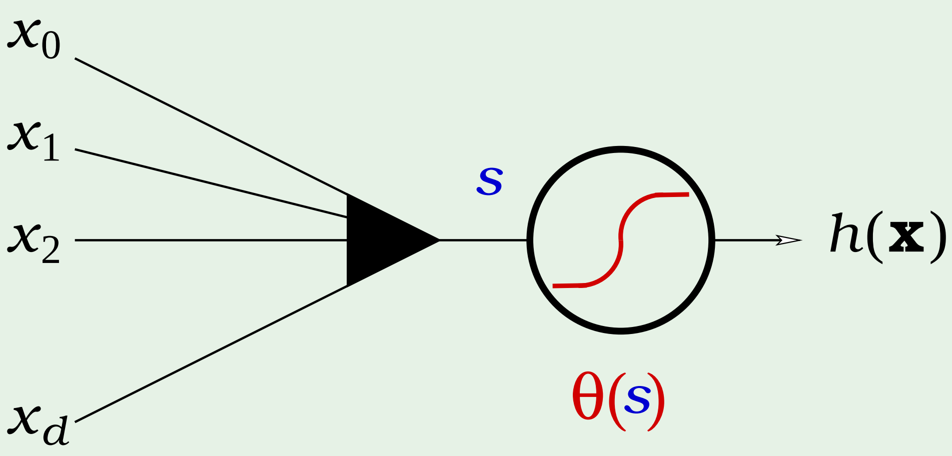
Review of Lecture 9

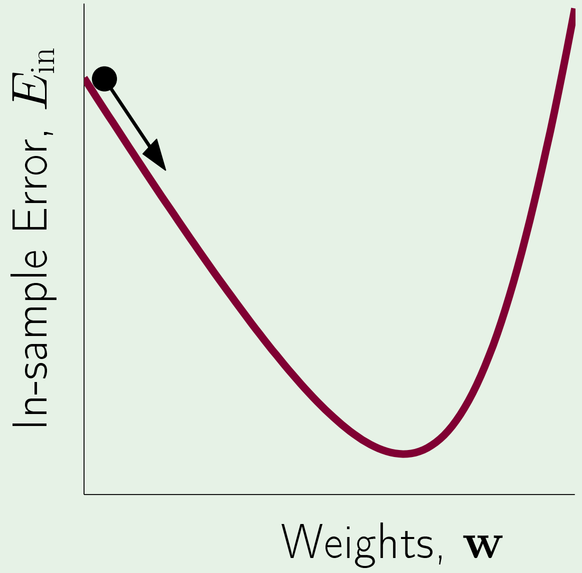
- Logistic regression



- 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:

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

$\longleftarrow$ in logistic regression

One step is termed an epoch - we call something an epoch when we have considered all the training examples at once

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 $(\mathbf{x}_n, y_n)$

We term this standard GD as "**batch**" GD

The stochastic aspect

Pick <u>one</u> $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$ (almost like the perceptron learning algorithm)

"Average" direction:

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

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

Every step, we go along this average direction + (some noise) since we only consider one example at a time - this noise introduces a stochastic aspect. So we essentially go along the direction we want, except we now only involve one example in the computation (which helps a lot) and we have a stochastic aspect. After many steps the noise associated with computing with a single example each time will average out and we travel along the average direction.

randomized version of GD

stochastic gradient descent (SGD)

Benefits of SGD

Situations where the randomization/stochastic nature of algorithm helps with annoying artifacts of the optimization of a surface - the direction is no longer deterministic so the 'up and down' nature tends to avoid terminating in 'silly' local minima (as in left example) and long plateaus before a minimum.

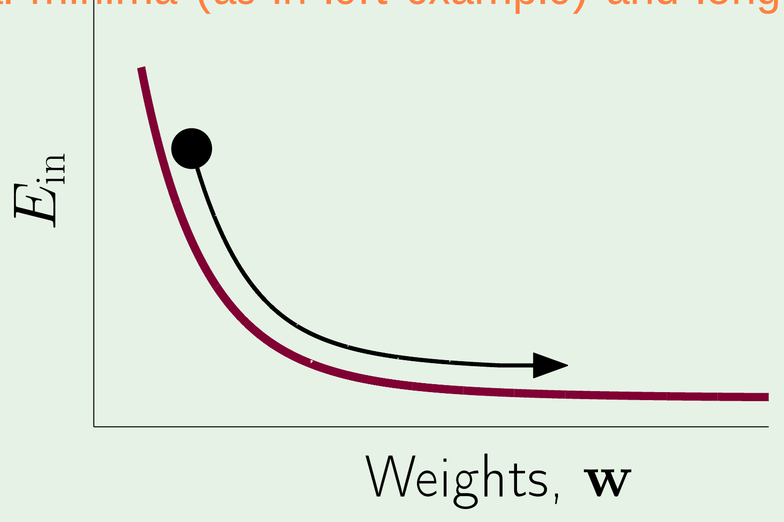
1. cheaper computation

2. randomization

3. simple

Rule of thumb:

$\eta = 0.1$ works

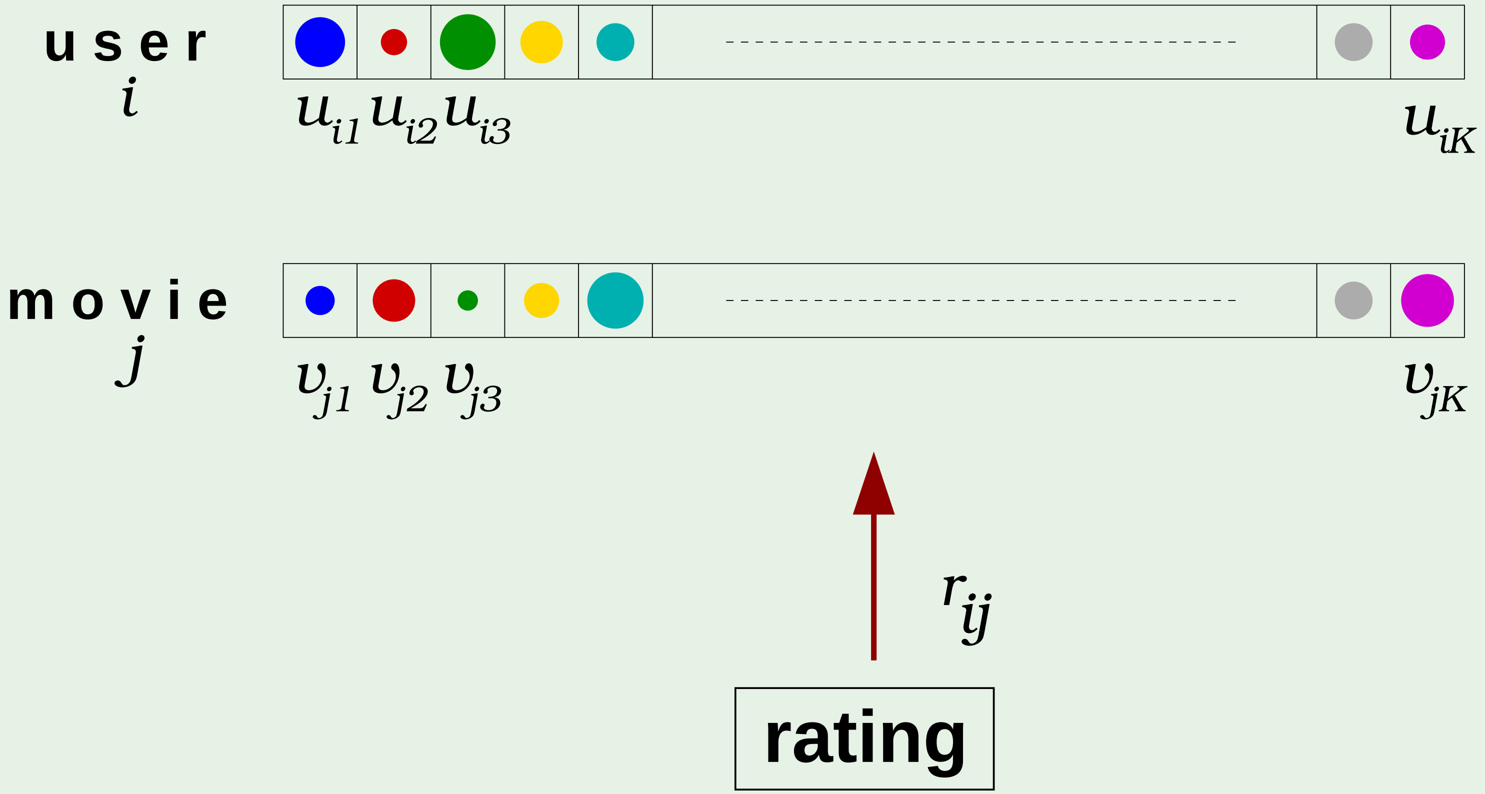


randomization helps

SGD in action

Remember movie ratings?

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

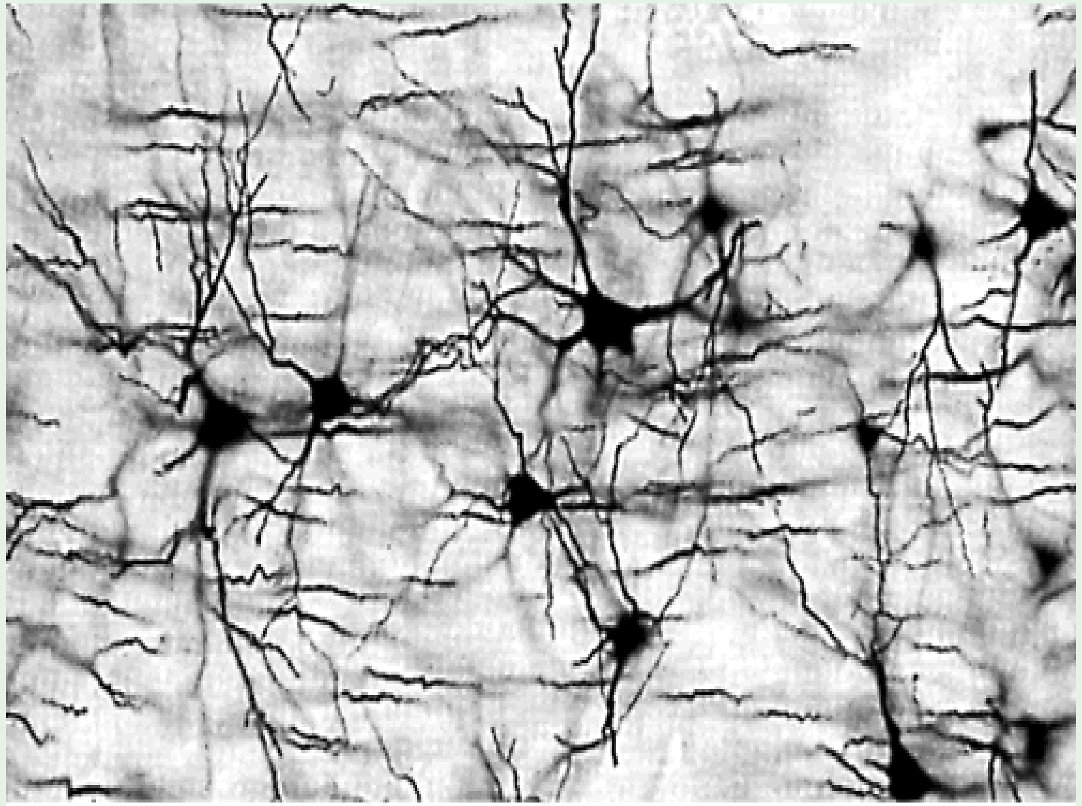


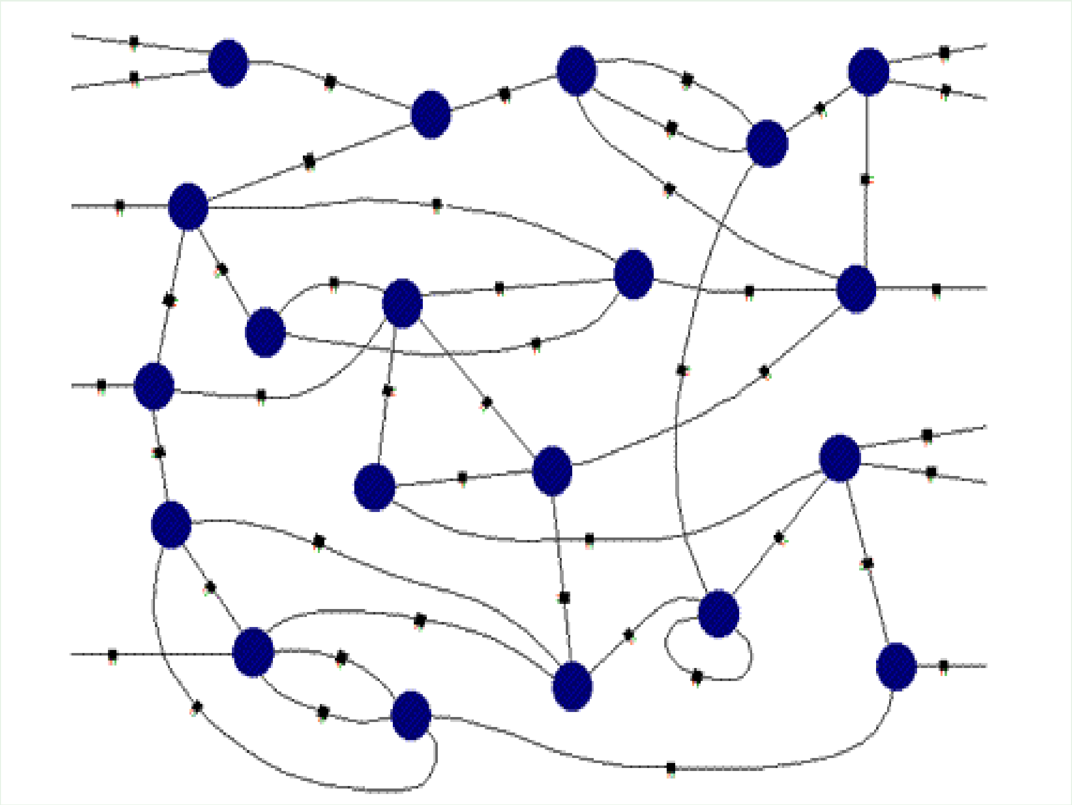
Outline

- Stochastic gradient descent
- Neural network model
- Backpropagation algorithm

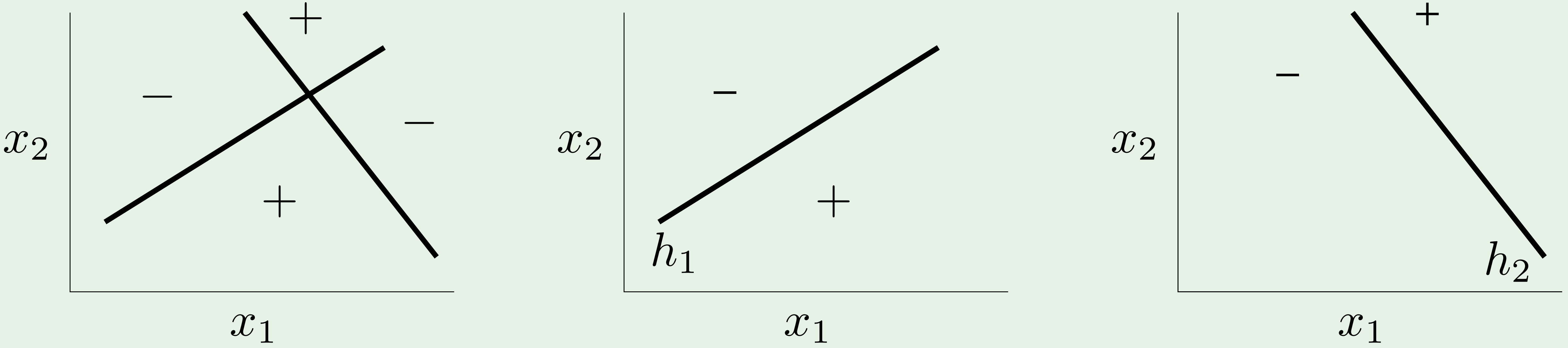
Biological inspiration

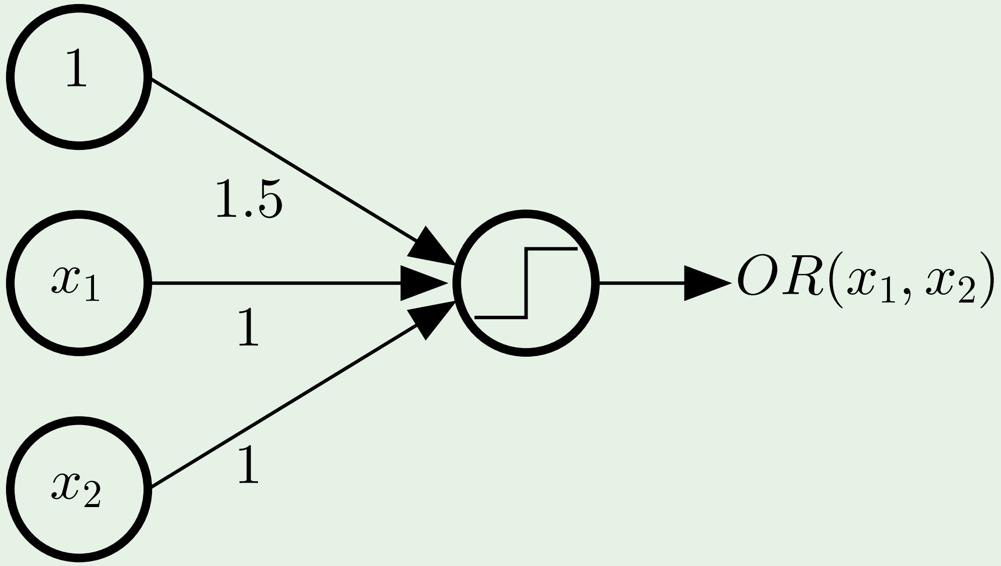
biological function $\longrightarrow$ biological structure

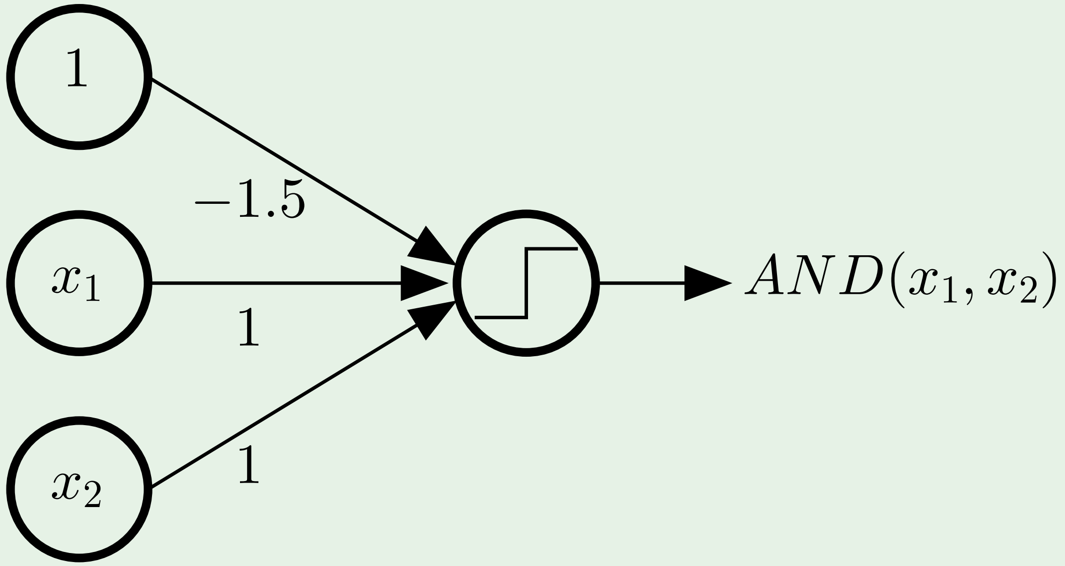




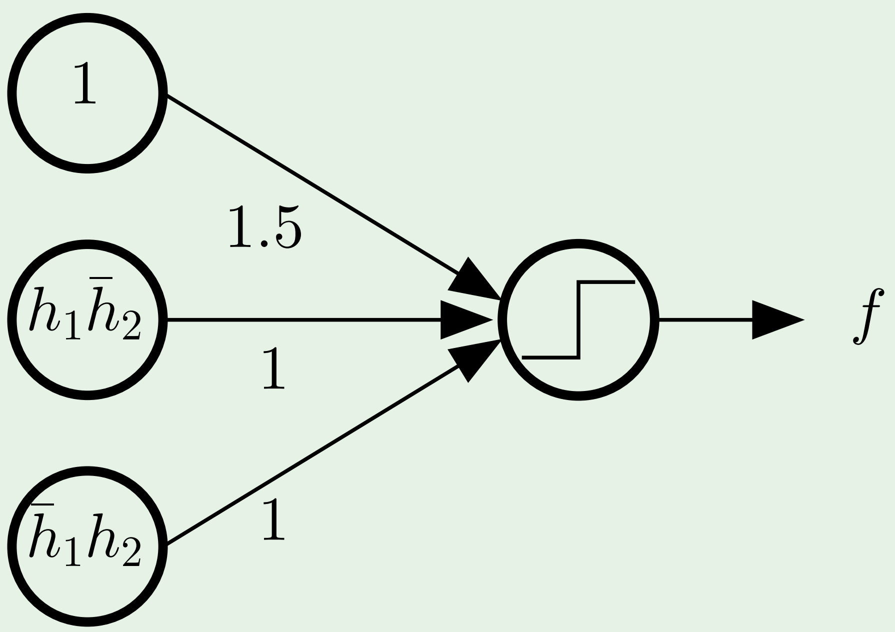
Combining perceptrons

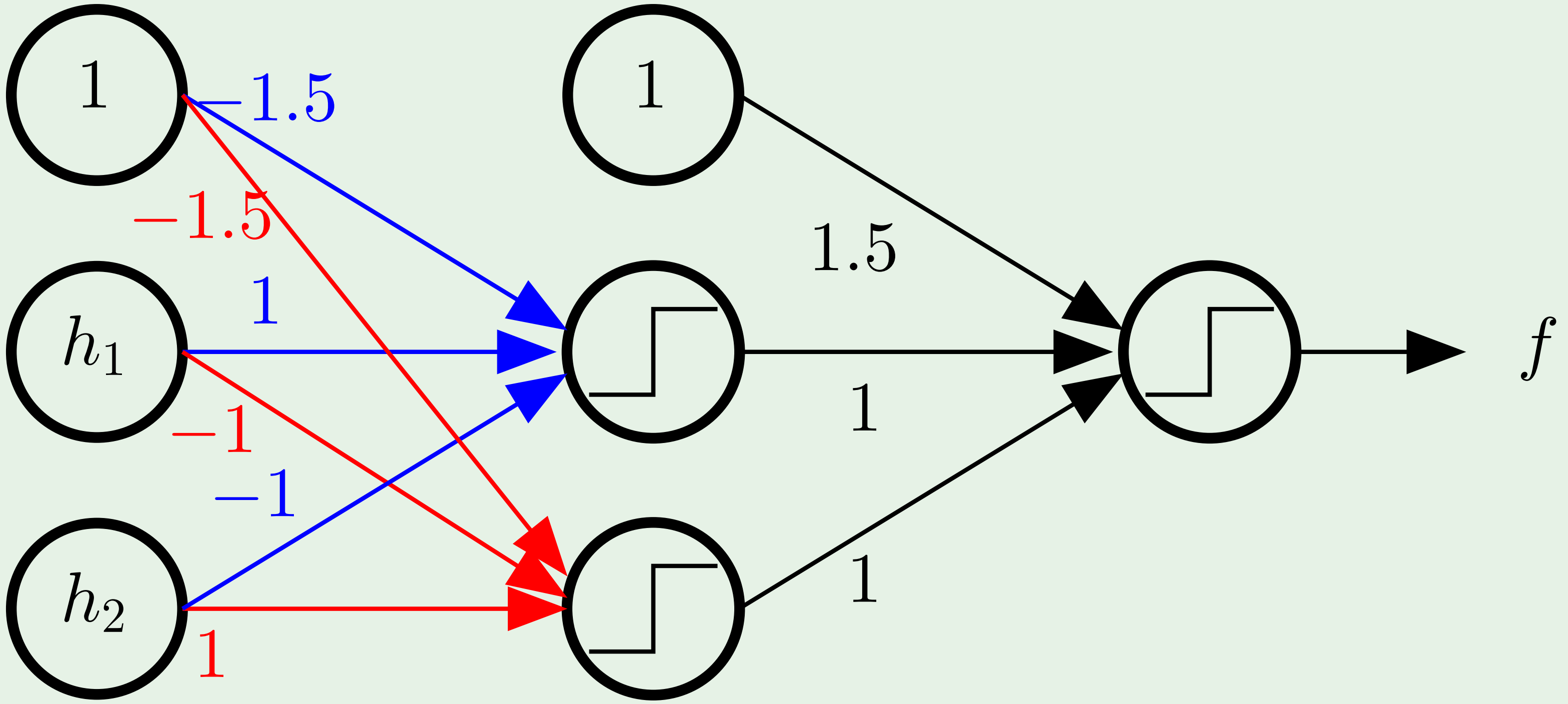




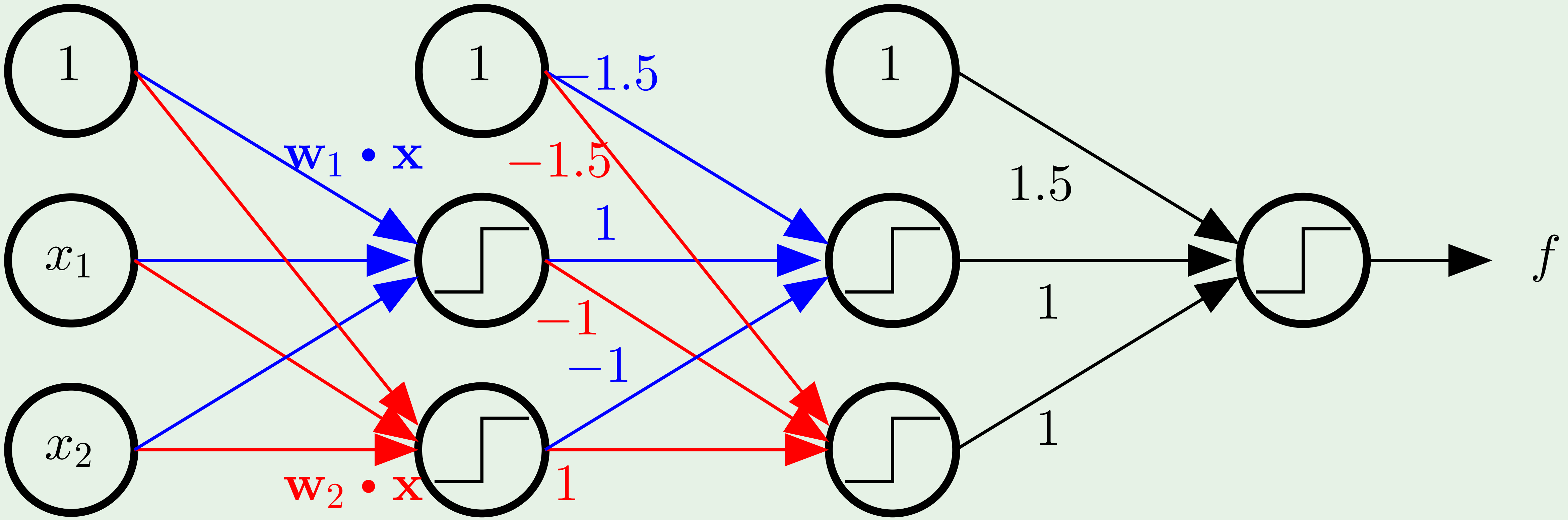


Creating layers



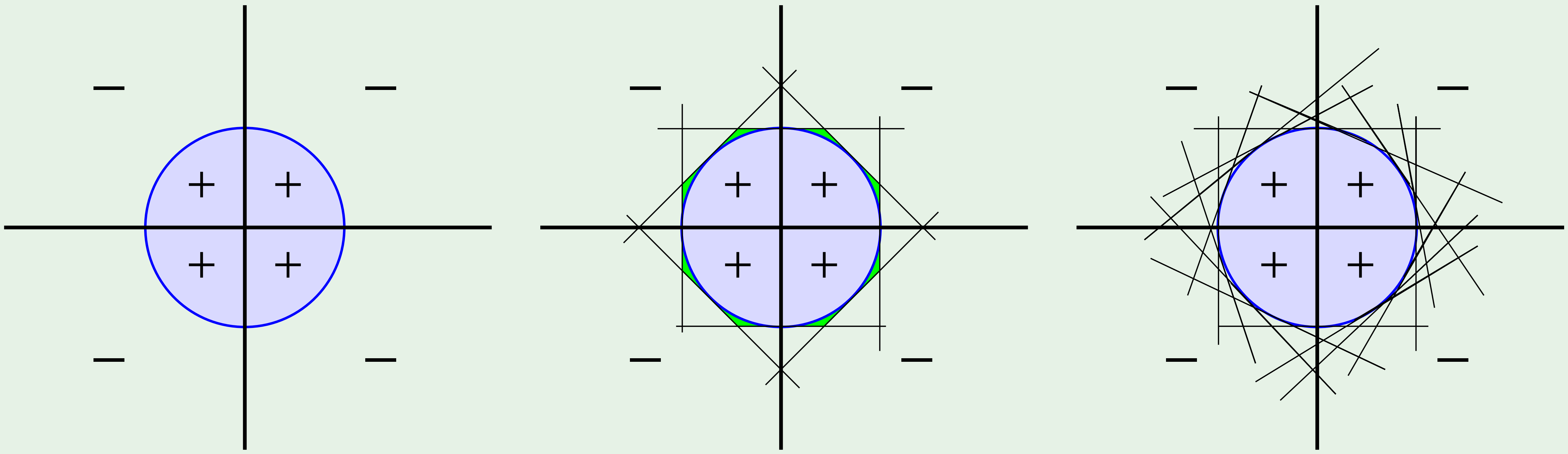


The multilayer perceptron



3 layers "feedforward"

A powerful model



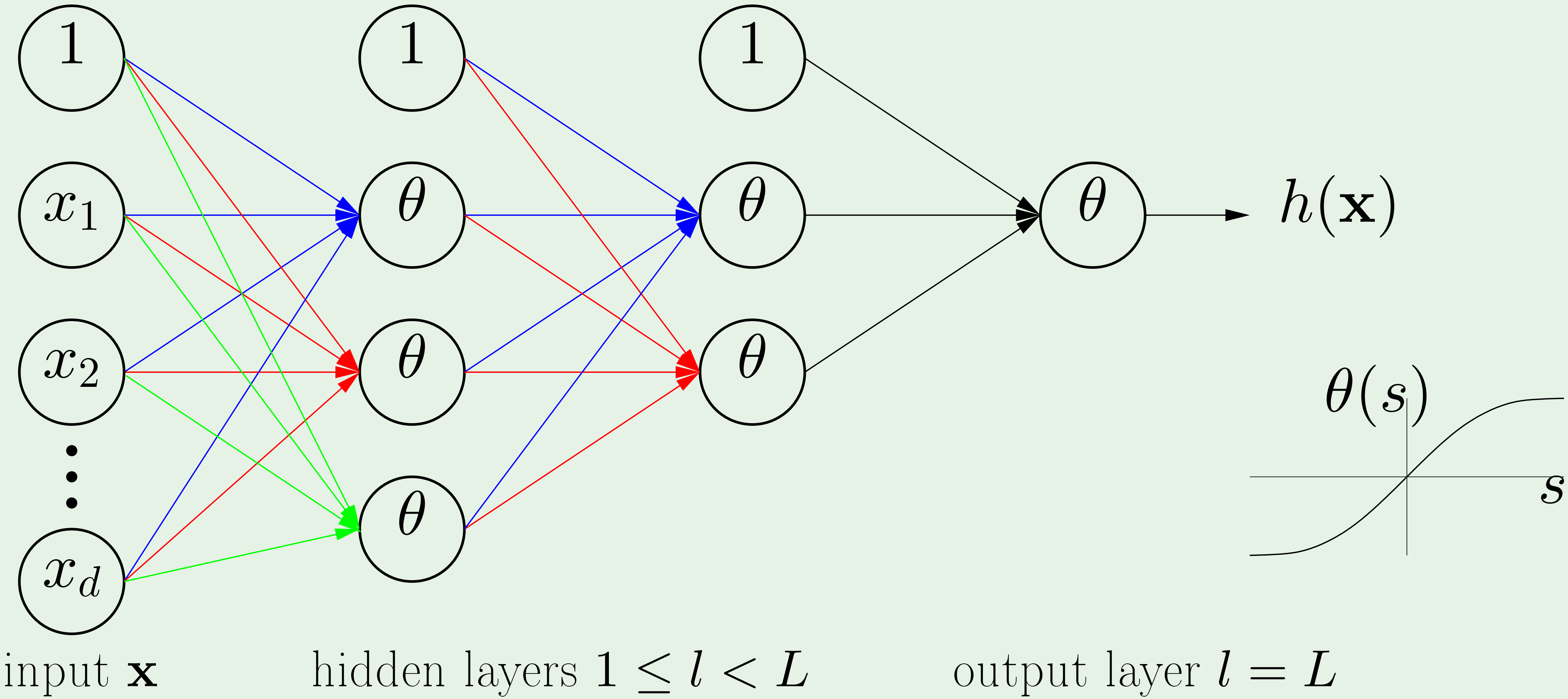
2 red flags for **generalization** and **optimization**

large VC dimension since many weights we can change/d.o.f. (depending on how many perceptrons) - therefore potentially large dataset needed

Very difficult to optimize weights for many perceptrons to match the function (which we do not know) - single perceptron combinatorial optimization was difficult, so multi-layer perceptron optimization will be very difficult

Theta represents the non-linearity which appears in each neuron. Technically each theta could be different and we can account for this in the algorithm, e.g. all soft threshold tanh except for linear last neuron (to replicate linear regression - implementing a real-valued function)

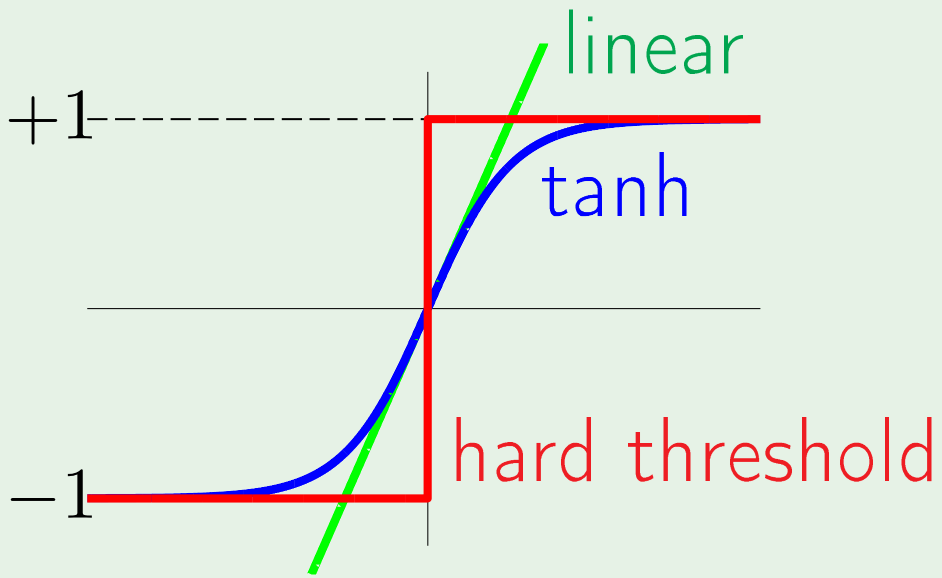
The neural network



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

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

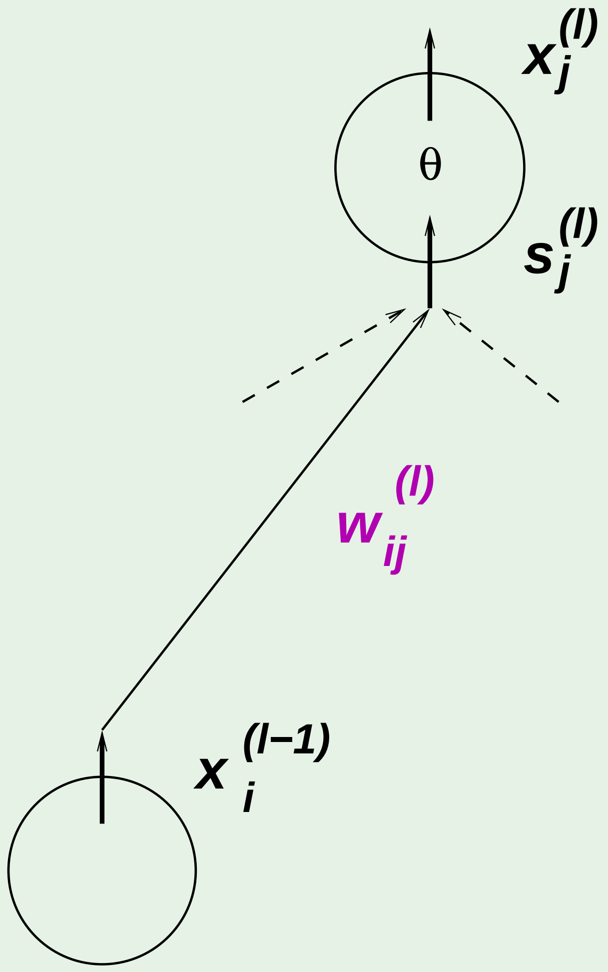
$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

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

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



A trick for efficient computation:

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

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

assign the name delta

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 $j = 1$:

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

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

applies to any error measure, mean squared error just an example

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

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

for the tanh

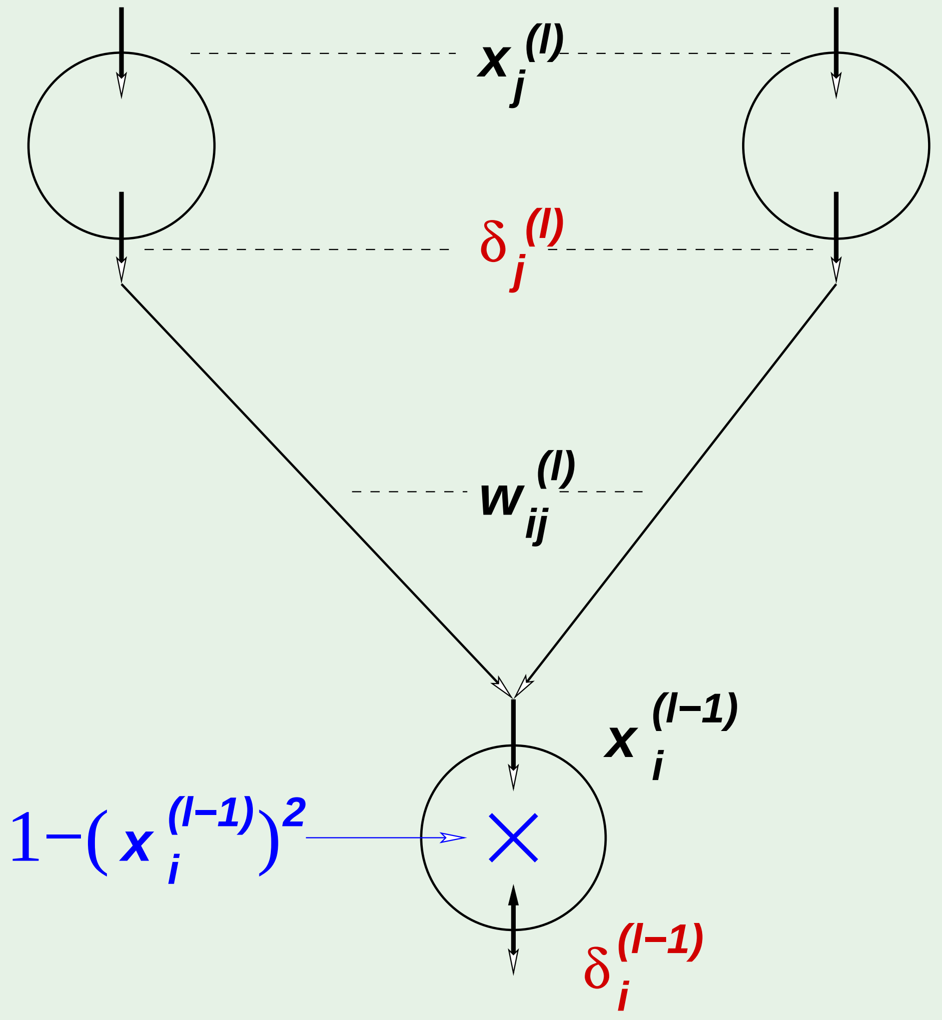
Back propagation of δ

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

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

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

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

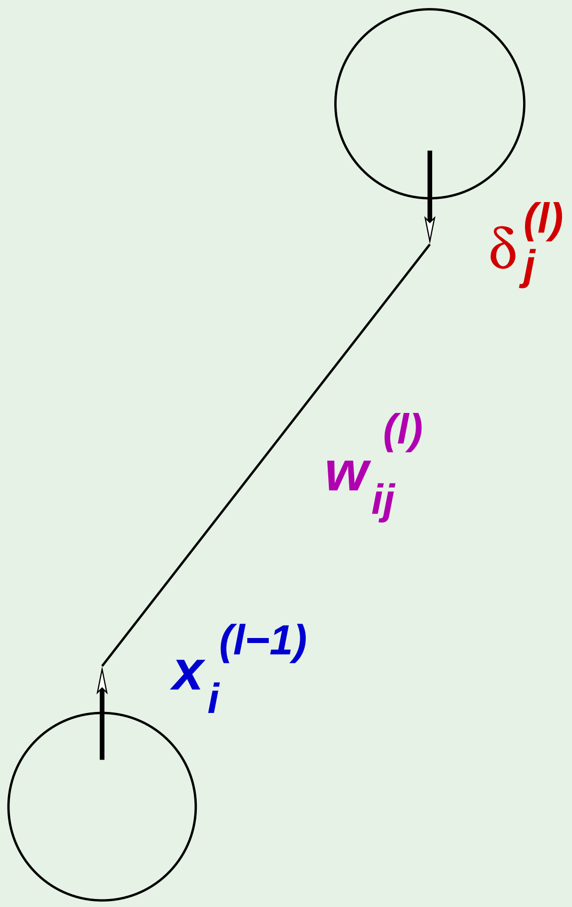


Looks exactly like the forward pass of the network: sum up the weights*something and instead of applying a non-linearity, we multiply by theta'. We get a delta for each neuron (where a signal is fed) and can adjust each weight sandwiched between this delta and the each previous x

Backpropagation algorithm

equal to grad(e(w)), see slide 16,17

Initialize all weights $w_{ij}^{(l)}$ **at random**
for $t = 0, 1, 2, \ldots$ **do**
    Pick $n \in \{1, 2, \cdots, N\}$ (at random - for SGD)
    Forward: Compute all $x_j^{(l)}$
    Backward: Compute all $\delta_j^{(l)}$
    Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \, x_i^{(l-1)} \delta_j^{(l)}$
    Iterate to the next step until it is time to stop
Return the final weights $w_{ij}^{(l)}$



Initialize all the weights at random because if they are all zero, with multiple layers, then either the x's or the delta's would be zero for each weight, so nothing would happen. This is because of the coincidence that we are perfectly at the top of a hill, unable to break the symmetry - the initialization of random and small weights breaks the symmetry

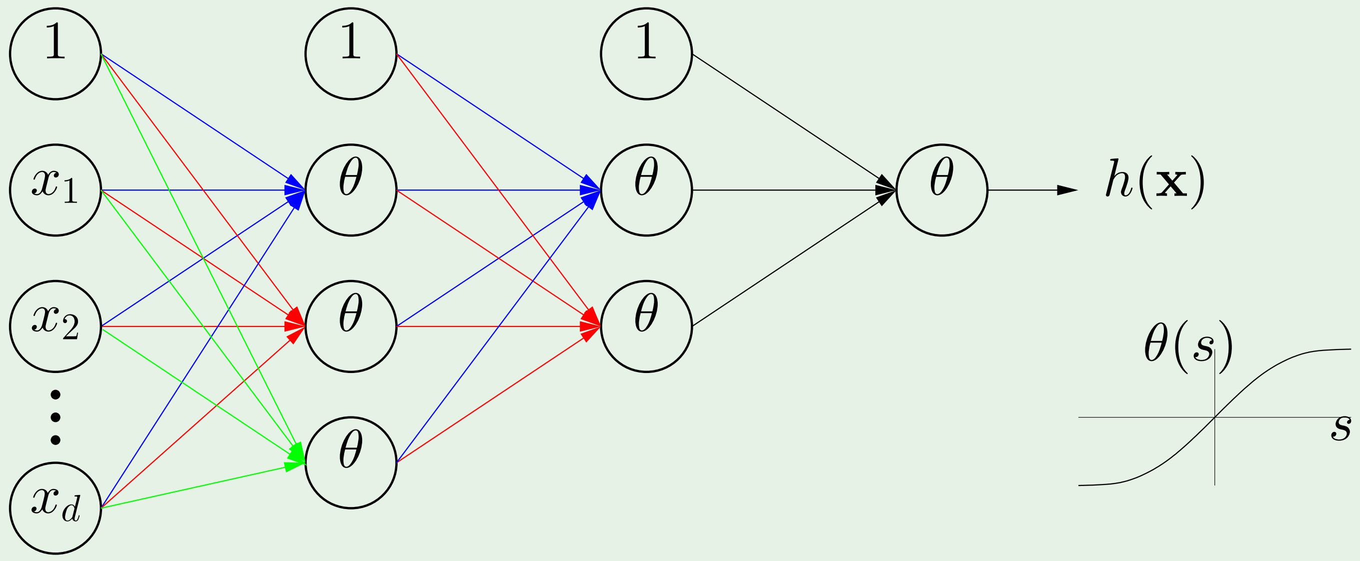
Final remark: hidden layers

learned nonlinear transform

Raw inputs passed through 1st hidden layer neurons which extract higher-order features from the data, the 2nd hidden layer extracts features of features etc. and these end up allowing us to better fit the data and approximate the target function.

interpretation?

When the learning algorithm tries to learn, it tries to produce the right hypothesis, not explain to you what the right hypothesis is/the interpretation of the features or why it gives a certain output for a given input



The neural network looks at the data and learns a non-linear transform/set of features from it - the weights are adjusted to get the proper transform that best fits the data. However, unlike data snooping, we have already 'charged' for the correct VC dimension (which is more or less the number of weights) - it is not looking at the data which is bad, but looking at it without accounting for it (but here it is built in that it is accounted for). This not a generic non-linear transform but one with a view to match specifically the dependency that we are after - so here is source of efficiency from using a neural network.