Neural Networks 1: Multilayer Perceptron

Multiple layers
Universal Approximation
The Neural Network

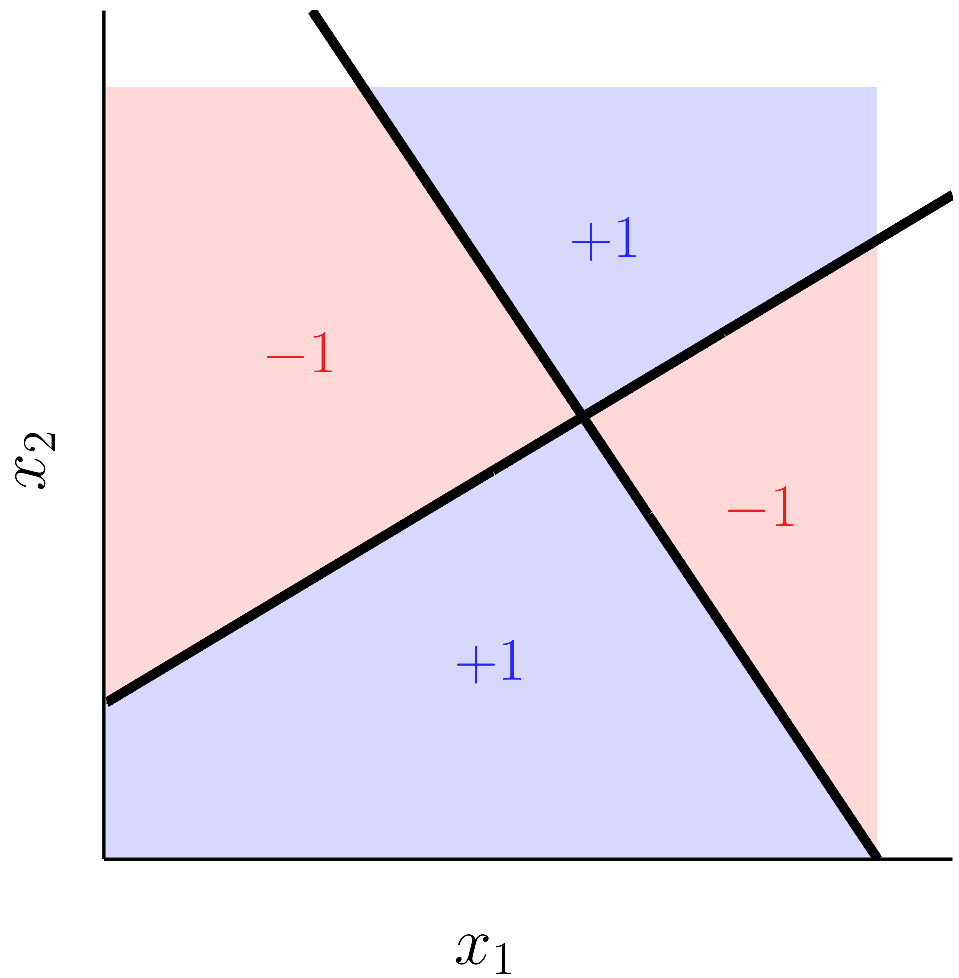
Contents adopted from M. Magdon-Ismail, CSCI 4100/6100

The Neural Network - Biologically Inspired

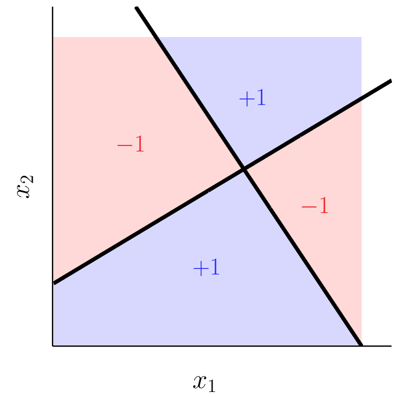
Planes Don't Flap Wings to Fly

Engineering success may start with biological inspiration, but then take a totally different path.

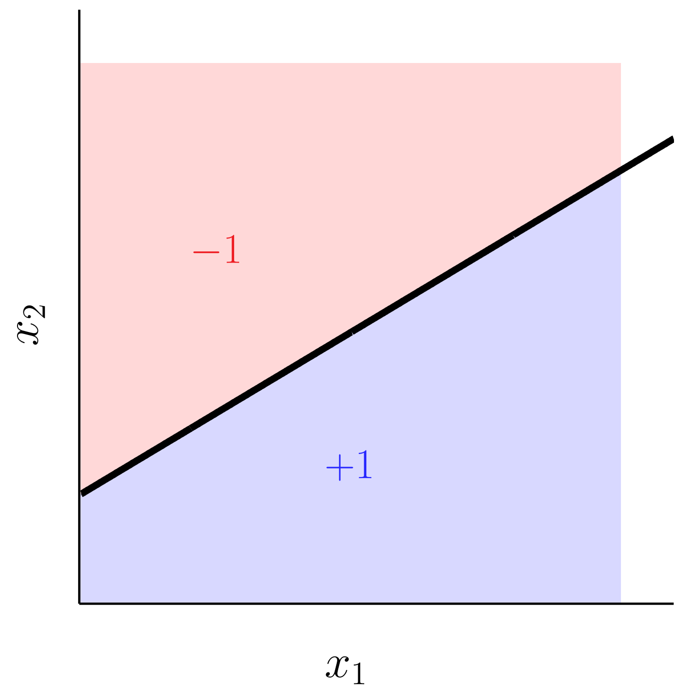
XOR: A Limitation of the Linear Model



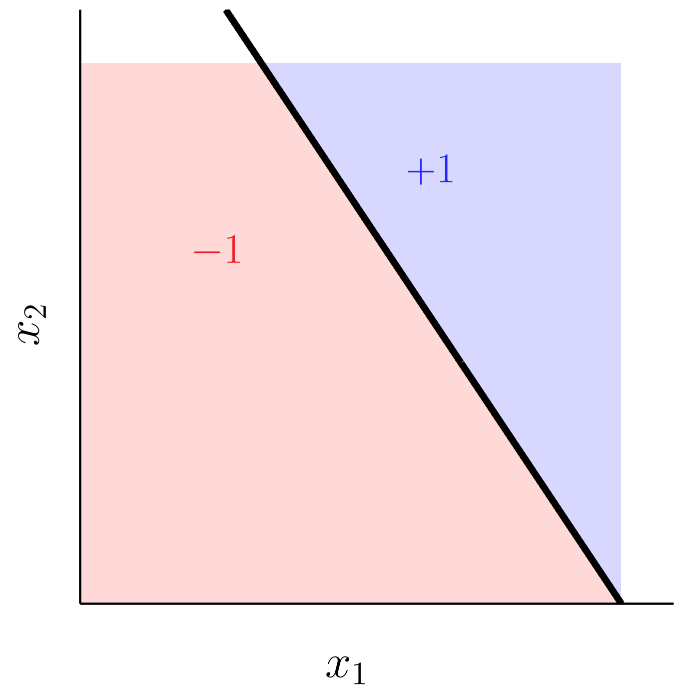
Decomposing XOR



$$f = h_1\overline{h_2} + \overline{h_1}h_2$$



$$h_1(\mathbf{x}) = \text{sign}(\mathbf{w}_1^{\mathrm{T}}\mathbf{x})$$

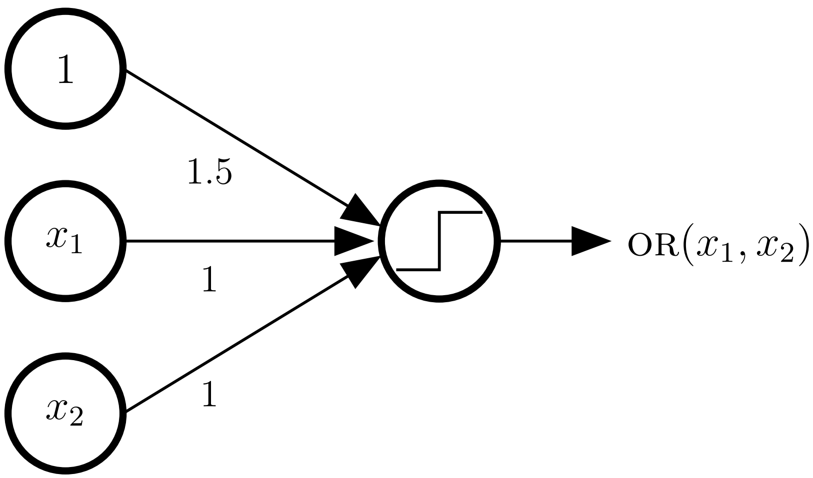


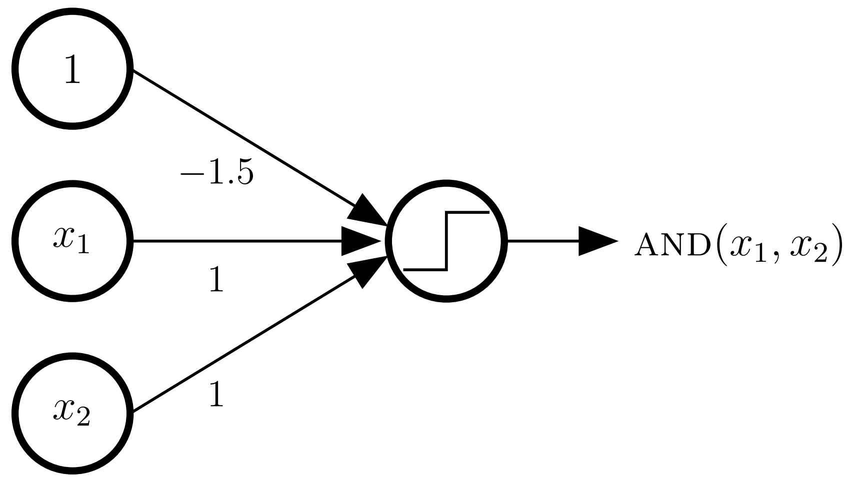
$$h_2(\mathbf{x}) = \text{sign}(\mathbf{w}_2^{\mathrm{T}}\mathbf{x})$$

Perceptrons for OR and AND

$$\text{OR}(x_1, x_2) = \text{sign}(x_1 + x_2 + 1.5)$$

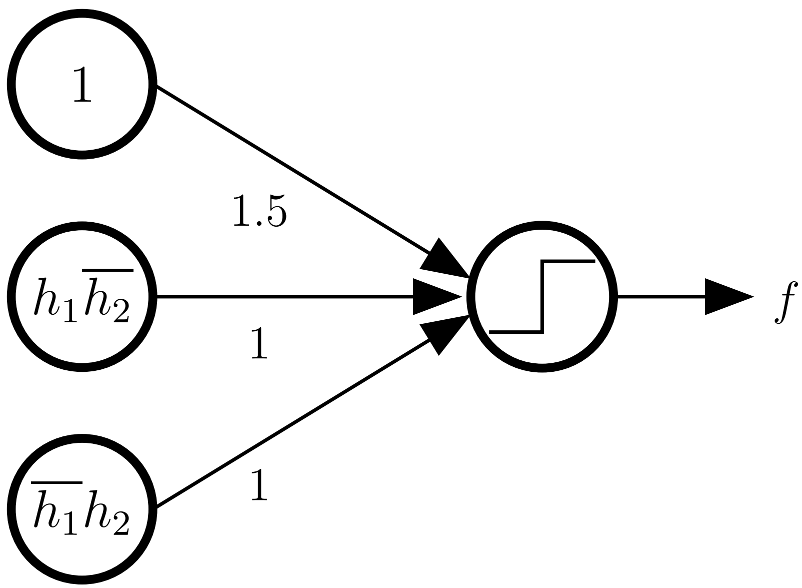
$$\text{AND}(x_1, x_2) = \text{sign}(x_1 + x_2 - 1.5)$$





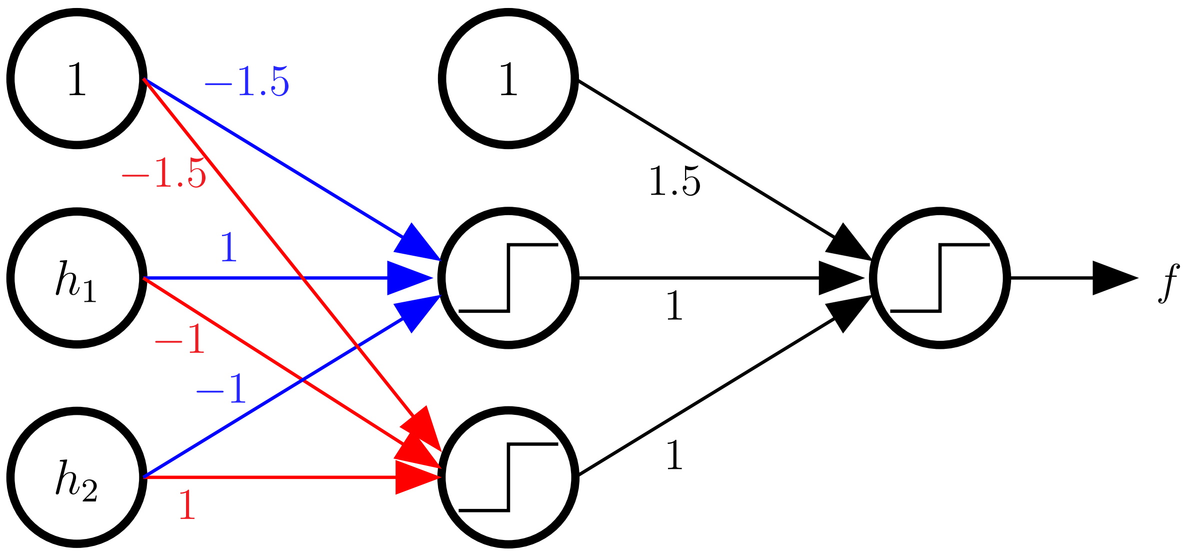
Representing f Using OR and AND

$$f = h_1\overline{h_2} + \overline{h_1}h_2$$



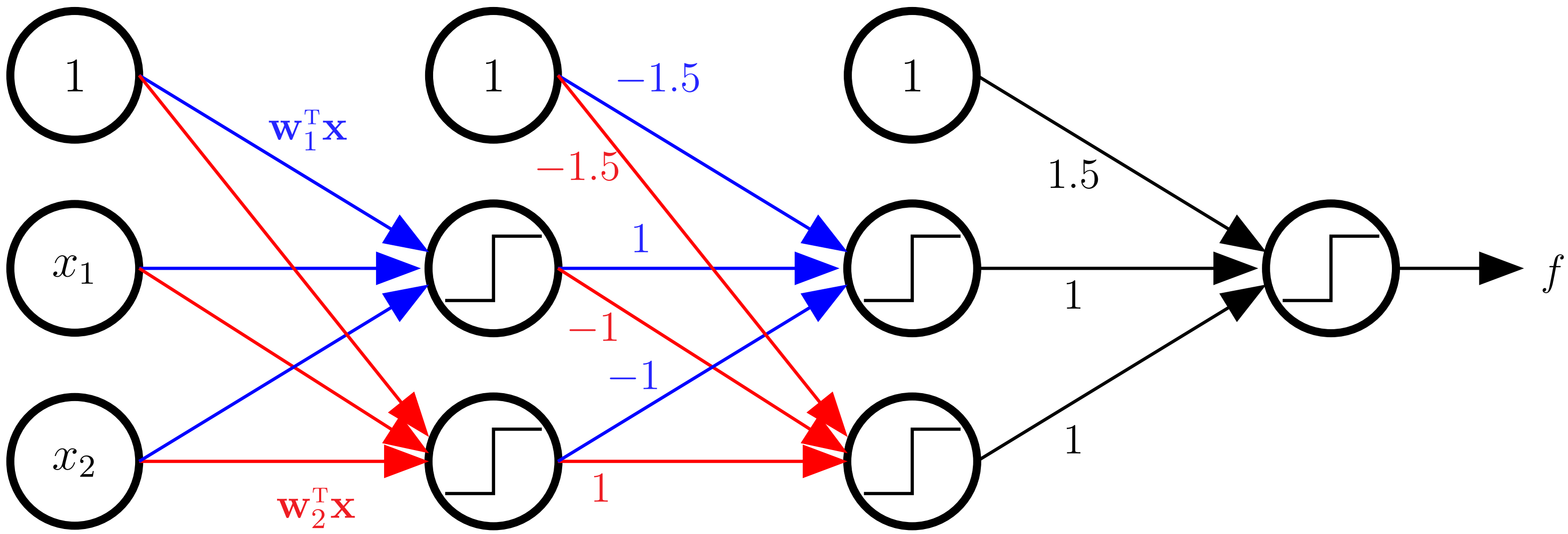
Representing f Using OR and AND

$$f = h_1\overline{h_2} + \overline{h_1}h_2$$

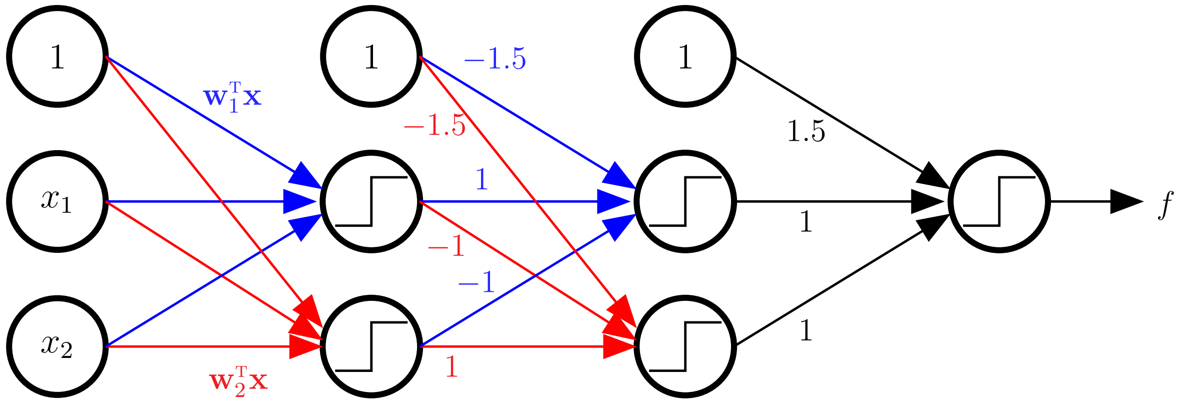


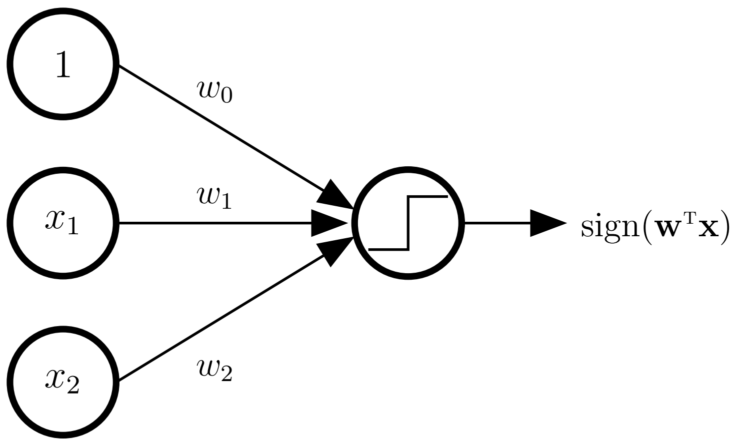
Representing f Using OR and AND

$$f = h_1\overline{h_2} + \overline{h_1}h_2$$



The Multilayer Perceptron (MLP)





More layers allow us to implement f

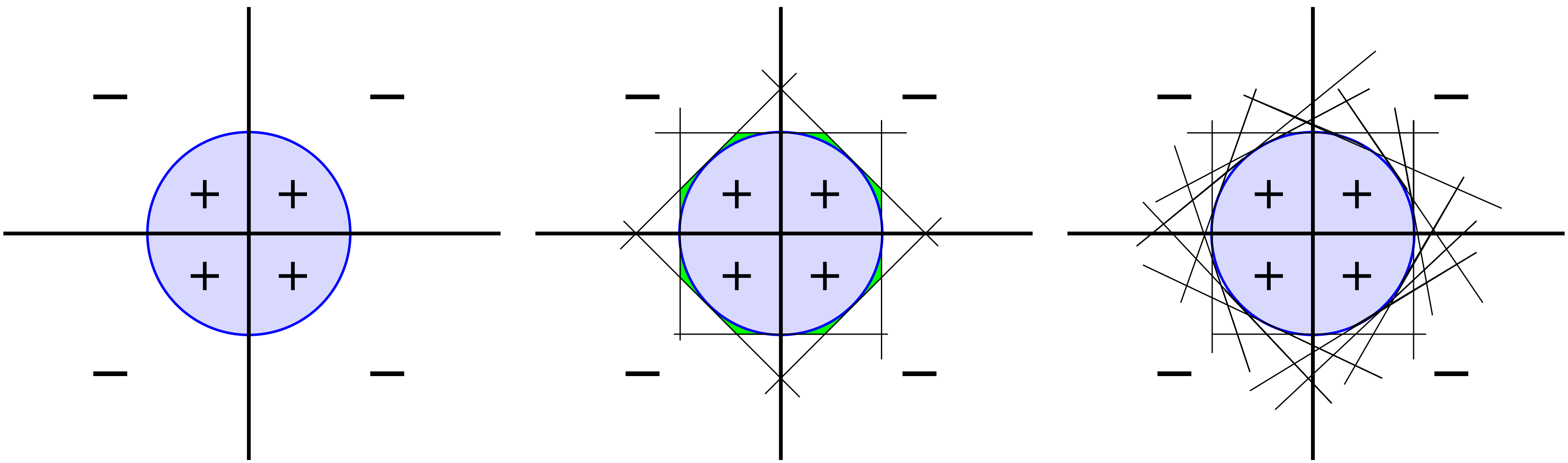
These additional layers are called *hidden layers*

Universal Approximation

Any target function f that can be decomposed into linear separators can be implemented by a 3-layer MLP.

Universal Approximation

A sufficiently smooth separator can "essentially" be decomposed into linear separators.



Target

8 perceptrons

16 perceptrons

Approximation Versus Generalization

The size of the MLP controls the approximation-generalization tradeoff.

More nodes per hidden layer $\implies$ approximation$\uparrow$ and generalization$\downarrow$

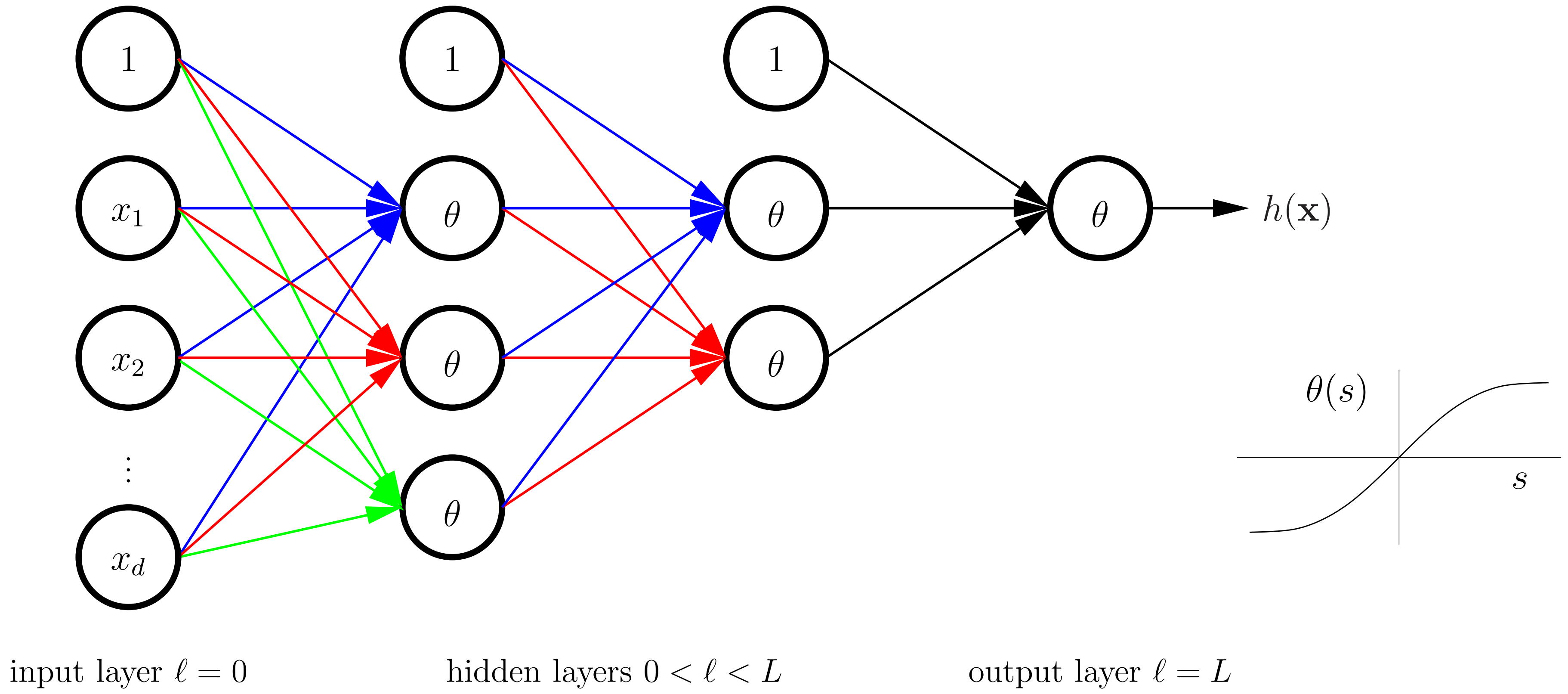
Minimizing E_{in}

A combinatorial problem even harder with the MLP than the Perceptron.

E_{in} is not smooth (due to sign function), so cannot use gradient descent.

$\text{sign}(x) \approx \tan(x) \longrightarrow$ gradient descent to minimize E_{in}.

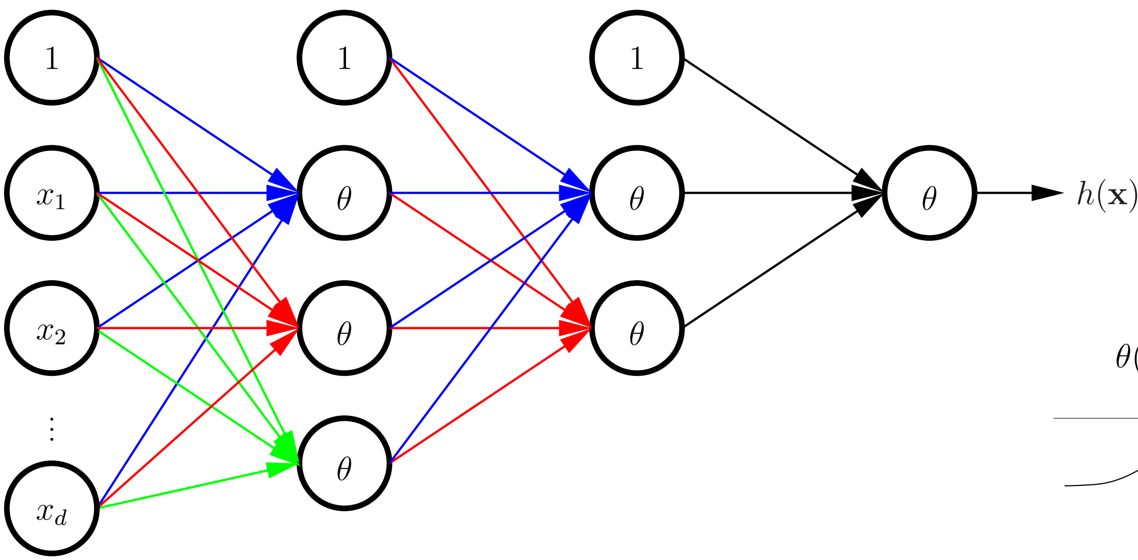
The Neural Network

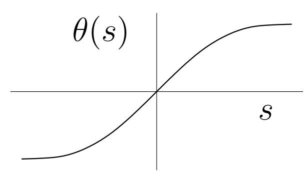


The Neural Network

Biology

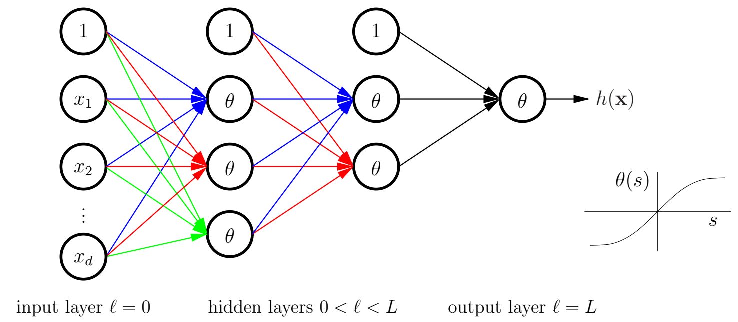
Engineering

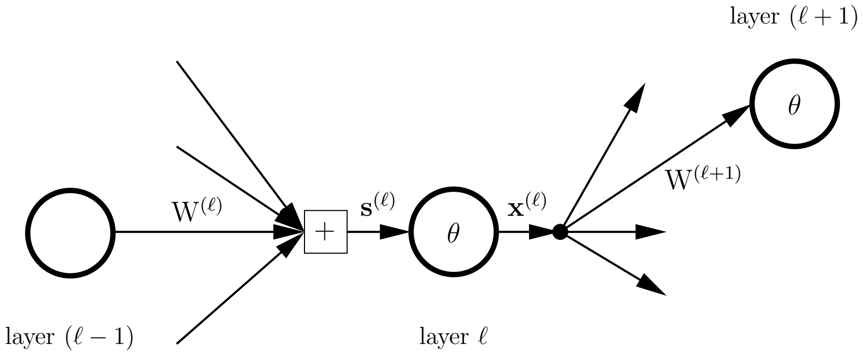




input layer $\ell = 0$ hidden layers $0 < \ell < L$ output layer $\ell = L$

Zooming into a Hidden Node





layer ℓ parameters

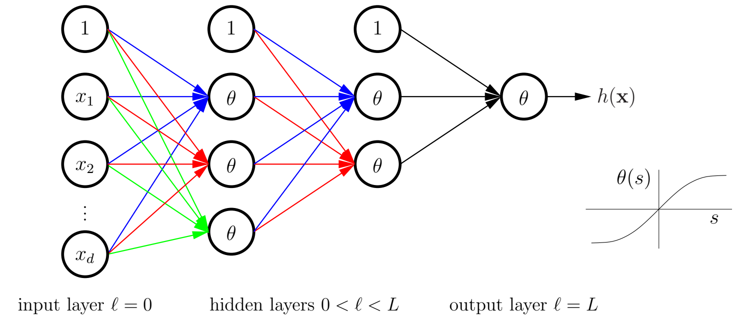
signals in	$\mathbf{s}^{(\ell)}$	$d^{(\ell)}$ dimensional input vector
outputs	$\mathbf{x}^{(\ell)}$	$d^{(\ell)} + 1$ dimensional output vector
weights in	$\mathrm{W}^{(\ell)}$	$(d^{(\ell-1)} + 1) \times d^{(\ell)}$ dimensional matrix
weights out	$\mathrm{W}^{(\ell+1)}$	$(d^{(\ell)} + 1) \times d^{(\ell+1)}$ dimensional matrix

layers $\ell = 0, 1, 2, \ldots, L$
layer ℓ has "dimension" $d^{(\ell)} \implies d^{(\ell)} + 1$ nodes

$$\mathrm{W}^{(\ell)} = \begin{bmatrix} \mathbf{w}_1^{(\ell)} & \mathbf{w}_2^{(\ell)} & \cdots & \mathbf{w}_{d^{(\ell)}}^{(\ell)} \\ \Big| & \Big| & \vdots & \Big| \end{bmatrix}$$

The Linear Signal

Input $\mathbf{s}^{(\ell)}$ is a linear combination (using weights) of the outputs of the previous layer $\mathbf{x}^{(\ell-1)}$.



$$\mathbf{s}^{(\ell)} = (\mathrm{W}^{(\ell)})^{\mathrm{T}} \mathbf{x}^{(\ell-1)}$$

$$\begin{bmatrix} s_1^{(\ell)} \\ s_2^{(\ell)} \\ \vdots \\ s_j^{(\ell)} \\ \vdots \\ s_{d^{(\ell)}}^{(\ell)} \end{bmatrix} = \begin{bmatrix} (\mathbf{w}_1^{(\ell)})^{\mathrm{T}} & \text{———} \\ (\mathbf{w}_2^{(\ell)})^{\mathrm{T}} & \text{———} \\ & \vdots \\ (\mathbf{w}_j^{(\ell)})^{\mathrm{T}} & \text{———} \\ & \vdots \\ (\mathbf{w}_{d^{(\ell)}}^{(\ell)})^{\mathrm{T}} & \text{———} \end{bmatrix} \mathbf{x}^{(\ell-1)}$$

$$s_j^{(\ell)} = (\mathbf{w}_j^{(\ell)})^{\mathrm{T}} \mathbf{x}^{(\ell-1)}$$

(recall the linear signal $s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$)

$$\mathbf{s}^{(\ell)} \xrightarrow{\quad\theta\quad} \mathbf{x}^{(\ell)}$$

Forward Propagation: Computing $h(\mathbf{x})$

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \xrightarrow{\mathrm{W}^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$

Forward propagation to compute $h(\mathbf{x})$:

$\mathbf{x}^{(0)} \leftarrow \mathbf{x}$ **[Initialization]**

for $\ell = 1$ to L **do** **[Forward Propagation]**

    $\mathbf{s}^{(\ell)} \leftarrow (\mathrm{W}^{(\ell)})^{\mathrm{T}} \mathbf{x}^{(\ell-1)}$

    $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$

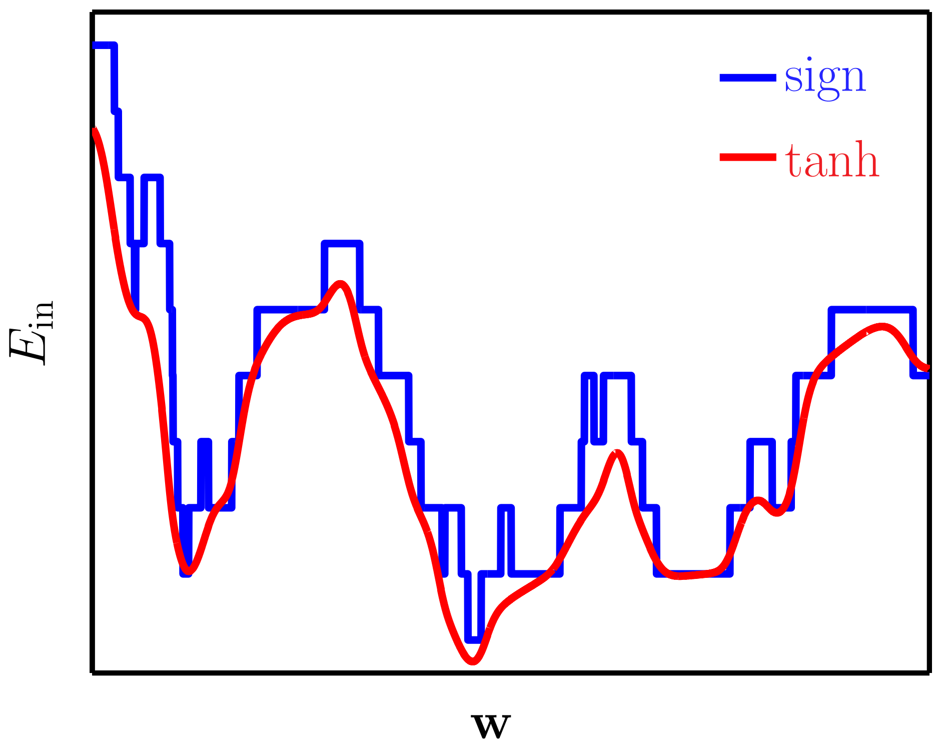
end for

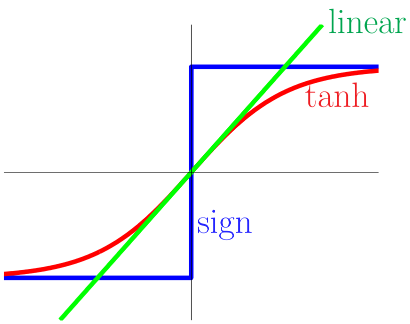
$h(\mathbf{x}) = \mathbf{x}^{(L)}$ **[Output]**

Minimizing E_{in}

$$E_{\text{in}}(h) = E_{\text{in}}(\text{W}) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - y_n)^2$$

$\text{W} = \{\text{W}^{(1)}, \text{W}^{(2)}, \ldots, \text{W}^{(L)}\}$





Using $\theta = \text{tanh}$ makes E_{in} differentiable so we can use gradient descent $\longrightarrow$ local minimum.

Gradient Descent

$$\mathrm{W}(t+1) = \mathrm{W}(t) - \eta \boldsymbol{\nabla} E_{\mathrm{in}}(\mathrm{W}(t))$$

Gradient of E_{in}

e_n

$$E_{\text{in}}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \mathsf{e}(h(\mathbf{x}_n), y_n)$$

$$\frac{\partial E_{\text{in}}(\mathbf{w})}{\partial \mathrm{W}^{(\ell)}} \;=\; \frac{1}{N}\sum_{n=1}^{N} \frac{\partial \mathsf{e}_n}{\partial \mathrm{W}^{(\ell)}}$$

We need

$$\frac{\partial \mathsf{e}(\mathbf{x})}{\partial \mathrm{W}^{(\ell)}}$$

Numerical Approach

$$\frac{\partial \mathsf{e}(\mathbf{x})}{\partial \mathrm{W}_{ij}^{(\ell)}} \approx \frac{\mathsf{e}(\mathbf{x}|\mathrm{W}_{ij}^{(\ell)} + \Delta) - \mathsf{e}(\mathbf{x}|\mathrm{W}_{ij}^{(\ell)} - \Delta)}{2\Delta}$$

approximate

inefficient

Example Data