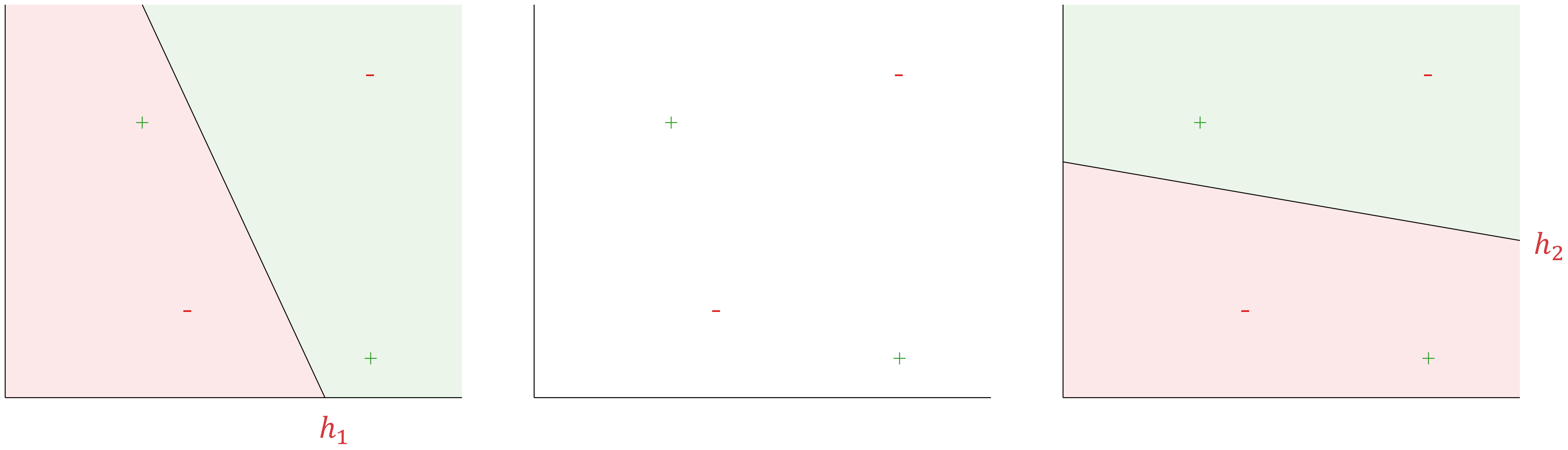
CSE 417T: Introduction to Machine Learning

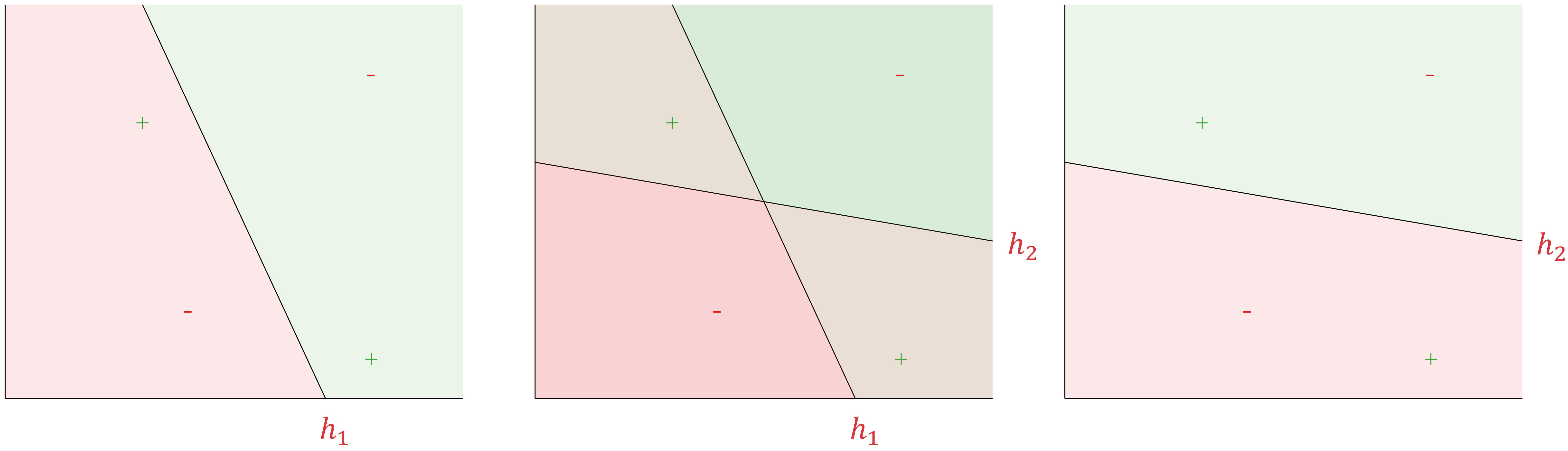
Lecture 25: Backpropagation

Henry Chai

11/29/18



Combining Perceptrons



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

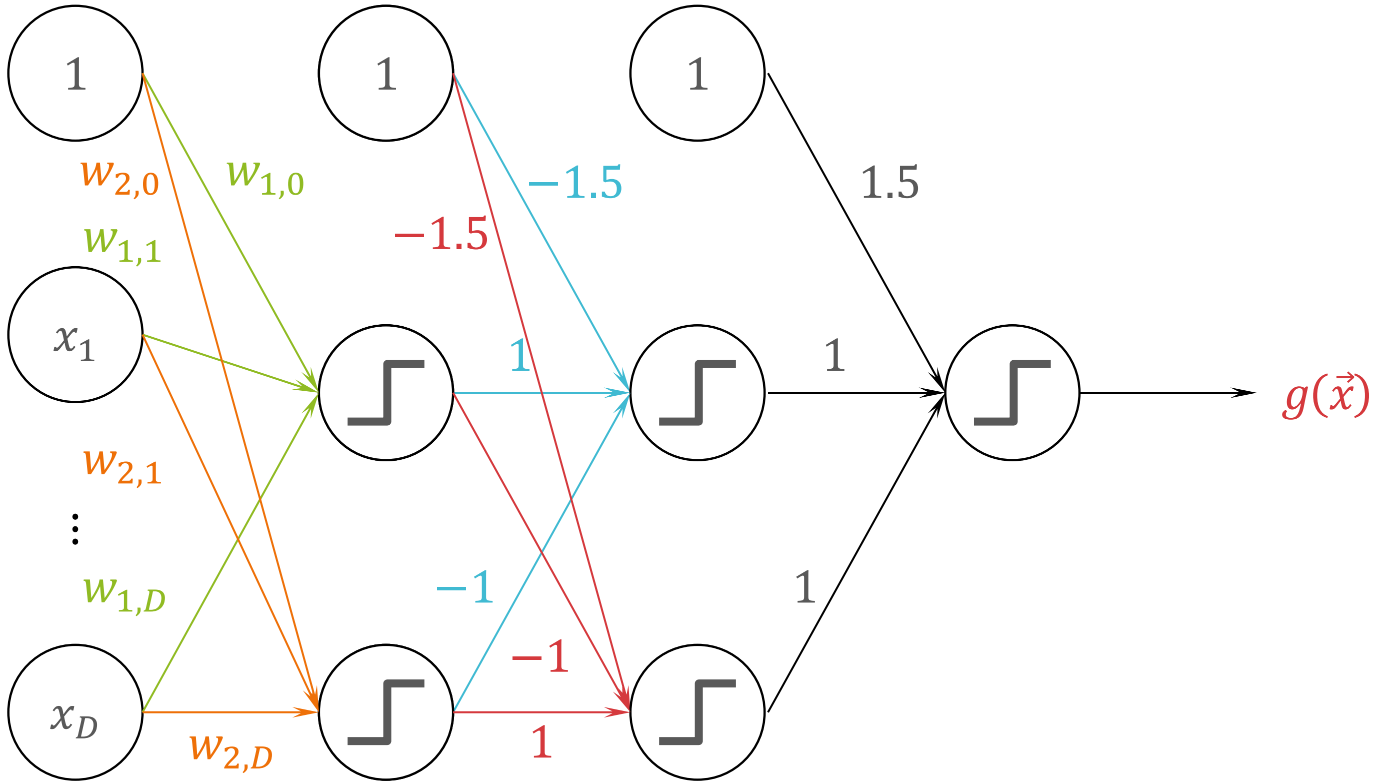
Boolean Algebra

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

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

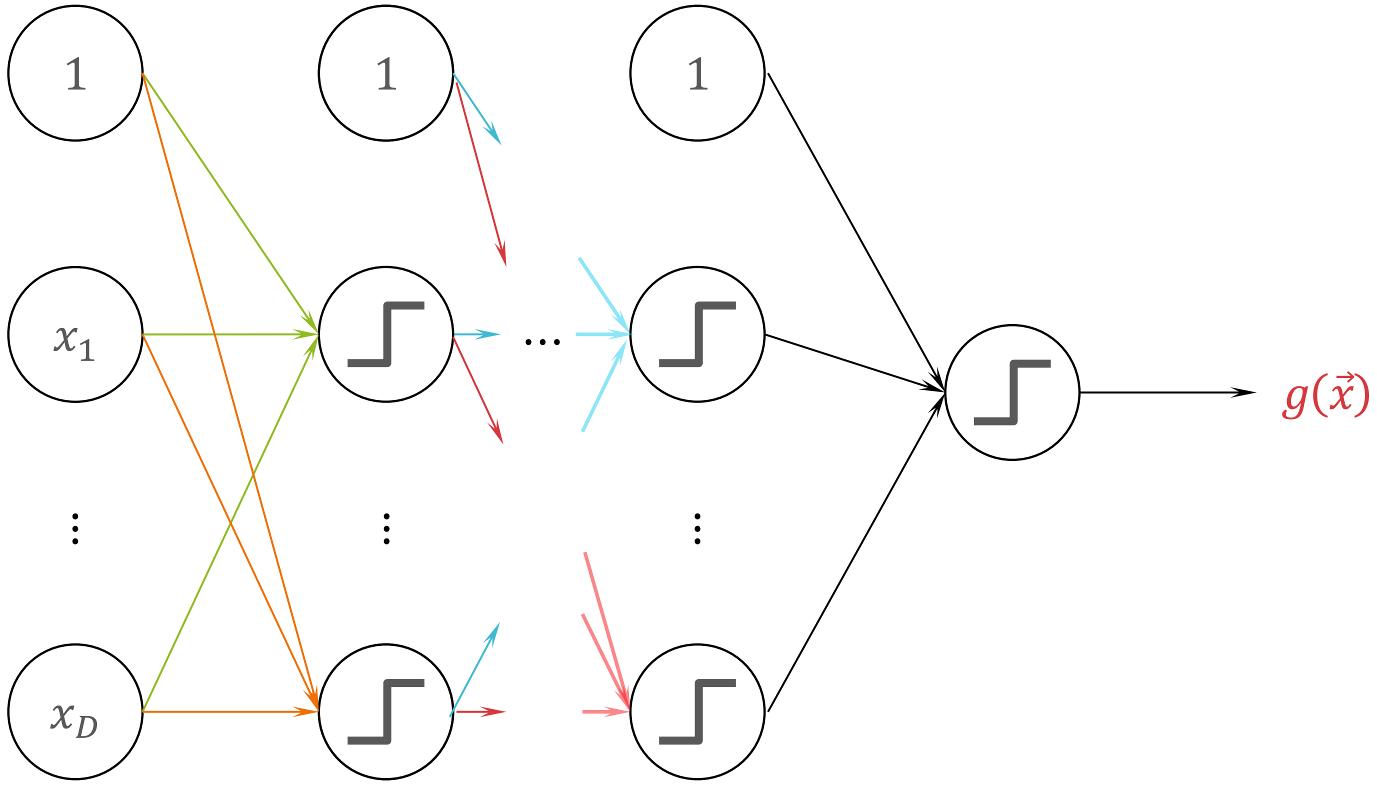
Building a Network

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

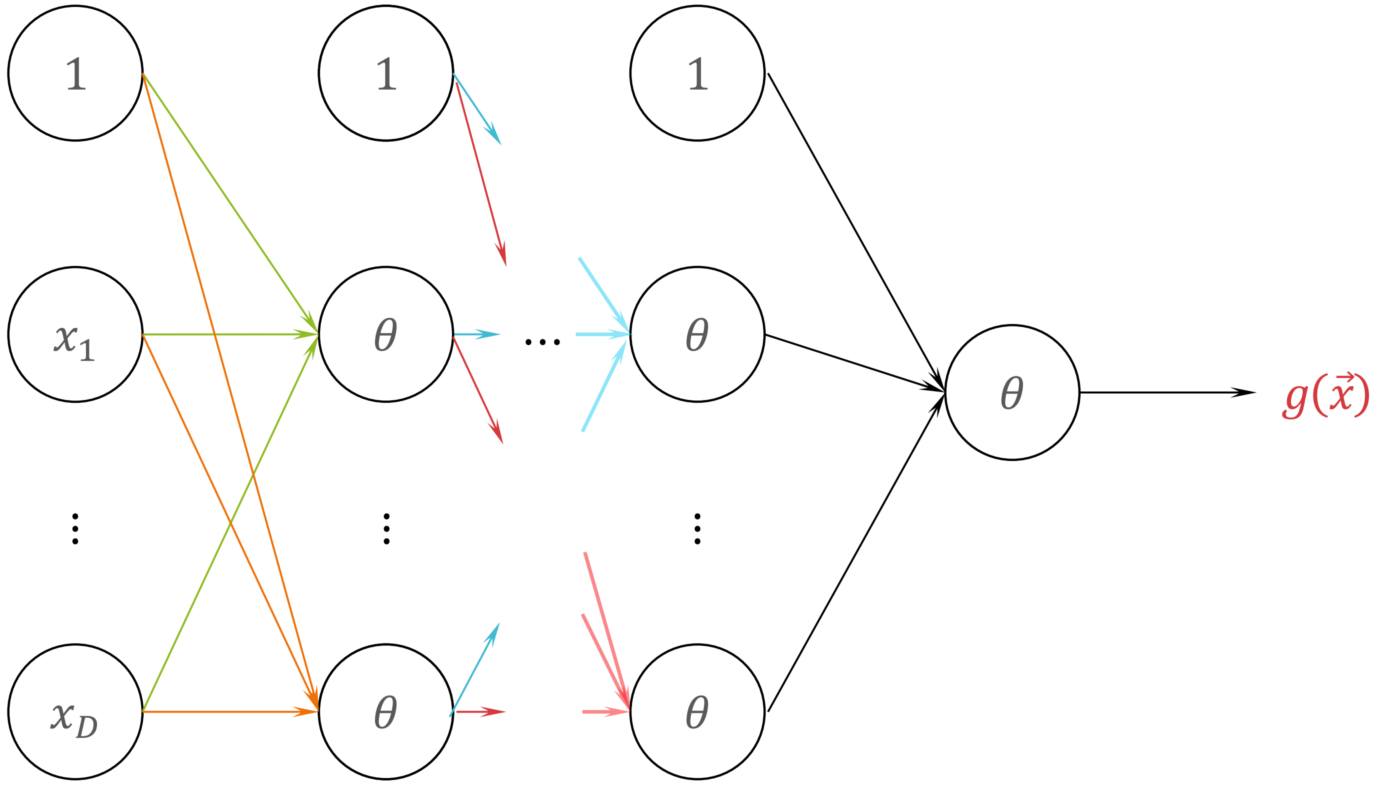


$$g(\vec{x}) = \text{sign}\left(\text{sign}\left(\text{sign}\left(\vec{w_1}^T\vec{x}\right) - \text{sign}\left(\vec{w_2}^T\vec{x}\right) - 1.5\right) + \text{sign}\left(-\text{sign}\left(\vec{w_1}^T\vec{x}\right) + \text{sign}\left(\vec{w_2}^T\vec{x}\right) - 1.5\right) + 1.5\right)$$

Multi-Layer Perceptron (MLP)

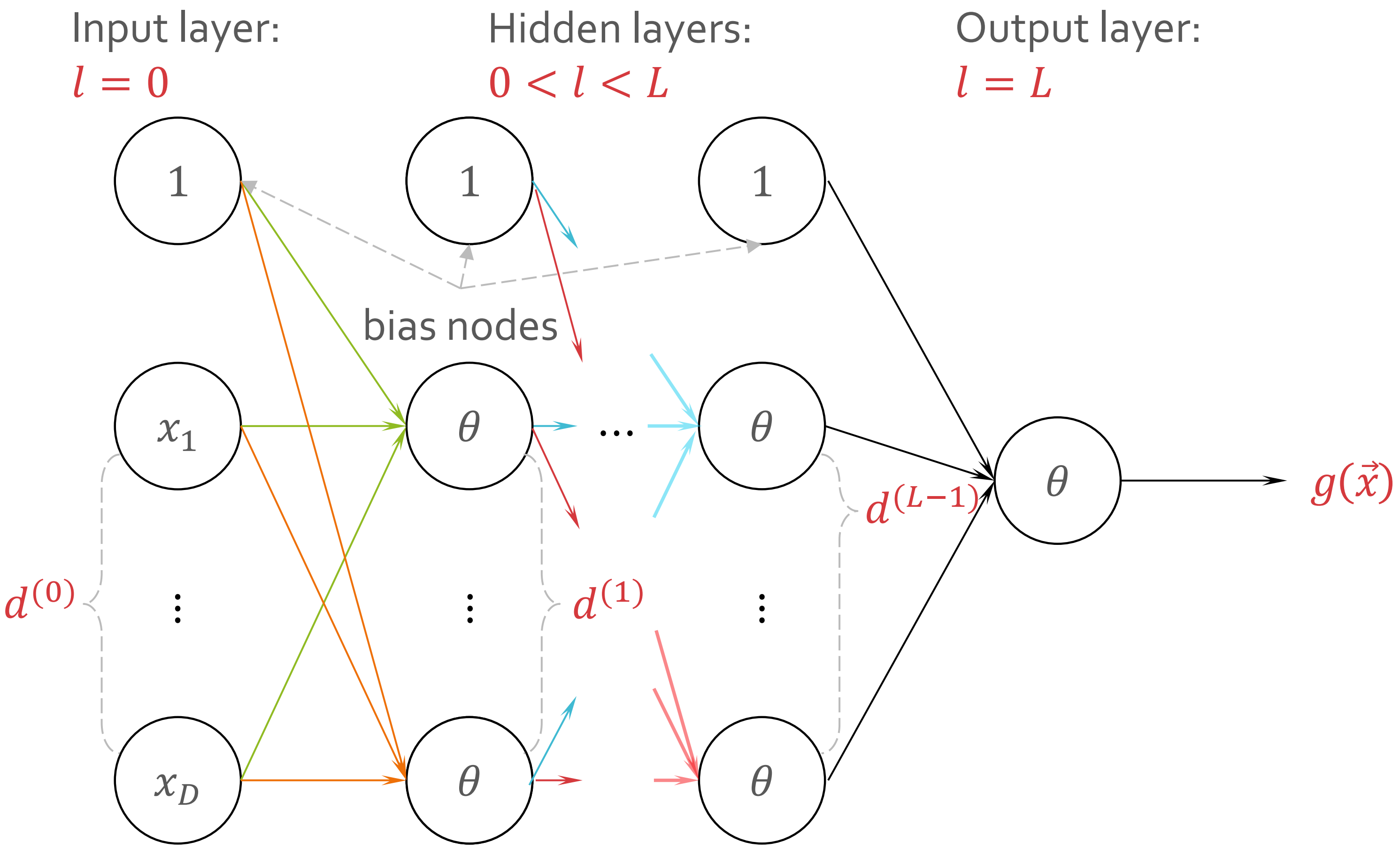


Feed-Forward Neural Network (NN)



What happens if $\theta(\cdot)$ is a linear function e.g. $\theta(x) = Cx$ for some constant C?

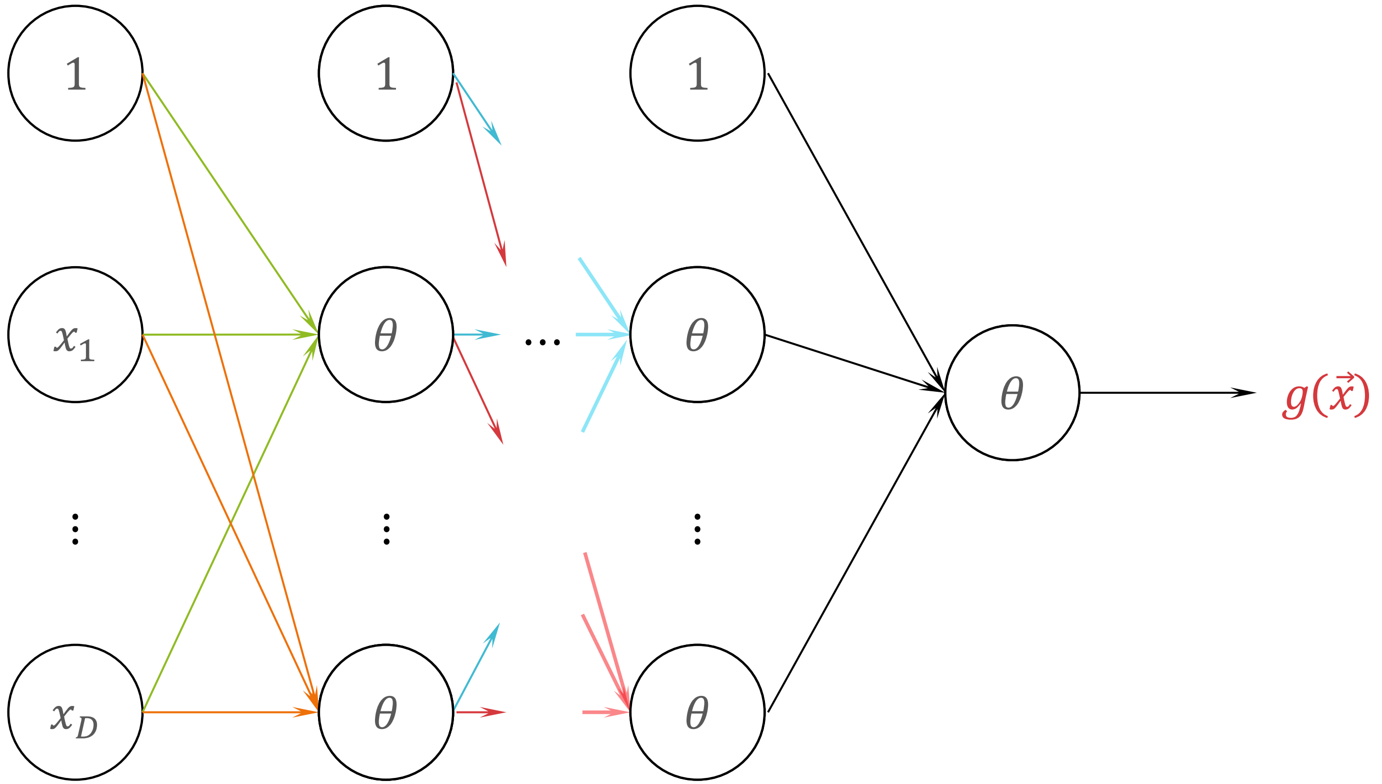
Feed-Forward Neural Network (NN)



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 $\vec{d} = \left[d^{(0)}, d^{(1)}, \dots, d^{(L)}\right]$

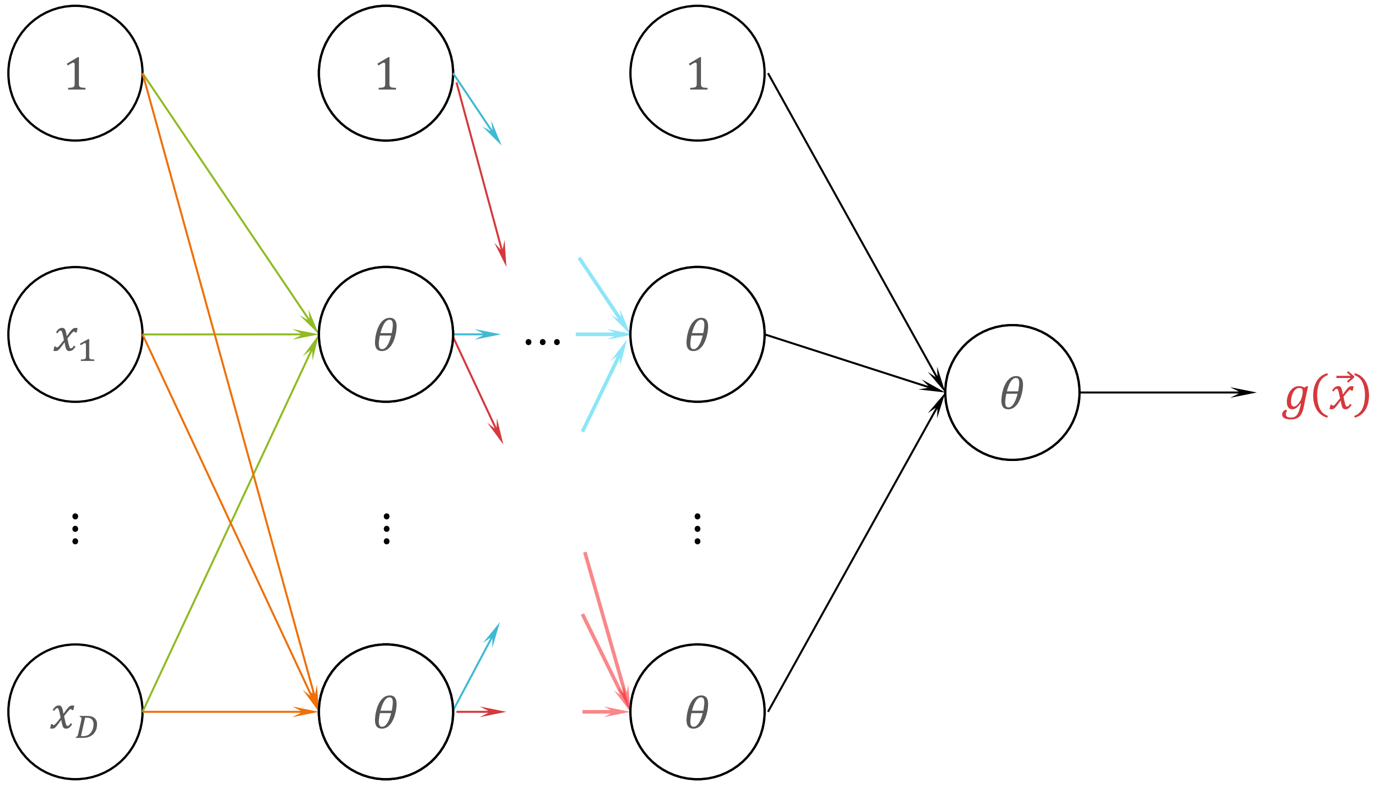


Every architecture corresponds to a hypothesis set: $\mathcal{H}_{NN}\left(\vec{d}\right)$

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

Weights

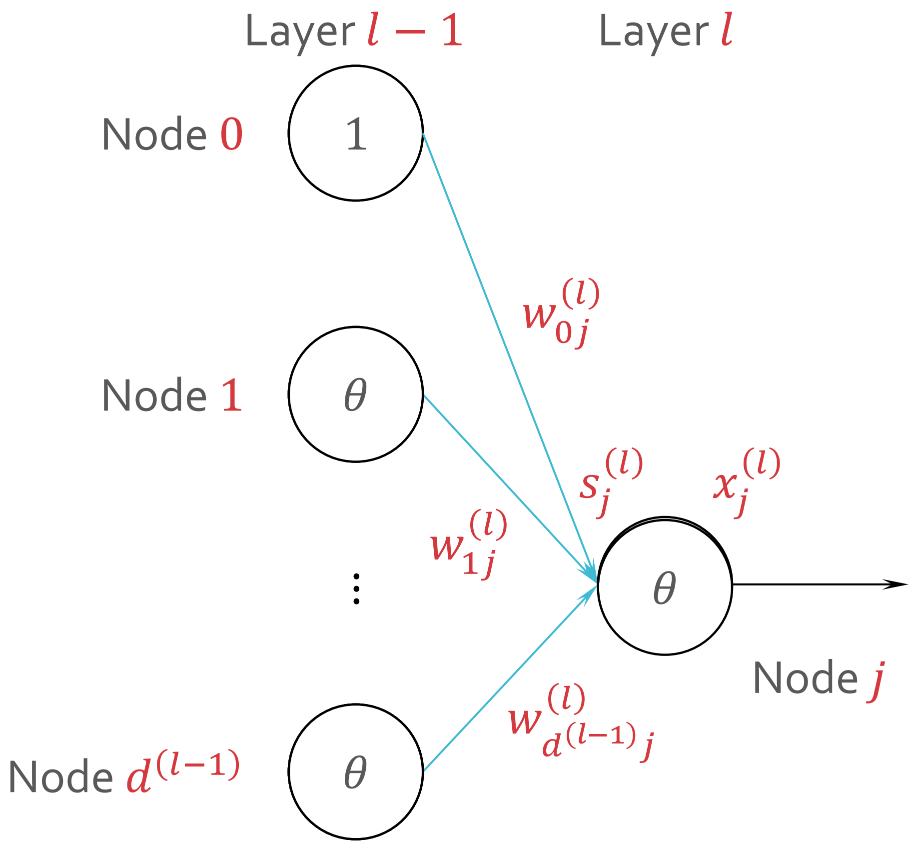
The weights between layer $l-1$ and layer l are a matrix: $W^{(l)} \in \mathbb{R}^{(d^{(l-1)}+1) \times d^{(l)}}$



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



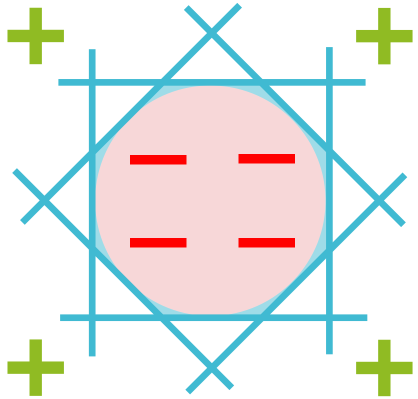
$$\overrightarrow{x^{(l)}} = \left[\begin{matrix}1\\ \theta\left(\overrightarrow{s^{(l)}}\right)\end{matrix}\right] \text{ and } \overrightarrow{s^{(l)}} = {W^{(l)}}^T \overrightarrow{x^{(l-1)}}$$

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: $\overrightarrow{x^{(1)}}, \dots, \overrightarrow{x^{(L)}}$

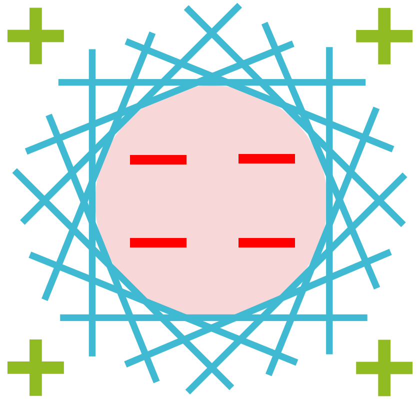
MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP

NNs as Universal Approximators

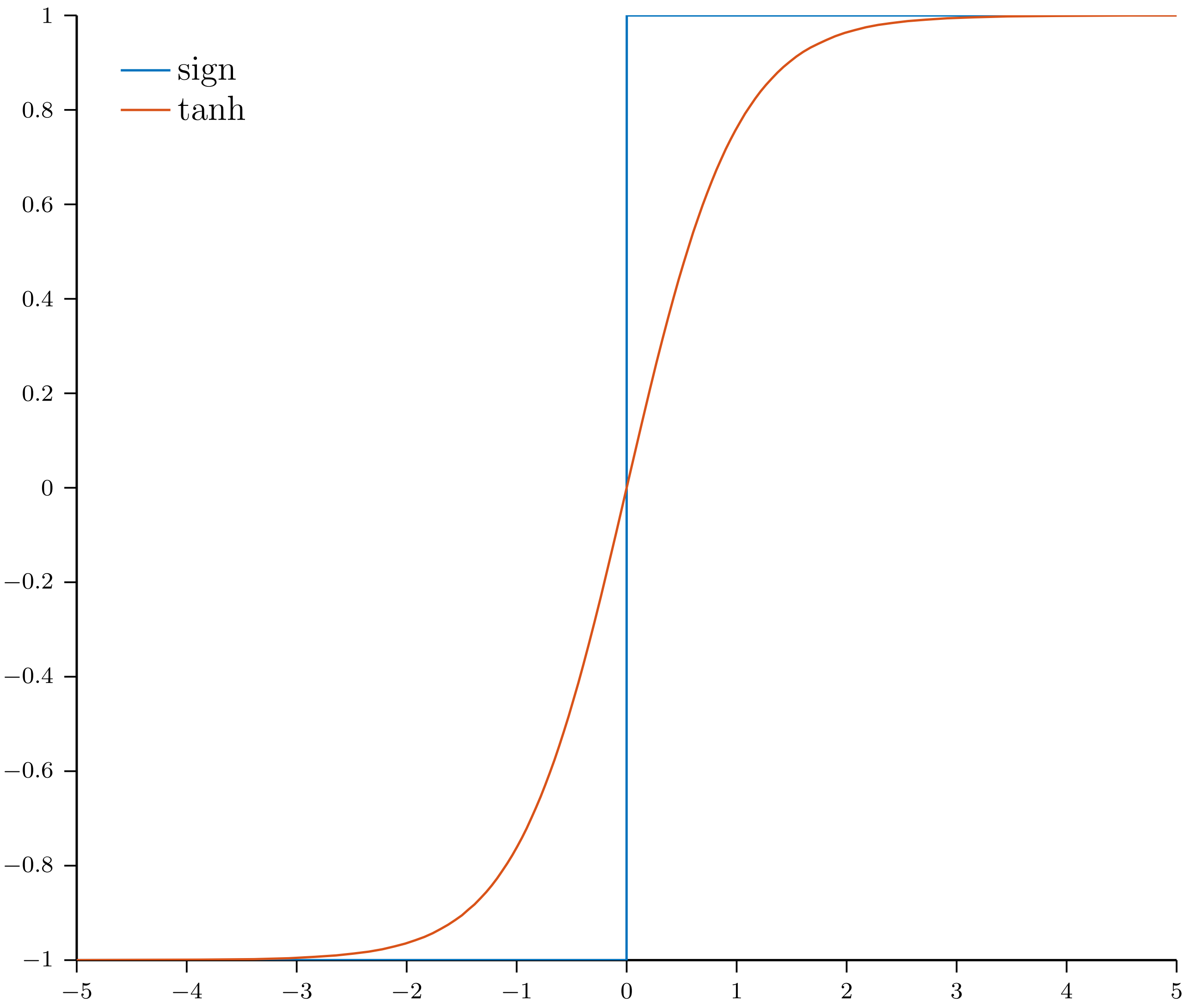
- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 2-layer feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.

$\theta(\cdot)$

- Hyperbolic tangent:

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

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



Error of a Neural Network

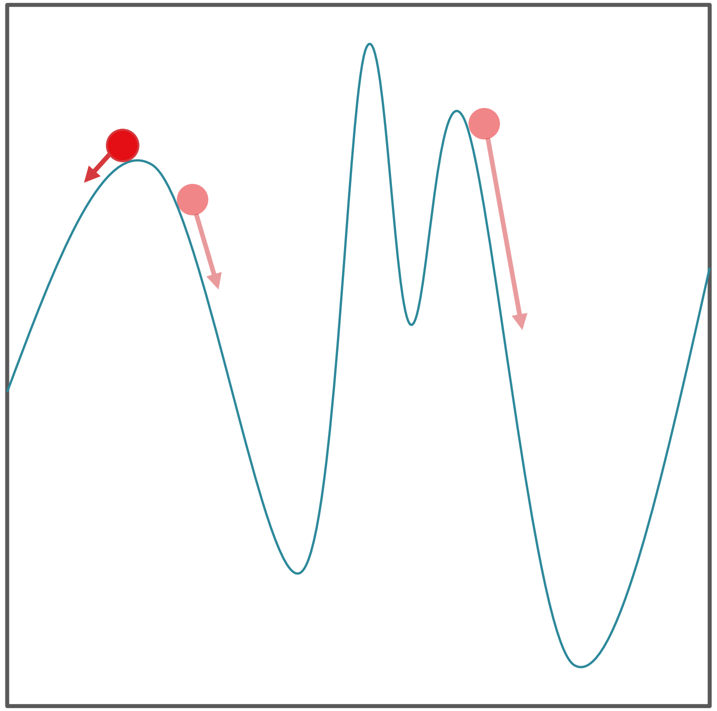
$$E_{in}\left(W^{(1)},\dots,W^{(L)}\right)=\frac{1}{n}\sum_{i=1}^{n}\left(h\left(\vec{x_i}\mid W^{(1)},\dots,W^{(L)}\right)-y_i\right)^2$$

$\ddots$

awkward silence

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: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$

Computing Gradients

$$E_{in}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \frac{1}{n}\sum_{i=1}^{n} e\left(h\left(\vec{x_i} \mid W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right), y_i\right)$$

$$\nabla_{W^{(l)}} E_{in}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$$

$$= \begin{bmatrix} \frac{\partial E_{in}}{\partial w_{01}^{(l)}} & \frac{\partial E_{in}}{\partial w_{02}^{(l)}} & \cdots & \frac{\partial E_{in}}{\partial w_{0d^{(l)}}^{(l)}} \\ \frac{\partial E_{in}}{\partial w_{11}^{(l)}} & \frac{\partial E_{in}}{\partial w_{12}^{(l)}} & \cdots & \frac{\partial E_{in}}{\partial w_{1d^{(l)}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E_{in}}{\partial w_{d^{(l-1)}1}^{(l)}} & \frac{\partial E_{in}}{\partial w_{d^{(l-1)}2}^{(l)}} & \cdots & \frac{\partial E_{in}}{\partial w_{d^{(l-1)}d^{(l)}}^{(l)}} \end{bmatrix}$$

Computing Gradients

$$E_{in}\left(W_{(t)}^{(1)},\dots,W_{(t)}^{(L)}\right)=\frac{1}{n}\sum_{i=1}^{n}e\left(h\left(\vec{x_i}\mid W_{(t)}^{(1)},\dots,W_{(t)}^{(L)}\right),y_i\right)$$

$$\frac{\partial E_{in}\left(W_{(t)}^{(1)},\dots,W_{(t)}^{(L)}\right)}{\partial w_{ab}^{(l)}}=\frac{1}{n}\sum_{i=1}^{n}\frac{\partial e\left(h\left(\vec{x_i}\mid W_{(t)}^{(1)},\dots,W_{(t)}^{(L)}\right),y_i\right)}{\partial w_{ab}^{(l)}}$$

$$=\frac{1}{n}\sum_{i=1}^{n}\frac{\partial e\left(\overrightarrow{x^{(L)}},y_i\right)}{\partial w_{ab}^{(l)}}$$

Computing Gradients

Computing $\nabla_{W^{(l)}} E_{in}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$ reduces to computing

$$\frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial w_{ab}^{(l)}}$$

Insight: $w_{ab}^{(l)}$ only affects $e\left(\overrightarrow{x^{(L)}}, y_i\right)$ via $s_b^{(l)}$

Use the chain rule: $\dfrac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial w_{ab}^{(l)}} = \dfrac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial s_b^{(l)}}\left(\dfrac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}}\right)$

Computing Gradients

To compute $\frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial s_b^{(l)}}\left(\frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}}\right)$, recall:

$$s_b^{(l)} = \sum_{a=0}^{d^{(l-1)}} w_{ab}^{(l)} x_a^{(l-1)} \rightarrow \frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}} = x_a^{(l-1)}$$

Can compute all outputs $\vec{x}^{(l)}\ \forall\, l \in \{0, \ldots, L\}$ using forward propagation

Let $\delta_b^{(l)} = \frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial s_b^{(l)}}$

$\overrightarrow{\delta^{(l)}}$ is called the sensitivity vector for layer l

Computing Gradients

Insight: $s_b^{(l)}$ only affects $e\left(\overrightarrow{x^{(L)}}, y_i\right)$ via $x_b^{(l)}$

Use the chain rule: $\delta_b^{(l)} = \frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial x_b^{(l)}} \left(\frac{\partial x_b^{(l)}}{\partial s_b^{(l)}}\right)$

Recall: $x_b^{(l)} = \theta\left(s_b^{(l)}\right) \rightarrow \frac{\partial x_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta\left(s_b^{(l)}\right)}{\partial s_b^{(l)}}$

$$= 1 - \left(x_b^{(l)}\right)^2$$

when $\theta(\cdot) = \tanh(\cdot)$

Computing Gradients

Insight: $x_b^{(l)}$ affects $e\left(\overrightarrow{x^{(L)}}, y_i\right)$ via $s_1^{(l+1)}, \dots, s_{d^{(l+1)}}^{(l+1)}$

Use the chain rule: $\dfrac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial x_b^{(l)}} = \displaystyle\sum_{c=1}^{d^{(l+1)}} \dfrac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial s_c^{(l+1)}} \left(\dfrac{\partial s_c^{(l+1)}}{\partial x_b^{(l)}}\right)$

$$= \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left(w_{bc}^{(l+1)}\right)$$

Computing Gradients

$$\delta_b^{(l)} = \frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial x_b^{(l)}} \left(\frac{\partial x_b^{(l)}}{\partial s_b^{(l)}}\right)$$

$$= \left(\sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left(w_{bc}^{(l+1)}\right)\right) \left(1 - \left(x_b^{(l)}\right)^2\right)$$

$$\overrightarrow{\delta^{(l)}} = W^{(l+1)} \overrightarrow{\delta^{(l+1)}} \otimes \left(1 - \overrightarrow{x^{(l)}} \otimes \overrightarrow{x^{(l)}}\right)$$

where $\otimes$ is the element-wise product operation

$$\frac{\partial e\left(\overrightarrow{x^{(L)}}, y_i\right)}{\partial w_{ab}^{(l)}} = \delta_b^{(l)} \left(\frac{\partial s_b^{(l)}}{\partial w_{ab}^{(l)}}\right) = \delta_b^{(l)} \left(x_a^{(l-1)}\right)$$

$$\nabla_{W^{(l)}} e\left(\overrightarrow{x^{(L)}}, y_i\right) = \overrightarrow{x^{(l-1)}} \left(\overrightarrow{\delta^{(l)}}\right)^T$$

Computing Gradients

Can recursively compute $\overrightarrow{\delta^{(l)}}$ using $\overrightarrow{\delta^{(l+1)}}$; need to compute the base case: $\overrightarrow{\delta^{(L)}}$

Assume the output layer is a single node and the error function is the squared error:

$\overrightarrow{\delta^{(L)}} = \delta_1^{(L)}$, $\overrightarrow{x^{(L)}} = x_1^{(L)}$ and $e\left(x_1^{(L)}, y_i\right) = \left(x_1^{(L)} - y_i\right)^2$

$$\delta_1^{(L)} = \frac{\partial e\left(x_1^{(L)}, y_i\right)}{\partial s_1^{(L)}} = \frac{\delta}{\partial s_1^{(L)}}\left(x_1^{(L)} - y_i\right)^2$$

$$= 2\left(x_1^{(L)} - y_i\right)\frac{\delta x_1^{(L)}}{\partial s_1^{(L)}} = 2\left(x_1^{(L)} - y_i\right)^2\left(1 - \left(x_1^{(L)}\right)^2\right)$$

when $\theta(\cdot) = \tanh(\cdot)$

Back-propagation

- Input: weights $W^{(1)}, \ldots, W^{(L)}$ and a query point $\vec{x}$
- Run forward propagation to get $\overrightarrow{x^{(1)}}, \ldots, \overrightarrow{x^{(L)}}$
- Initialize $\delta_1^{(L)} = 2\left(x_1^{(L)} - y_i\right)\left(1 - \left(x_1^{(L)}\right)^2\right)$
- For $l = L - 1, \ldots, 1$
  - Compute $\overrightarrow{\delta^{(l)}} = W^{(l+1)}\overrightarrow{\delta^{(l+1)}} \otimes \left(1 - \overrightarrow{x^{(l)}} \otimes \overrightarrow{x^{(l)}}\right)$
- Output: $\overrightarrow{\delta^{(1)}}, \ldots, \overrightarrow{\delta^{(L)}}$

Computing Gradients

- Input: $W^{(1)}, \dots, W^{(L)}$ and $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), \dots, (\overrightarrow{x_n}, y_n)\}$
- Initialize $E_{in} = 0$ and $G^{(l)} = 0 * W^{(l)}$ for $l = 1, \dots, L$
- For $i = 1, \dots, n$
  - Run forward propagation to get $\overrightarrow{x^{(1)}}, \dots, \overrightarrow{x^{(L)}}$
  - Run backpropagation to get $\overrightarrow{\delta^{(1)}}, \dots, \overrightarrow{\delta^{(L)}}$
  - Increment E_{in}: $E_{in} = E_{in} + \frac{1}{n}\left(\overrightarrow{x^{(L)}} - y_i\right)^2$
  - For $l = 1, \dots, L$
    - Compute $G_i^{(l)} = \overrightarrow{x^{(l-1)}}\left(\overrightarrow{\delta^{(l)}}\right)^T$
    - Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \frac{1}{n} G_i^{(l)}$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of E_{in} w.r.t $W^{(1)}, \dots, W^{(L)}$

Complexity

- Both forward and backpropagation contain matrix multiplications involving $W^{(1)}, \dots, W^{(L)}$ → both take time $O\left(\left|W^{(1)}\right| + \cdots + \left|W^{(L)}\right|\right)$ …
- Computing $G^{(1)}, \dots, G^{(L)}$ requires running forward and backpropagation for each training point $(\vec{x}, y) \in \mathcal{D}$ …
- Each iteration of gradient descent for a neural network takes time $O\left(n\left(\left|W^{(1)}\right| + \cdots + \left|W^{(L)}\right|\right)\right)$
- Use stochastic gradient descent instead!
- Also use parallelization and GPUs / TPUs!

Stochastic Gradient Descent for Neural Networks

- Input: $\mathcal{D} = \{(\overrightarrow{x_1}, y_1), \ldots, (\overrightarrow{x_n}, y_n)\}, \eta_0$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While some termination condition is not satisfied
  - For $l = 1, \ldots, L$
    - Randomly select a point $(\overrightarrow{x_1}, y_1) \in \mathcal{D}$
    - Compute $G^{(l)'} = \nabla_{W^{(l)}} e\left(h\left(\overrightarrow{x_i} \mid W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right), y_i\right)$
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)'}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

Initialization and Termination

- Initialization:
  - Randomness is good for non-convex optimization
  - Initialize weights by sampling from $N(0, \sigma^2)$
- Termination:
  - For complicated surfaces, the gradient's magnitude is not a good metric for proximity to a minimum
  - A simple solution: combine multiple termination criteria e.g. stop if enough iterations have passed and the improvement in error is small