CSE 417T: Introduction to Machine Learning

Lecture 24: Neural Networks

Henry Chai

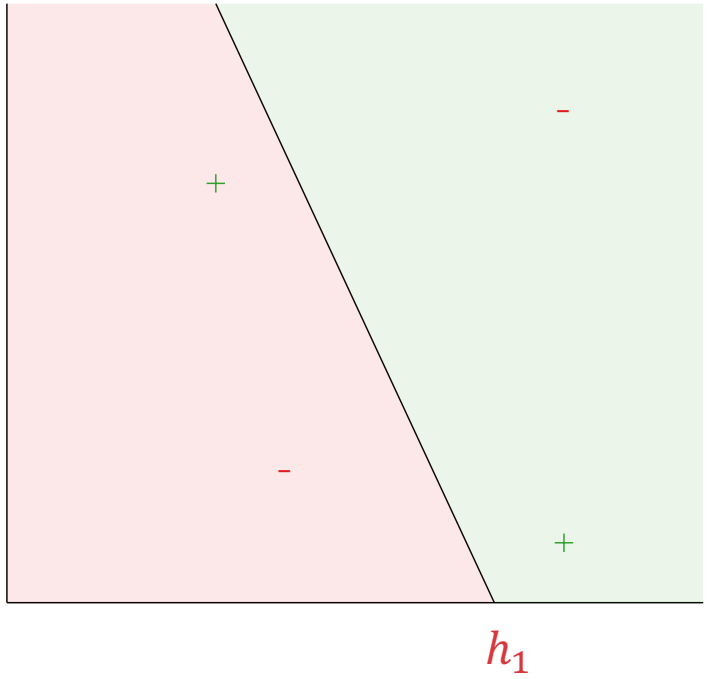
11/27/18

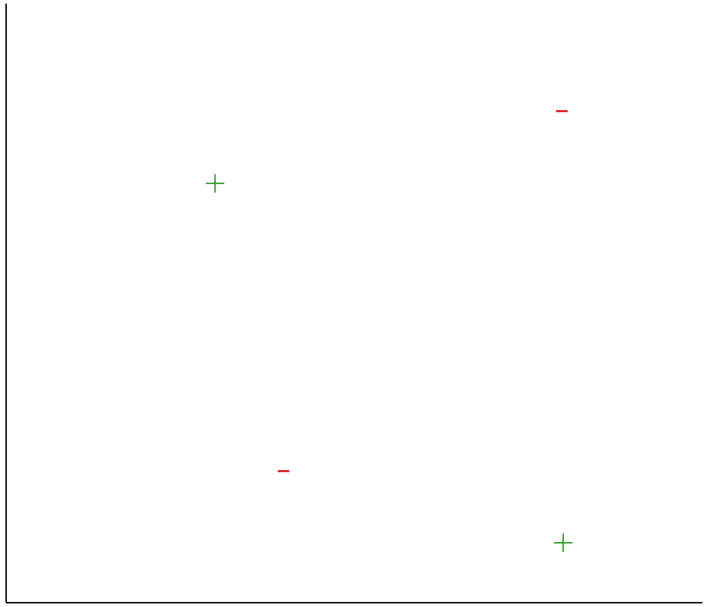
Biological Neural Network

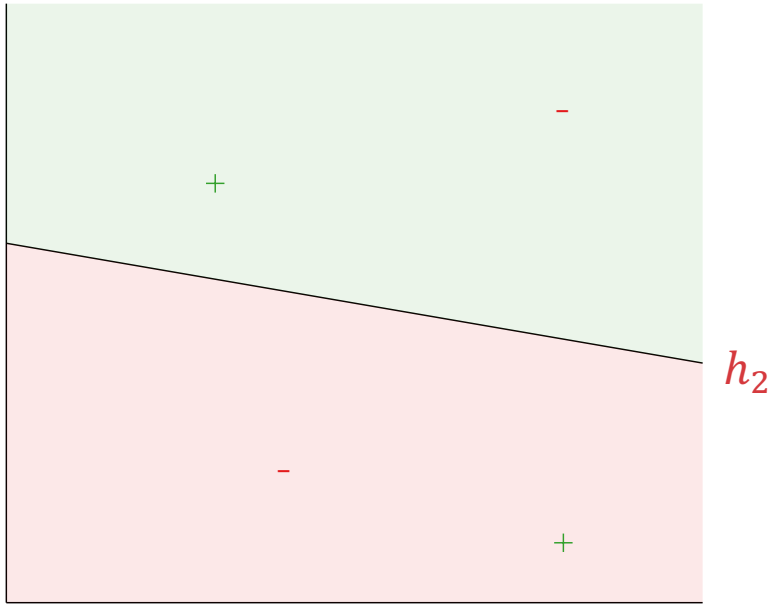
Source: https://science-art.com/image/?id=2971&m=168&pagename=neural_network_3d#.W_wrzJNKhUM

Recall: Perceptrons

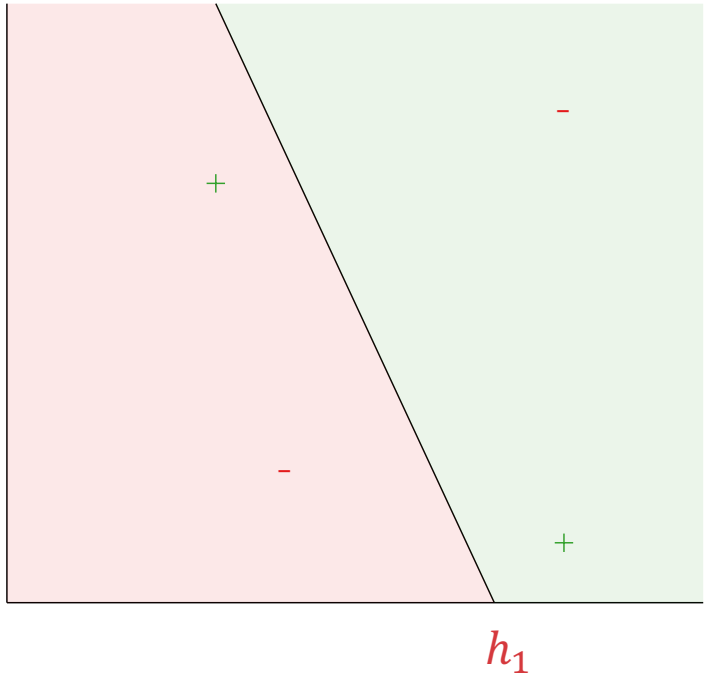
- Linear model for classification
- $h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})$
- Predictions are $+1$ or -1

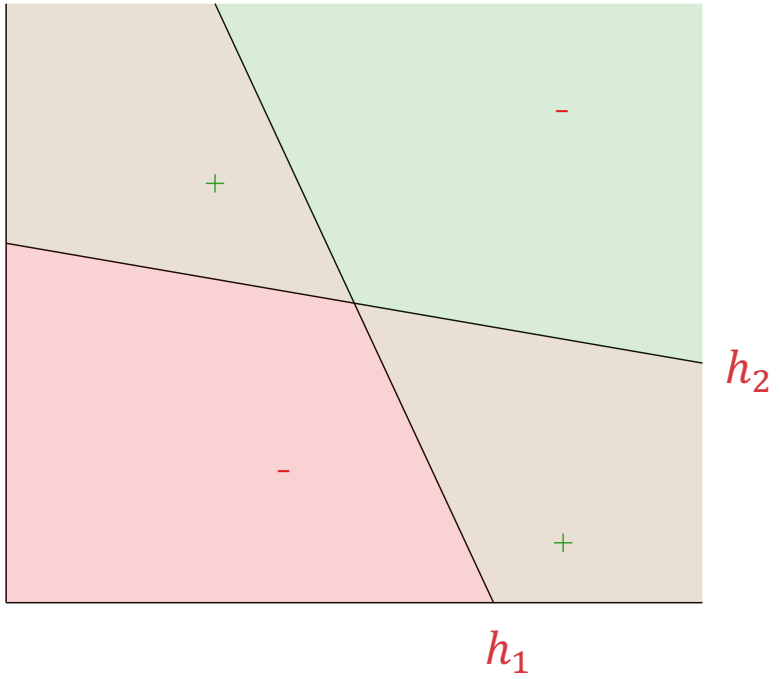


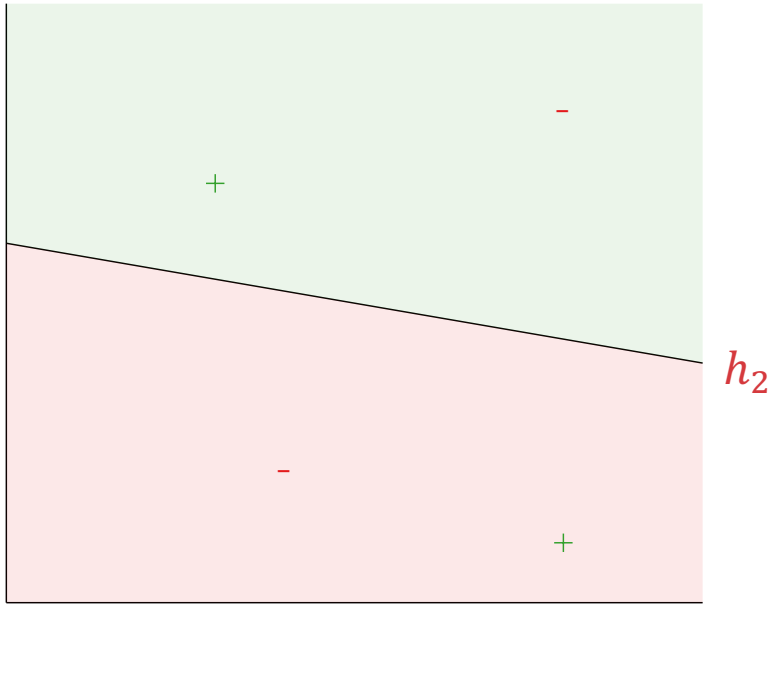




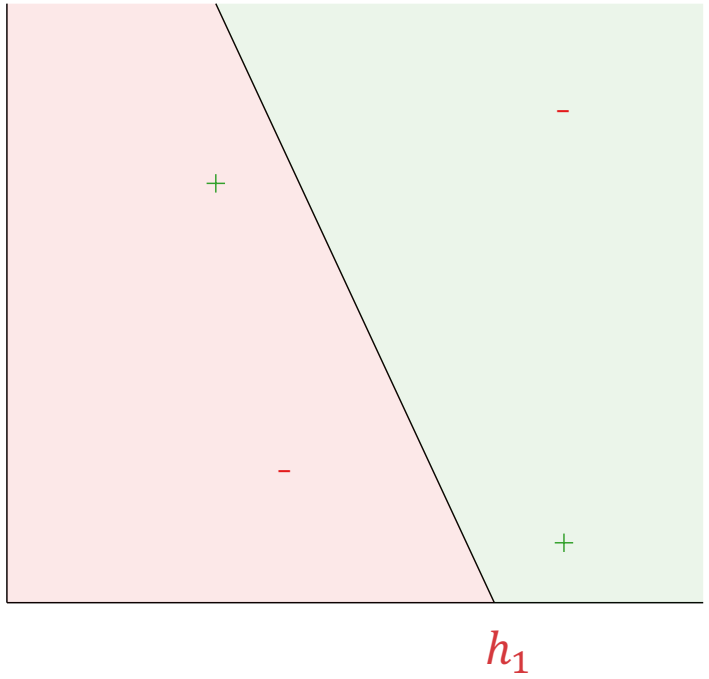
Perceptrons

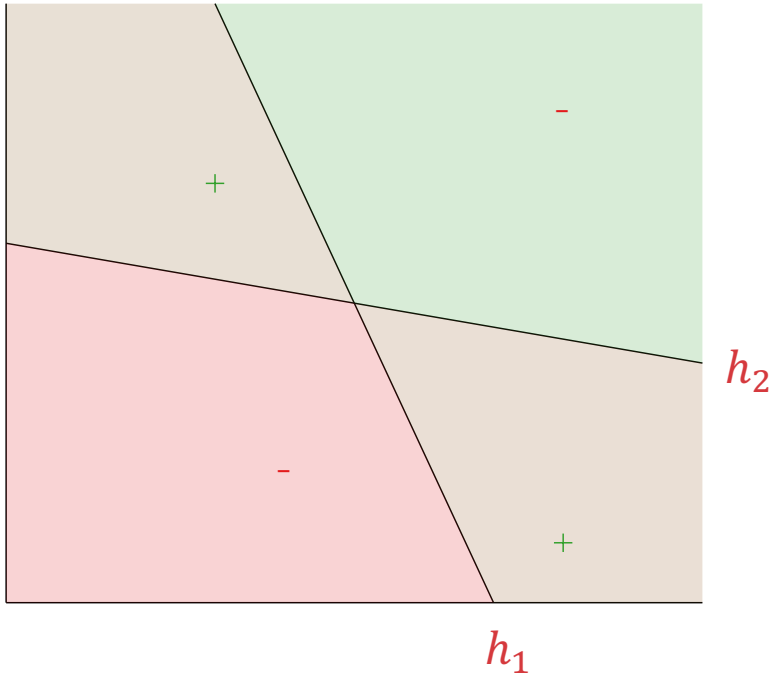


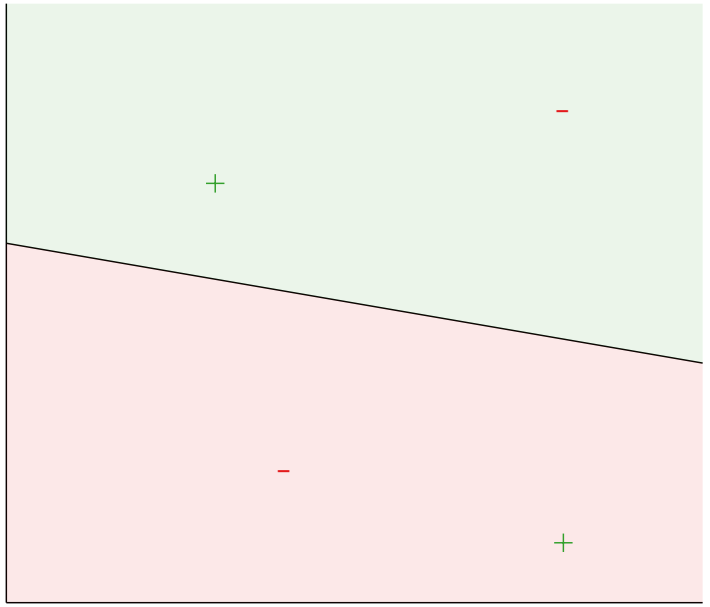




Combining Perceptrons





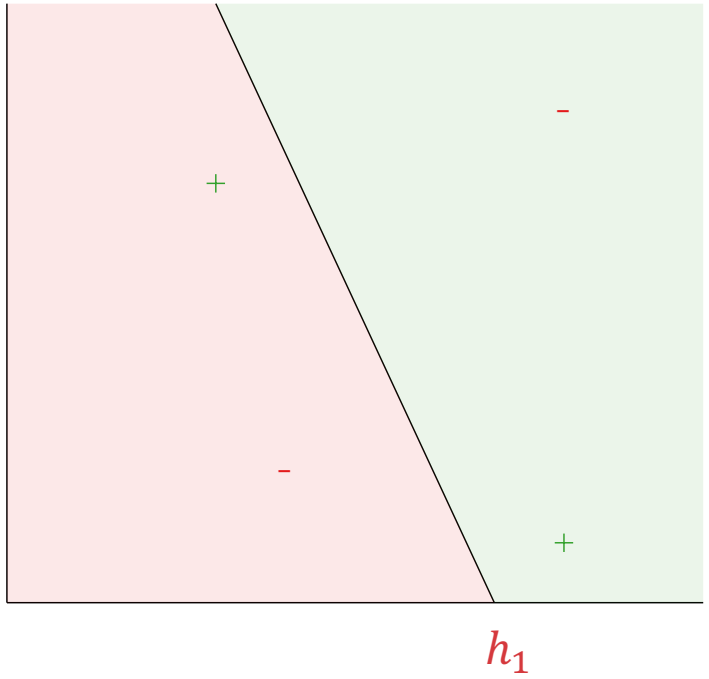


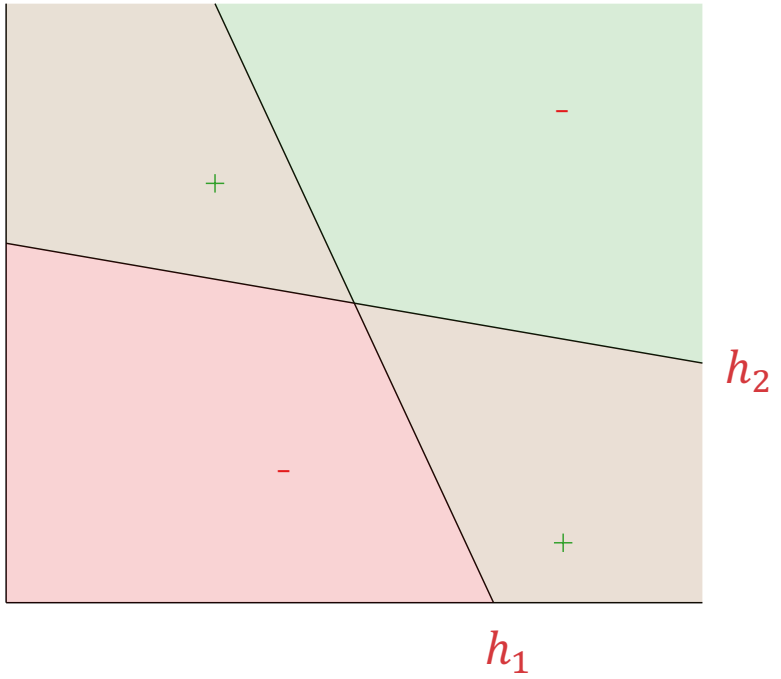
$$g(\vec{x}) = \begin{cases} +1 \text{ if } (h_1(\vec{x}) = +1 \text{ and } h_2(\vec{x}) = -1) \text{ or } (h_1(\vec{x}) = -1 \text{ and } h_2(\vec{x}) = +1) \\ -1 \text{ otherwise} \end{cases}$$

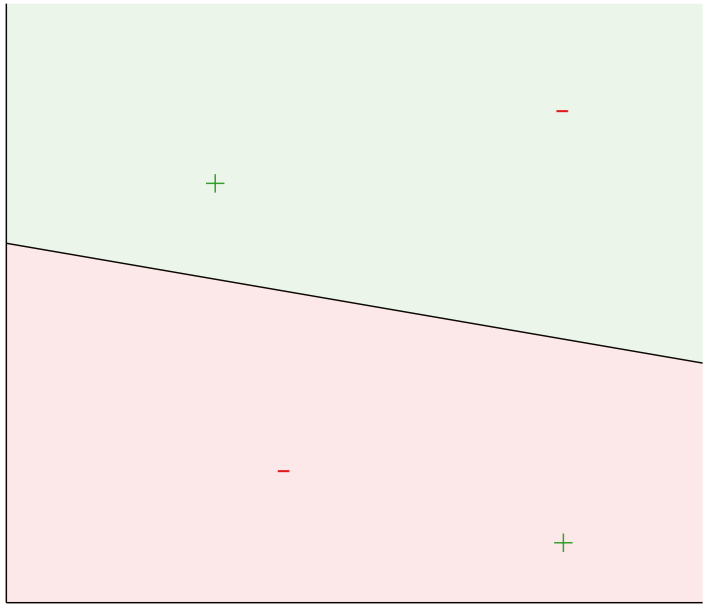
Boolean Algebra

- Boolean variables are either $+1$ ("true") or -1 ("false")

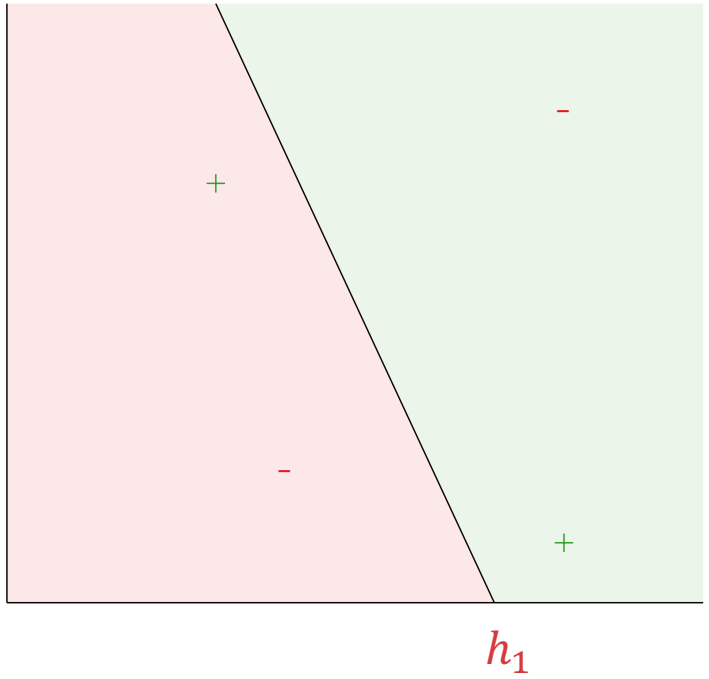
- Basic Boolean operations
  - Negation: $\bar{z} = -1 * z$
  - And: $AND(z_1, z_2) = \begin{cases} +1 \text{ if both } z_1 \text{ and } z_2 \text{ equal } +1 \\ -1 \text{ otherwise} \end{cases}$
  - Or: $OR(z_1, z_2) = \begin{cases} +1 \text{ if at least one of } z_1, z_2 \text{ equals } +1 \\ -1 \text{ otherwise} \end{cases}$

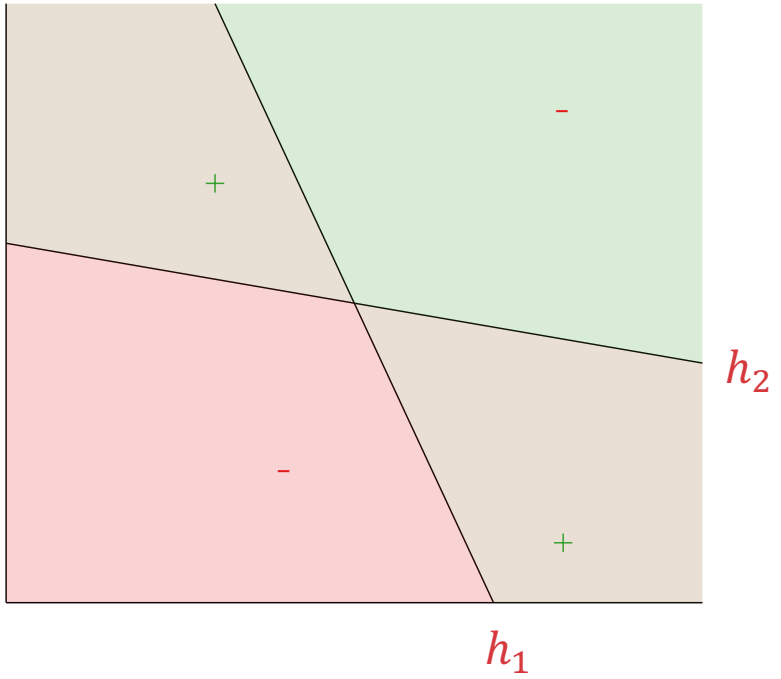


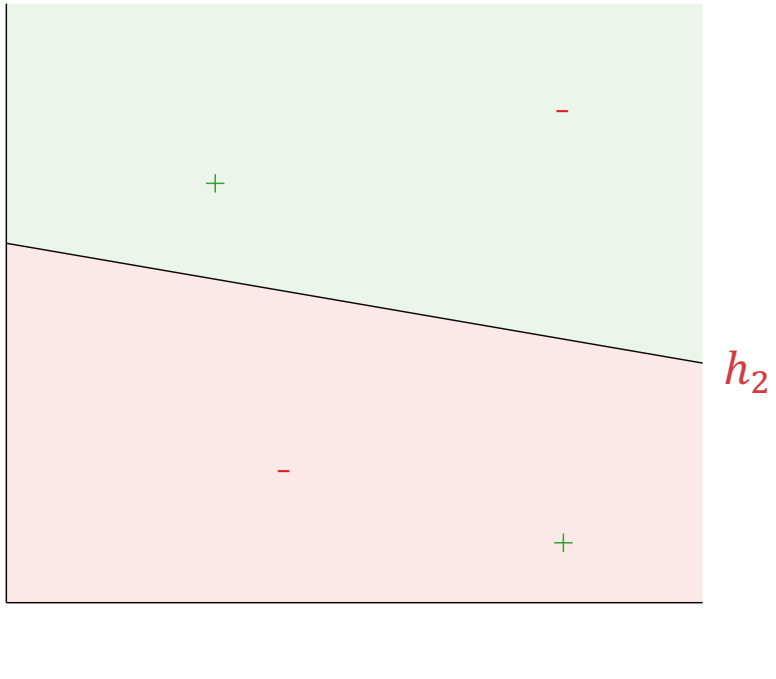




$$g(\vec{x}) = \begin{cases} +1 \text{ if } (h_1(\vec{x}) = +1 \text{ and } h_2(\vec{x}) = -1) \text{ or } (h_1(\vec{x}) = -1 \text{ and } h_2(\vec{x}) = +1) \\ -1 \text{ otherwise} \end{cases}$$







$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$

Boolean Algebra

- Boolean variables are either $+1$ ("true") or -1 ("false")

- Basic Boolean operations
  - Negation: $\bar{z} = -1 * z$
  - And: $AND(z_1, z_2) = \begin{cases} +1 \text{ if both } z_1 \text{ and } z_2 \text{ equal } +1 \\ -1 \text{ otherwise} \end{cases}$
  - Or: $OR(z_1, z_2) = \begin{cases} +1 \text{ if at least one of } z_1, z_2 \text{ equals } +1 \\ -1 \text{ otherwise} \end{cases}$

Boolean Algebra

- Boolean variables are either $+1$ ("true") or -1 ("false")
- Basic Boolean operations
  - Negation: $\bar{z} = -1 * z$
  - And: $AND(z_1, z_2) = \text{sign}(z_1 + z_2 - 1.5)$
  - Or: $OR(z_1, z_2) = \text{sign}(z_1 + z_2 + 1.5)$

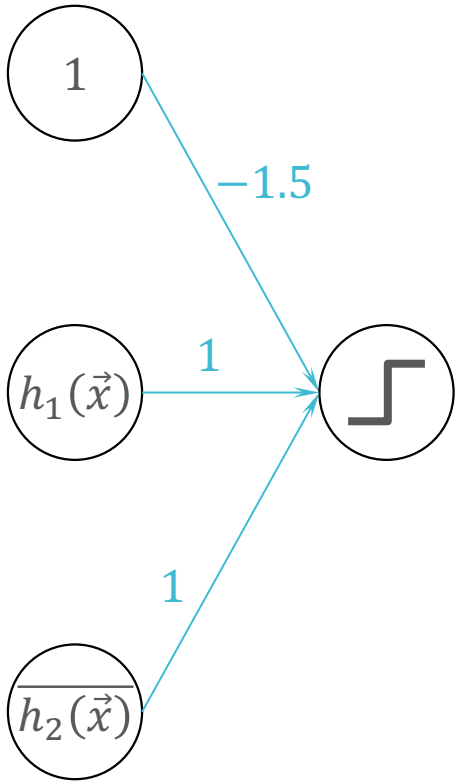
Boolean Algebra

- Boolean variables are either $+1$ ("true") or -1 ("false")

- Basic Boolean operations
  - Negation: $\bar{z} = -1 * z$
  - And: $AND(z_1, z_2) = \text{sign}\left([-1.5, 1, 1]\begin{bmatrix}1\\z_1\\z_2\end{bmatrix}\right)$
  - Or: $OR(z_1, z_2) = \text{sign}\left([1.5, 1, 1]\begin{bmatrix}1\\z_1\\z_2\end{bmatrix}\right)$

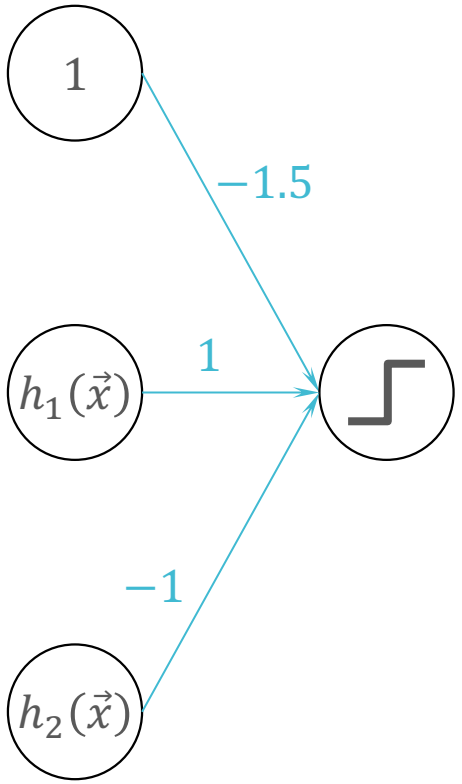
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



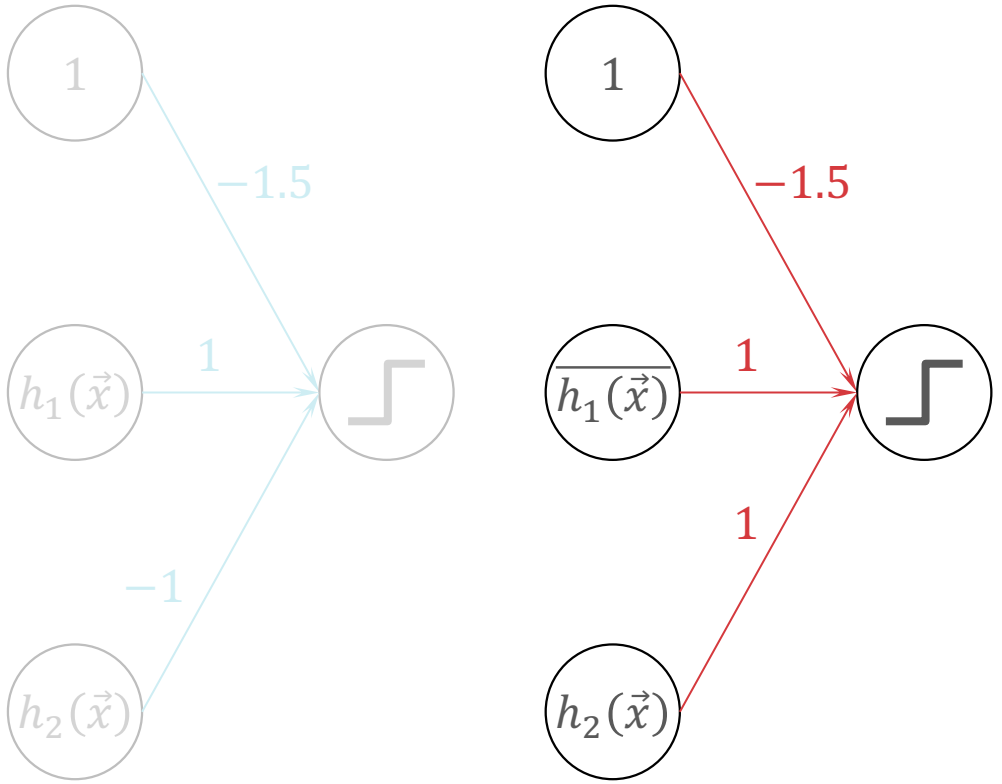
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



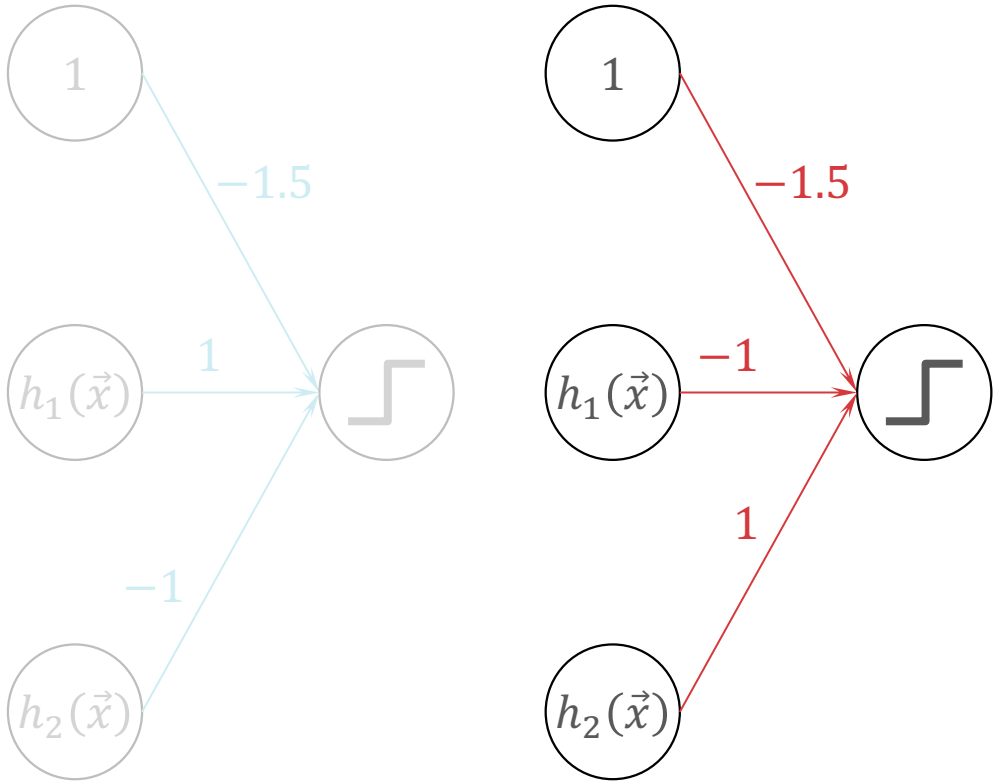
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



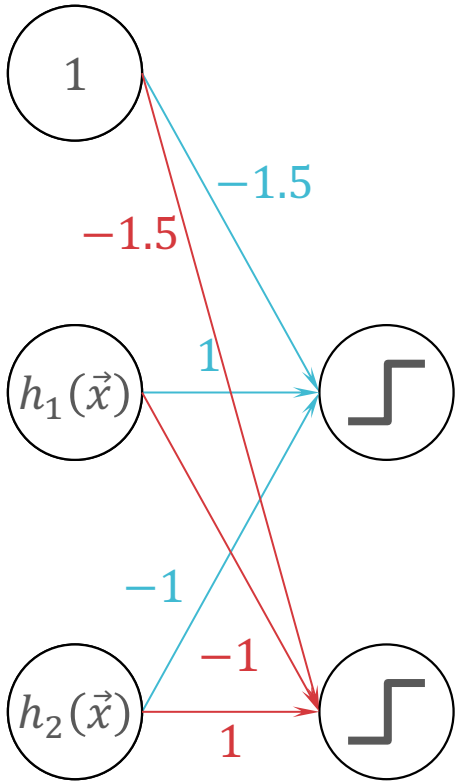
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



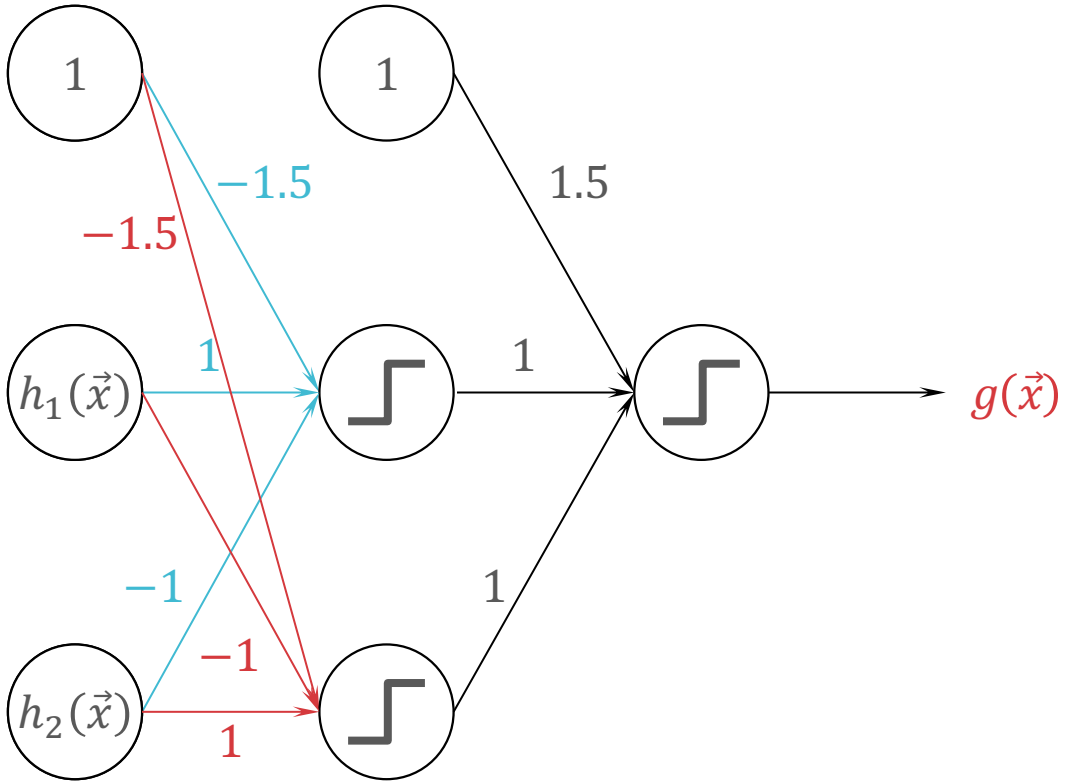
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



Building a Network

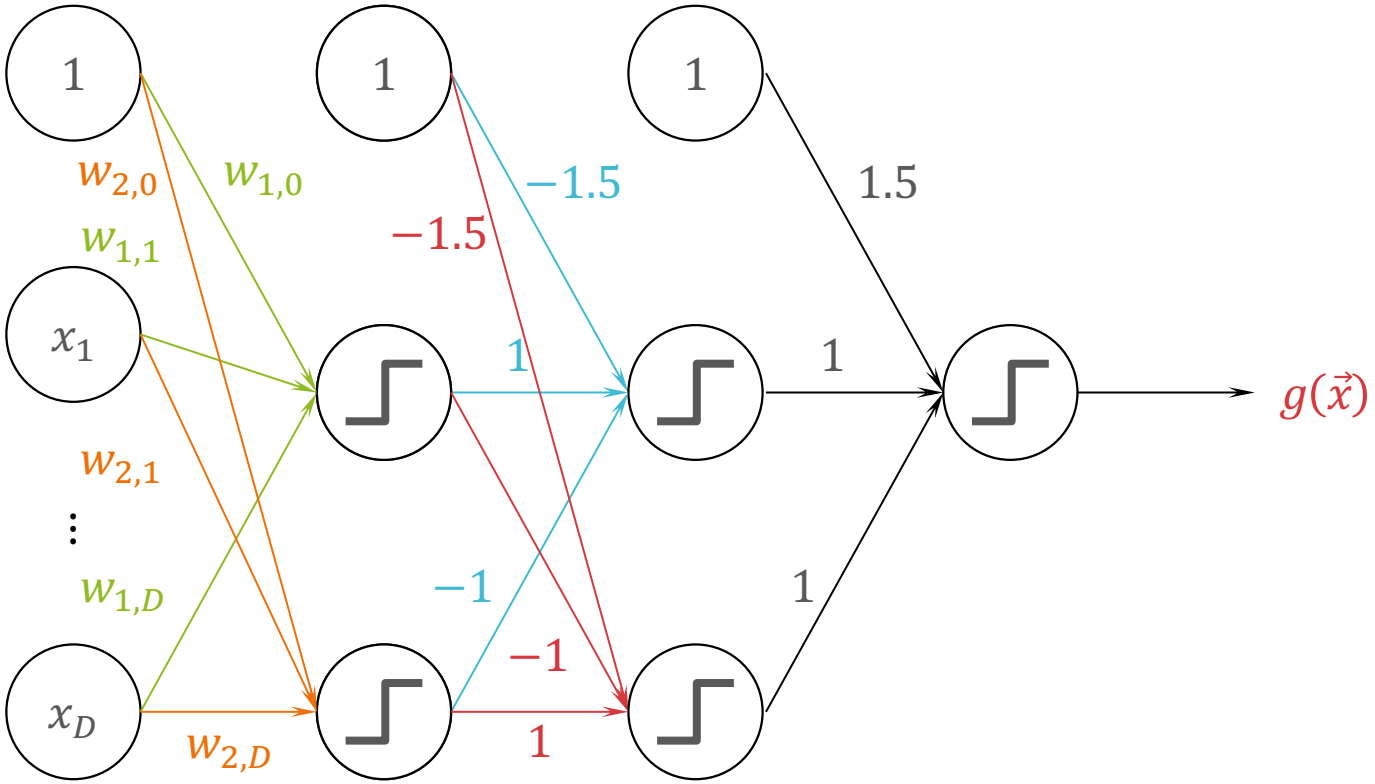
$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



$$h_i(\vec{x}) = \text{sign}\left(\vec{w_i}^T \vec{x}\right) = \text{sign}\left(\sum_{d=0}^{D} w_{i,d}\, x_d\right)$$

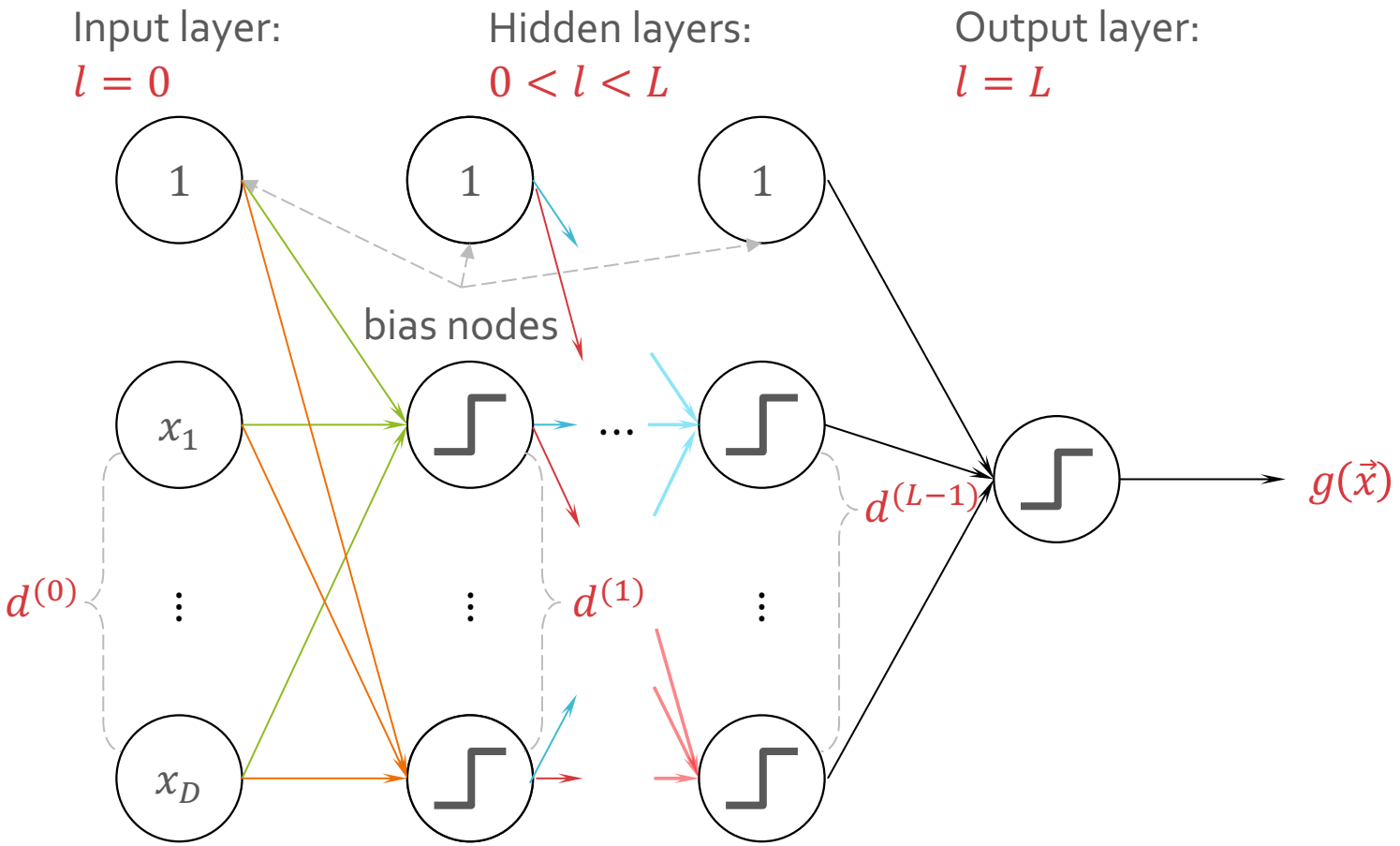
Building a Network

$$g(\vec{x}) = OR\left(AND\left(h_1(\vec{x}), \overline{h_2(\vec{x})}\right), AND\left(\overline{h_1(\vec{x})}, h_2(\vec{x})\right)\right)$$



$$g(\vec{x}) = \text{sign}\left(\text{sign}\left(\text{sign}\left(\overrightarrow{w_1}^T\vec{x}\right) - \text{sign}\left(\overrightarrow{w_2}^T\vec{x}\right) - 1.5\right) + \text{sign}\left(-\text{sign}\left(\overrightarrow{w_1}^T\vec{x}\right) + \text{sign}\left(\overrightarrow{w_2}^T\vec{x}\right) - 1.5\right) + 1.5\right)$$

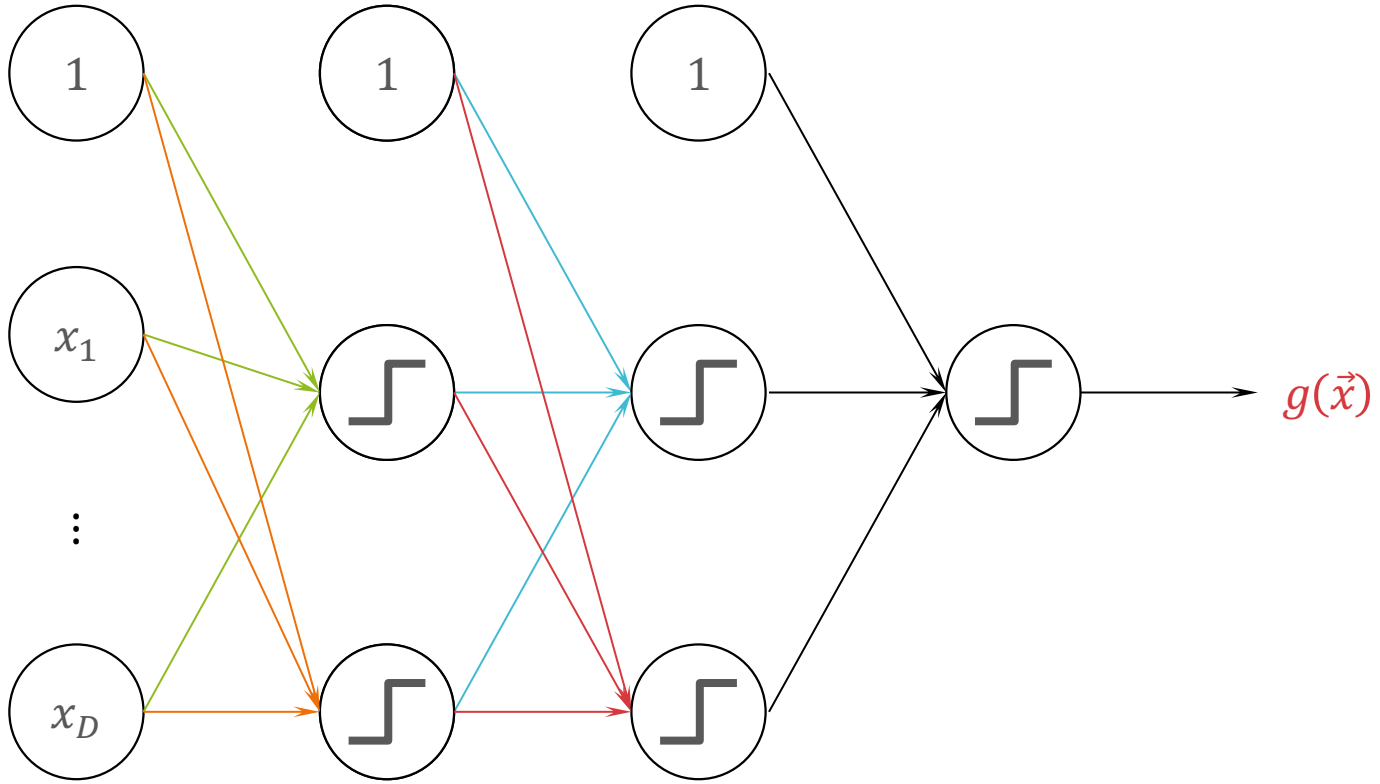
Multi-Layer Perceptron (MLP)



Layer l has dimension $d^{(l)}$ → Layer l has $d^{(l)} + 1$ nodes, counting the bias node

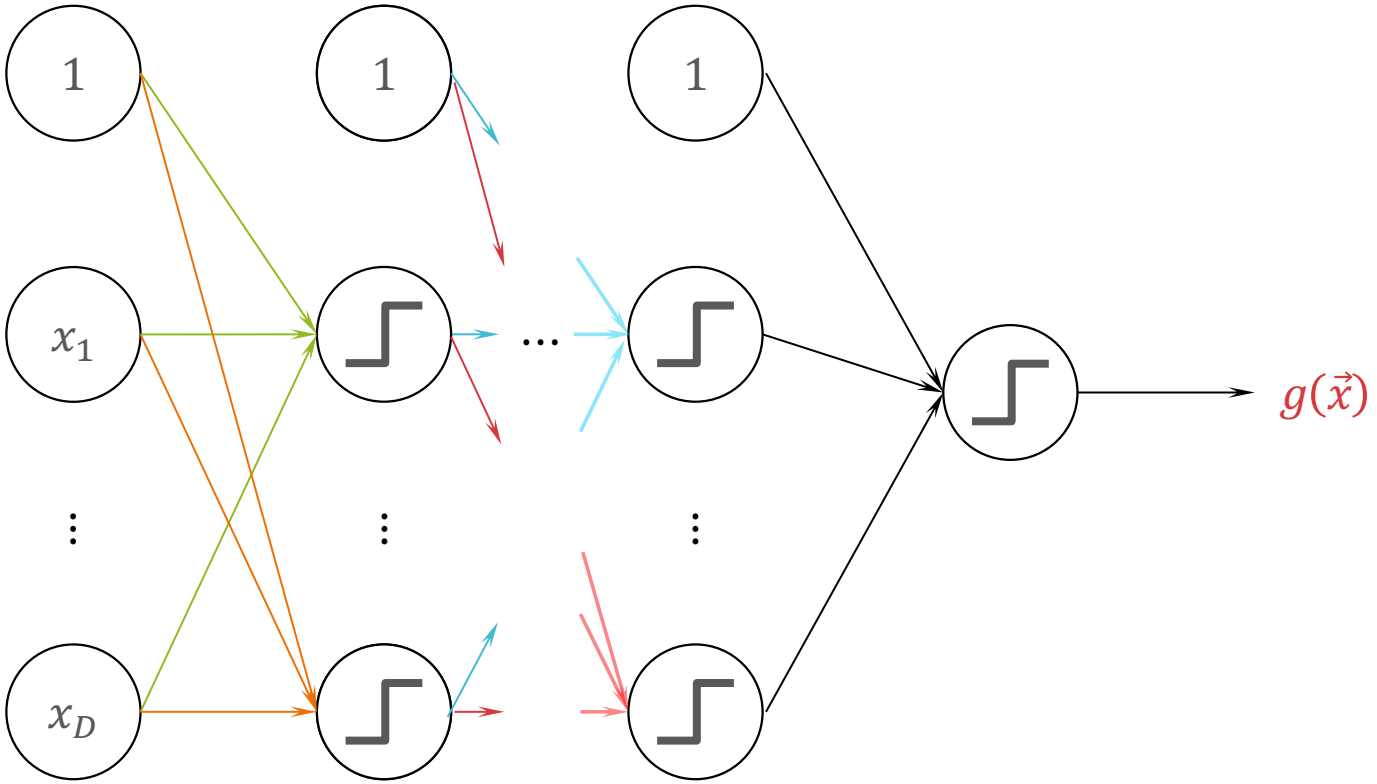
Example MLP

3-layer MLP ... with 2 hidden layers ... and 1 output layer



Architecture

The architecture of an MLP is the vector $\vec{d} = \left[d^{(0)}, d^{(1)}, \ldots, d^{(L)}\right]$

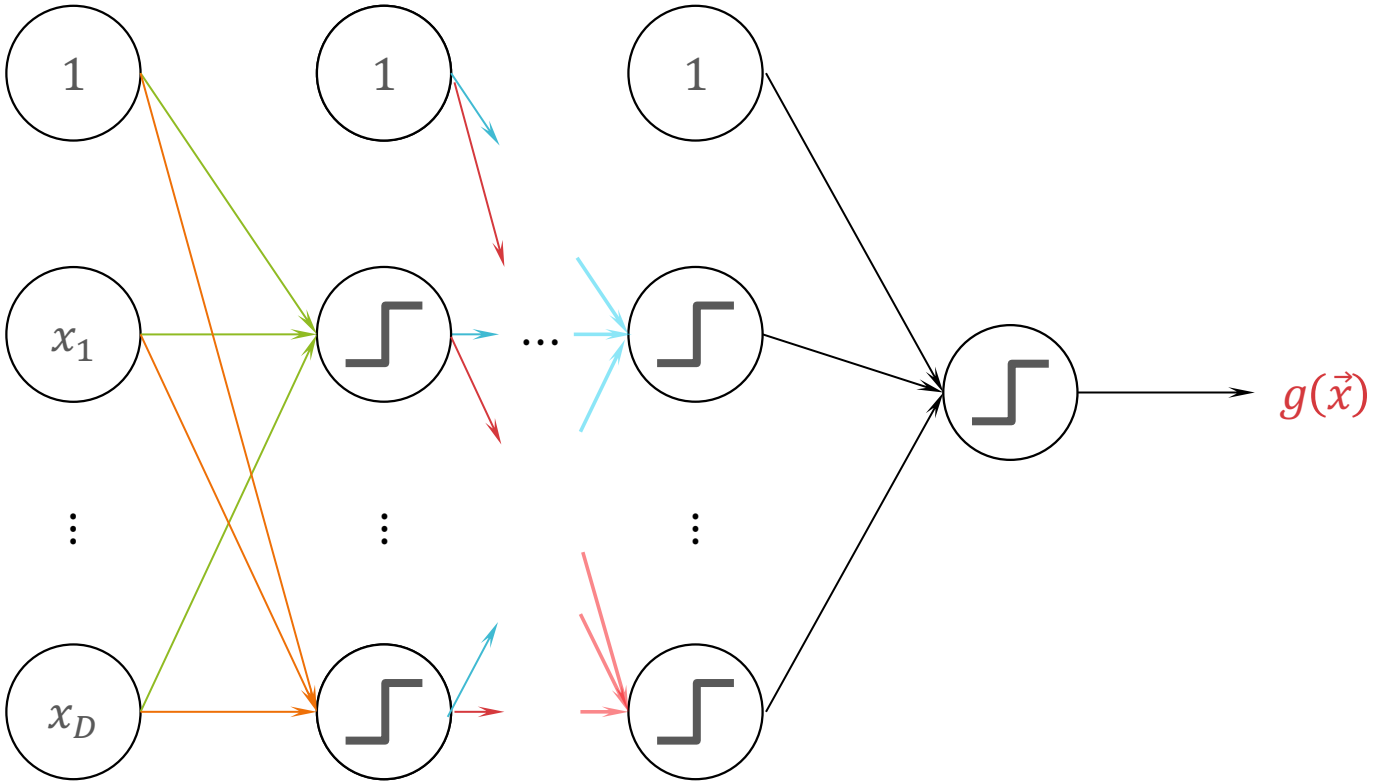


Every architecture corresponds to a hypothesis set: $\mathcal{H}_{MLP}\left(\vec{d}\right)$

A hypothesis $h \in \mathcal{H}_{MLP}\left(\vec{d}\right)$ is specified by setting all the weights between nodes

Weights

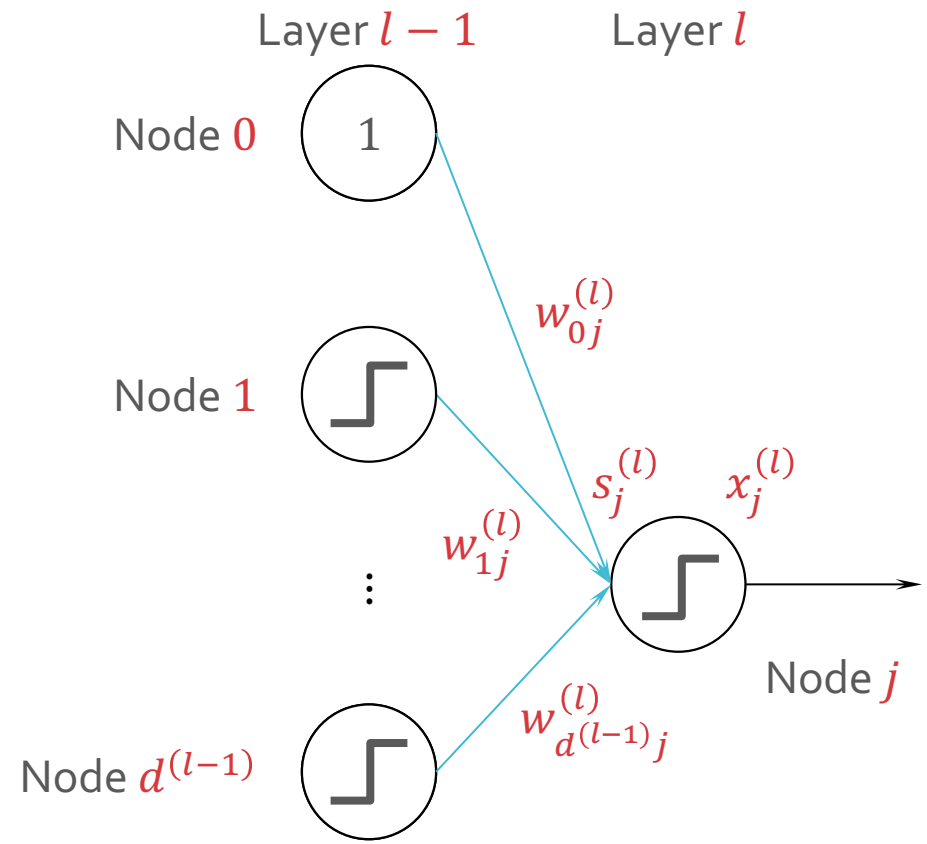
The weights between layer $l-1$ and layer l are a matrix: $W^{(l)} \in \mathbb{R}^{(d^{(l-1)}+1) \times d^{(l)}}$



$w_{ij}^{(l)}$ is the weight between node i in layer $l-1$ and node j in layer l

Signal and Outputs

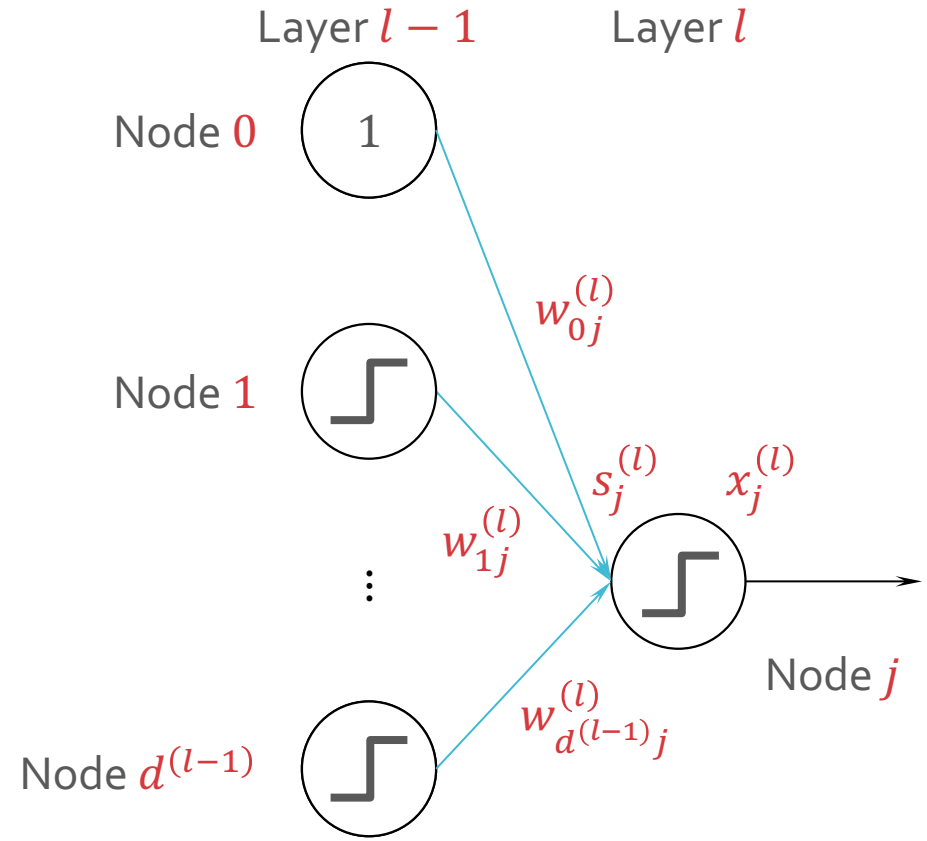
Every node has an incoming signal and outgoing output



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

Signal and Outputs

Every node has an incoming signal and outgoing output



$$\overrightarrow{x^{(l)}} = \begin{bmatrix} 1 \\ \text{sign}\left(\overrightarrow{s^{(l)}}\right) \end{bmatrix} \text{ and } \overrightarrow{s^{(l)}} = {W^{(l)}}^T \overrightarrow{x^{(l-1)}}$$

Recall: Binary Error

$$E_{in}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}[\![\text{sign}(\vec{w}^T\vec{x_i}) \neq y_i]\!]$$

awkward silence

Recall: Binary Error

$$E_{in}\left(W^{(1)}, \ldots, W^{(L)}\right) = \frac{1}{n}\sum_{i=1}^{n} [\![\text{sign}\left(W^{(L)^T}\text{sign}(\ldots)\right) \neq y_i]\!]$$

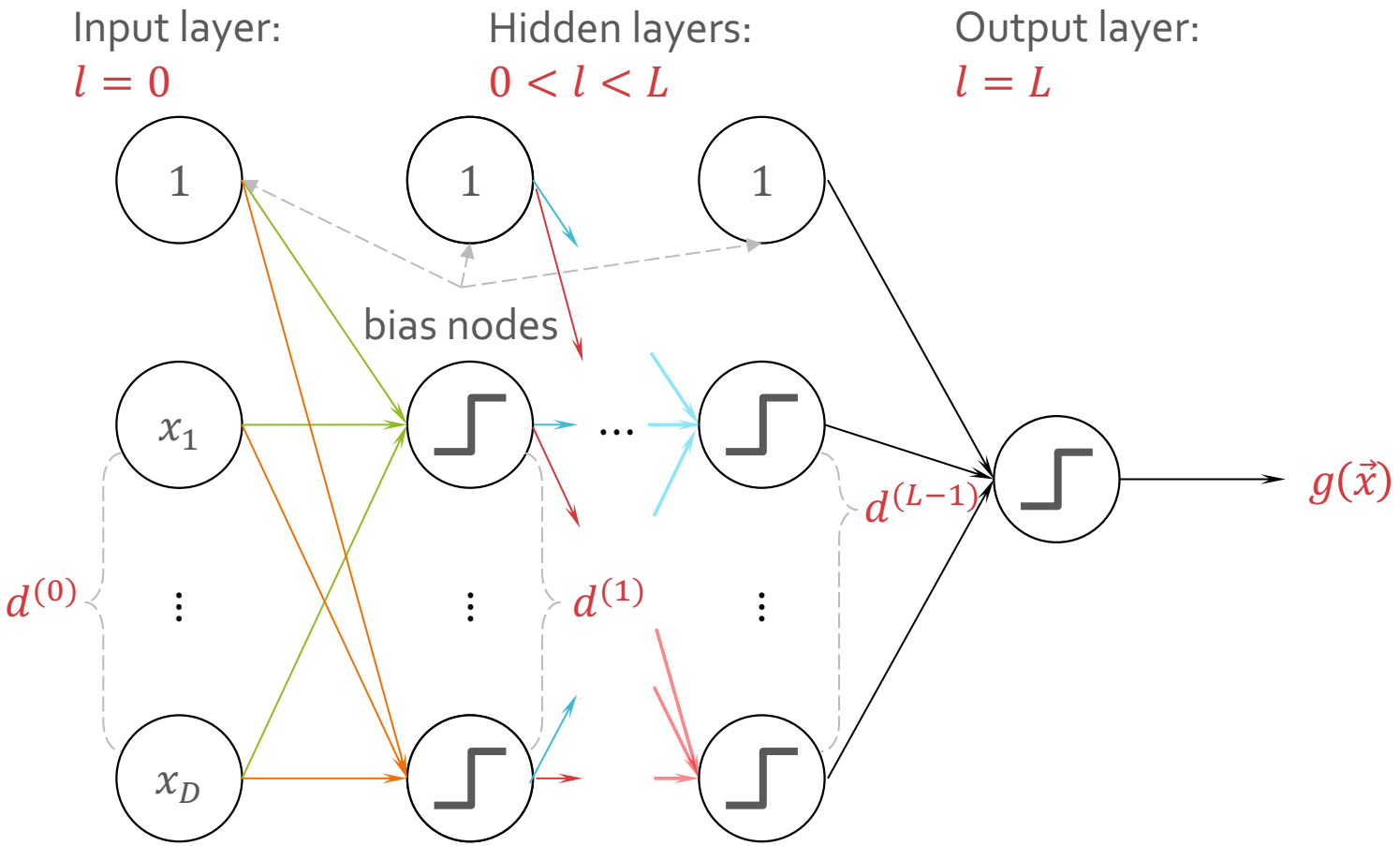
⋱

awkward silence intensifies

Softening MLPs

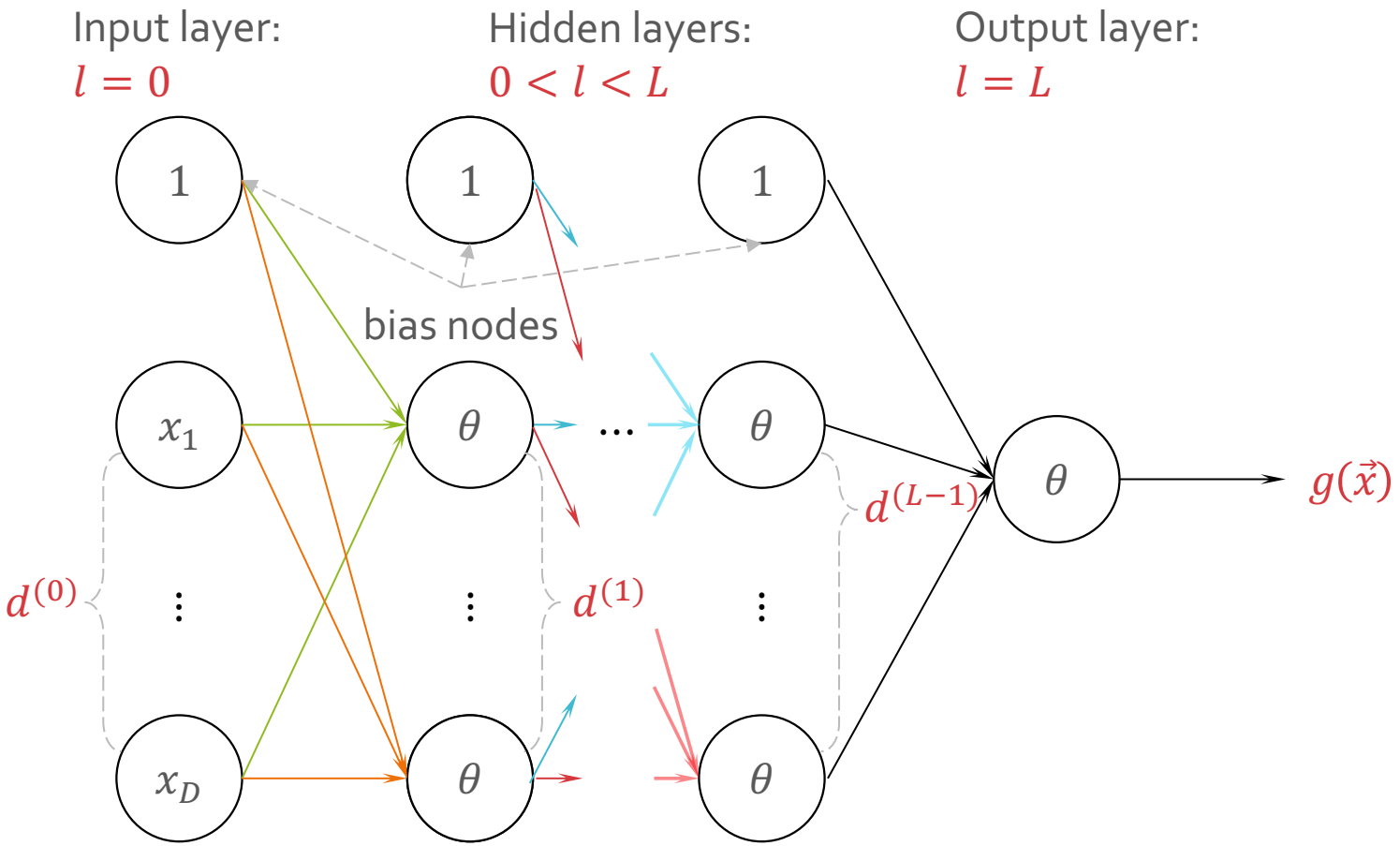
- Idea:
  - Replace the "hard" $\text{sign}(\cdot)$ function with a "soft", differentiable approximation, $\theta(\cdot)$
  - Learn all the weights $W^{(1)}, \ldots, W^{(L)}$ using $\theta(\cdot)$
  - To make predictions, use the learned weights and $\text{sign}(\cdot)$ at the output node to get valid predictions

Multi-Layer Perceptron (MLP)



Layer l has dimension $d^{(l)}$ → Layer l has $d^{(l)} + 1$ nodes, counting the bias node

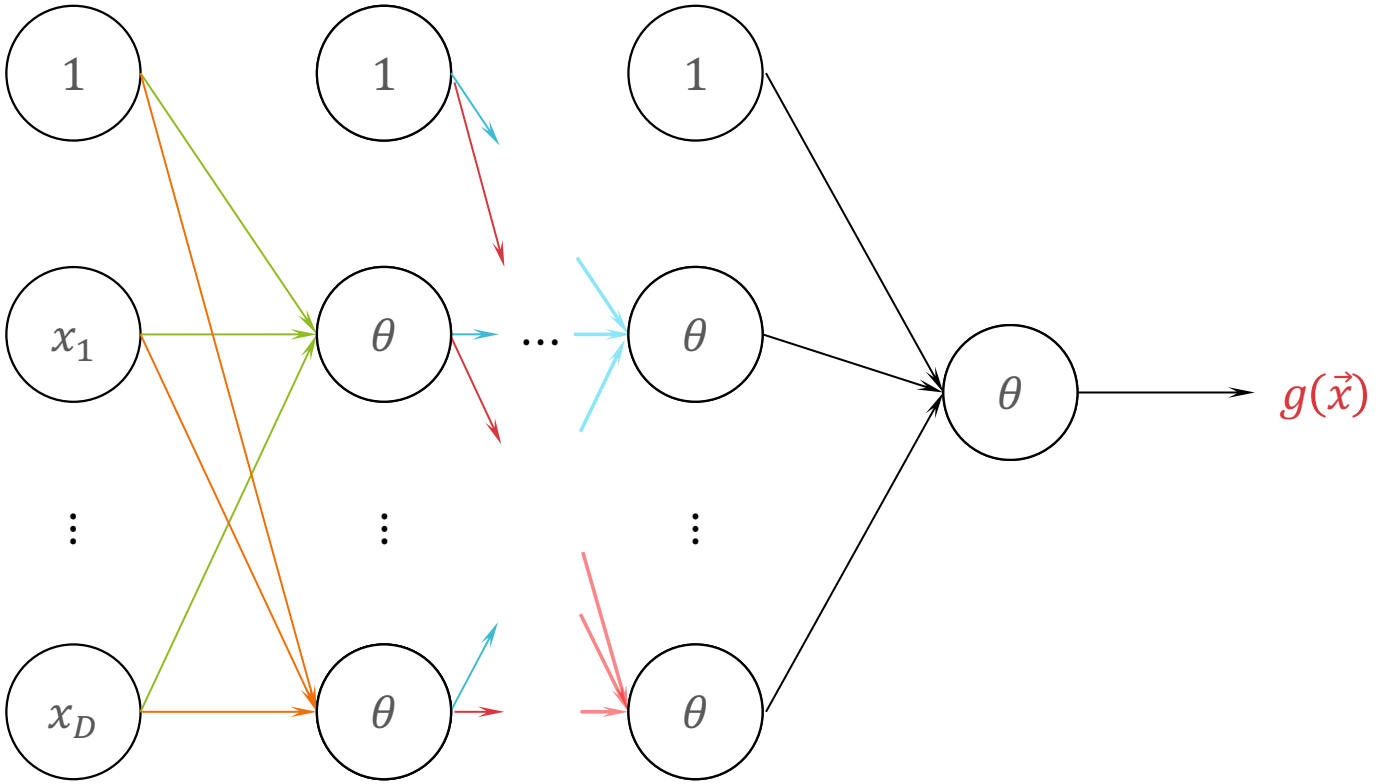
Feed-Forward Neural Network (NN)



Layer l has dimension $d^{(l)}$ → Layer l has $d^{(l)} + 1$ nodes, counting the bias node

Architecture

The architecture of a NN is the vector $\vec{d} = \left[d^{(0)}, d^{(1)}, \ldots, d^{(L)}\right]$

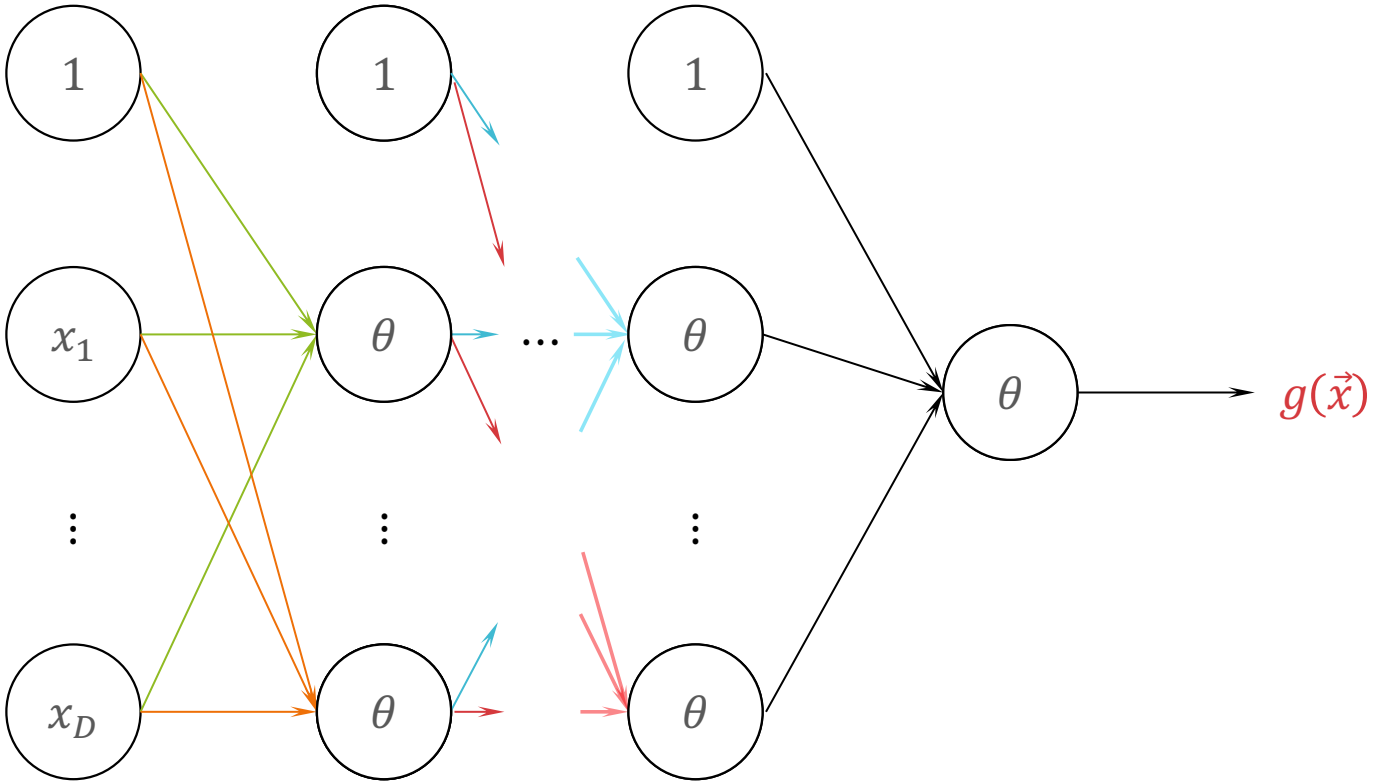


Every architecture corresponds to a hypothesis set: $\mathcal{H}_{NN}(\vec{d})$

A hypothesis $h \in \mathcal{H}_{NN}(\vec{d})$ is specified by setting all the weights between nodes

Weights

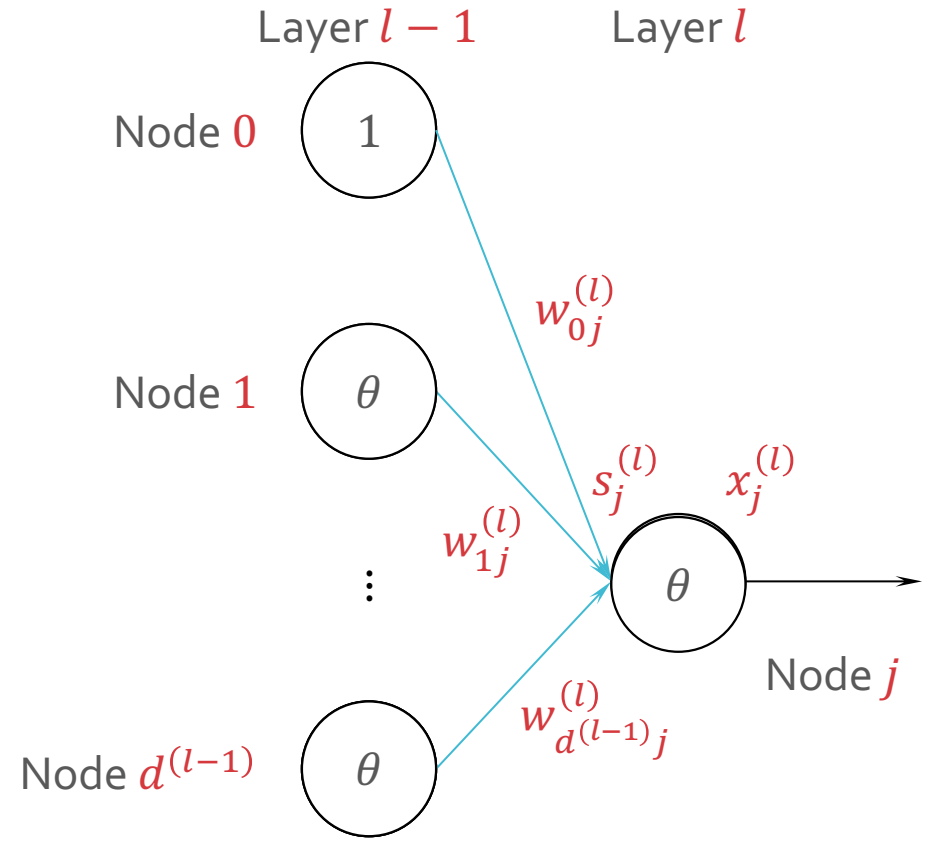
The weights between layer $l-1$ and layer l are a matrix: $W^{(l)} \in \mathbb{R}^{(d^{(l-1)}+1) \times d^{(l)}}$



$w_{ij}^{(l)}$ is the weight between node i in layer $l-1$ and node j in layer l

Signal and Outputs

Every node has an incoming signal and outgoing output



$$\overrightarrow{x^{(l)}} = \left[\theta\left(\overrightarrow{s^{(l)}}\right)\right] \text{ and } \overrightarrow{s^{(l)}} = {W^{(l)}}^T \overrightarrow{x^{(l-1)}}$$

Forward Propagation

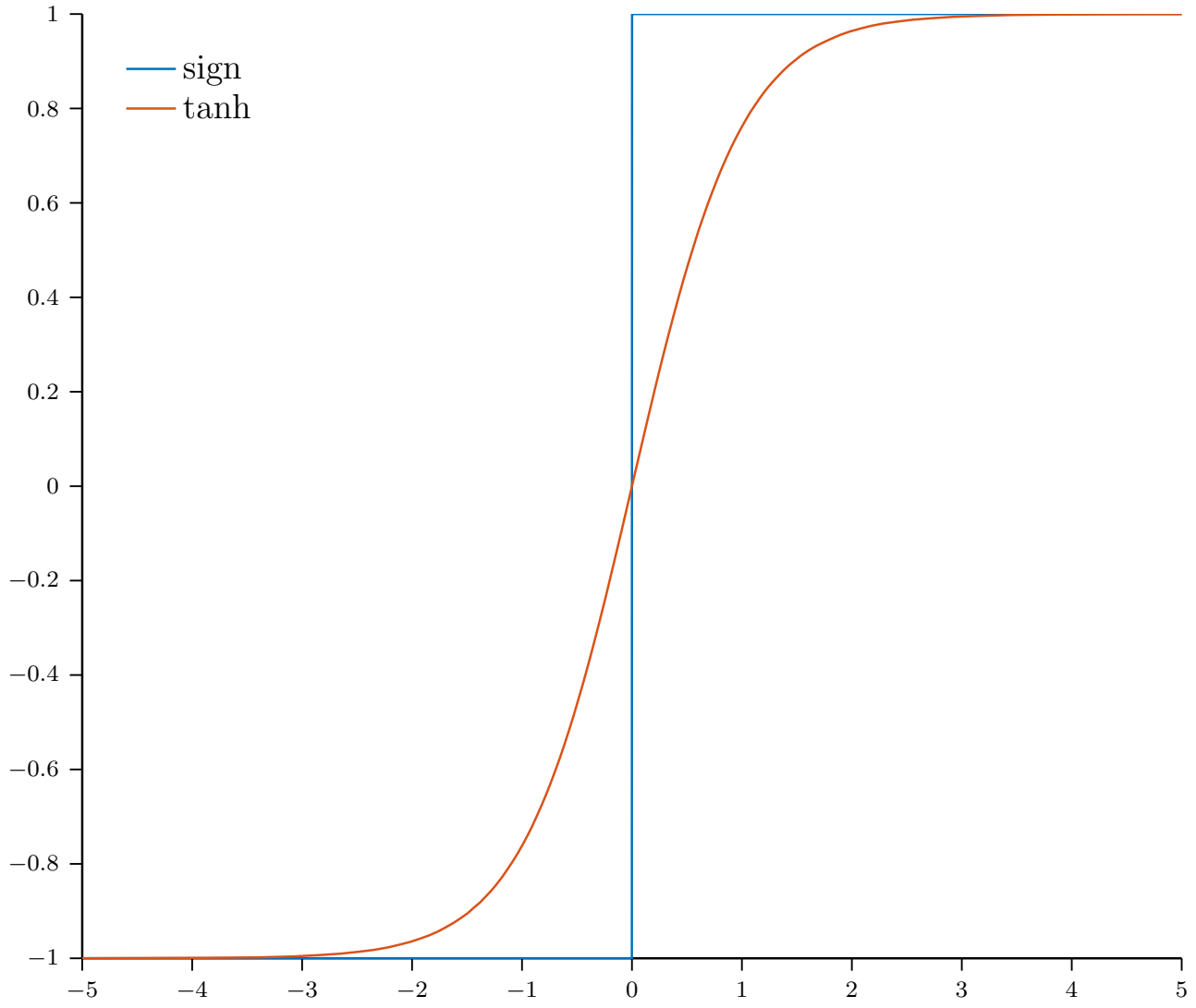
- Input: weights $W^{(1)}, \dots, W^{(L)}$ and a query point $\vec{x}$
- Initialize $\overrightarrow{x^{(0)}} = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix}$
- For $l = 1, \dots, L$
  - $\overrightarrow{s^{(l)}} = {W^{(l)}}^T \overrightarrow{x^{(l-1)}}$
  - $\overrightarrow{x^{(l)}} = \begin{bmatrix} 1 \\ \theta\left(\overrightarrow{s^{(l)}}\right) \end{bmatrix}$
- Output: $h\left(\vec{x} \mid W^{(1)}, \dots, W^{(L)}\right) = \overrightarrow{x^{(L)}}$

$\theta(\cdot)$

- Hyperbolic tangent:

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- $$\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh(z)^2$$



Error of a Neural Network

$$E_{in}\left(W^{(1)}, \ldots, W^{(L)}\right) = \frac{1}{n}\sum_{i=1}^{n}\left(h\left(\overrightarrow{x_i} \mid W^{(1)}, \ldots, W^{(L)}\right) - y_i\right)^2$$

⋱

awkward silence

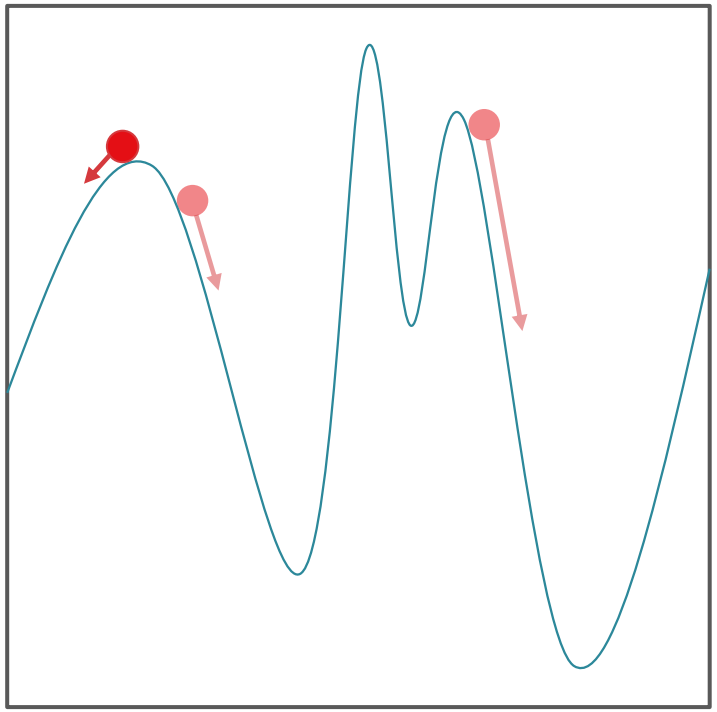
Recall: Cross-entropy Error

$$E_{in}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} \ln\left(1 + e^{-y_i \vec{w}^T \vec{x_i}}\right)$$

awkward silence

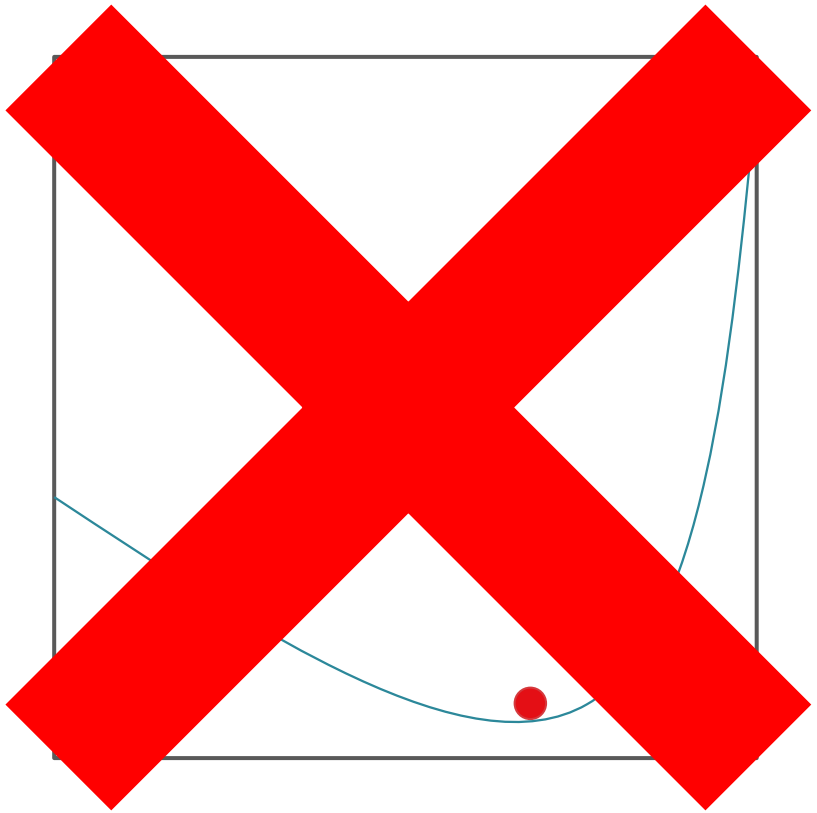
Recall: Gradient Descent

- Iterative method for minimizing functions
- Requires the gradient to exist everywhere



Recall: Gradient Descent

- Iterative method for minimizing functions
- Requires the gradient to exist everywhere



Gradient Descent for Neural Networks

- Input: $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), \dots, (\overrightarrow{x_n}, y_n)\}, \eta_0$
- Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While some termination condition is not satisfied
  - For $l = 1, \dots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} E_{in}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$ (???)
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$