ARTIFICIAL NEURAL NETWORKS

Federico Marini

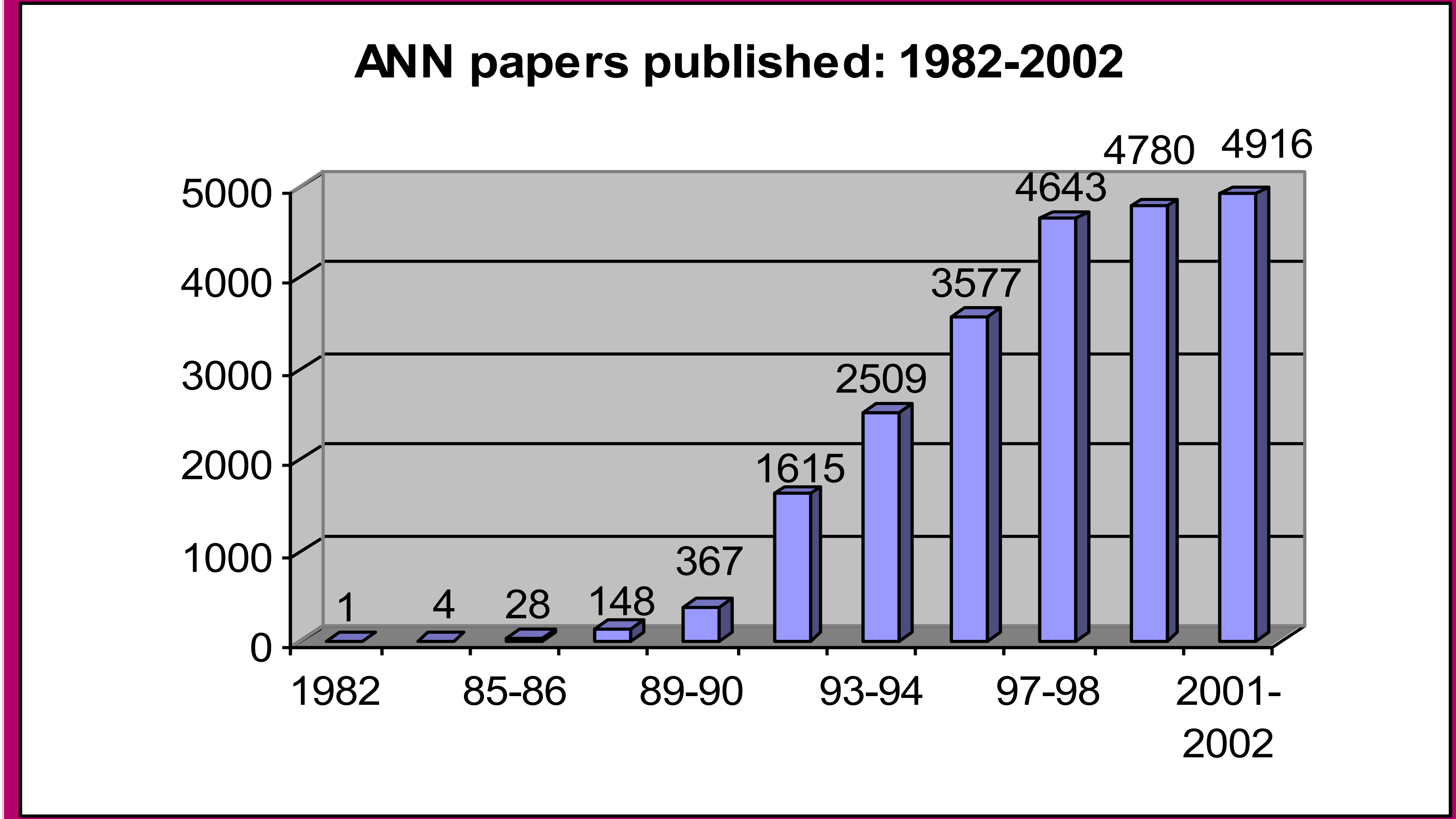
Dept. Chemistry, University of Rome "La Sapienza", Italy

SAPIENZA
UNIVERSITÀ DI ROMA

Artificial neural networks (ANNs)

Historical background

- 1943: McCulloch & Pitts model of a neuron
- 1949: Hebb postulates a learning rule
- 1958: Rosenblatt model of a neural net
- 1969: the book “Perceptrons” freezes most of the initial enthusiasm about ANNs
- 1982: Hopfield and its model of neural net act as “catalyst” in attracting the attention of many scientists towards NNs



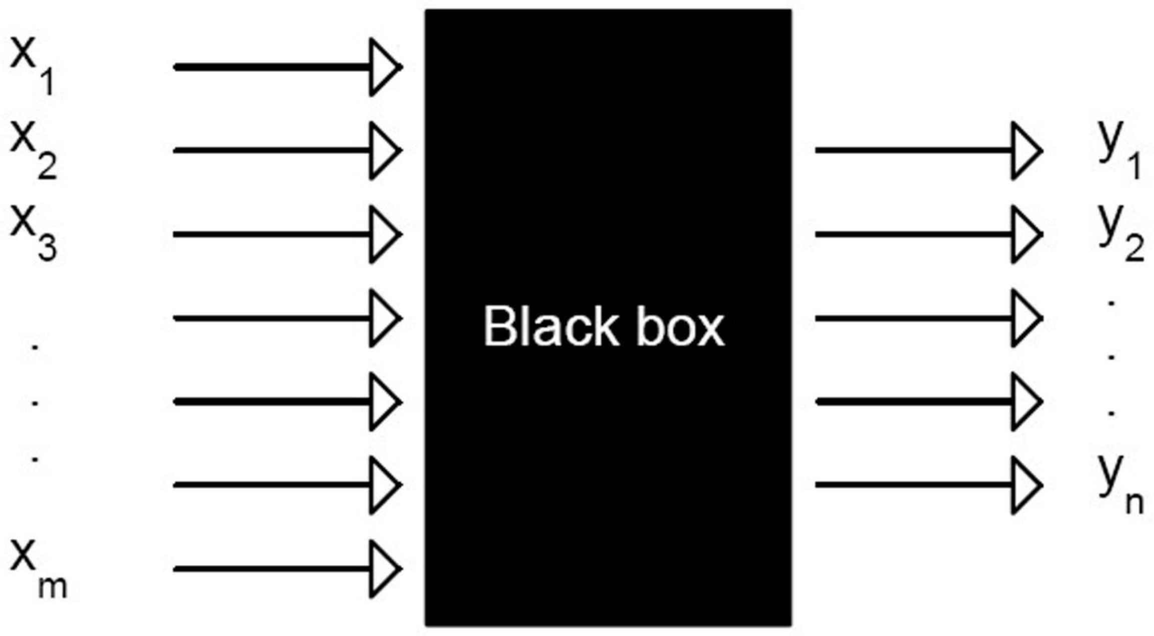
J. Hopfield, Neural Networks and Physical Systems with Emergent Collective Computational Ability, P. Nac. Acad. Sci. Biol., 79(2), (1982), 2554-2558

Strengths of a Neural Network

- **Power**: Model complex functions, nonlinearity built into the network
- **Ease of use**:
  - Learn by example
  - Very little user domain-specific expertise needed
- **Intuitively appealing**: based on model of biology, will it lead to genuinely intelligent computers/robots?

Neural networks cannot do anything that cannot be done using traditional computing techniques, **BUT** they can do some things which would otherwise be very difficult.

The roughest approach to NNs



Input variables	Non-linear relation	Output varibles

NNs in a nutshell

- From a computational point of view, ANNs represent a way to operate a non-linear functional mapping between an input and an output space.

$$\mathbf{y} = f(\mathbf{x})$$

- **Y** can be:
  - an n-D (usually 2D) vector of coordinates (***mapping***)
  - A multi-dimensional vector of response (***regression***)
  - A binary vector of class-memberships (***classification***)
- This functional relation is expressed in an implicit way

Opening the blackbox

- The peculiarity of ANNs relies on the fact that they operate using a large number of parallel connected simple arithmetic units (neurons).
- Mathematically, a neuron can be defined as *nonlinear, parameterized, bounded function* ➔ the variables this function depends on are called the inputs of the neuron and its value is called the output.

$$y = f\left(\sum_i w_i x_i + w_0\right)$$

Artificial neurons

Nonlinear generalization of the McCullogh-Pitts neuron:

$$y = f(x, w)$$

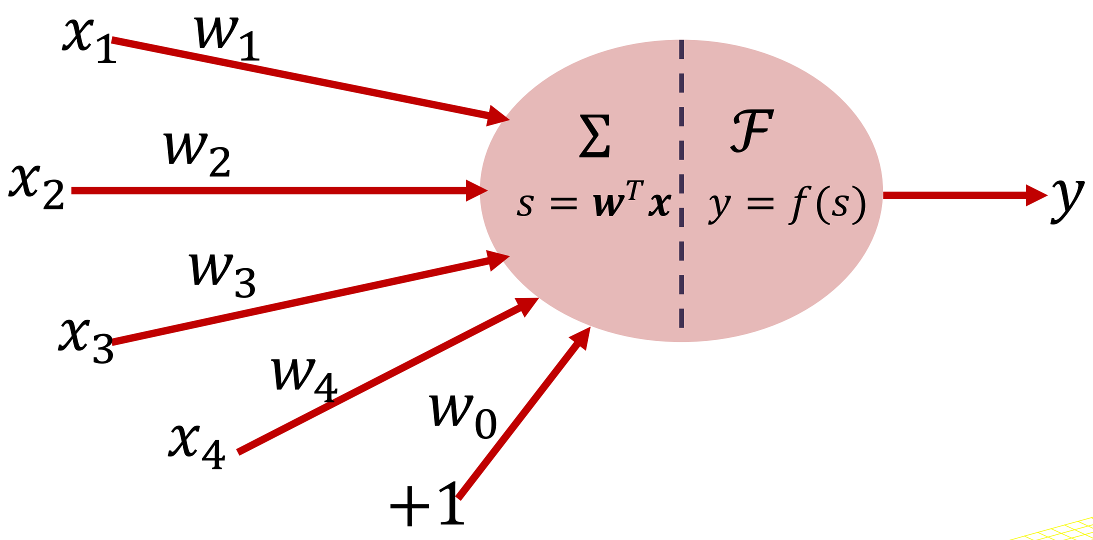
y is the neuron's output, x is the vector of inputs, and w is the vector of synaptic weights.

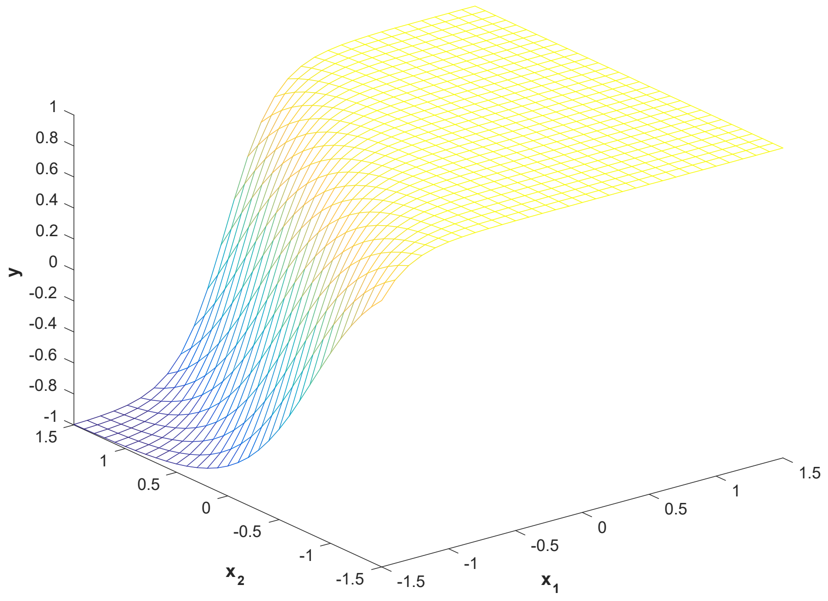
Examples:

$$y = \frac{1}{1 + e^{-w^T x - a}}$$ sigmoidal neuron

$$y = e^{-\frac{||x-w||^2}{2a^2}}$$ Gaussian neuron

Artificial neurons $y = f(x, w)$





Activation functions

Hard threshold

Piecewise linear

Sigmoid

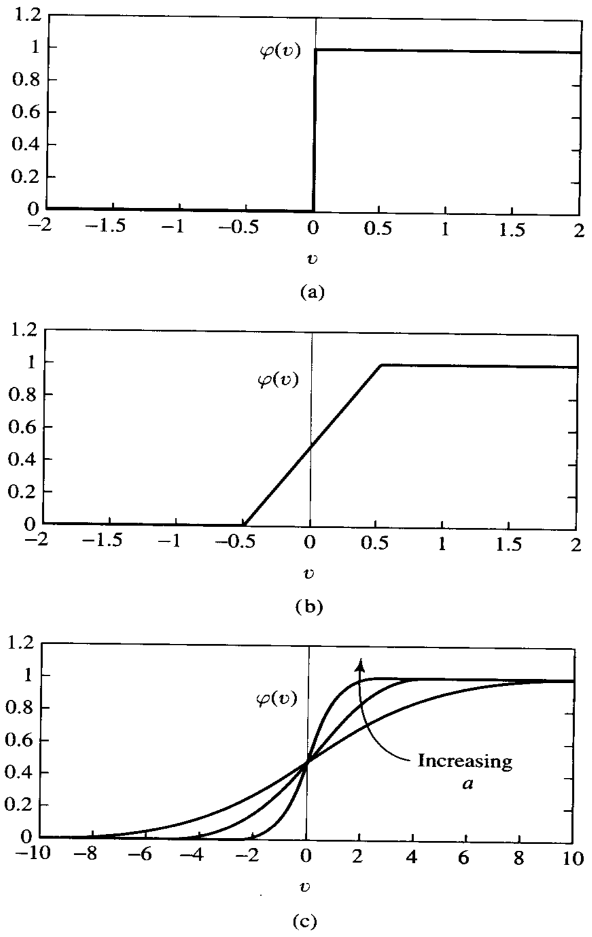
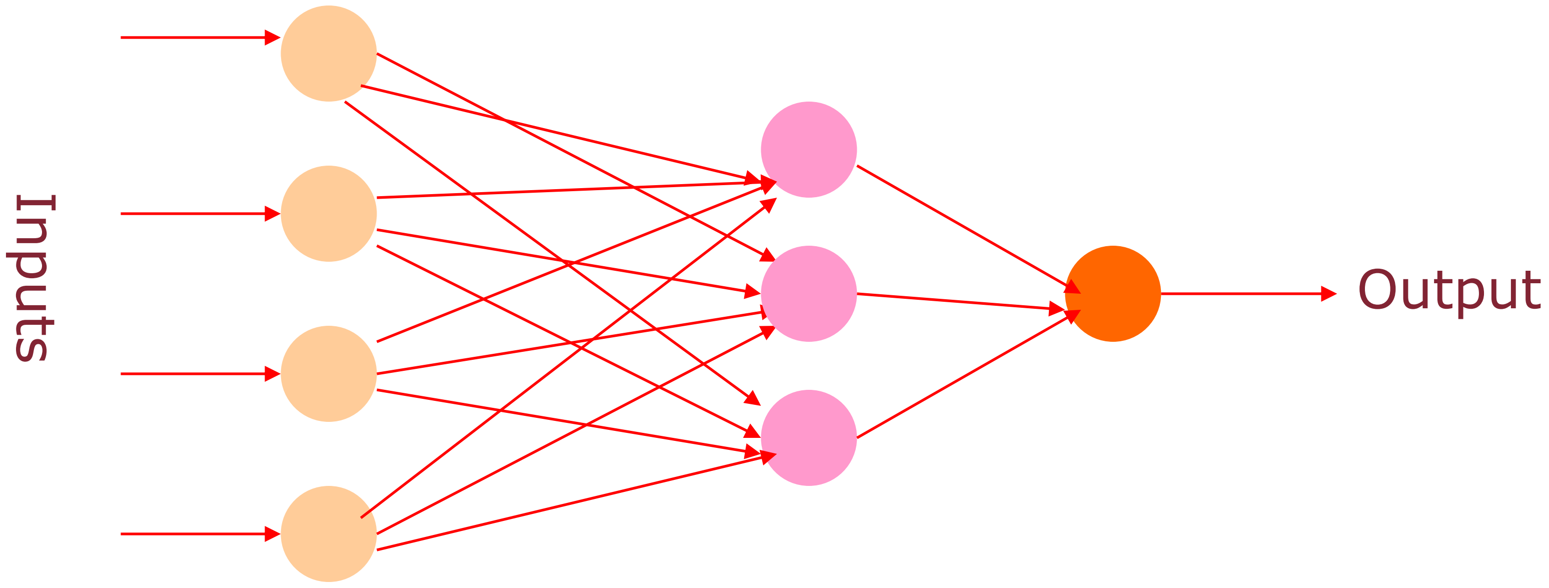


FIGURE 1.8 (a) Threshold function. (b) Piecewise-linear function. (c) Sigmoid function for varying slope parameter *a*.

From the neuron to the net

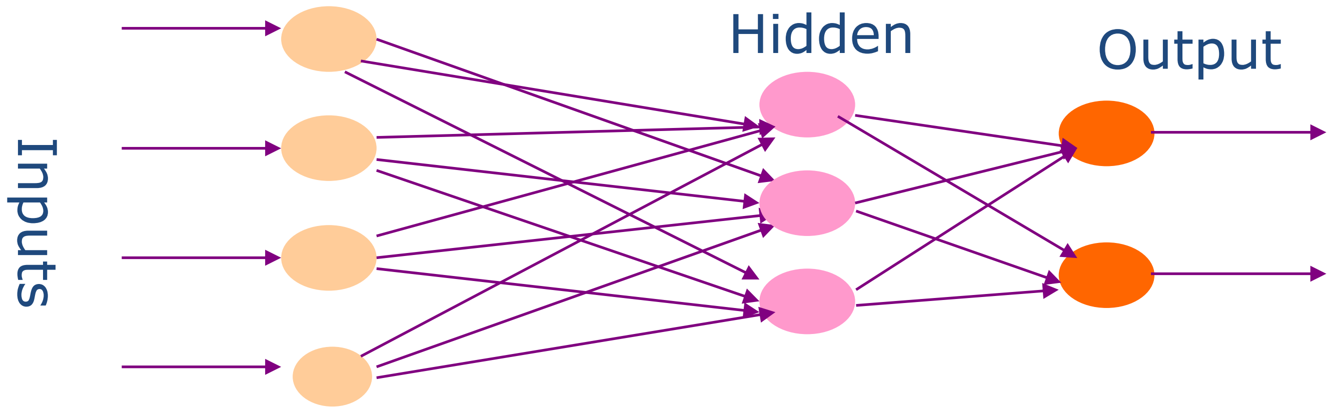
- Just as a neuron can be thought as a nonlinear function of its inputs, a network represents the composition of the nonlinear functions of two or more neurons
- The way the different units are connected among each other governs the way the different functions they describe are weighted and combined to produce the overall output.
- This pattern of interconnection among the neurons is called the network **"architecture"**, and can be conveniently represented on a graph: neurons operating on the same input variables are organized in *layers*, while the weights that modulate the combination of the nonlinear functions are represented as lines connecting units in different layers.

Artificial neural networks



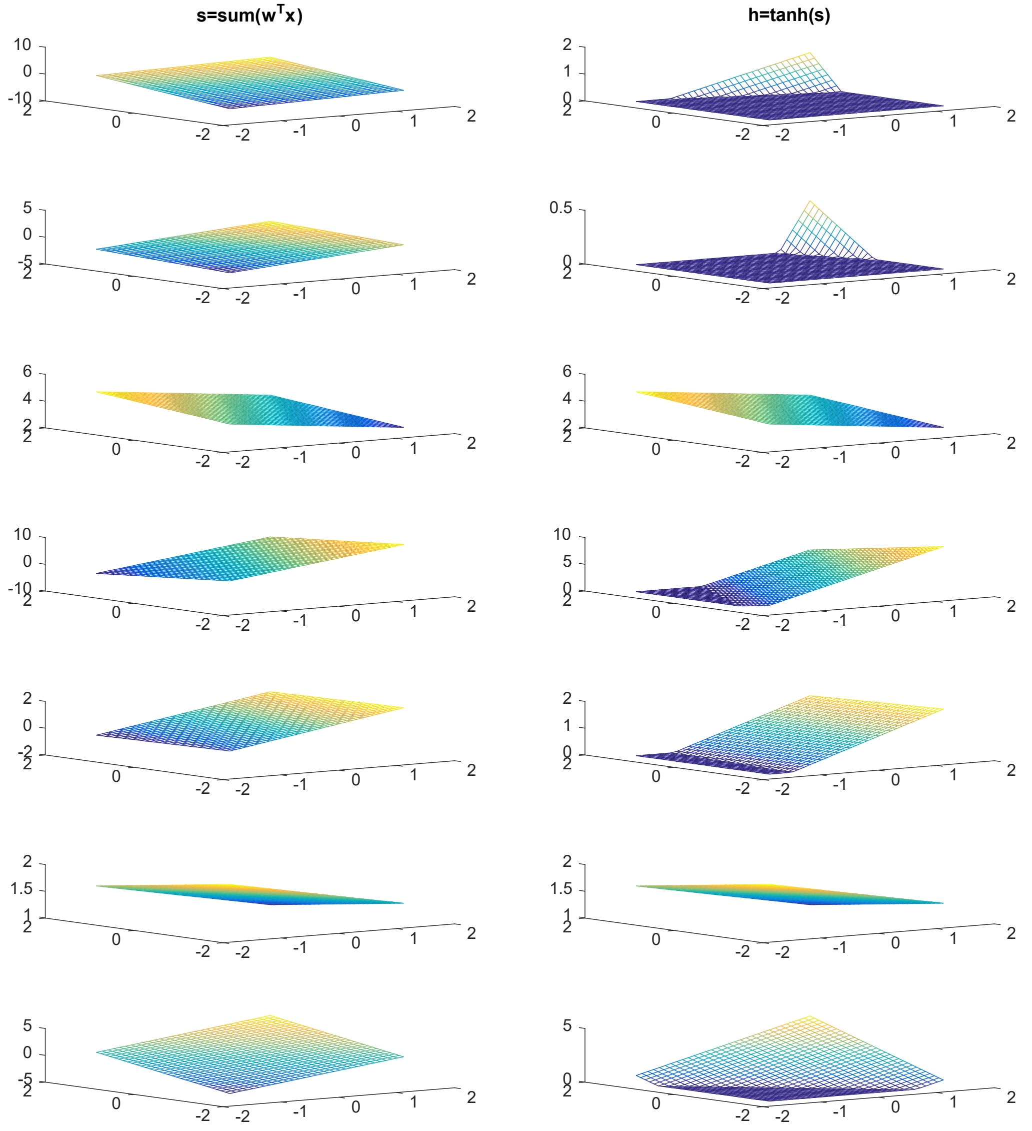
An artificial neural network is composed of many artificial neurons that are linked together according to a specific network architecture. The objective of the neural network is to transform the inputs into meaningful outputs.

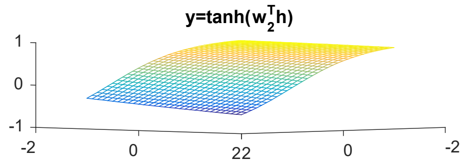
Artificial (multilayer feed-forward) NNs



$$y_j = g\left(\sum_{k=1}^{N_h} w_{jk} h_k + w_{j0}\right) = g\left(\sum_{k=1}^{N_h} w_{jk} f\left(\sum_{i=1}^{N_i} w_{ki} x_i + w_{k0}\right) + w_{j0}\right)$$

Artificial neural networks





How does a neural network learn?

- A neural network learns by determining the relation between the inputs and outputs.

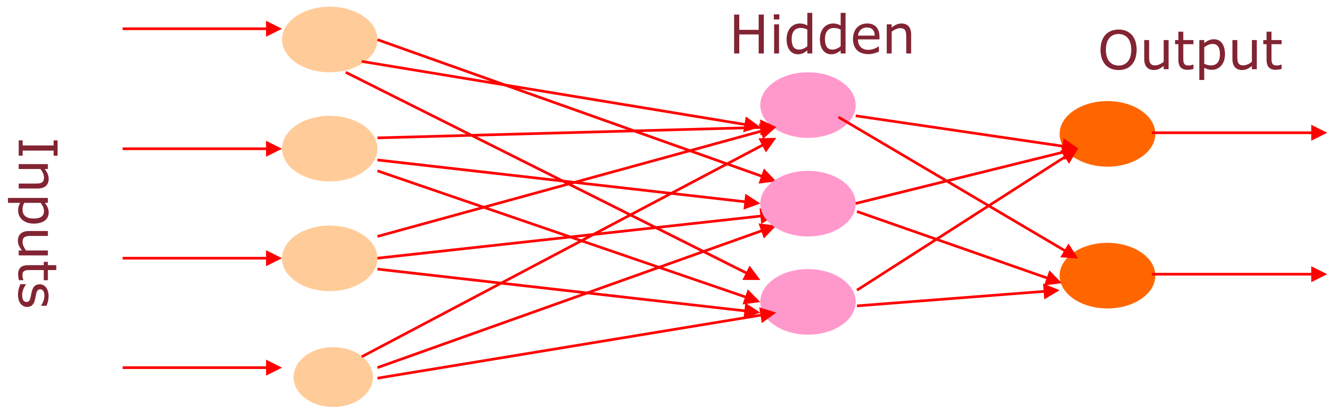
- By calculating the relative importance of the inputs and outputs the system can determine such relationships.

- Through trial and error the system compares its results with the expert provided results in the data until it has reached an accuracy level defined by the user.
  - With each trial the weight assigned to the inputs is changed until the desired results are reached.

Multilayer feed-forward networks

- The most common network architecture is the ***feed-forward*** one, in which the information flows only in the forward direction, from inputs to outputs.
- This means that its graph representation is acyclic: no path in the graph, following the connections, can lead back to the starting point.
- A great variety of network topologies can be imagined, under the sole constraint that the graph of connections be acyclic. However, as anticipated before, the vast majority of neural network applications implement multilayer networks.

Multilayer feed-forward NNs 2



The network computes as many functions of the input variables of the network as are the components of the output vector

• Neurons are organized in three kind of layers: input, hidden and output.
• The output neurons are the neurons that perform the final computation, i.e., whose outputs are the outputs of the network, while the other neurons, which perform intermediate computations, are termed hidden neurons.
• The units of the input layer just pass the inputs as variables to the hidden neurons, without doing any processing on them.
• Each output is a nonlinear function (computed by the corresponding output neuron) of the nonlinear functions computed by the hidden neurons.

$$y_j = g\left(\sum_{k=1}^{N_h} w_{jk} h_k + w_{j0}\right) = g\left(\sum_{k=1}^{N_h} w_{jk} f\left(\sum_{i=1}^{N_i} w_{ki} x_i + w_{k0}\right) + w_{j0}\right)$$

“Training” the net

- A feed-forward network with a single hidden layer can approximate with arbitrary accuracy any bounded and sufficiently regular function in a finite region of variable space.
- The procedure by whereby the parameters of the network are estimated, in order to approximate such a function is called *training* of the ANN.
- Since usually the nonlinear relationship between dependent and independent variables is not known analytically, but a finite number of numerical values of the function are known (because they are obtained through measurements performed on a physical, chemical, biological, etc. process): *the task that is assigned to the network is that of approximating the regression function of the available data*.

Supervised training

- This kind of training is referred to as “supervised” since the function that the network should implement is known in some or all points: a “teacher” provides “examples” of values of the inputs and of the corresponding values of the output,
- The goal of the training algorithm is to find the best set of model parameters, given the data ➔ find the numerical values of the network weights which minimize a cost function representing the distance between the prediction of the model and the measured values.

$$E^P(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{N_o}(y_k - \hat{y}_k)^2 = \frac{1}{2}\|\mathbf{y} - \hat{\mathbf{y}}\|^2$$

Supervised training

- When this cost function is the squared error of the residuals:

$$E^{\text{batch}}(\mathbf{w}) = \frac{1}{N_{\text{samples}}}\sum_{P=1}^{N_{\text{samples}}} E^{P}(\mathbf{w}) = \frac{1}{2N_{\text{samples}}}\sum_{P=1}^{N_{\text{samples}}}\sum_{k=1}^{N_o}\left(y_{k,P}-\hat{y}_{k,P}\right)^2 = \frac{1}{2N_{\text{samples}}}\sum_{P=1}^{N_{\text{samples}}}\|\mathbf{y}_P-\hat{\mathbf{y}}_P\|^2$$

- the resulting training algorithm is called back-propagation (BP) and is essentially an iterative weight update on the basis of a steepest descent criterion.

$$\Delta w_{ji}(t) = -\eta\frac{\partial E}{\partial w_{ji}} + \mu\Delta w_{ji}(t-1)$$

More on backpropagation

Weight update rule

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m),$$

Output neurons

$$\Delta w_{kj} = \eta\delta_k y_j = \eta(t_k - z_k) f'(net_k) y_j.$$

Hidden neurons

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \underbrace{\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(net_j)}_{\delta_j} x_i.$$

Second order methods

- Backpropagation is rather simple and easy to implement but can suffer from sever convergence problems.
- One way of coping with this is momentum.
- Another is to use second-order methods.
- Given an initial estimate of the weights, the error is Taylor-expanded up to the second order:

$$E^{\text{batch}}(\mathbf{w}) \cong E^{\text{batch}}(\mathbf{w}_0) + \left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)^{\mathrm{T}} \Delta\mathbf{w} + \frac{1}{2}(\Delta\mathbf{w})^{\mathrm{T}} \mathbf{H} \Delta\mathbf{w}$$

- where:

$$\nabla_{\mathbf{w}_0} E^{\text{batch}} = \left.\frac{\partial E^{\text{batch}}}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}_0} \qquad \mathbf{H} = \left.\frac{\partial^2 E^{\text{batch}}}{\partial \mathbf{w}^2}\right|_{\mathbf{w}=\mathbf{w}_0}$$

- Backpropagation corresponds to truncating the expansion to the first order.

Second order methods - 2

- Based on the second order expansion, solution is sought by differentiating the previous equation wrt to the weights and setting the derivative = 0 (Newton's method)

$$\Delta \mathbf{w}^* = -\mathbf{H}_{\mathbf{w}_0}^{-1} \nabla_{\mathbf{w}_0} E^{\text{batch}}$$

- Problems:
  - Hessian matrix H should be calculated and stored (computationally and memory intensive).
  - Hessian matrix should be nonsingular (not guaranteed, quite often H is not full rank).
- Solutions:
  - Quasi-second order methods:
    - Gauss-Newton
    - Levenberg-Marquardt

Levenberg-Marquardt method

- In Gauss-Newton, the Hessian matrix is approximated by:

$$\mathbf{H} \approx \left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)\left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)^{\mathrm{T}}$$

- However, this approximation is accurate only near the minimum and can suffer from initial guesses far from the optimal solution.
- LM method introduces an additional term to stabilize the estimate:

$$\Delta \mathbf{w}^{*}_{\mathrm{LM}} = -\mathbf{H}^{-1}_{\mathbf{w}_0} \nabla_{\mathbf{w}_0} E^{\text{batch}} \approx -\left[\left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)\left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)^{\mathrm{T}} + \mu \mathbf{I}\right]^{-1}\left(\nabla_{\mathbf{w}_0} E^{\text{batch}}\right)$$

- The parameter μ controls the property of the algorithm:
  - When μ is large the steepest descent term is dominant
  - When μ is small the Gauss-Newton term is dominant
- Fast and efficient

Specifically for classification

- When ANNs are used for classification:
  - Error criterion is cross-entropy instead of RMSE

$$E_{CE} = \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \ln \frac{y_{ij}}{\hat{y}_{ij}}$$

  - Output function is “softmax”

$$\hat{y}_j = \frac{e^{t_j}}{\sum_{k=1}^{M} e^{t_k}}$$

Generalization *vs* specialization

- Optimal number of hidden neurons
  - Too many hidden neurons: you get an over fit, training set is memorized, thus making the network useless on new data sets
  - Not enough hidden neurons:
    network is unable to learn problem concept

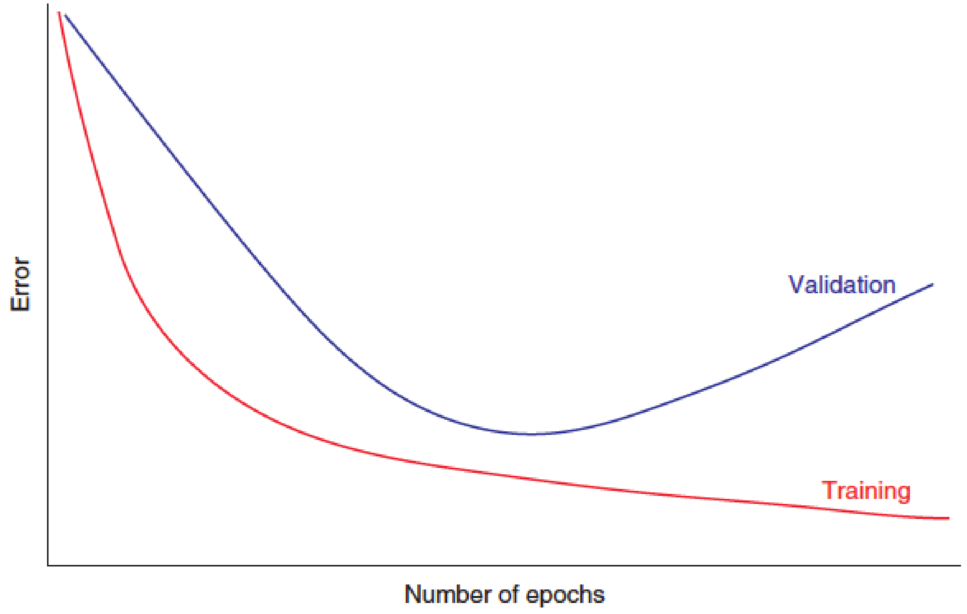
    ~ conceptually: the network's language isn't able to express the problem solution

WHY ISN'T MY CELL PHONE WORKING?
THAT'S A SHORT-RANGE CELL PHONE. YOU NEED TO BE IN THE SAME ROOM WITH THE PERSON YOU CALL.
ANSWER THE STINKIN' PHONE, ALICE.
WHY ARE YOU LISTEN-ING TO A TV REMOTE CONTROL?
scottadams@aol.com
www.dilbert.com
3/12/03 © 2003 United Feature Syndicate, Inc.

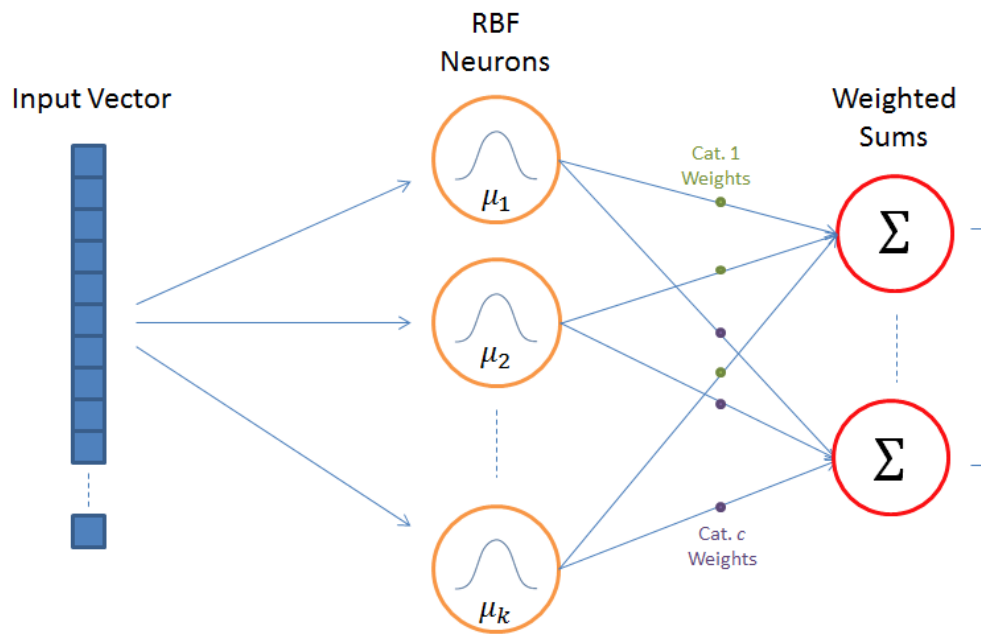
Generalization vs. specialization 2

- Overtraining:
  - Too much examples, the ANN memorizes the examples instead of the general idea
- Generalization vs. specialization trade-off:

  # hidden nodes & training samples



RADIAL BASIS FUNCTIONS - NN

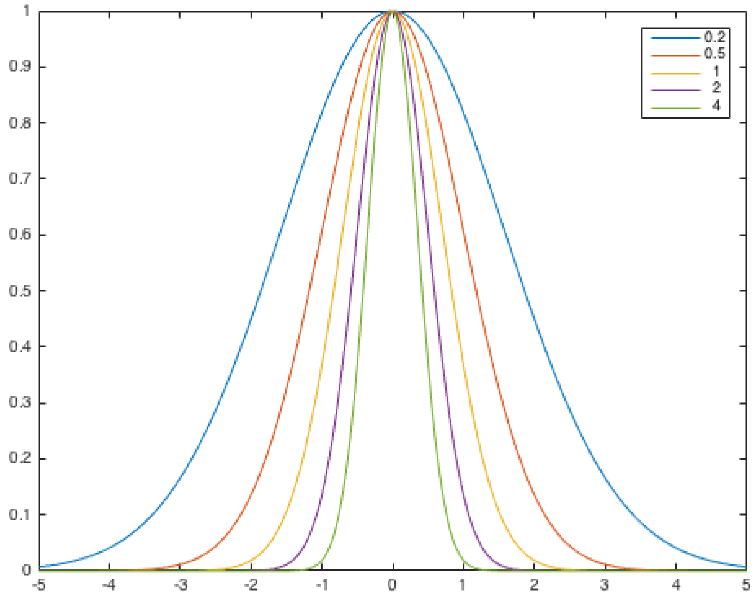


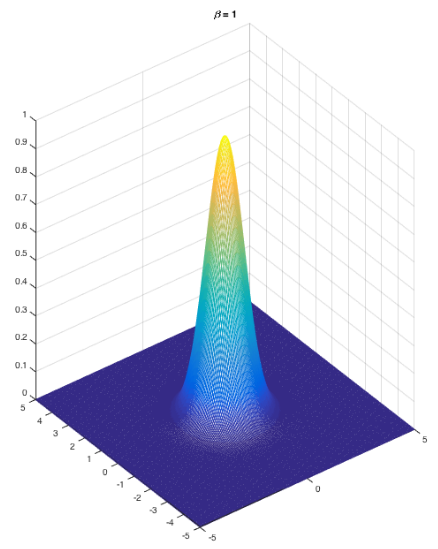
The RBF-NN

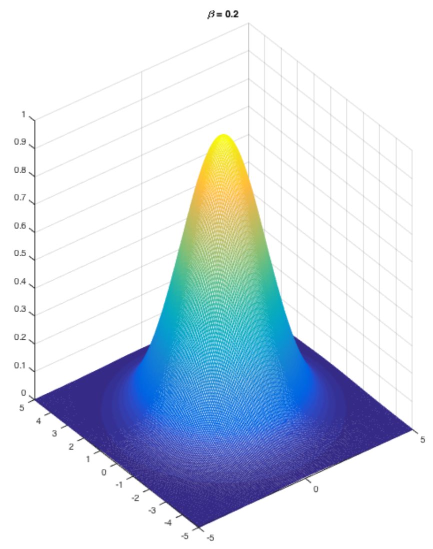
- Differently from MLP, RBF-NN performs classification and regression based on similarity with examples from the training set
- Its basic unit is the Gaussian (RBF) neuron:

$$y = e^{-\frac{\|\mathbf{x}_i - \mathbf{c}_k\|^2}{2\sigma^2}} = e^{-\beta\|\mathbf{x}_i - \mathbf{c}_k\|^2}$$

c_k is the center of the k^{th} RBF

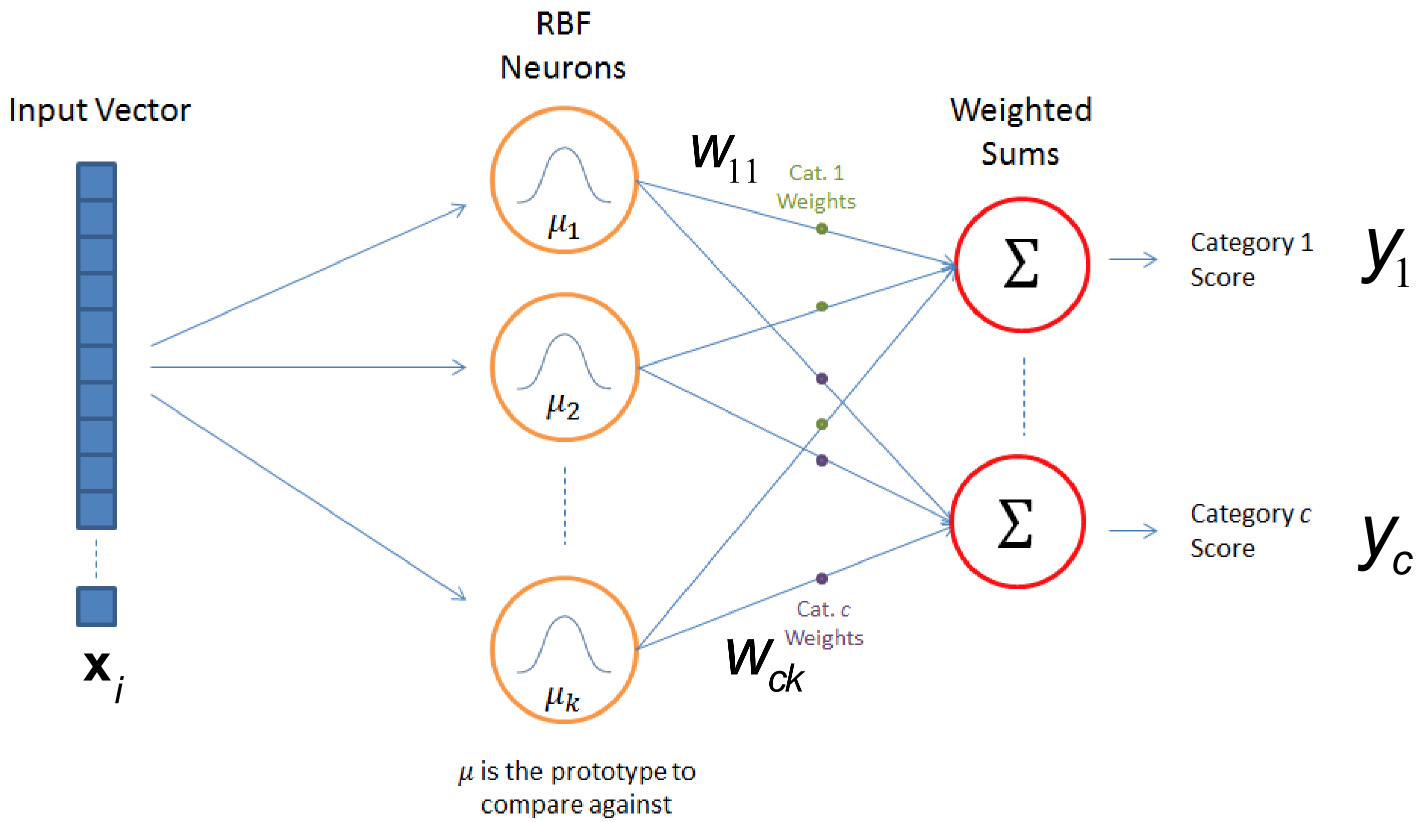






The RBF-NN for classification

$$y_j = \sum_{k=1}^{n_{RBF}} w_{jk} e^{-\beta \|\mathbf{x}_i - \mu_k\|^2}$$



There is the need to optimize the center and width of RBFs and the weights w_{jk}

Training the RBF-NN

- There are many different training algorithms for RBF-NN
- Apart from backpropagation, the most common are Orthogonal least squares and the following:
  1. Select the RBF centers by k-means clustering (possibly, applying clustering separately by category)
  2. Calculate the RBF width as the mean cluster distance to the centroid:

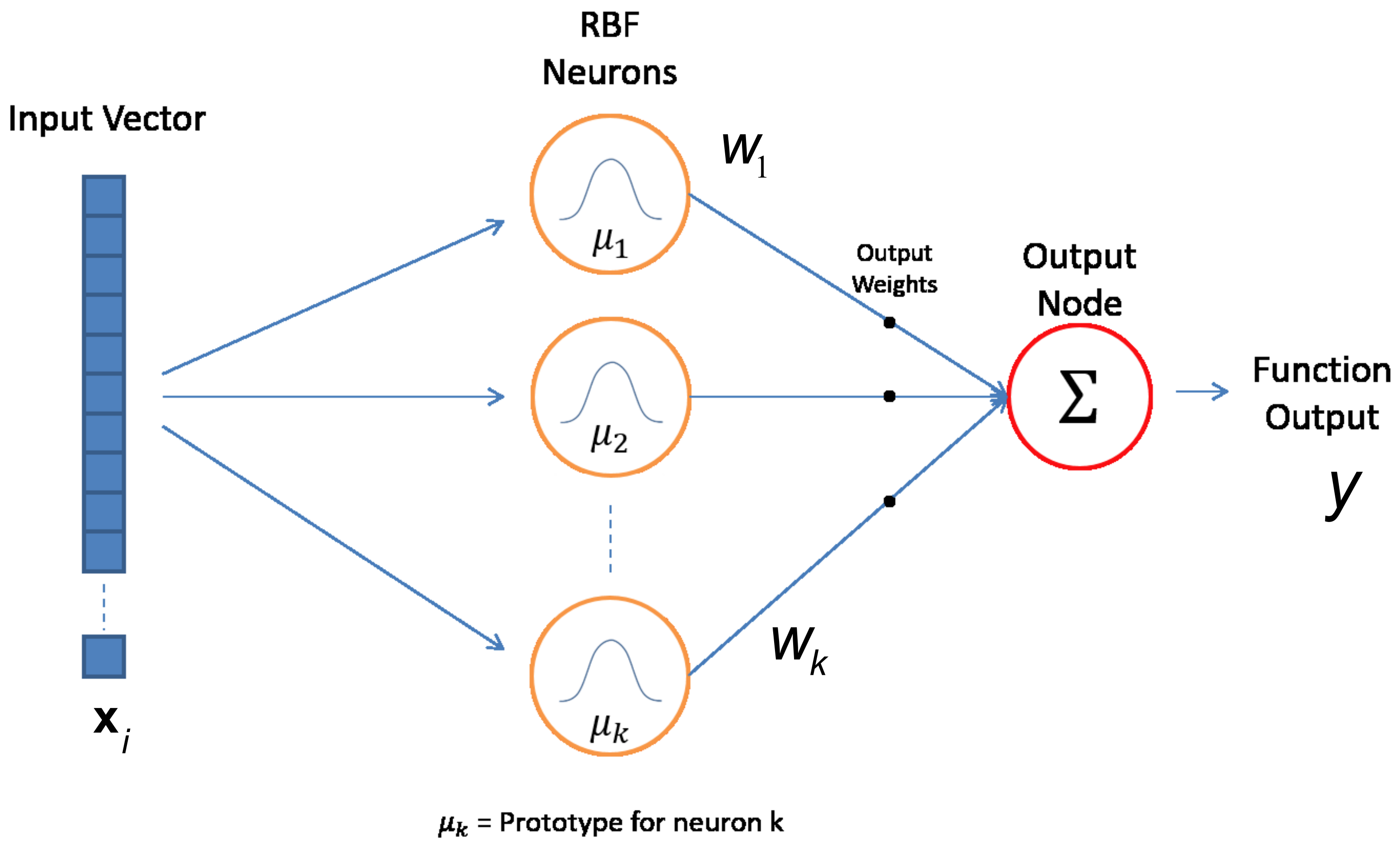
$$\sigma_k = \frac{1}{m} \sum_{i=1}^{m} \left\| \mathbf{x}_i - \mu_k \right\| \quad \text{and} \quad \beta_k = \frac{1}{2\sigma_k^2}$$

  1. Calculate the weight by LS regression:

$$\mathbf{W} = \mathbf{H}^{\dagger}\mathbf{Y} \quad \text{with } h_{ik} = e^{-\beta_k \left\| \mathbf{x}_i - \mu_k \right\|^2}$$

The RBF-NN for regression

$$y = \square_{k=1}^{n_{RBF}} w_k e^{-\beta \left\| \mathbf{x}_i - \mu_k \right\|^2}$$



There is the need to optimize the center and width of RBFs and the weights w_{jk}

Training the RBF-NN for regression

- The training algorithms for regression are the same as for classification, with two main differences:
  1. The width of the RBF is normally selected as equal for all nodes and based on cross-validation instead as on clustering.
  2. The output of the function is often scaled (normalized):

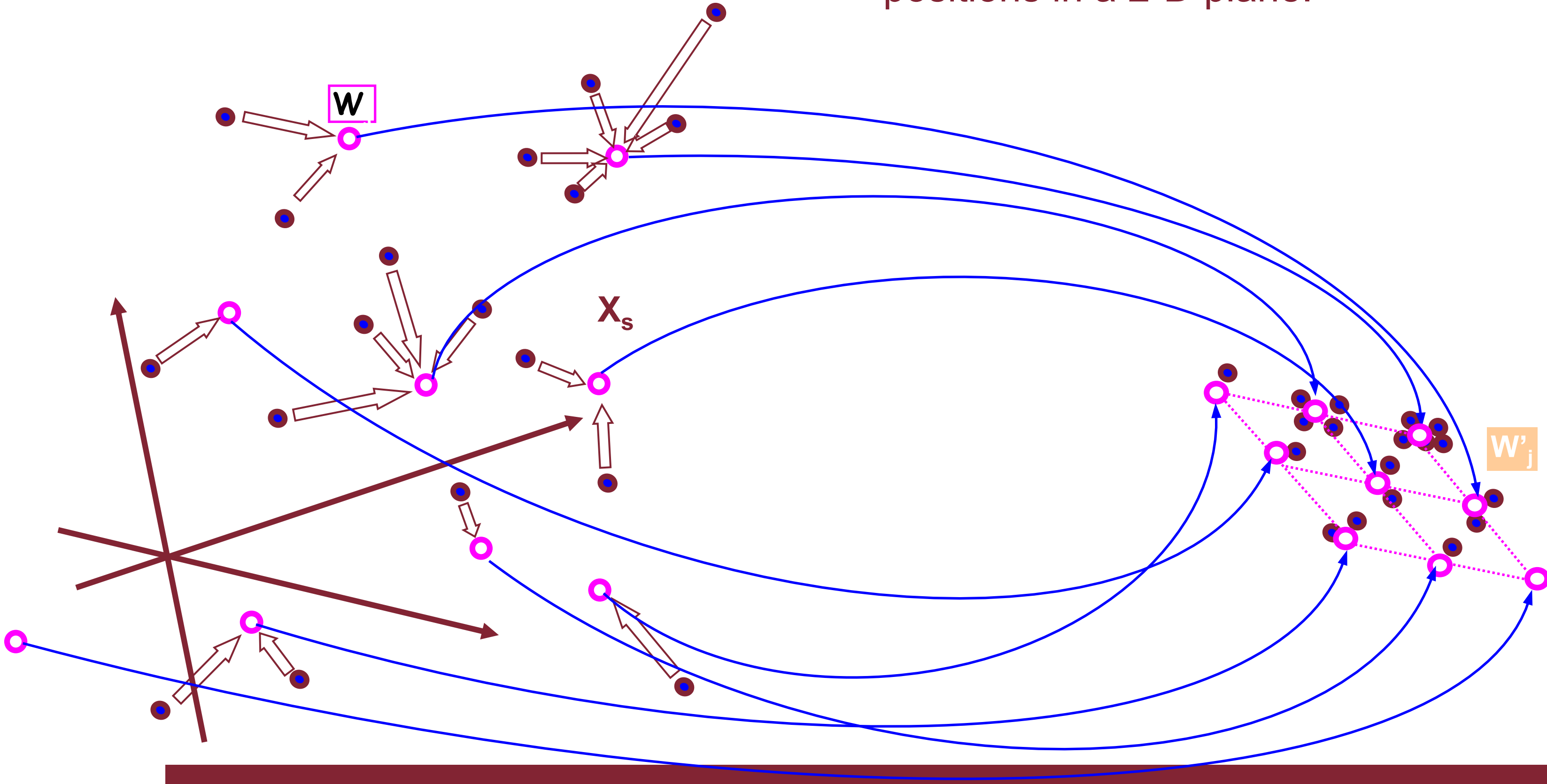
$$y = \frac{\square_{k=1}^{n_{RBF}} w_k e^{-\beta\left\|\mathbf{x}_i - \mu_k\right\|^2}}{\square_{k=1}^{n_{RBF}} e^{-\beta\left\|\mathbf{x}_i - \mu_k\right\|^2}}$$

Self organizing maps (Kohonen architectures)

Relative distance measure

The objects X_s are adjusted according to an absolute distance measure $d(X_s,W_j)$ to positions of the prespecified points W_j distributed in a topologically predefined scheme.

In the case of Kohonen neural networks the objects X_s and the arbitrarily distributed points W_j are, first, assigned to each other, and next, the points W_j with associated closest objects $\{X_s\}$ are pooled together to predefined positions in a 2-D plane.



The final result does not depend much on the distances between objects, but rather on the distances between the objects X_s and the pivot points W_j.

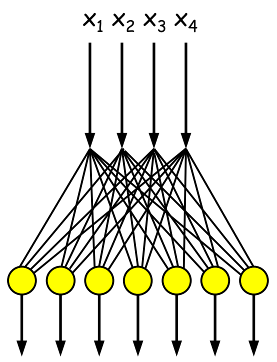
Kohonen self organizing maps (SOMs)

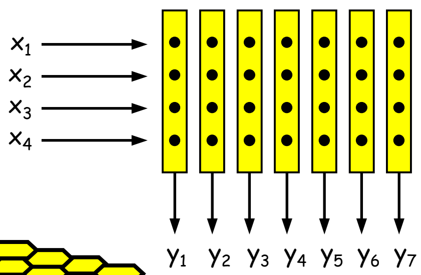
- The implicit functional relation that we want to approximate is a nonlinear mapping from an N_i-dimensional input space to a low-dimensional (usually 2D) discrete coordinate space (the map).
- Since there is no desired response to be obtained, training occurs by self-organization, i.e. a Kohonen network adapts itself so that similar input objects are associated with topological close neurons.
- In a self-organizing map, the target space used in Kohonen mapping is a two-dimensional array of neurons (the Kohonen layer or top-map), fully connected to the input layer, onto which the samples are mapped.
- Introducing the preservation of topology results in specifying for each node in the Kohonen layer a defined number of neurons as nearest neighbors, second-nearest neighbors and so on.

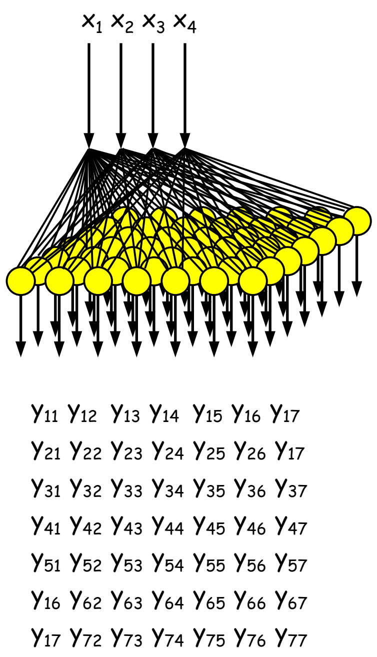
Kohonen self organizing maps 2

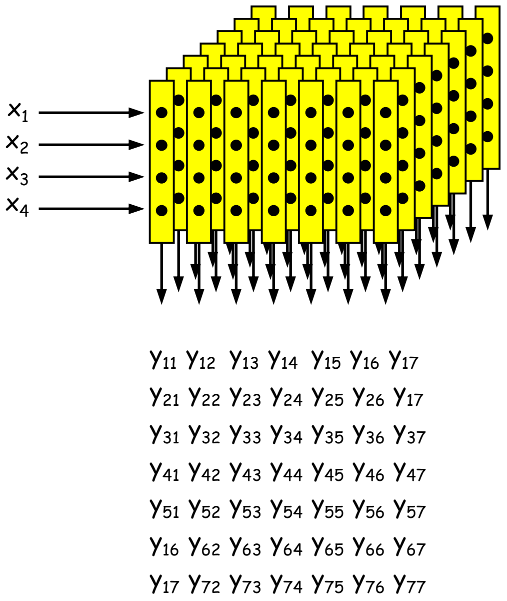
The most important feature of the Kohonen neural network is the topological order in which the neurons are combined together into the network.

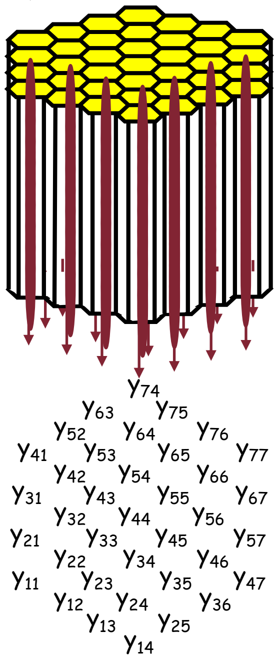
The lay-out of neurons in the Kohonen network can be linear (all neurons in one row ot line) or planar (in a rectangular or hexagonal lay-out).





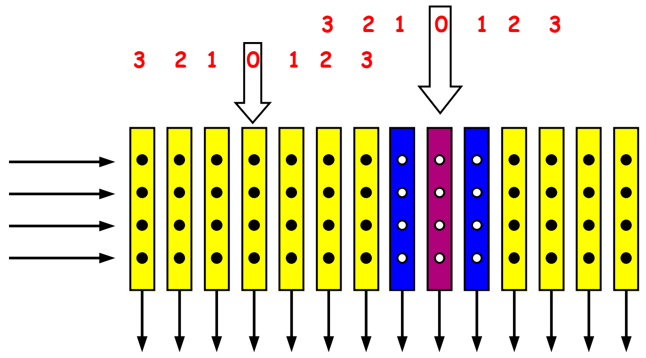

x1 x2 x3 x4
y11 y12 y13 y14 y15 y16 y17
y21 y22 y23 y24 y25 y26 y17
y31 y32 y33 y34 y35 y36 y37
y41 y42 y43 y44 y45 y46 y47
y51 y52 y53 y54 y55 y56 y57
y16 y62 y63 y64 y65 y66 y67
y17 y72 y73 y74 y75 y76 y77

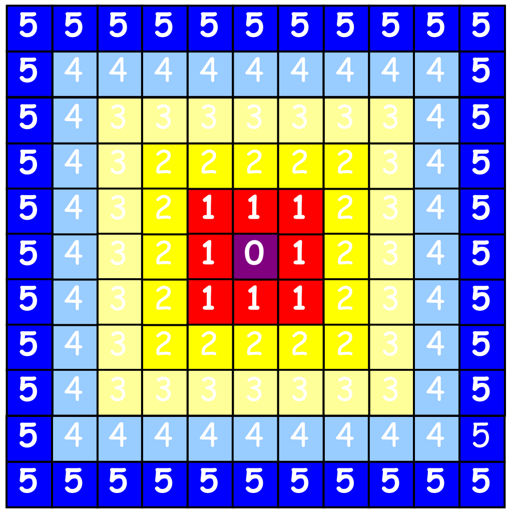

x1
x2
x3
x4
y11 y12 y13 y14 y15 y16 y17
y21 y22 y23 y24 y25 y26 y17
y31 y32 y33 y34 y35 y36 y37
y41 y42 y43 y44 y45 y46 y47
y51 y52 y53 y54 y55 y56 y57
y16 y62 y63 y64 y65 y66 y67
y17 y72 y73 y74 y75 y76 y77

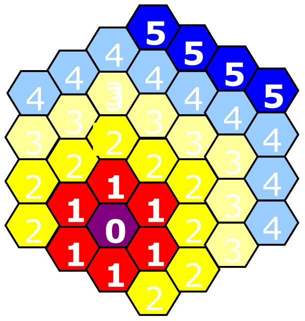


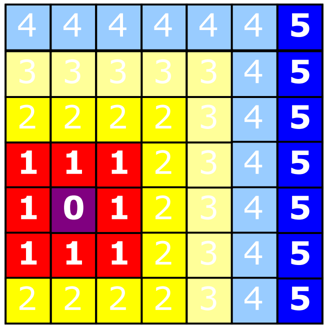
Defining the neighborhood

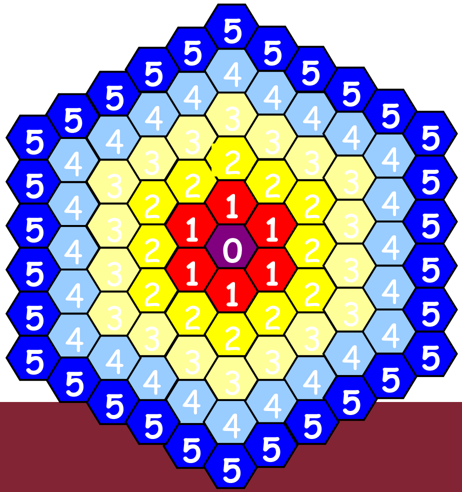
The *neighborhood* of a neuron is usually considered to be square or hexagonal which means that each neuron has 8 or 6 nearest neighbors respectively.







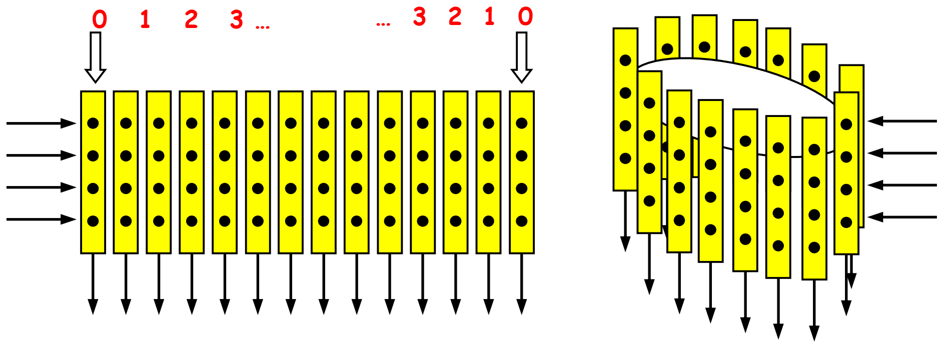




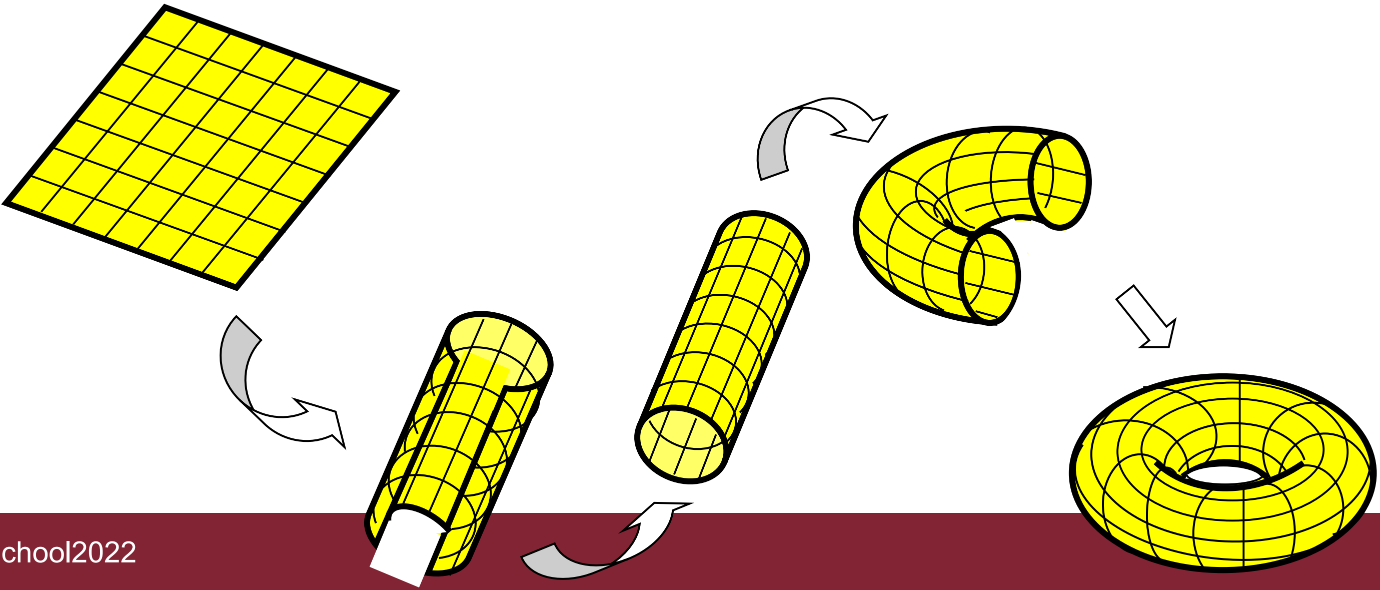
Cyclic and toroid conditions

The cyclic and the toroid boundary conditions. The edge one one side is linked to the edge on the opposite side.

Cyclic conditions in a line of neurons: 1st neighbours of W_1 are W_2 and W_n

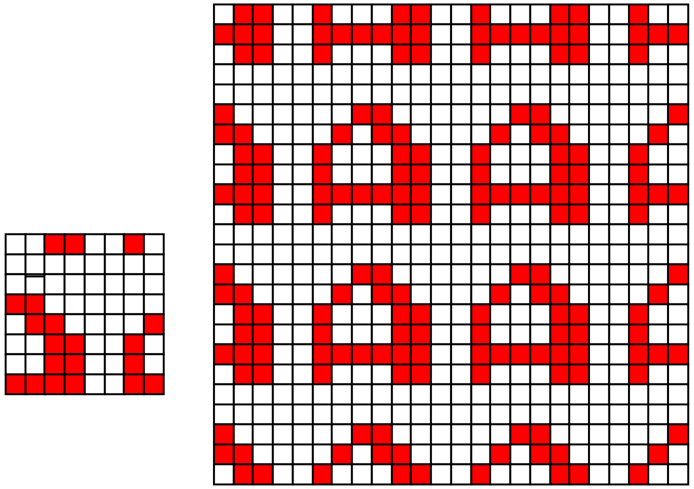


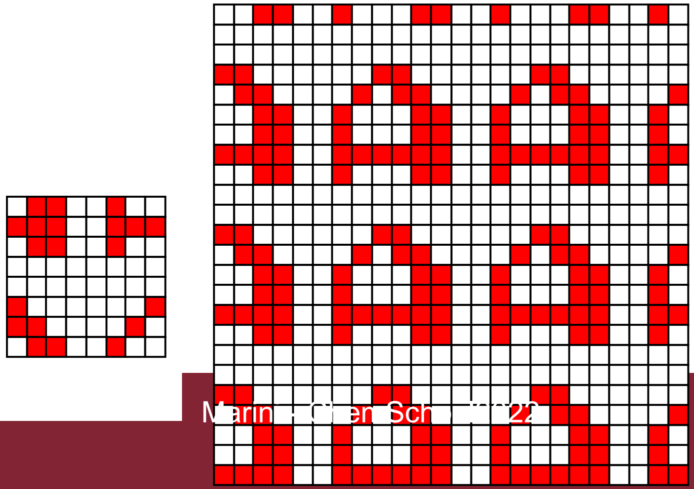
Toroid conditions in a rectangular plane of neurons: 1st neghbours of neurons on edges a and c are neurons on edges b and d respectively.

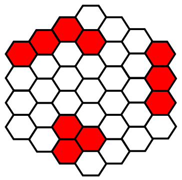


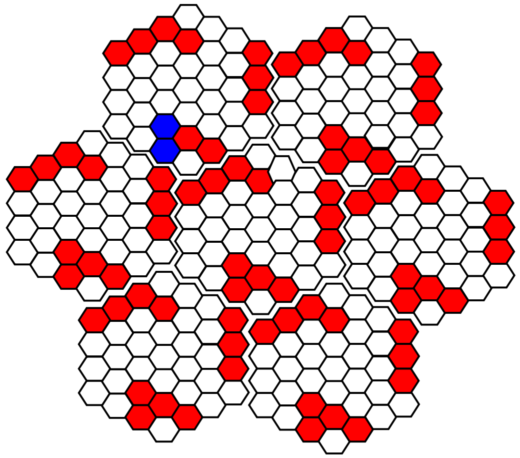
Cyclic and toroid conditions 2

The consideration of the toroidal conditions in Kohonen neural networks can sometimes lead to much clearer results.









Learning procedure in Kohonen networks

Learning in the Kohonen neural network is iterative, i.e., a set of objects $\{X_s\}$ is sent through the network several times.

After the pass of one object through the network the weights, which at the begining of learning are randomised, are changed. The learning procedure of a single pass consists of three steps:

1. selection of one neuron in the network according to a prespecified criterion (largest response, most similar input X_s to the neuron W_j, or similar),
2. correction of the weights of the selected neuron, and
3. correction of the weights of the neighbouring neurons up to a specified range around the selected neuron.

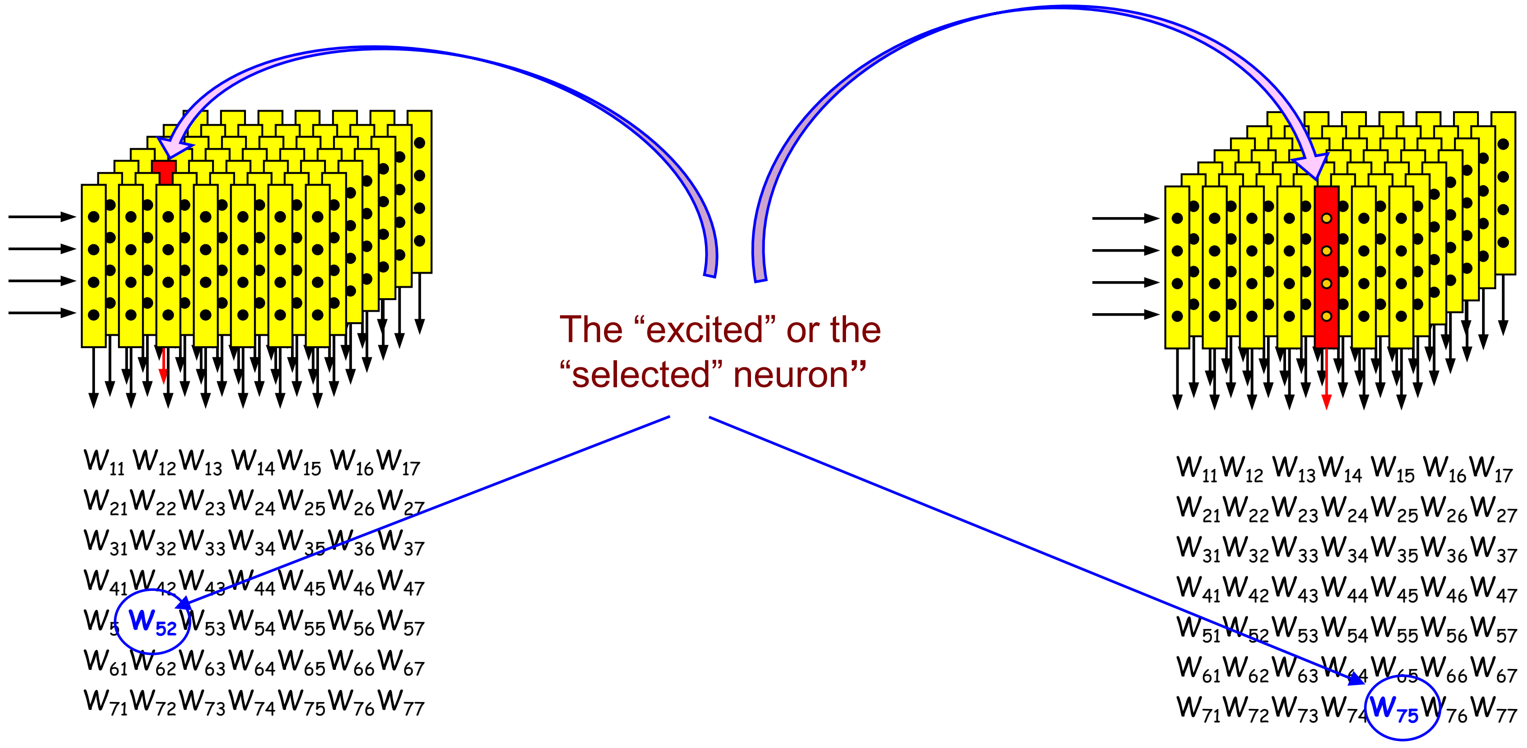
One pass of the entire data set through the network is called one epoch.

Training is stopped after a certain number of epochs. The main result of the Kohonen learning is the “top-map” or “self-organised map” (SOM) of objects associated with the excited neurons.

Kohonen networks in practice

Step one: selection of the “excited”, “central”, or “responding” neuron. In large majority of cases the selection of the neuron is made according to the smallest distance criterion:

$$d(X_s, W_j) = \sum_{i=1}^{m} (x_{si} - w_{ji})^2 \quad for\ all\ \ j = 1,\ \ldots,\ \ N \times N$$

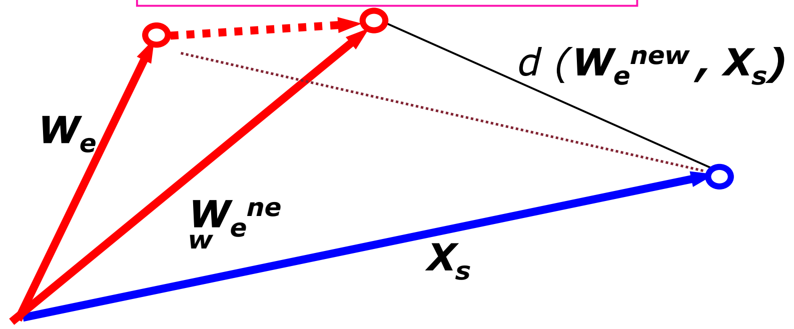

The “excited” or the “selected” neuron”
W11 W12 W13 W14 W15 W16 W17
W21 W22 W23 W24 W25 W26 W27
W31 W32 W33 W34 W35 W36 W37
W41 W42 W43 W44 W45 W46 W47
W51 W52 W53 W54 W55 W56 W57
W61 W62 W63 W64 W65 W66 W67
W71 W72 W73 W74 W75 W76 W77
W11 W12 W13 W14 W15 W16 W17
W21 W22 W23 W24 W25 W26 W27
W31 W32 W33 W34 W35 W36 W37
W41 W42 W43 W44 W45 W46 W47
W51 W52 W53 W54 W55 W56 W57
W61 W62 W63 W64 W65 W66 W67
W71 W72 W73 W74 W75 W76 W77

Kohonen networks in practice 2

Step two: correction of weights of the selected neuron W_e

Because after each pass only one neuron is selected the leaning procedure is called the “winner-takes-all” strategy.

$$\boxed{\otimes W_e = / \ (X_s - W_e^{old})}$$



$$\boxed{/ \ = \ (/^{start} - /^{final})(1 - n_{epoch}/n_{tot}) + /^{final}}$$

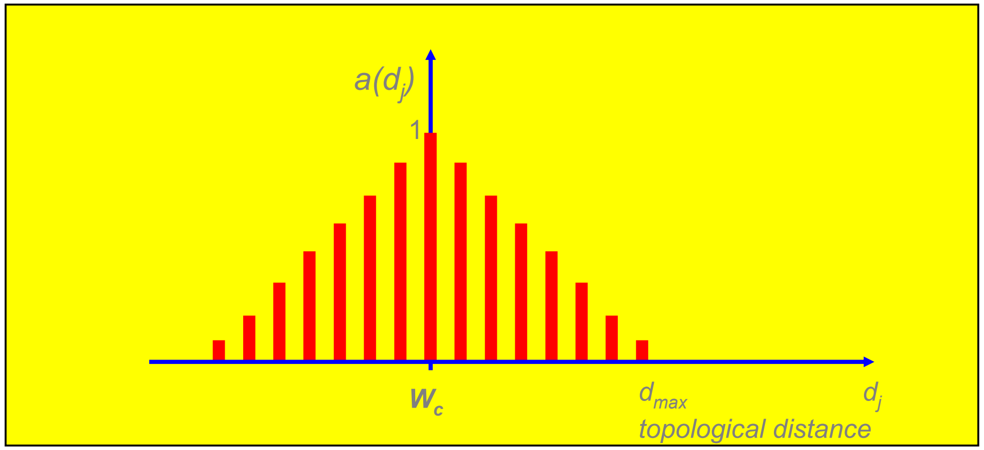
The parameter η is called the learning rate and is in most applications “time” dependent, i.e. dependent on the number of currently performed learning epochs.

The corrections $\otimes W_e$ are driving the weights w_{ei} of the excited neuron $W_e = (w_{e1}, w_{e2}, \dots w_{ei} \dots w_{em})$ closer to the variables x_{si} of object $X_s = (x_{s1}, x_{s2}, \dots x_{si}, \dots x_{sm})$ that has excited it.

Kohonen networks in practice 3

Step three: correction of the weights in neurons W_j surrounding the selected neuron W_e

$$\Delta w_{ji} = \eta a(d_j)\left(x_{si} - w_{ji}^{old}\right)$$



$$a(d_j) = 1 - \frac{d_j}{(d_{\max}+1)}$$

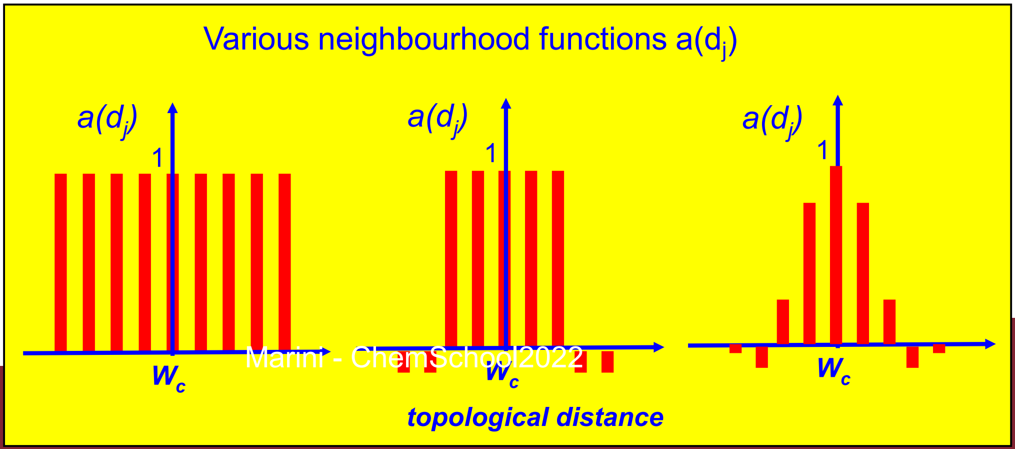
$$\Delta w_{ji} = \eta\left(1 - \frac{d_j}{d_{\max}+1}\right)(x_{si} - w_{ji}^{old})$$

$$\Delta w_{ji} = \eta\left(1 - \frac{d_j}{N_{net}\left(1 - \frac{n_{epoch}}{n_{tot}}\right)+1}\right)(x_{si} - w_{ji}^{old})$$

For the selected neuron *a(d_j) = 1* because *d_j = 0*.

For the neurons separated d_{max} from the selected neuron W_e the value $a(d_{max})$ is the smallest posible corection.

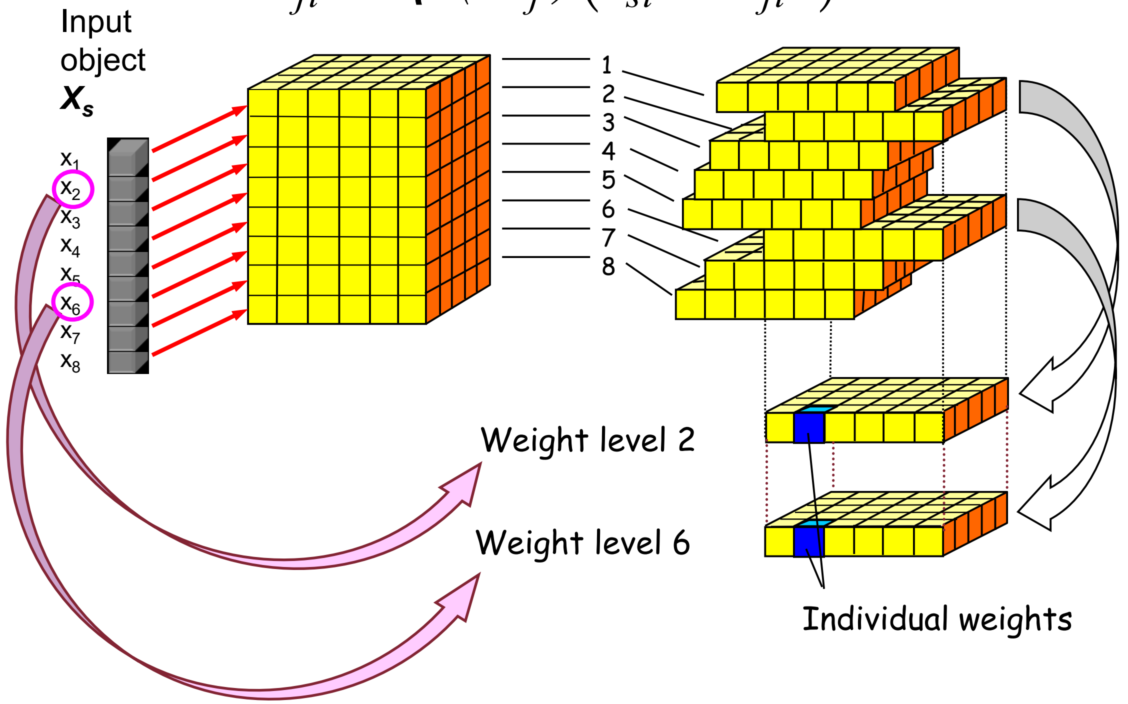
At the beginning of learning:
$n_{epoch} = 1$ ➔ $d_{max} = N_{net}$; correction covers the entire network
At the end of the learning:
$n_{epoch} = n_{tot}$ ➔ $d_{max} = 0$; correction is limited only to the selected neuron W_e.



Weight maps

Correction of the weights in neurons W_j is applied to all levels of weights. Due to the fact that weights w_{ji} in the neurons of the Kohonen network are alligned into levels according to the order of input variables x_i, each level of weights is influenced only by one variable and thus forms a map of weight values or weight map.

$$\Delta w_{ji} = \eta a(d_j)\left(x_{si} - w_{ji}^{old}\right)$$



Weight maps 2

.76	.85	.90	.93	.95	.97	.99
.63	.75	.84	.80	.85	.89	.95
.59	.66	.68	.65	.77	.75	.92
.32	.33	.52	.58	.60	.69	.85
.31	43	.46	.42	.53	.61	.81
.16	.25	.30	.33	.46	.55	.75
.03	.09	.08	.22	.38	.50	.67

Weight map of the variable x_2 - *normalised binder concentration*

A	A	A	A,C			B
A	A		A,C	C		B
A		C	C	C		B
	C	C	C			B
C	C	C			B	B
C	C	C,B	B	B	B	B
C	C		B	B	B	B

Top map showing the cells containing the information avout the overal quality of the paint-coat

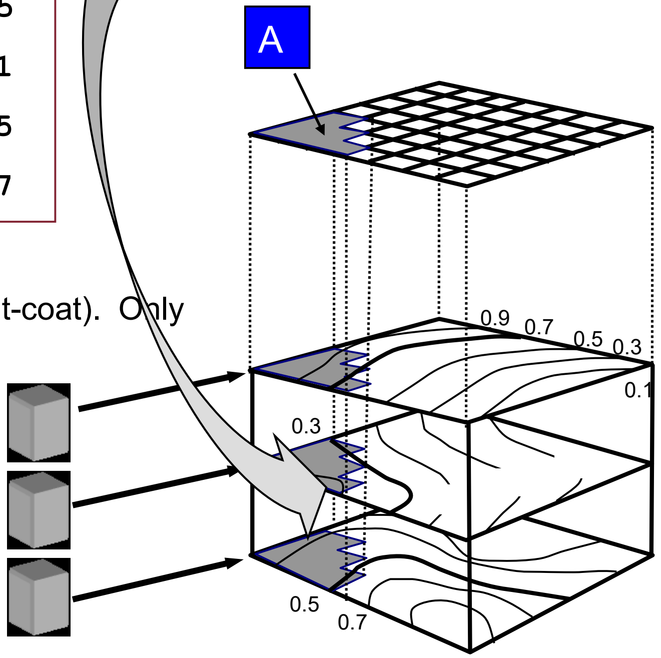
Labels **A**, **B**, and **C** refere to the excellent, passable, and below standard quality, respectively.

Input object $\boldsymbol{X}_s$ (recipe for the paint-coat). Only three variables are shown.

Solvent x_1

Binder x_2

Pigment x



The most important feature of the Kohonen network is the fact that all weights in all neurons regardless whether they were excited during the trainng or not bear valuable information.

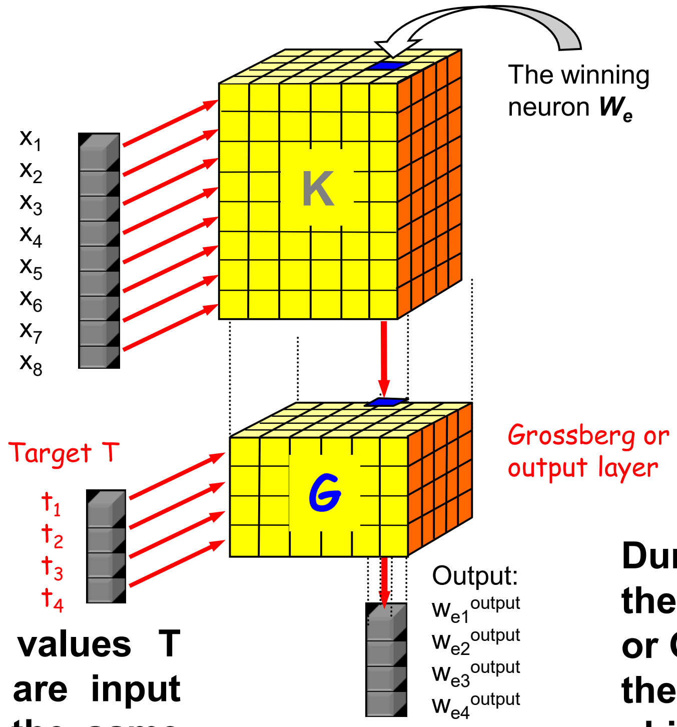
Counterpropagation networks

By the addition of an identical layer of weights to which the targets are input the Kohonen network is transformed into the counterpropagation network.

The additional layer (the output or Grossberg layer) has the same number and the same layout of neurons as the first (Kohonen) layer, however, in each layer the neurons have different number of weights.

$$\Delta w_{ji}^{K} = \eta a(d_j)\left(x_{si} - w_{ji}^{K,old}\right)$$

$$\Delta w_{ji}^{G} = \eta a(d_j)\left(t_{si} - w_{ji}^{G,old}\right)$$

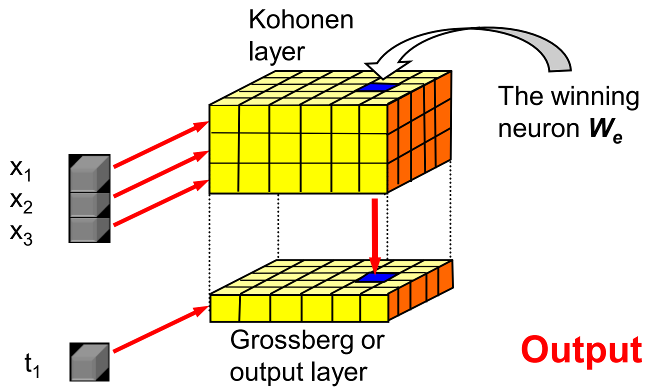


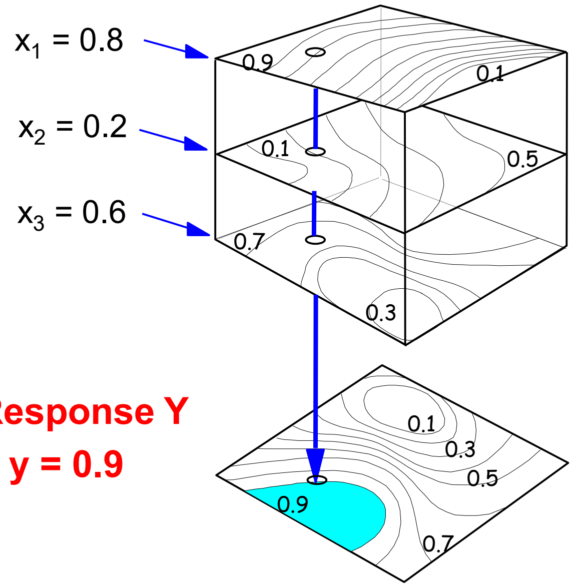
During the training the target values T associated with each object X are input into the output layer in exactly the same manner as the objects X are input into the Kohonen layer.

During the retrieval (prediction) the weights w_{ei}^{output} in the output or Grossberg layer selected by the winning neuron of the input object X (in the Kohonen layer) are used as the predictions.

Counterpropagation networks 2

Each counterpropagation network can be used as a direct and as an inverse "model".





Response Y

$y = 0.9$

Output X

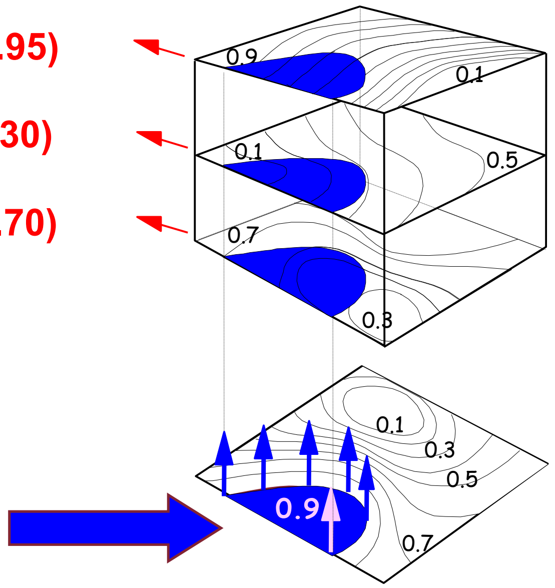
$x_1 = (0.45 - 0.95)$

$x_2 = (0.00 - 0.30)$

$x_3 = (0.30 - 0.70)$

Input Y

$y = 0.9$

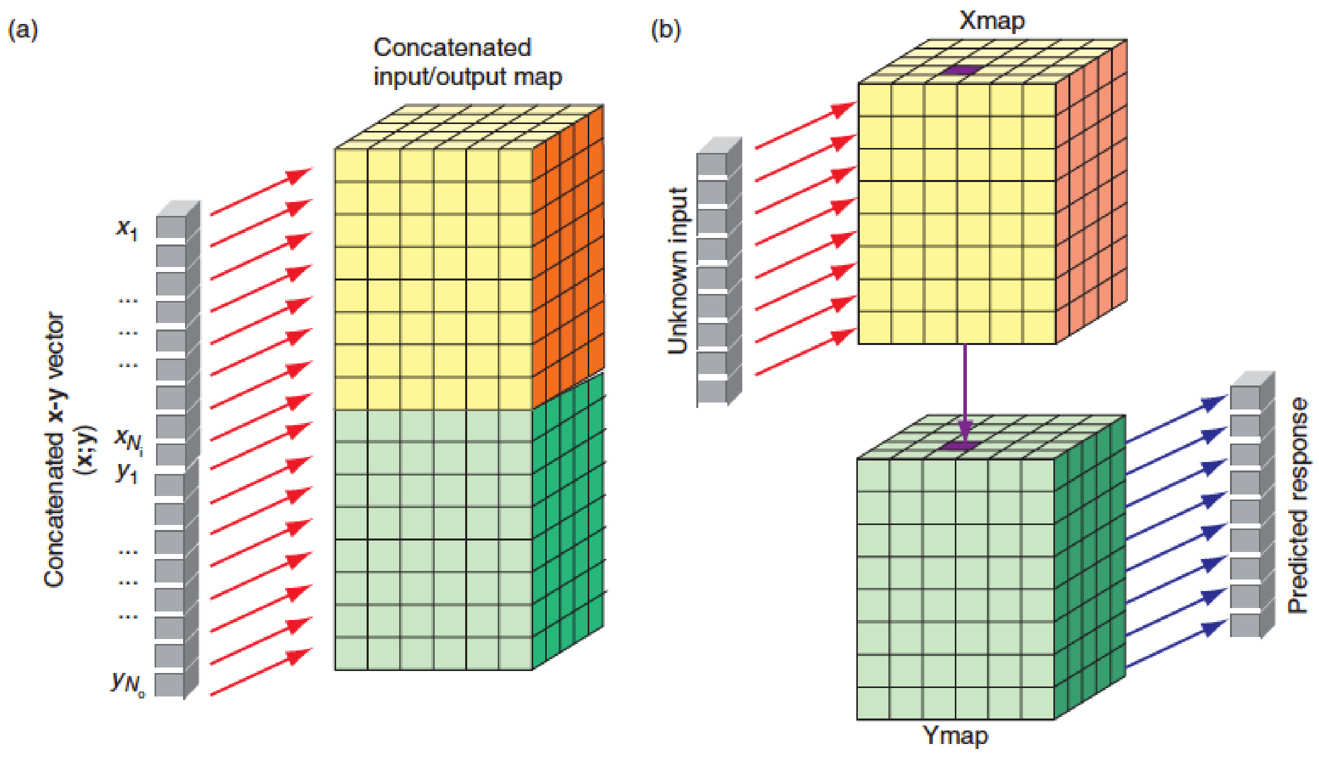


Supervised Kohonen architectures

- Counterpropagation networks are semi-supervised architectures, as the value of the Y vector doesn’t drive the selection of the winning neuron and, as a consequence, the direction of the training.
- There are other proposed architectures that are truly supervised and that can be used to build classification models:
    - Supervised Kohonen networks
    - XY-fused network
    - Bidirectional Kohonen networks

Supervised Kohonen networks

- X and Y variables are concatenated to train the network as in the standard Kohonen architecture.
- After training the two blocks are separated and prediction occur as in counterpropagation



XY-fused networks

- The winning neuron is decided by considering a similarity function which is a weighted sum of the similarity in the X space and in the Y space.
- The parameter α starts with a high value and then decreases, so that at the end only similarity in the Y space contributes, while at the beginning the X space is dominant.

$$S_{\text{fused}}(j) = \alpha(t)S(\mathbf{x}, \mathbf{w}_j) + (1-\alpha(t))S(\mathbf{y}, \mathbf{u}_j)$$

Bidirectional Kohonen networks

- The concept of BDK networks is similar to that of XY-fused, combining similarity in both spaces to update the weights.
- In BDK, however, X and Y weights are updated in an alternate fashion:
- The parameter α starts with a high value and then decreases, so in the beginning similarity in the Y space governs selection of the winning neuron in the X space while similarity in the X spaces de

$$S_{\text{Winner}X}(j) = (1-\alpha(t))S(\mathbf{x},\mathbf{w}_j) + \alpha(t)S(\mathbf{y},\mathbf{u}_j)$$

$$S_{\text{Winner}Y}(j) = \alpha(t)S(\mathbf{x},\mathbf{w}_j) + (1-\alpha(t))S(\mathbf{y},\mathbf{u}_j)$$

The fortunes and misfortunes of NNs

- Despite their increasing popularity up to the beginning of the 2000s, interest in neural networks seemed to vanish:
  - Curse of dimensionality
  - Inefficient learning algorithms
  - Lack of interpretability of the models
  - Performances heavily dependent on the choice of data representation (or features) on which they are applied.

- Much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations that result in a representation of the data that can support effective machine learning

Representation learning

- Learning representations of the data that make it easier to extract useful information when building classifiers or other predictors
  - Captures the posterior distribution of the underlying explanatory factors for the observed input.
  - Is useful as input to a supervised predictor.
- DEEP LEARNING:
  - composition of multiple non-linear transformations, with the goal of yielding more abstract – and ultimately more useful – representations
- Fundamental questions:
  - What makes one representation better than another?
  - Given an example, how should we compute its representation, i.e. perform feature extraction?
  - What are appropriate objectives for learning good representations?

Representation learning: “chemometric” concepts

- Representations → convenient to express many general (not task-specific) priors that are likely to be useful for a learning machine to solve AI-tasks.
- The revival experienced by NNs in the recent years has much to do with such priors
  - Their absence was one of the main reasons for the vanishing interest towards NN in the 2000s
  - They share much in common with essential ideas which make chemometric representations useful and versatile
  - Possibility of mutual benefit between the disciplines
- Some of these will be briefly discussed in the following

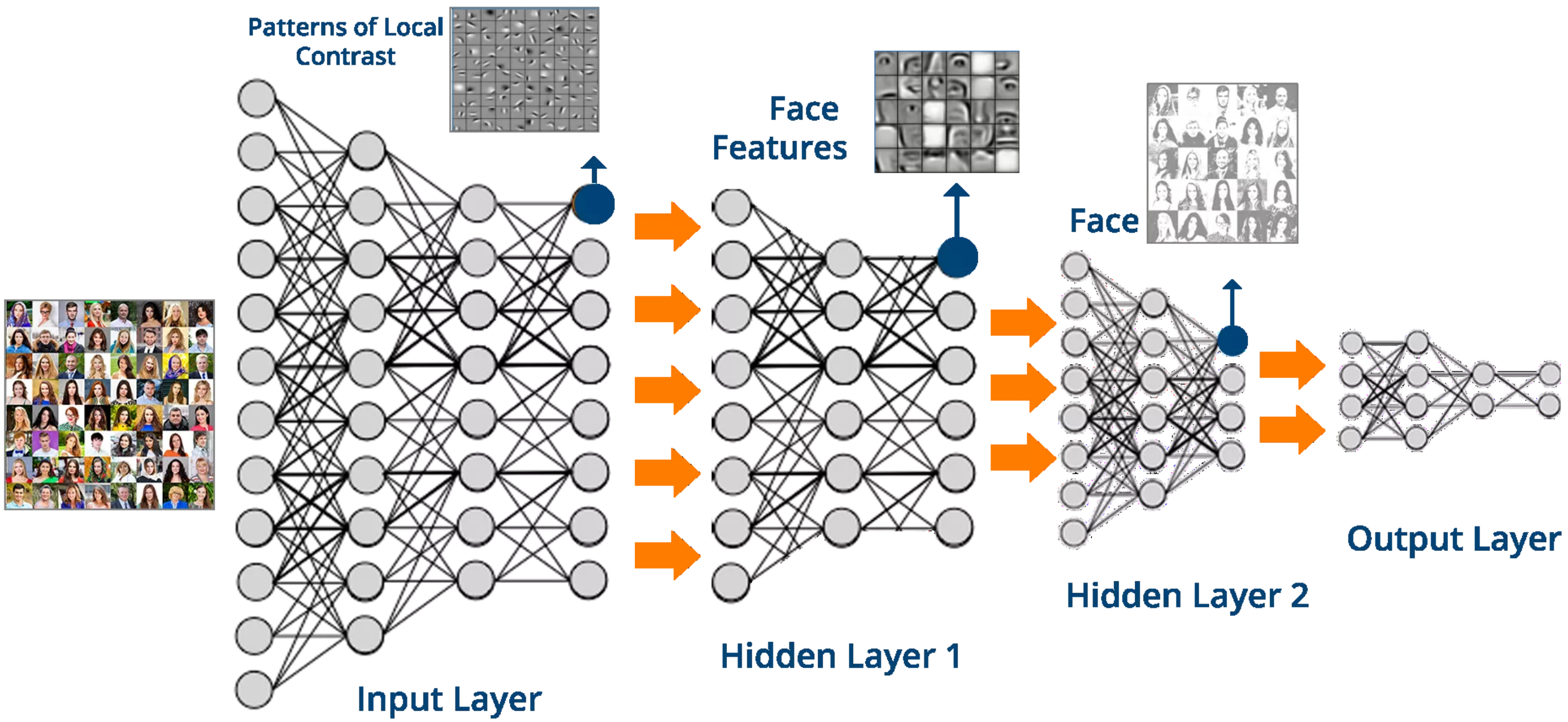
The general priors of representation learning

- **Smoothness**:
  - $x_1 \approx x_2 \Rightarrow f(x_1) \approx f(x_2)$
  - Insufficient to circumvent the curse of dimensionality
- **Multiple explanatory factors**:
  - Data distribution generated by different underlying factors
  - What one learns about one factor generalizes in many configurations of the other factors
  - The objective is to recover or at least disentangle these underlying factors of variation
- **Hierarchy**:
  - The features that are useful for describing the world around us can be defined in terms of other features, in a hierarchy
  - More abstract concepts higher in the hierarchy are defined in terms of less abstract ones $\rightarrow$ **Deep learning**

The general priors of representation learning - 2

- **Semi-supervised learning**:
  - A subset of the factors explaining X’s distribution explain much of Y, given X.
  - Representations that are useful for P(X) tend to be useful when learning P(Y|X)
  - Sharing of statistical strength between the unsupervised and supervised learning tasks
- **Manifolds**:
  - Probability mass concentrates near regions that have a much smaller dimensionality than the original space where the data lives.
- **Sparsity**:
  - For any given x, only a small fraction of the possible factors are relevant.
  - Features that are often zero or insensitive to small variations of x.
  - Priors on latent variables (peaked at 0), or by a nonlinearity whose value is often flat at 0 (e.g., ReLU)

Building deep representations

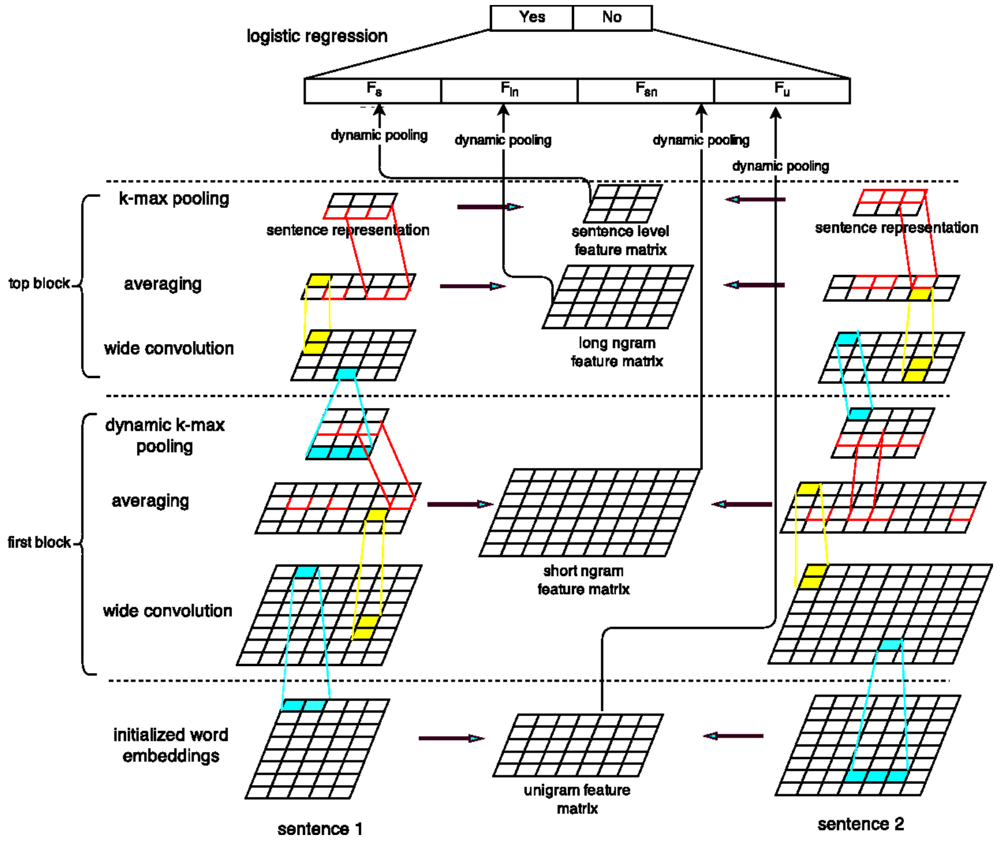


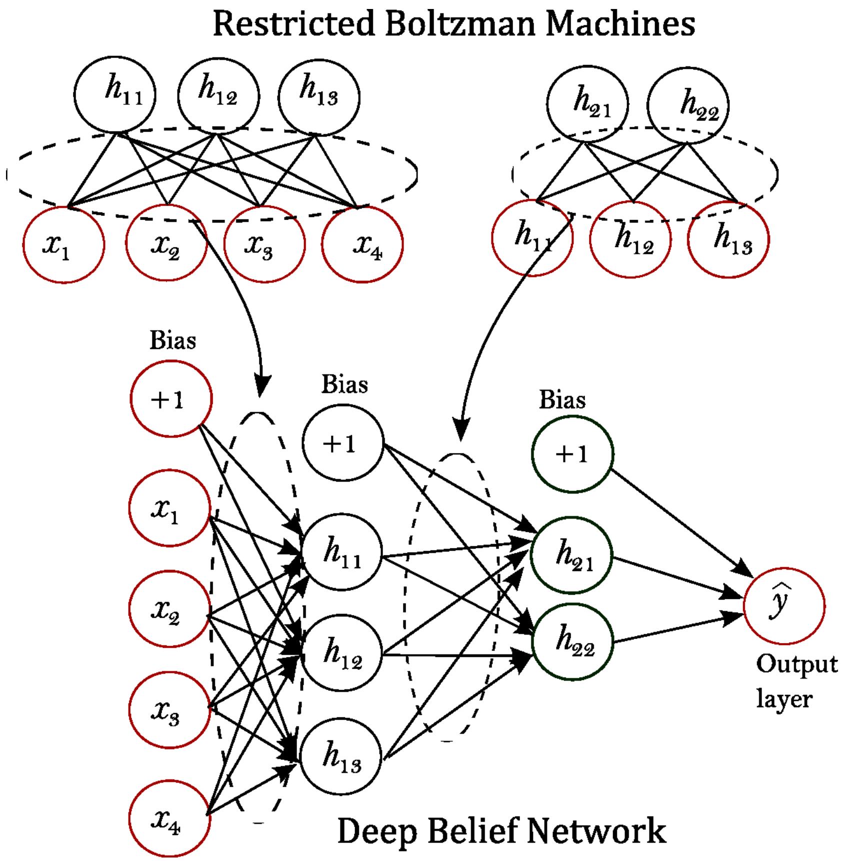
- Learn a hierarchy of features one level at a time, using unsupervised feature learning to learn a new transformation at each level to be composed with the previously learned transformations;
- The set of layers could be combined to initialize a deep supervised predictor, such as a neural network classifier, or a deep Boltzmann Machine

Two learning paradigms

- One rooted in probabilistic graphical models and one rooted in neural networks.
- Probabilistic modeling:
  - Attempt to recover a parsimonious set of latent random variables that describe a distribution over the observed data.
  - Feature values are conceived as the result of an inference process to determine the probability distribution of the latent variables given the data, i.e. p(h|x), the posterior probability.
- Main differences:
  - The layered architecture of a deep learning model is to be interpreted as describing a probabilistic graphical model or as describing a computation graph?
  - Are hidden units considered latent random variables or as computational nodes?

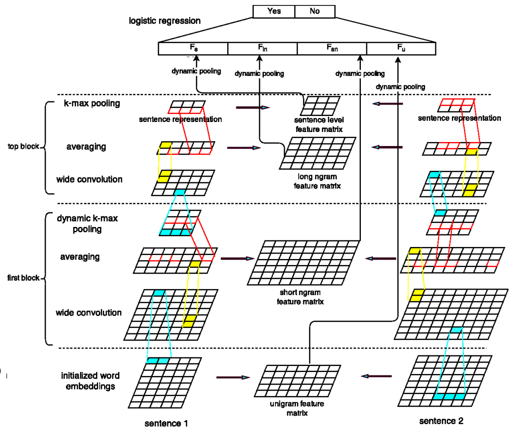
And two corresponding architectures





- Convolutional NN
- (Restricted) Boltzmann machines

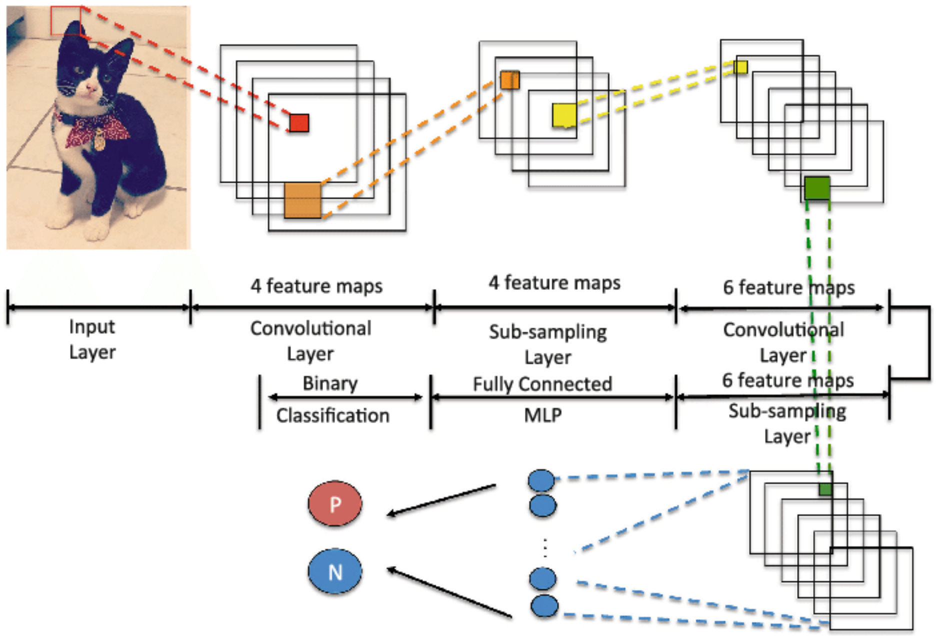
Convolutional Neural Networks



- Designed to process data that come in the form of multiple arrays:
  - a colour image composed of three 2D arrays containing pixel intensities in the three colour channels
  - 1D for signals and sequences
- There are four key ideas behind ConvNets that take advantage of the properties of natural signals:
  - local connections
  - shared weights
  - pooling
  - the use of many layers.

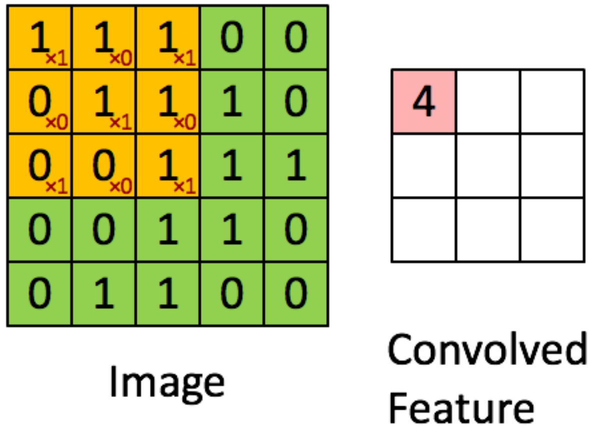
Convolutional Neural Networks

- The architecture is structured as a series of stages.
- The first few stages are composed of two types of layers: convolutional layers and pooling layers.
  - Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank.



The convolutional Concept

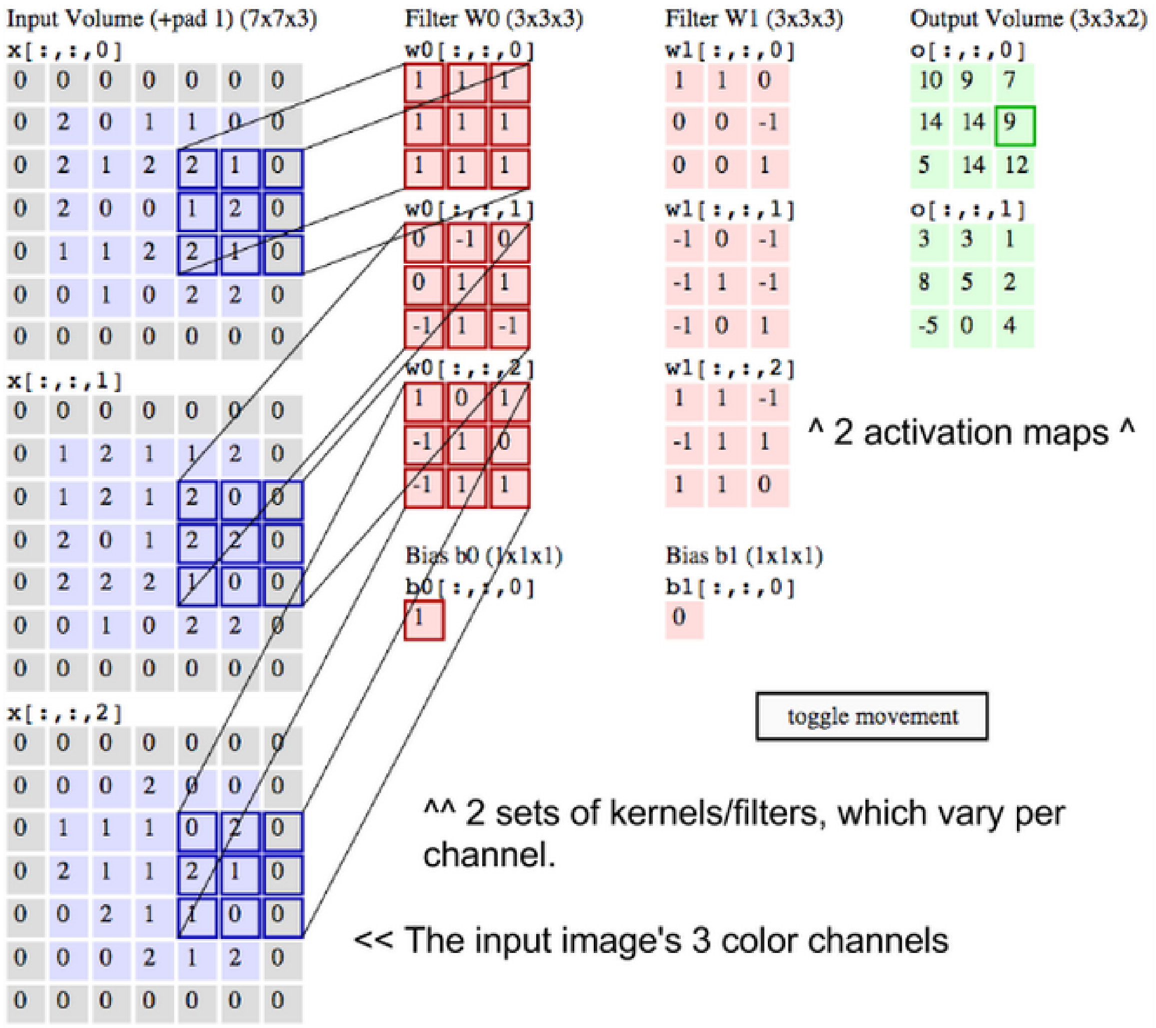
- Convolution extracts feature from the input image (data)
- Preserves the spatial relationship between pixels



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

And for multiple-layered inputs

- Filtering proceeds in parallel across the depth of the image



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	0	1	1	0	0
0	2	1	2	2	1	0
0	2	0	0	1	2	0
0	1	1	2	2	1	0
0	0	1	0	2	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	2	1	1	2	0
0	1	2	1	2	0	0
0	2	0	1	2	2	0
0	2	2	2	1	0	0
0	0	1	0	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	2	0	0	0
0	1	1	1	0	2	0
0	2	1	1	2	1	0
0	0	2	1	1	0	0
0	0	0	2	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	1
1	1	1
1	1	1

w0[:,:,1]

0	-1	0
0	1	1
-1	1	-1

w0[:,:,2]

1	0	1
-1	1	0
-1	1	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

1	1	0
0	0	-1
0	0	1

w1[:,:,1]

-1	0	-1
-1	1	-1
-1	0	1

w1[:,:,2]

1	1	-1
-1	1	1
1	1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

10	9	7
14	14	9
5	14	12

o[:,:,1]

3	3	1
8	5	2
-5	0	4

^ 2 activation maps ^

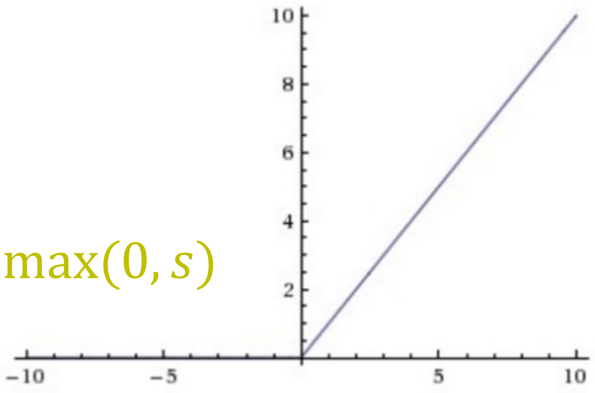
^^ 2 sets of kernels/filters, which vary per channel.

<< The input image's 3 color channels

Going nonlinear

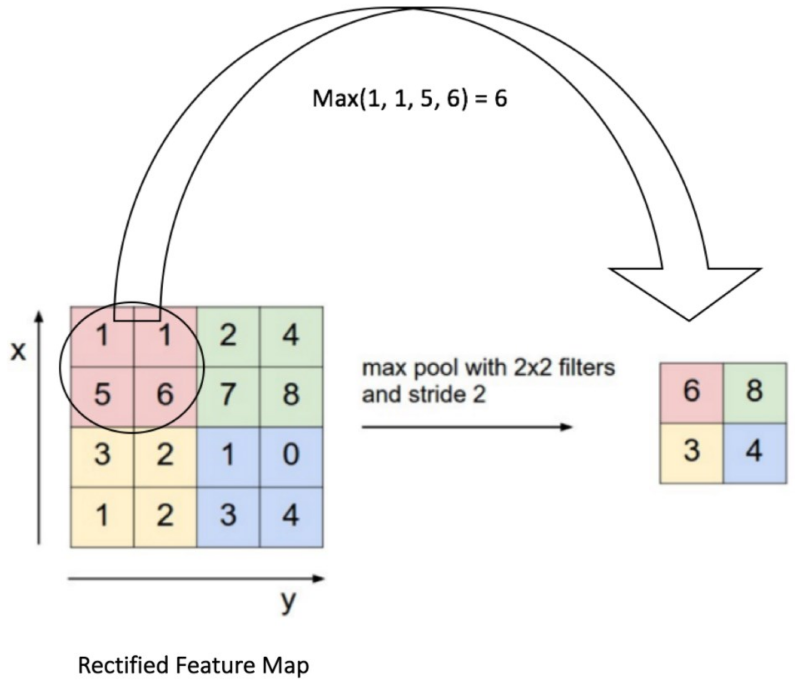
- A nonlinear operation is added on top of the convolution.
- Usually it is carried out by means of a Rectified Linear Unit (but one could also use, sigmoid or hyperbolic tangent)
- ReLU is an element wise operation (applied per pixel to the activation maps) and replaces all negative pixel values in the feature map by zero.

$$h = \max(0, s)$$



Compressing (Pooling)

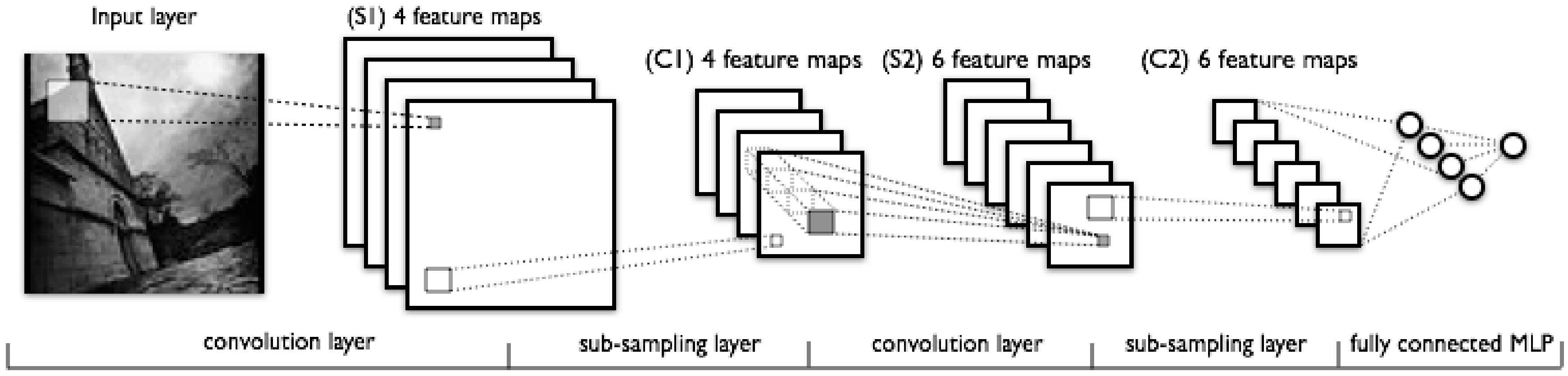
- Spatial Pooling reduces the dimensionality of each feature map but retains the most important information.
- Can be of different types: Max, Average, Sum etc.
- Define a spatial neighborhood (for example, a 2 × 2 window) and:
  - take the largest element from the rectified feature map within that window (Max pooling)
  - Take the average (Average Pooling) or sum of all elements in that window
  - Max Pooling has been shown to work better.



Compressing (Pooling) - 2

- Pooling is applied separately to each feature map
- The function of Pooling is to progressively reduce the spatial size of the input representation:
  - makes the input representations (feature dimension) smaller and more manageable
  - reduces the number of parameters and computations in the network, therefore, controlling overfitting
  - makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
  - almost scale invariant representation of our image (the exact term is “equivariant”)→detect objects in an image no matter where they are located.

Wrapping up


Input layer
(S1) 4 feature maps
(C1) 4 feature maps
(S2) 6 feature maps
(C2) 6 feature maps
convolution layer
sub-sampling layer
convolution layer
sub-sampling layer
fully connected MLP

Generative Topographic Mapping (GTM)

Eric Latrille – LBE –INRAE (France

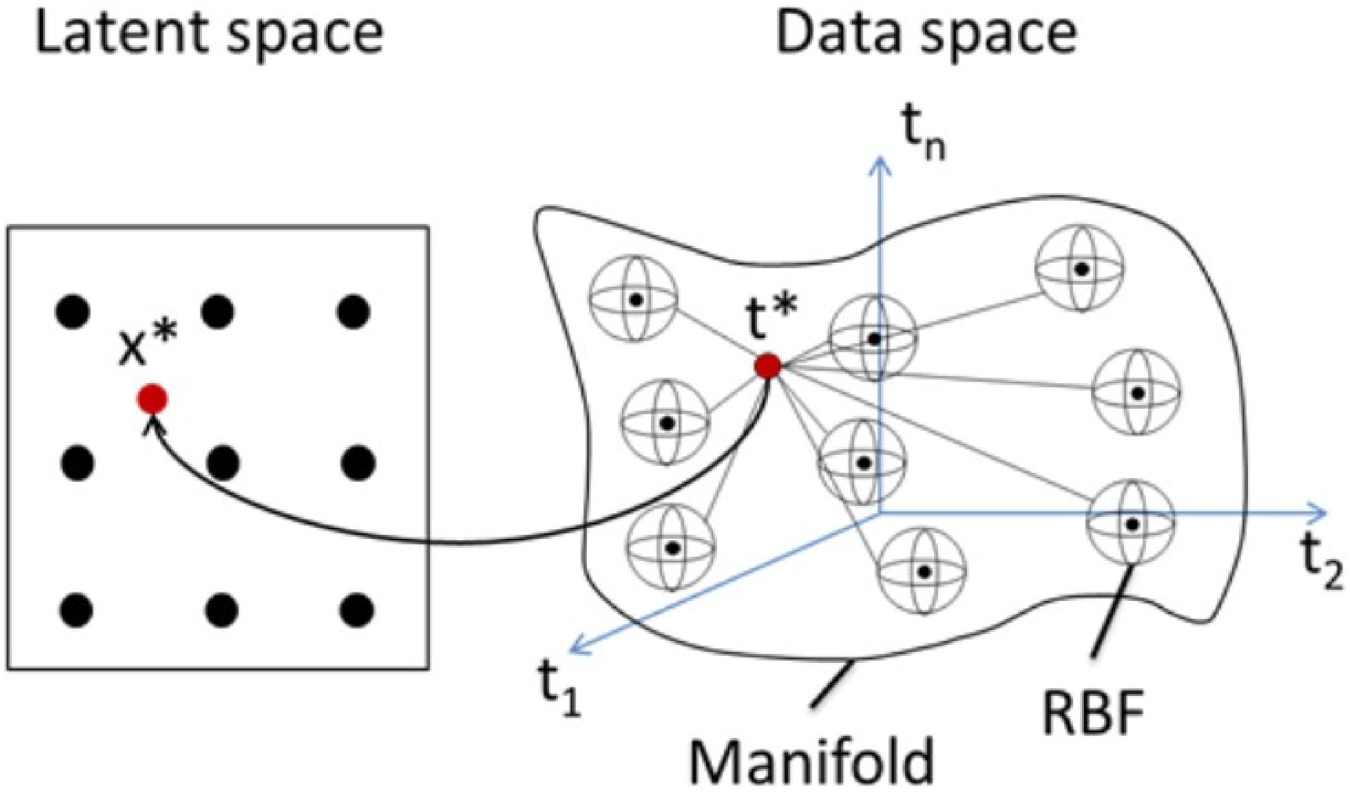
GTM is a dimensionality reduction algorithm well described by Bishop *et al*.

Briefly speaking, the algorithm injects a 2D hypersurface (*manifold*) into an initial *D*dimensional data space. The manifold is fitted to the data distribution by the ExpectationMaximization (*EM*) algorithm which minimizes the log-likelihood of the training data.

Once the fitting is done, each item from the data space is projected to a 2D latent grid of *K* nodes.

GTM is a probabilistic extension of SOM where log-likelihood is utilized as an objective function.

The manifold used to bind a data point **t*** in the data space and its projection **x*** in the latent space is described by a set of *M* Radial Basis Function centers (*RBF*; Gaussian functions are generally used).


Latent space
Data space
x*
t*
t_n
t_1
t_2
Manifold
RBF