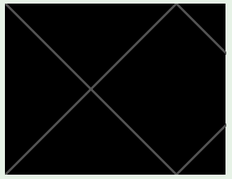
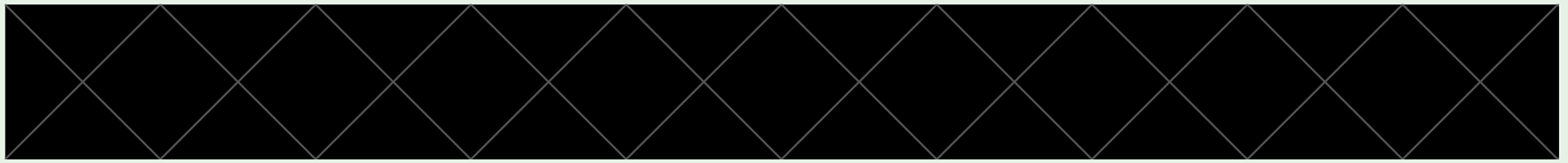
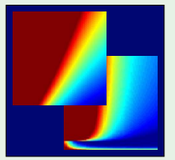
Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology



The Learning Problem





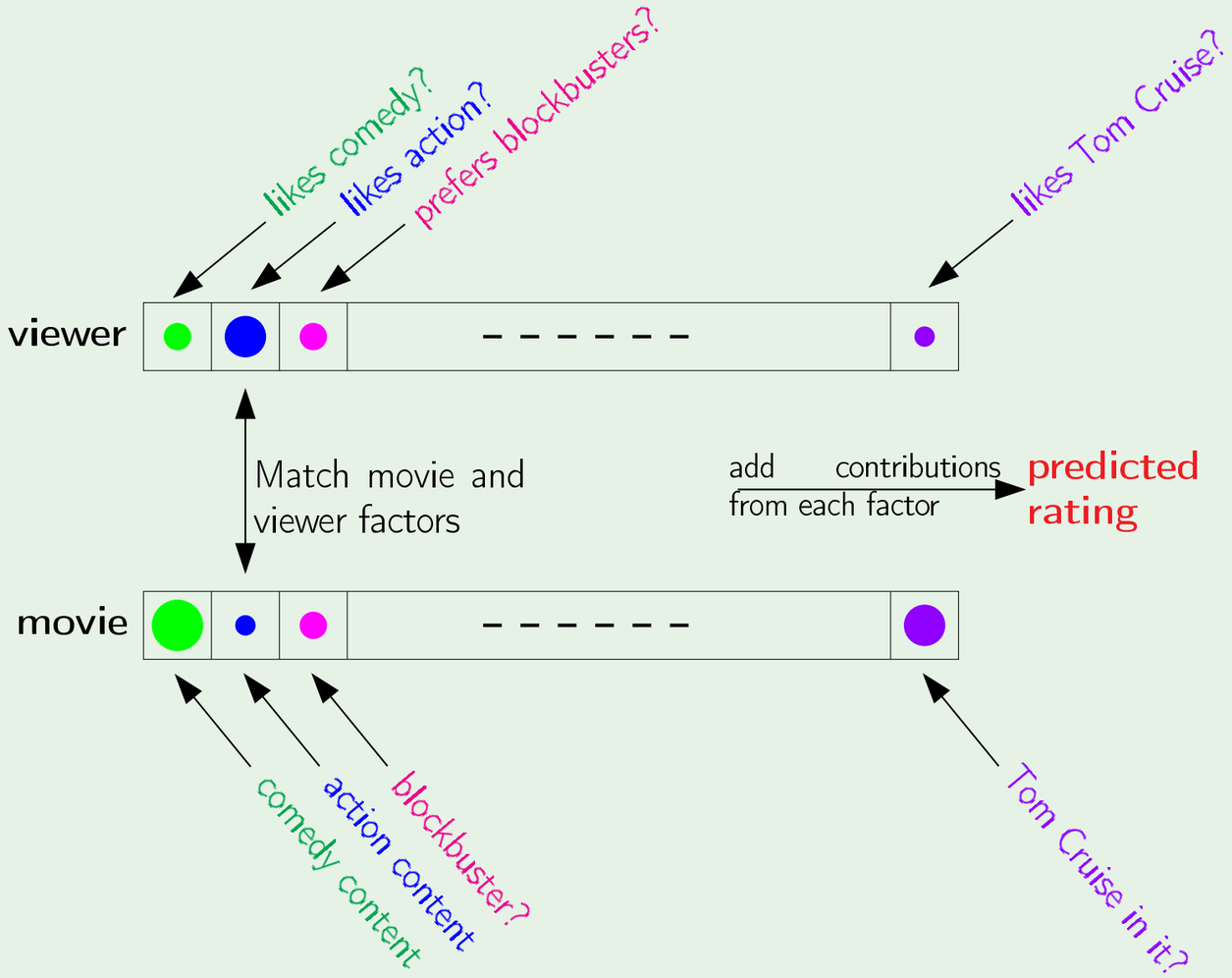
Example: Predicting how a viewer will rate a movie

10% improvement = 1 million dollar prize

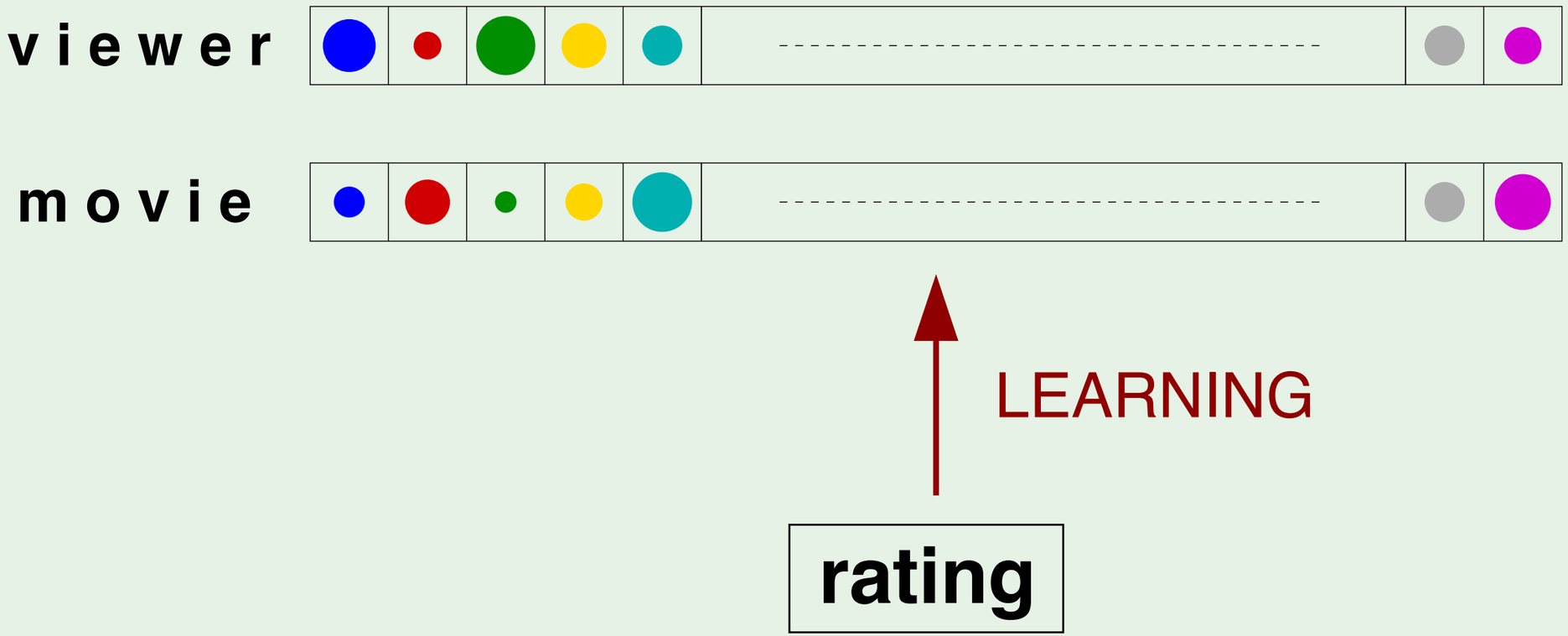
The essence of machine learning:

- A pattern exists.
- We cannot pin it down mathematically.
- We have data on it.

Movie rating - a solution



The learning approach



Components of learning

Metaphor: Credit approval

Applicant information:

age	23 years
gender	male
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000
...	...

Approve credit?

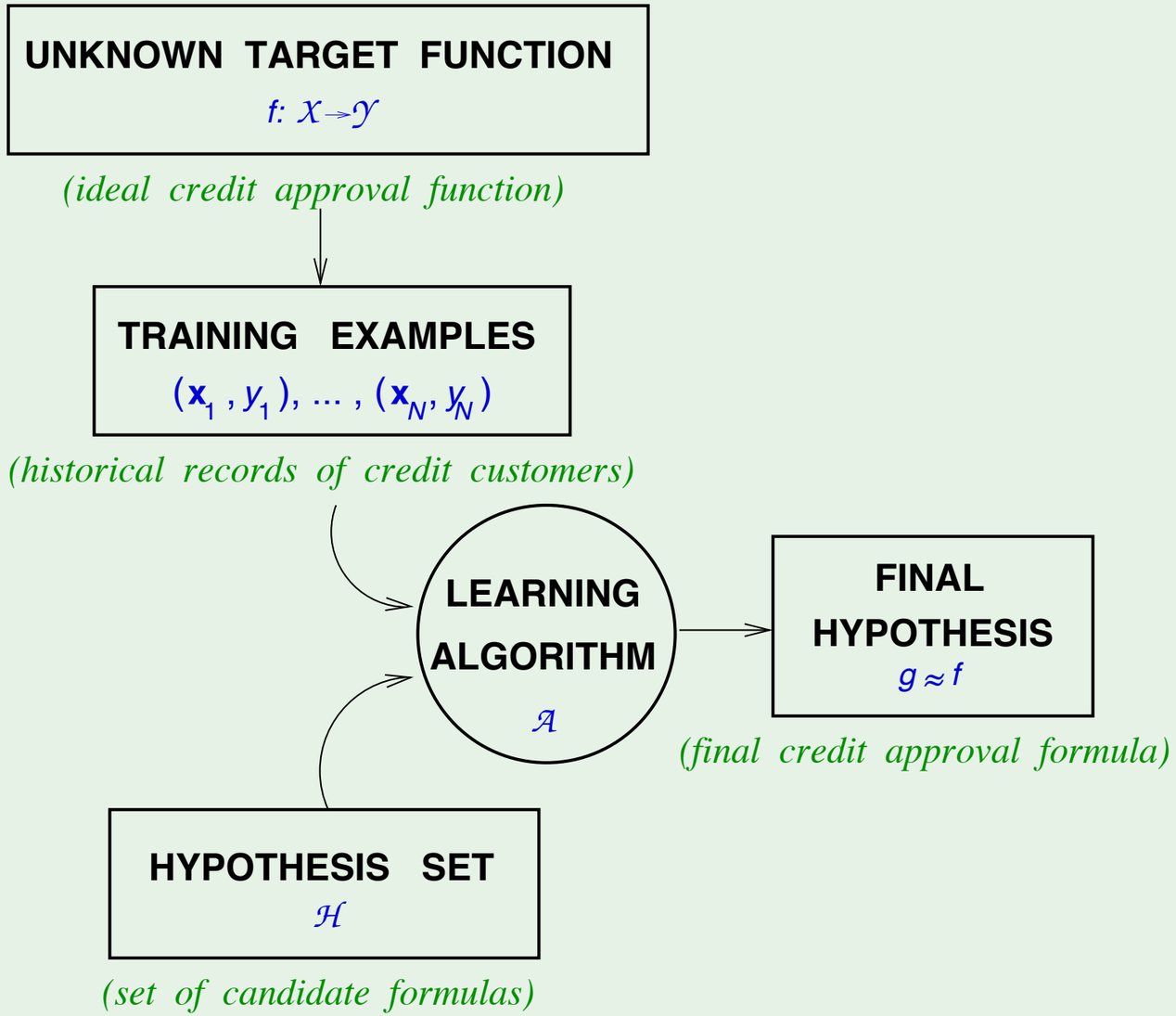
Components of learning

Formalization:

- Input: $\mathbf{x}$ (*customer application*)
- Output: y (*good/bad customer?*)
- Target function: $f : \mathcal{X} \to \mathcal{Y}$ (*ideal credit approval formula*)
- Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$ (*historical records*)

$\downarrow \quad \downarrow \quad \downarrow$

- Hypothesis: $g : \mathcal{X} \to \mathcal{Y}$ (*formula to be used*)


UNKNOWN TARGET FUNCTION
$f: \mathcal{X} \to \mathcal{Y}$
(ideal credit approval function)
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
(historical records of credit customers)
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
(final credit approval formula)
HYPOTHESIS SET
$\mathcal{H}$
(set of candidate formulas)

Solution components

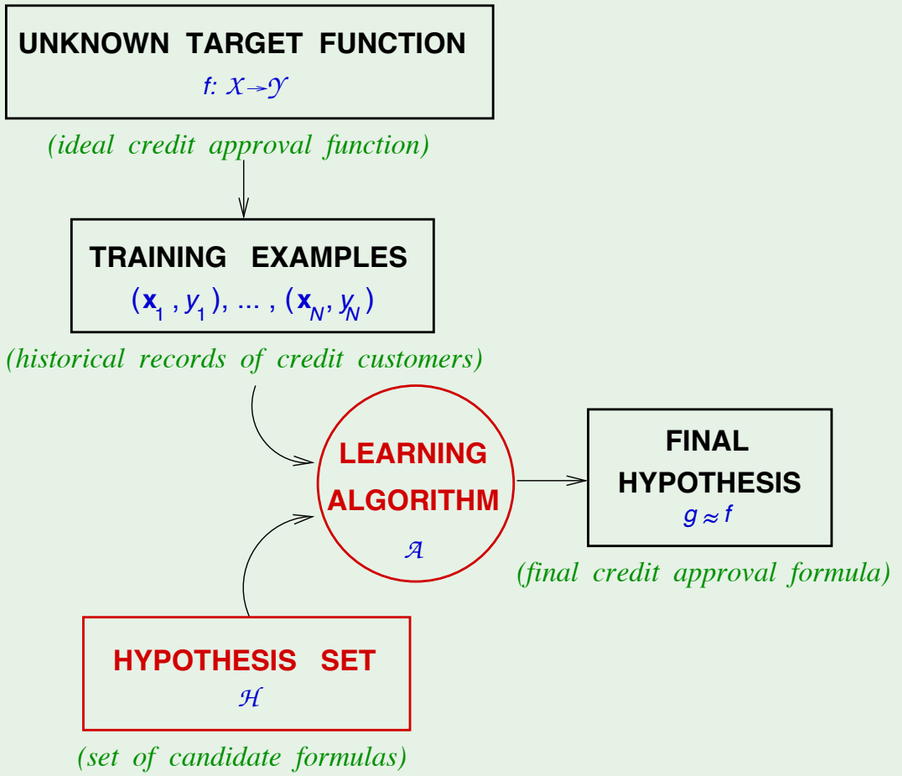
The 2 solution components of the learning problem:

- The Hypothesis Set

  $\mathcal{H} = \{h\}$ $\qquad$ $g \in \mathcal{H}$

- The Learning Algorithm

Together, they are referred to as the *learning model*.



A simple hypothesis set - the 'perceptron'

For input $\mathbf{x} = (x_1, \cdots, x_d)$ 'attributes of a customer'

$$\text{Approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold},$$

$$\text{Deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}.$$

This linear formula $h \in \mathcal{H}$ can be written as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

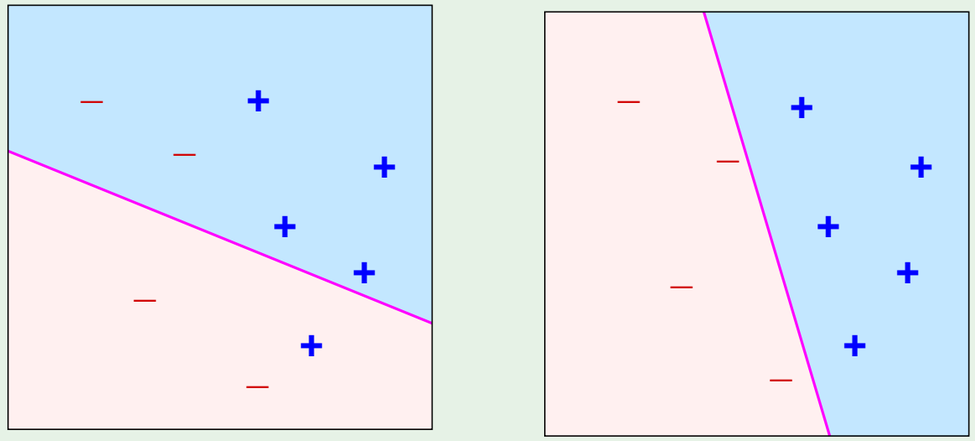
$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i\, x_i\right) + \quad w_0 \quad\right)$$

Introduce an artificial coordinate $x_0 = 1$:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^{d} w_i\, x_i\right)$$

In vector form, the perceptron implements

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x})$$



'linearly separable' data

A simple learning algorithm - PLA

The perceptron implements

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x})$$

Given the training set:

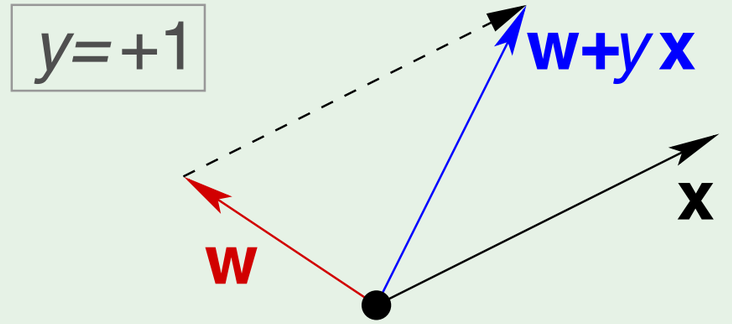
$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

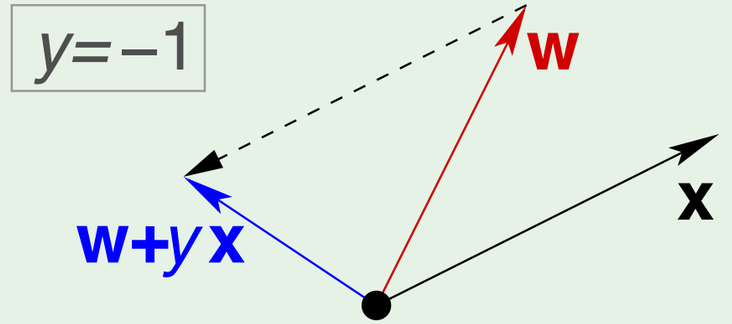
pick a misclassified point:

$$\text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n) \neq y_n$$

and update the weight vector:

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} + y_n\mathbf{x}_n}$$





Iterations of PLA

- One iteration of the PLA:

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$$

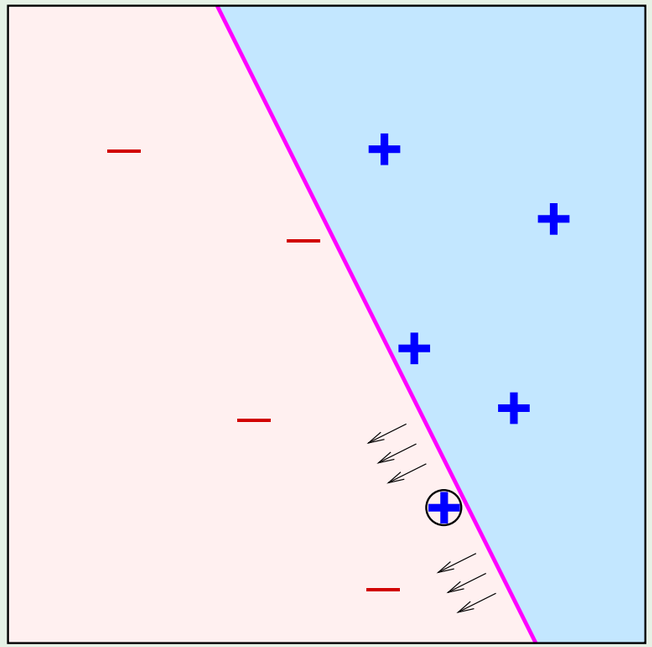
where $(\mathbf{x}, y)$ is a misclassified training point.

- At iteration $t = 1, 2, 3, \cdots$, pick a misclassified point from

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

and run a PLA iteration on it.

- That's it!



Basic premise of learning

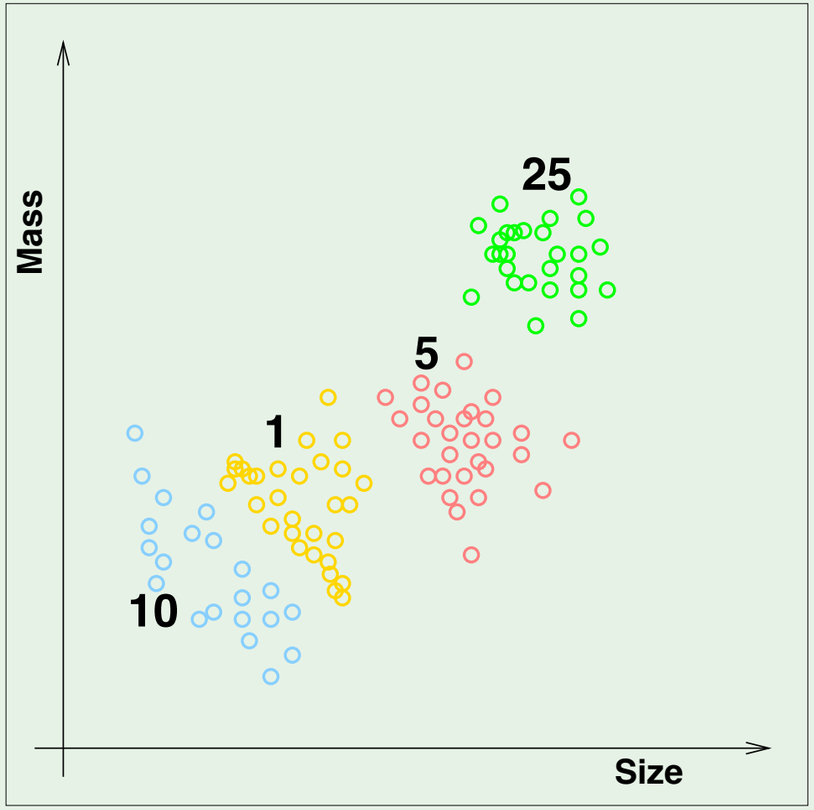
"using a set of observations to uncover an underlying process"

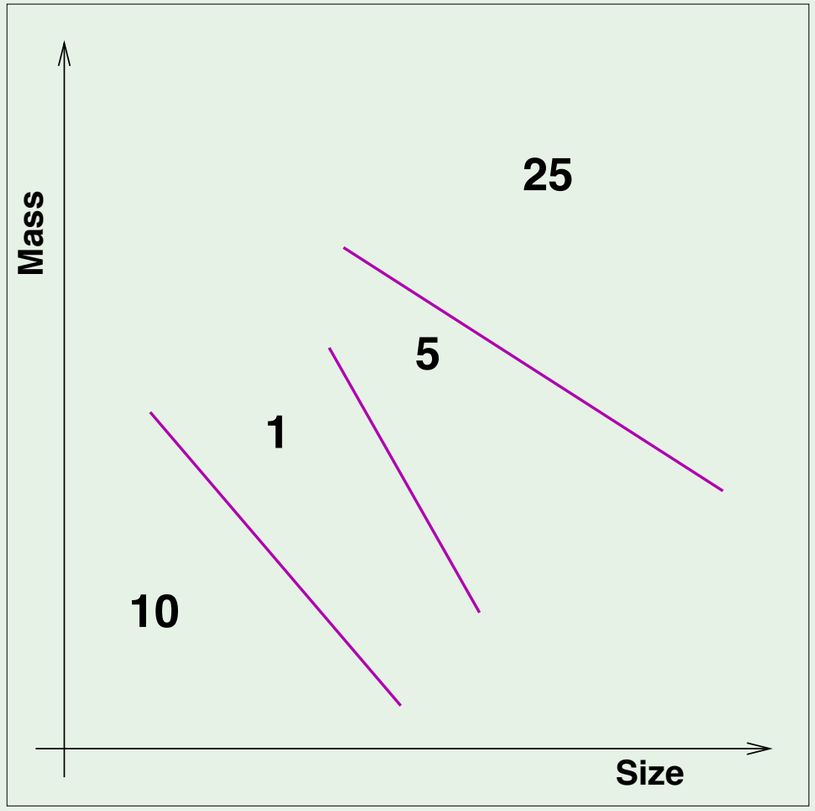
broad premise $\Longrightarrow$ many variations

- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**

Supervised learning

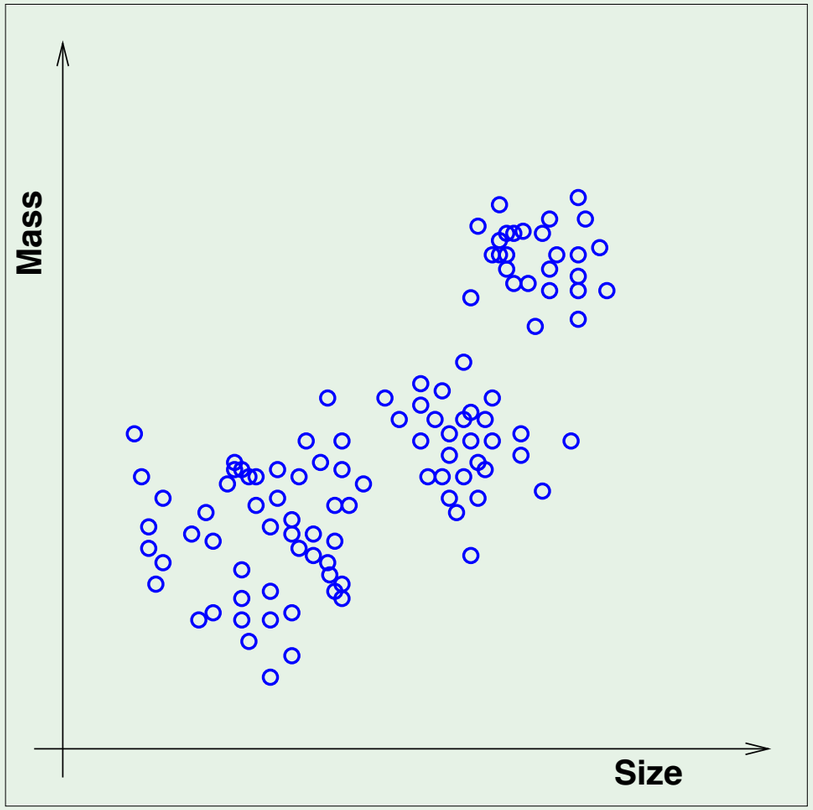
Example from vending machines – **coin recognition**





Unsupervised learning

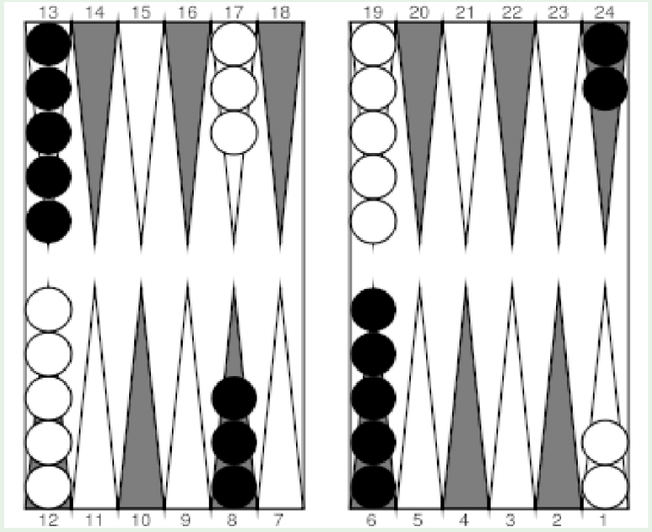
Instead of (input,correct output), we get (input, ?)



Reinforcement learning

Instead of (input,correct output),

we get (input,*some* output,grade for this output)

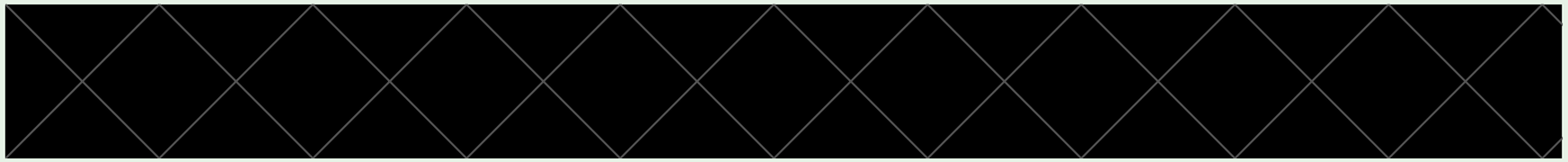


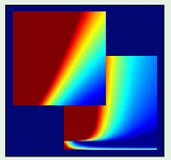
The world champion was a neural network!

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Linear Models I





A real data set

Input representation

'raw' input $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_{256})$

linear model: $(w_0, w_1, w_2, \cdots, w_{256})$

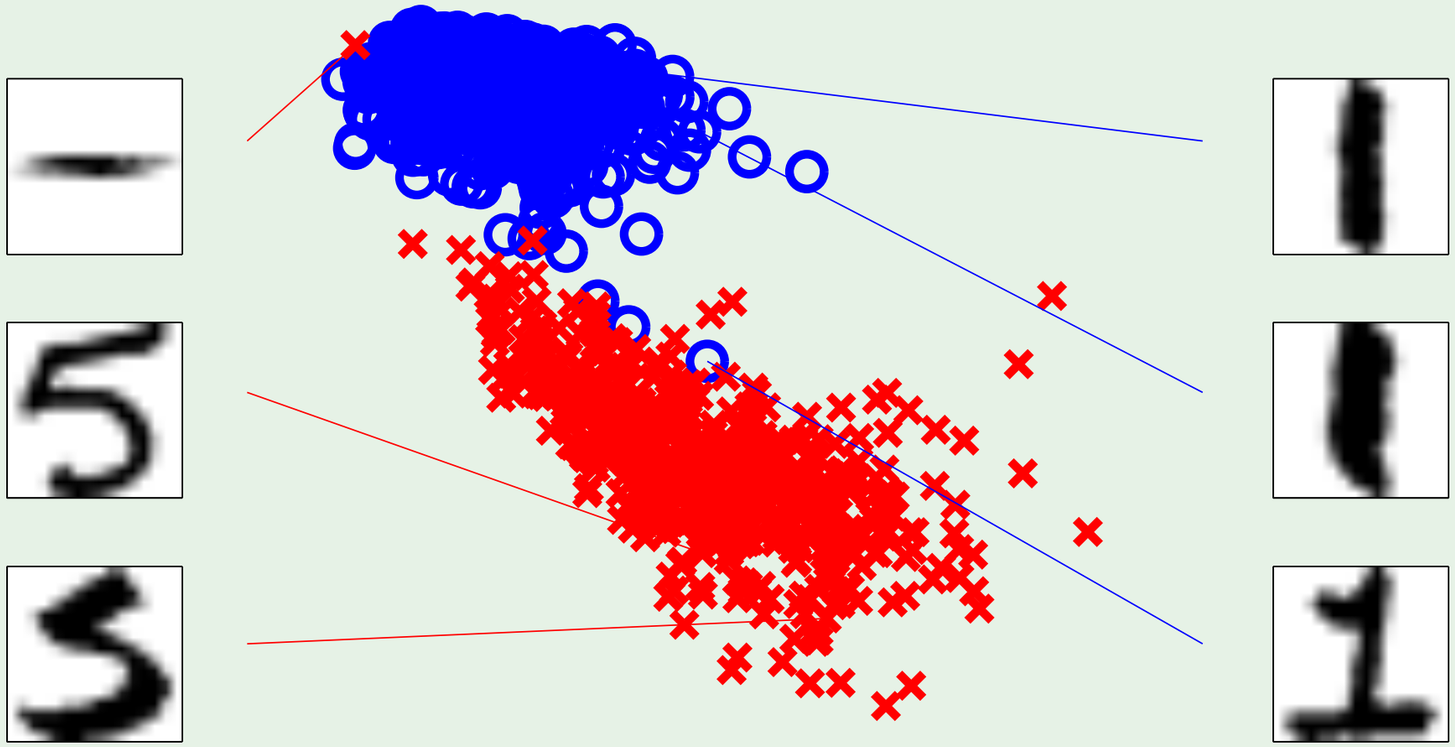
Features: Extract useful information, e.g.,

intensity and symmetry $\mathbf{x} = (x_0, x_1, x_2)$

linear model: (w_0, w_1, w_2)

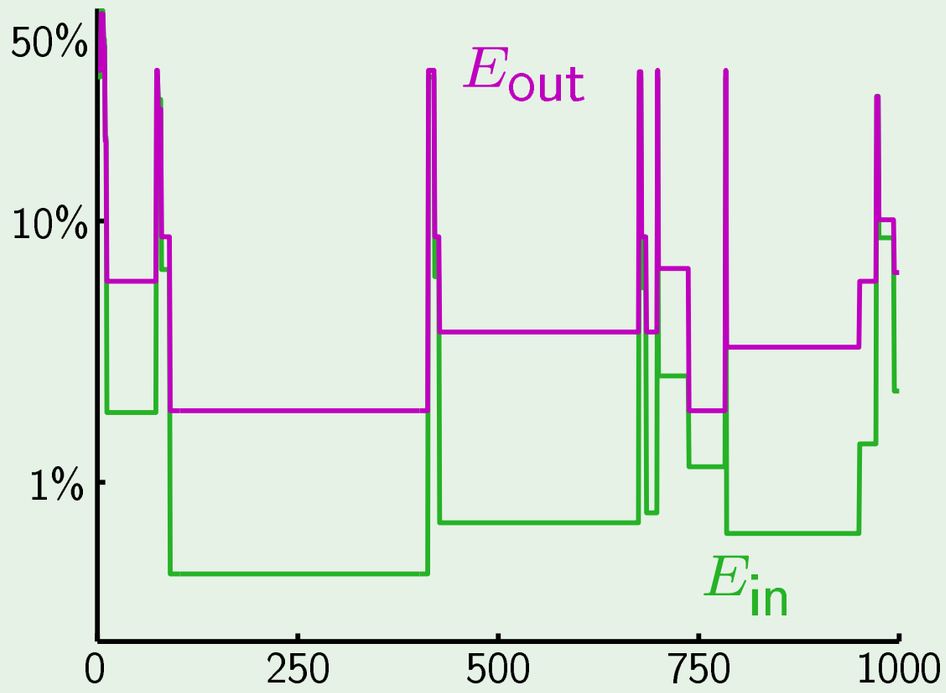
Illustration of features

$\mathbf{x} = (x_0, x_1, x_2)$ x_1: intensity x_2: symmetry

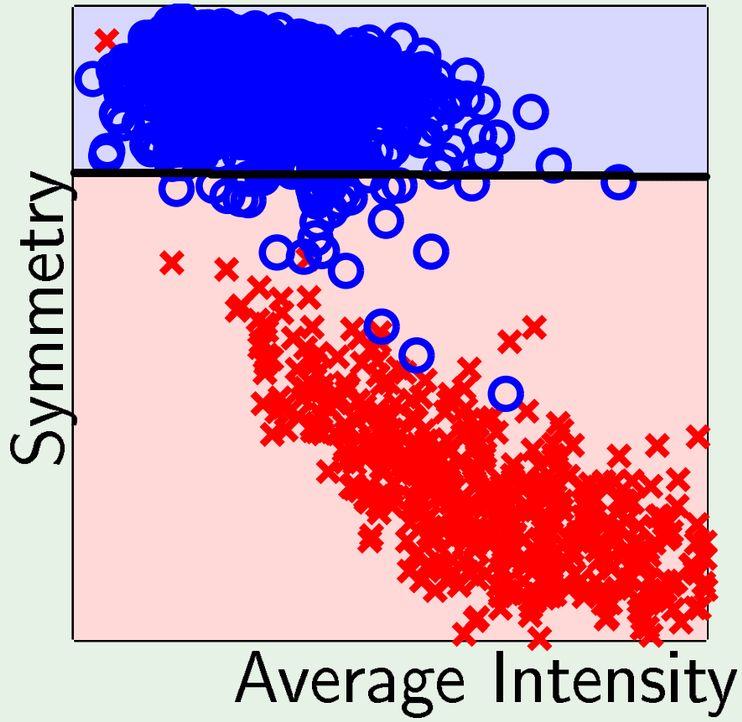


What PLA does

Evolution of E_{in} and E_{out}

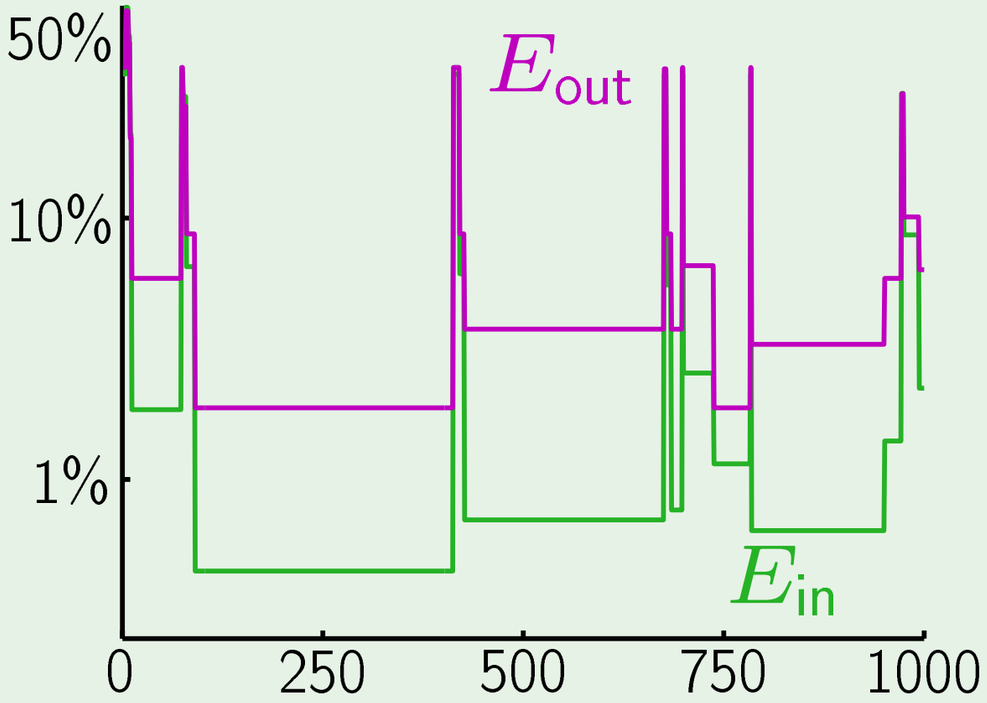


Final perceptron boundary

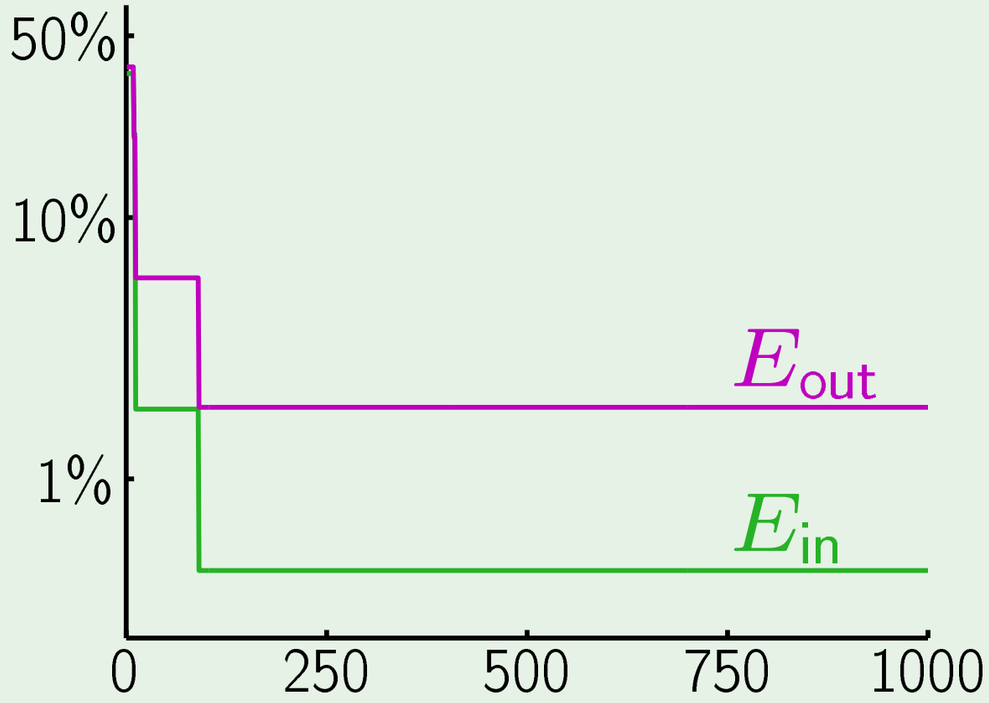


The ‘pocket’ algorithm

PLA:

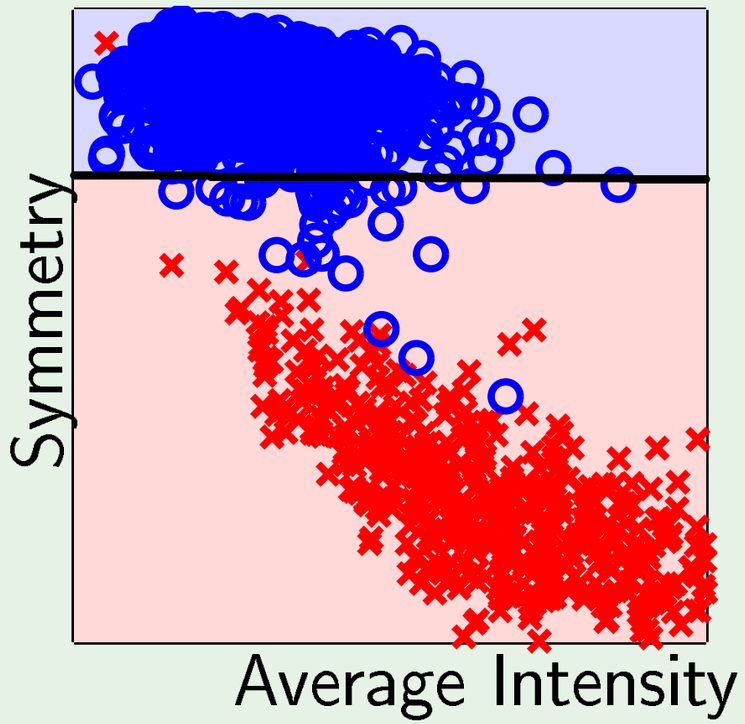


Pocket:

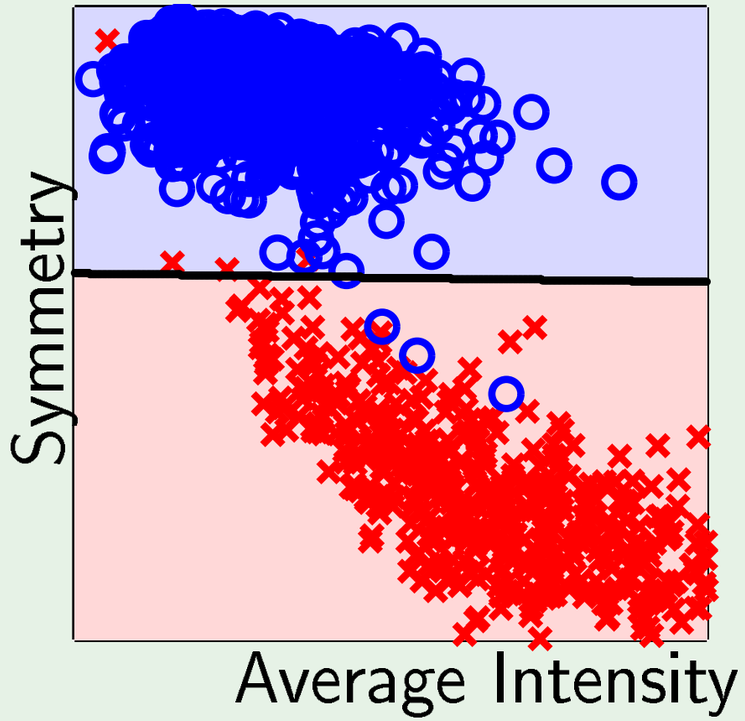


Classification boundary - PLA versus Pocket

PLA:



Pocket:



Credit again

Classification: Credit approval (yes/no)

Regression: Credit line (dollar amount)

Input: $\mathbf{x} =$

age	23 years
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000
...	...

Linear regression output: $h(\mathbf{x}) = \sum_{i=0}^{d} w_i \, x_i = \mathbf{w}^\mathsf{T}\mathbf{x}$

The data set

Credit officers decide on credit lines:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

$y_n \in \mathbb{R}$ is the credit line for customer $\mathbf{x}_n$.

Linear regression tries to replicate that.

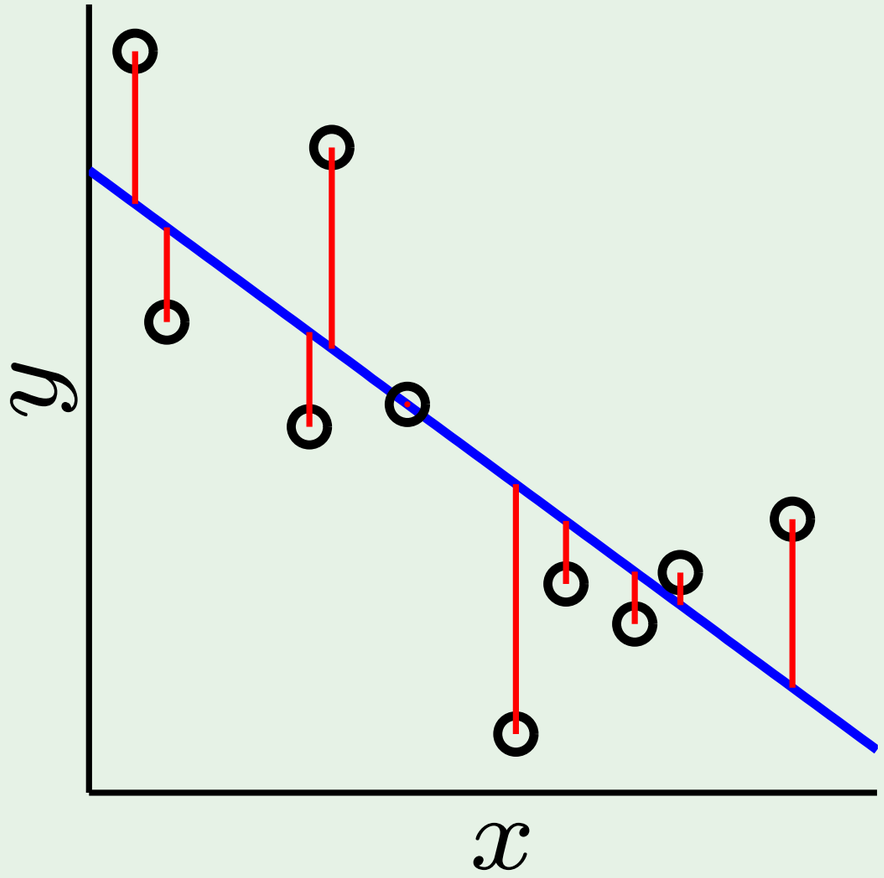
How to measure the error

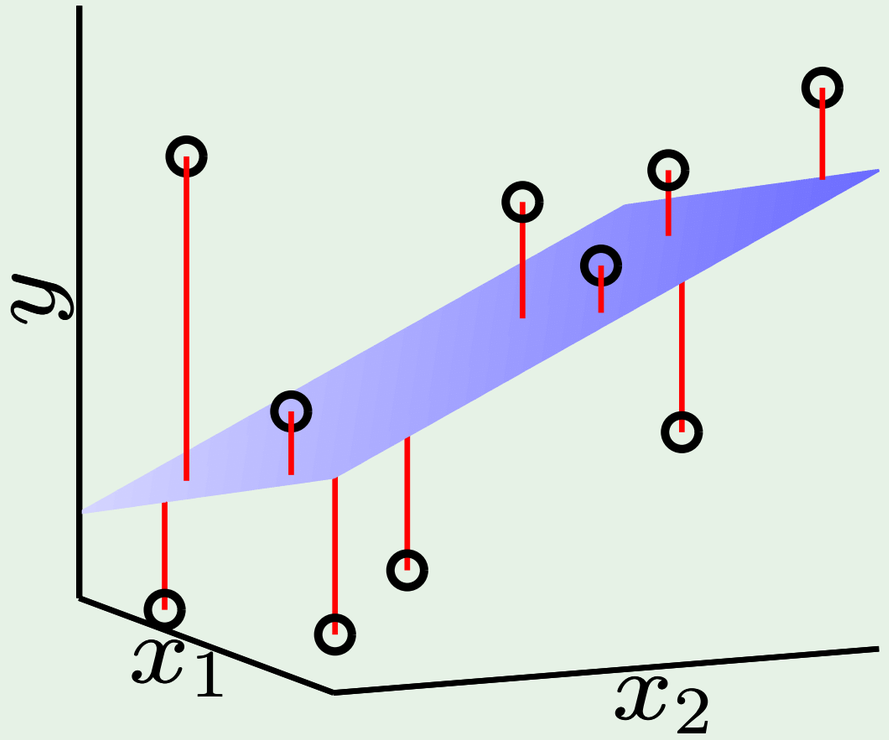
How well does $h(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x}$ approximate $f(\mathbf{x})$?

In linear regression, we use squared error $(h(\mathbf{x}) - f(\mathbf{x}))^2$

$$\textbf{in-sample error: } E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}\left(h(\mathbf{x}_n) - y_n\right)^2$$

Illustration of linear regression





The expression for E_{in}

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

$$= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

where $\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1{}^{\mathsf{T}}— \\ —\mathbf{x}_2{}^{\mathsf{T}}— \\ \vdots \\ —\mathbf{x}_N{}^{\mathsf{T}}— \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

Minimizing E_{in}

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\mathsf{T}}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

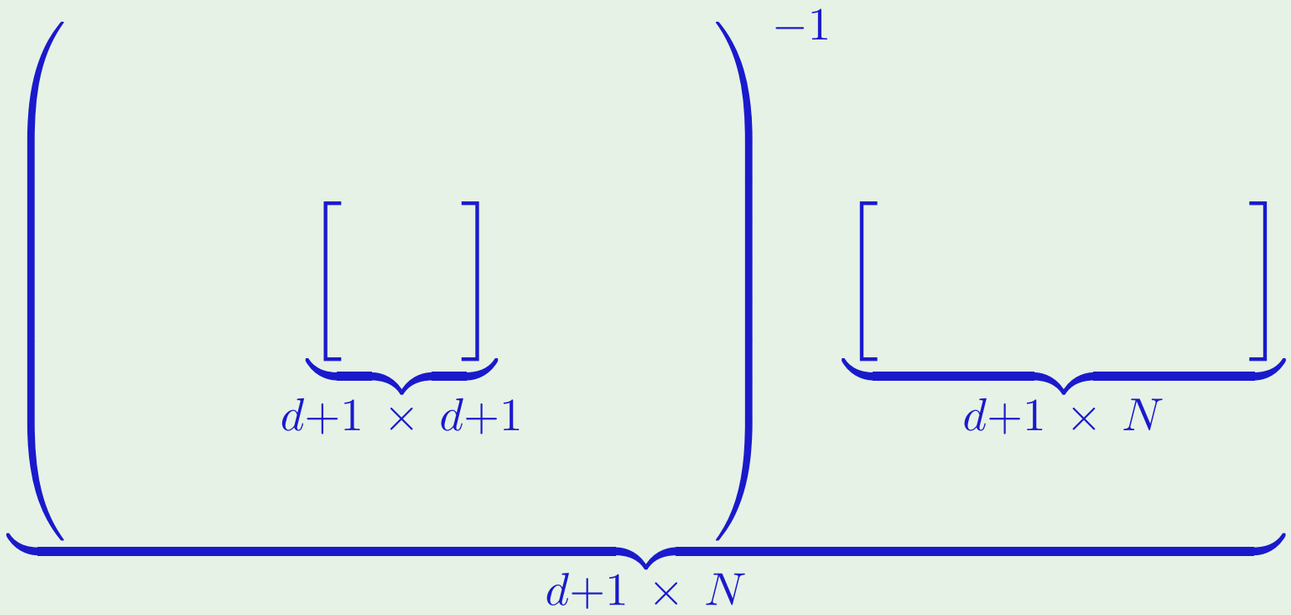
$$\mathrm{X}^{\mathsf{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathsf{T}}\mathbf{y}$$

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}$$

$\mathrm{X}^{\dagger}$ is the '**pseudo-inverse**' of X

The pseudo-inverse

$$\mathrm{X}^\dagger = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$$



The linear regression algorithm

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} \text{—}\mathbf{x}_1^\mathsf{T}\text{—} \\ \text{—}\mathbf{x}_2^\mathsf{T}\text{—} \\ \vdots \\ \text{—}\mathbf{x}_N^\mathsf{T}\text{—} \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathrm{X}^\dagger = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$.

3: Return $\mathbf{w} = \mathrm{X}^\dagger\mathbf{y}$.

Linear regression for classification

Linear regression learns a real-valued function $y = f(\mathbf{x}) \in \mathbb{R}$

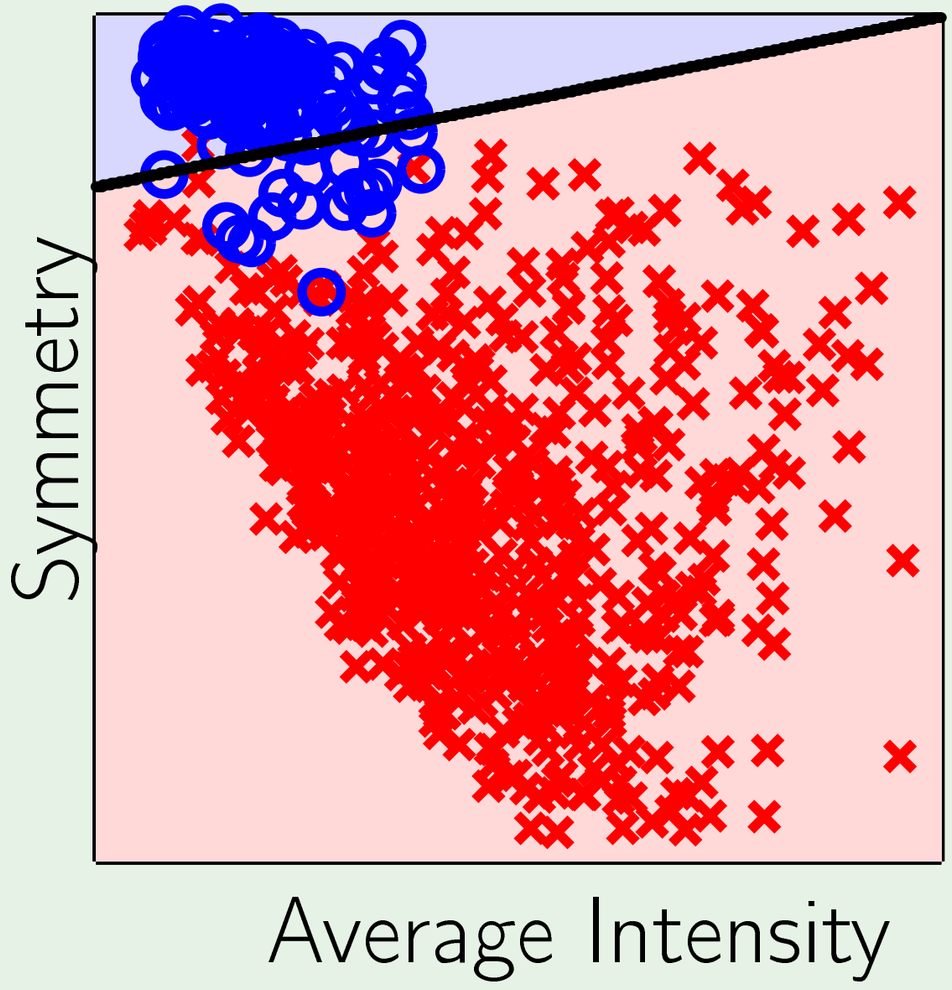
Binary-valued functions are also real-valued! $\pm 1 \in \mathbb{R}$

Use linear regression to get $\mathbf{w}$ where $\mathbf{w}^{\mathsf{T}}\mathbf{x}_n \approx y_n = \pm 1$

In this case, $\text{sign}\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n\right)$ is likely to agree with $y_n = \pm 1$

Good initial weights for classification

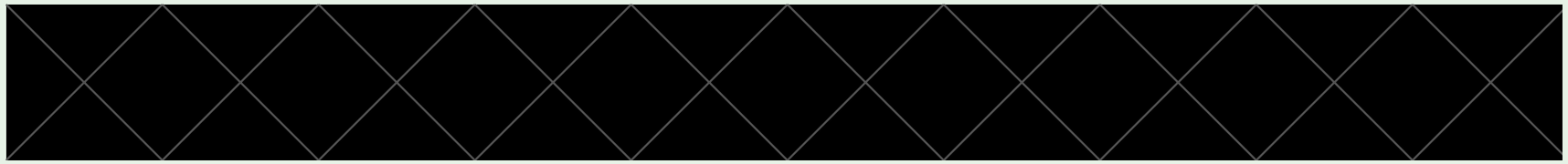
Linear regression boundary

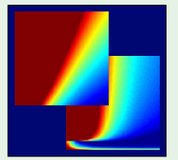


Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

The Linear Model II



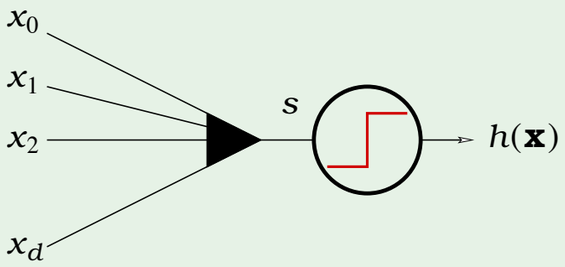


A third linear model

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

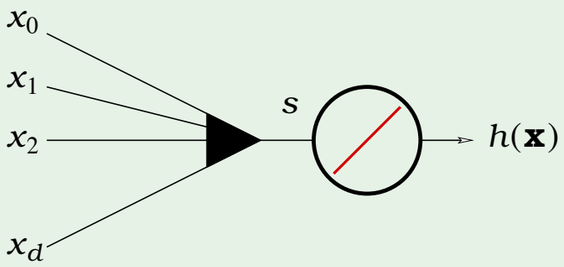
linear classification

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



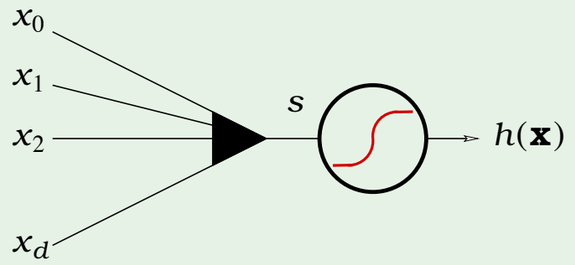
linear regression

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



logistic regression

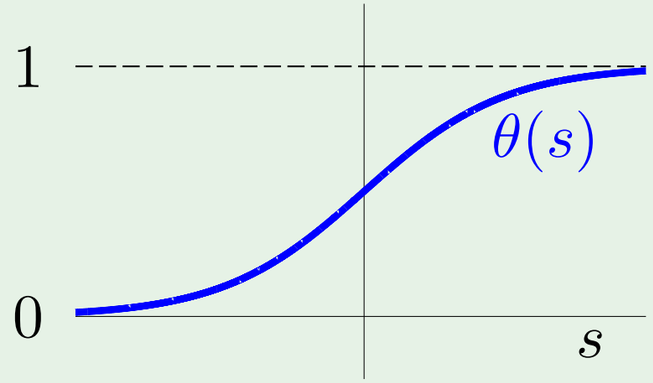
$$h(\mathbf{x}) = \theta(s)$$



The logistic function θ

The formula:

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



soft threshold: uncertainty

sigmoid: flattened out 's'

Probability interpretation

$h(\mathbf{x}) = \theta(s)$ is interpreted as a probability

Example. Prediction of heart attacks

Input $\mathbf{x}$: cholesterol level, age, weight, etc.

$\theta(s)$: probability of a heart attack

The signal $s = \mathbf{w}^{\mathsf{T}}\mathbf{x}$ "risk score"

Genuine probability

Data $(\mathbf{x}, y)$ with binary y, generated by a noisy target:

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

The target $f : \mathbb{R}^d \to [0, 1]$ is the probability

Learn $g(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T} \mathbf{x}) \approx f(\mathbf{x})$

Error measure

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

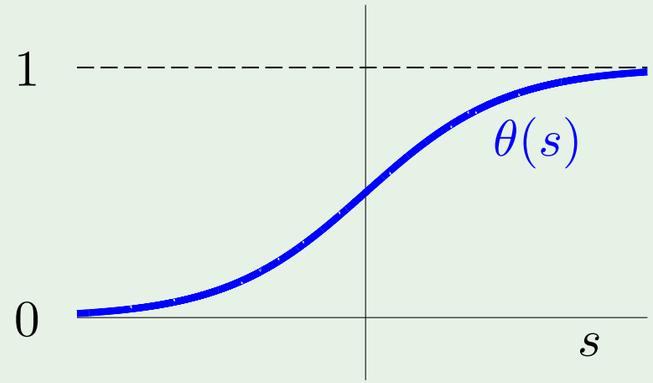
Plausible error measure based on **likelihood**:

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

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

$$\left[\theta(s)=\frac{1}{1\ +e^{-s}}\right]$$

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

"cross-entropy" error

How to minimize E_{in}

For logistic regression,

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

Compare to linear regression:

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

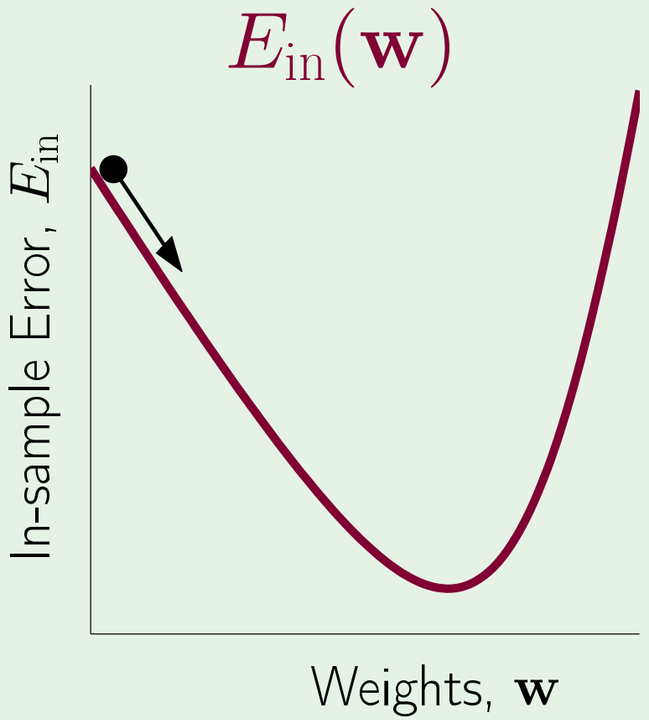
Iterative method: gradient descent

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 $\hat{\mathbf{v}}$

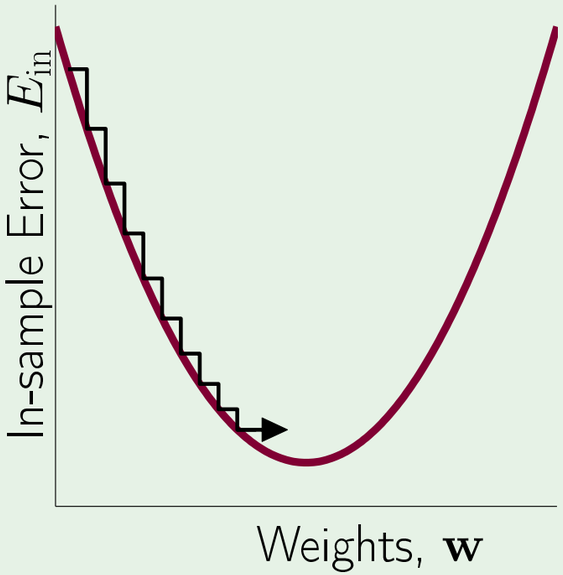
$$\begin{aligned}
\Delta E_{\text{in}} &= E_{\text{in}}(\,\mathbf{w}(0) + \eta\hat{\mathbf{v}}\,) - E_{\text{in}}(\mathbf{w}(0)) \\
&= \eta\nabla E_{\text{in}}(\mathbf{w}(0))^{\text{T}}\hat{\mathbf{v}} + O(\eta^2) \\
&\geq -\eta\|\nabla E_{\text{in}}(\mathbf{w}(0))\|
\end{aligned}$$

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

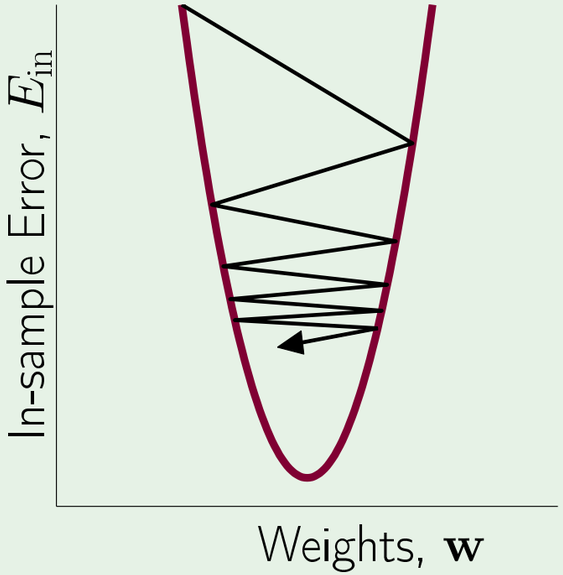
$$\hat{\mathbf{v}} = -\frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$

Fixed-size step?

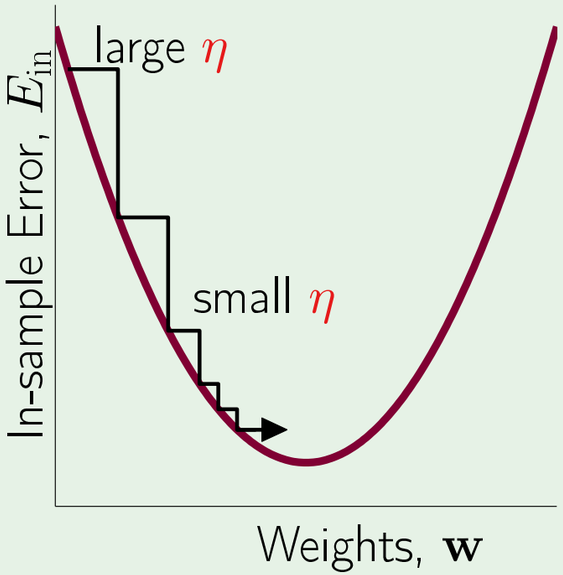
How η affects the algorithm:



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

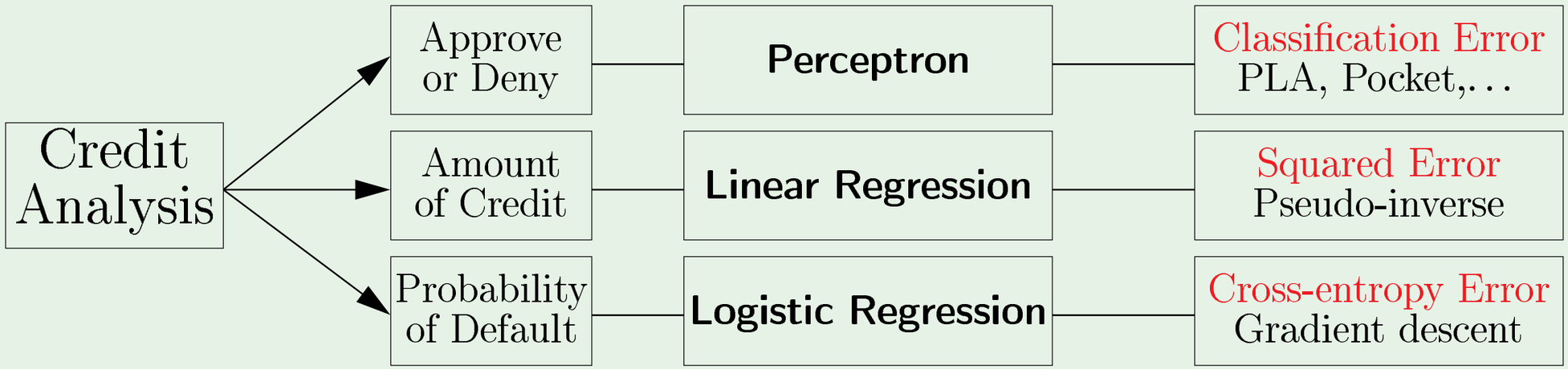
$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathsf{T}}(t) \mathbf{x}_n}}$$

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

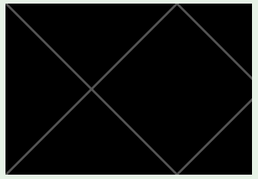
6: Return the final weights $\mathbf{w}$

Summary of Linear Models

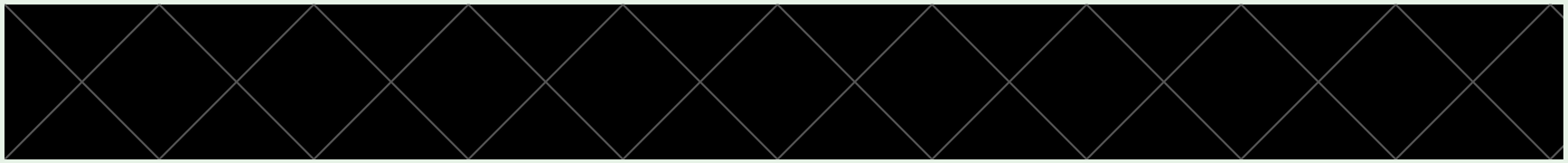


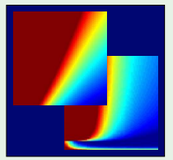
Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology



Neural Networks





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

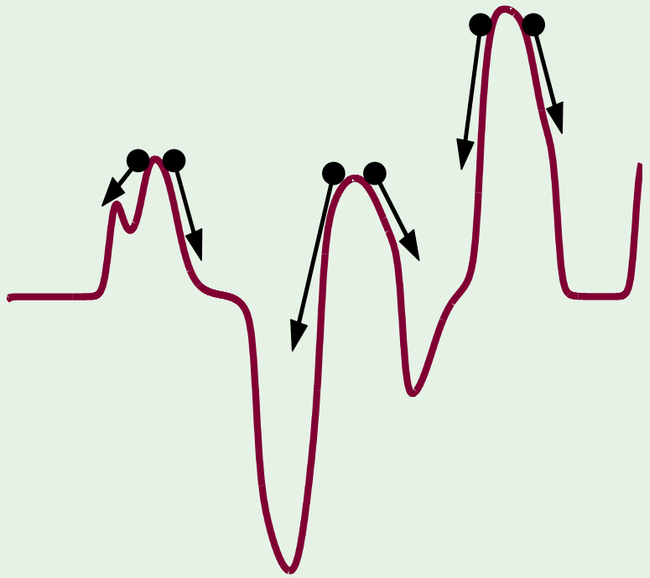
stochastic gradient descent (SGD)

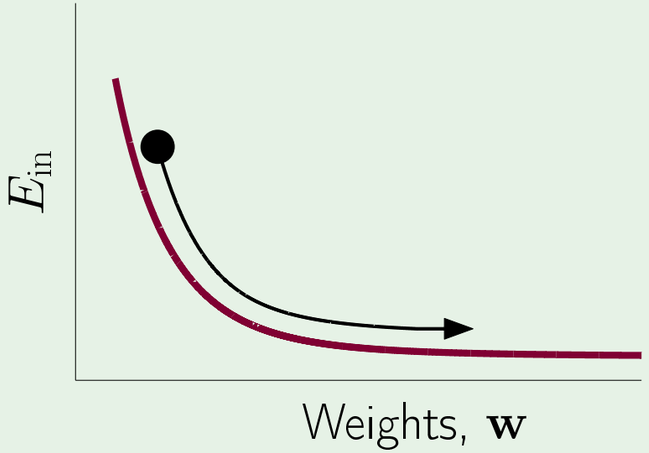
Benefits of SGD

1. cheaper computation
2. randomization
3. simple

Rule of thumb:

$\eta = 0.1$ works



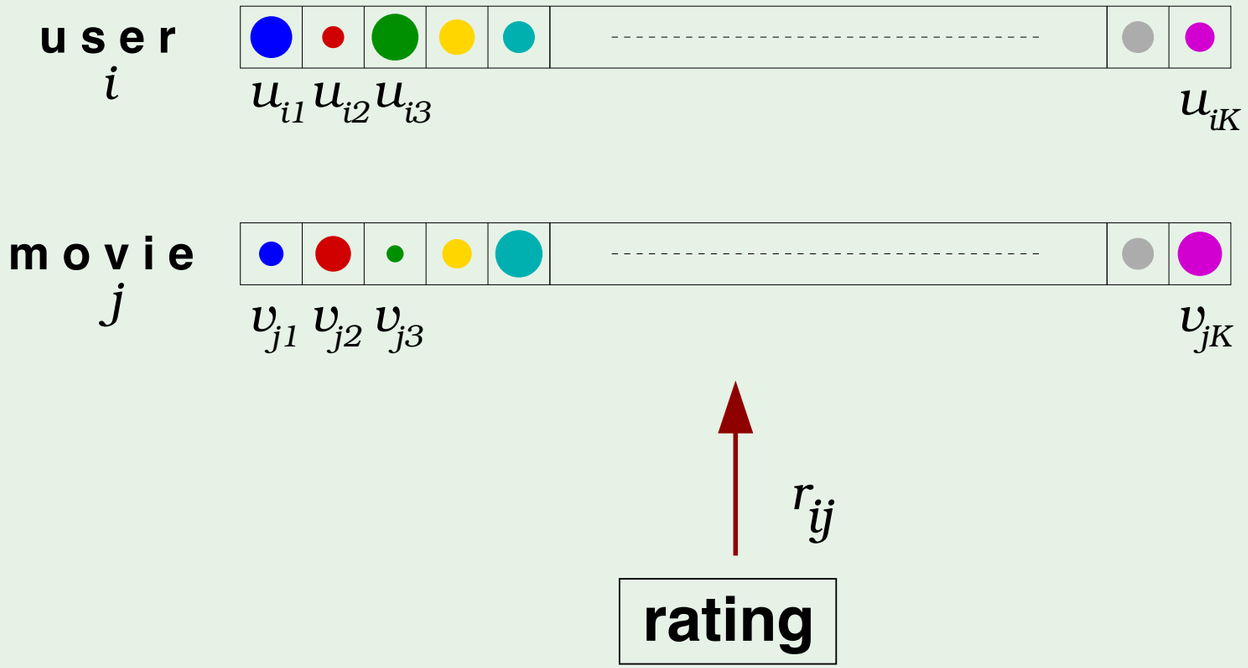


randomization helps

SGD in action

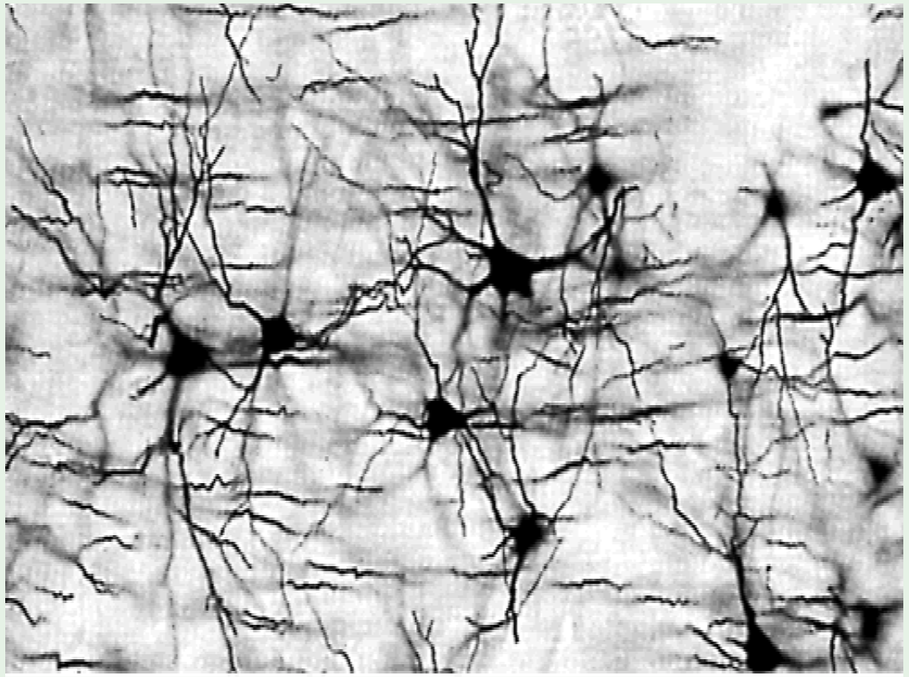
Remember movie ratings?

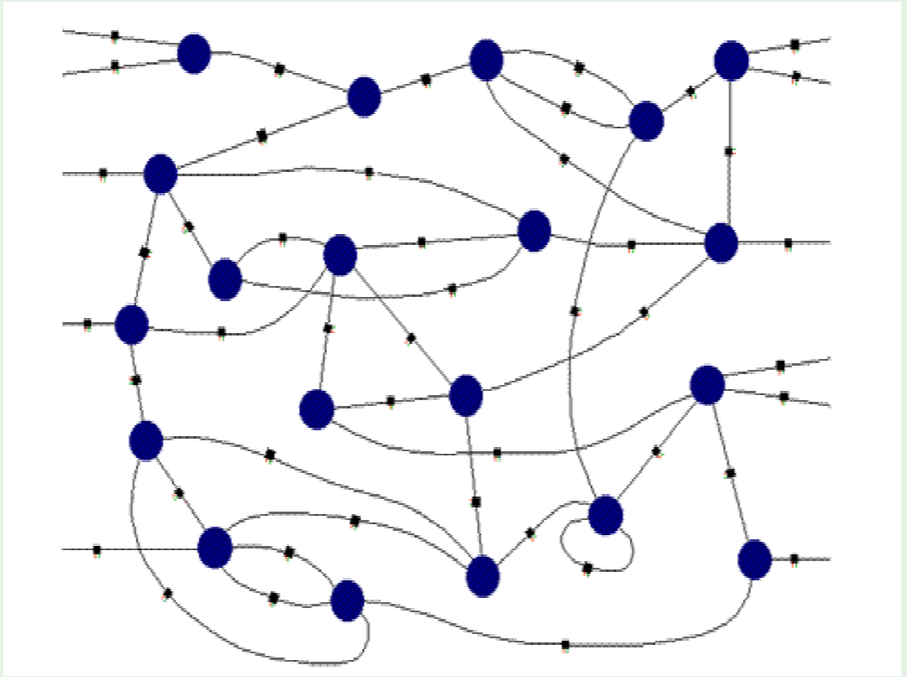
$$\mathbf{e}_{ij} = \left(r_{ij} - \sum_{k=1}^{K} u_{ik} v_{jk} \right)^2$$



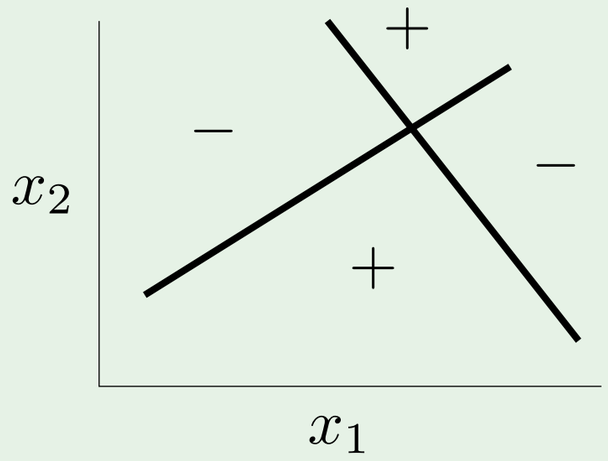
Biological inspiration

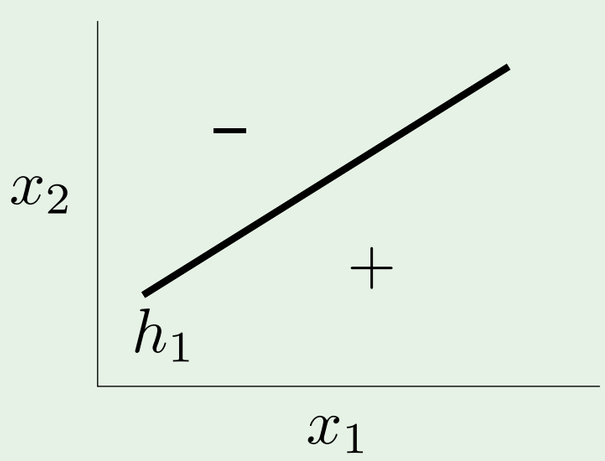
biological function $\longrightarrow$ biological structure

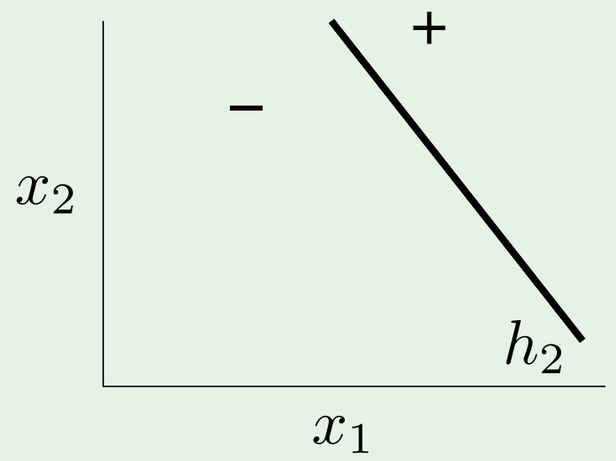


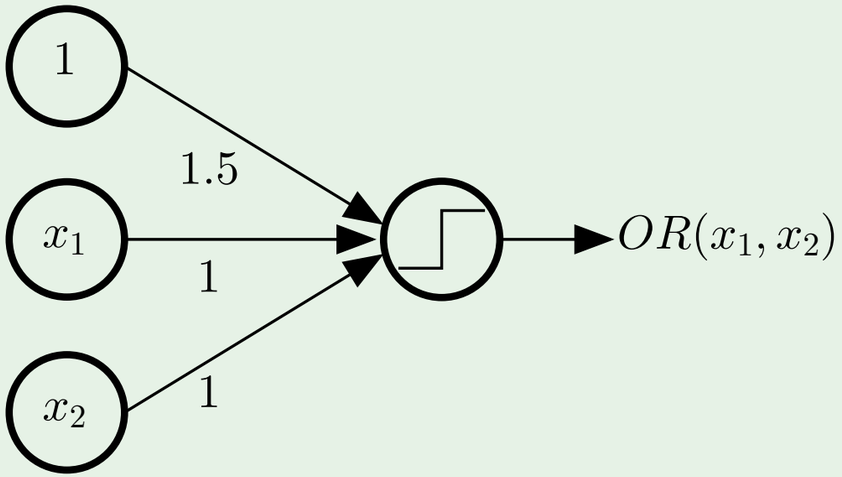


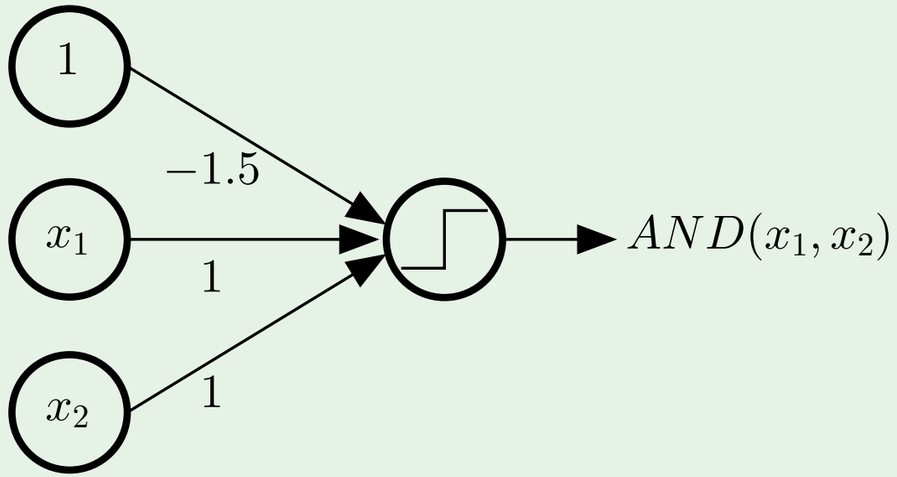
Combining perceptrons



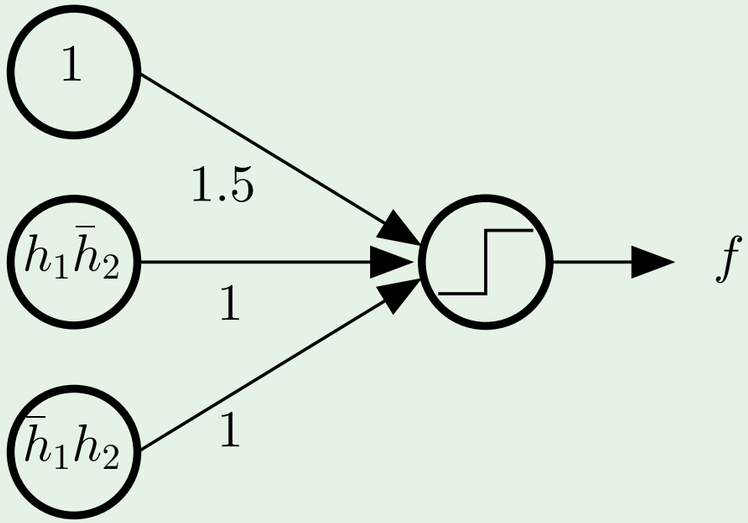


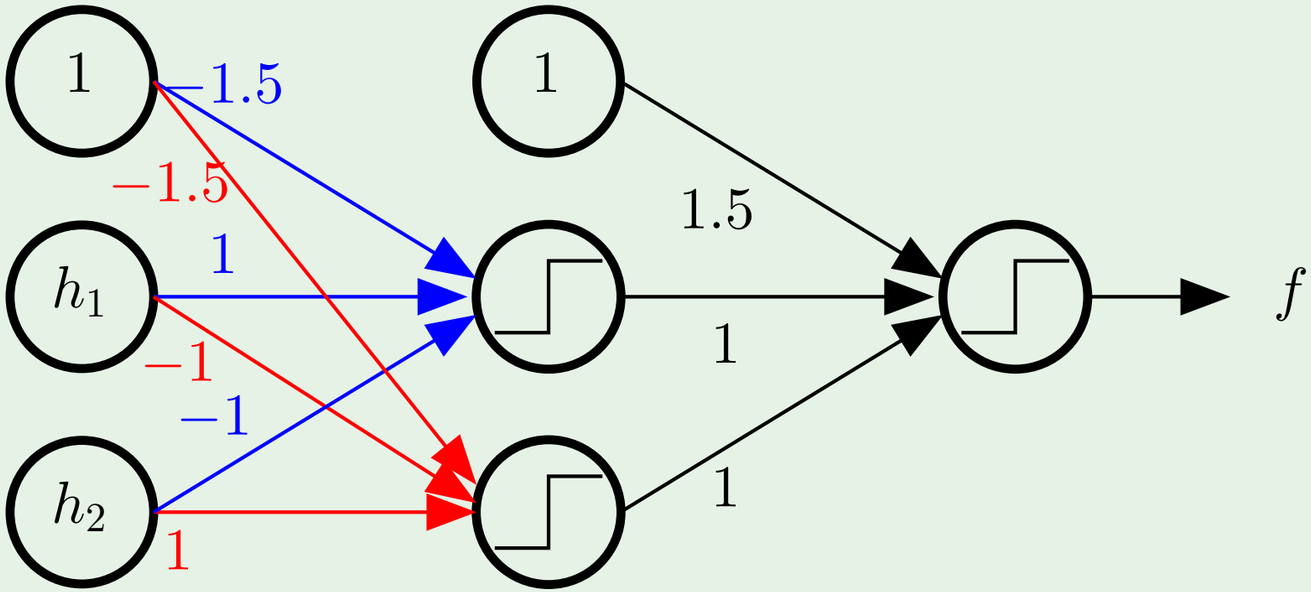




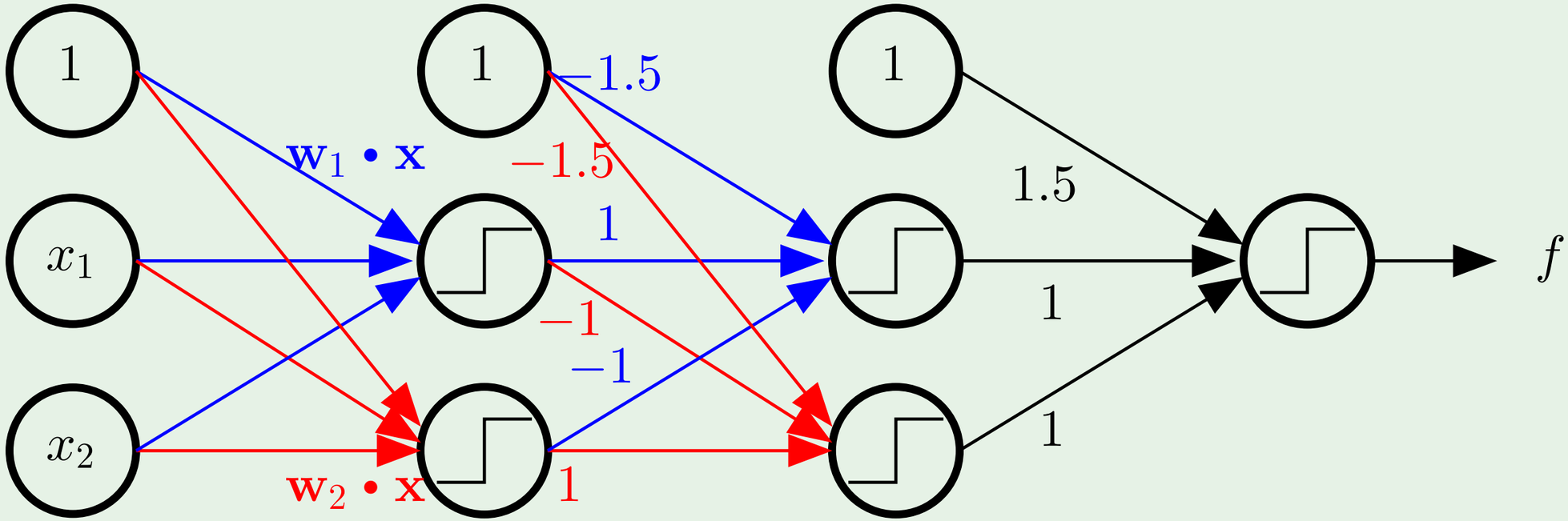


Creating layers



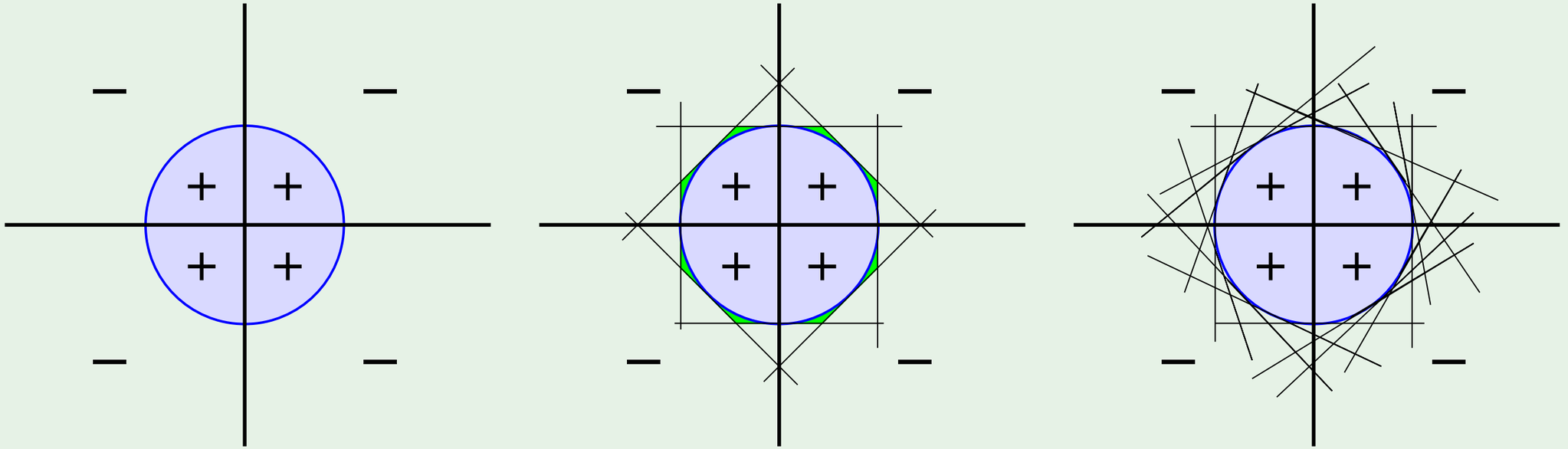


The multilayer perceptron



3 layers “feedforward”

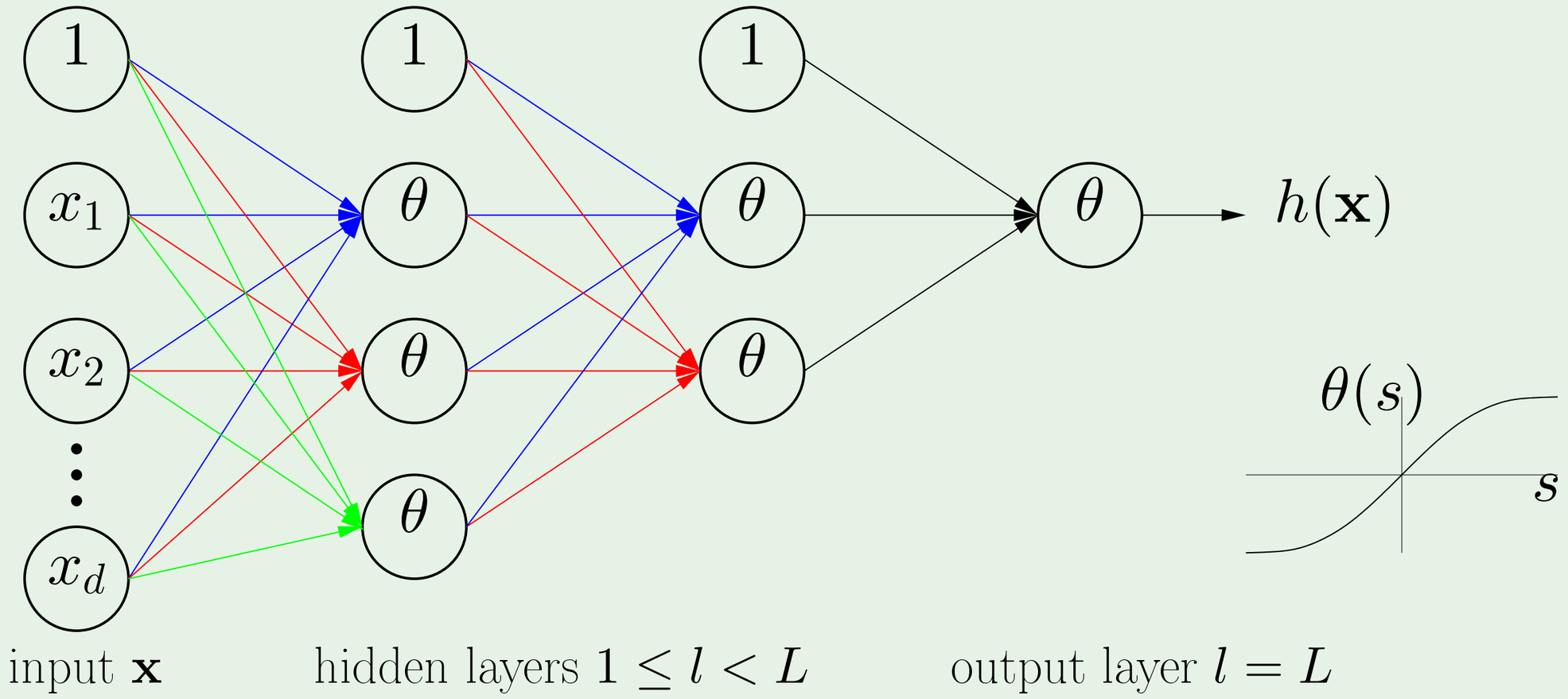
A powerful model



Target | 8 perceptrons | 16 perceptrons

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

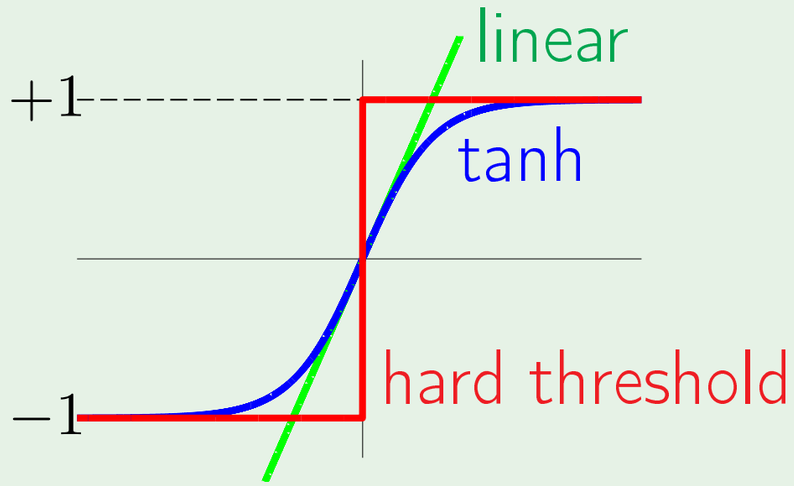
The neural network



How the network operates

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

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



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

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

Applying SGD

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

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

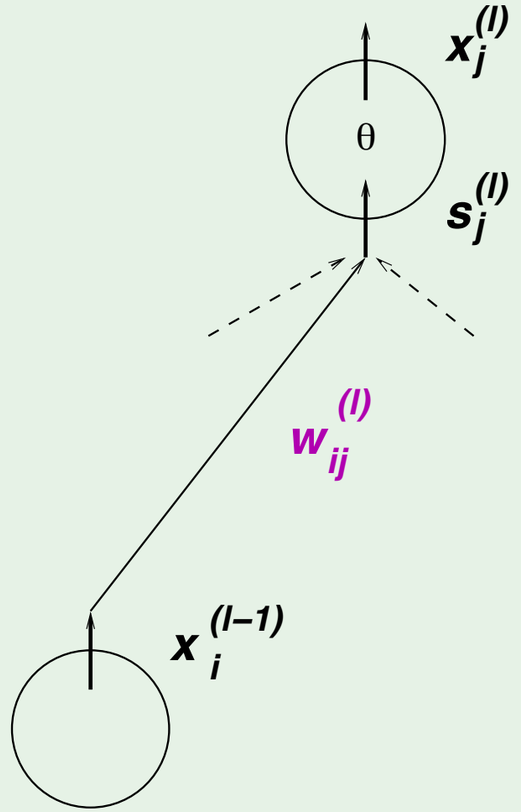
$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

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 $j = 1$:

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

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

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

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

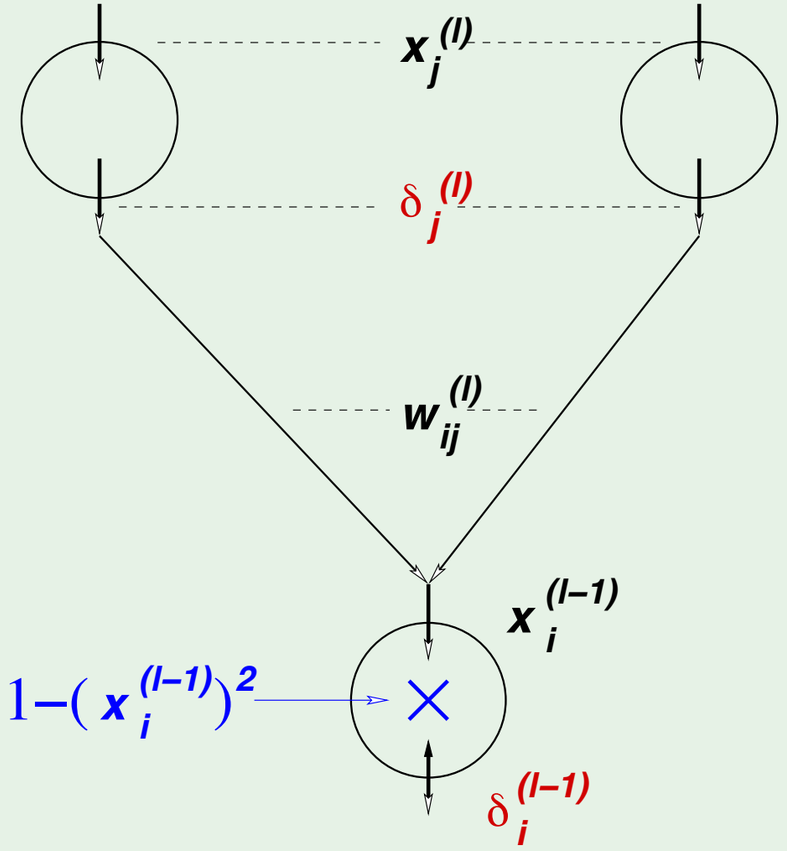
Back propagation of δ

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

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

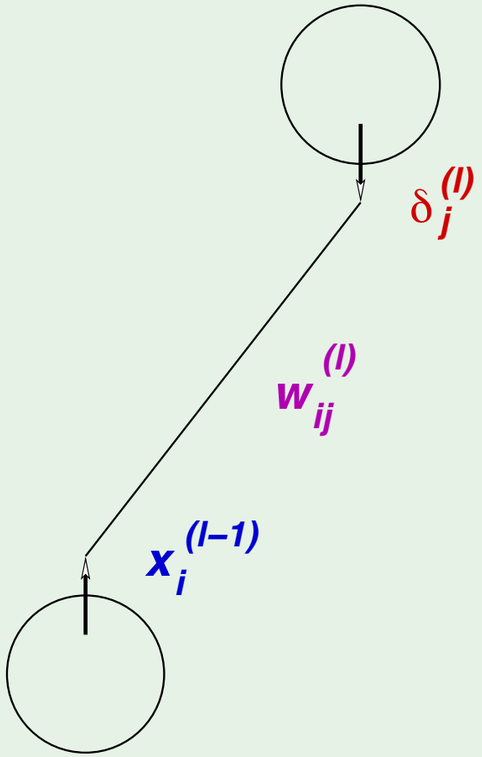
$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})$$

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



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?