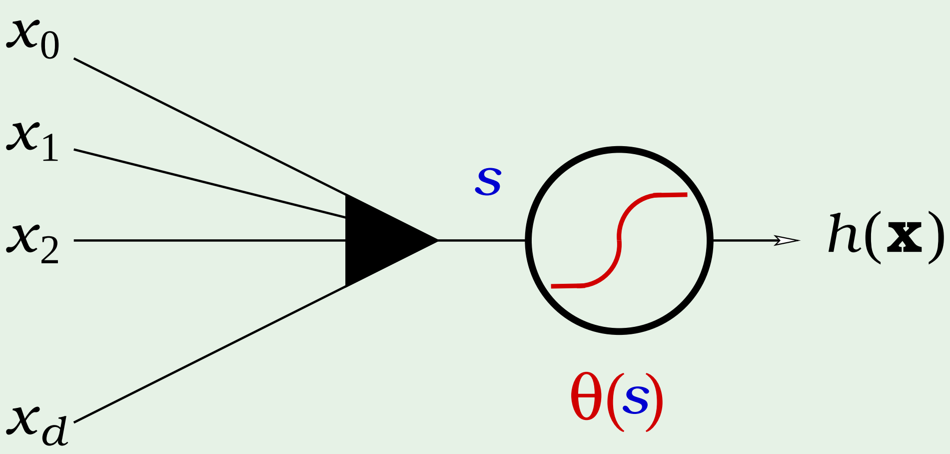
Review of Lecture 9

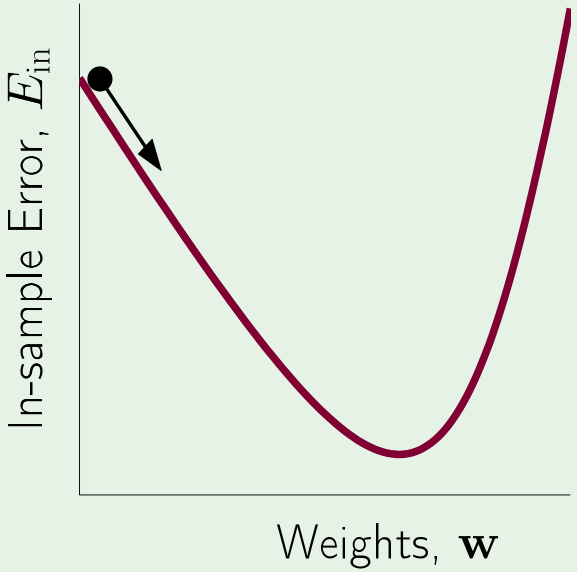
- Logistic regression



- 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**

Sponsored by Caltech's Provost Office, E&AS Division, and IST • Thursday, May 3, 2012

Outline

- Stochastic gradient descent

- Neural network model

- Backpropagation algorithm

Stochastic gradient descent

GD minimizes:

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

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 one $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$

"Average" direction:
$$\mathbb{E}_n\left[-\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)\right] = \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$$
$$= -\nabla\, E_{\text{in}}$$

randomized version of GD

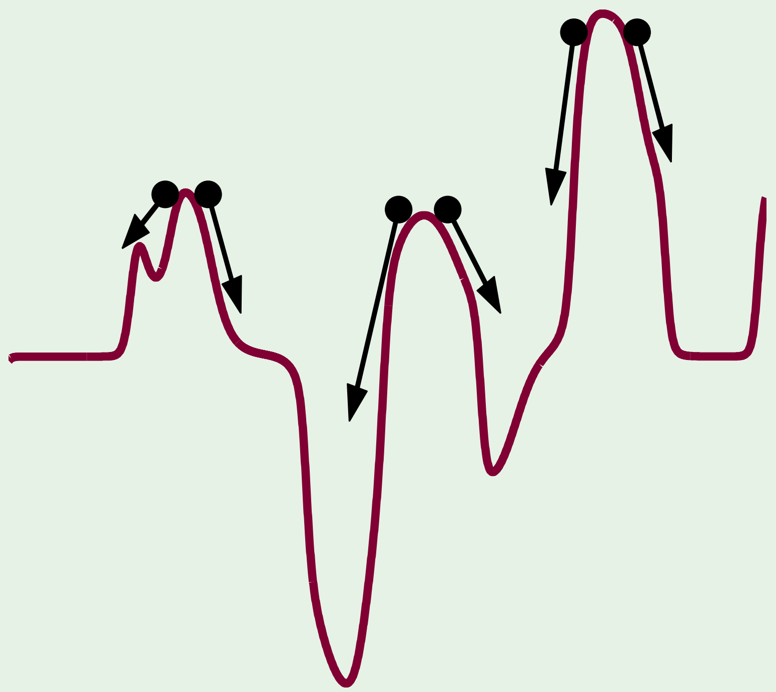
stochastic gradient descent (SGD)

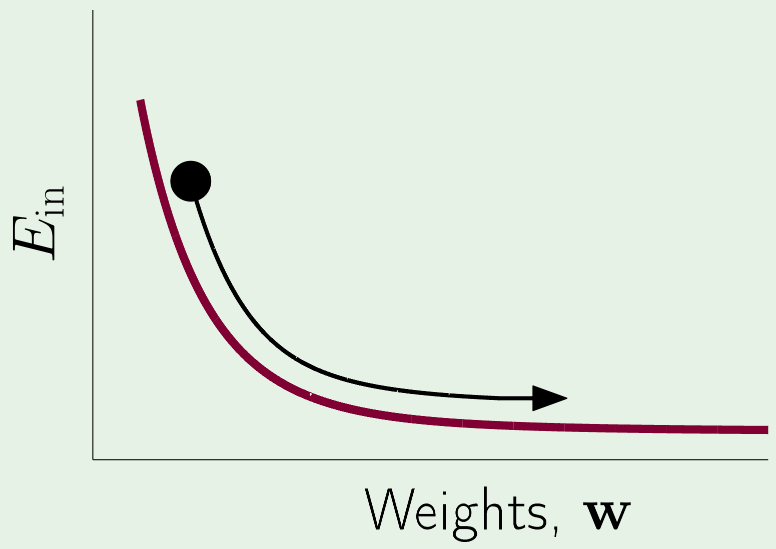
Benefits of SGD

1. cheaper computation

2. randomization

3. simple





Rule of thumb:

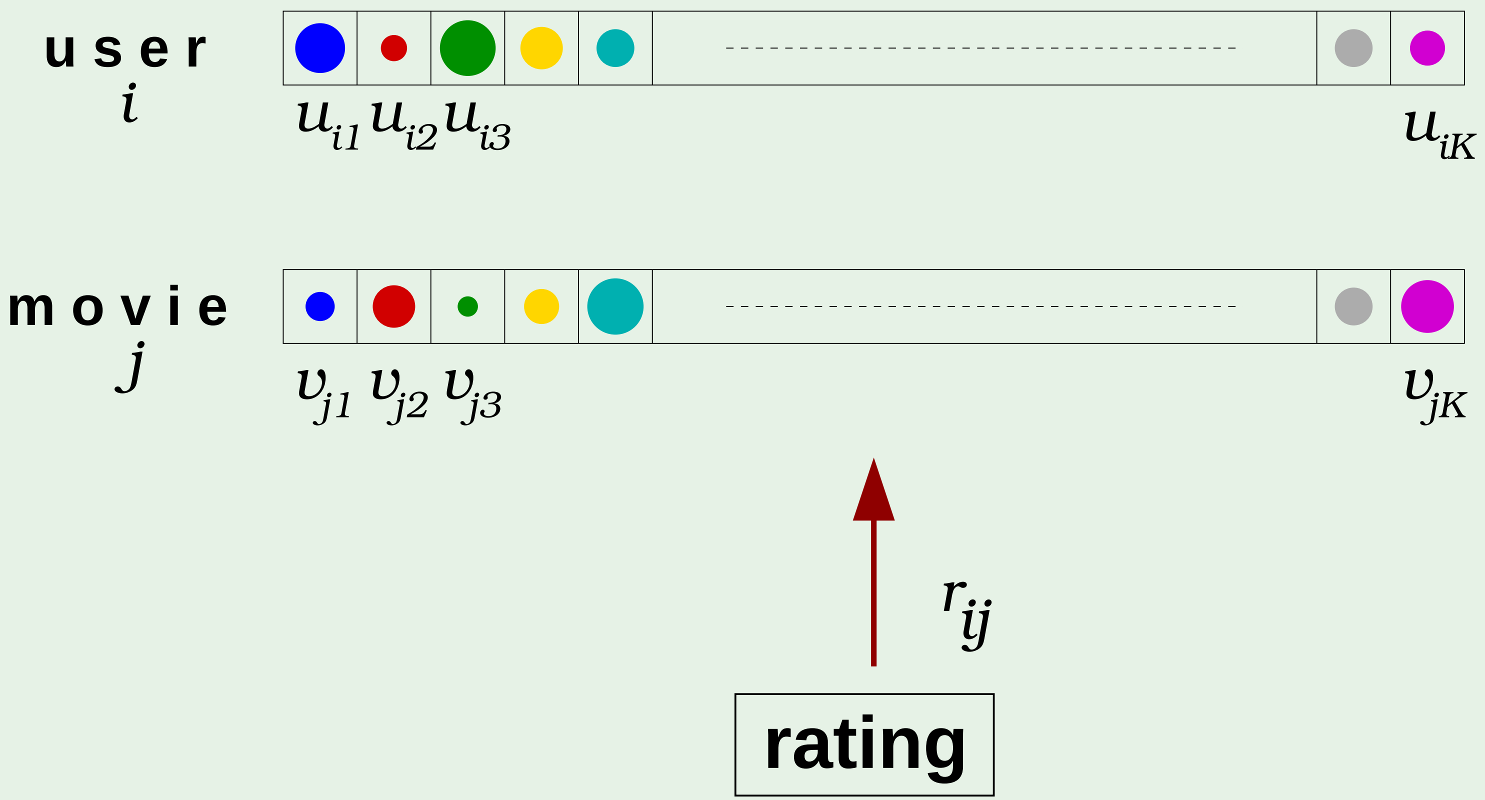
$\eta = 0.1$ works

randomization helps

SGD in action

Remember movie ratings?

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



Outline

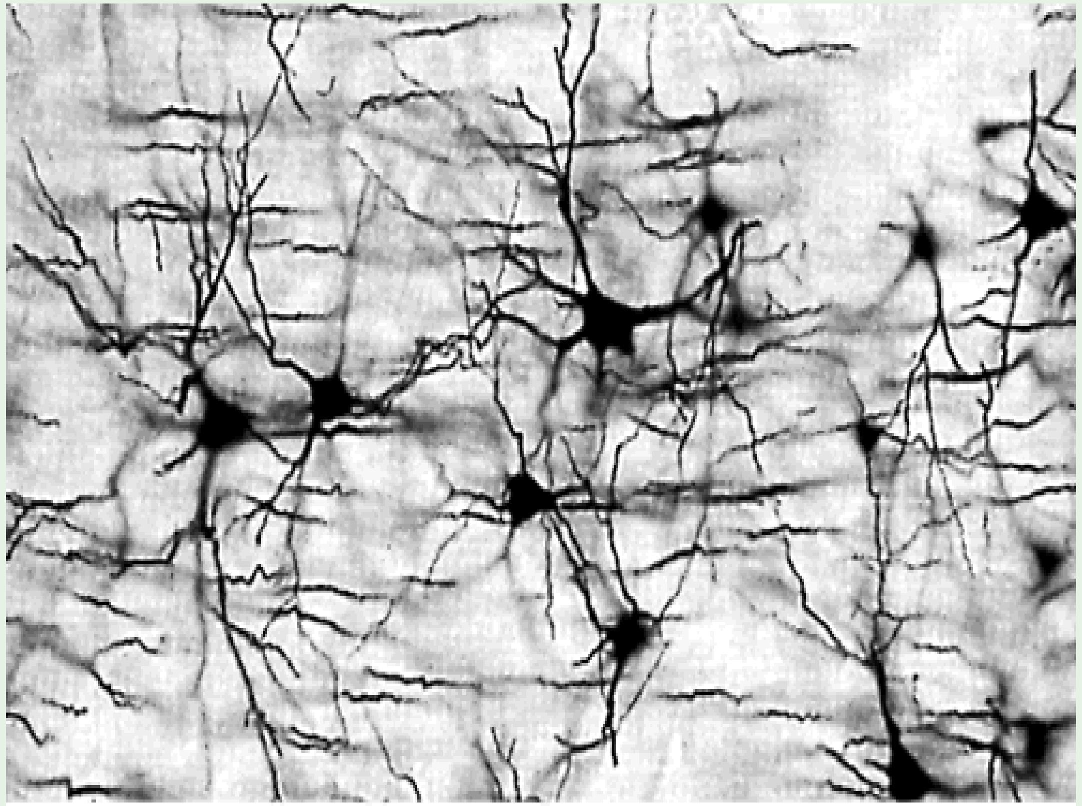
- Stochastic gradient descent

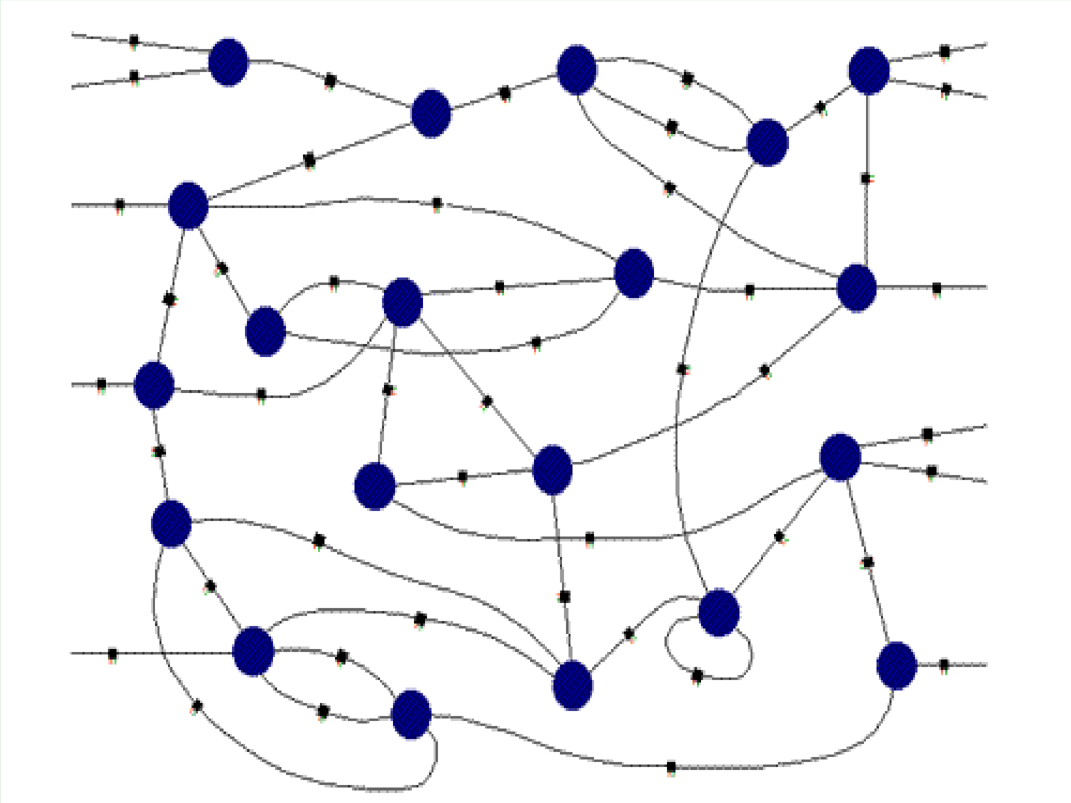
- Neural network model

- Backpropagation algorithm

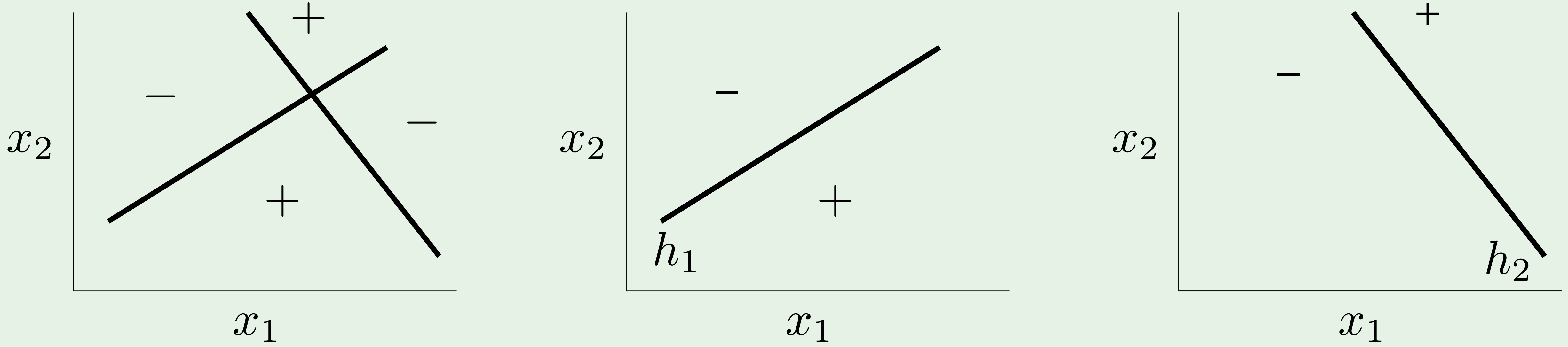
Biological inspiration

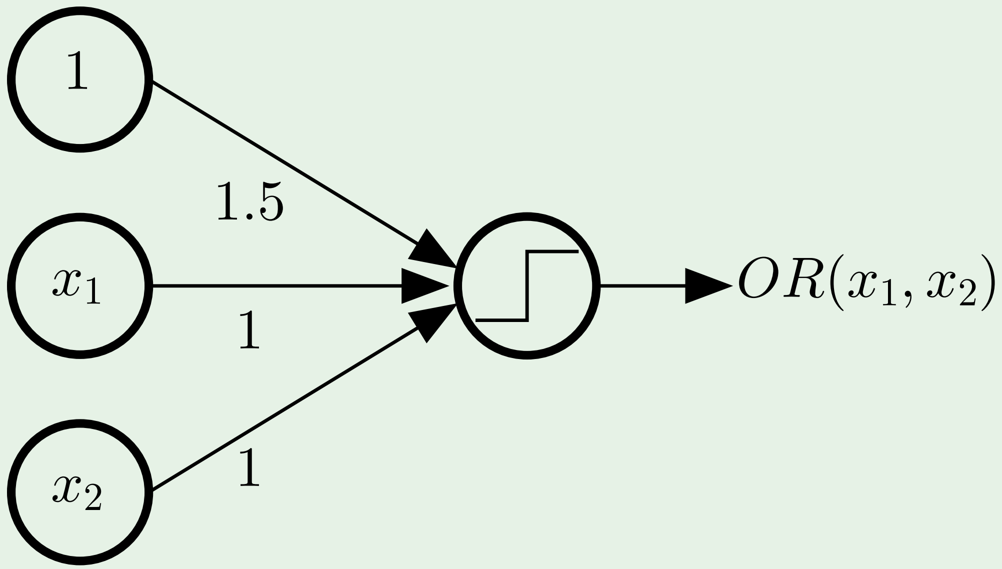
biological function $\longrightarrow$ biological structure

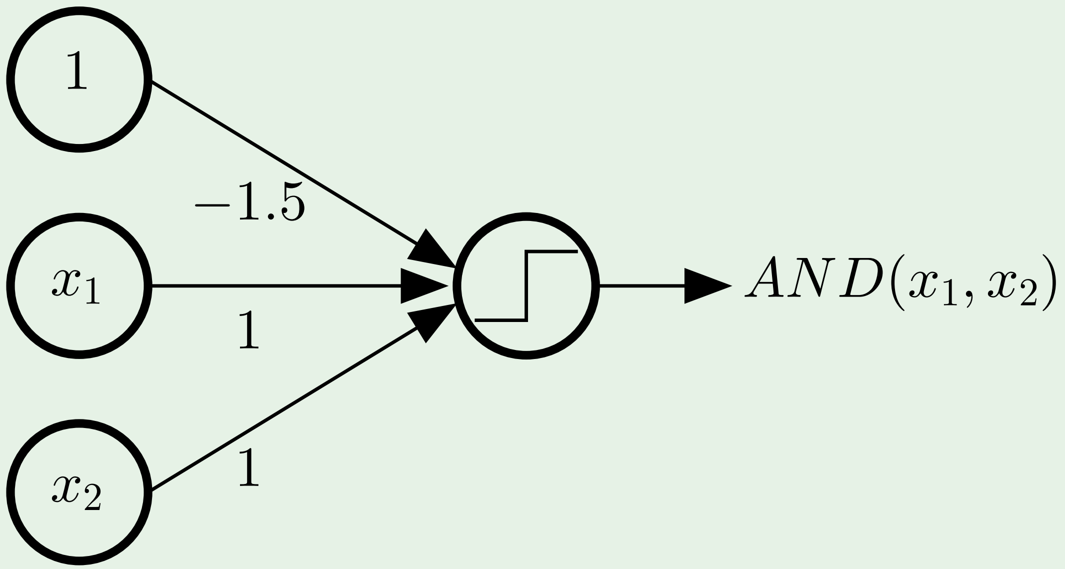




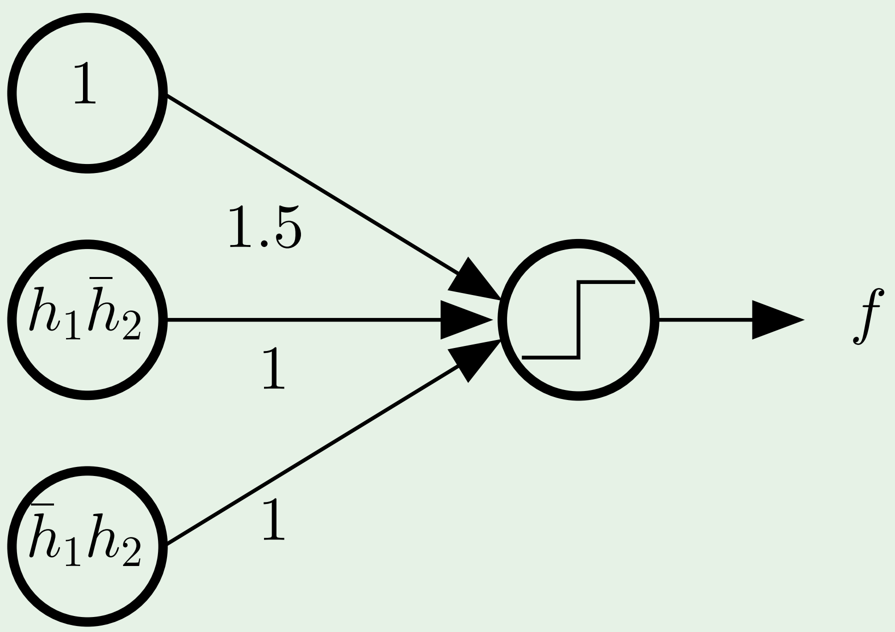
Combining perceptrons

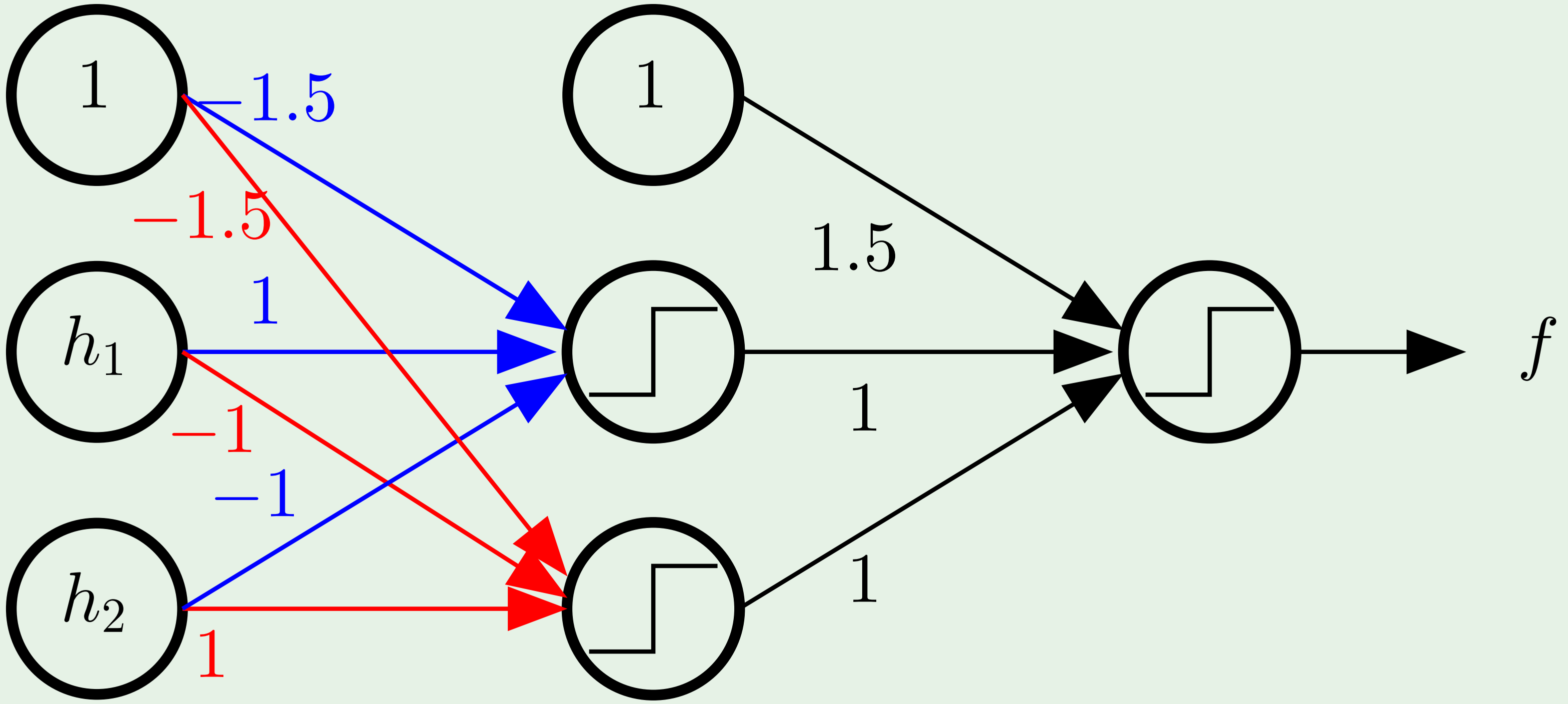




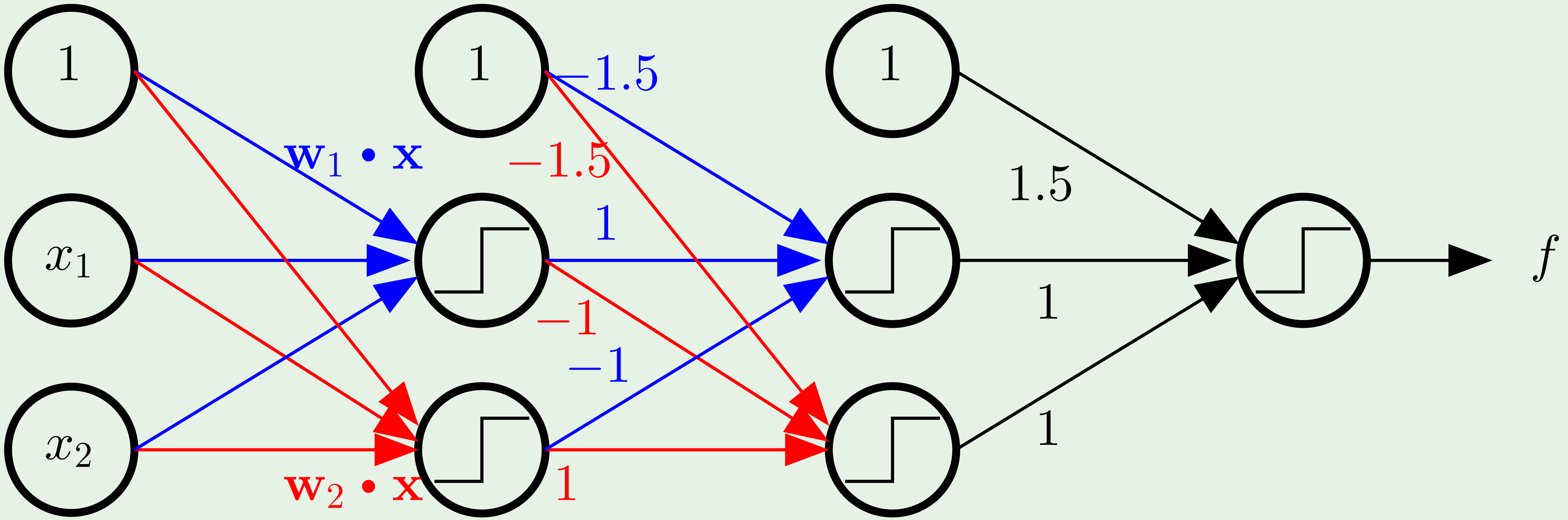


Creating layers



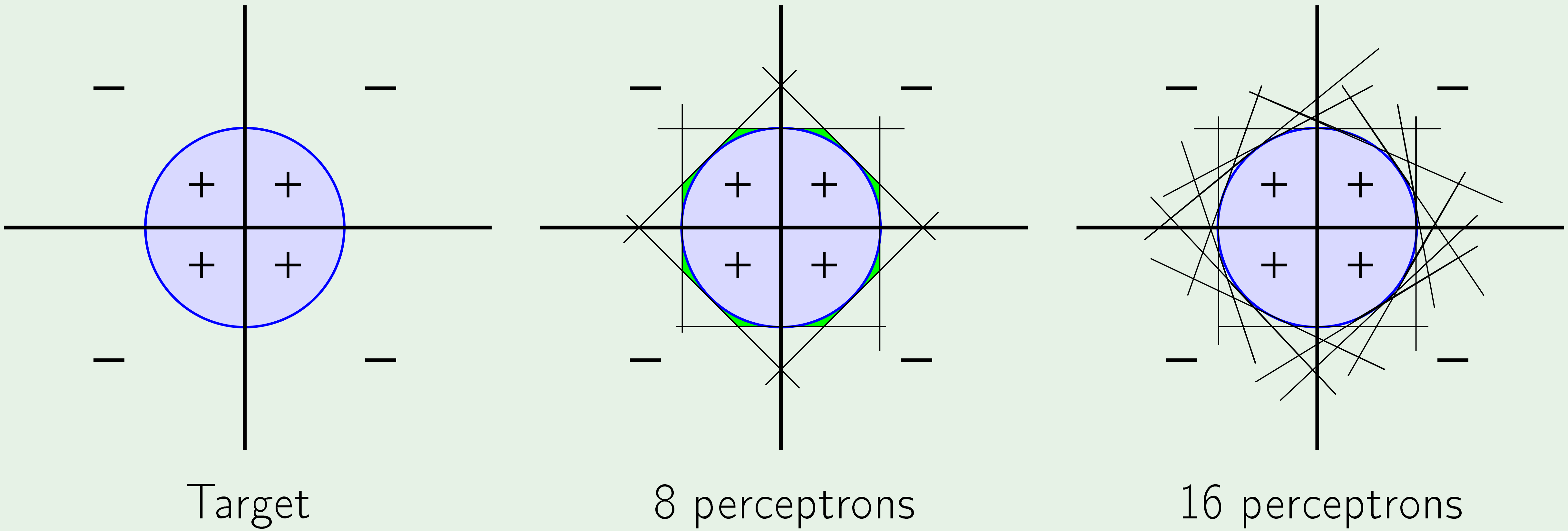


The multilayer perceptron



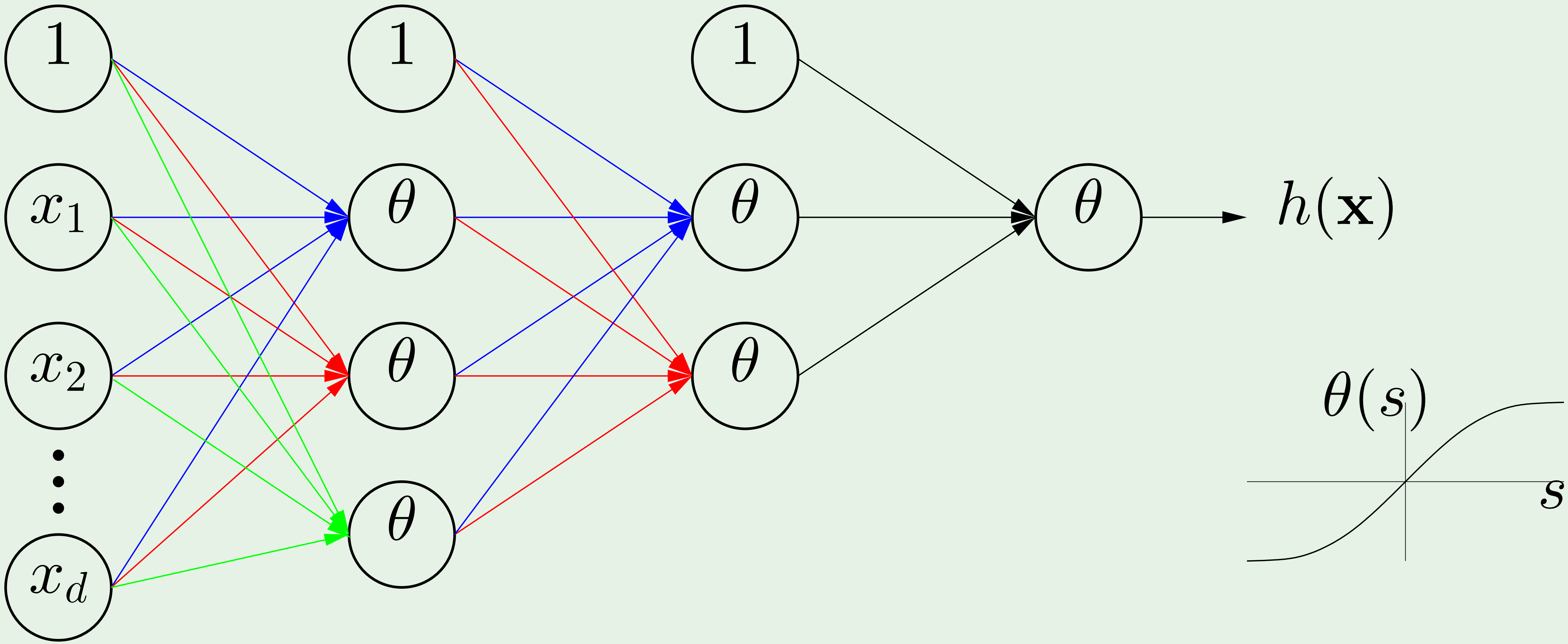
3 layers "feedforward"

A powerful model



2 red flags for **generalization** and **optimization**

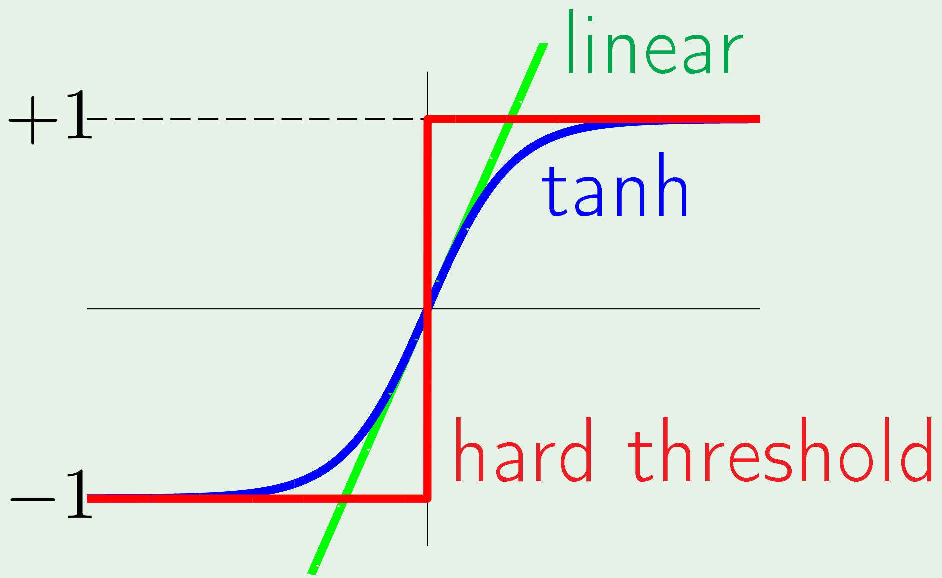
The neural network



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

How the network operates

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



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

if bias is used then: $x_j^{(l)} = \theta(s_j^{(l)}) = \theta(\mathrm{sum}(w_{ij}^{(l)} x_i^{(l-1)} + b_j))$

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

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

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

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

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

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

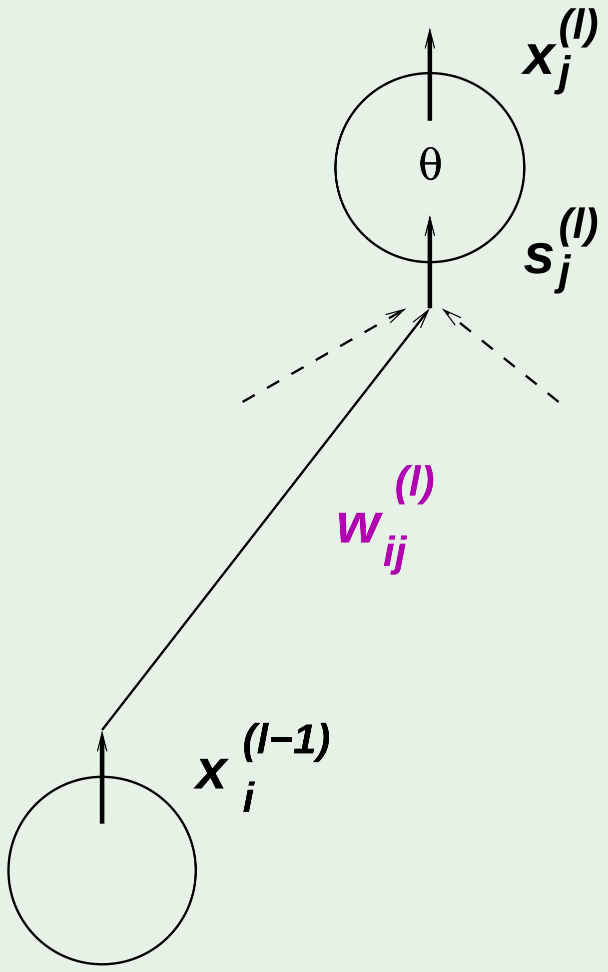
A trick for efficient computation:

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

if bias is used then: $x_j^{(l)} = \theta(s_j^{(l)}) = \theta(\mathrm{sum}(w_{ij}^{(l)}.x_i^{(l-1)}+b_j^{(l)})$

so PD e(w)/ PD $b_j^{(l)}$ = PD e(w)/PD $s_j^{(l)}$ x PD $s_j^{(l)}$/PD $b_j^{(l)}$ = $\delta_j^{(l)}$ x 1 = $\delta_j^{(l)}$

We have $\dfrac{\partial\ s_j^{(l)}}{\partial\ w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}} = \delta_j^{(l)}$



δ for the final layer

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

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

$$\delta_1^{(L)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_1^{(L)}}$$

$$\mathrm{e}(\mathbf{w}) = (\, x_1^{(L)} - y_n)^2$$

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

$$\theta'(s) = 1 - \theta^2(s)$$ for the tanh

$\delta^{(L)}{}_1=2*(\theta(S^{(L)}{}_1)-y_n)*(1-\theta^2(S^{(L)}{}_1))$ $\delta^{(L)}{}_1=2*(x^{(L)}{}_1-y_n)*(1-x^{(L)}{}_1\text{\^{}}2)$

Back propagation of δ

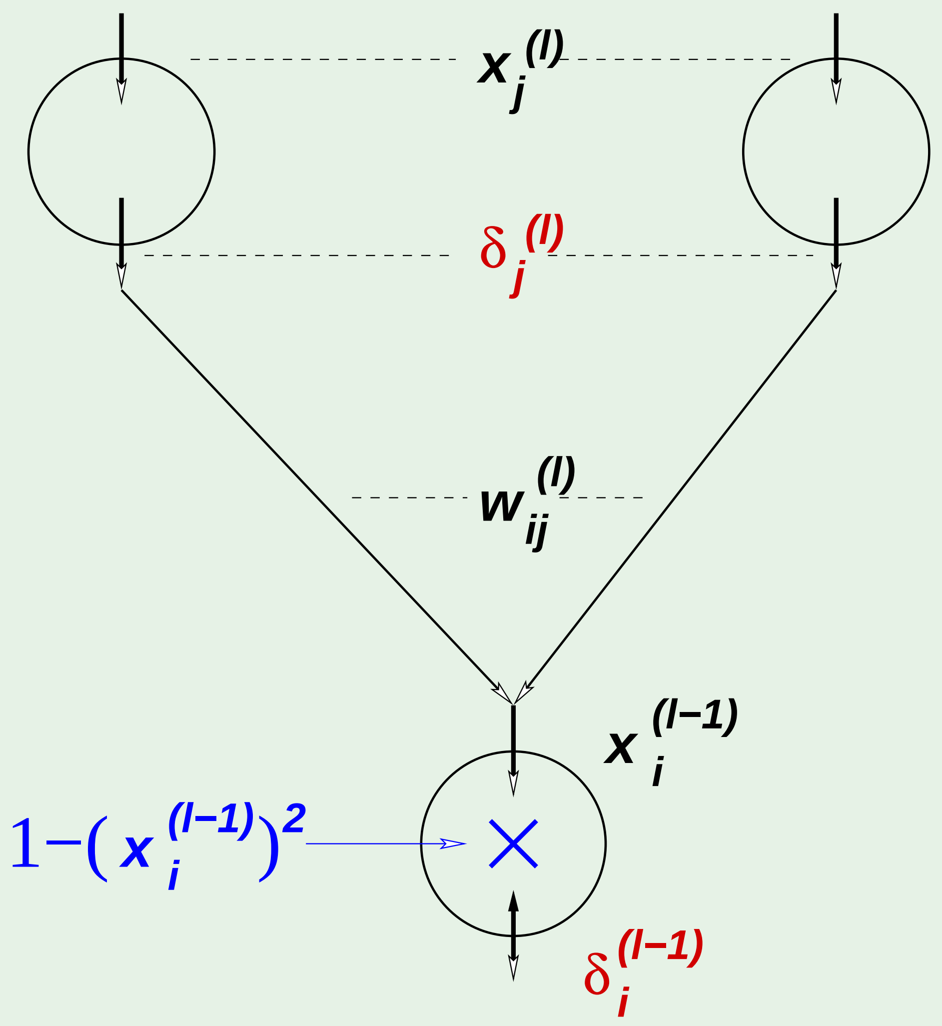
$$\delta_i^{(l-1)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}} = \sum_{j=1}^{d^{(l)}} \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, s_i^{(l-1)}}$$

chain rule for partial derivatives

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

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

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



The overall error is: $\nabla E = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right)$

Since we want to reduce the error by changing the weights in every iteration.

Hence, $\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}}$

Also refer to slide 21/24 of Logistic regression lecture.

$\Delta w_{ij} = -\eta \frac{\partial E(w)}{\partial w_{ij}}$ where,

$\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, w_{ij}^{(l)}}$; $\frac{\partial\, s_j^{(l)}}{\partial\, w_{ij}^{(l)}} = x_i^{(l-1)}$ and $\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} = \delta_j^{(l)}$

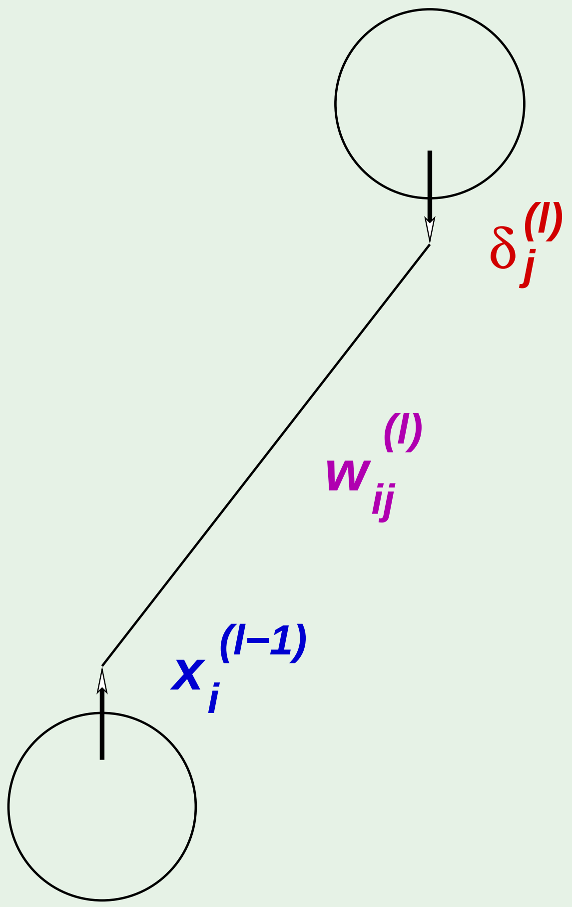
$\therefore \Delta w_{ij}^{(l)} = -\eta \delta_j^{(l)} x_i^{(l-1)} = w_{ij}^{(l)}(t+1) - w_{ij}^{(l)}(t)$

$\therefore w_{ij}^{(l)}(t+1) - w_{ij}^{(l)}(t) = -\eta \delta_j^{(l)} x_i^{(l-1)}$ or

$w_{ij}^{(l)}(t+1) = w_{ij}^{(l)}(t) - \eta \delta_j^{(l)} x_i^{(l-1)}$

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 $\delta_j^{(l)}$
5.5: update bias: $B_j^{(l)} \leftarrow B_j^{(l)} - \eta * 1 * \delta_j^{(l)}$
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 $w_{ij}^{(l)}$



Final remark: hidden layers

learned nonlinear transform

interpretation?