Where we are

- Linear classification ✓
- Linear regression ✓
- Logistic regression
- Nonlinear transforms

Nonlinear transforms

$$\mathbf{x} = (x_0, x_1, \cdots, x_d) \quad \xrightarrow{\Phi} \quad \mathbf{z} = (z_0, z_1, \cdots\cdots\cdots\cdots, z_{\tilde{d}})$$

Each $z_i = \phi_i(\mathbf{x})$ $\qquad$ $\mathbf{z} = \Phi(\mathbf{x})$

Example: $\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$

Final hypothesis $g(\mathbf{x})$ in $\mathcal{X}$ space:

$$\text{sign}\left(\tilde{\mathbf{w}}^\mathsf{T}\Phi(\mathbf{x})\right) \qquad \text{or} \qquad \tilde{\mathbf{w}}^\mathsf{T}\Phi(\mathbf{x})$$

The price we pay

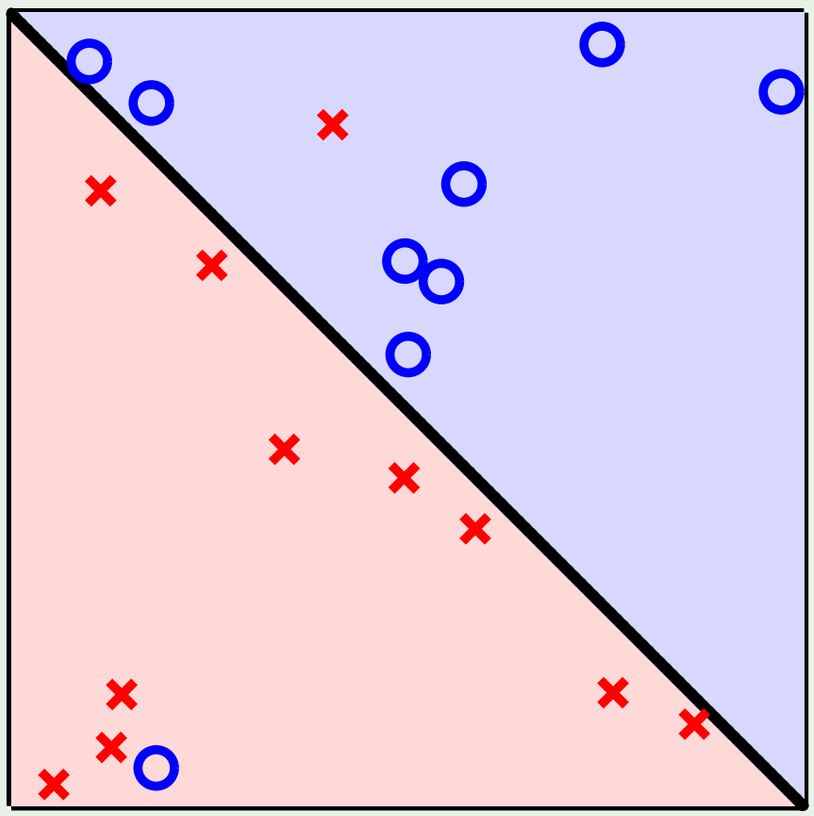
$$\mathbf{x} = (x_0, x_1, \cdots, x_d) \quad \xrightarrow{\Phi} \quad \mathbf{z} = (z_0, z_1, \cdots\cdots\cdots\cdots, z_{\tilde{d}})$$

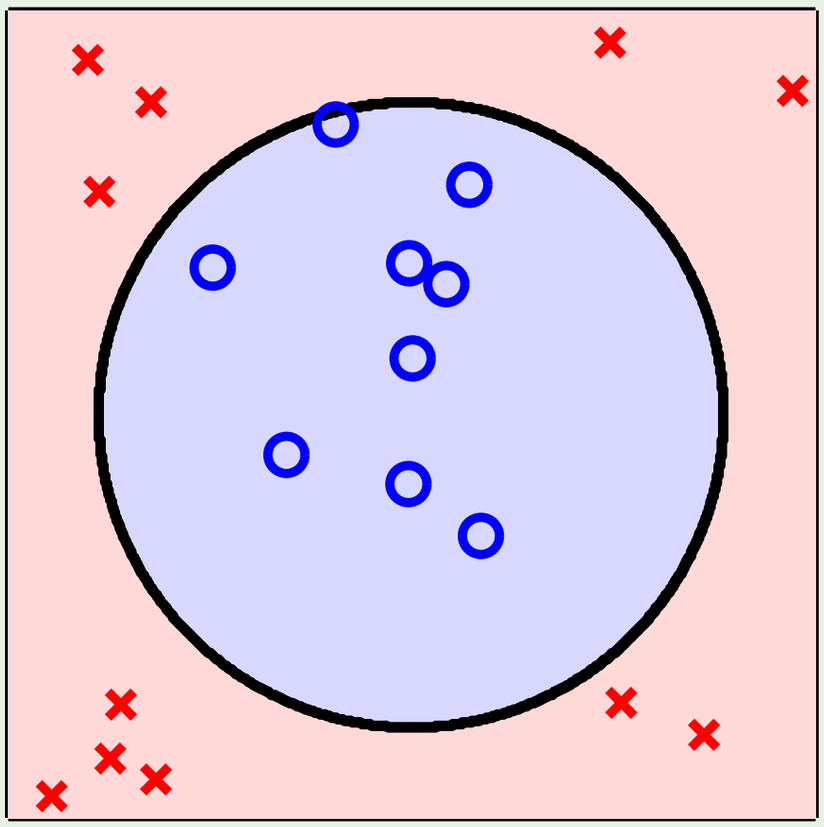
$$\downarrow \qquad\qquad\qquad\qquad\qquad\qquad \downarrow$$

$$\mathbf{w} \qquad\qquad\qquad\qquad\qquad\qquad \tilde{\mathbf{w}}$$

$$d_{\text{VC}} = d + 1 \qquad\qquad\qquad\qquad d_{\text{VC}} \leq \tilde{d} + 1$$

Two non-separable cases



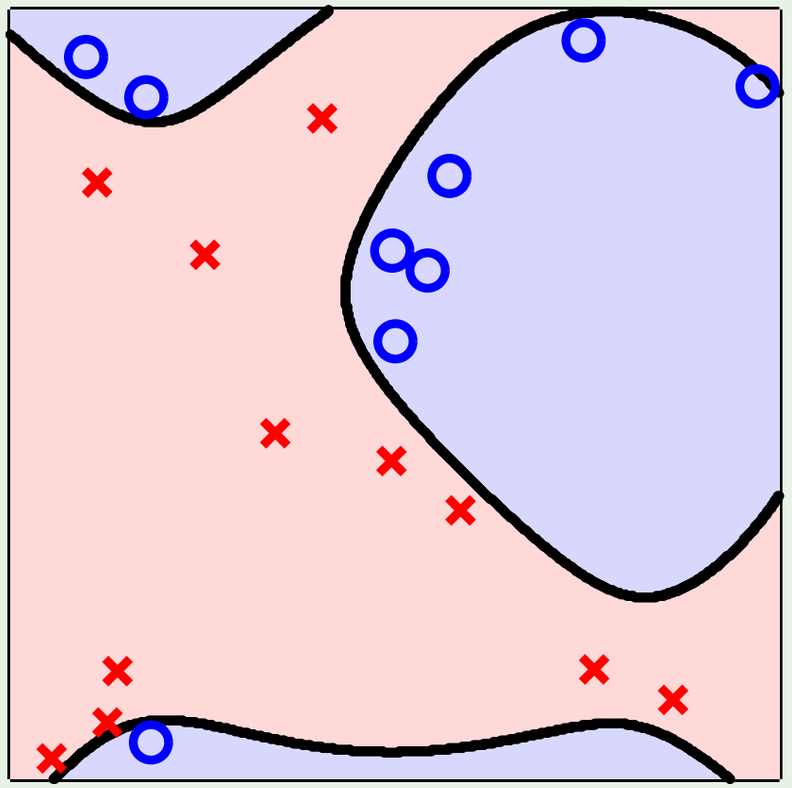


First case

Use a linear model in $\mathcal{X}$; accept $E_{\text{in}} > 0$

or

Insist on $E_{\text{in}} = 0$; go to high-dimensional $\mathcal{Z}$



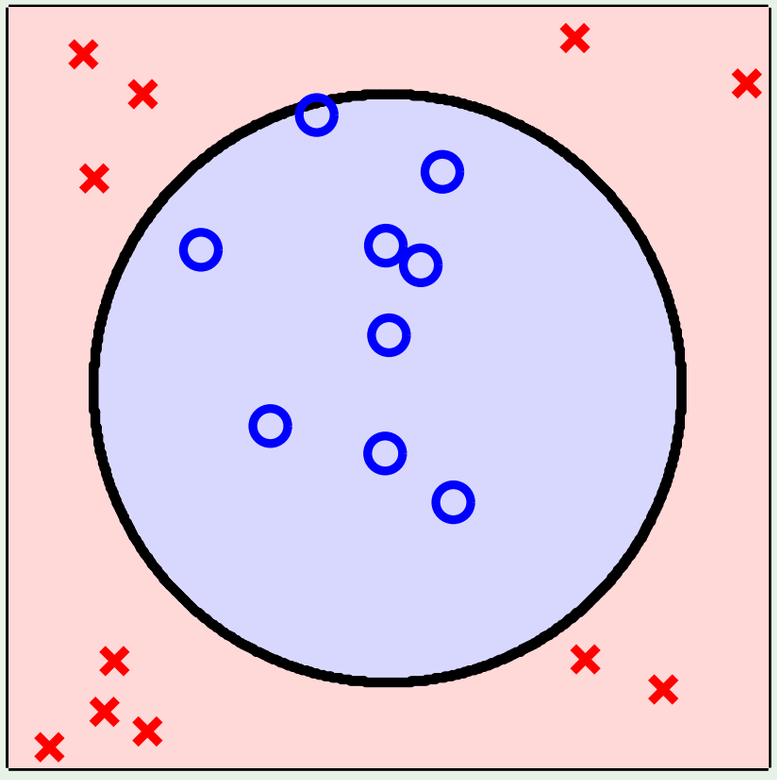
Second case

$$\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

Why not: $\mathbf{z} = (1, x_1^2, x_2^2)$

or better yet: $\mathbf{z} = (1, x_1^2 + x_2^2)$

or even: $\mathbf{z} = (x_1^2 + x_2^2 - 0.6)$



Lesson learned

Looking at the data *before* choosing the model can be hazardous to your E_{out}

Data snooping

Logistic regression - Outline

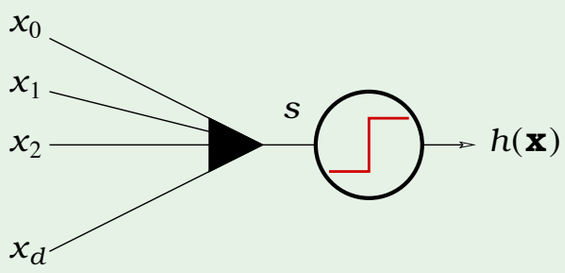
- The model
- Error measure
- Learning algorithm

A third linear model

$$s = \sum_{i=0}^{d} w_i x_i$$

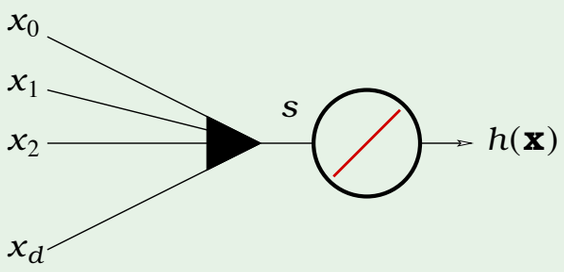
linear classification

$$h(\mathbf{x}) = \text{sign}(s)$$



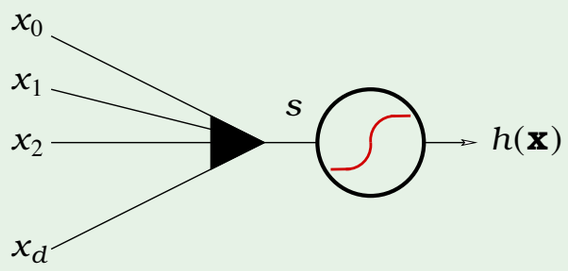
linear regression

$$h(\mathbf{x}) = s$$



logistic regression

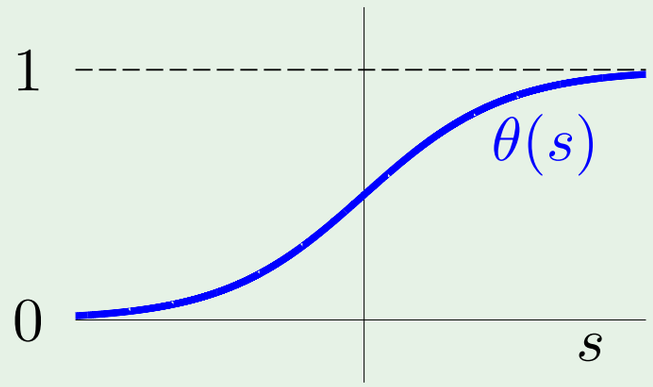
$$h(\mathbf{x}) = \theta(s)$$



The logistic function θ

The formula:

$$\theta(s) = \frac{e^s}{1 + e^s}$$



soft threshold: uncertainty

sigmoid: flattened out 's'

Error measure

For each $(\mathbf{x}, y)$, y is generated by probability $f(\mathbf{x})$

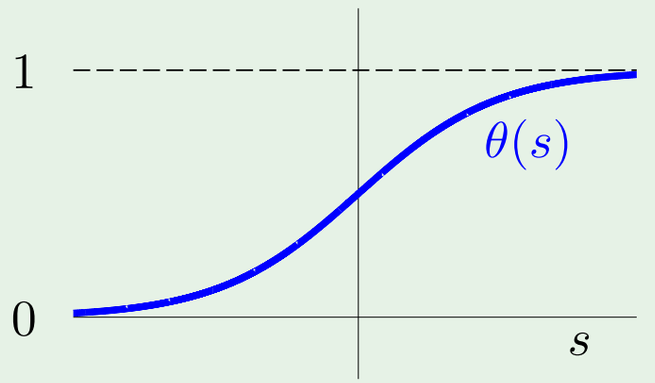
Plausible error measure based on **likelihood:**

If $h = f$, how likely to get y from $\mathbf{x}$?

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

Formula for likelihood

$$P(y \mid \mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1; \\ 1 - h(\mathbf{x}) & \text{for } y = -1. \end{cases}$$



Substitute $h(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T}\mathbf{x})$, noting $\theta(-s) = 1 - \theta(s)$

$$P(y \mid \mathbf{x}) = \theta(y\ \mathbf{w}^\mathsf{T}\mathbf{x})$$

Likelihood of $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ is

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

Maximizing the likelihood

Minimize $$-\frac{1}{N}\ln\left(\prod_{n=1}^{N}\theta(y_n\ \mathbf{w}^{\mathsf{T}}\ \mathbf{x}_n)\right)$$

$$=\frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{1}{\theta(y_n\ \mathbf{w}^{\mathsf{T}}\ \mathbf{x}_n)}\right) \qquad \left[\theta(s)=\frac{1}{1\ +e^{-s}}\right]$$

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

"cross-entropy" error

Logistic regression - Outline

- The model
- Error measure
- Learning algorithm

How to minimize E_{in}

For logistic regression,

$$E_{\text{in}}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right) \qquad \longleftarrow \textbf{iterative} \text{ solution}$$

Compare to linear regression:

$$E_{\text{in}}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2 \qquad \longleftarrow \text{closed-form solution}$$

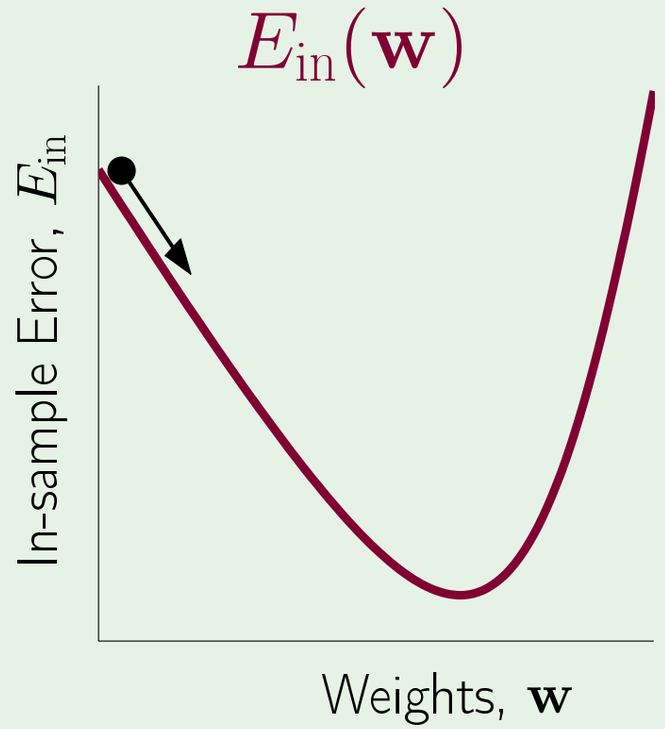
Iterative method: gradient descent

General method for nonlinear optimization

Start at $\mathbf{w}(0)$; take a step along steepest slope

Fixed step size: $\mathbf{w}(1) = \mathbf{w}(0) + \eta\,\hat{\mathbf{v}}$

What is the direction $\hat{\mathbf{v}}$?



Formula for the direction $\hat{\mathbf{v}}$

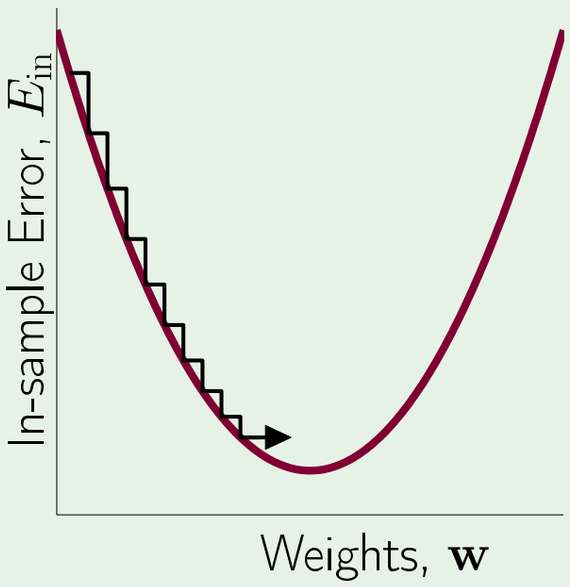
$$\begin{aligned}
\Delta E_{\text{in}} &= E_{\text{in}}(\,\mathbf{w}(0) + \eta\hat{\mathbf{v}}\,) - E_{\text{in}}(\mathbf{w}(0)) \\
&= \eta\nabla E_{\text{in}}(\mathbf{w}(0))^{\text{T}}\hat{\mathbf{v}} + O(\eta^2) \\
&\geq -\eta\|\nabla E_{\text{in}}(\mathbf{w}(0))\|
\end{aligned}$$

Since $\hat{\mathbf{v}}$ is a unit vector,

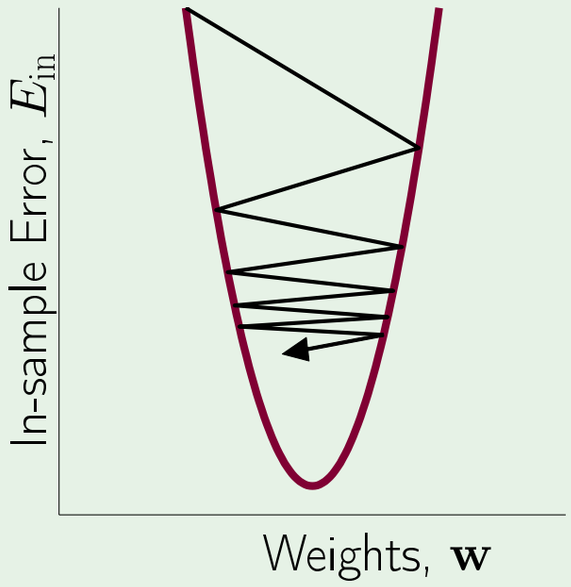
$$\hat{\mathbf{v}} = -\frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$

Fixed-size step?

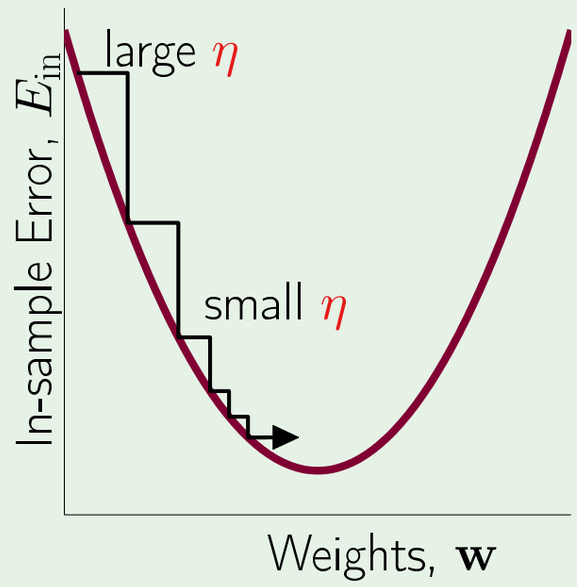
How η affects the algorithm:



η too small



η too large



variable η – just right

η should increase with the slope

Easy implementation

Instead of

$$\Delta \mathbf{w} \;=\; \eta\, \hat{\mathbf{v}}$$

$$=\; -\,\eta\, \frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$

Have

$$\Delta \mathbf{w} \;=\; -\,\eta\;\; \nabla E_{\text{in}}(\mathbf{w}(0))$$

Fixed **learning rate** η

Logistic regression algorithm

1: Initialize the weights at $t = 0$ to $\mathbf{w}(0)$

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

3: Compute the gradient

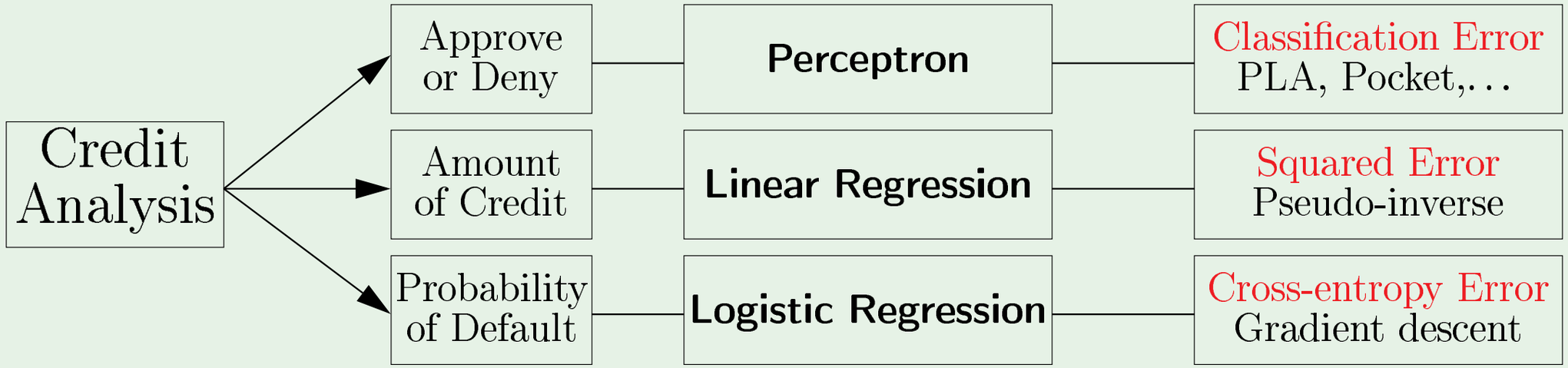
$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathsf{T}}(t) \mathbf{x}_n}}$$

4: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}$

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

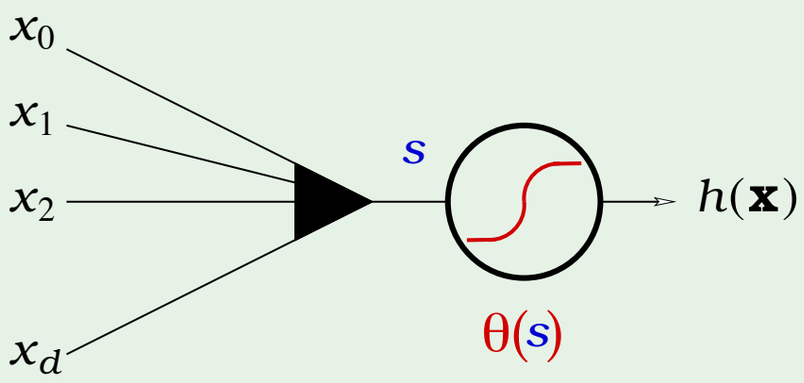
6: Return the final weights $\mathbf{w}$

Summary of Linear Models



Review of Lecture 9

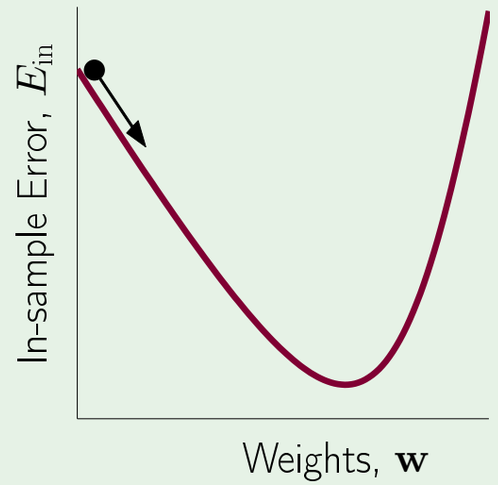
- Logistic regression



- 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:

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

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

"Average" direction:

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

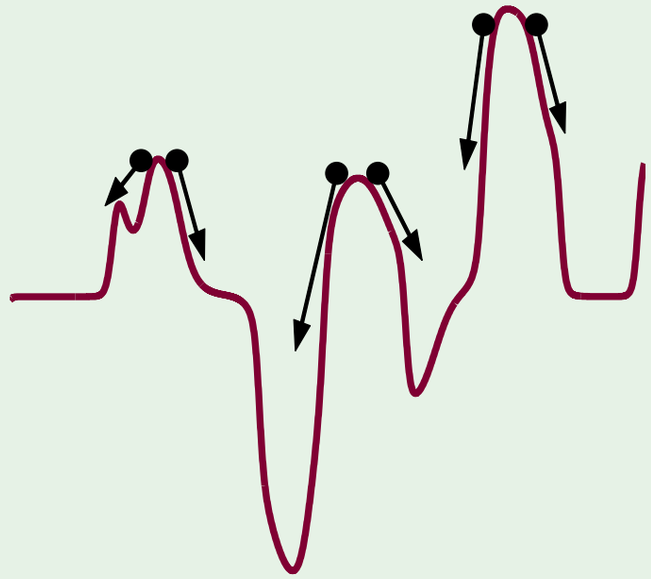
$$= -\nabla E_{\text{in}}$$

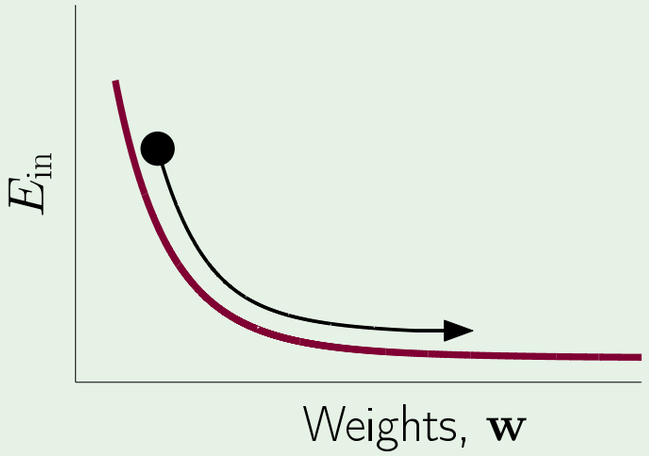
randomized version of GD

stochastic gradient descent (SGD)

Benefits of SGD

1. cheaper computation
2. randomization
3. simple





randomization helps

Rule of thumb:

$\eta = 0.1$ works

Outline

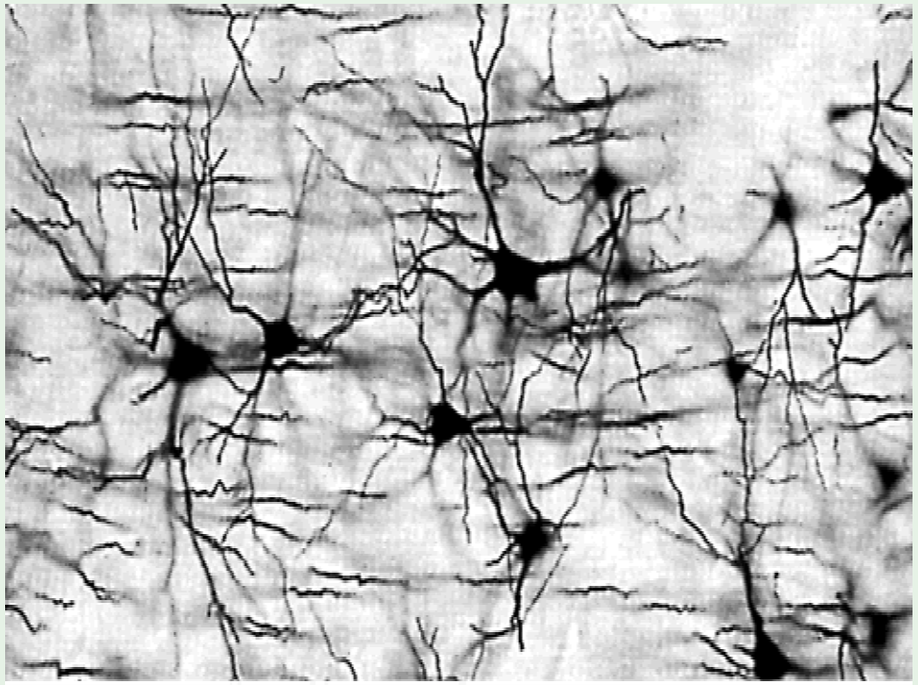
- Stochastic gradient descent

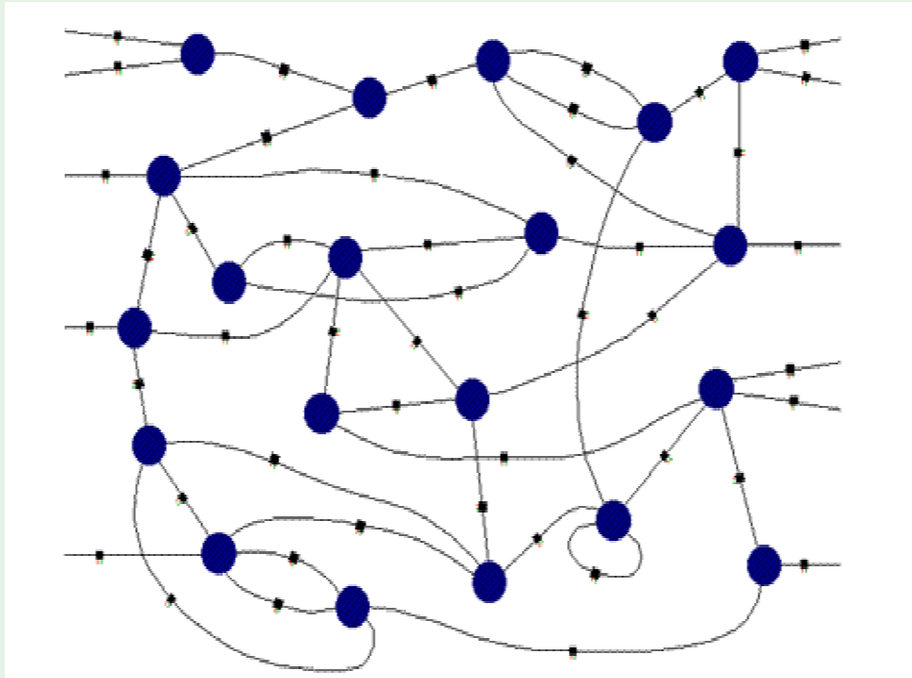
- Neural network model

- Backpropagation algorithm

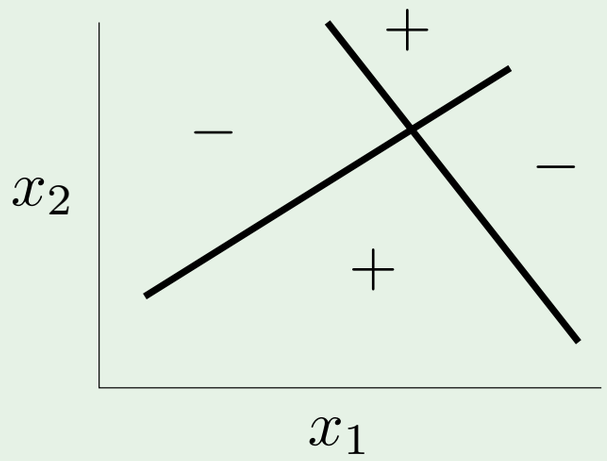
Biological inspiration

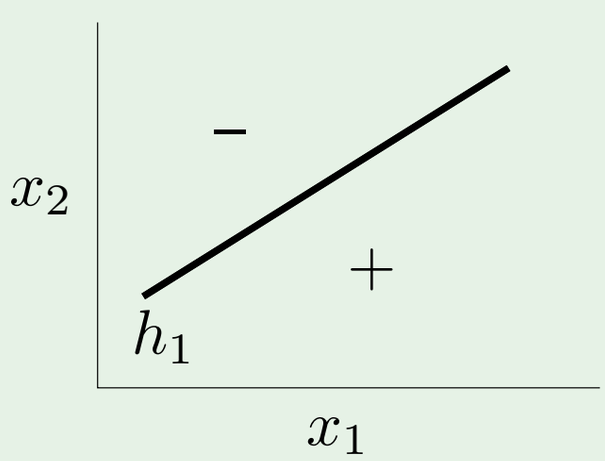
biological function $\longrightarrow$ biological structure

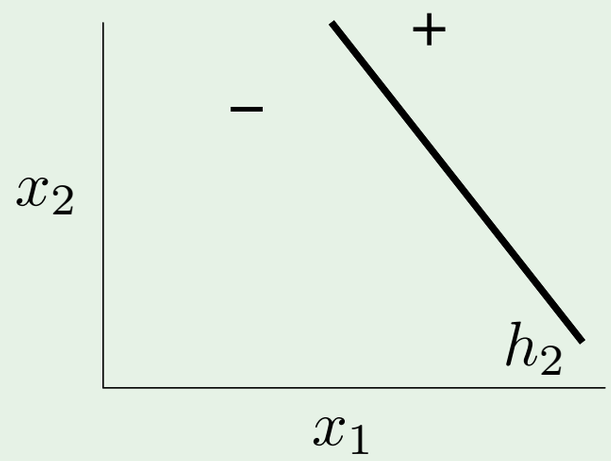


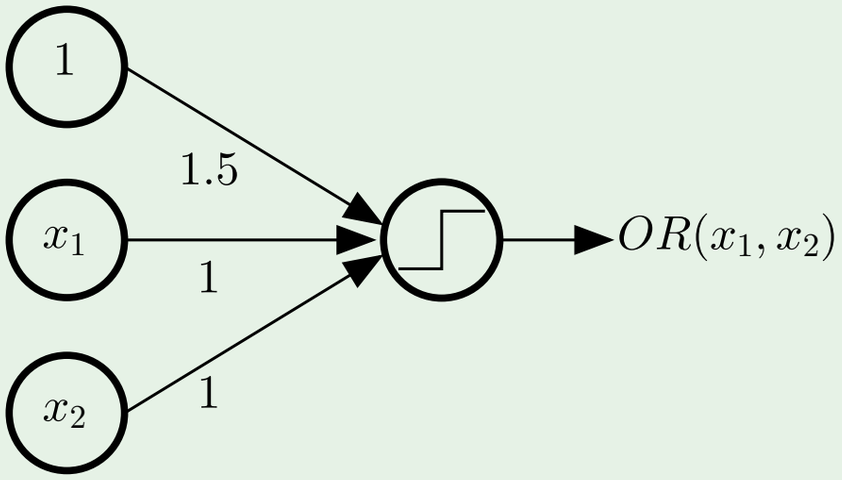


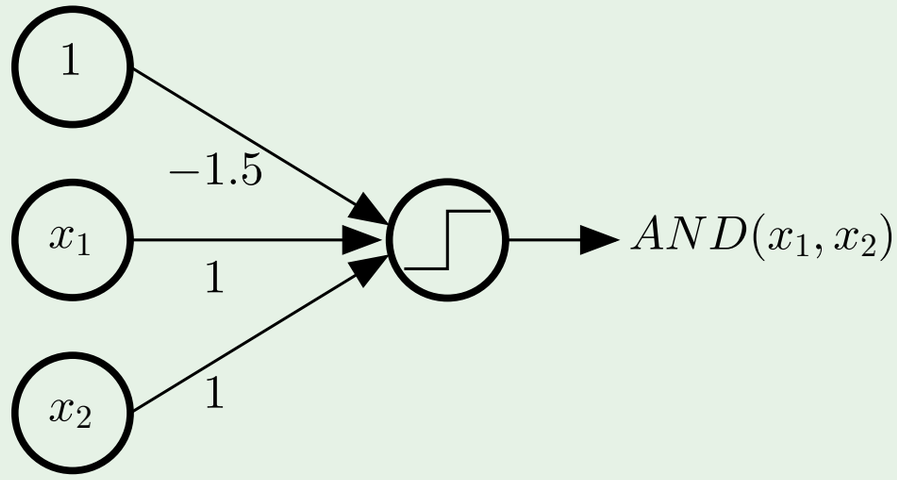
Combining perceptrons



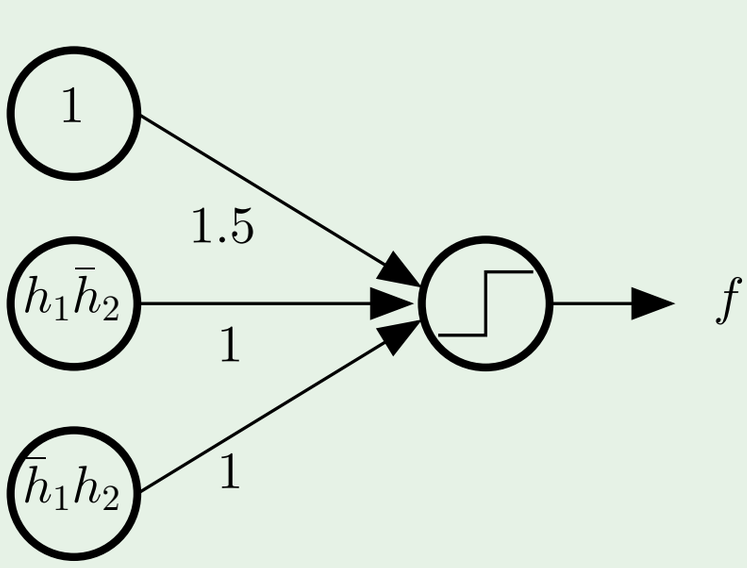


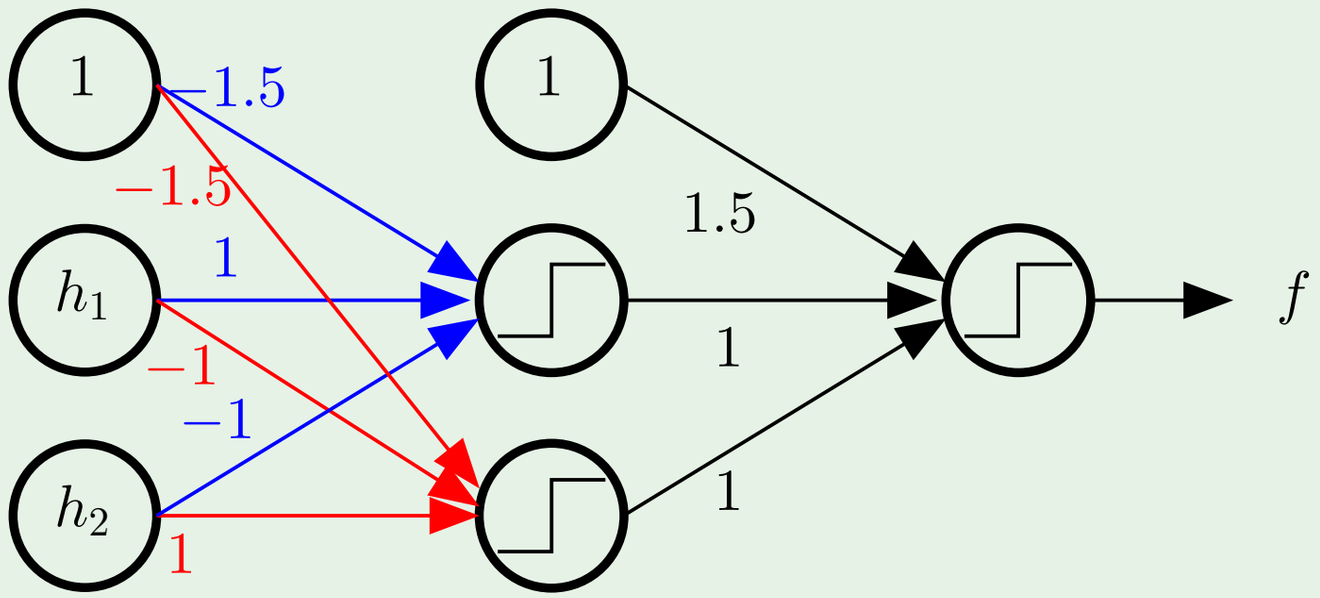




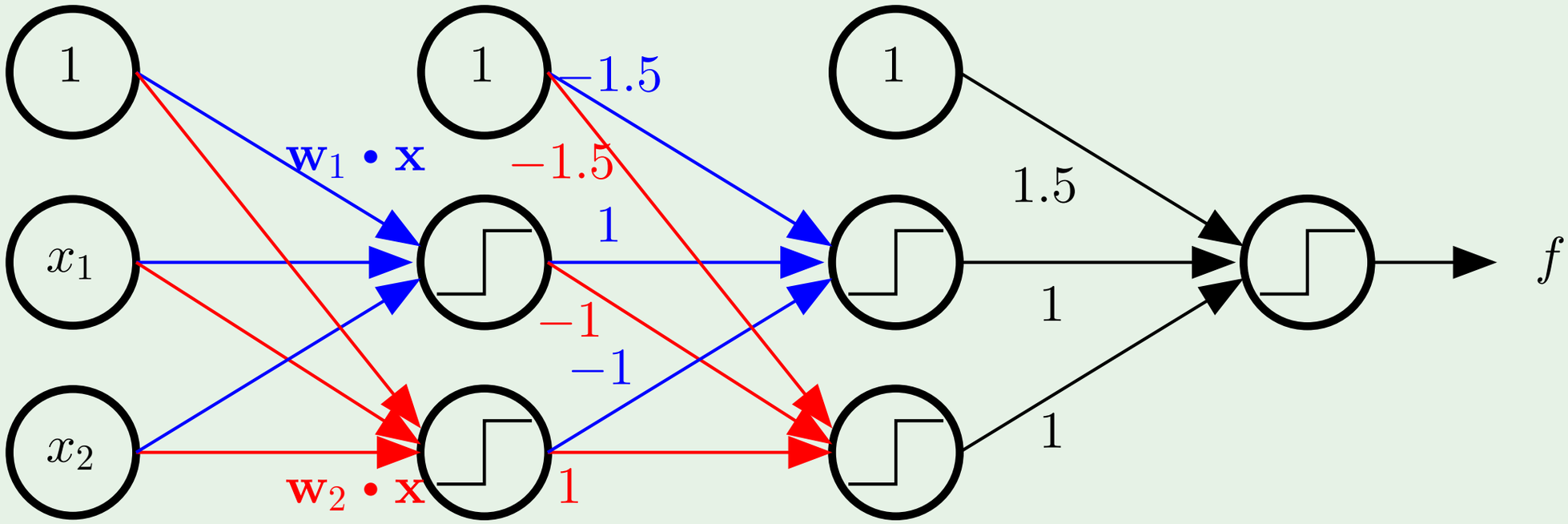


Creating layers



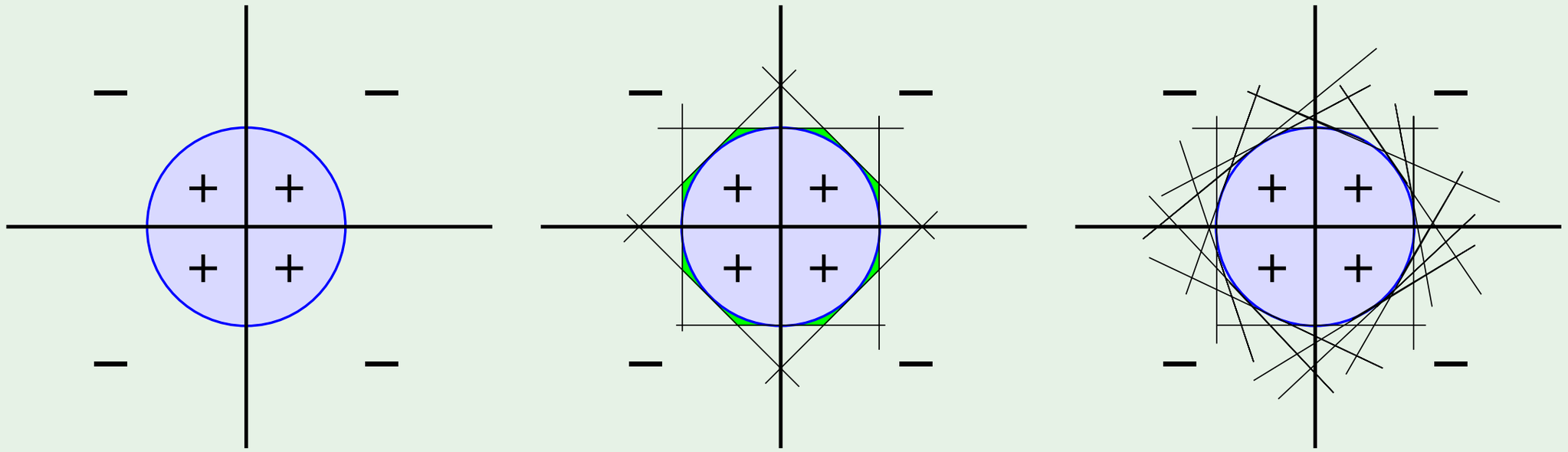


The multilayer perceptron



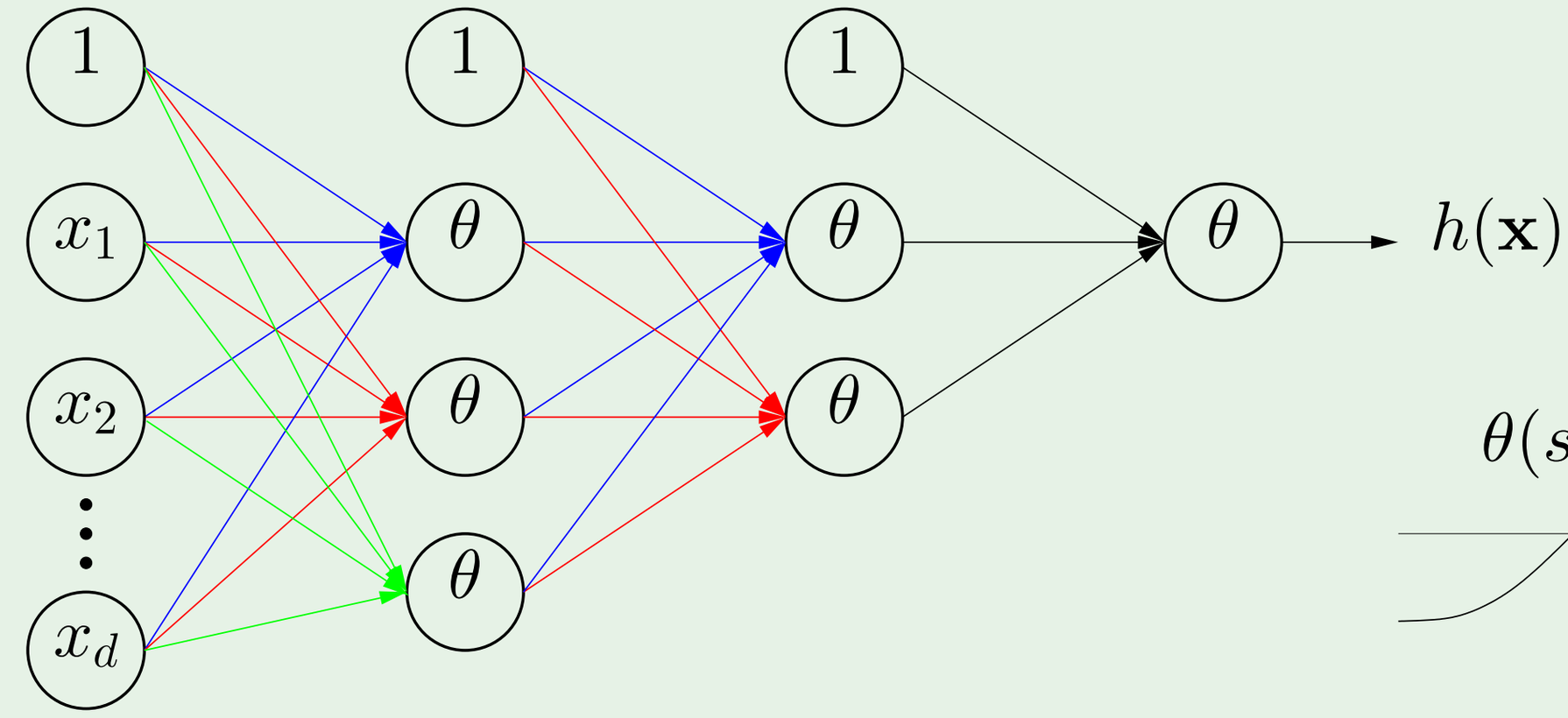
3 layers “feedforward”

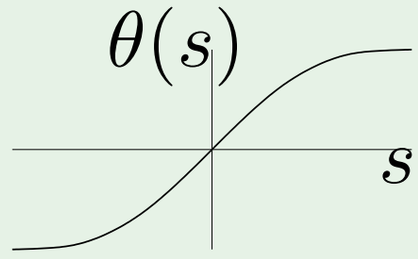
A powerful model



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

The neural network





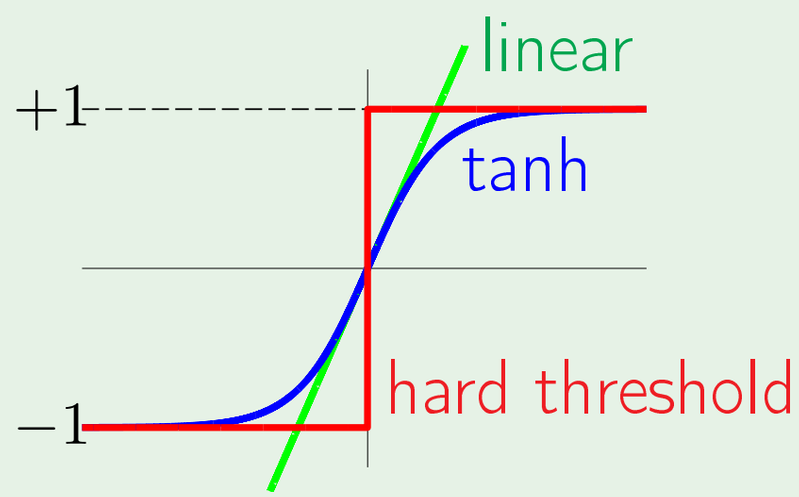
input $\mathbf{x}$ hidden layers $1 \leq l < L$ output layer $l = L$

How the network operates

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

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

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



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

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

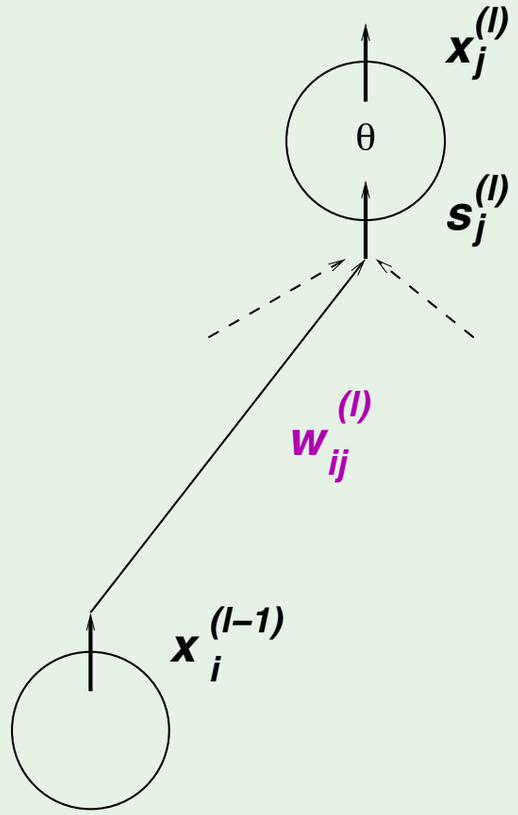
$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

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

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

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

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 $j = 1$:

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

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

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

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

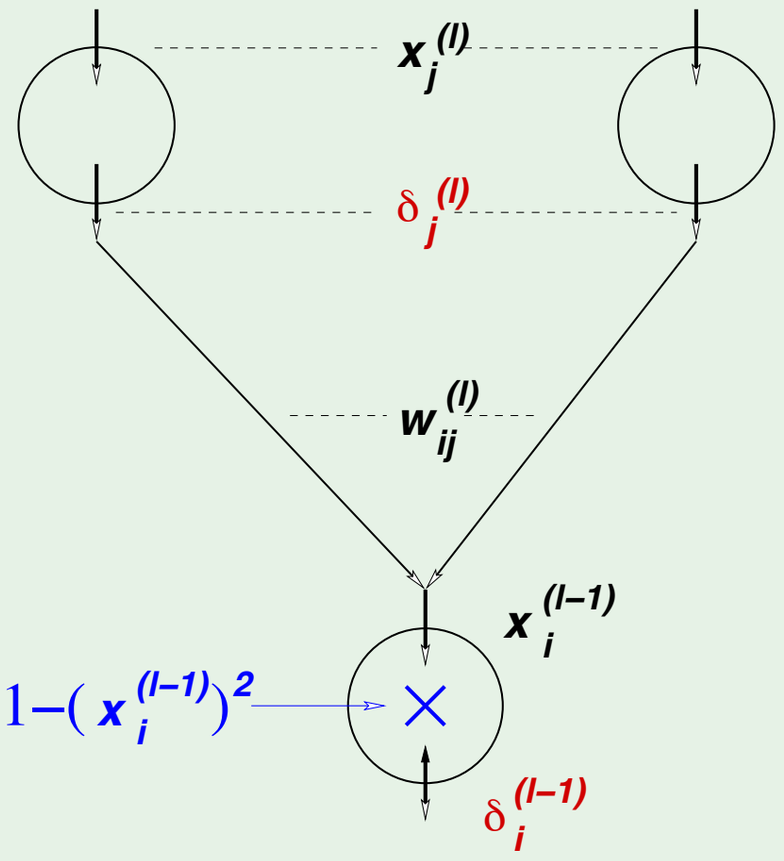
Back propagation of δ

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

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

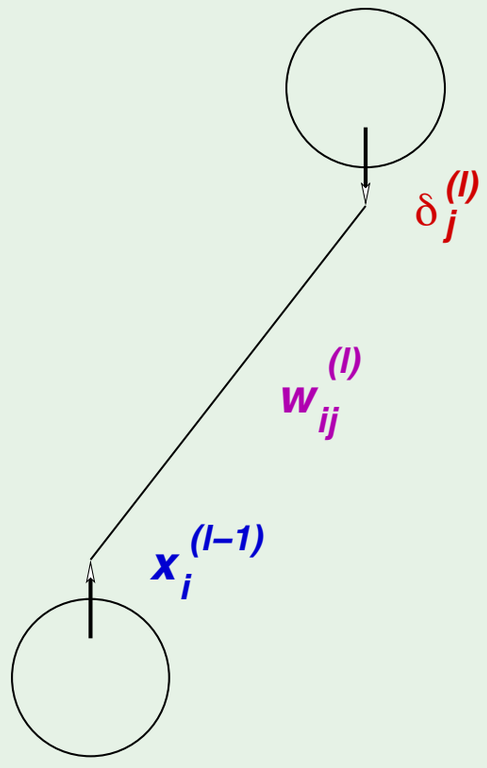
$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

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



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



Final remark: hidden layers

learned nonlinear transform

interpretation?