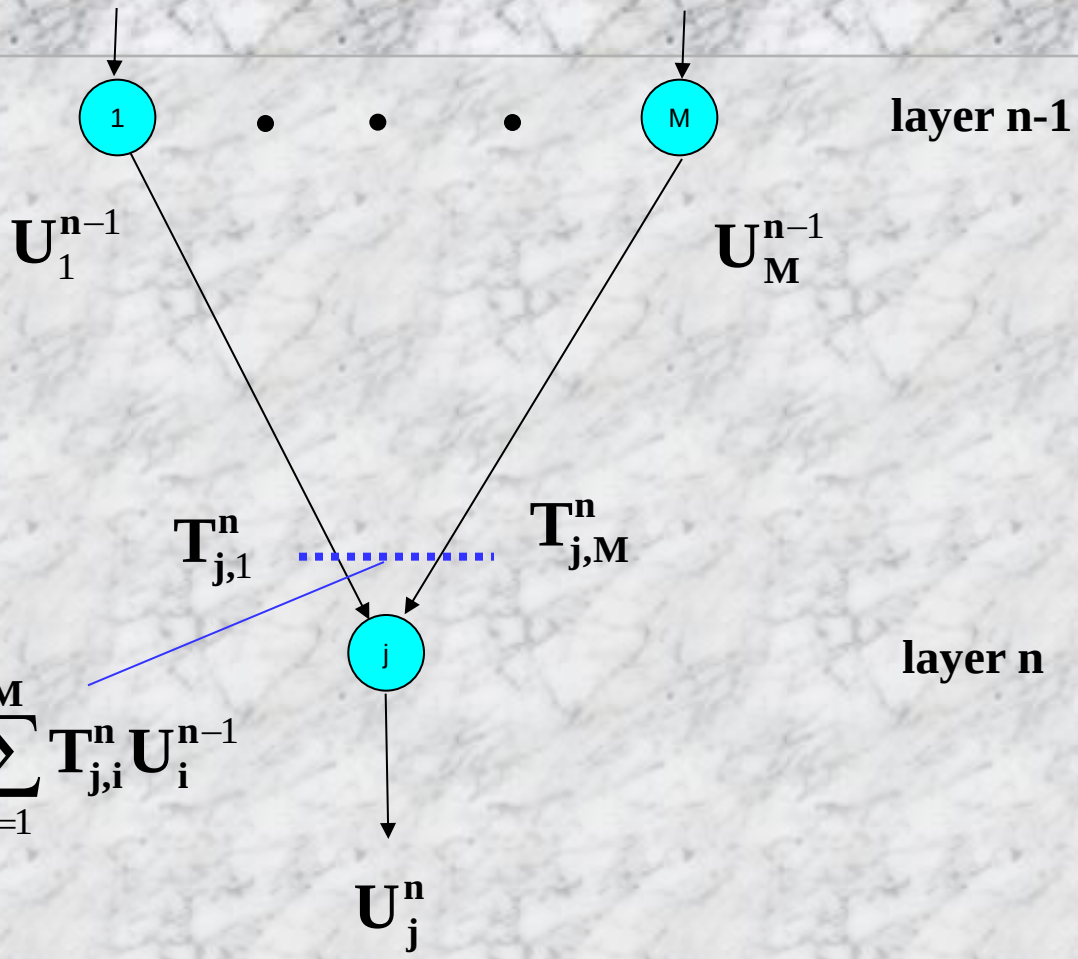
where

$\mathbf{E}_{\mathbf{j}}^{\mathbf{n}}$ weighted input sum of *j*-th neuron in layer *n*

$\mathbf{T}_{\mathbf{j,i}}^{\mathbf{n}}$ connection weight between *i*-th neuron inlayer *n*-1 and *j*-th neuron in layer *n*

$\mathbf{U}_{\mathbf{i}}^{\mathbf{n-1}}$ out put signal of *i*-th neuron in layer *n*-1

Backpropagation



Backpropagation

output signal of *j*-th neuron in layer *n* is defined by

$$\mathbf{U_j^n} = f(\mathbf{E_j^n}) \qquad \textbf{(2)}$$

where *f* is the neuron transfer function

$$\mathbf{U_j^n} = f(\mathbf{E_j^n}) = \frac{1}{1 + exp[-(\mathbf{E_j^n} - \Theta_j^n)/\Theta_0]} \qquad \textbf{(3)}$$

f – sigmoid function, the output values $\in$ (0;1),

Θ_j^n - the threshold, Θ_0 - slope

Backpropagation

The global error **D** is a differentiable function of weights

$$D = \frac{1}{2} \cdot \sum_{j=1}^{M} (U_j^* - U_j^{out})^2 \qquad \textbf{(4)}$$

where $\mathbf{U_j^*}$ desired output (a target) of *j*-th neuron in the output layer

U_j^{out} actual output of *j*-th neuron in the output layer

Backpropagation

Aim: minimization of a global error **D**, modifying the network's weights.

Goal: define the rules of the weights adaptation to minimize the global error

Solution: the gradient descent method

$$\Delta \mathbf{T}_{\mathbf{j,i}}^{\mathbf{n}} = -\eta \frac{\partial \mathbf{D}}{\partial \mathbf{T}_{\mathbf{j,i}}^{\mathbf{n}}} \qquad \mathbf{(5)}$$

where η is the learning coefficient

Backpropagation

Each weight is changed according to the value and direction of negative gradient on the hyperplane **D(T)**.

The partial derivative in (5) can be calculated by use of the chain rule.

Backpropagation

$$\frac{\partial D}{\partial T_{j,i}^{n}} = \frac{\partial D}{\partial E_{j}^{n}} \cdot \frac{\partial E_{j}^{n}}{\partial T_{j,i}^{n}} \qquad (6)$$

We define the change in error as a function of the change in the net inputs to a *j*-th neuron as

$$d_{j}^{n} = -\frac{\partial D}{\partial E_{j}^{n}}$$

Backpropagation

$$\frac{\partial E_j^n}{\partial T_{j,i}^n} = -\frac{\partial}{\partial T_{j,i}^n} \sum_{k=1}^{M} T_{j,k}^n U_k^{n-1} = \sum_{k=1}^{M} \frac{\partial T_{j,k}^n}{\partial T_{j,i}^n} U_k^{n-1} = U_i^{n-1} \qquad (7)$$

because

$$\frac{\partial T_{j,k}^n}{\partial T_{j,i}^n} = \begin{cases} 0 \text{ for } k \neq i \\ 1 \text{ for } k = i \end{cases}$$

Backpropagation

Finally, we get

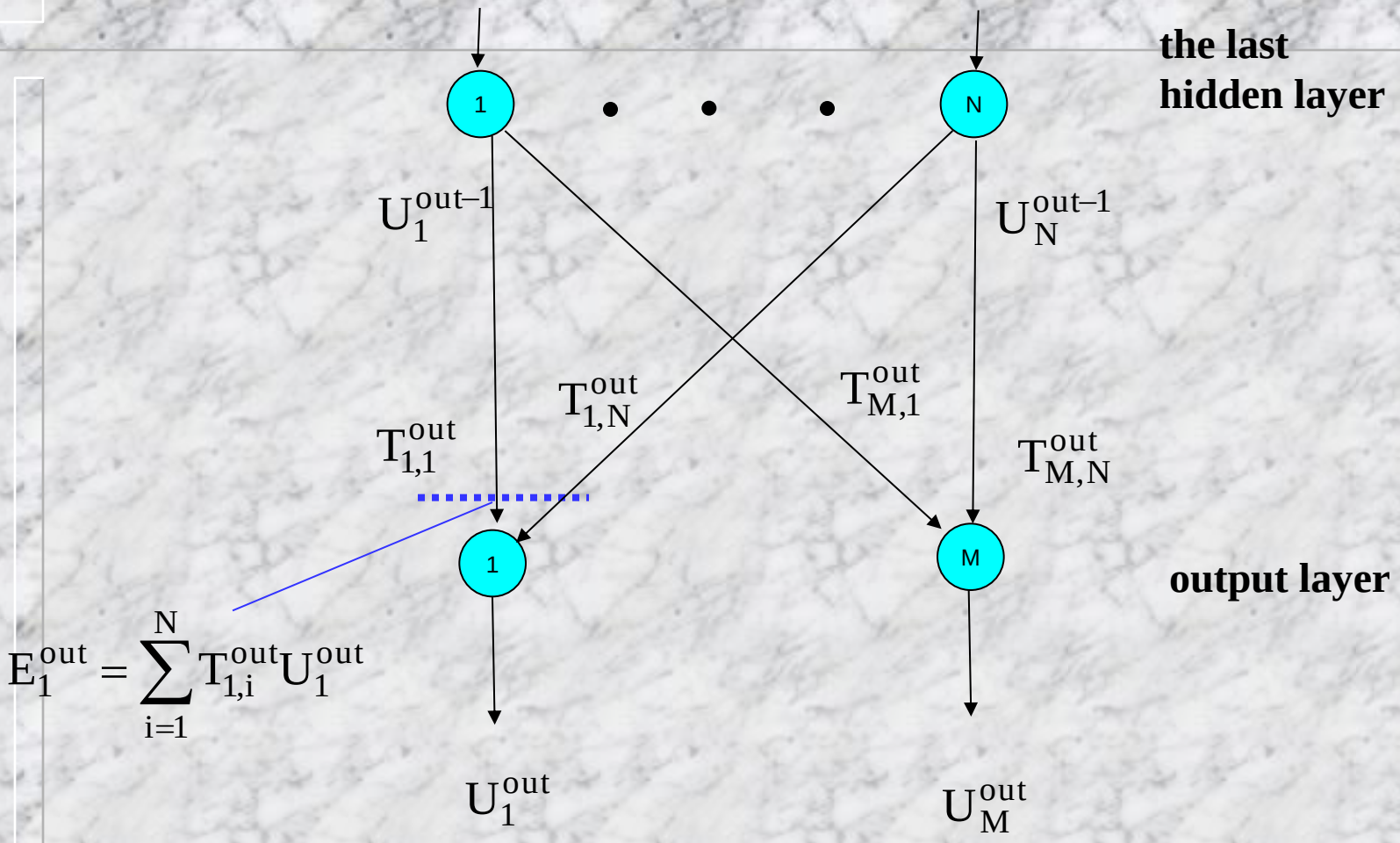
$$\Delta \mathbf{T}_{j,i}^{n} = \eta \mathbf{d}_{j}^{n} \mathbf{U}_{i}^{n-1} \qquad \textbf{(8)}$$

For the elements in the output layer

$$\Delta \mathbf{T}_{j,i}^{out} = \eta \mathbf{d}_{j}^{out} \mathbf{U}_{i}^{out-1} \qquad \textbf{(9)}$$

Decrease of **D** means the changes proportional to $\mathbf{d}_{j}^{out} \mathbf{U}_{i}^{out-1}$

Backpropagation



Backpropagation

Now, we need to know what $\mathbf{d_j^{out}}$ is for each of the units

$$\mathbf{d_j^{out}} = -\frac{\partial \mathbf{D}}{\partial \mathbf{E_j^{out}}} = \frac{\partial \mathbf{D}}{\partial \mathbf{U_j^{out}}} \frac{\partial \mathbf{U_j^{out}}}{\partial \mathbf{E_j^{out}}} \qquad \mathbf{(10)}$$

where

$$\frac{\partial \mathbf{U_j^{out}}}{\partial \mathbf{E_j^{out}}} = \mathrm{f}'(\mathbf{E_j^{out}})$$

Backpropagation

differentiate $\mathbf{D}$ with respect to $\mathbf{U_j^{out}}$ giving

$$\frac{\partial \mathbf{D}}{\partial \mathbf{U_j^{out}}} = -(\mathbf{U_j^{*}} - \mathbf{U_j^{out}}) \qquad \textbf{(11)}$$

thus

$$\mathbf{d_j^{out}} = (\mathbf{U_j^{*}} - \mathbf{U_j^{out}}) \cdot \mathrm{f}'(\mathbf{E_j^{out}}) \qquad \textbf{(12)}$$

Backpropagation

This is useful for the output units to modify weights $T_{j,i}^{out}$ between the neurons of the output layer and the neurons of the last hidden layer (since the target and output both are available)

$$T_{j,i}^{out}(t) = T_{j,i}^{out}(t-1) + \tau \Delta T_{j,i}^{out}$$

where τ is a learning coefficient (13)

Backpropagation

The formula (9) can be rewritten to avoid oscillations

$$\Delta T_{j,i}^{out}(t) = \alpha \Delta T_{j,i}^{out}(t-1) + (1-\alpha) d_j^{out} f(E_i^{out-1}) \quad (14)$$

where E_j^{wy-1} is the weighted input sum of *i*-th element from the last hidden layer, α (so called smoothing parameter), is a constant value defining the effect of the previous weights modification on the actual modification .

Backpropagation

This method is very useful for the output layer elements because we have access both to the output signal, target signal.

However, for the elements located in the hidden layers unfortunately does not work.

Backpropagation

If *j*-th neuron **does not belong** to the output layer but is an element of a hidden layer *s*, then its local weight modification is calculated from

$$\mathbf{d}_j^s = -\frac{\partial \mathbf{D}}{\partial \mathbf{E}_j^s} \qquad (15)$$

Because

$$\frac{\partial \mathbf{D}}{\partial \mathbf{E}_j^n} = \frac{\partial \mathbf{D}}{\partial \mathbf{U}_j^n} \cdot \frac{\partial \mathbf{U}_j^n}{\partial \mathbf{E}_j^n} = f'(\mathbf{E}_j^n) \frac{\partial \mathbf{D}}{\partial \mathbf{U}_j^n} \qquad (16)$$

Backpropagation

also

$$\frac{\partial \mathbf{D}}{\partial \mathbf{U}_j^n} = \sum_{k=1}^{N} \frac{\partial \mathbf{D}}{\partial \mathbf{E}_k^{n+1}} \frac{\partial \mathbf{E}_k^{n+1}}{\partial \mathbf{U}_j^n} = \sum_{k=1}^{N} \left[\frac{\partial \mathbf{D}}{\partial \mathbf{E}_k^{n+1}} \frac{\partial}{\partial \mathbf{U}_j^n} \left(\sum_{i=1}^{M} \mathbf{T}_{k,i}^{n+1} \mathbf{U}_i^n \right) \right] =$$

$$= \sum_{k=1}^{N} \frac{\partial \mathbf{D}}{\partial \mathbf{E}_k^{n+1}} \mathbf{T}_{k,j}^{n+1} = \sum_{k=1}^{N} \mathbf{d}_k^{n+1} \mathbf{T}_{k,j}^{n+1} \qquad \textbf{(17)}$$

then

$$\mathbf{d}_j^n = -\frac{\partial \mathbf{D}}{\partial \mathbf{E}_j^n} = f'(\mathbf{E}_j^n) \sum_{k=1}^{N} \mathbf{d}_j^{n+1} \mathbf{T}_{k,j}^{n+1} \qquad \textbf{(18)}$$

Backpropagation

it modify the weight $\mathbf{T}_{j,i}^{n}$ between *i*-th element of the layer *n*-1 and *j*-th element of the layer *n*

$$\mathbf{T}_{j,i}^{n}(t) = \mathbf{T}_{j,i}^{n}(t-1) + \tau \ \Delta\mathbf{T}_{j,i}^{n} \qquad (19)$$

a

$$\Delta\mathbf{T}_{j,i}^{n}(t) = \alpha\Delta\mathbf{T}_{j,i}^{n}(t-1) + (1-\alpha)\,\mathbf{d}_{j}^{n}\, f(\mathbf{E}_{i}^{n-1}) \qquad (20)$$

Iterative calculations correct weight „backwards”, to the input layer.

Backpropagation

This learning procedure is repeated for each learning pattern, until the network will generate the correct answer for every input signal (pattern) – of course with the defined accuracy.

Backpropagation

Example:

If $$f(\mathbf{E}) = \frac{1}{1 + exp(-k\mathbf{E})}$$

$f(\mathbf{E}) \in (0;1)$, $\mathbf{k>0}$

when $\mathbf{k} \Rightarrow \infty$

then $f(\mathbf{E}) \Rightarrow$ step function

$$\mathbf{U_j^{out}} = \mathrm{f}(\mathbf{E_j^{out}}) = \frac{1}{1 + \exp(-\mathrm{k}\mathbf{E_j^{out}})}$$

Backpropagation

calculating the derivative $f'(\mathbf{E})$ for j-th element:

$$f'(\mathbf{E}) = \frac{k\mathbf{e}^{-k\mathbf{E}}}{(1+\mathbf{e}^{-k\mathbf{E}})^2} = kf(\mathbf{E})\cdot[1-f(\mathbf{E})] =$$

$$= k\mathbf{U}_j^{out}(1-\mathbf{U}_j^{out})$$

(this derivative simplify he calculations)

Backpropagation

for the elements in the output layer

$$\mathbf{d_j^{out}} = (\mathbf{U_j^*} - \mathbf{U_j^{out}}) \cdot \mathrm{f}'(\mathbf{E_j^{out}}) =$$

$$= \mathrm{k} \cdot \mathbf{U_j^{out}} \cdot (\mathbf{1} - \mathbf{U_j^{out}}) \cdot (\mathbf{U_j^*} - \mathbf{U_j^{out}})$$

for the elements in the hidden layers

$$\mathbf{d_j^n} = k \cdot \mathbf{U_j^n} \cdot (\mathbf{1} - \mathbf{U_j^n}) \cdot \sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{d_k^{n+1}} \mathbf{T_{j,k}^{n+1}}$$

Recurrent MultiLayer Perceptron (RMLP)

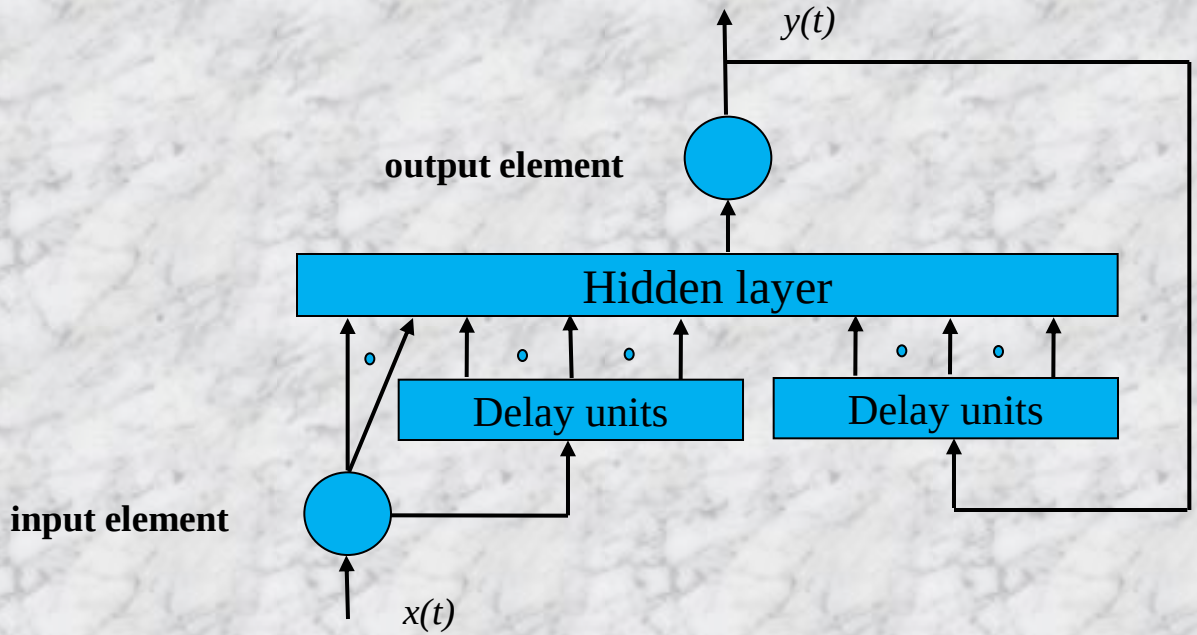
RMLP

Recurrent neural network architectures consists of a standard Multi-Layer Perceptron (MLP) plus added loops.

RMLP

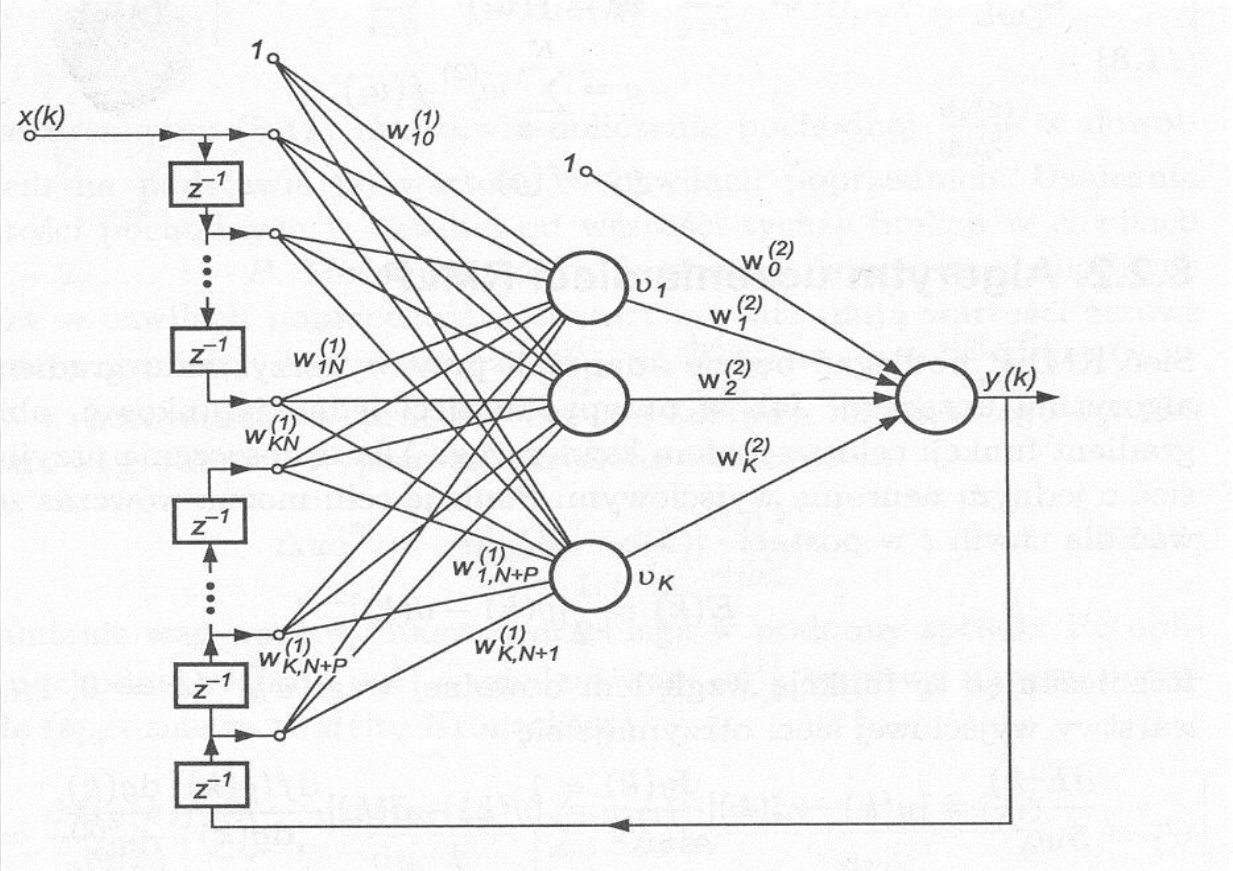
Assumption: only one input nod (input signal *x(t)*) and one output nod (output *signal y(t)*), and one hidden layer.

The network input signal is composed from input *x*, delayed inputs and external recurrence output signal *y(t)* with some unit time delays.



RMLP

Structure of the RMLP network



RMLP

RMLP is a dynamic network with delayed input and output signals.

The farther analysis is performed for only one input node (signal *x(t)*), one output node (signal *y(t)*) and one hidden layer.

The function

$$y(t+1)=f(x(t),x(t-1),\ldots,x(t-(N-1)),y(t-1),y(t-2),\ldots,y(t-P))$$

where

- $N-1$ ⇨ number of delayed input signals,
- P ⇨ number of delayed output signals,
- K ⇨ number of neurons in the hidden layer,

RMLP

Vector input signal ***x*** applied to the network input in the time *t*

x(*t*)=[1,*x*(*t*),*x*(*t*-1), …, *x*(*t*-(*N*-1)),*y*(*t*-*P*),*y*(*t*-*P*+1),…,*y*(*t*-1)]

Denote

$$u_i = \sum_{j=0}^{N+P} w_{ij}^{(1)} x_j$$

weighted input signal of the *i*-th neuron of the hidden layer

$$v_i = f(u_i)$$

input signal of the *i*-th neuron of the hidden layer

$$g = \sum_{i=0}^{K} w_i^{(2)} f(u_j)$$

weighted input signal of the output neuron

$$y = f(g)$$

output signal of the output neuron

(1-4)

RMLP

RMLP Learning Algorythm

The gradient learning algorithm is used. The gradient of ab error function with respect to each networks' weight is calculated. For RMLP network the error function can be define by:

$$E(t) = \frac{1}{2}[y(t) - d(t)]^2$$

Differentiating with respect to each weight $w_{\alpha}^{(2)}$ in the second layer

RMLP

$$\frac{\partial E(t)}{\partial w_\alpha^{(2)}} = [y(t) - d(t)] \frac{dy(t)}{dw_\alpha^{(2)}} = [y(t) - d(t)] \frac{df(g(t))}{dg(t)} \frac{dg(t)}{dw_\alpha^{(2)}} =$$

$$= [y(t) - d(t)] \frac{df(g(t))}{dg(t)} \sum_{i=0}^{K} \frac{d(w_\alpha^{(2)} v_i(t))}{dw_\alpha^{(2)}} =$$

$$= [y(t) - d(t)] \frac{df(g(t))}{dg(t)} \left[v_\alpha(t) + \sum_{i=0}^{K} w_\alpha^{(2)} \frac{dv_i(t)}{dw_\alpha^{(2)}} \right]$$

because

$$\frac{d(w_\alpha^{(2)})}{dw_\alpha^{(2)}} = \begin{Bmatrix} 1 \ \text{for i} = \alpha \\ 0 \ \text{for i} \neq \alpha \end{Bmatrix}$$

RMLP

$$\frac{dv_i(t)}{dw_\alpha^{(2)}} = \frac{df(u_i(t))}{du_i(t)} \sum_{j=0}^{N+P} w_{ij}^{(1)} \frac{dx_j}{dw_\alpha^{(2)}} =$$

$$= \frac{df(u_i(t))}{du_i(t)} \sum_{j=N+1}^{N+P} w_{ij}^{(1)} \frac{dy(t-P-1+(j-N))}{dw_\alpha^{(2)}} =$$

$$= \frac{df(u_i(t))}{du_i(t)} \sum_{j=0}^{P} w_{ij}^{(1)} \frac{dy(t-P-1+j)}{dw_\alpha^{(2)}}$$

RMLP

next

$$\frac{dy(t)}{dw_{\alpha}^{(2)}} =$$

$$= \frac{df(g(t))}{dg(t)} \left[v_{\alpha}(t) + \sum_{i=0}^{K} w_i^{(2)} \frac{df(u_i(t))}{du_i(t)} \sum_{j=1}^{P} w_{i,j+N}^{(1)} \frac{dy(t-P-1+j)}{dw_{\alpha}^{(2)}} \right]$$

this formula enable calculater the $\frac{dy(t)}{dw_{\alpha}^{(2)}}$ in the epoch t on the base of its previus values

RMLP

Using the gradient descent method the weight change of the output layer is defined by

$$\Delta w_{\alpha}^{(2)} = -\eta[y(t) - d(t)]\frac{dy(t)}{dw_{\alpha}^{(2)}} \qquad (5)$$

Similarly can be calculated the weight change in the hidden layer. $w_{\alpha,\beta}^{(1)}$

After calualtion of the derivative of a signal *y(t)* with respect of the weight ijn the hidden layer

RMLP

$$\frac{dy(t)}{dw_{\alpha,b}^{(1)}} = \frac{df(g(t))}{dg(t)} \sum_{i=0}^{K} w_i^{(2)} \frac{df(u_i(t))}{du_i(t)} \left[\sum_{j=1}^{P} w_{i,j+N}^{(1)} \frac{dy(t-P-1+j)}{dw_{\alpha}^{(2)}} \right]$$

and the formula for the weight $w_{\alpha,\beta}^{(1)}$ change in the hidden layer

$$\Delta w_{\alpha,\beta}^{(1)} = -\eta[y(t) - d(t)] \frac{dy(t)}{dw_{\alpha,\beta}^{(1)}} \qquad (6)$$

RMLP

Summary of a learning algorithm

1. Initialize the weight vectors in the hidden layer and output layer
2. Evaluate the neuron state in epoch (t) with input x (1-4)
3. Calculates the values of $\frac{dy(t)}{dw_{\alpha}^{(2)}}$ and $\frac{dy(t)}{dw_{\alpha\beta}^{(1)}}$ for every $\alpha\beta$
4. Updates the weights according to 5 i 6
5. Go to step 2 algorithm.

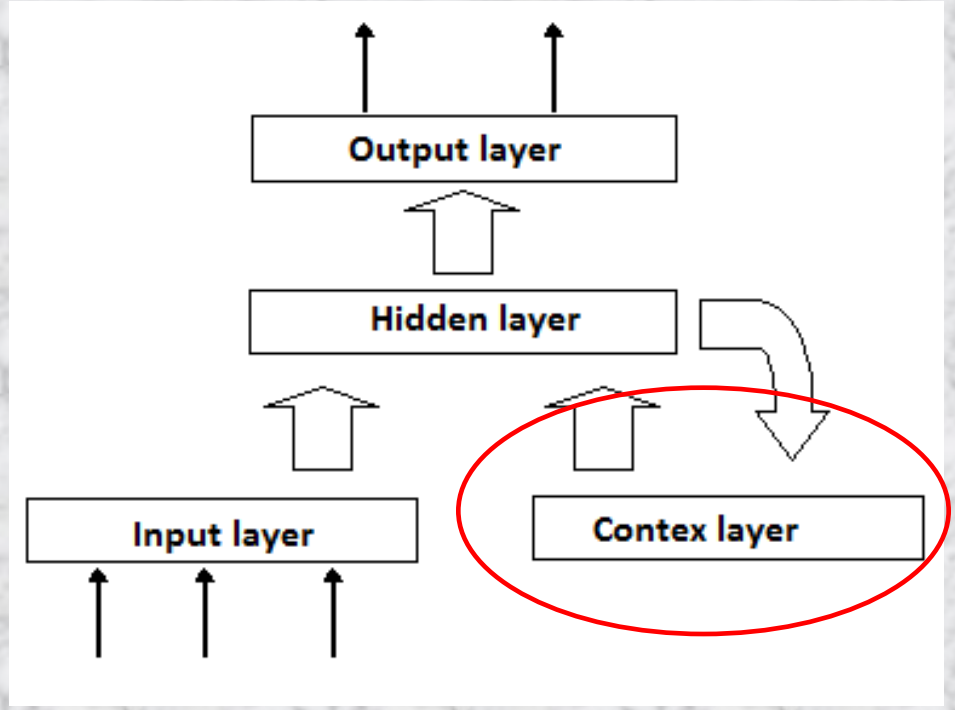
Elman network

Elman Network

An Elman network is an MLP with a single hidden layer and in addition it contains connections from the hidden layer's neurons to the ***context units***. The context units store the output values from the hidden neurons in a time unit and these values are fed as additional inputs to the hidden neurons in the next time unit.

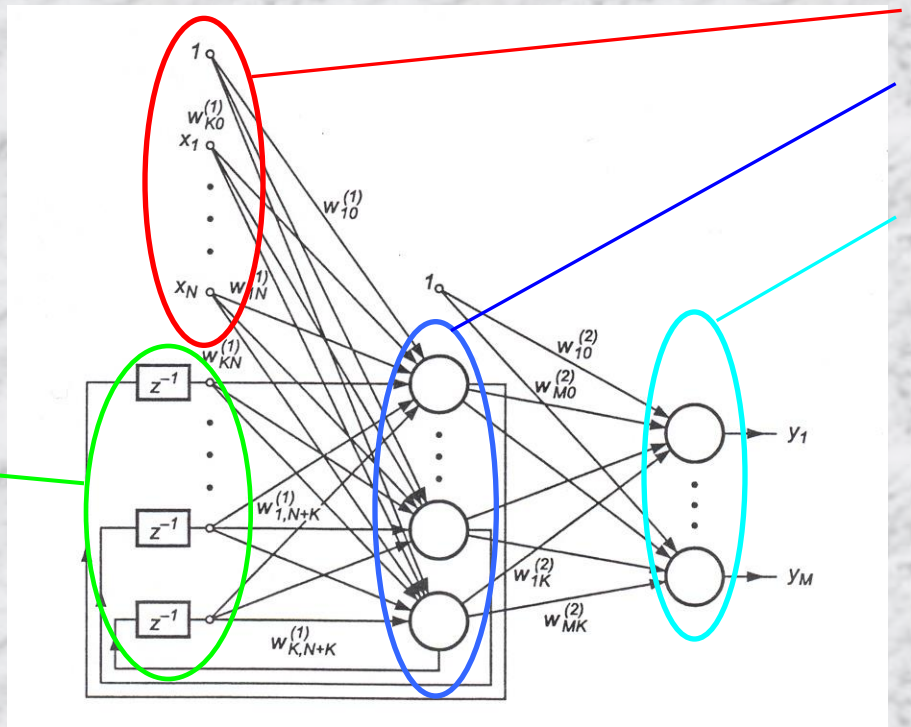
Elman Network

Elman Network – SRN (Simple Recurrent Network) is a simplification of Multi Layer Perceptron



Elman Network

A structure of an Elman Network



Contex layer

N- number of external inputs

K – number of neurons in a hidden layer

M – number of neurons in an output layer

Elman Network

Vector input signal:

$$[1, x_1^{(1)}(t),\ldots, x_N^{(1)}(t), x_{N+1}^{(1)}(t),\ldots, x_{N+K}^{(1)}(t)]$$

the elements denoted $j=N+1,..,N+K$ are from the hidden layer from the previous epoch

$$[1, x_1^{(1)}(t),\ldots, x_N^{(1)}(t), y_1^{(1)}(t-1),\ldots, y_K^{(1)}(t-1)]$$

Elman Network

Notation

$$u_i(t) = \sum_{j=0}^{N+K} w_{ij}^{(1)} x_j(t)$$

weighted input signal of the *i*-th neuron in a hidden layer

$$v_i = f_1(u_i(t))$$

output signal of the *i*-th neuron of a hidden layer

$$g_i(t) = \sum_{j=0}^{K} w_{ij}^{(2)} v_j(t)$$

weighted input signal of the *i*-th neuron of the output layer

$$y_i = f_2(g_i(t))$$

output signal of the *i*-th neuron of the output layer

Elman Network

Elman learning algorithm

The network will be learn according to steepest descent algorithm. Similarly to feedforward network a gradient of the cost function will be calculated with respect to every network`s weight. For the Elman network a cost function can be defined by

$$E(t) = \frac{1}{2}\sum_{i=1}^{M}[y_i(t) - d_i(t)]^2 = \frac{1}{2}\sum_{i=1}^{M}[e_i(t)]^2$$

Differentiating this function with respect to any output layer weight $w_{\alpha\beta}^{(2)}$ we obtain

Elman Network

$$\nabla^{(2)}_{\alpha\beta} E(t) = \frac{\partial E(t)}{\partial w^{(2)}_{\alpha\beta}} = \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \frac{dg_i(t)}{dw^{(2)}_{\alpha\beta}} =$$

$$= \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \sum_{j=0}^{K} \frac{d(w^{(2)}_{ij} v_j(t))}{dw^{(2)}_{\alpha\beta}} =$$

$$= \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \sum_{j=0}^{K} \left[\frac{d(v_j(t)}{dw^{(2)}_{\alpha\beta}} w^{(2)}_{ij} + \frac{d(w^{(2)}_{ij}(t)}{dw^{(2)}_{\alpha\beta}} v_j(t) \right]$$

Because connections between the hidden layer and output layer are unidirectional

$$\frac{d(v_j(t))}{dw^{(2)}_{\alpha\beta}} = 0$$

Elman Network

yields to

$$\nabla^{(2)}_{\alpha\beta} E(t) = \frac{\partial E(t)}{\partial w^{(2)}_{\alpha\beta}} = \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \sum_{j=0}^{K} \frac{d(w^{(2)}_{ij}(t)}{dw^{(2)}_{\alpha\beta}} v_j(t) \quad (7)$$

According to the method of steepest descent the weights' change in output layer are defined by

$$w^{(2)}_{\alpha\beta}(t+1) = w^{(2)}_{\alpha\beta}(t) - \eta \nabla^{(2)}_{\alpha\beta} E(t) \quad (8)$$

Elman Network

Weights change in the hidden layer is more complicated because of the feedbacks.

$$\nabla^{(1)}_{\alpha\beta} E(t) = \frac{\partial E(t)}{\partial w^{(1)}_{\alpha\beta}} = \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \sum_{j=0}^{K} \frac{d(w^{(2)}_{ij} v_j(t))}{dw^{(1)}_{\alpha\beta}} =$$

$$= \sum_{i=1}^{M} [y_i(t) - d_i(t)] \frac{df_2(g_i(t))}{dg_i(t)} \sum_{j=0}^{K} \left[\frac{d(v_j(t)}{dw^{(1)}_{\alpha\beta}} w^{(2)}_{ij} \right] \qquad (9)$$

$$\frac{dv_j(t)}{dw^{(1)}_{\alpha\beta}} = \frac{df_1(u_j)}{du_j} \sum_{m=0}^{N+K} \frac{d(x_m(t) w^{(1)}_{jm})}{dw^{(1)}_{\alpha\beta}} = \frac{df_1(u_j)}{du_j} \left[\delta_{j\alpha} x_\beta + \sum_{m=0}^{N+K} \frac{dx_m(t)}{dw^{(1)}_{\alpha\beta}} w^{(1)}_{jm} \right]$$

Elman Network

and next

$$\frac{dv_j(t)}{dw_{\alpha\beta}^{(1)}} = \frac{df_1(u_j)}{du_j}\left[\delta_{j\alpha}x_\beta + \sum_{m=N+1}^{N+K}\frac{dv_{(m-N)}(t-1)}{dw_{\alpha\beta}^{(1)}}w_{jm}^{(1)}\right] = \tag{10}$$

$$= \frac{df_1(u_j)}{du_j}\left[\delta_{j\alpha}x_\beta + \sum_{m=1}^{K}\frac{dv_m(t-1)}{dw_{\alpha\beta}^{(1)}}w_{j,m+N}^{(1)}\right]$$

The last formula (10) allows to calculate the derivatives of cost function with respect to weights of hidden layer in moment *t.* It is the recurrent formula defining derivative in a moment *t* dependence of a derivative in a moment *t*-1.

Elman Network

Assuming the initial values in a moment $t = 0$

$$\frac{dv_1(0)}{dw_{\alpha\beta}^{(2)}} = \frac{dv_2(0)}{dw_{\alpha\beta}^{(2)}} = \ldots = \frac{dv_K(0)}{dw_{\alpha\beta}^{(2)}} = 0$$

and basing on the steepest descent method the weights' change in the hidden layer is defined by

$$w_{\alpha\beta}^{(1)}(t+1) = w_{\alpha\beta}^{(1)}(t) - \eta \nabla_{\alpha\beta}^{(1)} E(t) \qquad (11)$$

Elman Network

Elman training algorithm

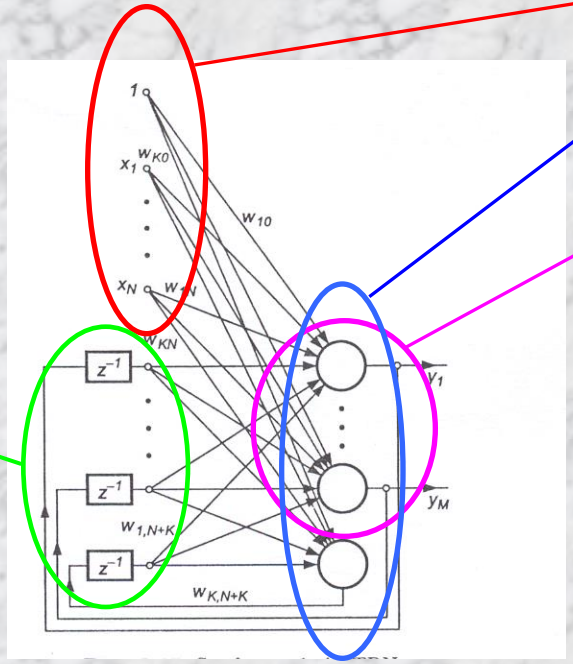
1. Initialize the weight vector with random values in , the learning rate , the repetitions counter () and the epochs counter (). Initialize the context nodes at 0.5.
2. Let the network's weight vector in the beginning of epoch
   - 2.1 Start of epoch. Store the current values of the weight vector
   - 2.2 For n=1,2,...,N
     - 2.2.1 Select the training example (x^n, t^n)and apply the error backpropagation in order to compute the partial derivatives $\eta\frac{\partial E^n}{\partial w_i}$
     - 2.2.2 Update the weights $w_i(k+1) = w_i(k) - \eta\frac{\partial E^n}{\partial w_i}$
     - 2.2.3 Copy the hidden nodes' values to the context units.
   - 2.3 End of epoch k. Termination check. If true, terminate.
3. k=k+1. Go to step 2.

Real Time Recurrent Network (RTRN)

RTRN Network

RTRN Network (Real Time Recurrent Network) is the network for real-time signal processing

Network structure



N- number of inputs
K – number of neurons in hidden layer
M – number of output neurons (from the *K* hidden elements)

K element context layer

RTRN Network

$$u_i(t) = \sum_{j=0}^{N+K} w_{ij}^{(1)} x_j(t)$$

weighted input signal of *i*-th neuron in hidden layer (12)

$$y_i = f(u_i(t))$$

output signal from *i*-th neuron in hidden layer (13)

Vector input signal ***x(t)*** and delayed one cycle vector ***y(t-1)*** create the excitation network signal

$$[1, x_1(t), \ldots, x_2(t), x_N(t), y_1(t-1), \ldots, y_K(t-1)] \quad (14)$$

RTRN Network

This is the special case of the Elman network with constant values of output weights ($w_{\alpha}^{(2)}$) and defined by

$$w_{ij}^{(2)} = \delta_{ij} = \begin{cases} 1 \text{ for } i = j \\ 0 \text{ for } i \neq j \end{cases}$$

Modifying the Elman`s algorithms one obtain the new learning algorithm for that network

RTRN Network

RTRN training algorithm

1. Initialization of weights (usually uniform distribution from [-1;1]).
2. Calculation of naurons' state in (t=0,1,...) (signals u_i and y_i formulas (12) and (13) and next vector excitation signal (14).
3. Calculation of $\frac{dy_j(t)}{dw_{\alpha\beta}} = 0$ according to

$$\frac{dy_j(t)}{dw_{\alpha\beta}} = \frac{df(u_j)}{du_j}\left[\delta_{j\alpha} + \sum_{t=1}^{K} \frac{dy_j(t-1)}{dw_{\alpha\beta}} w_{j,t+N}\right]$$

RTRN Network

RTRN training algorithm

4. Upgrades of weights according to steepest descent algorithm according to

$$w_{\alpha\beta}(t+1) = w_{\alpha\beta}(t) - \eta \sum_{j=1}^{M} [y_j(t) - d_j(t)] \frac{dy_j(t)}{dw_{\alpha\beta}}$$

for $\alpha=1,2,\ldots,K$, $\beta=0,1,2,\ldots,N+K$

5. Go to step 2.

Radial Networks

Network structure

Radial networks

For special purposes sometimes we are using the radial neurons i.e. neurons with *Radial Basis Function RBF*. They have non typical aggregation methods of input data, they uses non typical transfer function (Gauss function) and they learned by the special way.

Data aggregation consist on the calculation of a distance between an input signal (vector **X**), and established in a learning process centroid of a certain set **T**.

Network structure

Sigmoid and radial neuron

Sigmoidal neuron represents in the multidimensional space hyperplane separating space into two categories (Fig.A). Radial neuron represents hypersphere performing circular separation around the central point (Fig.B).

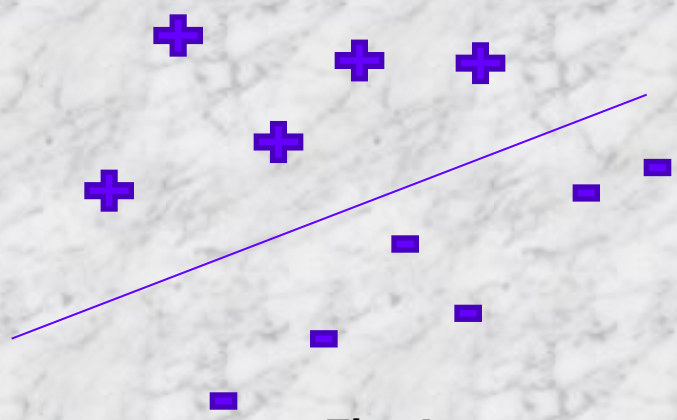


Fig. A

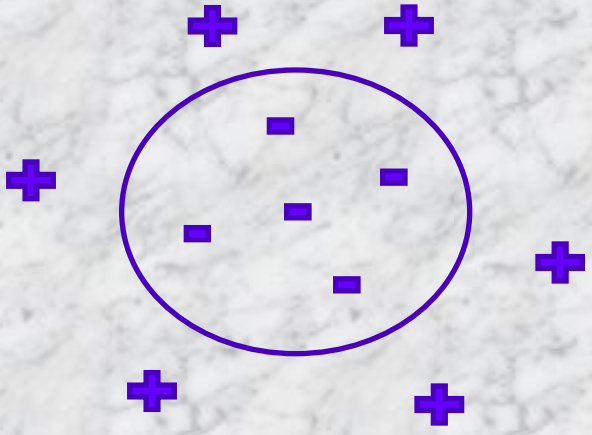


Fig. B

Radial basis function network

Radial basis function network (**RBF**) uses radial basis functions as activation functions typically have three layers: an input layer, a hidden layer with a non-linear **RBF** activation function and a linear output layer.

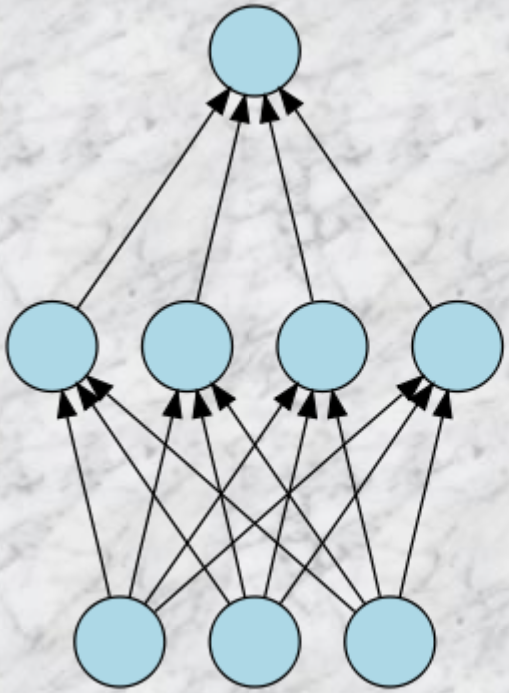
Functions that depend only on the distance from a center vector are radially symmetric about that vector, hence the name *radial basis function*. In the basic form all inputs are connected to each hidden neuron. The norm is typically taken to be the Euclidean distance and the radial basis function is commonly taken to be Gaussian.

Radial basis function network

The output of the network is then a scalar function of the input vector, and is given by

$$\varphi(\mathbf{x}) = \sum_{i=1}^{N} a_i \rho(||\mathbf{x} - \mathbf{c}_i||)$$

where N is the number of neurons in the hidden layer, c_i is the center vector for neuron , and a_i is the weight of neuron i in the linear output neuron. Functions that depend only on the distance from a center vector are radially symmetric about that vector. The radial basis function is commonly taken to be Gaussian.



$$\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp\left[-\beta \|\mathbf{x} - \mathbf{c}_i\|^2\right]$$