DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Lecture 5: **Radial basis function NN and introduction to competitive learning**

Pawel Herman

Computational Science and Technology (CST)

KTH Royal Institute of Technology

January 2018

Lecture overview

- Interpolation problem and radial-basis functions (RBFs)
- RBF networks – hybrid learning
- Weight interpretation
- Competitive mechanisms for unsupervised learning

The separability of patterns

Cover's theorem

"A complex pattern classification problem, projected nonlinearly to a high-dimensional space, is more likely to be linearly separable than in low dimensional space, especially if it is not populated too densely."

Cover, 1965

The separability of patterns

Cover's theorem

"A complex pattern classification problem, projected nonlinearly to a high-dimensional space, is more likely to be linearly separable than in low dimensional space, especially if it is not populated too densely."

Cover, 1965

So, we need to have:

1) Many nonlinear mappings $\varphi_i(\mathbf{x} \in \mathrm{R}^M)$: $\mathrm{R}^M \rightarrow \mathrm{R}^1$, where i =1,.., N (large)
2) Linear function for separability in high N-dimensional space

The radial-basis-function (RBF) technique

Nonlinear mapping with the use of *radial-basis-functions* (RBFs):

$$\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

$\mathbf{x}_i$ – RBF centre

$\|\cdot\|$ – vector norm, often Euclidean

$\varphi_i(r)$ – kernel function, often Gaussian

The radial-basis-function (RBF) technique

Nonlinear mapping with the use of *radial-basis-functions* (RBFs):

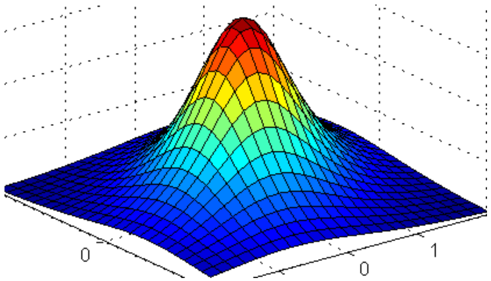
$$\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

$\mathbf{x}_i$ – RBF centre

$\|\cdot\|$ – vector norm, often Euclidean

$\varphi_i(r)$ – kernel function, often Gaussian

$$\varphi_i\left(r=\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)=\exp\left(-\frac{r^2}{2\sigma^2}\right)$$



The radial-basis-function (RBF) technique

Nonlinear mapping with the use of *radial-basis-functions* (RBFs):

$$\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

$\mathbf{x}_i$ – RBF centre

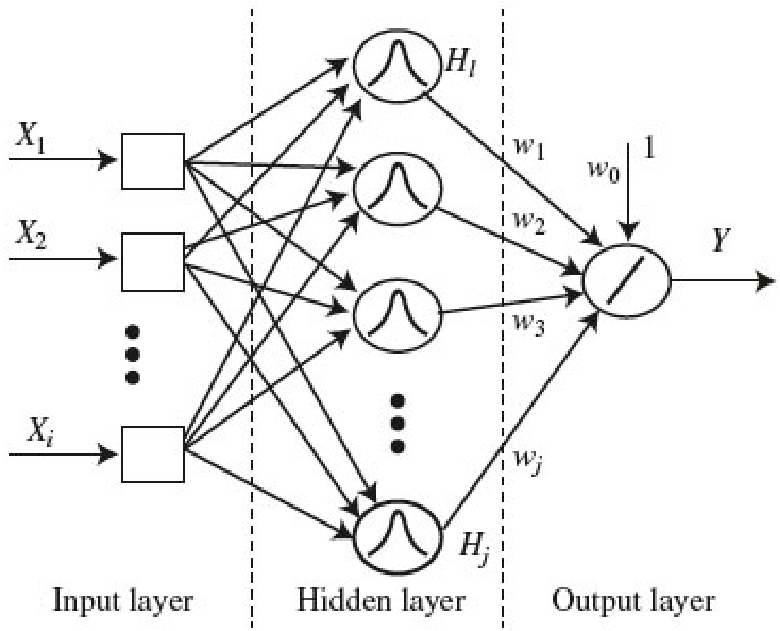
$\|\cdot\|$ – vector norm, often Euclidean

$\varphi_i(r)$ – kernel function, often Gaussian

Linear operation in N-dimensional space:

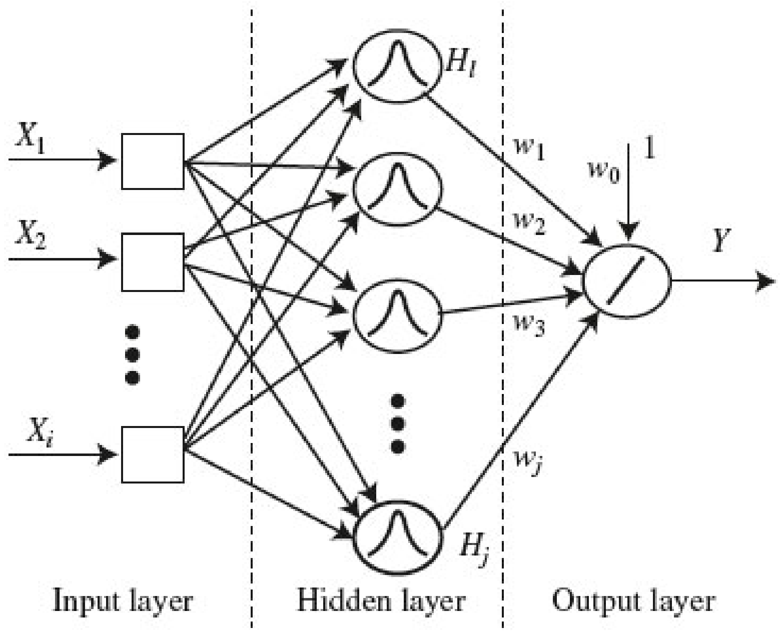
$$F(\mathbf{x})=\sum_{i=1}^{N} w_i\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

The RBF neural network concept



In the *exact interpolation*, the size of the hidden layer, N, is equal to the number of samples n

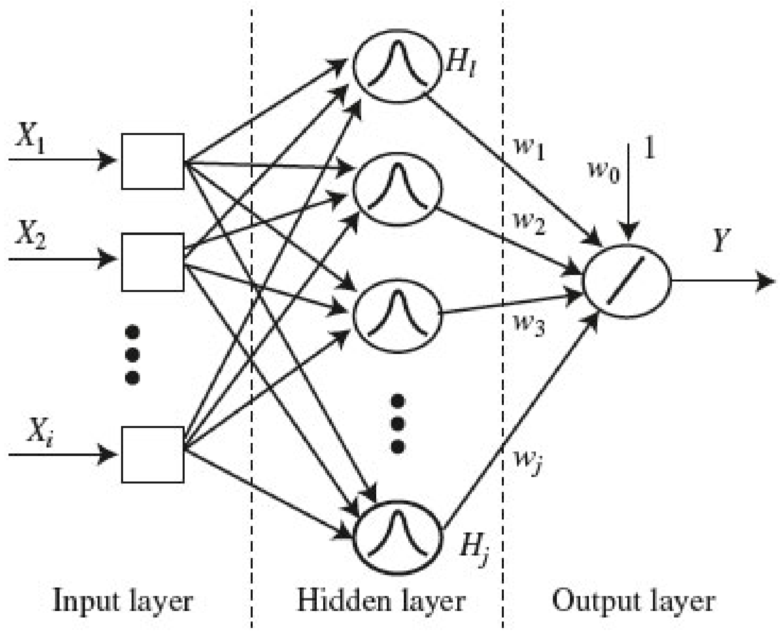
The RBF neural network concept



In the *exact interpolation*, the size of the hidden layer, N, is equal to the number of samples n

BUT this is not robust especially if a lot of samples are corrupted with noise!

The RBF neural network concept



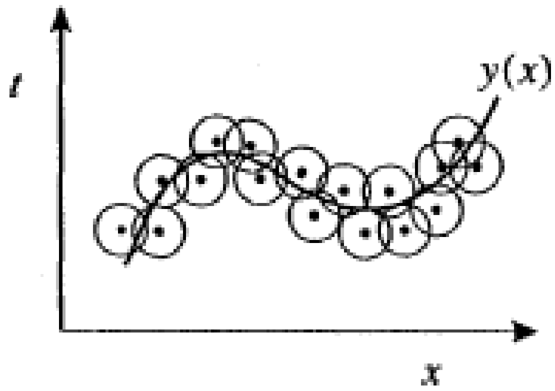
In the *exact interpolation*, the size of the hidden layer, N, is equal to the number of samples n

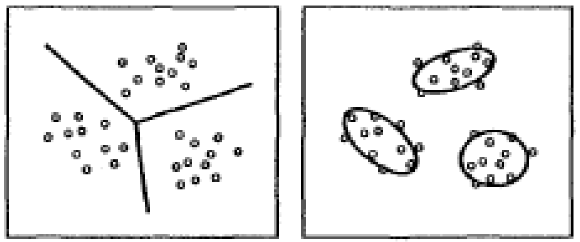
BUT this is not robust especially if a lot of samples are corrupted with noise!

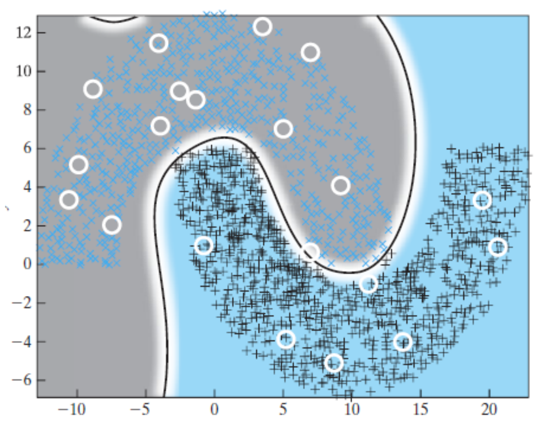
We need modifications:

1) $N < n$
2) centres $\mathbf{x}_i$ different from samples
3) widths, σ, also differ across RBF nodes
4) it is possible to include biases

Examples

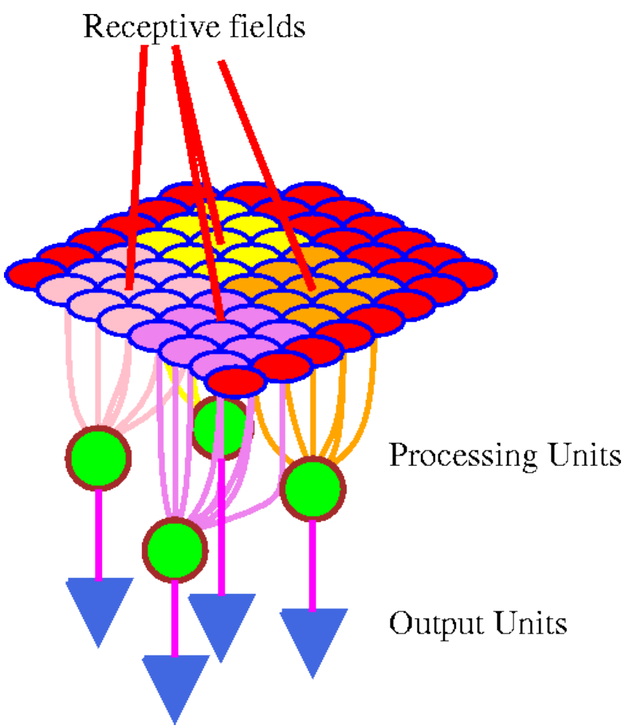






See *Bishop (1995)* for Bayesian interpretation of classification with RBF networks

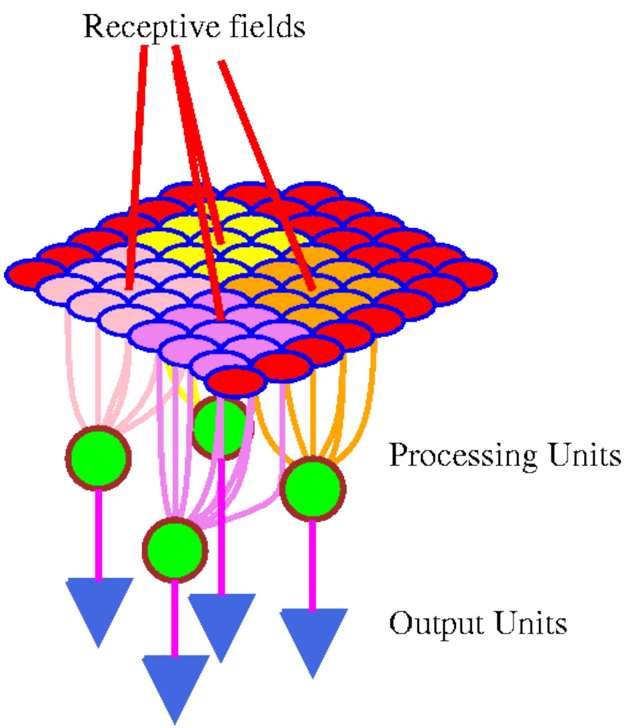
Analogy with receptive fields



The **receptive field** of a unit is the region of the input space from which a stimulus pattern evokes a response.

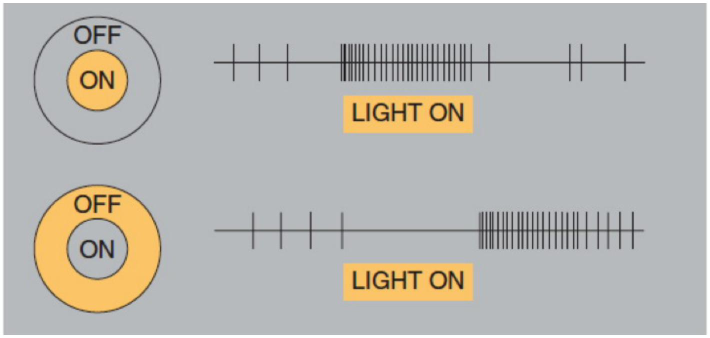
Haykin, 2009

Analogy with receptive fields

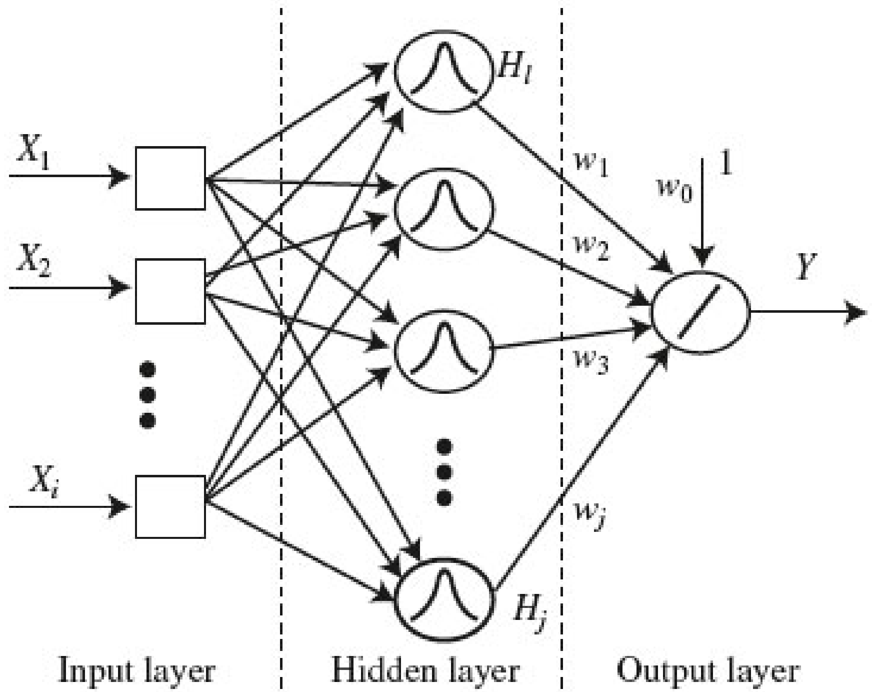


Example:

Retinal ON-centre and OFF-centre cells



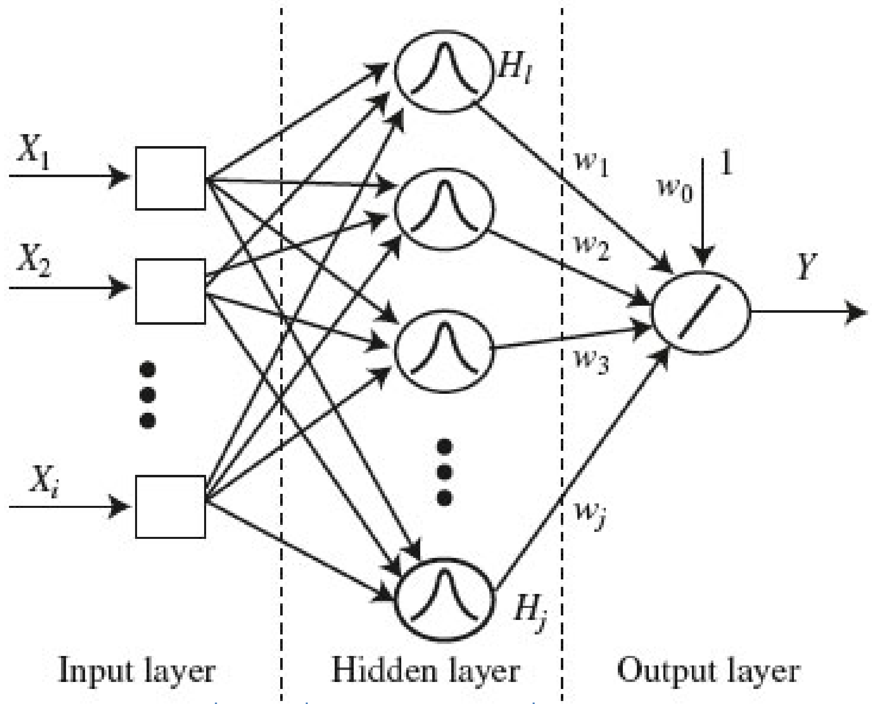
Hybrid learning algorithm



M **input** units N **hidden** units

n **training** samples (each sample is N-dimensional)

Hybrid learning algorithm



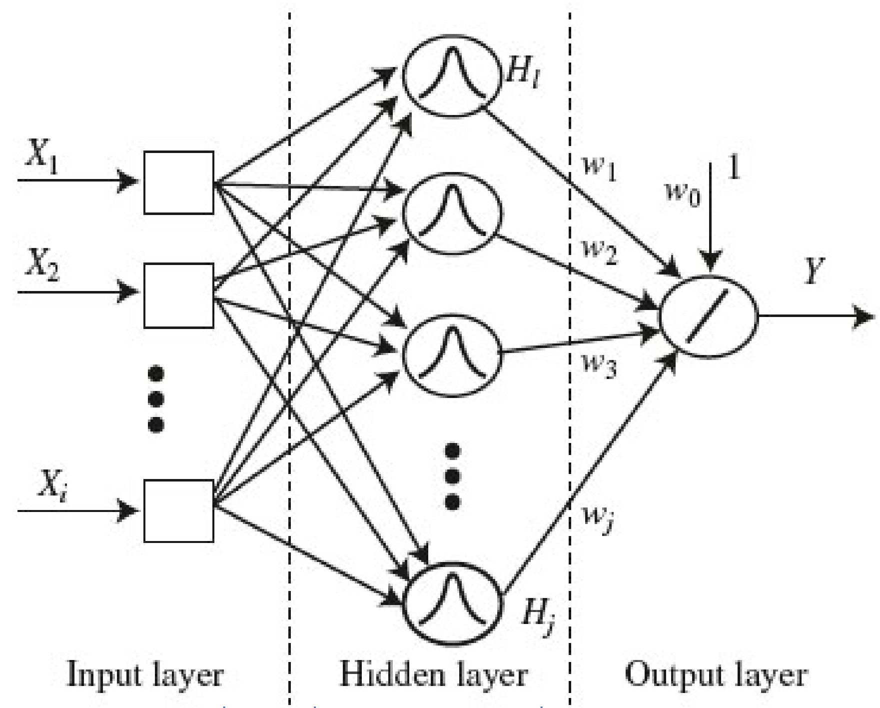
M **input** units

N **hidden** units

n **training** samples (each sample is N-dimensional)

Size of the **input layer** determined by the dimensionality of the input.

Hybrid learning algorithm



M **input** units

N **hidden** units

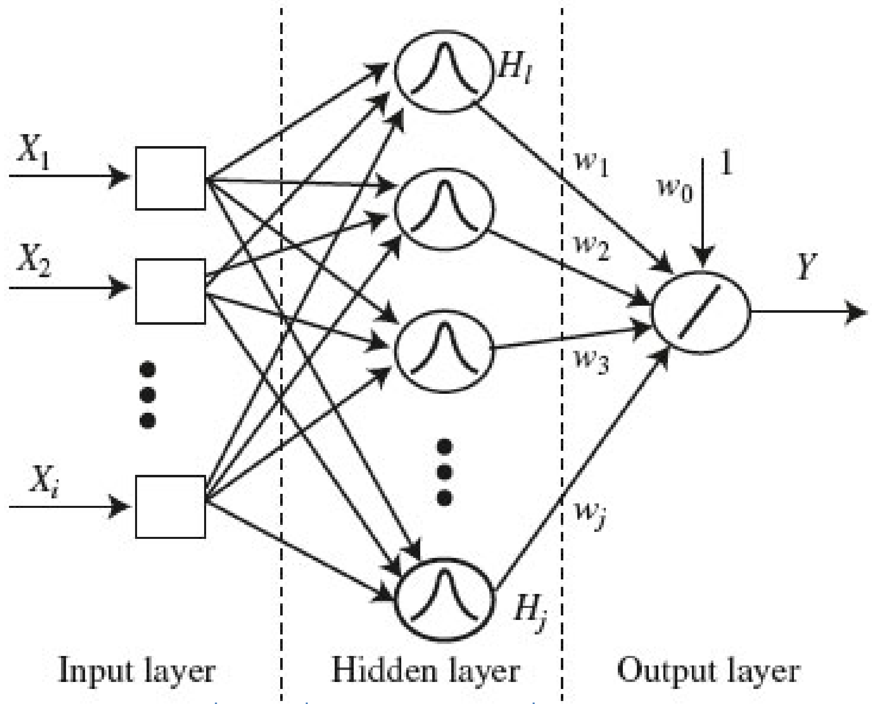
n **training** samples (each sample is N-dimensional)

Size of the **input layer** determined by the dimensionality of the input.

Hidden layer

- N has to be decided
- centres and widths of hidden units have to be identified

Hybrid learning algorithm



M **input** units

N **hidden** units

n **training** samples (each sample is N-dimensional)

Size of the **input layer** determined by the dimensionality of the input.

Hidden layer

- N has to be decided
- centres and widths of hidden units have to be identified

Output layer performs a linear mapping – e.g., training with least square methods

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ . & . & . & . \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ . \\ d_n \end{bmatrix}$$

Hybrid learning – RBF layer

- Clustering algorithms
  - k-means clustering
  - clustering with Kohonen feature maps (SOMs), vector quantization (VQ)
  - estimate of the cluster width (variance problem)
- Gaussian mixture models: expectation-maximization (EM) alg.

Hybrid learning – RBF layer

- Clustering algorithms
  - k-means clustering
  - clustering with Kohonen feature maps (SOMs), vector quantization (VQ)
  - estimate of the cluster width (variance problem)
- Gaussian mixture models: expectation-maximization (EM) alg.
- Supervised tuning of the RBF parameters with gradient descent

$$\frac{\partial E}{\partial \mu_{ji}} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} \exp\left(-\frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \frac{(x_i^n - \mu_{ji})}{\sigma_j^2}$$

$$\frac{\partial E}{\partial \sigma_j} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} \exp\left(-\frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{\sigma_j^3}$$

Bishop, 1995

Hybrid learning – RBF layer

- Clustering algorithms
  - k-means clustering
  - clustering with Kohonen feature maps (SOMs), vector quantization (VQ)
  - estimate of the cluster width (variance problem)
- Gaussian mixture models: expectation-maximization (EM) alg.
- Supervised tuning of the RBF parameters with gradient descent

$$\frac{\partial E}{\partial \mu_{ji}} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} \exp\left(-\frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \frac{(x_i^n - \mu_{ji})}{\sigma_j^2}$$

$$\frac{\partial E}{\partial \sigma_j} = \sum_n \sum_k \{y_k(\mathbf{x}^n) - t_k^n\} w_{kj} \exp\left(-\frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \frac{\|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2}{\sigma_j^3}$$

Bishop, 1995

- Subset selection with orthogonal least squares

Hybrid learning – output layer

- Least-squares fitting – batch approach

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ . & . & . & . \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ . \\ d_n \end{bmatrix}$$

Hybrid learning – output layer

- Least-squares fitting – batch approach

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ . & . & . & . \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ . \\ d_n \end{bmatrix}$$

- Recursive least-squares estimation (see *Haykin*, RLS algorithm)

delta rule:

$$\begin{aligned} \mathbf{\Delta w} &= -\eta \nabla_{\mathbf{w}} \hat{\xi} \\ &= -\eta \frac{1}{2} \nabla_{\mathbf{w}} (f(x_k) - \mathbf{\Phi}(x_k)^\top \mathbf{w})^2 \\ &= \eta (f(x_k) - \mathbf{\Phi}(x_k)^\top \mathbf{w}) \mathbf{\Phi}(x_k) \\ &= \eta e \mathbf{\Phi}(x_k) \end{aligned}$$

$$\mathbf{\Phi}(x_k) = \begin{pmatrix} \phi_1(x_k) \\ \phi_2(x_k) \\ \vdots \\ \phi_n(x_k) \end{pmatrix}$$

RBF NN vs MLP

- Hidden units in MLP rely on weighted linear summations of inputs (a matter of interpretation)
  - RBFs rely on distance to prototype vectors

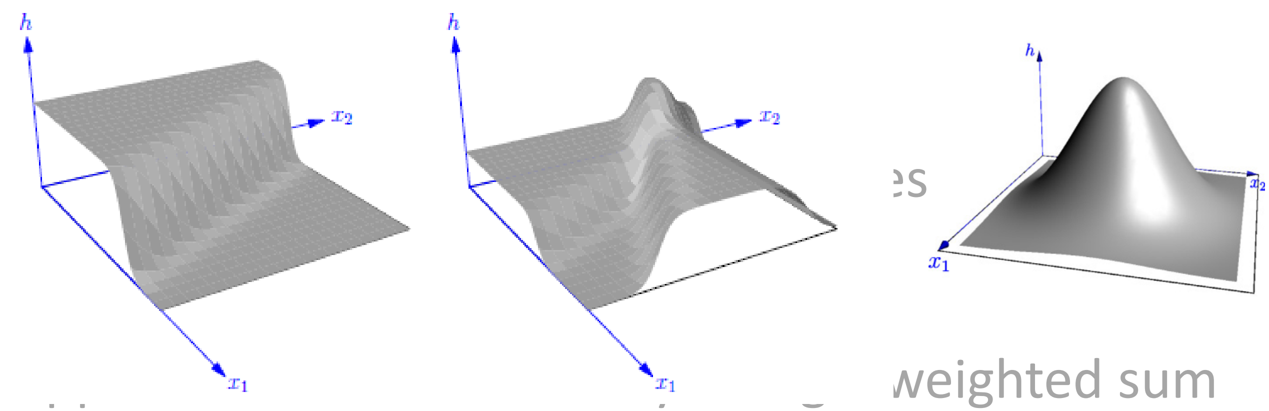
RBF NN vs MLP

- Hidden units in MLP rely on weighted linear summations of inputs (a matter of interpretation)
  - RBFs rely on distance to prototype vectors
- In MLPs, function approximation is defined as by a nested sum of weighted summations
  - In RBFs, the approximation is defined by a single weighted sum

RBF NN vs MLP

- Hidden units in MLP rely on weighted linear summations of inputs (a matter of i
  - RBFs rely
- In MLPs, fun ... es weighted su
  - In RBFs, t ... weighted sum

Weighted sum of Base Functions



- MLPs form distributed activations (many hidden units contribute to the output for a given input, which partly leads to local minima etc.)
  - In RBFs, very few local basis functions (wrt. input) are activated for a given input

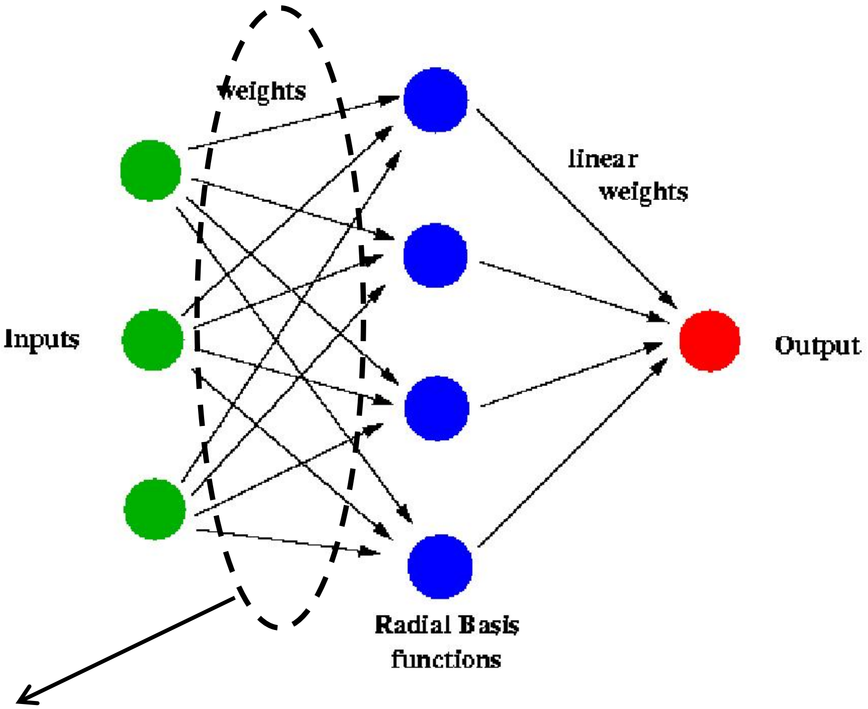
RBF NN vs MLP

- In MLPs, usually all parameters/weights are trained at the same time
    - In RBFs, hybrid two-stage training is used

RBF NN vs MLP

- In MLPs, usually all parameters/weights are trained at the same time
  - In RBFs, hybrid two-stage training is used
- MLPs rely on complex multi-layer architecture
  - RBF NNs have simple one-hidden-layer architecture

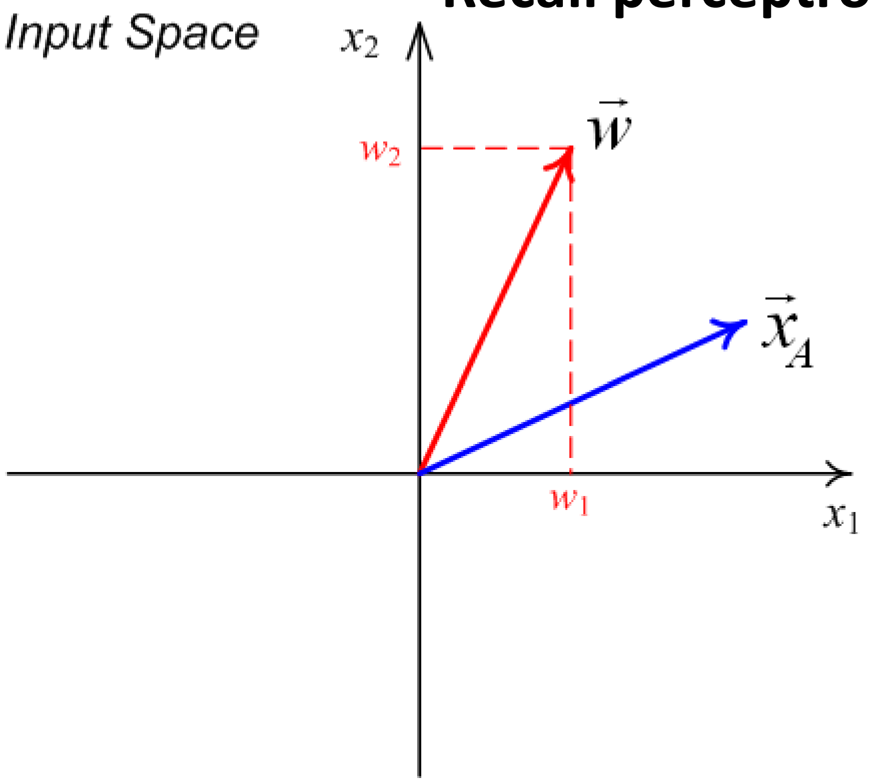
Interpretation of the weights to hidden layer

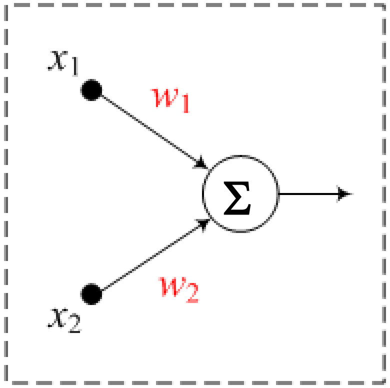


Do these connections have weights?

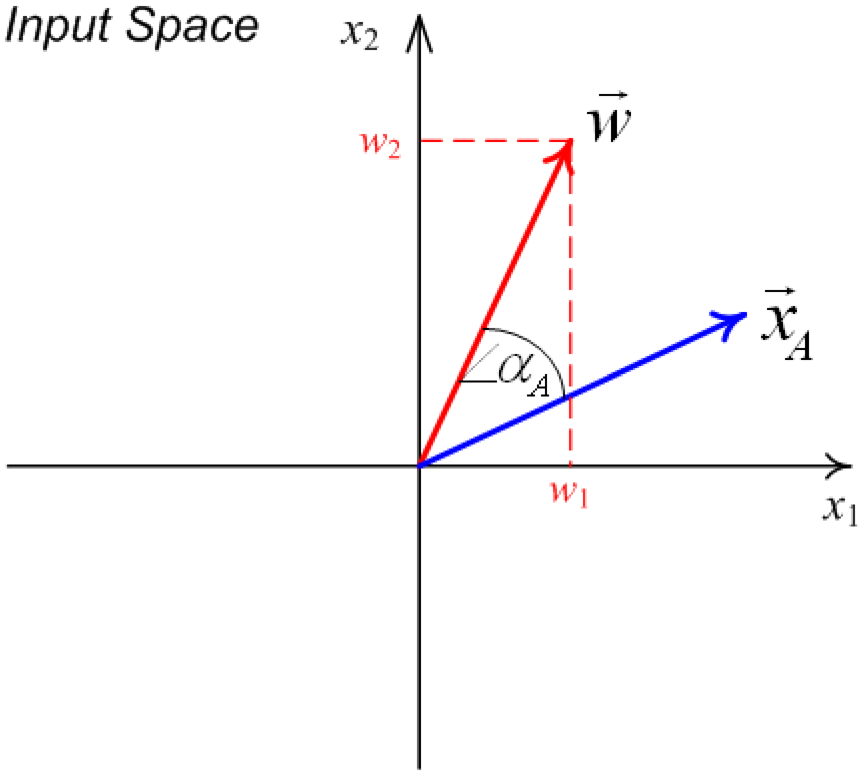
Weights in the input space

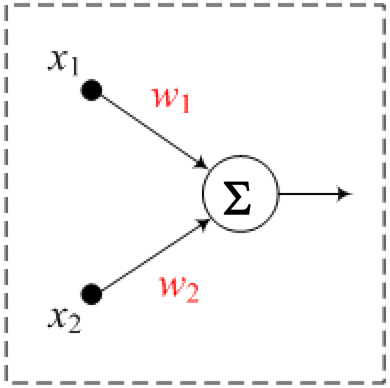
Recall perceptron





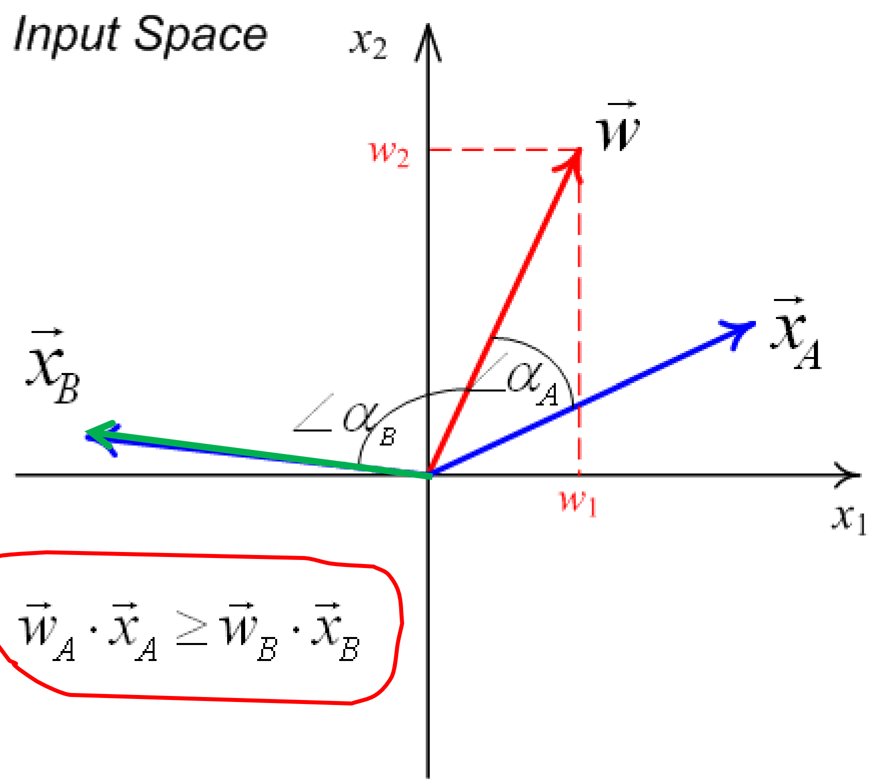
Weights in the input space – scalar product



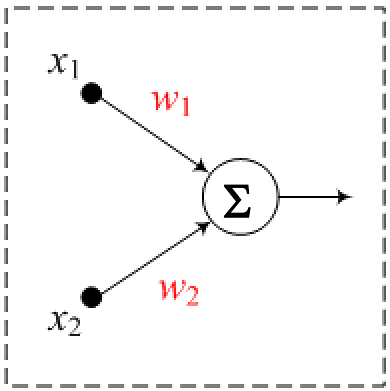


$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 = \|\vec{w}\| \|\vec{x}\| \cos(\angle \alpha)$$

Weights in the input space – scalar product

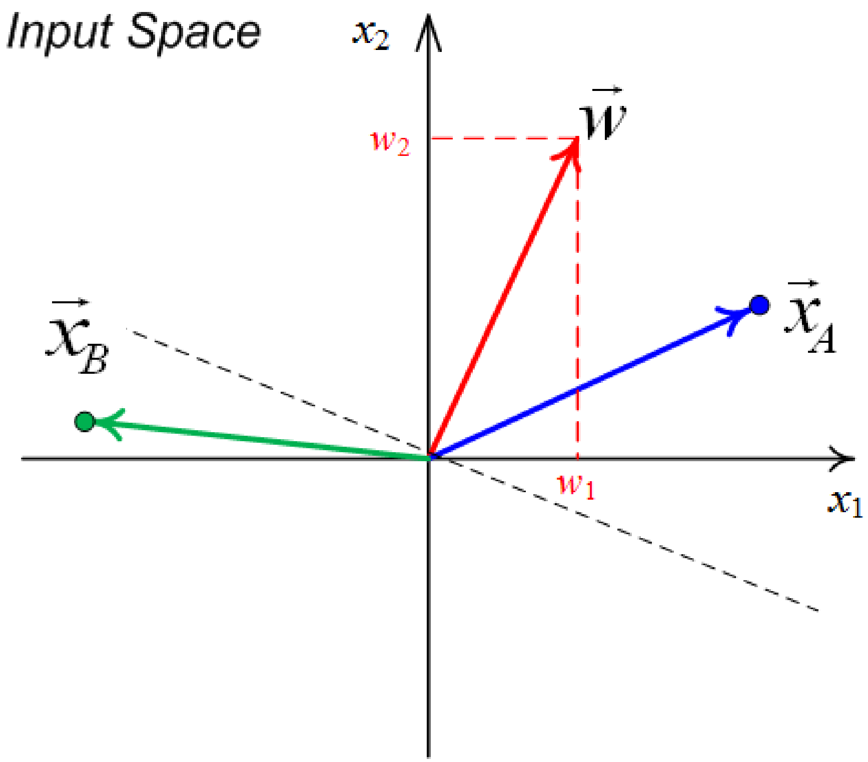


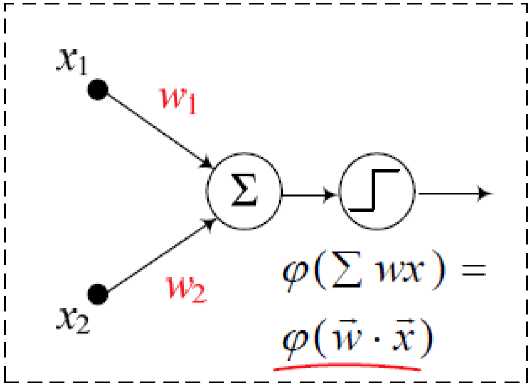
$$\vec{w}_A \cdot \vec{x}_A \geq \vec{w}_B \cdot \vec{x}_B$$



$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 = \|\vec{w}\| \|\vec{x}\| \cos(\angle \alpha)$$

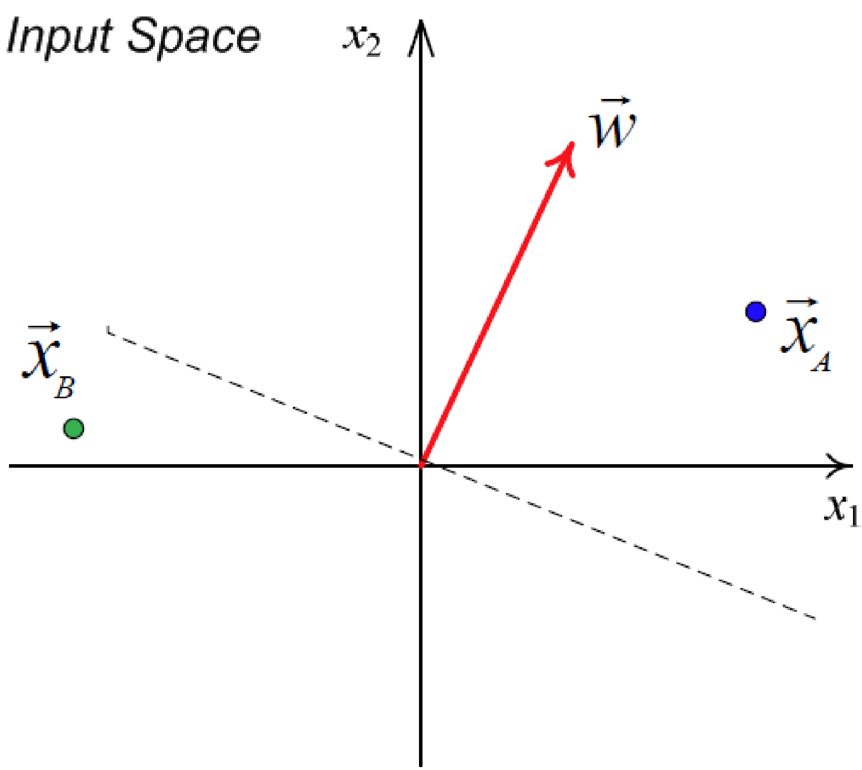
Weights in the input space – perceptron classifier

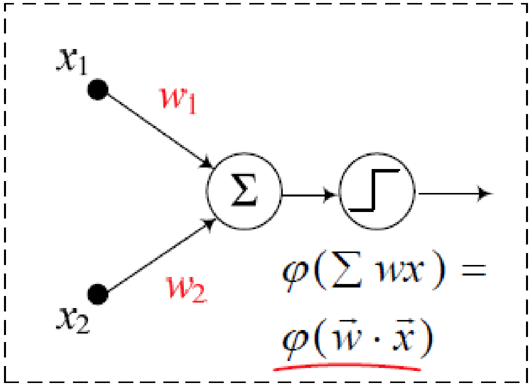




$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 = \|\vec{w}\| \|\vec{x}\| \cos(\angle \alpha)$$

Weights in the input space – perceptron classifier

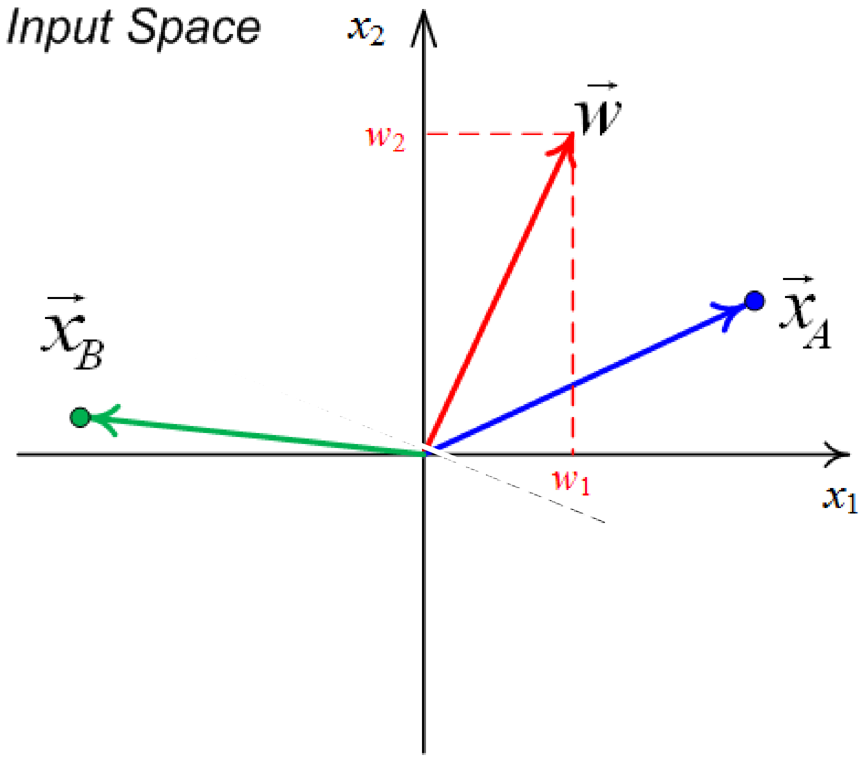


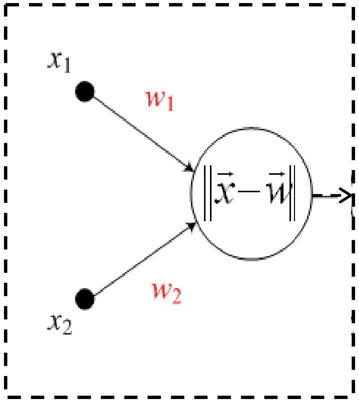


based on scalar product

$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{2} w_i x_i = w_1 x_1 + w_2 x_2 = \|\vec{w}\| \|\vec{x}\| \cos(\angle \alpha)$$

Weights in the input space – Euclidean distance

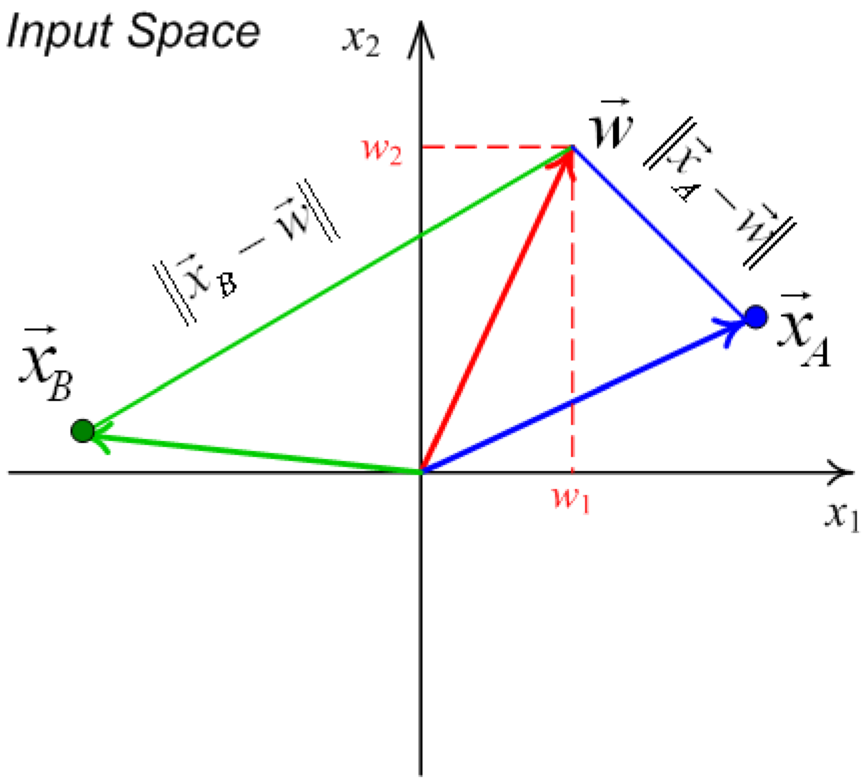


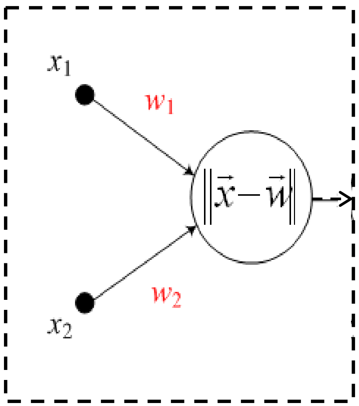


Euclidean distance measure: $\|\vec{x}-\vec{w}\|$

$$\|\vec{x}-\vec{w}\| = \sqrt{\sum_{i=1}^{2}(x_i - w_i)^2}$$

Weights in the input space – Euclidean distance

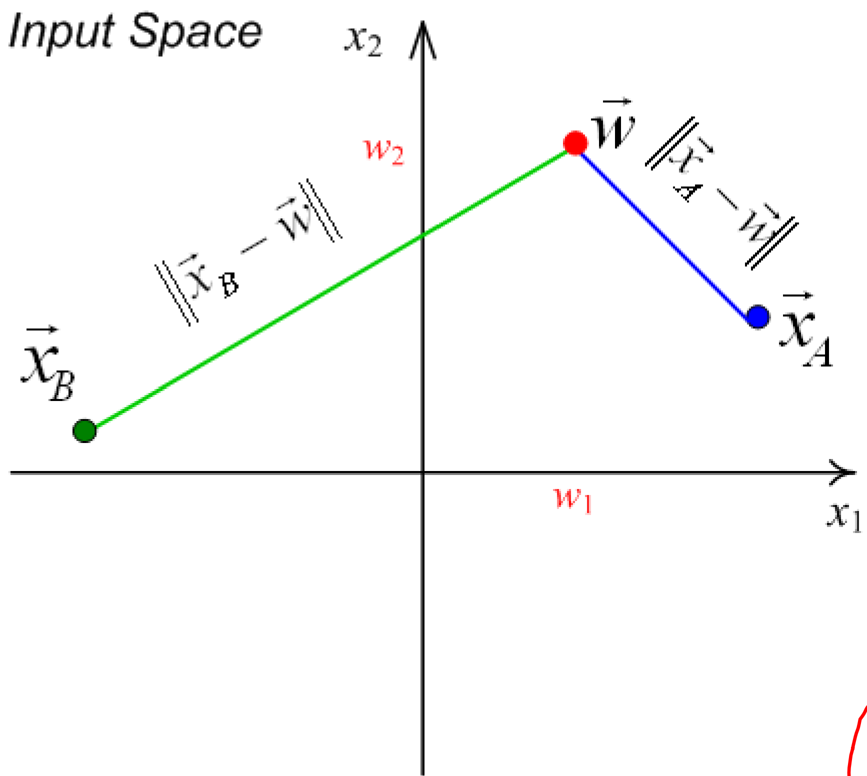




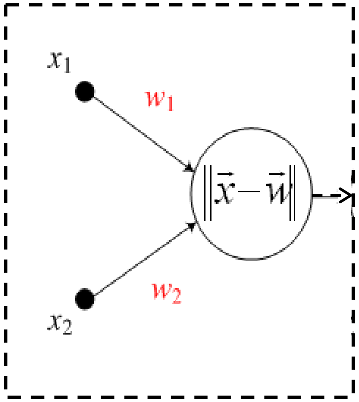
Euclidean distance measure: $\|\vec{x} - \vec{w}\|$

$$\|\vec{x} - \vec{w}\| = \sqrt{\sum_{i=1}^{2} (x_i - w_i)^2}$$

Weights in the input space – Euclidean distance



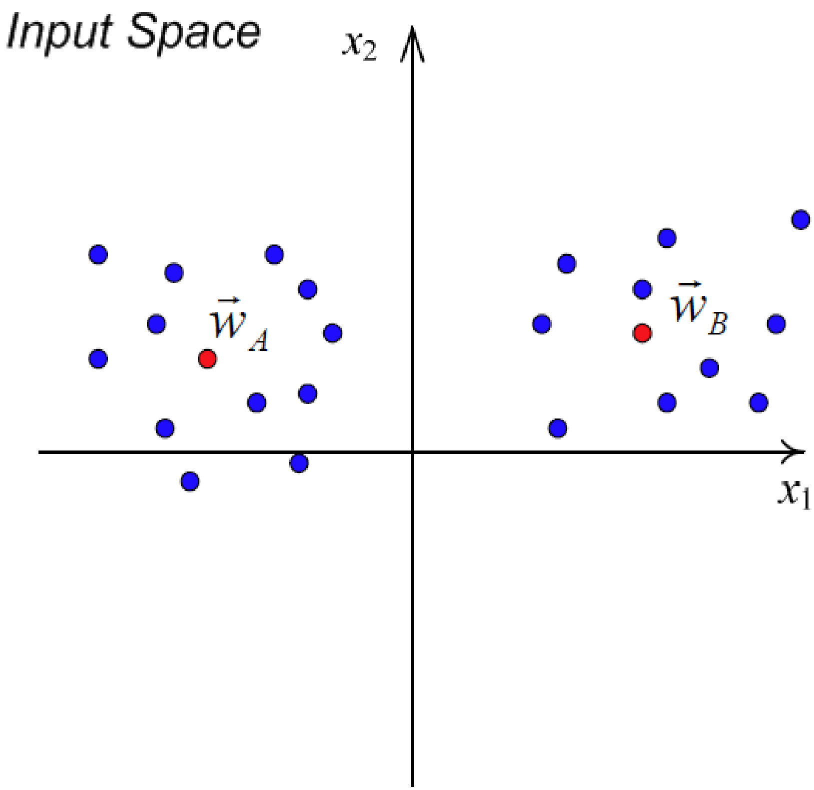
$$\|\vec{x} - \vec{w}\| = \sqrt{\sum_{i=1}^{2} (x_i - w_i)^2}$$

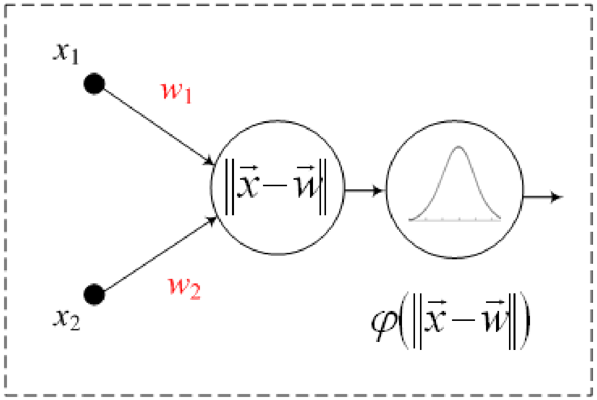


Euclidean distance measure: $\|\vec{x} - \vec{w}\|$

$$\|\vec{x}_A - \vec{w}\| < \|\vec{x}_B - \vec{w}\|$$

Weights in the input space – proximity measure

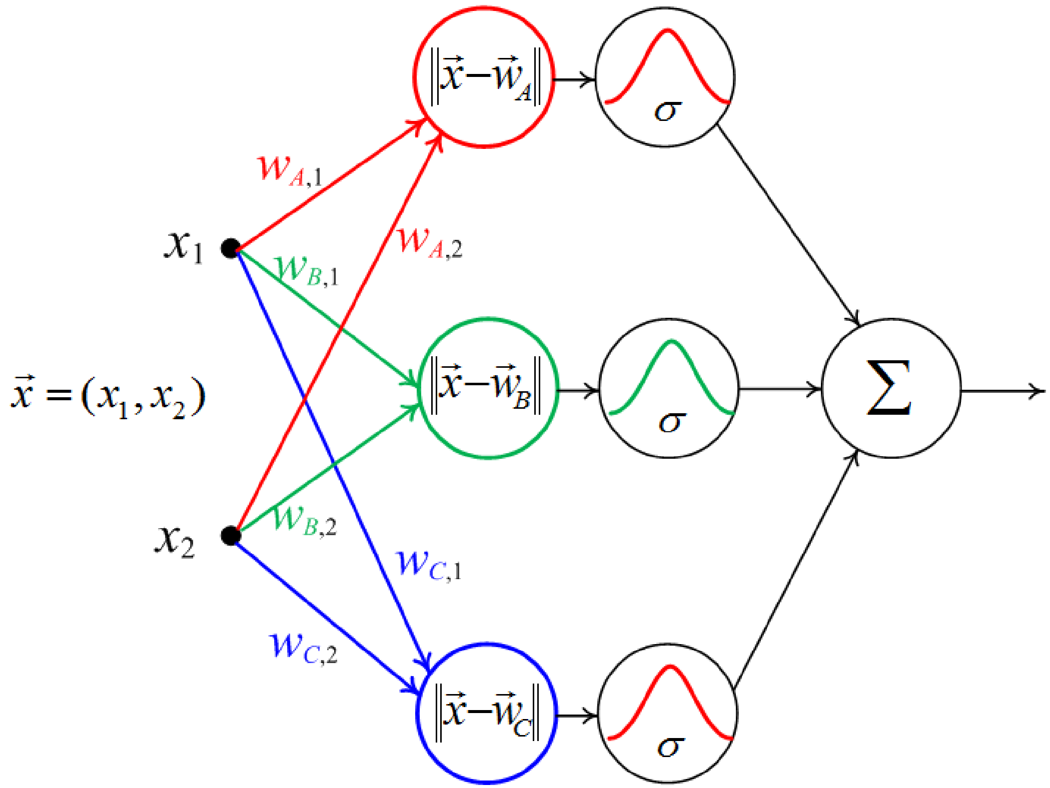




based on Euclidean distance

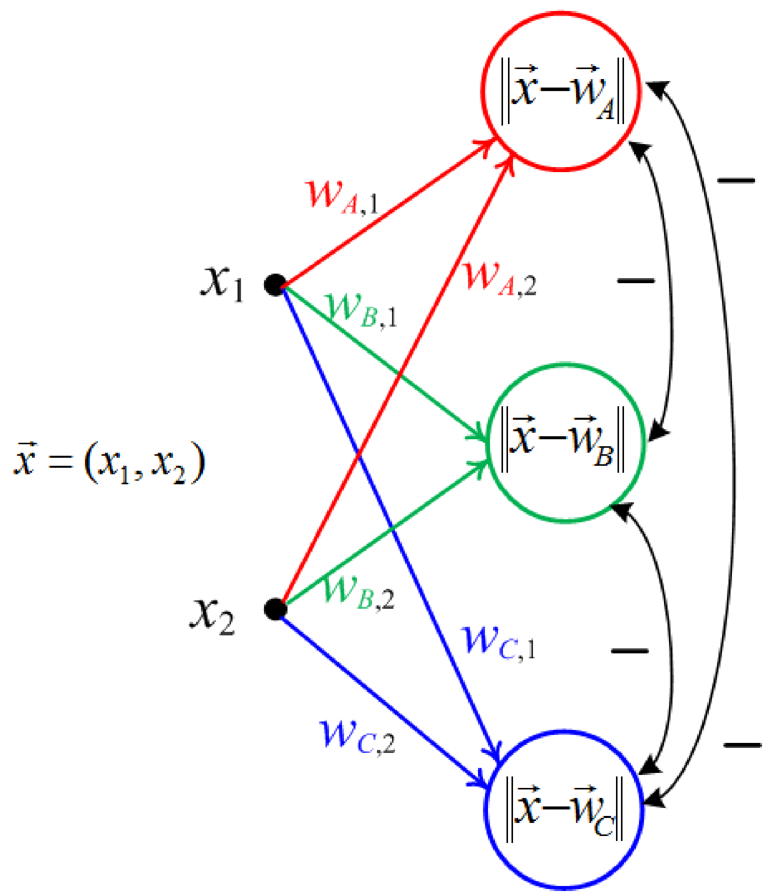
$$\varphi(\|\vec{x}-\vec{w}\|) = \exp\left(-\frac{\|\vec{x}-\vec{w}\|}{\sigma^2}\right)$$

RBF network – interpretation of the input weights



$$\varphi(\vec{x}, \vec{w}) = \exp\left(\frac{-\|\vec{x} - \vec{w}\|}{\sigma^2}\right)$$

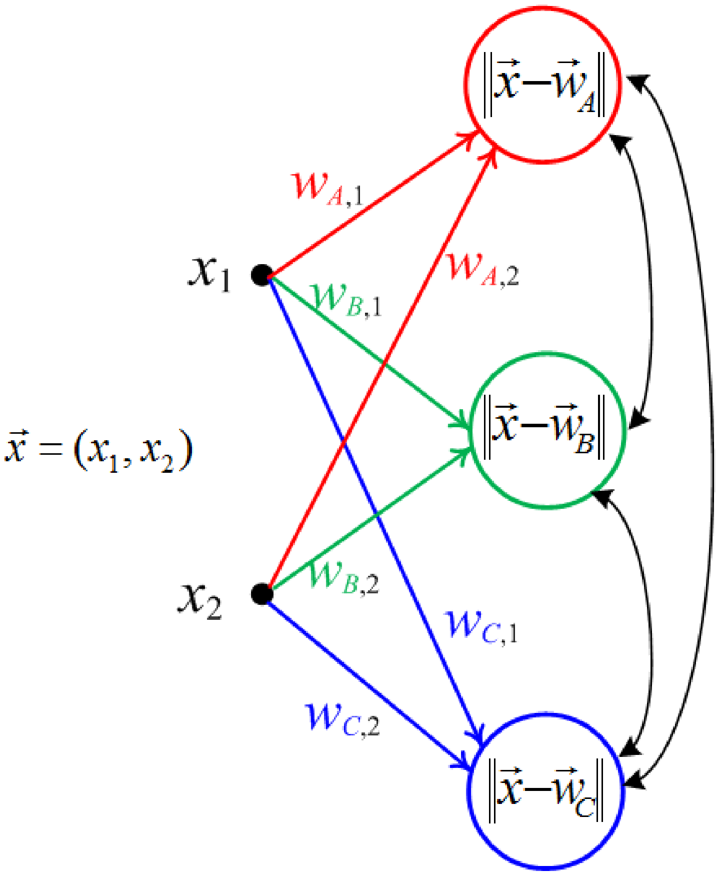
Competition and winner-take-all mechanism



Competition between nodes – **Winner Takes All** (WTA)

$$\vec{w}_A = (w_{A,1}, w_{A,2})$$
$$\vec{w}_B = (w_{B,1}, w_{B,2})$$
$$\vec{w}_C = (w_{C,1}, w_{C,2})$$

Competitive mechanisms



If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

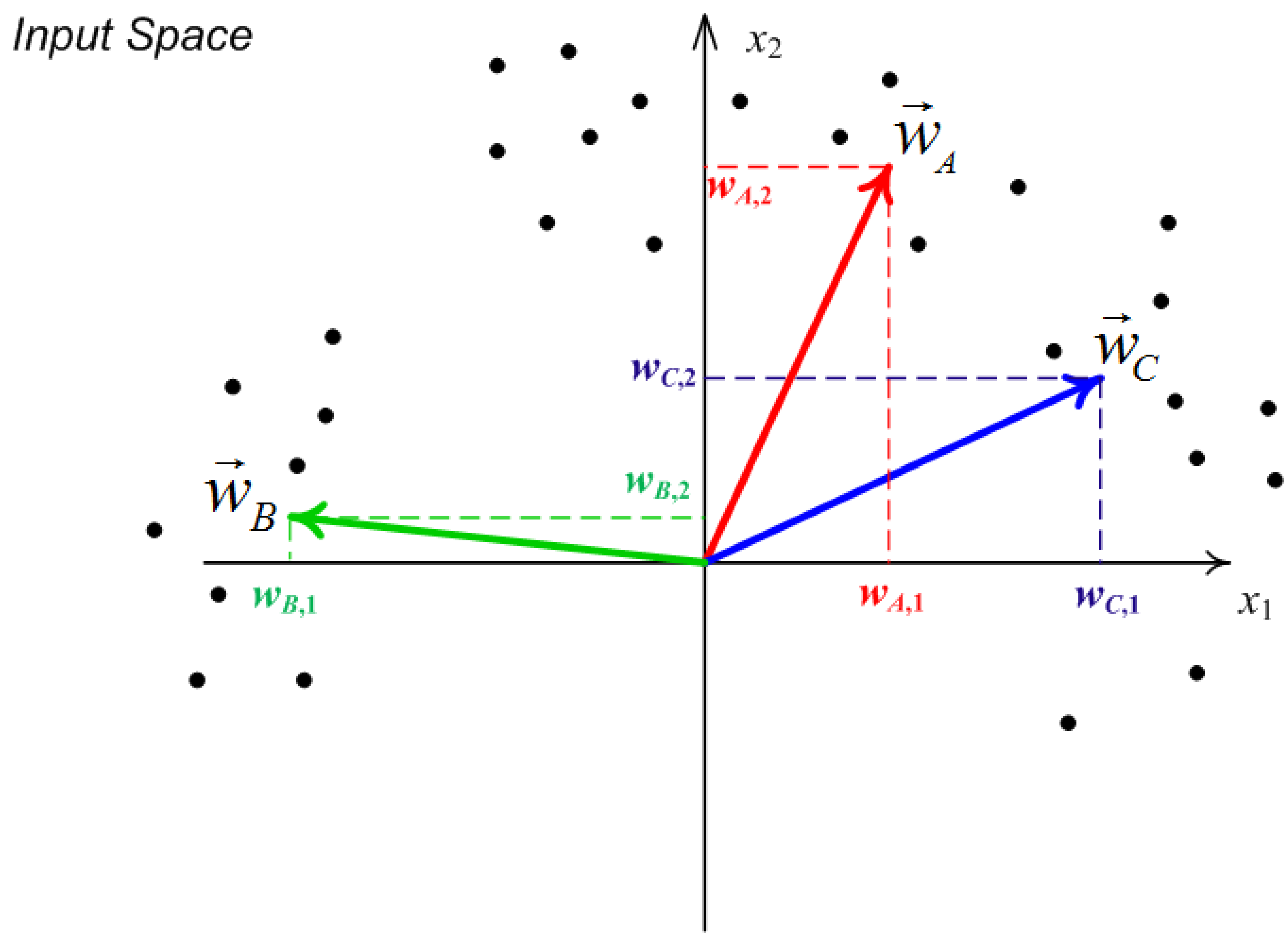
$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

$$\vec{w}_A = (w_{A,1}, w_{A,2})$$

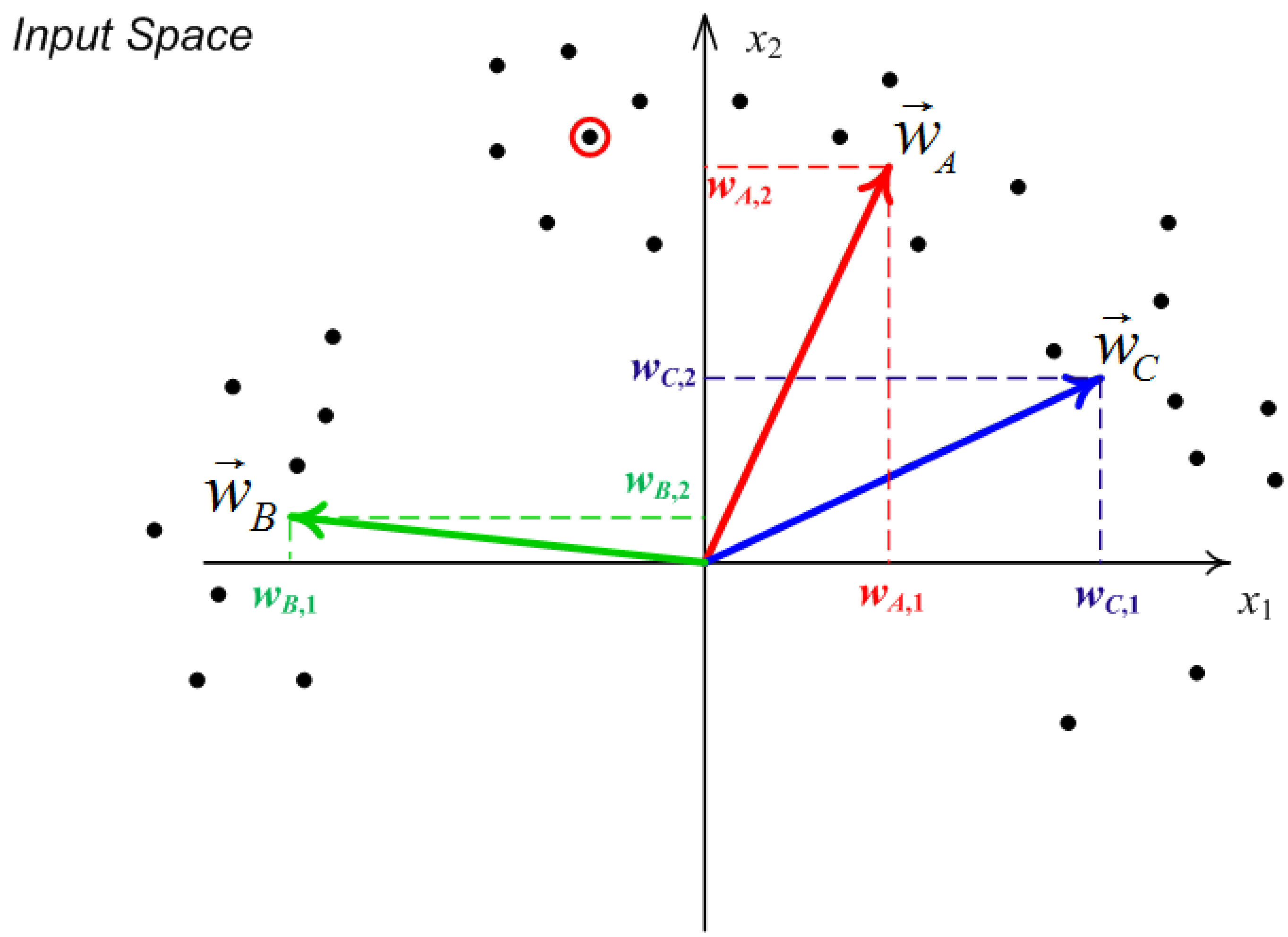
$$\vec{w}_B = (w_{B,1}, w_{B,2})$$

$$\vec{w}_C = (w_{C,1}, w_{C,2})$$

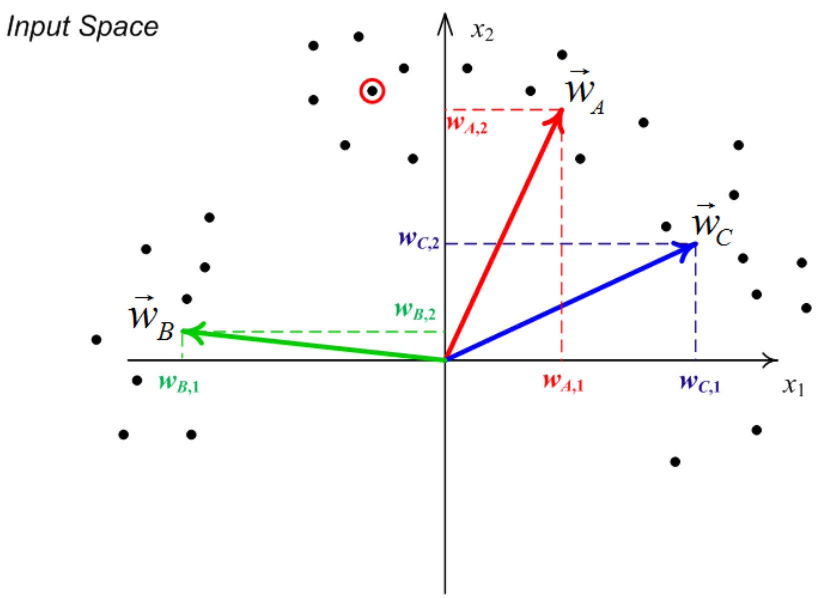
Competition – update of the input weights, example



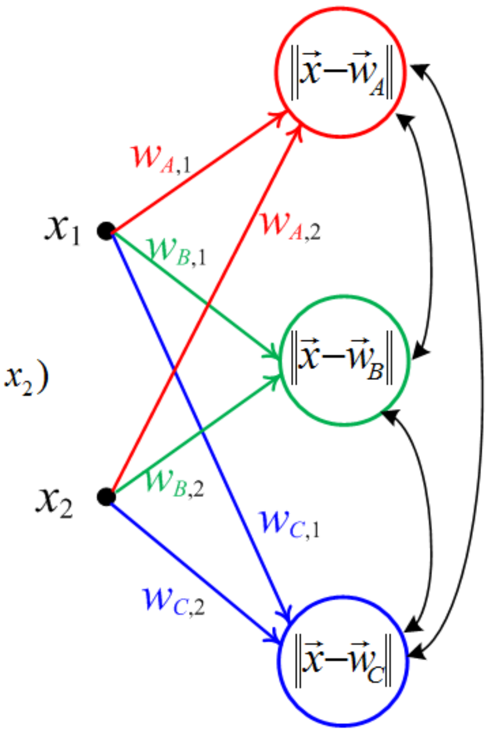
Competition – update of the input weights



Competition – update of the input weights



$\vec{x} = (x_1, x_2)$



$$\vec{w}_A = (w_{A,1}, w_{A,2})$$
$$\vec{w}_B = (w_{B,1}, w_{B,2})$$
$$\vec{w}_C = (w_{C,1}, w_{C,2})$$

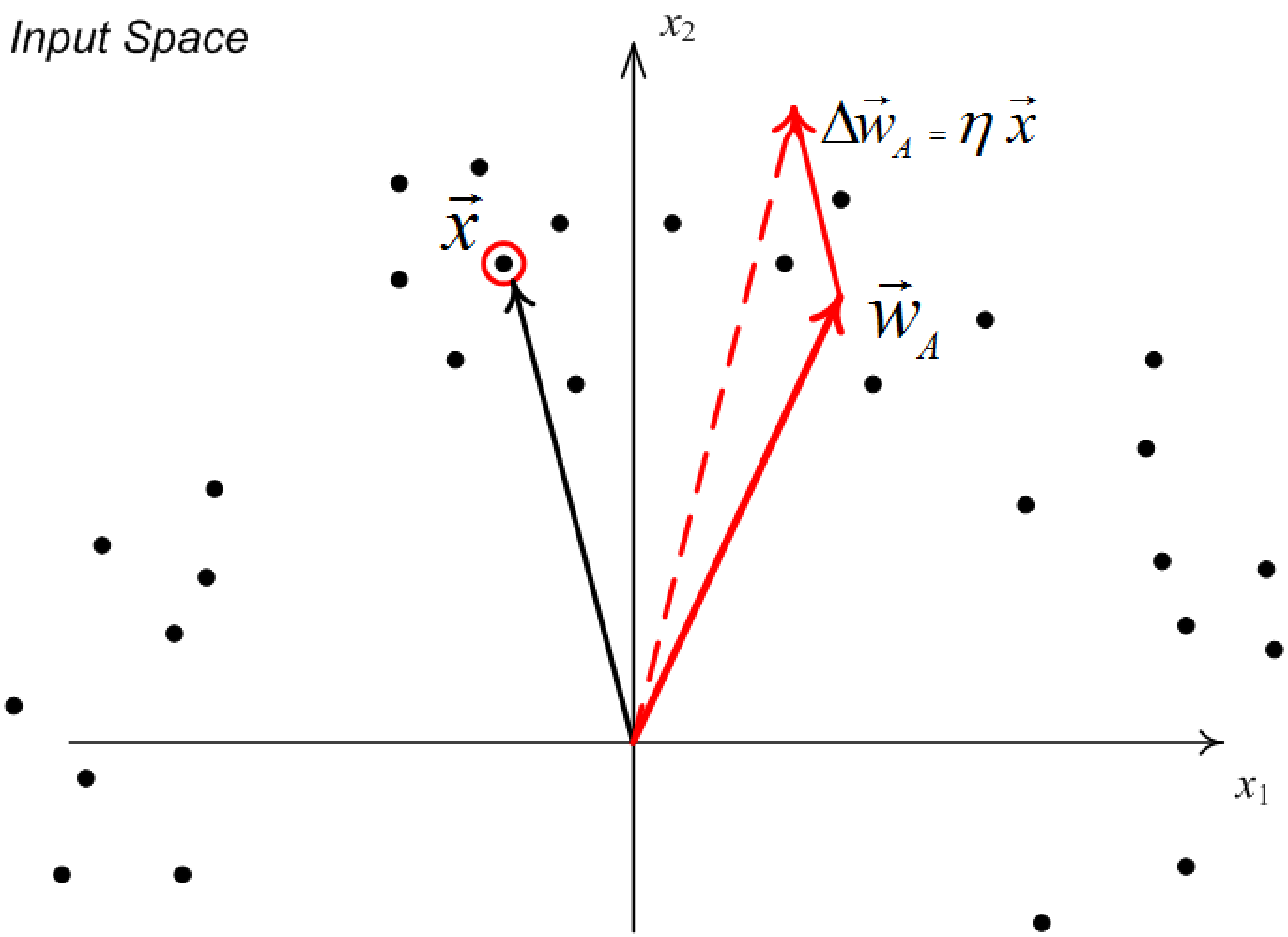
If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

Competition – update of the input weights, ver. 1



Competition – update of the input weights, ver. 2