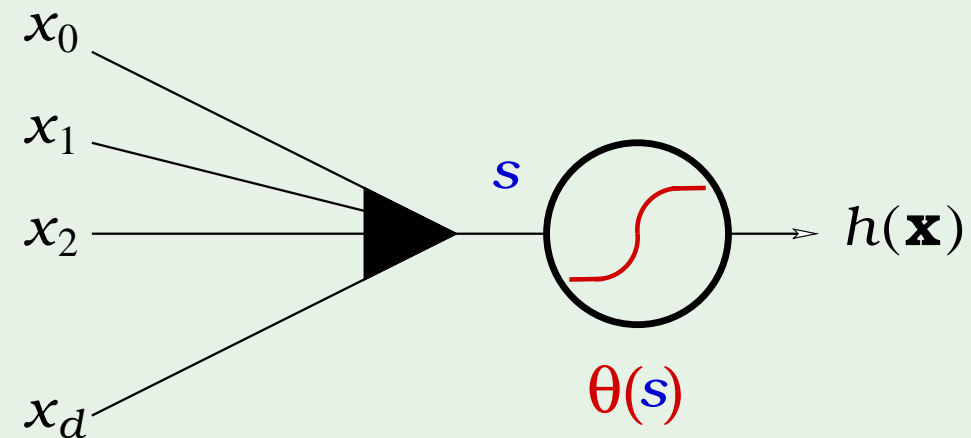# Review of Lecture 9
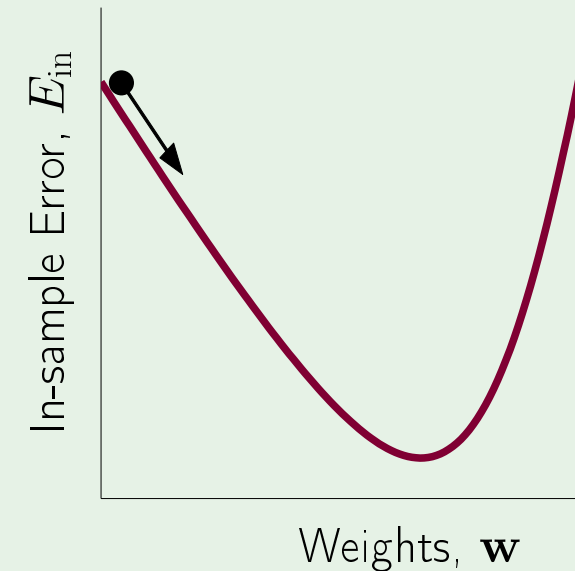
- Logistic regression

- Likelihood measure

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)$$

- Gradient descent

- Initialize $\mathbf{w}(0)$

- For $t = 0, 1, 2, \cdots$ [to termination]

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \, \nabla E_{\text{in}}(\mathbf{w}(t))$$
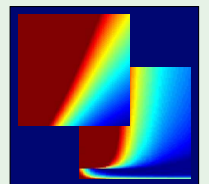
- Return final $\mathbf{w}$

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 10: Neural Networks

Outline

- Stochastic gradient descent

- Neural network model

- Backpropagation algorithm

Stochastic gradient descent

GD minimizes:

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \underbrace{\text{e}\left(h(\mathbf{x}_n), y_n\right)}_{\ln\left(1+e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)} \longleftarrow \text{in logistic regression}$$

by iterative steps along $-\nabla E_{\text{in}}$:

$$\Delta\mathbf{w} = -\eta \, \nabla E_{\text{in}}(\mathbf{w})$$

∇E_{in} is based on all examples $(\mathbf{x}_n, y_n)$

"**batch**" GD

The stochastic aspect

Pick <u>one</u> $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\big(h(\mathbf{x}_n), y_n\big)$

"Average" direction:

$$\mathbb{E}_n\left[-\nabla\mathbf{e}\big(h(\mathbf{x}_n), y_n\big)\right] = \frac{1}{N}\sum_{n=1}^{N} -\nabla\mathbf{e}\big(h(\mathbf{x}_n), y_n\big)$$

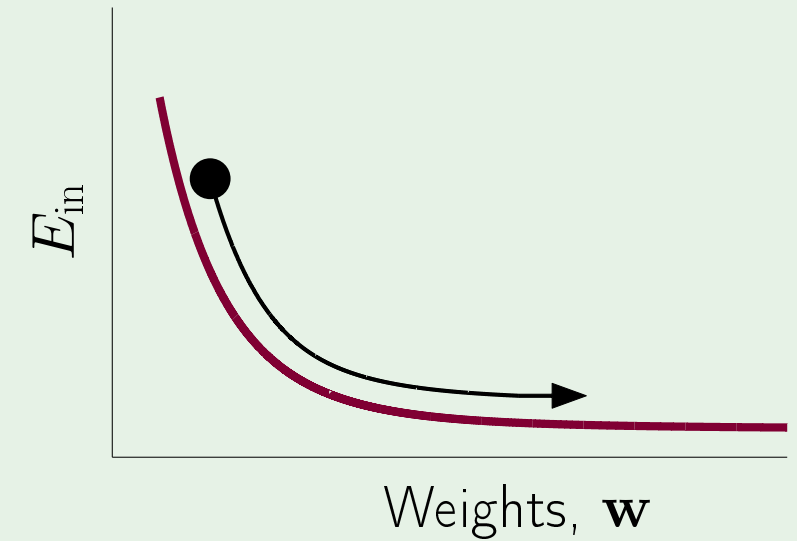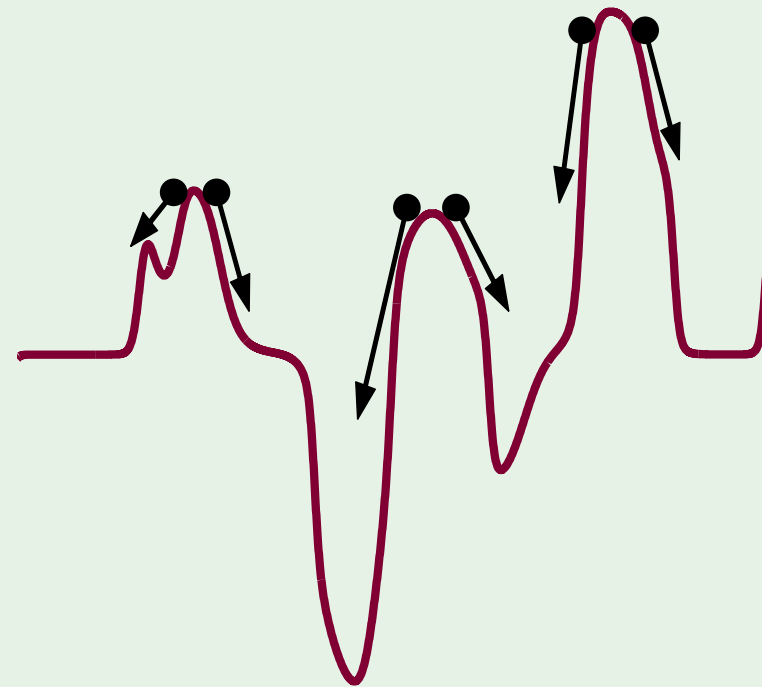$$= -\nabla\, E_{\text{in}}$$

randomized version of GD

stochastic gradient descent (SGD)

Benefits of SGD

1. cheaper computation

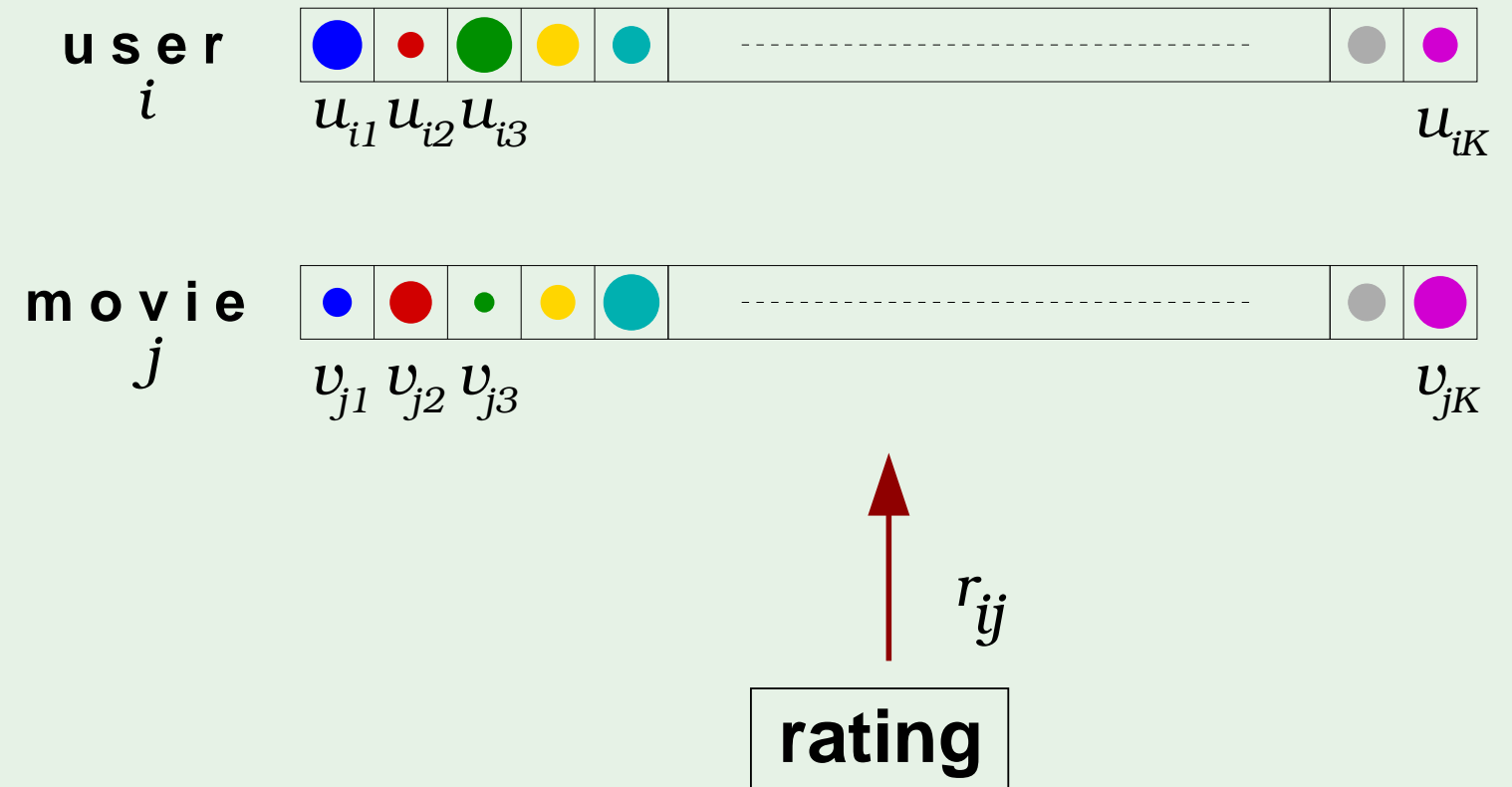2. randomization

3. simple

Rule of thumb:

$$\eta = 0.1 \quad \text{works}$$

randomization helps

E_{in}

Weights, $\mathbf{w}$

SGD in action

Remember movie ratings?

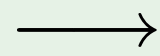$$\mathbf{e}_{ij} = \left(r_{ij} - \sum_{k=1}^{K} u_{ik} v_{jk} \right)^2$$
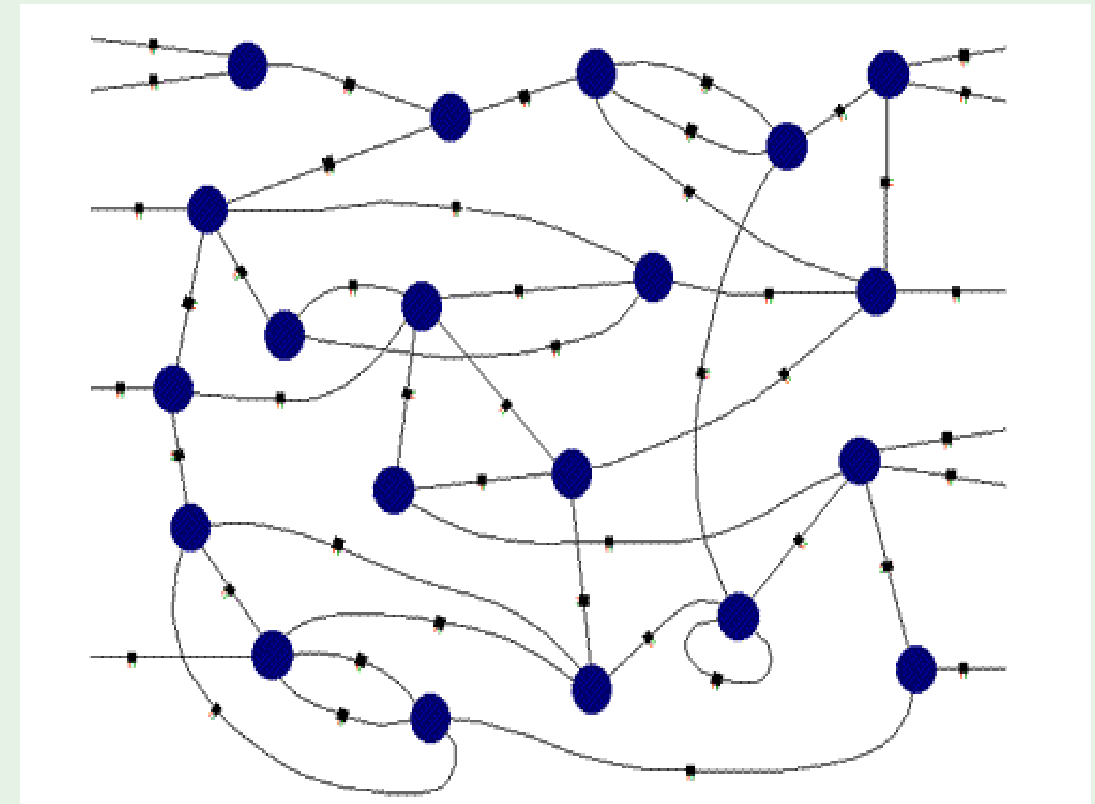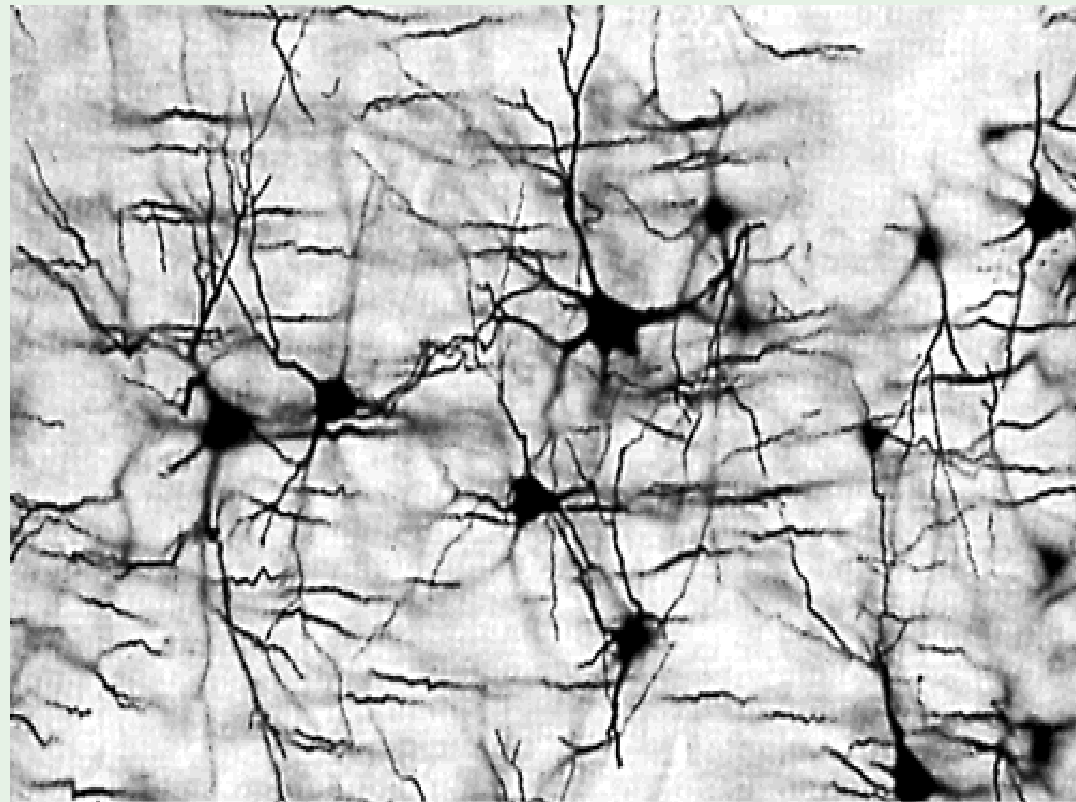
Outline

- Stochastic gradient descent

- Neural network model

- Backpropagation algorithm

Biological inspiration

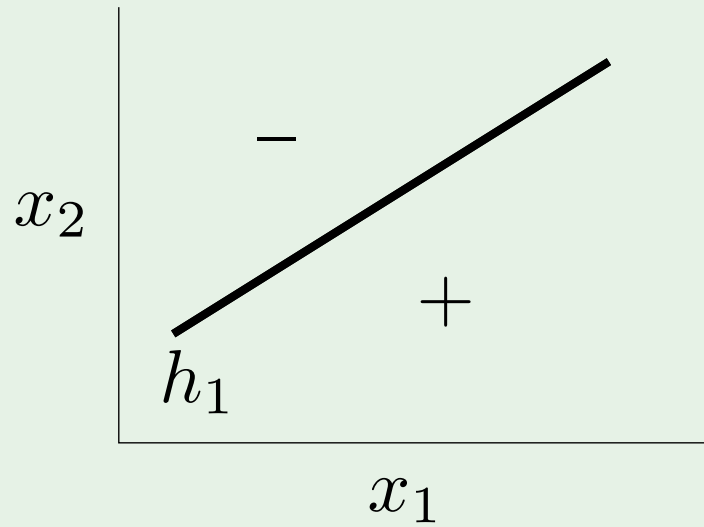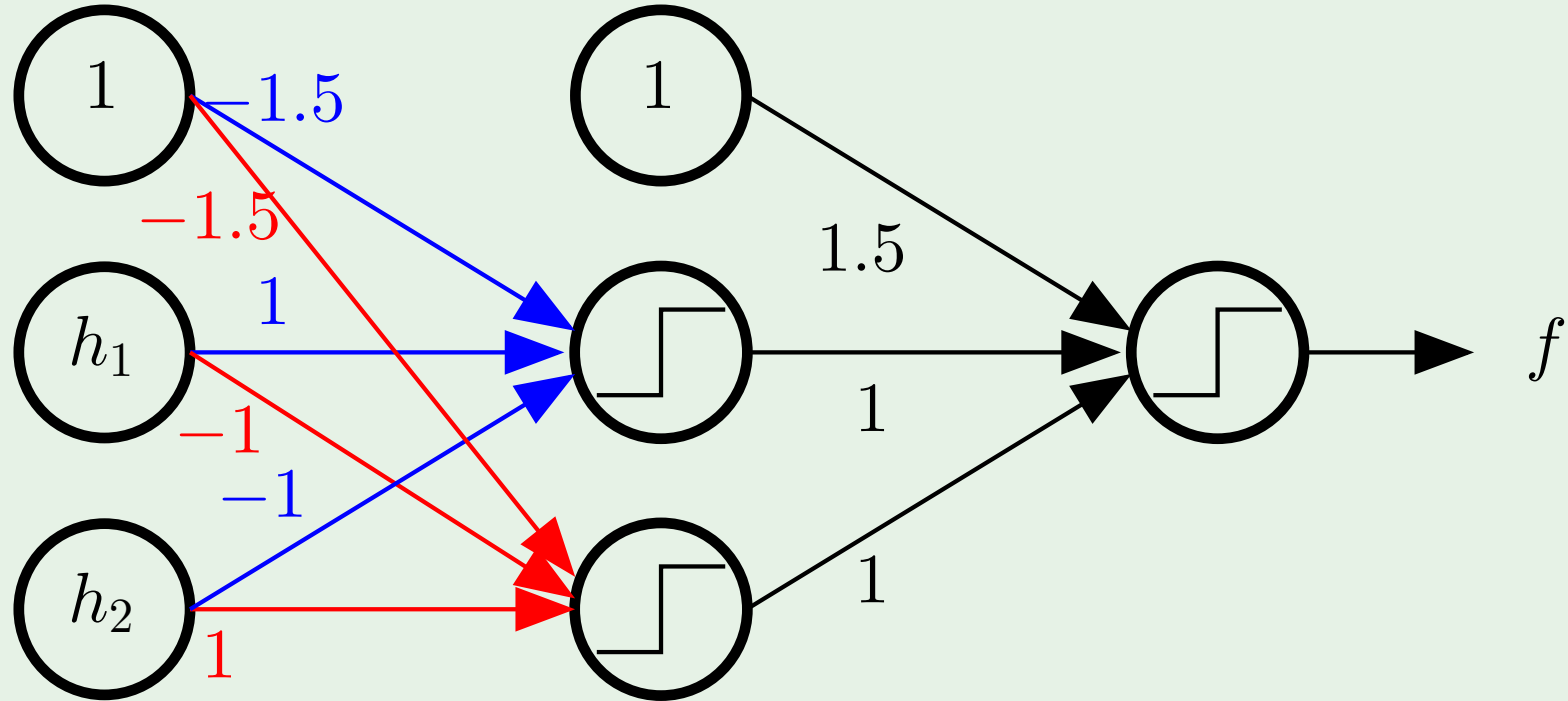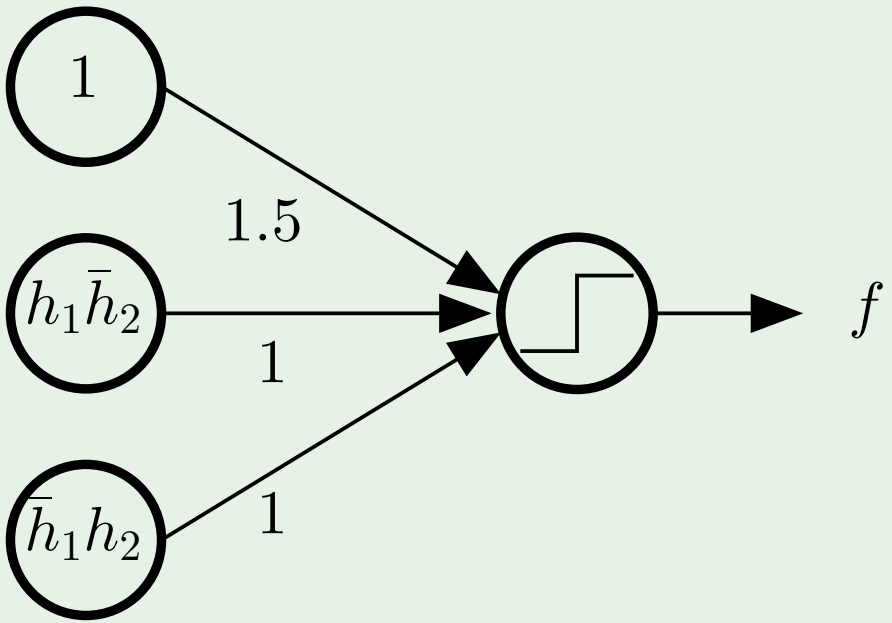biological function $\longrightarrow$ biological structure
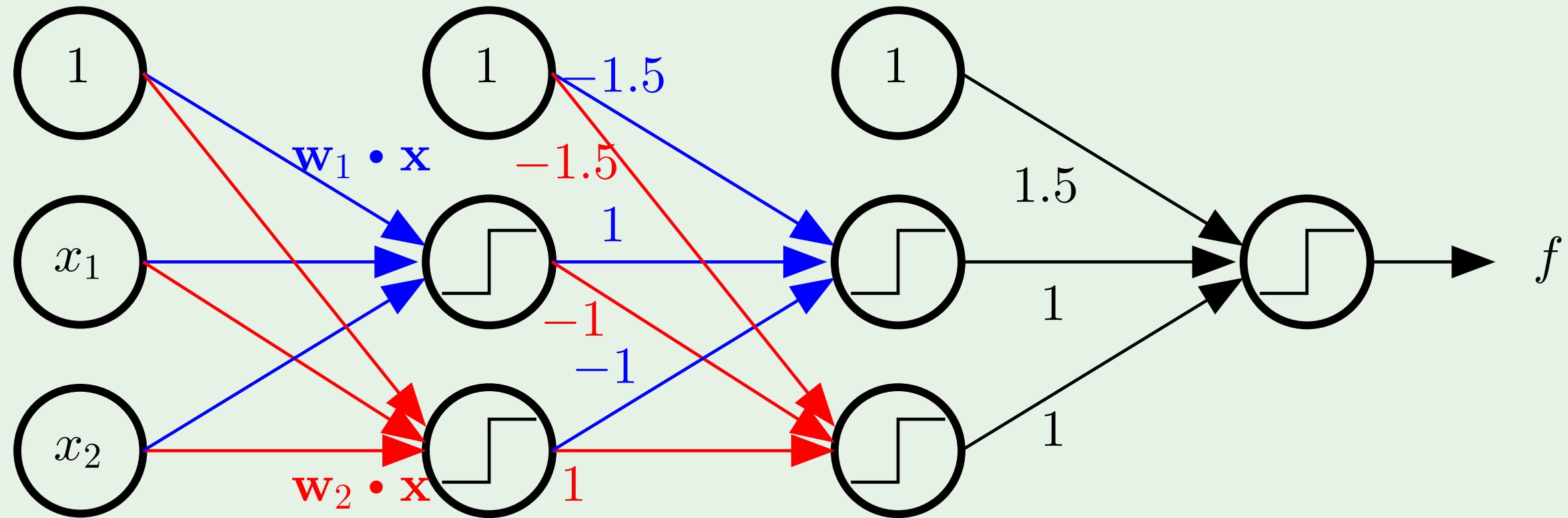
Combining perceptrons
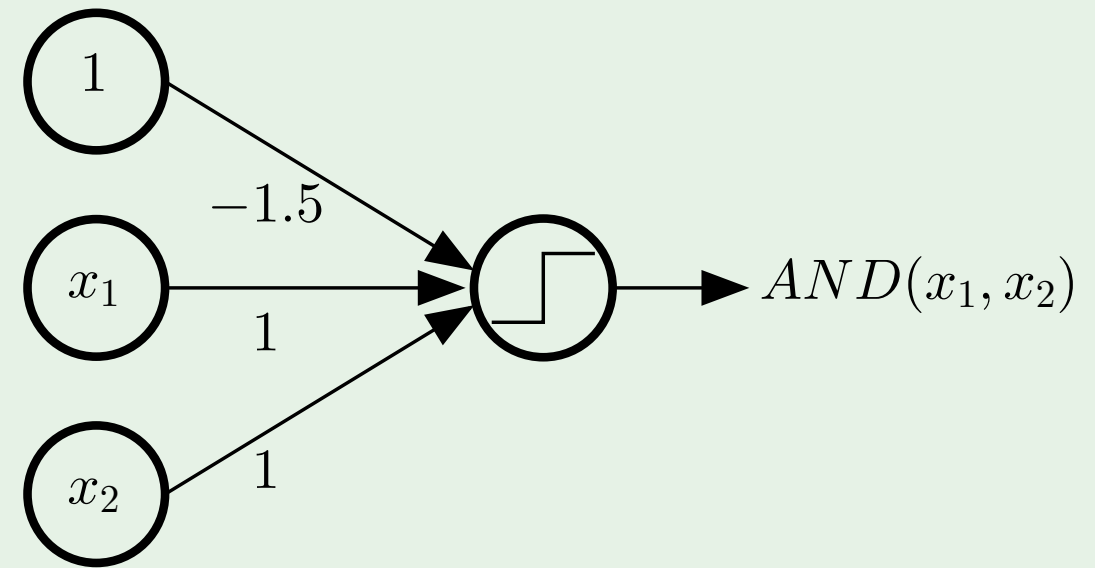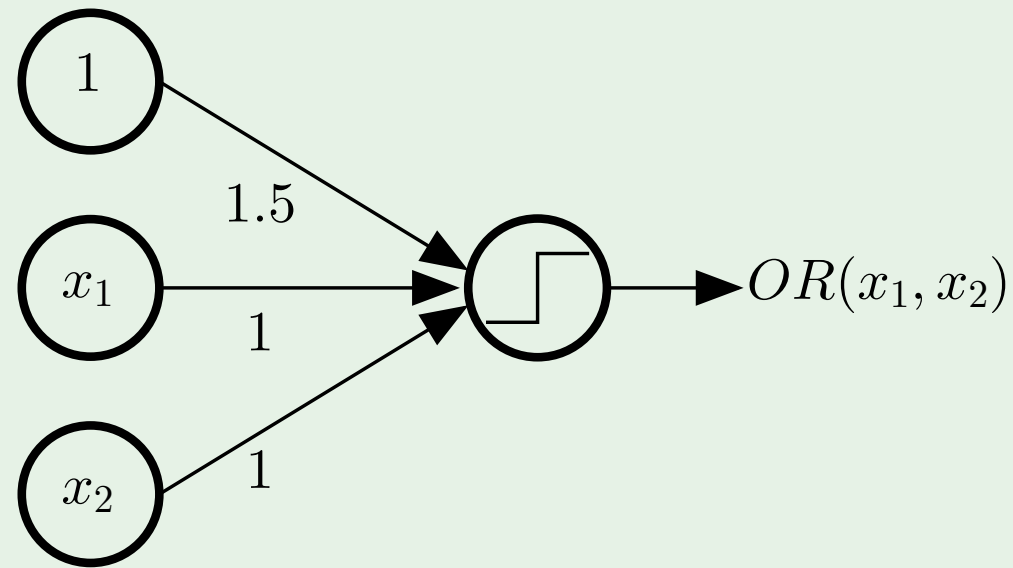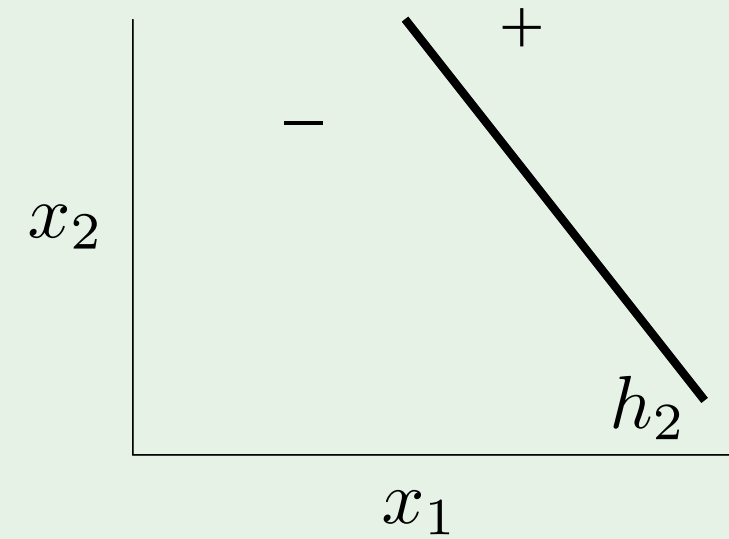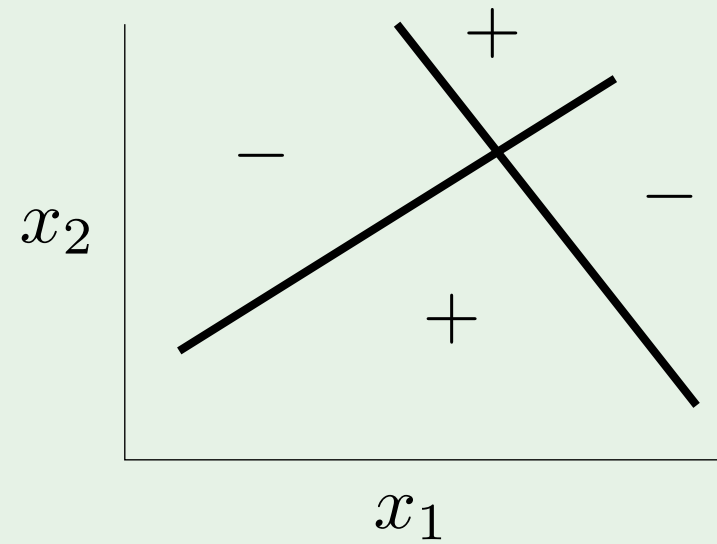
Creating layers

The multilayer perceptron

3 layers   "feedforward"

A powerful model
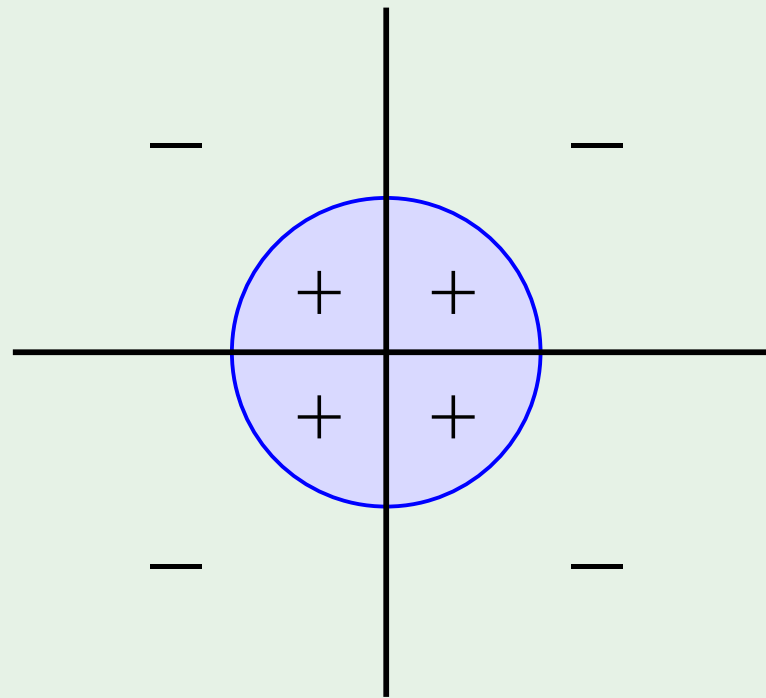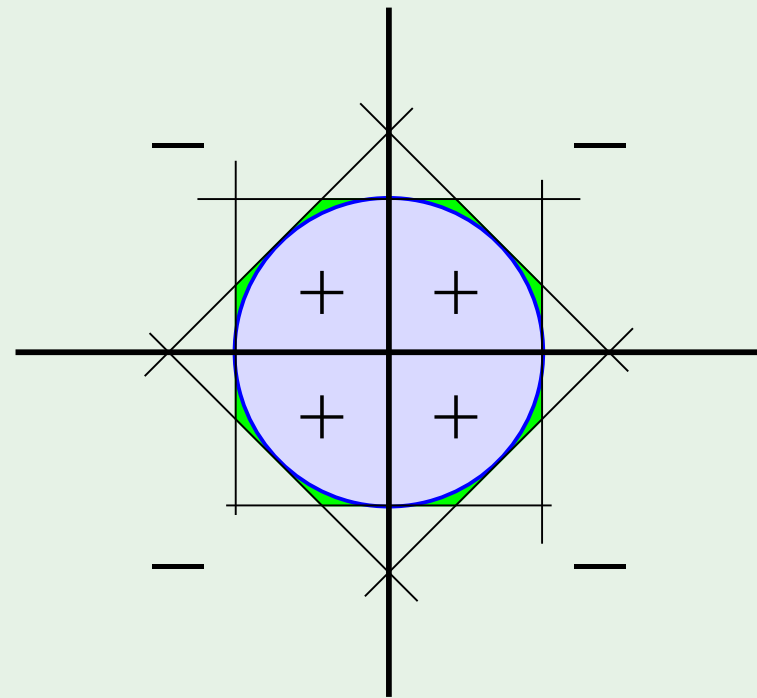
Target   8 perceptrons   16 perceptrons
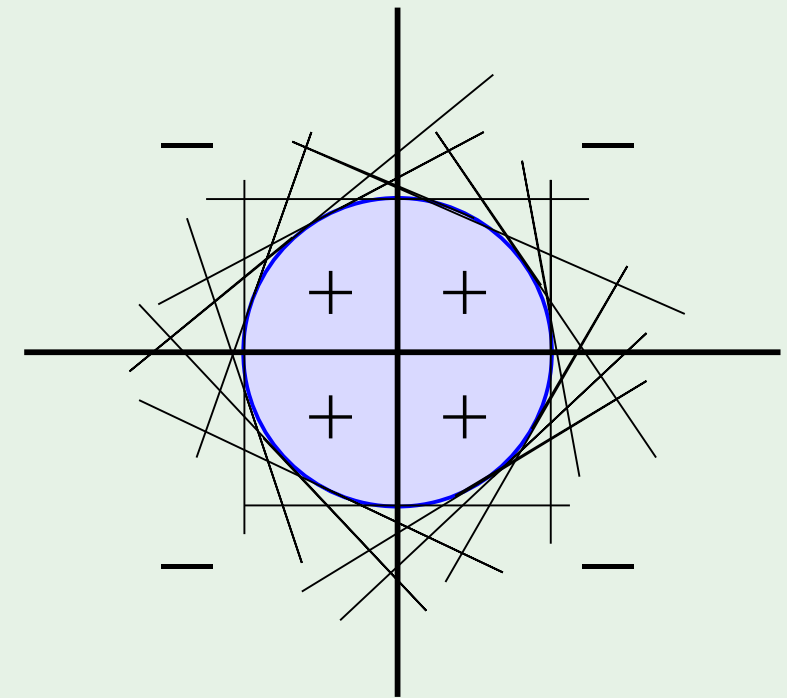
2 red flags for **generalization** and **optimization**

The neural network

input $\mathbf{x}$ hidden layers $1 \le l < L$ output layer $l = L$
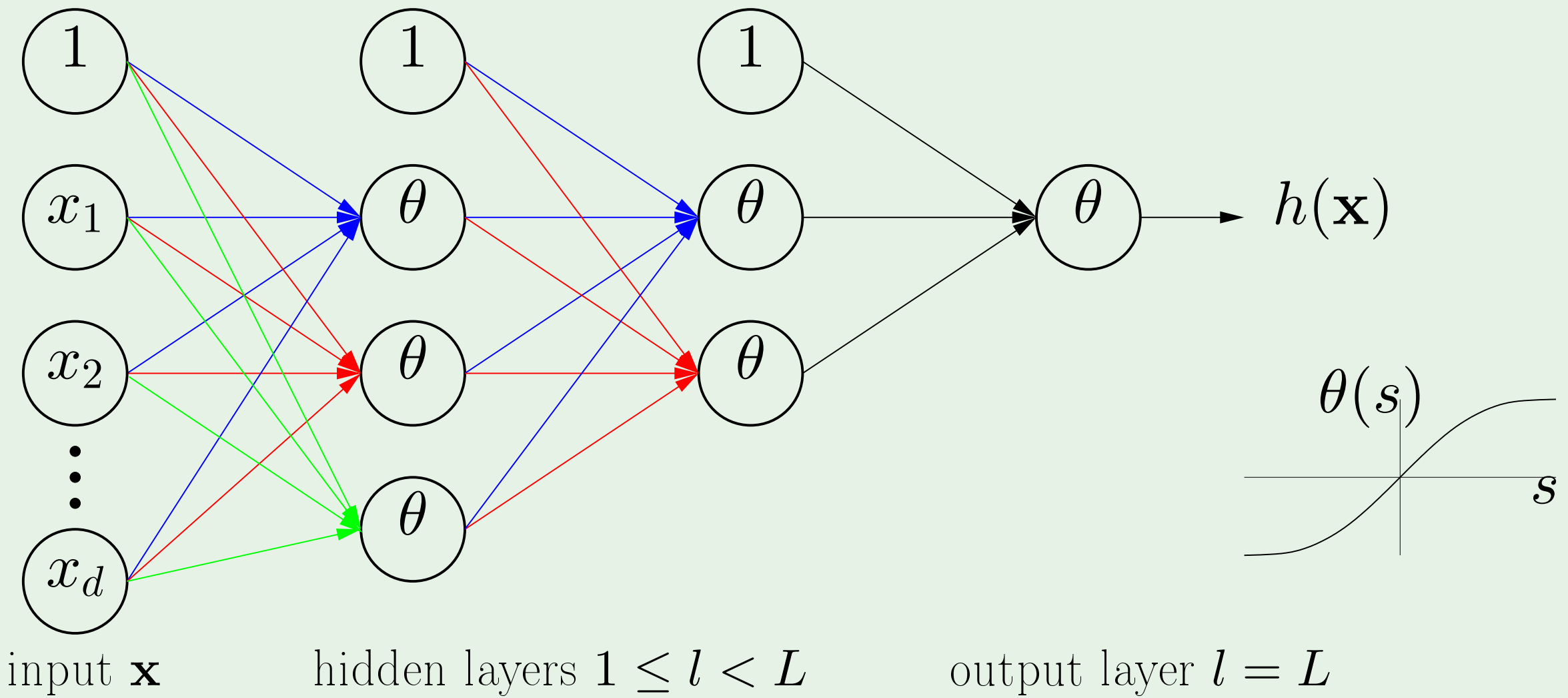
How the network operates

$$w_{ij}^{(l)} \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta \left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} \, x_i^{(l-1)} \right)$$

Apply $\mathbf{x}$ to $x_1^{(0)} \cdots x_{d^{(0)}}^{(0)} \rightarrow \rightarrow x_1^{(L)} = h(\mathbf{x})$

linear

$+1$ -------

tanh

-1

hard threshold
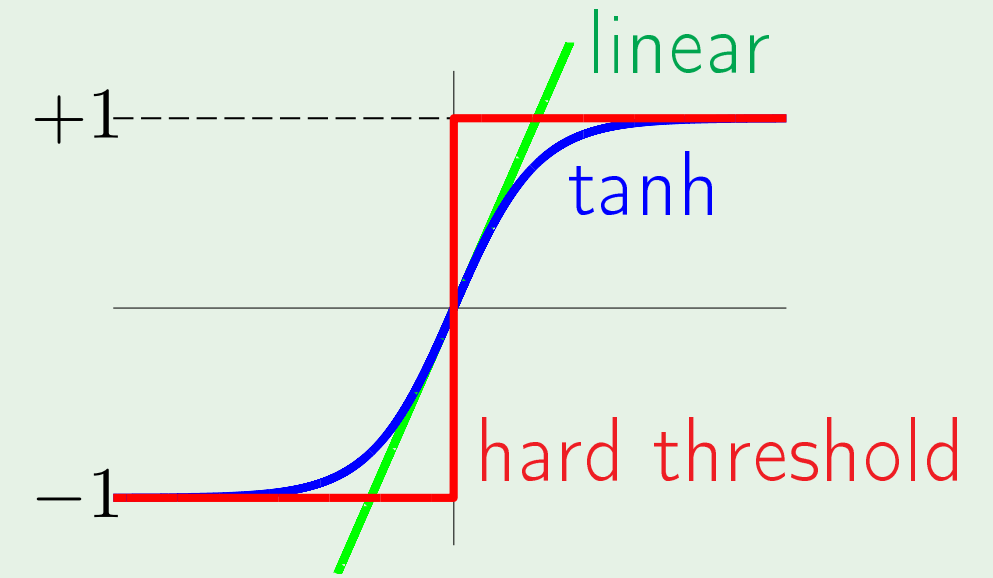
$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Outline

- Stochastic gradient descent

- Neural network model

- Backpropagation algorithm

Applying SGD

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

$$\nabla\mathbf{e}(\mathbf{w}): \quad \frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} \quad \text{for all}\ \ i, j, l$$

Computing $\dfrac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

$$\frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} = \frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ s_{j}^{(l)}} \times \frac{\partial\ s_{j}^{(l)}}{\partial\ w_{ij}^{(l)}}$$

We have $\dfrac{\partial\ s_{j}^{(l)}}{\partial\ w_{ij}^{(l)}} = x_{i}^{(l-1)}$ We only need: $\dfrac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ s_{j}^{(l)}} = \delta_{j}^{(l)}$
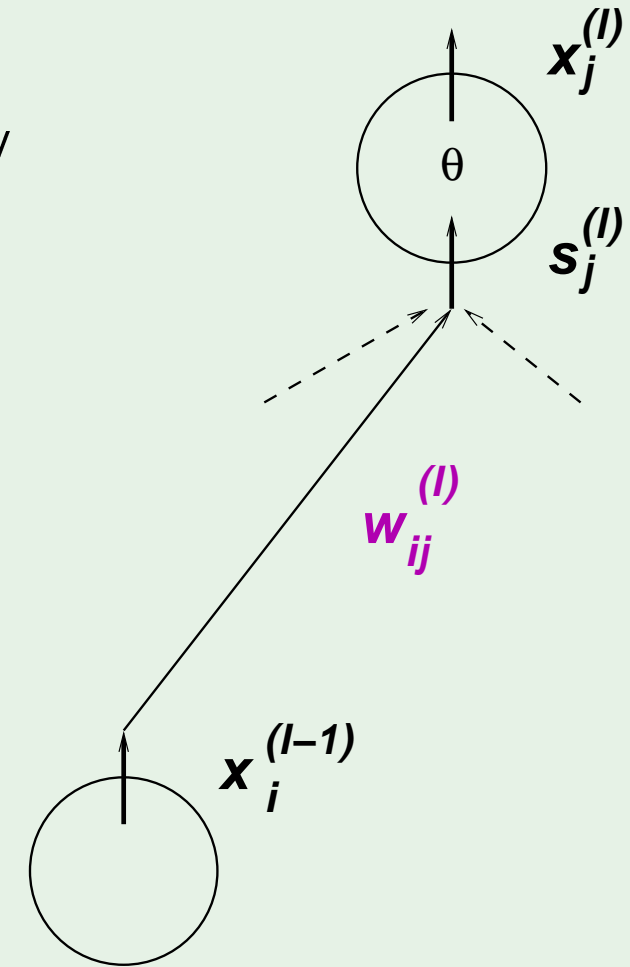
δ for the final layer

$$\delta_j^{(l)} = \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\, \mathbf{e}(\mathbf{w})}{\partial\, s_1^{(L)}}$$
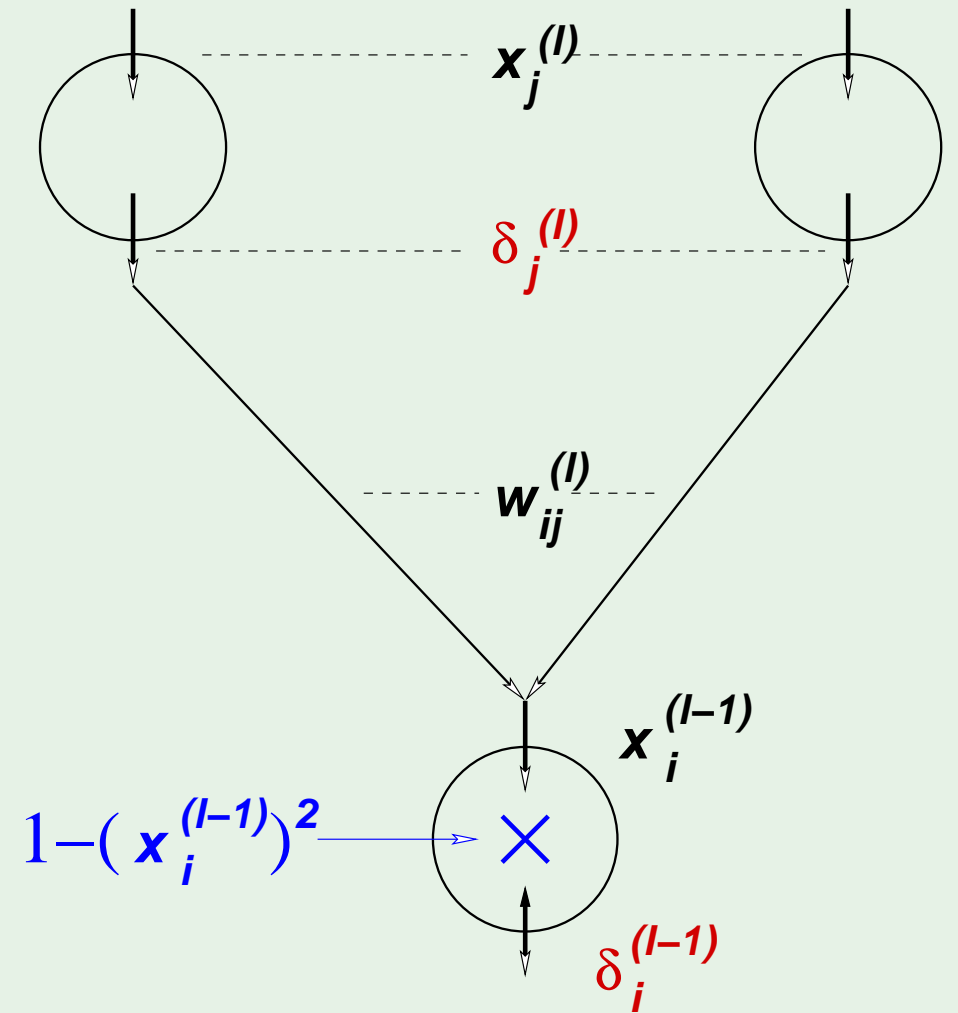
$$\mathbf{e}(\mathbf{w}) = (\, x_1^{(L)} - y_n)^2$$

$$x_1^{(L)} = \theta(s_1^{(L)})$$

$$\theta'(s) = 1 - \theta^2(s) \qquad \text{for the tanh}$$

Back propagation of δ

$$\delta_i^{(l-1)} = \frac{\partial \, \mathbf{e}(\mathbf{w})}{\partial \, s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \frac{\partial \, \mathbf{e}(\mathbf{w})}{\partial \, s_j^{(l)}} \times \frac{\partial \, s_j^{(l)}}{\partial \, x_i^{(l-1)}} \times \frac{\partial \, x_i^{(l-1)}}{\partial \, s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

$$\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \, \delta_j^{(l)}$$

Backpropagation algorithm

1: Initialize all weights $w_{ij}^{(l)}$ **at random**

2: **for** $t = 0, 1, 2, \ldots$ **do**

3:      Pick $n \in \{1, 2, \cdots, N\}$

4:      *Forward:* Compute all $x_j^{(l)}$

5:      *Backward:* Compute all $\delta_j^{(l)}$

6:      Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \, x_i^{(l-1)} \delta_j^{(l)}$

7:      Iterate to the next step until it is time to stop

8: Return the final weights $w_{ij}^{(l)}$

$\delta_j^{(l)}$

$w_{ij}^{(l)}$

$x_i^{(l-1)}$
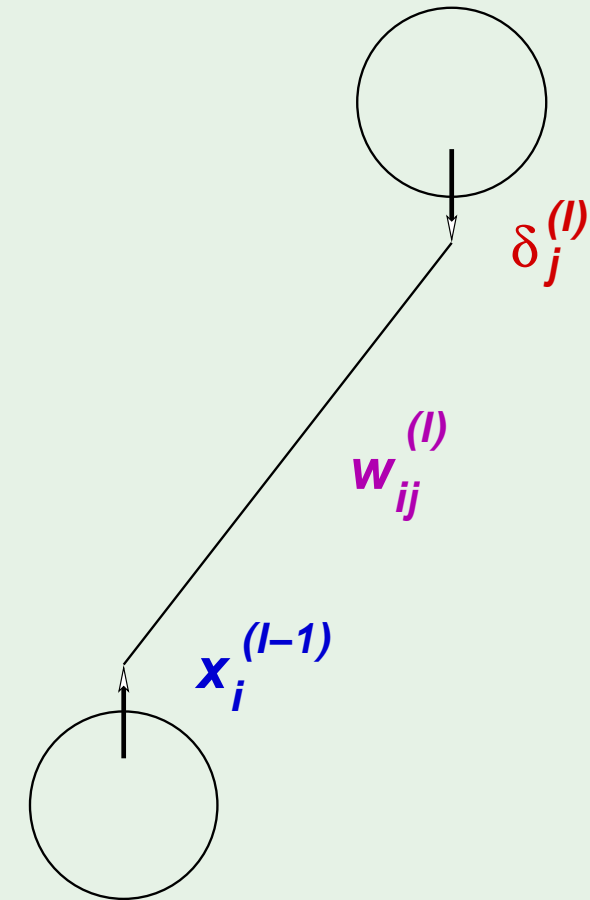
Final remark: hidden layers

learned nonlinear transform

interpretation?