CS2109S: Introduction to AI and Machine Learning

Lecture 9:
Intro to Neural Networks

25 March 2025

Outline

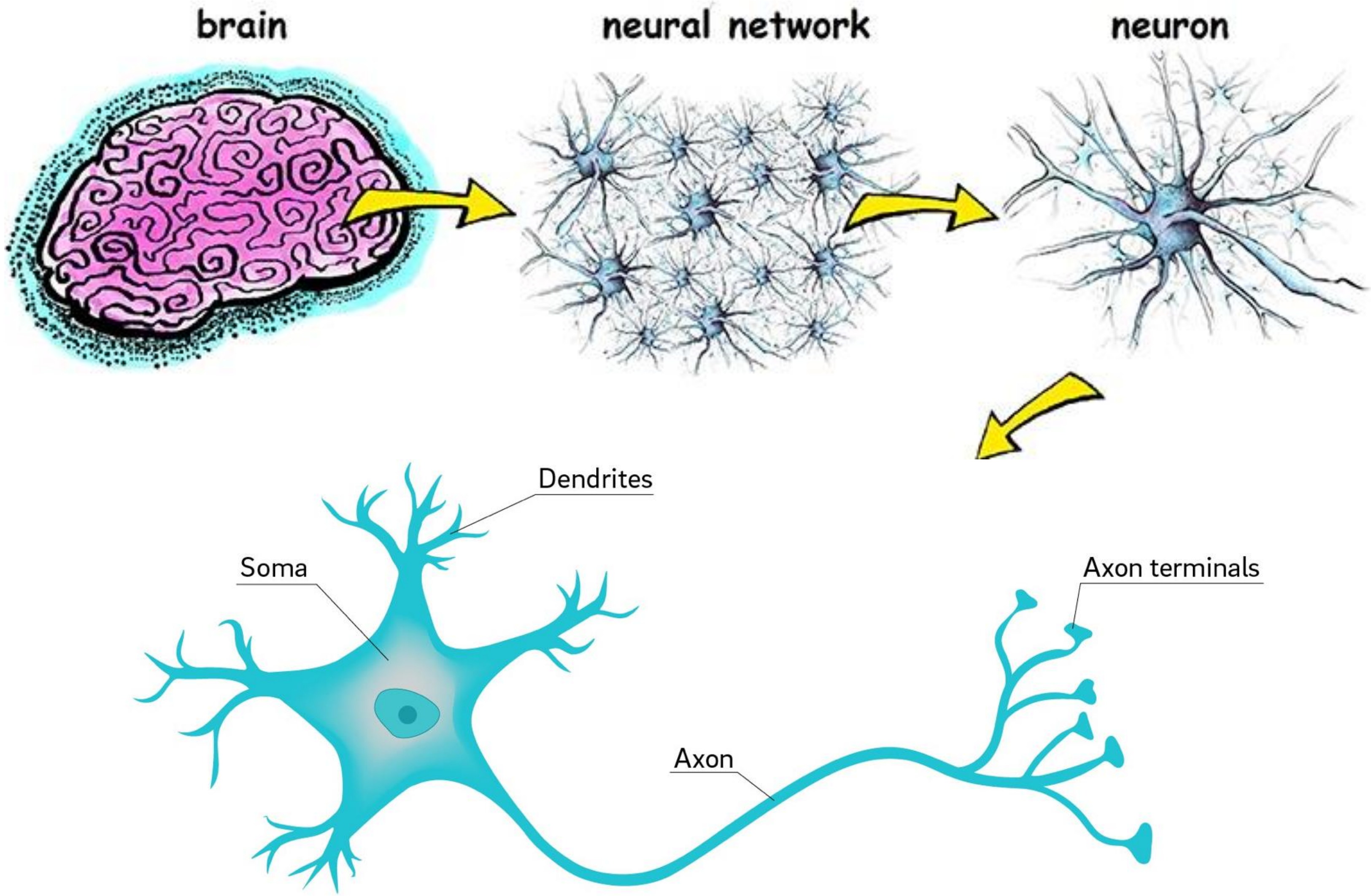
- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Network
  - Neuron
  - AND Gate Modelling
  - XNOR Gate Modelling
  - Single-layer and Multi-layer Neural Networks
- Multi-class Classification

Outline

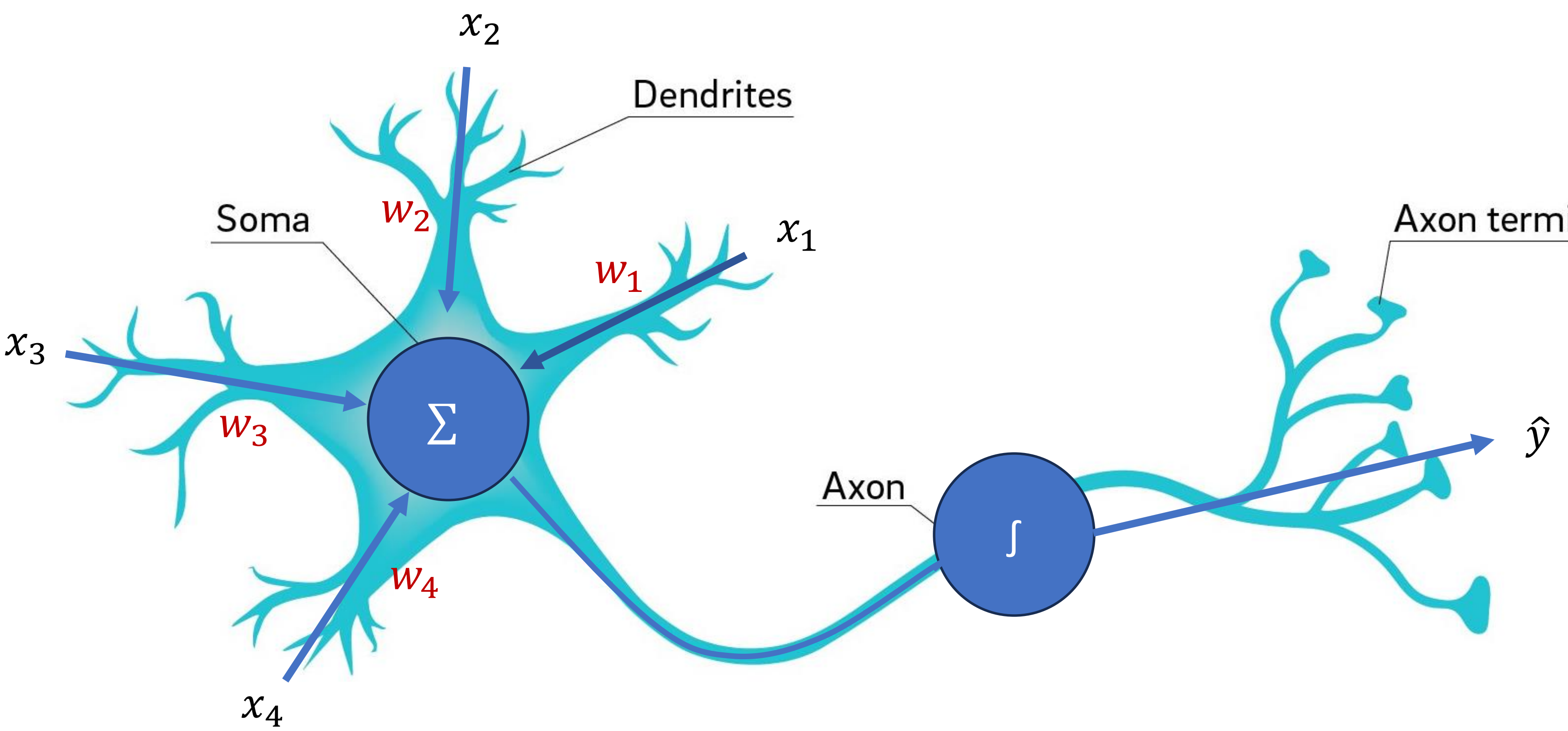
- **Perceptron**
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Network
  - Neuron
  - AND Gate Modelling
  - XNOR Gate Modelling
  - Single-layer and Multi-layer Neural Networks
- Multi-class Classification

Image credit:
https://kids.frontiersin.org/articles/10.3389/frym.2021.560631
https://www.upgrad.com/blog/biological-neural-network/

Brain and Neuron

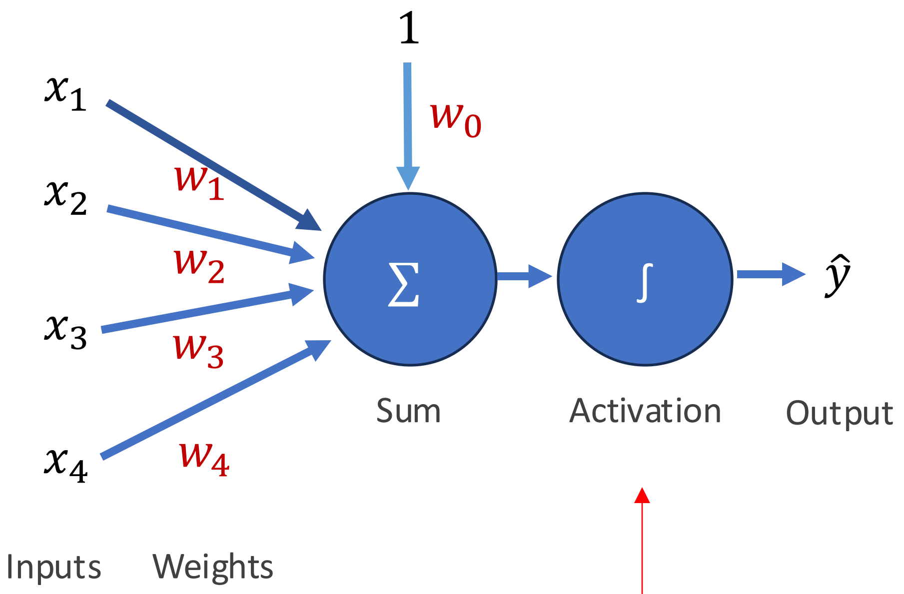


Neuron



x_2
Dendrites
Soma
w_2
x_1
w_1
Axon terminals
x_3
Σ
w_3
$\hat{y}$
Axon
$\int$
w_4
x_4

Perceptron



Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Perceptron: Data

Suppose:

- We are given N data points.
- Each data point consists of features and a target variable.
- The features are described by a vector of real numbers in dimension d.
- The target is {-1,1}, where -1 is “negative” class and 1 is “positive” class.

Perceptron: Data – Math

Suppose:

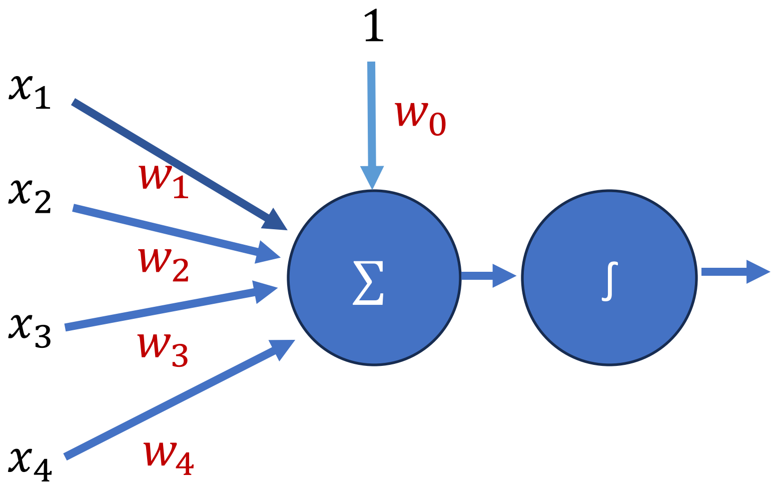
$$D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)},\ y^{(2)}), \dots, (\boldsymbol{x}^{(N)},\ y^{(N)})\},$$

where for all $i \in \{1, \dots, N\}$

Features: $\boldsymbol{x}^{(i)} \in \mathbb{R}^{d}$

Targets: $y^{(i)} \in \{-1,1\}$

Perceptron



Given an input vector $\boldsymbol{x}$ of dimension d, the Perceptron model is defined as:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{w}^T\boldsymbol{x}) = g(\sum_{j=0}^{d} w_j x_j) = g(w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d)$$

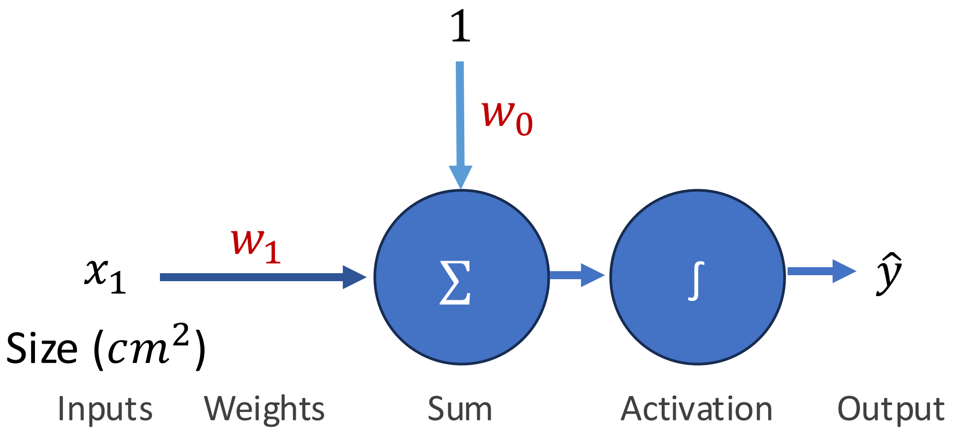
where $w_0, \ldots, w_d$ are **parameters/weights**, $x_0 = 1$ is a dummy variable.
g is sign function:

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Perceptron: An Example

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)



Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

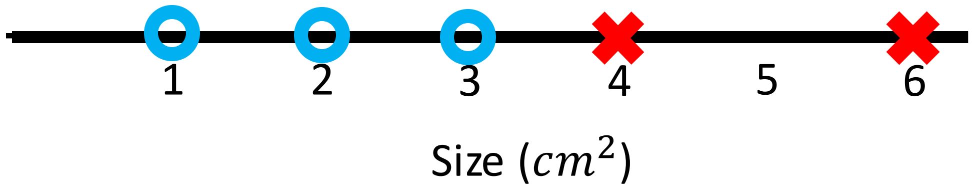
$$\hat{y} = \begin{cases} +1 & \text{if } w_0 x_0 + w_1 x_1 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$w_0 = -1.5$

$w_1 = 1$

$$\hat{y} = \begin{cases} +1 & \text{if } -1.5 + 1x_1 \geq 0 \longrightarrow x_1 \geq 1.5 \\ -1 & \text{otherwise} \end{cases}$$

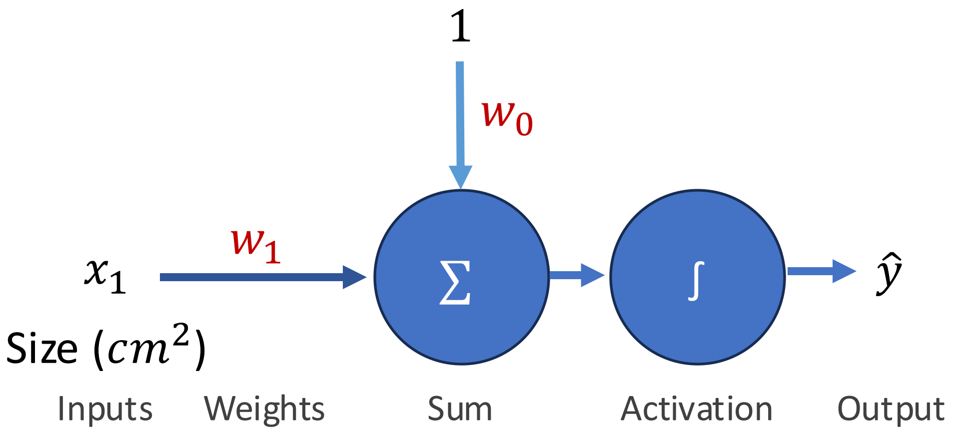
Cancer prediction: benign (-1), malignant (+1)



Perceptron: An Example

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)



Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

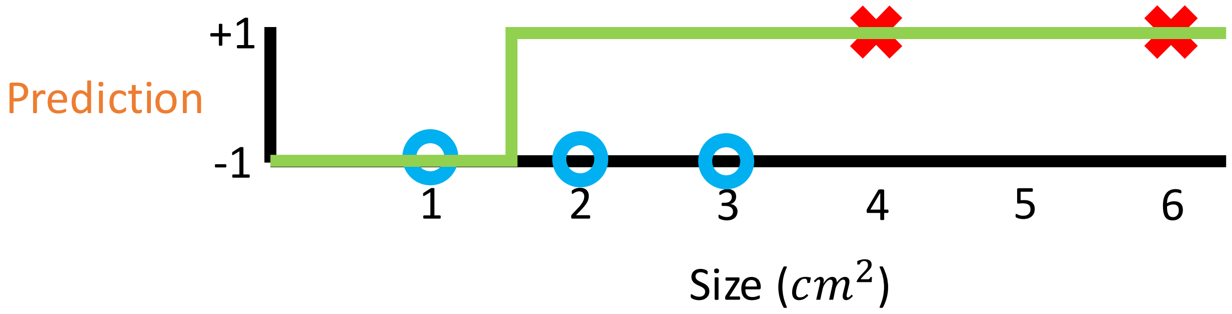
$$\hat{y} = \begin{cases} +1 & \text{if } w_0 x_0 + w_1 x_1 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$w_0 = -1.5$

$w_1 = 1$

$$\hat{y} = \begin{cases} +1 & \text{if } -1.5 + 1x_1 \geq 0 \longrightarrow x_1 \geq 1.5 \\ -1 & \text{otherwise} \end{cases}$$

Cancer prediction: benign (-1), malignant (+1)



Current model cannot predict all the labels correctly

-> Need to update $\boldsymbol{w}$

Perceptron Learning Algorithm

- Initialize $\boldsymbol{w}$
- Loop (until convergence or max steps reached)
  - For each instance $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$, classify $\hat{y}^{(i)} = h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})$
  - Select one misclassified instance $\left(\boldsymbol{x}^{(k)}, y^{(k)}\right)$
  - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right)\boldsymbol{x}^{(k)}$

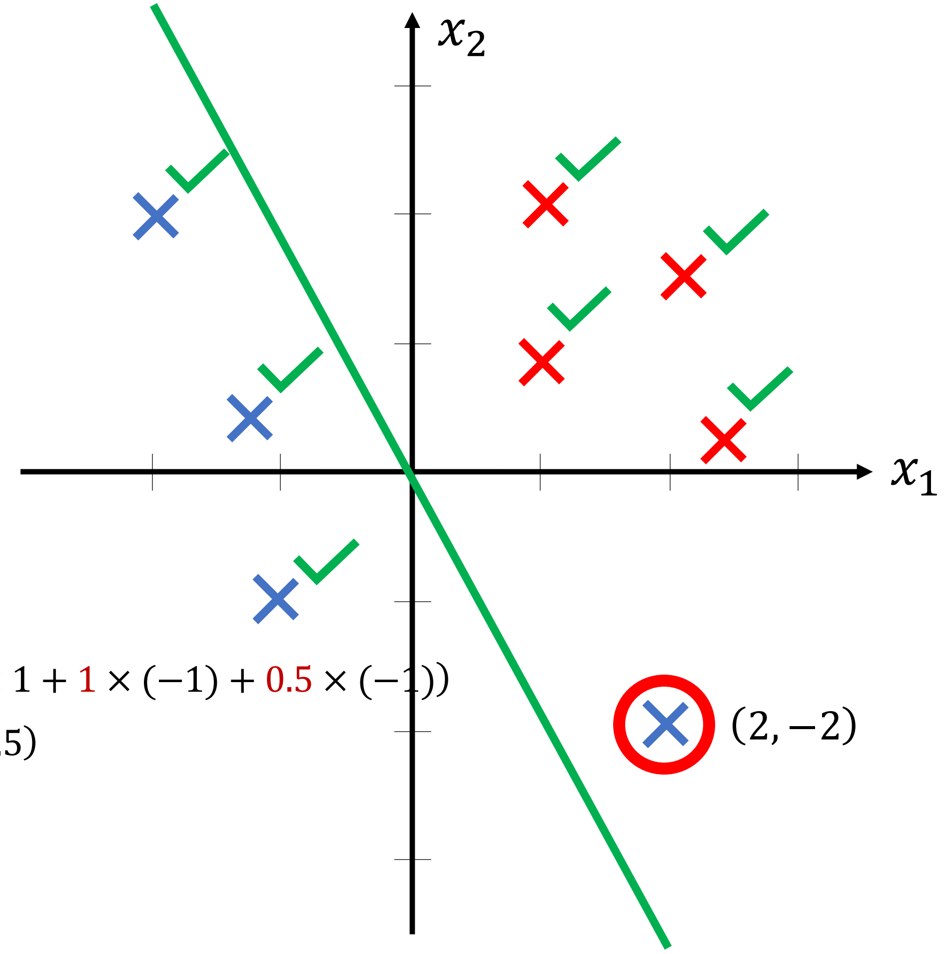
Learning rate

Use $y^{(i)}$ during training, supervised Learning!

Perceptron Learning Algorithm: An Example

$0x_0 + 1x_1 + 0.5x_2 = 0$

$\times$:+1 $\quad$ $\times$:−1



$g(0 \times 1 + 1 \times (-1) + 0.5 \times (-1))$
$= g(-1.5)$
$= -1$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\,w_0 x_0 + w_1 x_1 + w_2 x_2)$$
$$= g(0x_0 + 1x_1 + 0.5x_2)$$

weights initialization

$$g(0 \times 1 + 1 \times 2 + 0.5 \times (-2)) = g(1) = +1$$

misclassified instance

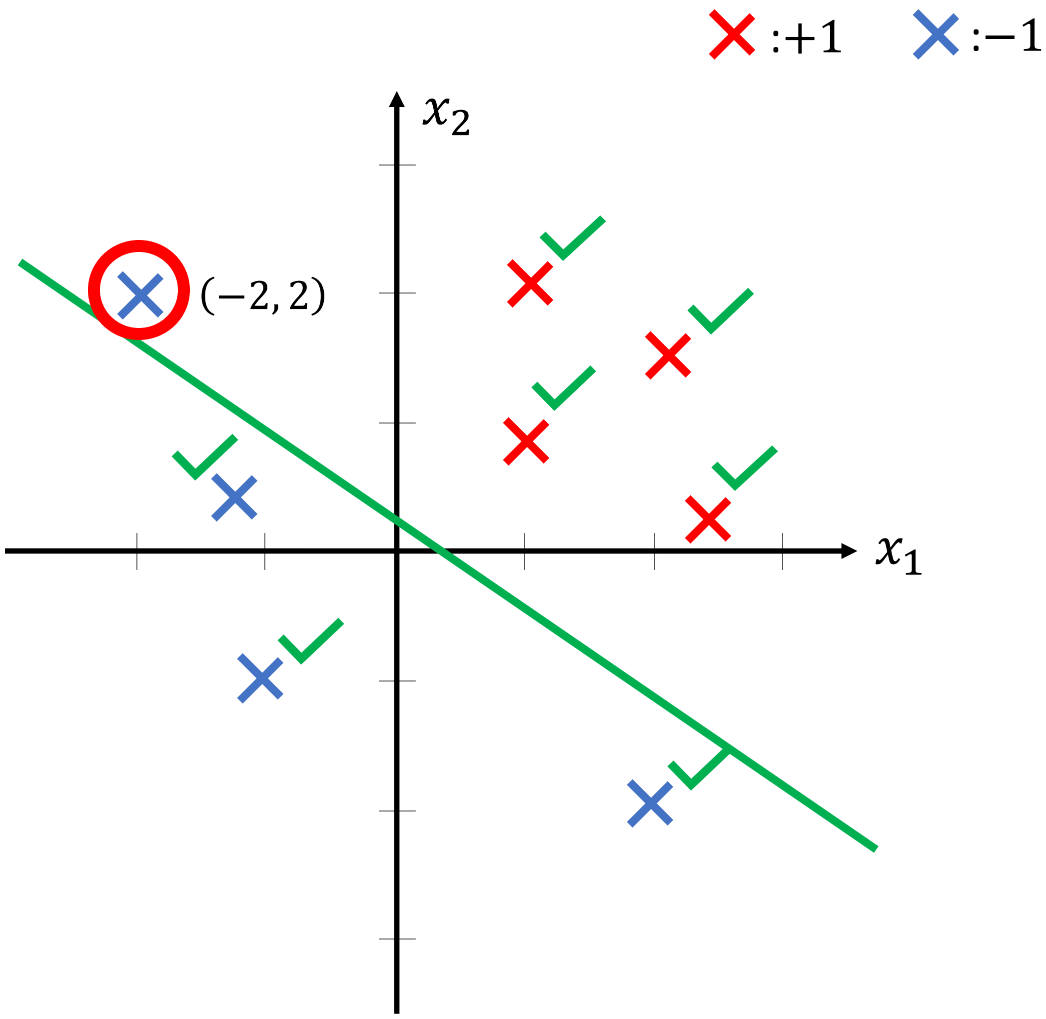
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right) \boldsymbol{x}^{(k)}$$

$$\begin{bmatrix} 0 \\ 1 \\ 0.5 \end{bmatrix} + 0.1(-1-1)\begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.6 \\ 0.9 \end{bmatrix}$$

New $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-0.2x_0 + 0.6x_1 + 0.9x_2)$

update weights

Perceptron Learning Algorithm: An Example



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\,w_0 x_0 + w_1 x_1 + w_2 x_2)$$

$$= g(-0.2x_0 + 0.6x_1 + 0.9x_2)$$

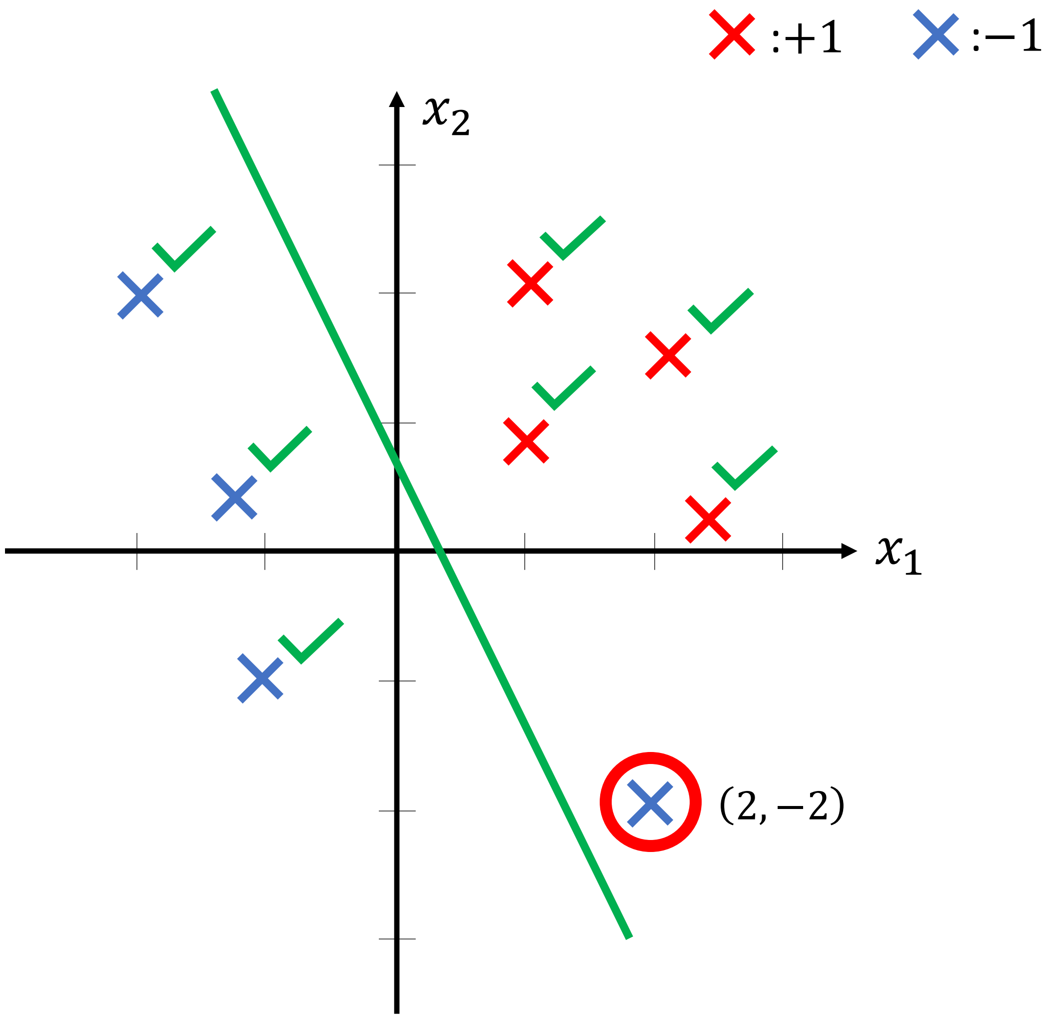
$g(-0.2 \times 1 + 0.6 \times (-2) + 0.9 \times 2) = g(0.4)$
$= +1$

$$\boldsymbol{w} \leftarrow \quad \boldsymbol{w} \quad + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right) \boldsymbol{x}^{(k)}$$

$$\begin{bmatrix} -0.2 \\ 0.6 \\ 0.9 \end{bmatrix} + 0.1(-1 - 1)\begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.4 \\ 1 \\ 0.5 \end{bmatrix}$$

New $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-0.4x_0 + 1x_1 + 0.5x_2)$

Perceptron Learning Algorithm: An Example



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\, w_0 x_0 + w_1 x_1 + w_2 x_2)$$

$$= g(-0.4 x_0 + 1 x_1 + 0.5 x_2)$$

$$g(-0.4 \times 1 + 1 \times 2 + 0.5 \times (-2)) = g(0.6)$$

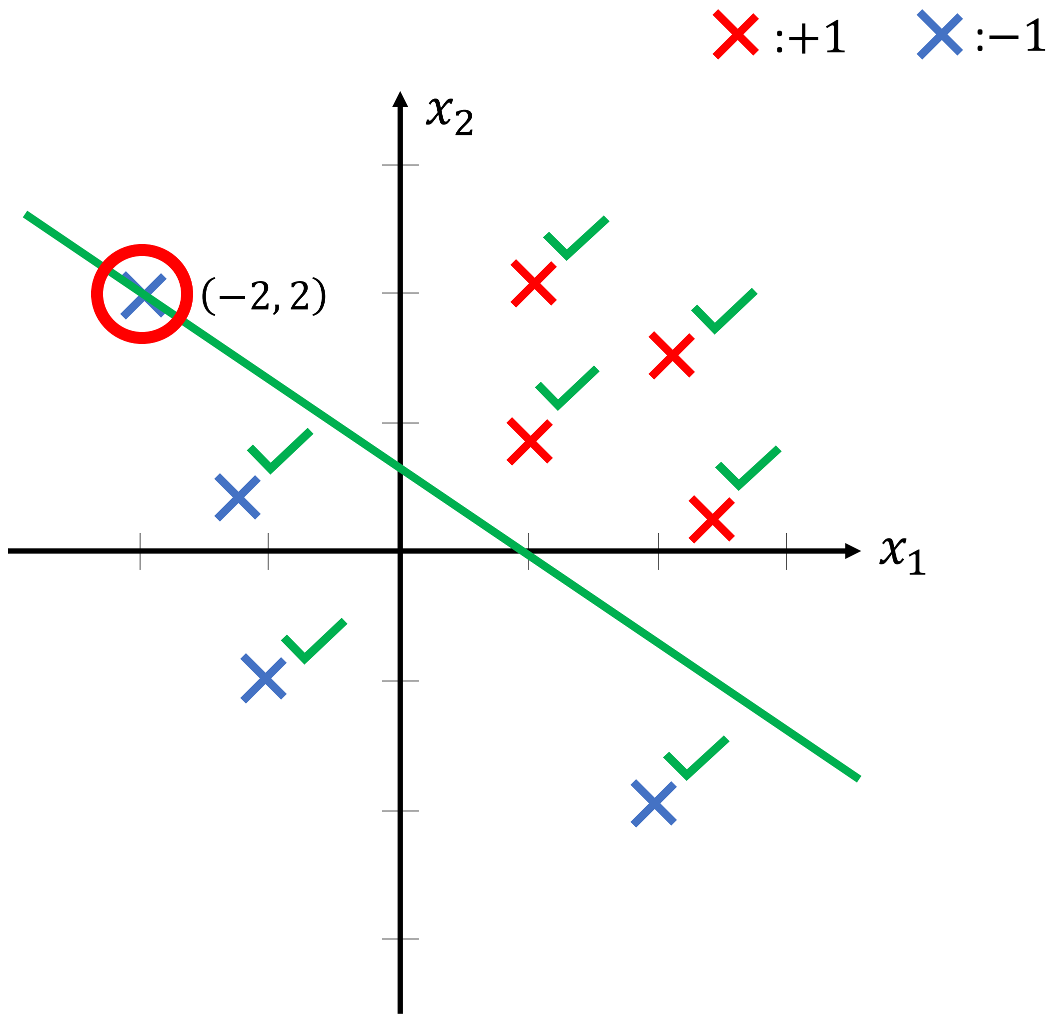
$$= +1$$

$$\boldsymbol{w} \leftarrow \quad \boldsymbol{w} \quad + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right) \boldsymbol{x}^{(k)}$$

$$\begin{bmatrix} -0.4 \\ 1 \\ 0.5 \end{bmatrix} + 0.1(-1 - 1) \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} -0.6 \\ 0.6 \\ 0.9 \end{bmatrix}$$

New $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-0.6 x_0 + 0.6 x_1 + 0.9 x_2)$

Perceptron Learning Algorithm: An Example



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\, w_0 x_0 + w_1 x_1 + w_2 x_2)$$

$$= g(-0.6 x_0 + 0.6 x_1 + 0.9 x_2)$$

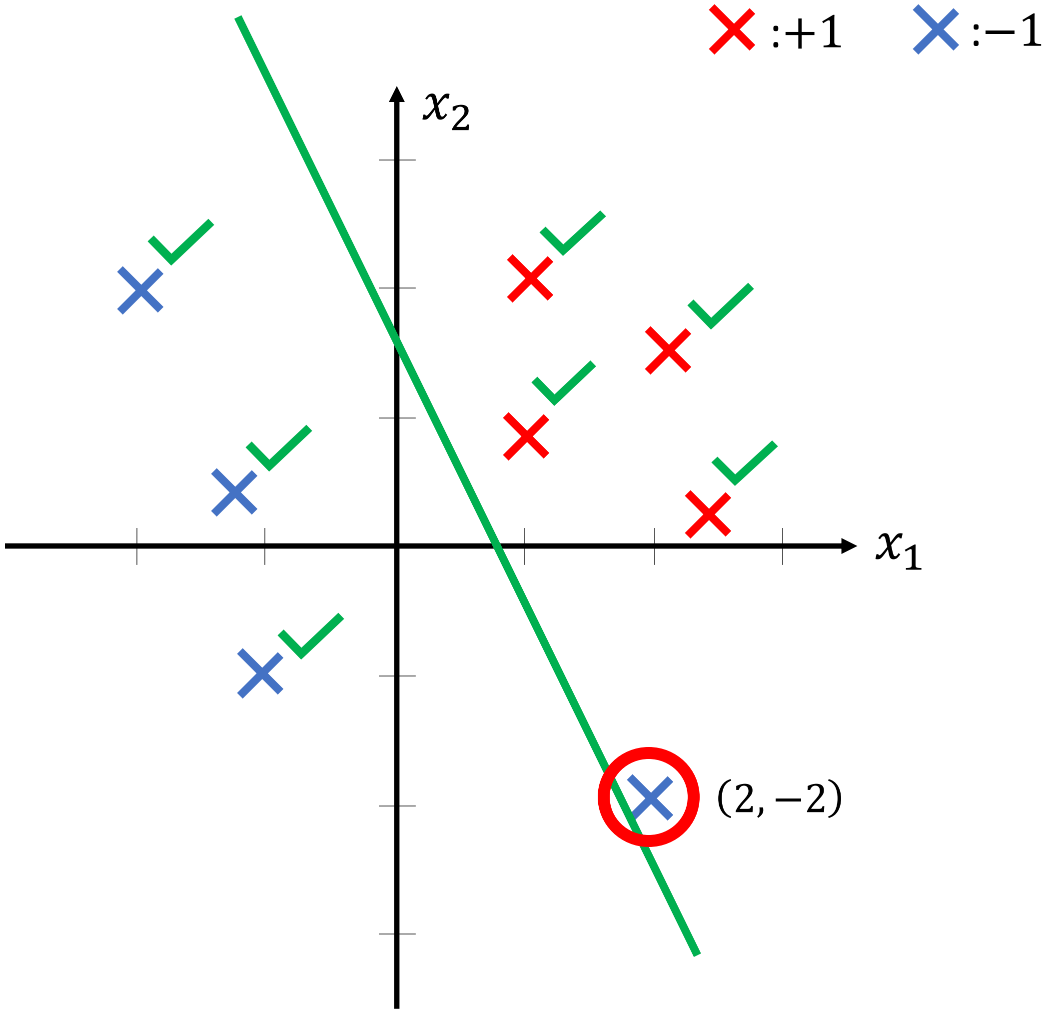
$g(-0.6 \times 1 + 0.6 \times (-2) + 0.9 \times 2) = g(0)$
$= +1$

$$\boldsymbol{w} \leftarrow \quad \boldsymbol{w} \quad + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right) \boldsymbol{x}^{(k)}$$

$$\begin{bmatrix} -0.6 \\ 0.6 \\ 0.9 \end{bmatrix} + 0.1(-1 - 1)\begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.8 \\ 1 \\ 0.5 \end{bmatrix}$$

New $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-0.8 x_0 + 1 x_1 + 0.5 x_2)$

Perceptron Learning Algorithm: An Example



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g(w_0 x_0 + w_1 x_1 + w_2 x_2)$$

$$= g(-0.8 x_0 + 1 x_1 + 0.5 x_2)$$

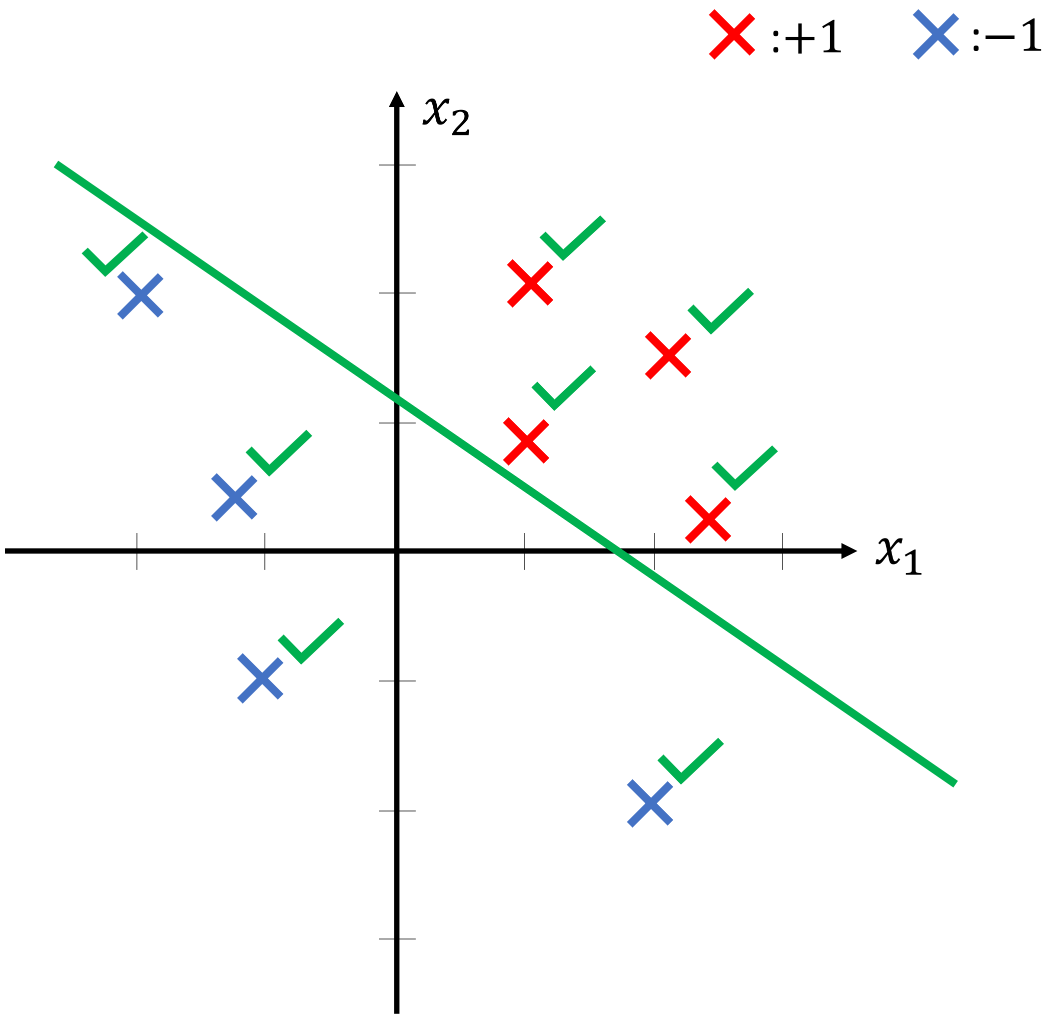
$$g(-0.8 \times 1 + 1 \times 2 + 0.5 \times (-2)) = g(0.2)$$
$$= +1$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma\left(y^{(k)} - \hat{y}^{(k)}\right) \boldsymbol{x}^{(k)}$$

$$\begin{bmatrix} -0.8 \\ 1 \\ 0.5 \end{bmatrix} + 0.1(-1-1) \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.6 \\ 0.9 \end{bmatrix}$$

New $h_{\boldsymbol{w}}(\boldsymbol{x}) = g(-1 x_0 + 0.6 x_1 + 0.9 x_2)$

Perceptron Learning Algorithm: An Example



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$

$x_0 = 1$ (dummy variable)

Sign function

$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

$$\hat{y} = h_w(x) = g(\, w_0 x_0 + w_1 x_1 + w_2 x_2)$$
$$= g(-1 x_0 + 0.6 x_1 + 0.9 x_2)$$

No misclassifications! Converged!

Different from SVM, we do not maximize margin here.

What if it's not linearly separable?

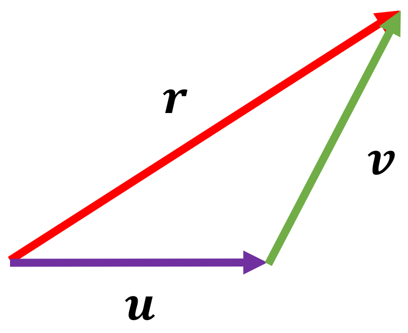
The algorithm will not converge

Background: Vector Addition

Suppose that we have two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$,

$$\boldsymbol{r} = \boldsymbol{u} + \boldsymbol{v}$$

Vector addition can be visualized in a vector diagram by drawing the vectors to be added tip-to-tail:



Background: Dot Product and Angle

Suppose that we have two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$, dot product:

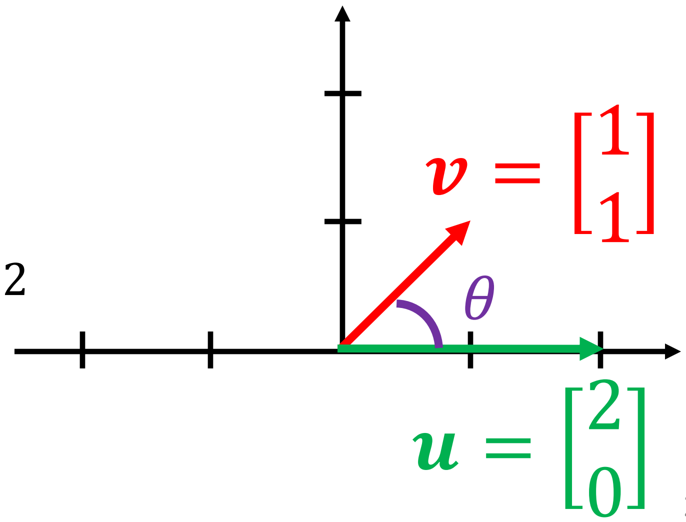
$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v} = \|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos\theta$$

where θ is the angle between two vectors and $0 \leq \theta \leq \pi$.

If $\boldsymbol{u} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\boldsymbol{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, we have $\|\boldsymbol{u}\| = 2, \|\boldsymbol{v}\| = \sqrt{2}, \theta = \frac{\pi}{4}, \cos\theta = \frac{\sqrt{2}}{2}$

$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v} = \begin{bmatrix} 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2$$

$$\|\boldsymbol{u}\| \|\boldsymbol{v}\| \cos\theta = 2 \times \sqrt{2} \times \frac{\sqrt{2}}{2} = 2$$



Perceptron Learning Algorithm: Why?

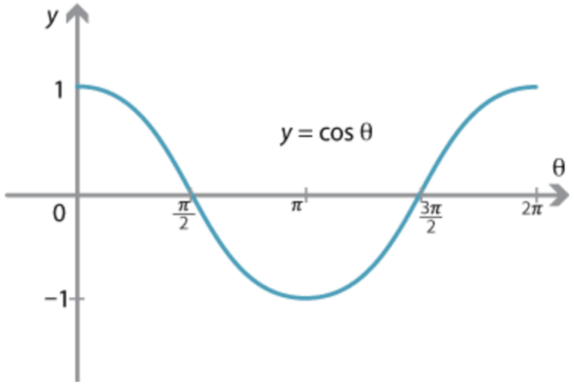
$$\hat{y} = g\left(\sum_{j=0}^{d} w_j x_j\right) = g(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} +1, & if\ \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1, & if\ \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

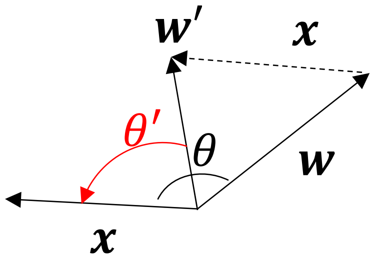
When there is a misclassification:

- **Case 1**: $y = +1, \hat{y} = -1$

What we have:		What we want:
$\hat{y} = -1$		$\hat{y} = +1$
$\boldsymbol{w}^T\boldsymbol{x} < 0$		$\boldsymbol{w}^T\boldsymbol{x} \geq 0$
$\boldsymbol{w} \cdot \boldsymbol{x} < 0$		$\boldsymbol{w} \cdot \boldsymbol{x} \geq 0$
$\Vert\boldsymbol{w}\Vert\Vert\boldsymbol{x}\Vert \cos\theta < 0$		$\Vert\boldsymbol{w}\Vert\Vert\boldsymbol{x}\Vert \cos\theta \geq 0$
$\cos\theta < 0$		$\cos\theta \geq 0$
$\frac{\pi}{2} < \theta \leq \pi$	$\xrightarrow{\text{reduce } \theta}$	$0 \leq \theta \leq \frac{\pi}{2}$

Doesn't affect the sign





$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \boldsymbol{x}$

θ' will be smaller (as required)

Image credit: https://amsi.org.au/ESA_Senior_Years/SeniorTopic2/2d/2d_2content_6.html

Perceptron Learning Algorithm: Why?

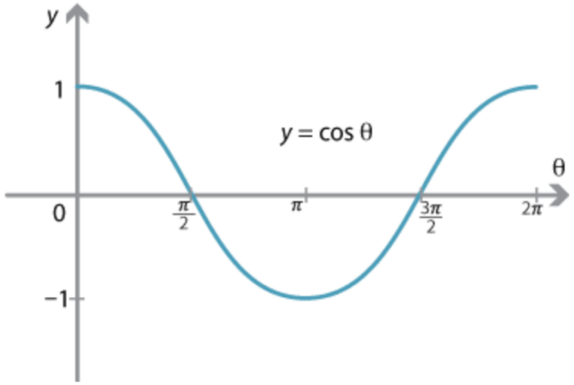
$$\hat{y} = g\left(\sum_{j=0}^{d} w_j x_j\right) = g(\boldsymbol{w}^T \boldsymbol{x}) = \begin{cases} +1, & if\ \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1, & if\ \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

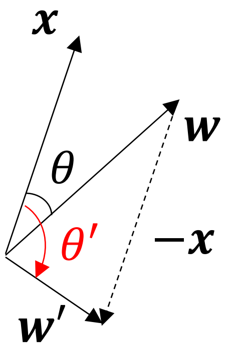
When there is a misclassification:

- **Case 2**: $y = -1, \hat{y} = +1$

What we have:		What we want:
$\hat{y} = +1$		$\hat{y} = -1$
$\boldsymbol{w}^T\boldsymbol{x} \geq 0$		$\boldsymbol{w}^T\boldsymbol{x} < 0$
$\boldsymbol{w} \cdot \boldsymbol{x} \geq 0$		$\boldsymbol{w} \cdot \boldsymbol{x} < 0$
$\lVert\boldsymbol{w}\rVert\lVert\boldsymbol{x}\rVert \cos\theta \geq 0$		$\lVert\boldsymbol{w}\rVert\lVert\boldsymbol{x}\rVert \cos\theta < 0$
$\cos\theta \geq 0$		$\cos\theta < 0$
$0 \leq \theta \leq \frac{\pi}{2}$	increase θ	$\frac{\pi}{2} < \theta \leq \pi$

Doesn't affect the sign





$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \boldsymbol{x}$

θ' will be larger (as required)

Image credit: https://amsi.org.au/ESA_Senior_Years/SeniorTopic2/2d/2d_2content_6.html

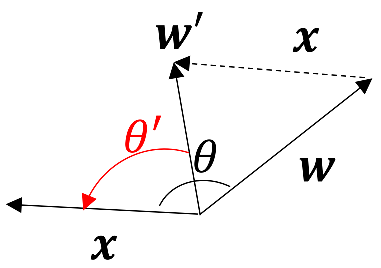
Perceptron Learning Algorithm: Why?

$$\hat{y} = g\left(\sum_{j=0}^{d} w_j x_j\right) = g(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} +1, & if\ \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1, & if\ \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

When there is a misclassification:

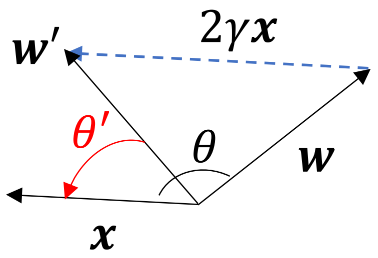
$\hat{y} = -1$

$y = +1$



$\boldsymbol{w}' \leftarrow \boldsymbol{w} + \boldsymbol{x}$

θ' will be smaller (as required)



$\boldsymbol{w}' \leftarrow \boldsymbol{w} + 2\gamma\boldsymbol{x}$ $\quad$ γ is constant

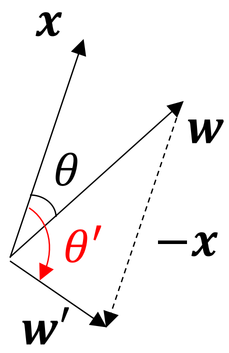
$\boldsymbol{w} + \gamma(+1 - (-1))\boldsymbol{x}$

$\boldsymbol{w} + \gamma(y - \hat{y})\boldsymbol{x}$

Weight updating rule in PLA

$\hat{y} = +1$

$y = -1$



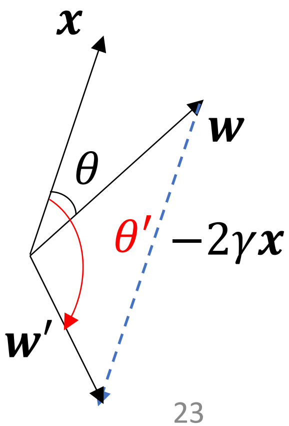
$\boldsymbol{w}' \leftarrow \boldsymbol{w} - \boldsymbol{x}$

θ' will be larger (as required)

$\boldsymbol{w}' \leftarrow \boldsymbol{w} - 2\gamma\boldsymbol{x}$

$\boldsymbol{w} + \gamma(-1 - (+1))\boldsymbol{x}$

$\boldsymbol{w} + \gamma(y - \hat{y})\boldsymbol{x}$

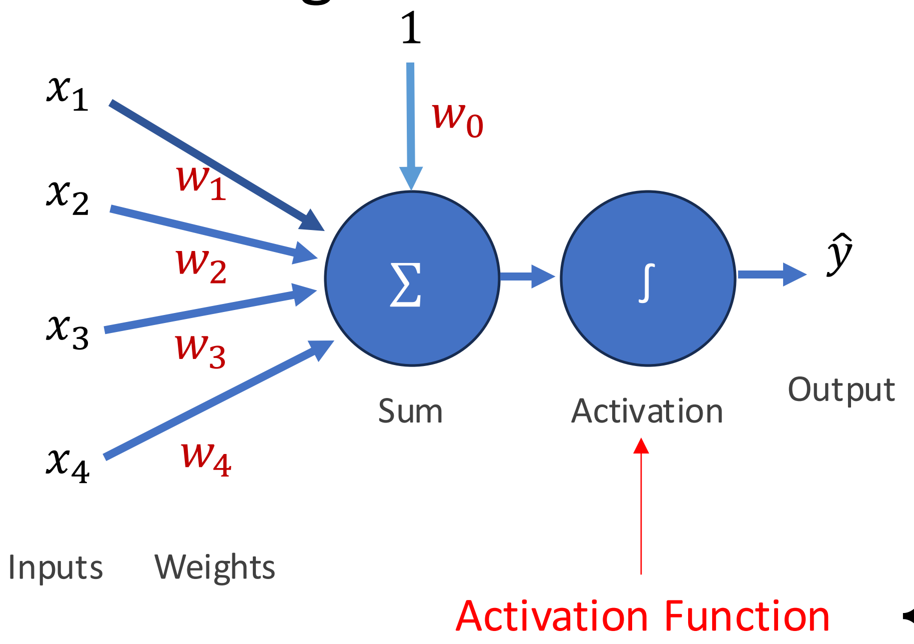


Outline

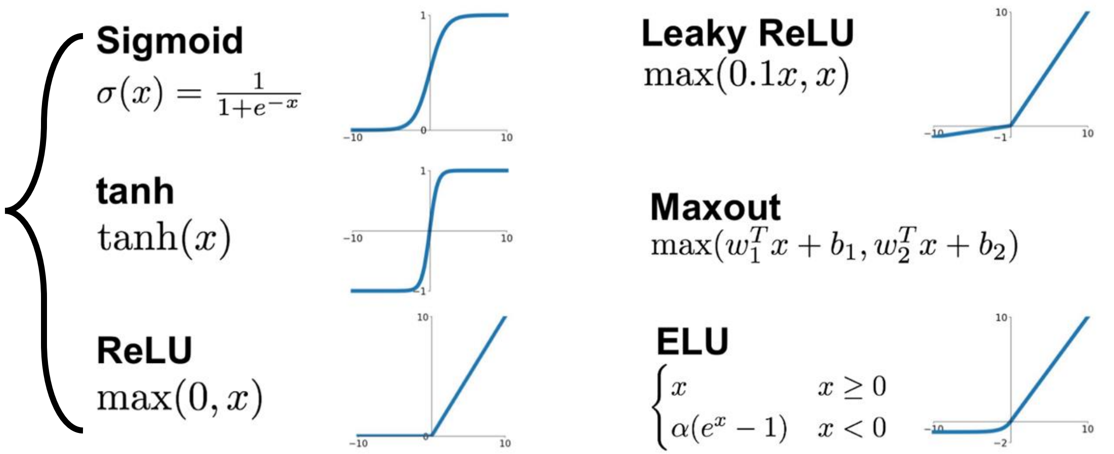
- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- **Neural Network**
  - Neuron
  - AND Gate Modelling
  - XNOR Gate Modelling
  - Single-layer and Multi-layer Neural Networks
- Multi-class Classification

Neuron

- Neuron is the building block of neural networks. An artificial neuron is a more generalized version of the perceptron.



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right)$$



Linear Regression Model

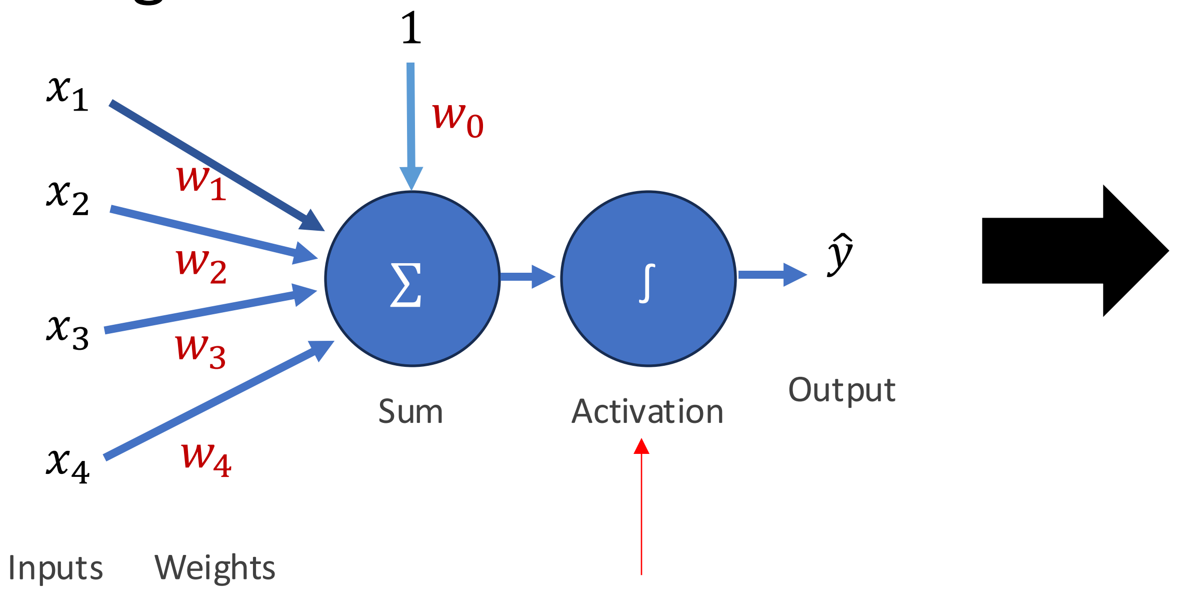
Given an input vector x of dimension d, the hypothesis class of linear models is defined as the set of functions:

$$h_w(x) = \sum_{j=0}^{d} w_j x_j$$

Where $w_0, \ldots, w_d$ are **parameters/weights.**

Neuron v.s. Linear Regression Model

- A neuron with linear activation function is equivalent to linear regression model.



Identity function/
Linear activation function
$g(z) = z$

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right) = \sum_{j=0}^{d} w_j x_j$$

Logistic Regression Model

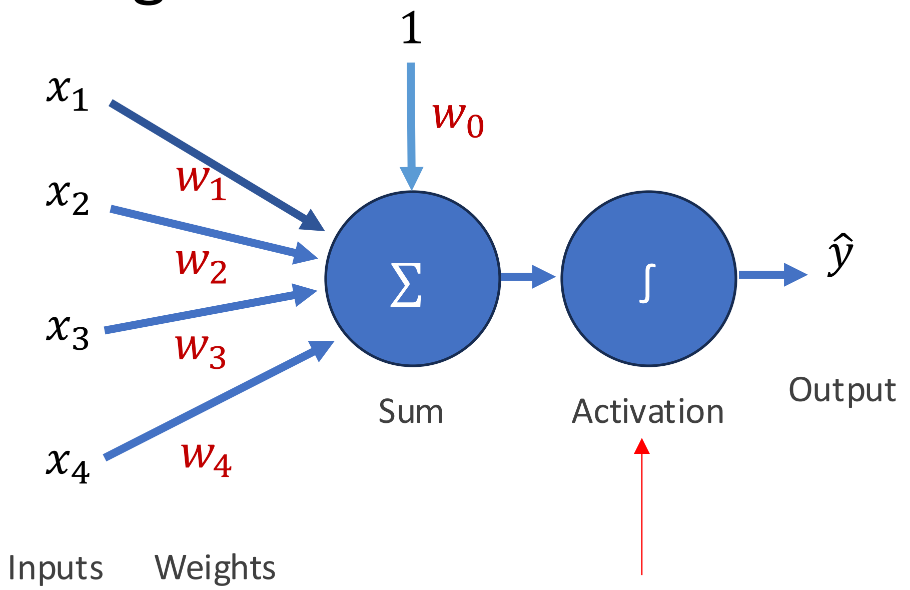
Given an input vector x of dimension d, the hypothesis class of linear models is defined as the set of functions:

$$h_w(x) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right)$$

Where $w_0, \dots, w_d$ are **parameters/weights** and σ is the sigmoid function.

Neuron v.s. Logistic Regression Model

- A neuron with sigmoid activation function is equivalent to logistic regression model.



Sigmoid function

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

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} w_j x_j\right) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right)$$

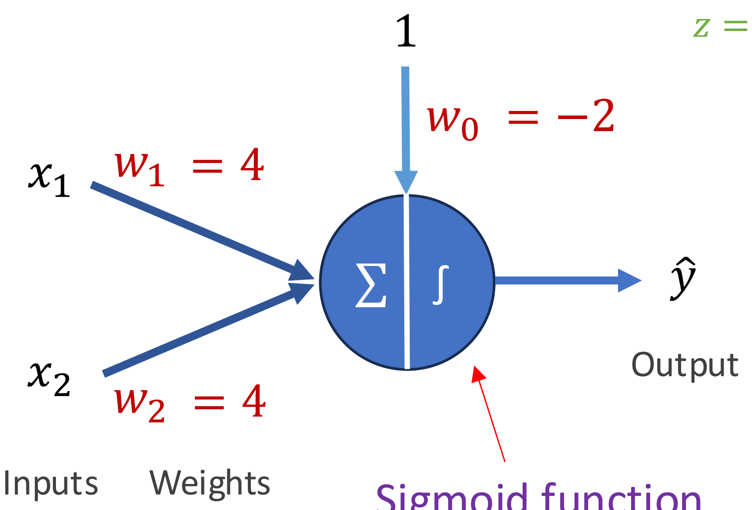
$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) \qquad x_0 = 1 \text{ (dummy variable)}$$

OR Gate Modelling

With Logistic Regression Model

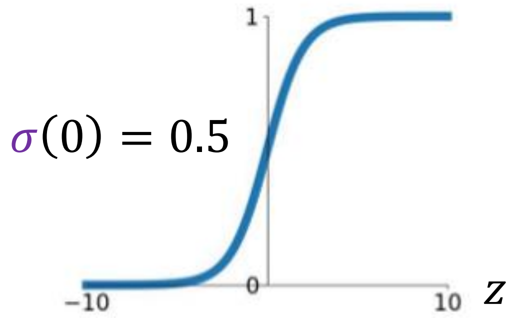
Decision threshold: $\hat{y} = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) = 0.5 = \sigma(0)$

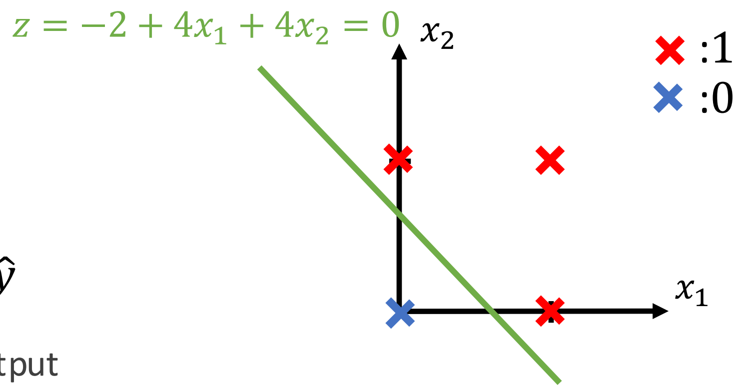
Decision boundary: $z = \sum_{j=0}^{d} w_j x_j = 0$



Sigmoid function

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





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

x_0	x_1	x_2	y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$\hat{y} = \sigma(z), z = w_0 x_0 + w_1 x_1 + w_2 x_2$

$z \geq 0$, predicted as 1

$z < 0$, predicted as 0

$$\boldsymbol{x}^{(1)} = \begin{bmatrix}1\\0\\0\end{bmatrix}, y^{(1)} = 0: \qquad \boldsymbol{x}^{(2)} = \begin{bmatrix}1\\0\\1\end{bmatrix}, y^{(2)} = 1:$$

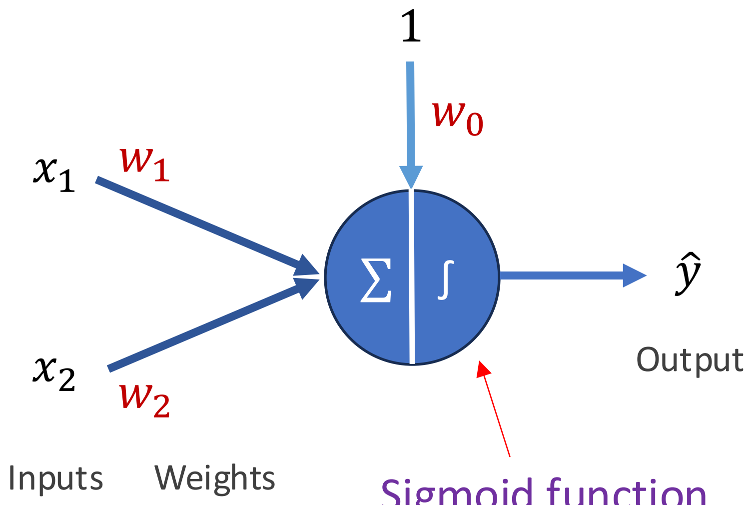
$$z^{(1)} = w_0 < 0 \qquad z^{(2)} = w_0 + w_2 \geq 0$$

$$\boldsymbol{x}^{(3)} = \begin{bmatrix}1\\1\\0\end{bmatrix}, y^{(3)} = 1: \qquad \boldsymbol{x}^{(4)} = \begin{bmatrix}1\\1\\1\end{bmatrix}, y^{(4)} = 1$$

$$z^{(3)} = w_0 + w_1 \geq 0 \qquad z^{(4)} = w_0 + w_1 + w_2 \geq 0$$

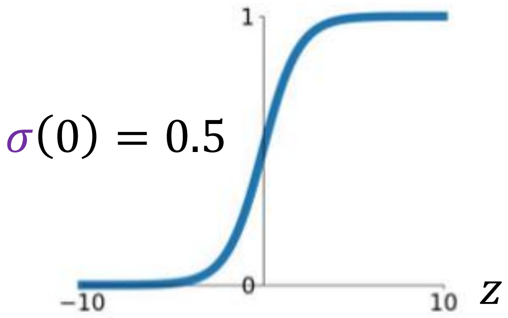
AND Gate Modelling

With Logistic Regression Model



Sigmoid function

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



$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) \qquad x_0 = 1 \text{ (dummy variable)}$$

Decision threshold: $\hat{y} = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) = 0.5 = \sigma(0)$

Decision boundary: $z = \sum_{j=0}^{d} w_j x_j = 0$

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

x_0	x_1	x_2	y
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$\hat{y} = \sigma(z), z = w_0 x_0 + w_1 x_1 + w_2 x_2$

$z \geq 0$, predicted as 1
$z < 0$, predicted as 0

Poll Everywhere

To correctly model the AND gate, which of the following options can be used to set the parameters in the logistic regression model?

A: $w_0 = -6, w_1 = 4, w_2 = 4$

B: $w_0 = -5, w_1 = 2, w_2 = 2$

C: $w_0 = 3, w_1 = -2, w_2 = -2$

D: $w_0 = -4, w_1 = -2, w_2 = -2$

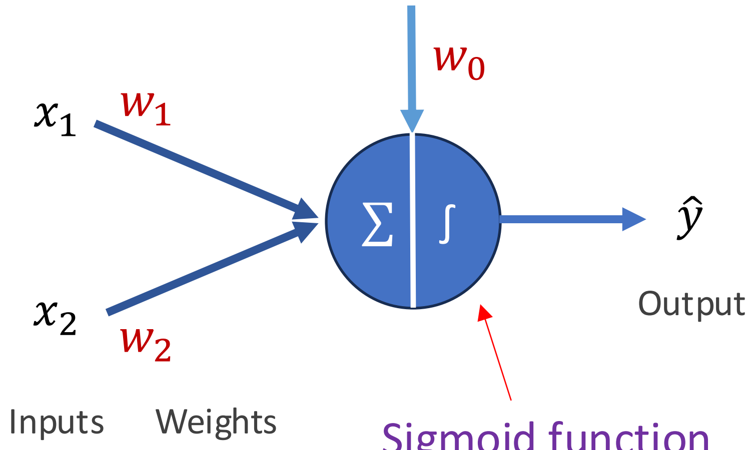
Consider $x_1, x_2 \in \{1,0\}$

x_0	x_1	x_2	y
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$\hat{y} = \sigma(z), z = w_0x_0 + w_1x_1 + w_2x_2$

$z \geq 0$, predicted as 1

$z < 0$, predicted as 0



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

Poll Everywhere

To correctly model the AND gate, which of the following options can be used to set the parameters in the logistic regression model?

A: $\boldsymbol{w_0 = -6, w_1 = 4, w_2 = 4}$

B: $w_0 = -5, w_1 = 2, w_2 = 2$

C: $w_0 = 3, w_1 = -2, w_2 = -2$

D: $w_0 = -4, w_1 = -2, w_2 = -2$

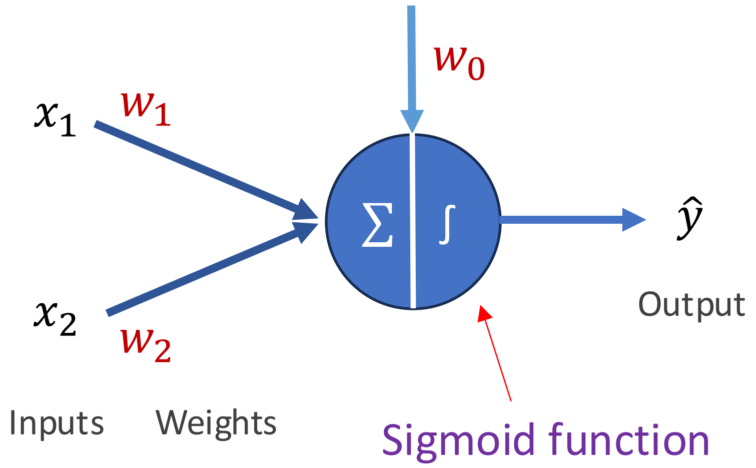
Consider $x_1, x_2 \in \{1,0\}$

x_0	x_1	x_2	y
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$\hat{y} = \sigma(z), z = w_0 x_0 + w_1 x_1 + w_2 x_2$

$z \geq 0$, predicted as 1

$z < 0$, predicted as 0



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

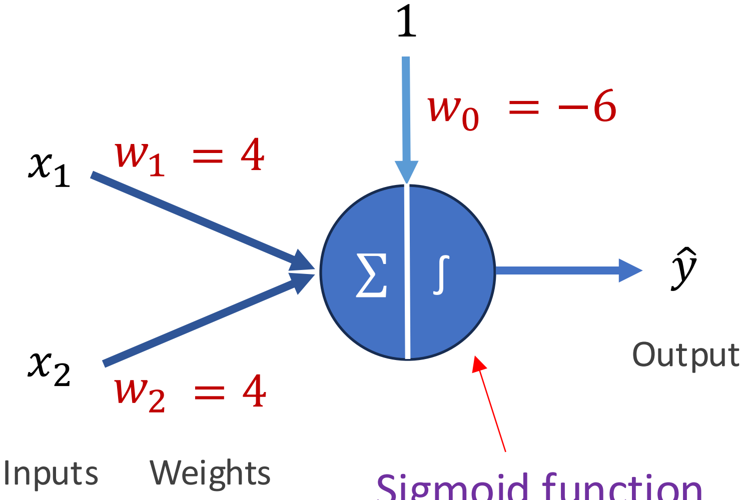
AND Gate Modelling

With Logistic Regression Model

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) \qquad x_0 = 1 \text{ (dummy variable)}$$

Decision threshold: $\hat{y} = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) = 0.5 = \sigma(0)$

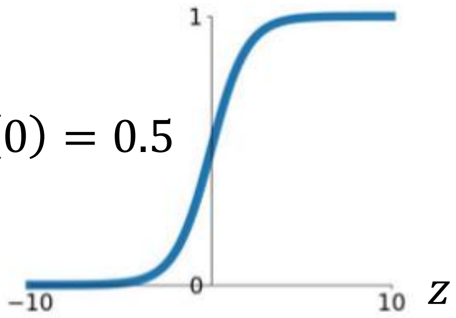
Decision boundary: $z = \sum_{j=0}^{d} w_j x_j = 0$



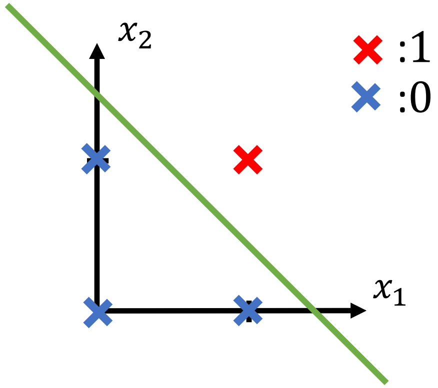
Sigmoid function

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

$\sigma(0) = 0.5$



$z = -6 + 4x_1 + 4x_2 = 0$



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

x_0	x_1	x_2	y
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$\hat{y} = \sigma(z), z = w_0 x_0 + w_1 x_1 + w_2 x_2$

$z \geq 0$, predicted as 1

$z < 0$, predicted as 0

$\boldsymbol{x}^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, y^{(1)} = 0$: $\qquad \boldsymbol{x}^{(2)} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, y^{(2)} = 0$:

$z^{(1)} = w_0 < 0 \qquad z^{(2)} = w_0 + w_2 < 0$

$\boldsymbol{x}^{(3)} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, y^{(3)} = 0 \qquad \boldsymbol{x}^{(4)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, y^{(4)} = 1$

$z^{(3)} = w_0 + w_1 < 0 \qquad z^{(4)} = w_0 + w_1 + w_2 \geq 0$

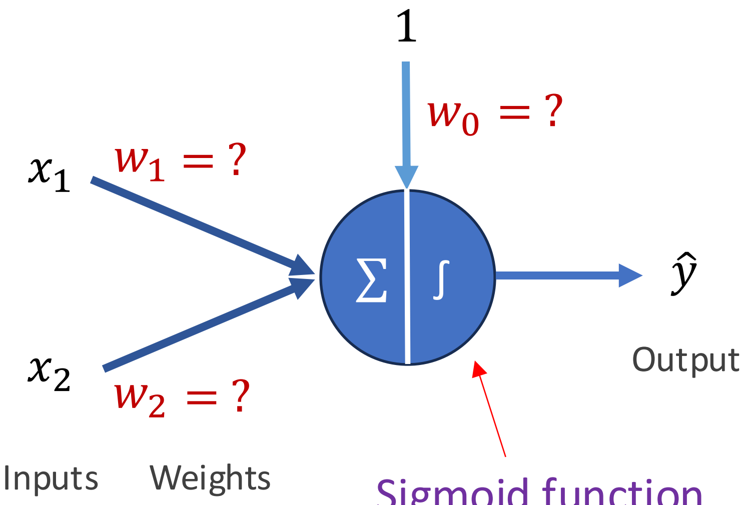
XNOR Gate Modelling

With Logistic Regression Model

$$\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) \qquad x_0 = 1 \text{ (dummy variable)}$$

Decision threshold: $\hat{y} = \sigma\left(\sum_{j=0}^{d} w_j x_j\right) = 0.5 = \sigma(0)$

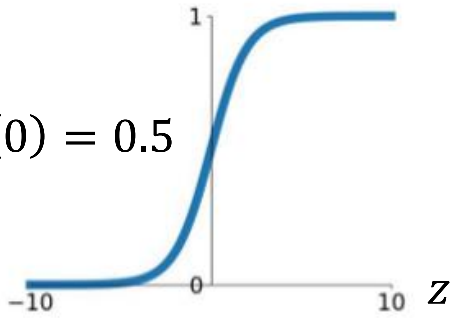
Decision boundary: $z = \sum_{j=0}^{d} w_j x_j = 0$



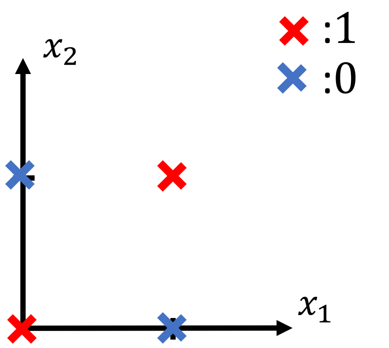
Sigmoid function

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

$\sigma(0) = 0.5$



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



x_0	x_1	x_2	y
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$\hat{y} = \sigma(z), z = w_0 x_0 + w_1 x_1 + w_2 x_2$

Not linearly separable

Feature engineering is needed to transform existing features:

- Manually create new features, e.g., ${x_1}^2, e^{x_2}, x_1 x_2$ …
- Use neurons to transform features

Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

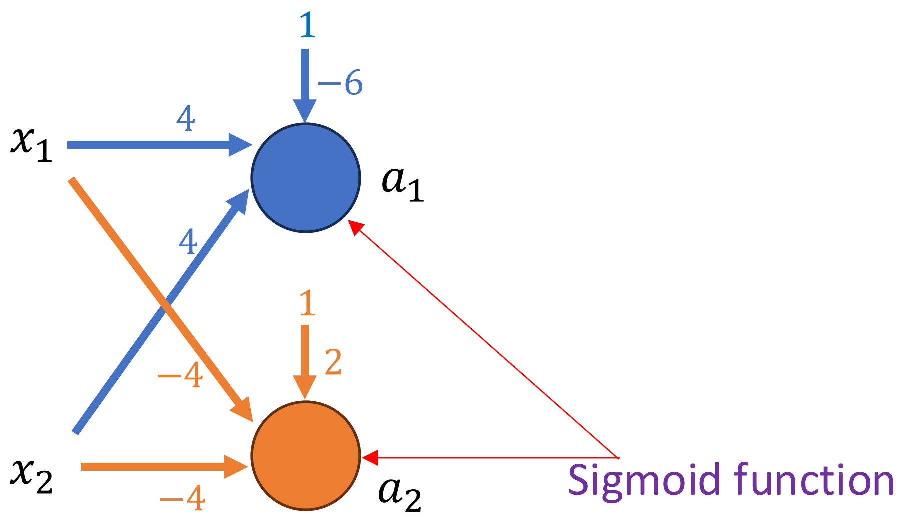
XNOR Gate Modelling

With More Neurons

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$

a_1 and a_2 are new features created based on existing features x_1 and x_2.



Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

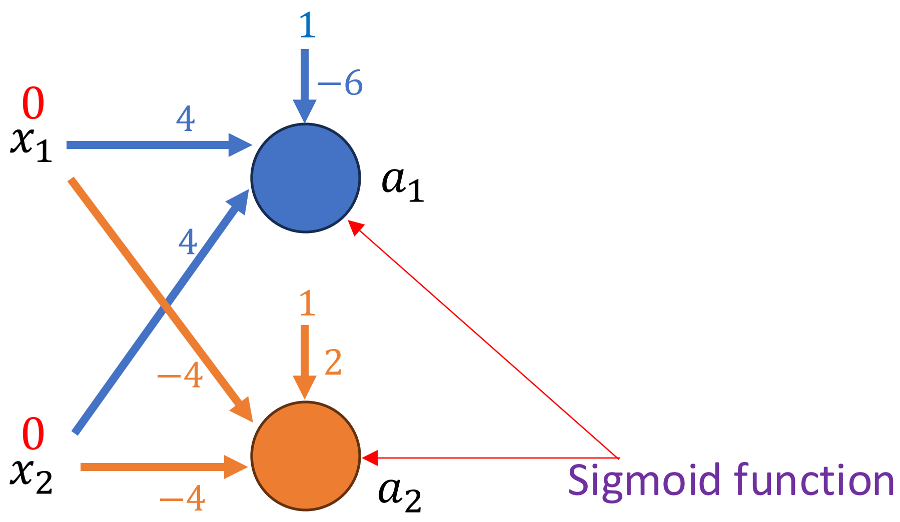
XNOR Gate Modelling

With More Neurons

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4 \times 0 + 4 \times 0) = 0.002$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4) \times 0 + (-4) \times 0) = 0.881$$

a_1 and a_2 are new features created based on existing features x_1 and x_2.



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

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881

Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

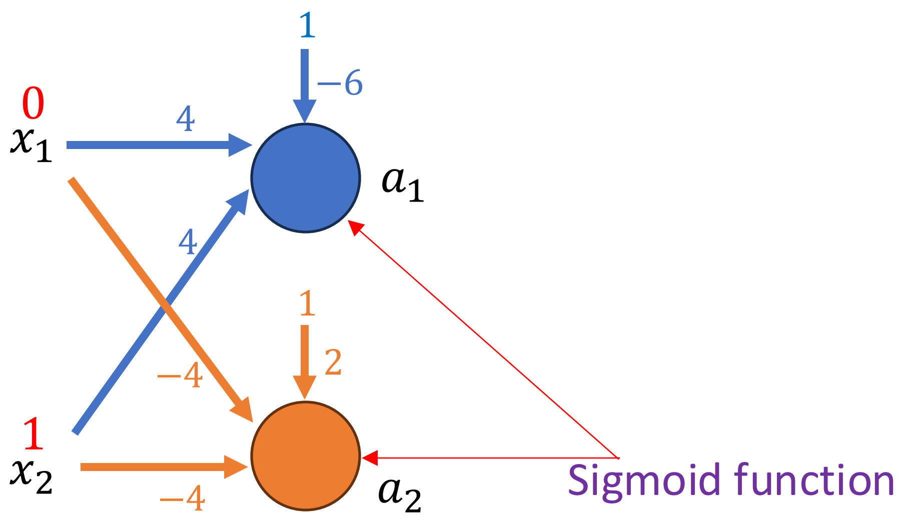
XNOR Gate Modelling

With More Neurons

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4 \times 0 + 4 \times 1) = 0.119$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4) \times 0 + (-4) \times 1) = 0.119$$

a_1 and a_2 are new features created based on existing features x_1 and x_2.



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

x_1	x_2	y	a_1	a_2
0	0	1	0.002	0.881
0	1	0	0.119	0.119
1	0	0		
1	1	1		

Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

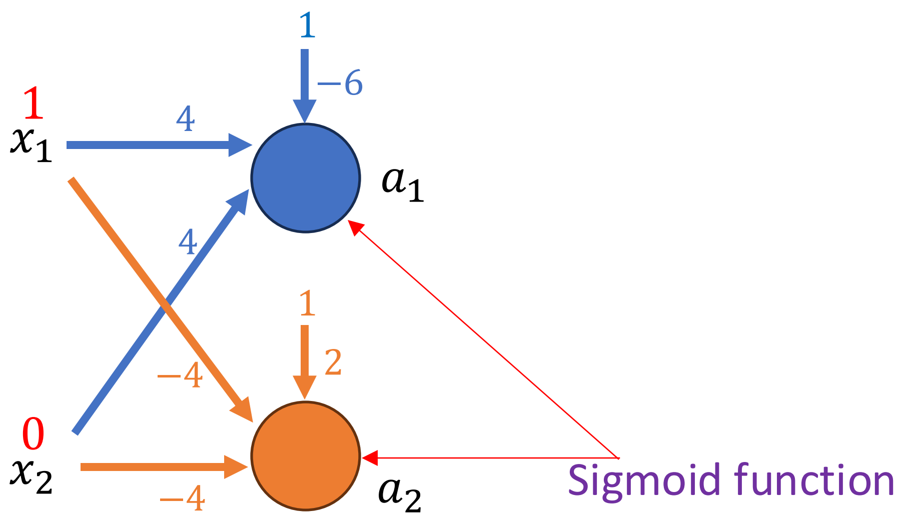
XNOR Gate Modelling

With More Neurons

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4 \times 1 + 4 \times 0) = 0.119$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4) \times 1 + (-4) \times 0) = 0.119$$

a_1 and a_2 are new features created based on existing features x_1 and x_2.



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

x_1	x_2	y	a_1	a_2
0	0	1	0.002	0.881
0	1	0	0.119	0.119
1	0	0	0.119	0.119
1	1	1		

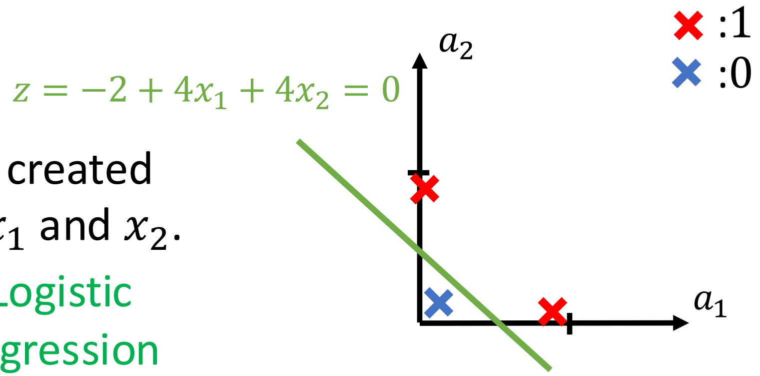
Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

XNOR Gate Modelling

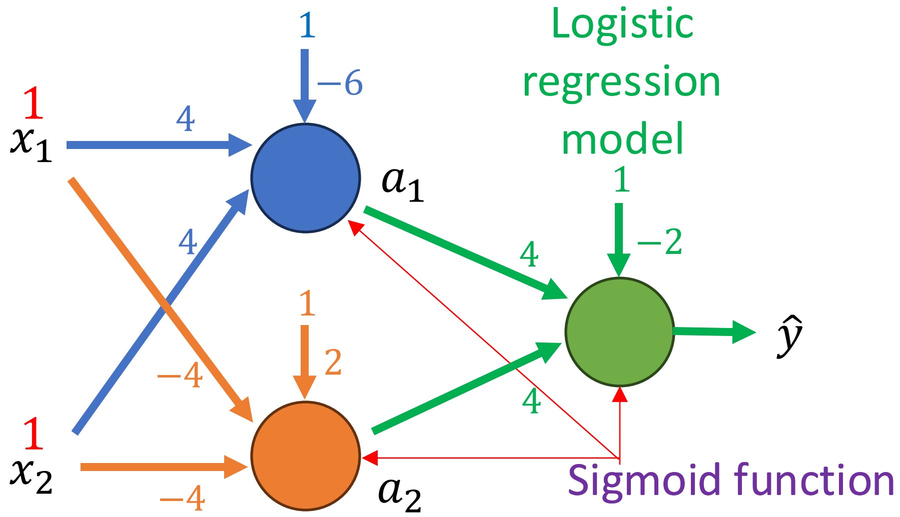
With More Neurons

$$a_1 = \sigma(-6 + 4x_1 + 4x_2)$$
$$= \sigma(-6 + 4 \times 1 + 4 \times 1) = 0.881$$

$$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$$
$$= \sigma(2 + (-4) \times 1 + (-4) \times 1) = 0.002$$



a_1 and a_2 are new features created based on existing features x_1 and x_2.



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

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

a_1	a_2
0.002	0.881
0.119	0.119
0.119	0.119
0.881	0.002

Decision threshold: $\hat{y} = 0.5 = \sigma(0)$

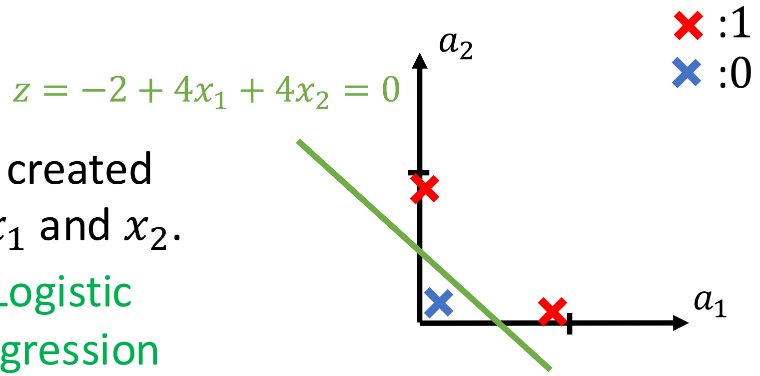
XNOR Gate Modelling

With More Neurons

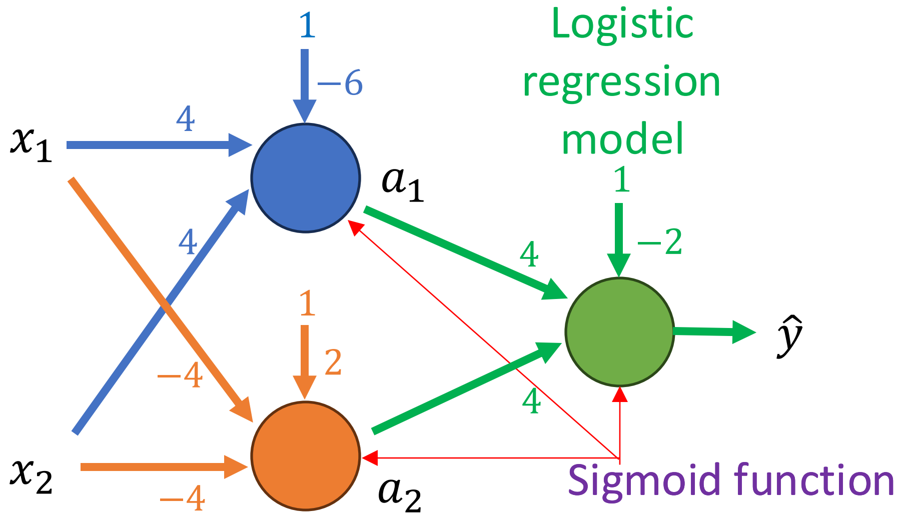
$a_1 = \sigma(-6 + 4x_1 + 4x_2)$

$a_2 = \sigma(2 + (-4)x_1 + (-4)x_2)$

$\hat{y} = \sigma(-2 + 4a_1 + 4a_2)$



a_1 and a_2 are new features created based on existing features x_1 and x_2.



Multi-layer neural network

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

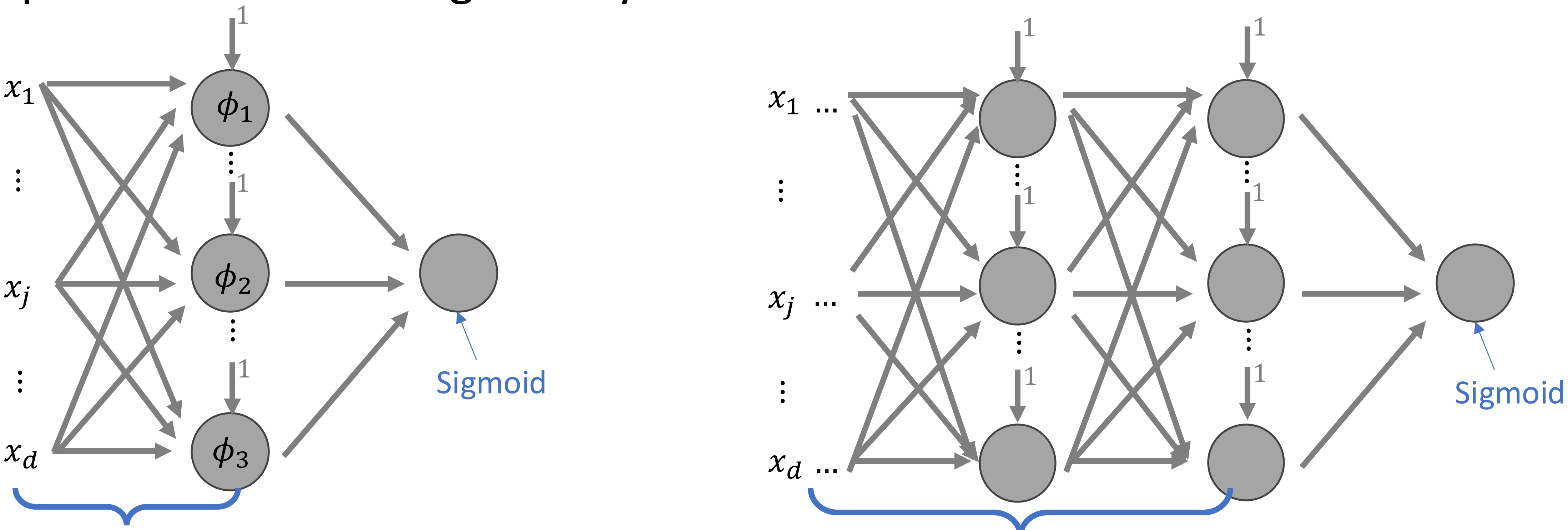
x_1	x_2	y	a_1	a_2	$\hat{y}$	Pred. label
0	0	1	0.002	0.881	1.532	1
0	1	0	0.119	0.119	−1.048	0
1	0	0	0.119	0.119	−1.048	0
1	1	1	0.881	0.002	1.532	1

How to learn the weights for neurons will be covered in the next lecture.

Neural Network v.s. Logistic Regression Model

With Feature Engineering

Logistic regression relies on manual feature engineering to capture complex patterns, while a multi-layer neural network learns its own feature representations through its layers with activation functions.



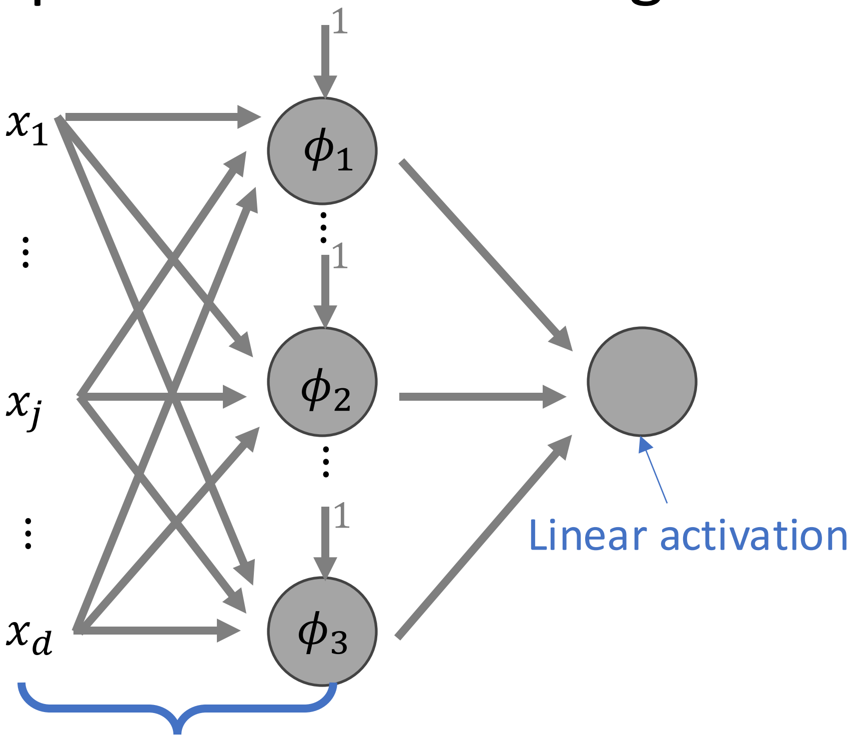
Manual feature engineering in logistic regression

Learned feature engineering in multi-layer neural network

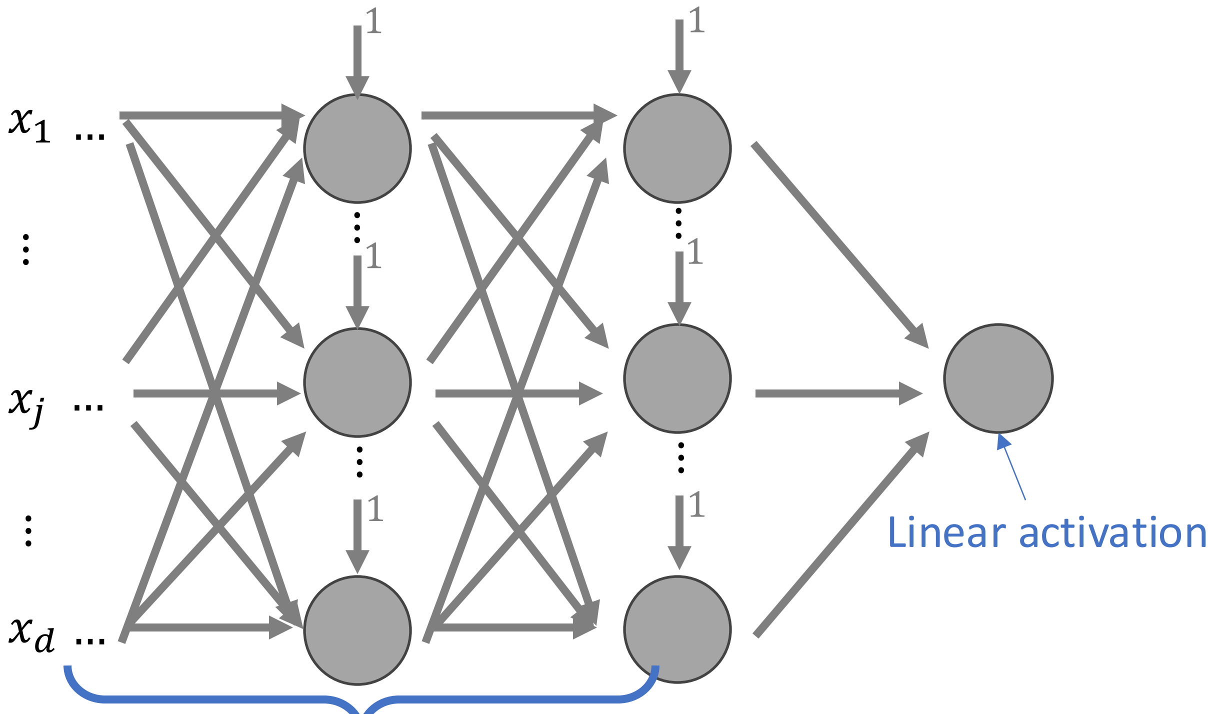
Neural Network v.s. Linear Regression Model

With Feature Engineering

Linear regression relies on manual feature engineering to capture complex patterns, while a multi-layer neural network learns its own feature representations through its layers with activation functions.

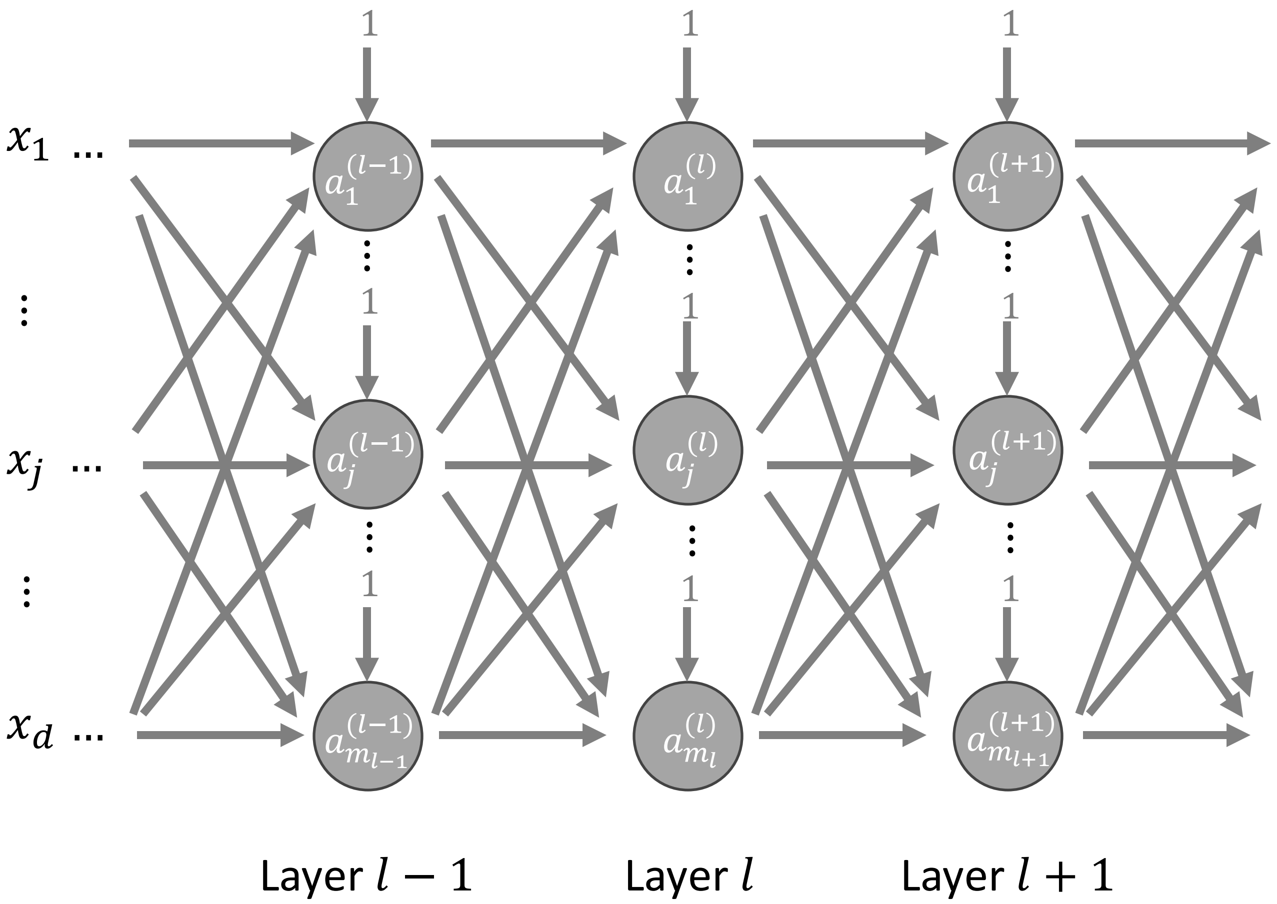


Manual feature engineering in linear regression

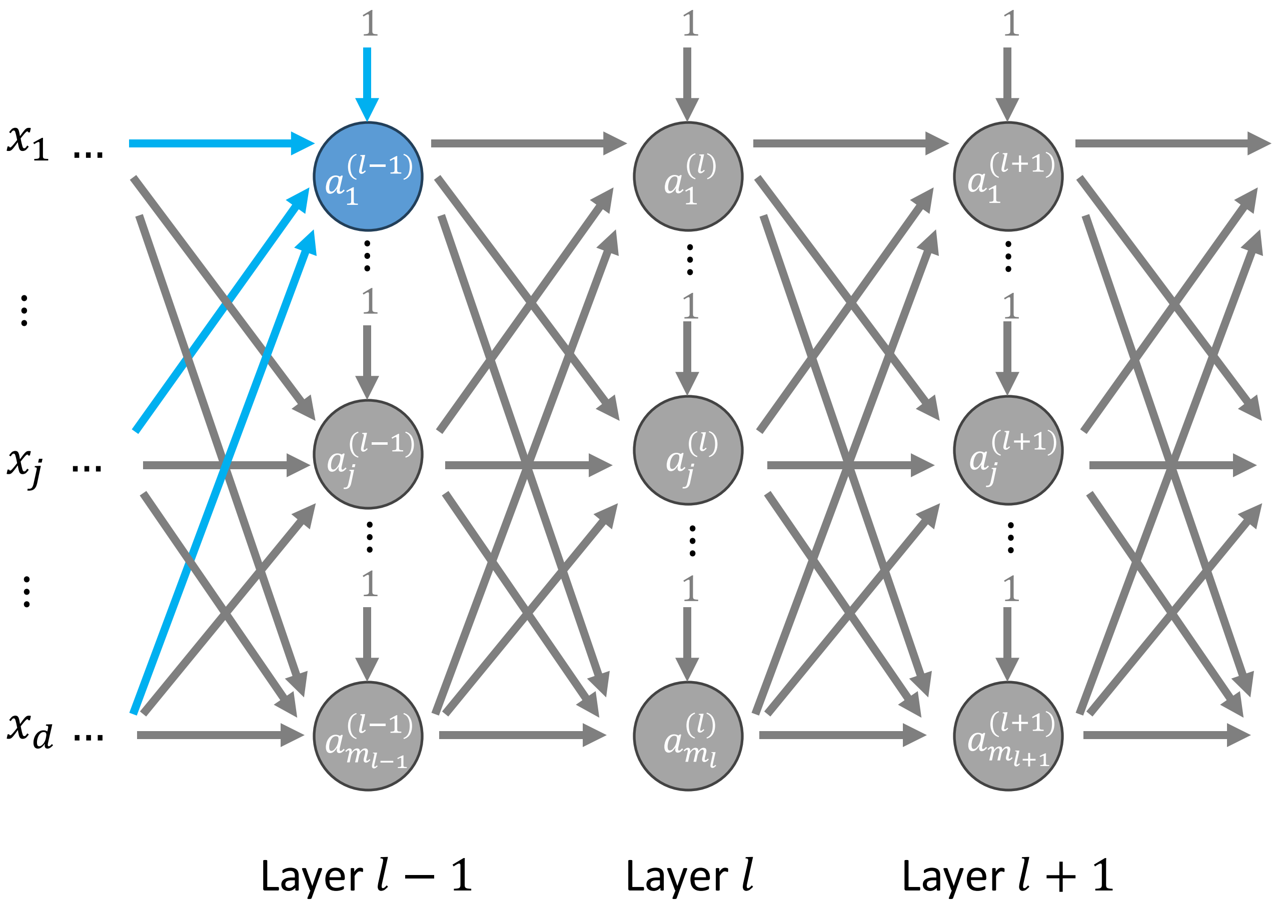


Learned feature engineering in multi-layer neural network

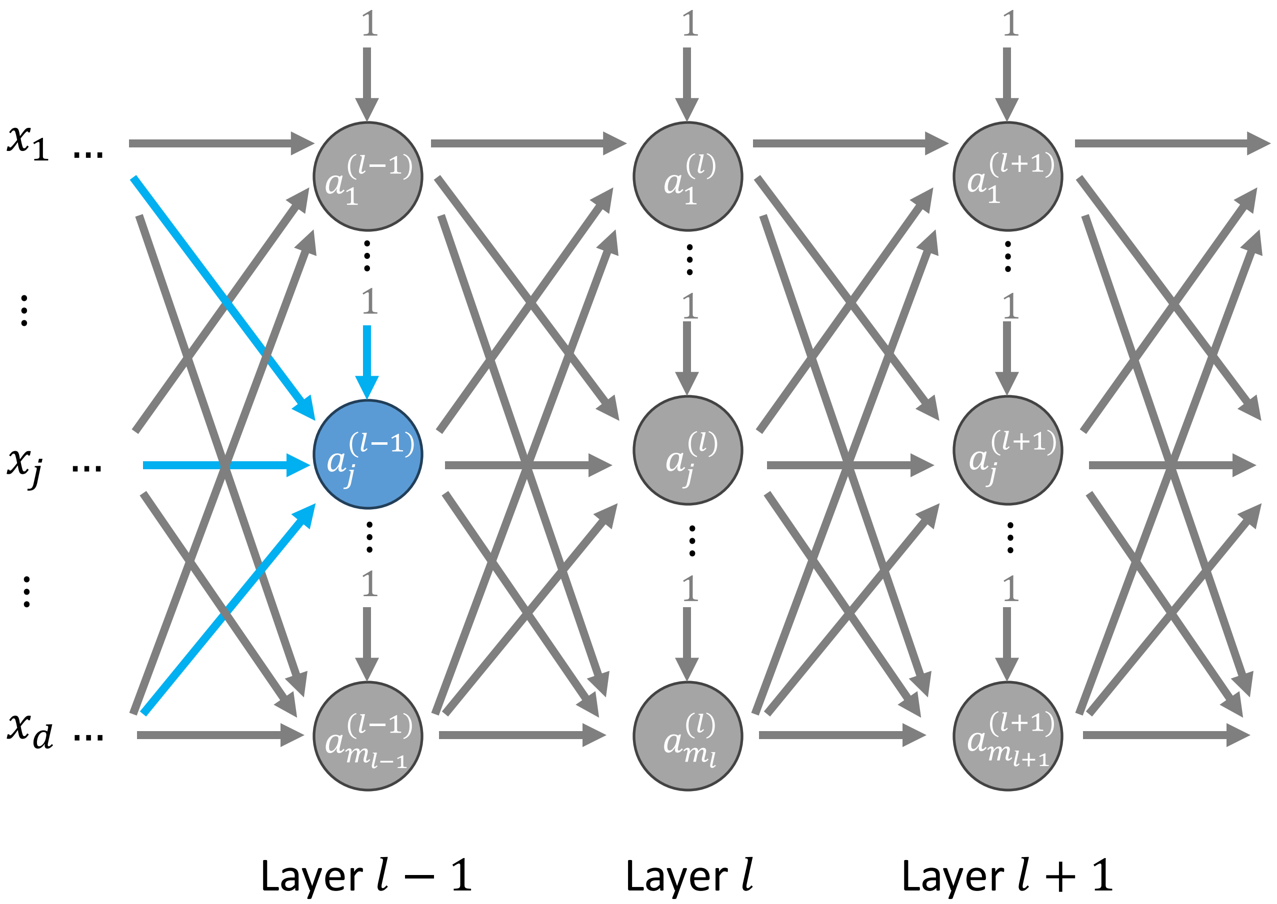
Multi-layer Neural Networks



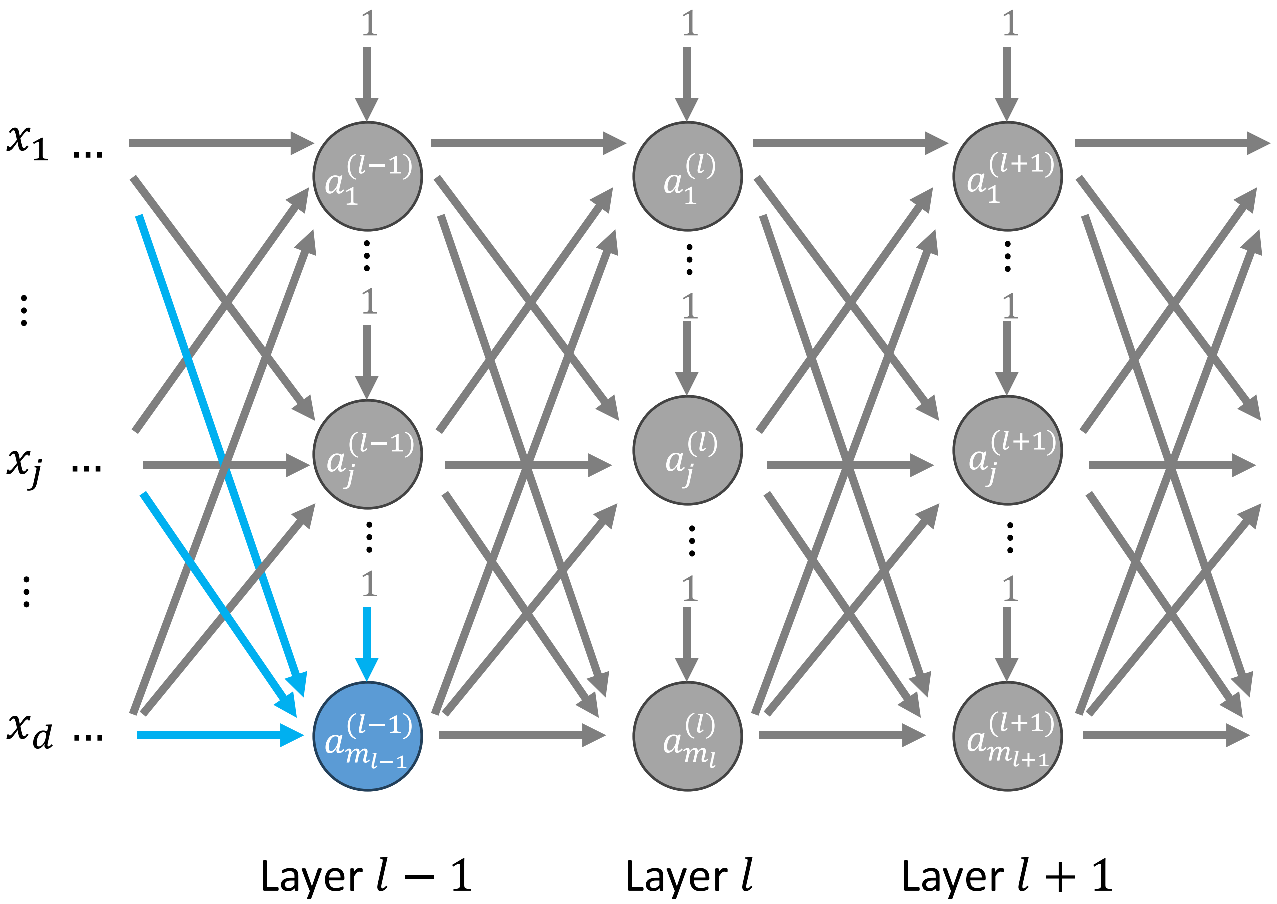
Multi-layer Neural Networks



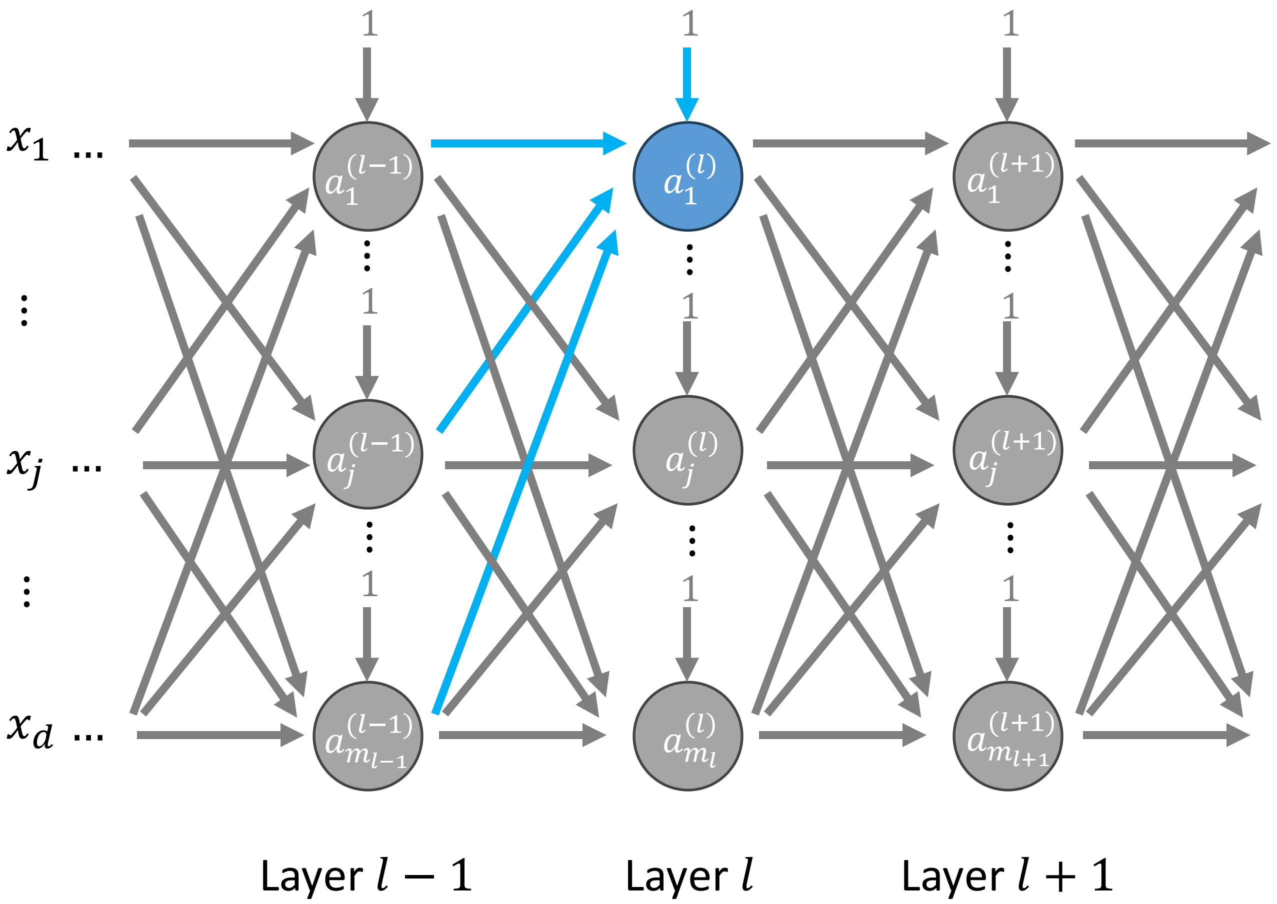
Multi-layer Neural Networks



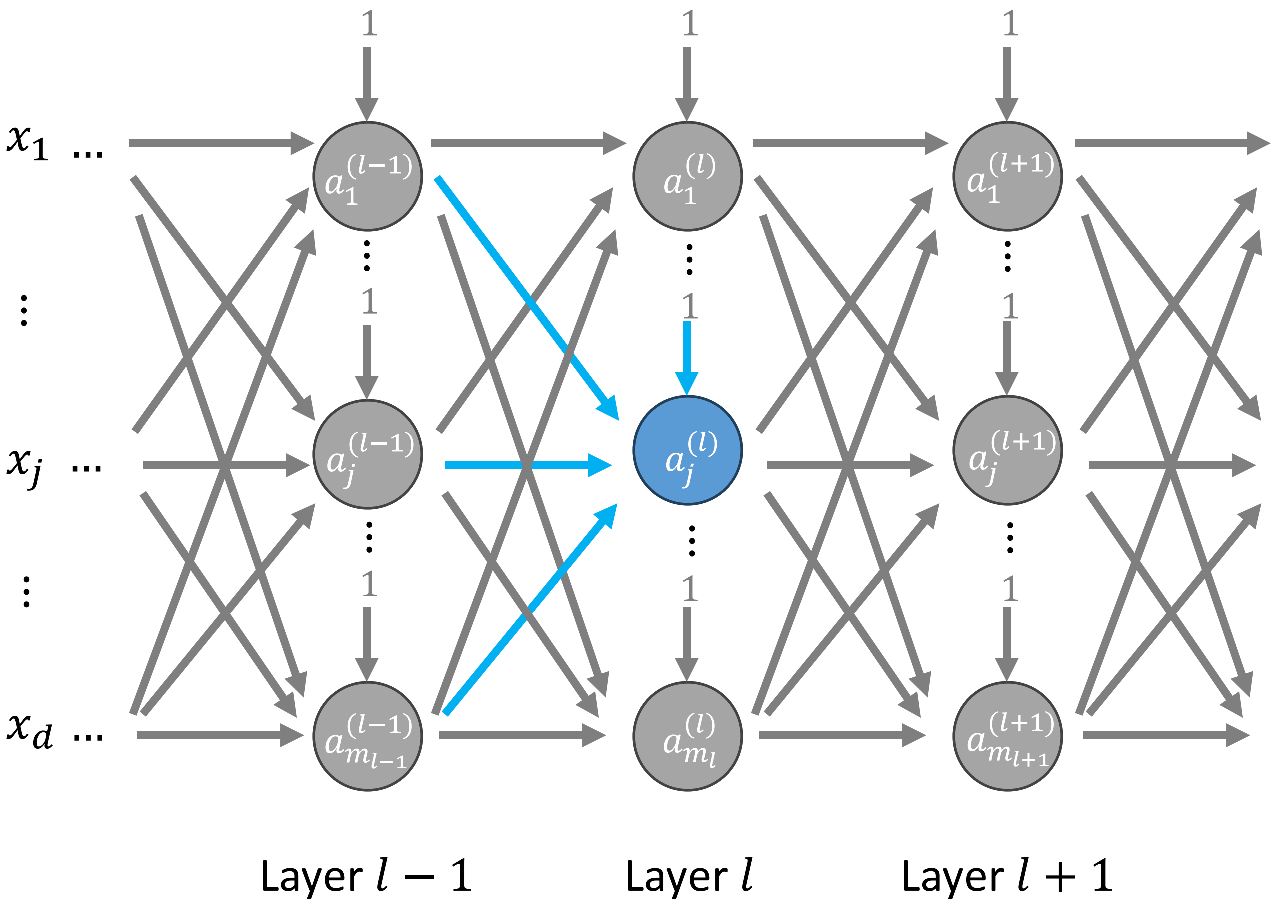
Multi-layer Neural Networks



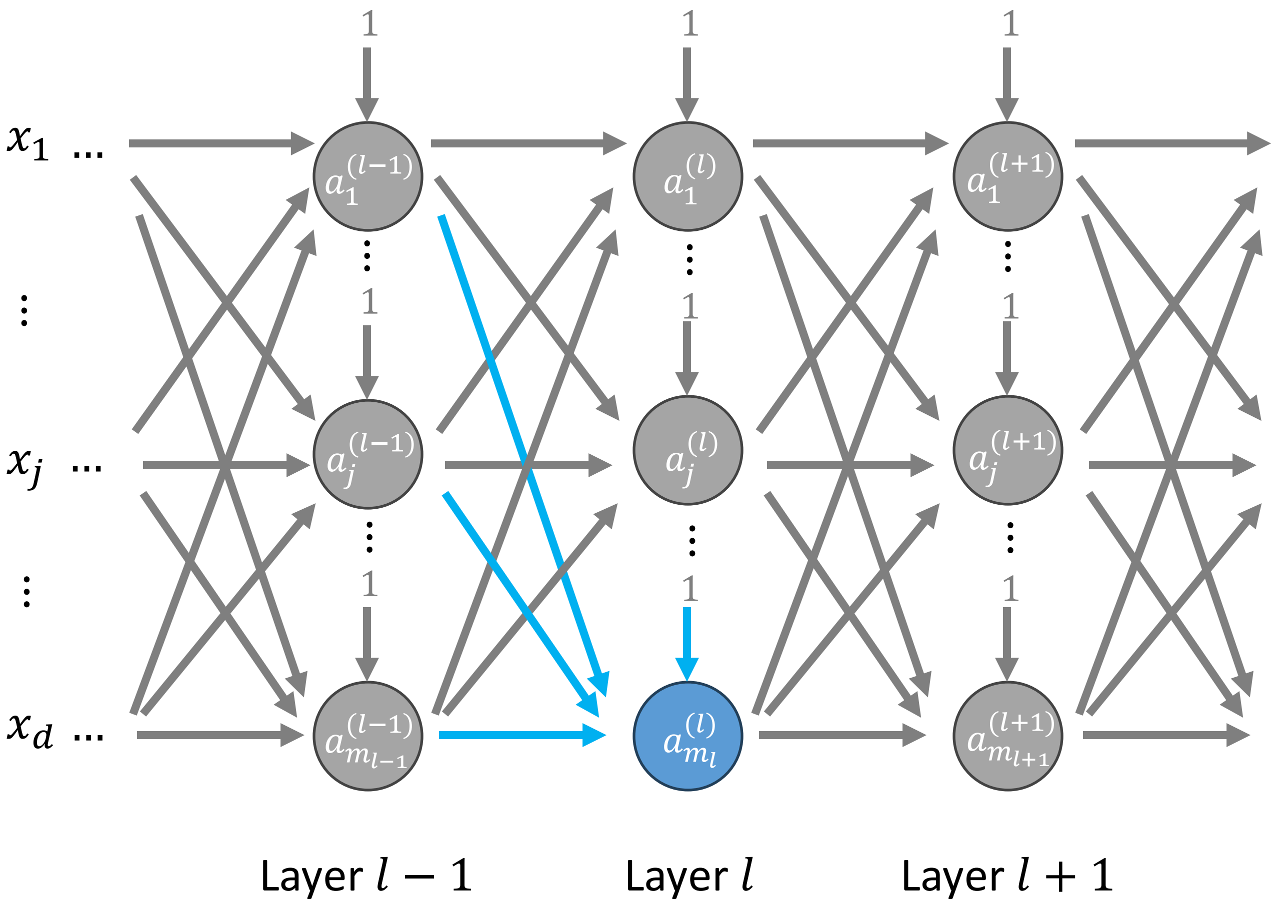
Multi-layer Neural Networks



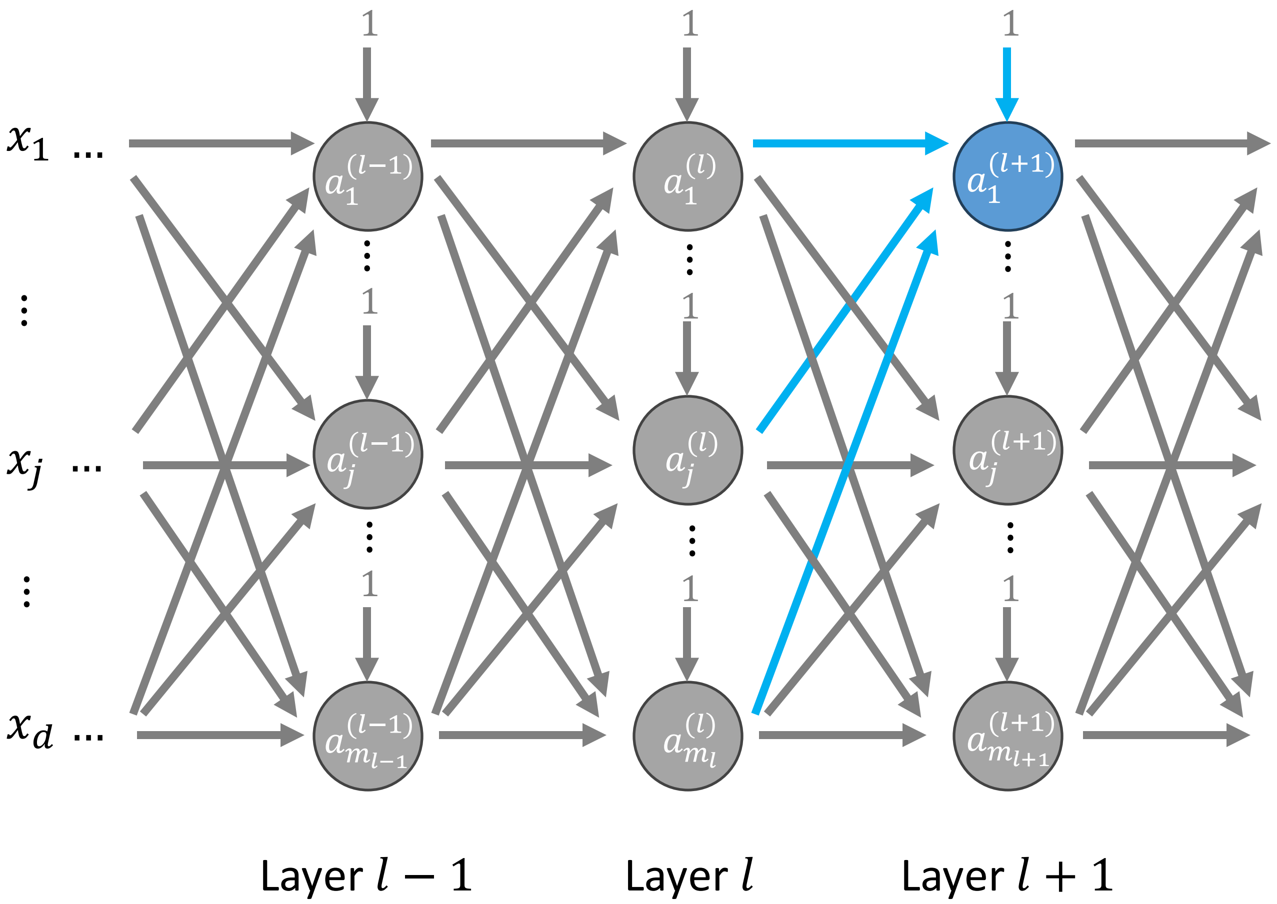
Multi-layer Neural Networks



Multi-layer Neural Networks

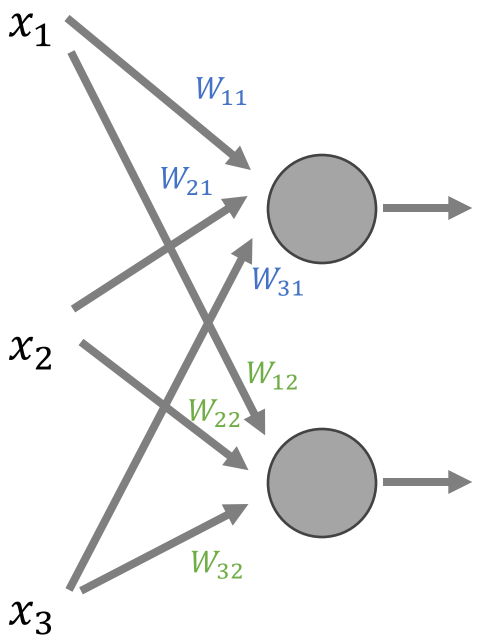


Multi-layer Neural Networks



Neural Networks and Matrix Multiplication

Single-layer



$$\hat{y}_1 = g(W_{11}x_1 + W_{21}x_2 + W_{31}x_3)$$

$$\hat{y}_2 = g(W_{12}x_1 + W_{22}x_2 + W_{32}x_3)$$

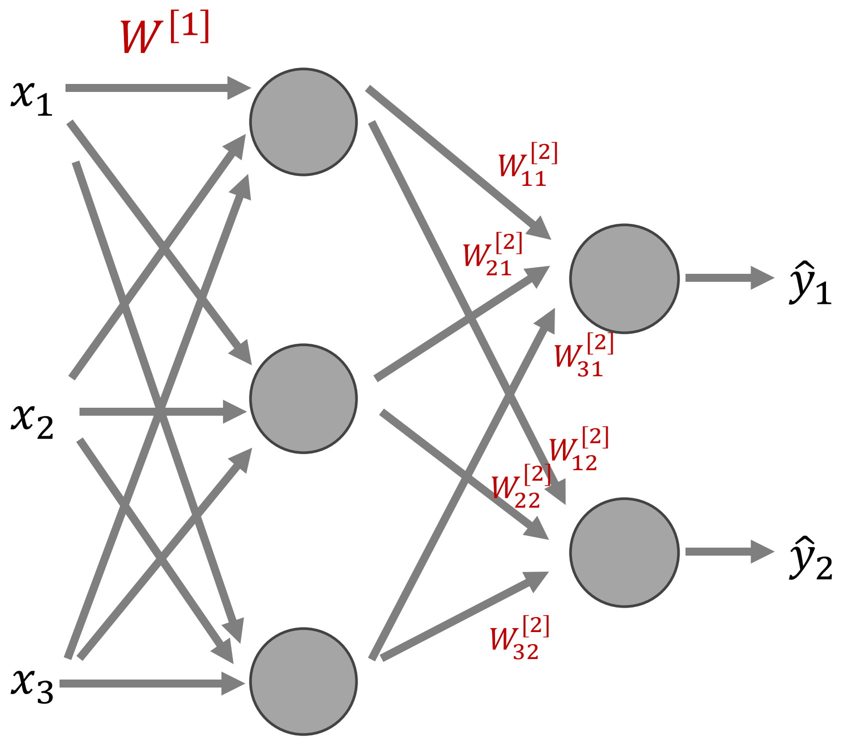
Input (number of weights per neuron / input variables)

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} \quad \hat{\boldsymbol{y}} = g(W^T\boldsymbol{x}) = g\left(\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Output (number of layer’s neurons / output variables)

Neural Networks and Matrix Multiplication

Multi-layer



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

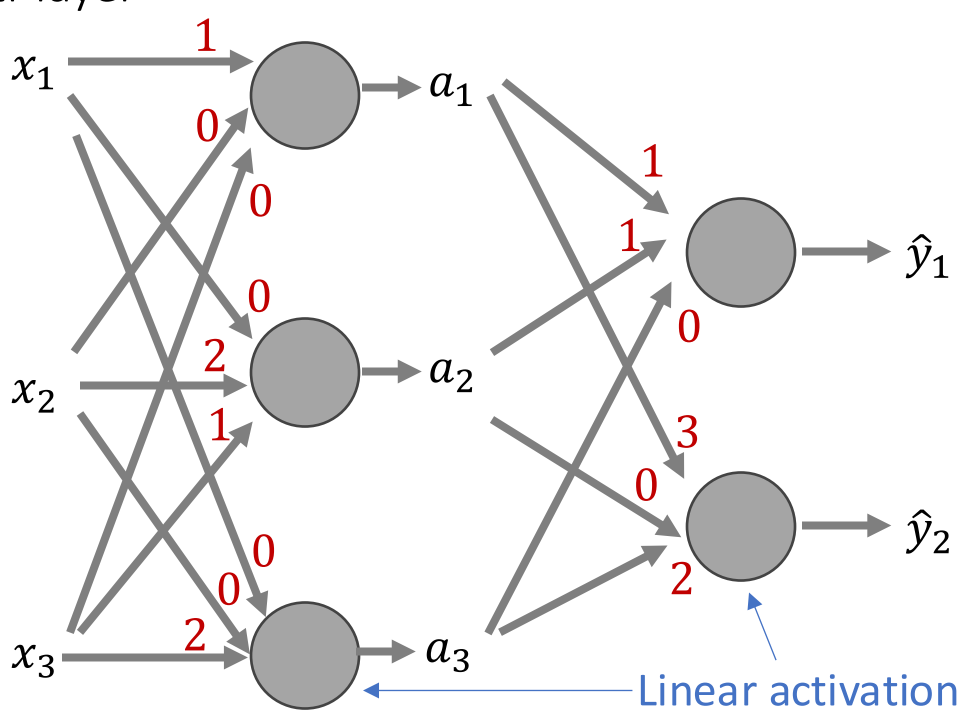
$$W^{[1]} = \begin{bmatrix} W^{[1]}_{11} & W^{[1]}_{12} & W^{[1]}_{13} \\ W^{[1]}_{21} & W^{[1]}_{22} & W^{[1]}_{23} \\ W^{[1]}_{31} & W^{[1]}_{32} & W^{[1]}_{33} \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} W^{[2]}_{11} & W^{[2]}_{12} \\ W^{[2]}_{21} & W^{[2]}_{22} \\ W^{[2]}_{31} & W^{[2]}_{32} \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T}\boldsymbol{x}\right)\right) = g^{[2]}\left(\begin{bmatrix} W^{[2]}_{11} & W^{[2]}_{12} \\ W^{[2]}_{21} & W^{[2]}_{22} \\ W^{[2]}_{31} & W^{[2]}_{32} \end{bmatrix}^T g^{[1]}\left(\begin{bmatrix} W^{[1]}_{11} & W^{[1]}_{12} & W^{[1]}_{13} \\ W^{[1]}_{21} & W^{[1]}_{22} & W^{[1]}_{23} \\ W^{[1]}_{31} & W^{[1]}_{32} & W^{[1]}_{33} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right)\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Neural Networks and Matrix Multiplication

Multi-layer



$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad W^{[1]} = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} & W_{33}^{[1]} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = W^{[1]^T}\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix}$$

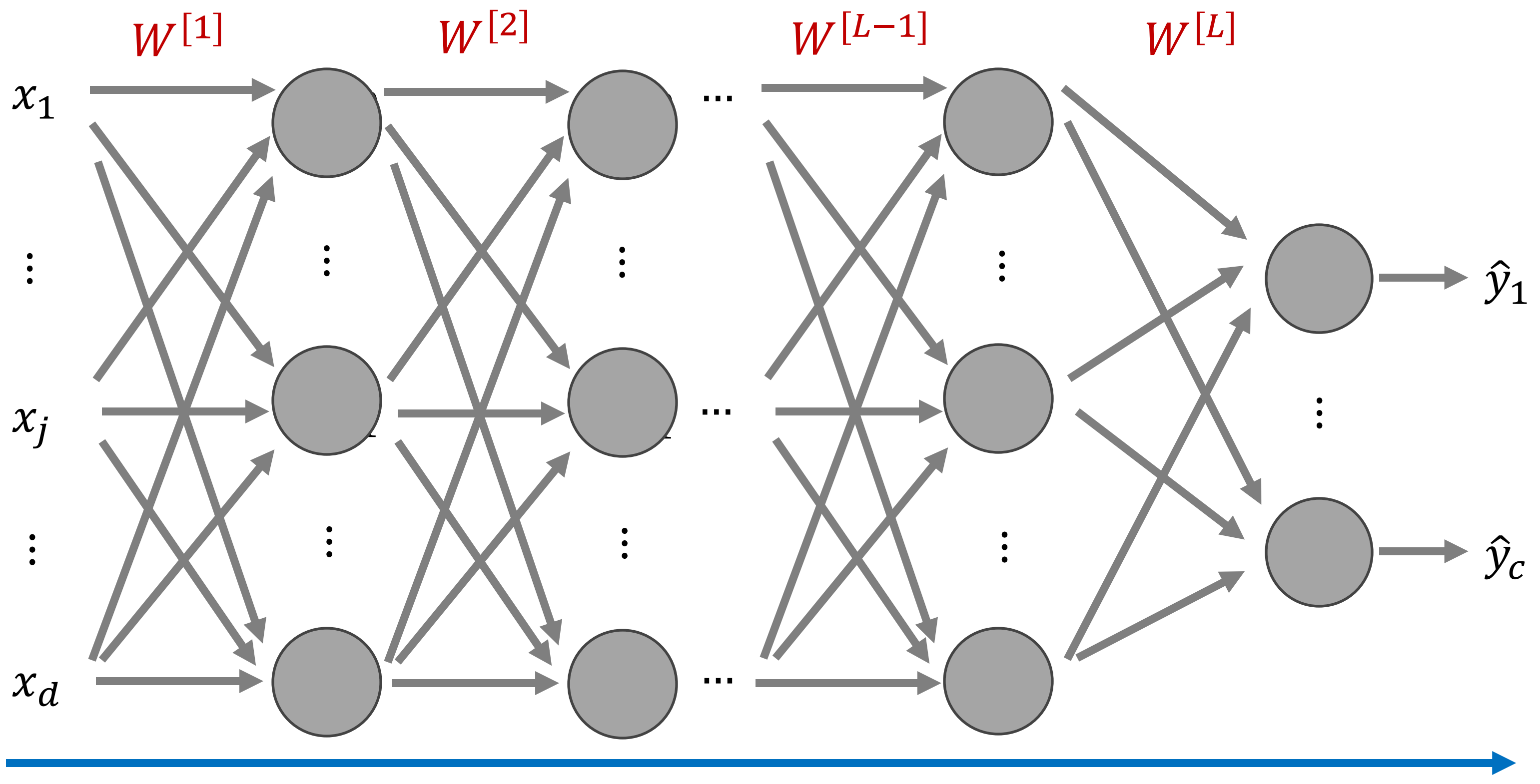
$$W^{[2]} = \begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = W^{[2]^T}\boldsymbol{a} = \begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 3 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 7 \end{bmatrix}$$

$$\hat{\boldsymbol{y}} = g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T}\boldsymbol{x}\right)\right) = g^{[2]}\left(\begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix}^T g^{[1]}\left(\begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} & W_{33}^{[1]} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right)\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Neural Networks and Matrix Multiplication

Multi-layer



The process by which input data passes through a neural network to generate an output prediction.

Forward Propagation

$$\hat{\boldsymbol{y}} = g^{[L]}\left(W^{[L]^T} \dots g^{[L-1]}\left(W^{[L-1]^T} \dots g^{[l]}\left(W^{[l]^T} \dots g^{[2]}\left(W^{[2]^T} g^{[1]}\left(W^{[1]^T}\boldsymbol{x}\right)\right)\right)\right)\right) = \begin{bmatrix} \hat{y}_1 \\ \dots \\ \hat{y}_c \end{bmatrix}$$

Outline

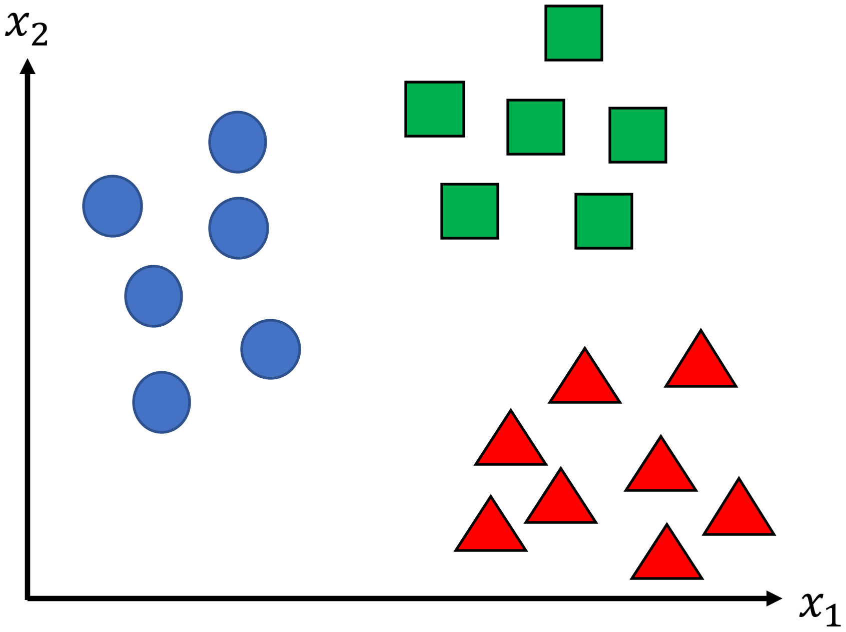
- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Network
  - Neuron
  - AND Gate Modelling
  - XNOR Gate Modelling
  - Single-layer and Multi-layer Neural Networks
- **Multi-class Classification**

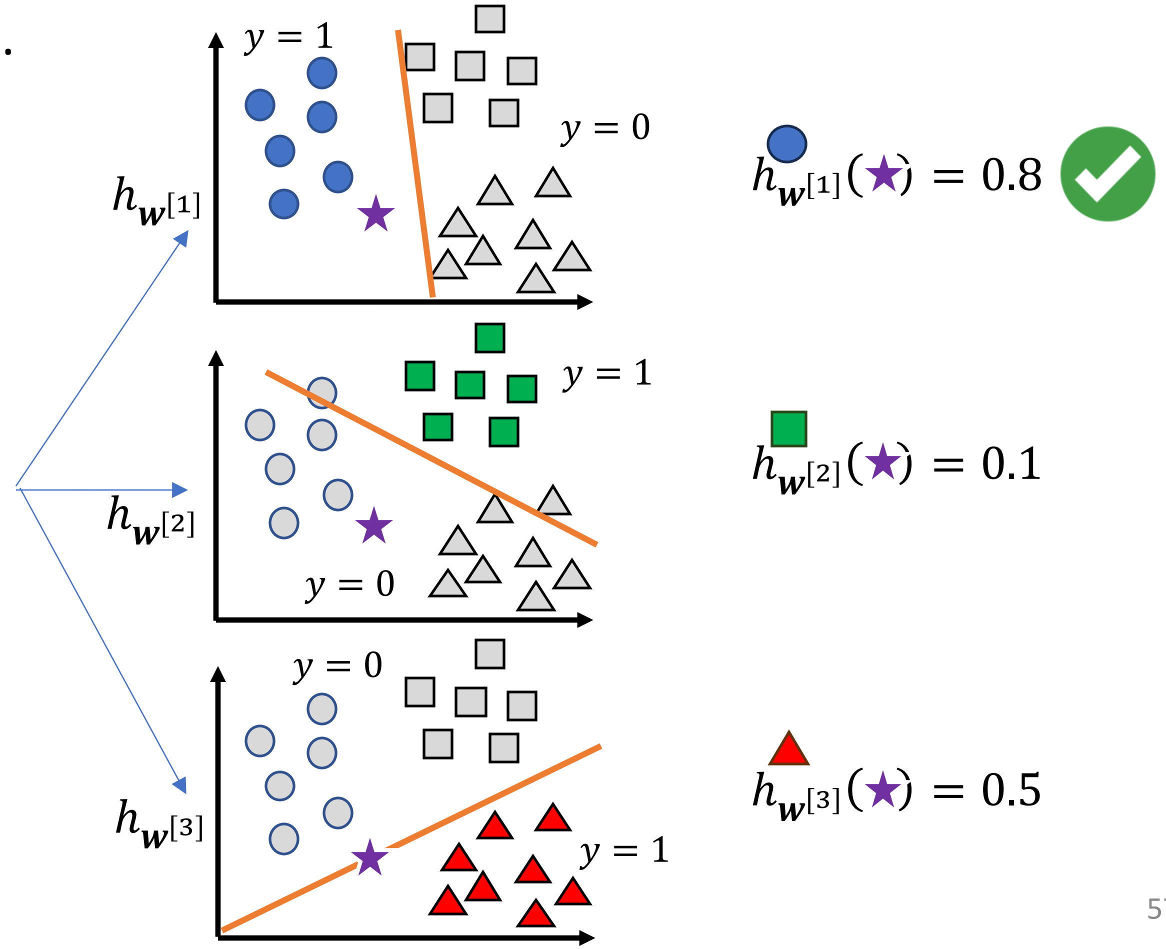
Multi-class Classification

With Logistic Regression Model

Given a set of features (e.g., x_1, x_2),

Predict whether it is belongs to class 1, 2, or 3.

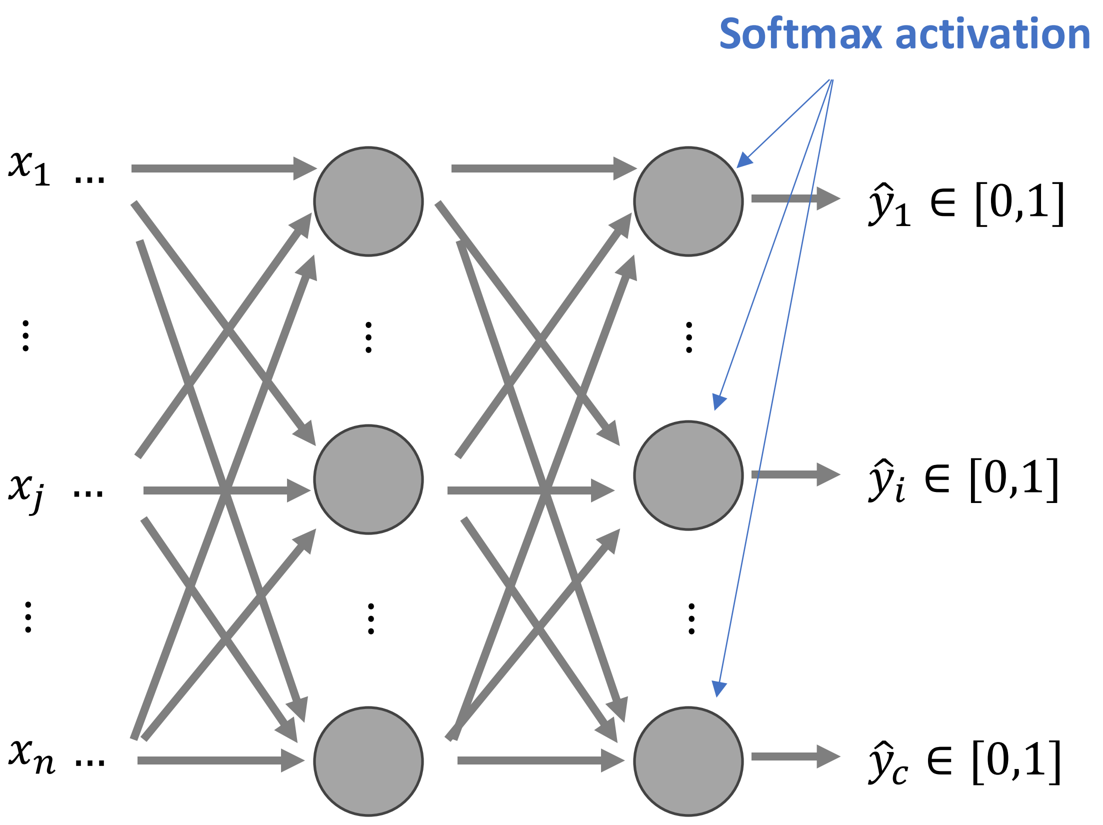




- Fit one classifier per class,
- Fit against all other classes
- Pick **highest probability**

Multi-class Classification

With Neural Network



- Set the number of neurons in the last layer as c
- Predict the probability of each class
- Pick **highest probability**

c-class Classification

Softmax Function

Given a vector $\boldsymbol{z} = [z_1, z_2 \ldots z_c]$, the softmax function computes the output for each z_i as:

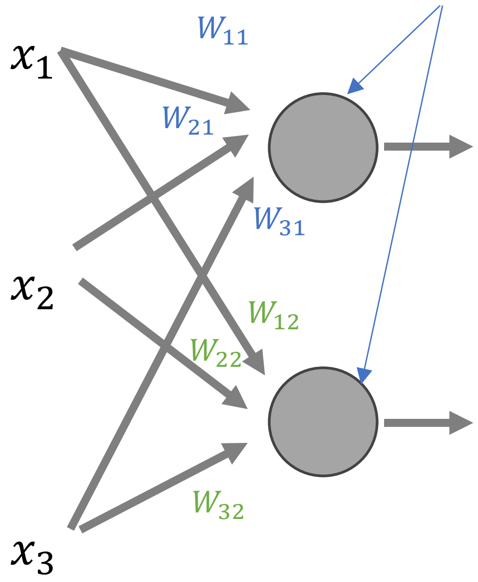
$$g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{c} e^{z_j}} \quad \in [0,1]$$

And $g(z_1) + g(z_2) \ldots + g(z_c) = 1$.

Softmax function can be used to

- Convert raw scores z_i into probabilities
- Ensure that the output values sum to 1.

Softmax activation



$$z_1 = W_{11}x_1 + W_{21}x_2 + W_{31}x_3$$

$$\hat{y}_1 = g(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} \in [0,1]$$

$$z_2 = W_{12}x_1 + W_{22}x_2 + W_{32}x_3$$

$$\hat{y}_2 = g(z_2) = \frac{e^{z_2}}{e^{z_1} + e^{z_2}} \in [0,1]$$

$$g(z_1) + g(z_2) = 1$$

Summary

- Perceptron
  - Biological inspiration: brain, neural network, neuron
  - Perceptron Learning Algorithm:
    - $w \leftarrow w + \gamma\left(y^{(j)} - \hat{y}^{(j)}\right)x^{(j)}$ on a misclassified instance
- Neural Networks
  - Linear regression model: A neuron with linear activation function
  - Logistic regression model: A neuron with sigmoid activation function
  - Neurons can be used to transform features
  - Multi-layer Neural Networks
- Multi-class Classification
  - Softmax activation function should be applied.

Coming Up Next Week

- **More on Neural Networks**
  - Backpropagation
  - Convolution Neural Networks
  - …

To Do

- **Lecture Training 9**
  - +250 Free EXP
  - +100 Early bird bonus