Training Radial Basis Function Networks

Radial Basis Function Networks: Initialization

Let $L_{\text{fixed}} = \{l_1, \ldots, l_m\}$ be a fixed learning task,
consisting of m training patterns $l = (\vec{\imath}^{\,(l)}, \vec{o}^{\,(l)})$.

Simple radial basis function network:
One hidden neuron v_k, $k = 1, \ldots, m$, for each training pattern:

$$\forall k \in \{1, \ldots, m\}: \qquad \vec{w}_{v_k} = \vec{\imath}^{\,(l_k)}.$$

If the activation function is the Gaussian function,
the radii σ_k are chosen heuristically

$$\forall k \in \{1, \ldots, m\}: \qquad \sigma_k = \frac{d_{\max}}{\sqrt{2m}},$$

where

$$d_{\max} = \max_{l_j, l_k \in L_{\text{fixed}}} d\left(\vec{\imath}^{\,(l_j)}, \vec{\imath}^{\,(l_k)}\right).$$

Radial Basis Function Networks: Initialization

Initializing the connections from the hidden to the output neurons

$$\forall u : \sum_{k=1}^{m} w_{uv_m} \mathrm{out}_{v_m}^{(l)} - \theta_u = o_u^{(l)} \qquad \text{or abbreviated} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u,$$

where $\vec{o}_u = (o_u^{(l_1)}, \ldots, o_u^{(l_m)})^\top$ is the vector of desired outputs, $\theta_u = 0$, and

$$\mathbf{A} = \begin{pmatrix} \mathrm{out}_{v_1}^{(l_1)} & \mathrm{out}_{v_2}^{(l_1)} & \ldots & \mathrm{out}_{v_m}^{(l_1)} \\ \mathrm{out}_{v_1}^{(l_2)} & \mathrm{out}_{v_2}^{(l_2)} & \ldots & \mathrm{out}_{v_m}^{(l_2)} \\ \vdots & \vdots & & \vdots \\ \mathrm{out}_{v_1}^{(l_m)} & \mathrm{out}_{v_2}^{(l_m)} & \ldots & \mathrm{out}_{v_m}^{(l_m)} \end{pmatrix}.$$

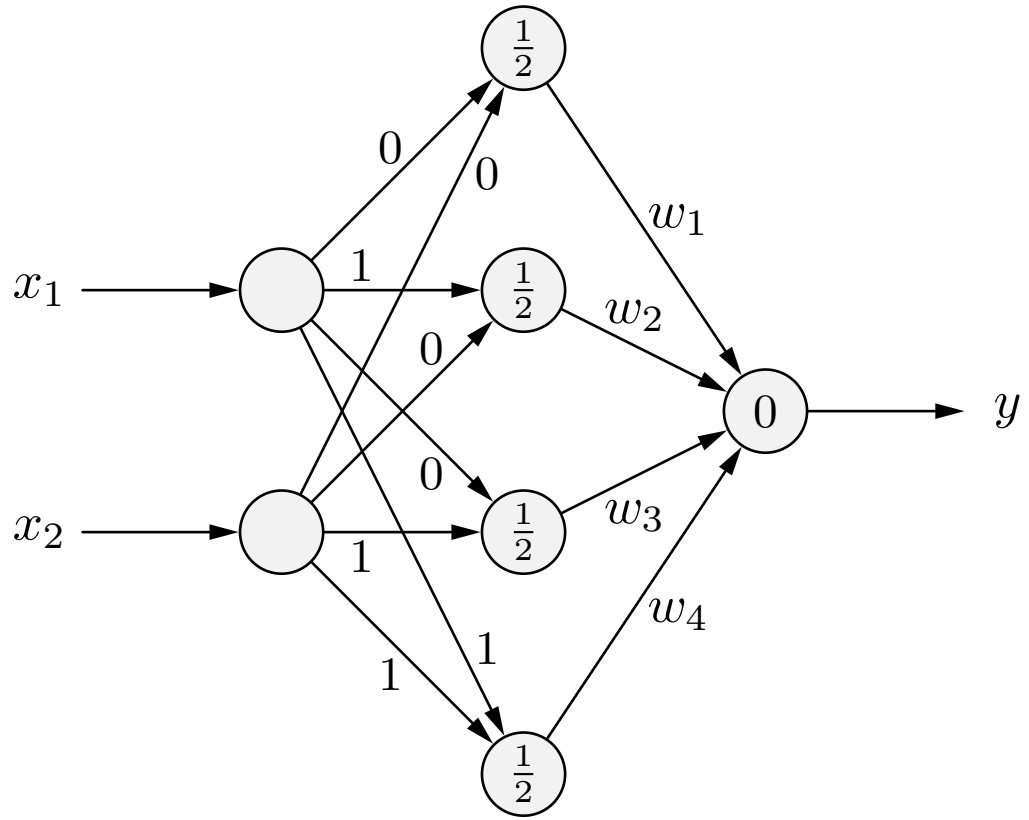
This is a linear equation system, that can be solved by inverting the matrix $\mathbf{A}$:

$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u.$$

RBFN Initialization: Example

Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



RBFN Initialization: Example

Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & e^{-2} & e^{-2} & e^{-4} \\ e^{-2} & 1 & e^{-4} & e^{-2} \\ e^{-2} & e^{-4} & 1 & e^{-2} \\ e^{-4} & e^{-2} & e^{-2} & 1 \end{pmatrix} \qquad \mathbf{A}^{-1} = \begin{pmatrix} \frac{a}{D} & \frac{b}{D} & \frac{b}{D} & \frac{c}{D} \\ \frac{b}{D} & \frac{a}{D} & \frac{c}{D} & \frac{b}{D} \\ \frac{b}{D} & \frac{c}{D} & \frac{a}{D} & \frac{b}{D} \\ \frac{c}{D} & \frac{b}{D} & \frac{b}{D} & \frac{a}{D} \end{pmatrix}$$

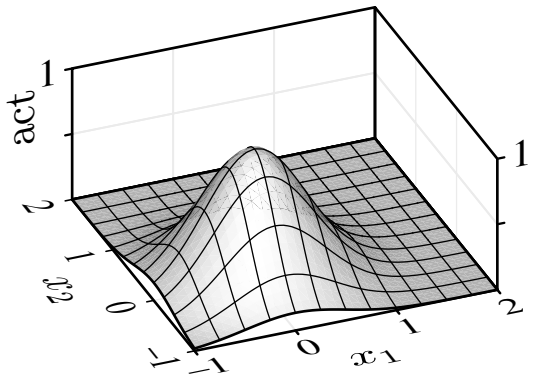
where

$$\begin{aligned} D &= 1 - 4e^{-4} + 6e^{-8} - 4e^{-12} + e^{-16} &&\approx \quad 0.9287 \\ a &= \quad 1 \quad - 2e^{-4} + e^{-8} &&\approx \quad 0.9637 \\ b &= -e^{-2} + 2e^{-6} - e^{-10} &&\approx -0.1304 \\ c &= \quad e^{-4} - 2e^{-8} + e^{-12} &&\approx \quad 0.0177 \end{aligned}$$

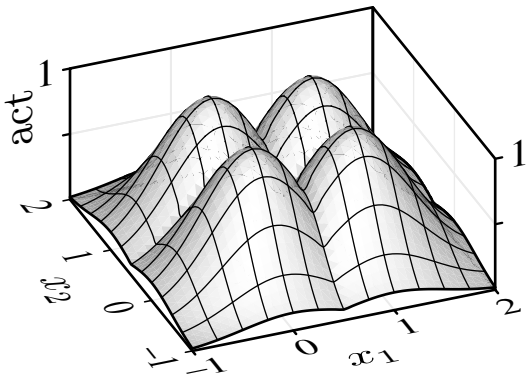
$$\vec{w}_u = \mathbf{A}^{-1} \cdot \vec{o}_u = \frac{1}{D} \begin{pmatrix} a+c \\ 2b \\ 2b \\ a+c \end{pmatrix} \approx \begin{pmatrix} 1.0567 \\ -0.2809 \\ -0.2809 \\ 1.0567 \end{pmatrix}$$

RBFN Initialization: Example

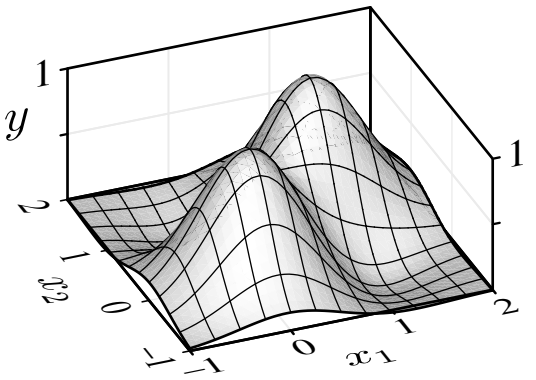
Simple radial basis function network for the biimplication $x_1 \leftrightarrow x_2$



single basis function



all basis functions



output

- Initialization leads already to a perfect solution of the learning task.
- Subsequent training is not necessary.

Radial Basis Function Networks: Initialization

Normal radial basis function networks:
Select subset of k training patterns as centers.

$$\mathbf{A} = \begin{pmatrix} 1 & \text{out}_{v_1}^{(l_1)} & \text{out}_{v_2}^{(l_1)} & \ldots & \text{out}_{v_k}^{(l_1)} \\ 1 & \text{out}_{v_1}^{(l_2)} & \text{out}_{v_2}^{(l_2)} & \ldots & \text{out}_{v_k}^{(l_2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \text{out}_{v_1}^{(l_m)} & \text{out}_{v_2}^{(l_m)} & \ldots & \text{out}_{v_k}^{(l_m)} \end{pmatrix} \qquad \mathbf{A} \cdot \vec{w}_u = \vec{o}_u$$

Compute (Moore–Penrose) pseudo inverse:

$$\mathbf{A}^+ = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top.$$

The weights can then be computed by

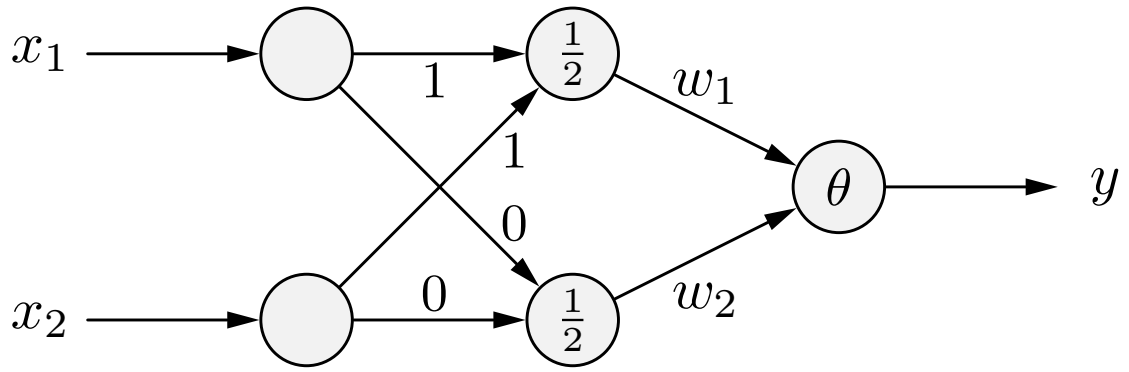
$$\vec{w}_u = \mathbf{A}^+ \cdot \vec{o}_u = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \cdot \vec{o}_u$$

RBFN Initialization: Example

Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

Select two training patterns:

- $l_1 = (\vec{\imath}^{(l_1)}, \vec{o}^{(l_1)}) = ((0,0),(1))$
- $l_4 = (\vec{\imath}^{(l_4)}, \vec{o}^{(l_4)}) = ((1,1),(1))$



RBFN Initialization: Example

Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & e^{-4} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-2} & e^{-2} \\ 1 & e^{-4} & 1 \end{pmatrix} \qquad \mathbf{A}^+ = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top = \begin{pmatrix} a & b & b & a \\ c & d & d & e \\ e & d & d & c \end{pmatrix}$$

where

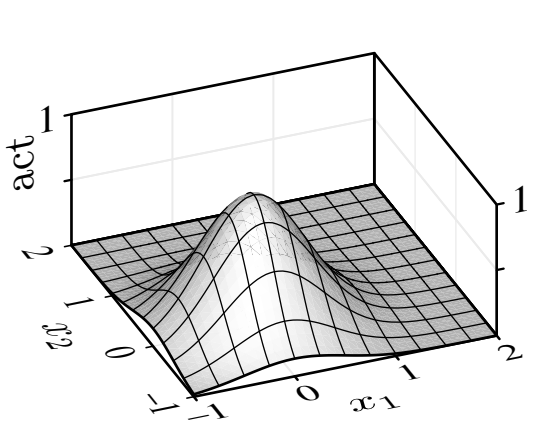
$$\begin{aligned} a &\approx -0.1810, & b &\approx \quad 0.6810, \\ c &\approx \quad 1.1781, & d &\approx -0.6688, & e &\approx 0.1594. \end{aligned}$$

Resulting weights:

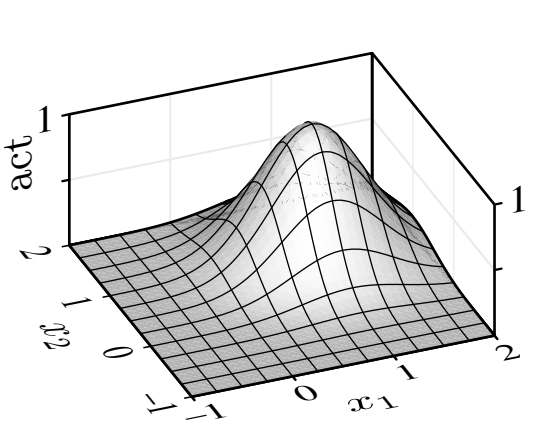
$$\vec{w}_u = \begin{pmatrix} -\theta \\ w_1 \\ w_2 \end{pmatrix} = \mathbf{A}^+ \cdot \vec{o}_u \approx \begin{pmatrix} -0.3620 \\ 1.3375 \\ 1.3375 \end{pmatrix}.$$

RBFN Initialization: Example

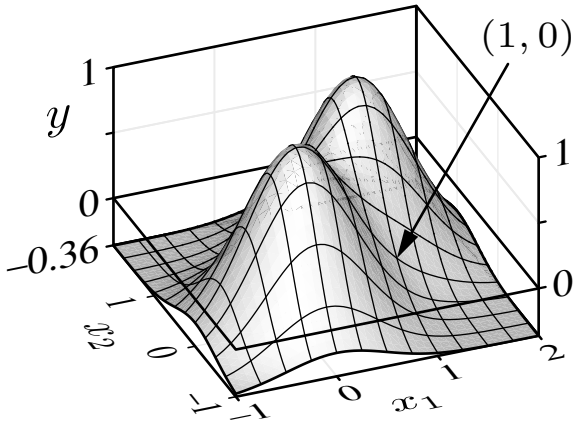
Normal radial basis function network for the biimplication $x_1 \leftrightarrow x_2$



basis function (0,0)



basis function (1,1)



output

- Initialization leads already to a perfect solution of the learning task.
- This is an accident, because the linear equation system is not over-determined, due to linearly dependent equations.

Radial Basis Function Networks: Initialization

How to choose the radial basis function centers?

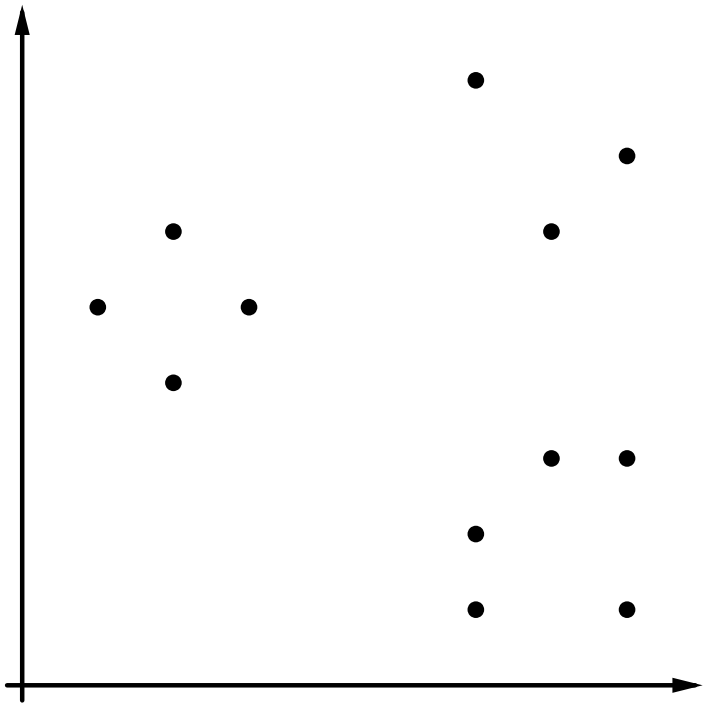
- Use **all data points** as centers for the radial basis functions.
  - Advantages: Only radius and output weights need to be determined; desired output values can be achieved exactly (unless there are inconsistencies).
  - Disadvantage: Often much too many radial basis functions; computing the weights to the output neuron via a pseudo-inverse can become infeasible.
- Use a **random subset** of data points as centers for the radial basis functions.
  - Advantages: Fast; only radius and output weights need to be determined.
  - Disadvantages: Performance depends heavily on the choice of data points.
- Use the **result of clustering** as centers for the radial basis functions, e.g.
  - c-means clustering (on the next slides)
  - Learning vector quantization (to be discussed later)

RBFN Initialization: c-means Clustering

- Choose a number c of clusters to be found (user input).
- Initialize the cluster centers randomly
  (for instance, by randomly selecting c data points).
- **Data point assignment**:
  Assign each data point to the cluster center that is closest to it
  (that is, closer than any other cluster center).
- **Cluster center update**:
  Compute new cluster centers as the mean vectors of the assigned data points.
  (Intuitively: center of gravity if each data point has unit weight.)
- Repeat these two steps (data point assignment and cluster center update)
  until the clusters centers do not change anymore.

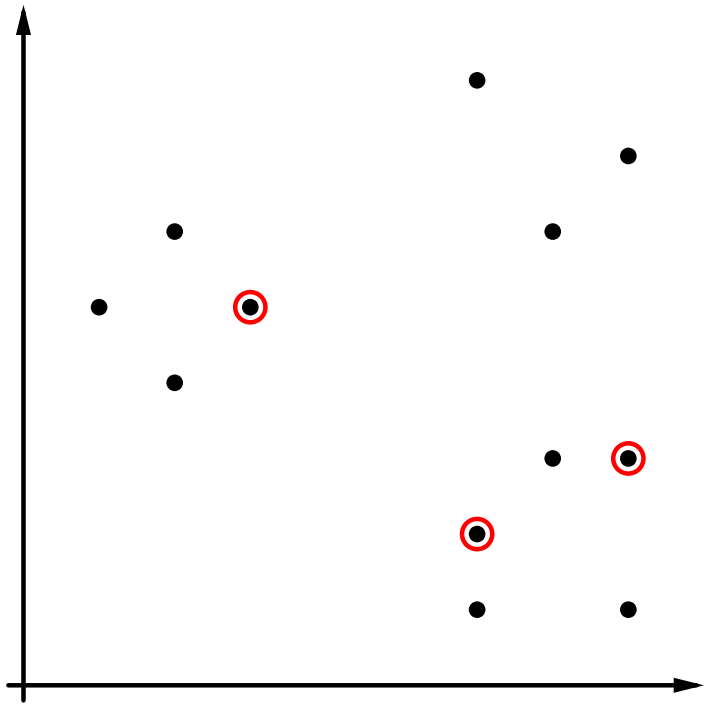
It can be shown that this scheme must converge,
that is, the update of the cluster centers cannot go on forever.

c-Means Clustering: Example



Data set to cluster.

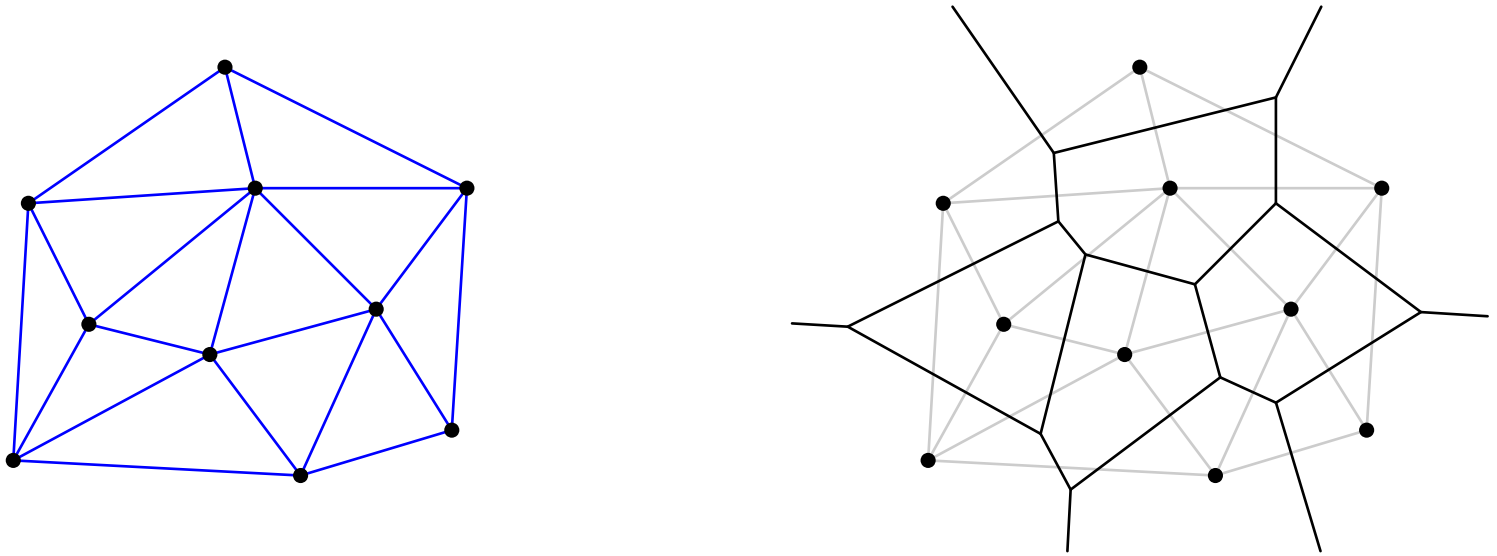
Choose $c = 3$ clusters.
(From visual inspection, can be difficult to determine in general.)



Initial position of cluster centers.

Randomly selected data points.
(Alternative methods include e.g. latin hypercube sampling)

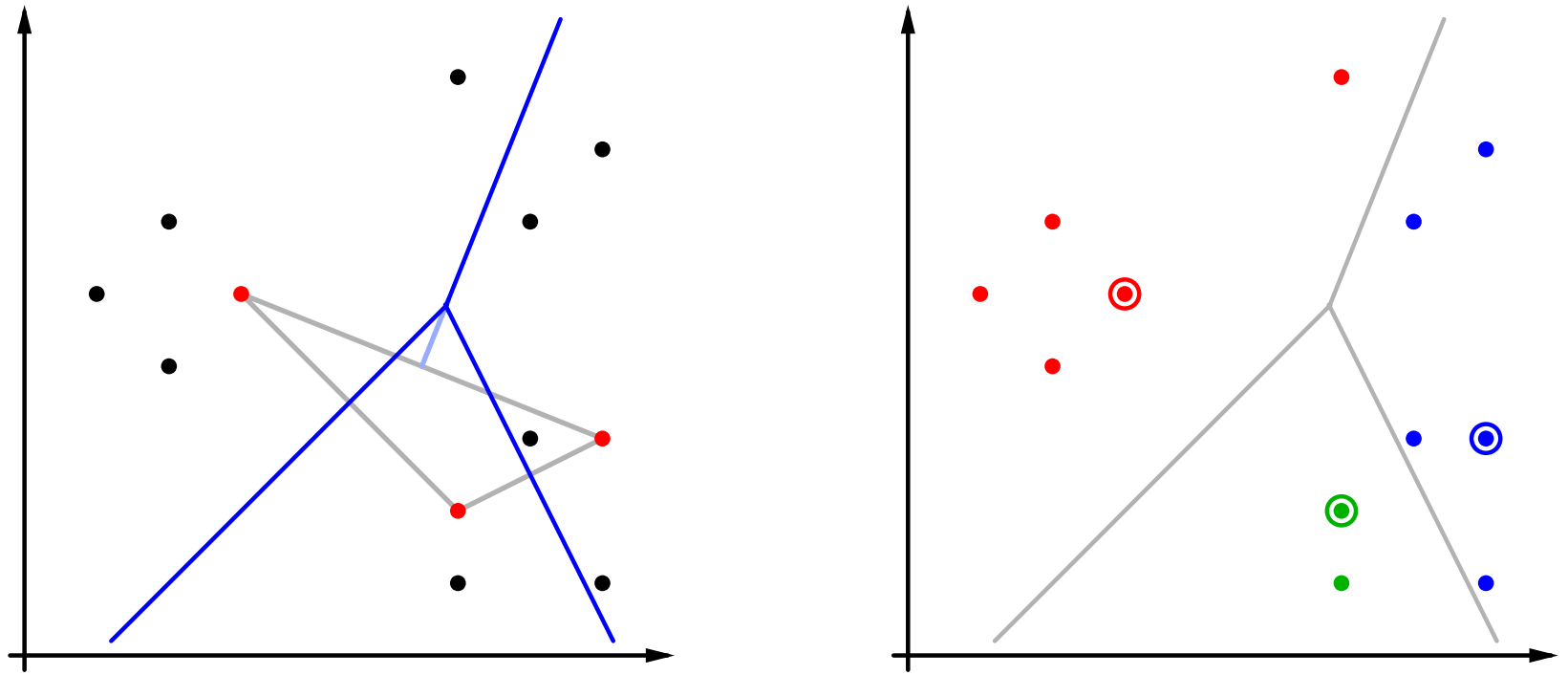
Delaunay Triangulations and Voronoi Diagrams



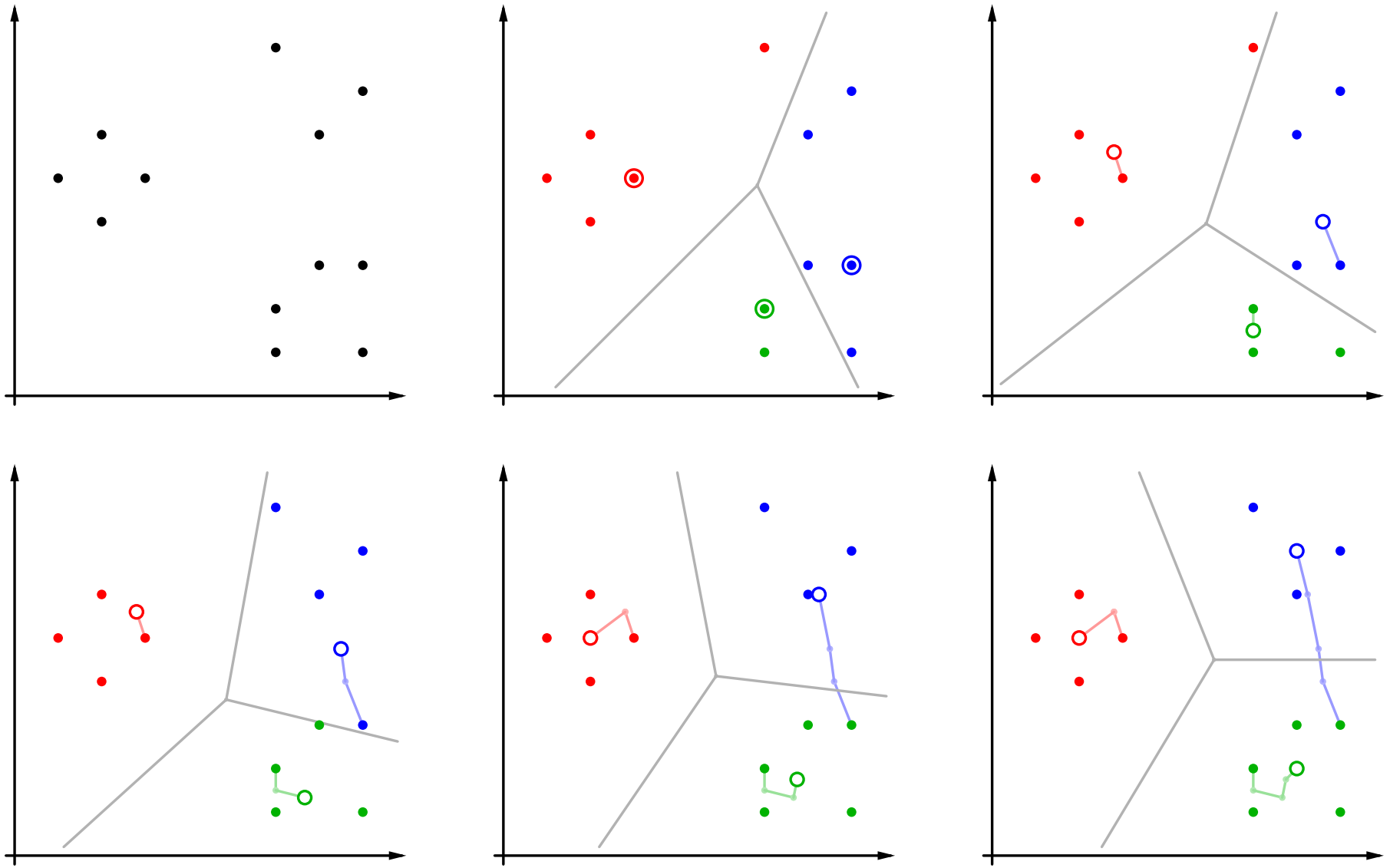
- Dots represent cluster centers.
- Left: **Delaunay Triangulation**
  The circle through the corners of a triangle does not contain another point.
- Right: **Voronoi Diagram / Tesselation**
  Midperpendiculars of the Delaunay triangulation: boundaries of the regions of points that are closest to the enclosed cluster center (Voronoi cells).

Delaunay Triangulations and Voronoi Diagrams

- **Delaunay Triangulation:** simple triangle (shown in gray on the left)
- **Voronoi Diagram:** midperpendiculars of the triangle's edges (shown in blue on the left, in gray on the right)



c-Means Clustering: Example



Radial Basis Function Networks: Training

Training radial basis function networks:
Derivation of update rules is analogous to that of multi-layer perceptrons.

Weights from the hidden to the output neurons.

Gradient:

$$\vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \frac{\partial e_u^{(l)}}{\partial \vec{w}_u} = -2(o_u^{(l)} - \mathrm{out}_u^{(l)})\,\vec{\mathrm{in}}_u^{(l)},$$

Weight update rule:

$$\Delta \vec{w}_u^{(l)} = -\frac{\eta_3}{2} \vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \eta_3 (o_u^{(l)} - \mathrm{out}_u^{(l)})\,\vec{\mathrm{in}}_u^{(l)}$$

Typical learning rate: $\eta_3 \approx 0.001$.

(Two more learning rates are needed for the center coordinates and the radii.)

Radial Basis Function Networks: Training

Training radial basis function networks:
Center coordinates (weights from the input to the hidden neurons).

Gradient:

$$\vec{\nabla}_{\vec{w}_v} e^{(l)} = \frac{\partial e^{(l)}}{\partial \vec{w}_v} = -2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{su} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \mathrm{net}_v^{(l)}} \frac{\partial \mathrm{net}_v^{(l)}}{\partial \vec{w}_v}$$

Weight update rule:

$$\Delta \vec{w}_v^{(l)} = -\frac{\eta_1}{2} \vec{\nabla}_{\vec{w}_v} e^{(l)} = \eta_1 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{sv} \frac{\partial \mathrm{out}_v^{(l)}}{\partial \mathrm{net}_v^{(l)}} \frac{\partial \mathrm{net}_v^{(l)}}{\partial \vec{w}_v}$$

Typical learning rate: $\eta_1 \approx 0.02$.

Radial Basis Function Networks: Training

Training radial basis function networks:
Center coordinates (weights from the input to the hidden neurons).

Special case: **Euclidean distance**

$$\frac{\partial \operatorname{net}_v^{(l)}}{\partial \vec{w}_v} = \left(\sum_{i=1}^{n} (w_{vp_i} - \operatorname{out}_{p_i}^{(l)})^2 \right)^{-\frac{1}{2}} (\vec{w}_v - \vec{\operatorname{in}}_v^{(l)}).$$

Special case: **Gaussian activation function**

$$\frac{\partial \operatorname{out}_v^{(l)}}{\partial \operatorname{net}_v^{(l)}} = \frac{\partial f_{\text{act}}(\operatorname{net}_v^{(l)}, \sigma_v)}{\partial \operatorname{net}_v^{(l)}} = \frac{\partial}{\partial \operatorname{net}_v^{(l)}} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = -\frac{\operatorname{net}_v^{(l)}}{\sigma_v^2} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

Radial Basis Function Networks: Training

Training radial basis function networks:
Radii of radial basis functions.

Gradient:

$$\frac{\partial e^{(l)}}{\partial \sigma_v} = -2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{su} \frac{\partial \, \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Weight update rule:

$$\Delta \sigma_v^{(l)} = -\frac{\eta_2}{2} \frac{\partial e^{(l)}}{\partial \sigma_v} = \eta_2 \sum_{s \in \mathrm{succ}(v)} (o_s^{(l)} - \mathrm{out}_s^{(l)}) w_{sv} \frac{\partial \, \mathrm{out}_v^{(l)}}{\partial \sigma_v}.$$

Typical learning rate: $\eta_2 \approx 0.01$.

Radial Basis Function Networks: Training

Training radial basis function networks:
Radii of radial basis functions.

Special case: **Gaussian activation function**

$$\frac{\partial \operatorname{out}_v^{(l)}}{\partial \sigma_v} = \frac{\partial}{\partial \sigma_v} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}} = \frac{\left(\operatorname{net}_v^{(l)}\right)^2}{\sigma_v^3} e^{-\frac{\left(\operatorname{net}_v^{(l)}\right)^2}{2\sigma_v^2}}.$$

(The distance function is irrelevant for the radius update,
since it only enters the network input function.)

Radial Basis Function Networks: Generalization

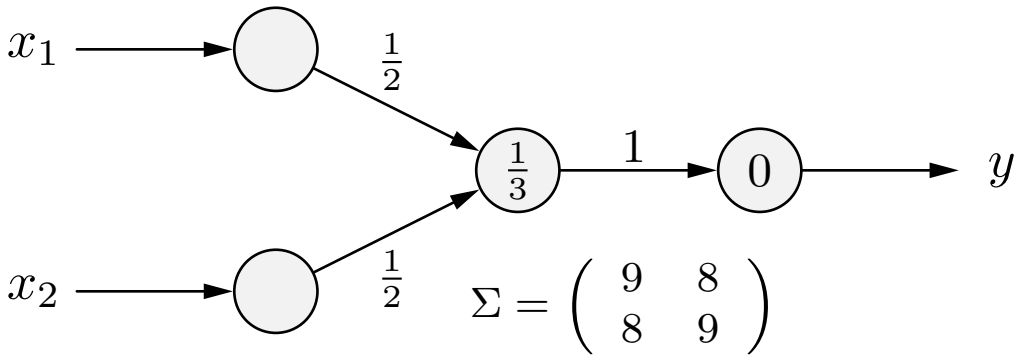
Generalization of the distance function

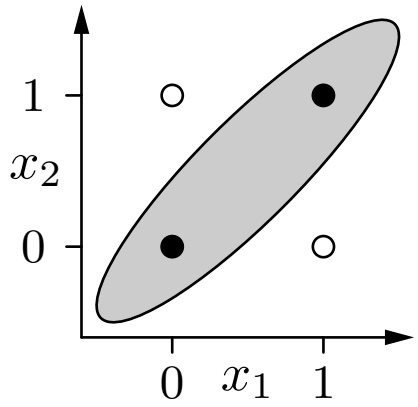
Idea: Use anisotropic (direction dependent) distance function.

Example: **Mahalanobis distance**

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})}.$$

Example: **biimplication**





Application: Recognition of Handwritten Digits

- **Comparison of various classifiers:**
  - Nearest Neighbor (1NN)
  - Decision Tree (C4.5)
  - Multi-Layer Perceptron (MLP)
  - Learning Vector Quantization (LVQ)
  - Radial Basis Function Network (RBF)
  - Support Vector Machine (SVM)
- **Distinction of the number of RBF training phases:**
  - 1 phase: find output connection weights e.g. with pseudo-inverse.
  - 2 phase: find RBF centers e.g. with some clustering plus 1 phase.
  - 3 phase: 2 phase plus error backpropagation training.
- **Initialization of radial basis function centers:**
  - Random choice of data points
  - c-means Clustering
  - Learning Vector Quantization
  - Decision Tree (one RBF center per leaf)

Application: Recognition of Handwritten Digits

Classification results:

Classifier	Accuracy
Nearest Neighbor (1NN)	97.68%
Learning Vector Quantization (LVQ)	96.99%
Decision Tree (C4.5)	91.12%
2-Phase-RBF (data points)	95.24%
2-Phase-RBF (c-means)	96.94%
2-Phase-RBF (LVQ)	95.86%
2-Phase-RBF (C4.5)	92.72%
3-Phase-RBF (data points)	97.23%
3-Phase-RBF (c-means)	98.06%
3-Phase-RBF (LVQ)	98.49%
3-Phase-RBF (C4.5)	94.83%
Support Vector Machine (SVM)	98.76%
Multi-Layer Perceptron (MLP)	97.59%

- LVQ: 200 vectors (20 per class)
  C4.5: 505 leaves
  c-means: 60 centers(?) (6 per class)
  SVM: 10 classifiers, $\approx$ 4200 vectors
  MLP: 1 hidden layer with 200 neurons
- Results are medians of three training/test runs.
- Error backpropagation improves RBF results.