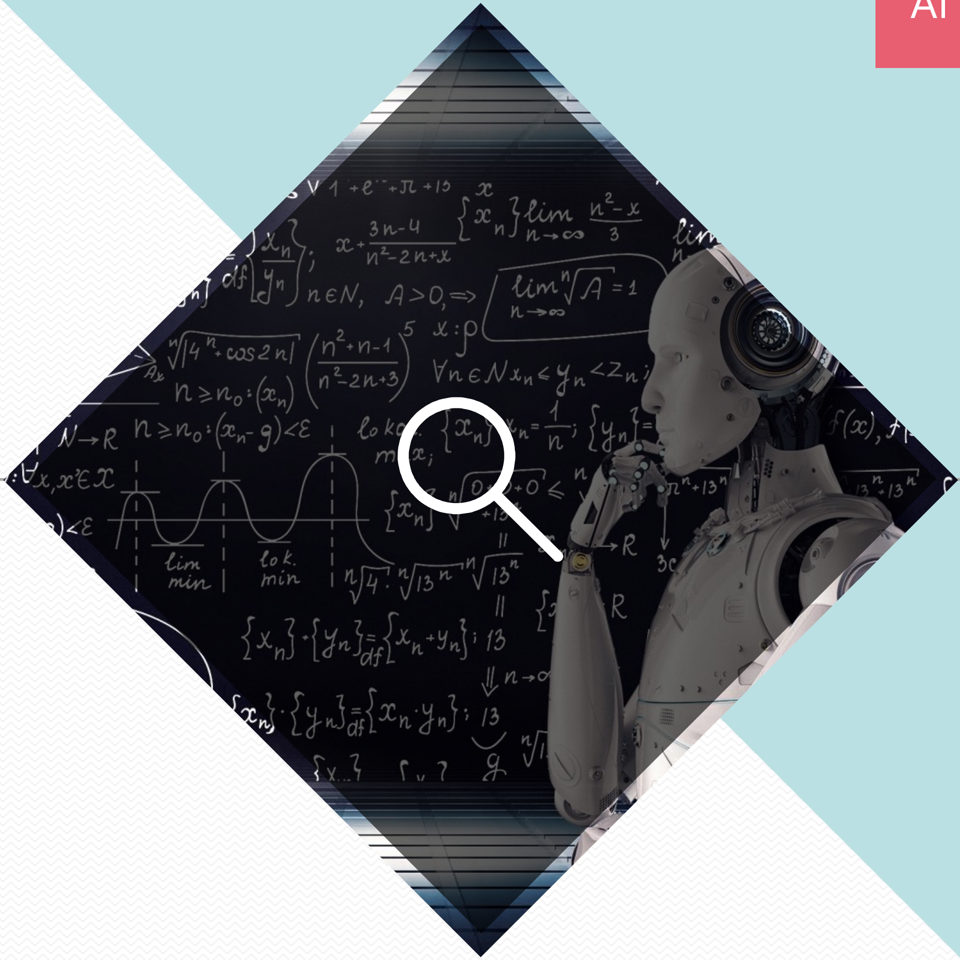
AI TRAINING SERIES



AI Math

기본적인 선형대수내용을 공부하고 이를 AI 문제에 적용

Agenda

- *Why we need Math. in AI?*
- *Math. You Need to Know for AI*

- Linear Algebra

- Chain rule of derivative

- Gradient decent

- Regression(Linear/Logistic)

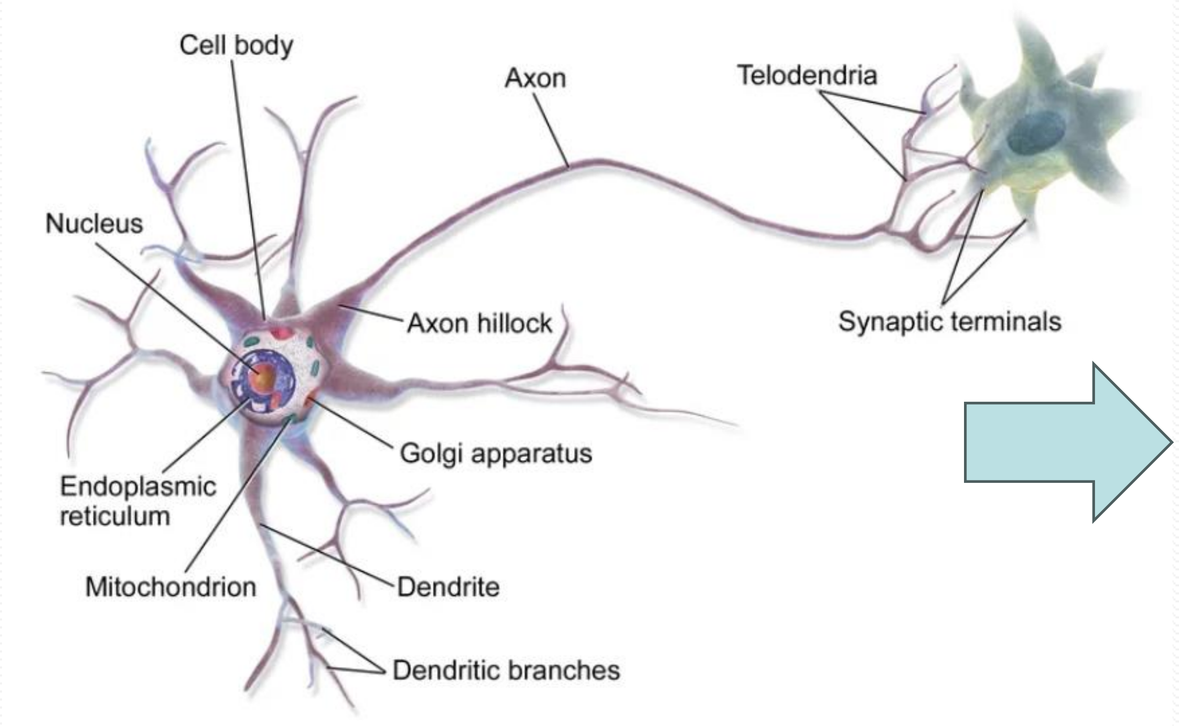
- *Hands on*

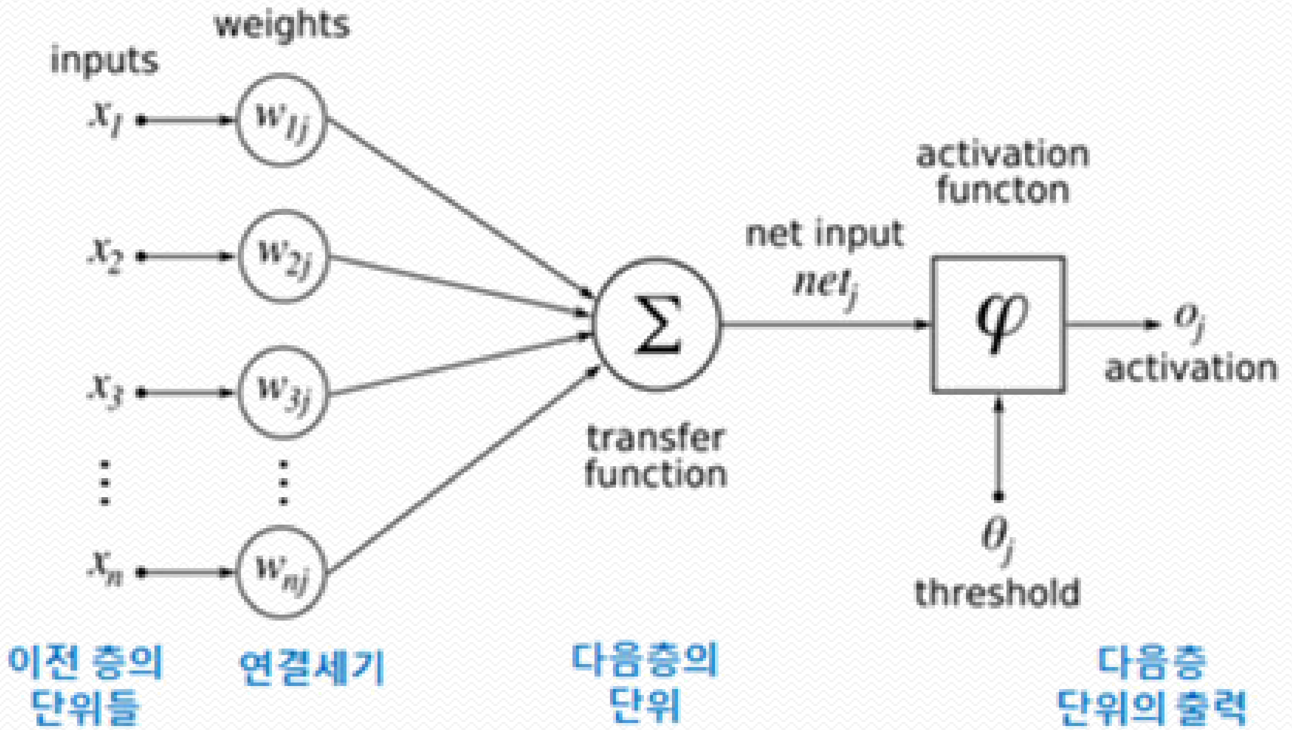
- Basic python lib.

- Basic Math for AI

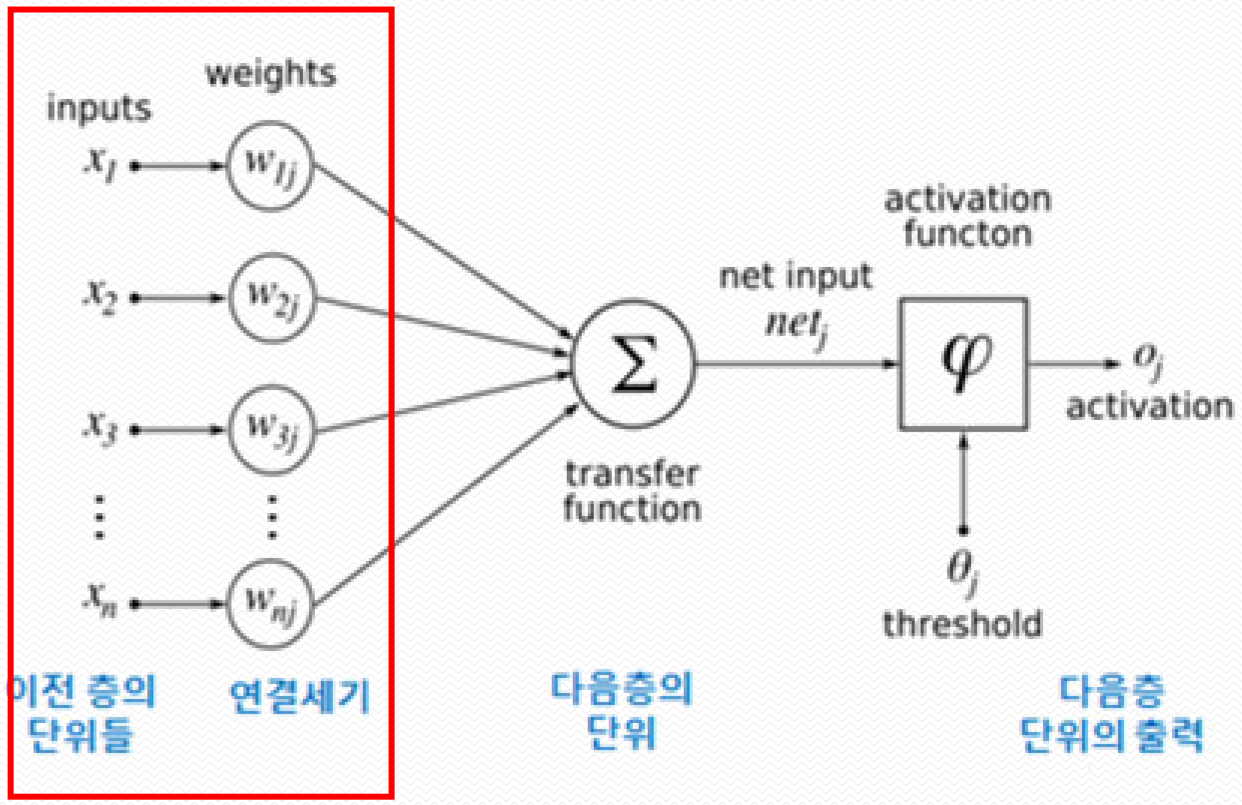
Why we need Math in AI?

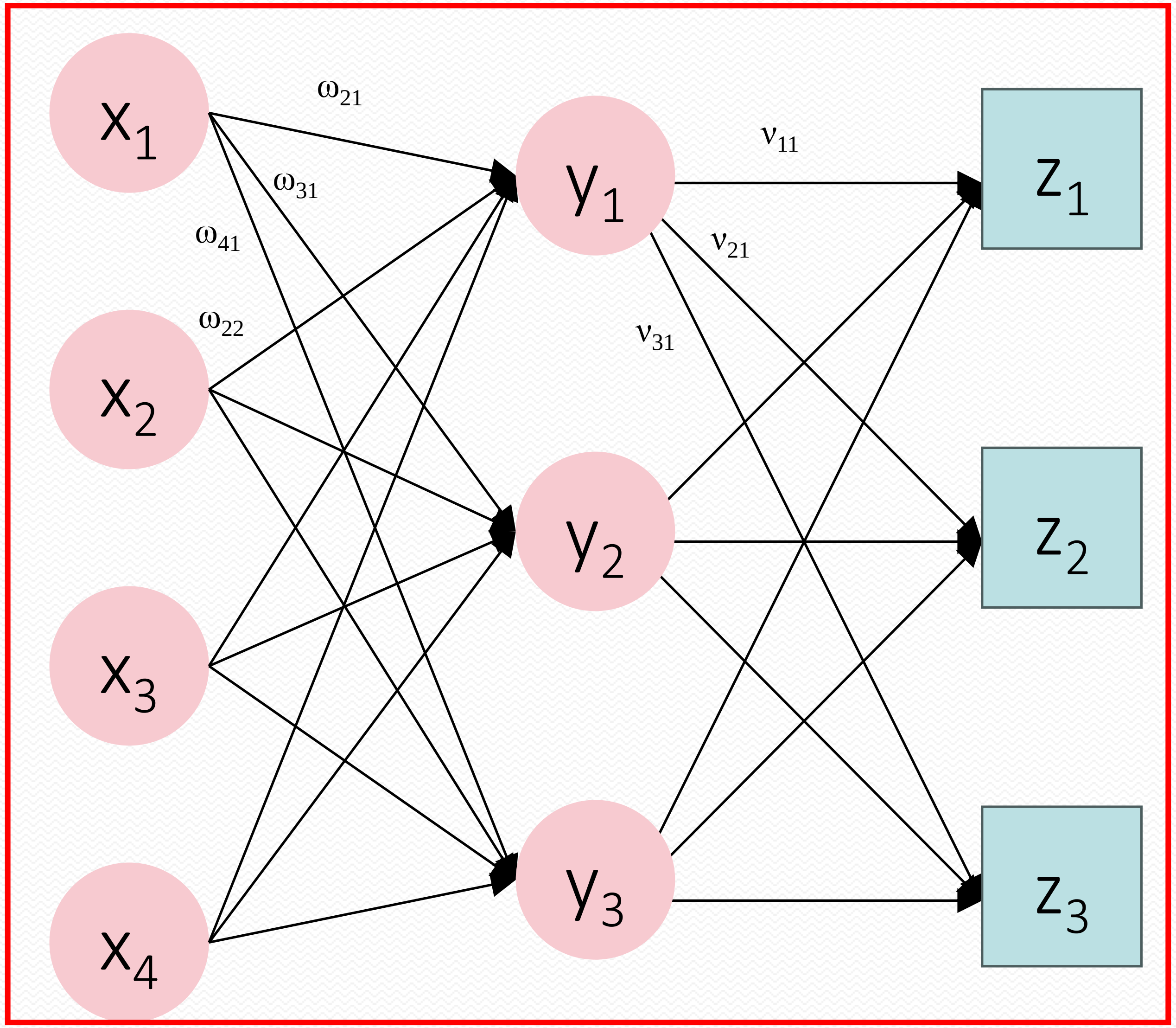
Neuron


Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches

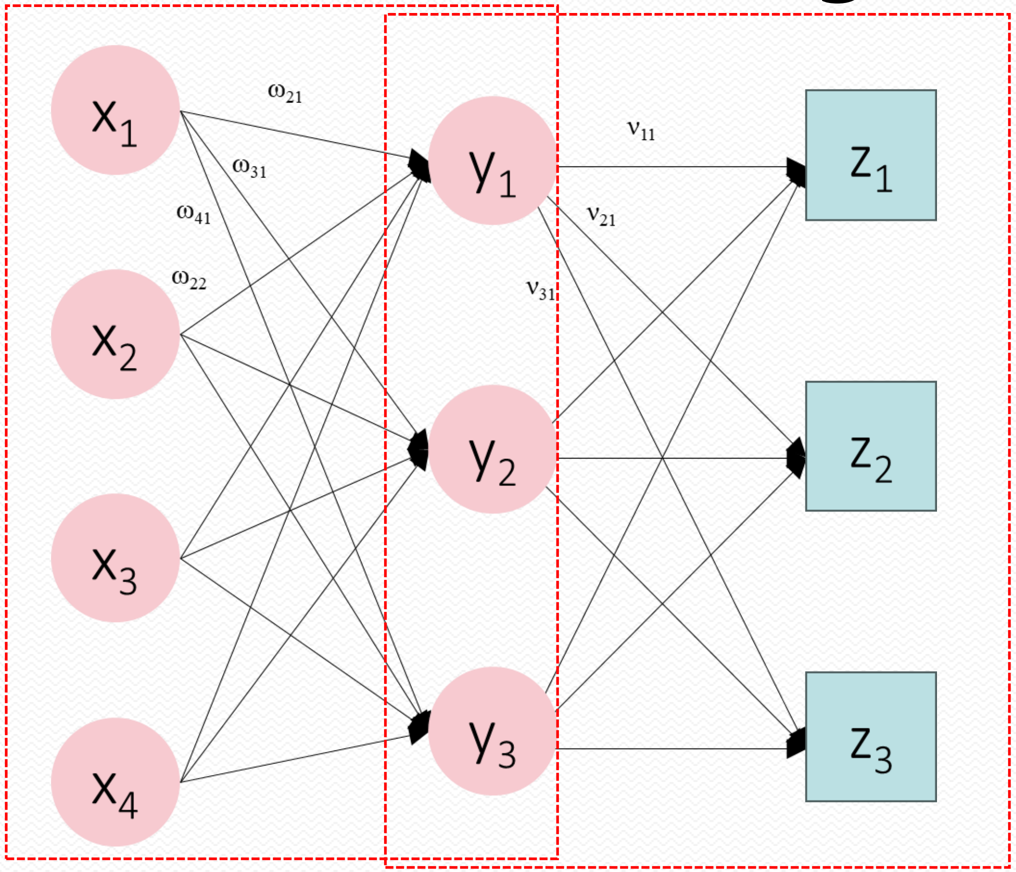

weights
inputs
x_1 w_{1j}
x_2 w_{2j}
x_3 w_{3j}
x_n w_{nj}
activation functon
net input
net_j
Σ
transfer function
φ
o_j
activation
θ_j
threshold
이전 층의 단위들
연결세기
다음층의 단위
다음층 단위의 출력

Neural network





Linear algebra



$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}\begin{pmatrix} \omega_{11} & \omega_{21} & \omega_{31} \\ \omega_{12} & \omega_{22} & \omega_{32} \\ \omega_{13} & \omega_{23} & \omega_{33} \\ \omega_{14} & \omega_{24} & \omega_{34} \end{pmatrix}\begin{pmatrix} \nu_{11} & \nu_{11} & \nu_{11} \\ \nu_{12} & \nu_{12} & \nu_{12} \\ \nu_{13} & \nu_{13} & \nu_{13} \end{pmatrix}$$

$$=\begin{pmatrix} z_1 & z_2 & z_3 \end{pmatrix}$$

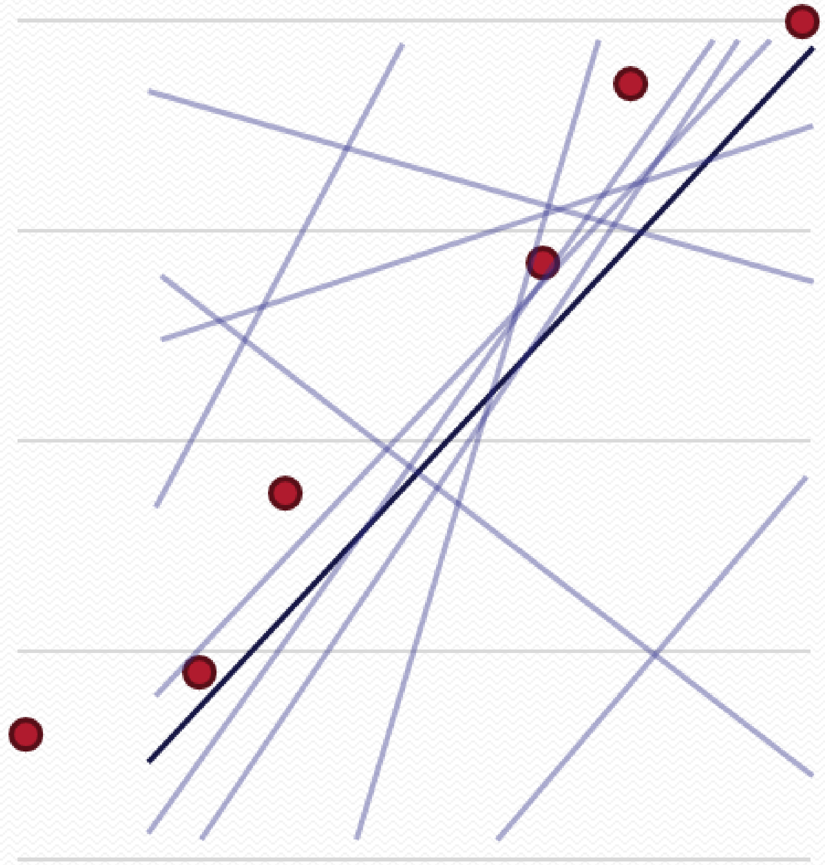
$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}\begin{pmatrix} \omega_{11} & \omega_{21} & \omega_{31} \\ \omega_{12} & \omega_{22} & \omega_{32} \\ \omega_{13} & \omega_{23} & \omega_{33} \\ \omega_{14} & \omega_{24} & \omega_{34} \end{pmatrix}=\begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$$

$$\begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}\begin{pmatrix} \nu_{11} & \nu_{11} & \nu_{11} \\ \nu_{12} & \nu_{12} & \nu_{12} \\ \nu_{13} & \nu_{13} & \nu_{13} \end{pmatrix}=\begin{pmatrix} z_1 & z_2 & z_3 \end{pmatrix}$$

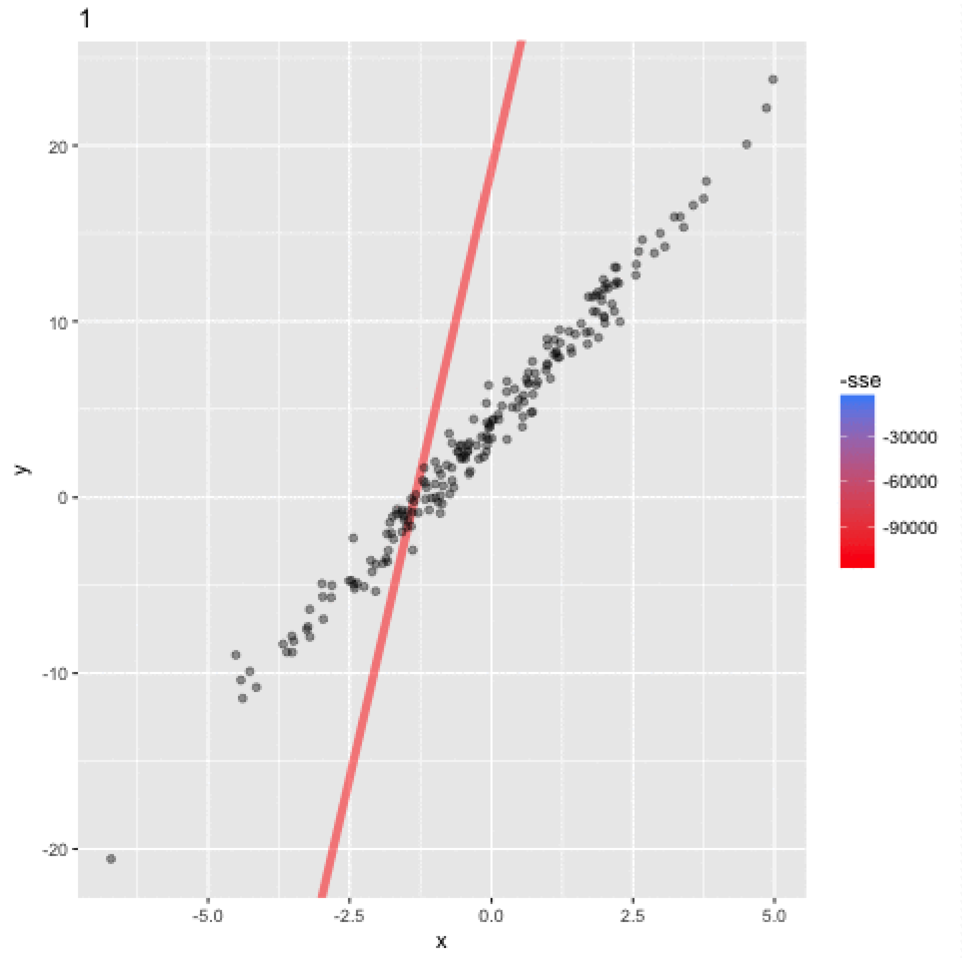
Linear Regression

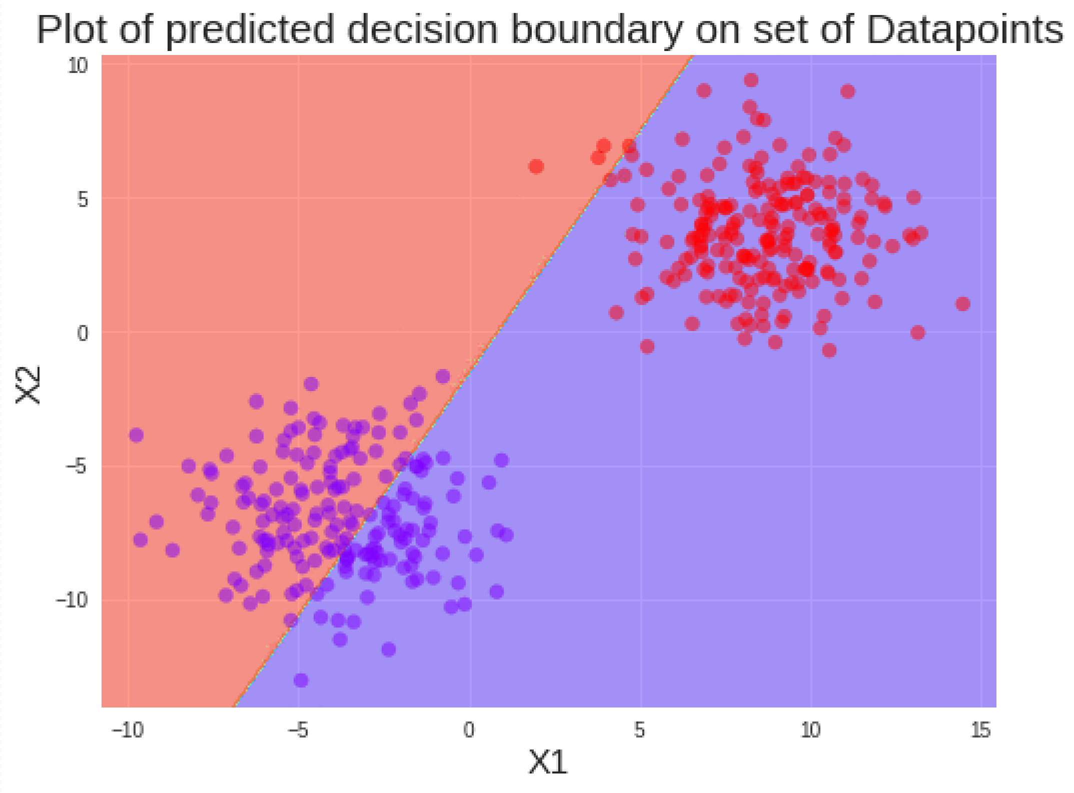
$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix} \begin{pmatrix} \omega_{11} & \omega_{21} & \omega_{31} \\ \omega_{12} & \omega_{22} & \omega_{32} \\ \omega_{13} & \omega_{23} & \omega_{33} \\ \omega_{14} & \omega_{24} & \omega_{34} \end{pmatrix} = \begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix}$$

$$A\mathbf{x} = \mathbf{Y}$$



Regression

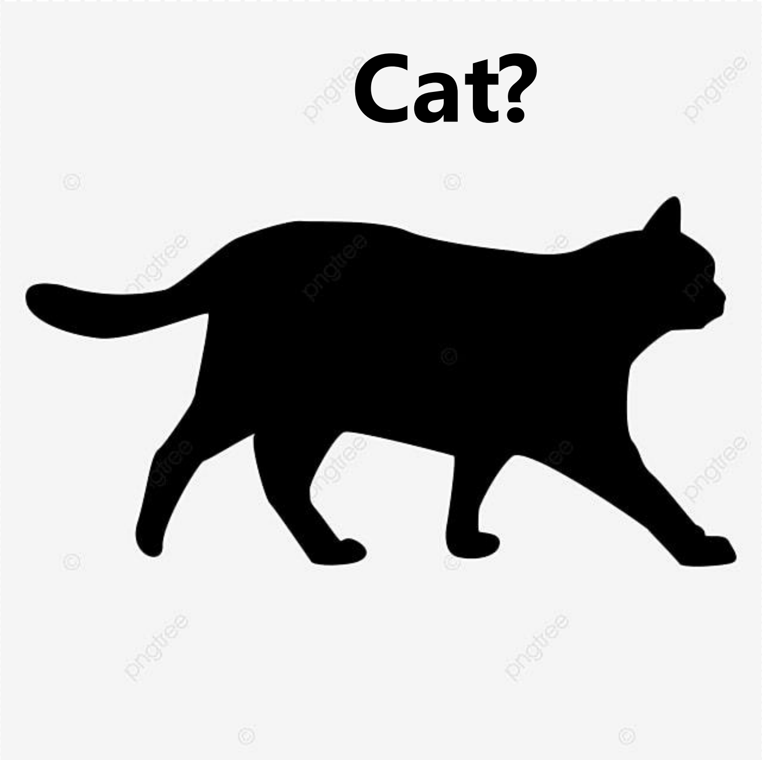

1
20
10
0
-10
-20
y
-5.0
-2.5
0.0
2.5
5.0
x
-sse
-30000
-60000
-90000


Plot of predicted decision boundary on set of Datapoints
10
5
0
−5
−10
X2
−10
−5
0
5
10
15
X1

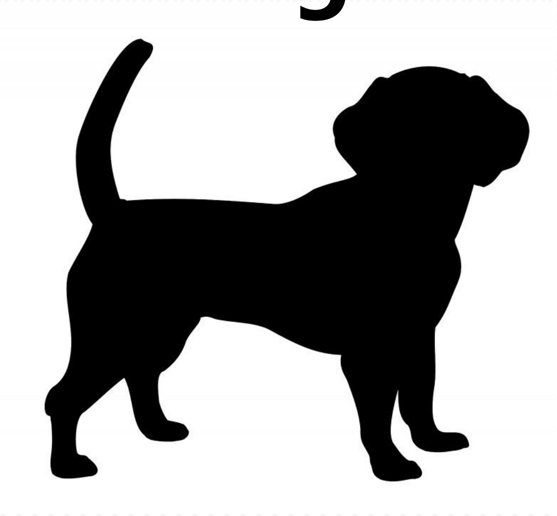
Logistic Regression

- **What do human think those image are, cat or dog?**
- **What do ai think those image are, cat or dog?**

Cat?

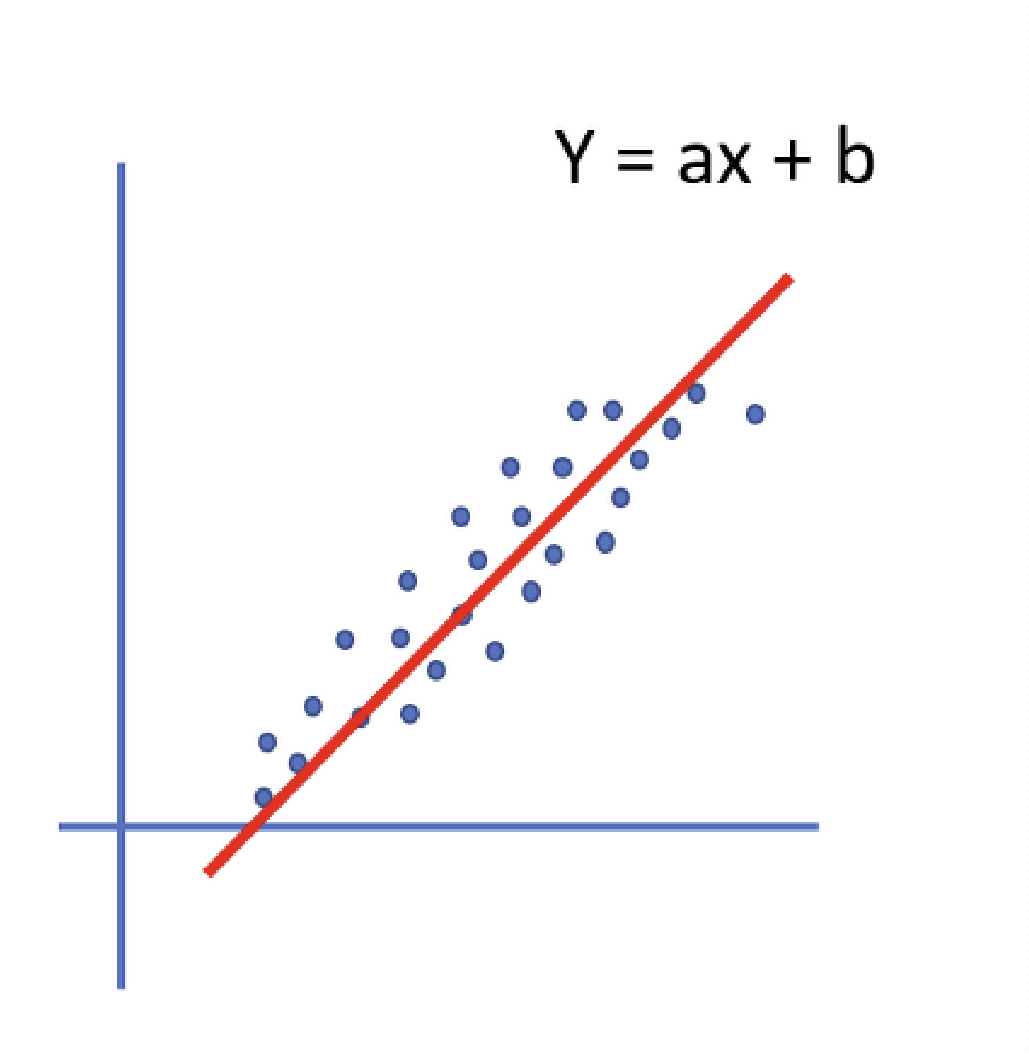


Dog?



Math. You Need to Know for AI
- **Linear Algebra**

Linear algebra (1)



$$A_1 x_1 = b_1$$
$$A_2 x_2 = b_2$$

$$A\vec{x} = \vec{b}$$

$$A\mathbf{x} = \mathbf{b}$$

Linear algebra (2)

- How to convert lineare equation to matrixs

Linear Equations

$$ax_1 + bx_2 = e$$

$$cx_1 + dx_2 = f$$

Matrixs

$$\begin{bmatrix} \boxed{a \quad b} \\ c \quad d \end{bmatrix} \boxed{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}} = \begin{bmatrix} \boxed{a \cdot x_1 + b \cdot x_2} \\ c \cdot x_1 + d \cdot x_2 \end{bmatrix}$$

$$\begin{bmatrix} a \quad b \\ \boxed{c \quad d} \end{bmatrix} \boxed{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}} = \begin{bmatrix} a \cdot x_1 + b \cdot x_2 \\ \boxed{c \cdot x_1 + d \cdot x_2} \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

Linear algebra: inverse

Does the equation have the root?

How can we know the equations have the root?

Linear algebra: inverse

$$\boxed{AA^{-1} = A^{-1}A = I}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix}$$

$$ad - bc \neq 0$$

- The equation cannot be solved when the determinant of the equation is zero.

Linear algebra: 1 and 2 variable

Equation	Solution	판별식
$ax = b$	$x = a^{-1}b$	$a \neq 0$
$ax_1 + bx_2 = e$ $cx_1 + dx_2 = f$	$x_1 = \dfrac{de - bf}{ad - bc}$ $x_2 = \dfrac{af - ce}{ad - bc}$	$ad - bc \neq 0$
$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$	$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} \begin{bmatrix} e \\ f \end{bmatrix}$	$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) \neq 0$

Linear algebra: 1 and 2 variable

- Determine if the following linear equations have the roots.

$$\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 \\ 6 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 \\ 6 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Linear algebra: 1 and 2 variable

- Determine if the following linear equations have the roots.

  - **A solution exists.**

$$\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

  - **No solution exists.**

$$\begin{bmatrix} 4 & 2 \\ 6 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

  - **Infinitely many solutions exist.**

$$\begin{bmatrix} 4 & 2 \\ 6 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Linear algebra: Transpose

- Transpose of Matrix

$$\begin{bmatrix} 4 & 5 & 2 & 1 \\ 2 & 3 & 8 & 0 \\ 1 & 0 & 7 & 2 \end{bmatrix}^T = \begin{bmatrix} 4 & 2 & 1 \\ 5 & 3 & 0 \\ 2 & 8 & 7 \\ 1 & 0 & 2 \end{bmatrix}$$

- Symmetric of Matrix

$$A^T = A$$

$$\begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 0 \\ 1 & 0 & 7 \end{bmatrix}^T = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 0 \\ 1 & 0 & 7 \end{bmatrix}$$

Linear algebra: inner product / dot product

$$\langle x, y \rangle = x^T y$$

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, y = \begin{bmatrix} 5 \\ -2 \\ 1 \end{bmatrix} \qquad x^T y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ -2 \\ 1 \end{bmatrix} = 4$$

Linear algebra: Multivariable

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$\Downarrow$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Linear algebra: product

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \boxed{a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4} \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 4 \cdot 2 + 2 \cdot 3 + 0 \cdot 4 \\ 9 \cdot 1 + 5 \cdot 2 + 0 \cdot 3 + 0 \cdot 4 \\ 4 \cdot 1 + 0 \cdot 2 + 2 \cdot 3 + 4 \cdot 4 \\ 6 \cdot 1 + 1 \cdot 2 + 8 \cdot 3 + 3 \cdot 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 19 \\ 26 \\ 44 \end{bmatrix}$$

Linear algebra: inverse

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$AA^{-1} = A^{-1}A = I$$

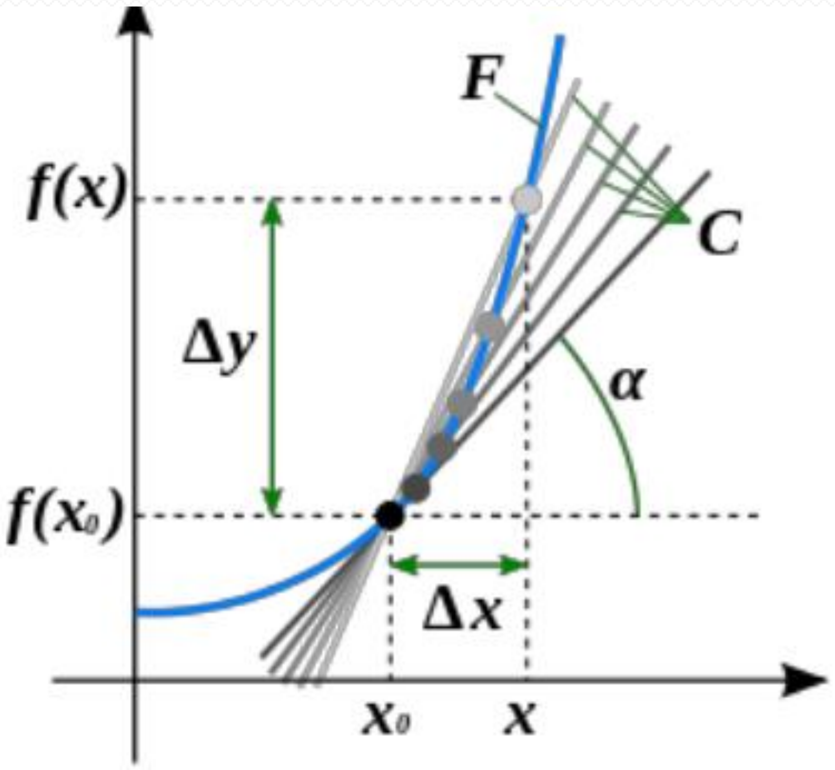
- The equation cannot be solved when the determinant of the equation is zero.

Math. You Need to Know for AI
- Chain rule of derivative

Derivative of 1 variable

- First order derivative

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$



Derivative of 1 variable: table

$f(x)$	$f'(x)$	$f(x)$	$f'(x)$
x^n	nx^{n-1}	e^x	e^x
$\ln(x)$	$1/x$	$\sin(x)$	$\cos(x)$
$\cos(x)$	$-\sin(x)$	$\tan(x)$	$\sec^2(x)$
$\cot(x)$	$-\text{cosec}^2(x)$	$\sec(x)$	$\sec(x)\tan(x)$
$\text{cosec}(x)$	$-\text{cosec}(x)\cot(x)$	$\tan^{-1}(x)$	$1/(1+x^2)$
$\sin^{-1}(x)$	$1/\sqrt{1-x^2}$ for $\|x\|<1$	$\cos^{-1}(x)$	$-1/\sqrt{1-x^2}$ for $\|x\|<1$
$\sinh(x)$	$\cosh(x)$	$\cosh(x)$	$\sinh(x)$
$\tanh(x)$	$\text{sech}^2(x)$	$\coth(x)$	$-\text{cosech}^2(x)$
$\text{sech}(x)$	$-\text{sech}(x)\tanh(x)$	$\text{cosech}(x)$	$-\text{cosech}(x)\coth(x)$
$\sinh^{-1}(x)$	$1/\sqrt{x^2+1}$	$\cosh^{-1}(x)$	$1/\sqrt{x^2-1}$ for $x>1$
$\tanh^{-1}(x)$	$1/(1-x^2)$ for $\|x\|<1$	$\coth^{-1}(x)$	$-1/(x^2-1)$ for $\|x\|>1$

Derivative of 1 variable: chain rule

- case1

$$\boxed{\frac{\partial}{\partial x} f(x)g(x) = f'(x)g(x) + f(x)g'(x)}$$

- case2

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

$$f(x) = \frac{1}{2}x^2 \quad g(x) = b - ax$$

$$f'(x) = x \qquad g'(x) = -a$$

Direct method

$$f(x)g(x) = \frac{bx^2 - ax^3}{2}$$

$$(f(x)g(x))' = bx - \frac{3a}{2}x^2$$

Chain rule

$$f'(x)g(x) + f(x)g'(x)$$

$$= bx - ax^2 - \frac{1}{2}x^2$$

Derivative of 1 variable: chain rule

- case1

$$\frac{\partial}{\partial x} f(x)g(x) = f'(x)g(x) + f(x)g'(x)$$

- case2

$$\boxed{\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)}$$

$$f(x) = \frac{1}{2}x^2 \quad g(x) = b - ax$$

$$f'(x) = x \qquad g'(x) = -a$$

<u>Direct method</u>

$$f(g(x)) = \frac{1}{2}\left(a^2x^2 - 2abx + b^2\right)$$

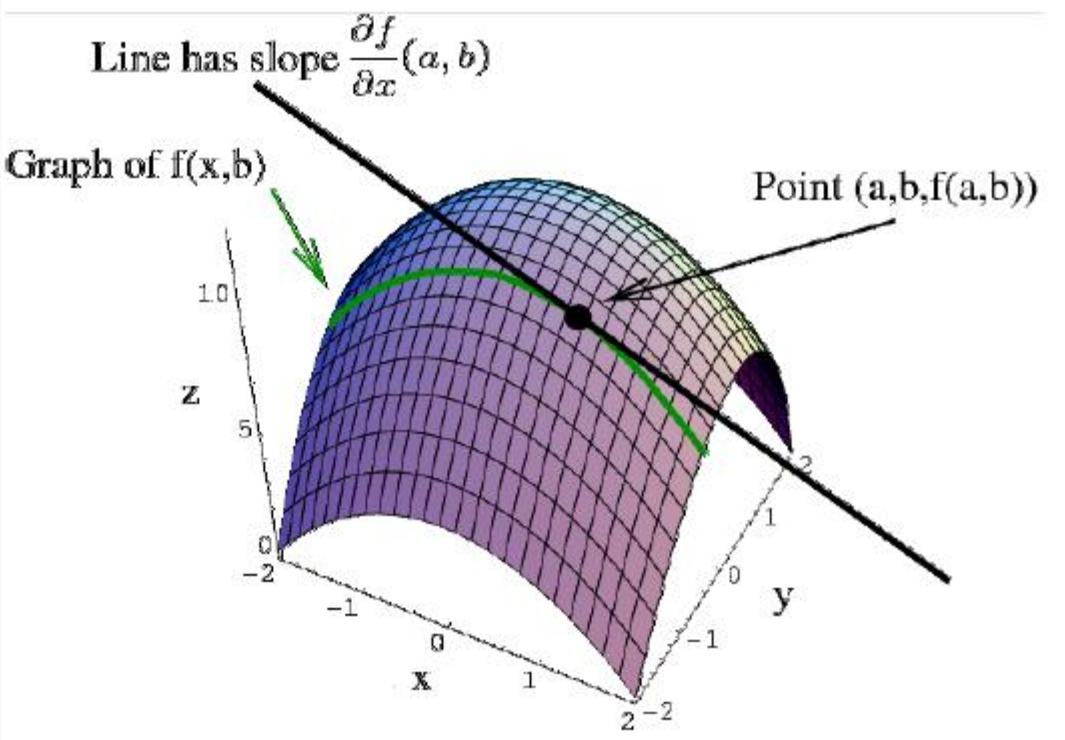
$$f(g(x))' = a^2x - ab$$

<u>Chain rule</u>

$$f'(g(x))g'(x)$$

$$= -a(b - ax)$$

Derivatives of multivariable: partial derivatives



$$\frac{\partial f(x_0, y_0)}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x, y_0) - f(x_0, y_0)}{\Delta x}$$

$$\frac{\partial f(x_0, y_0)}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x_0, y_0 + \Delta y) - f(x_0, y_0)}{\Delta y}$$

Derivatives of vector: Gradient (∇)

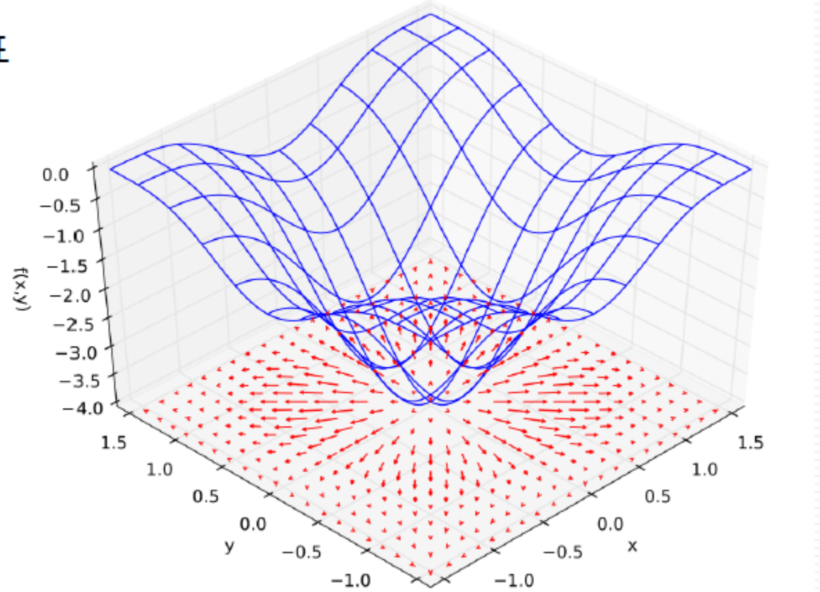
- The gradient is a vector of partial derivatives representing the rate and direction of the steepest ascent of a function.

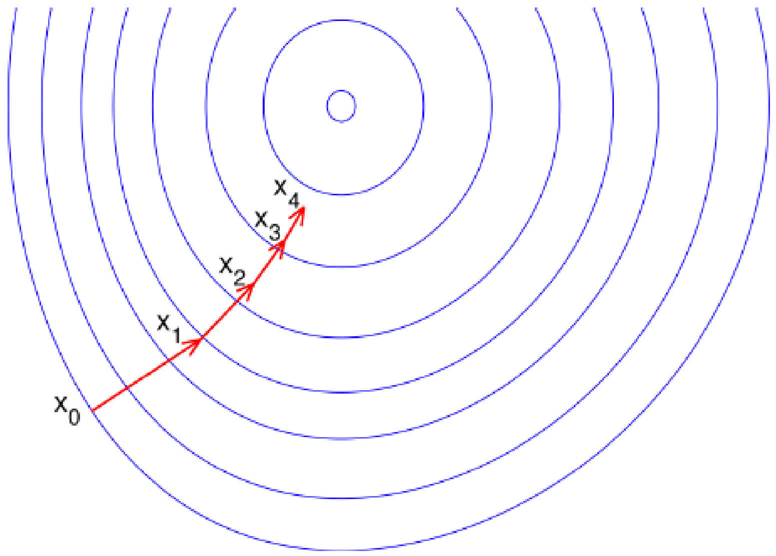
- Gradient는 편미분값의 벡터 표현입니다.

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- n차원에서는

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$





Summary of Derivatives and Linear algebra

$$\nabla\left(x^T \mathrm{a}\right) = \mathrm{a}$$

$$\nabla\left(\frac{1}{2}\mathbf{x}^T\mathbf{x}\right) = \mathbf{x}$$

$$\nabla\left(\frac{1}{2}\mathbf{x}^T A\mathbf{x}\right) = \frac{1}{2}(A + A^T)\mathbf{x}$$

Math. You Need to Know for AI
- **gradient decent**

Gradient descent algorithm

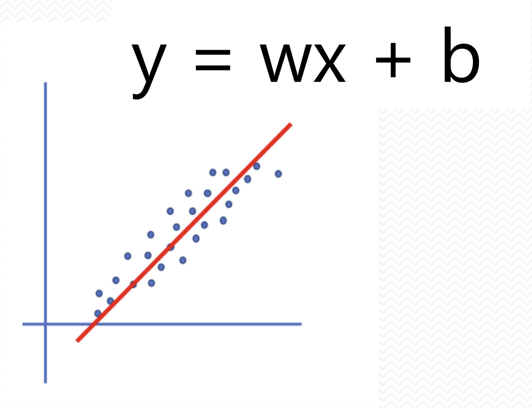
- 비용 함수 최소화
- Gradient descent는 주로 최적화 문제에 사용된다.
- 주어진 비용 함수, 비용 (W, b)에 대해 비용을 최소화하기 위해 W, b를 찾는다.
- 일반적인 함수 : 비용 (w1, w2,…)에 적용 가능

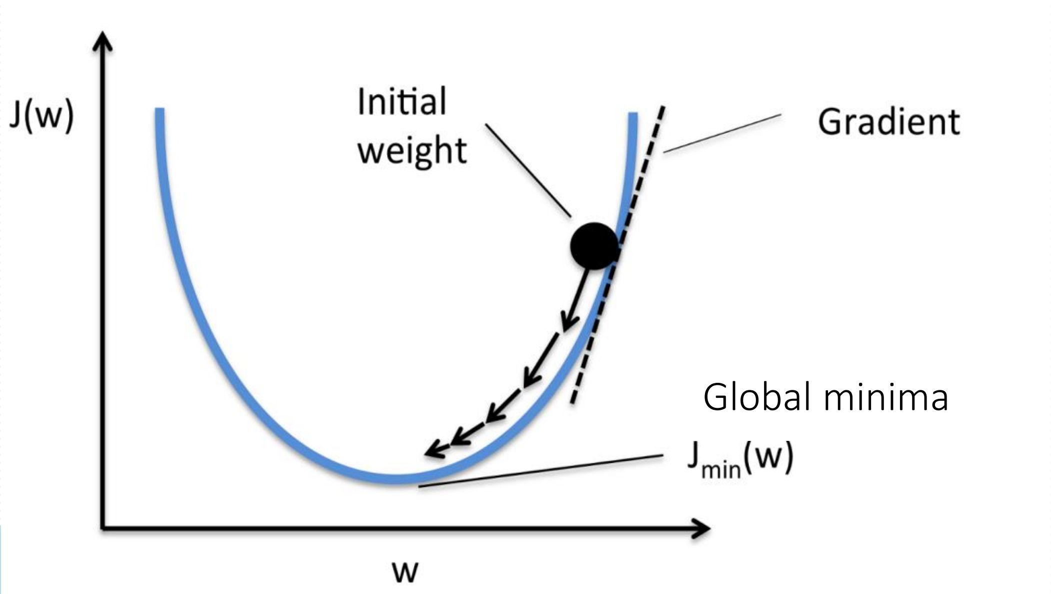
Gradient descent algorithm

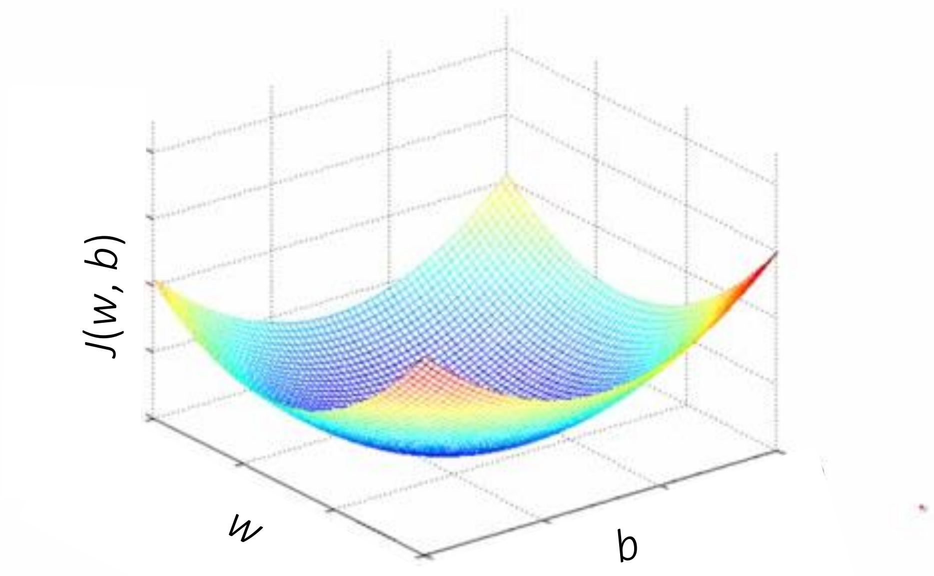
- Initialize parameters
- Compute predictions
- Calculate loss
- Compute gradients
- Update parameters
- Repeat

Gradient Descent

- Example: mean squared error as loss function
- Cost function is a convex function
- **Goal**: Find values of *w* and *b* such that *J* is minimum,
  i.e. find the global minima of the cost function
- Move in the direction of negative gradient

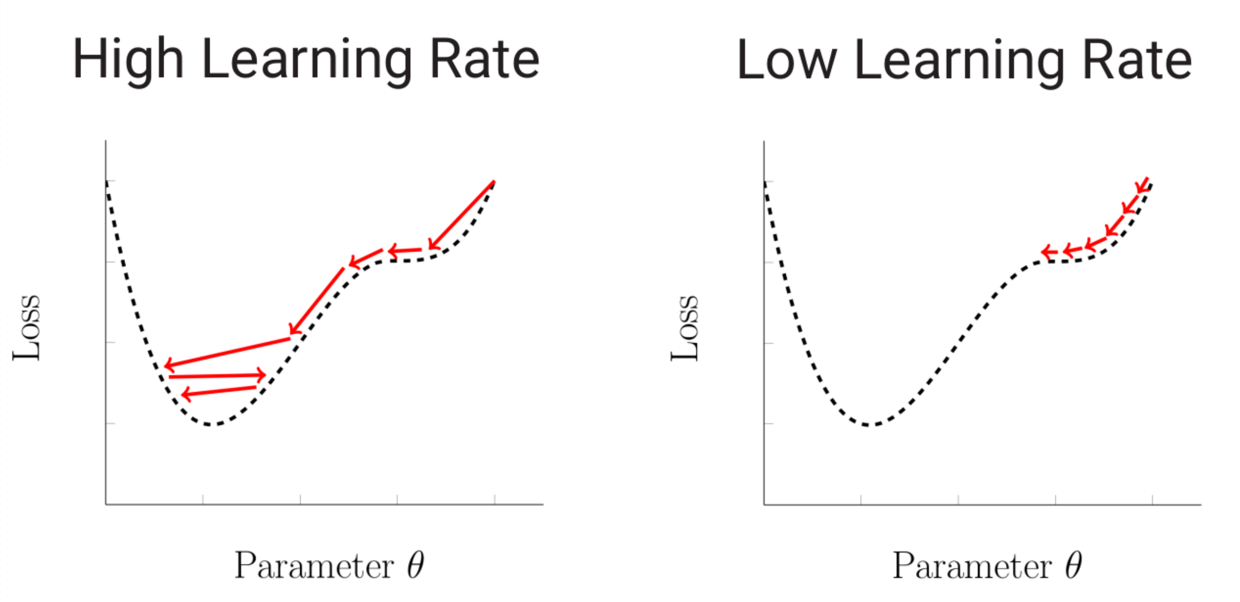




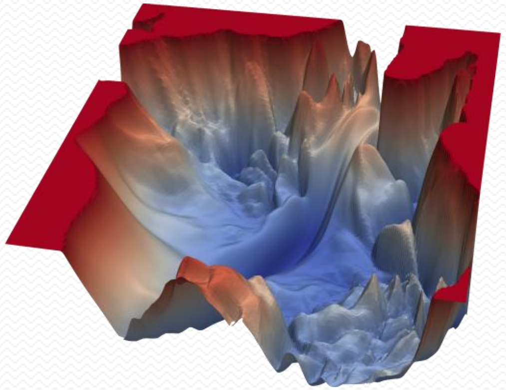


Learning Rate

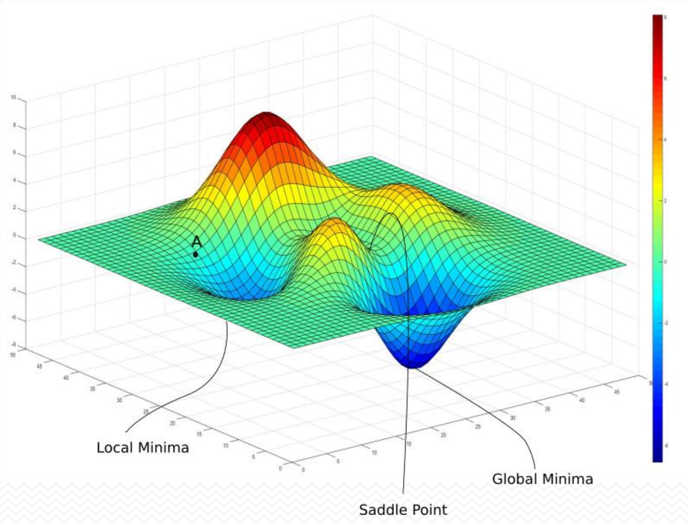
- A constant that determines the step size of gradient descent
- Too large – risk of overshooting the minima
- Too small – too slow to reach the minima



Local minima



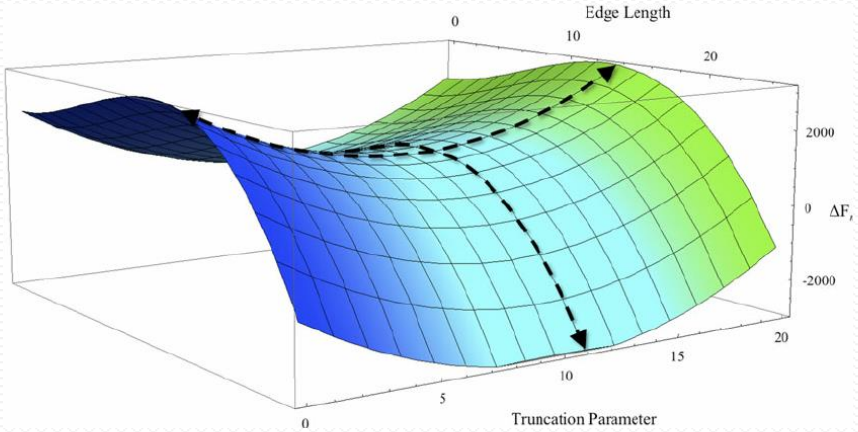
loss contour of a VGG-56 DNN’s loss function



- Gradient of local and global minima is zero
- Improper initialization point may cause convergence to a local minima – you’re doomed!

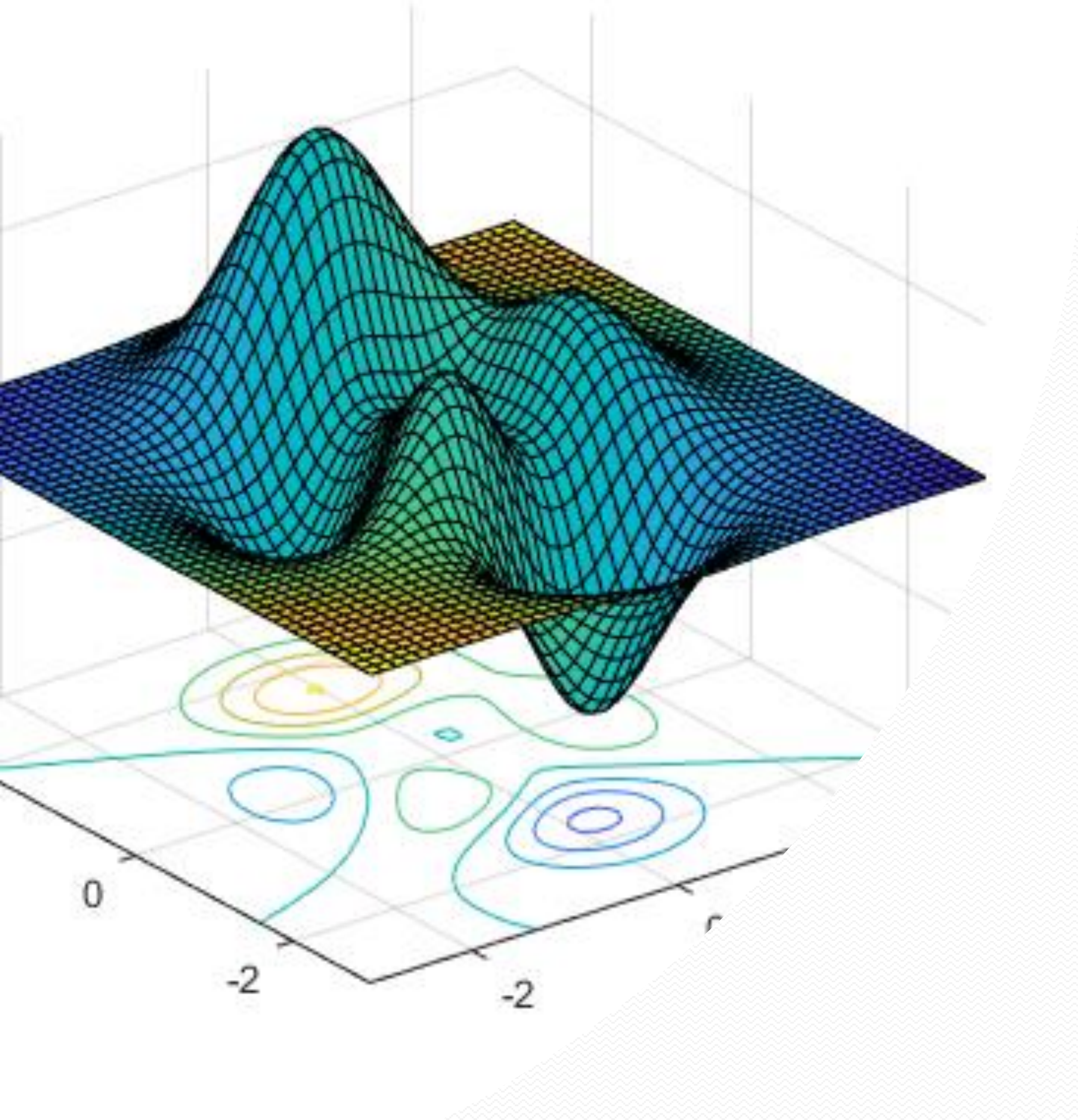
Multi dimension

Saddle Points



- A minima in one direction, a maxima in another direction
- Occurs where two maxima meet

Math. You Need to Know for AI
- ***Regression(Linear/Logistic)***



Linear Regression

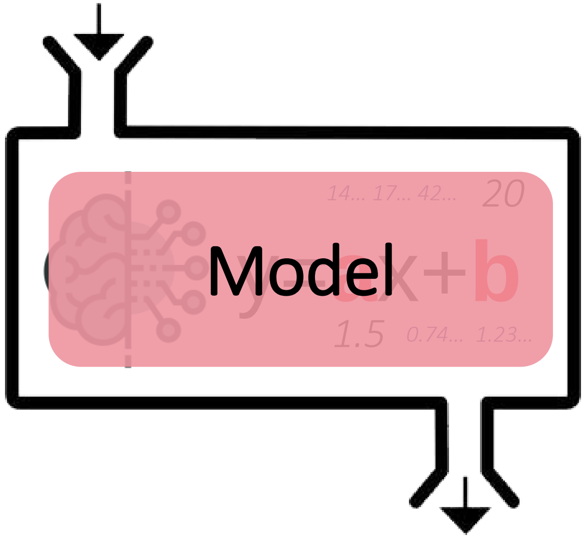
ARTIFICIAL INTELLIGENCE?

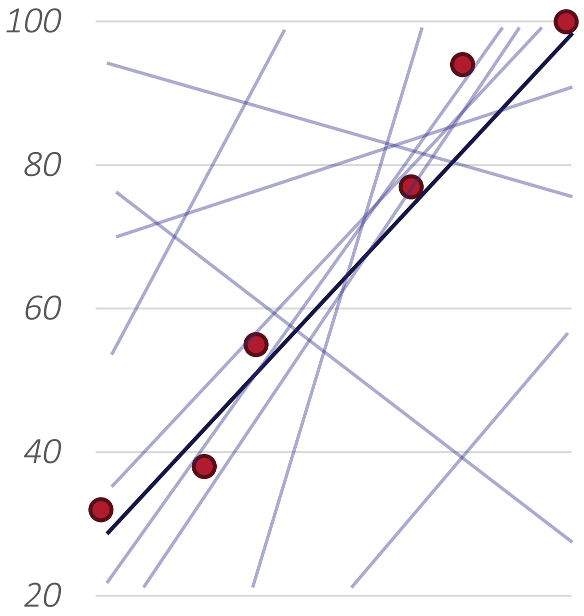
THINK OF AI AS ANOTHER KIND OF LIBRARY WITH MANY TUNING VALUES

Data	*X*	*Y*
#1	6	29
#2	14	41
#3	22	53
#4	38	77
#5	44	86
#6	52	98

⋮

A.I. Powered library libX



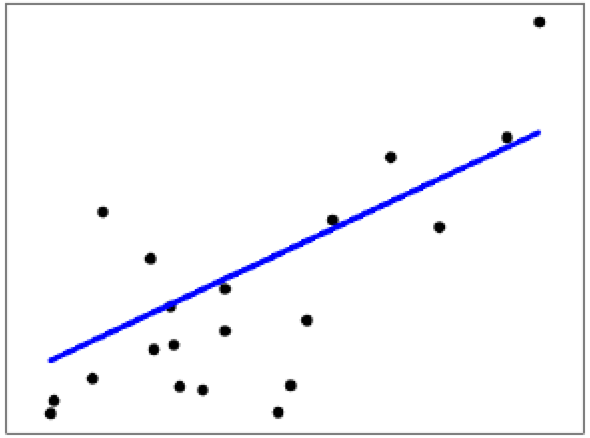


Optimization methods: Linear regression

1.1.1. Ordinary Least Squares

`LinearRegression` fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} ||Xw - y||_2{}^2$$



Optimization methods: Linear regression

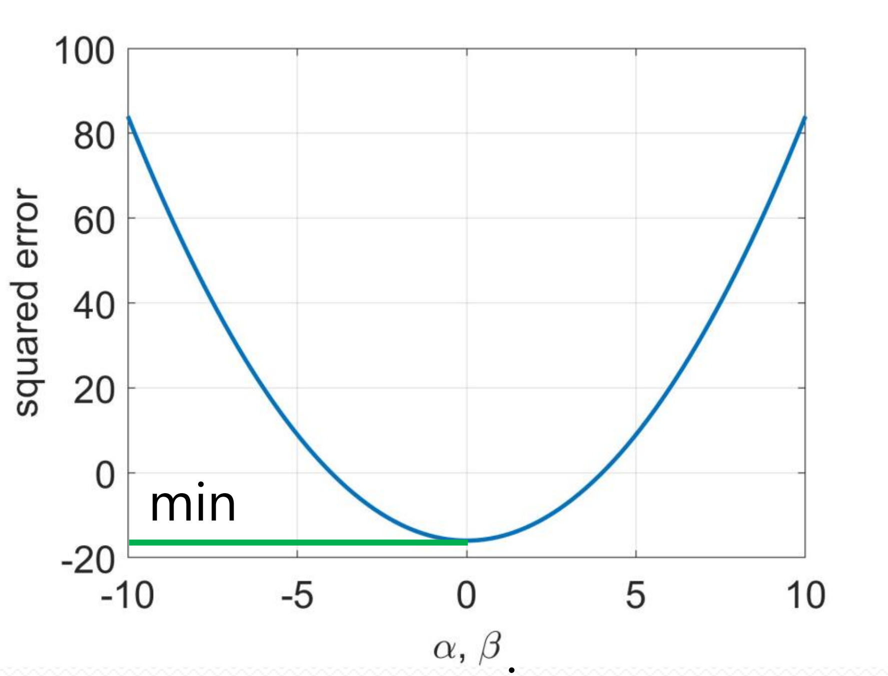
1. **Suggest data**: student
2. **Data of interest (DOI)** : age, grade, etc.
3. **Value of interest (VOI):** minimum value
4. **Find the DOV with DOI:** Hong Gildong

$$\text{Hong Gildong} = \underset{\text{student}}{\arg\min}\ \text{age}$$

Optimization methods: Linear regression

1. **Suggest data**: *X, Y* (y=ax+b)
2. **Data of interest (DOI)** : squared error
3. **Value of interest (VOI):** least
4. **Find the DOI data with VOI:** (α, β)

$$(\alpha, \beta) = \arg\min_{a,b} \sum_{j=1}^{n} |Y_j - (ax_j + b)|^2$$



argmin

Optimization methods: Linear regression

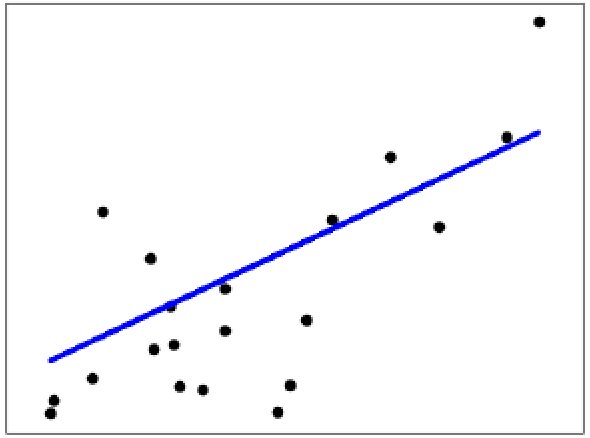
1.1.1. Ordinary Least Squares

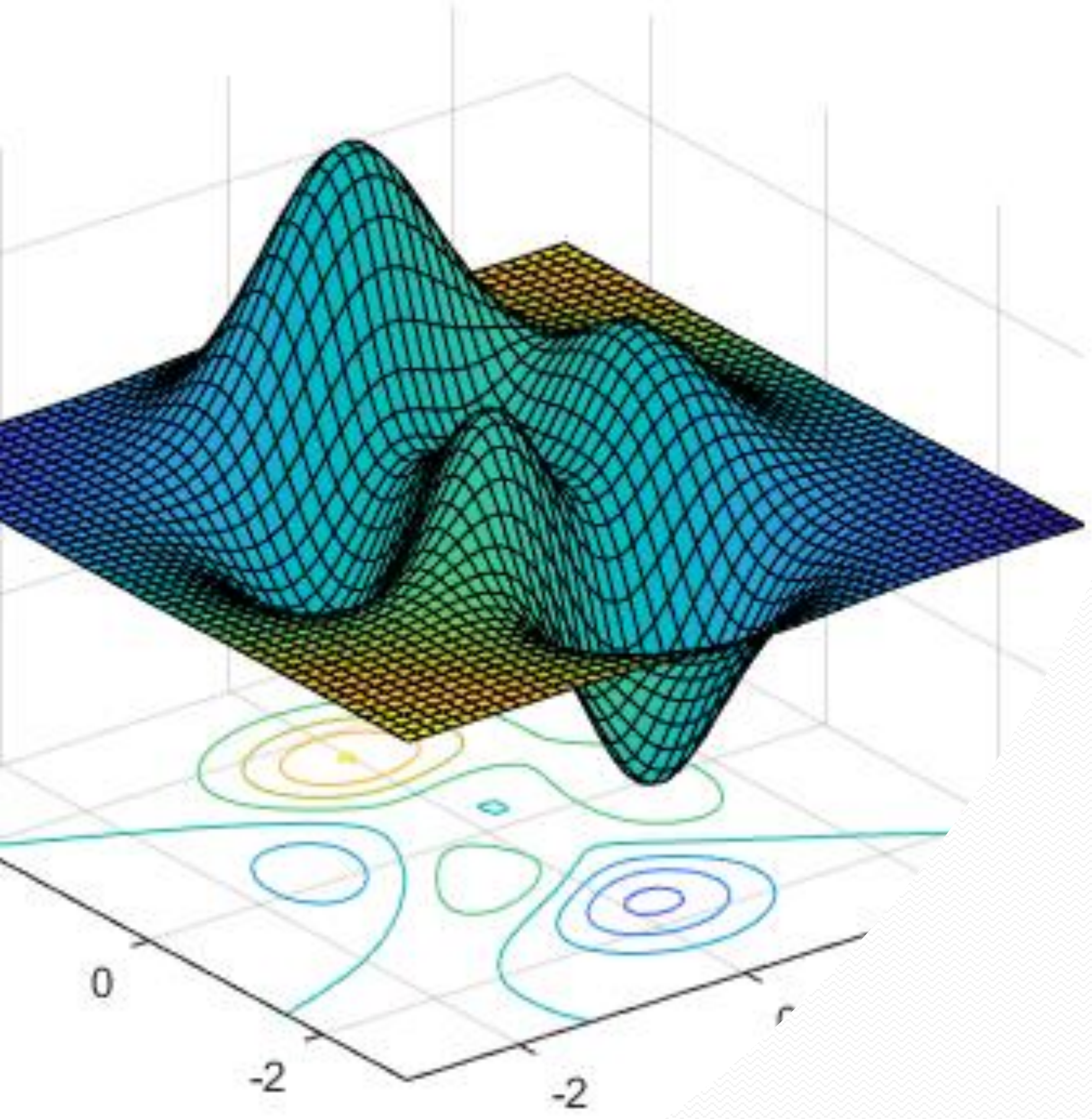
`LinearRegression` fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{\boxed{w}} \boxed{||Xw - y||_2^2}$$

- Weight
- Control Variables

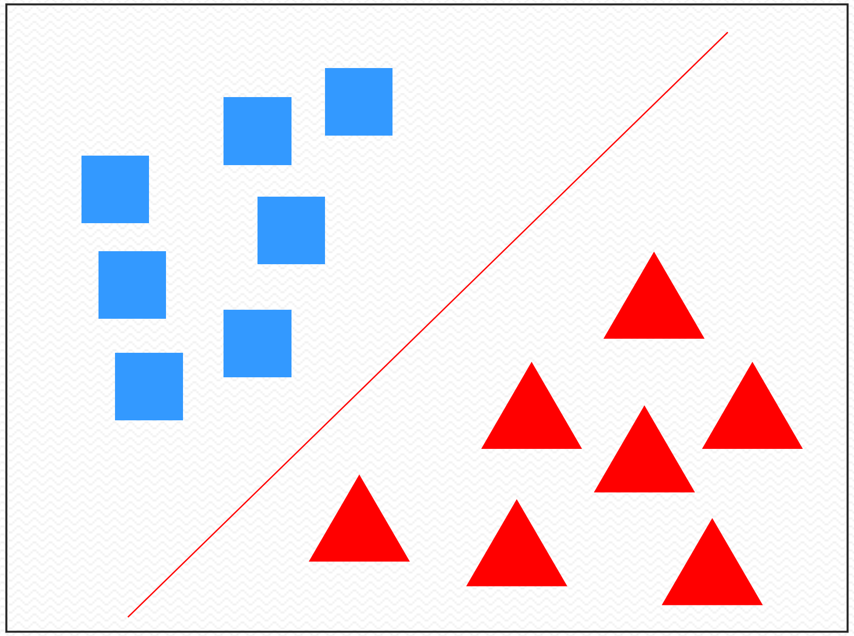
- Objective function
- Loss / Cost Function

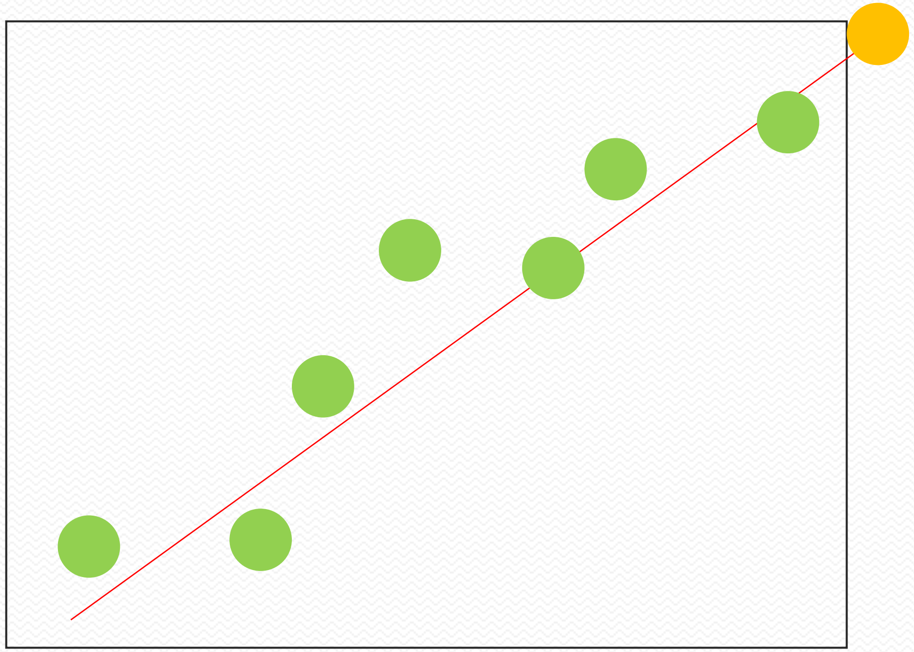




Logistic Regression

Logistic Vs Linear





Discrete : classification

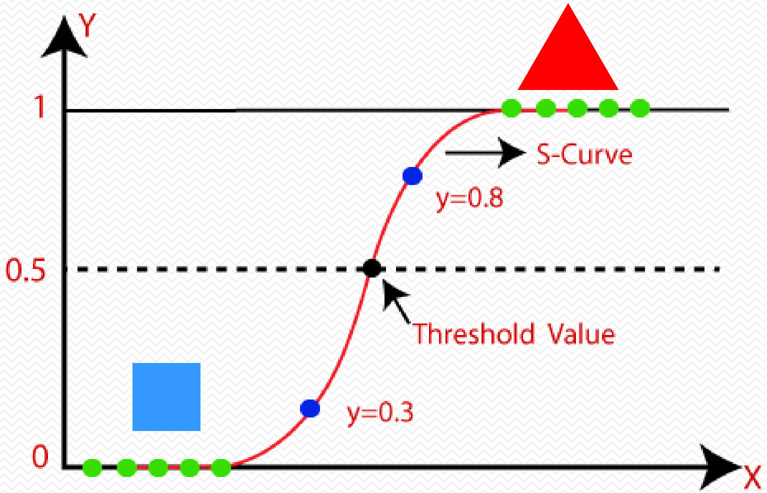
Shoues size / employee of group

Continuous : prediction

Time/ weight/ height

Logistic regression

- Logistic is for classification
- Logistic Regression is a method for predicting binary outcomes



Sigmoid function

A mathematical expression called the sigmoid function converts any real-valued input to a number

between 0 and 1

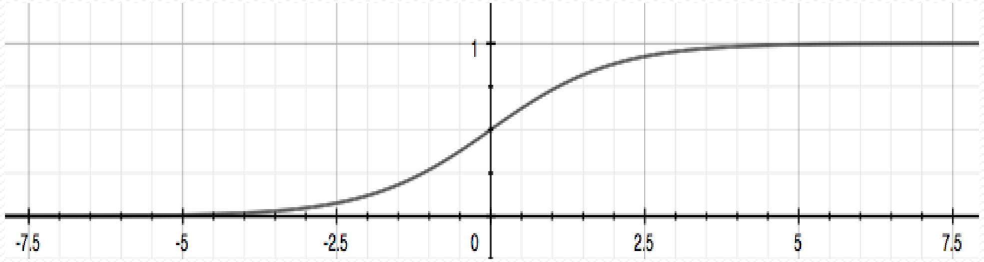
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z = w^T x + b$$

https://ai.plainenglish.io/activation-functions-in-machine-learning-86eafce4b750

Logistic regression: cost function

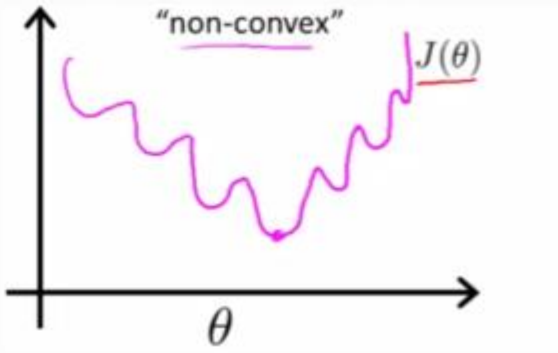
- Assume we apply **MSE** as cost function for Sigmoid

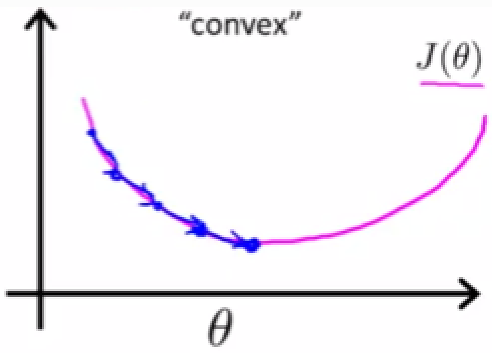


$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$Cost(h_\theta(x^{(i)}),\, y) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

- Cost function will be shown as "Non-convex", which is not expected





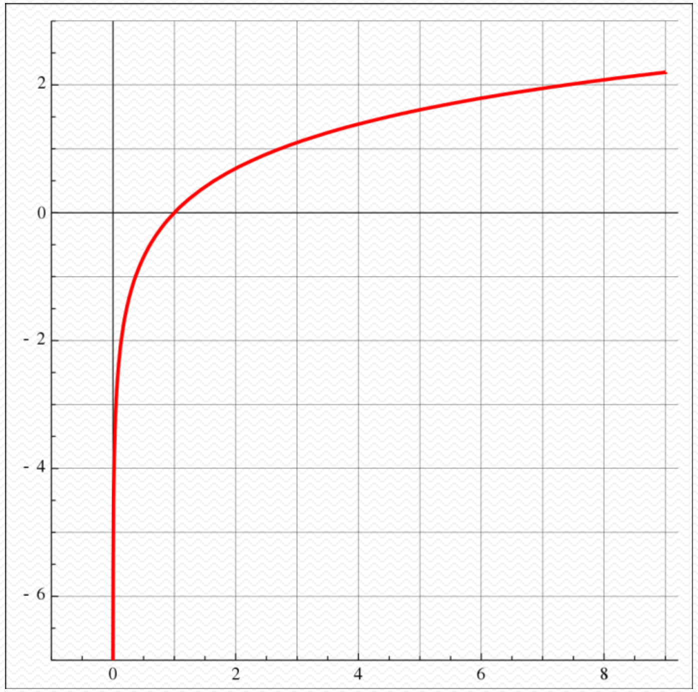
https://en.wikipedia.org/wiki/Logistic_regression

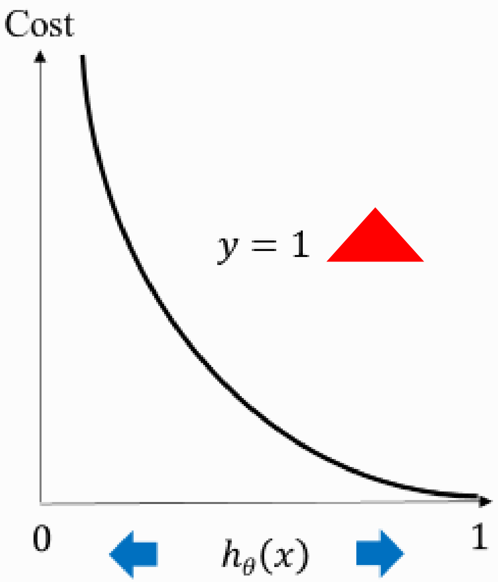
Logistic regression: cost function

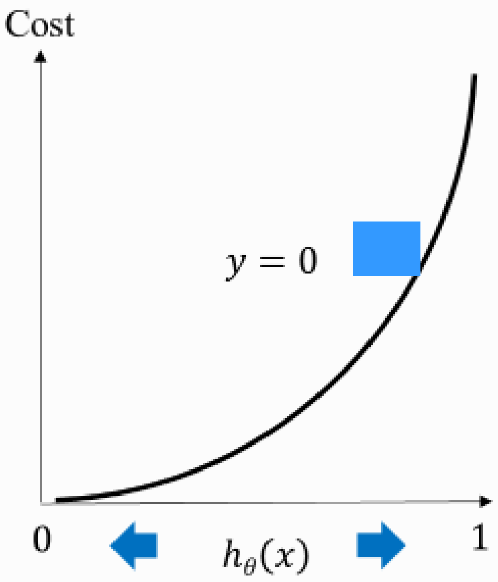
- **Binary Cross Entropy** represents Logistic regression

$$Cost(h_\theta(x^{(i)}),\ y) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$

Log function







https://en.wikipedia.org/wiki/Logistic_regression

Logistic regression: cost function

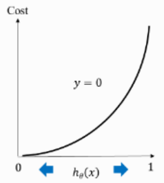
- Cost function with Condition

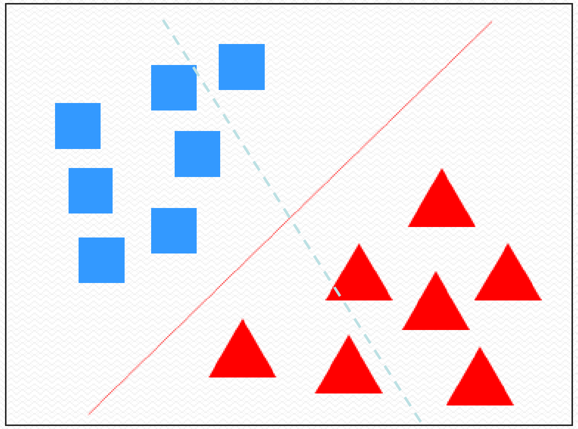
$$C(H(x), y) = \begin{cases} -\log(H(X)) & if\ y = 1 \\ -\log(1 - H(x)) & if\ y = 0 \end{cases}$$

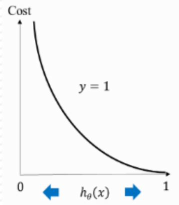
Single Function expression

$$\boxed{C(H(x), y) = -y * log(H(X)) - (1 - y) * log(1 - H(X))}$$

Categorical crossentropy func.







Summary: logistic regression

- Cost Function of Logistic Regression

$$C(H(x), y) = -y * log(H(X)) - (1-y) * log(1-H(X))$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[logloss(h_\theta(x^{(i)}), y^{(i)})]$$

- Gradient Descent of above cost function

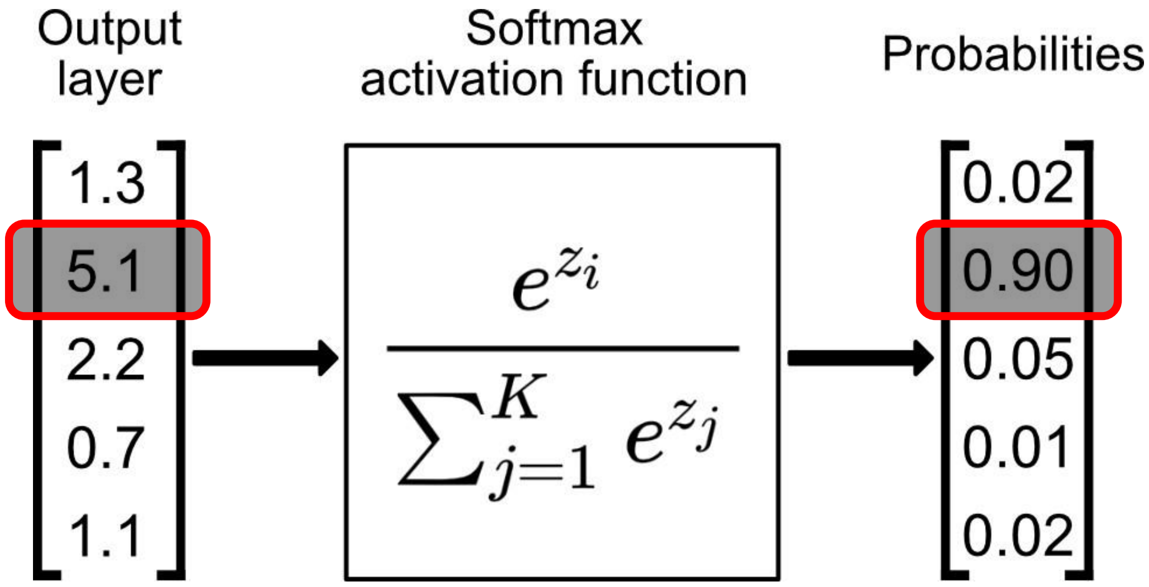
$$\theta = \theta - \alpha\frac{\partial}{\partial\theta}J(\theta)$$

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

CALCULATE THE OUTPUT DATA / SOFTMAX

CALCULATE THE CONFIDENCE SCORE (Probabilities)

- Suitable for classification problems
  multi-class classification
- Sum of scores will be 1
- Highest score is the prediction class

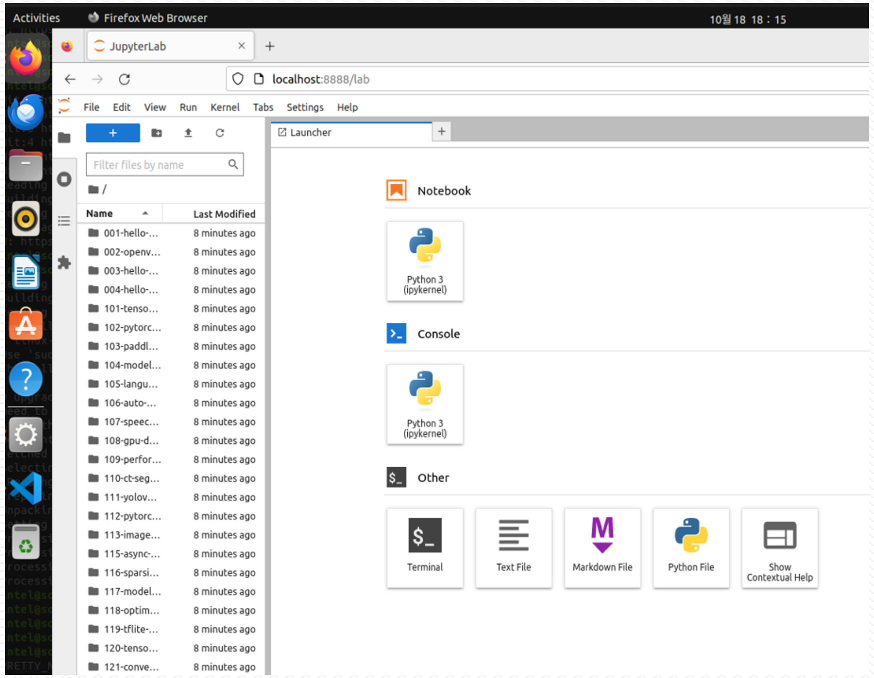


Hands on

- Basic python lib.

Install the OpenVINO (for Jupyter Notebook)

- Input the command below,
  - (.openvino_env) $ git clone --depth=1 https://github.com/openvinotoolkit/openvino_notebooks.git
  - (.openvino_env) $ cd openvino_notebooks
  - (.openvino_env) $ pip install -U pip
  - (.openvino_env) $ pip install wheel setuptools
  - (.openvino_env) $ pip install -r requirements.txt
  - (.openvino_env) $ jupyter lab notebooks



- Refer to the link
  https://docs.openvino.ai/2023.1/notebooks_installation.html
  https://github.com/openvinotoolkit/openvino_notebooks

Python lib

matplotlib

NumPy

SymPy

TensorFlow

scikit learn

Python lib

NumPy

Example

A 2-dimensional array of size 2 x 3, composed of 4-byte integer elements:

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
>>> type(x)
<class 'numpy.ndarray'>
>>> x.shape
(2, 3)
>>> x.dtype
dtype('int32')
```

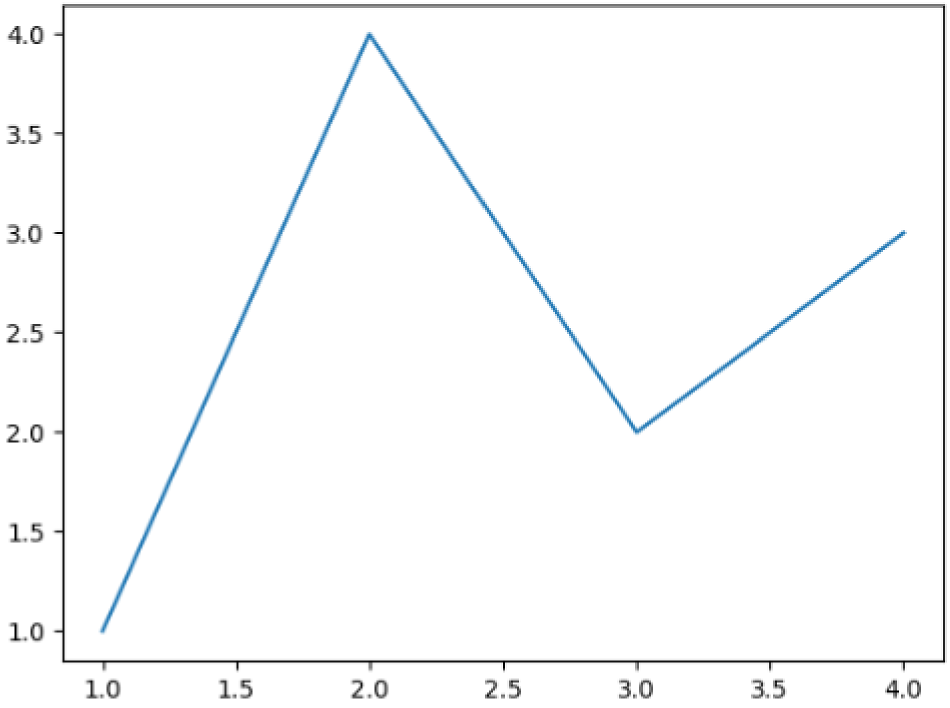
The array can be indexed using Python container-like syntax:

```
>>> # The element of x in the *second* row, *third* column, namely,
>>> x[1, 2]
6
```

For example slicing can produce views of the array:

```
>>> y = x[:,1]
>>> y
array([2, 5], dtype=int32)
>>> y[0] = 9 # this also changes the corresponding element in x
>>> y
array([9, 5], dtype=int32)
>>> x
array([[1, 9, 3],
       [4, 5, 6]], dtype=int32)
```

matplotlib



Hands on

- Basic python lib. (basic)

Python basic (변수선언)

[1]:
```python
a = 1 #int로 선언
b = 2. #float으로 선언
c = "String" #string으로 선언
```

[2]:
```python
print(a)
print(b)
print(c)
```

[3]:
```python
print(type(a))
print(type(b))
print(type(c))
```

Python basic (함수선언)

```
[4]: def f(x, y):
         val = x + y
         return val
```

```
[5]: a = 1
     b = 2.
     d = f(a,b)
     print(d)
```

Python basic (익명함수선언)

```
f = lambda x,y : x + y
```

```
a = 1
b = 2.
d = f(a,b)
print(d)
```

Python basic (주요변수타입:리스트)

[8]:
```
a = [1, 3, 4]
print(a)
```

[9]:
```
a[0] = 9
print(a)
```

[10]:
```
b = [1, 3, 'string']
print(b)
```

[11]:
```
b.append(6.24)
print(b)
```

```
cars = ["Ford", "Volvo", "BMW"]

cars.pop(1)

print(cars)
```

```
cars = ["Ford", "Volvo", "BMW"]

cars.remove("Volvo")

print(cars)
```

Python basic(주요변수타입:딕셔너리)

```
info = {'A' : 2.3, 'B' : 'C', 5 : 'D'}
print(info)
```

```
info['A'] = 5.2
print(info)
```

```
info['Hello'] = [1, 2, 3, 4, 'World.']
print(info)
```

Python basic (여러가지 for loop)

```
items = [[1,2], [3,4], [5,6]]
for item in items:
    print(item[0], item[1])
```

```
for item1, item2 in items:
    print(item1, item2)
```

```
items = [(1,2), (3,4), (5,6)]
for item1, item2 in items:
    print(item1, item2)
```

Python basic(여러가지 for loop)

```
info = {'A' : 1, 'B' : 2, 'C' : 3}
for key in info:
    print(key, info[key])
```

```
for key, value in info.items():
    print(key, value)
```

Python basic(zip이 들어간 for loop)

```python
items1 = [[1,2], [3,4], [5,6]]
items2 = [['A','B'], ['C','D'], ['E','F']]
print(items1)
print(items2)
```

```python
for digits, characters in zip(items1, items2):
    print(digits, characters)
```

Python basic(한줄 for loop)

```
a = []
for k in range(0,5):
    a.append(k)
print(a)
```

```
a = [k for k in range(0,5)]
print(a)
```

```
a = [k for k in range(0,5) if k % 2 == 0]
print(a)
```

```
a = {k : k*10 for k in range(0,5) }
print(a)
```

Hands on
- Basic python lib.
(numpy)

Python numpy (일반배열)

```
import numpy as np
a = np.array([1,2,3,4])
print(a)
```

```
print(a + a)
```

```
b = [1,2,3,4]
print(b + b)
```

- 2x2 행렬의 생성

```
a = np.array([[1,2],[3,4]])
print(a)
```

Python numpy (행렬)

```
a = np.array([[1,2],[3,4]])
print(a)
```

```
a = np.array([[[1,2],[3,4]], [[1,2],[3,4]]])
print(a)
```

Python numpy (행렬)

```
a = np.array([1,2,3,4])
b = np.array([[1],[2],[3],[4]])
print(a)
print(b)
```

```
print(a)
print(a.shape)
print(b)
print(b.shape)
```

Python numpy (전치행렬 과 reshape)

```
a = np.array([[1],[2],[3],[4]])
print(a) #shape = (4,1)
print(a.T) #shape = (1,4)
print(a.T.reshape(-1,4))
print(a.shape)
print(a.T.reshape(-1,4).T.shape)
```

```
a = np.array([1,2,3,4])
b= a.reshape(4,-1)
print(a.reshape(4,-1))
print(a.shape,",",b.shape)
```

x.reshape(-1, 1) => shape(12,1)	x.reshape(-1, 2) => shape(6, 2)	x.reshape(-1, 3) => shape(4, 3)	x.reshape(-1, 4) => shape(3, 4)
In [5]: x.reshape(-1, 1) Out[5]: array([[0], [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]])	In [6]: x.reshape(-1, 2) Out[6]: array([[0, 1], [2, 3], [4, 5], [6, 7], [8, 9], [10, 11]])	In [7]: x.reshape(-1, 3) Out[7]: array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])	In [8]: x.reshape(-1, 4) Out[8]: array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])

Python numpy (전치행렬 과 reshape)

```
a = np.array([1,2,3,4,5,6])
print(a.reshape(3,2))
print(a.shape)
print(a.reshape(3,-1))
print(a.shape)
print(a.reshape(-1,2))
print(a.shape)
```

```
a = np.array([1,2,3,4])
print(a)
print(a.T)
b=a.reshape(4,-1)
print(b.shape)
print(b)
print(b.T.shape)
```

Python numpy (배열 인덱싱)

```
a = np.array([10,20,30,40,50,60])
print(a)
b = a[[4,2,0]]
print(b)
```

- shuffle: Modify a sequence in-place by shuffling its contents.

```
idx = np.arange(0, len(a))
print(idx)
np.random.shuffle(idx)
print(idx)
print(a[idx])
```

Hands on

- Basic python lib. (matplot)

Python Matplot lib.

```
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
```

```
x = np.linspace(-2, 2, 11)
f = lambda x: x ** 2
fx = f(x)
```

```
print(x)
print(fx)
```

Python Matplot lib.

```
plt.plot(x, fx, '-o')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.title('This is an example for 1d graph')
plt.show()
```

Python Matplot lib.

```
x = np.linspace(-2,2, 11)
y = np.linspace(-2,2, 11)

print(x)
print(y)
```

```
x,y = np.meshgrid(x,y)
print(x)
print(y)
# x.head()
```

Python Matplot lib.

```
f = lambda x,y : (x-1)**2 + (y-1)**2
```

```
z = f(x,y)
print(z)
```

```
from mpl_toolkits.mplot3d import Axes3D

ax = plt.axes(projection='3d', elev=50, azim=-50)
ax.plot_surface(x, y, z, cmap=plt.cm.jet)

ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$z$')

plt.show()
```

Python Matplot lib.

```
ax = plt.axes()
ax.contour(x, y, z, levels=np.linspace(0, 20, 20), cmap=plt.
ax.grid()
ax.axis('equal')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
plt.show()
```

Hands on

- ***Basic Math for AI (Linear Alge.)***

행렬/벡터 곱: 프로그램 예제

1. Jupyter를 사용하여 직사각 행렬(n x m)과 m차원 벡터의 곱을 구현해봅니다.
2. 작성한 알고리즘을 사용하여 다음 2가지 예제를 풀고, 답을 확인해봅니다.

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$\begin{bmatrix} 4 & 5 & 2 & 1 \\ 2 & 3 & 8 & 0 \\ 1 & 0 & 7 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

행렬/벡터 곱 : 프로그램 예제(cont'd)

$$A = \begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

```python
import numpy as np
```

```python
A = np.matrix('\
1, 4, 2, 0; \
9, 5, 0, 0;\
4, 0, 2, 4;\
6, 1, 8, 3')
x = np.matrix('1;2;3;4')
print(A)
print(x)
```

```python
import numpy as np
```

```python
A = np.array([\
              [1, 4, 2, 0],\
              [9, 5, 0, 0],\
              [4, 0, 2, 4],\
              [6, 1, 8, 3]])
x = np.array([1,2,3,4]).reshape(1,-1)
```

```python
b=A*x.T
print("A*x=",b)
```

```
A*x= [[15]
 [19]
 [26]
 [44]]
```

행렬/벡터 곱: 프로그램 예제

1. Python을 사용하여 직사각 행렬(n x m)과 m차원 벡터의 곱을 구현해봅니다.
2. 작성한 알고리즘을 사용하여 다음 2가지 예제를 풀고, 답을 확인해봅니다.

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 19 \\ 26 \\ 44 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 5 & 2 & 1 \\ 2 & 3 & 8 & 0 \\ 1 & 0 & 7 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 32 \\ 30 \end{bmatrix}$$

역행렬 : 프로그램 예제 (cont'd)

- 아래 선형 방정식의 해를 구해 봅니다.

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 15 \\ 19 \\ 26 \\ 44 \end{bmatrix}$$

역행렬 : 프로그램 예제(cont'd)

- 아래 선형 방정식의 해를 구해 봅니다.

$$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 9 & 5 & 0 & 0 \\ 4 & 0 & 2 & 4 \\ 6 & 1 & 8 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 15 \\ 19 \\ 26 \\ 44 \end{bmatrix}$$

```
import numpy as np
A = np.matrix('1, 4, 2, 0;\
9, 5, 0, 0;\
4, 0, 2, 4;\
6, 1, 8, 3')
b = np.matrix('15, 19, 26, 44');
Ainv = np.linalg.inv(A)
btrn = b.transpose();
x = Ainv*btrn
x
```

Hands on
- *Basic Math for AI(Derivative)*

'sympy' 라이브러리를 이용한 수학식 계산

```
import sympy as sp
```

```
x = sp.symbols('x')
y = sp.symbols('y')
z = sp.symbols('z')
```

Take a derivative of $f(x)$ with respect to x

$$f(x) = x^2 + 4x + 6$$

```
f = x**2 + 4*x + 6
f
```

$x^2 + 4x + 6$

- diff 함수를 이용해서 상기 f(x)를 x에 대해 미분해 봅시다.
- Diff 함수를 이용해서 상기 f(x)를 y에 대해 미분해 봅시다.

'sympy'를 이용한 미분계산

```
f = x**2+4*x+6
f
```

$$x^2 + 4x + 6$$

```
sp.diff(f, x)
```

$$2x + 4$$

```
sp.diff(f, y)
```

$$0$$

합성함수의 미분

Make a arbituray functions $f(x)$ with respect to x, y, and z

$$f_1(x,y,z) = (a_1cos(\beta_x x) + b_1sin(\beta_x x))(c_1cos(\beta_y y) + d_1sin(\beta_y y))(e_1cos(\beta_z z) + f_1sin(\beta_z z))$$
$$f_2(x,y,z) = (a_1cos(\beta_x x) + b_1sin(\beta_x x))(c_1cos(\beta_y y) + d_1sin(\beta_y y))(e_1cos(\beta_z z) + f_1sin(\beta_z (d-z)))$$

symbol 정의

•[11]:

```
a1 = sp.symbols('a1')
b1 = sp.symbols('b1')
c1 = sp.symbols('c1')
d1 = sp.symbols('d1')
e1 = sp.symbols('e1')
f1 = sp.symbols('f1')

betax = sp.symbols('betax')
betay = sp.symbols('betay')
betaz = sp.symbols('betaz')
d = sp.symbols('d')
```

미분실습 (symbolic lib.)

Make a arbituray functions $f(x)$ with respect to x, y, and z

$$f_1(x,y,z) = (a_1 cos(\beta_x x) + b_1 sin(\beta_x x))(c_1 cos(\beta_y y) + d_1 sin(\beta_y y))(e_1 cos(\beta_z z) + f_1 sin(\beta_z z))$$
$$f_2(x,y,z) = (a_1 cos(\beta_x x) + b_1 sin(\beta_x x))(c_1 cos(\beta_y y) + d_1 sin(\beta_y y))(e_1 cos(\beta_z z) + f_1 sin(\beta_z (d-z)))$$

함수 생성

```
# f = (a1*sp.sin(x))
ez1 = (a1*sp.cos(betax*x)+b1*sp.sin(betax*x))*(c1*sp.cos(betay*y)+d1*sp.sin(betay*y))* \
(e1*sp.cos(betaz*z)+f1*sp.sin(betaz*z))
ez1
```

$$(a_1 \cos(betaxx) + b_1 \sin(betaxx))(c_1 \cos(betayy) + d_1 \sin(betayy))(e_1 \cos(betazz) + f_1 \sin(betazz))$$

```
ez2 = (a1*sp.cos(betax*x)+b1*sp.sin(betax*x))*(c1*sp.cos(betay*y)+d1*sp.sin(betay*y))* \
(e1*sp.cos(betaz*z)+f1*sp.sin(betaz*(d-z)))
ez2
```

$$(a_1 \cos(betaxx) + b_1 \sin(betaxx))(c_1 \cos(betayy) + d_1 \sin(betayy))(e_1 \cos(betazz) + f_1 \sin(betaz(d-z)))$$

- diff 함수를 이용해서 상기 f1(x, y, z)를 x와 z에 대해 미분해 봅시다.
- diff 함수를 이용해서 상기 f2(x, y, z)를 x에 z에 대해 미분해 봅시다.

미분실습 (symbolic lib.)

Take a derivative of $f_1(x, y, z)$ with respect to x and z

$$\frac{\partial^2 f_1(x, y, z)}{\partial x \partial z}$$

```
sp.diff(sp.diff(ez1,x),z)
```

$$(c_1 \cos(betayy) + d_1 \sin(betayy))(-a_1 betax \sin(betaxx) + b_1 betax \cos(betaxx))(-betaze_1 \sin(betazz) + betazf_1 \cos(betaz$$

Take a derivative of $f_2(x, y, z)$ with respect to x and z

$$\frac{\partial^2 f_2(x, y, z)}{\partial x \partial z}$$

```
sp.diff(sp.diff(ez2,x),z)
```

$$(c_1 \cos(betayy) + d_1 \sin(betayy))(-a_1 betax \sin(betaxx) + b_1 betax \cos(betaxx))(-betaze_1 \sin(betazz) - betazf_1 \cos(betaz$$

Gradient (jacobian 함수 이용)

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

```
type(ez1)
```

```
sympy.core.mul.Mul
```

```
F = sp.Matrix(1,1,[ez1])
type(F)
```

```
sympy.matrices.dense.MutableDenseMatrix
```

```
grad = F.jacobian([x,y,z]).T
grad
```

$$\begin{bmatrix} \left(c_1 \cos\left(betayy\right) + d_1 \sin\left(betayy\right)\right)\left(e_1 \cos\left(betazz\right) + f_1 \sin\left(betazz\right)\right)\left(-a_1 betax \sin\left(betaxx\right) + b_1 betax \cos\left(betaxx\right)\right) \\ \left(a_1 \cos\left(betaxx\right) + b_1 \sin\left(betaxx\right)\right)\left(e_1 \cos\left(betazz\right) + f_1 \sin\left(betazz\right)\right)\left(-betayc_1 \sin\left(betayy\right) + betayd_1 \cos\left(betayy\right)\right) \\ \left(a_1 \cos\left(betaxx\right) + b_1 \sin\left(betaxx\right)\right)\left(c_1 \cos\left(betayy\right) + d_1 \sin\left(betayy\right)\right)\left(-betaze_1 \sin\left(betazz\right) + betazf_1 \cos\left(betazz\right)\right) \end{bmatrix}$$

Hands on

- Basic Math for AI (regression)

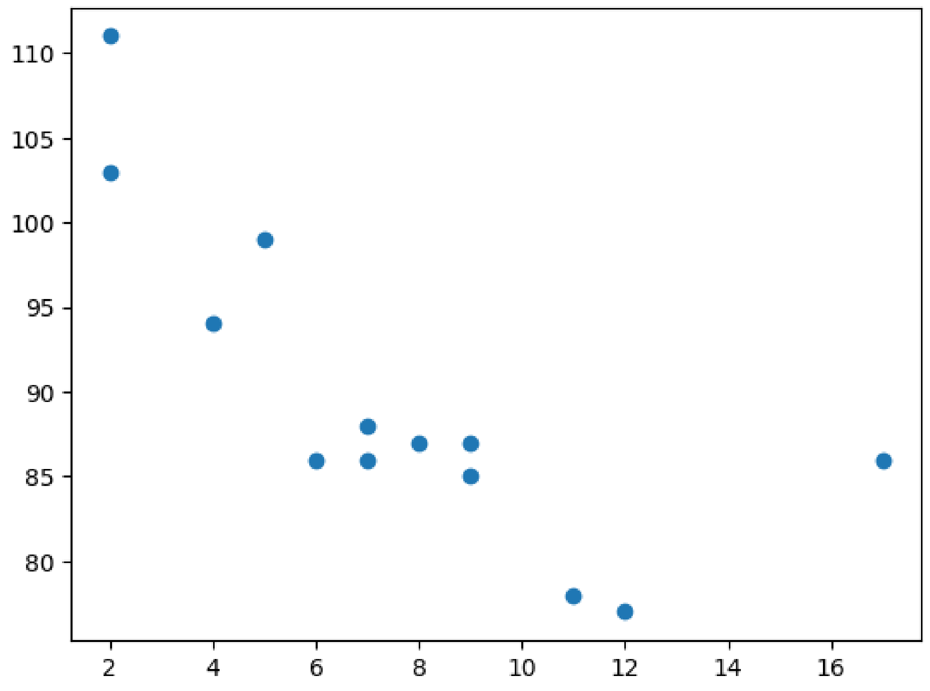
Linear regression

- 데이터 준비

```python
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```



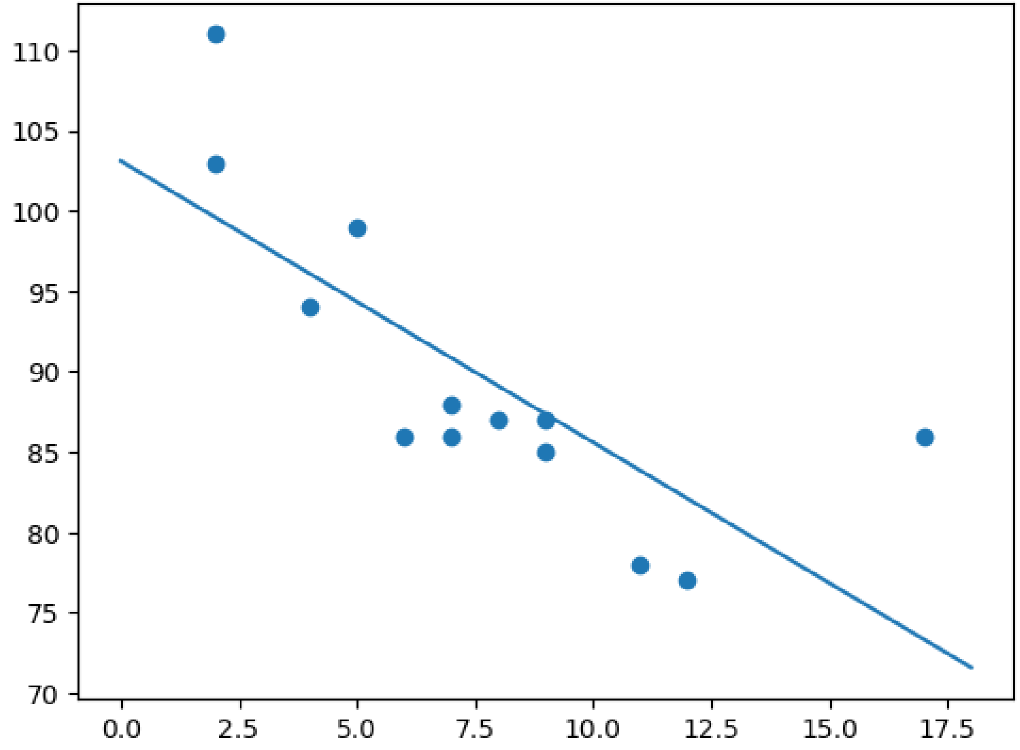
Linear regression

```
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

xx=np.linspace(0, 18, 180)
yy=slope * xx + intercept

plt.scatter(x, y)
plt.plot(xx, yy)
plt.show()
```



THANK YOU