Knowledge-Based Control Systems (SC4081)

Lecture 5: Artificial neural networks

Alfredo Núñez
Section of Railway Engineering
CiTG, Delft University of Technology
The Netherlands

a.a.nunezvicencio@tudelft.nl
tel: 015-27 89355

Robert Babuška
Delft Center for Systems and Control
3mE, Delft University of Technology
The Netherlands

r.babuska@tudelft.nl
tel: 015-27 85117

Outline

1. Introduction to artificial neural networks.

2. Feedforward neural network.

3. Backpropagation.

4. Radial basis function network.

5. Neuro-fuzzy systems.

6. Training and validation aspects.

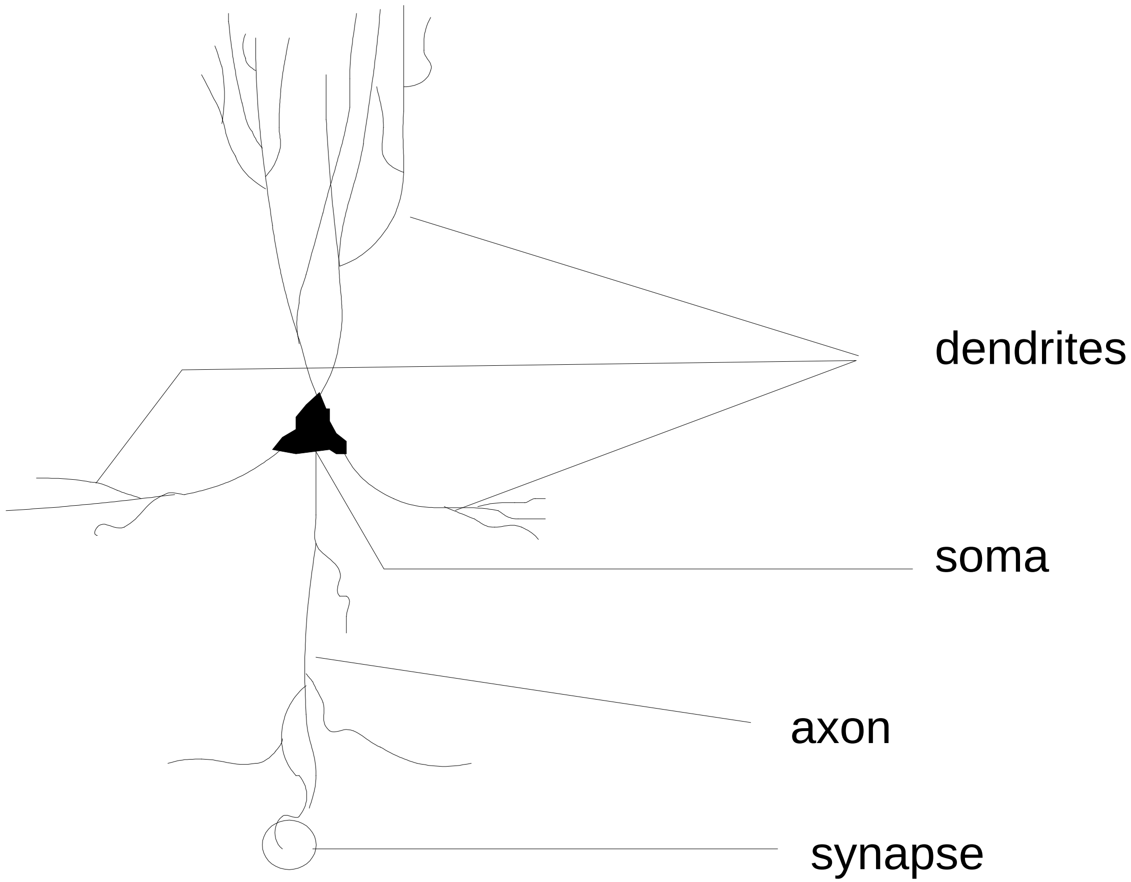
Motivation: biological neural networks

- Humans are able to process complex tasks efficiently (perception, pattern recognition, reasoning, etc.).
- Learning from examples.
- Adaptivity and fault tolerance.

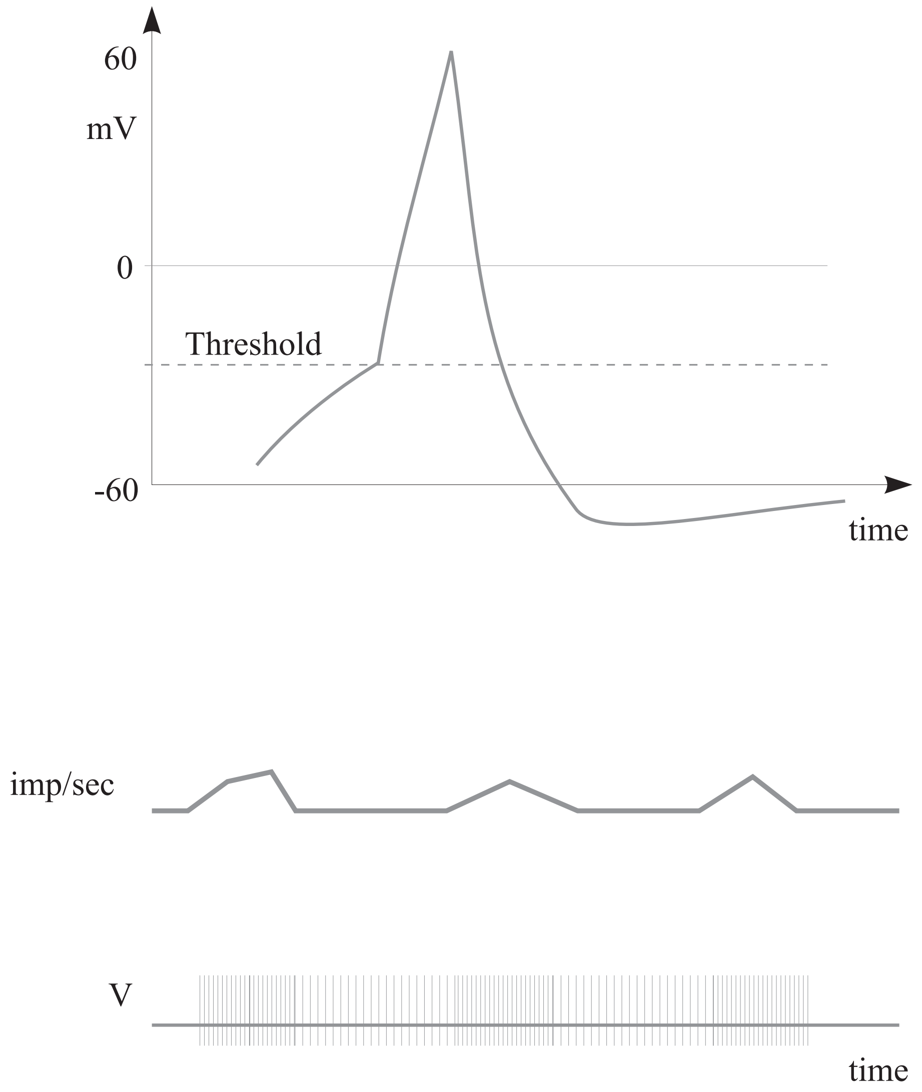
In engineering applications:

- Nonlinear approximation and classification.
- Learning and adaptation from data (black-box models).
- VLSI implementation.

Biological neuron



Signal transfer in biological networks



Learning in neural networks

Biological neural networks:

- Synaptic connections among neurons which simultaneously exhibit high activity are strengthened.

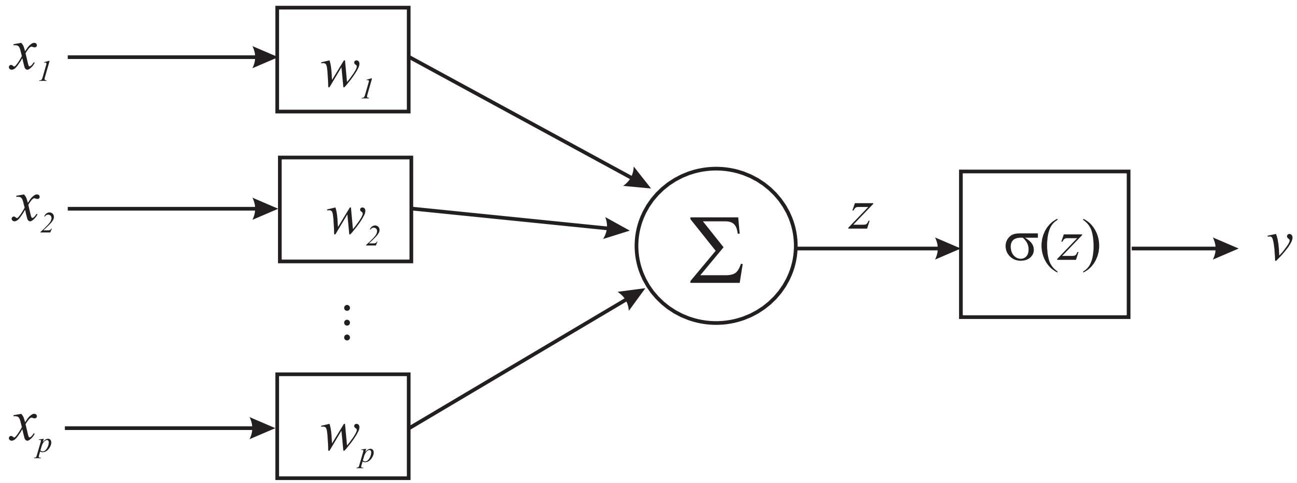
Artificial neural networks:

- Mathematical approximation of biological learning: Hebbian learning (neurocomputing).
- Error minimization, energy minimization (function approximation, classification, optimization).

A bit of history

1943	McCulloch & Pits (first model of neurons)
1949	Hebb (learning)
1957	Rosenblatt (perceptron)
1959	Widrow (ADALINE)
1969	Minsky (critique of ADALINE)
1977	Rummelhart (backpropagation learning)
1982	Hopfield (recurrent network)
1989	Cybenko (approximation theory)
1990–	Jang et.al. (neuro-fuzzy systems)
1993	Barron (complexity vers. accuracy)

Artificial neuron



x_i : ith input of the neuron

w_i : adaptive weight (synaptic strength) for x_i

z : weighted sum of inputs: $z = \sum_{i=1}^{p} w_i x_i = \mathbf{w}^T \mathbf{x}$

$\sigma(z)$: activation function

v : output of the neuron

Activation functions

Purpose: transformation of the input space (squeezing).

Two main types:

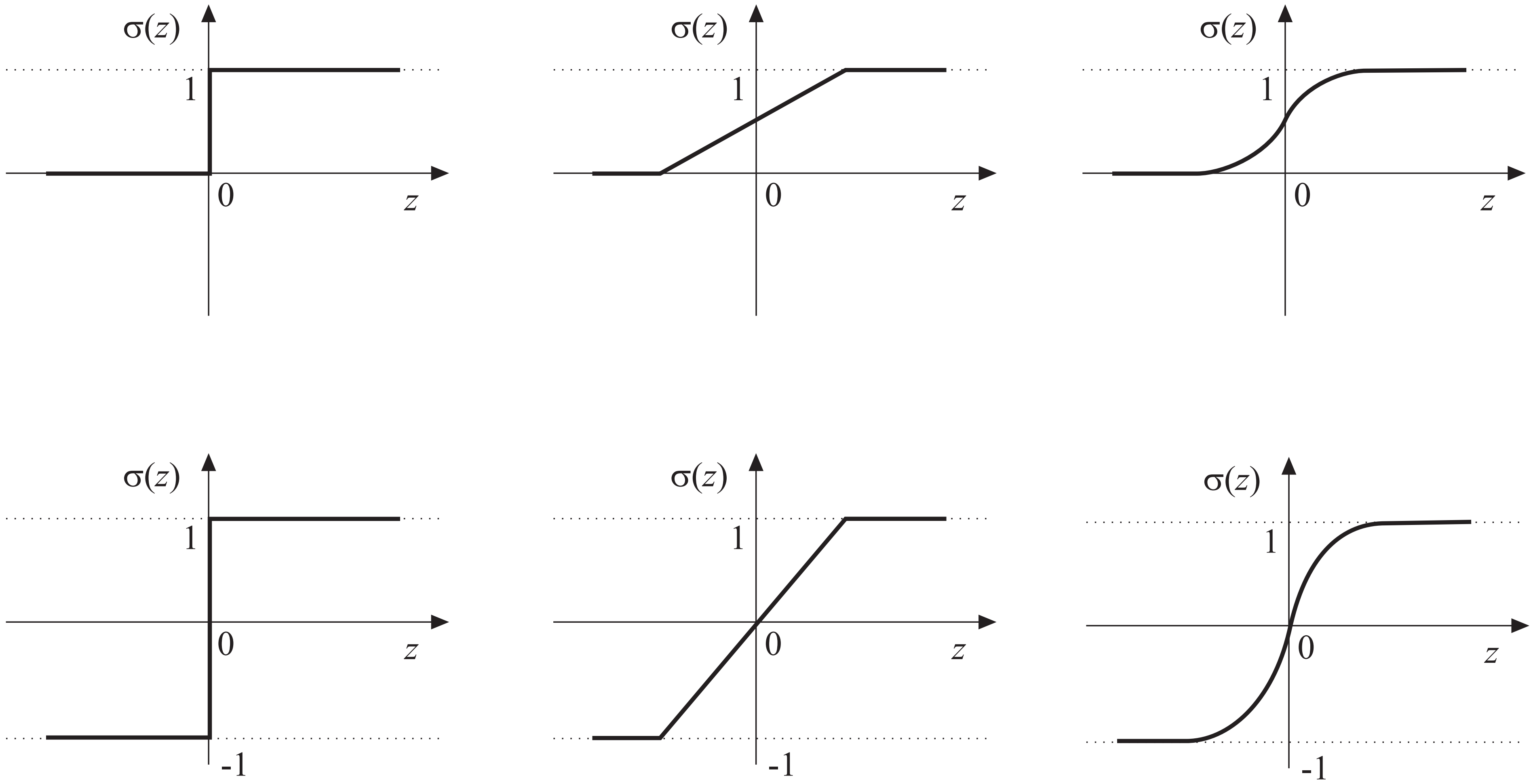
1. **Projection functions: threshold function, piece-wise linear function, tangent hyperbolic, sigmoidal function:**

$$\sigma(z) = 1/(1 + \exp(-2z))$$

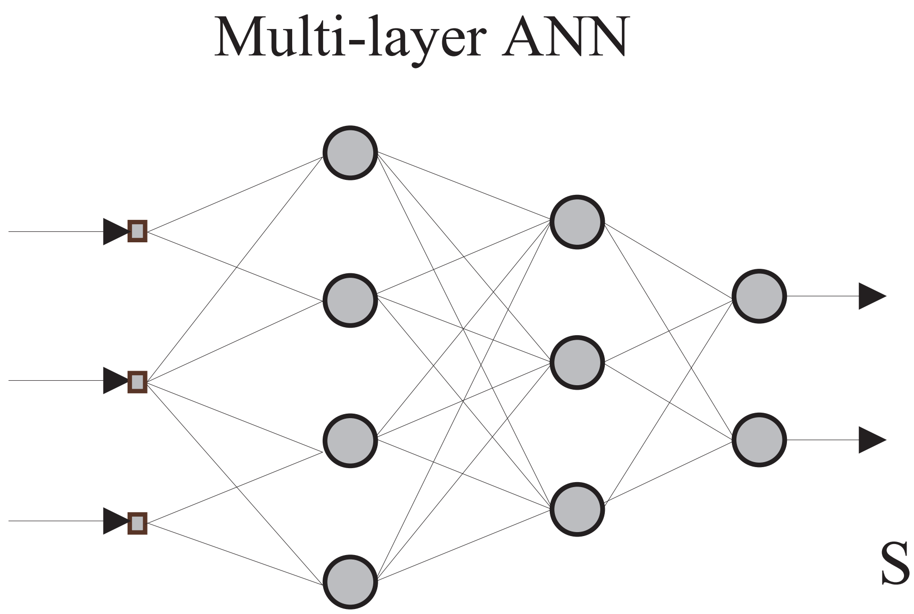
2. **Kernel functions (radial basis functions):**

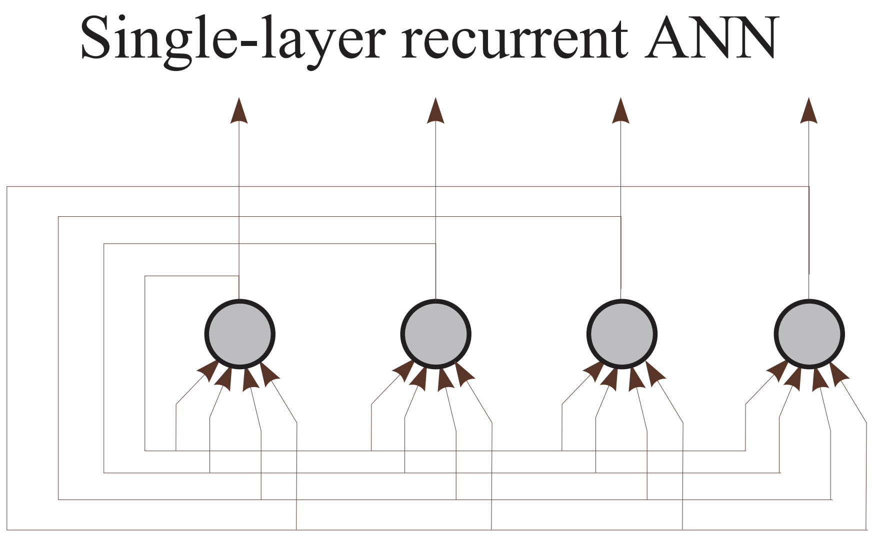
$$\sigma(\mathbf{x}) = \exp\left(-(\mathbf{x} - \mathbf{c})^2/s^2\right)$$

Activation functions

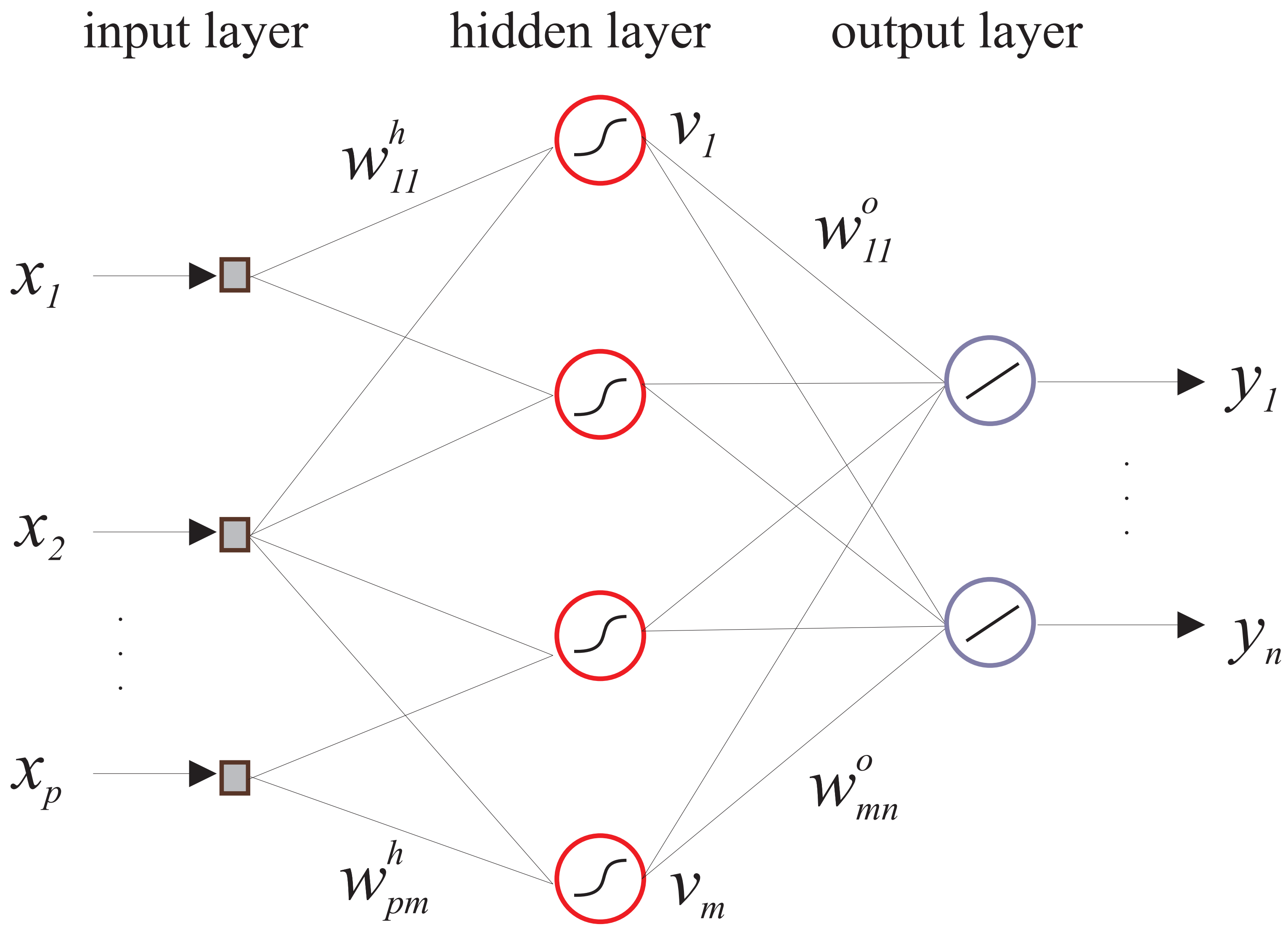


Neural Network: Interconnected Neurons





Feedforward neural network



Feedforward neural network (cont'd)

1. Activation of hidden-layer neuron j:

$$z_j = \sum_{i=1}^{p} w_{ij}^h x_i + b_j^h$$

2. Output of hidden-layer neuron j:

$$v_j = \sigma\left(z_j\right)$$

3. Output of output-layer neuron l:

$$y_l = \sum_{j=1}^{h} w_{jl}^o v_j + b_l^o$$

Input–Output Mapping

Matrix notation:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X}_b \mathbf{W}^h \\ \mathbf{V} &= \sigma(\mathbf{Z}) \\ \mathbf{Y} &= \mathbf{V}_b \mathbf{W}^o \end{aligned}$$

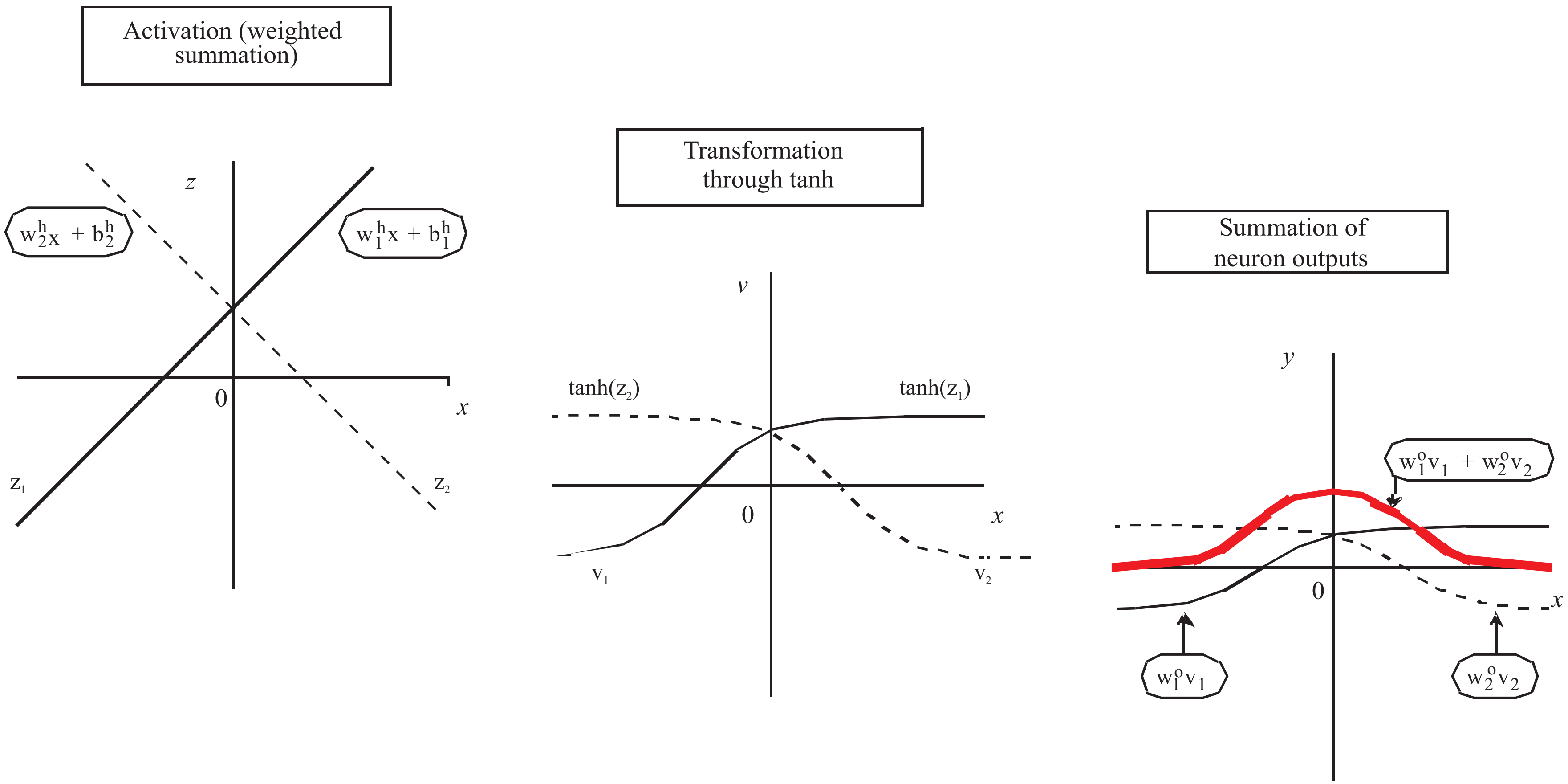
with $\mathbf{X}_b = [\mathbf{X}\ \mathbf{1}]$ and $\mathbf{V}_b = [\mathbf{V}\ \mathbf{1}]$.

Compact formula:

$$\mathbf{Y} = [\sigma([\mathbf{X}\ \mathbf{1}]\mathbf{W}^h)\ \mathbf{1}]\mathbf{W}^o$$

Function approximation with neural nets

$$y = w_1^o \tanh\left(w_1^h x + b_1^h\right) + w_2^o \tanh\left(w_2^h x + b_2^h\right)$$



Approximation properties of neural nets

[Cybenko, 1989]: A feedforward neural net with at least one hidden layer can approximate any continuous nonlinear function $\mathbb{R}^p \rightarrow \mathbb{R}^n$ arbitrarily well, provided that sufficient number of hidden neurons are available (not constructive).

Approximation properties of neural nets

[Barron, 1993]: A feedforward neural net with one hidden layer with sigmoidal activation functions can achieve an integrated squared error of the order

$$J = \mathcal{O}\left(\frac{1}{h}\right)$$

independently of the dimension of the input space p**, where** h **denotes the number of hidden neurons.**

Approximation properties of neural nets

[Barron, 1993]: A feedforward neural net with one hidden layer with sigmoidal activation functions can achieve an integrated squared error of the order

$$J = \mathcal{O}\left(\frac{1}{h}\right)$$

independently of the dimension of the input space p**, where** h **denotes the number of hidden neurons.**

For a basis function expansion (polynomial, trigonometric expansion, singleton fuzzy model, etc.) with h **terms, in which only the parameters of the linear combination are adjusted**

$$J = \mathcal{O}\left(\frac{1}{h^{2/p}}\right)$$

Approximation properties: example

1) $p = 2$ **(function of two variables):**

$$\textbf{polynomial} \; J = \mathcal{O}\left(\frac{1}{h^{2/2}}\right) = \mathcal{O}\left(\frac{1}{h}\right)$$

$$\textbf{neural net} \quad J = \mathcal{O}\left(\frac{1}{h}\right)$$

$\longrightarrow$ **no difference**

Approximation properties: example

2) $p = 10$ **(function of ten variables) and** $h = 21$**:**

$$\textbf{polynomial} \quad J = \mathcal{O}(\frac{1}{21^{2/10}}) = 0.54$$

$$\textbf{neural net} \quad J = \mathcal{O}(\frac{1}{21}) = 0.048$$

Approximation properties: example

2) $p = 10$ **(function of ten variables) and** $h = 21$**:**

$$\textbf{polynomial} \quad J = \mathcal{O}(\frac{1}{21^{2/10}}) = 0.54$$

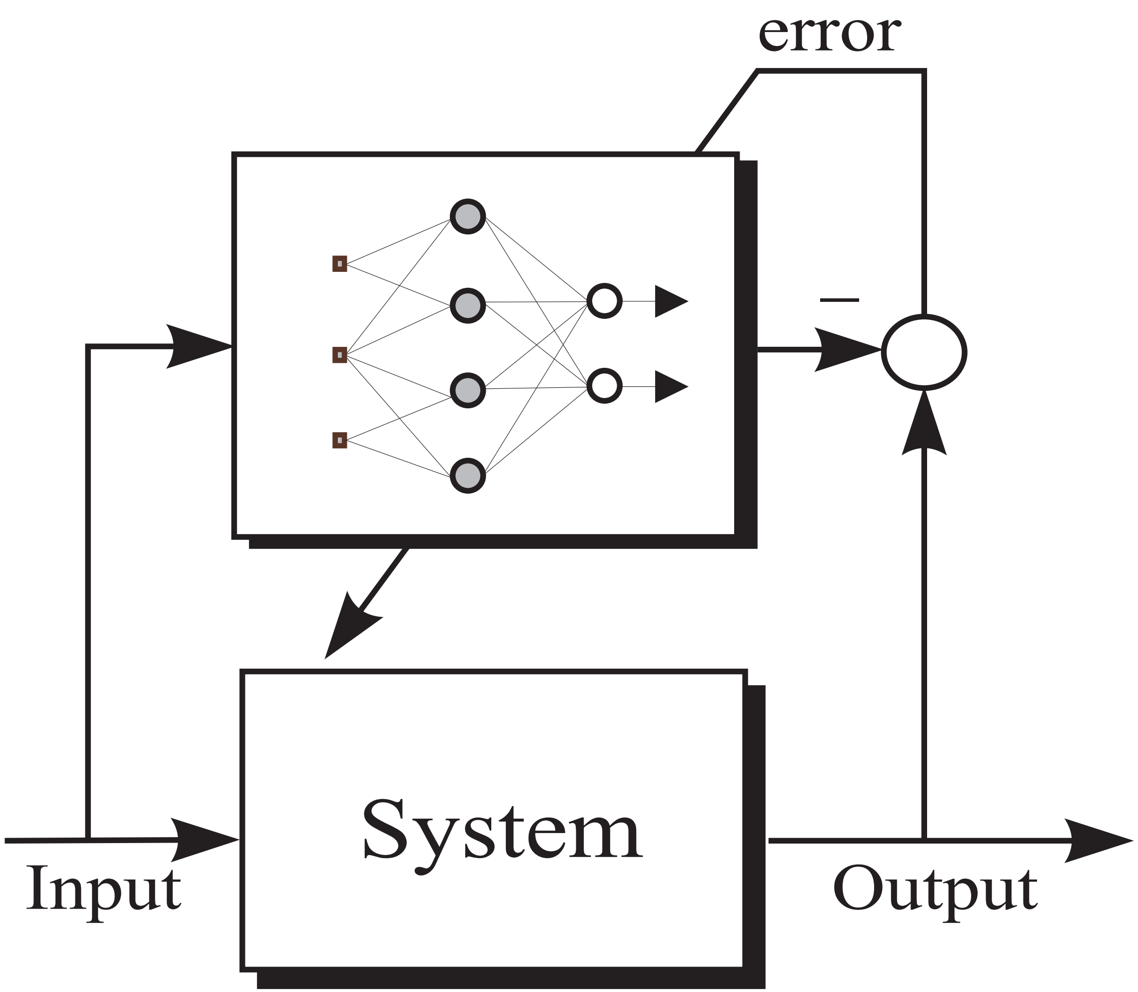
$$\textbf{neural net} \quad J = \mathcal{O}(\frac{1}{21}) = 0.048$$

To achieve the same accuracy:

$$\mathcal{O}(\frac{1}{h_n}) = \mathcal{O}(\frac{1}{h_b})$$

$$h_n = h_b^{2/p} \quad \Rightarrow \quad h_b = \sqrt{h_n^p} = \sqrt{21^{10}} \approx 4 \cdot 10^6$$

Supervised learning



Learning in feedforward nets

1. **Feedforward computation.** From the inputs proceed through the hidden layers to the output.

$$\mathbf{Z} = \mathbf{X}_b \mathbf{W}^h, \quad \mathbf{X}_b = [\mathbf{X}\ \mathbf{1}]$$

$$\mathbf{V} = \sigma(\mathbf{Z})$$

$$\mathbf{Y} = \mathbf{V}_b \mathbf{W}^o, \quad \mathbf{V}_b = [\mathbf{V}\ \mathbf{1}]$$

Learning in feedforward nets

2. **Weight adaptation.** Compare the net output with the desired output:

$$\mathbf{E} = \mathbf{D} - \mathbf{Y}$$

Adjust the weights such that the following cost function is minimized:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{N} \sum_{j=1}^{l} e_{kj}^2 = \text{trace}\left(\mathbf{E}\mathbf{E}^T\right)$$

$$\mathbf{w} = \left[\mathbf{W}^h \; \mathbf{W}^o\right]$$

Optimization methods

Training of neural nets is a *nonlinear* optimization problem.

Methods:

- Error backpropagation (first-order gradient).
- Newton methods (second-order gradient).
- Levenberg-Marquardt (second-order gradient).
- Conjugate gradients.
- Variable projection.
- ... and many others

First-order gradient methods

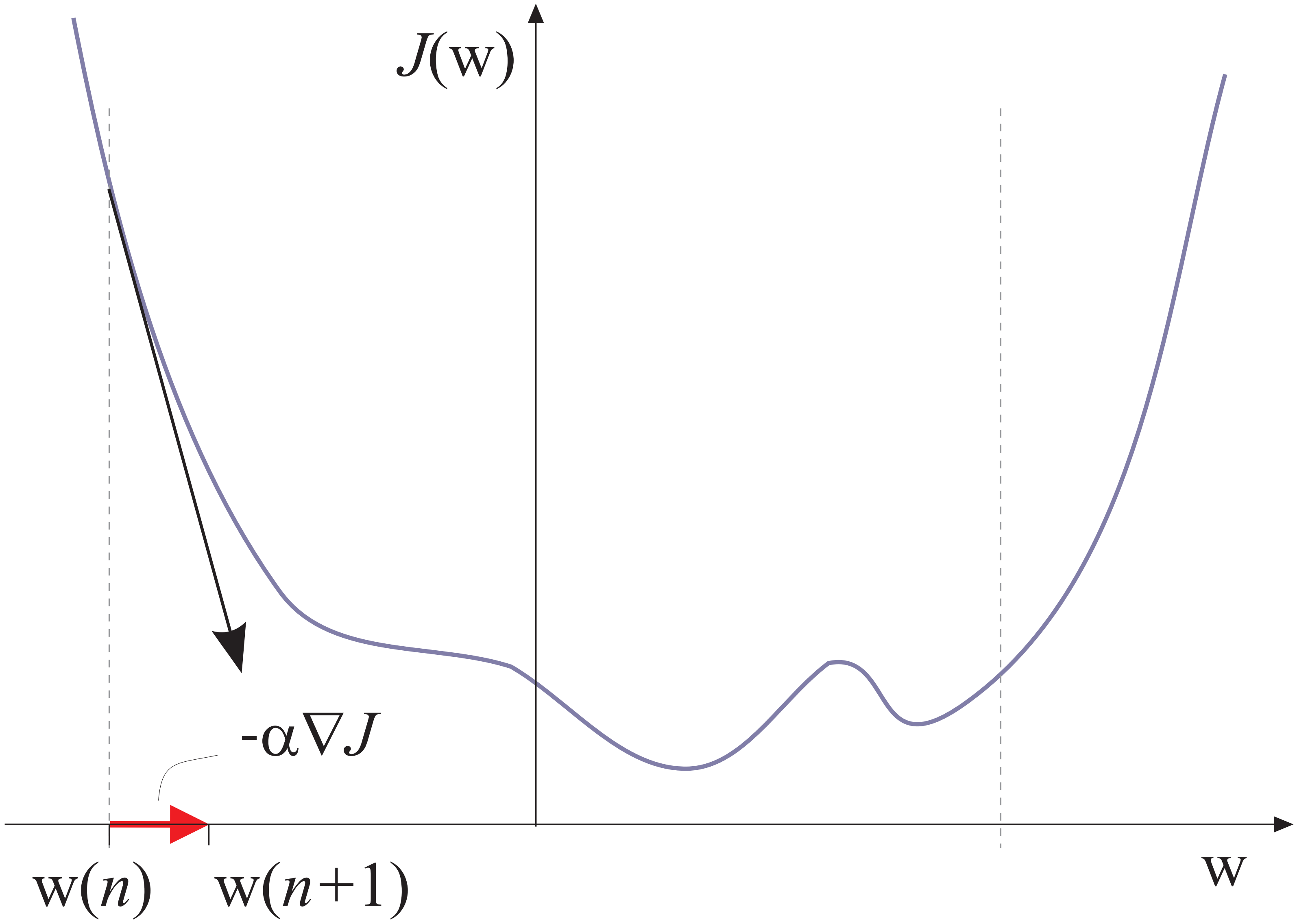
Update rule for the weights:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha_n \nabla J(\mathbf{w}_n)$$

with the Jacobian $\nabla J(\mathbf{w}_n)$

$$\nabla J(\mathbf{w}) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_M} \right)^T$$

First-order gradient methods



Second-order gradient methods

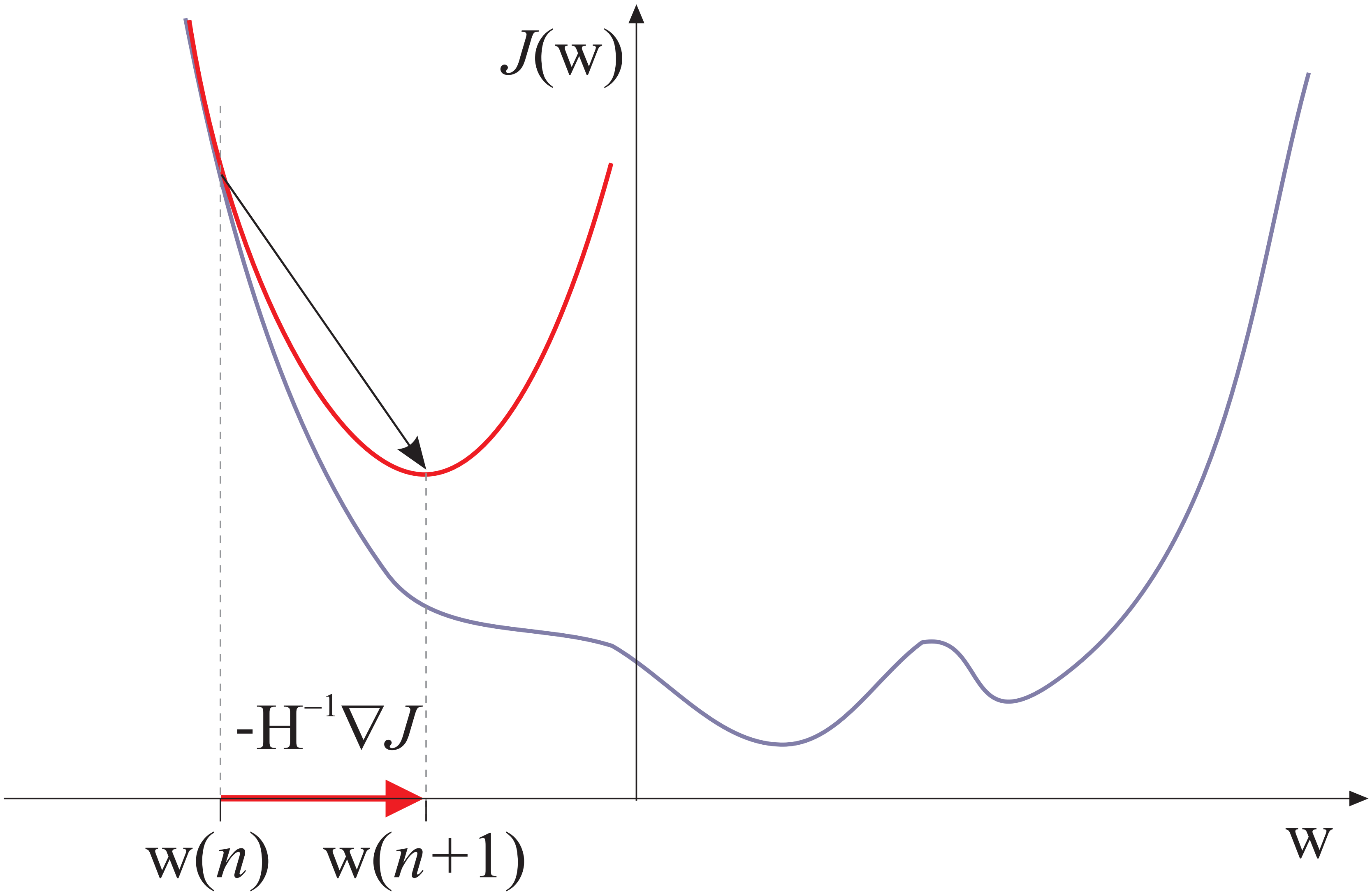
$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + \nabla J(\mathbf{w}_0)^T(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T\mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0)$$

where $\mathbf{H}(\mathbf{w}_0)$ is the Hessian in $\mathbf{w}_0$.

Update rule for the weights:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mathbf{H}^{-1}(\mathbf{w}_n)\nabla J(\mathbf{w}_n)$$

Second-order gradient methods



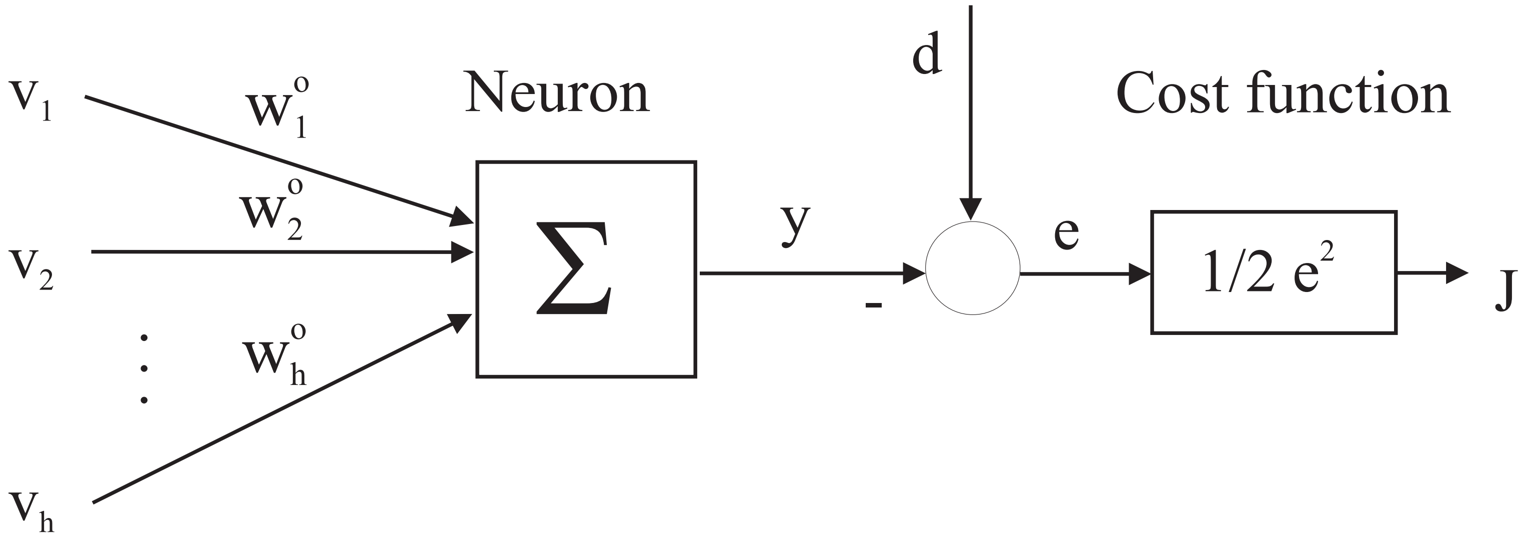
Error backpropagation

- First-order gradient method
  $\longrightarrow$ not so effective but instructive for the principle.

- Main idea:
  - compute errors at the outputs,
  - adjust output weights,
  - propagate error backwards through the net and adjust hidden-layer weights.

- Process the data set pattern by pattern
  (suitable for both on-line and off-line learning).

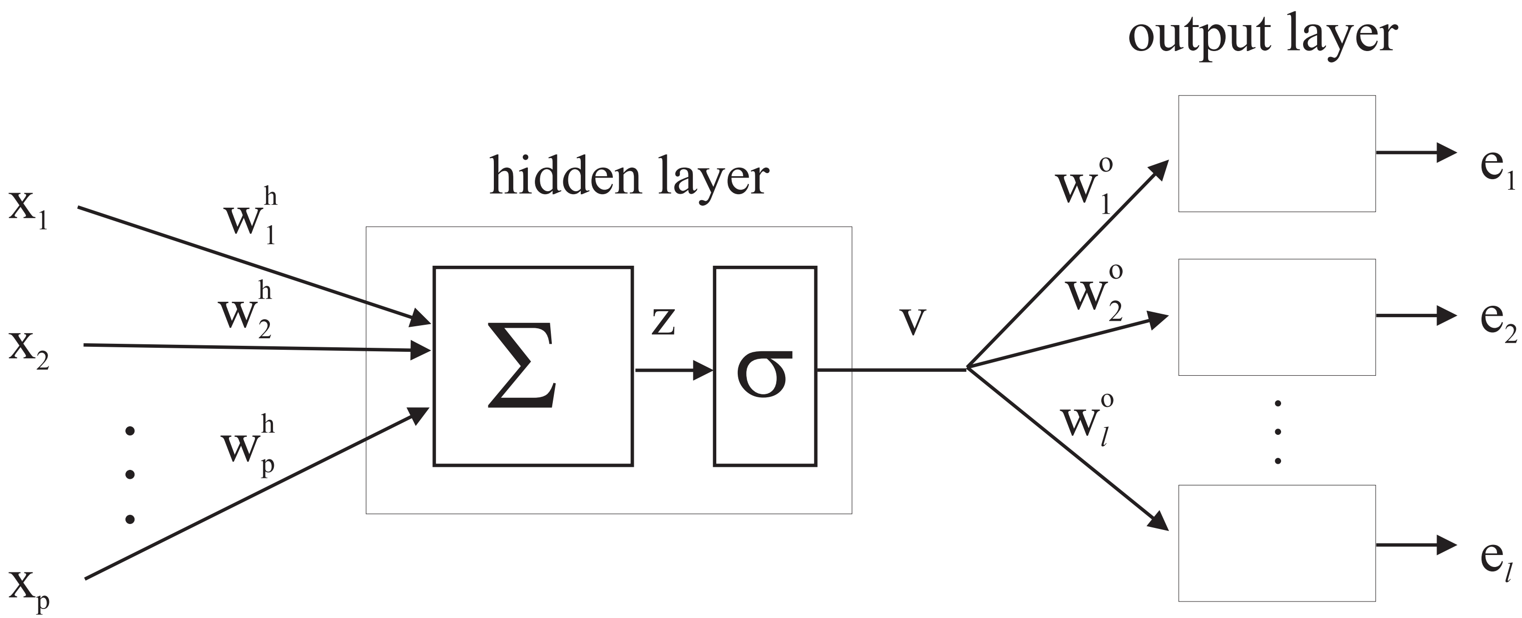
Output-layer weights



$$J = \frac{1}{2}\sum_l e_l^2, \quad e_l = d_l - y_l, \quad y_l = \sum_j w_j^o v_j$$

$$\frac{\partial J}{\partial w_{jl}^o} = \frac{\partial J}{\partial e_l} \cdot \frac{\partial e_l}{\partial y_l} \cdot \frac{\partial y_l}{\partial w_{jl}^o} = -v_j e_l$$

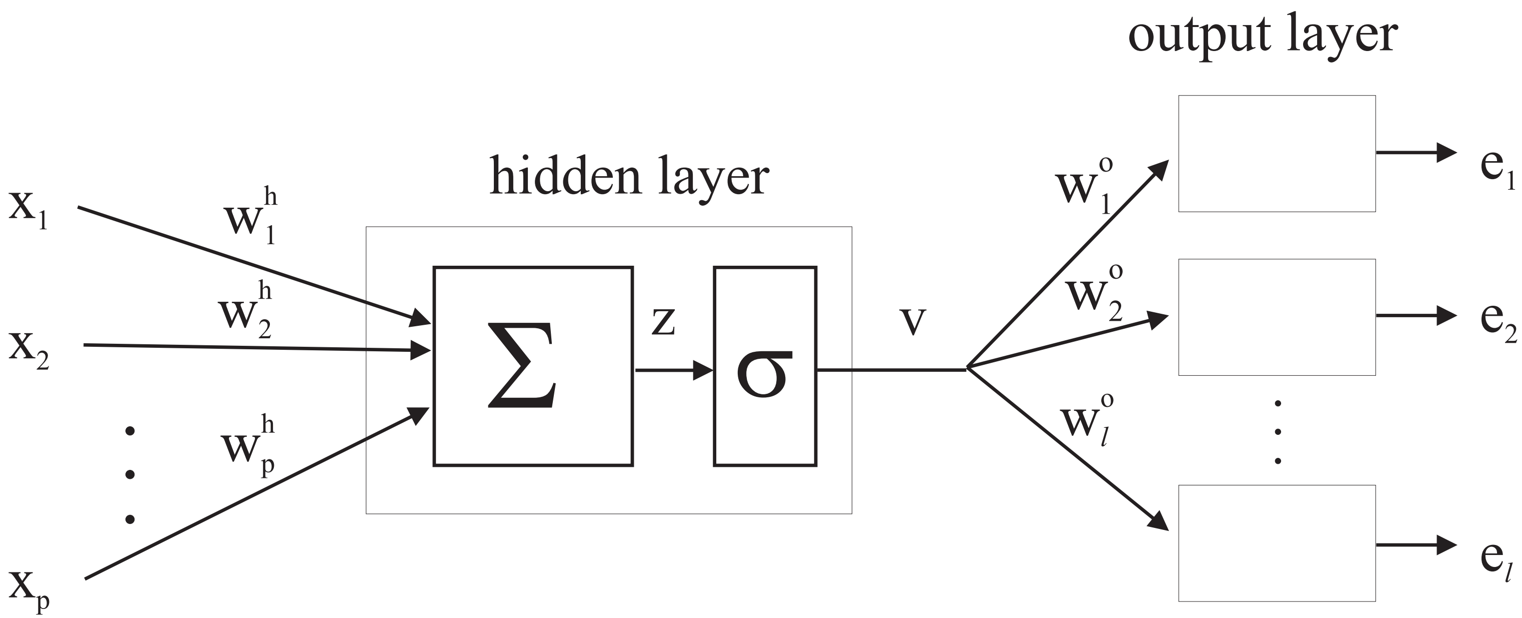
Hidden-layer weights



$$\frac{\partial J}{\partial w^h_{ij}} = \frac{\partial J}{\partial v_j} \cdot \frac{\partial v_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w^h_{ij}}$$

$$\frac{\partial J}{\partial v_j} = \sum_l -e_l w^o_{jl}, \qquad \frac{\partial v_j}{\partial z_j} = \sigma'_j(z_j), \qquad \frac{\partial z_j}{\partial w^h_{ij}} = x_i$$

Hidden-layer weights



$$\frac{\partial J}{\partial w^h_{ij}} = \frac{\partial J}{\partial v_j} \cdot \frac{\partial v_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w^h_{ij}} = -x_i \cdot \sigma'_j(z_j) \cdot \sum_l e_l w^o_{jl}$$

$$\frac{\partial J}{\partial v_j} = \sum_l -e_l w^o_{jl}, \qquad \frac{\partial v_j}{\partial z_j} = \sigma'_j(z_j), \qquad \frac{\partial z_j}{\partial w^h_{ij}} = x_i$$

Error backpropagation: summary

1. Initialize the weights (at random).

2. Present inputs and desired outputs, calculate actual outputs and errors.

3. Compute gradients and update weights.

   for the output layer:

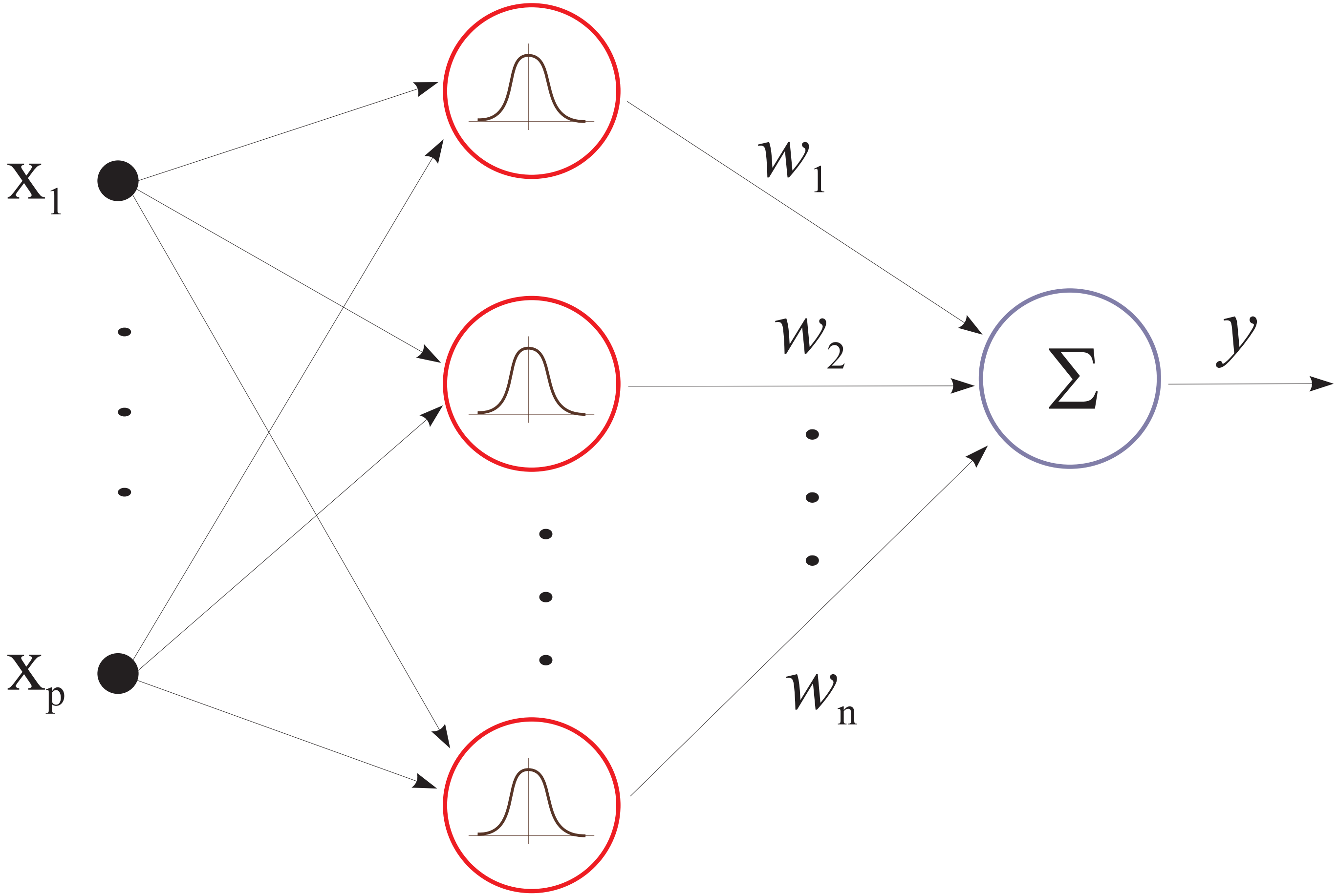
$$w_{jl}^o := w_{jl}^o + \alpha v_j e_l$$

   and for the hidden layer(s):

$$w_{ij}^h := w_{ij}^h + \alpha x_i \cdot \sigma_j'(z_j) \cdot \sum_l e_l w_{jl}^o$$

4. Repeat by going to Step 2.

Radial basis function network



Radial basis function network

Input–output mapping:

$$y = \sum_{i=1}^{n} w_i e^{-\frac{(\mathbf{x}-\mathbf{c}_i)^2}{s_i^2}}$$

n, $\mathbf{c}_i$ **and** s_i **are usually fixed (determined a priori)**

w_i **estimated by least squares**

Notice similarity with the singleton fuzzy model.

Least-squares estimate of weights

Given A_{ij} and a set of input–output data:

$$\{\langle \mathbf{x}_k, y_k \rangle \mid k = 1, 2, \ldots, N\}$$

1. Compute the output of the neurons:

$$z_{ki} = e^{-\frac{(\mathbf{x}_k - \mathbf{c}_i)^2}{s_i^2}}, \quad k = 1, 2, \ldots, N, \quad i = 1, 2, \ldots, n$$

The output is linear in the weights:

$$\mathbf{y} = \mathbf{Z}\mathbf{w}$$

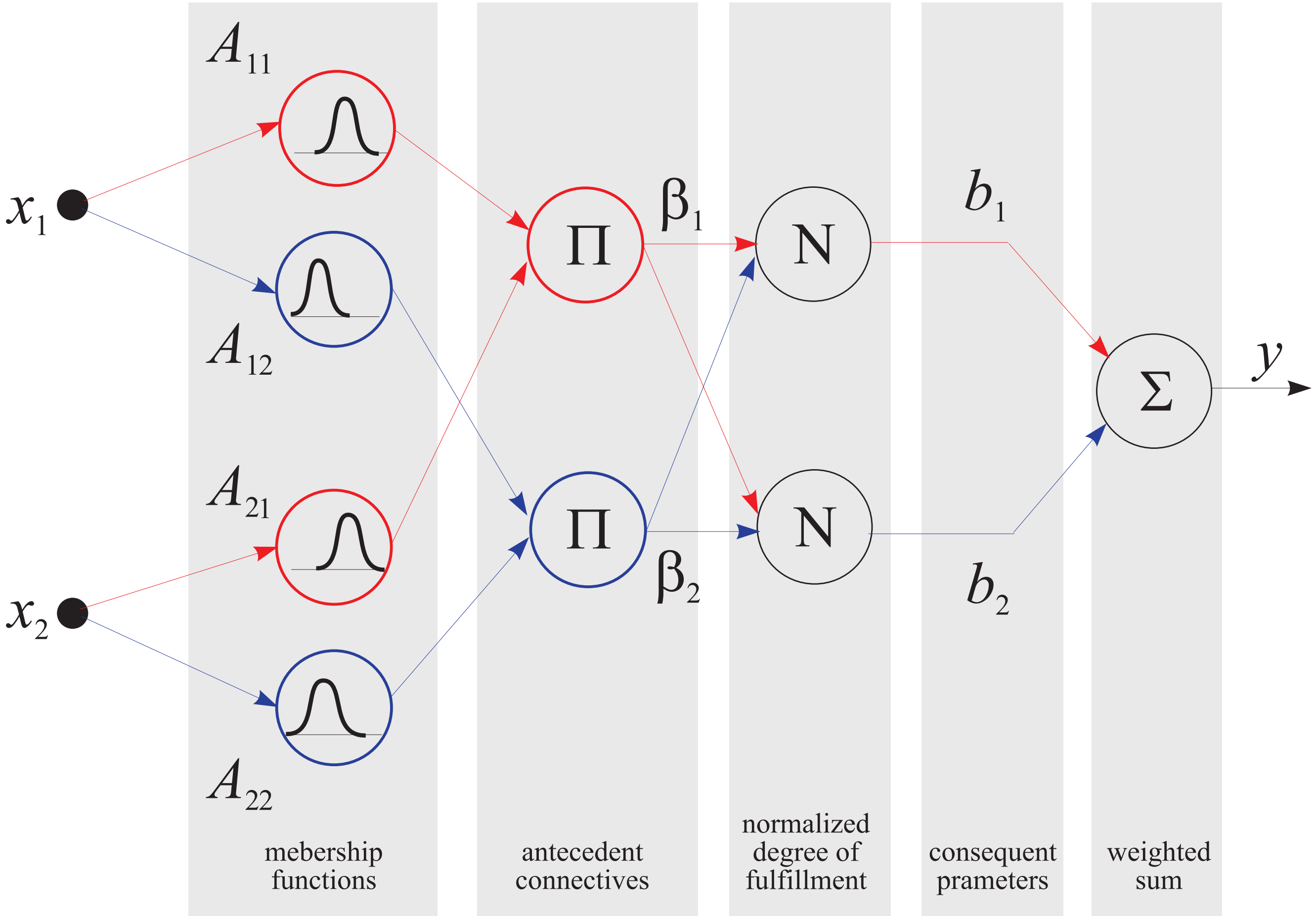
2. Least-squares estimate:

$$\mathbf{w} = \left[\mathbf{Z}^T \mathbf{Z}\right]^{-1} \mathbf{Z}^T \mathbf{y}$$

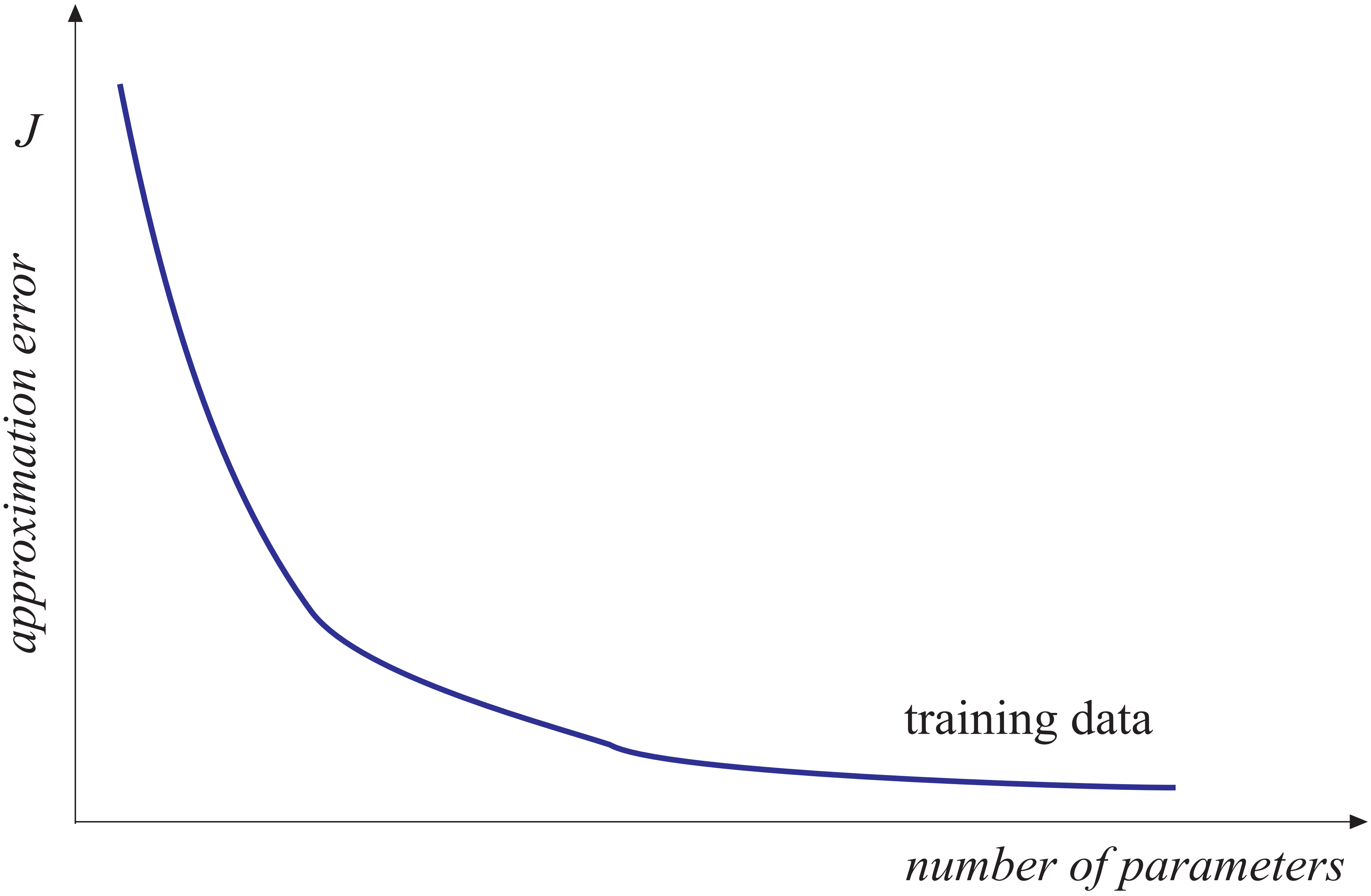
Neuro–fuzzy learning

If x_1 **is** A_{11} **and** x_2 **is** A_{21} **then** $y = b_1$

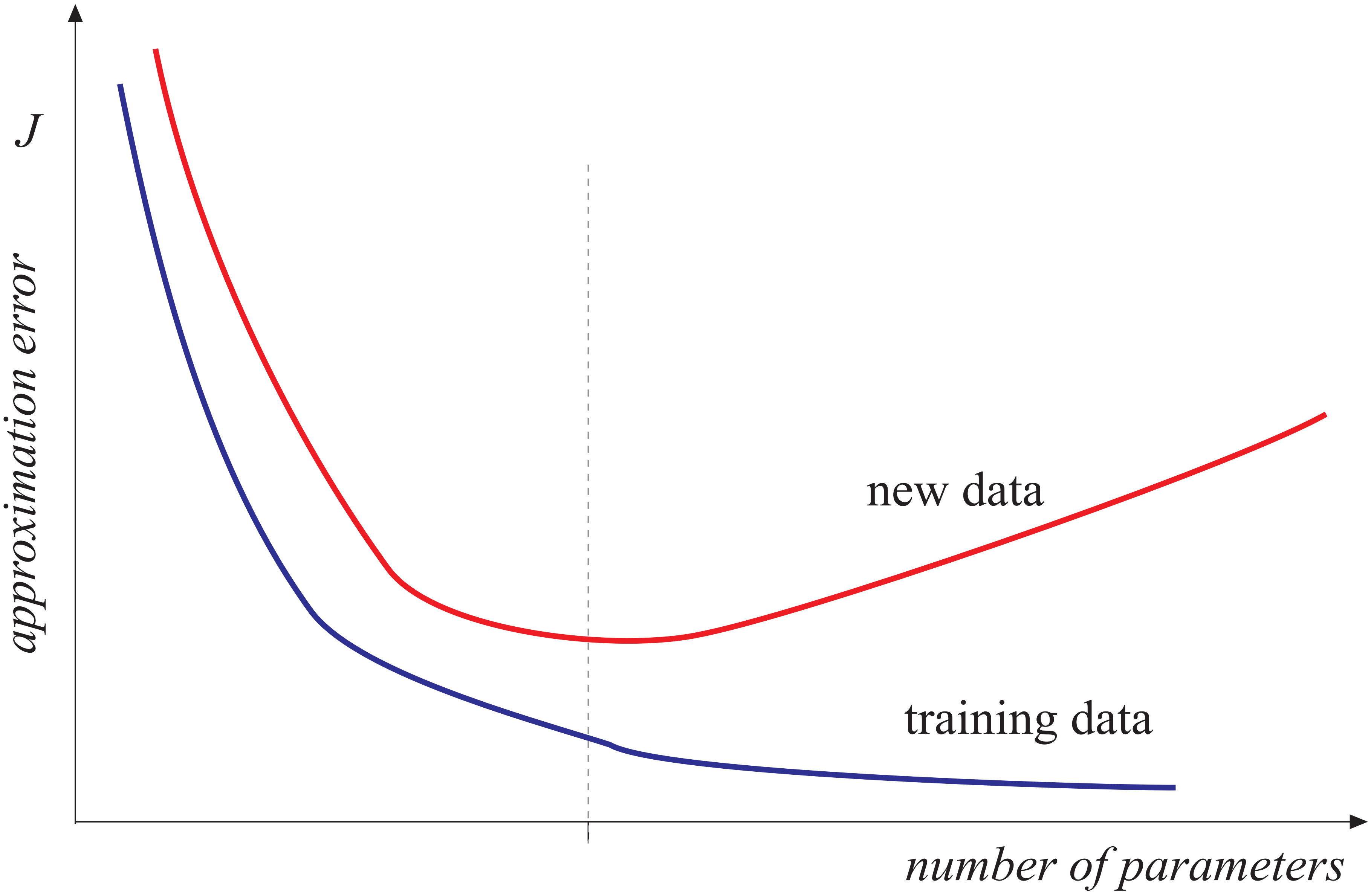
If x_1 **is** A_{12} **and** x_2 **is** A_{22} **then** $y = b_2$



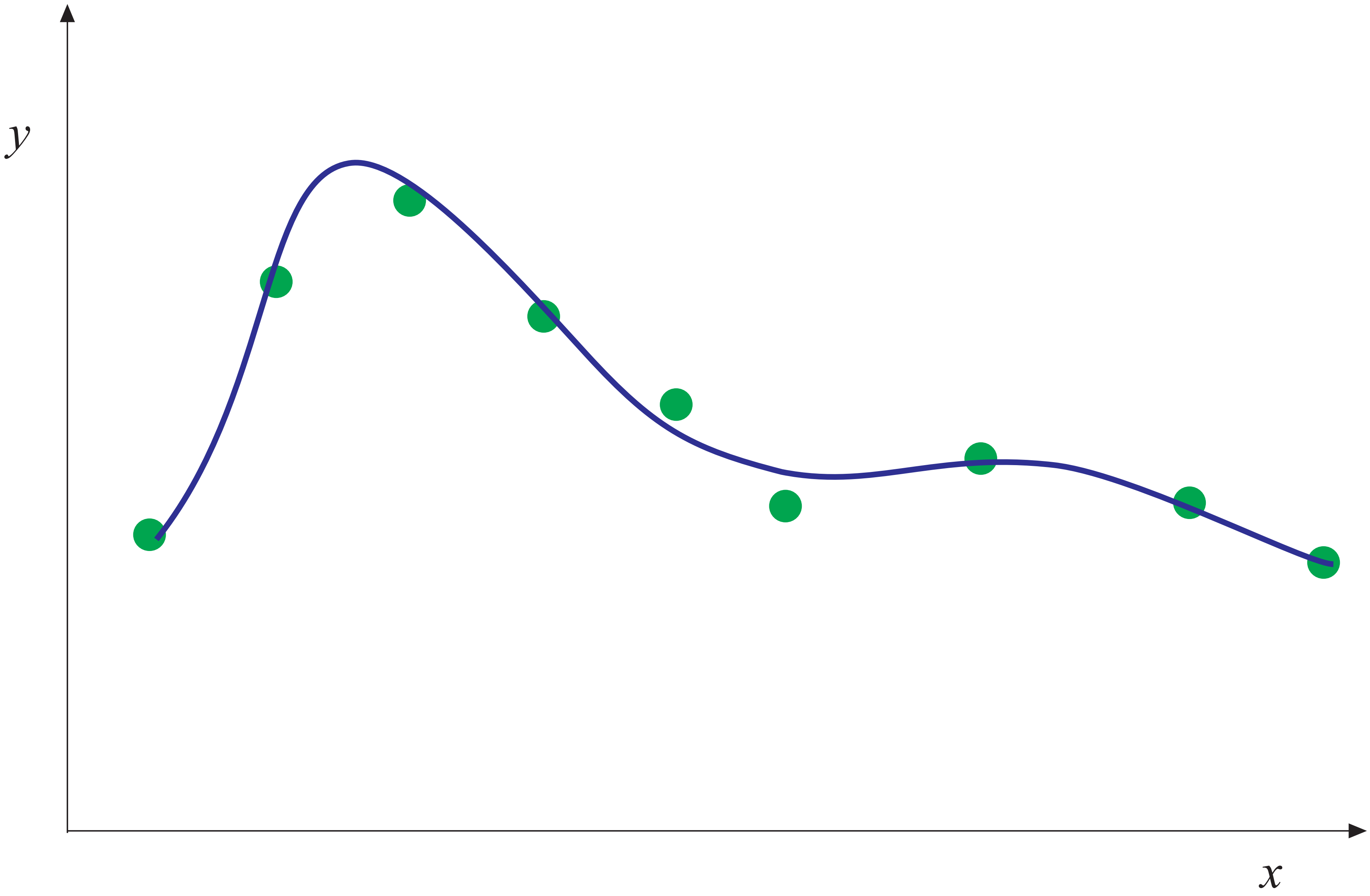
Approximation error vs. number of parameters



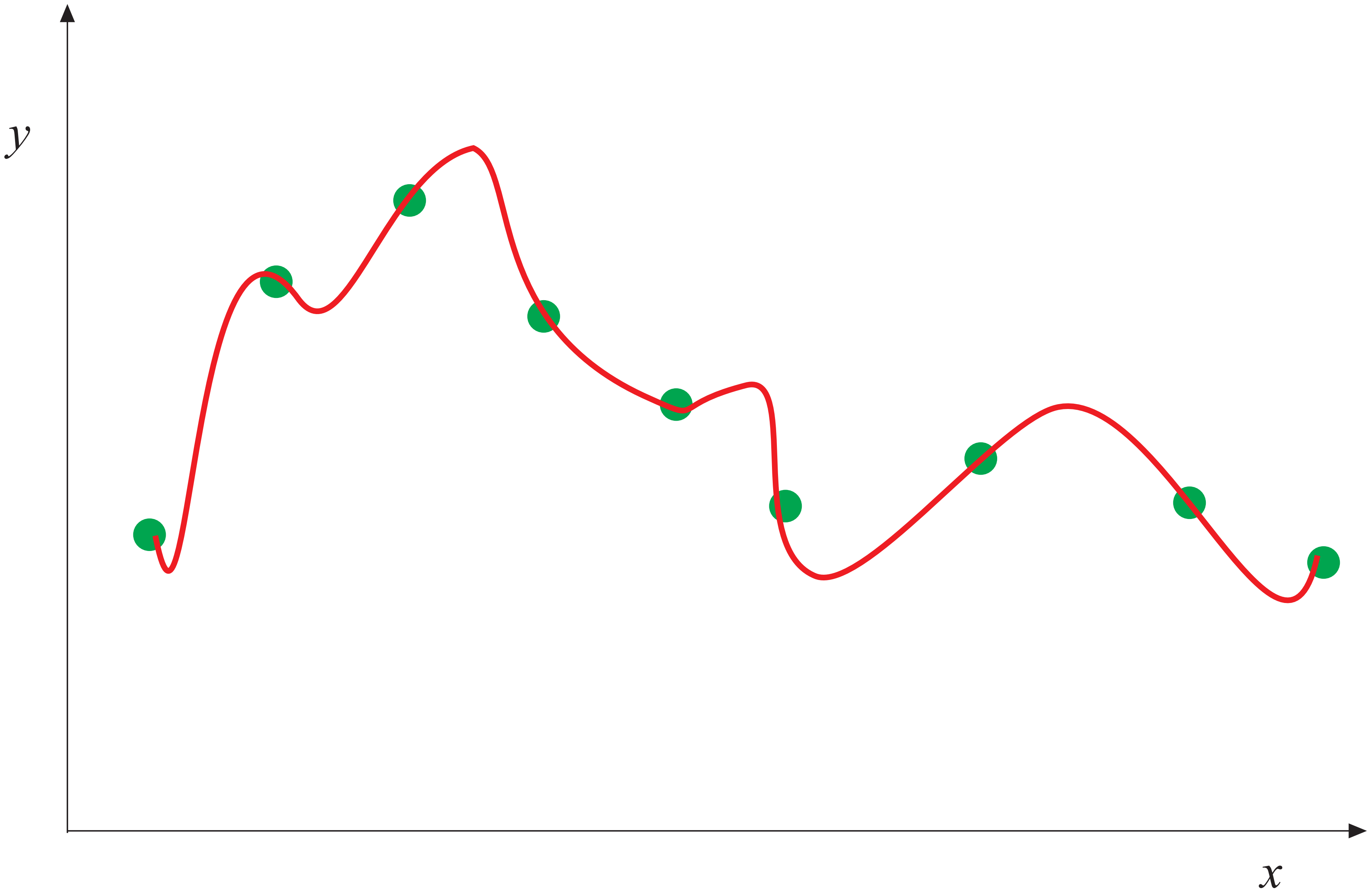
Approximation error vs. number of parameters



Good fit



Overfitting



Validation

System: $y = f(\mathbf{x})$ **or** $y(k+1) = f(\mathbf{x}(k), \mathbf{u}(k))$

Model: $\hat{y} = F(\mathbf{x}; \theta)$ **or** $\hat{y}(k+1) = F(\mathbf{x}(k), \mathbf{u}(k); \theta)$

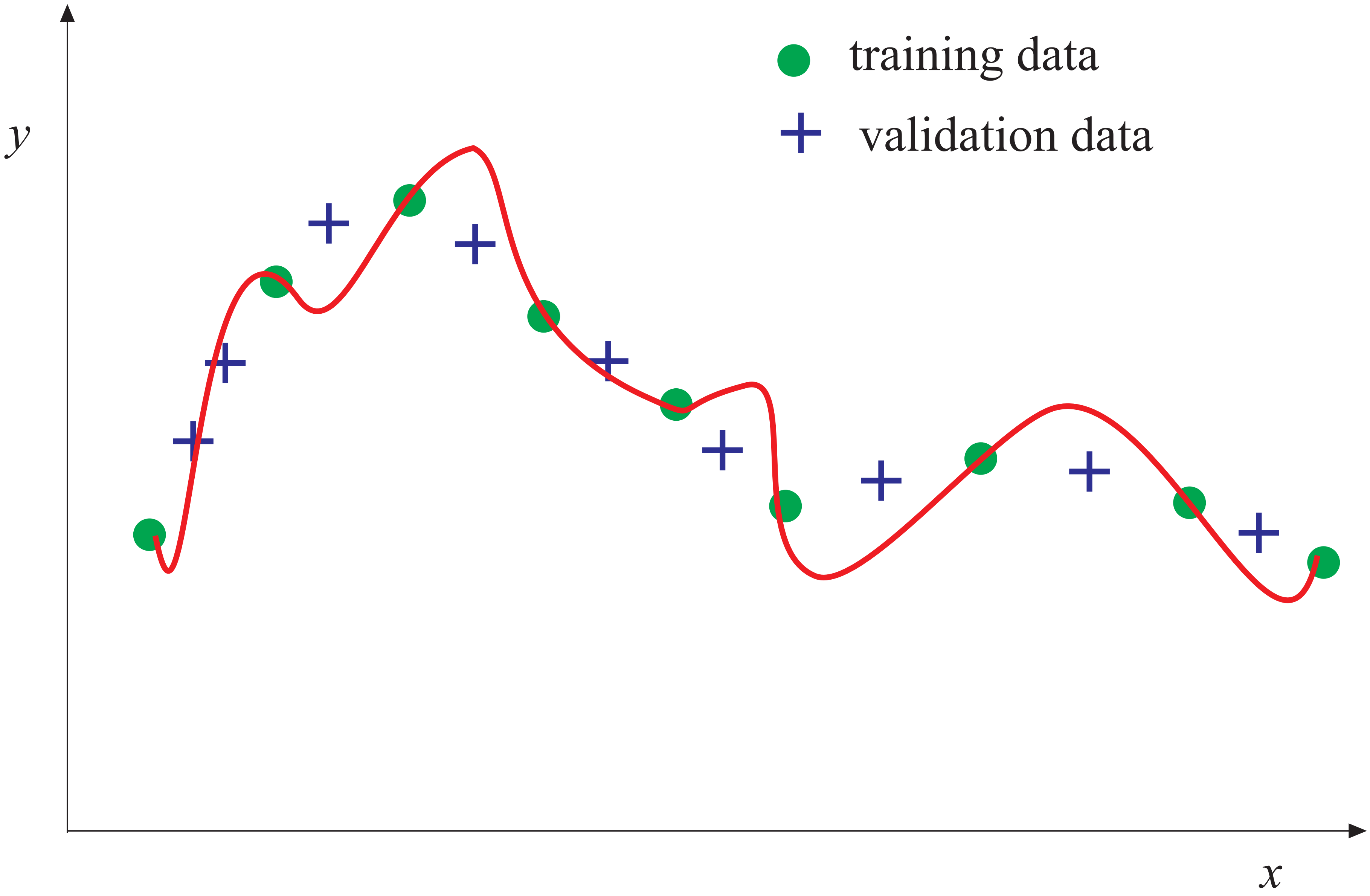
True criterion:

$$I = \int_X \|f(\mathbf{x}) - F(\mathbf{x})\| \mathbf{dx} \qquad (1)$$

Usually cannot be computed as $f(\mathbf{x})$ is not available, use available data to numerically compute (1)

- **use a validation set**
- **cross-validation (randomize)**

Validation Data Set



Cross-Validation

- **Regularity criterion (for two data sets):**

$$RC = \frac{1}{2}\left[\frac{1}{N_A}\sum_{i=1}^{N_A}(y^A(i) - \hat{y}^A_B(i))^2 + \frac{1}{N_B}\sum_{i=1}^{N_B}(y^B(i) - \hat{y}^B_A(i))^2\right]$$

- **leave-one-out method**

- v**-fold cross-validation**

Some Common Criteria

- **Mean squared error (root mean square error):**

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y(i) - \hat{y}(i))^2$$

- **Variance accounted for (VAF):**

$$\mathbf{VAF} = 100\% \cdot \left[1 - \frac{\mathbf{var}(y - \hat{y})}{\mathbf{var}(y)}\right]$$

- **Check the correlation of the residual $y - \hat{y}$ to u, y and itself.**

Applications of neural nets

- Black-box modeling of systems from input-output data.
- Reconstruction (estimation) – soft sensors.
- Classification.
- Neurocomputing.
- Neurocontrol.