COMP 7990
Principles and Practices of Data Analytics

Lecture 2: Multivariate Linear Regression, Perception and Artificial Neural Network

Dr. Kevin Wang

Slide prepared by: Dr. Eric Lu Zhang

What we learned last week?

- **Data Preprocessing**
- **Supervised learning**
- Regression
  1. Linear regression with one variable
  2. Linear Regression with multiple variables
- Classification
  1. Perceptron
  2. Artificial Neural Network
  3. Support Vector Machine
  4. K Nearest Neighbor
- **Unsupervised learning**
  1. K-means Clustering
  2. Hierarchical Clustering

Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**
- **Supervised learning**

❖ Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables

❖ Classification

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

Multiple variables

Size ($feet^2$)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

Notation:

n = number of variables (n=4)

$x^{(i)}$ = input (variable) of i^{th} training example.

$x_j^{(i)}$ = value of variable j in i^{th} training example.

Hypothesis

Previously: $h_\theta(x) = \theta_0 + \theta_1 x$

Multivariate linear regression:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Gradient descent

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$\ldots$

Matrix Multiplication

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

$\rightarrow 1 \times 7 + 2 \times 9 + 3 \times 11$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

Matrix Transpose

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Input

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Output

Matrix Inverse

The product of A and its inverse is the identity:

$$\begin{bmatrix} -3 & 1 \\ 5 & 0 \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{5} \\ 1 & \frac{3}{5} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

matrix A

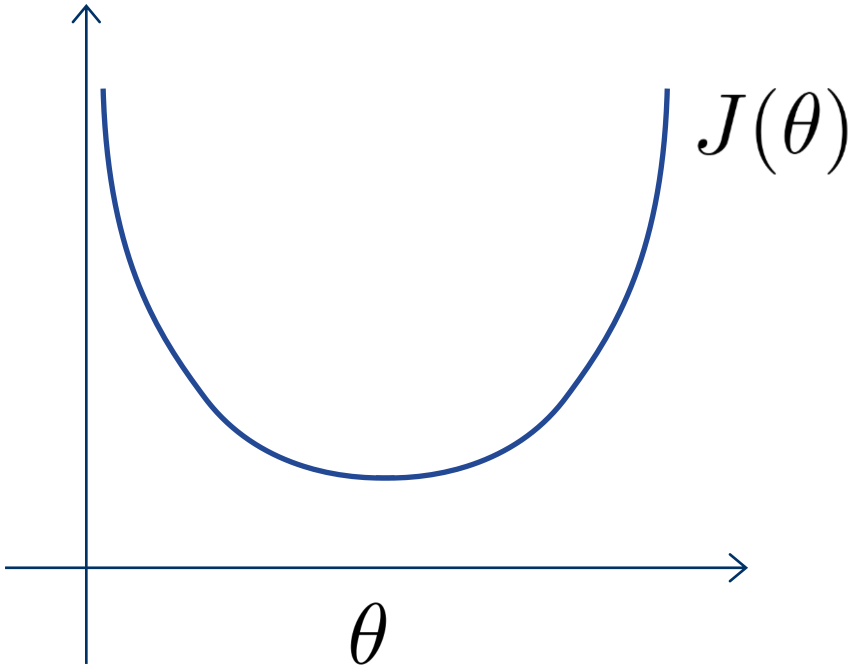
matrix A^{-1}

2 x 2
identity matrix

Normal equation

Gradient Descent:
Method to solve for θ numerically.

Normal equation:
Method to solve for θ analytically.



Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$

$$\theta = (X^T X)^{-1} X^T y$$

Normal equation (optional)

Linear Regression in Matrix Format

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x^{(11)} & \cdots & x^{(1n)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x^{(m1)} & \cdots & x^{(mn)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

- **y:** m*1**; X:** m*(n+1); θ : (n+1)*1
- The weighted sum of squared errors can be written as

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(X^{(i)}\theta^T - y^{(i)})^2 = \frac{1}{2m}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$

Linear Regression Solution (optional)

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(X^{(i)}\theta^T - y^{(i)})^2 = \frac{1}{2m}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$

- Taking derivative with respect to θ, and equating to zero, we get

$$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{m}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\theta) = 0 \Longrightarrow \mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}$$

$$\Longrightarrow \theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Gradient descent and normal equation

m **training cases**, n **variables.**

Gradient Descent

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α.
- Don't need to iterate.
- Need to compute $(X^TX)^{-1}$
- Slow if n is very large.

Normal equation (optional)

Examples: $m = 4$.

x_0	Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

Example (optional)

Student	Test score	IQ	Study hours
1	100	110	40
2	90	120	30
3	80	100	20
4	70	90	0
5	60	80	10

develop a regression equation to predict test scores (y), based on students'IQs (x_1) and the number of hours that the student studied (x_2).

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

(optional)

$$\mathbf{X} = \begin{bmatrix} 1 & 110 & 40 \\ 1 & 120 & 30 \\ 1 & 100 & 20 \\ 1 & 90 & 0 \\ 1 & 80 & 10 \end{bmatrix}$$

$$\mathbf{X}^{\mathbf{T}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 110 & 120 & 100 & 90 & 80 \\ 40 & 30 & 20 & 0 & 10 \end{bmatrix}$$

$$\mathbf{X}^{\mathbf{T}}\,\mathbf{X} = \begin{bmatrix} 5 & 500 & 100 \\ 500 & 51{,}000 & 10{,}800 \\ 100 & 10{,}800 & 3{,}000 \end{bmatrix}$$

$$(\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1} = \begin{bmatrix} 101/5 & -7/30 & 1/6 \\ -7/30 & 1/360 & -1/450 \\ 1/6 & -1/450 & 1/360 \end{bmatrix}$$

Calculation Steps (optional)

$$\mathbf{Y} = \begin{bmatrix} 100 \\ 90 \\ 80 \\ 70 \\ 60 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 20 \\ 0.5 \\ 0.5 \end{bmatrix}$$

$$y = 20 + 0.5x_1 + 0.5x_2$$

Outline for data analytics and data mining

- Data Preprocessing
- **Supervised learning**

❖ Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables

❖ Classification

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

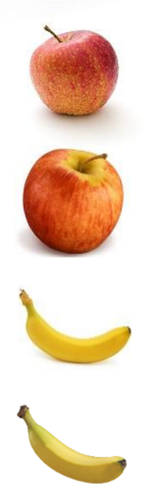
- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

An Example of Classification Problem

- Learn to recognize apple or banana

Training Data with Class Label



Label

apple

apple

banana

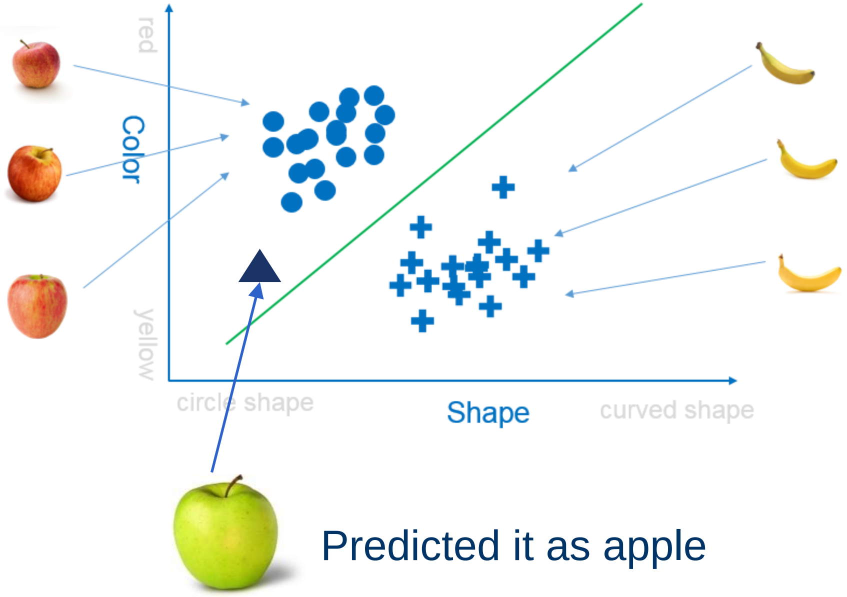
banana

… …

Feature Engineering

color(x_1)	shape (x_2)	y
0.8	0.1	1
0.75	0.12	1
0.12	0.6	0
0.2	0.8	0
…	…	…
…	…	…

Model Building

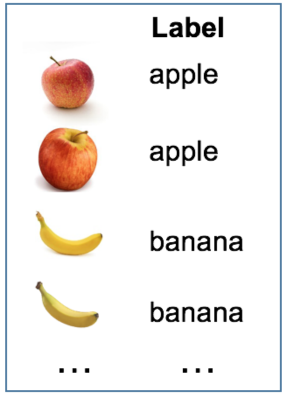


Predicted it as apple

Test Data

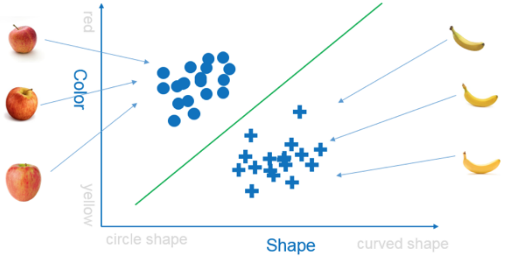
To predict it is apple or banana

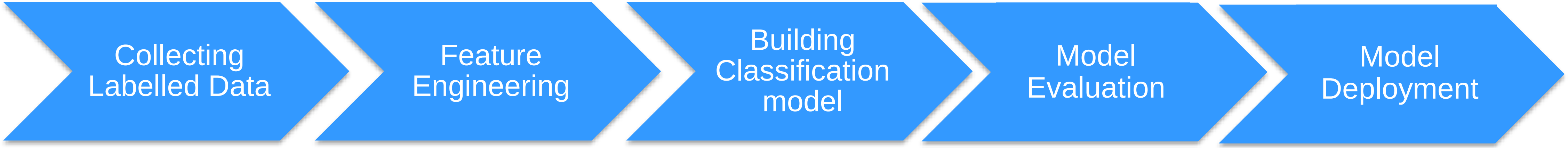
A Typical Classification Pipeline



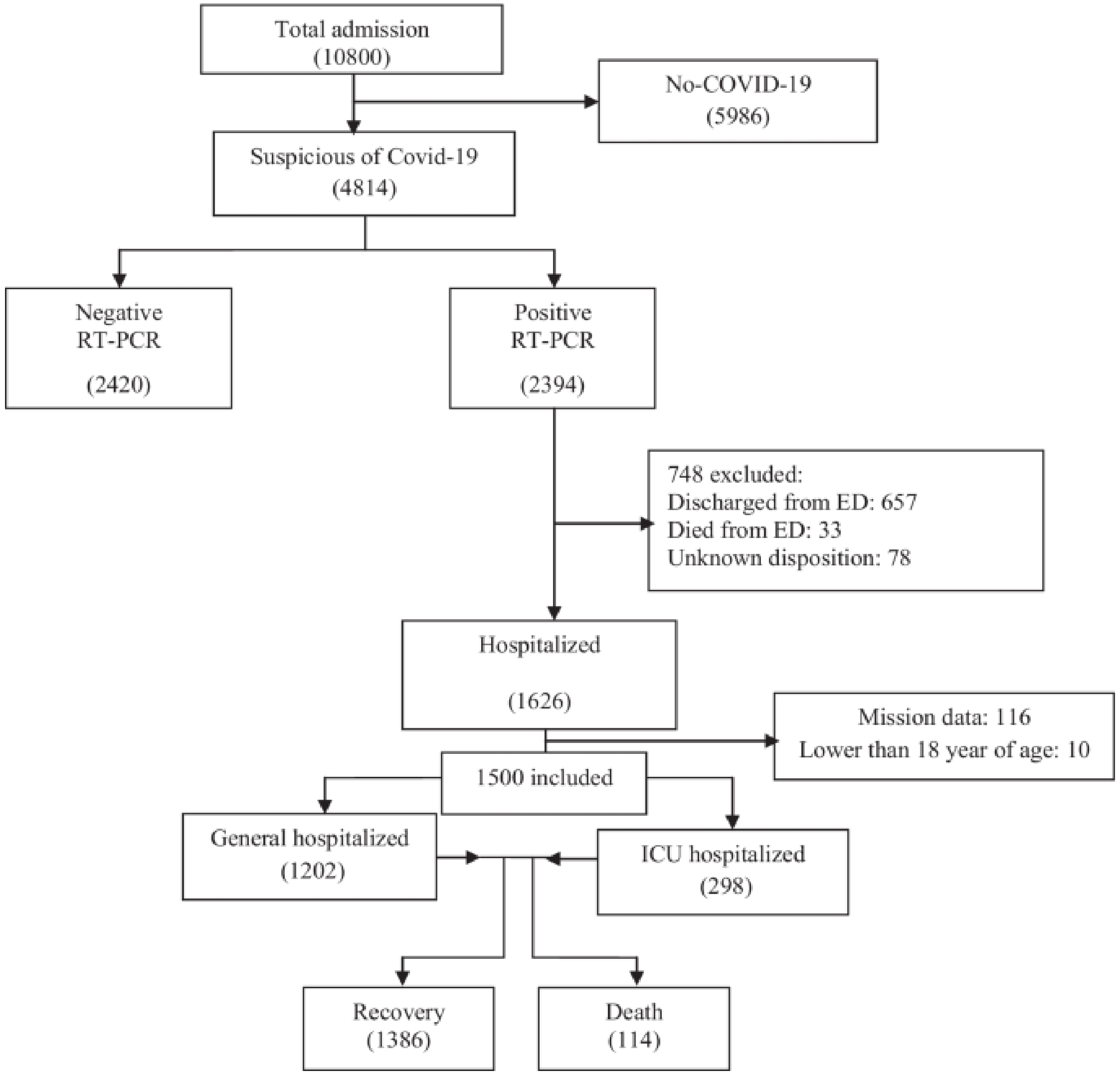
Feature Engineering

Color(x1)	Shape (x2)	y
0.8	0.1	1
0.75	0.12	1
0.12	0.6	0
0.2	0.8	0
…	…	…
…	…	…





Application: Covid-19 Mortality Prediction (Data Preprocessing)



https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-021-01742-0#Tab3

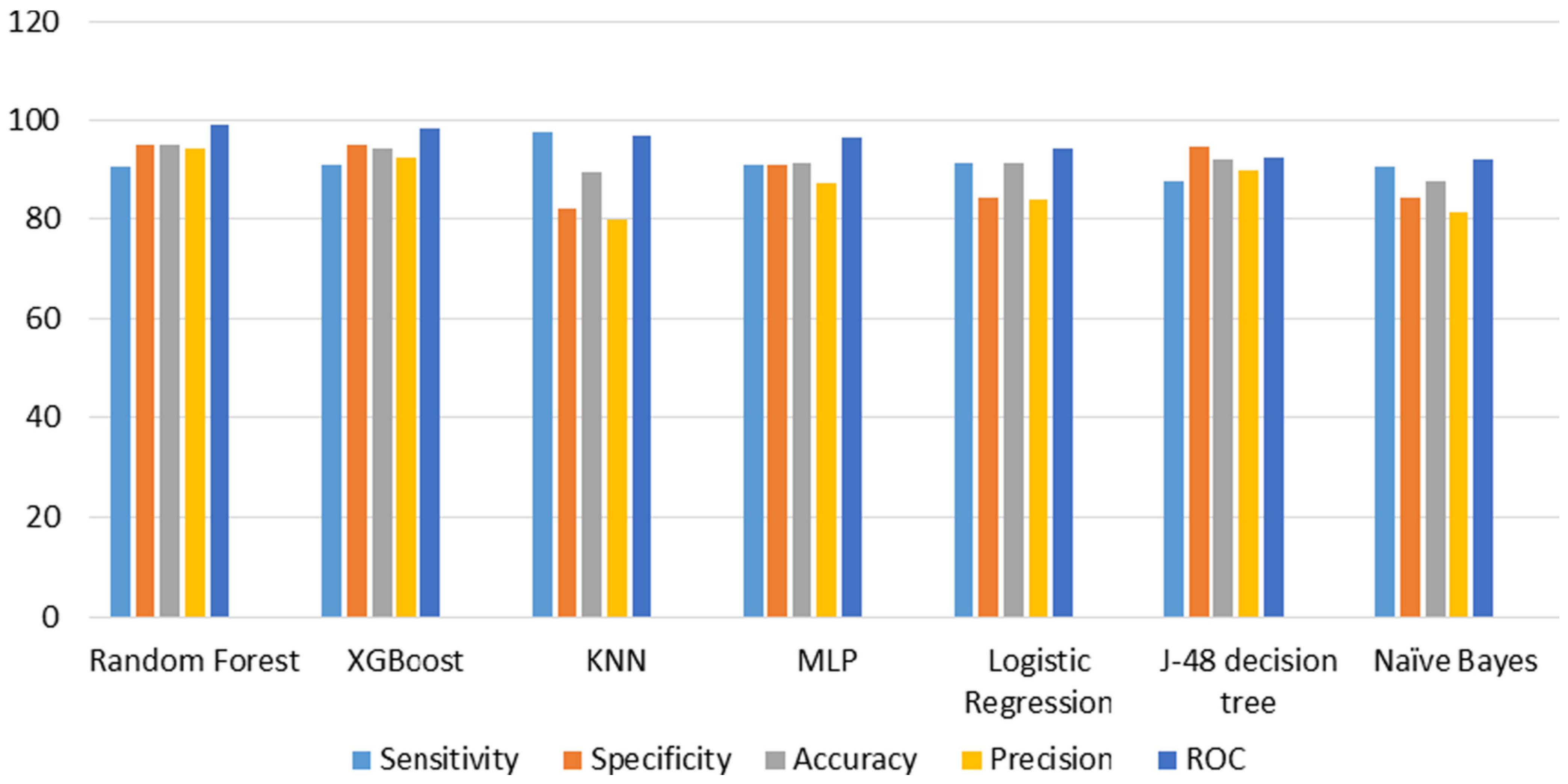
Application: Covid-19 Mortality Prediction (Feature Selection)

Row	Features name	Degree of importance	Row	Features name	Degree of importance
1	Dyspnea	0.5532	21	Chest pain and pressure	0.2256
2	ICU admission	0.5409	22	Absolute neutrophil count	0.2123
3	Oxygen therapy	0.3789	23	Headache	0.1992
4	Age	0.3207	24	Gender	0.186
5	Fever	0.3142	25	Gastrointestinal symptoms	0.1802
6	Cough	0.3072	26	White cell count	0.1702
7	Loss of taste	0.2944	27	C-reactive protein	0.1574
8	Loss of smell	0.2923	28	Hypersensitive troponin	0.1428
9	Hypertension	0.2768	29	Pneumonia	0.1066
10	Contusion	0.2744	30	Glucose	0.0906
11	Muscular Pain	0.2731	31	Erythrocyte sedimentation rate	0.0826
12	Chill	0.2537	32	Creatinine	0.0716
13	Runny noise	0.2532	33	Alkaline phosphatase	0.0678
14	Blood urea nitrogen	0.2524	34	Length of hospitalization	0.0626
15	Diabetes	0.2506	35	Aspartate aminotransferase	0.0445
16	Sore throat	0.25	36	Smoking	0.0427
17	Absolute lymphocyte count	0.2339	37	Alanine aminotransferase	0.0319
18	Nausea/vomiting	0.2301	38	Platelet count	0.0210
19	Other under line disease	0.2282	39		
20	Cardiac disease	0.2274			

Application: Covid-19 Mortality Prediction (Feature Statistics)

Features (quantitative)	Range	Mean (SD)
Age (year)	18–100	57.25 (17.8)
Leng of hospitalization	1–32	61.89 (13.25)
Creatinine (mg/dL)	0.1–17.9	51.39 (14.4)
White-cell count	1300–63,000	82.34 (4897.4)
Platelet count	108,000–691,000	66.2 (38.1)
Absolute lymphocyte count	2–95	23.74 (11.8)
Absolute neutrophil count	8–98	74.52 (12.3)
Blood urea nitrogen	0.5–251	42.52 (31.7)
Aspartate aminotransferase	3.8–924	44.45 (53.5)
Alanine aminotransferase	2–672	38.29 (41.6)
Glucose	18–994	36.09 (74.2)
Lactate dehydrogenase	4.6–6973	55.68 (339.0)
Prothrombin time	0.9–46.8	42.82 (23.9)
Alkaline phosphatase	9.6–2846	21.12 (39.2)
Erythrocyte sedimentation rate	2–258	40.65 (28.8)

Application: Covid-19 Mortality Prediction (Model Selection)

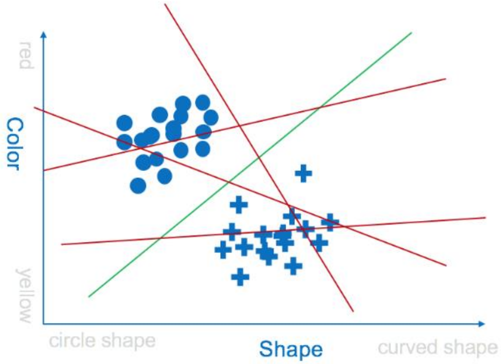


Application: Covid-19 Mortality Prediction (Benefit)

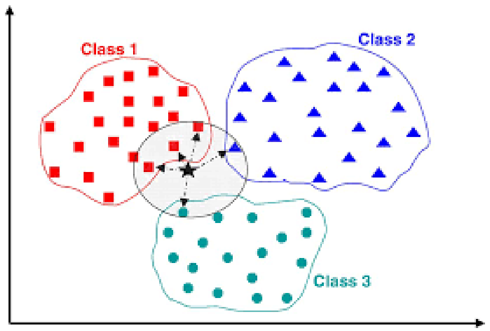
- Optimal use of hospital resources for
  - treating the patients with more critical conditions and
  - assisting in providing more qualitative care and
  - reducing medical errors due to fatigue and long working hours in the ICU

Overview of Classification Models

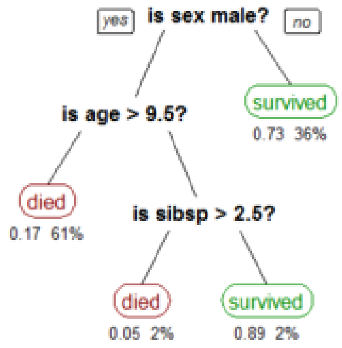
- Simple Models



Perceptron

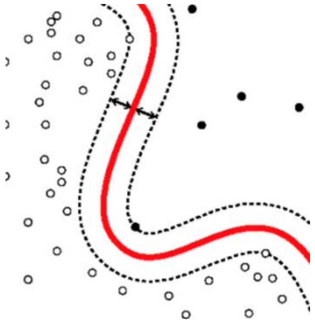


K nearest Neighbors

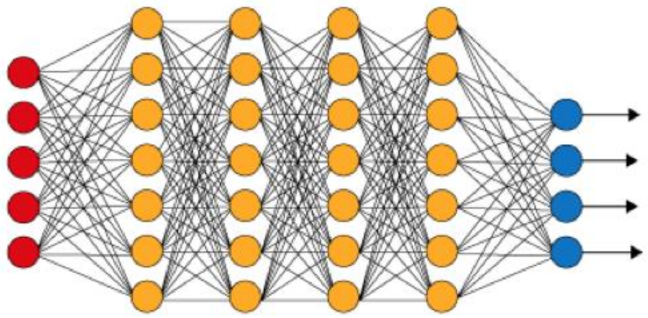


Decision Tree

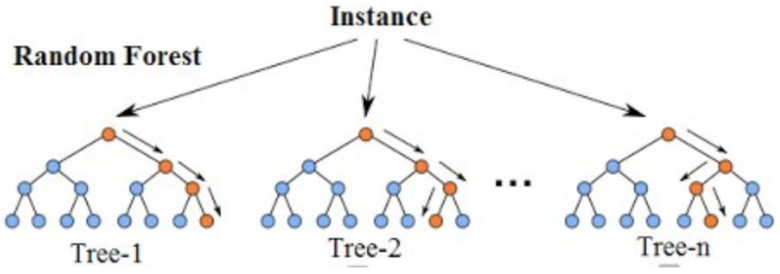
- Complex Models



Support Vector Machine



(Deep) Neural Networks



Ensemble methods:
Random Forest; Gradient Boosting Tree

Notation

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.3	3.3	6	2.5	virginica
…	…	…	…	…

X **y**

- **X**: the *m*n* feature matrix
  - *m*: the number of data samples
  - *n*: the dimensionality of each data sample
- **y**: *m*1* label vector
- **X**(*i*,:): the *i*-th row in matrix **X**
  - i.e., the *i*-th data sample
- $\mathbf{x}_i$: the *i*-th sample in column vector representation
  - $\mathbf{x}_i = \mathbf{X}(i,:)^T$
- $x_{(i,j)}$: the *j*-th feature of the *i*-th sample
- y_i: the label of the *i*-th sample

Capital bold letter for matrix, small bold letter for vector, small (italic) letter for scalar. Vectors are column vectors

Notation

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.3	3.3	6	2.5	virginica
…	…	…	…	…

X

y

x_{11}

x_{13}

y_1

$\mathbf{X}(2,:)$ = [4.9, 3, 1.4, 0.2] it is a row vector

$$\mathbf{x}_2 = \begin{bmatrix} 4.9 \\ 3 \\ 1.4 \\ 0.2 \end{bmatrix}$$

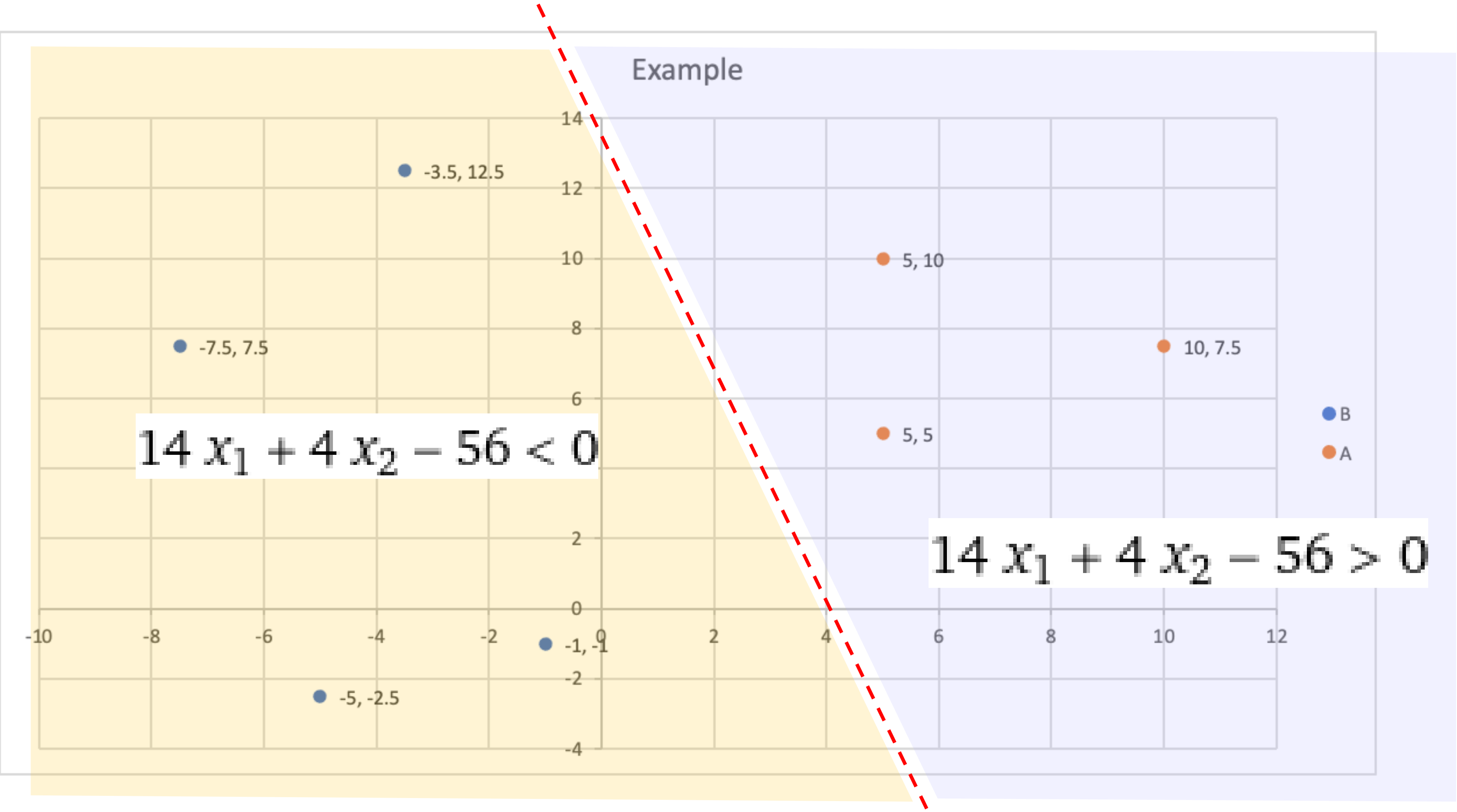
The subscript T means transpose.

Both of them are presenting the second sample for X. $\mathbf{x}_i = \mathbf{X}(i,:)^T$

Linear Classification Task – An Illustrated Example

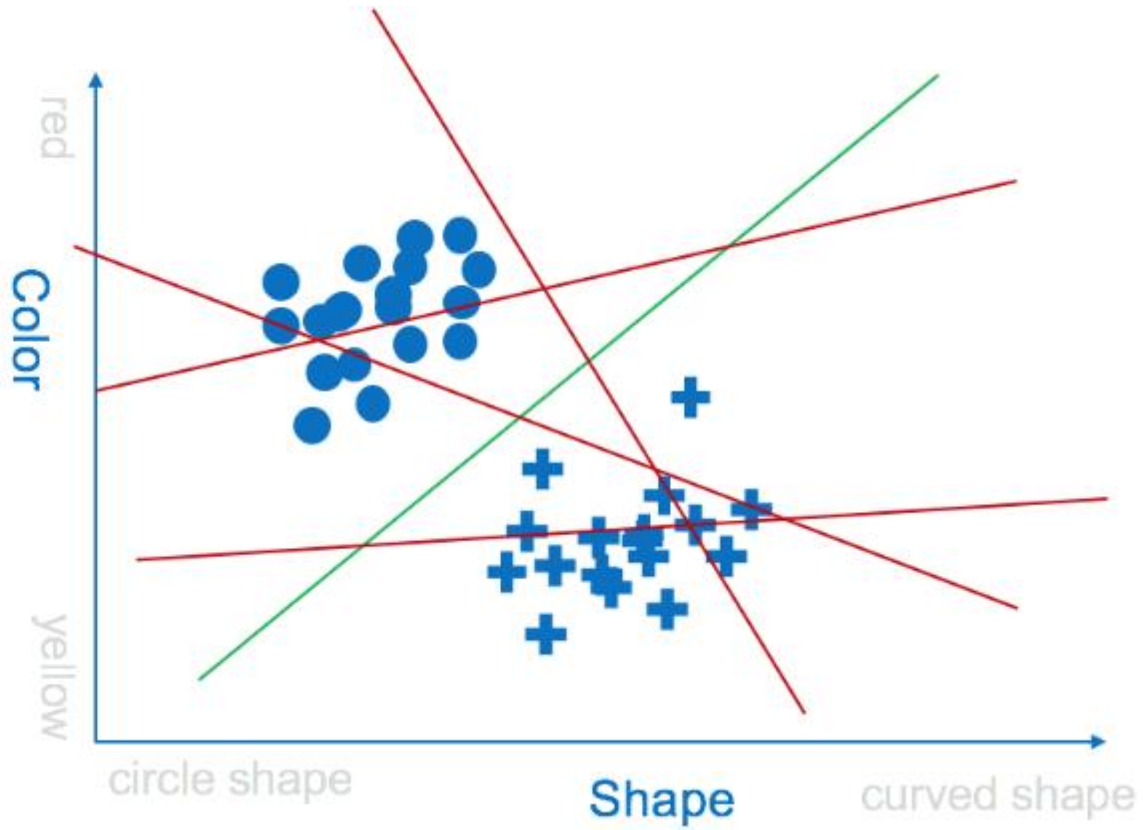
How about non-linear?

x1	x2	y
-1	-1	0
-5	-2.5	0
-7.5	7.5	0
10	7.5	1
-2.5	12.5	0
5	10	1
5	5	1



$$\boxed{14}\,x_1 + \boxed{4}\,x_2 - \boxed{56} = 0$$

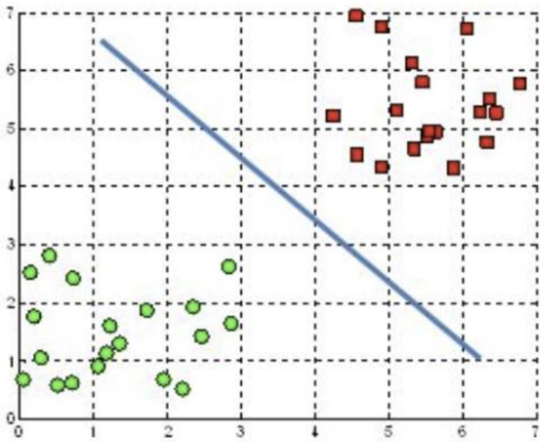
Perceptron Algorithm



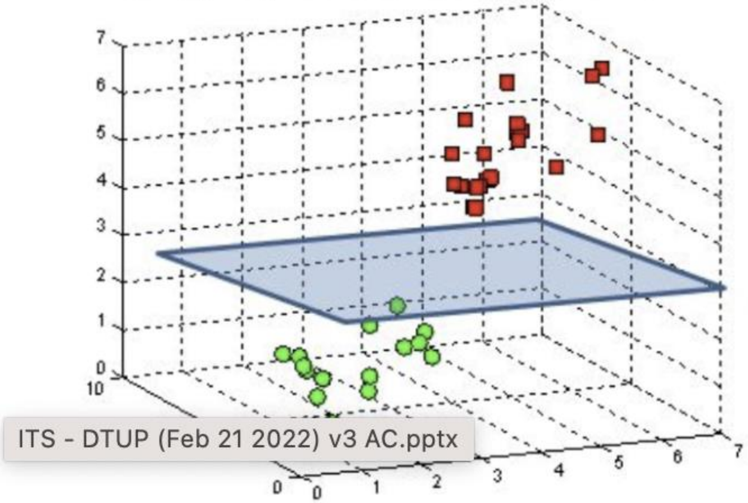
Introduction to Perceptron Algorithm

- One of the earliest algorithms for linear classification (Rosenblatt, 1958)
- Try to find a hyperplane separating the labeled data
- Guaranteed to find a separating hyperplane if the data is linearly separable

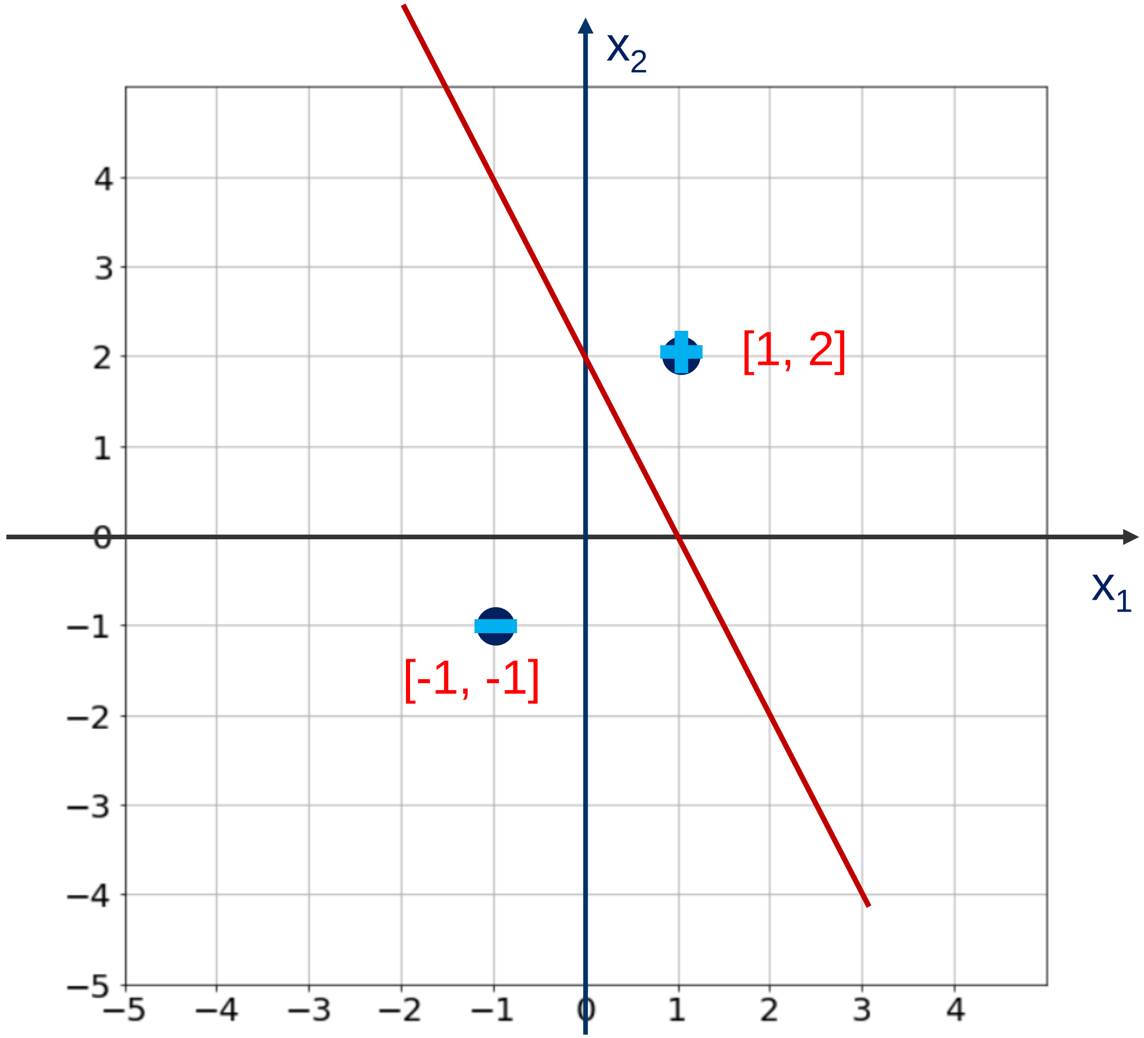
A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



Linear Decision Boundary for Classification: Example



- What is the formula for this linear boundary?
  - **$x_2 = -2x_1+2 \Rightarrow 2x_1+x_2-2 = 0$**
  - General form:

    $$f(\mathbf{x}) = \sum_{j=1}^{d} w_j x_j + b = \mathbf{w}^T\mathbf{x} + b = 0$$

- What label would we predict for a new data point **x**?
  - [1, 2]
    - 2*1+2-2=2 > 0
    - predicted it as **positive**
  - [-1, -1]
    - 2*-1+-1-2 = -5 < 0
    - Predicted it as **negative**

Introduction to Perceptron Algorithm

- Problem Definition
  - Given a training dataset $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$, where $\mathbf{x}_i$ is a *n* dimensional input feature vector, $y_i \in \{-1,1\}$ is the corresponding class label.
  - Objective: Train a linear classifier that can separate positive and negative samples.
  - **Training:** Learn a linear classification function $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b$ from training data
    - $y = 1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b > 0$
    - $y = -1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b < 0$
    - w_j, b are the model parameters that we need to learn from training data
  - **Prediction:** for any new input **x,** predict its class label as $y = \mathrm{sign}(f(\mathbf{x}))$
- Simplify the notation
  - By introducing an artificial feature $x_0 = 1$, $\mathbf{x} \to [x_0, x_1, \ldots, x_n]$, $f(\mathbf{x})$ can be rewritten in vector form as

$$f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b = \sum_{j=1}^{n} w_j x_j + b x_0 = \sum_{j=0}^{n} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Introduction to Perceptron Algorithm

Initialize **w = 0**

Repeat

if $\boxed{y_i(\mathbf{w}^T\mathbf{x}_i) \le 0}$ then

$\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$

end if

Iteratively do prediction on the training data,

If the current data point is correctly classified

do nothing

If the current point is wrongly classified

Update the model

$y_i(\mathbf{w}^T\mathbf{x}_i) \le 0$ means the training data point $\mathbf{x}_i$ is misclassified.

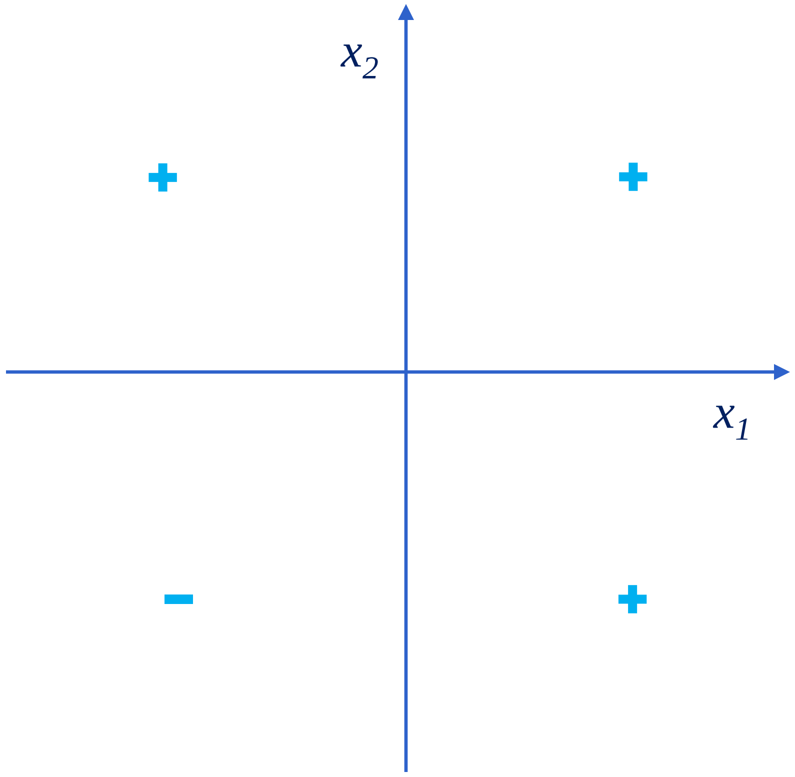
- true label $y_i = 1$, but the prediction $\text{sign}(\mathbf{w}^T\mathbf{x}) = -1$, or
- true label $y_i = -1$, but the prediction $\text{sign}(\mathbf{w}^T\mathbf{x}) = 1$

Perceptron: An Illustrated Example

Suppose we have a small training data with only 4 labelled data samples. Each data sample has only two features.

x_1	x_2	y
1	-1	1
1	1	1
-1	1	1
-1	-1	-1

x_0	x_1	x_2	y
1	1	-1	1
1	1	1	1
1	-1	1	1
1	-1	-1	-1



An Illustrated Example

- Initialize $\mathbf{w}$ = [0, 0, 0]
- Cyclically go through the data

x_0	x_1	x_2	y
1	1	-1	1
1	1	1	1
1	-1	1	1
1	-1	-1	-1

The 1st pass

([1, 1, -1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (0 * 1 + 0 * 1 + 0 * (-1)) = 0 \leq 0$

$\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i = [0, 0, 0] + 1 * [1, 1, -1] = [1, 1, -1]$

([1, 1, 1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + (-1) * 1) = 1 > 0$

do not update **w**

([1, -1, 1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (1 * 1 + 1 * (-1) + (-1) * 1) = -1 \leq 0$

$\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i = [1, 1, -1] + 1 * [1, -1, 1] = [2, 0, 0]$

([1, -1,-1],-1): $y_i(\mathbf{w}^T\mathbf{x}_i) = -1 * (2 * 1 + 0 * (-1) + 0 * (-1)) = 0 \leq 0$

$\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i = [2, 0, 0] + (-1) * [1, -1, -1] = [1, 1, 1]$

The 2nd pass

([1,1, -1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + 1 * (-1)) = 1 > 0$

do not update **w**

([1,1, 1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (1 * 1 + 1 * 1 + 1 * (1)) = 3 > 0$

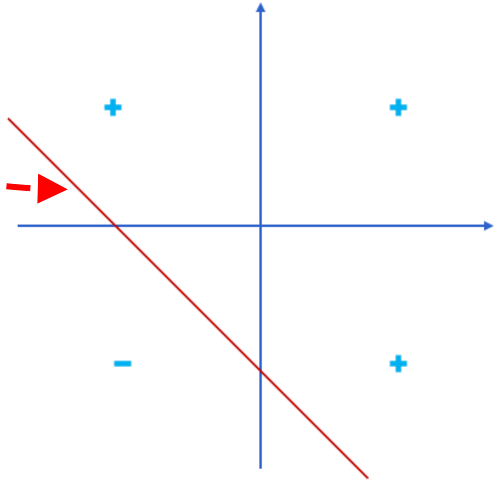
do not update **w**

([1,-1, 1], 1): $y_i(\mathbf{w}^T\mathbf{x}_i) = 1 * (1 * 1 + 1 * (-1) + 1 * (1)) = 1 > 0$

do not update **w**

([1,-1,-1],-1): $y_i(\mathbf{w}^T\mathbf{x}_i) = -1 * (1 * 1 + 1 * (-1) + 1 * (-1)) = 1 > 0$

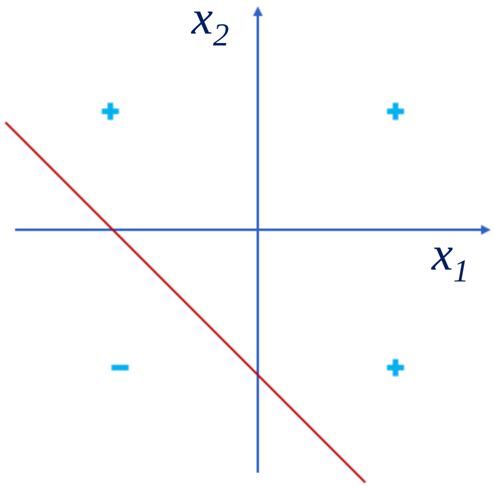
do not update **w**

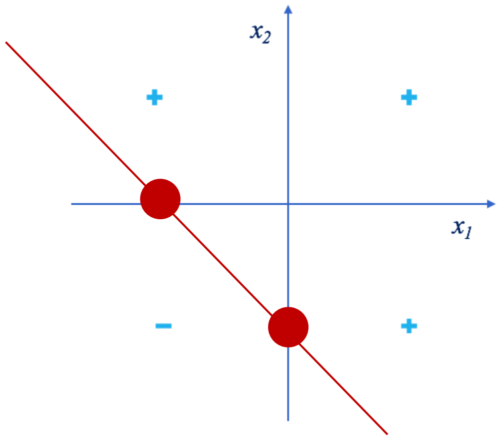


How to Plot the Decision Boundary?

- The learnt **w** by using perceptron algorithm is [1, 1, 1], then the corresponding decision boundary is $w_0 * x_0 + w_1 * x_1 + w_2 * x_2 = 0$.
- We know that w_0 is 1 and x_0 is an artificial feature which always equal to 1. Also we know $w_1 = 1$ and $w_2 = 1$. Therefore, the decision boundary is:

  $1 + x_1 + x_2 = 0 \rightarrow x_1 + x_2 = -1$
- The line (i.e. decision boundary) can be plotted by connecting two data points lies on the line.
  - Suppose $x_1 = 0$, then $x_2 = -1$
  - Suppose $x_2 = 0$ then $x_1 = -1$



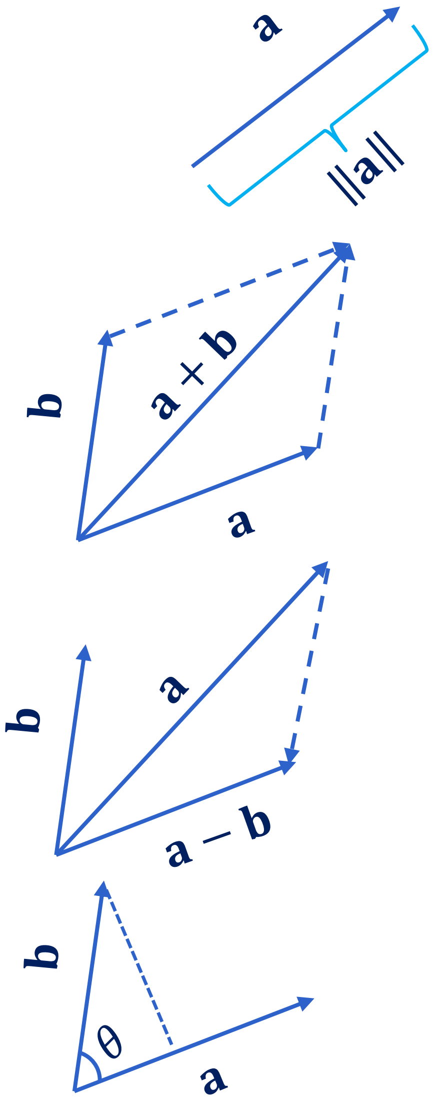


Why perceptron works?

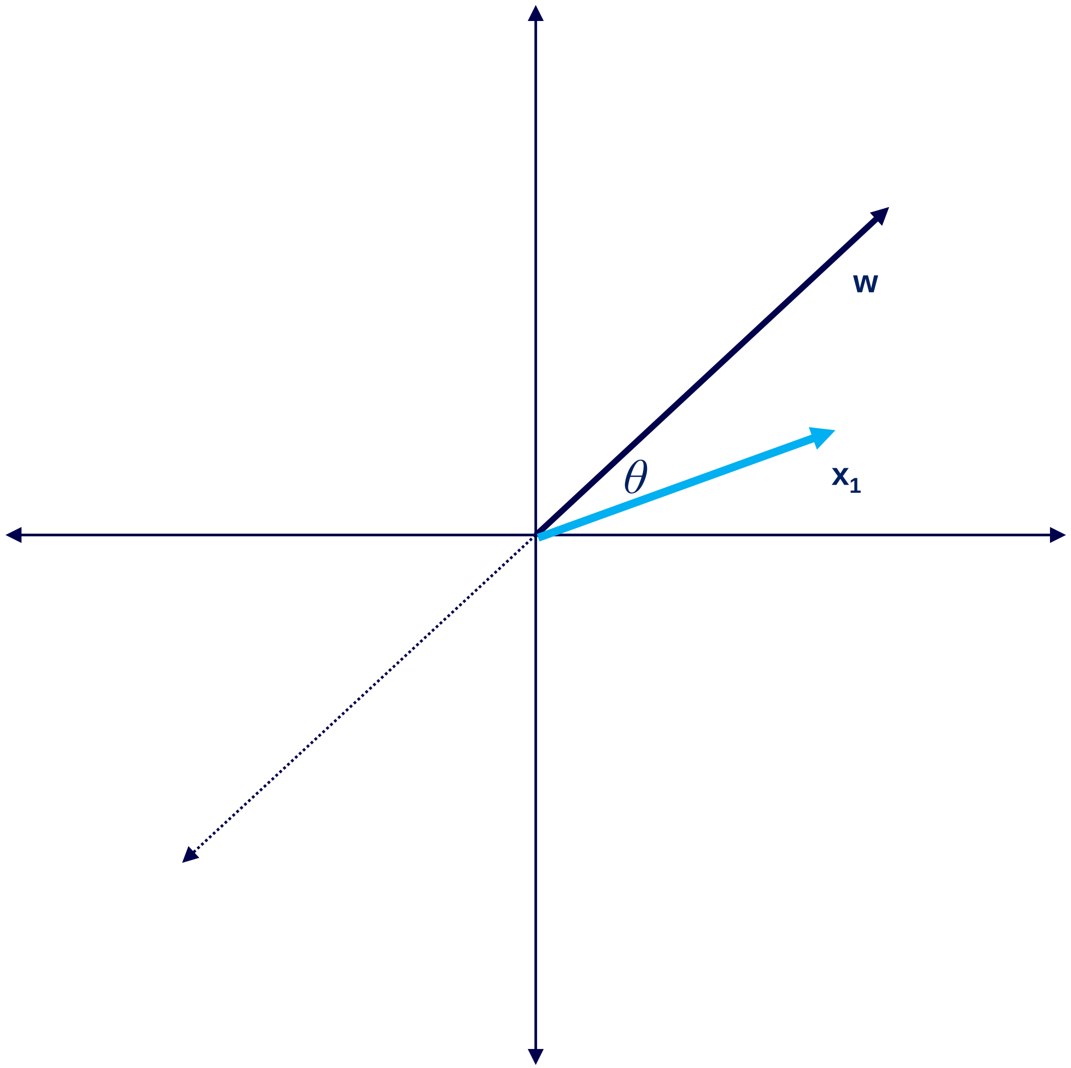
- Some preliminary on vectors
  - The magnitude $\|\mathbf{a}\|$ of a vector $\mathbf{a} = (a_1, a_2)$ is

    $$\|\mathbf{a}\| = \sqrt{a_1^2 + a_2^2}$$

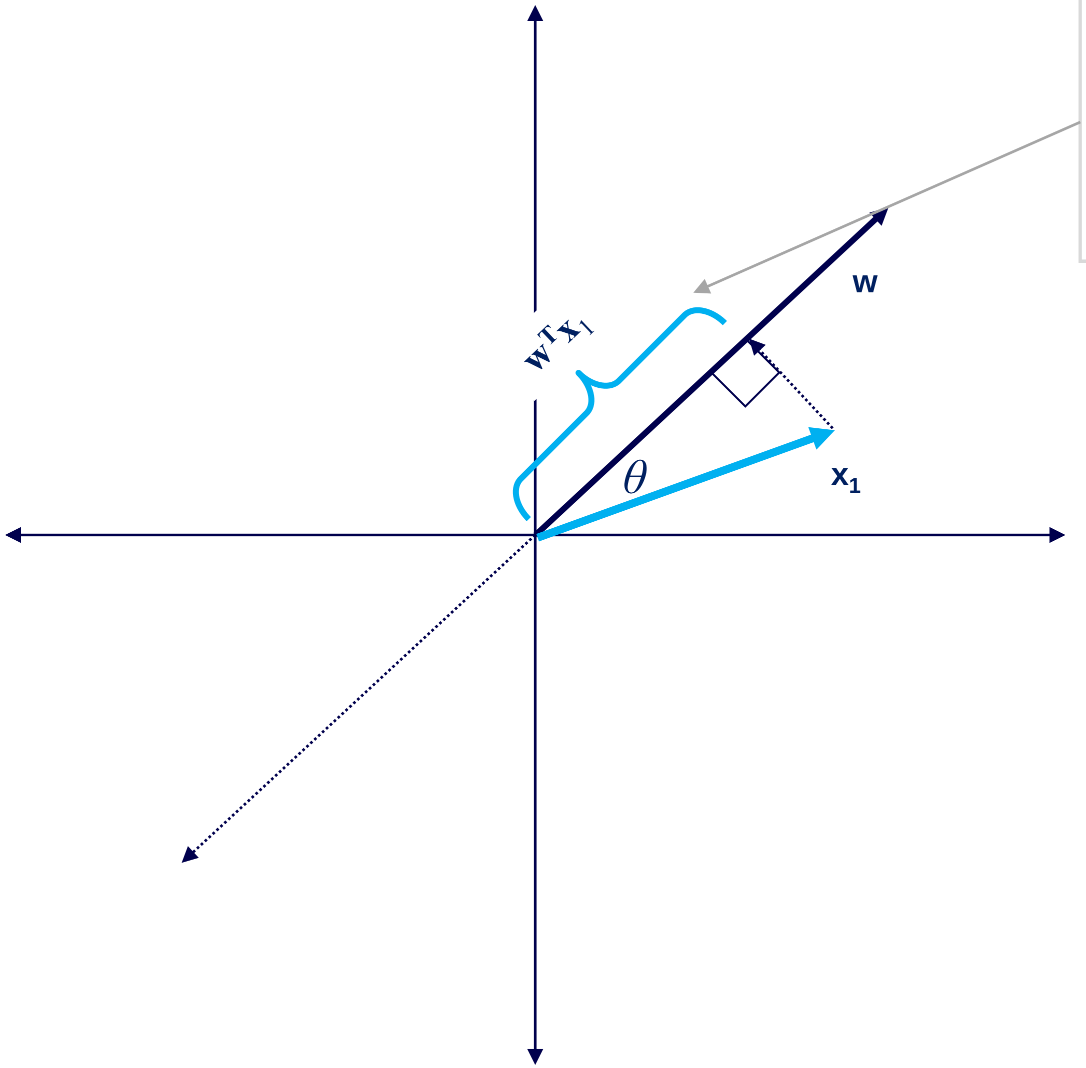
  - Vector addition: $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$
    - ➢ $\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2)$
  - Vector subtraction: $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$
    - ➢ $\mathbf{a} - \mathbf{b} = (a_1 - b_1, a_2 - b_2)$
  - Dot product: $\mathbf{a} = (a_1, a_2)$, $\mathbf{b} = (b_1, b_2)$
    - ➢ $\mathbf{a} \cdot \mathbf{b} = \mathbf{ab} = a_1 b_1 + a_2 b_2 = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta$



Why Perceptron Works?

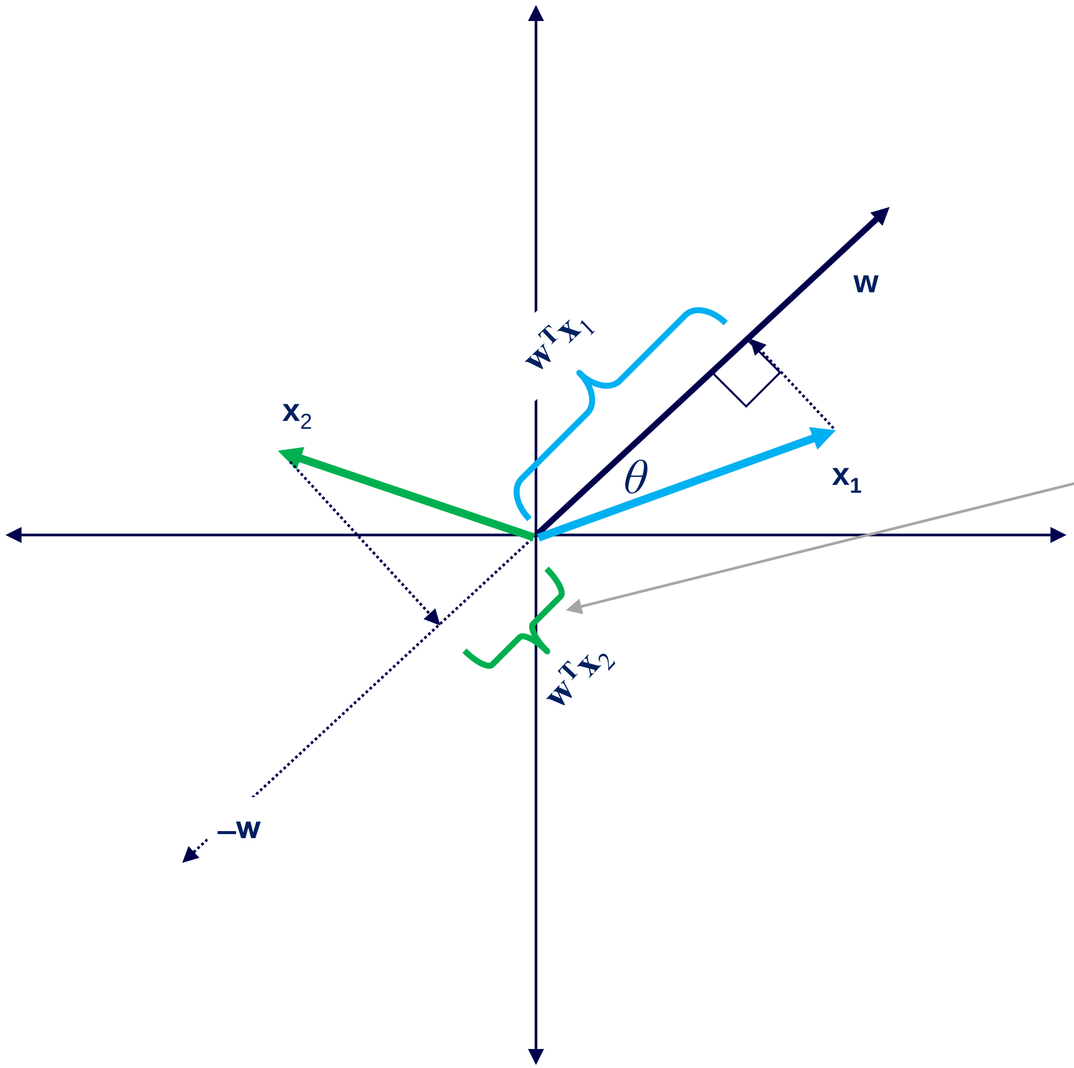


Why Perceptron Works?



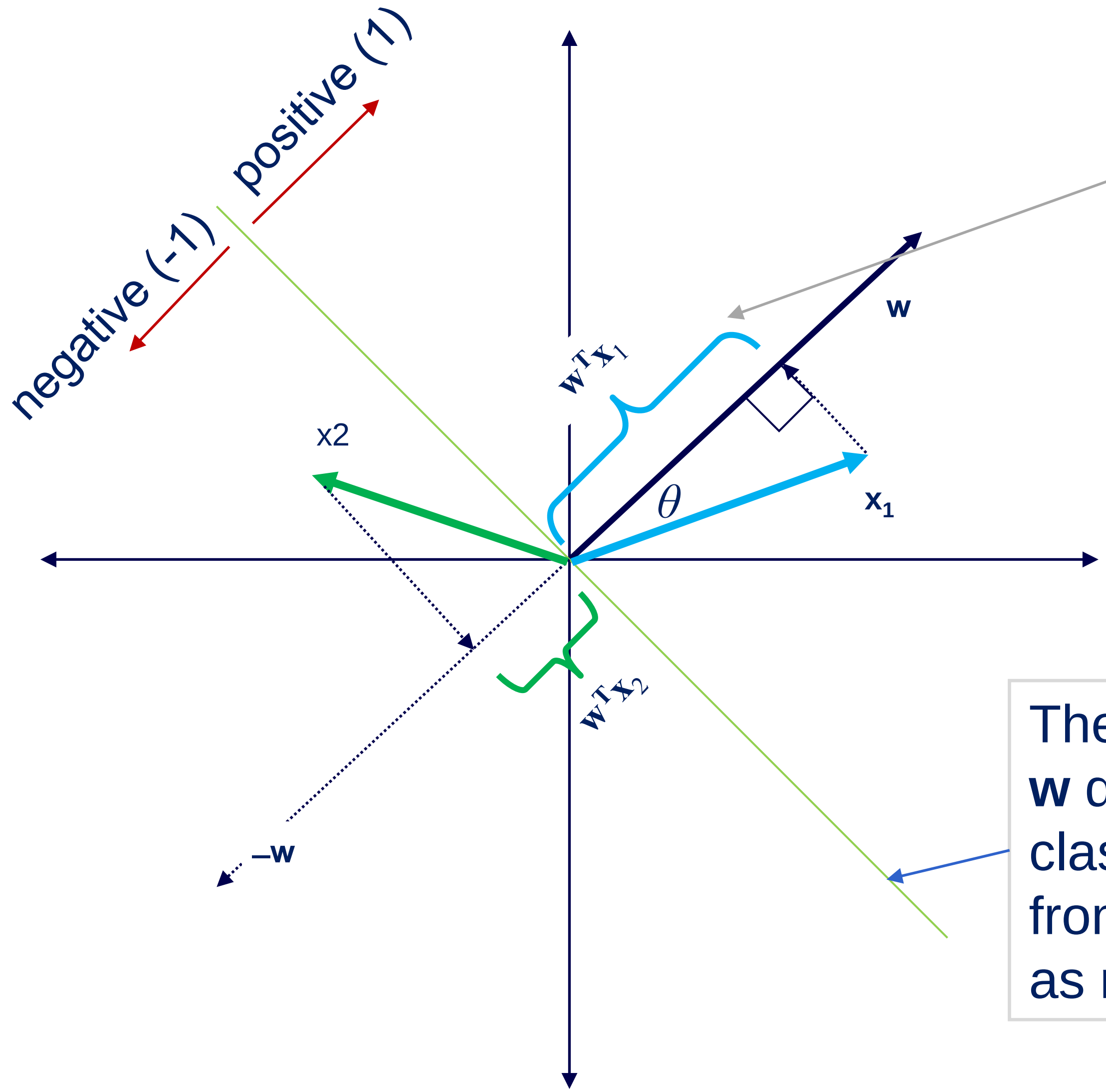
Without loss of generality, we assume $||\mathbf{w}|| = 1$.
Visually, $\mathbf{wx} = ||\mathbf{w}||||\mathbf{x}||\cos\theta$ is the scalar value you get if you "project **x** onto **w**".
If the value is positive, predict it as positive.

Why Perceptron Works?



Without loss of generality, we assume $||\mathbf{w}|| = 1$.
Visually, $\mathbf{wx} = ||\mathbf{w}||||\mathbf{x}||\cos\theta$ is the scalar value you get if you "project **x** onto **w**".
If the value is positive, predict it as positive.
If the value is negative, predict is as negative.

Why Perceptron Works?



Without loss of generality, we assume ||**w**|| = 1.
Visually, $\mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|\|\mathbf{x}\|\cos\theta$ is the scalar value you get if you “project **x** onto **w**”.
If the value is positive, predict it as positive.
If the value is negative, predict is as negative.

The line perpendicular to **w** divides the vectors classified as positive (1) from the vectors classified as negative -1.

Why Perceptron Works?

- Consider a positive data point (y_i = 1) was wrongly classified as negative
  - Perceptron predict as -1 since $\mathbf{w}^T\mathbf{x}_i < 0$

- Model Update

  $\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + y_i\mathbf{x}_i$

  Note that,

$$
\begin{aligned}
\mathbf{w}_{new}^T\mathbf{x}_i &= (\mathbf{w}_{old} + y_i\mathbf{x}_i)^T\mathbf{x}_i \\
&= \mathbf{w}_{old}^T\mathbf{x}_i + \boxed{\mathbf{x}_i^T\mathbf{x}_i}
\end{aligned}
$$

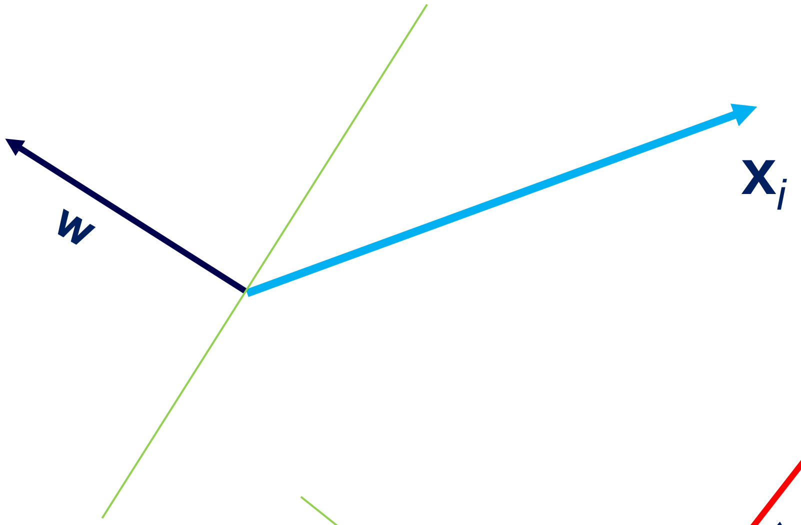
$\|\mathbf{x}_i\|_2^2 > 0$

- Therefore, $\mathbf{w}_{new}^T\mathbf{x}_i$ is **less negative** than $\mathbf{w}_{old}^T\mathbf{x}_i$
  - The update makes the classifier more correct on this data point $(\mathbf{x}_i, y_i)$

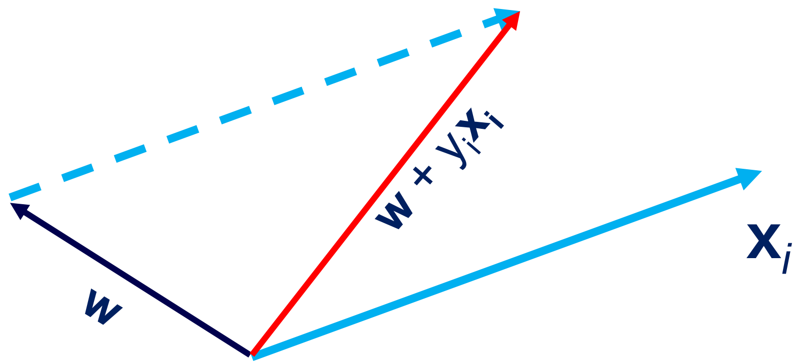
Why Perceptron Works? (Visually)

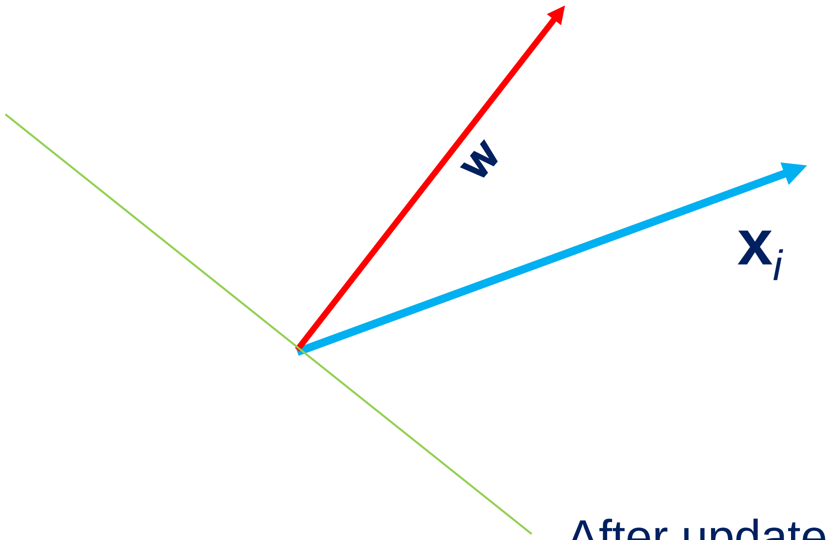
- Consider a positive data point was wrongly classified as negative

$y_i = 1$



$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$$





After update **w** to **w** + $y_i\mathbf{x}_i$, this positive data point will be correctly classified as positive

After update

Why Perceptron Works?

- Consider a negative data point ($y_i = -1$) was wrongly classified as positive
  - Perceptron predict as 1 since $\mathbf{w}^T\mathbf{x}_i > 0$

- Model Update

  $\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + y_i\mathbf{x}_i$

  Note that,

$$\begin{aligned}\mathbf{w}_{new}^T\mathbf{x}_i &= (\mathbf{w}_{old} + y_i\mathbf{x}_i)^T\mathbf{x}_i \\ &= \mathbf{w}_{old}^T\mathbf{x}_i - \boxed{\mathbf{x}_i^T\mathbf{x}_i}\end{aligned}$$

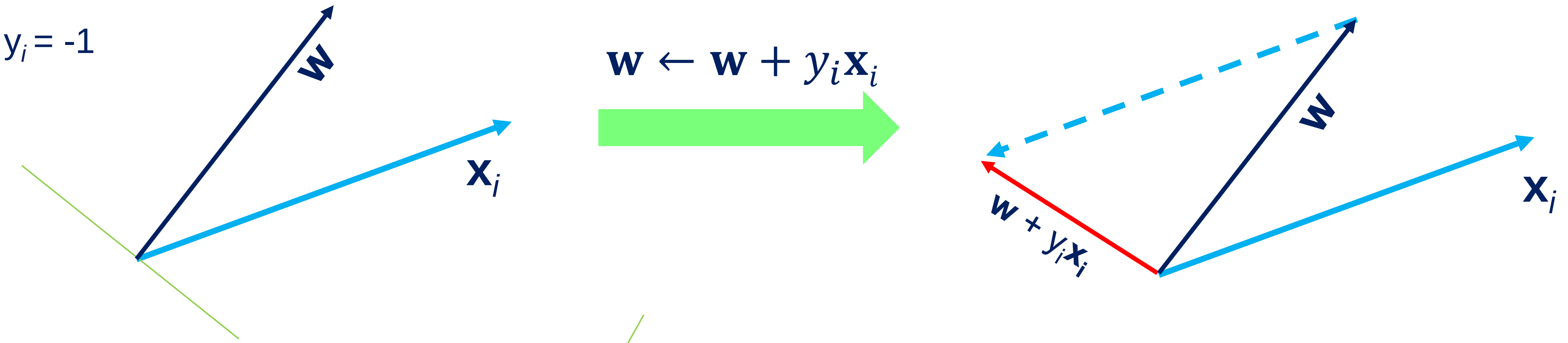
$\|\mathbf{x}_i\|_2^2 > 0$

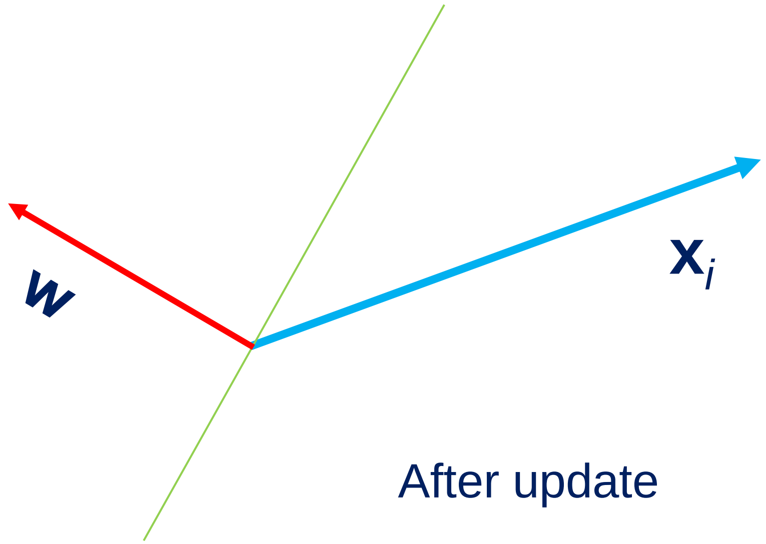
- Therefore, $\mathbf{w}_{new}^T\mathbf{x}_i$ is **less positive** than $\mathbf{w}_{old}^T\mathbf{x}_i$
  - The update makes the classifier more correct on this data point $(\mathbf{x}_i, y_i)$

Why Perceptron Works? (Visually)

- Consider a negative data point was wrongly classified as positive

$y_i = -1$





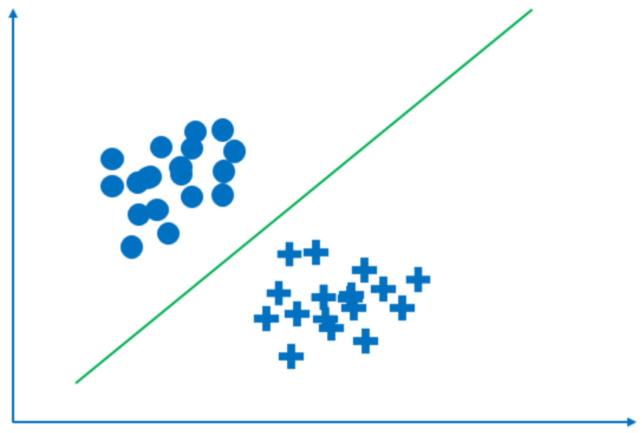
After update **w** to **w** + $y_i\mathbf{x}_i$, this negative data point will be correctly classified as negative

After update

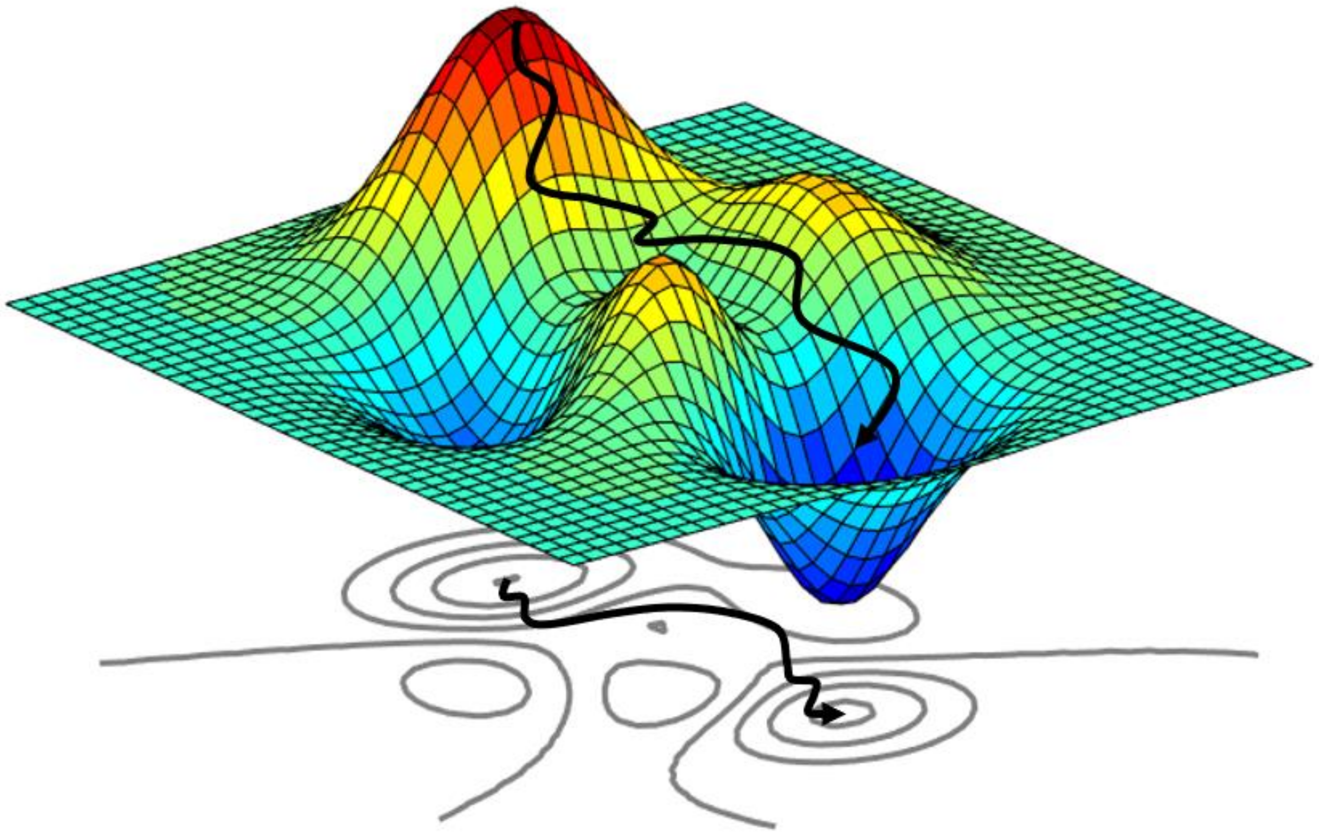
Perceptron: Some Additional Notes

- Given a linearly separable training set $\{\mathbf{x}_i, y_i\}_{i=1}^{n}$, perceptron algorithm finds a classification model with "zero training error".

- We can formulate the goal of learning of the separating hyperplane as an optimization problem (*minimizing a loss function related to the training error*).



Learning as Optimization

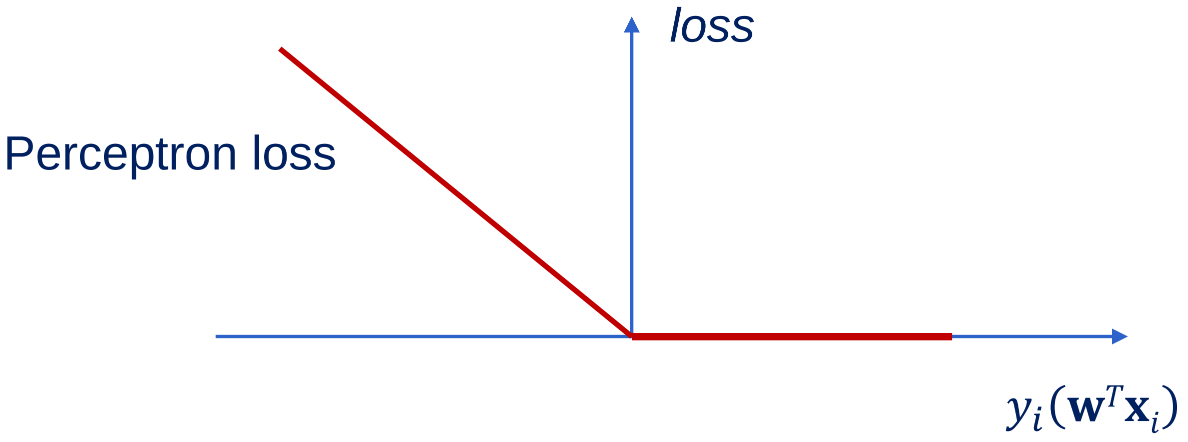


Learning as Optimization

- The loss function of perceptron:

$$l(\mathbf{w}) = \sum_{i=1}^{n} \max\{0, -y_i(\mathbf{w}^T\mathbf{x}_i)\}$$

  - Loss = 0 on samples where Perceptron prediction is correct, i.e., $y_i(\mathbf{w}^T\mathbf{x}_i) > 0$
  - Loss > 0 on samples where Perceptron prediction is wrong, i.e., $y_i(\mathbf{w}^T\mathbf{x}_i) < 0$



Learning as Optimization

- Therefore, the classification model **w** is by *minimizing the perceptron loss function*:

$$\min_{\mathbf{w}} l(\mathbf{w}) = \sum_{i=1}^{n} \max\{0, -y_i(\mathbf{w}^T\mathbf{x}_i)\}$$

learning rate

- Gradient Descent: $\mathbf{w} = \mathbf{w} - \eta \boxed{\frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}}$ gradient
  – Gradient is computed based on the entire training data
- Stochastic Gradient Descent
  – Works like gradient descent, now the ('stochastic') gradient is computed based on only one data sample

$$\mathbf{w} = \mathbf{w} - \eta \boxed{\frac{\partial \max\{0, -y_i(\mathbf{w}^T\mathbf{x}_i)\}}{\partial \mathbf{w}}} \rightarrow \begin{cases} 0 & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i) > 0 \\ -y_i\mathbf{x}_i & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i) \leq 0 \end{cases}$$

By setting $\eta = 1$, it returns the perceptron updating rule:

If $y_i(\mathrm{w}^T\mathrm{x}_i) \leq 0$
$\quad \mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$

Learning as Optimization

- Optimization is one of the core components of machine learning algorithms.

- Most machine learning algorithms is to learn the parameters by minimizing the loss function based on training data.

- Linear Regression:

$$\min_{\mathbf{w}} l(\mathbf{w}) = \sum_{i=1}^{n} (y_i - (\mathbf{w}^T \mathbf{x}_i + b))^2$$

- Support Vector Machine:

$$\min \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^{n} \xi_i$$

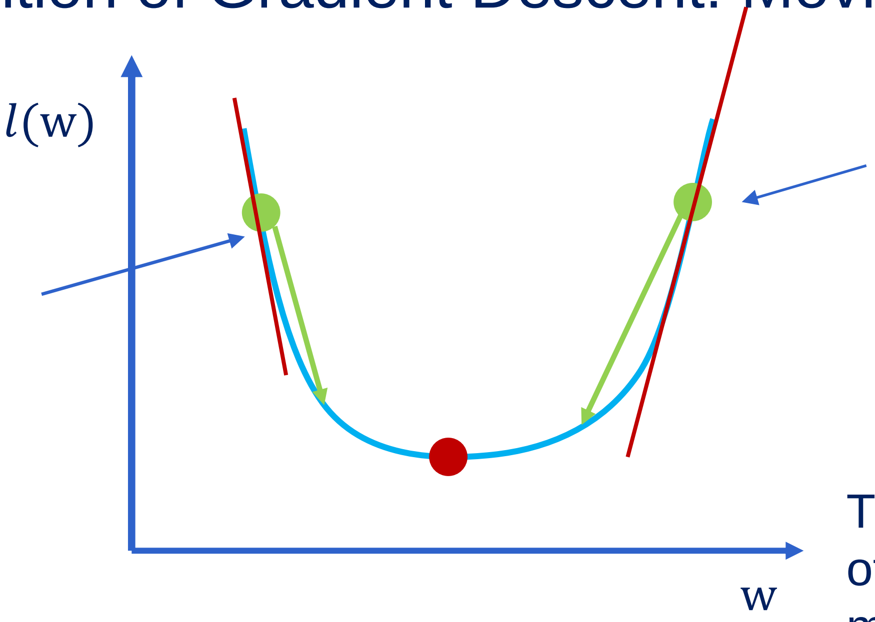
$$\text{subject to} \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \dots, n$$
$$\xi_i \geq 0$$

Recap: Optimization using Gradient Descent

- Gradient Descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the *negative* of the gradient. It is a general algorithm that can be applied to many machine learning model.

- Intuition of Gradient Descent: Moving in the direction of steepest descent

negative
slope



What is the gradient of *l*(w) at this point?

- The gradient of the function at this point is the slope of the tangent line (i.e., the straight line that "just touches" the curve at this point)
- This is a positive slope (increasing)
- Therefore, we need to go to opposite direction.

The gradient determines the direction of steepest increase of *l*(w). We need to go to opposite direction of gradient to minimize the *l*(w).

Gradient Descent Algorithm

Gradient Descent

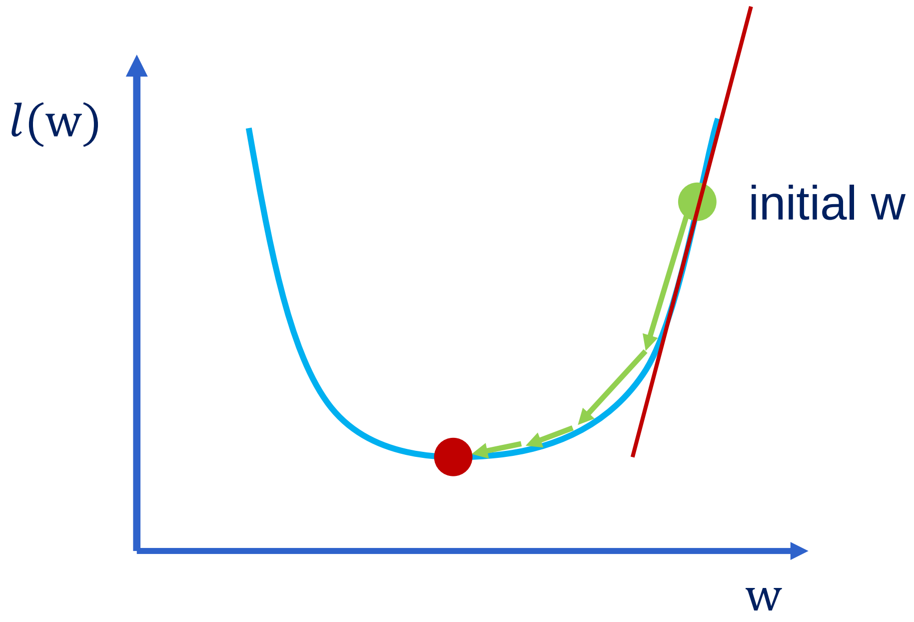
Initialize $\mathbf{w}$

Repeat T times

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \boxed{\frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}}$$

Output: $\mathbf{w}$

Learning rate (e.g., 0.1)   gradient



Stochastic Gradient Descent

- Works like gradient descent, now the ('stochastic') gradient is computed based on only one data sample

Stochastic Gradient Descent

Initialize $\mathbf{w}$

Repeat T times

For a data sample $(\mathbf{x}_i, y_i)$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \boxed{\frac{\partial l(\mathbf{w}, \mathbf{x}_i, y_i)}{\partial \mathbf{w}}}$$

Output: $\mathbf{w}$

stepsize (e.g., 0.1)

stochastic gradient based on one data sample

Perceptron: additional notes

Frank Rosenblatt

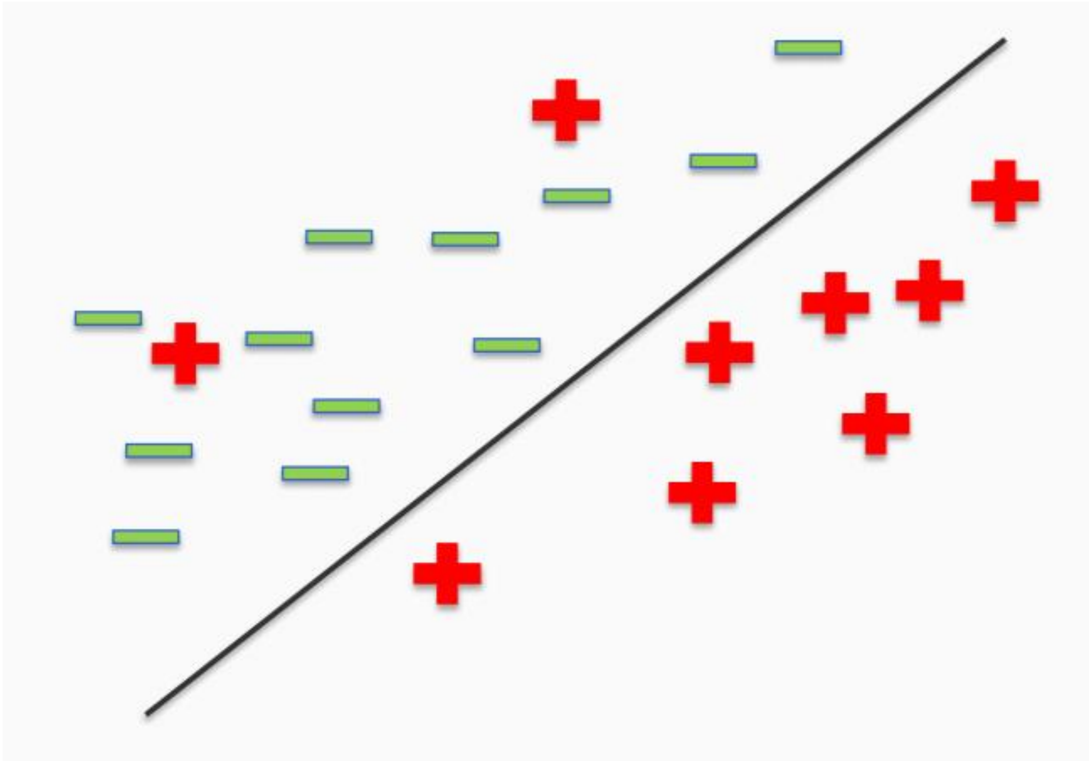
1928–1969

Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minksy, whose objections were published in the book "Perceptrons", co-authored with Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

Perceptron was criticized for its inability to handle non linearly separable problem (e.g. XOR function).

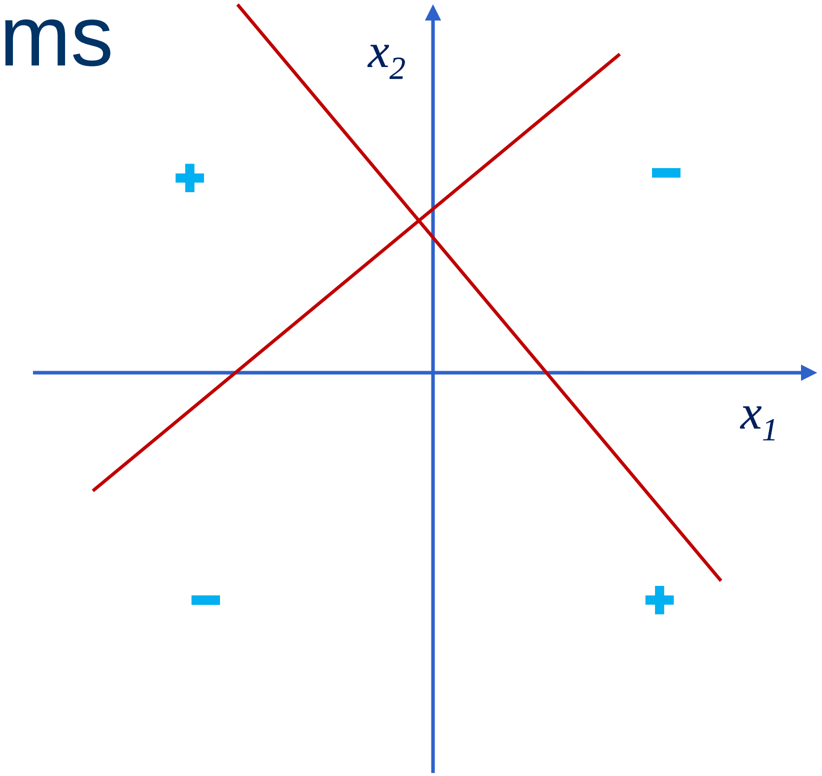
If the data is not linearly separable

- The perceptron algorithm would not converge if the training data is not linear separable.



Perceptron: Nonlinear separable problems

x_1	x_2	y
1	-1	1
1	1	-1
-1	1	1
-1	-1	-1

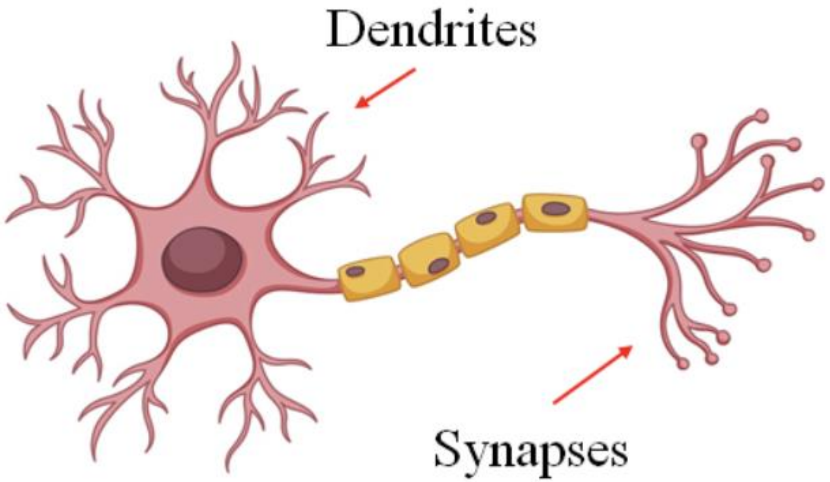


- Perceptron can not solve nonlinear separable problems
- Nonlinear separable problems can be solved by
  - Using multiple layers perceptron (**Artificial Neural Networks**)
  - Making it linearly separable using kernel (**Support Vector Machine**)

Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**
- **Supervised learning**
- Regression
  1. Linear regression with one variable
  2. Linear Regression with multiple variables
- Classification
  1. Perceptron
  2. Artificial Neural Network
  3. K Nearest Neighbor
  4. Support Vector Machine
- **Unsupervised learning**
  1. K-means Clustering
  2. Hierarchical Clustering

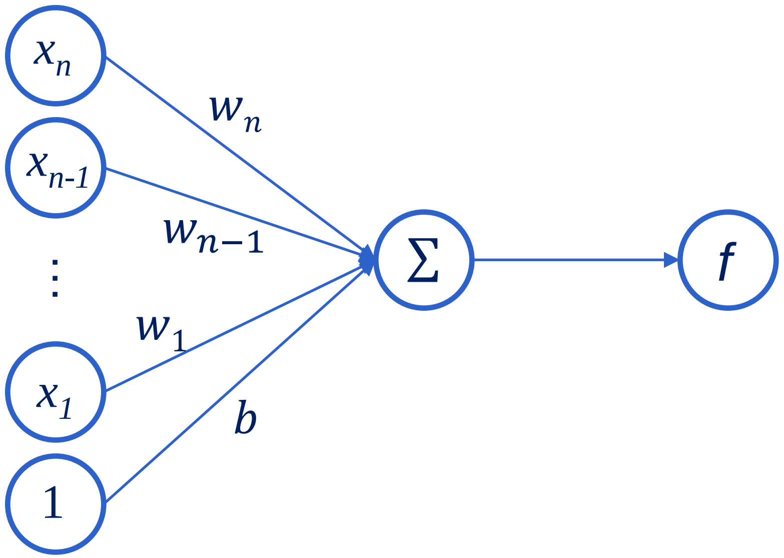
Artificial Neural Networks



NEURON

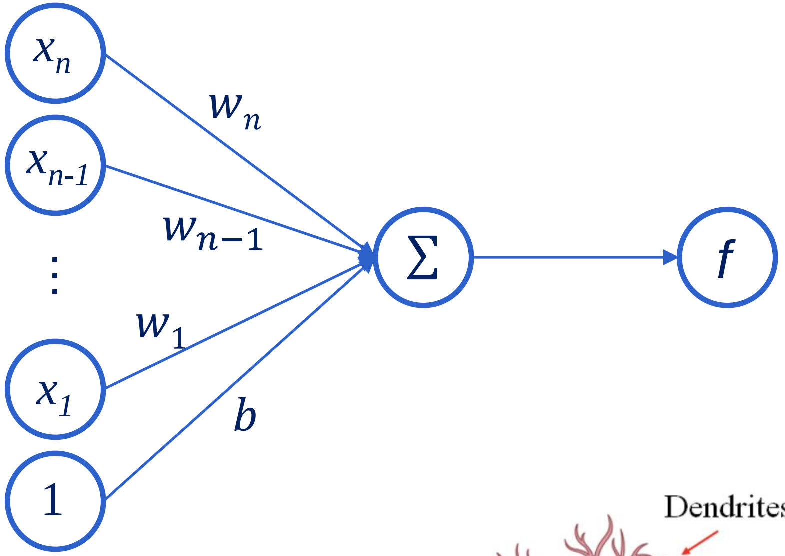
Recall on Perceptron

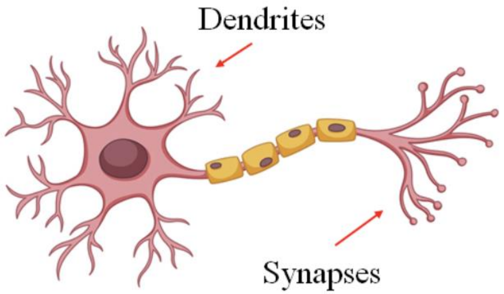
- A linear classification function $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b$
  - $y = 1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b > 0$
  - $y = -1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b < 0$



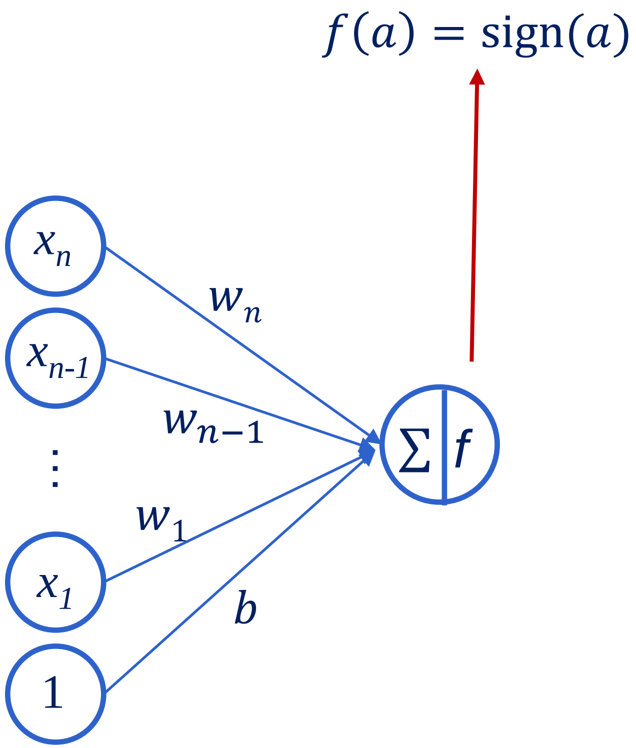
- $[x_1, \ldots, x_n]$: inputs
- $[w_1, \ldots, w_n]$: weights
- b: bias term
- Σ: summation
- f: activation function (sign function used)

Simplified Illustration of a Neuron



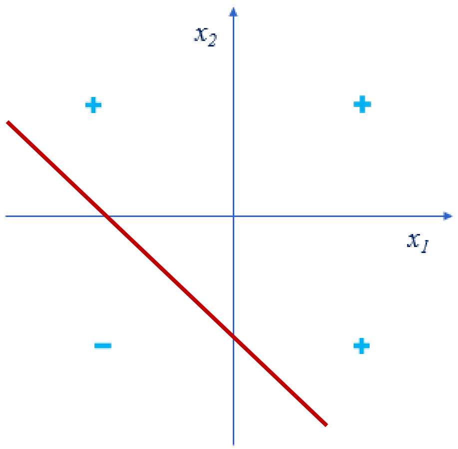


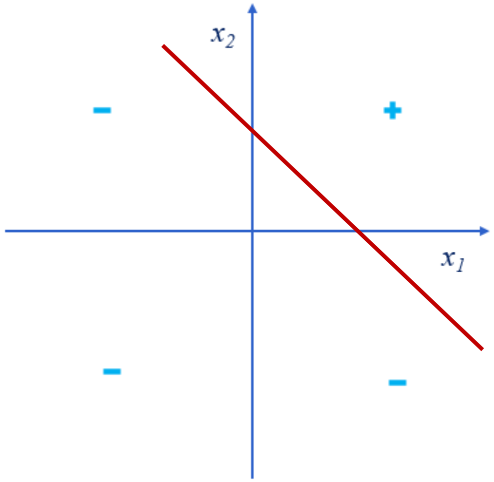
NEURON

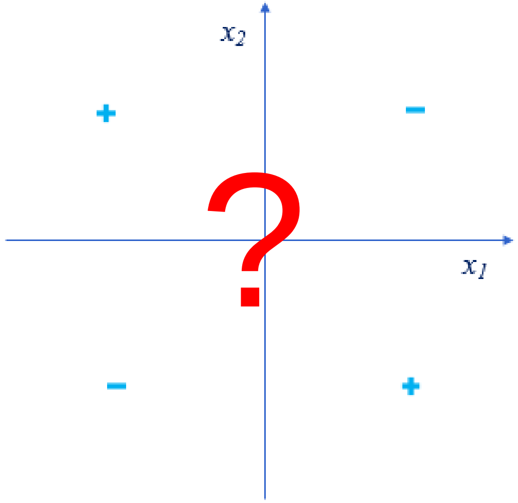


Limitation of Perceptron

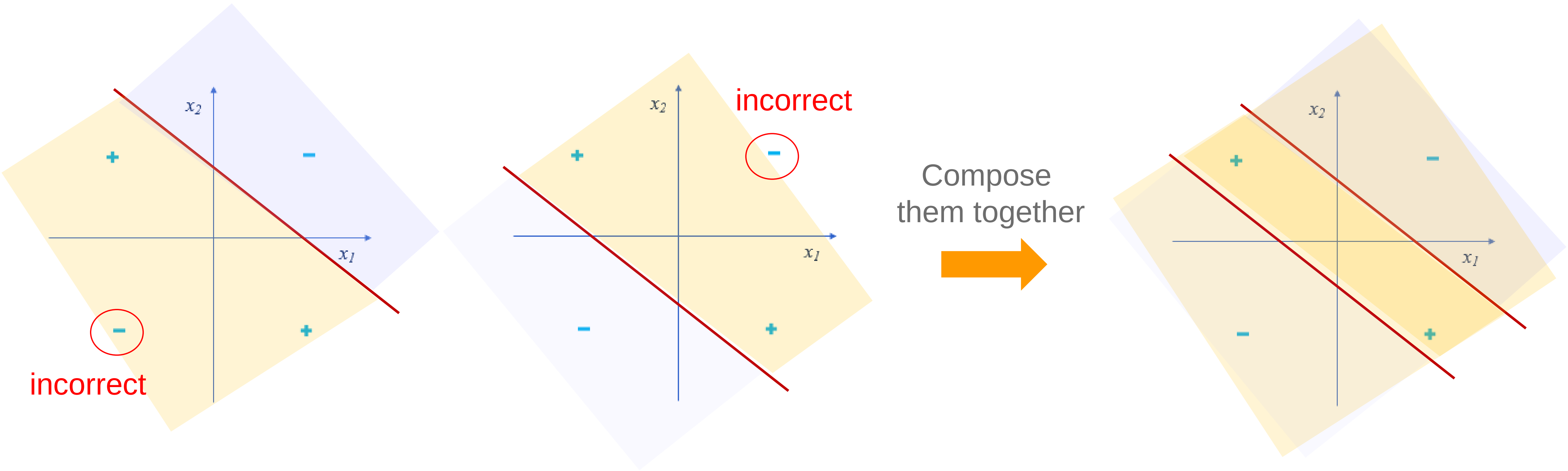
- Many classification problems are NOT linearly separable.



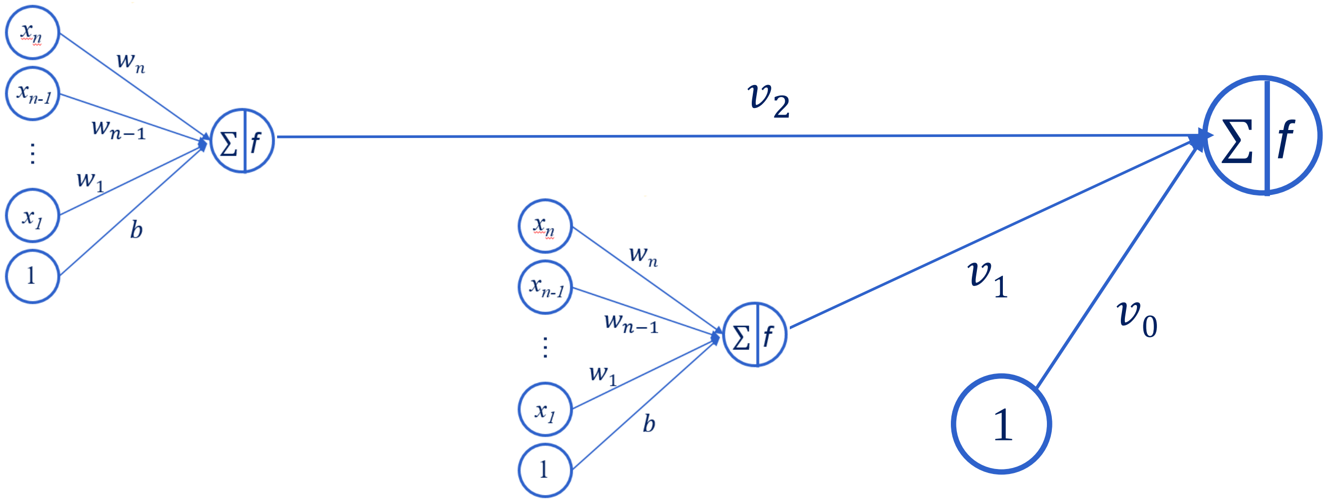




- Possible solution for nonlinear classification problem: Composition
  – Multiple Layer Perceptron (also called Feedforward Neural Network)

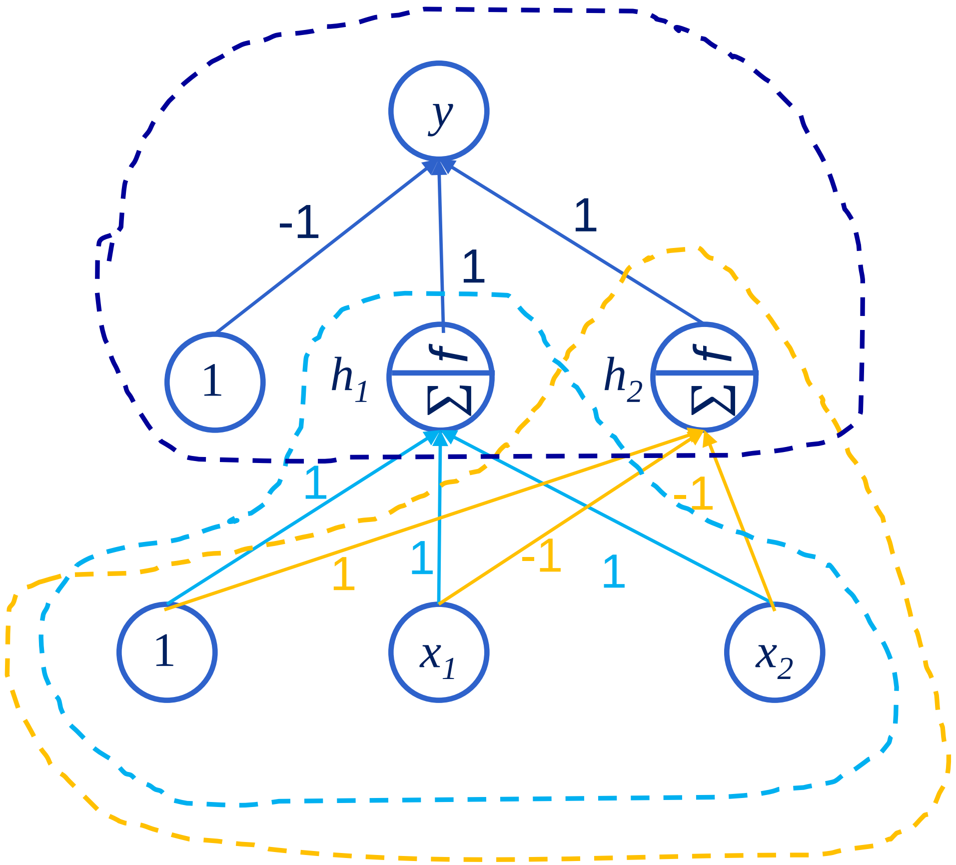


x_2
x_1
incorrect
x_2
x_1
incorrect
Compose
them together
x_2
x_1

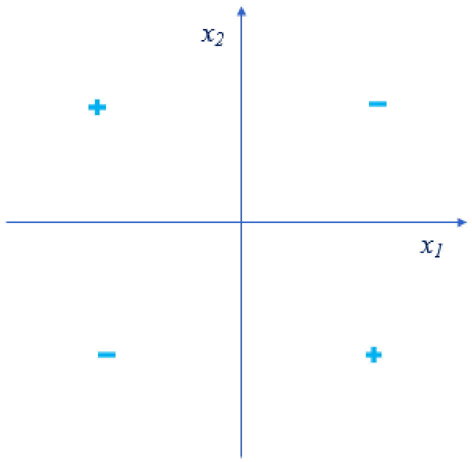


x_n
x_{n-1}
x_1
1
w_n
w_{n-1}
w_1
b
Σ f
v_2
x_n
x_{n-1}
x_1
1
w_n
w_{n-1}
w_1
b
Σ f
v_1
v_0
1
Σ f

Multi-Layer Perceptron for Nonlinear Classification



x_1	x_2	y
1	-1	1
1	1	-1
-1	1	1
-1	-1	-1

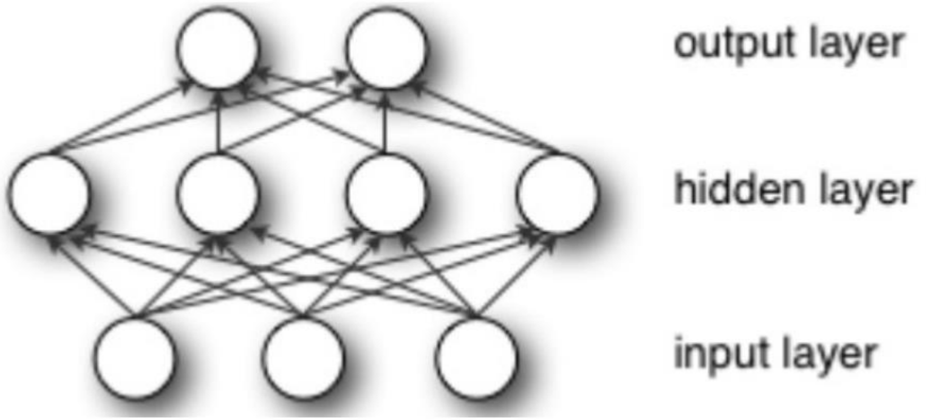


This multi-layer perceptron can solve this nonlinear classification problem.

How to learn the model parameter (i.e., weights on the edge) from data?

Multi-Layer Perceptron

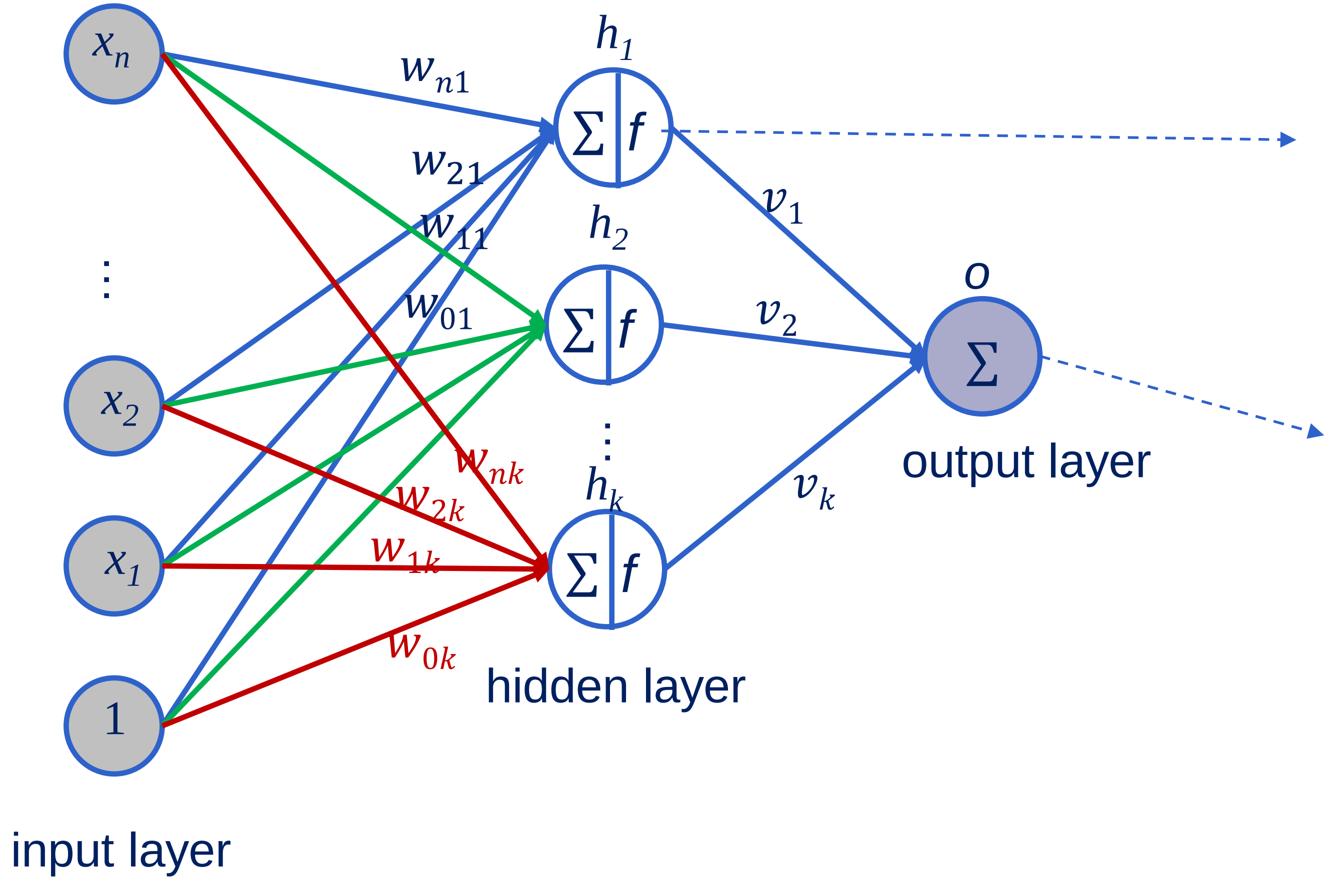
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers carry out nonlinear transformation of the inputs
- Universal Approximator (Hornik, 1991)

Multi-Layer Perceptron (MLP) with One Hidden Layer

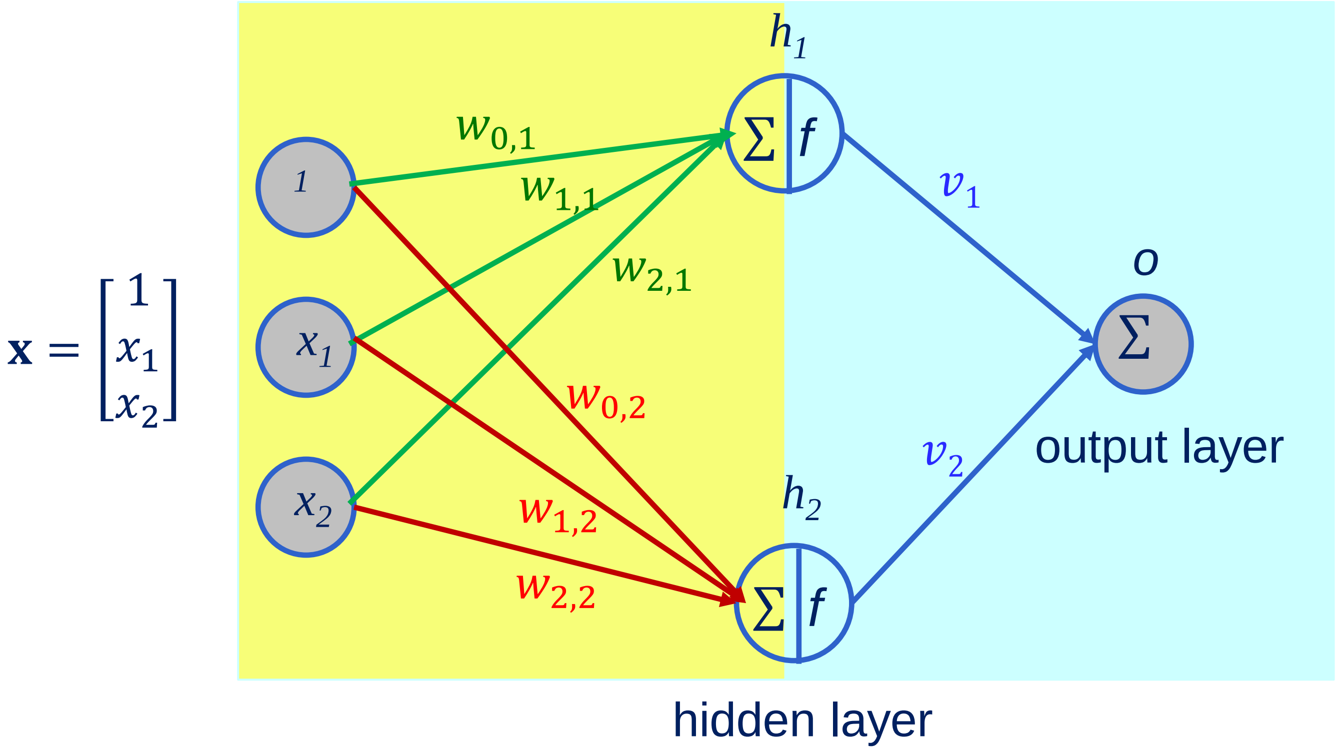
- Multi-Layer Perceptron with *d* inputs (i.e. the dimension of input samples is *d*), one hidden layer with *k* nodes, and one output.



Nonlinear "activation function" is required. Otherwise, the model would reduce to a linear model. (Why?)

The output *o* is a weighted sum of the preceding layer's hidden nodes

Output of MLP with 2 Hidden Nodes and 2-dimensional Input



$$o = v_1 h_1 + v_2 h_2 = \sum_{i=1}^{2} v_i h_i$$

$$h_1 = f\left(\sum_{i=0}^{2} w_{i1} x_i\right) = f(\mathbf{w}_1^T \mathbf{x})$$

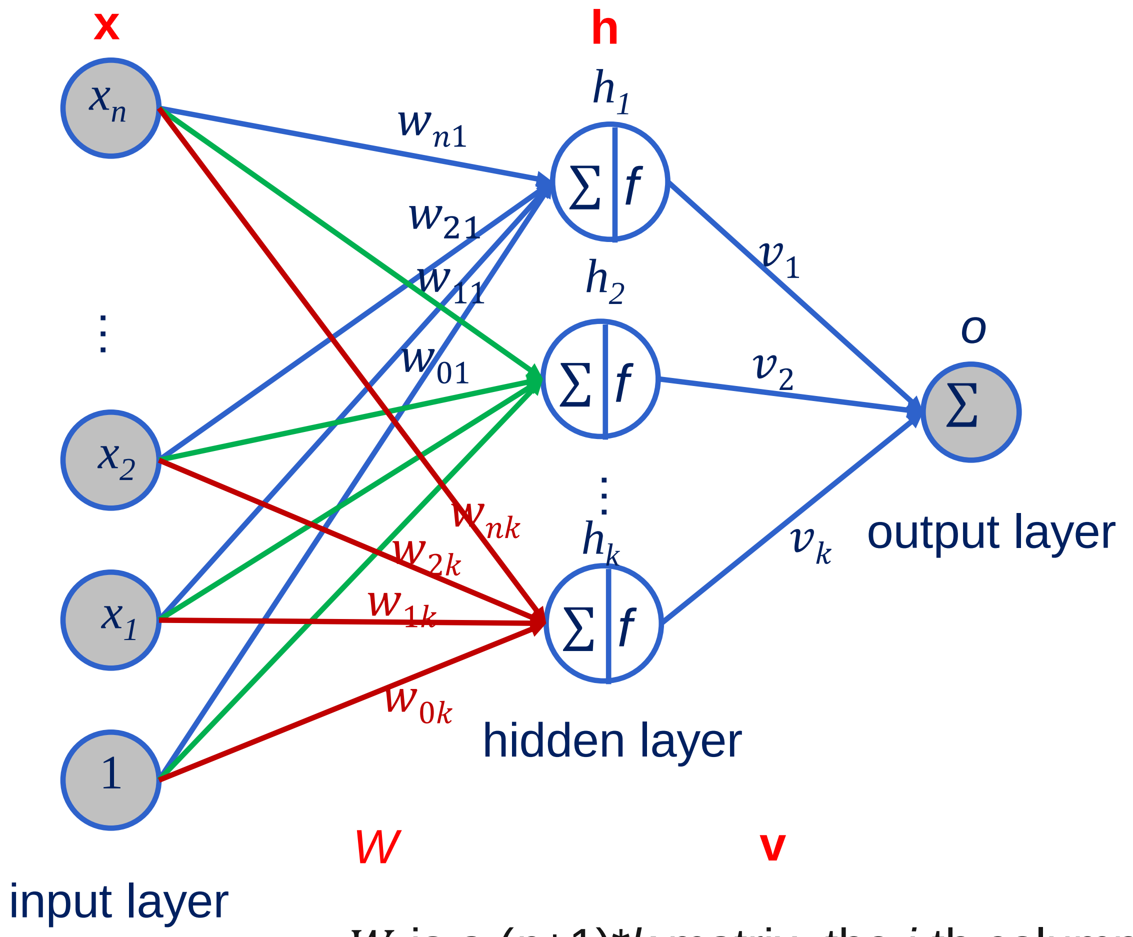
$$h_2 = f\left(\sum_{i=0}^{2} w_{i2} x_i\right) = f(\mathbf{w}_2^T \mathbf{x})$$

$$W = \begin{bmatrix} w_{0,1} & w_{0,2} \\ w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} = [\mathbf{w}_1 \quad w_2]$$

$$W^T = \begin{bmatrix} w_{0,1} & w_{1,1} & w_{2,1} \\ w_{0,2} & w_{1,2} & w_{2,2} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1{}^T \\ \mathbf{w}_2{}^T \end{bmatrix}$$

W is a 3-by-2 matrix with the j-th column in W denoting the weights of the j-th node in the hidden layer.

Output of MLP with *k* Hidden Nodes and *d*-dimensional Input



$$o = \sum_{i=1}^{k} v_i h_i = \mathbf{v}^T\mathbf{h}$$

$$\mathbf{h} = f(W^T\mathbf{x})$$

$$h_1 = f\Big(\sum_{i=0}^{n} w_{i1}x_i\Big) = f(\mathbf{w}_1^T\mathbf{x})$$

$$h_2 = f\Big(\sum_{i=0}^{n} w_{i2}x_i\Big) = f(\mathbf{w}_2^T\mathbf{x})$$

$$\vdots \qquad\qquad \vdots$$

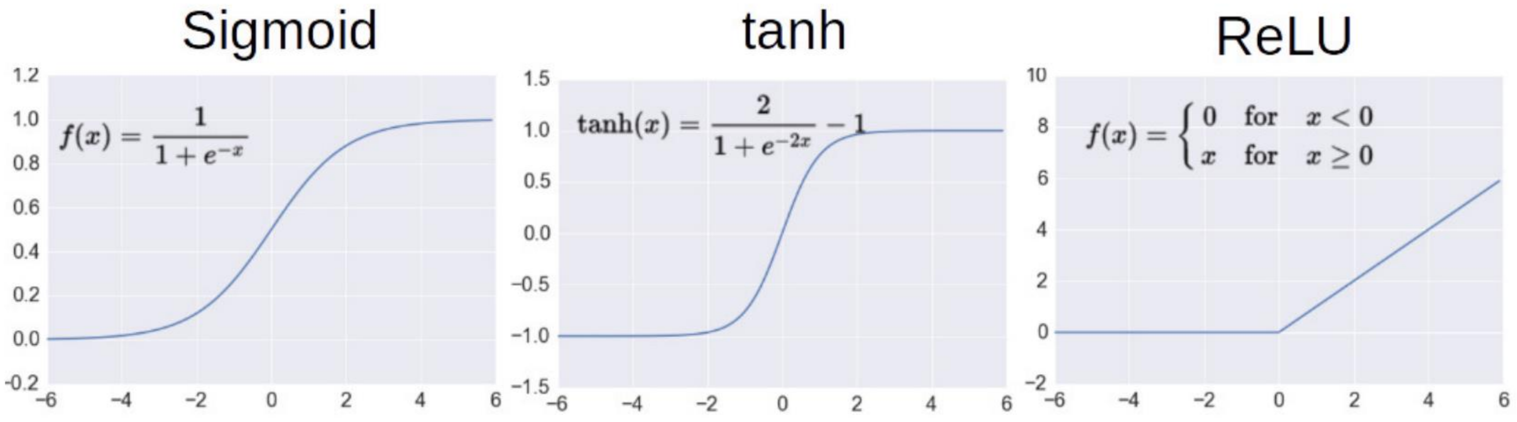
$$h_k = f\Big(\sum_{i=0}^{n} w_{ik}x_i\Big) = f(\mathbf{w}_k^T\mathbf{x})$$

$$o = \mathbf{v}^T f(W^T\mathbf{x})$$

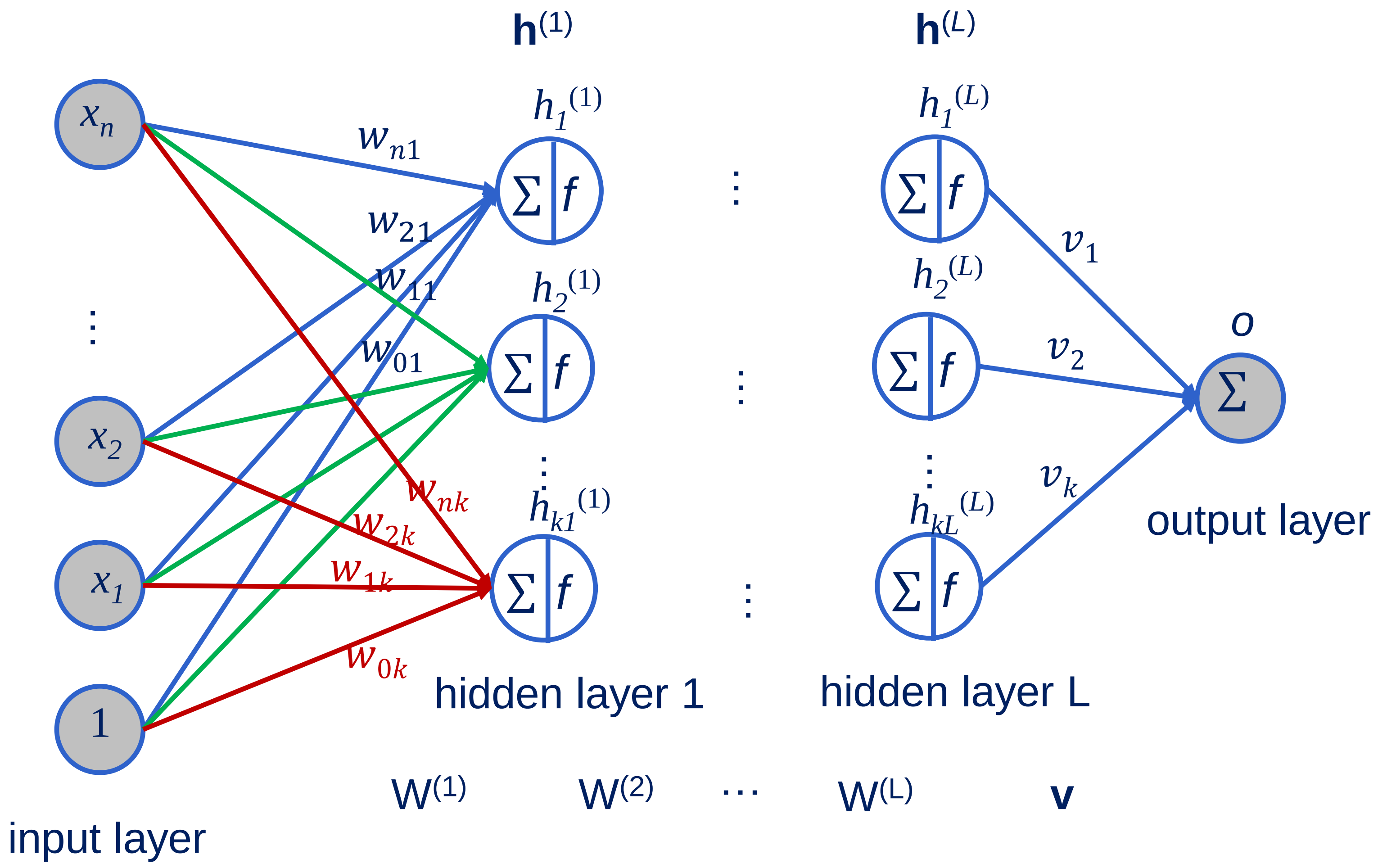
W is a (n+1)*k matrix, the j-th column in W denotes the weights of the j-th node in the hidden layer.

Commonly Used Nonlinear Activation Functions

- Some common choices for nonlinear activation function *f*
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - Tanh: $f(x) = 2\sigma(2x) - 1 = \frac{2}{1+\exp(-2x)} - 1$(range between -1 and +1)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



Multi-Layer Perceptron



- For an MLP with L hidden layers, $\mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(L)}$ and the scalar-valued output is computed as

$$o = \sum_{i=1}^{k} h_i v_i = \mathbf{v}^T \mathbf{h}^{(L)}$$

$$\mathbf{h}^{(L)} = f({W^{(L)}}^T \mathbf{h}^{(L-1)})$$
$$\mathbf{h}^{(L-1)} = f({W^{(L-1)}}^T \mathbf{h}^{(L-2)})$$
$$\vdots \qquad \vdots$$
$$\mathbf{h}^{(2)} = f({W^{(2)}}^T \mathbf{h}^{(1)})$$
$$\mathbf{h}^{(1)} = f({W^{(1)}}^T \mathbf{x})$$

Why nonlinear "activation function" is required?

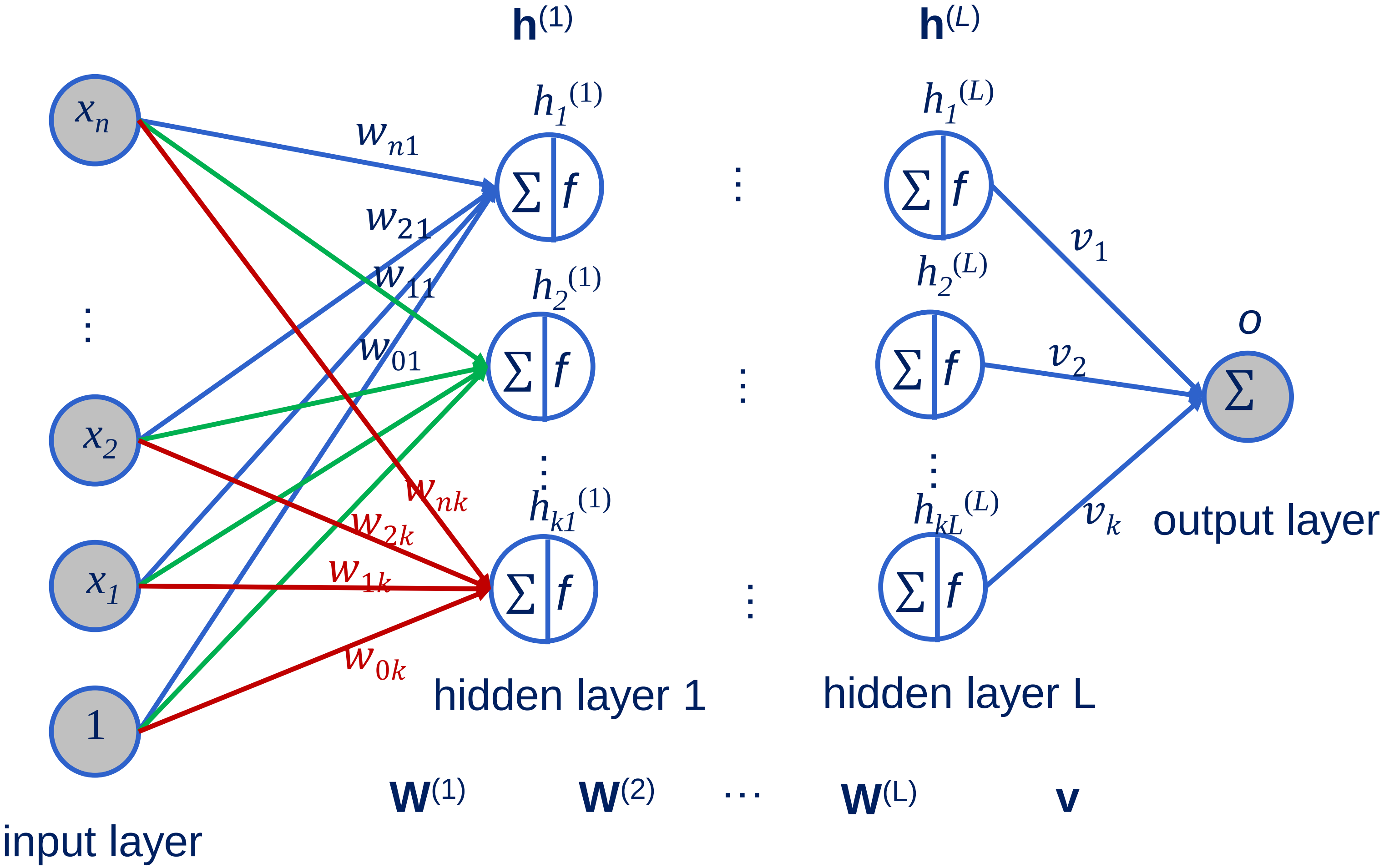
Why nonlinear “activation function” is required

- If we remove the nonlinear “activation function”, the output of the neural network will be reduced to

$$
\begin{aligned}
o &= \sum_{i=1}^{k} h_i v_i = \mathbf{v}^T \mathbf{h}^{(L)} \\
&= \mathbf{v}^T {W^{(L)}}^T \mathbf{h}^{(L-1)} \\
&= \mathbf{v}^T {W^{(L)}}^T {W^{(L-1)}}^T \mathbf{h}^{(L-2)} \\
&\dots \\
&= \underbrace{\mathbf{v}^T {W^{(L)}}^T {W^{(L-1)}}^T \dots {W^{(2)}}^T {W^{(1)}}^T}\mathbf{x}
\end{aligned}
$$

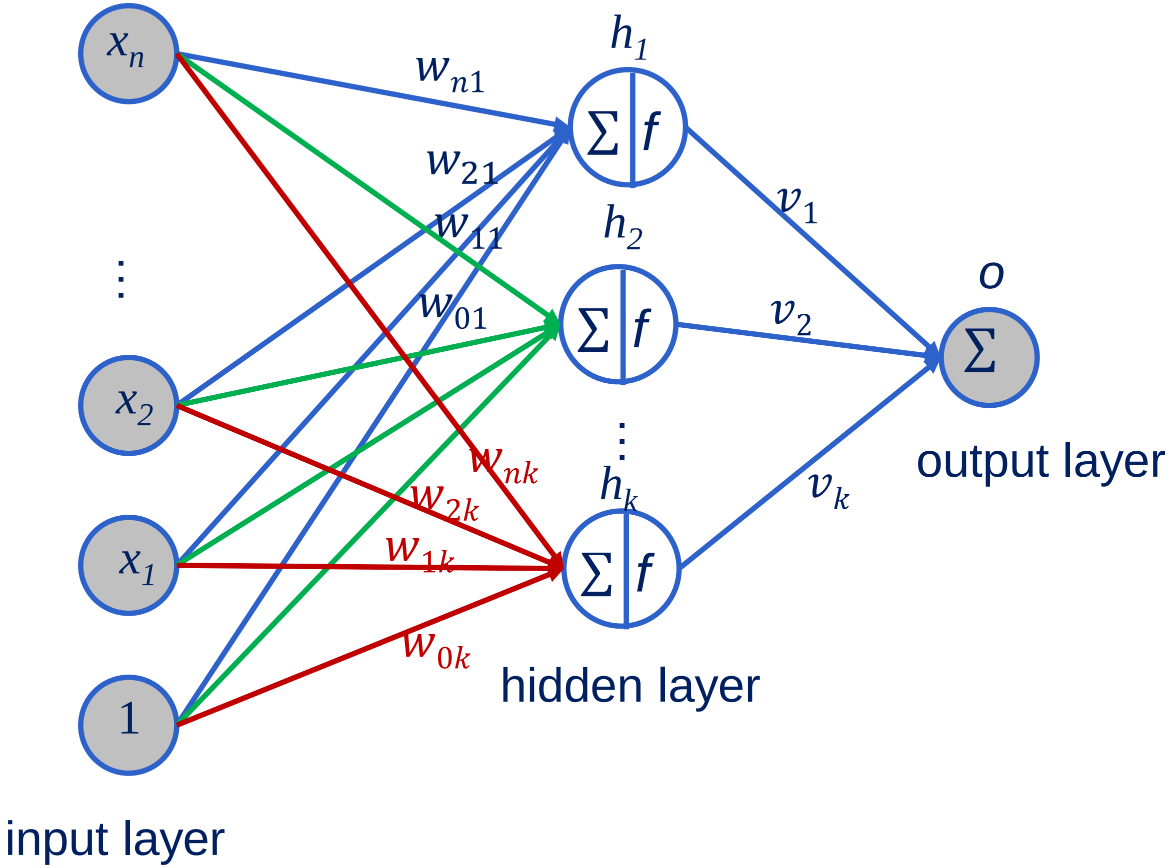
All these operations are linear transformation. Therefore, without nonlinear activation function, it will reduce to a simple linear model.

Learning MLP Via Backpropagation



- For a given training dataset, we want to learn the parameter ($W^{(1)}$, …, $W^{(L)}$, **v**) by minimizing some loss function.
- **Backpropagation** (gradient descent + chain rule for derivatives) is commonly used to do this efficiently.

Learning MLP (one hidden layer)



- Given one data sample of input and true output $\{\boldsymbol{x}, y\}$, our goal is to minimize $(y - o)^2 = (y - \mathbf{v}^T f(W^T \mathbf{x}))^2$

- Given n data sample of input and true output $\{\{\boldsymbol{x}_1, y_1\}, \{\boldsymbol{x}_2, y_2\}, \dots, \{\boldsymbol{x}_m, y_m\}\}$, the goal becomes:

$$\min_{W,\mathbf{v}} \frac{1}{2} \sum_{i=1}^{m} (y_i - o_i)^2$$

$$\min_{W,\mathbf{v}} \frac{1}{2} \sum_{i=1}^{m} \left(y_i - \mathbf{v}^T f(W^T \mathbf{x}_i)\right)^2$$

$$= \min_{W,\mathbf{v}} \frac{1}{2} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{k} v_j f(\mathbf{w}_j^T \mathbf{x}_i) \right)^2$$

Note: We use square loss here for simply illustrating the key idea of learning MLP. In practice, we usually use hinge loss or cross entropy loss (log loss) for classification problem.

Learning MLP (one hidden layer) (optional)

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function (loss function)

$$L = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \sum_{j=1}^{k} v_j f(\mathbf{w}_j^T \mathbf{x}_i)\right)^2 = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{v}^T\mathbf{h}_i)^2$$

- Compute gradient w.r.t $\boldsymbol{v} = [v_1, v_2, \ldots, v_k]$ is straightforward

$$\frac{\partial L}{\partial \mathbf{v}} = -\sum_{i=1}^{m}\left(y_i - \sum_{j=1}^{k} v_j f\left(\mathbf{w}_j^T \mathbf{x}_i\right)\right)\mathbf{h}_{\boldsymbol{i}} = -\sum_{i=1}^{m} e_i \boldsymbol{h}_{\boldsymbol{i}}$$

- where e_i denotes the error between the true output and MLP output for the i^{th} data sample.

Learning MLP (one hidden layer) (optional)

- To learn minimize the loss w.r.t. $W = [\boldsymbol{w}_1 \boldsymbol{w}_2 \ldots \boldsymbol{w}_k]$

$$L = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \sum_{j=1}^{k} v_j f(\mathbf{w}_j^T \mathbf{x}_i)\right)^2 = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{v}^T \mathbf{h}_i)^2$$

- Gradient w.r.t the weights $W = [\boldsymbol{w}_1 \boldsymbol{w}_2 \ldots \boldsymbol{w}_k]$ is bit more complicated due to the presence of the nonlinear activation function and the chain rule is used.

$$\frac{\partial L}{\partial \mathbf{w}_j} = \frac{\partial L}{\partial \mathbf{f}_j}\frac{\partial \mathbf{f}_j}{\partial \mathbf{w}_j}$$

Note: $\mathbf{f}_j = f(\mathbf{w}_j^T \mathbf{x}_i)$ for the j-th hidden node

- We have $\frac{\partial L}{\partial \mathbf{f}_j} = -\sum_{i=1}^{m}\left(y_i - \sum_{j=1}^{k} v_j f\left(\mathbf{w}_j^T \mathbf{x}_i\right)\right) v_j = -\sum_{i=1}^{m} e_i\, v_j$
- We have $\frac{\partial \mathbf{f}_j}{\partial \mathbf{w}_j} = \sum_{i=1}^{m} f'(\mathbf{w}_j^T \mathbf{x}_i)\mathbf{x}_i$, where $f'\left(\mathbf{w}_j^T \mathbf{x}_i\right)$ is function *f*'s derivative at $\mathbf{w}_j^T \mathbf{x}_i$

Learning MLP (one hidden layer) (optional)

α is a small non-negative scalar (learning rate)

- Gradient with the respect to **v**:

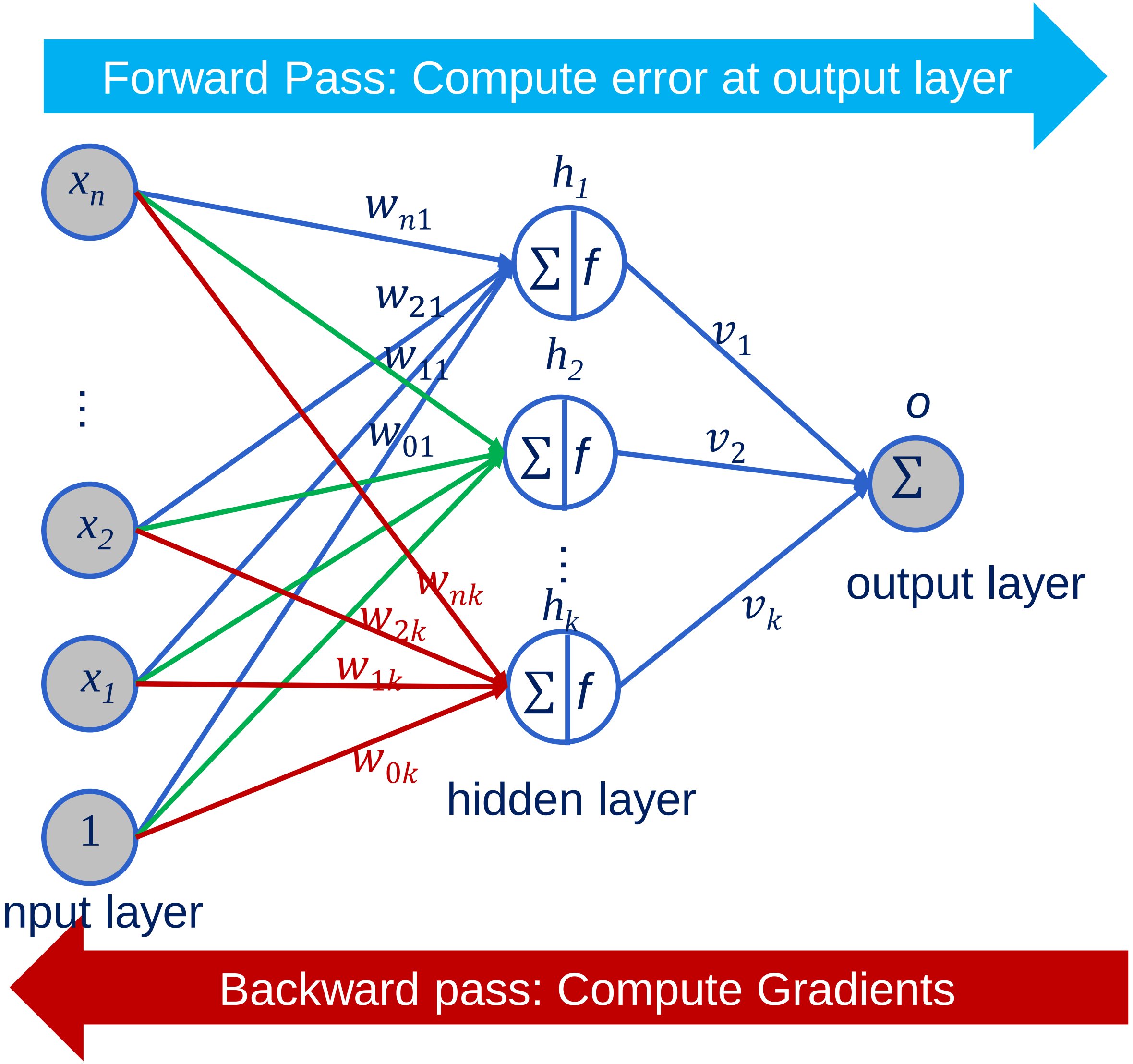
$$\frac{\partial L}{\partial \mathbf{v}} = -\sum_{i=1}^{m} e_i \mathbf{h}_i \longrightarrow \mathbf{v}_{\text{new}} \leftarrow \mathbf{v}_{old} - \alpha\left(-\sum_{i=1}^{m} e_i \mathbf{h}_i\right)$$

- Gradient w.r.t. the weights W

$$\frac{\partial L}{\partial \mathbf{w}_j} = \frac{\partial L}{\partial \mathbf{f}_j}\frac{\partial \mathbf{f}_j}{\partial \mathbf{w}_j} = -\sum_{i=1}^{m} e_i\, v_j \sum_{i=1}^{m} f'(\mathbf{w}_j^T \mathbf{x}_i)\mathbf{x}_i \longrightarrow (\mathbf{w}_j)_{\text{new}} \leftarrow (\mathbf{w}_j)_{\text{old}} - \alpha\left(-\sum_{i=1}^{m} e_i\, v_j \sum_{i=1}^{m} f'(\mathbf{w}_j^T \mathbf{x}_i)\mathbf{x}_i\right)$$

These calculations can be done efficiently using **backpropagation**.

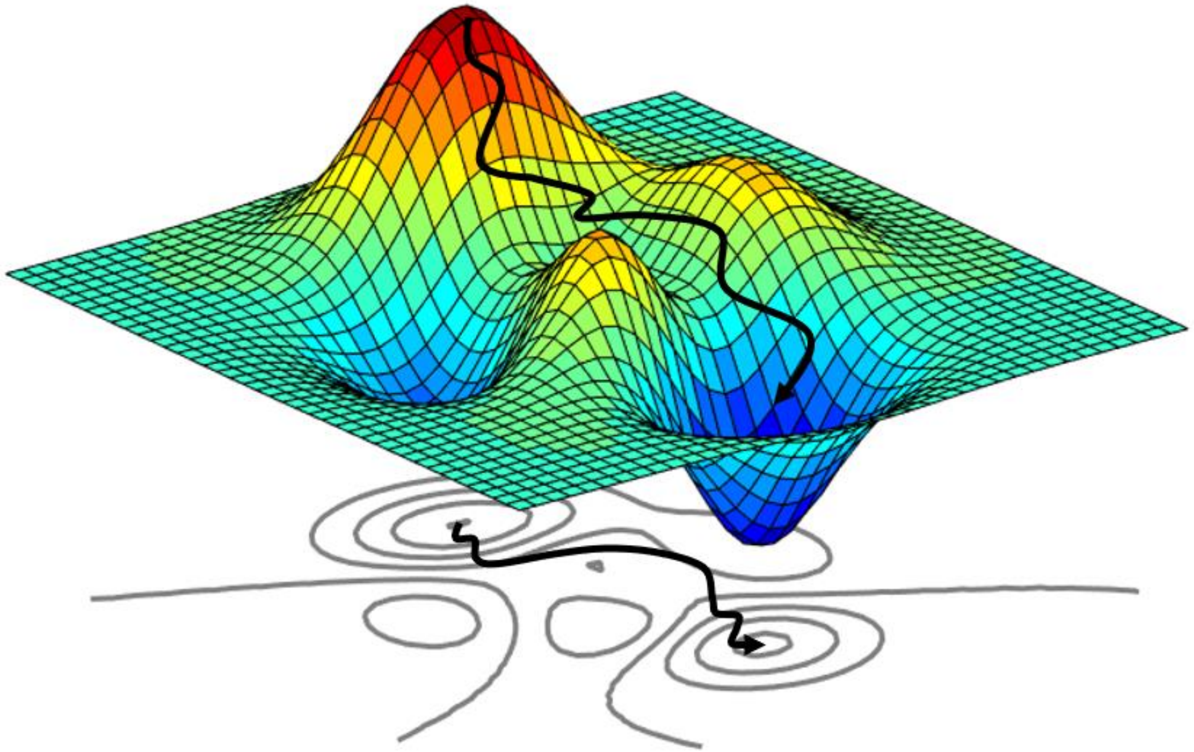
Backpropagation



- Basically consists of a forward pass and a backward pass
- Forward pass computes the error e_i using the current parameters
- Backwards pass computes the gradient and updates the parameters, starting from the parameter at the rightmost layer and the moving backwards.
- Also good at reusing previous computations (updates of parameters at any layer depend on parameters of the upper layer)

Solutions are Local Minima

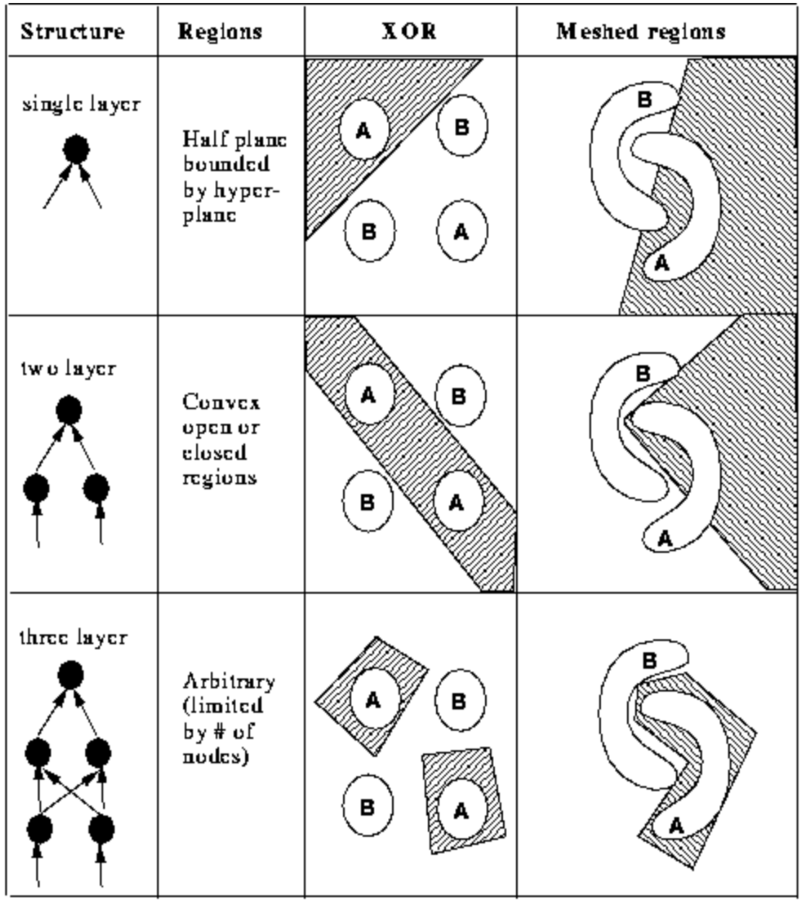
Parameter Initialization and Learning Rate Do Matter



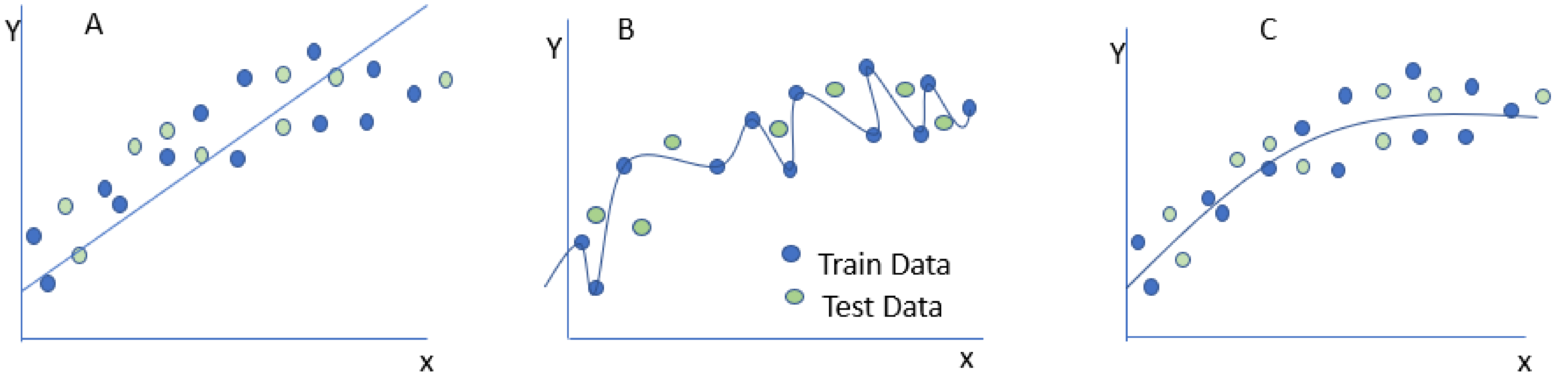
Pros and Cons of MLP

- Pros
  - Versatile: Adaptive to many datasets
  - Can capture nonlinear dependence of input and output

- Cons
  - Does not work for small data sets
  - Blackbox; Hard to interpret
  - Speed of convergence
  - Local minimum
  - Overfitting issue (how to select the structure; how to achieve good generalization)



Underfitting/Overfitting and Model Complexity



Under fitting　　Over fitting　　Well fitting