Neural Networks

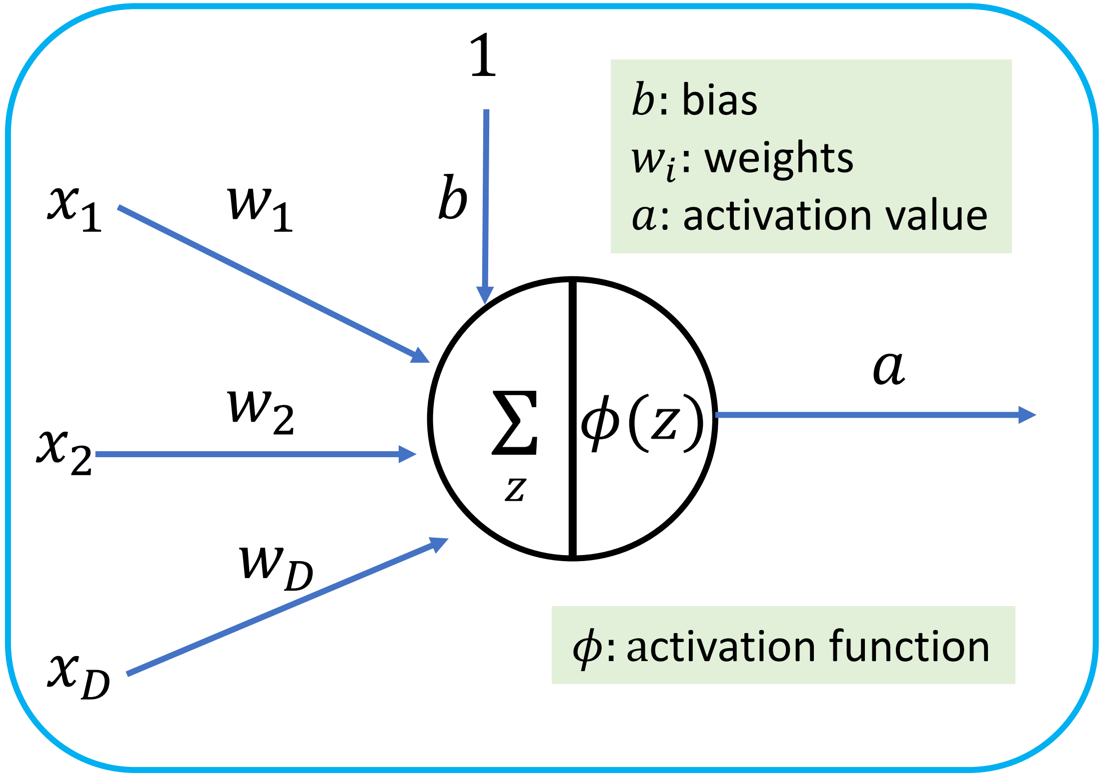
CSCC11 – Topic 5

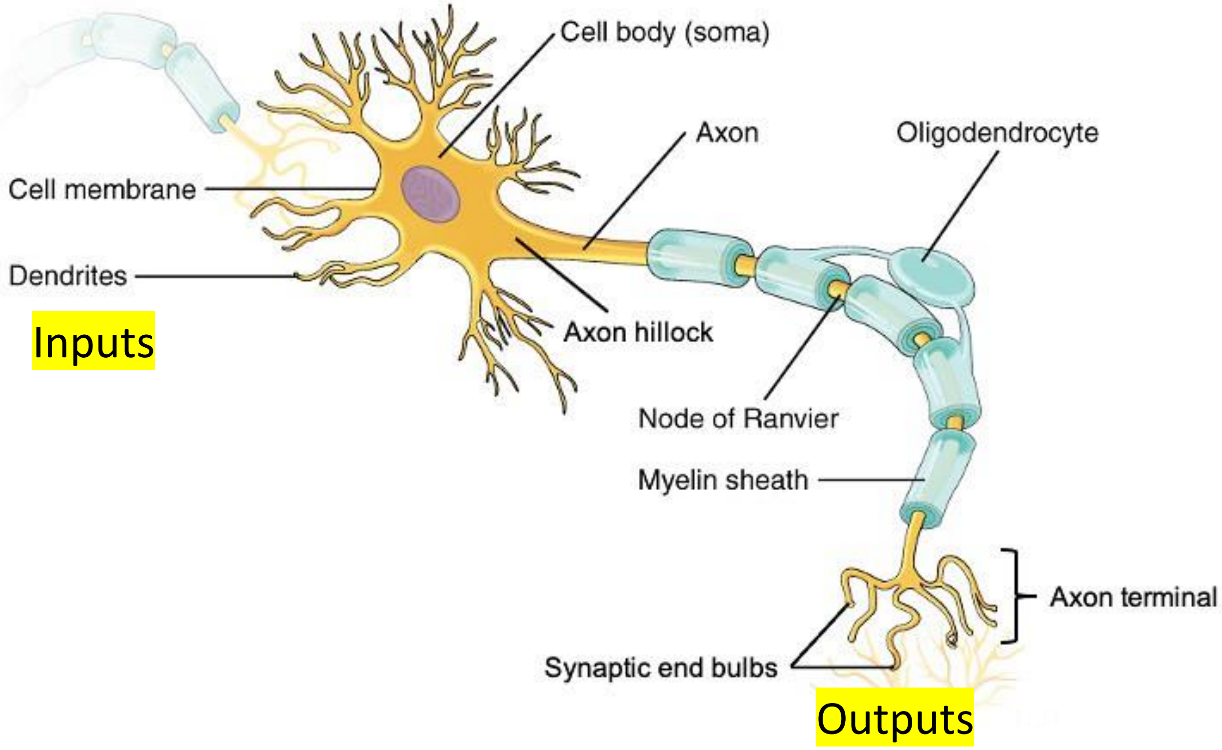
Computer & Mathematical Sciences
UNIVERSITY OF TORONTO
SCARBOROUGH

Learning Feature Mapping

- Feature engineering makes linear methods non linear by $\mathbf{x} \rightarrow \phi(\mathbf{x})$ mapping.
- We want to learn the feature mappings instead of manually crafting them.
- Neural networks, invented by Frank Rosenblatt in 1963, can be thought as a method to explicitly learn the feature map ϕ

Artificial Neuron





$$a = \phi\left(\sum_{j=1}^{D} w_j x_j + b\right) = \phi(\mathbf{w}^T \mathbf{x} + b)$$

Activation Function

- Sigmoid

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

- ReLU: Rectified Linear Unit

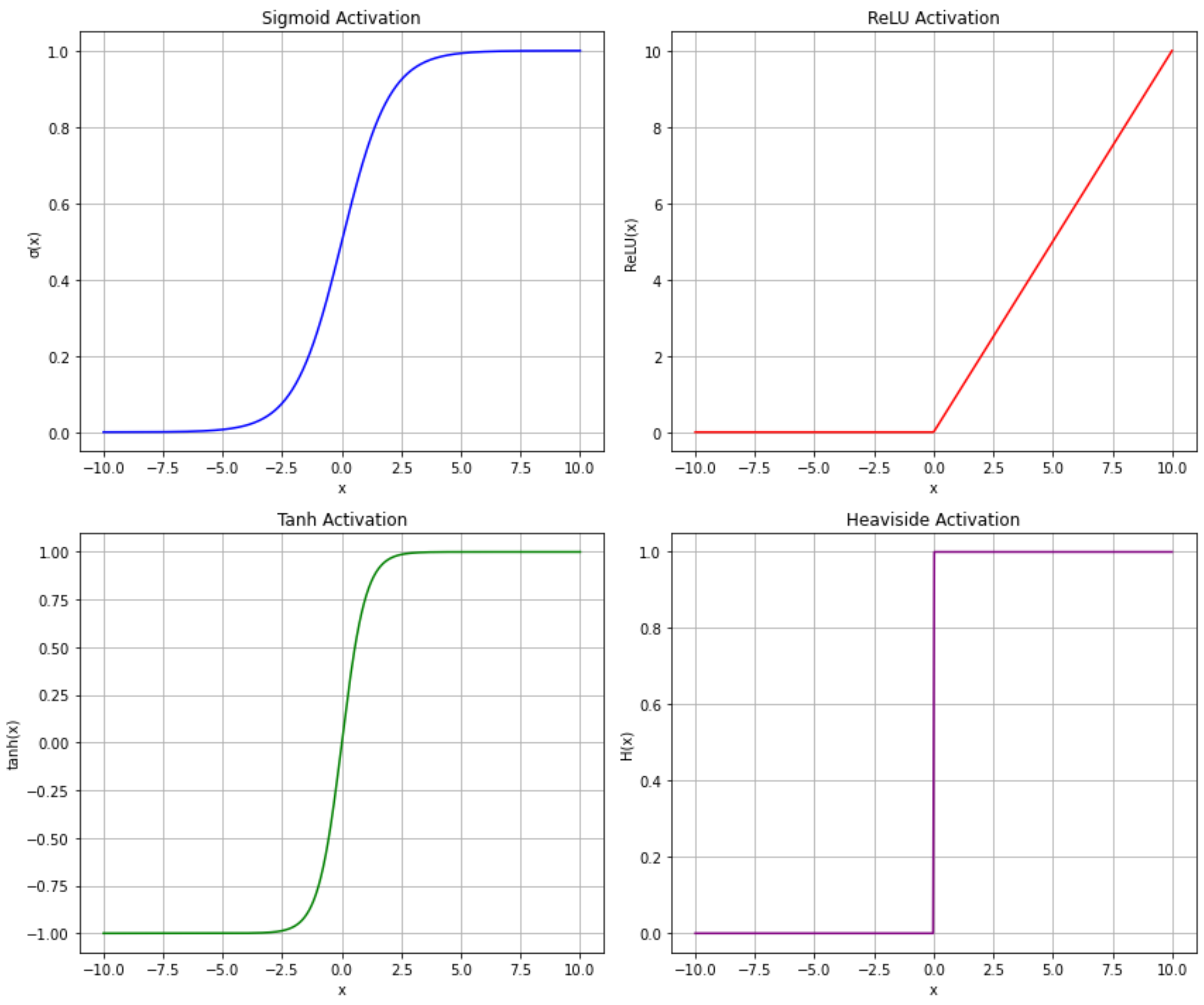
$$\mathrm{ReLU}(z) \equiv \max(z, 0)$$

- Hyperbolic tangent

$$\tanh(z) \equiv \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

- Heaviside function

$$H(z) \equiv \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if z} < 0 \end{cases}$$

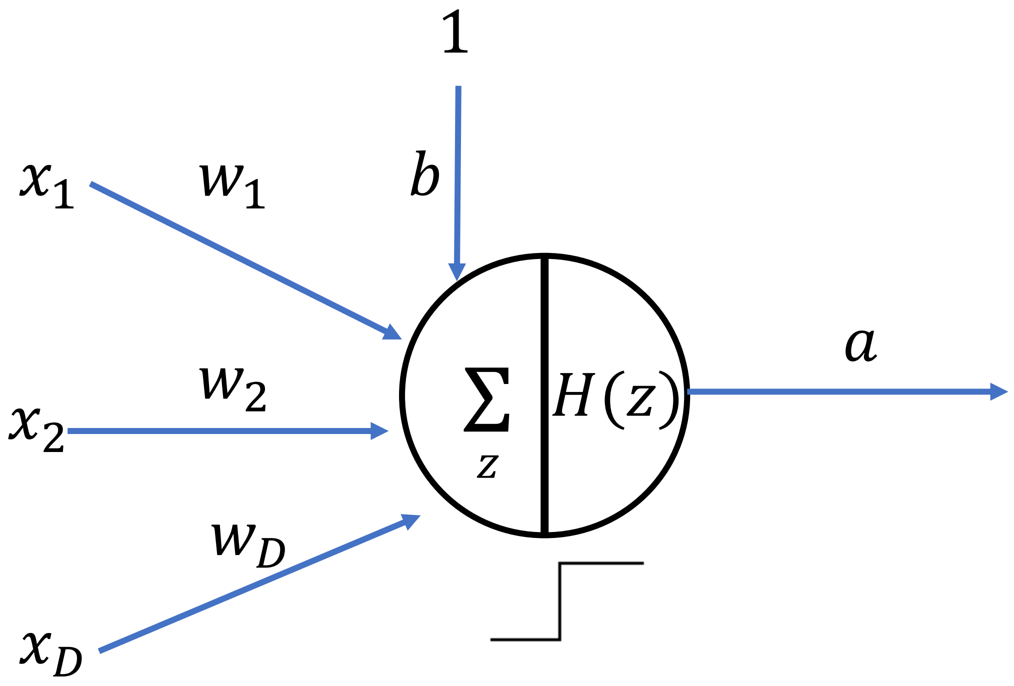


Which ones are differentiable?

Perceptron

- Theory was invented in 1958, realized in 1963 by Frank Rosenblatt
- A single neuron for binary classification
- Heaviside step function as the activation function.
- A Learning algorithm was created.

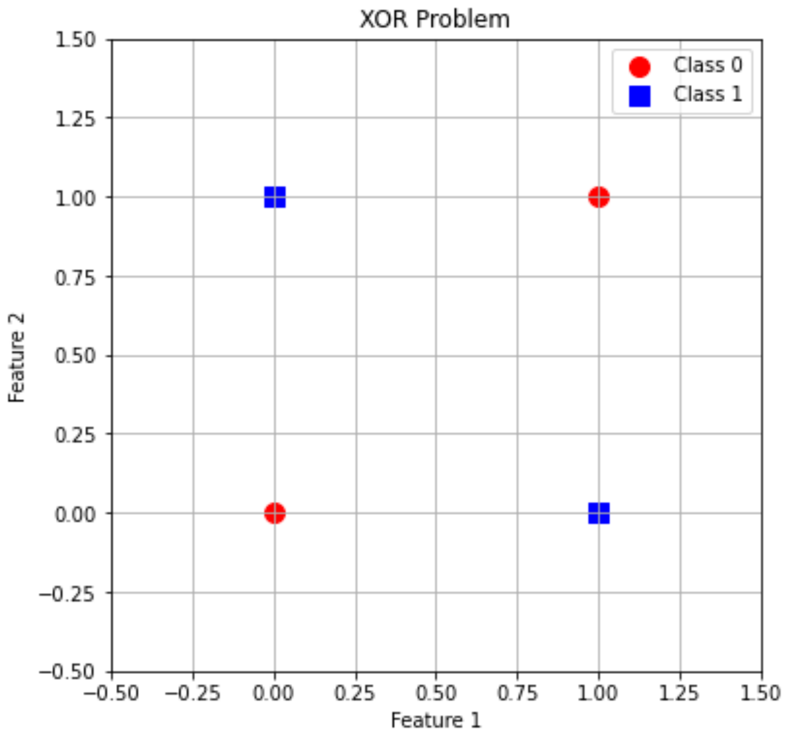
$$\hat{y} = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0, & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$



The XOR Problem

- In 1969, Minsky & Papert highlighted the limitations of the Perceptron, leading to a temporary decline in interest of NN.
- First AI winter: 1974-1980

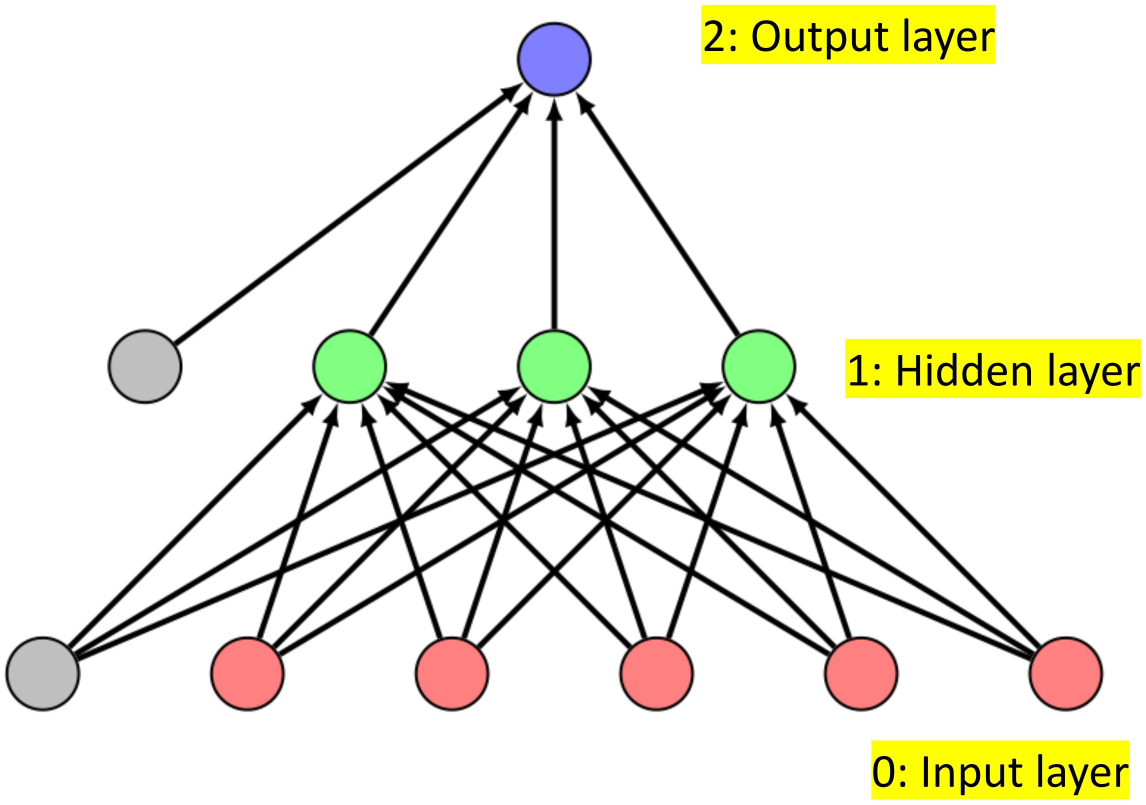
x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



- Linear decision boundary does not exist

Multi-layer Perceptron (MLP)

- Concept started in 1960s, before 1980s
  - No effective training algorithms
  - Single layer network was easy to implement and learn.
- 1980s: backpropagation algorithm by Rumelhart, Hinton, and Williams renewed research in MLP
- Single perceptron can learn AND, OR, NOT
- Stacking up layers of perceptrons to learn any binary function



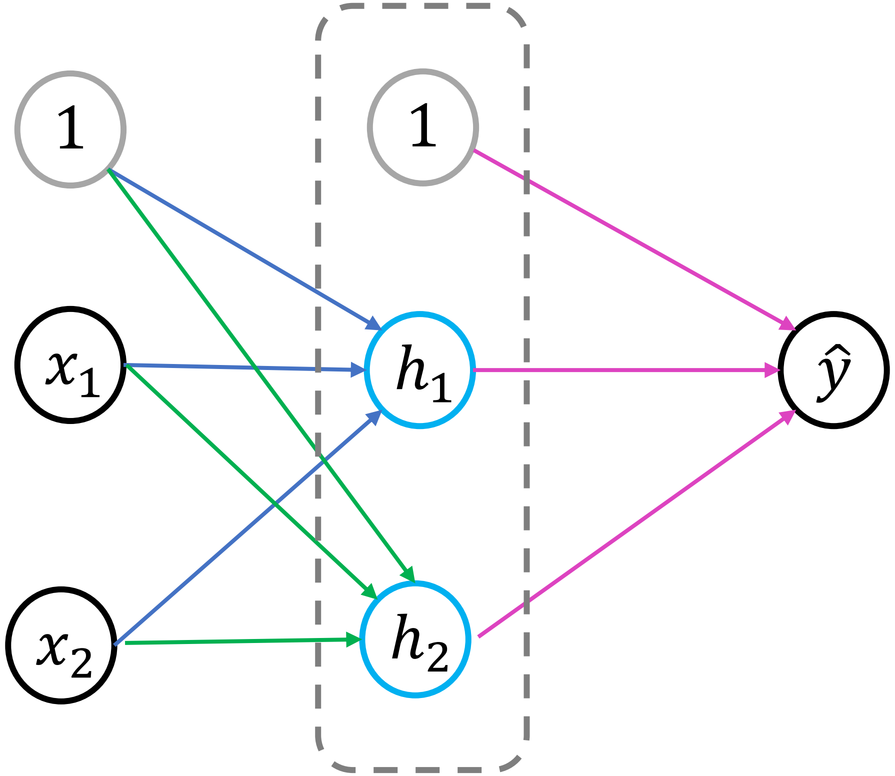
A 2-layer Neural Network
The input layer does not count as a layer
[Image Courtesy Wikipedia]

AND, OR and NOT

- Exercise: Design Perceptron to represent logic operators

XOR under MLP

- A 2-layer Neural Network to compute XOR



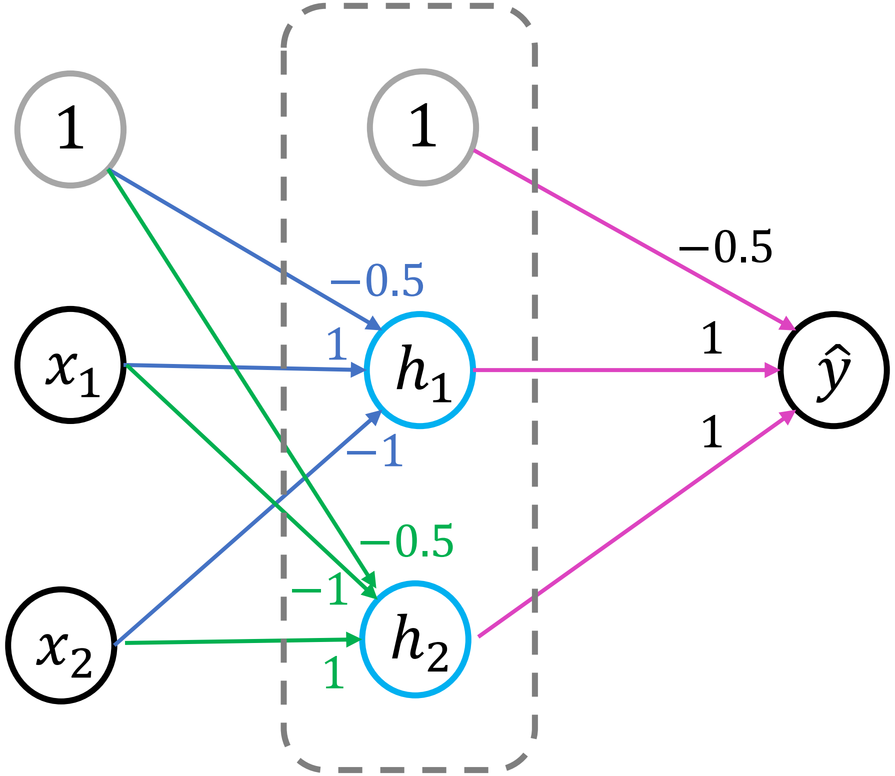
x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- Heaviside function

$$H(z) \equiv \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

XOR: a 2-layer NN Solution

$$x_1 \oplus x_2 = (x_1 \land \neg x_2) \lor (\neg x_1 \land x_2)$$



x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- Heaviside function

$$H(z) \equiv \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

Universal Approximation Theorem

- A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, under some assumptions about the activation function (non-linear).
  - Arbitrary width and bounded depth: 1989 by Cybenko and Hornik
  - Arbitrary depth and bounded width: 2017 by Lu et. al.
  - Bounded depth and width: 2022 by Maiorov and Pinkus
    - Two hidden layers are enough to approximate any functions.
- Remarks
  - It guarantees the representation power of neural networks for approximating virtually any continuous function to arbitrary precision
  - It only states the existence, but does not provide a method to find it
  - It does not address the learnability of the function or the computational efficiency of training

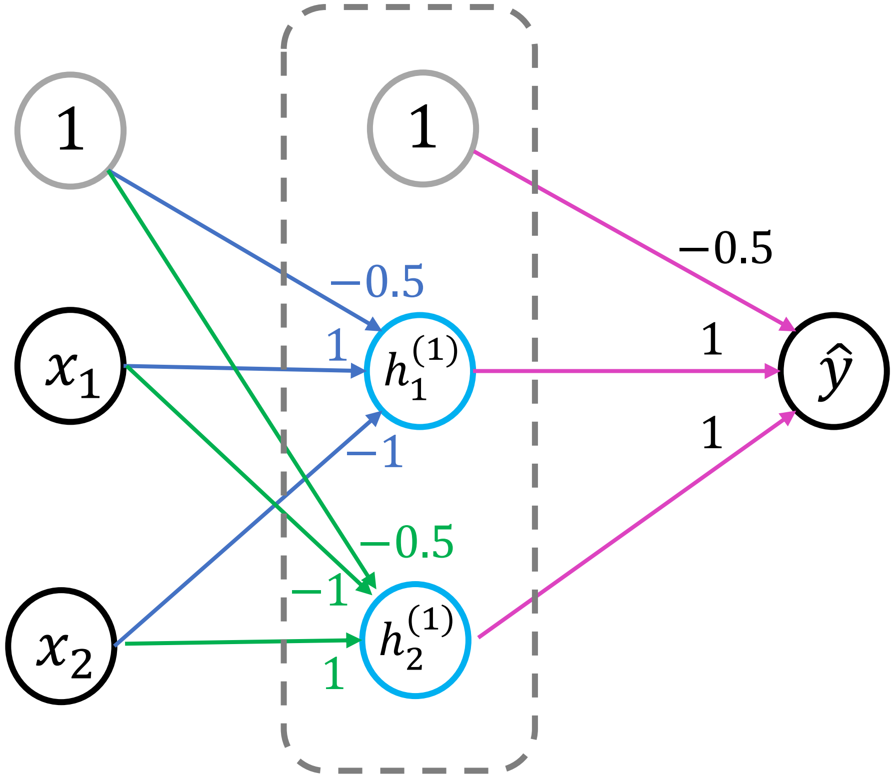
Changes to Revive NN

- ReLU activation function
  - The vanishing gradient problem
- GPU
  - Efficient matrix multiplication possible
- Stochastic Gradient Descent
- Rebranding
  - Deep Learning

XOR: a 2-layer NN Solution

x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$



- Heaviside function

$$H(z) \equiv \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

$\mathbf{w}_j^{(l)}$ $\quad l^{th}$ layer $\quad j^{th}$ node

$$\mathbf{w}_1^{(1)} = \begin{bmatrix} b_1^{(1)} \\ w_{1,1}^{(1)} \\ w_{1,2}^{(1)} \end{bmatrix}$$

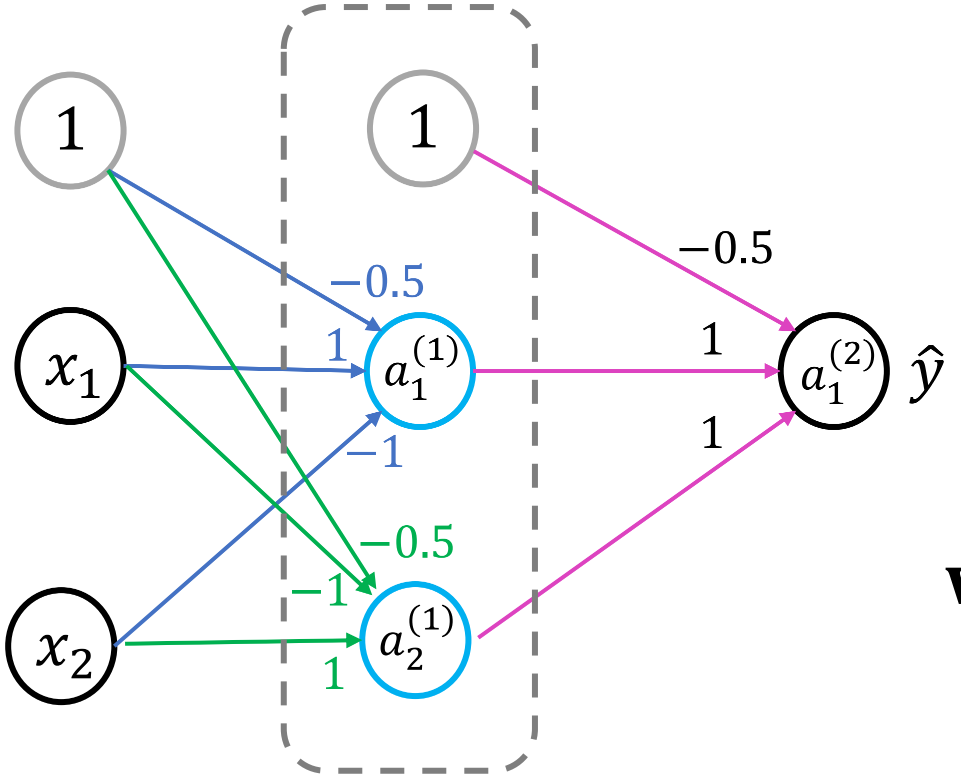
$$\mathbf{W}^{(1)} = \begin{bmatrix} b_1^{(1)} & w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ b_2^{(1)} & w_{2,1}^{(1)} & w_{2,2}^{(1)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} b_1^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} \end{bmatrix}$$

XOR: a 2-layer NN Solution

x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$



- Heaviside function

$$H(z) \equiv \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

$\mathbf{w}_j^{(l)}$ l^{th} layer, j^{th} node

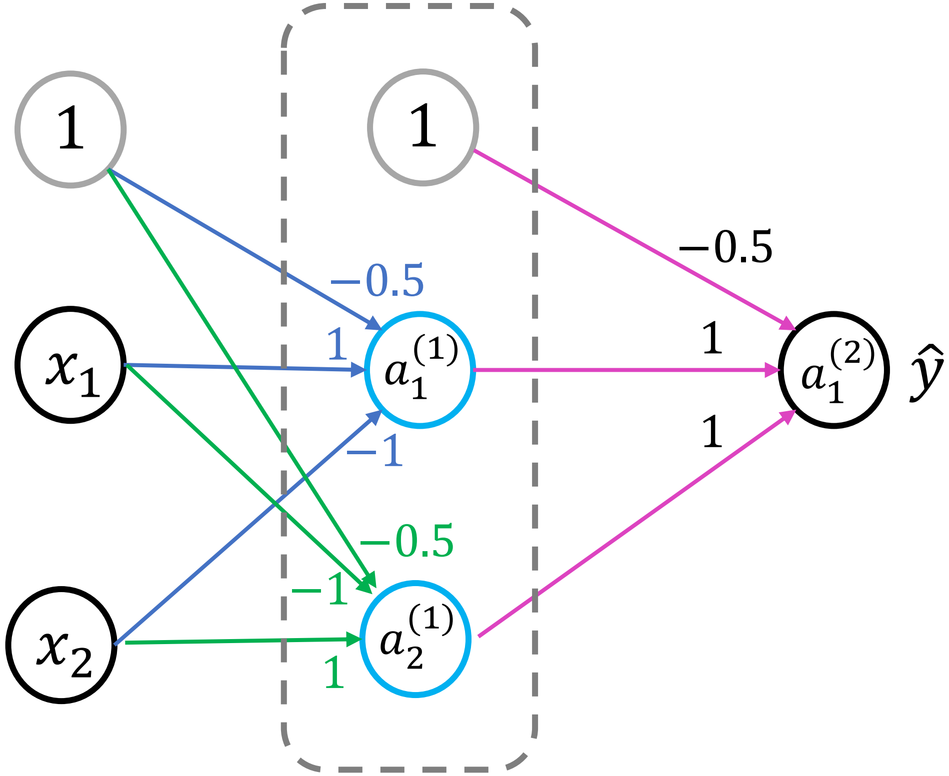
$$\mathbf{w}_1^{(1)} = \begin{bmatrix} b_1^{(1)} \\ w_{1,1}^{(1)} \\ w_{1,2}^{(1)} \end{bmatrix} \qquad \mathbf{W}^{(1)} = \begin{bmatrix} b_1^{(1)} & w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ b_2^{(1)} & w_{2,1}^{(1)} & w_{2,2}^{(1)} \end{bmatrix}$$

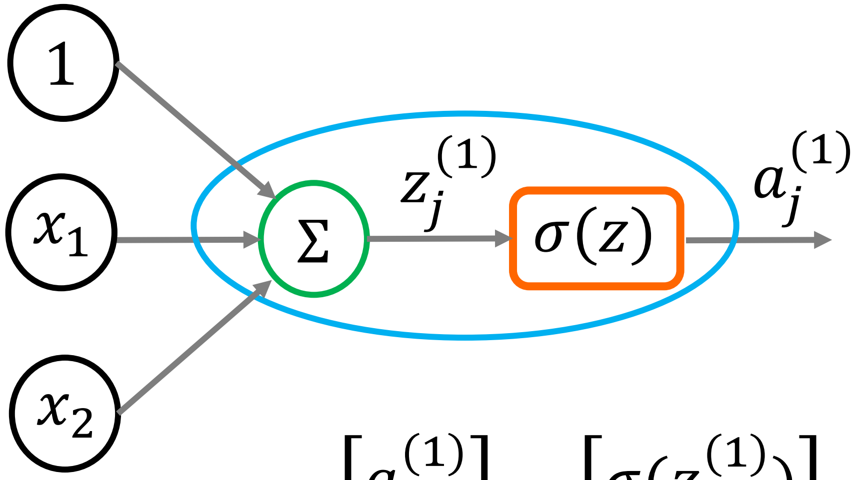
$$\mathbf{W}^{(2)} = \begin{bmatrix} b_1^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} \end{bmatrix}$$

Weight Matrices

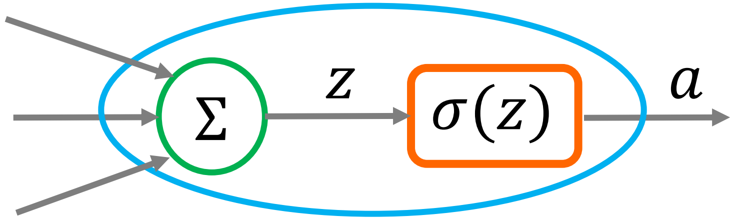
$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{W}^{(1)} = \begin{bmatrix} b_1^{(1)} & w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ b_2^{(1)} & w_{2,1}^{(1)} & w_{2,2}^{(1)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} b_1^{(2)} & w_{1,1}^{(2)} & w_{1,2}^{(2)} \end{bmatrix}$$



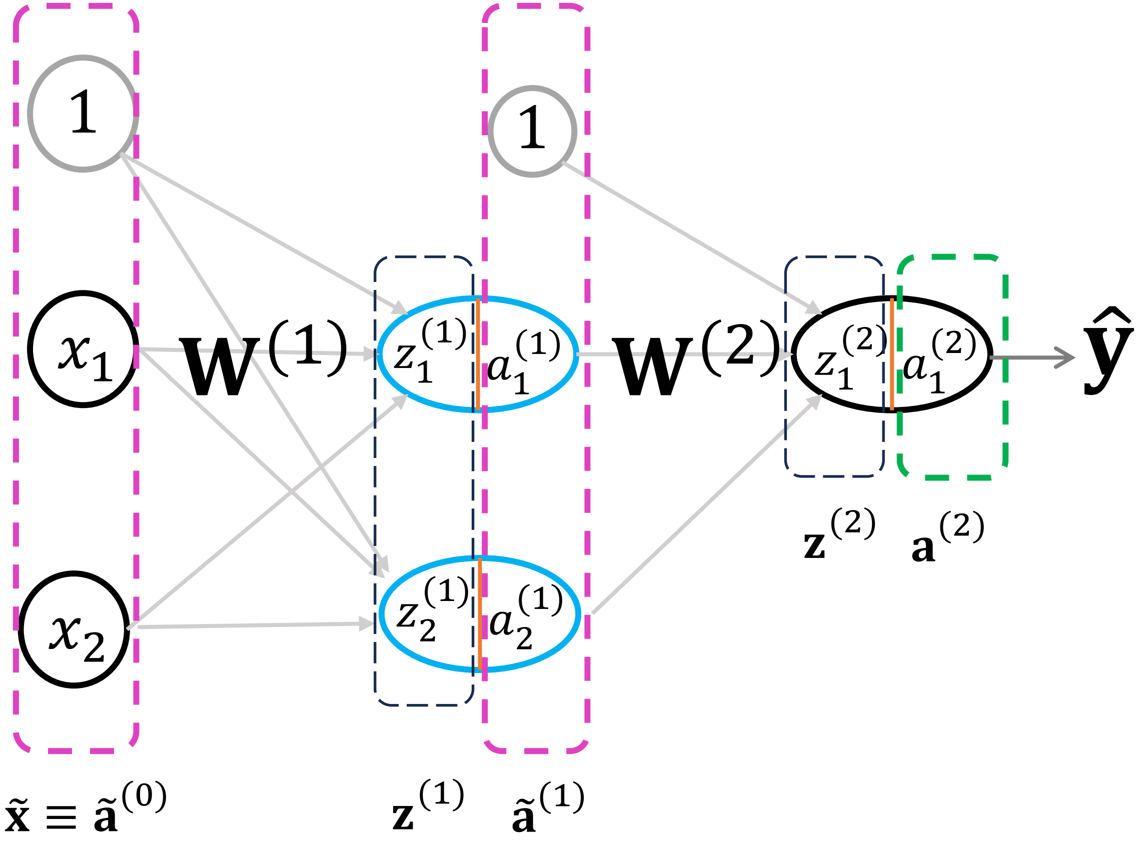


$$\mathbf{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{(1)}) \\ \sigma(z_2^{(1)}) \end{bmatrix} = \sigma\left(\mathbf{W}^{(1)}\tilde{\mathbf{x}}\right)$$



$$\tilde{\mathbf{a}} = \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \qquad \mathbf{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{(2)}) \end{bmatrix} = \sigma\left(\mathbf{W}^{(2)}\tilde{\mathbf{a}}^{(1)}\right)$$

Weight Matrices



$$\tilde{\sigma}(\mathbf{z}) \equiv \begin{bmatrix} 1 \\ \sigma(z_1) \\ \vdots \\ \sigma(z_m) \end{bmatrix} = \begin{bmatrix} 1 \\ \sigma(\mathbf{z}) \end{bmatrix}$$

$$\begin{aligned} \hat{\mathbf{y}} = \mathbf{a}^{(2)} &= \left[a_1^{(2)}\right] = \sigma\left(\mathbf{z}^{(2)}\right) \\ &= \sigma\left(\mathbf{W}^{(2)}\tilde{\mathbf{a}}^{(1)}\right) \\ &= \sigma\left(\mathbf{W}^{(2)}\tilde{\sigma}\left(\mathbf{W}^{(1)}\tilde{\mathbf{a}}^{(0)}\right)\right) \\ &= \sigma\Big(\underbrace{\mathbf{W}^{(2)}\tilde{\sigma}\left(\mathbf{z}^{(1)}\right)}_{\mathbf{z}^{(2)}}\Big) \end{aligned}$$

Learning Weights

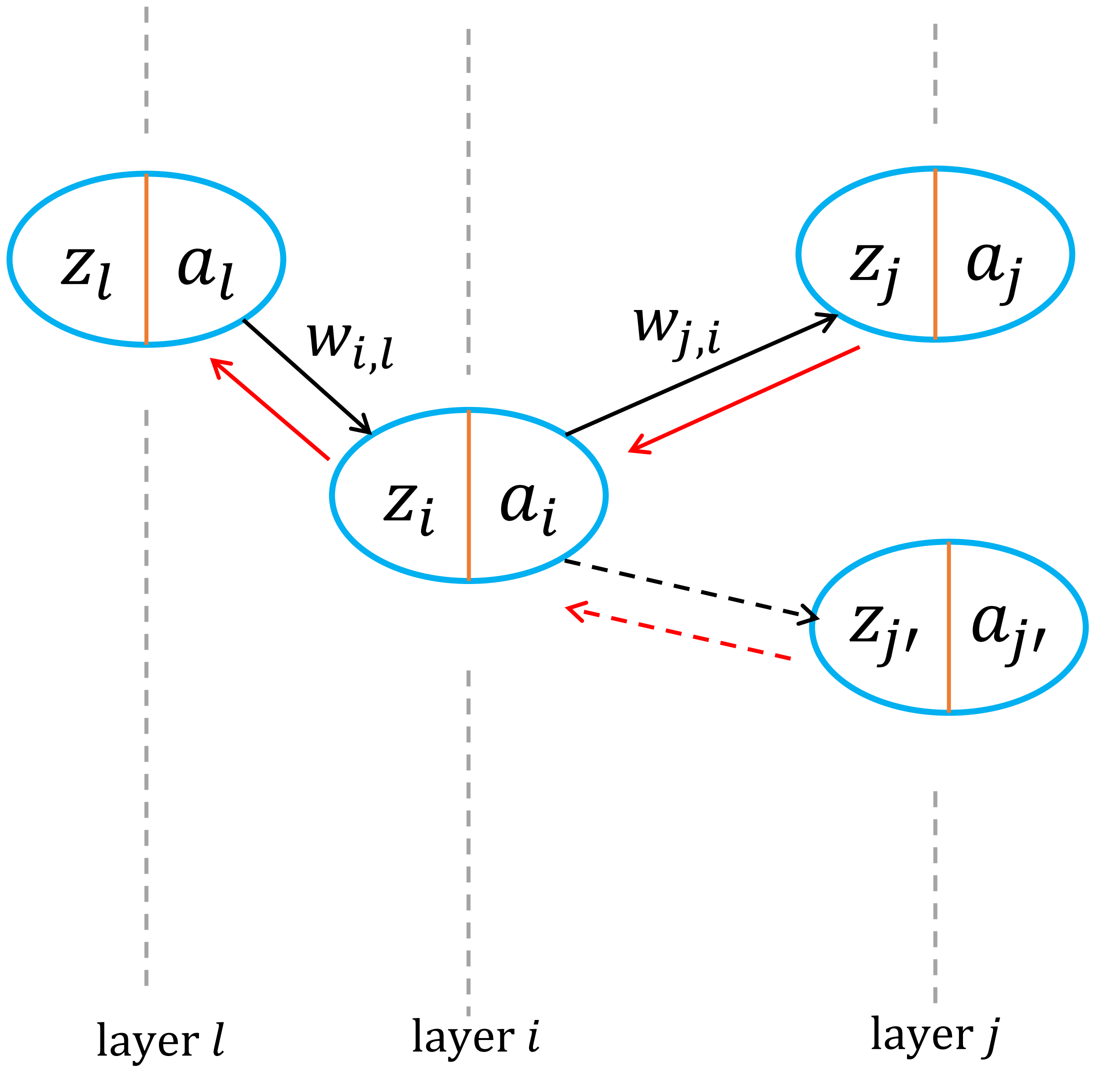
- Define loss function $\mathcal{L}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k; \left\{\mathbf{W}^{(l)}\right\}_{l=1:L}\right) \equiv \mathcal{L}_k\left(\left\{\mathbf{W}^{(l)}\right\}_{l=1:L}\right)$
- Cost function

$$E\left(\{(\mathbf{y}_k, \hat{\mathbf{y}}_k)\}_{k=1:N}; \left\{\mathbf{W}^{(l)}\right\}_{l=1:L}\right) = c\sum_{k=1}^{N} \mathcal{L}_k\left(\left\{\mathbf{W}^{(l)}\right\}_{l=1:L}\right), \text{ e.g. } c = \frac{1}{N}$$

- Gradient Descent $\quad w_{j,i}^{(l)} \leftarrow w_{j,i}^{(l)} - \lambda \frac{\partial E}{\partial w_{j,i}^{(l)}}$

$$\frac{\partial E}{\partial w_{j,i}^{(l)}} = \sum_{k=1}^{N} \frac{\partial \mathcal{L}_k\left(\left\{\mathbf{W}^{(l)}\right\}_{l=1:L}\right)}{\partial w_{j,i}^{(l)}}$$

Backpropagation



Backpropagation

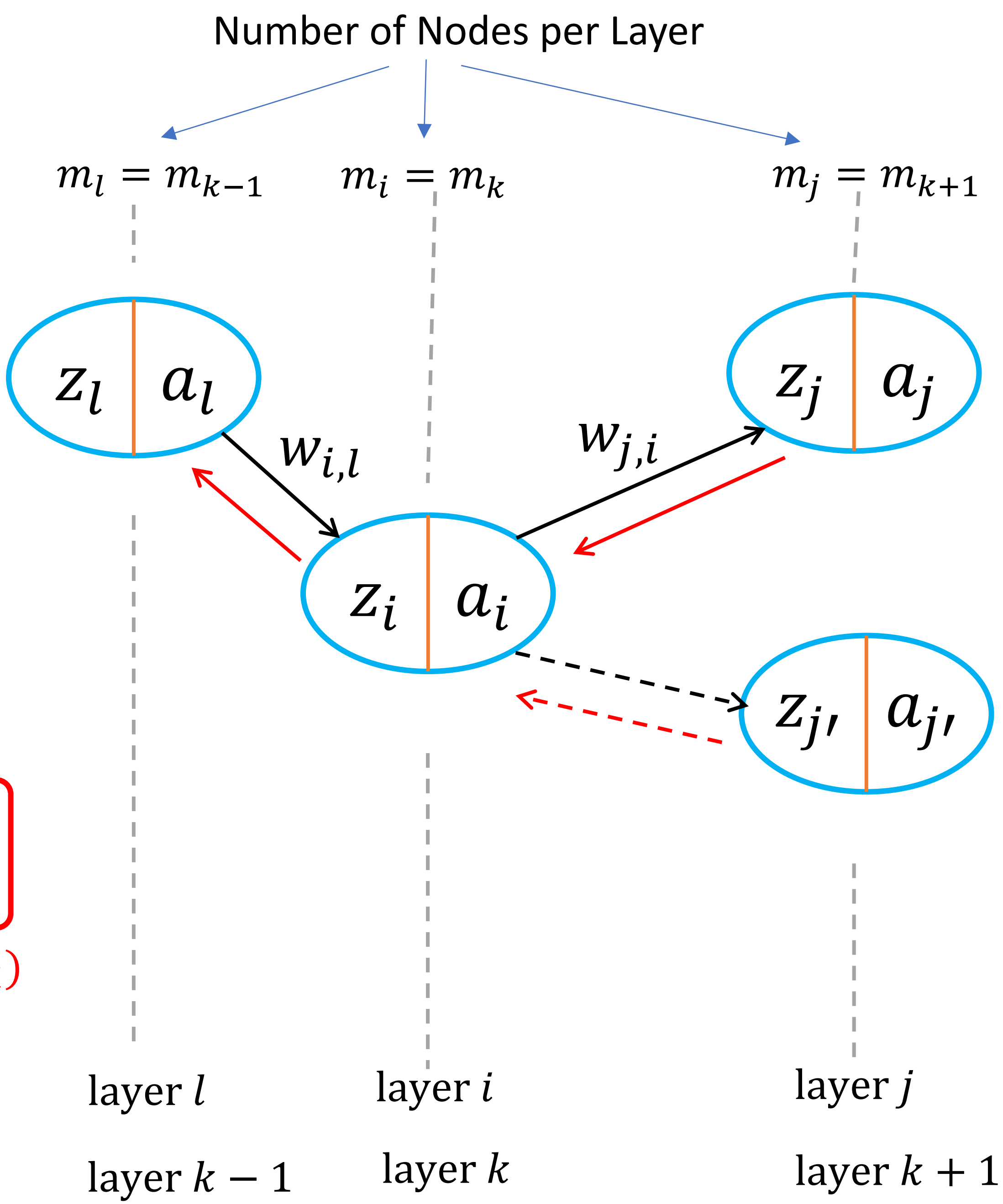
$$\frac{\partial \mathcal{L}}{\partial w_{i,l}} = \overset{\delta_i}{\boxed{\frac{\partial \mathcal{L}}{\partial z_i}}} \times \overset{a_l}{\boxed{\frac{\partial z_i}{\partial w_{i,l}}}}$$

$$\frac{\partial z_i}{\partial w_{i,l}} = \frac{\partial \sum_{l=1}^{m_l} w_{i,l}\, a_l}{\partial w_{i,l}} = a_l \qquad\qquad \frac{\partial z_i}{\partial a_l} = w_{i,l}$$

$$\delta_i = \frac{\partial \mathcal{L}}{\partial z_i} = \sum_{j=1}^{m_j} \frac{\partial \mathcal{L}}{\partial z_j} \times \frac{\partial z_j}{\partial z_i} = \sum_{j=1}^{m_j} \underset{\delta_j}{\boxed{\frac{\partial \mathcal{L}}{\partial z_j}}} \times \underset{w_{j,i}}{\boxed{\frac{\partial z_j}{\partial a_i}}} \times \underset{\sigma'(z_i)}{\boxed{\frac{\partial a_i}{\partial z_i}}}$$

$$\delta_i = \sigma'(z_i) \sum_{j=1}^{m_j} \delta_j w_{j,i}$$

$$\delta_i^{(k)} = \sigma'\left(z_i^{(k)}\right) \sum_{j=1}^{m_{k+1}} \delta_j^{(k+1)} w_{j,i}^{(k+1)}$$



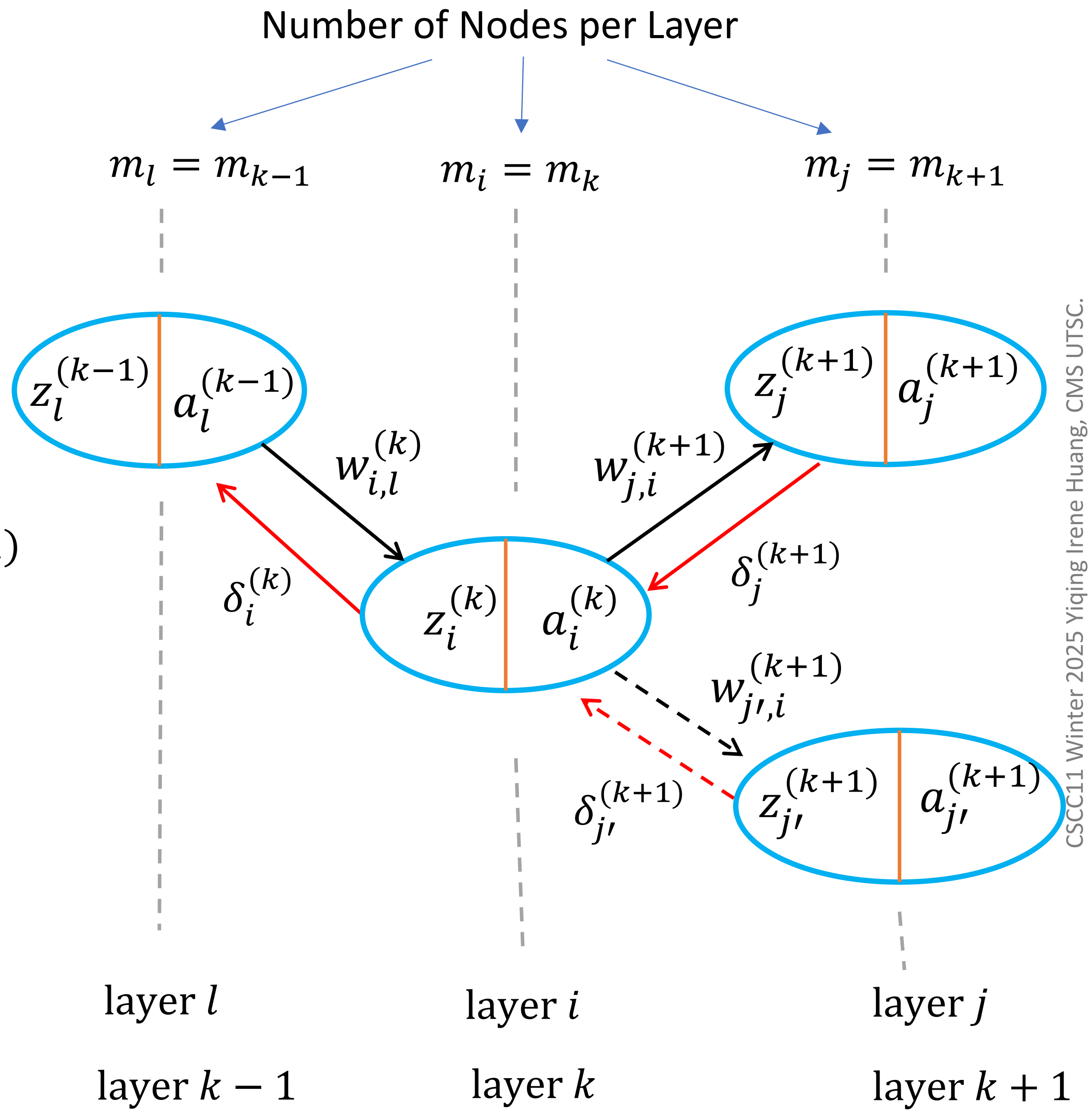
BackPropagation

$$\frac{\partial \mathcal{L}}{\partial w_{i,l}^{(k)}} = \delta_i^{(k)} a_l^{(k-1)}$$

$$\delta_i^{(k)} \equiv \frac{\partial \mathcal{L}}{\partial z_i^{(k)}} = \sigma'\left(z_i^{(k)}\right) \sum_{j=1}^{m_{k+1}} \delta_j^{(k+1)} w_{j,i}^{(k+1)}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(k)}} = \boldsymbol{\delta}^{(k)} \left(\tilde{\mathbf{a}}^{(k-1)}\right)^T \equiv \boldsymbol{\delta}^{(k)} \otimes \tilde{\mathbf{a}}^{(k-1)}$$

$$\boldsymbol{\delta}^{(k)} = ?$$



Backprop: Forward Pass

1. Random initialize the weights to small numbers (close to zeros)
2. Feed $\mathbf{x}$ into the FFNN input layer and compute the outputs of all input neurons
3. Propagate the outputs of each hidden layer forward, one hidden layer at a time, and compute the outputs of all hidden neurons
4. Compute the final output neuron
5. Compute the loss function

Backprop: Backward Pass

1. Compute $\boldsymbol{\delta}^{(L)}$
2. Compute $\delta_i^{(k)} = \sigma'\left(z_i^{(k)}\right)\sum_{j=1}^{m_l} \delta_j^{(k+1)} w_{j,i}^{(k+1)}$ from $k = L - 1$
3. Compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(k-1)}}$
4. Update weights according to gradient based method

Acknowledgement

- Prof. David Fleet developed the course. He made his notes and courseware available to all of us.
- Prof. Francisco (Paco) Estrada shared his assignments and insights.
- Prof. Rawad A. Assi shared past assignments and advices.

Neural Networks Brief History

- **1943:** McCulloch & Pitts introduce the first artificial neural networks to
- **1958:** Rosenblatt develops the Perceptron, a basic learning algorithm.
- **1969:** Minsky & Papert highlight the limitations of the Perceptron, leading to a temporary decline in interest.
- **1980s:** Backpropagation is popularized (Rumelhart, Hinton & Williams), sparking renewed research in multi-layer networks.
- **1990s:** Alternative methods (e.g., Support Vector Machines) gain prominence; neural networks face skepticism.
- **2000s:** Advancements in computing power and big data lead to the resurgence of neural networks (deep learning begins).
- **2012:** AlexNet's breakthrough on the ImageNet competition demonstrates the power of deep CNNs.
- **2010s – 2020s:** Explosion of deep learning applications with RNNs, LSTMs, GANs, and Transformers driving AI innovations.

BackPropagation

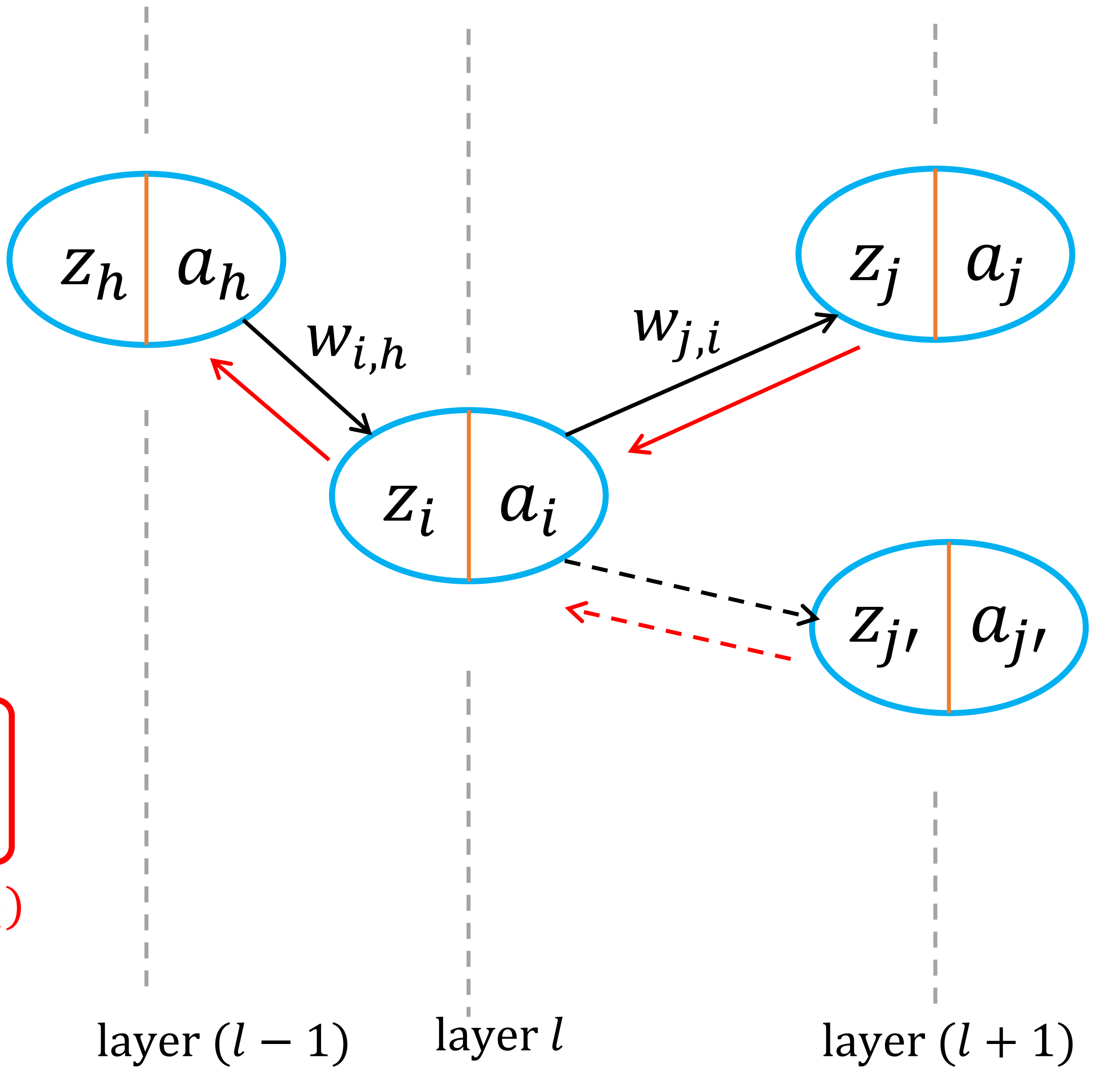
$$\frac{\partial \mathcal{L}}{\partial w_{i,h}} = \overbrace{\boxed{\frac{\partial \mathcal{L}}{\partial z_i}}}^{\delta_i} \times \overbrace{\boxed{\frac{\partial z_i}{\partial w_{i,h}}}}^{a_h}$$

$$\frac{\partial z_i}{\partial w_{i,h}} = \frac{\partial \sum_{h=1}^{m_{l-1}} w_{i,h}\, a_h}{\partial w_{i,h}} = a_h \qquad \frac{\partial z_i}{\partial a_h} = w_{i,h}$$

$$\delta_i = \frac{\partial \mathcal{L}}{\partial z_i} = \sum_{j=1}^{m_j} \frac{\partial \mathcal{L}}{\partial z_j} \times \frac{\partial z_j}{\partial z_i} = \sum_{j=1}^{m_j} \underbrace{\boxed{\frac{\partial \mathcal{L}}{\partial z_j}}}_{\delta_j} \times \underbrace{\boxed{\frac{\partial z_j}{\partial a_i}}}_{w_{j,i}} \times \underbrace{\boxed{\frac{\partial a_i}{\partial z_i}}}_{\sigma'(z_i)}$$

$$\delta_i = \sigma'(z_i) \sum_{j=1}^{m_j} \delta_j w_{j,i}$$

$$\delta_i^{(l)} = \sigma'\left(z_i^{(l)}\right) \sum_{j=1}^{m_{l+1}} \delta_j^{(l+1)} w_{j,i}^{(l+1)}$$



BackPropagation

$$\frac{\partial \mathcal{L}}{\partial w_{i,l}^{(k)}} = \delta_i^{(k)} a_l^{(k-1)}$$

$$\delta_i^{(k)} \equiv \frac{\partial \mathcal{L}}{\partial z_i^{(k)}} = \sigma'\left(z_i^{(k)}\right) \sum_{j=1}^{m_{k+1}} \delta_j^{(k+1)} w_{j,i}^{(k+1)}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(k)}} = \boldsymbol{\delta}^{(k)} \left(\tilde{\mathbf{a}}^{(k-1)}\right)^T \equiv \boldsymbol{\delta}^{(k)} \otimes \tilde{\mathbf{a}}^{(k-1)}$$

$$\boldsymbol{\delta}^{(k)} = \begin{bmatrix} \sigma'\left(z_1^{(k)}\right) & & \\ & \ddots & \\ & & \sigma'\left(z_k^{(k)}\right) \end{bmatrix} \left(\mathbf{W}[\mathbf{1}:]^{(k+1)}\right)^T \boldsymbol{\delta}^{(k+1)}$$

All rows after the first row