Artificial Neural Networks from Scratch

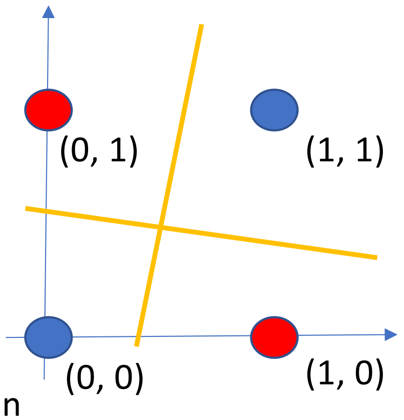
- Learn to build neural network from scratch.
  - Focus on multi-level feedforward neural networks (multi-level perceptrons)
- Training large neural networks is one of the most important workload in large scale parallel and distributed systems
  - Programming assignments throughout the semester will use this.

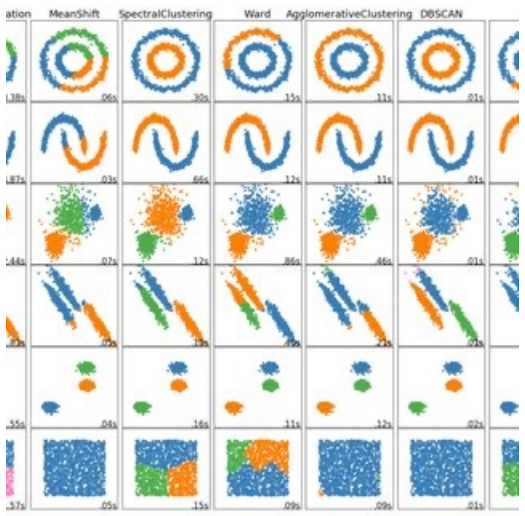
What do (deep) neural networks do?

- Learning (highly) non-linear functions.

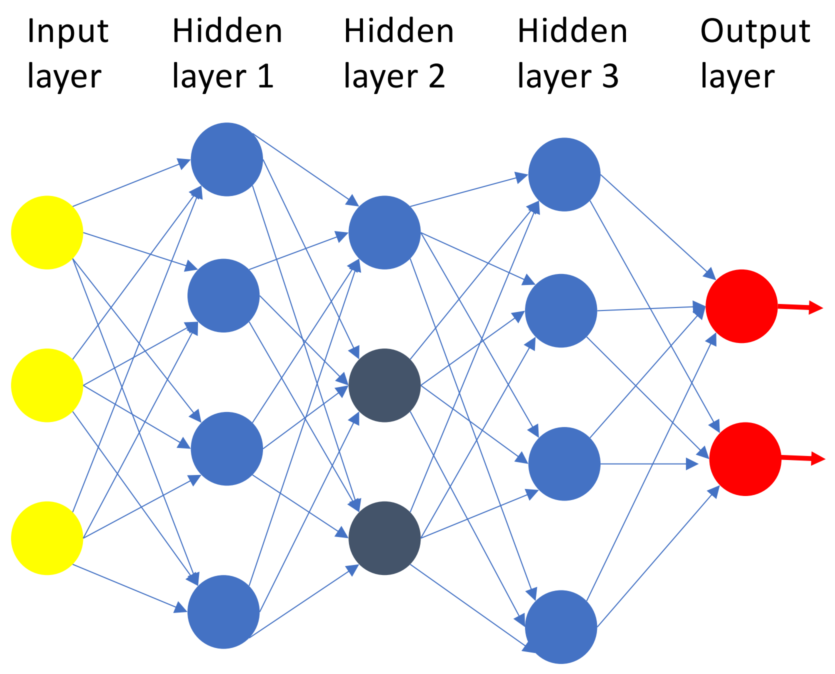
$x1$	$x2$	$x1 \oplus x2$
0	0	0
0	1	1
1	0	1
1	1	0

Logic XOR ($\oplus$) operation





Artificial neural network example



- A neural network consists of layers of artificial neurons and connections between them.
- Each connection is associated with a weight.
- Training of a neural network is to get to the right weights (and biases) such that the error across the training data is minimized.

Training a neural network

- A neural network is trained with m training samples

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots(x^{(m)}, y^{(m)})$$

  $x^{(i)}$ is an input vector, $y^{(i)}$ is an output vector
- Training objective: minimize the prediction error (loss)

$$min \sum_{i=1}^{m} (y^{(i)} - f_W(x^{(i)}))^2$$

  $f_W(x^{(i)})$ is the predicted output vector for the input vector $x^{(i)}$
- Approach: Gradient descent (stochastic gradient descent, batch gradient descent, mini-batch gradient descent).
  - Use error to adjust the weight value to reduce the loss. The adjustment amount is proportional to the contribution of each weight to the loss – Given an error, adjust the weight a little to reduce the error.

Stochastic gradient descent

- Given one training sample $(x^{(i)}, y^{(i)})$
- Compute the output of the neural network $f_{\overrightarrow{W}}(x^{(i)})$
- Training objective: minimize the prediction error (loss) – there are different ways to define error. The following is an example:

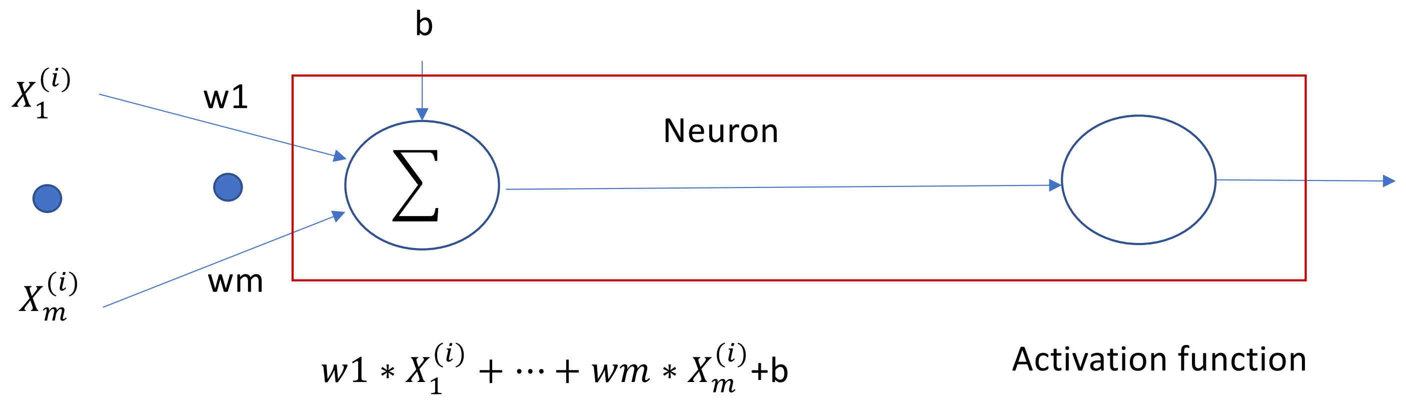
$$E = \frac{1}{2}(y^{(i)} - f_{\overrightarrow{W}}(x^{(i)}))^2$$

- Estimate how much each weight w_k in $\overrightarrow{W}$ contributes to the error: $\frac{\partial E}{\partial w_k}$
- Update the weight w_k by $w_k = w_k - \alpha \frac{\partial E}{\partial w_k}$. Here α is the learning rate.

Algorithm for learning artificial neural network

- Initialize the weights $\overrightarrow{W} = [W_0 \ , W_1 \ , \dots \dots , W_k \ ]$
- Training
  - For each training data $(x^{(i)}, y^{(i)})$, Using forward propagation to compute the neural network output vector $f_{\overrightarrow{W}}(x^{(i)})$
  - Compute the error E (various definitions)
  - Use backward propagation to compute $\frac{\partial E}{\partial W_k}$ for each weight W_k
  - Update W_k= $W_k - \alpha \frac{\partial E}{\partial W_k}$
  - Repeat until E is sufficiently small.

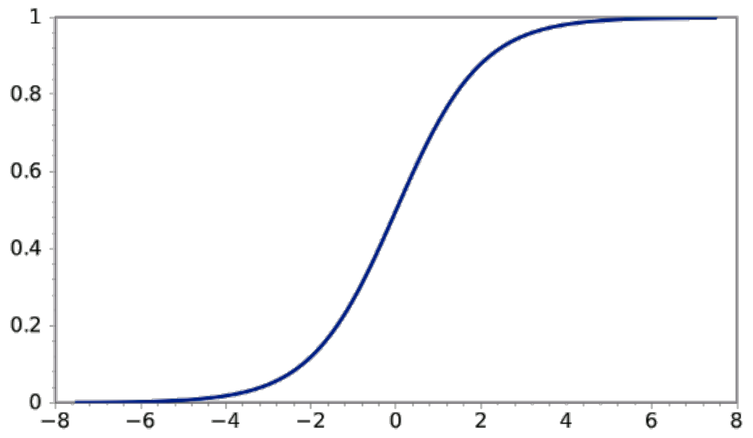
A single neuron



- An artificial neuron has two components: (1) weighted sum and activation function.
    - Many activation functions: Sigmoid, ReLU, etc.

Sigmoid function

- $\sigma(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$
- The derivative of the sigmoid function: $\frac{d}{dx}sigmoid(x) = sigmoid(x)\left(1 - sigmoid(x)\right)$
- $\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$

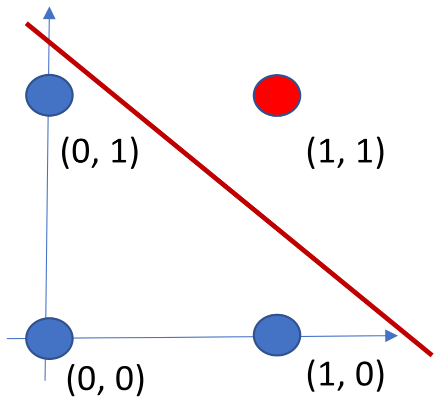


Training for the logic AND with a single neuron

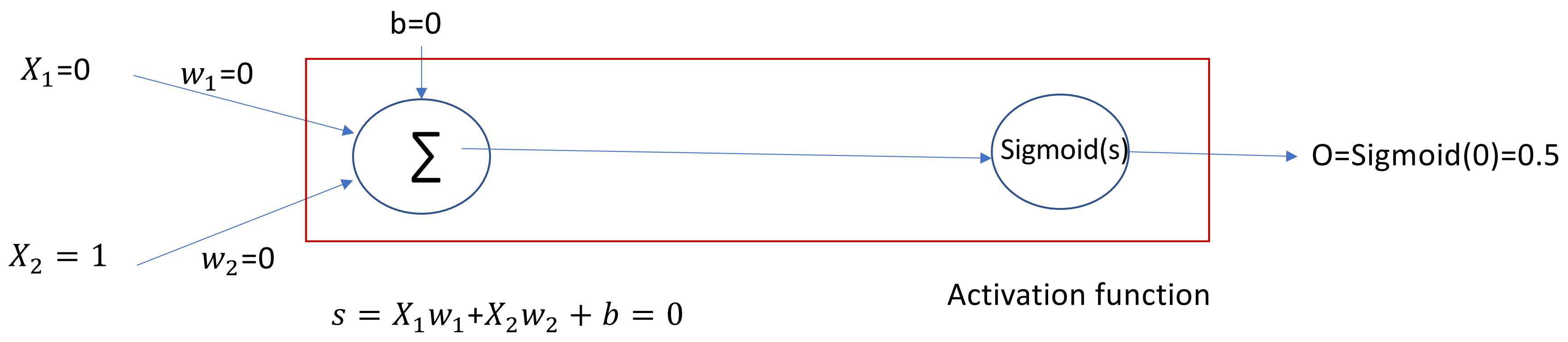
- In general, one neuron can be trained to realize a linear function.
- Logic AND function is a linear function:

$x1$	$x2$	$x1 \wedge x2$
0	0	0
0	1	0
1	0	0
1	1	1

Logic AND (∧) operation

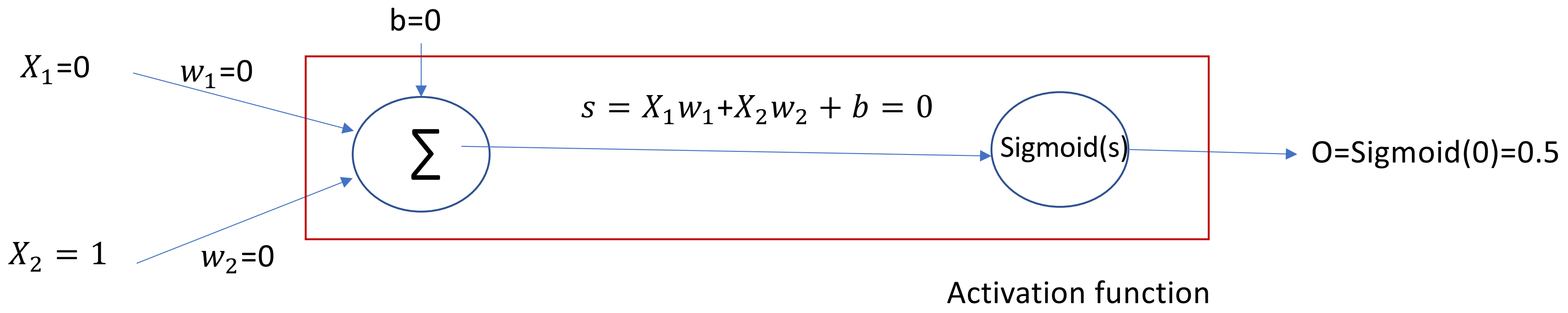


Training for the logic AND with a single neuron



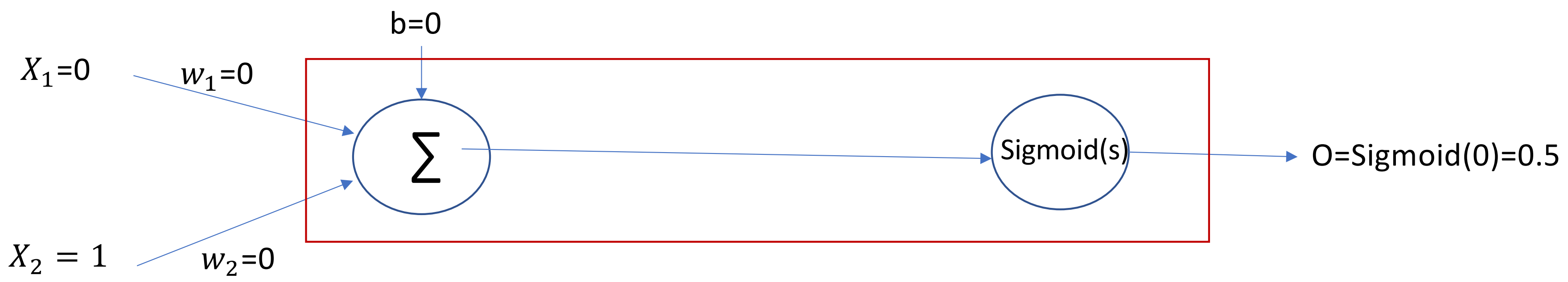
- Consider training data input (X_1=0, $X_2 = 1$), output Y=0.
- NN Output = 0.5
- Error: $E = \frac{1}{2}(Y - O)^2 = 0.125$
- To update w_1, w_2, and b, gradient descent needs to compute $\frac{\partial E}{\partial w_1}$, $\frac{\partial E}{\partial w_2}$, and $\frac{\partial E}{\partial b}$

Chain rules for calculating $\frac{\partial E}{\partial w_1}$, $\frac{\partial E}{\partial w_2}$, and $\frac{\partial E}{\partial b}$



- If a variable *z* depends on the variable *y*, which itself depends on the variable *x*, then *z* depends on *x* as well, via the intermediate variable *y*. The **chain rule** is a formula that expresses the derivative as : $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$
- $\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial s}\frac{\partial s}{\partial W_1}$

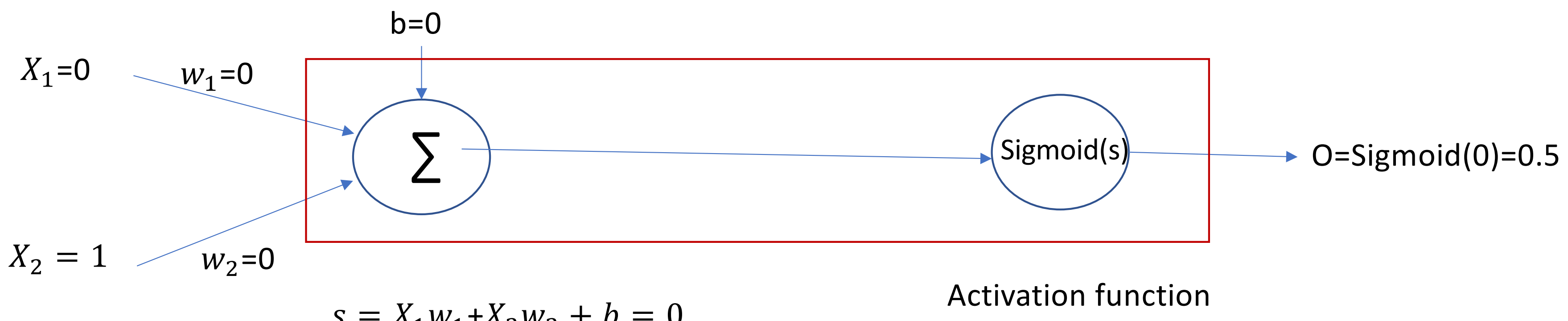
Training for the logic AND with a single neuron



$s = X_1 w_1 + X_2 w_2 + b = 0$

- $\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial O} \frac{\partial O}{\partial s} \frac{\partial s}{\partial W_1}$ $\frac{\partial E}{\partial O} = \frac{\partial(\frac{1}{2}(Y-O)^2)}{\partial O} = \mathrm{O} - \mathrm{Y} = 0.5 - 0 = 0.5$
- $\frac{\partial O}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s)) = 0.5 (1-0.5) = 0.25, $\frac{\partial s}{\partial W_1} = \frac{\partial(X_1 w_1 + X_2 w_2 + b)}{\partial W_1} =$ X_1 = 0
- To update w_1: $w_1 = w_1 - rate * \frac{\partial E}{\partial W_1}$ = 0 – 0.1*0.5*0.25*0 = 0
- Assume rate = 0.1

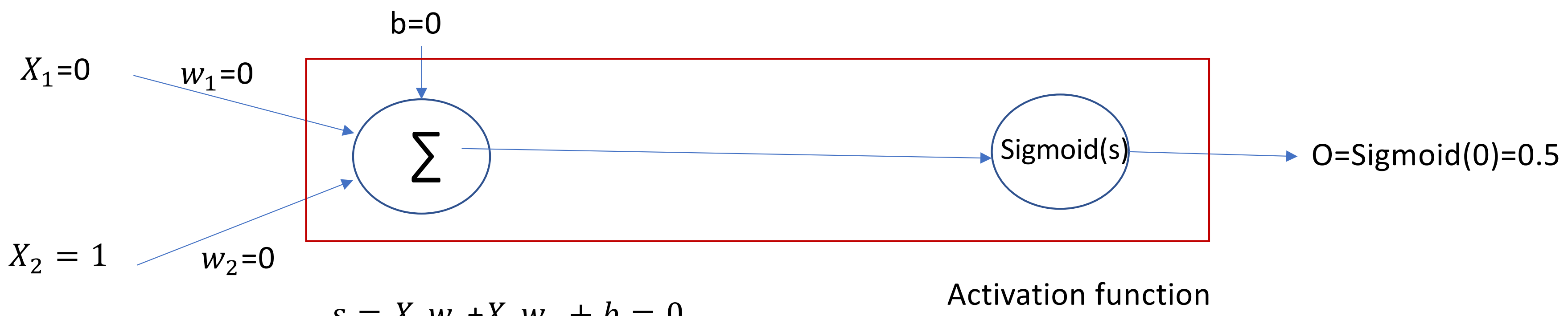
Training for the logic AND with a single neuron



$s = X_1 w_1 + X_2 w_2 + b = 0$

- $\frac{\partial E}{\partial W_2} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial s}\frac{\partial s}{\partial W_2}$ $\frac{\partial E}{\partial O} = \frac{\partial(\frac{1}{2}(Y-O)^2)}{\partial O} = \mathrm{O} - \mathrm{Y} = 0.5 - 0 = 0.5$
- $\frac{\partial O}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s)) = 0.5 (1-0.5) = 0.25, $\frac{\partial s}{\partial W_2}$= $\frac{\partial(X_1 w_1 + X_2 w_2 + b)}{\partial W_2} = X_2$ = 1
- To update w_2: $w_2 = w_2 - rate * \frac{\partial E}{\partial W_2}$ = 0 – 0.1*0.5*0.25*1 = -0.0125

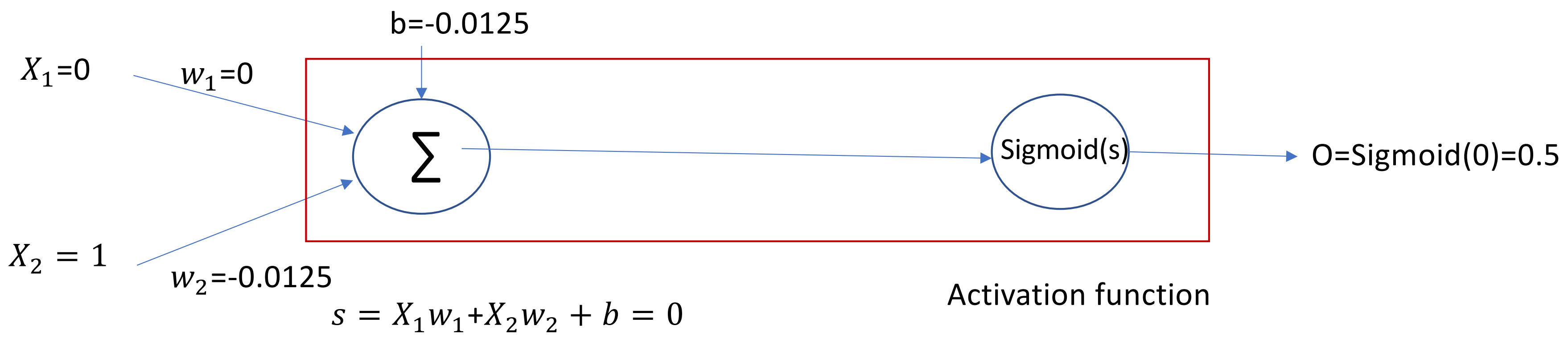
Training for the logic AND with a single neuron



$$s = X_1 w_1 + X_2 w_2 + b = 0$$

- $\frac{\partial E}{\partial b} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial s}\frac{\partial s}{\partial b}$ $\qquad \frac{\partial E}{\partial O} = \frac{\partial(\frac{1}{2}(Y-O)^2)}{\partial O} = \mathrm{O} - \mathrm{Y} = 0.5 - 0 = 0.5$
- $\frac{\partial O}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s)) = 0.5 (1-0.5) = 0.25, $\frac{\partial s}{\partial b} = \frac{\partial(X_1 w_1 + X_2 w_2 + b)}{\partial b} = 1$
- To update b: b= $b - rate * \frac{\partial E}{\partial b}$ = 0 – 0.1*0.5*0.25*1 = -0.0125

Training for the logic AND with a single neuron



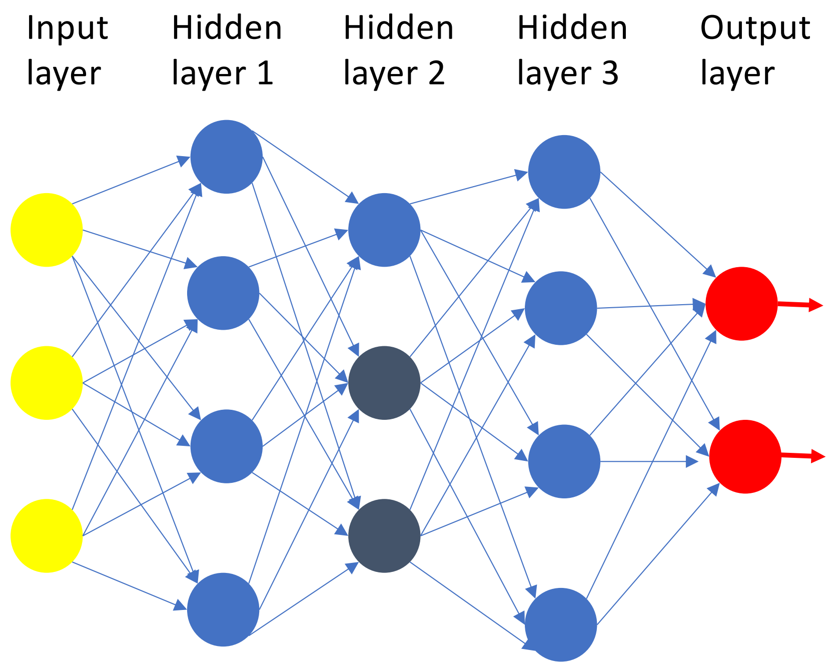
- This process is repeated until the error is sufficiently small
- The initial weight should be randomized. Gradient descent can get stuck in the local optimal.

Multi-level feedforward neural networks

- A multi-level feedforward neural network is a neural network that consists of multiple levels of neurons. Each level can have many neurons and connections between neurons in different levels do not form loops.
  - Information moves in one direction (forward) from input nodes, through hidden nodes, to output nodes.
- One artificial neuron can only realize a linear function
- Many levels of neurons can combine linear functions can train arbitrarily complex functions.

Multi-level feedforward neural networks examples

- A layer of neurons that do not directly connect to outputs is called a hidden layer.



3-level feedforward neural network

Output: OO[N2]

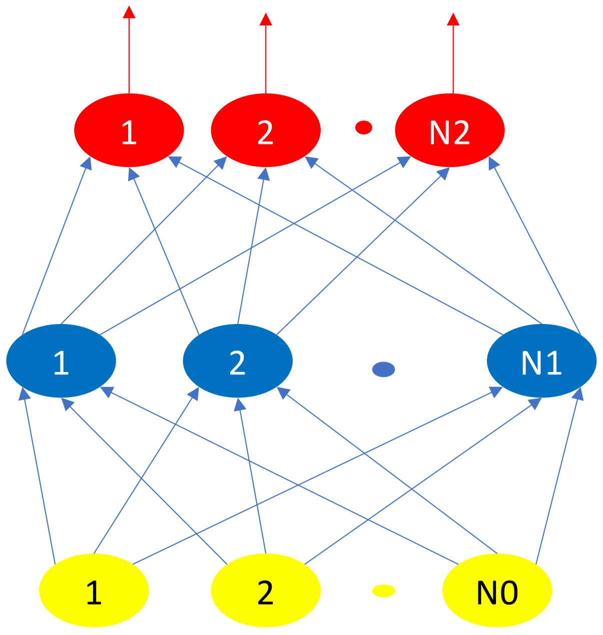
Output layer

Hidden layer
Output: HO[N1]

Hidden layer

Input layer

Input: IN[N0]



Hidden layer biases: B2[N2]

Weight: W1[N1][N2]

Hidden layer biases: B1[N1]

Weight: W0[N0][N1]

Hidden layer weighted sum: HS[N1]
Output layer weighted sum: HS[N2]

Build a 3-level neural network from scratch

- 3 levels: Input level, hidden level, output level
  - Other assumptions: fully connected between layers, all neurons use sigmoid (σ) as the activation function.
- Notations:
  - *N0*: size of the input level. Input: $IN[N0] = [IN_1, IN_2, \ldots, IN_{No}]$
  - *N1*: size of the hidden layer
  - *N2*: size of the output layer. Output: OO$[N2] = [OUT_1, OUT_2, \ldots, OUT_{N2}]$

Build a 3-level neural network from scratch

- Notations:
  - *N0, N1, N2*: sizes of the input layer, hidden layer, and output layer, respectively
  - *N0*×*N1* weights from input layer to hidden layer. $W0_{ij}$: the weight from input unit i to hidden unit j. *B0[N1]* biases. *B0*$[N1] = [B0_1, B0_2, \dots, B0_{N1}]$
  - *N1*×*N2* weights from hidden layer to output layer. $W1_{ij}$: the weight from hidden unit i to output unit j. *B1[N2]* biases. *B1*$[N2] = [B1_1, B1_2, \dots, B1_{N2}]$
  - $W0[N0][N1] = \begin{pmatrix} W0_{1,1} & \cdots & W0_{1,N1} \\ \vdots & \ddots & \vdots \\ W0_{N0,1} & \cdots & W0_{N0,N1} \end{pmatrix}$, $W1[N1][N2] = \begin{pmatrix} W0_{1,1} & \cdots & W0_{1,N2} \\ \vdots & \ddots & \vdots \\ W0_{N1,1} & \cdots & W0_{N1,N2} \end{pmatrix}$

Forward propogation (compute OO and E)

- Compute hidden layer weighted sum: $HS[N1] = [HS_1, HS_2, \ldots, HS_{N1}]$
    - $HS_i = IN_1 \times W0_{1,i} + IN_2 \times W0_{2,i} + \cdots + IN_{N0} \times W0_{N0,i} + B1_i$
    - In matrix form: $HS = IN \times W0 + B1$
- Compute hidden layer output: $HO[N1] = [HO_1, HO_2, \ldots, HO_{N1}]$
    - $HO_i = \sigma(HS_i)$
    - In matrix form: $HO = \sigma(HS)$

Forward propogation

- From input (IN[N0]), compute output (OO[N2]) and error E.
- Compute output layer weighted sum: $OS[N2] = [OS_1, OS_2, \ldots, OS_{N2}]$
  - $OS_i = HO_1 \times W1_{1,i} + HO_2 \times W1_{2,i} + \cdots + HO_{N1} \times W1_{N1,i} + B2_i$
  - In matrix form: $HS = HO \times W1 + B2$
- Compute final output: $OO[N2] = [OO_1, OO_2, \ldots, OO_{N1}]$
  - $OO_i = \sigma(OS_i)$
  - In matrix form: $OO = \sigma(OS)$
- Let us use mean square error: $E = \frac{1}{N2}\sum_{i=1}^{N2}(OO_i - Y_i)^2$

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_{i}}$, and $\frac{\partial E}{\partial B2_{i}}$.

- $\frac{\partial E}{\partial OO} = \left[\frac{\partial E}{\partial OO_1}, \frac{\partial E}{\partial OO_2}, \ldots, \frac{\partial E}{\partial OO_{N2}}\right] = [\frac{2}{N2}(OO_1 - Y_1), \frac{2}{N2}(OO_2 - Y_2), \ldots, \frac{2}{N2}(OO_{N2} - Y_{N2})]$
- In matrix form: $\frac{\partial E}{\partial OO} = \frac{2}{N2}(OO - Y)$
- This can be stored in an array dE_OO[N2];

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}$ is done
- $\frac{\partial E}{\partial OS} = \left[\frac{\partial E}{\partial OO_1}\frac{\partial OO_1}{\partial OS_1}, \frac{\partial E}{\partial OO_2}\frac{\partial OO_2}{\partial OS_2}, \ldots, \frac{\partial E}{\partial OO_{N2}}\frac{\partial OO_{N2}}{\partial OS_{N2}}\right]$ =

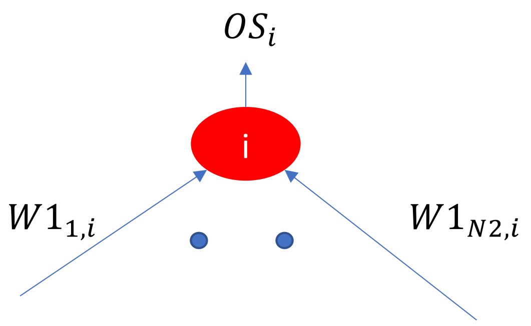
  $\left[\frac{\partial E}{\partial OO_1}\sigma(OS_1)(1-\sigma(OS_1)), \ldots, \frac{\partial E}{\partial OO_{N2}}\sigma(OS_{N2})(1-\sigma(OS_{N2}))\right]$
- In matrix form: $\frac{\partial E}{\partial OS}$ = $\frac{2}{N2}(OO - Y) \odot OO \odot (1 - OO)$
- This can be stored in an array dE_OS[N2];

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}, \frac{\partial E}{\partial OS}$ are done
- $\frac{\partial E}{\partial B2} = \left[\frac{\partial E}{\partial Os_1}\frac{\partial Os_1}{\partial B2_1}, \frac{\partial E}{\partial Os_2}\frac{\partial OS_2}{\partial B2_2}, \dots, \frac{\partial E}{\partial OO_{N2}}\frac{\partial OO_{N2}}{\partial B2_{N2}}\right]$
- $OS_i = HO_1 \times W1_{1,i} + HO_2 \times W1_{2,i} + \cdots + HO_{N1} \times W1_{N1,i} + B2_i$
- Hence, $\frac{\partial Os_i}{\partial B2_i} = 1.$
- $\frac{\partial E}{\partial B2} = \left[\frac{\partial E}{\partial Os_1}, \frac{\partial E}{\partial Os_2}, \dots, \frac{\partial E}{\partial OO_{N2}}\right] = \quad \frac{\partial E}{\partial OS}$

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}$, $\frac{\partial E}{\partial OS}$, $\frac{\partial E}{\partial B2}$ are done
- $\frac{\partial E}{\partial W1} = \begin{pmatrix} \frac{\partial E}{\partial OS_1}\frac{\partial OS_1}{\partial W1_{1,1}} & \cdots & \frac{\partial E}{\partial OS_{N2}}\frac{\partial OS_{N2}}{\partial W1_{1,N2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial OS_1}\frac{\partial OS_1}{\partial W1_{N1,1}} & \cdots & \frac{\partial E}{\partial OS_{N2}}\frac{\partial OS_{N2}}{\partial W1_{N1,,N2}} \end{pmatrix}$
- $OS_i = HO_1 \times W1_{1,i} + HO_2 \times W1_{2,i} + \cdots + HO_{N1} \times W1_{N1,i} + B2_i$
- Hence, $\frac{\partial Os_i}{\partial W1j_{,i}} = HO_j$.



Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}, \frac{\partial E}{\partial OS}, \frac{\partial E}{\partial B2}$ are done
- $\frac{\partial E}{\partial W1} = \begin{pmatrix} \frac{\partial E}{\partial OS_1}\frac{\partial OS_1}{\partial W1_{1,1}} & \cdots & \frac{\partial E}{\partial OS_{N2}}\frac{\partial OS_{N2}}{\partial W1_{1,N2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial OS_1}\frac{\partial OS_1}{\partial W1_{N1,1}} & \cdots & \frac{\partial E}{\partial OS_{N1}}\frac{\partial OS_{N2}}{\partial W1_{N1,,N2}} \end{pmatrix} = \begin{pmatrix} \frac{\partial E}{\partial OS_1}HO_1 & \cdots & \frac{\partial E}{\partial OS_{N2}}HO_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial OS_1}HO_{N1} & \cdots & \frac{\partial E}{\partial OS_{N2}}HO_{N1} \end{pmatrix}$
- In matrix form: $\frac{\partial E}{\partial W1} = HO^T \frac{\partial E}{\partial OS}$

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}, \frac{\partial E}{\partial OS}, \frac{\partial E}{\partial B2}, \frac{\partial E}{\partial W1}$ are done
- $\frac{\partial E}{\partial HO} = [\frac{\partial E}{\partial HO_1}, \frac{\partial E}{\partial HO_2}, \dots \dots, \frac{\partial E}{\partial HO_{N1}}]$
- $\frac{\partial E}{\partial HO_i} = \frac{\partial E}{\partial OS_1}\frac{\partial OS_1}{\partial HO_i} + \frac{\partial E}{\partial OS_2}\frac{\partial OS_2}{\partial HO_i} + \dots + \frac{\partial E}{\partial OS_{N2}}\frac{\partial OS_{N2}}{\partial HO_i} = \frac{\partial E}{\partial OS_1} W1_{i,1} + \frac{\partial E}{\partial OS_2} W1_{i,2} + \dots + \frac{\partial E}{\partial OS_{N2}} W1_{i,N2}$

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}, \frac{\partial E}{\partial OS}, \frac{\partial E}{\partial B2}, \frac{\partial E}{\partial W1}$ are done
- $\frac{\partial E}{\partial HO} = [\frac{\partial E}{\partial HO_1}, \frac{\partial E}{\partial HO_2}, \dots \dots, \frac{\partial E}{\partial HO_{N1}}] = \frac{\partial E}{\partial OS} W1^T$

Backward propogation

- To goal is to compute $\frac{\partial E}{\partial W0_{i,j}}$, $\frac{\partial E}{\partial W1_{i,j}}$, $\frac{\partial E}{\partial B1_i}$, and $\frac{\partial E}{\partial B2_i}$.
- $\frac{\partial E}{\partial OO}$, $\frac{\partial E}{\partial OS}$, $\frac{\partial E}{\partial B2}$, $\frac{\partial E}{\partial W1}$, $\frac{\partial E}{\partial HO}$ are done
- Once $\frac{\partial E}{\partial HO}$ is computed, we can repeat the process for the hidden layer by replacing OO with HO, OS with HS, B2 with B1 and W2 with W1, in the differential equation. Also the input is IN[N0] and the output is HO[N1].

Summary

$$\begin{tikzcd} IN \arrow[r] & \boxed{\text{layer}} \arrow[r] & O \end{tikzcd}$$

$$\begin{tikzcd} X \arrow[r] & \boxed{\text{Layer 1}} \arrow[r, "H1"] & \boxed{\text{Layer 2}} \arrow[r, "H2"] & \boxed{\text{Layer 3}} \arrow[r] & Y \end{tikzcd}$$

$$\begin{tikzcd} \frac{\partial E}{\partial IN} & \boxed{\text{layer}} \arrow[l] & \frac{\partial E}{\partial O} \arrow[l] \end{tikzcd}$$

- The output of a layer is the input of the next layer.
- Backward propagation uses results from forward propagation.
  - $\frac{\partial E}{\partial IN} = \frac{\partial E}{\partial O} W^T, \frac{\partial E}{\partial W} = IN^T \frac{\partial E}{\partial O}, \frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}$