CS2109S: Introduction to AI and Machine Learning

Lecture 8:
Intro to Neural Networks

13 October 2023

Recap

- Overfitting
- Regularization
    - Linear and logistic regression
- Support Vector Machine (SVM)
    - Hard-margin SVM
    - Soft-margin SVM
- Kernel
    - SVM with Kernel Trick

Logistic Regression / SVM
With x as features

Logistic Regression / SVM
With $\boldsymbol{\phi}(\boldsymbol{x})$ as features

SVM with Kernel Trick
With $\boldsymbol{\phi}$(x) mapping to
finite-dimensional
features

SVM with Kernel Trick
With $\boldsymbol{\phi}$(x) mapping to
infinite-dimensional
features

ERRATA

Linear Regression w/ Regz: Gradient Descent

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_w(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{i=1}^{n} w_i^2\right]$$

$\frac{\partial J(w_0, w_{1,\dots})}{\partial w_j}$

Optimization goal: $\min_{w} J(w)$

$\frac{\partial J(w_0, w_{1,\dots})}{\partial w_j}$

Repeat {

$$w_0 := w_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right).x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m}\left[\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right).x_1^{(i)} + \lambda w_1\right]$$

$$\vdots$$

$$w_n := w_n - \alpha \frac{1}{m}\left[\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right).x_n^{(i)} + \lambda w_n\right]$$

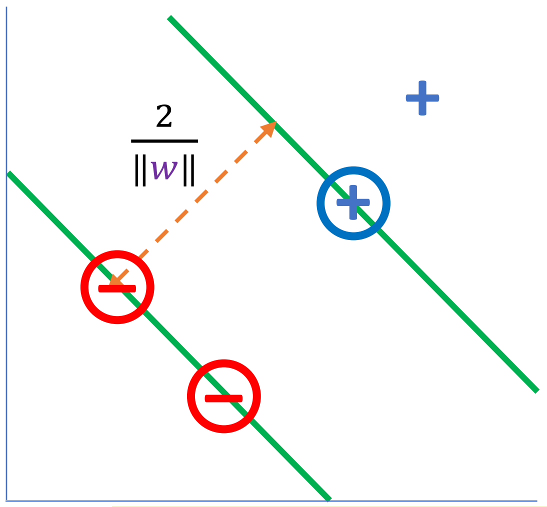
}

Why does this work?

ERRATA

Support Vector Machines (SVM)

3



Objective:

$$\max \frac{2}{\|w\|} \rightarrow \max \frac{1}{\|w\|} \rightarrow \min \|w\| \rightarrow \min \frac{1}{2}\|w\|^2$$ "Maximize gap"

$$\text{s.t. } \bar{y}^{(i)}\left(w \cdot x^{(i)} + b\right) - 1 \geq 0$$ "Classify correctly"

Assume $x^{(i)}$ in margin

$$\boxed{b = \bar{y}^{(i)} - w \cdot x^{(i)}}$$

Objective (Lagrange Dual):

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_i \alpha^{(i)}\left[\bar{y}^{(i)}\left(w \cdot x^{(i)} + b\right) - 1\right], \forall_i\ \alpha^{(i)} \geq 0$$

$$\max_{\alpha} \min_{w,b} L(w, b, \alpha)$$

Minimized when gradient = 0

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_i \alpha^{(i)} \bar{y}^{(i)} x^{(i)} = 0 \quad \Rightarrow \quad \boxed{w = \sum_i \alpha^{(i)} \bar{y}^{(i)} x^{(i)}}$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_i \alpha^{(i)} \bar{y}^{(i)} = 0$$

Samples with non-zero $\alpha^{(i)}$ = support vectors

… a few math later …

Maximize $\longrightarrow$ $$L(\alpha) = \sum_i \alpha^{(i)} - \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)}\bar{y}^{(i)}\bar{y}^{(j)} x^{(i)} \cdot x^{(j)}$$

ERRATA

SVM with Kernel Trick

From Before: $\boxed{w = \sum_i \alpha^{(i)} \hat{y}^{(i)} x^{(i)}}$

Objective:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_i \alpha^{(i)}\left[\bar{y}^{(i)}\left(w \cdot \phi\left(x^{(i)}\right) + b\right) - 1\right]$$

… a few math later …

$$L(\alpha) = \sum_i \alpha^{(i)} - \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)}\bar{y}^{(i)}\bar{y}^{(j)}\phi\left(x^{(i)}\right) \cdot \phi\left(x^{(j)}\right)$$

$$L(\alpha) = \sum_i \alpha^{(i)} - \frac{1}{2}\sum_i \sum_j \alpha^{(i)}\alpha^{(j)}\bar{y}^{(i)}\bar{y}^{(j)}K\left(x^{(i)}, x^{(j)}\right)$$

Decision Rule:

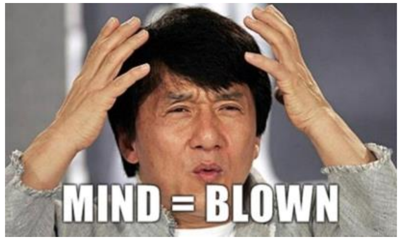
$$w \cdot \phi(x) + b \geq 0 \text{ then } +$$

$$\sum_i \alpha^{(i)} \hat{y}^{(i)} \phi\left(x^{(i)}\right) \cdot \phi(x) + b \geq 0 \text{ then } +$$

$$\sum_i \alpha^{(i)} \hat{y}^{(i)} \quad K\left(x^{(i)}, x\right) \quad + b \geq 0 \text{ then } +$$

There is no need to compute the transformed features **explicitly**!

Can have SVM with **infinite-dimensional** features!



Outline

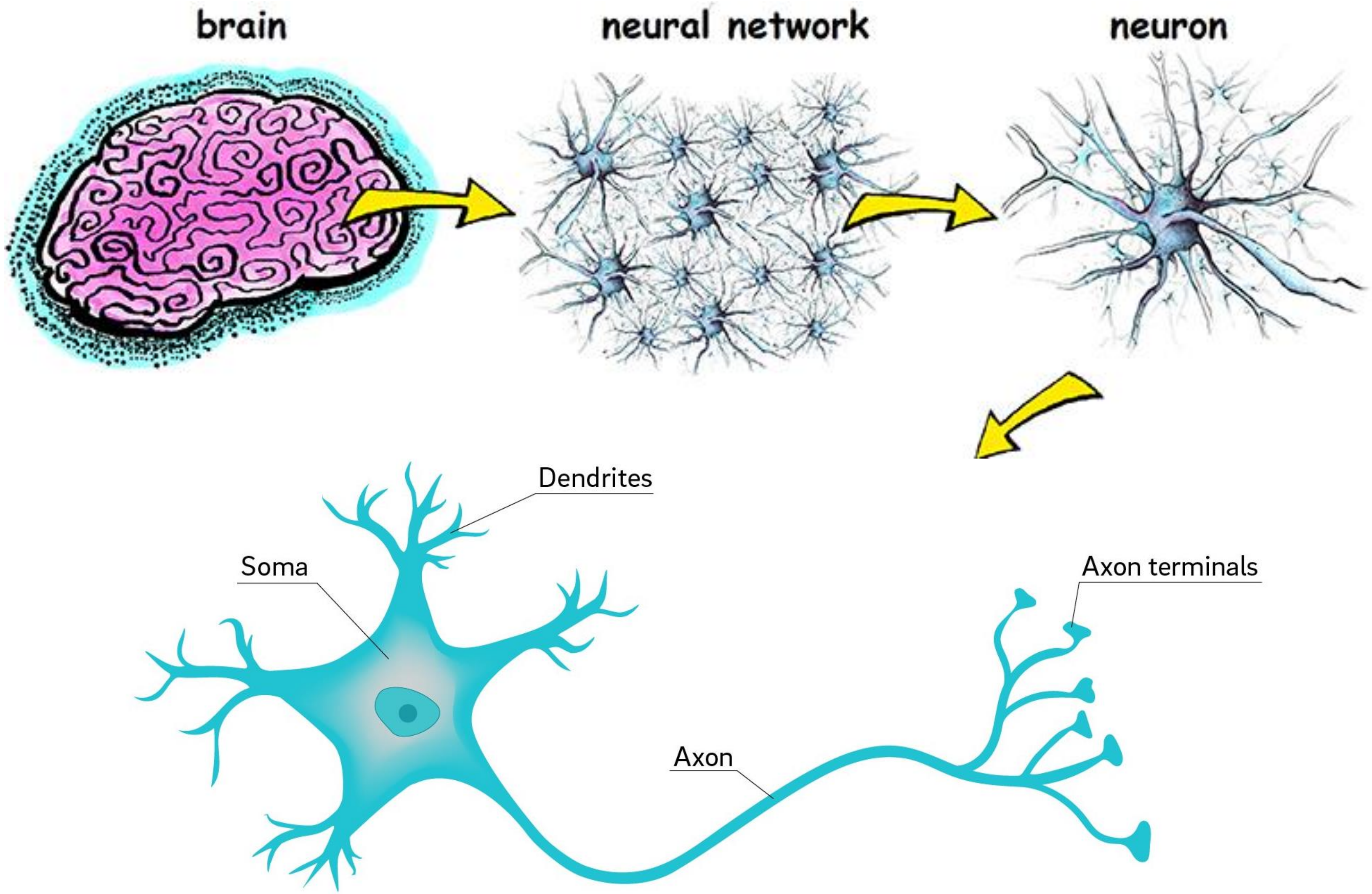
- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Networks
  - Single-layer Neural Networks
  - Multi-layer Neural Networks
  - Regression and Classification
- Neural Networks with Gradient Descent
- Neural Networks vs Other Models

Outline

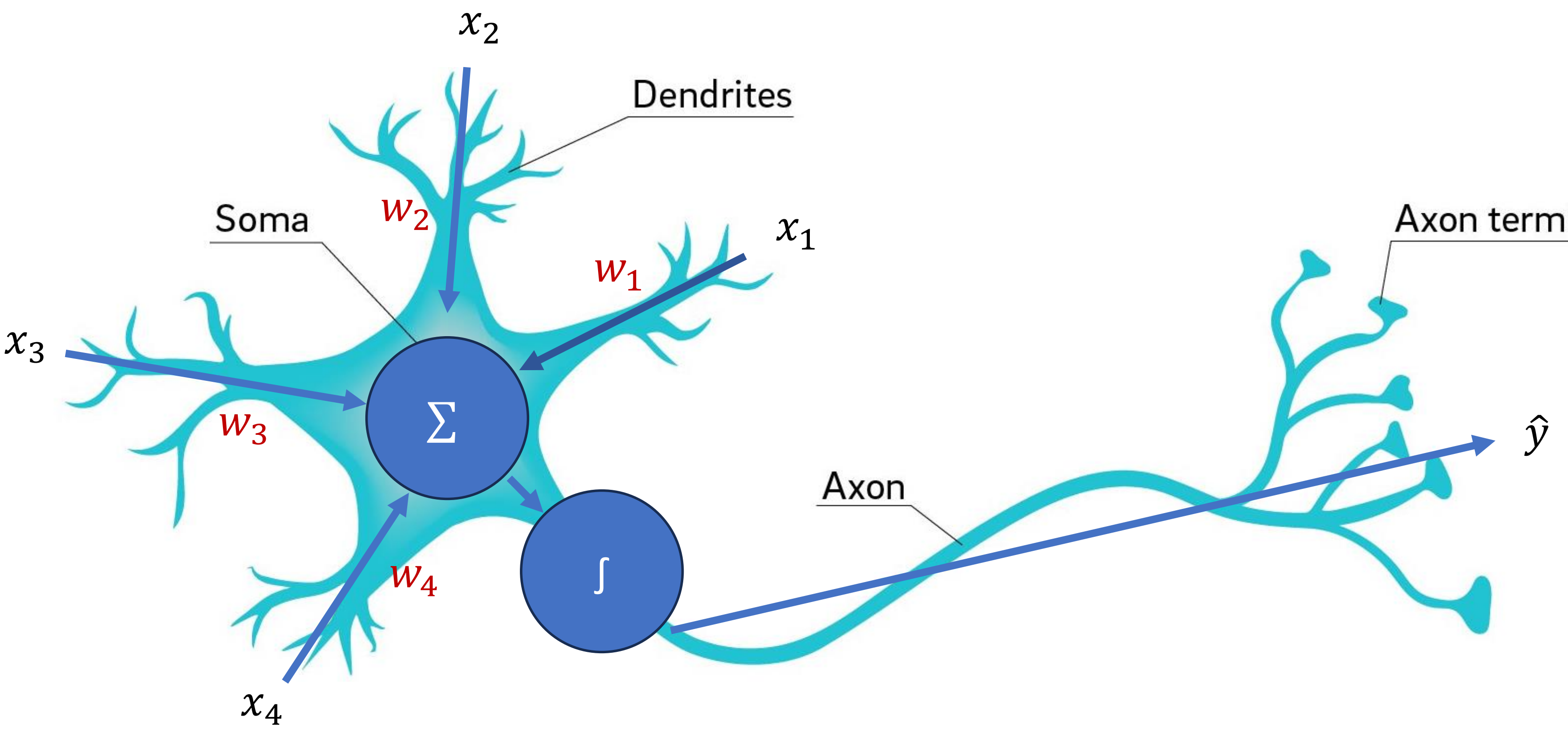
- **Perceptron**
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Networks
  - Single-layer Neural Networks
  - Multi-layer Neural Networks
  - Regression and Classification
- Neural Networks with Gradient Descent
- Neural Networks vs Other Models

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

Brain and Neuron



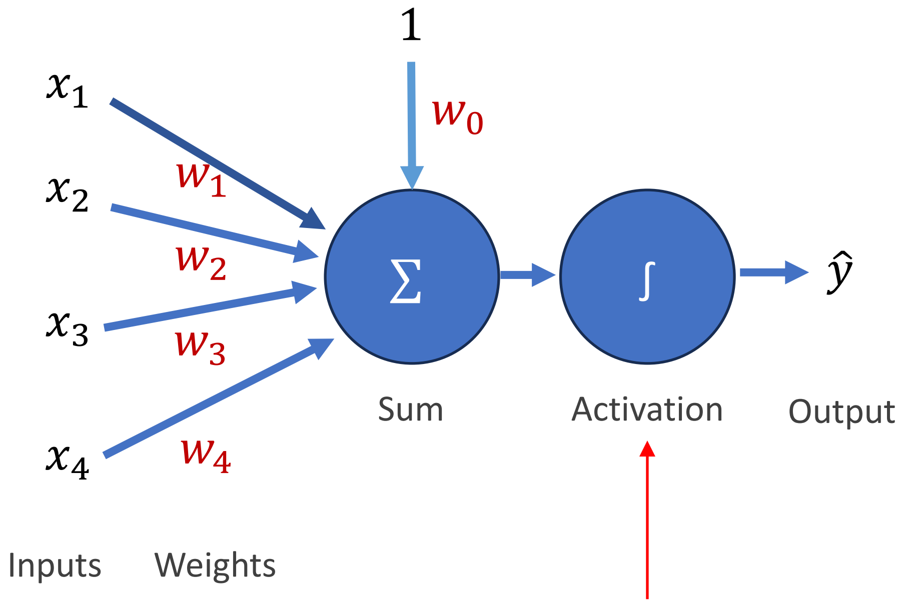
Neuron



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

Frank Rossenblatt (1943)

Perceptron



$$\hat{y} = h_w(x) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

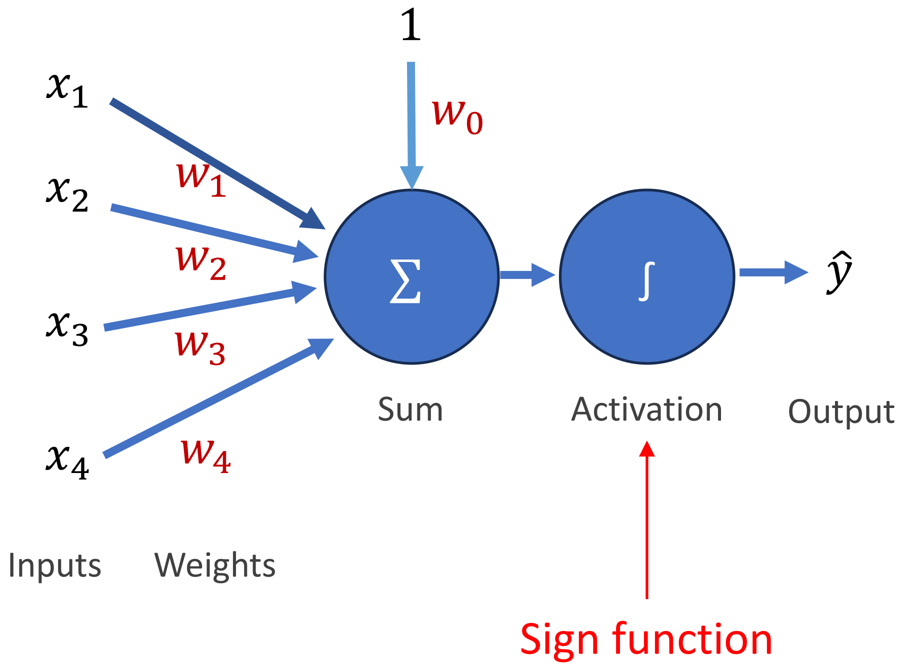
Sign function

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

Frank Rossenblatt (1943)

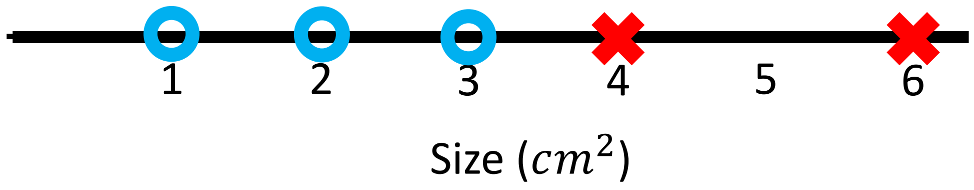
Perceptron: An Example

$$\hat{y} = h_w(x) = g(\sum_{i=0}^{n} w_i x_i)$$



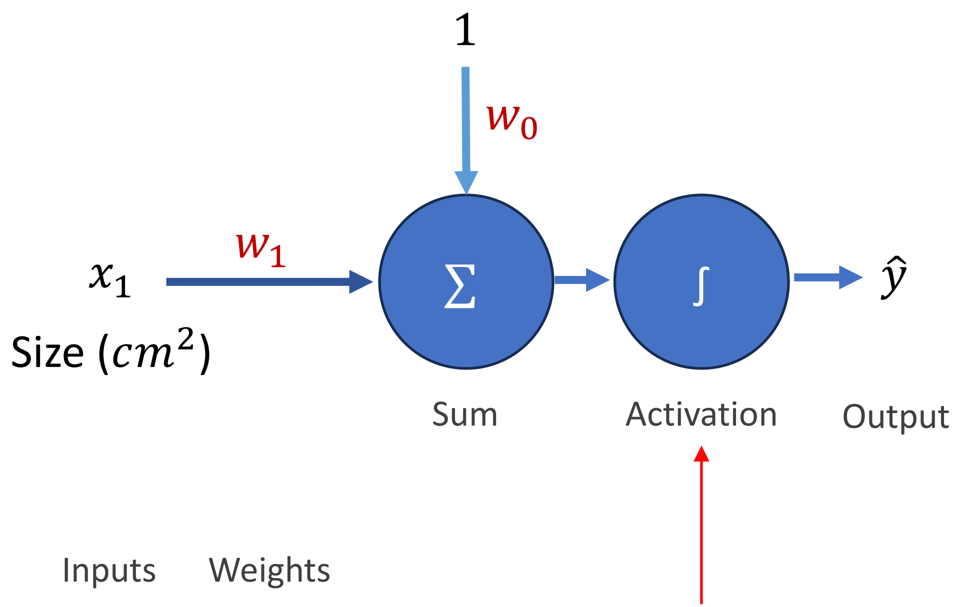
$$g(z) = \begin{cases} +1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

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



Perceptron: An Example

$$\hat{y} = h_w(x) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$



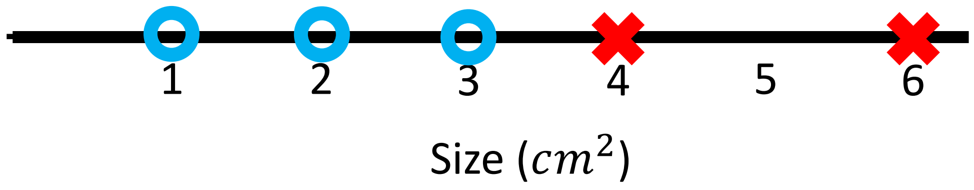
Sign function

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

$$h_w(x) = \begin{cases} +1 & \text{if } w_0 + w_1 x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$h_w(x) = \begin{cases} +1 & \text{if } -3.5 + 1x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

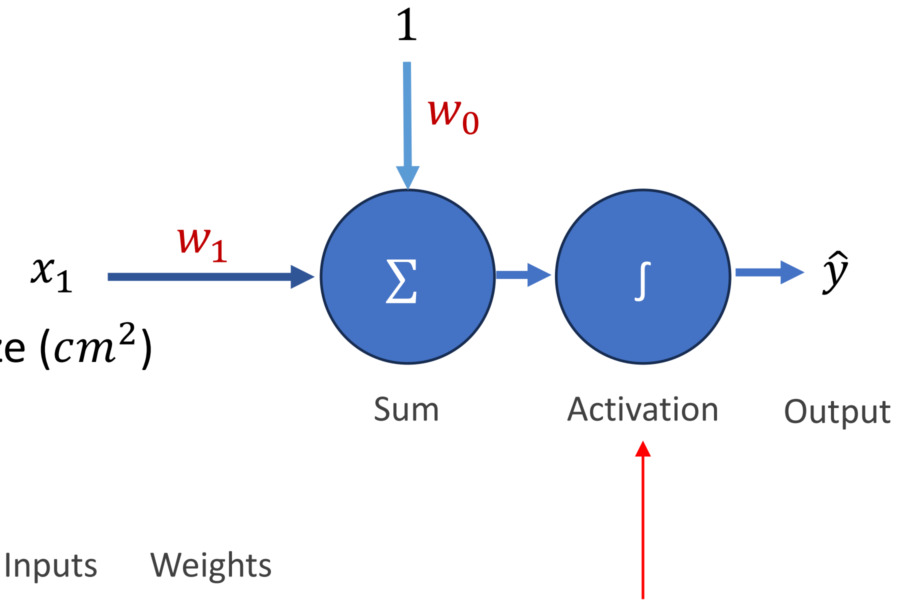
Cancer prediction: benign (-1), malignant (1)



Frank Rossenblatt (1943)

Perceptron: An Example

$$\hat{y} = h_w(x) = g(\sum_{i=0}^{n} w_i x_i)$$



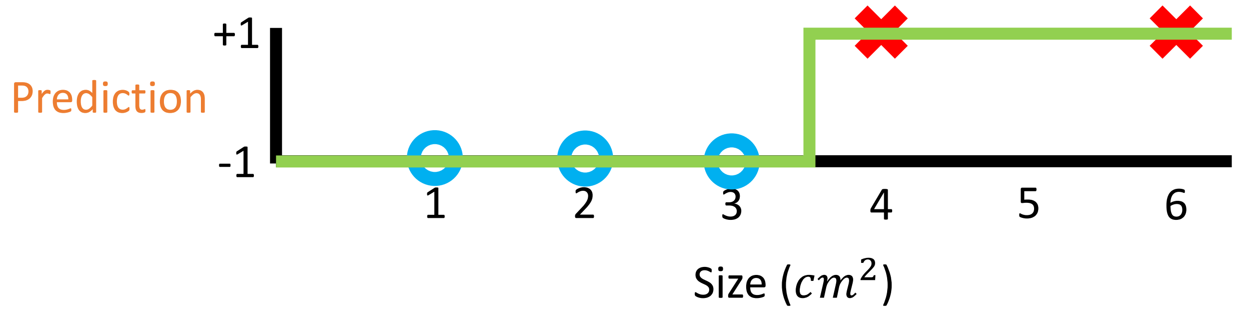
$$h_w(x) = \begin{cases} +1 & \text{if } w_0 + w_1 x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$h_w(x) = \begin{cases} +1 & \text{if } -3.5 + 1x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

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

Sign function

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



How do we learn w while g is not differentiable?

Frank Rossenblatt (1943)

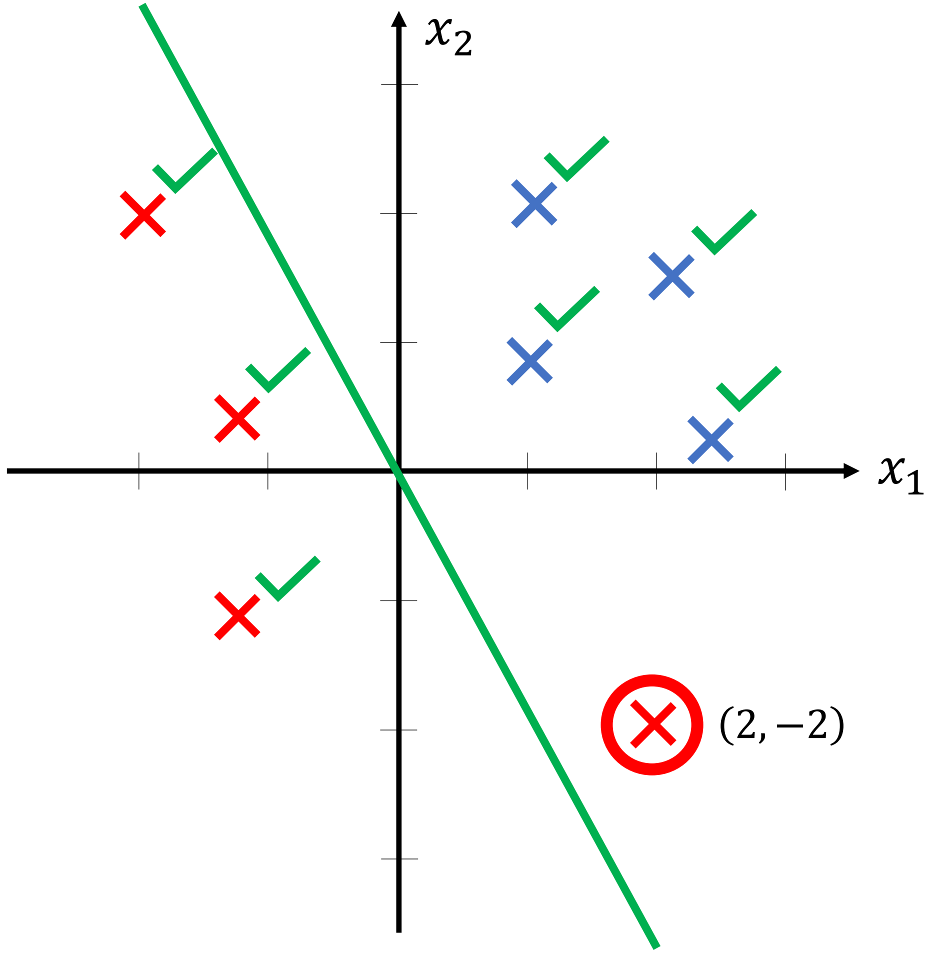
Perceptron Learning Algorithm

- Initialize $\forall_i w_i = 0$
- Loop (until convergence or max steps reached)
  - For each instance $\left(x^{(i)}, y^{(i)}\right)$, classify $\hat{y}^{(i)} = h_w(x^{(i)})$
  - Select one misclassified instance $\left(x^{(j)}, y^{(j)}\right)$
  - Update weights: $w \leftarrow w + \gamma\left(y^{(j)} - \hat{y}^{(j)}\right)x^{(j)}$

Learning rate

Frank Rossenblatt (1943)

Perceptron Learning Algorithm: An Example



Sign function

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

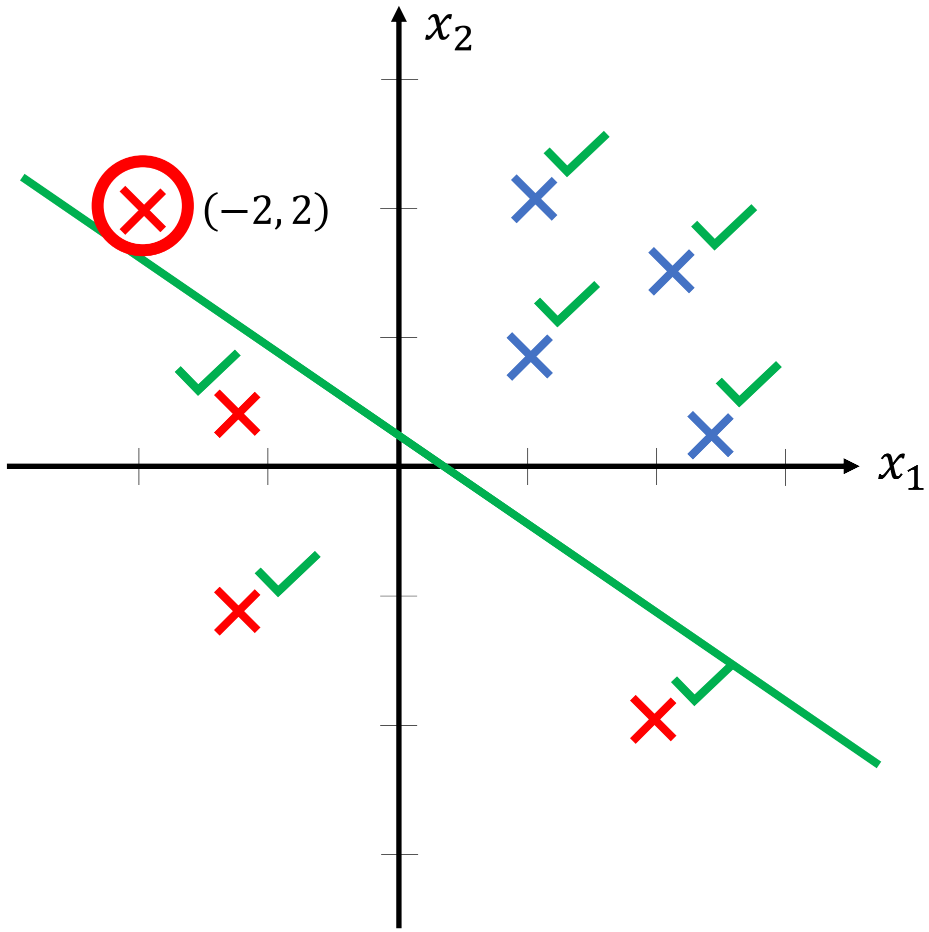
$$\hat{y} = h_w(x) = g(\,w_0 + w_1 x_1 + w_2 x_2)$$
$$= g(0 + 1x_1 + 0.5x_2)$$

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

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

New $h_w(x) = g(-0.2 + 0.6x_1 + 0.9x_2)$

Perceptron Learning Algorithm: An Example



Sign function

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

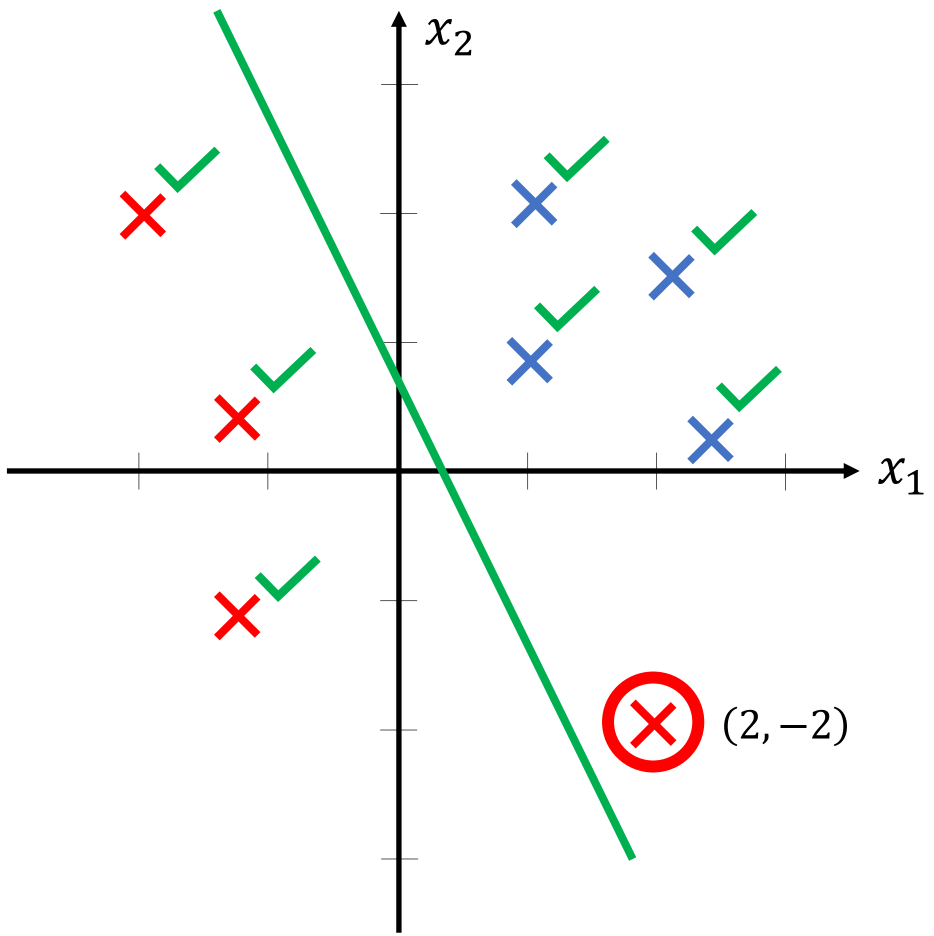
$$\hat{y} = h_w(x) = g(\, w_0 + w_1 x_1 + w_2 x_2)$$
$$= g(-0.2 + 0.6 x_1 + 0.9 x_2)$$

$$w \leftarrow \quad w \quad + \gamma\left(y^{(i)} - \hat{y}^{(i)}\right) x^{(i)}$$
$$\begin{bmatrix} -0.2 \\ 0.6 \\ 0.9 \end{bmatrix} + 0.1(-1-1)\begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.4 \\ 1 \\ 0.5 \end{bmatrix}$$

New $h_w(x) = g(-0.4 + 1 x_1 + 0.5 x_2)$

Frank Rossenblatt (1943)

Perceptron Learning Algorithm: An Example



Sign function

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

$$\hat{y} = h_w(x) = g(\,w_0 + w_1 x_1 + w_2 x_2)$$
$$= g(-0.4 + 1x_1 + 0.5x_2)$$

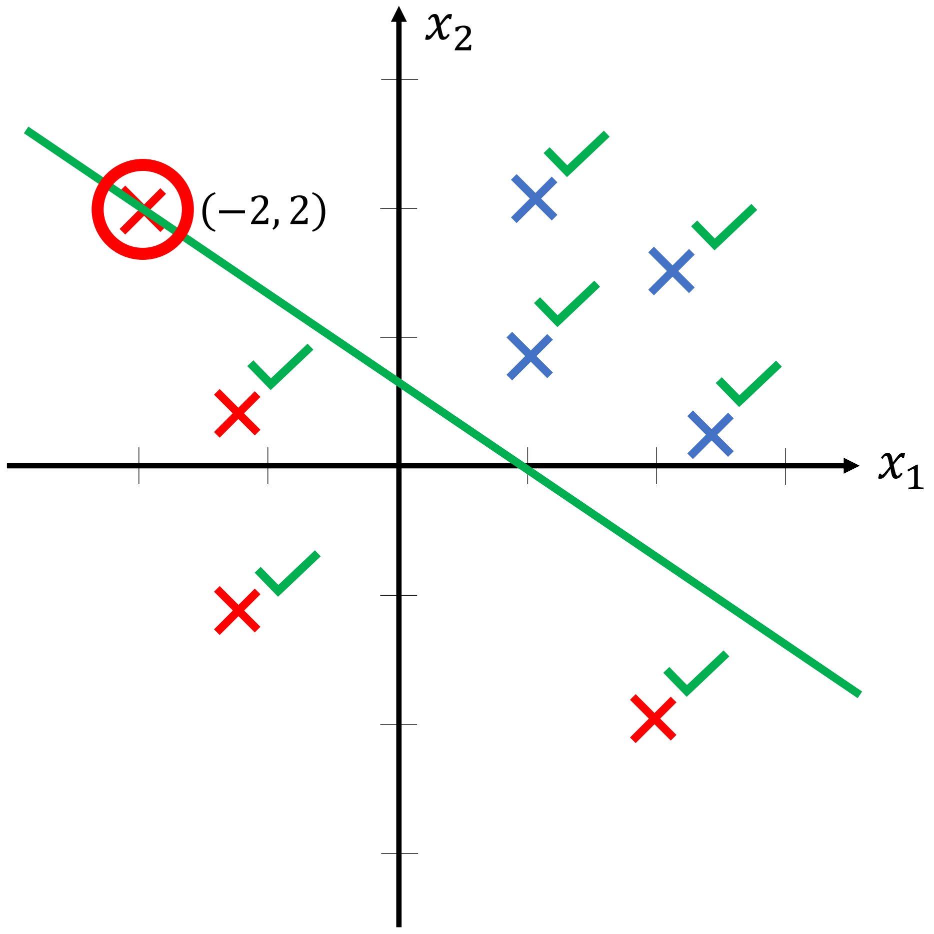
$$w \leftarrow \quad w \quad + \gamma\left(y^{(i)} - \hat{y}^{(i)}\right) x^{(i)}$$

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

New $h_w(x) = g(-0.6 + 0.6x_1 + 0.9x_2)$

Frank Rossenblatt (1943)

Perceptron Learning Algorithm: An Example



Sign function

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

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

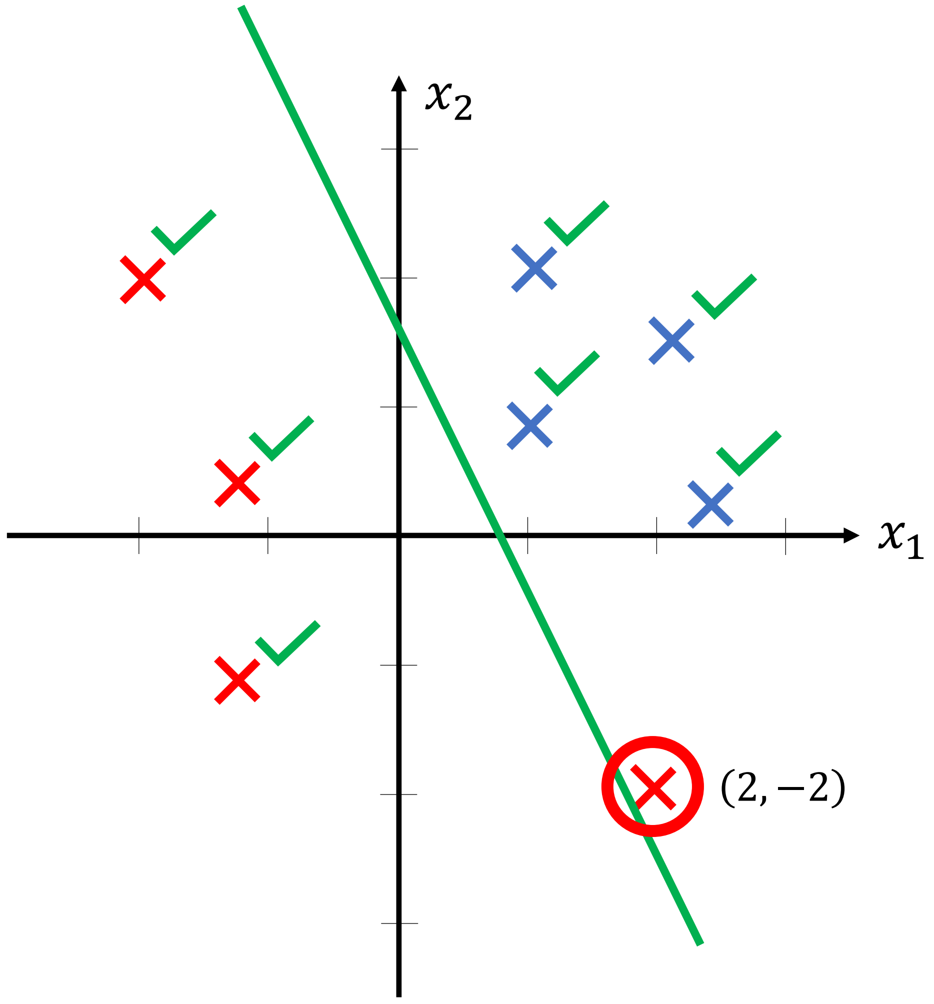
$$w \leftarrow \quad w \quad + \gamma\left(y^{(i)} - \hat{y}^{(i)}\right) x^{(i)}$$

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

New $h_w(x) = g(-0.8 + 1x_1 + 0.5x_2)$

Frank Rossenblatt (1943)

Perceptron Learning Algorithm: An Example



Sign function

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

$$\hat{y} = h_w(x) = g(\, w_0 + w_1 x_1 + w_2 x_2)$$
$$= g(-0.8 + 1x_1 + 0.5x_2)$$

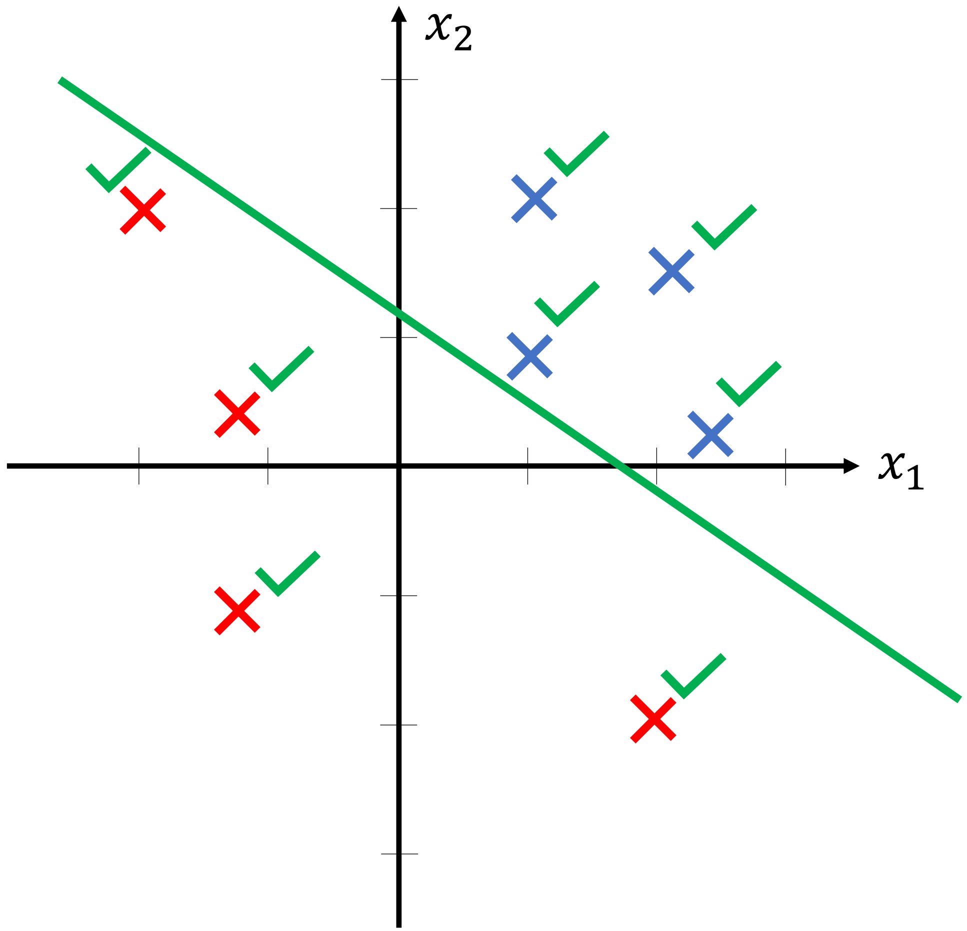
$$w \leftarrow \quad w \quad + \gamma\left(y^{(i)} - \hat{y}^{(i)}\right) x^{(i)}$$

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

New $h_w(x) = g(-1 + 0.6x_1 + 0.9x_2)$

Frank Rossenblatt (1943)

Perceptron Learning Algorithm: An Example



Sign function

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

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

No misclassifications! Converged!

What if it's not linearly separable?
The algorithm will not converge

Frank Rossenblatt (1943)

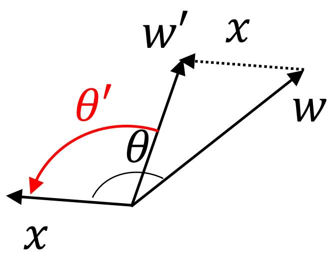
Perceptron Learning Algorithm: Why?

$$\hat{y} = g\left(\sum_{i=0}^{n} w_i x_i\right) \qquad g(z) = \begin{cases} +1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

When there is a misclassification: $y = +1, \hat{y} = -1$, or vice versa

$$\hat{y} = -1$$
$$w^T x < 0$$
$$w \cdot x < 0$$
$$||w||.||x|| \cos\theta < 0$$
$$\cos\theta < 0$$
$$\frac{\pi}{2} < \theta < \pi$$

Doesn't affect the sign



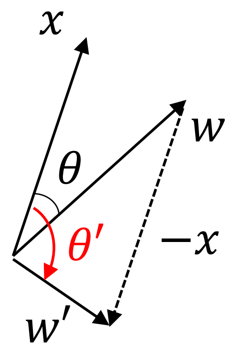
$$w' \leftarrow w + x$$

θ' will be smaller (as required)

What we want:
$$y = +1$$
$$\cos\theta > 0$$
$$0 < \theta < \frac{\pi}{2}$$

$$\hat{y} = +1$$
$$w^T x > 0$$
$$w \cdot x > 0$$
$$||w||.||x|| \cos\theta > 0$$
$$\cos\theta > 0$$
$$0 < \theta < \frac{\pi}{2}$$

Doesn't affect the sign



$$w' \leftarrow w - x$$

θ' will be larger (as required)

What we want:
$$y = -1$$
$$\cos\theta < 0$$
$$\frac{\pi}{2} < \theta < \pi$$

Frank Rossenblatt (1943)

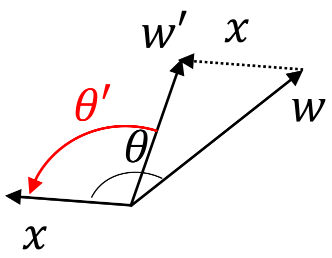
Perceptron Learning Algorithm: Why?

$$\hat{y} = g\left(\sum_{i=0}^{n} w_i x_i\right) \qquad g(z) = \begin{cases} +1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

When there is a misclassification: $y = +1, \hat{y} = -1$, or vice versa

$\hat{y} = -1$

$y = +1$



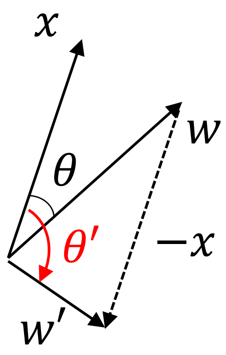
$$w' \leftarrow \boxed{w + x}$$

θ' will be smaller (as required)

$$\begin{aligned} w &\leftarrow w + \gamma(y - \hat{y})x \\ &\leftarrow w + \gamma(+1 - (-1))x \\ &\leftarrow \boxed{w + 2\gamma x} \end{aligned}$$

$\hat{y} = +1$

$y = -1$



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

θ' will be larger (as required)

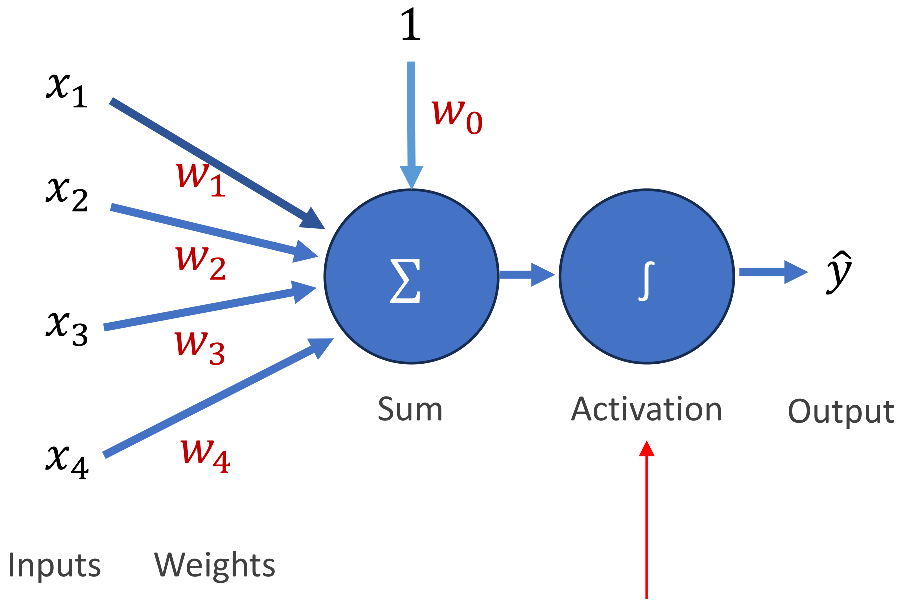
$$\begin{aligned} w &\leftarrow w + \gamma(y - \hat{y})x \\ &\leftarrow w + \gamma(-1 + (-1))x \\ &\leftarrow \boxed{w - 2\gamma x} \end{aligned}$$

Outline

- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- **Neural Networks**
  - Single-layer Neural Networks
  - Multi-layer Neural Networks
  - Regression and Classification
- Neural Networks with Gradient Descent
- Neural Networks vs Other Models

Frank Rossenblatt (1943)

Perceptron

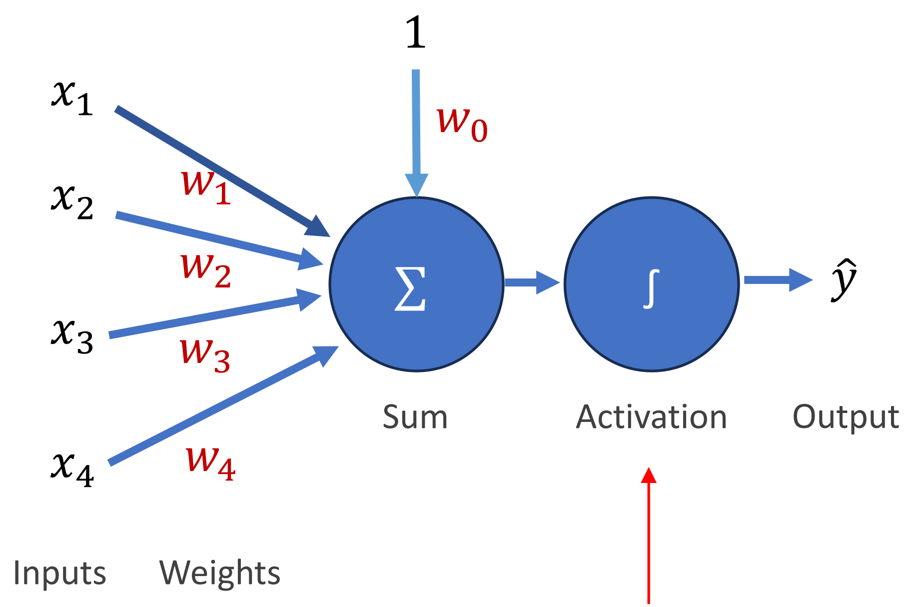


$$\hat{y} = h_w(x) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

Sign function

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

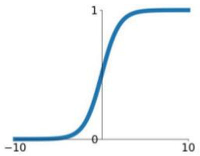
Single-layer Neural Networks



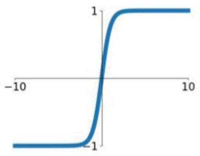
$$\hat{y} = h_w(x) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

Activation Function
(usually non-linear)

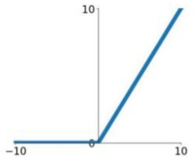
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



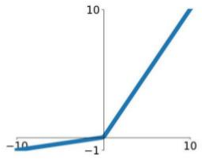
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

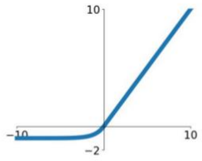


Leaky ReLU
$\max(0.1x, x)$

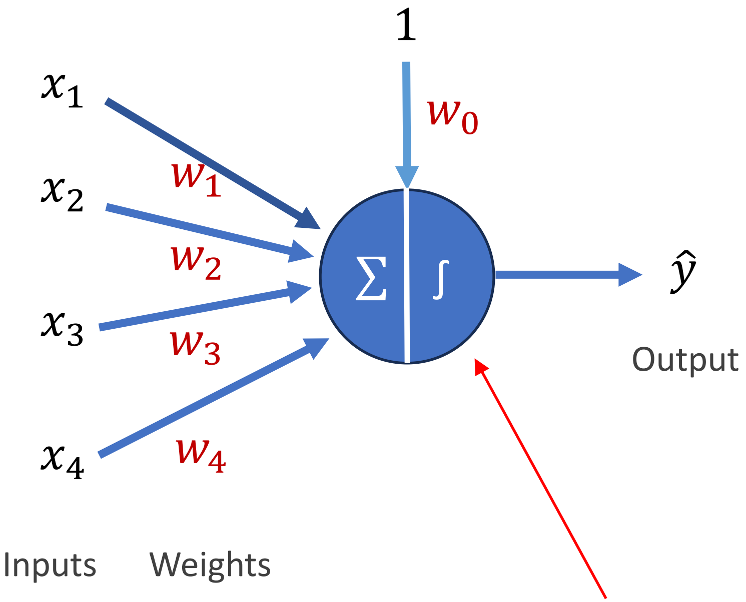


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



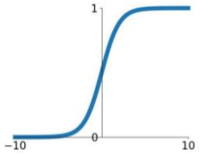
Single-layer Neural Networks



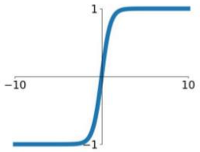
Activation Function
(usually non-linear)

$$\hat{y} = h_w(x) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

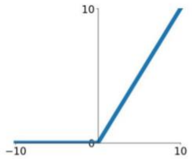
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



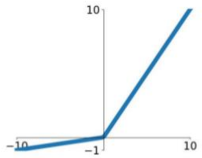
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

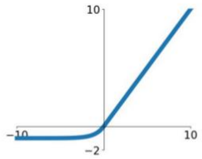


Leaky ReLU
$\max(0.1x, x)$

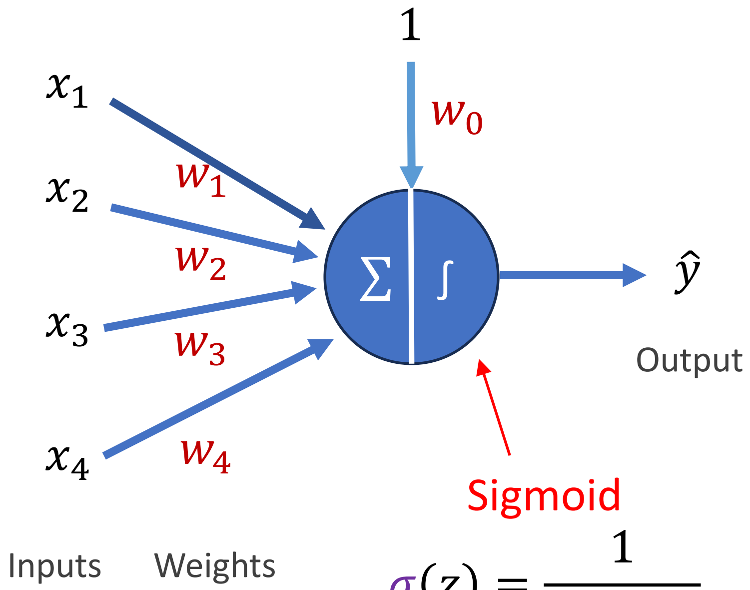


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



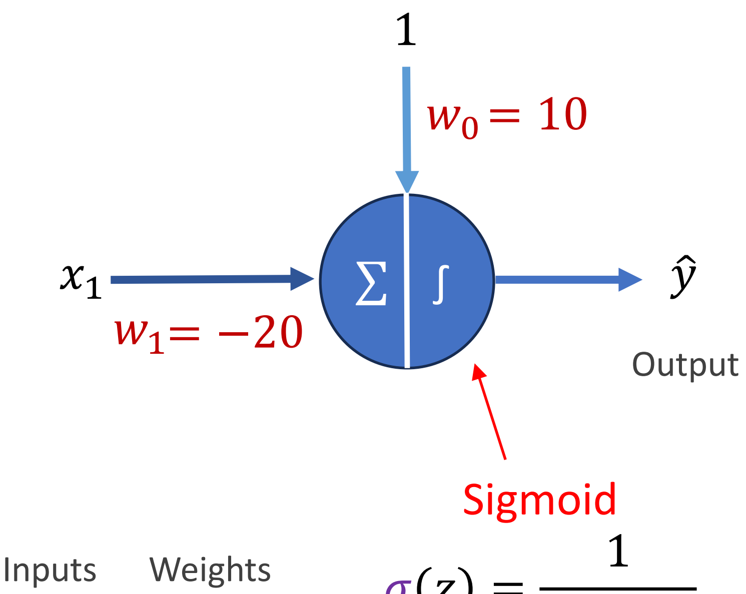
Single-layer Neural Networks



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

$$\hat{y} = h_w(x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

Single-layer Neural Networks: NOT



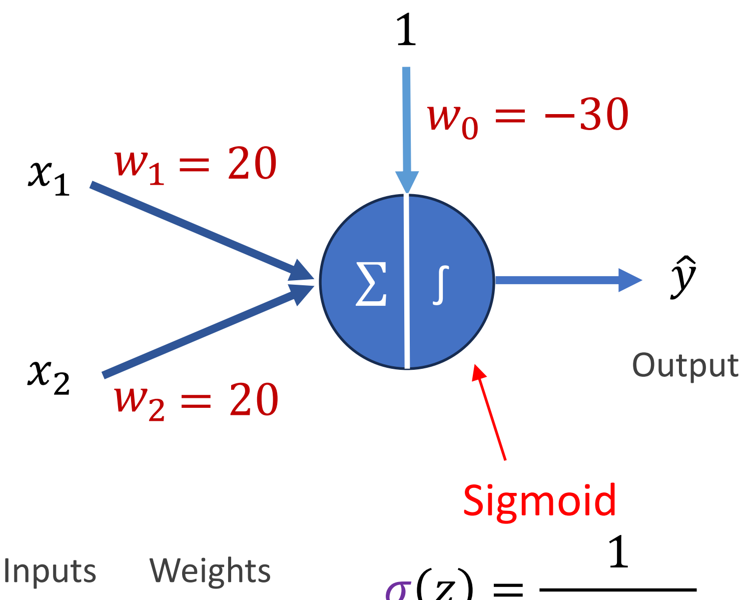
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\hat{y} = h_w(x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

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

x_1	y	Σ	$\hat{y}$
0	1	10	0.999
1	0	−10	0.00004

Single-layer Neural Networks: AND



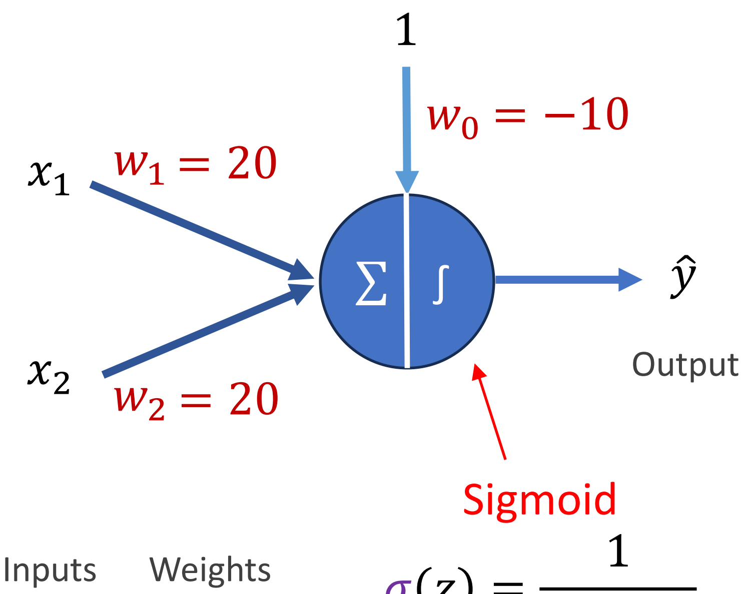
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = h_w(x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

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

x_1	x_2	y	Σ	$\hat{y}$
0	0	0	-30	0.000 ...
0	1	0	-10	0.00004
1	0	0	-10	0.00004
1	1	1	10	0.999

Single-layer Neural Networks: OR



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

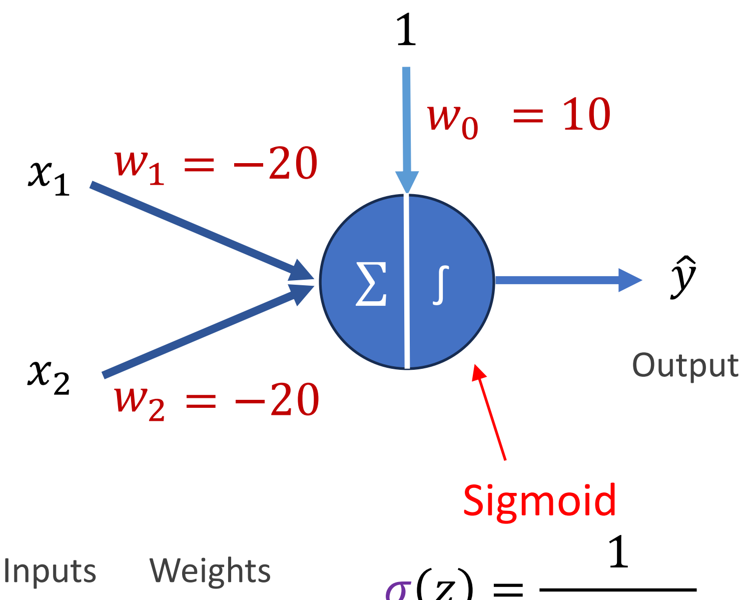
$$\hat{y} = h_w(x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

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

x_1	x_2	y	Σ	$\hat{y}$
0	0	0	-10	0.00004
0	1	1	10	0.999
1	0	1	10	0.999
1	1	1	30	0.999

Single-layer Neural Networks: NOR

(NOT x_1) AND (NOT x_2)



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

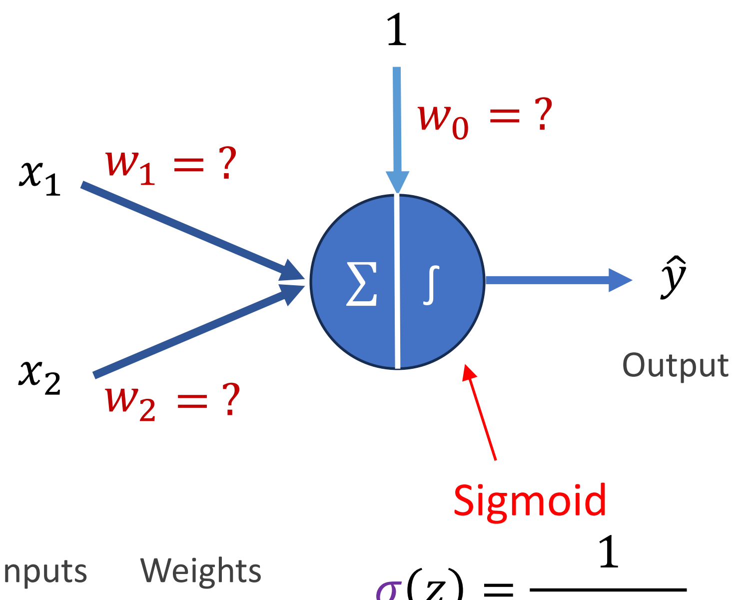
$$\hat{y} = h_w(x) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right)$$

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

x_1	x_2	y	Σ	$\hat{y}$
0	0	1	10	0.999
0	1	0	−10	0.00004
1	0	0	−10	0.00004
1	1	0	−30	0.000 …

Single-layer Neural Networks: XNOR

eXclusive Not OR

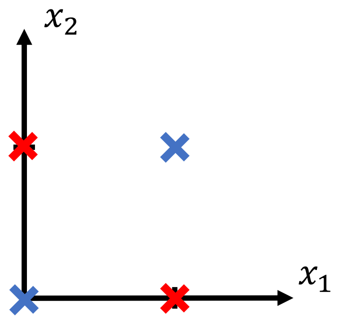


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\hat{y} = h_w(x) = \sigma\left(\textstyle\sum_{i=0}^{n} w_i x_i\right)$$

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

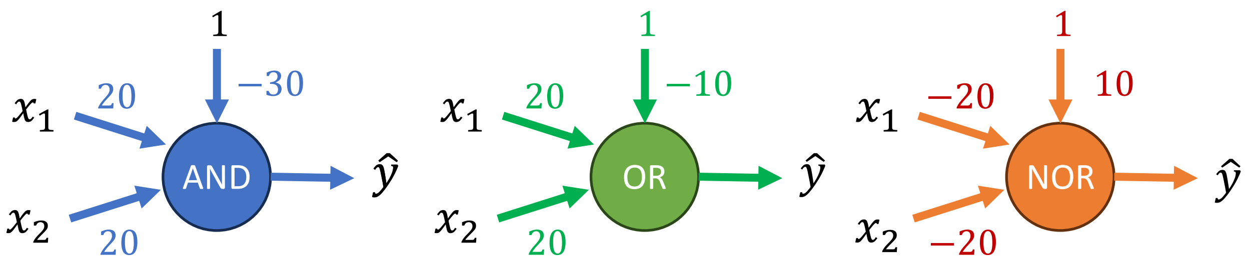
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1



Not linearly separable!

Single-layer Neural Networks: XNOR

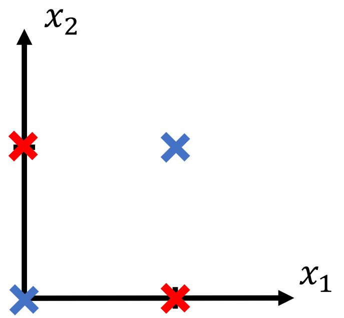
eXclusive Not OR



$\hat{y} = h_w(x) = \sigma(\sum_{i=0}^{n} w_i x_i)$

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

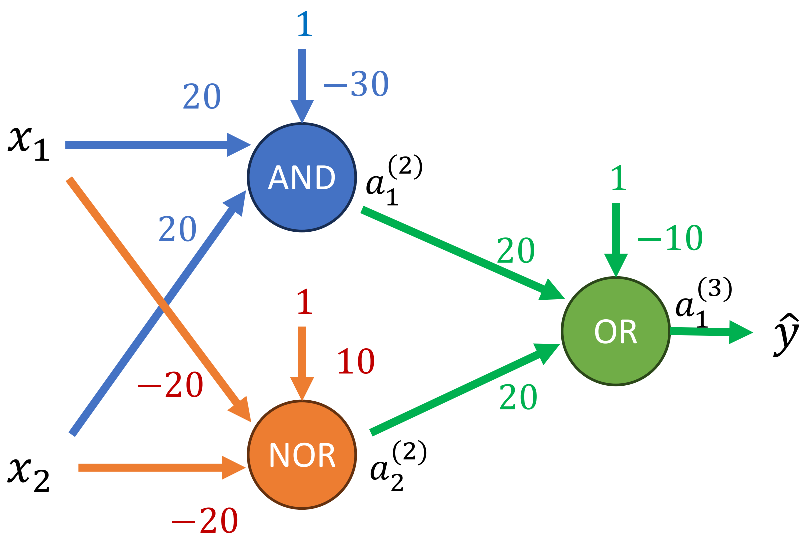


Not linearly separable!

Multi ~~Single~~-layer Neural Networks: XNOR

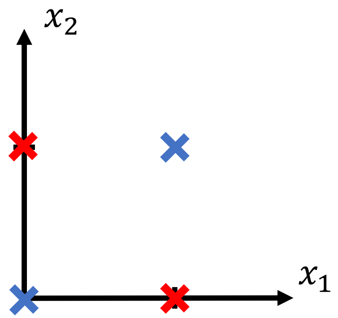
eXclusive Not OR

$$\cancel{\hat{y} = h_w(x) = \sigma(\textstyle\sum_{i=0}^{n} w_i x_i)}$$



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

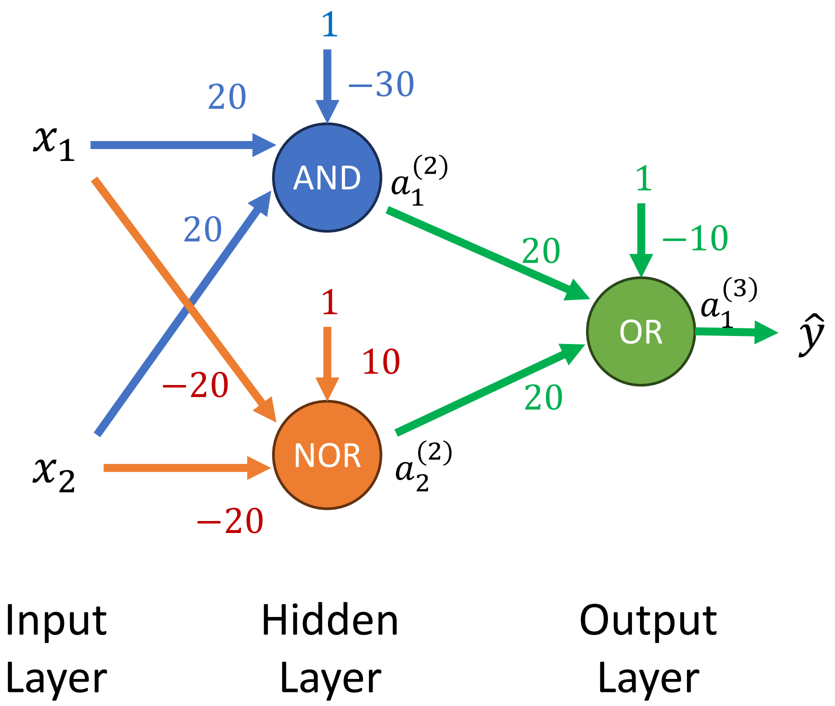
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1



<u>Not</u> linearly separable!

Multi-layer Neural Networks: XNOR

eXclusive Not OR



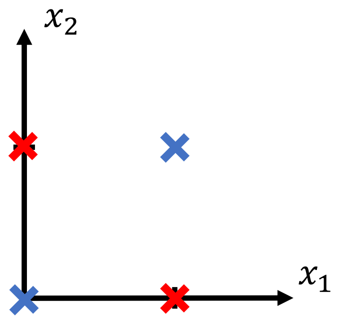
$$\hat{y} = \sigma\left(\sum_{i=0}^{n} w_{1i}^{(3)} a_i^{(2)}\right) \qquad a_i^{(2)} = \sigma\left(\sum_{j=0}^{n} w_{ij}^{(2)} x_j\right)$$

Function composition $\qquad a_0^{(2)} = 1$

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

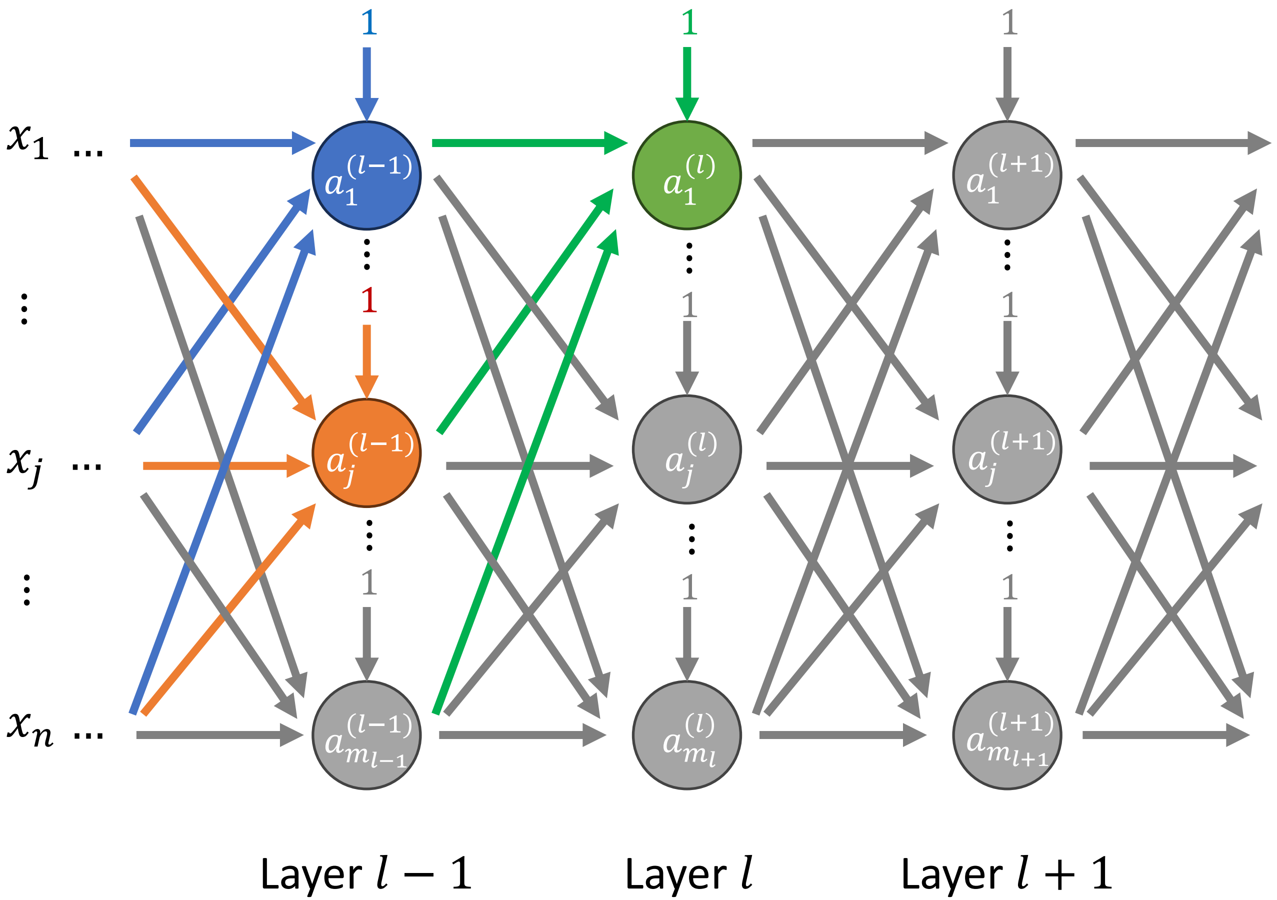
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$a_1^{(2)}$	$a_2^{(2)}$	$\hat{y}$
0	1	1
0	0	0
0	0	0
1	0	1

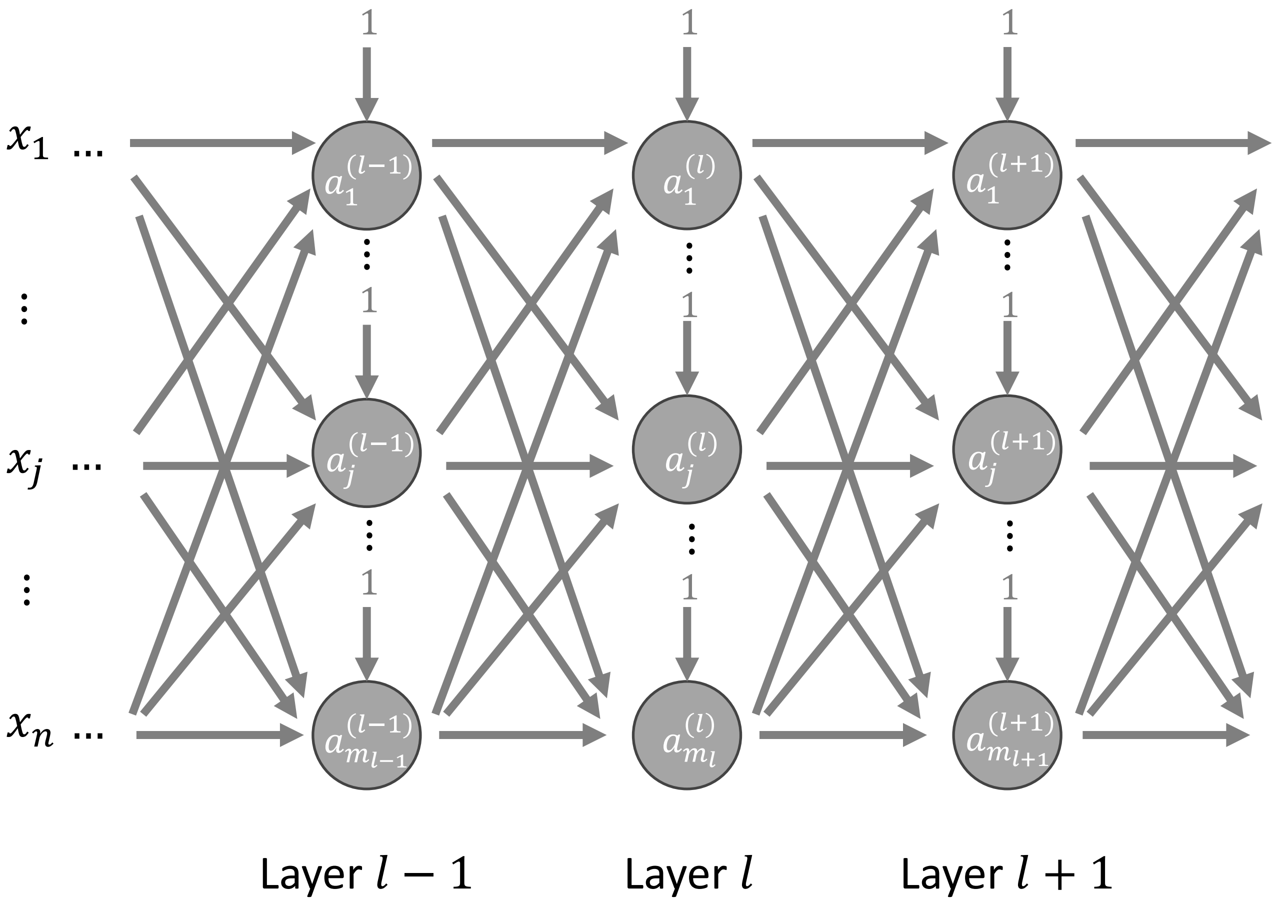


Not linearly separable!

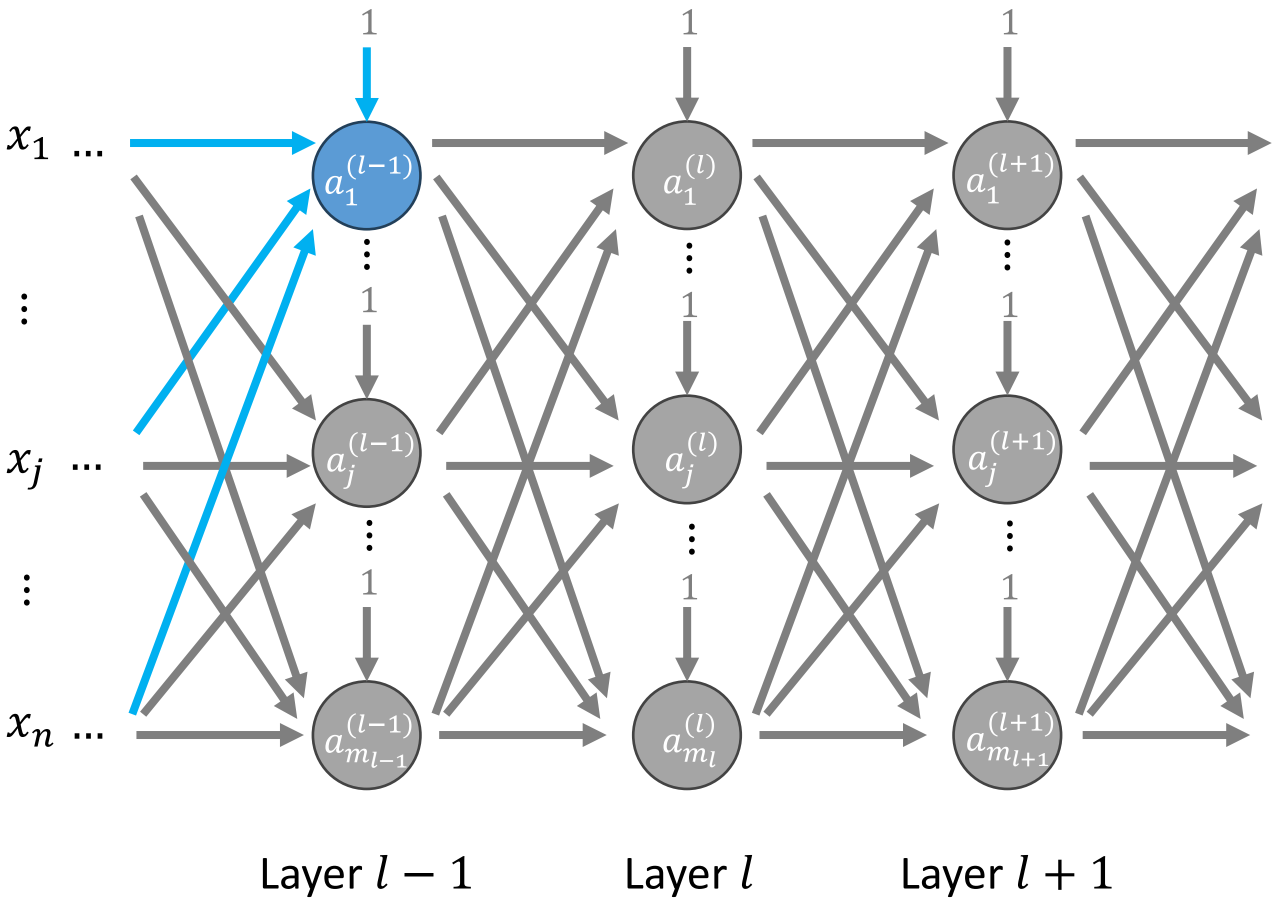
Multi-layer Neural Networks



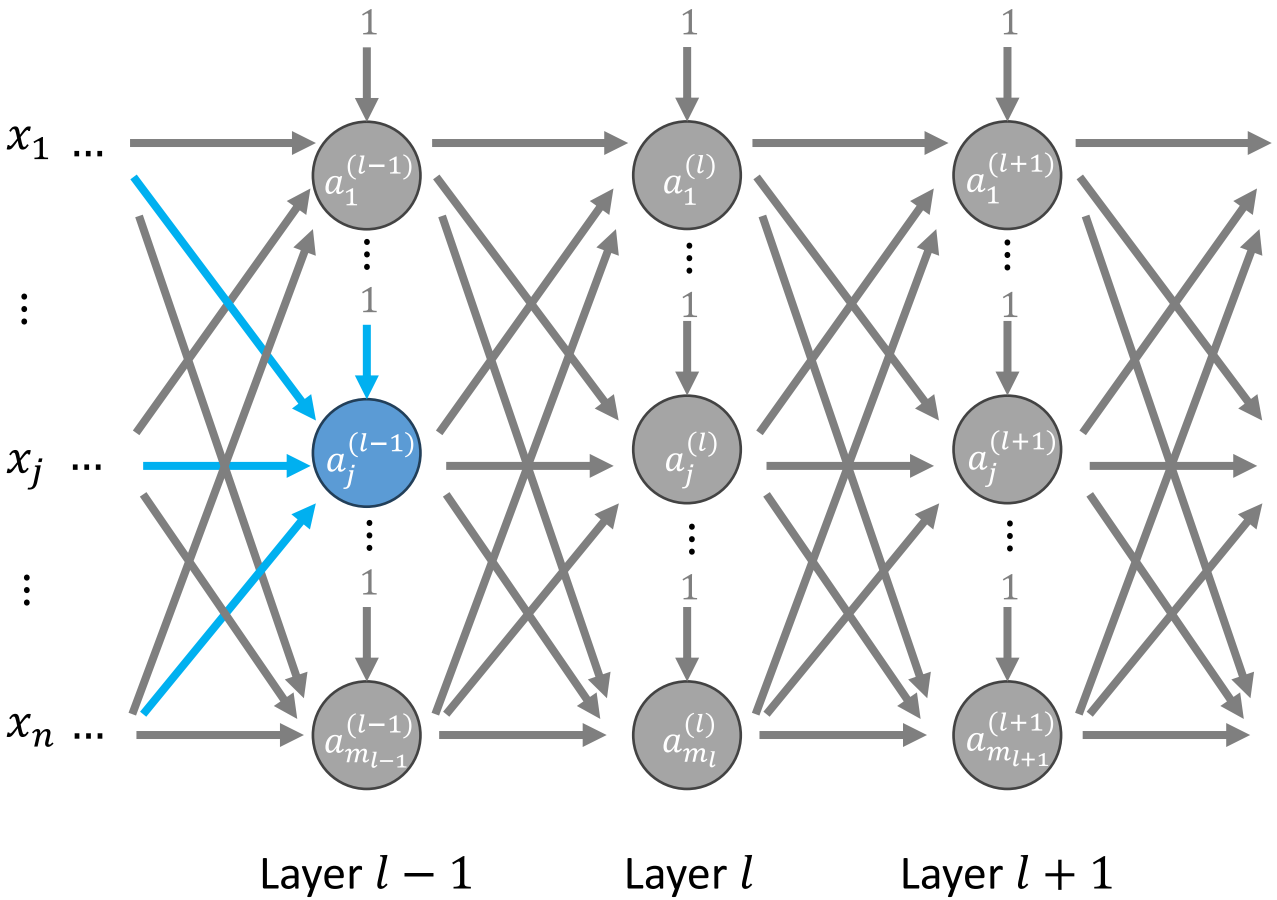
Multi-layer Neural Networks



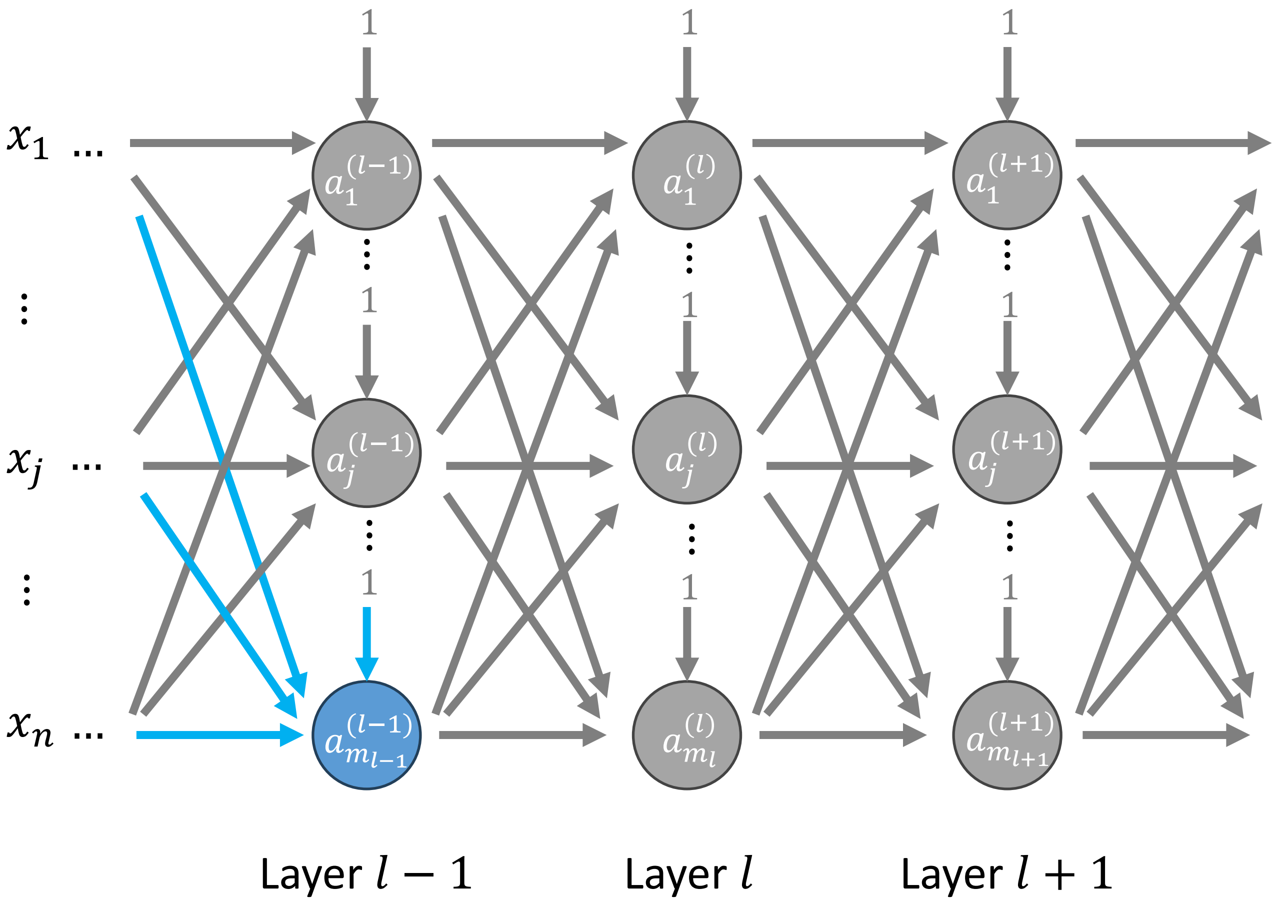
Multi-layer Neural Networks



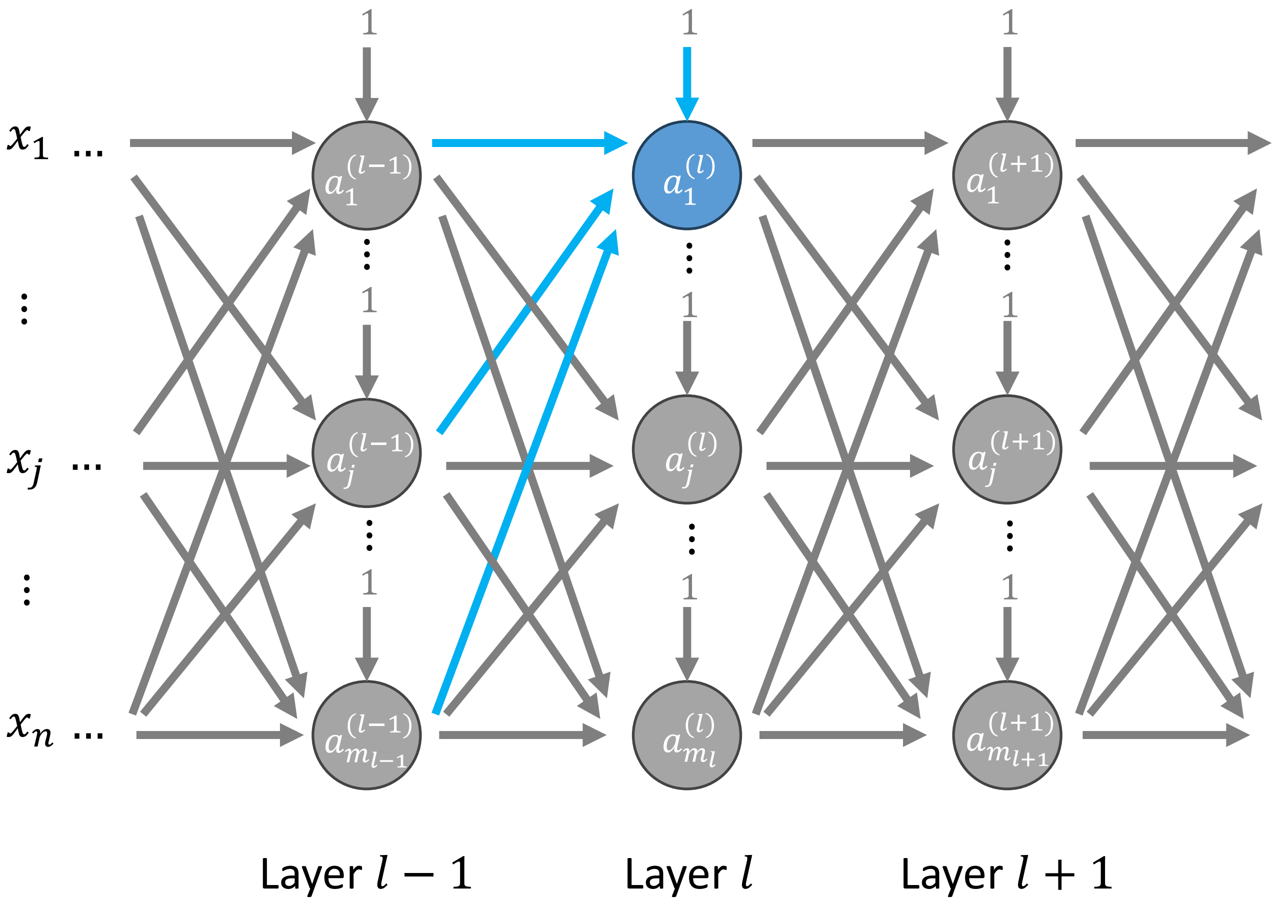
Multi-layer Neural Networks



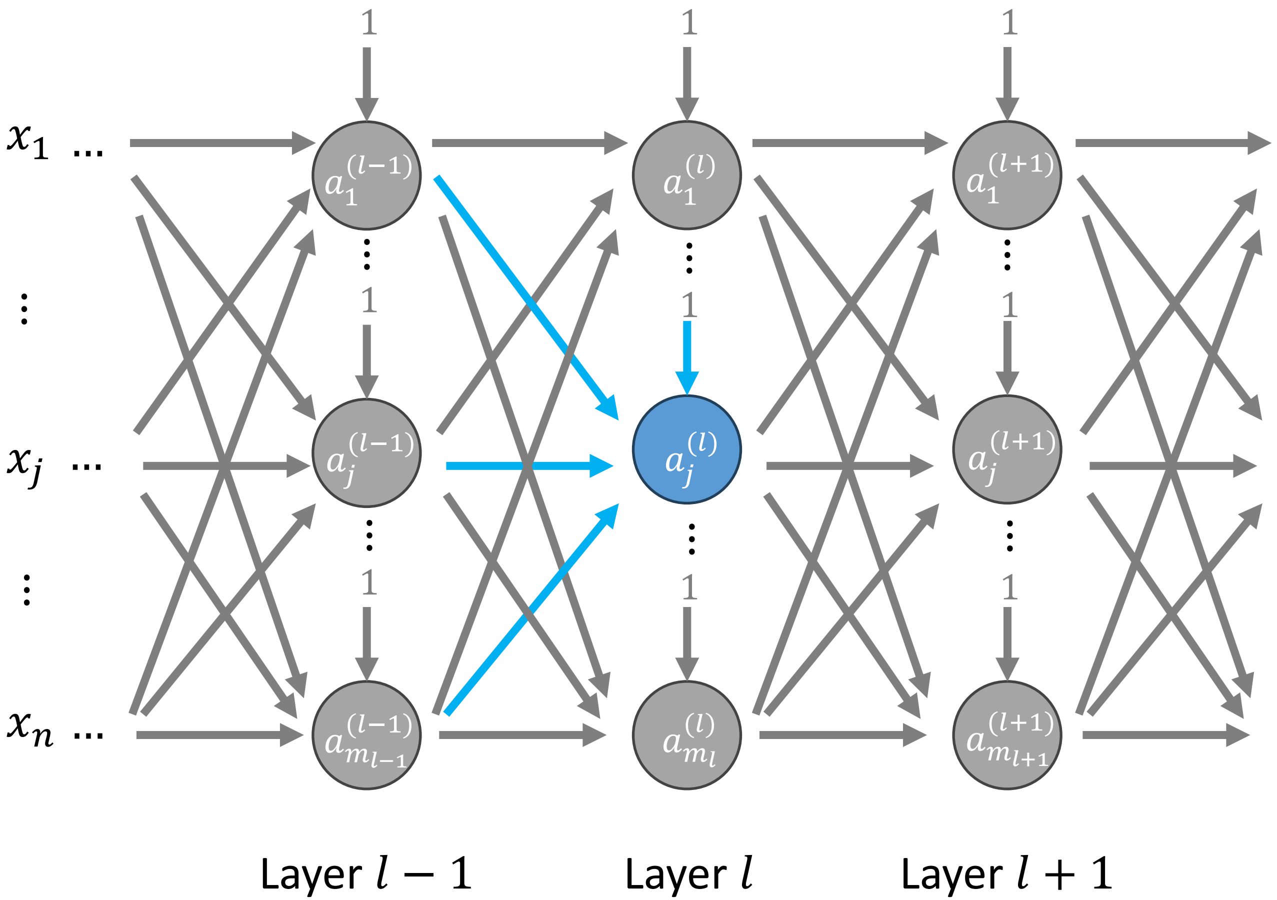
Multi-layer Neural Networks



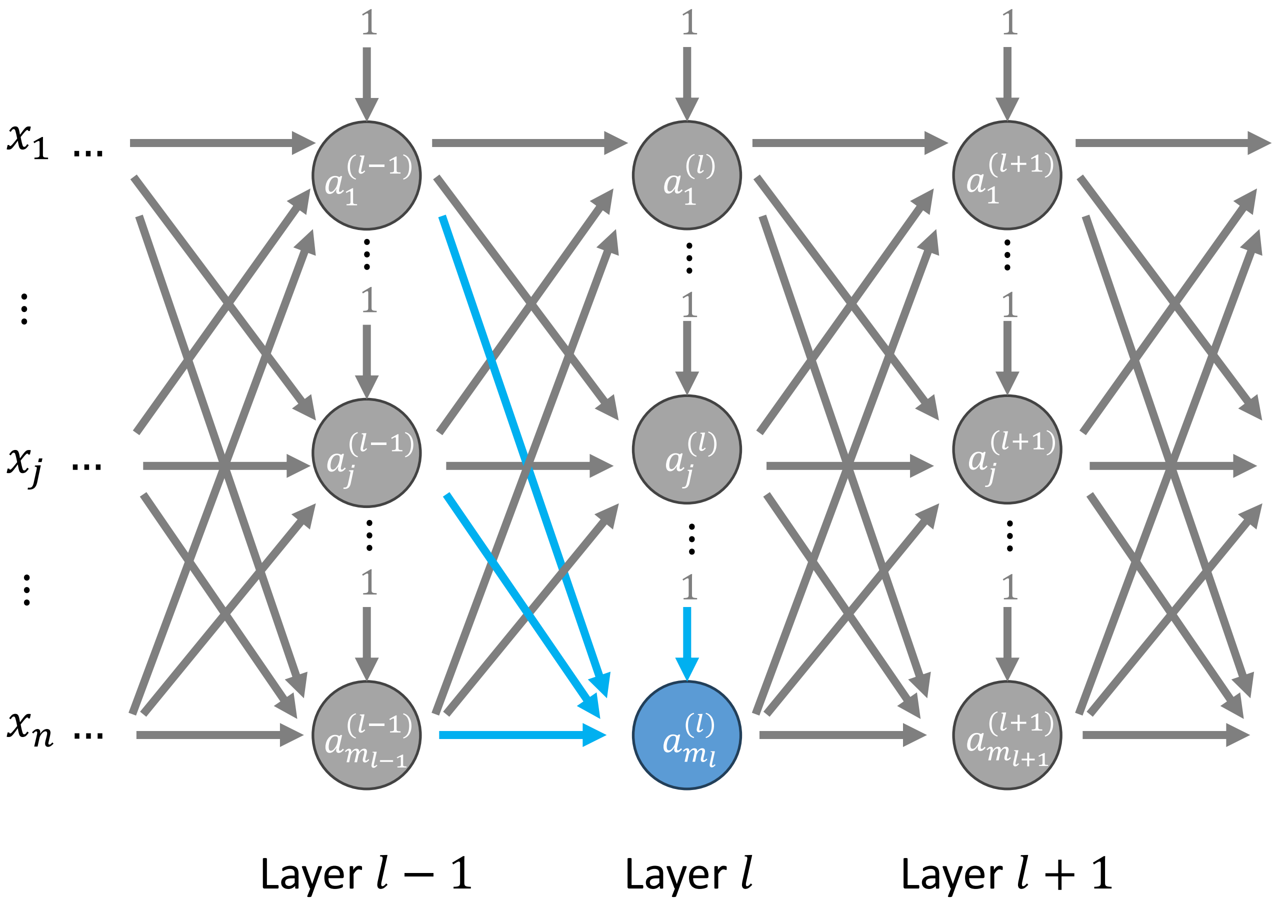
Multi-layer Neural Networks



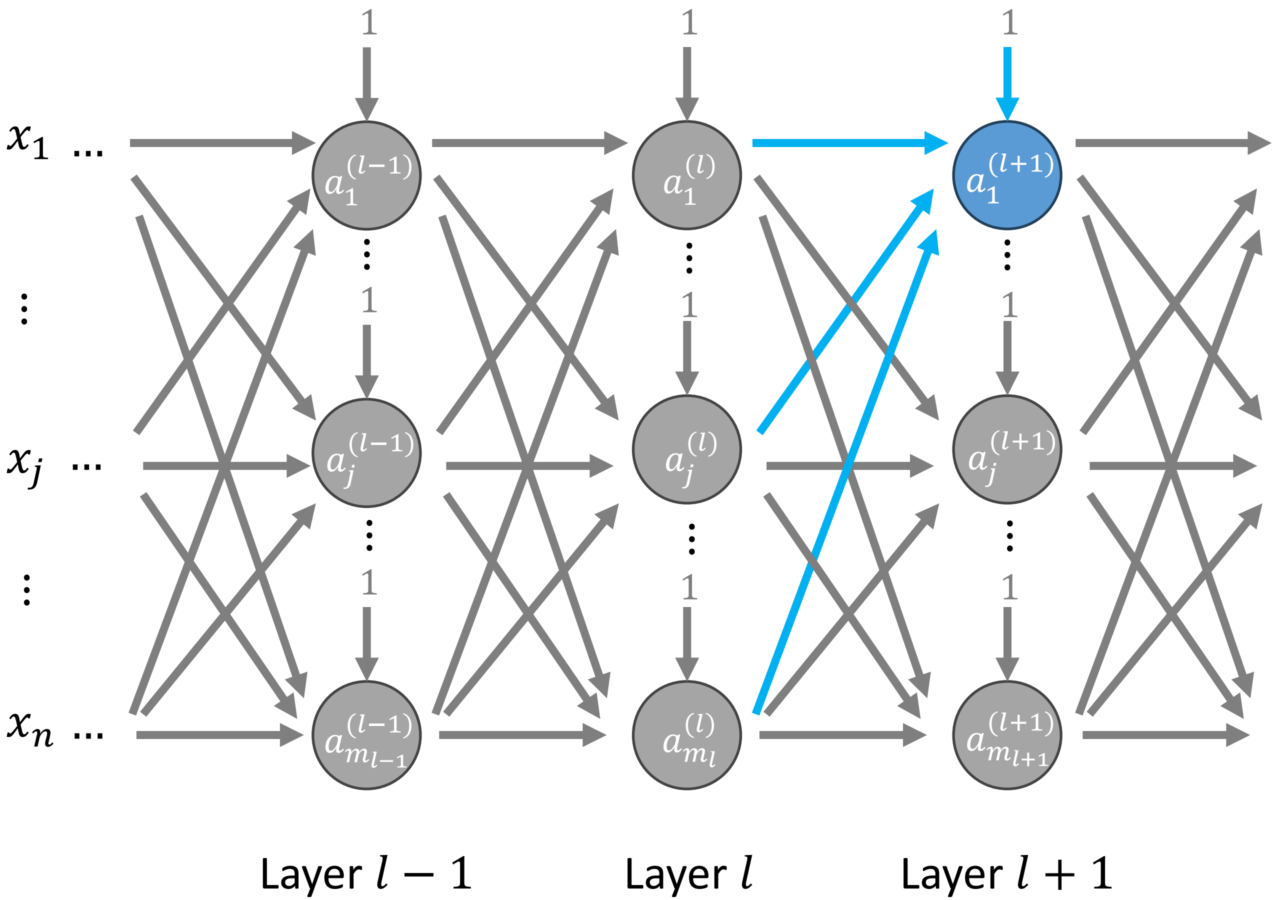
Multi-layer Neural Networks



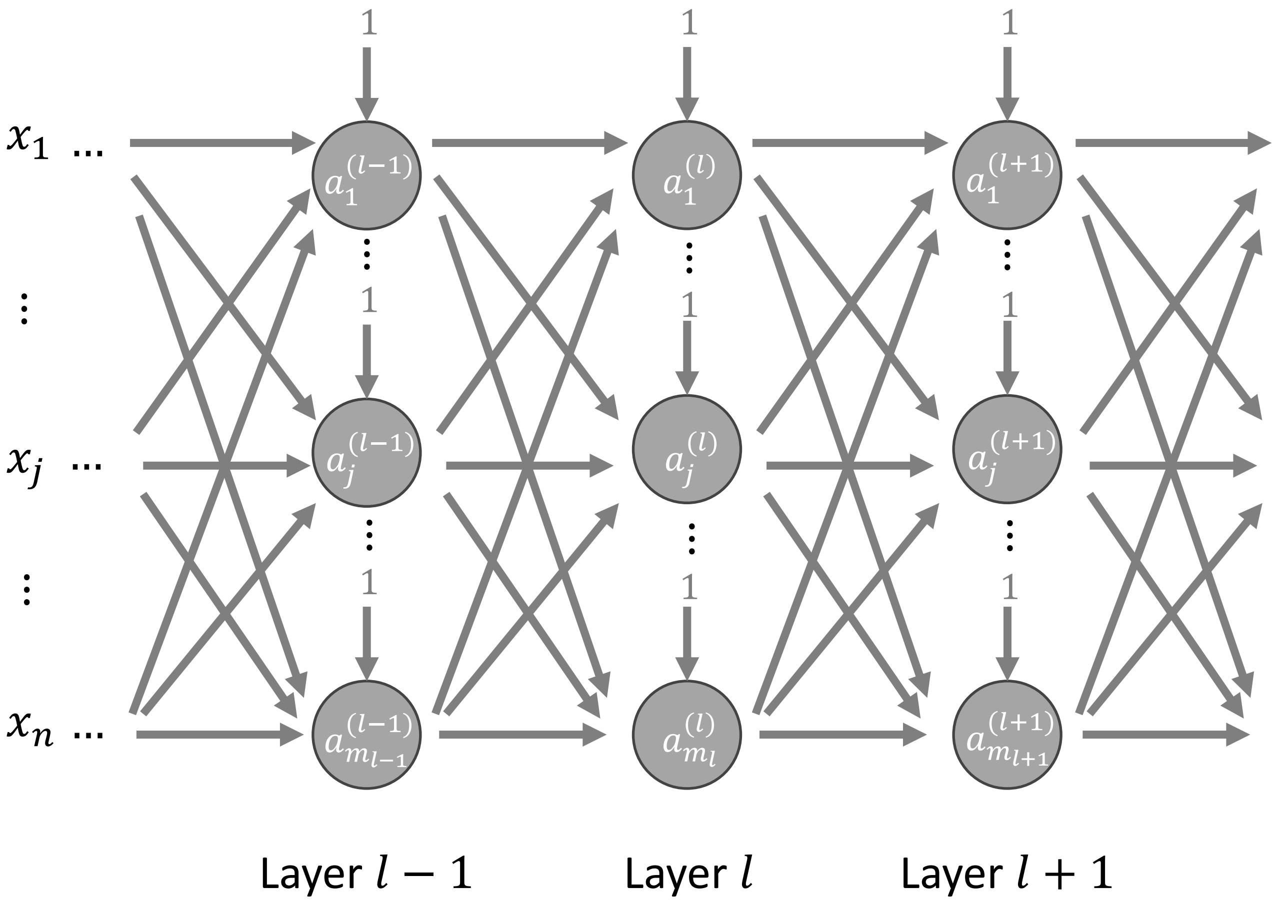
Multi-layer Neural Networks



Multi-layer Neural Networks



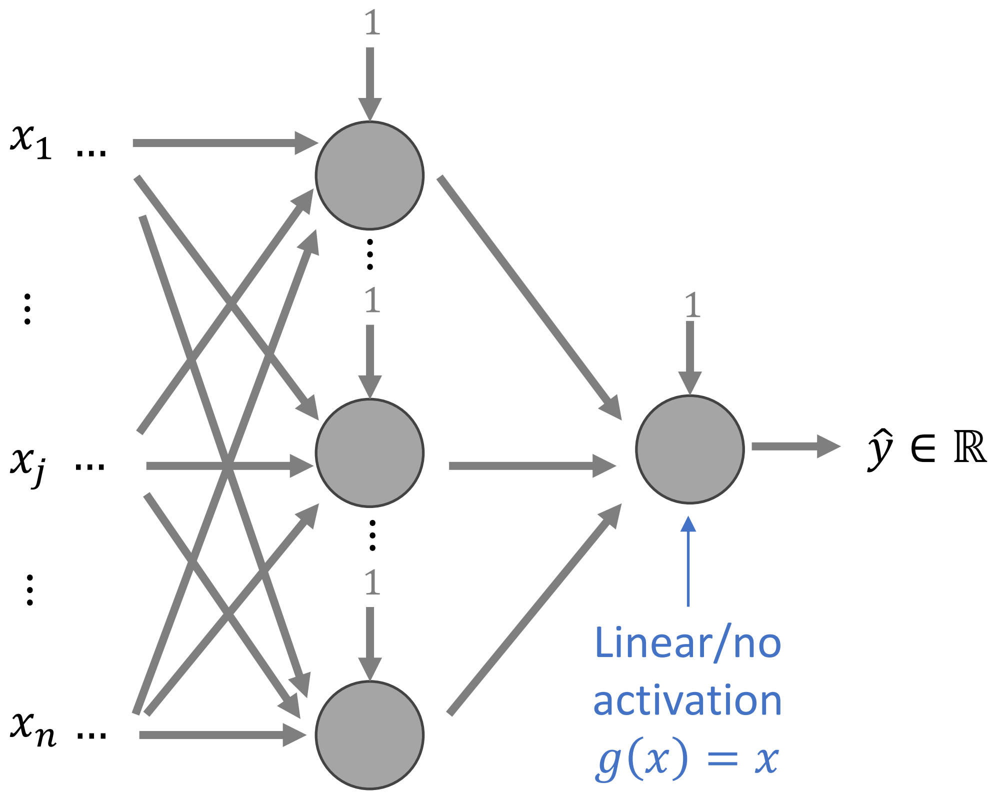
Multi-layer Neural Networks



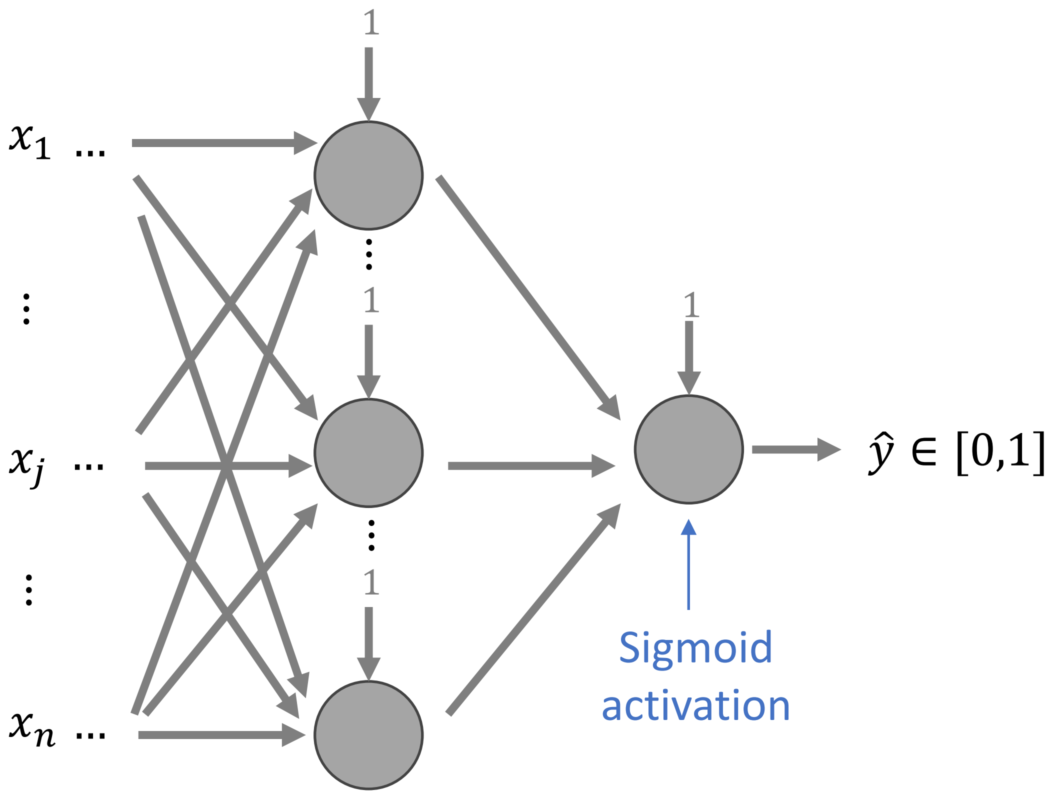
Universal Function Approximation Theorem
Neural networks can *represent* a wide variety of interesting functions with appropriate weights

- A <u>single hidden layer</u> network can approximate any continuous function within a specific range

Regression and Classification

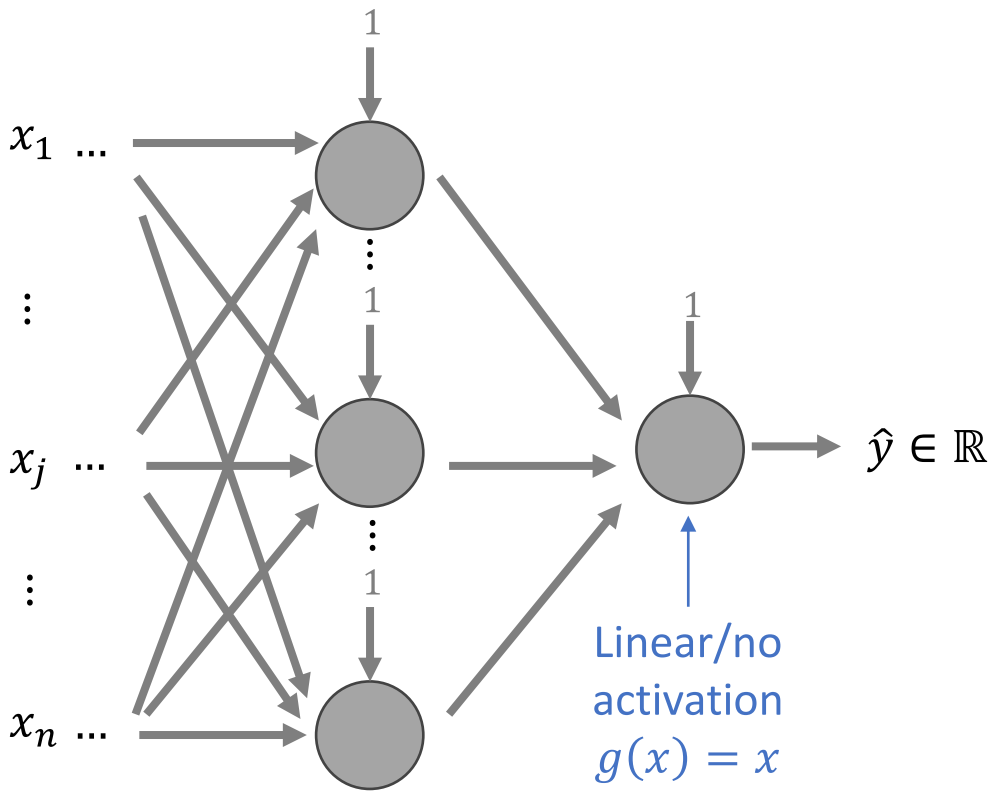


Regression

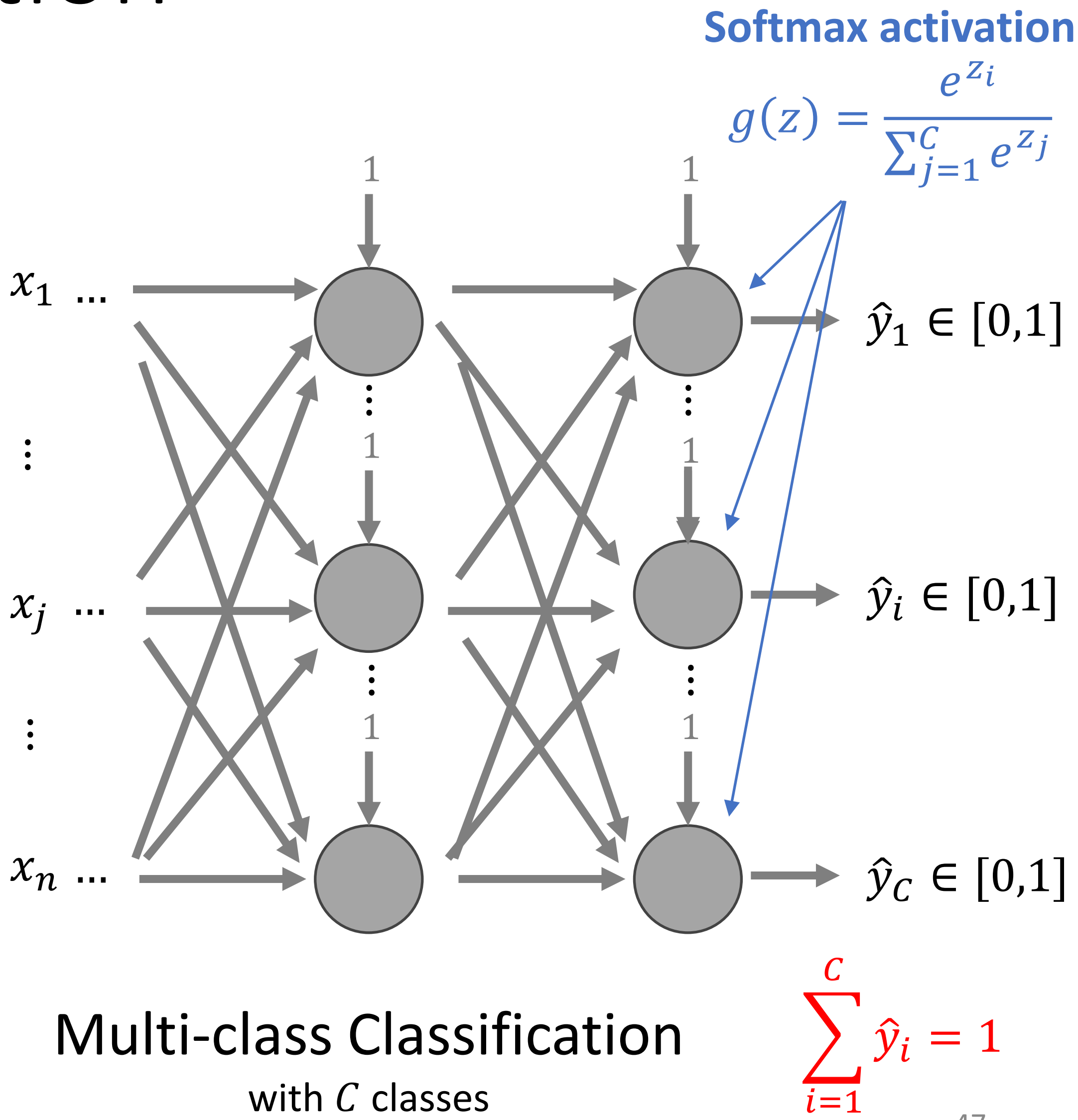


Binary Classification

Regression and Classification



Regression



Multi-class Classification
with C classes

Outline

- Perceptron
  - Biological inspiration
  - Perceptron Learning Algorithm
- Neural Networks
  - Single-layer Neural Networks
  - Multi-layer Neural Networks
  - Regression and Classification
- **Gradient Descent with Neural Networks**
- Neural Networks vs Other Models

Background: Chain Rule

$$z(x) = h\left(g\left(f(x)\right)\right)$$

$$\frac{dz}{dx} = \frac{dz}{dh}\frac{dh}{dg}\frac{dg}{df}\frac{df}{dx}$$

Aside: Derivative of Sigmoid Function

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

$$\begin{aligned}
\sigma'(x) &= \frac{d\left(\frac{1}{1 + e^{-x}}\right)}{dx} \\
&= \frac{d\left((1 + e^{-x})^{-1}\right)}{dx} \\
&= -(1 + e^{-x})^{-2}\,(-e^{-x}) \\
&= \frac{e^{-x}}{(1 + e^{-x})^2} \\
&= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\
&= \frac{1}{1 + e^{-x}}\left(1 - \frac{1}{1 + e^{-x}}\right) \\
&= \boxed{\sigma(x)\left(1 - \sigma(x)\right)}
\end{aligned}$$

Gradient Descent on Single-layer NN

$$\hat{y} = \sigma\big(f(x)\big) \qquad f(x) = \sum_{i=0}^{n} w_i x_i \qquad L = \frac{1}{2}(\hat{y} - y)^2$$

Mean-squared Error

$$w_i \leftarrow w_i - \gamma \frac{dL}{dw_i}$$

$$w_i \leftarrow w_i - \gamma(\hat{y} - y)\hat{y}(1 - \hat{y})x_i$$

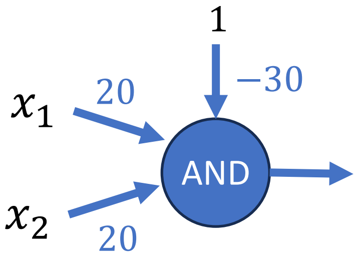
$$\frac{dL}{dw_i} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{df}\frac{df}{dw_i}$$

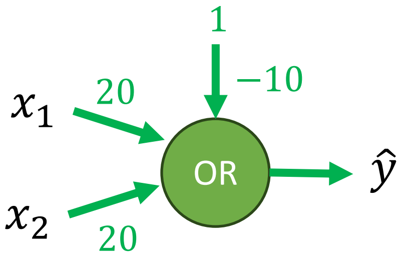
$$= (\hat{y} - y)\hat{y}(1 - \hat{y})x_i$$

$$\frac{dL}{d\hat{y}} = \frac{d\left(\frac{1}{2}(\hat{y} - y)^2\right)}{d\hat{y}} = (\hat{y} - y)$$

$$\frac{d\hat{y}}{df} = \frac{d\Big(\sigma\big(f(x)\big)\Big)}{f} = \sigma\big(f(x)\big)\Big(1 - \sigma\big(f(x)\big)\Big) = \hat{y}(1 - \hat{y})$$

$$\frac{df}{dw_i} = \frac{d(\sum_{i=0} w_i x_i)}{dw_i} = \frac{d(w_i x_i)}{dw_i} + \cancelto{0}{\frac{d(\sum_{k \neq i} w_k x_k)}{dw_i}} = x_i$$





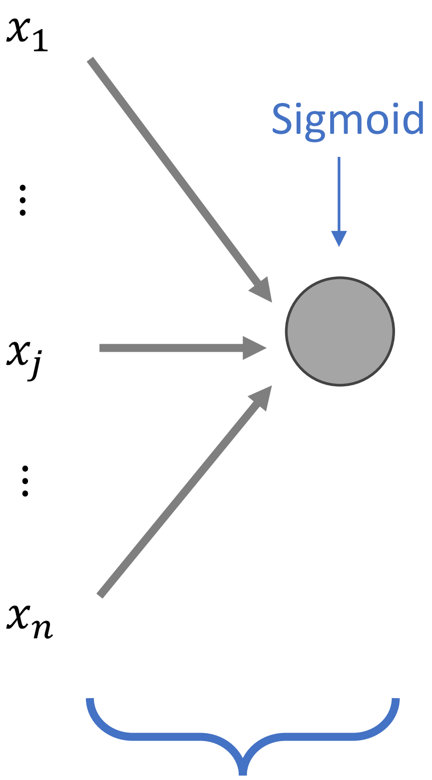
Gradient Descent on Multi-layer NN

Backpropagation is covered in the next Lecture!

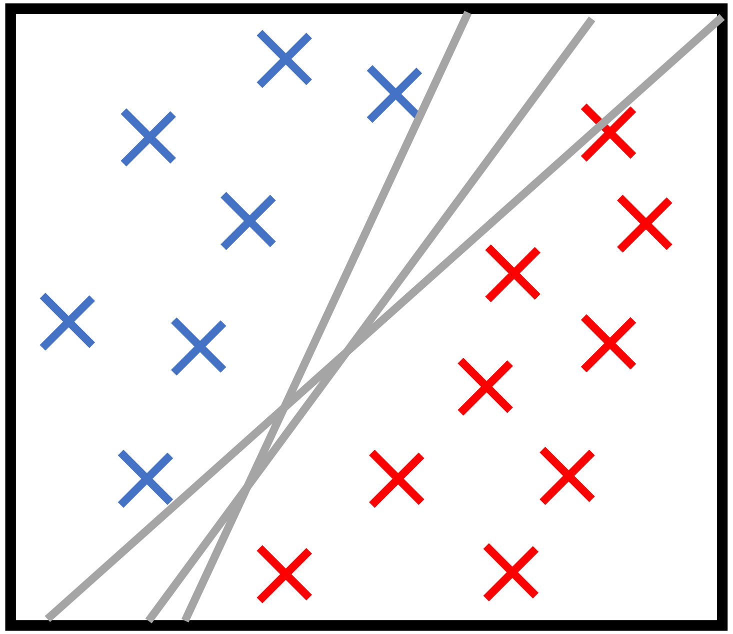
Outline

- Perceptron
    - Biological inspiration
    - Perceptron Learning Algorithm
- Neural Networks
    - Single-layer Neural Networks
    - Multi-layer Neural Networks
    - Regression and Classification
- Neural Networks with Gradient Descent
- **Neural Networks vs Other Models**

Neural Networks vs **Logistic Regression**



Linear Model

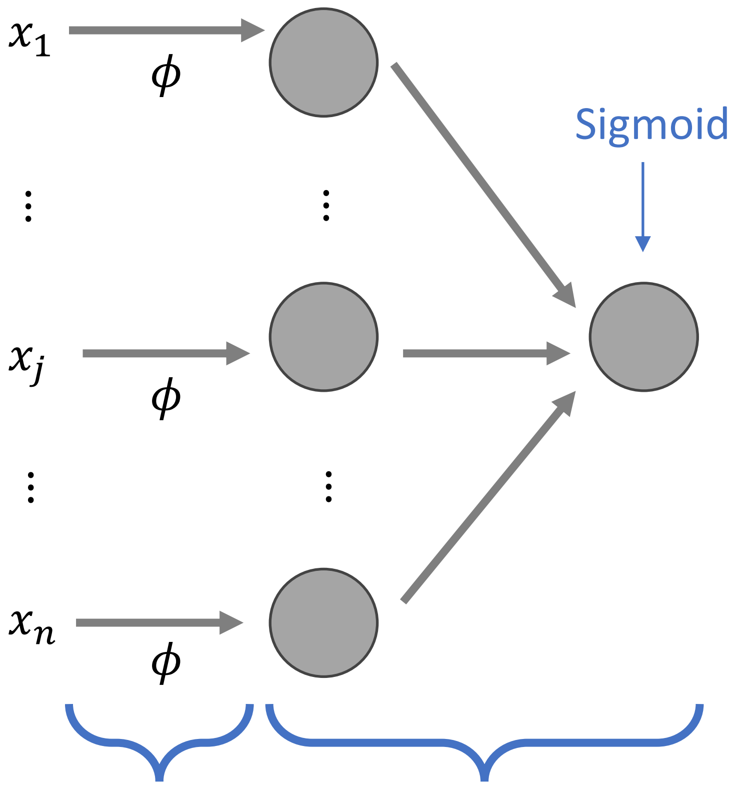


Linear, **non-robust** decision boundary

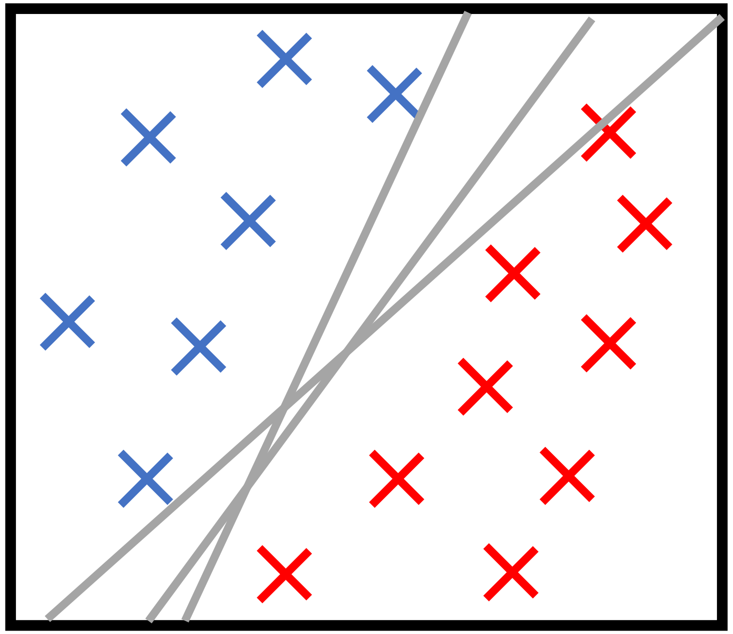
Prone to misclassification since the decision boundary can be **too close to data points**

Neural Networks vs **Logistic Regression**

With Feature Mapping



Handcrafted Feature Mapping Linear Model

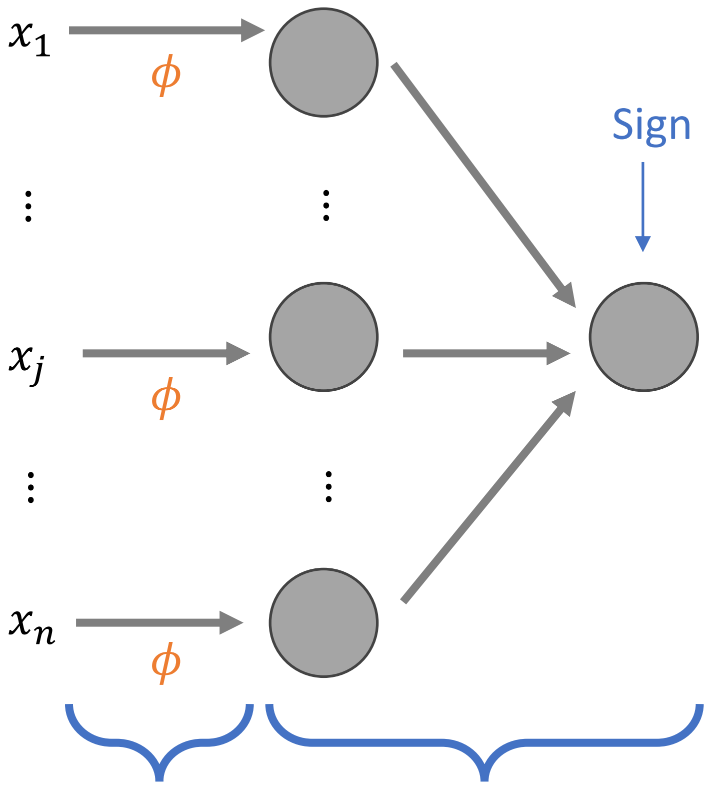


Non-linear, **non-robust** decision boundary

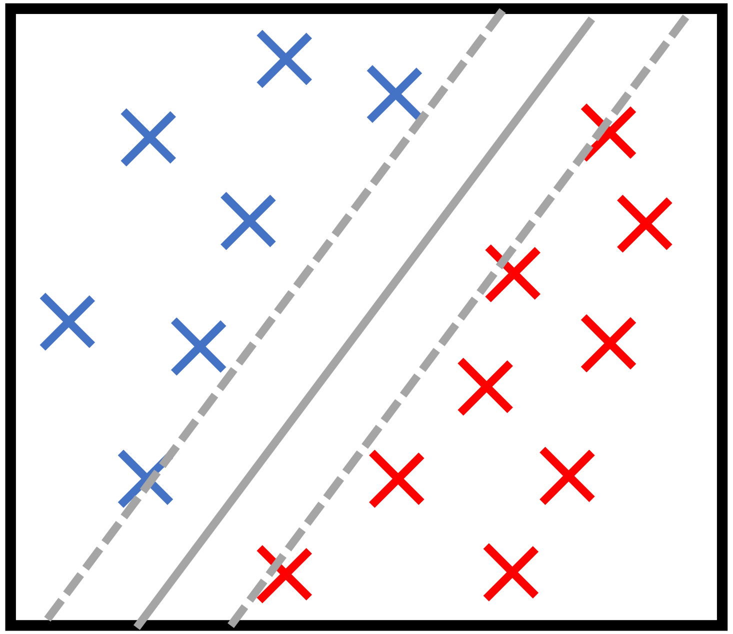
Prone to misclassification since the decision boundary can be **too close to data points**

Neural Networks vs **Support Vector Machines**

Implicit with **Kernel**: can have many, possibly infinite dimensional features



Handcrafted Feature Mapping Linear Model

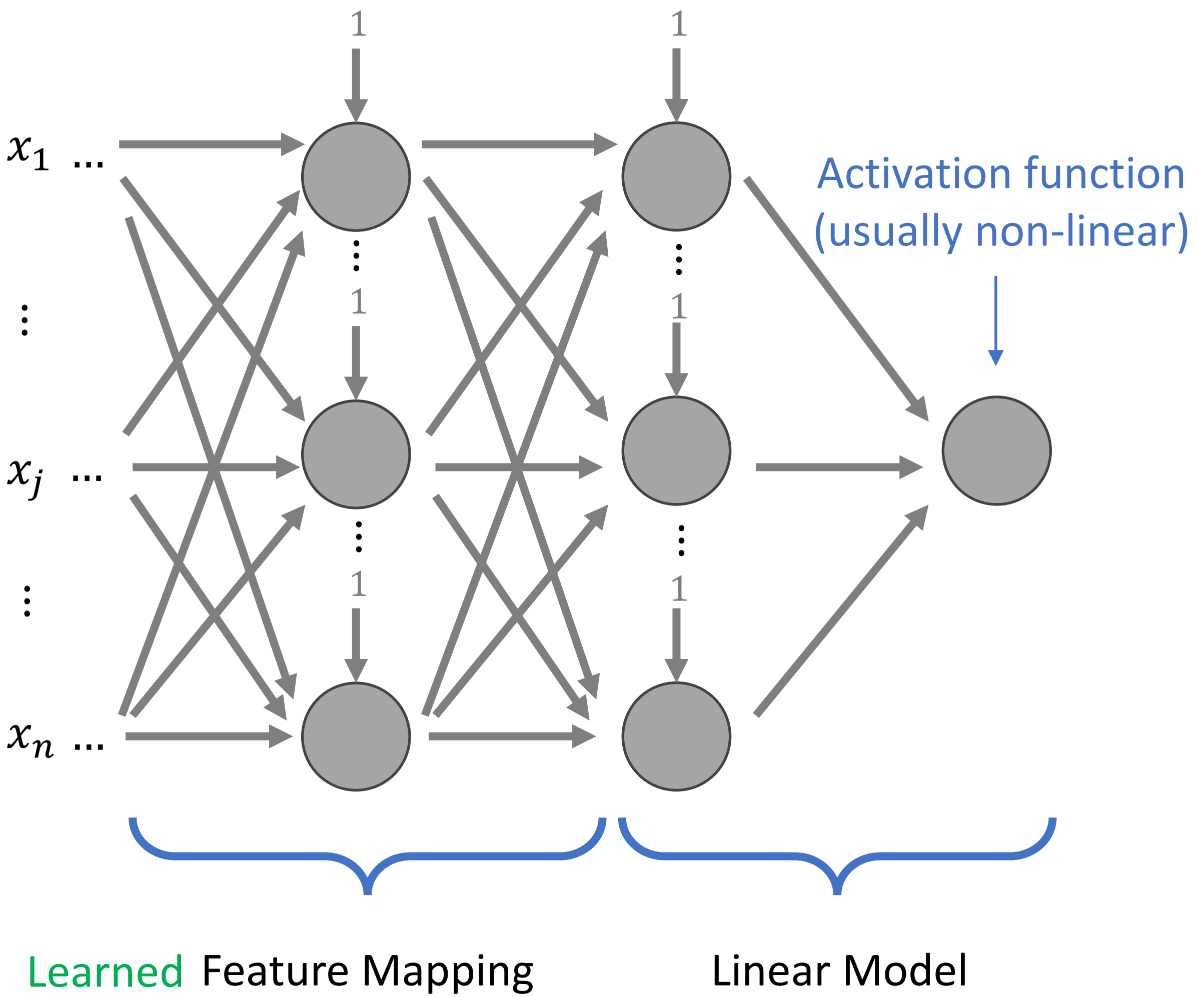


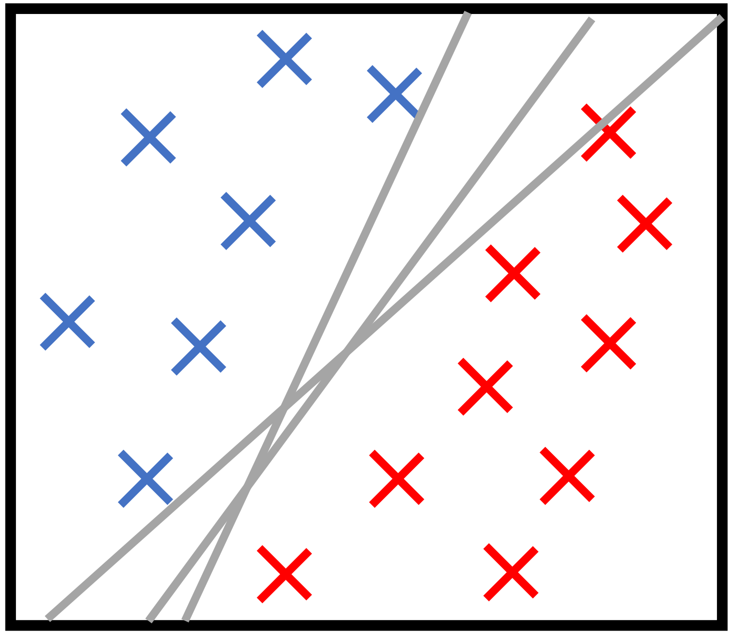
Non-linear, **robust** decision boundary

Decision boundary is guaranteed to be **far from the data points**

Neural Networks vs Other Methods

Credit: Internet meme, original source unknown





Non-linear, non-robust decision boundary

Prone to misclassification since the decision boundary can be **too close to data points**

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
  - Single-layer Neural Networks: AND, OR, NOR
  - Multi-layer Neural Networks: XNOR, Universal Approximation Theorem
  - Regression and Classification
- Neural Networks with Gradient Descent
- Neural Networks vs Other Models: learned feature mapping!

Coming Up Next Week

- **Math revision**
  - Linear algebra: scalar, vector, matrix, and their operations
  - Calculus: partial derivative, matrix calculus, chain rule
- **Backpropagation**
- **Introduction to PyTorch**
  - Training neural networks with gradient descent

To Do

- **Lecture Training 8**
  - +100 Free EXP
  - +50 Early bird bonus
- **PS5**
  - Due **tomorrow** (Saturday, 21st October) 23:59
- **PS6 is out today!**
  - May need some knowledge from the next lecture (Lecture 9)