THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

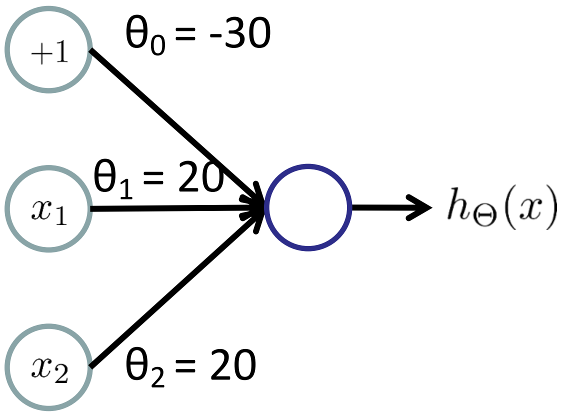
# COMP4434 Big Data Analytics

# Lab 6 Logic Gate Neural Network, Multilayer perceptron

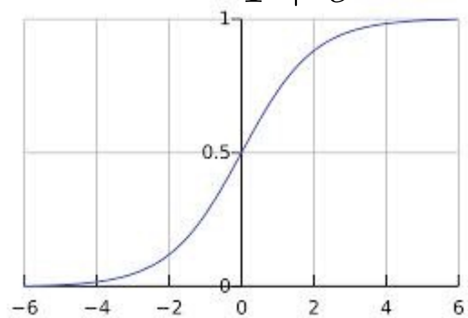
HUANG Xiao
xiaohuang@comp.polyu.edu.hk

# AND Example

- AND: $y = x_1 \wedge x_2$



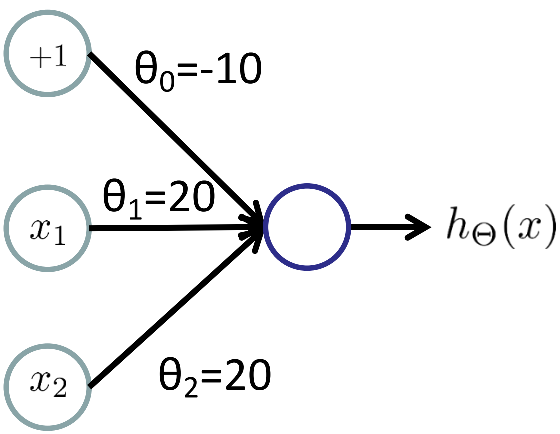

$$g(z) = \frac{1}{1+e^{-z}}$$



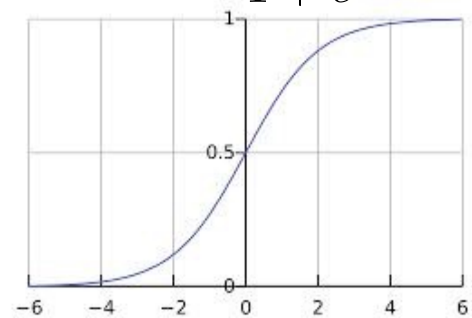

| $x_1$ | $x_2$ | $g(\theta_0+\theta_1 x_1+\theta_2 x_2)$ |
|---|---|---|
| 0 | 0 | $g(-30) \approx 0$ |
| 0 | 1 | $g(-10) \approx 0$ |
| 1 | 0 | $g(-10) \approx 0$ |
| 1 | 1 | $g(10) \approx 1$ |

# OR Example

- OR: $y = x_1 \vee x_2$



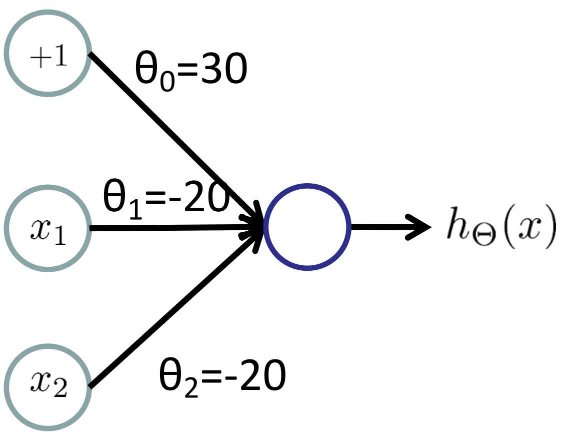

$$g(z) = \frac{1}{1 + e^{-z}}$$



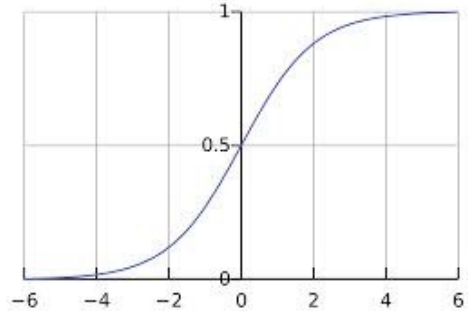

| $x_1$ | $x_2$ | $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ |
|---|---|---|
| 0 | 0 | $g(-10) \approx 0$ |
| 0 | 1 | $g(10) \approx 1$ |
| 1 | 0 | $g(10) \approx 1$ |
| 1 | 1 | $g(30) \approx 1$ |

# NAND Example

- $y = x_1 NAND x_2$



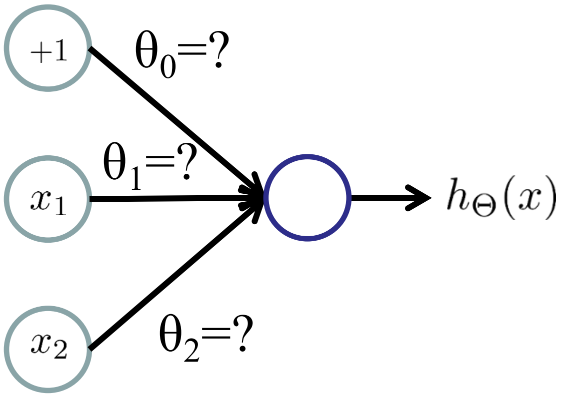

$$g(z) = \frac{1}{1+e^{-z}}$$




| $x_1$ | $x_2$ | $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ |
|---|---|---|
| 0 | 0 | $g(30) \approx 1$ |
| 0 | 1 | $g(10) \approx 1$ |
| 1 | 0 | $g(10) \approx 1$ |
| 1 | 1 | $g(-10) \approx 0$ |

# Feature Learning: NOR Example

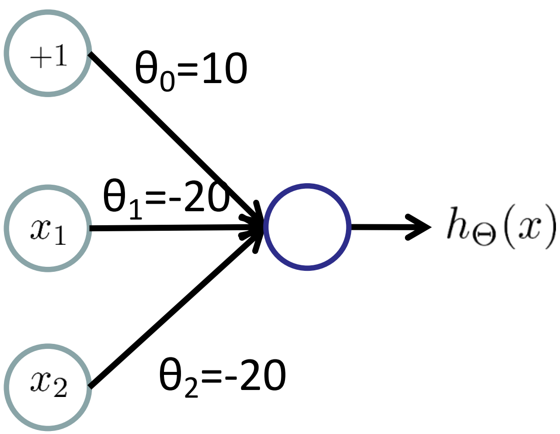
- $y = x_1 NOR x_2$



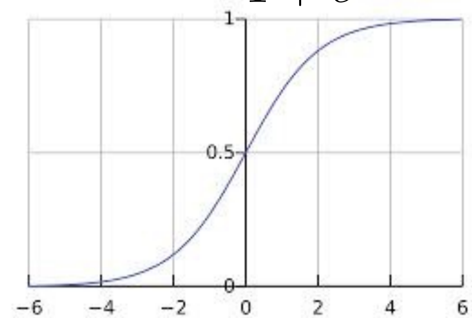

| $x_1$ | $x_2$ | $h$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

# NOR Example

- $y = x_1 NOR x_2$



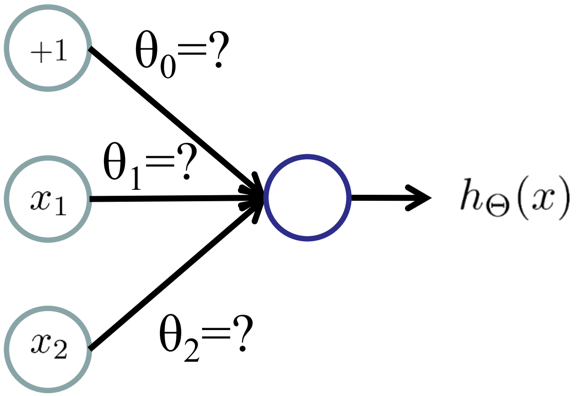

$$g(z) = \frac{1}{1+e^{-z}}$$




| $x_1$ | $x_2$ | $g(\theta_0+\theta_1 x_1+\theta_2 x_2)$ |
|---|---|---|
| 0 | 0 | $g(10) \approx 1$ |
| 0 | 1 | $g(-10) \approx 0$ |
| 1 | 0 | $g(-10) \approx 0$ |
| 1 | 1 | $g(-30) \approx 0$ |

# Feature Learning: XOR Example

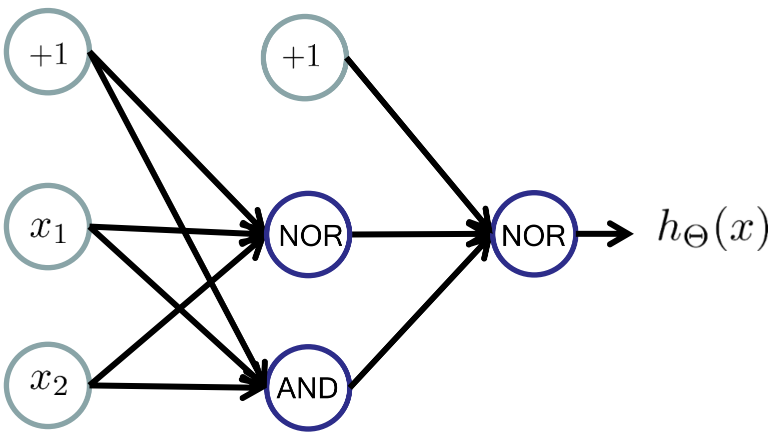
- $y = x_1 XOR x_2$



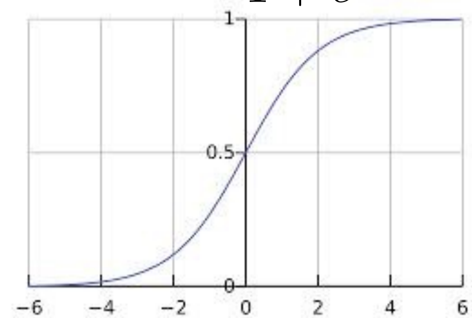

| $x_1$ | $x_2$ | $h$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# XOR Example

- $y = x_1 XOR x_2$



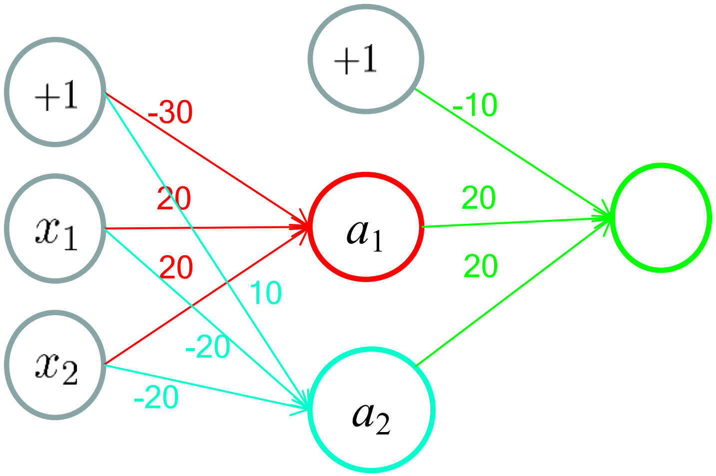

$$g(z) = \frac{1}{1+e^{-z}}$$




| $x_1$ | $x_2$ | $(NOR, AND)$ | $NOR$ |
|---|---|---|---|
| 0 | 0 | $(1, 0)$ | 0 |
| 0 | 1 | $(0, 0)$ | 1 |
| 1 | 0 | $(0, 0)$ | 1 |
| 1 | 1 | $(0, 1)$ | 0 |

# XNOR Example

| $x_1$ | $x_2$ | $a_1$ | $a_2$ | $h$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |




$$a_1 = g(-30 + 20x_1 + 20x_2)$$

$$a_2 = g(10 - 20x_1 - 20x_2)$$

$$h(x) = g(-10 + 20a_1 + 20a_2) = g(-10 + 20g(-30 + 20x_1 + 20x_2) + 20g(10 - 20x_1 - 20x_2))$$

# Further Practice

## Further tasks:

- Implement XOR gate using another different model
- Implement XNOR gate by yourself

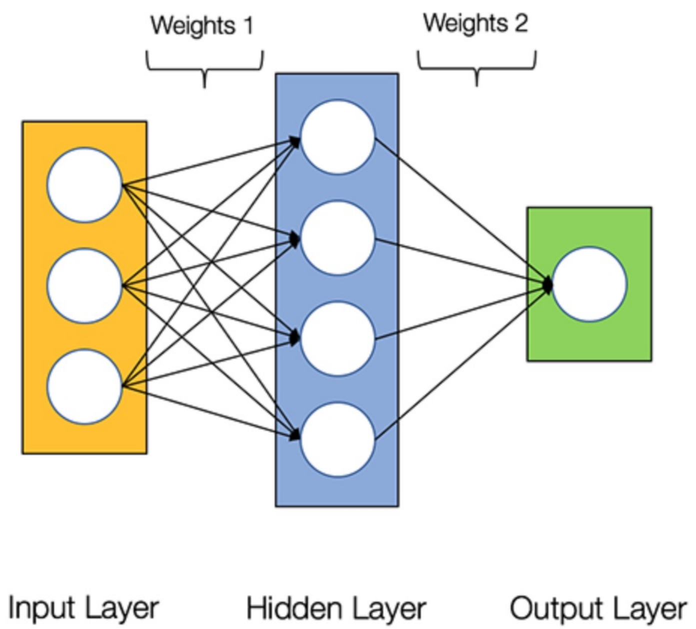
## Further readings:

- https://towardsdatascience.com/implementing-logic-gates-using-neural-networks-part-2-b284cc159fce
- https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1
- https://towardsdatascience.com/emulating-logical-gates-with-a-neural-network-75c229ec4cc9
- https://en.wikipedia.org/wiki/Logic_gate

# Recap: Neural Network

Neural Networks consist of the following components：

- An **input layer**, $\boldsymbol{x}$
- An arbitrary amount of **hidden layers**
- An **output layer**, $\hat{y}$
- A set of **weights** and **biases** between each layer, $\boldsymbol{W}$ **and** $\boldsymbol{b}$
- A choice of **activation function** for each hidden layer, $\boldsymbol{\sigma}$.

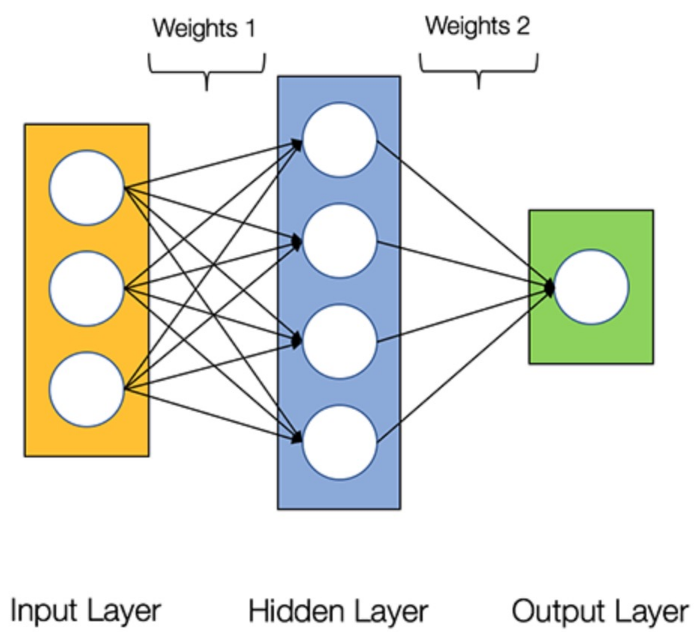



Architecture of a 2-layer Neural Network

In this tutorial, we'll use a Sigmoid activation function.

# Creating a Neural Network Class

```
In [ ]: class NeuralNetwork:
            def __init__(self, x, y):
                self.input      = x
                self.weights1   = np.random.rand(self.input.shape[1],4)
                self.weights2   = np.random.rand(4,1)
                self.y          = y
                self.output     = np.zeros(y.shape)
```



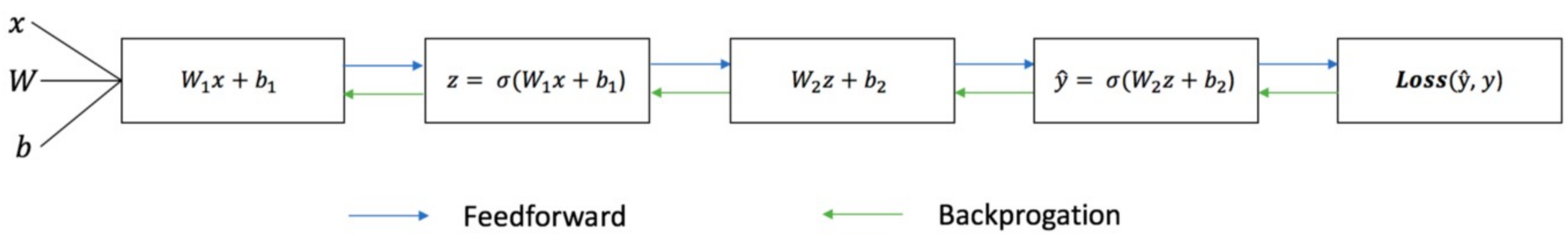

Architecture of a 2-layer Neural Network

# Training the Neural Network

The output $\hat{y}$ of a simple 2-layer Neural Network is:

$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

Each iteration of the training process consists of the following steps:



# Feedforward

$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

In [ ]:
```
class NeuralNetwork:
    def __init__(self, x, y):
        self.input      = x
        self.weights1   = np.random.rand(self.input.shape[1],4)
        self.weights2   = np.random.rand(4,1)
        self.y          = y
        self.output     = np.zeros(self.y.shape)

    def feedforward(self):
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.output = sigmoid(np.dot(self.layer1, self.weights2))
```

# Loss Function and Backpropagation

$$Loss(y,\hat{y}) = \sum_{i=1}^{n}(y - \hat{y})^2$$

$$\frac{\partial\, Loss(y,\hat{y})}{\partial W} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial W} \quad \text{where } z = Wx + b$$

$$= 2(y - \hat{y}) * \text{derivative of sigmoid function} * x$$

$$= 2(y - \hat{y}) * \mathrm{z(1\text{-}z)} * x$$

```
In [ ]: class NeuralNetwork:
            def __init__(self, x, y):
                self.input      = x
                self.weights1   = np.random.rand(self.input.shape[1],4)
                self.weights2   = np.random.rand(4,1)
                self.y          = y
                self.output     = np.zeros(self.y.shape)

            def feedforward(self):
                self.layer1 = sigmoid(np.dot(self.input, self.weights1))
                self.output = sigmoid(np.dot(self.layer1, self.weights2))

            def backprop(self):
                # application of the chain rule to find derivative of the loss function with respe
                d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) * sigmoid_derivative(
                d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) * sigmoid_deri

                # update the weights with the derivative (slope) of the loss function
                self.weights1 += d_weights1
                self.weights2 += d_weights2
```

# Putting it all together

| $X1$ | $X2$ | $X3$ | $Y$ |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

# Results


| Prediction | $Y$(Actual) |
|---|---|
| 0.0099 | 0 |
| 0.969 | 1 |
| 0.968 | 1 |
| 0.037 | 0 |

# Further Practice

## Further tasks:

- Derive the presentation function weights1's derivative by hand.
- Using 3rd party package to implement the rating prediction model for the individual project.

## Further readings:

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html https://realpython.com/python-ai-neural-network/
- https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53