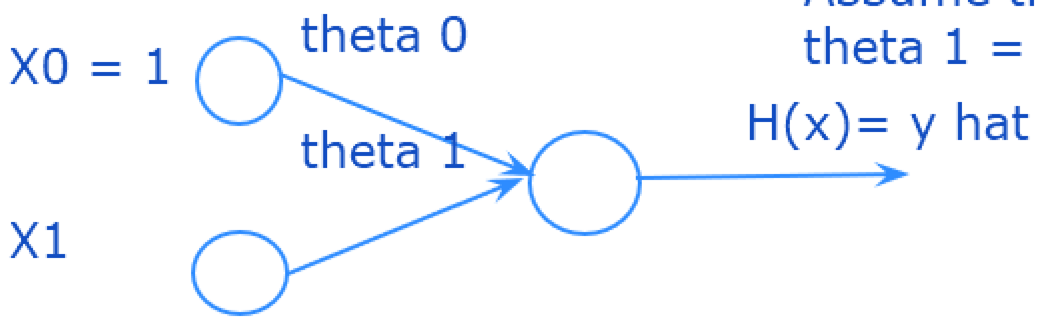
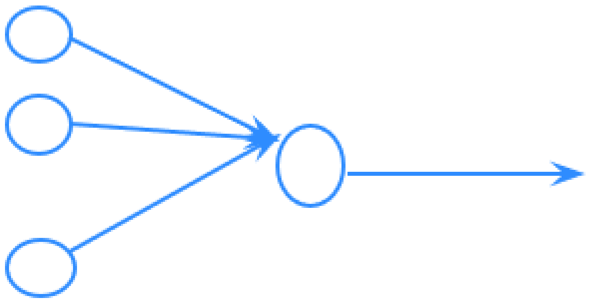
Example: Logic Gate (NOT)

Assume theta 0 = 20
theta 1 = -30

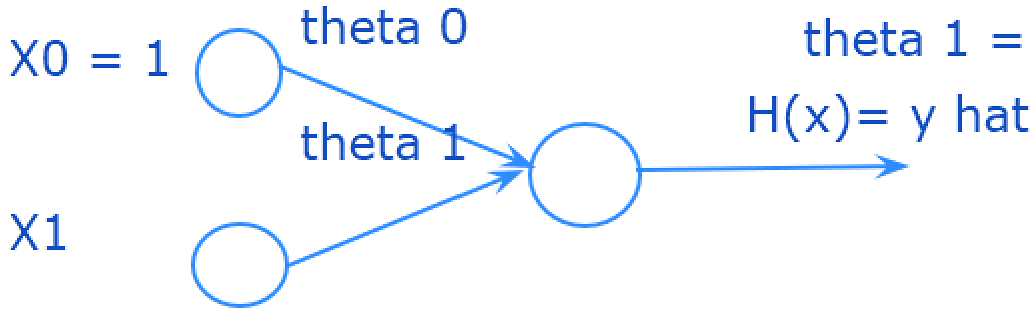


X1	h(X)
0	1
1	0

h(x) = g(20 -30X1) = y hat
if x1 = 0; h(x) = g(20) = y hat = 1
if x1 = 1; h(x) = g(20-30) = g(-10) = y hat = 0



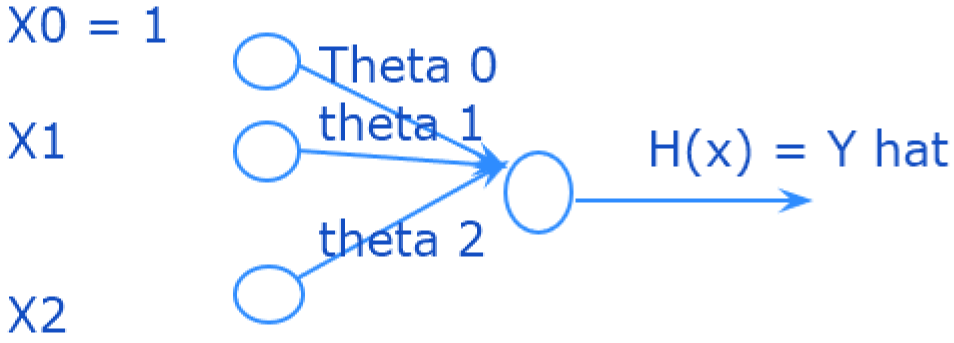
Example: Logic Gate (NOT)



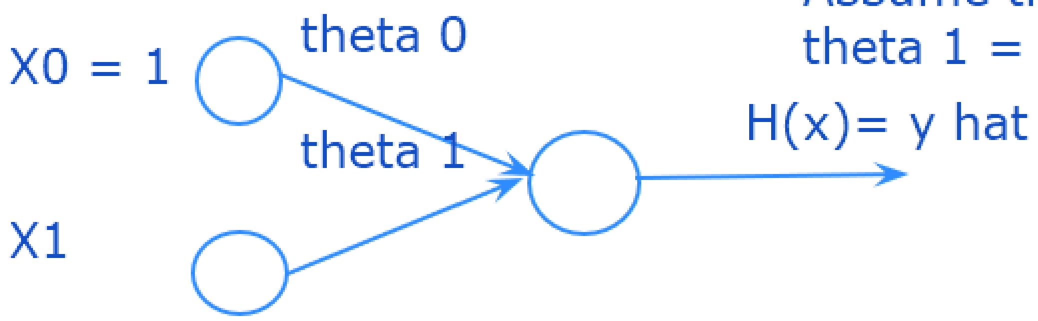
Assume theta 0 = 20
theta 1 = -30

X1	h(X)
0	1
1	0

h(x) = g(20 -30X1) = y hat
if x1 = 0; h(x) = g(20) = y hat = 1
if x1 = 1; h(x) = g(20-30) = g(-10) = y hat = 0



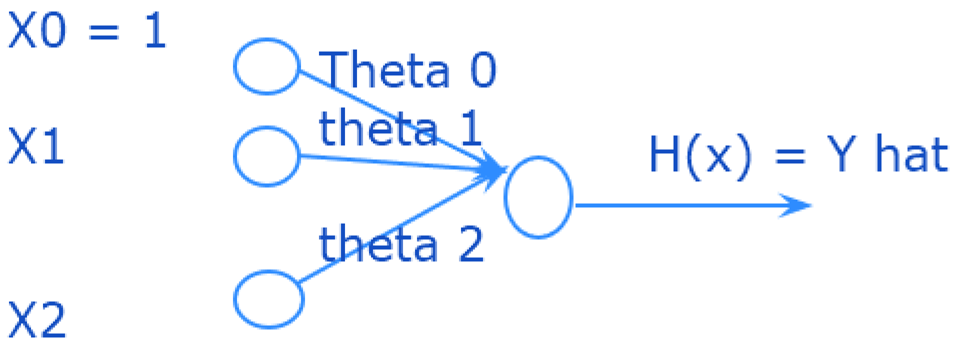
Example: Logic Gate (NOT)



Assume theta 0 = 20
theta 1 = -30

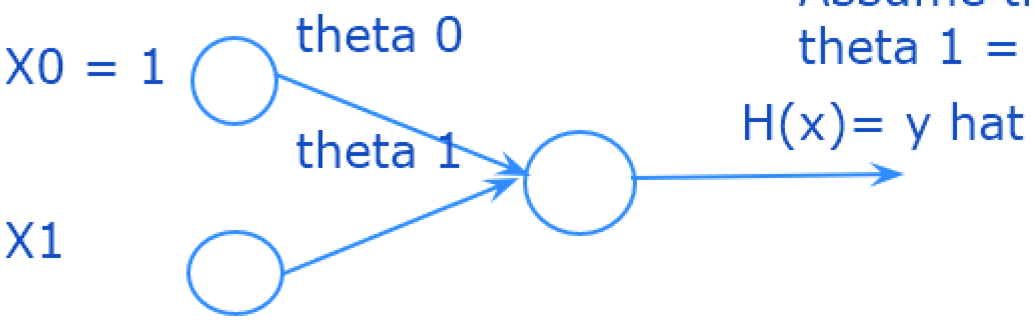
X1	h(X)
0	1
1	0

h(x) = g(20 -30X1) = y hat
if x1 = 0; h(x) = g(20) = y hat = 1
if x1 = 1; h(x) = g(20-30) = g(-10) = y hat = 0



Assume: theta 0 = -40
theta 1 = 30
theta 2 = 30

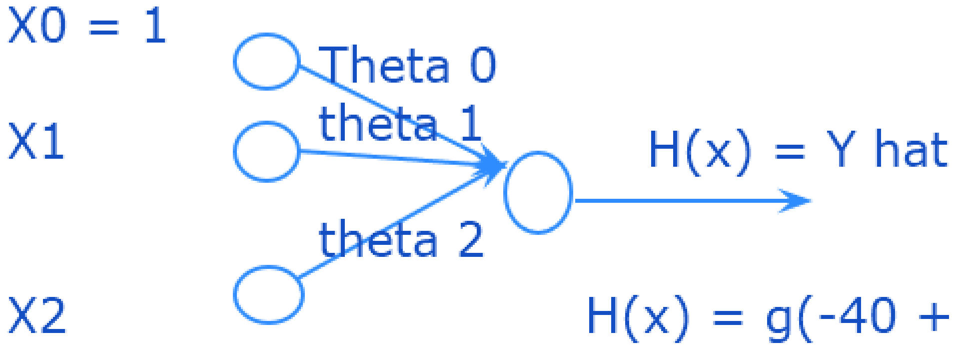
Example: Logic Gate (NOT)



Assume theta 0 = 20
theta 1 = -30

X1	h(X)
0	1
1	0

h(x) = g(20 -30X1) = y hat
if x1 = 0; h(x) = g(20) = y hat = 1
if x1 = 1; h(x) = g(20-30) = g(-10) = y hat = 0



Assume: theta 0 = -40
theta 1 = 30
theta 2 = 30

H(x) = g(-40 + 30 X1 + 30 X2) = Y hat

$$H(x) = g(-40 + 30\ X1 + 30\ X2)$$

$H(x) = g(-40 + 30\ X1 + 30\ X2)$

X1	X2	H(x)/ Y hat
0	0	$g(-40) \sim 0$
0	1	$g(-10) \sim 0$
1	0	$g(-10) \sim 0$
1	1	$g(20) \sim 1$

$H(x) = g(-40 + 30\ X1 + 30\ X2)$

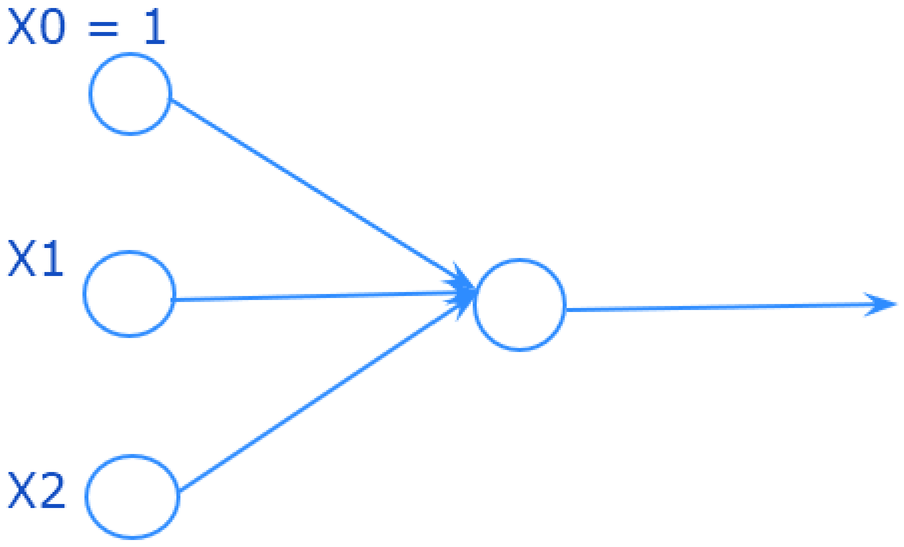
Logic Gate (AND)

X1	X2	H(x)/ Y hat
0	0	$g(-40) \sim 0$
0	1	$g(-10) \sim 0$
1	0	$g(-10) \sim 0$
1	1	$g(20) \sim 1$

$$H(x) = g(-40 + 30\,X1 + 30\,X2)$$

Logic Gate (AND)

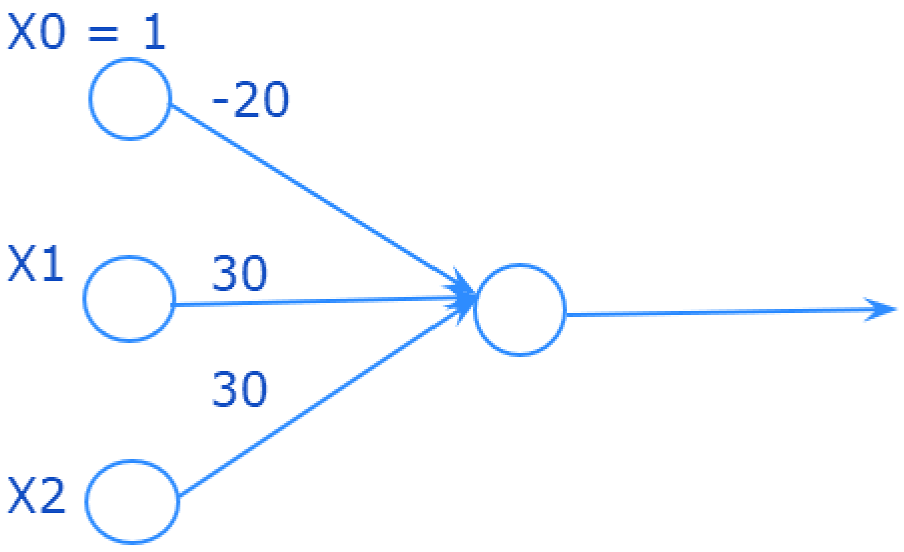
X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1



$H(x) = g(-40 + 30\ X1 + 30\ X2)$

Logic Gate (AND)

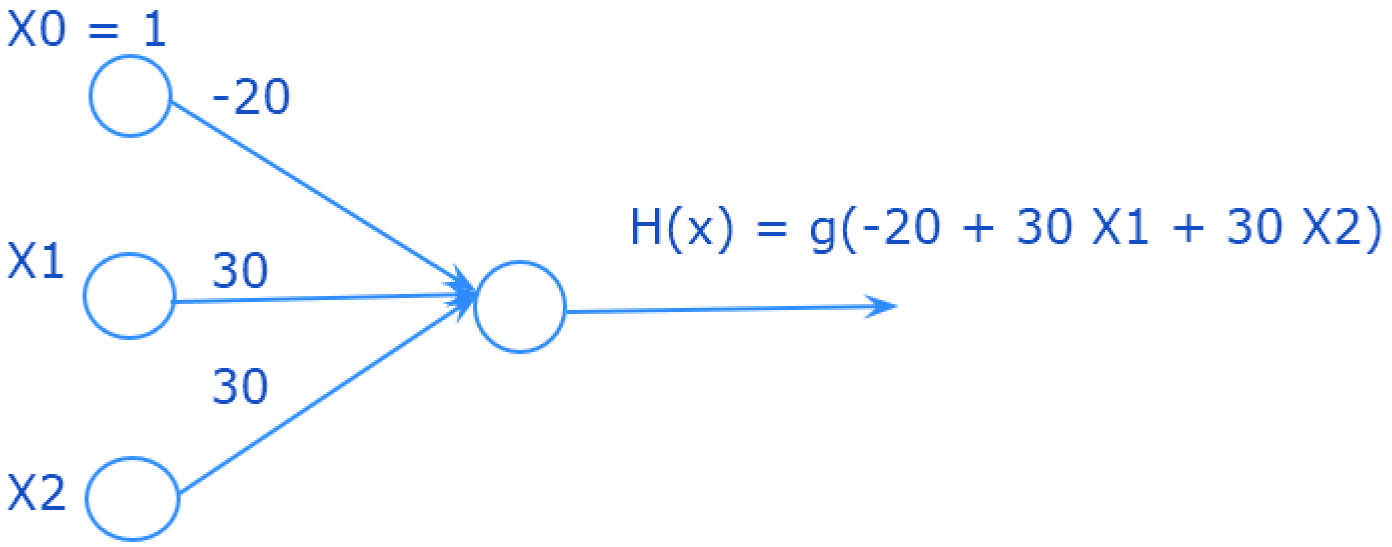
X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1



$$H(x) = g(-40 + 30\,X1 + 30\,X2)$$

Logic Gate (AND)

X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1

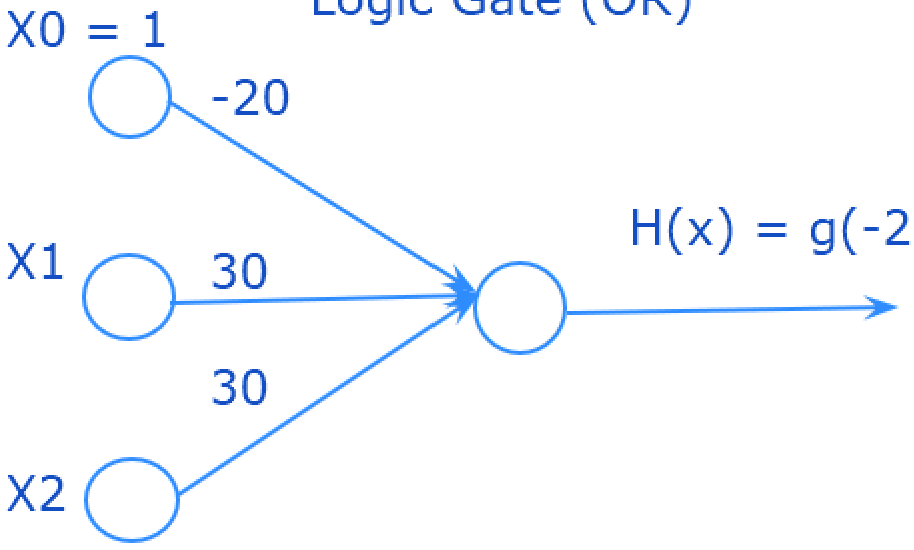


$H(x) = g(-40 + 30\,X1 + 30\,X2)$

Logic Gate (AND)

X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1

Logic Gate (OR)



$H(x) = g(-20 + 30\,X1 + 30\,X2)$

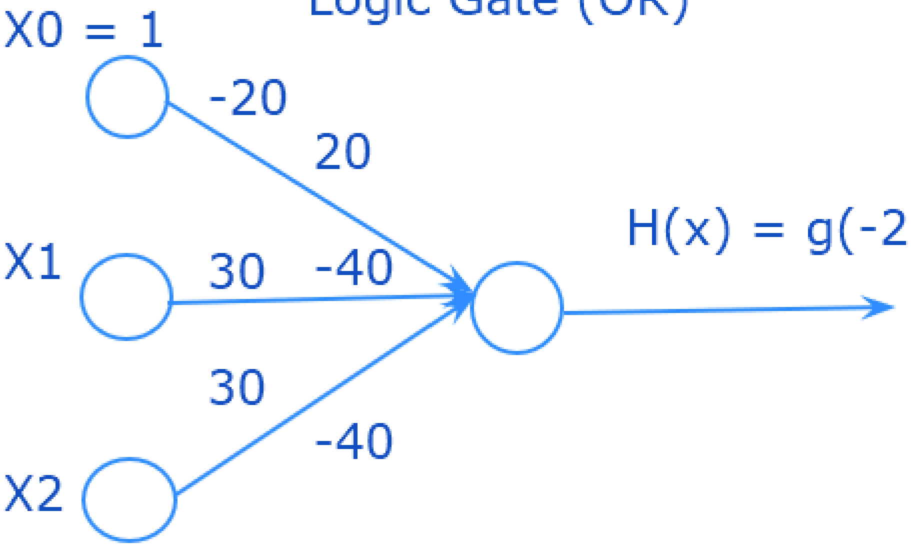
X1	X2	H(x)/ Y hat
0	0	g(-20)~0
0	1	g(10) ~1
1	0	g(10) ~1
1	1	g(40) ~1

$H(x) = g(-40 + 30\ X1 + 30\ X2)$

Logic Gate (AND)

X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1

Logic Gate (OR)



$H(x) = g(-20 + 30\ X1 + 30\ X2)$

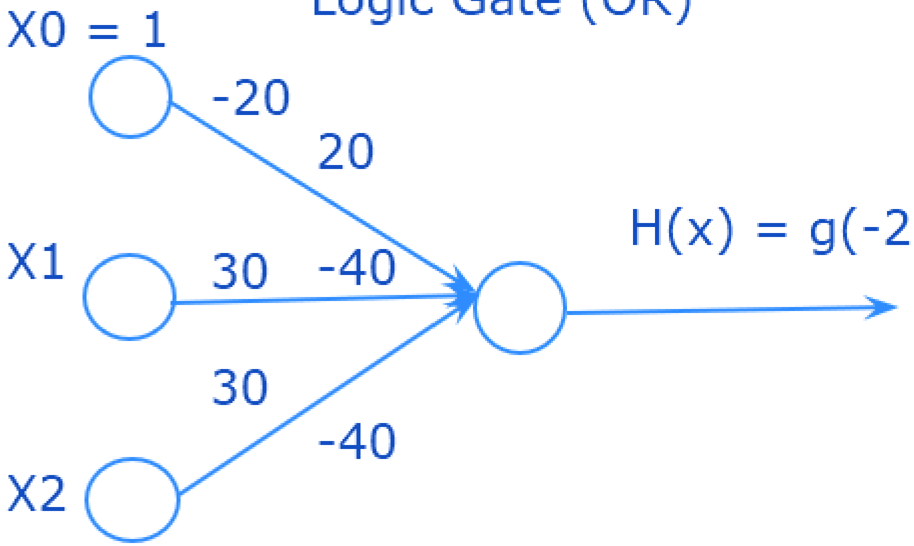
X1	X2	H(x)/ Y hat
0	0	g(-20)~0
0	1	g(10) ~1
1	0	g(10) ~1
1	1	g(40) ~1

$$H(x) = g(-40 + 30\,X1 + 30\,X2)$$

Logic Gate (AND)

X1	X2	H(x)/ Y hat
0	0	g(-40)~0
0	1	g(-10)~0
1	0	g(-10)~0
1	1	g(20) ~1

Logic Gate (OR)



$$H(x) = g(-20 + 30\,X1 + 30\,X2)$$

X1	X2	H(x)
0	0	g(20)~1
0	1	g(-20)~0
1	0	g(-20)~0
1	1	g(-60)~0

X1	X2	H(x)/ Y hat
0	0	g(-20)~0
0	1	g(10) ~1
1	0	g(10) ~1
1	1	g(40) ~1

Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 -30X2)$

Theta 0 = 40
theta 1 = -30
Theta 2 = -30

H(X) = g(40 - 30X1 -30X2)

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Theta 0 = 40
theta 1 = -30
Theta 2 = -30

H(X) = g(40 - 30X1 -30X2)

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Theta 0 = 40
theta 1 = -30
Theta 2 = -30

H(X) = g(40 - 30X1 -30X2)

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR

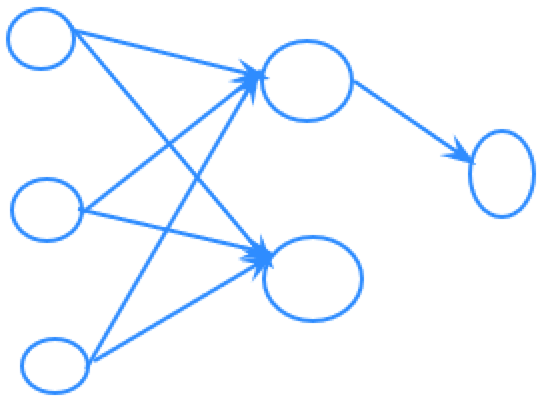
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 - 30X2)$

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR



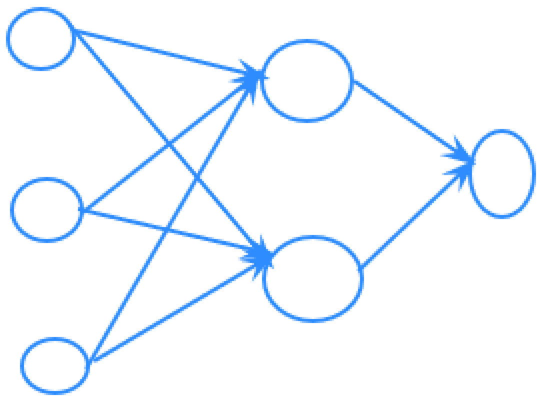
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 - 30X2)$

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR



Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 - 30X2)$

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

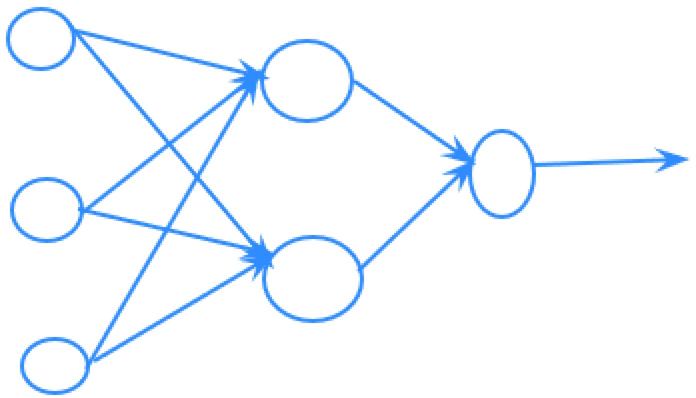
Logic gate: (NAND)

Logic gate: XNOR

X0 = 1

X1

X2



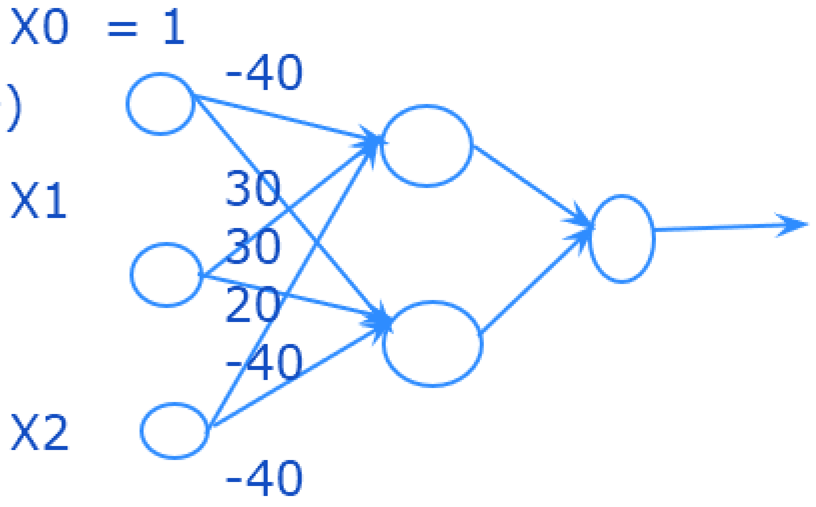
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 - 30X2)$

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR



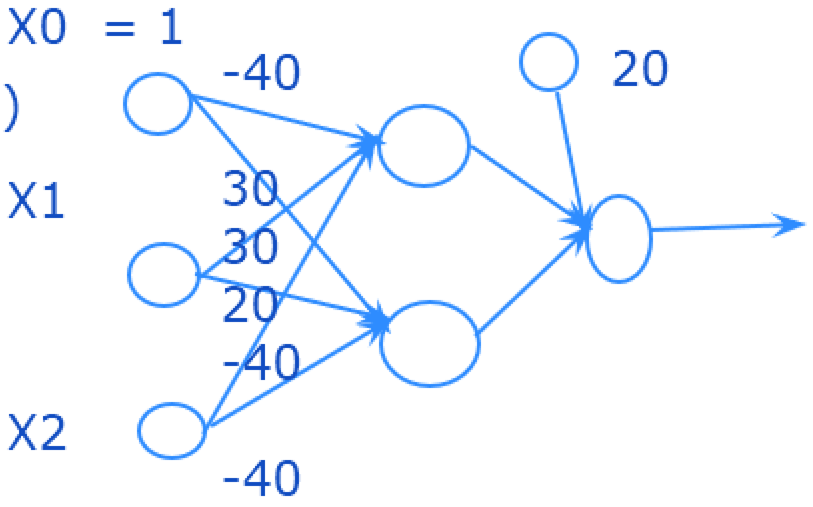
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

H(X) = g(40 - 30X1 -30X2)

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR



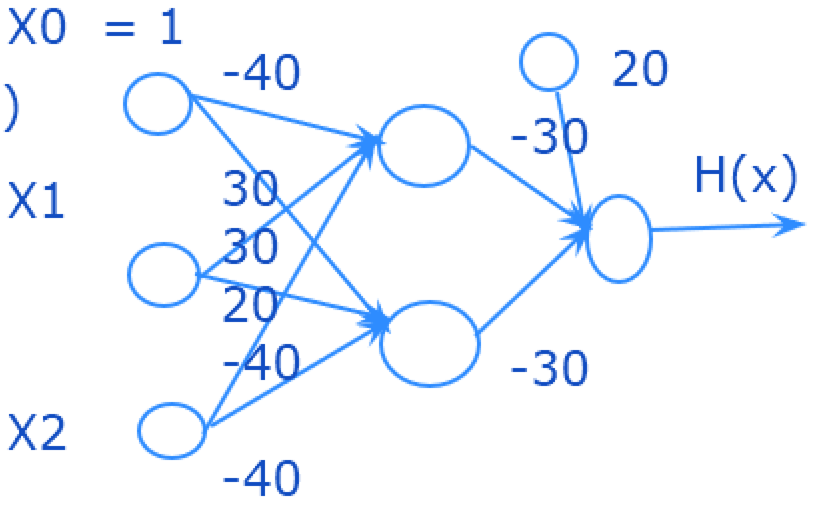
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

$H(X) = g(40 - 30X1 - 30X2)$

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

Logic gate: (NAND)

Logic gate: XNOR



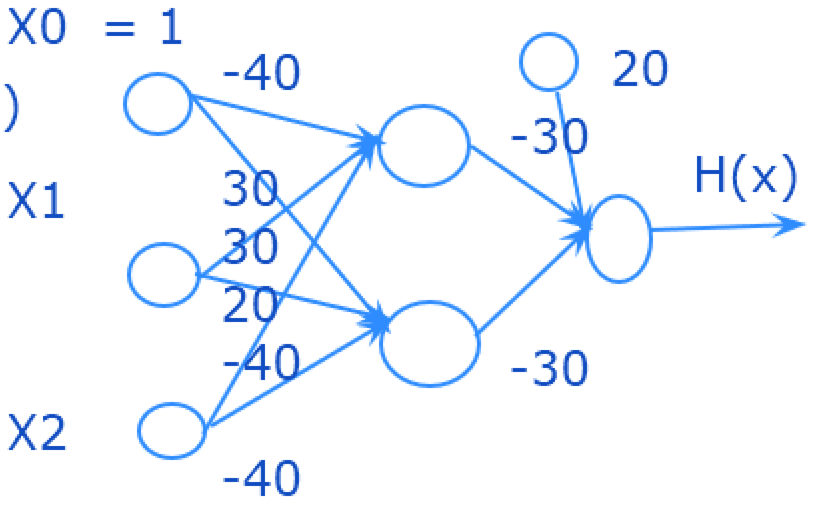
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

H(X) = g(40 - 30X1 -30X2)

Logic gate: (NAND)

Logic gate: XNOR



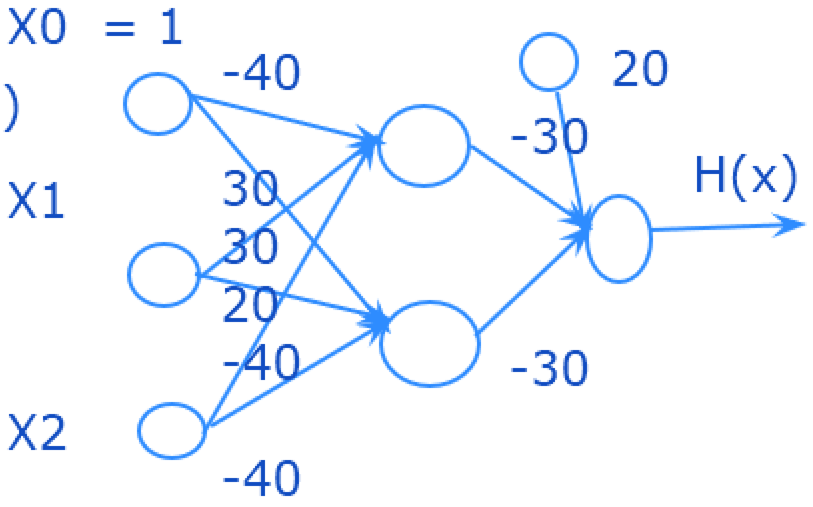
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

$H(X) = g(40 - 30X1 -30X2)$

Logic gate: (NAND)

Logic gate: XNOR



X1	X2	a1	a2	h(X)
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

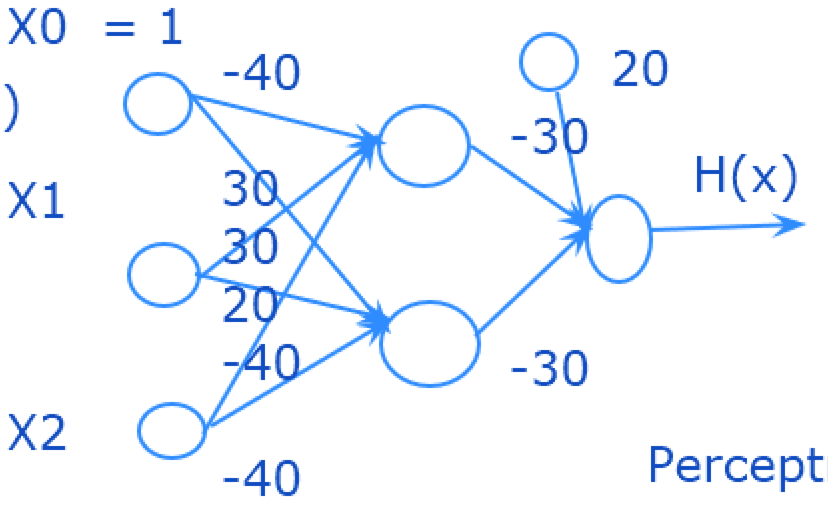
Theta 0 = 40
theta 1 = -30
Theta 2 = -30

X1	X2	H(x)
0	0	g(40)~1
0	1	g(10)~1
1	0	g(10)~1
1	1	g(-20)~0

$H(X) = g(40 - 30X1 - 30X2)$

Logic gate: (NAND)

Logic gate: XNOR



Perceptron

X1	X2	a1	a2	h(X)
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

Logic Gates

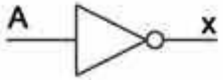
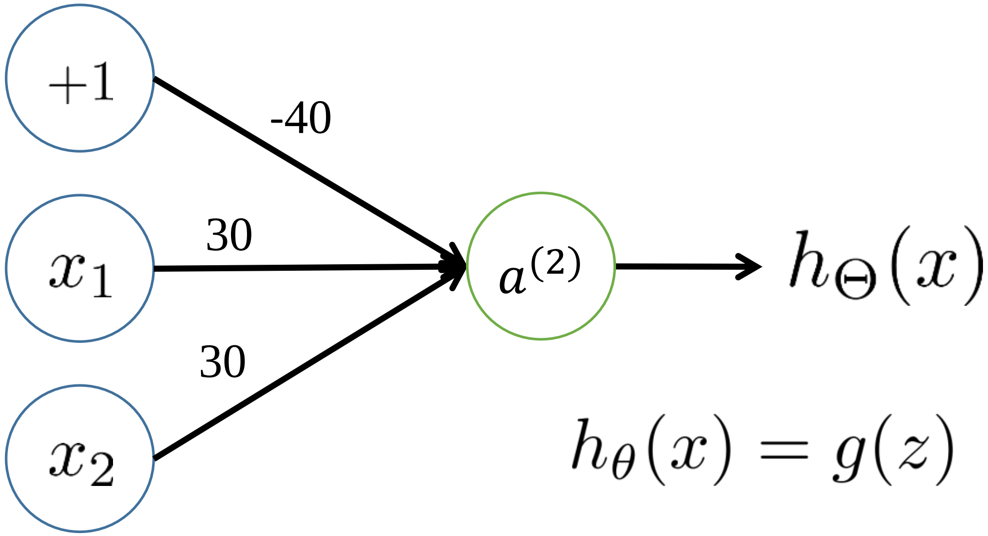
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Image Source: https://medium.com/autonomous-agents/how-to-teach-logic-to-your-neuralnetworks-116215c71a49

Implementation of Logic Gates with Perceptron

Example: Logic Gate (AND)

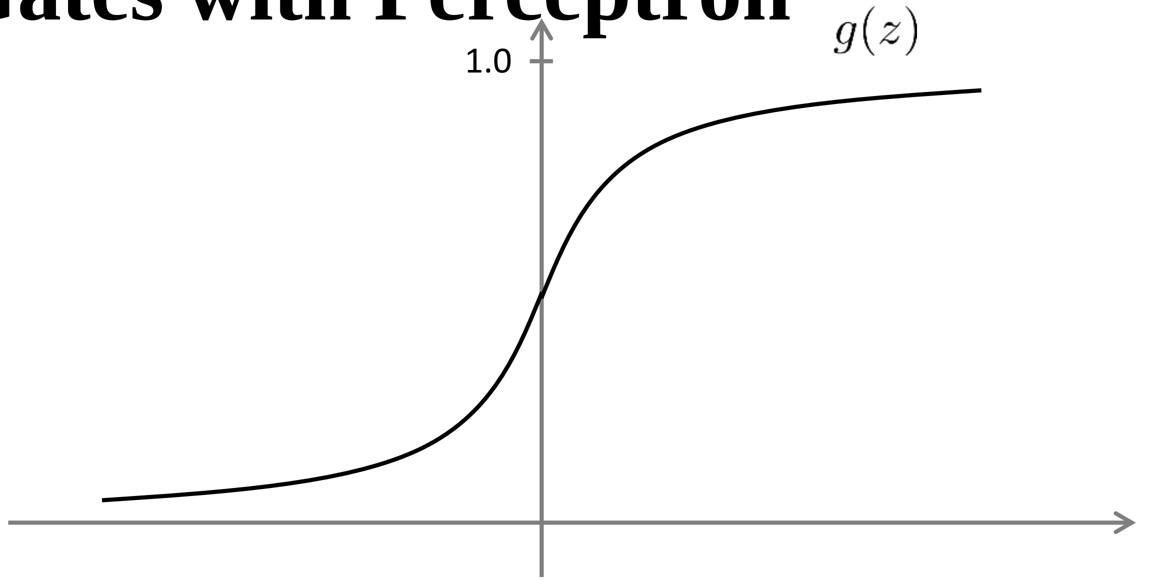
$$x_1, x_2 \in \{0, 1\}$$
$$y = x_1 \text{ AND } x_2$$



$$h_\theta(x) = g(z)$$

= g (-40 + 30 x1 + 30 x2)

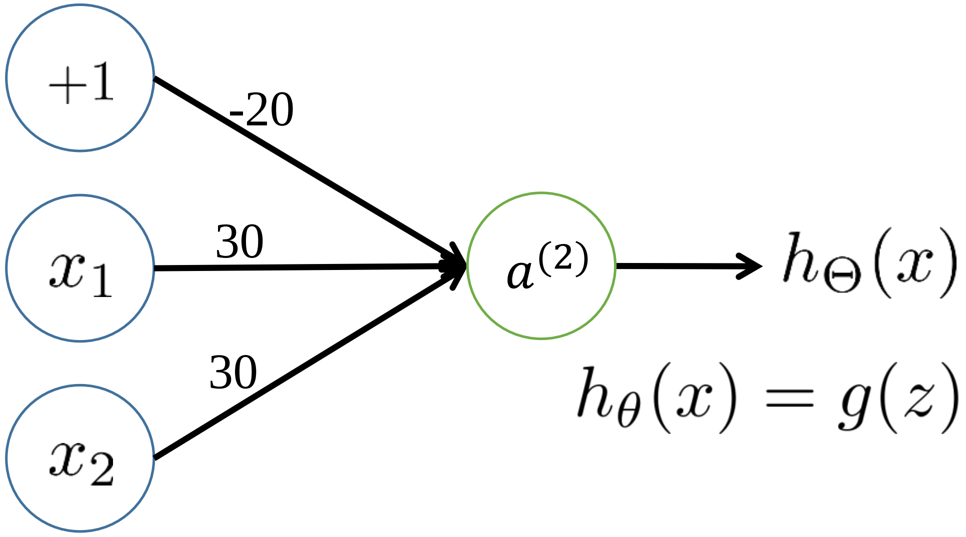
h (x) = x1 AND x2



x_1	x_2	$h_\Theta(x)$
0	0	g (-40) ~ 0
0	1	g (-10) ~ 0
1	0	g (-10) ~ 0
1	1	g (20) ~ 1

Implementation of Logic Gates with Perceptron

Example: Logic Gate (OR)



$$h_\theta(x) = g(z)$$

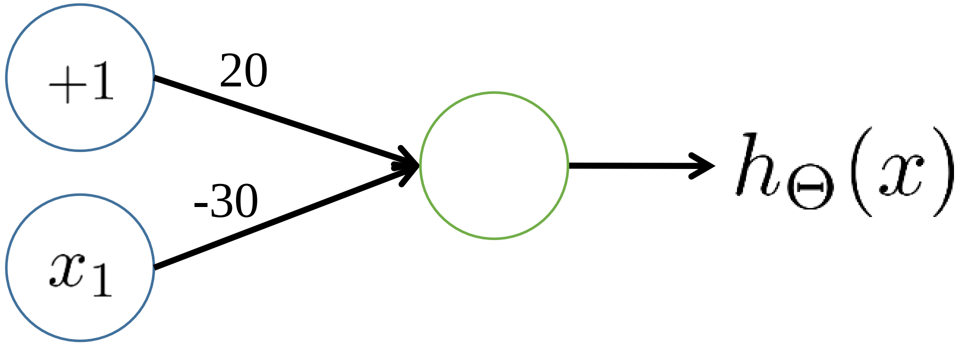
= g (-20 + 30 x1 + 30 x2)

h (x) = x1 OR x2

x_1	x_2	$h_\Theta(x)$
0	0	g (-20) ~ 0
0	1	g (10) ~ 1
1	0	g (10) ~ 1
1	1	g (40) ~ 1

Implementation of Logic Gates with Perceptron

Example: Logic Gate (NOT)



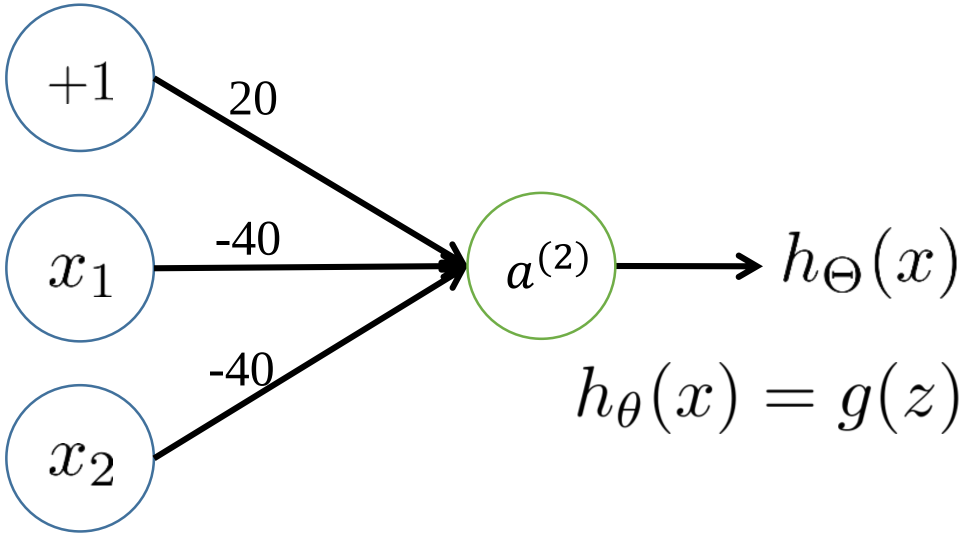
$$h_\theta(x) = g(z)$$

= g (20 - 30 x1)

x_1	$h_\Theta(x)$
0	g (20) ~ 1
1	g (-10) ~ 0

Implementation of Logic Gates with Perceptron

Example: Logic Gate (NOR)



$$h_\theta(x) = g(z)$$

= g (20 - 40 x1 - 40 x2)

h (x) = x1 NOR x2

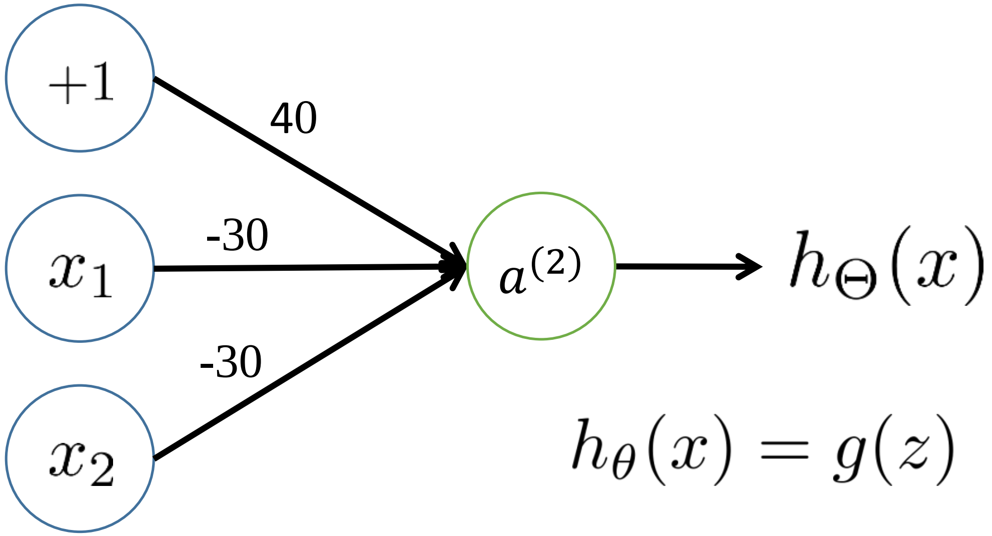
$$(\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$$

x_1	x_2	$h_\Theta(x)$
0	0	g (20) ~ 1
0	1	g (-20) ~ 0
1	0	g (-20) ~ 0
1	1	g (-60) ~ 0

Implementation of Logic Gates with Perceptron

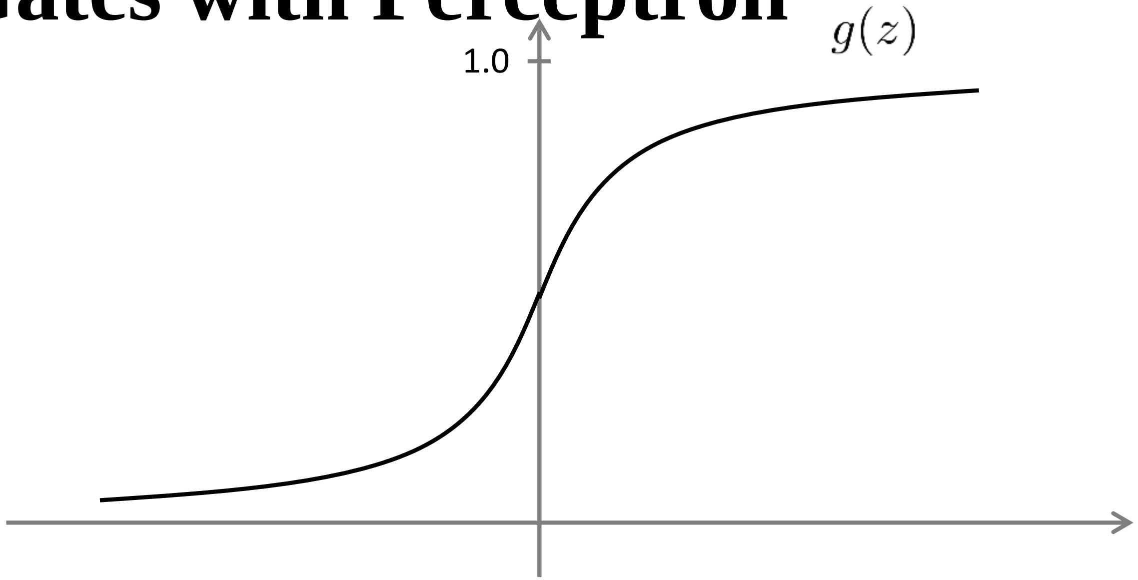
Example: Logic Gate (NAND)

$$x_1, x_2 \in \{0, 1\}$$

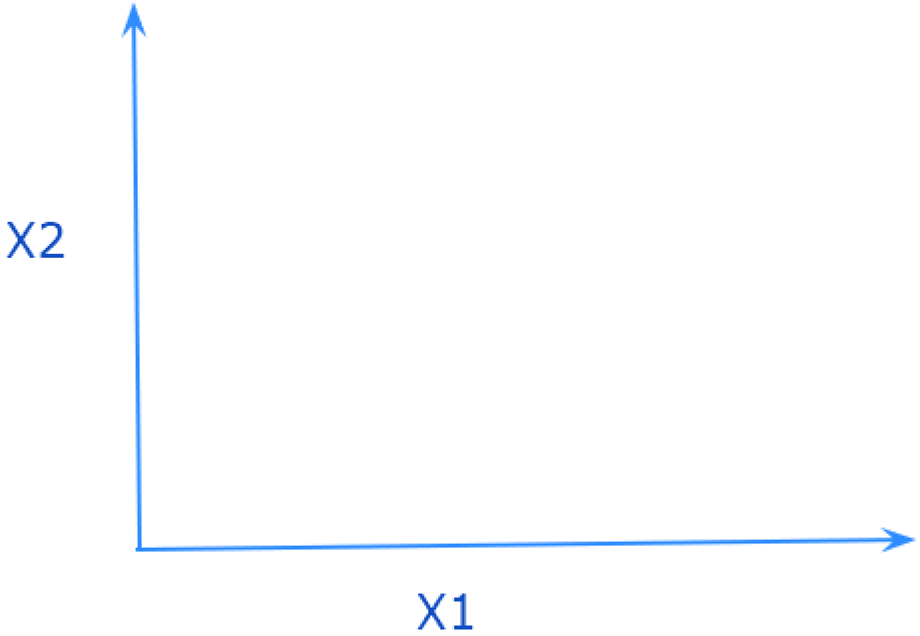


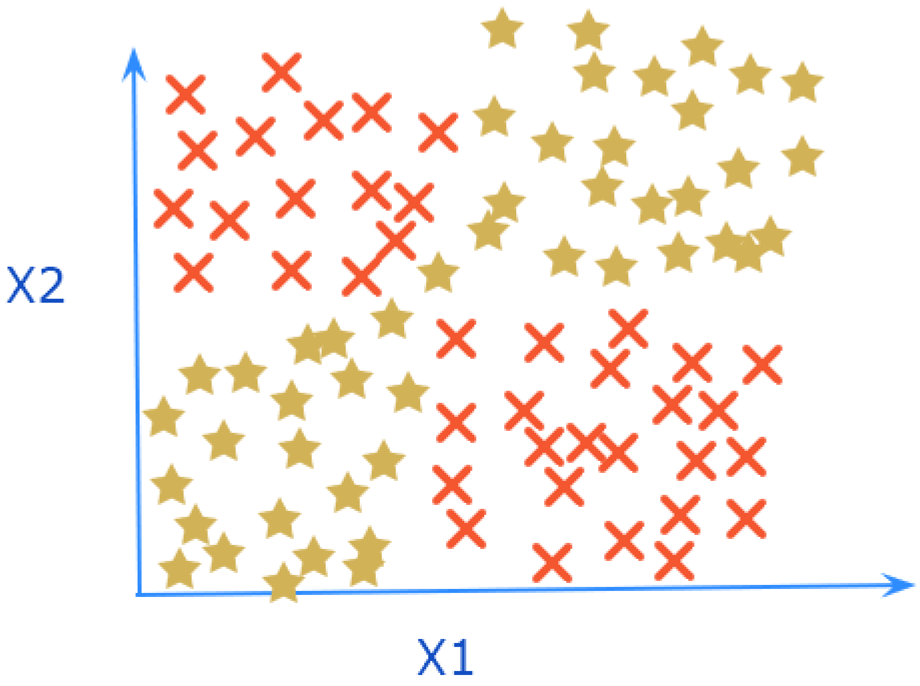
$$h_\theta(x) = g(z)$$

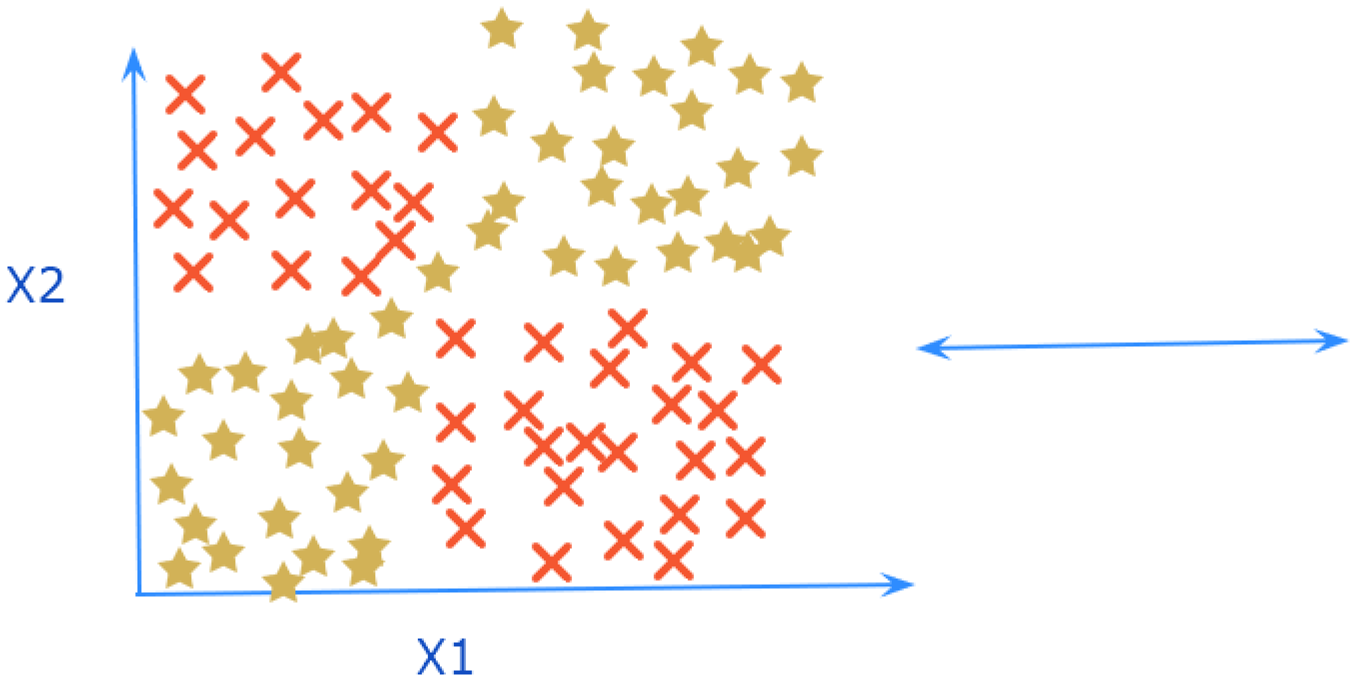
= g (40 - 30 x1 - 30 x2)



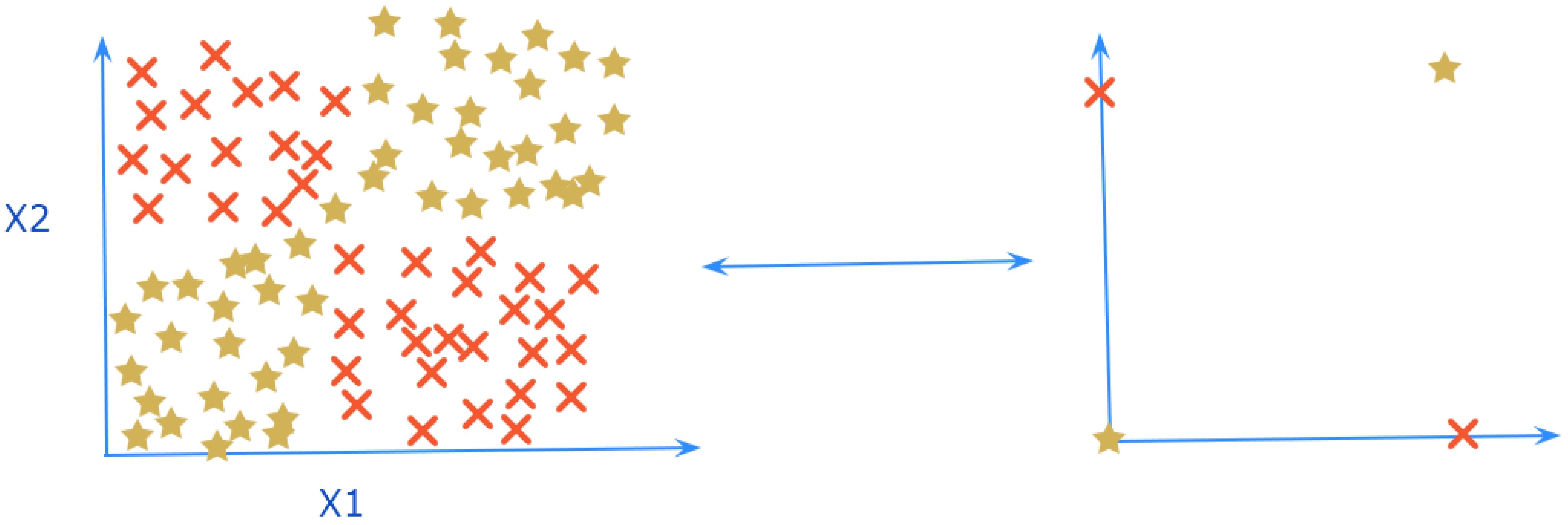
x_1	x_2	$h_\Theta(x)$
0	0	g (40) ~ 1
0	1	g (10) ~ 1
1	0	g (10) ~ 1
1	1	g (-20) ~ 0

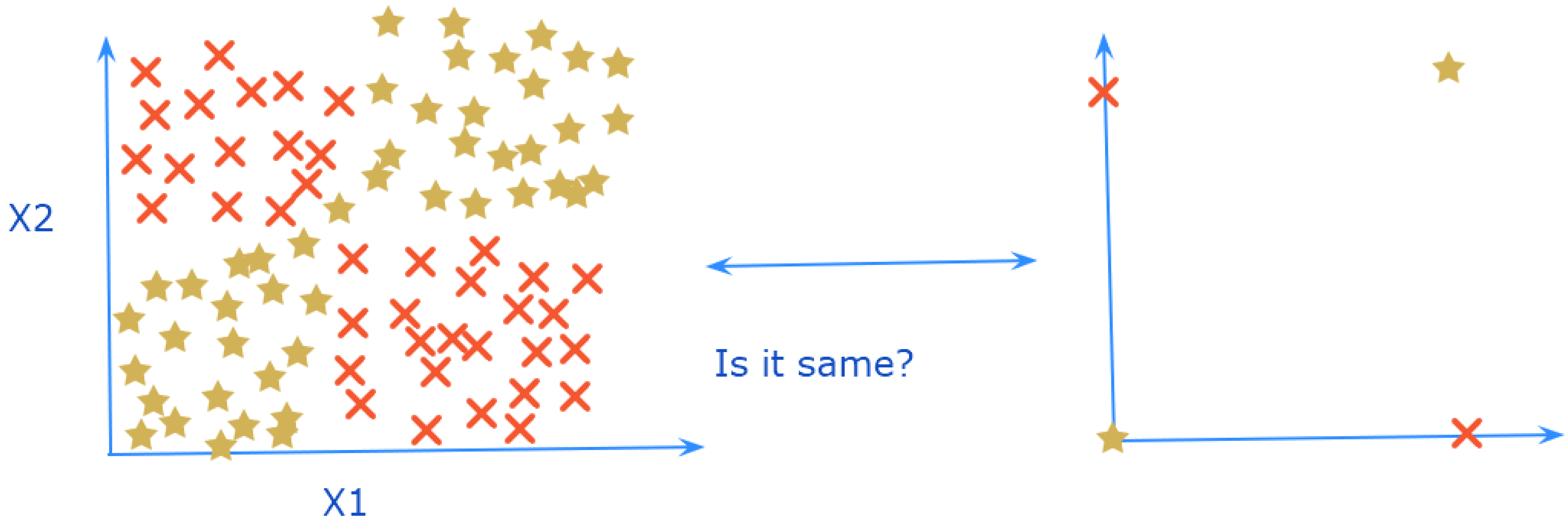

X2
X1

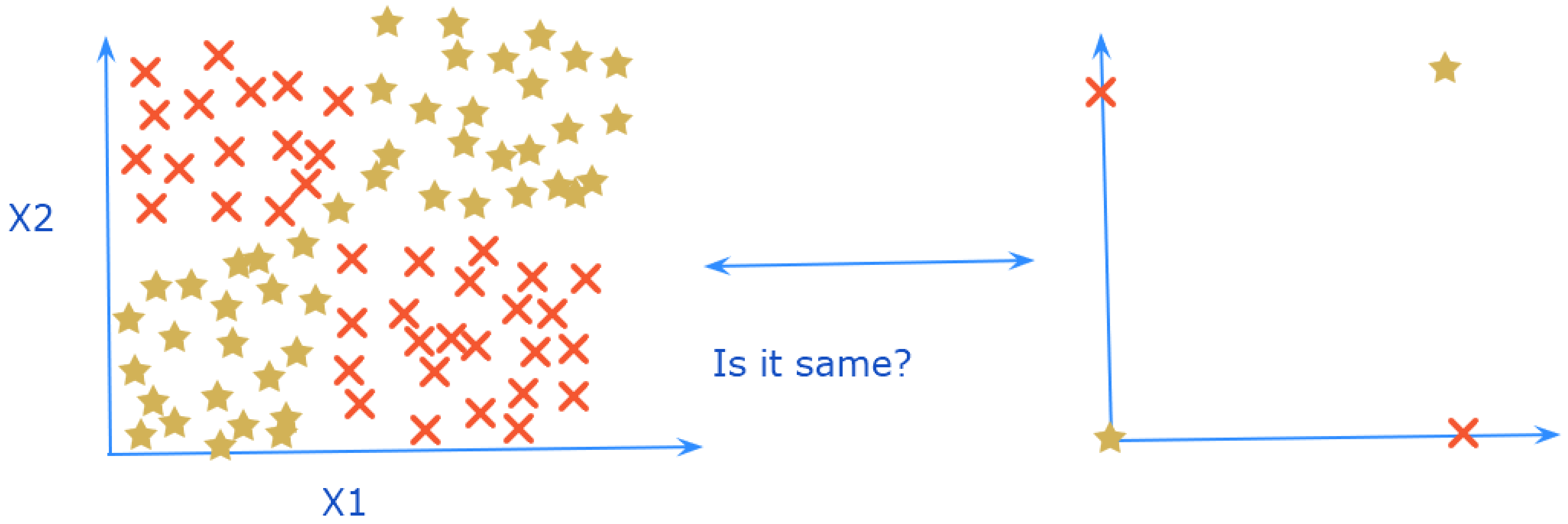

X2
X1



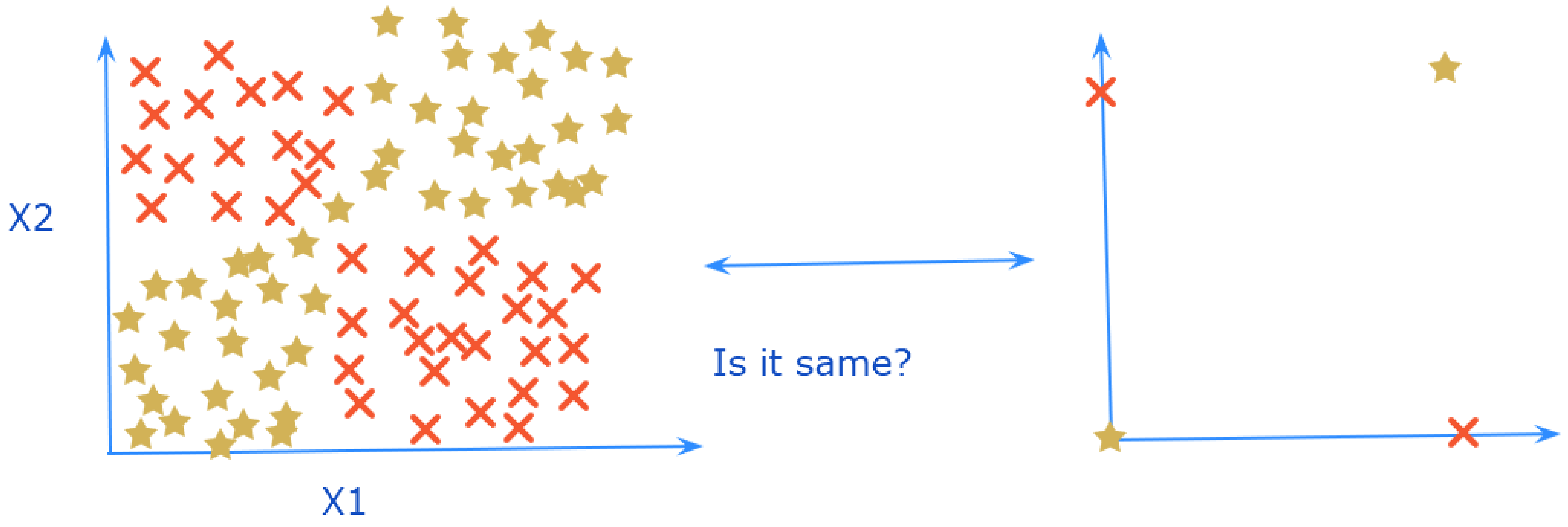
X2
X1


X2
X1

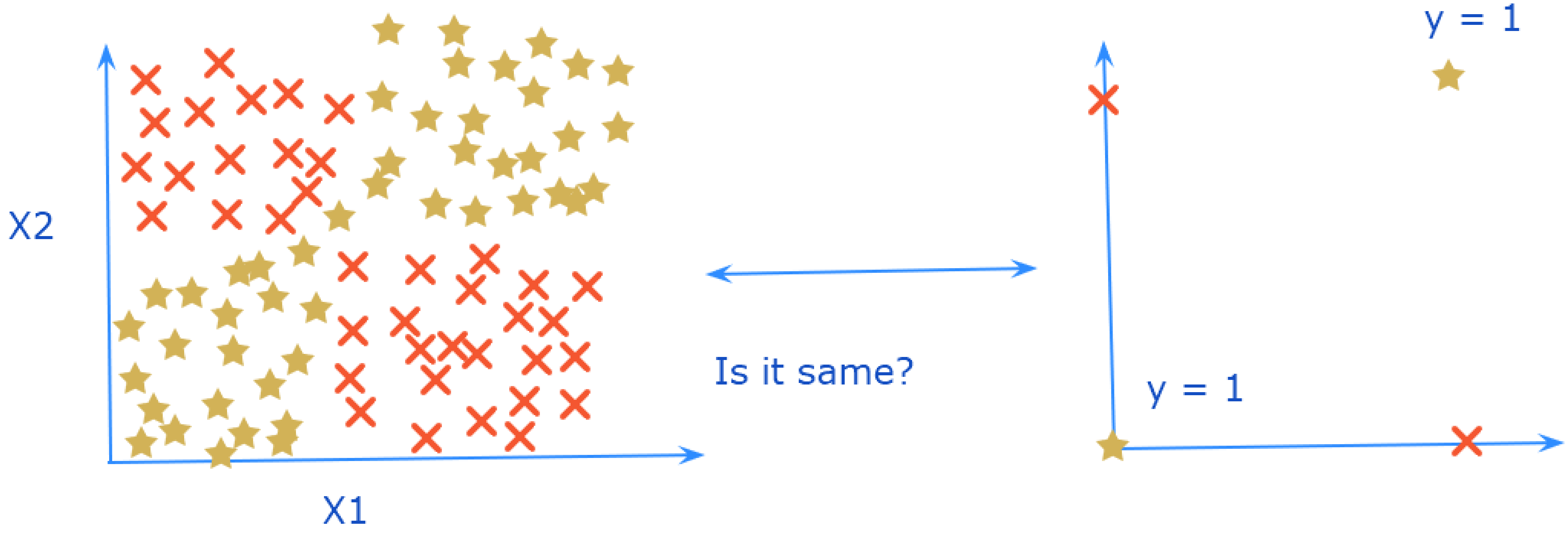

X2
X1
Is it same?



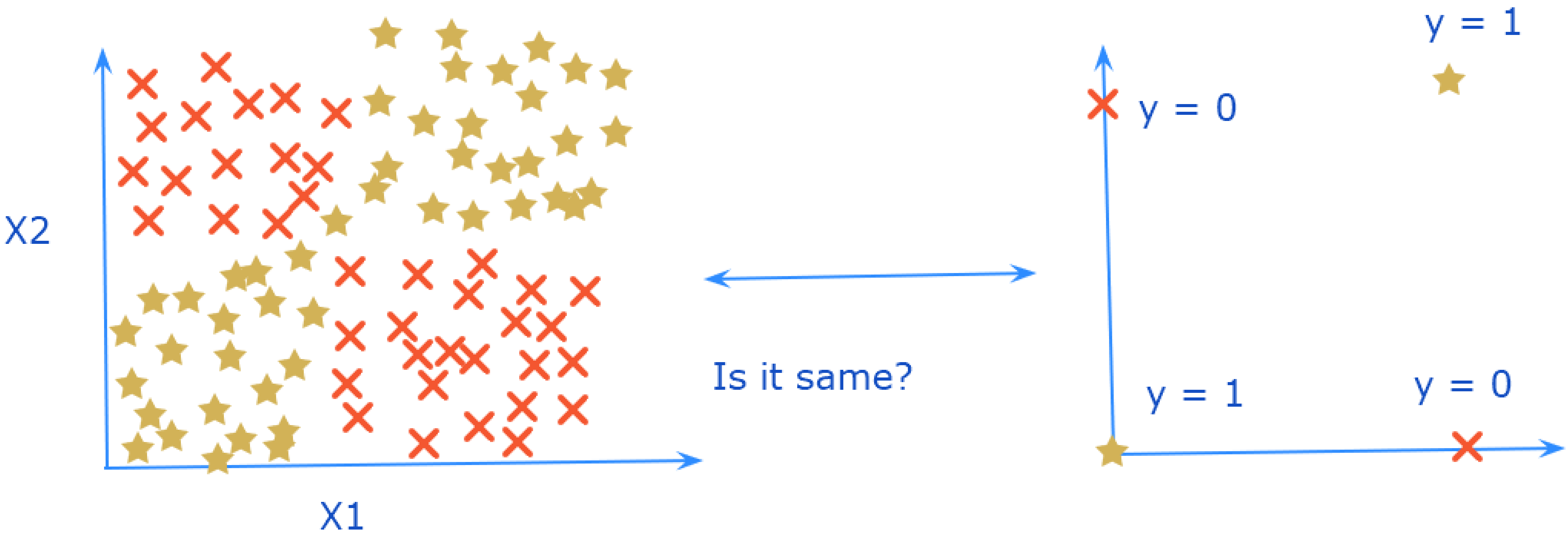
X2	X1	Output
0	0	1
0	1	0
1	0	0
1	1	1



X2	X1	Output
0	0	1
0	1	0
1	0	0
1	1	1



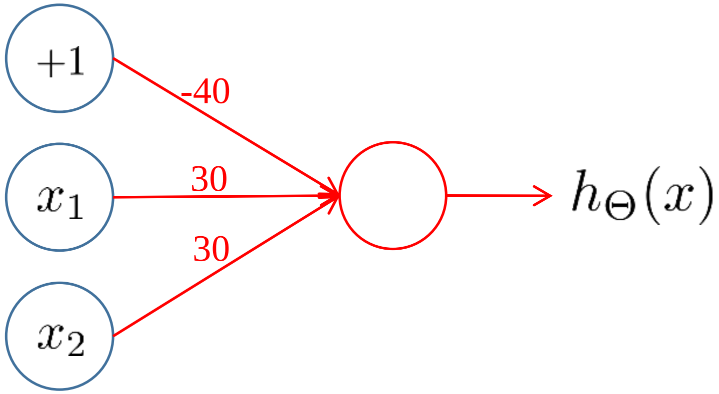
X2	X1	Output
0	0	1
0	1	0
1	0	0
1	1	1



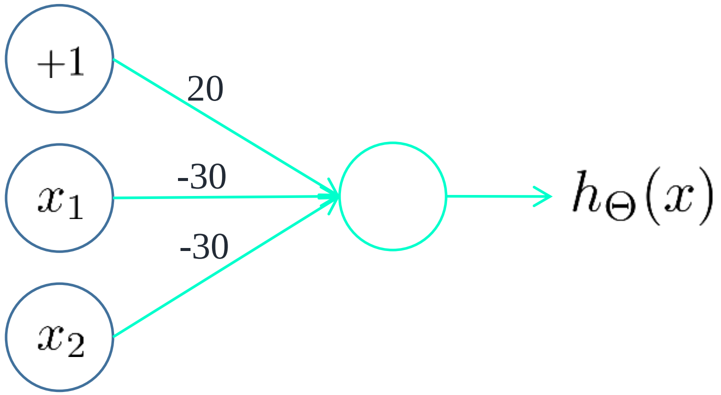
X2	X1	Output
0	0	1
0	1	0
1	0	0
1	1	1

Implementation of Logic Gates with Perceptron

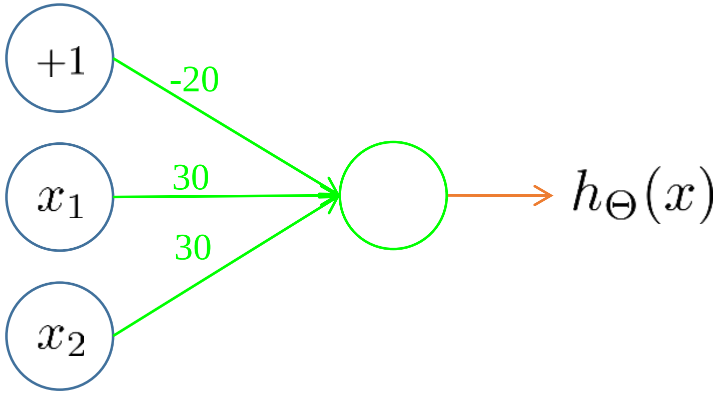
Example: Logic Gate (XNOR)



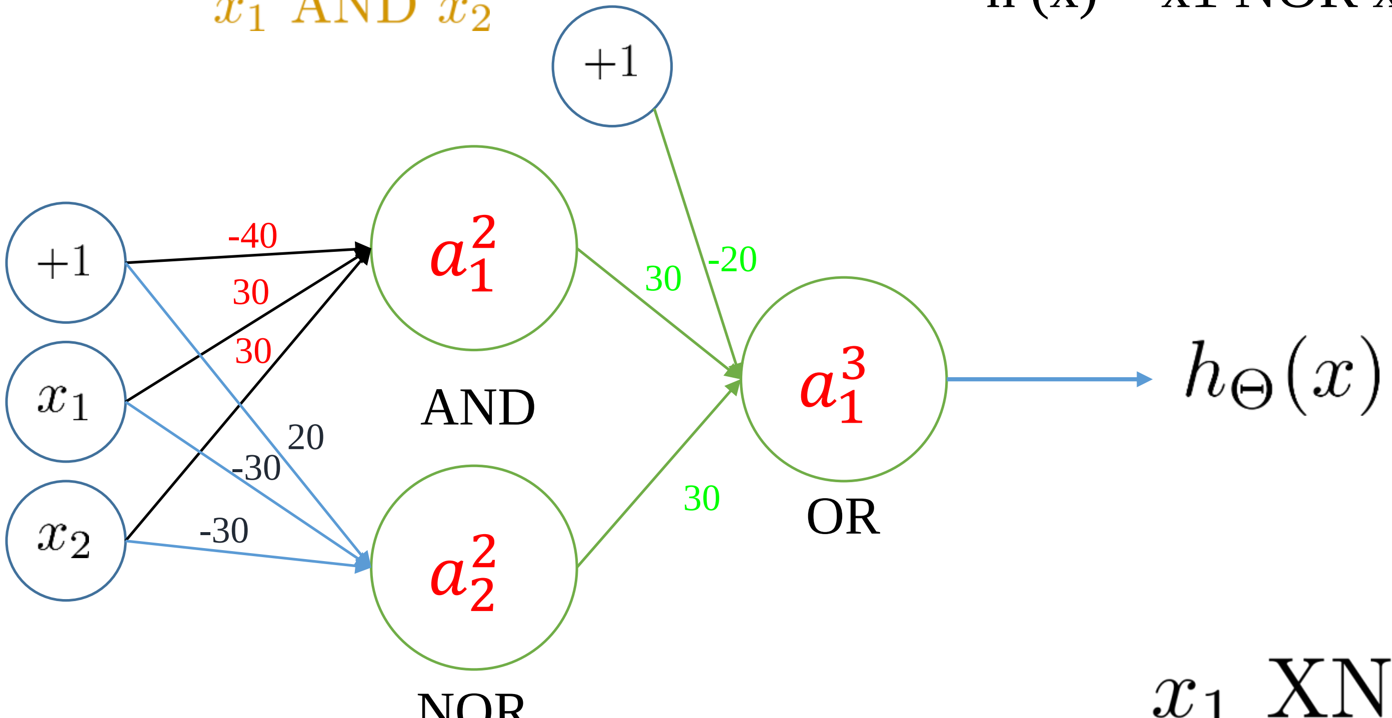
x_1 AND x_2



h (x) = x1 NOR x2



x_1 OR x_2

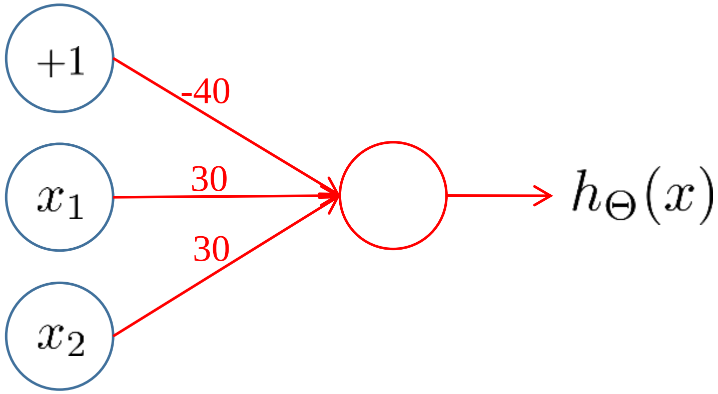


x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

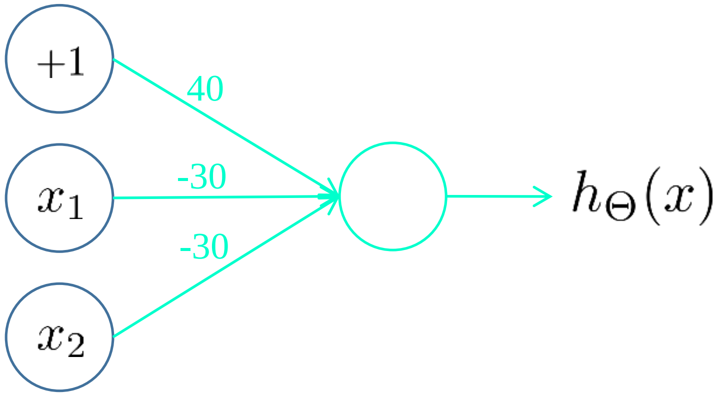
x_1 XNOR x_2

Implementation of Logic Gates with Perceptron

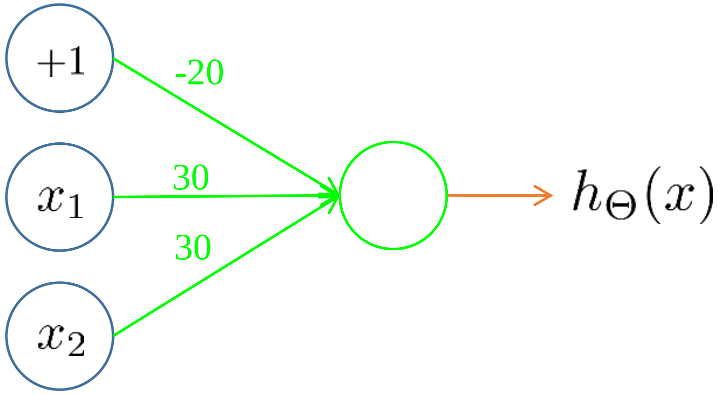
Example: Logic Gate (XOR)



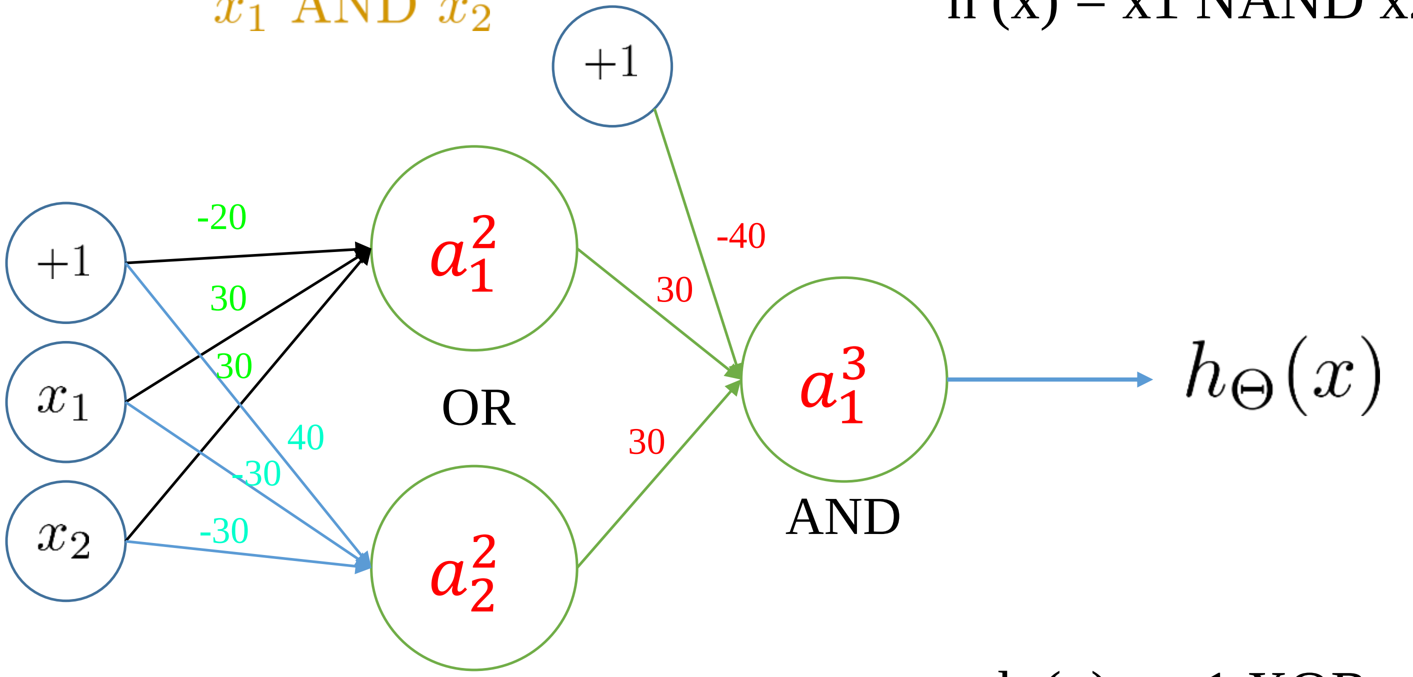
x_1 AND x_2



h (x) = x1 NAND x2



x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

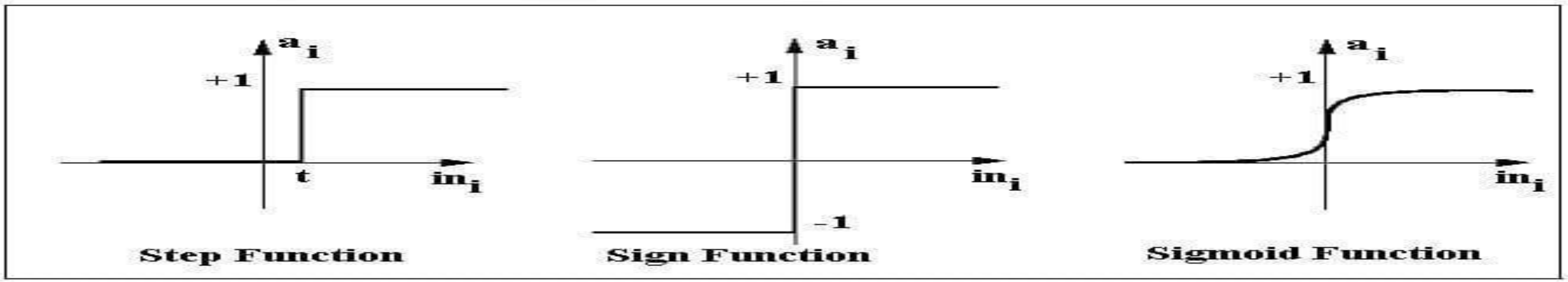
h (x) = x1 XOR x2

Applications of Neural Networks

- **Can** we **use** a **neural network** to solve a **regression problem**?
  - **Yes**, in case of more **complex model**, applying a neural network to the **problem can provide better prediction power compared to a traditional regression** technique.

- Handwriting Recognition
- Financial Forecasting
- Targeted Marketing
- Portfolio Management
- Target Recognition

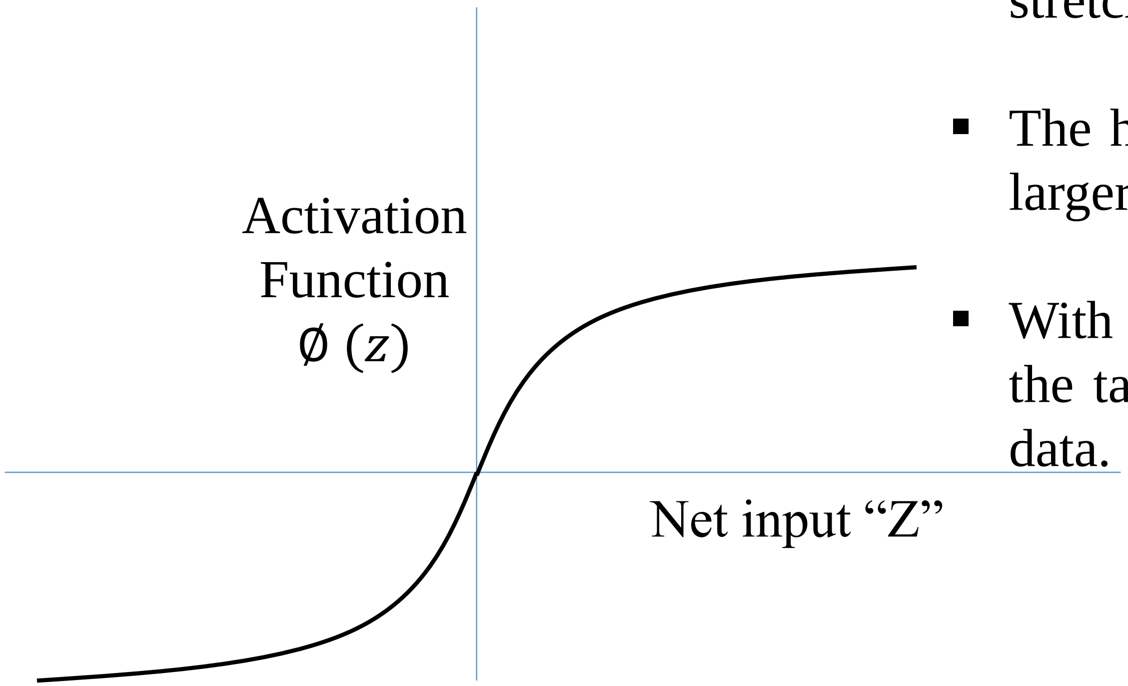
Activation Functions of Perceptron (Artificial Neuron)



- For activation (Step function): the artificial neuron gets triggered above a certain value of the neuron output; else it outputs zero.
- For activation (Sign Function): the artificial neuron gets outputs +1 or -1 depending on whether neuron output is greater than zero or not.
- For activation (Sigmoid Function): the artificial neuron gets the output values between 0 and 1. The logistic sigmoid function leads to a probability of the value between 0 and 1. It is useful when one is interested in probability mapping rather than precise values of input parameter.
- For highly negative input, the sigmoid function output is close to zero which might create problem in neural network training and can lead to slow learning.
- In such cases, hyperbolic tangent function is more preferable as an activation function in hidden layers of a neural network.

Source: https://www.simplilearn.com/what-is-perceptron-tutorial

Hyperbolic Tangent Function

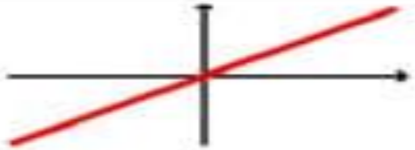
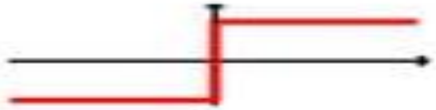


- It provides output between -1 and +1. This is an extension of logistic sigmoid where the output stretches between -1 and +1.

- The hyperbolic tangent (tanh) function has two times larger output space than the logistic function.

- With larger output space and symmetry around zero, the tanh function leads to the more even handling of data.

- The advantage of the hyperbolic tangent over the logistic function is that it has a broader output spectrum and ranges in the open interval (-1, 1), which can improve the convergence.

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Source: https://www.simplilearn.com/what-is-perceptron-tutorial

Why an Activation Function is needed in ANN?

- An **activation function** is added into an artificial neural network to help the **network to learn complex patterns in the data**.
- It helps in deciding at the end **whether neuron will be fired or not**.
- Basically, helps in **keeping the value** of the output from the neuron **to a certain limit** according to our requirement apart from **maintaining the biological similarity**.
  - If **not restricted** to a certain limit then it **can go very high in magnitude** (especially in case of very deep neural networks that have millions of parameters) and it can lead to computational issues.
- The **most important** feature in an activation function is its ability to **add non-linearity into a neural network**.

Multiple output units: One-vs-all.

Dolphin

Sea Lion

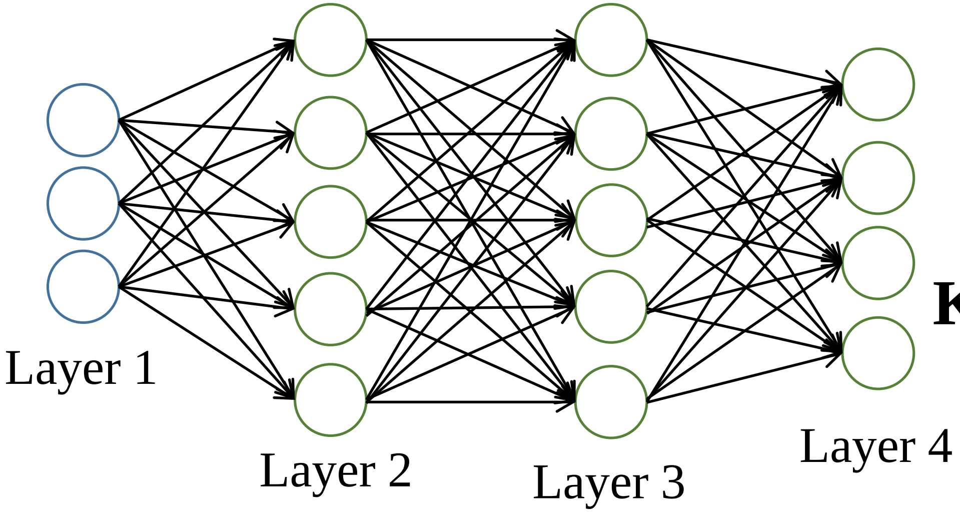
Bird

Car

<u>Binary classification</u>

$y = 0 \text{ or } 1$

1 output unit



$h_\Theta(x) \in \mathbb{R}^4$

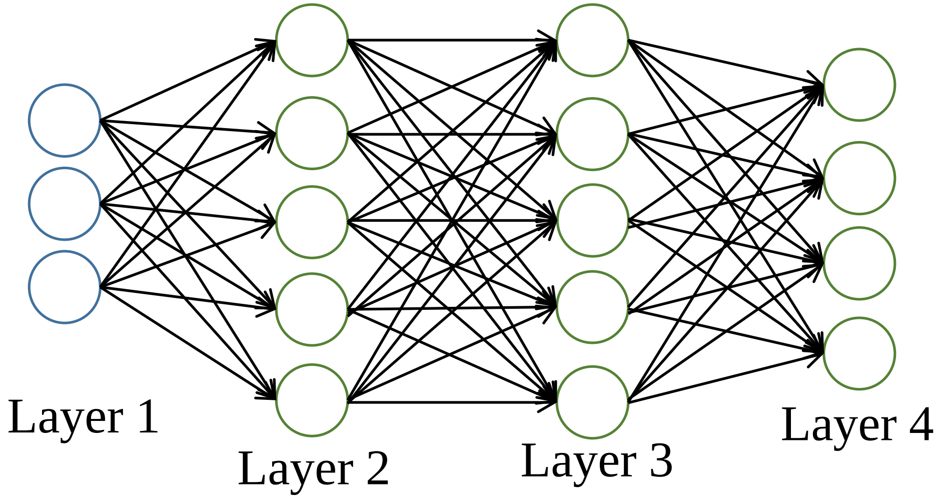
K classes, K output units;

$y \in \mathbb{R}^K$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when Dolphin when car when Bird

Multiple output units: One-vs-all (Multi-class classification).



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

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer

$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when Dolphin when car when Bird

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

$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ **K classes, K output units;** $y \in \mathbb{R}^K$

Dolphin Sea Lion Bird Car

Cost function for Neural Network

Logistic regression cost function:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural Network cost function:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Gradient Descent Algorithm

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

- $J(\Theta)$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$$

Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l .

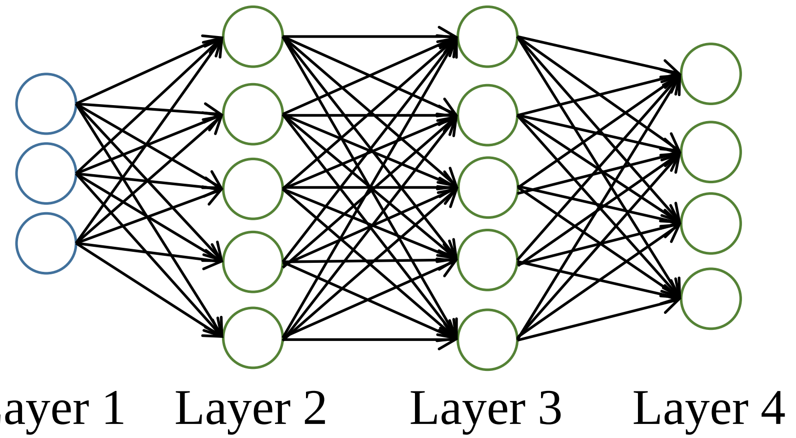
❖ **Goal is to compute partial derivatives**

For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$



- **Back-propagation** is the method of fine-tuning the parameters (thetas) of a **neural network** based on the error rate obtained in the previous epoch.
- Proper tuning of the parameters (thetas) allows you to reduce error rates and to make the model reliable by increasing its generalization

Backpropagation algorithm

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

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

- **It actually gives us detailed insights into how changing the parameters and biases changes the overall behavior of the network.**

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \qquad \text{if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Derivative

Which is better: Feed-Forward or Backpropagation algorithm in ANN?

- **Backpropagation algorithm** is used to train (adjust parameter) the neural network.
  - **Information flows from output layer to the input layer**.
- **Feed-forward** or Forward algorithm is used to calculate output vector from input vector.
  - **Information flows from input layer to the output layer**.
- For training a neural network, **we need to use both the algorithms**.
  - Basically, we will propagate in forward direction to get the output and compare it with the real value to get the error then, we will back propagate and use gradient descent to update new values of parameters. Then again we will propagate in forward direction to observe the behavior of ANN and if it is not satisfactory we will again propagate in backward direction to find the new values of parameter. The process will be repeated until we achieve the minimum value of error.
- Basic type of neural network is **multi-layer perceptron** that is Feed-forward backpropagation network.

ChatGPT – Recent AI Development

- ChatGPT is an artificial-intelligence (AI) chatbot developed by OpenAI.
- Launched in November 2022.
- It is built on top of OpenAI's GPT-3.5 and GPT-4 families of large language models (LLMs) and has been fine-tuned using both supervised and reinforcement learning techniques.

- **Limitation:** It can struggle to grasp the subtle nuances of human communication. Example: If a user were to use sarcasm or humor in their message, ChatGPT may fail to pick up on the intended meaning and instead provide a response that is inappropriate or irrelevant