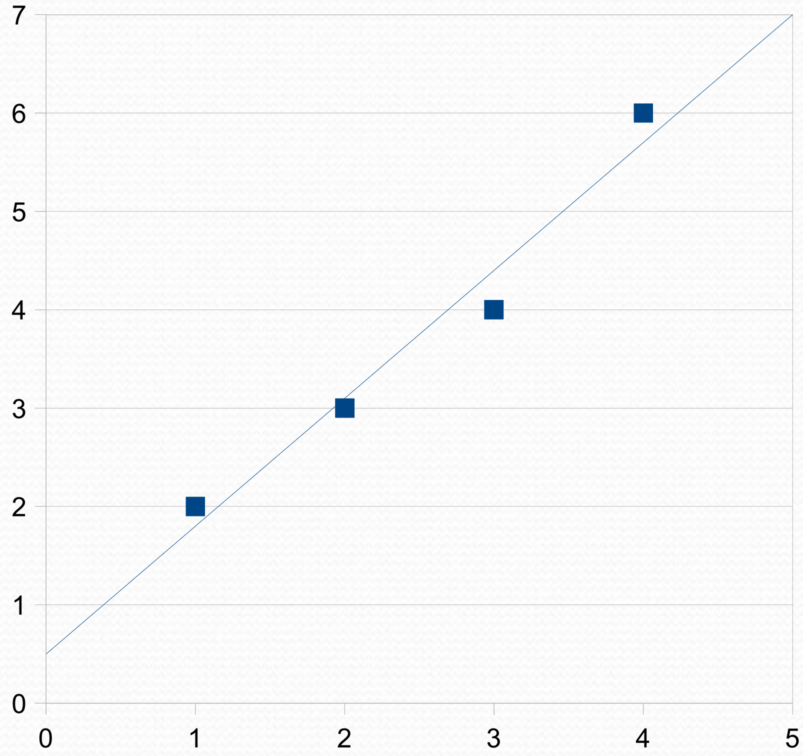
Neural Networks

- Linear regression (again)
- Radial basis function networks
- Self-organizing maps
- Recurrent networks

Partially based on slides by John A. Bullinaria and J. Kok

Linear Regression



$$\mathbf{X} = \begin{pmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vec{x}_3^T \\ \vec{x}_4^T \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 6 \end{pmatrix}$$

Linear Regression

Search for $\vec{\beta}$ such that

$$|\beta_1 x_{i1} + \beta_2 x_{i2} + \cdots \beta_n x_{in} - y_i| = |\vec{x}_i^T \vec{\beta} - y_i|$$

is small for all *i*

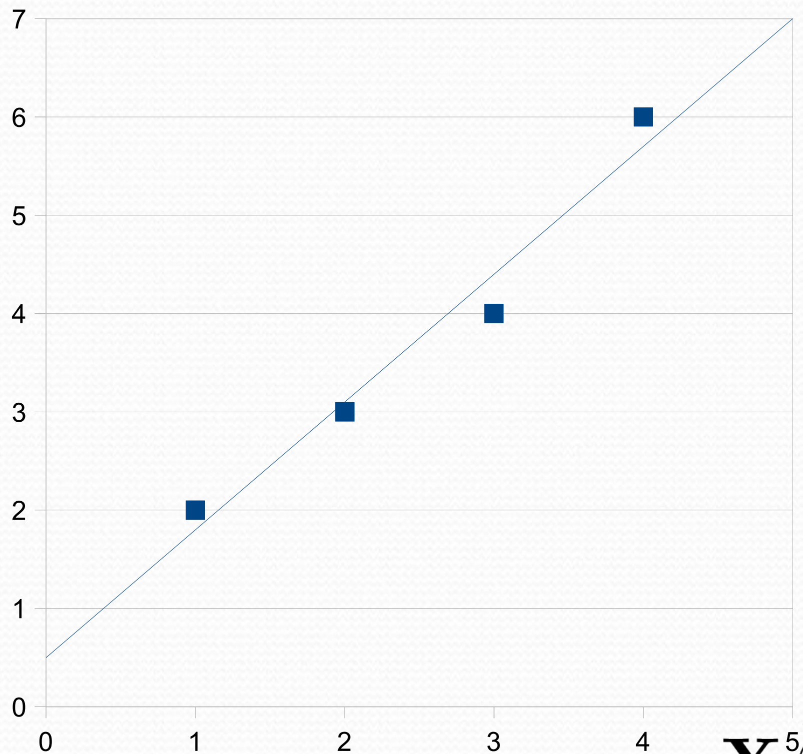
Add to find intercept

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 6 \end{pmatrix} \qquad \mathbf{X} = \begin{pmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vec{x}_3^T \\ \vec{x}_4^T \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$

Linear Regression

$$\mathbf{X} = \begin{pmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vec{x}_3^T \\ \vec{x}_4^T \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 6 \end{pmatrix}$$



Example: $\vec{\beta} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$$\mathbf{X}\vec{\beta} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \quad \mathbf{X}\vec{\beta} - \vec{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

Linear Regression

- Error function:

$$E = \frac{1}{2}\sum_i (\vec{x}_i^T\vec{\beta} - y_i)^2 = \frac{1}{2}(\mathbf{X}\vec{\beta} - \vec{y})^T(\mathbf{X}\vec{\beta} - \vec{y})$$

- Compute global minimum by means of derivative:

$$\nabla_{\vec{\beta}} E = \begin{pmatrix} \frac{\partial E}{\partial \beta_1} \\ \vdots \\ \frac{\partial E}{\partial \beta_n} \end{pmatrix} = \begin{pmatrix} \sum_i (\vec{x}_i^T\vec{\beta} - y_i)\vec{x}_{i1} \\ \vdots \\ \sum_i (\vec{x}_i^T\vec{\beta} - y_i)\vec{x}_{in} \end{pmatrix}$$

$$= \sum_i (\vec{x}_i^T\vec{\beta} - y_i)\vec{x}_i$$

Linear Regression

- Compute global minimum by means of derivative:

$$\nabla_{\vec{\beta}} E = \sum_i (\vec{x}_i^T \vec{\beta} - y_i)\vec{x}_i = \mathbf{0}$$

$$\nabla_{\vec{\beta}} E = \begin{pmatrix} | & & | \\ \vec{x}_1 & \ldots & \vec{x}_n \\ | & & | \end{pmatrix} \begin{pmatrix} \vec{x}_1^T \vec{\beta} - y_1 \\ \vdots \\ \vec{x}_n^T \vec{\beta} - y_n \end{pmatrix} = \mathbf{0}$$

$$\nabla_{\vec{\beta}} E = \begin{pmatrix} | & & | \\ \vec{x}_1 & \ldots & \vec{x}_n \\ | & & | \end{pmatrix} \left[\begin{pmatrix} - & \vec{x}_1^T & - \\ & \vdots & \\ - & \vec{x}_n^T & - \end{pmatrix} \vec{\beta} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right] = \mathbf{0}$$

Linear Regression

- Compute global minimum by means of derivative:

$$\nabla_{\vec{\beta}} E = \begin{pmatrix} | & & | \\ \vec{x}_1 & \dots & \vec{x}_n \\ | & & | \end{pmatrix} \left[\begin{pmatrix} - & \vec{x}_1^T & - \\ & \vdots & \\ - & \vec{x}_n^T & - \end{pmatrix} \vec{\beta} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right] = \mathbf{0}$$

$$\mathbf{X}^T(\mathbf{X}\vec{\beta} - \vec{y}) = \mathbf{0}$$

$$\mathbf{X}^T\mathbf{X}\vec{\beta} = \mathbf{X}^T\vec{y}$$

$$\vec{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}$$

Linear Regression

- Online learning; given one example, the error is:

$$E = \frac{1}{2}(\vec{x}^T\vec{\beta} - y)^2$$

- Taking the derivative with respect to one weight:

$$\frac{\partial E}{\partial \beta_i} = (\vec{x}^T\vec{\beta} - y)x_i$$

- Update weight:

$$\beta_i \leftarrow \beta_i + \eta(y - \vec{x}^T\vec{\beta})x_i$$

Radial Basis Function Networks

RBF is not a multi-layered perceptron

- Localized activation function

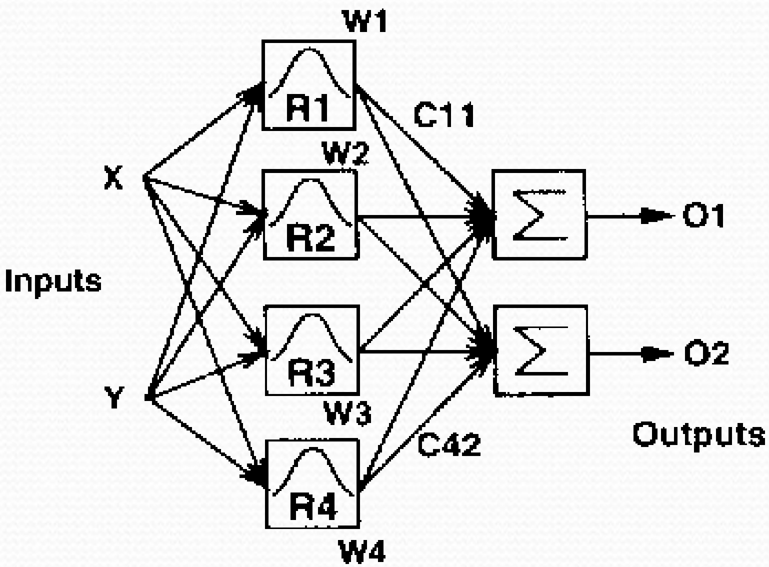
$$R_i(x) = e^{\left(-\frac{\|x-u_i\|^2}{2\sigma_i^2}\right)} \qquad R_i(x) = 1/\left(1+e^{-\frac{\|x-u_i\|^2}{\sigma_i^2}}\right)$$

- Weighted sum or average output

$$d(x) = \sum_{i=1}^{H} c_i R_i(x) \qquad d(x) = \frac{\sum_{i=1}^{H} c_i R_i(x)}{\sum_{i=1}^{H} R_i(x)}$$

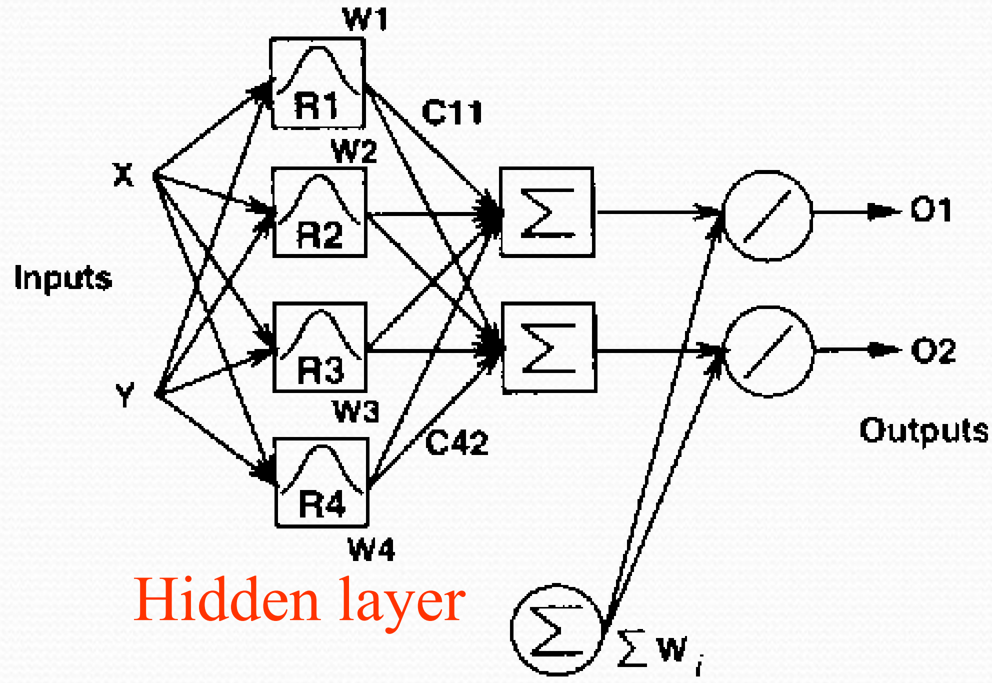
RBFN architecture

Weighted sum



Hidden layer

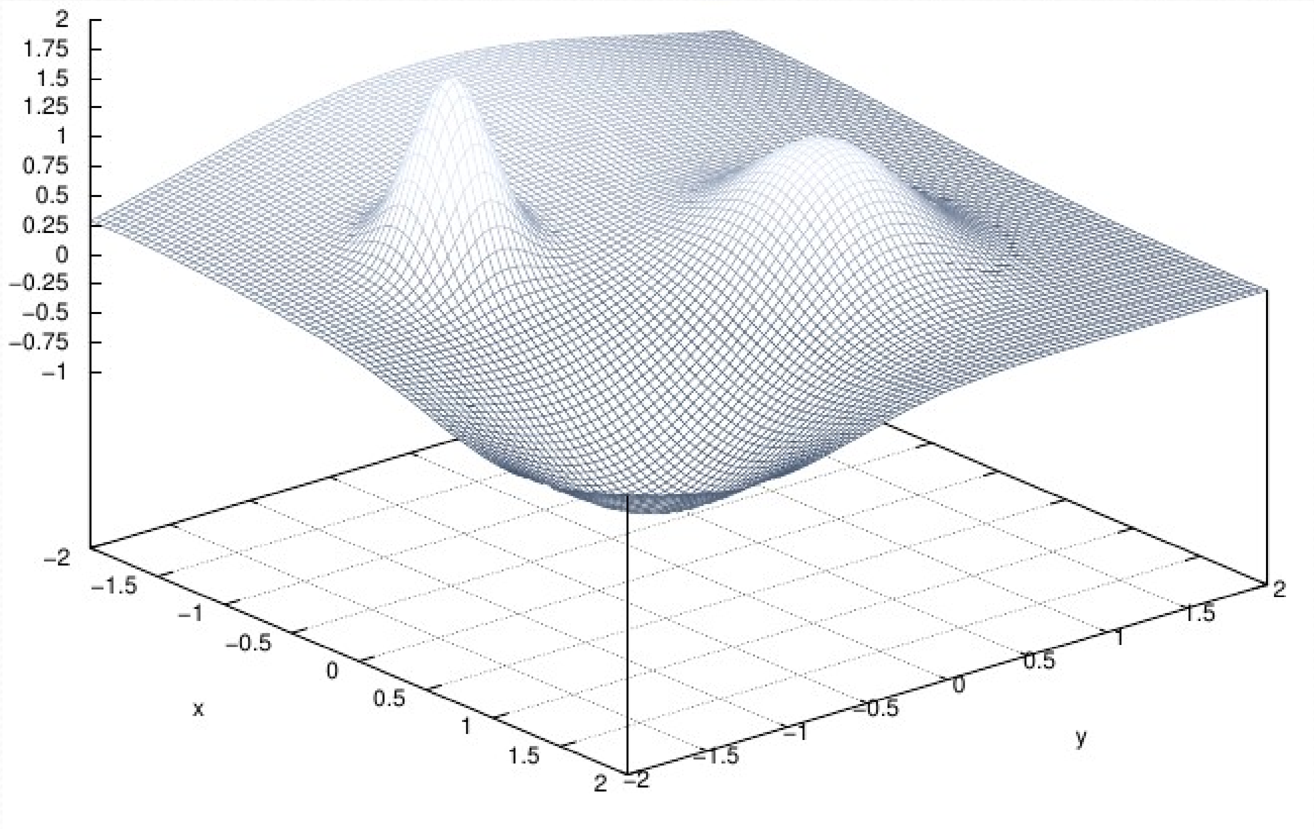
Weighted average



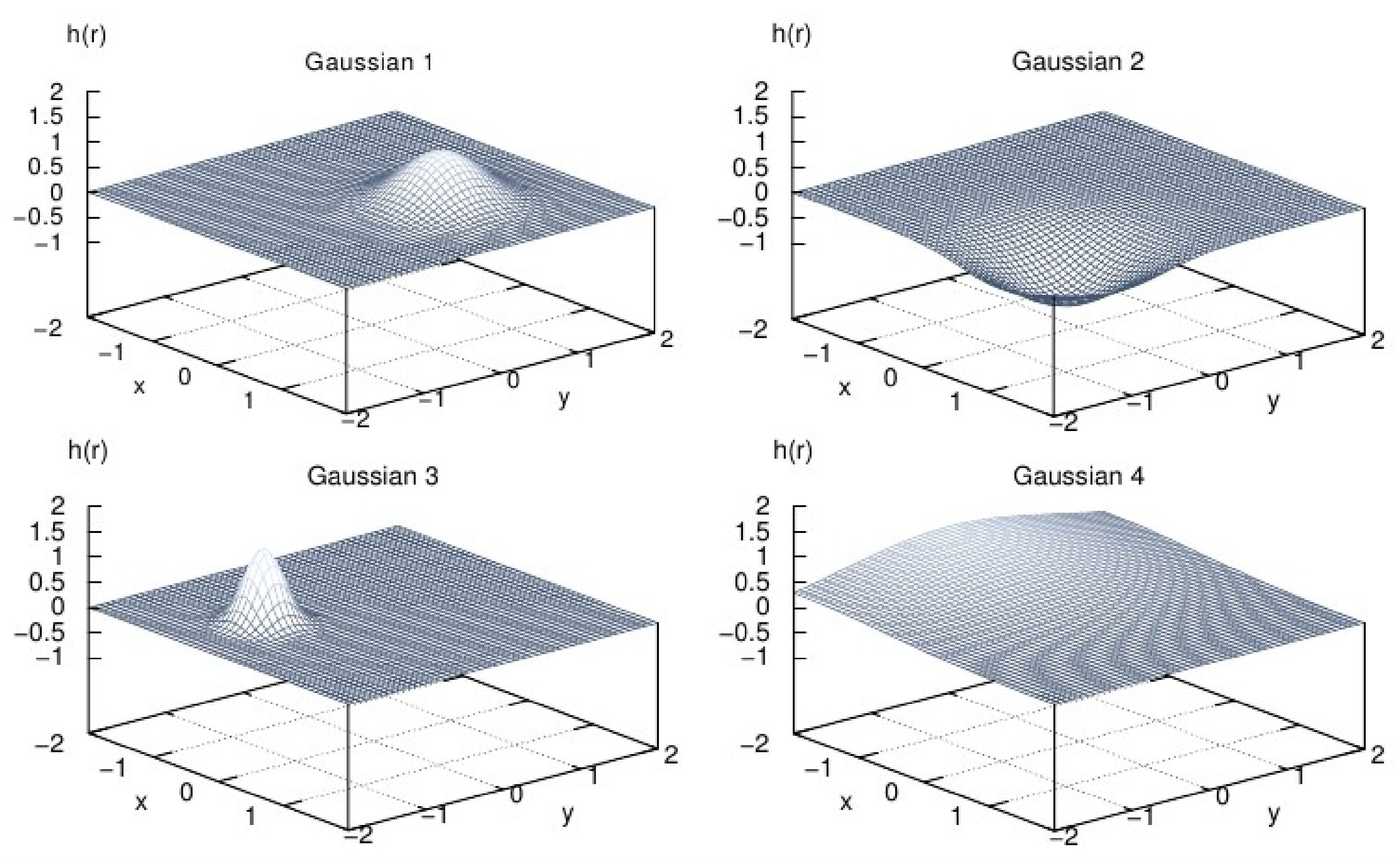
Hidden layer

Localized activation functions
in the hidden layer

RBFN Example



RBFN Example



RBFN Learning

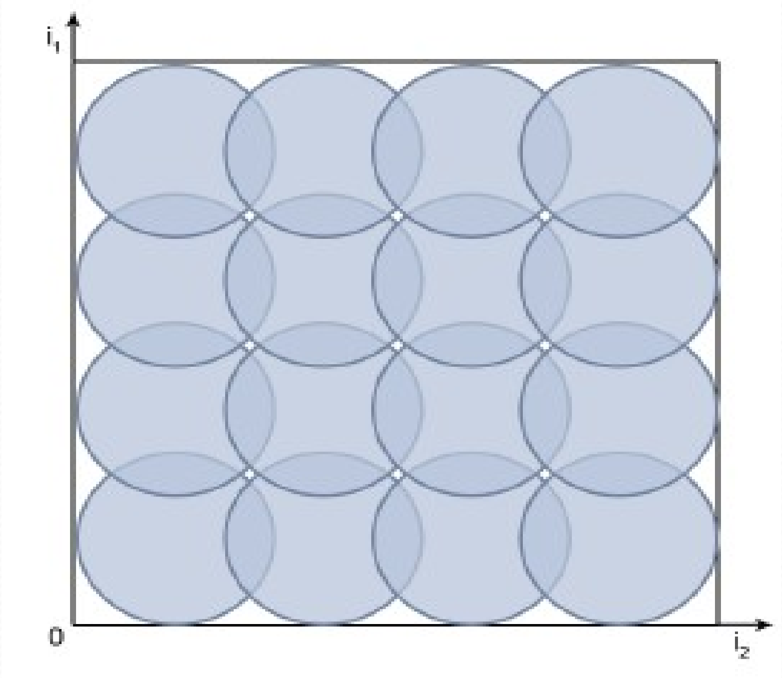
- Three types of parameters:
  - centers of the radial basis functions
  - width of the radial basis functions
  - weights for each radial basis function
- "Obvious" algorithm: backpropagation?

RBFN Hybrid Learning

- **Step 1:** Fix the RBF centers and widths
- **Step 2:** Learn the linear weights

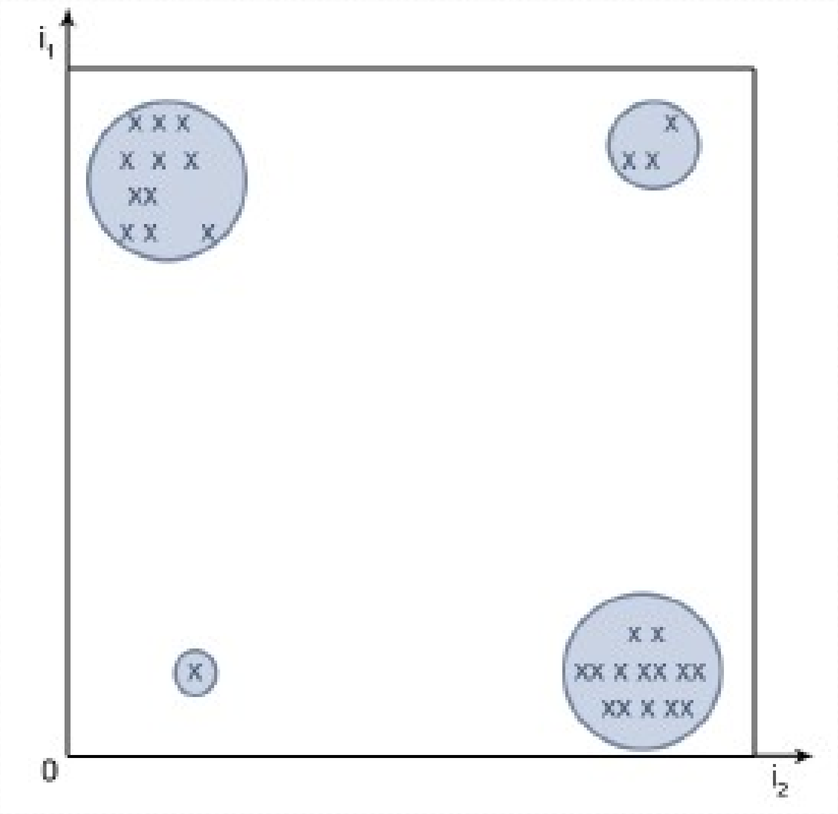
RBFN Hybrid Learning

- **Step 1:** Fixed selection



RBFN Hybrid Learning

- **Step 1:** Clustering



RBFN Hybrid Learning

- **Step 2:** linear regression!

1. Calculate for each pattern its (normalized) RBF value, for each of the neurons

2. Create a table:

Output Neuron 1	Output Neuron 2	...	Output Neuron *n*	Desired Output
...	...	...	...	...

3. Linear regression

RBFN vs MLP

- The hidden layer of a RBFN does *not* compute a weighted sum, but a distance to a center
- The layers of a RBFN are usually trained one layer at a time
- RBFNs consitute a set of *local* models, MPLs represent a global model
- A RBFN will predict 0 when it doesn't know anything
- The number of neurons in a RBFN for accurate prediction can be high
- Removing one neuron can have a large influence

RBFN vs Sugeno Systems?

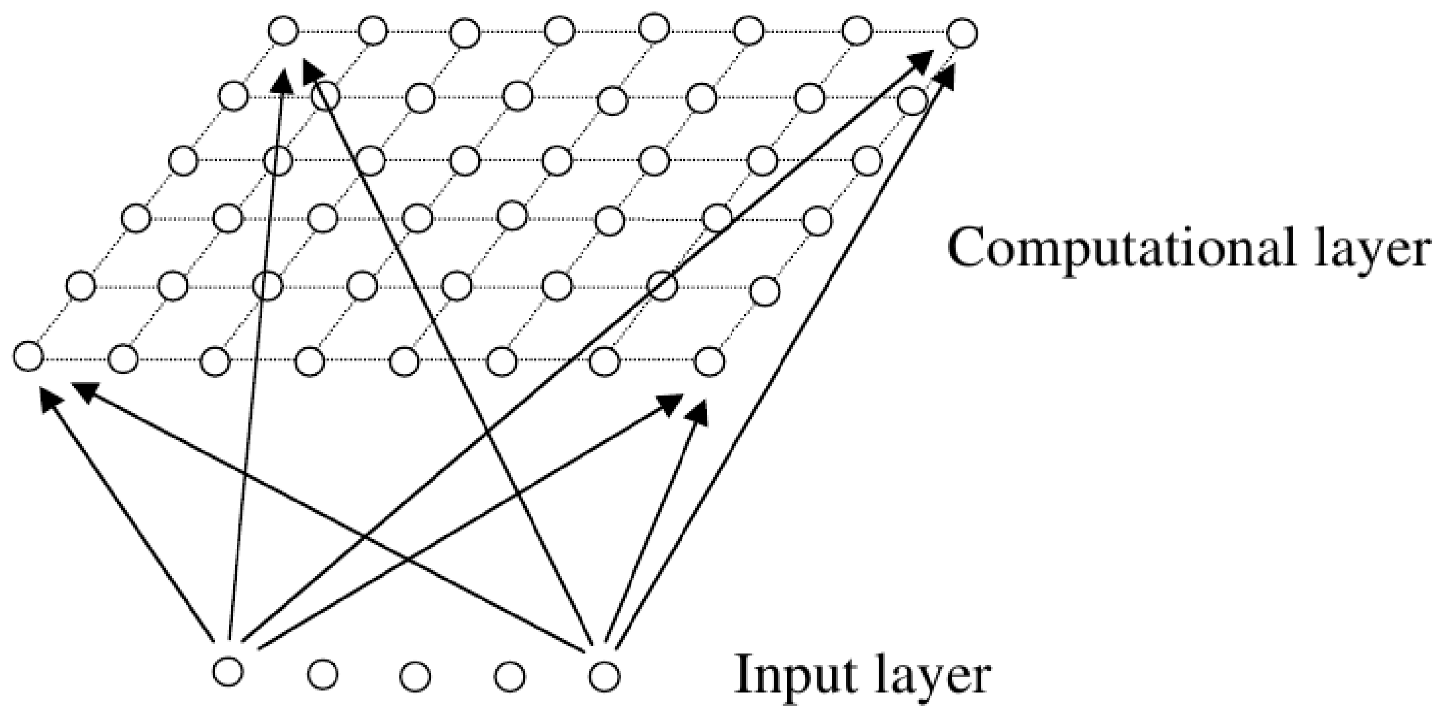
Self-Organising Maps (Kohonen Networks)

- Unsupervised setting

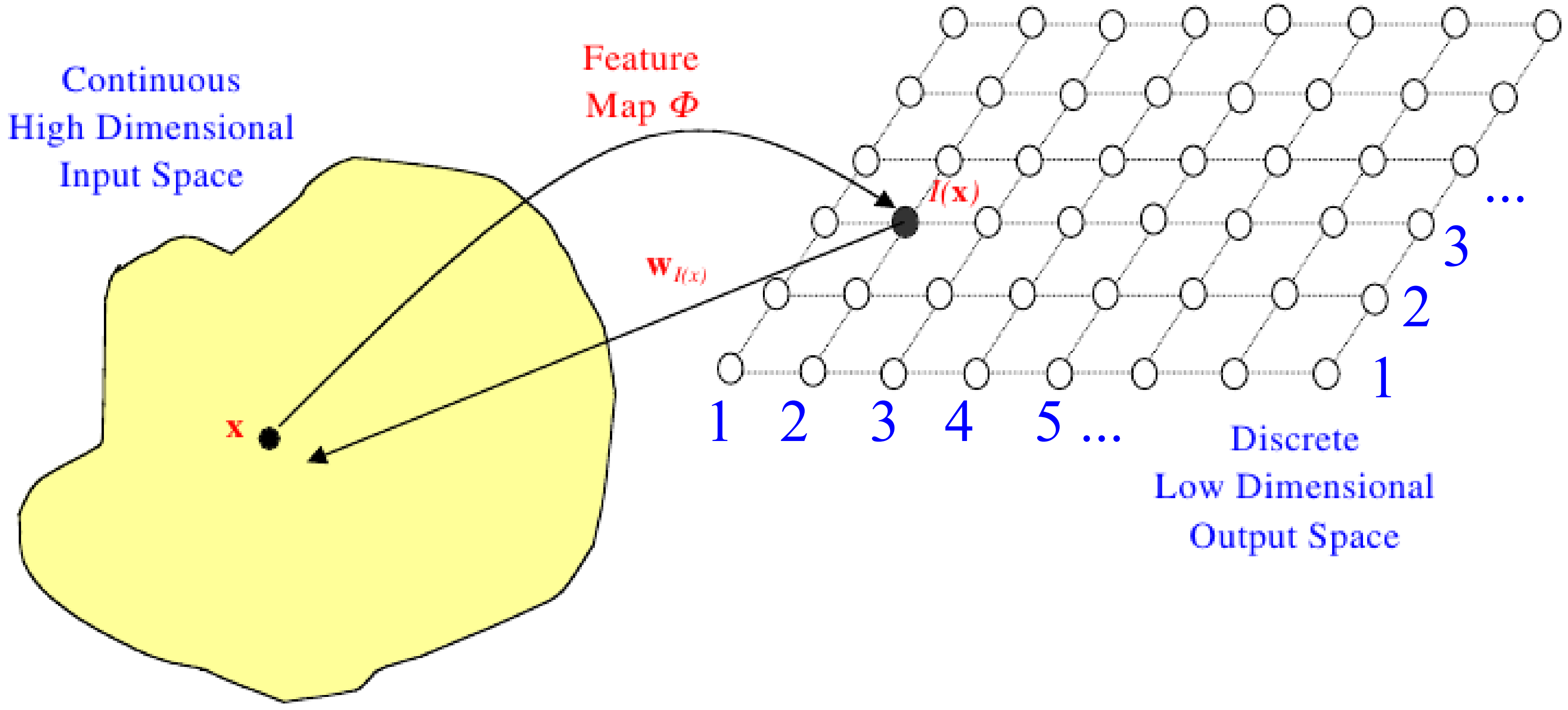
- These networks can be used to
  - cluster a space of patterns
  - learn nodes in hidden layer of a RBFN
  - map a high dimensional space to a lower dimensional one
  - solving traveling salesman problems heuristically

Kohonen Networks

- Example: network in a grid structure



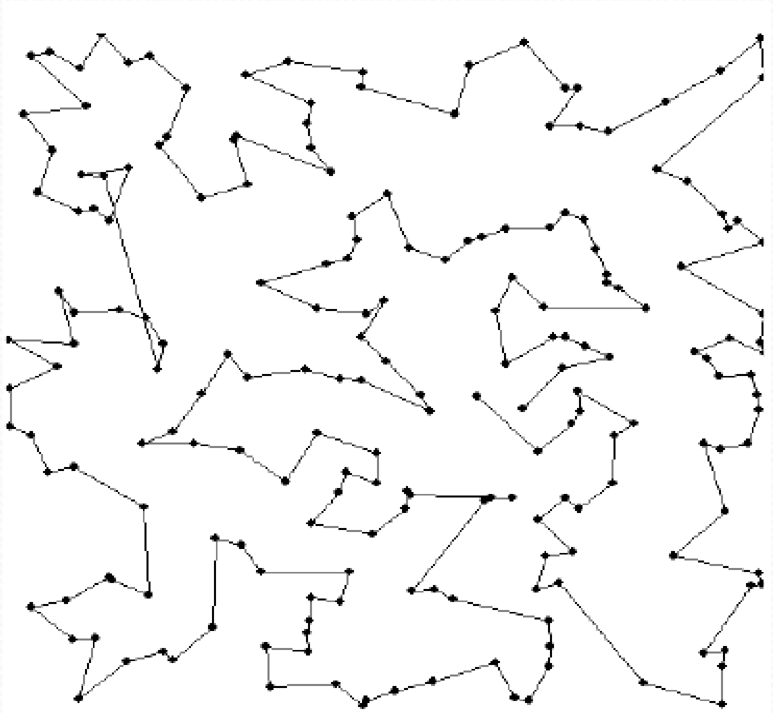
Kohonen Networks



Mapping such that points close in input are close in output

Kohonen Networks

- Solving a traveling salesman problem using a network in a circular structure: cities close on a map should be close on the tour



(Elastic net)

Kohonen Networks: Algorithm

- **Step 1**: initialize weights for each node at random
- **Step 2:** sample a training pattern
- **Step 3:** compute which node is closest to the sample
- **Step 4:** adapt the weights of this node such that this node is even closer to the pattern next time
- **Step 5:** adapt the weight of *closeby* nodes (in the grid, on the line, …) such that also these other nodes are close
- Go to step 2

Kohonen Networks: Algorithm

- **Step 3:** distance calculation for node *i*

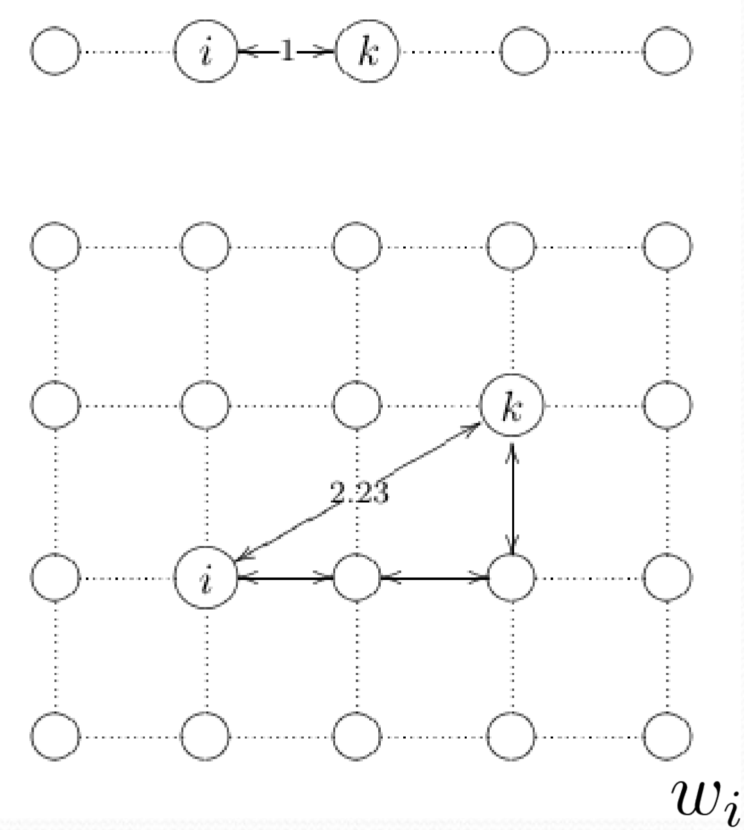
$$||\vec{x} - \vec{w}_i|| = \sum_j (x_j - w_{ij})^2$$

- **Step 4:** adapt weights for node *i* (update rule)

$$w_{ij} \leftarrow w_{ij} + \eta(x_i - w_{ij})$$

Kohonen Networks: Algorithm

- **Step 5:** adapt weights of nodes *closeby*



Step 5a: calculate distance $d(i,i^*)$ between two nodes in the grid / on the line

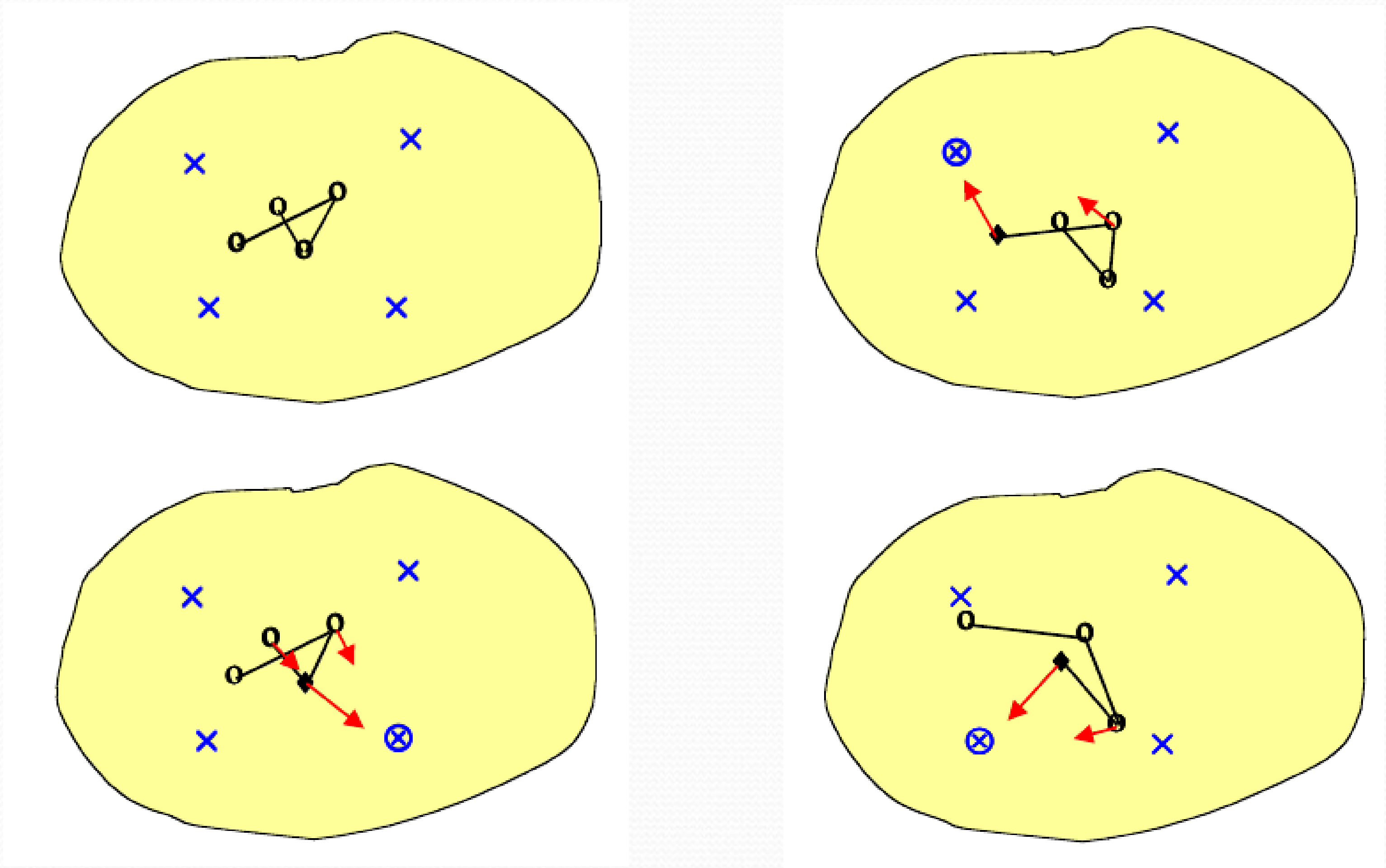
Step 5b: reweigh the distance (closeby = high weight)

$$\Lambda(i, i^*) = e^{-(d(i,i^*))^2/2\sigma^2}$$

Step 5c: update weight nearby

$$w_{i^*j} \leftarrow w_{i^*j} + \eta\Lambda(i, i^*)(x_i - w_{ij})$$

Kohonen Networks: Illustration

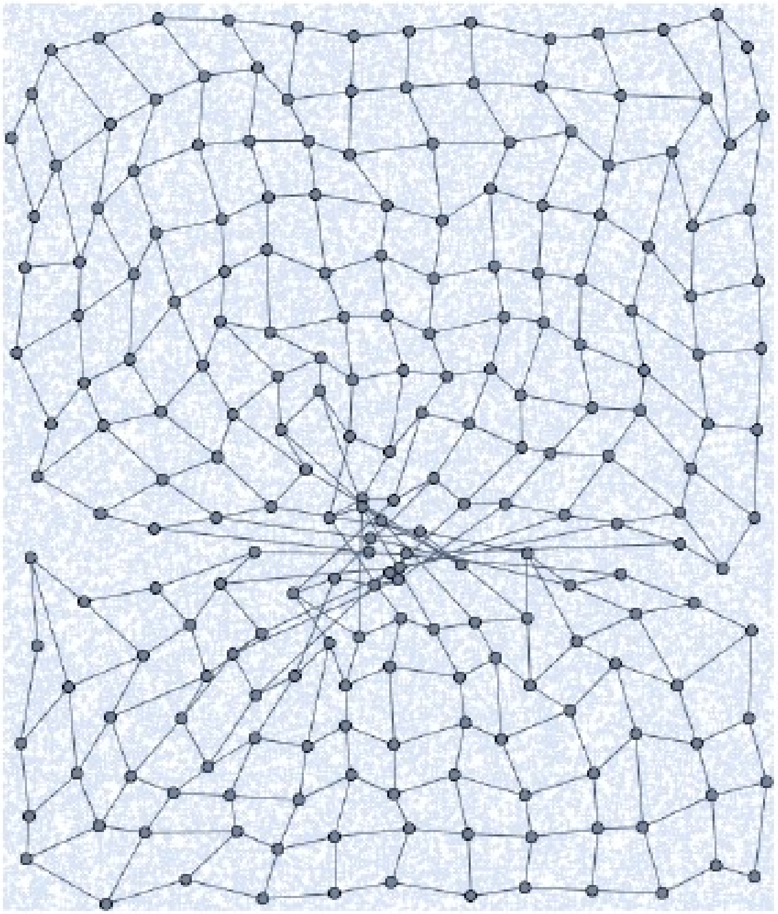


Kohonen Networks: Defects

- Avoiding "knots":
  - higher σ
  - higher learning rate

    $\eta(t) = \eta_0 e^{-t/\tau}$

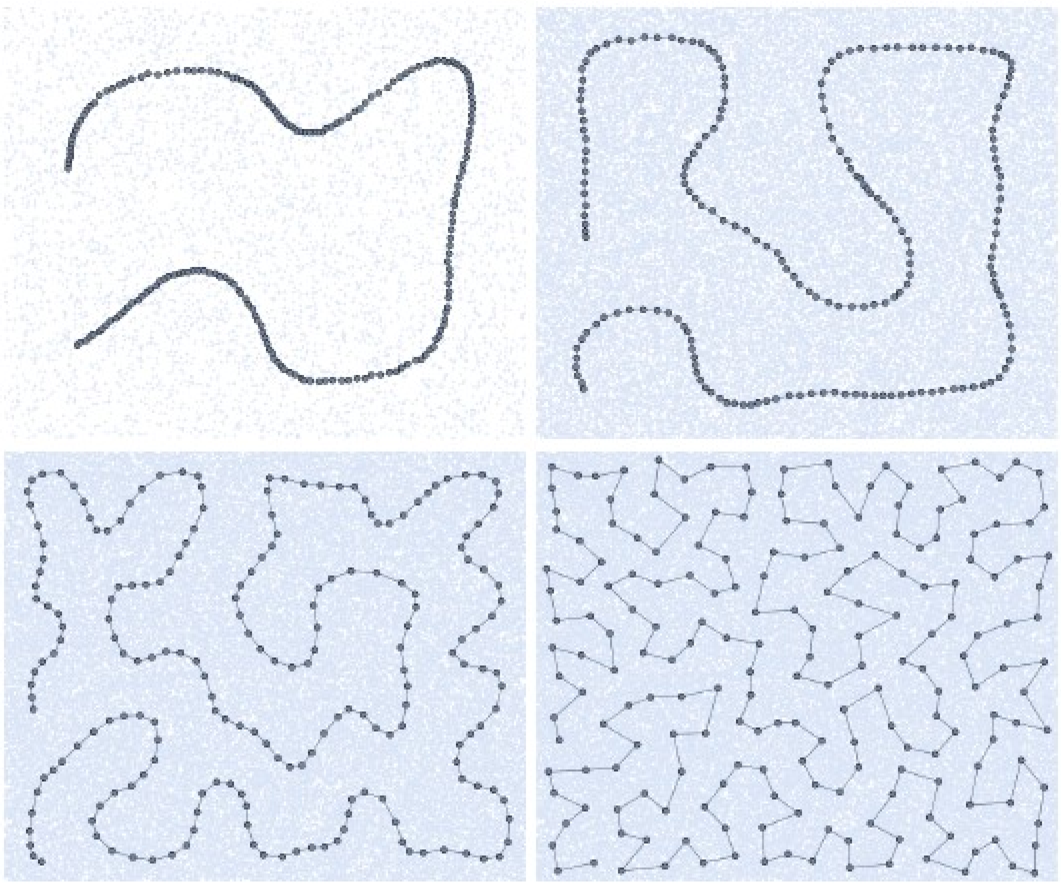
  in early iterations



Kohonen Networks: Examples

5000 uniform samples from 2D space

50000 samples

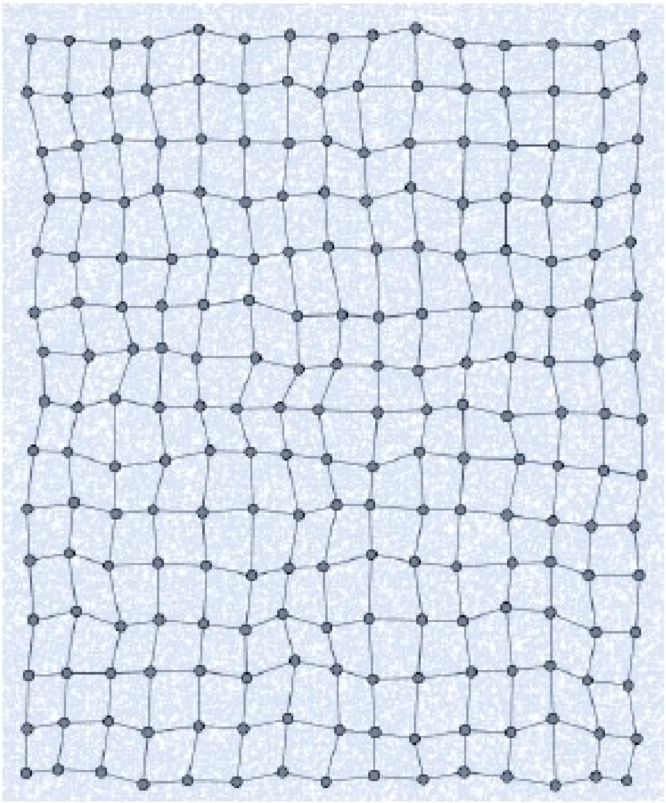


70000 samples

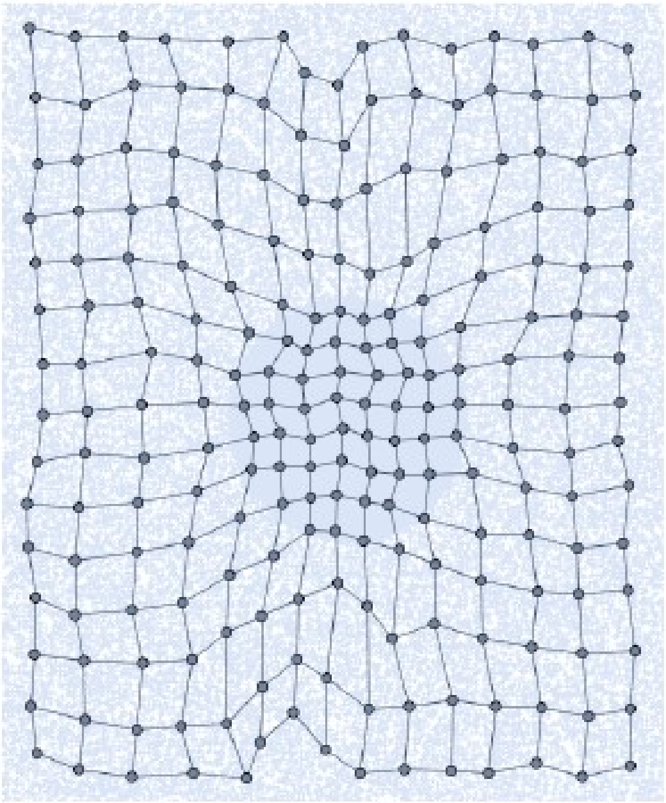
80000 samples

Kohonen Networks: Examples

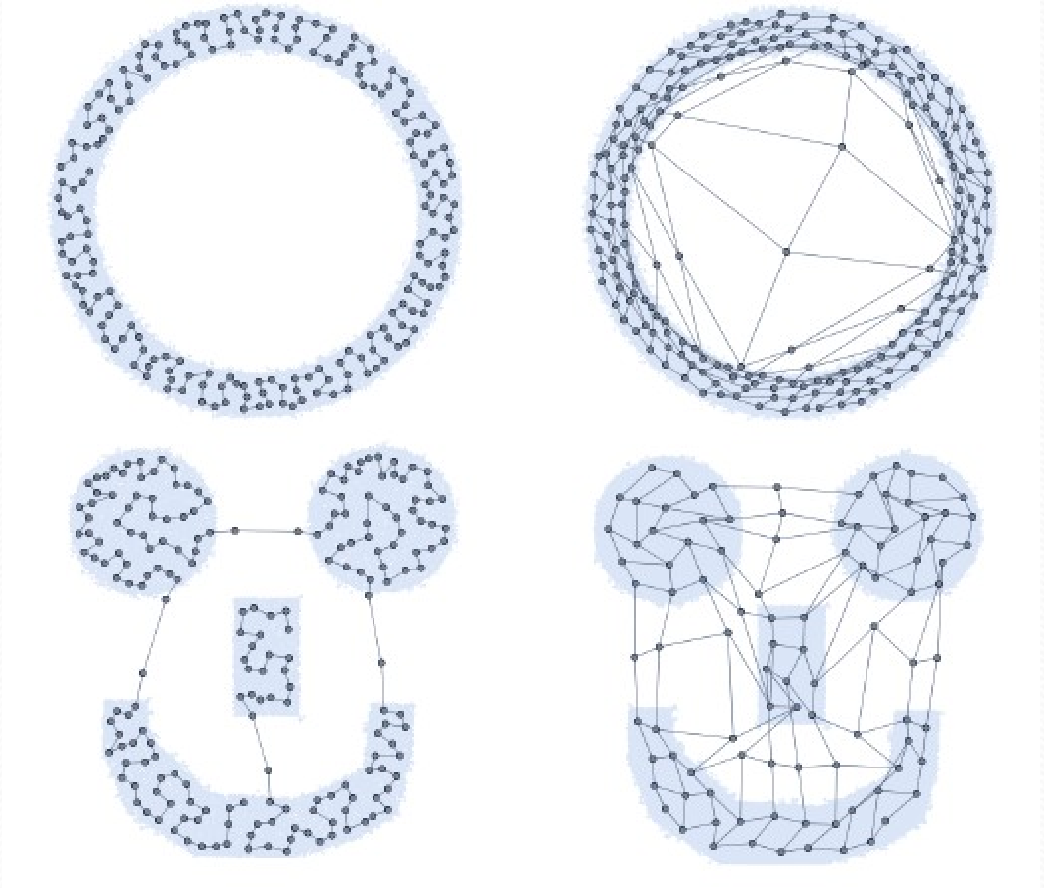
Uniform



Not uniform



Kohonen Networks: Examples

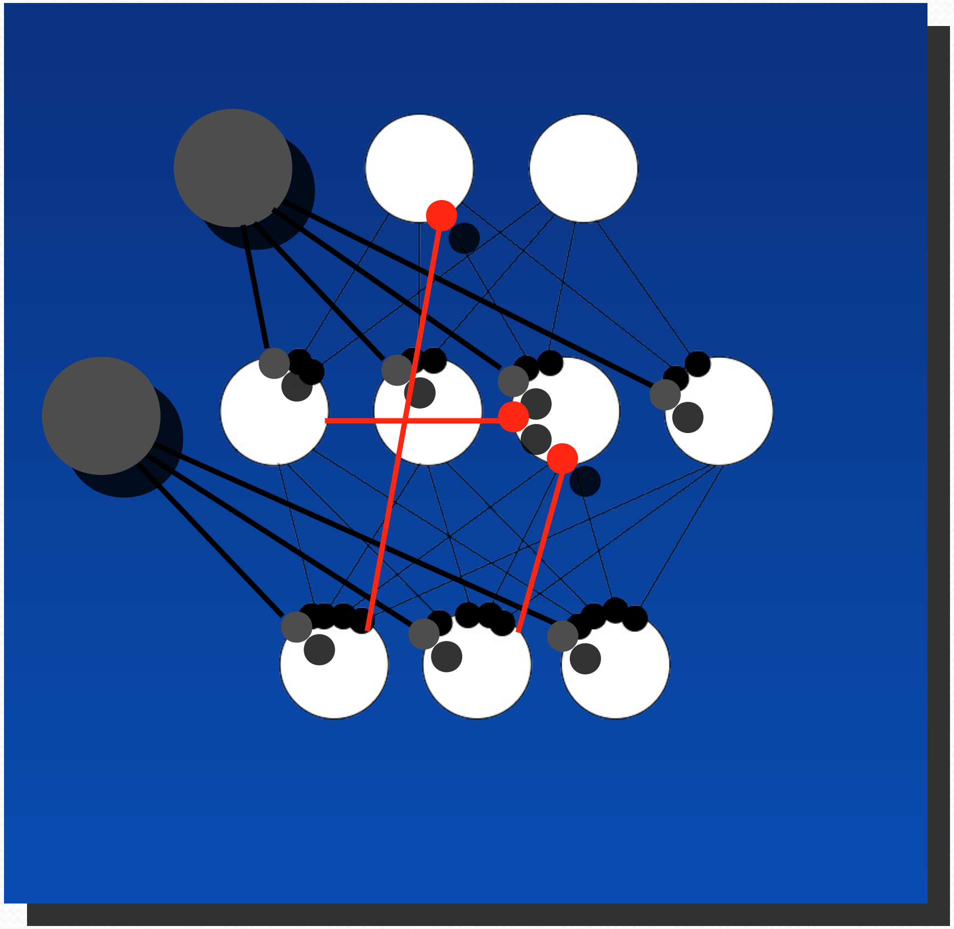


Kohonen Networks

- How to use for clustering?
- How to use to build RBF networks?

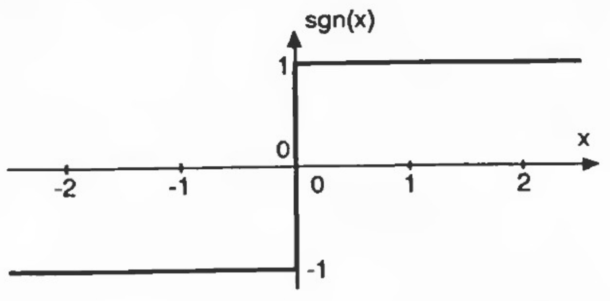
Recurrent Networks

- The output of any neuron can be the input of any other

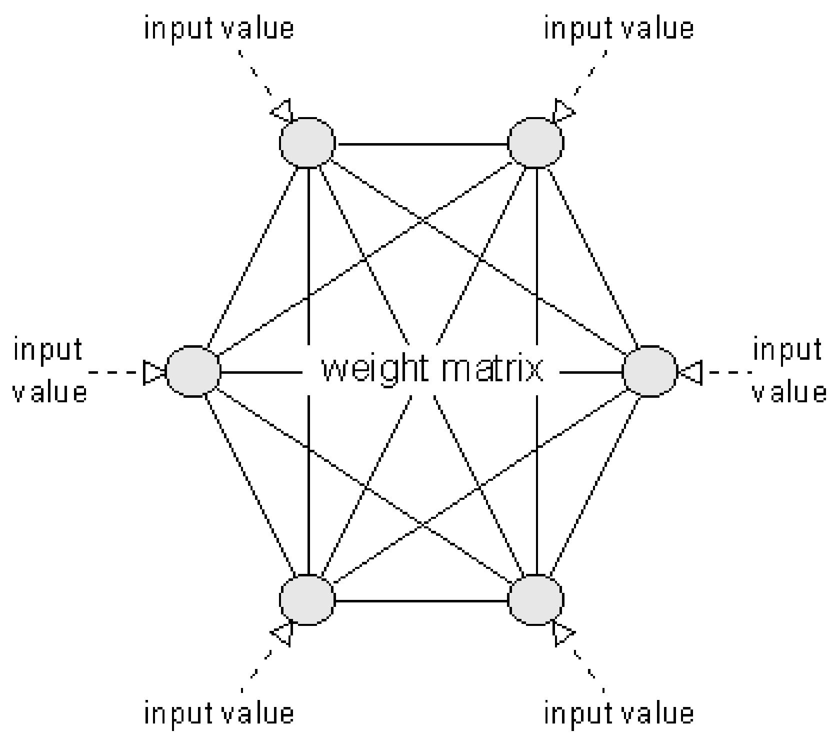


Hopfield (Recurrent) Network

Activation function:



Input = activation: {-1,1}

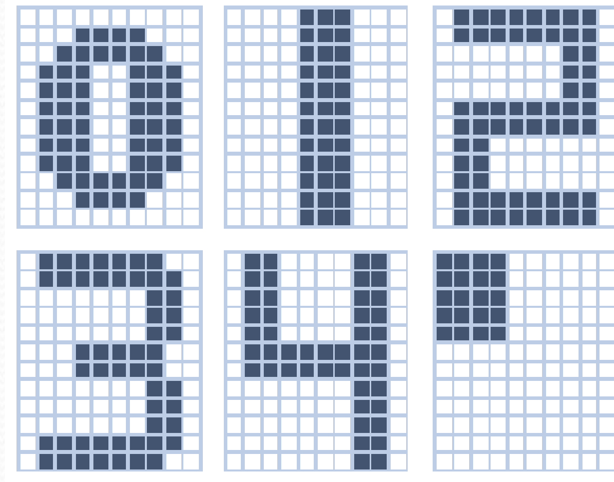


Hopfield Network: Input Processing

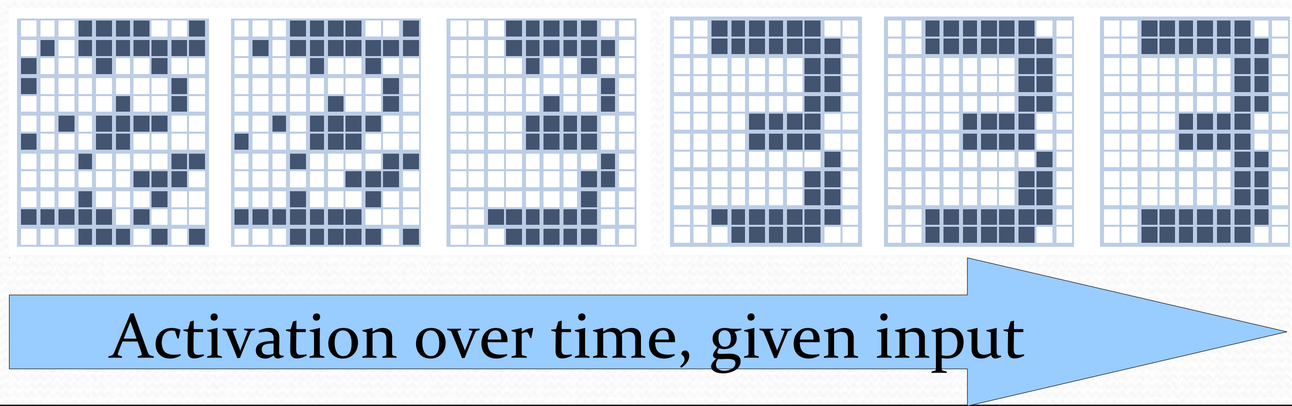
- Given an input $\vec{x}$
- Asynchronously: (Common)
  - **Step 1:** sample an arbitrary unit
  - **Step 2:** update its activation
  - **Step 3:** if activation does not change, stop, otherwise repeat
- Synchronously:
  - **Step 1:** save all current activations (time *t*)
  - **Step 2:** recompute activation for all units a time *t+1* using activations at time *t*
  - **Step 3:** if activation does not change, stop, otherwise repeat

Hopfield Network: Associative Memory

- Patterns “stored” in the network:



- Retrieval task: for given input, find the input that is closest:



Hopfield Network: Learning

- Activation:

$$S_i(t+1) = f\left(\sum_j w_{ij} S_j(t)\right)$$

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Hopfield Network: Learning

- **Definition** A network is stable for one pattern if:

$$f\left(\sum_j w_{ij}\xi_j\right) = \xi_i$$

  where $\vec{\xi}$ is a pattern

- If we pick the weights as follows, the network will be stable for pattern $\vec{\xi}$: (*N* is number of units)

$$w_{ij} = \frac{1}{N}\xi_i\xi_j$$

Hopfield Network: Learning

- Proof for stability:

$$
\begin{aligned}
f\left(\sum_j w_{ij}\xi_j\right) &= f\left(\sum_j \frac{1}{N}\xi_i\xi_j\xi_j\right) \\
&= f\left(\xi_i \sum_j \frac{1}{N}\xi_j^2\right) \\
&= f\left(\xi_i\right) \\
&= \xi_i
\end{aligned}
$$

Hopfield Network: Learning

- Learning multiple patterns:

$$w_{ij} = \frac{1}{N} \sum_{\mu} \xi_i^{\mu} \xi_j^{\mu}$$

- "Hebb rule"
- Ensures that with a high probability approximately 0.139N arbitrary patterns can be stored (no proof given)
- **Simple learning algorithm:** assign all weights once!

Hopfield Network: Learning

- Intuition

$$f\left(\sum_j w_{ij}\xi_j^{\mu}\right) = f\left(\sum_j \frac{1}{N}\sum_{\mu'} \xi^{\mu'}_i \xi^{\mu'}_j \xi_j^{\mu}\right)$$

$$= f\left(\xi_i^{\mu} + \underbrace{\sum_j \frac{1}{N}\sum_{\mu' \neq \mu} \xi^{\mu'}_i \xi^{\mu'}_j \xi_j^{\mu}}_{<0.5}\right)$$

with high probability for 0.139*N* patterns

Hopfield Network: Energy Function

- We define the energy of network activation as:

$$H = C - \sum_{(ij)} w_{ij} S_i S_j$$

- We will show that energy always goes down when updating activations
- Assume we recalculate unit *i*:

$$S_i' = f\left(\sum_j w_{ij} S_j\right)$$

... and that its activation changes $S_i' = -S_i$

Hopfield Network: Energy Function

- Calculate change in energy

$$
\begin{aligned}
H' - H &= -S_i' \sum_{j \neq i} w_{ij} S_j + S_i \sum_{j \neq i} w_{ij} S_j && (1) \\
&= S_i \sum_{j \neq i} w_{ij} S_j + S_i \sum_{j \neq i} w_{ij} S_j && (2) \\
&= 2S_i \sum_{j \neq i} w_{ij} S_j && (3) \\
&= 2S_i \sum_{j} w_{ij} S_j - 2w_{ij} \qquad \left(sign\left(\sum_j w_{ij} S_j \right) \neq S_i \right) && (4) \\
&= 2S_i \sum_{j} w_{ij} S_j - 2w_{ij} < 0 && (5)
\end{aligned}
$$

Hopfield Network: Energy Function

- Choose as energy function

Note: if $S_i = \xi_i^\mu$, this is 1, sum total is N (maximal)

$$H = -\frac{1}{2N}\sum_\mu \left(\sum_i S_i \xi_i^\mu\right)^2$$

this function has local minima at each of the patterns

- Rewrite:

$$\begin{aligned} H &= -\frac{1}{2N}\sum_\mu \left(\sum_i S_i \xi_i^\mu\right)\left(\sum_j S_i \xi_j^\mu\right) \\ &= -\frac{1}{2}\sum_{ij}\left(\frac{1}{N}\sum_\mu \xi_i^\mu \xi_j^\mu\right) S_i S_j \end{aligned}$$

Next week

- More on recurrent networks
- Deep belief networks
- Slowly moving to variations of evolutionary algorithms