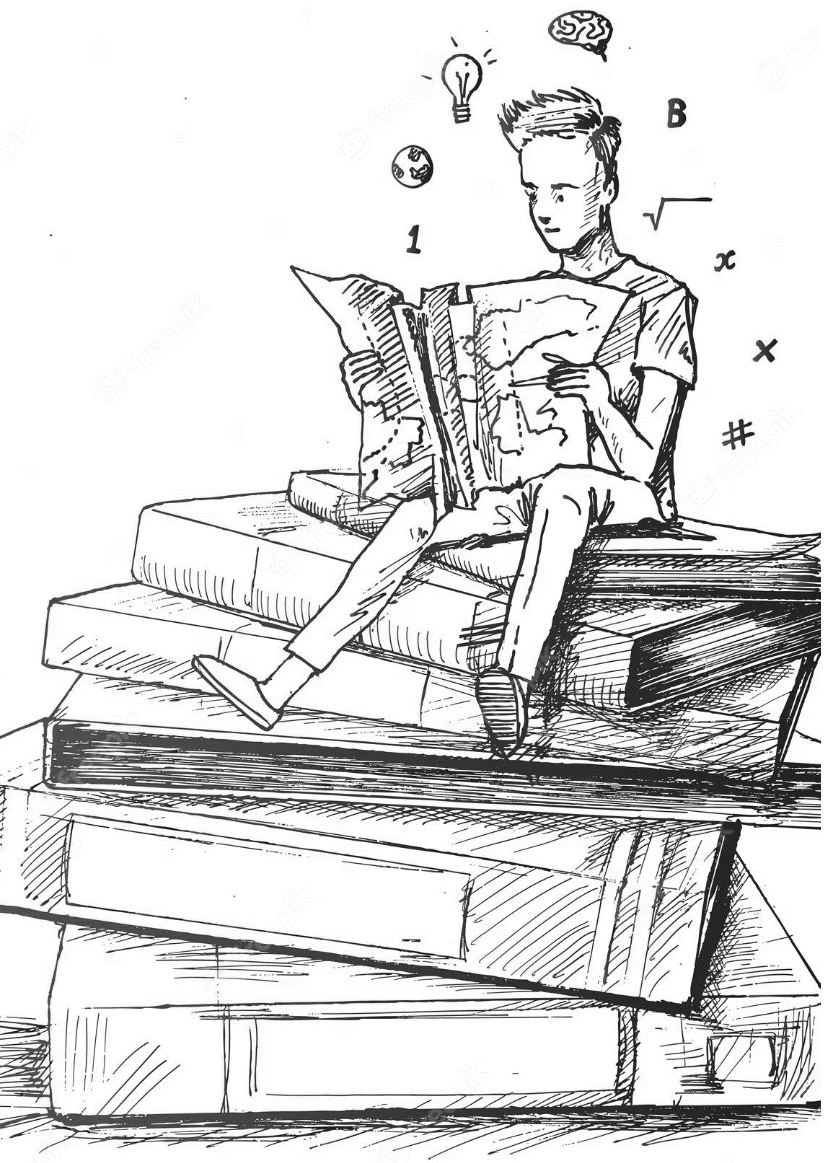
Indonesia AI
AI for Everyone, AI for Indonesia

x Ilma Aliya Fiddien

Mathematics in Deep Learning

Saturday, Januari 19th 2024



Learning Objective

Understand the essential mathematical concepts to gain **a deeper understanding** about the underlying algorithm of artificial neural networks (ANN)

Learning Objective

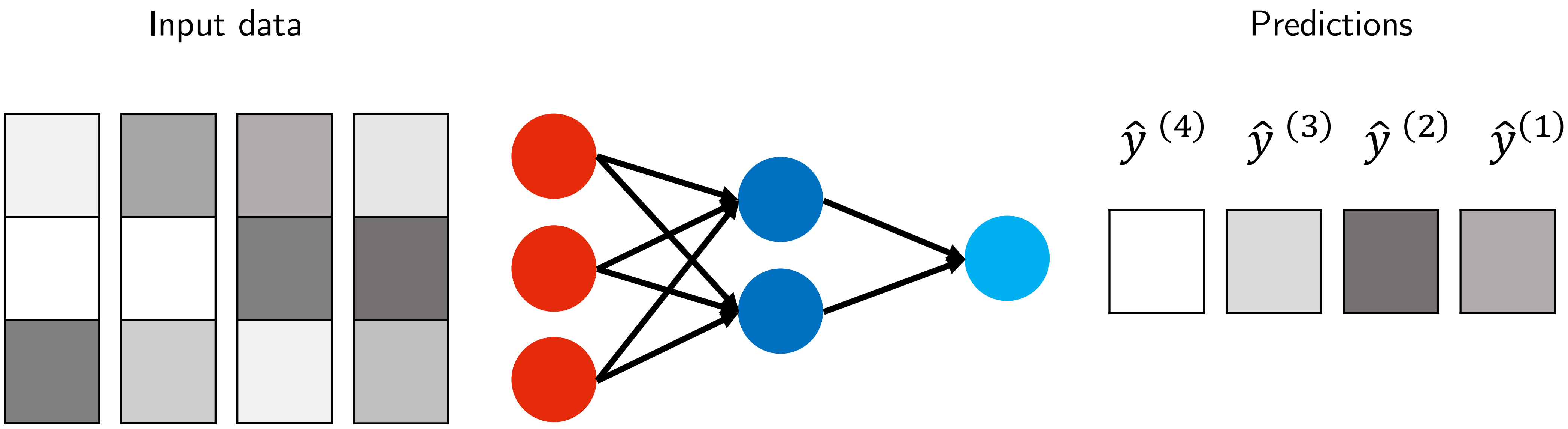
Matriks/tensor
Matrix multiplication
Function
Derivatives
Partial derivatives
Gradient
Chain rule

Forward pass
Backward pass
- gradient descent

Understand the essential mathematical concepts to gain **a deeper understanding** about the underlying algorithm of artificial neural networks (ANN)

The most basic architecture of deep learning

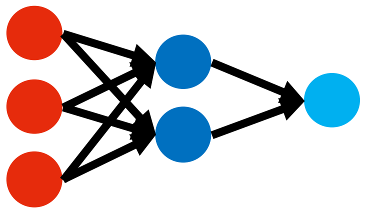
Forward Pass →



$$\sigma_1(X^{(n)} \cdot W_1 + b_1) = A_1$$

$$\sigma_2(A_1 \cdot W_2 + b_2) = A_2 = \hat{y}^{(n)}$$

Tensor Operations

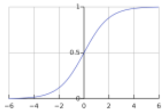


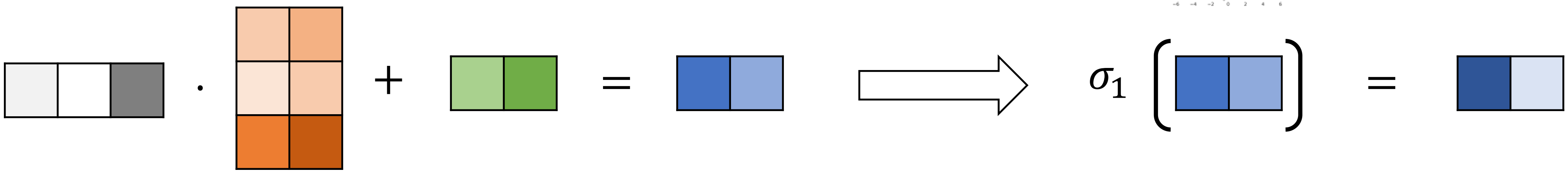
$$a_1(X^{(n)} \cdot W_1 + b_1) = A_1$$

$\dim(X^{(4)}) =$ $(1, 3)$

$\dim(W_1) =$ $(3,2)$

$\dim(b_1) =$ $(1,2)$



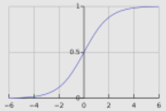


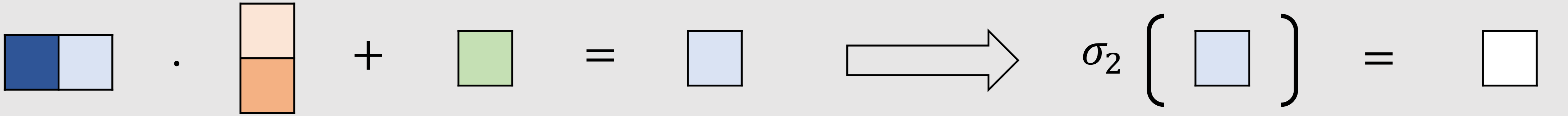
$$a_2(\ A_1 \cdot W_2 + b_2) = A_2 = \hat{y}^{(n)}$$

$\dim(A_1) =$ $(1, 2)$

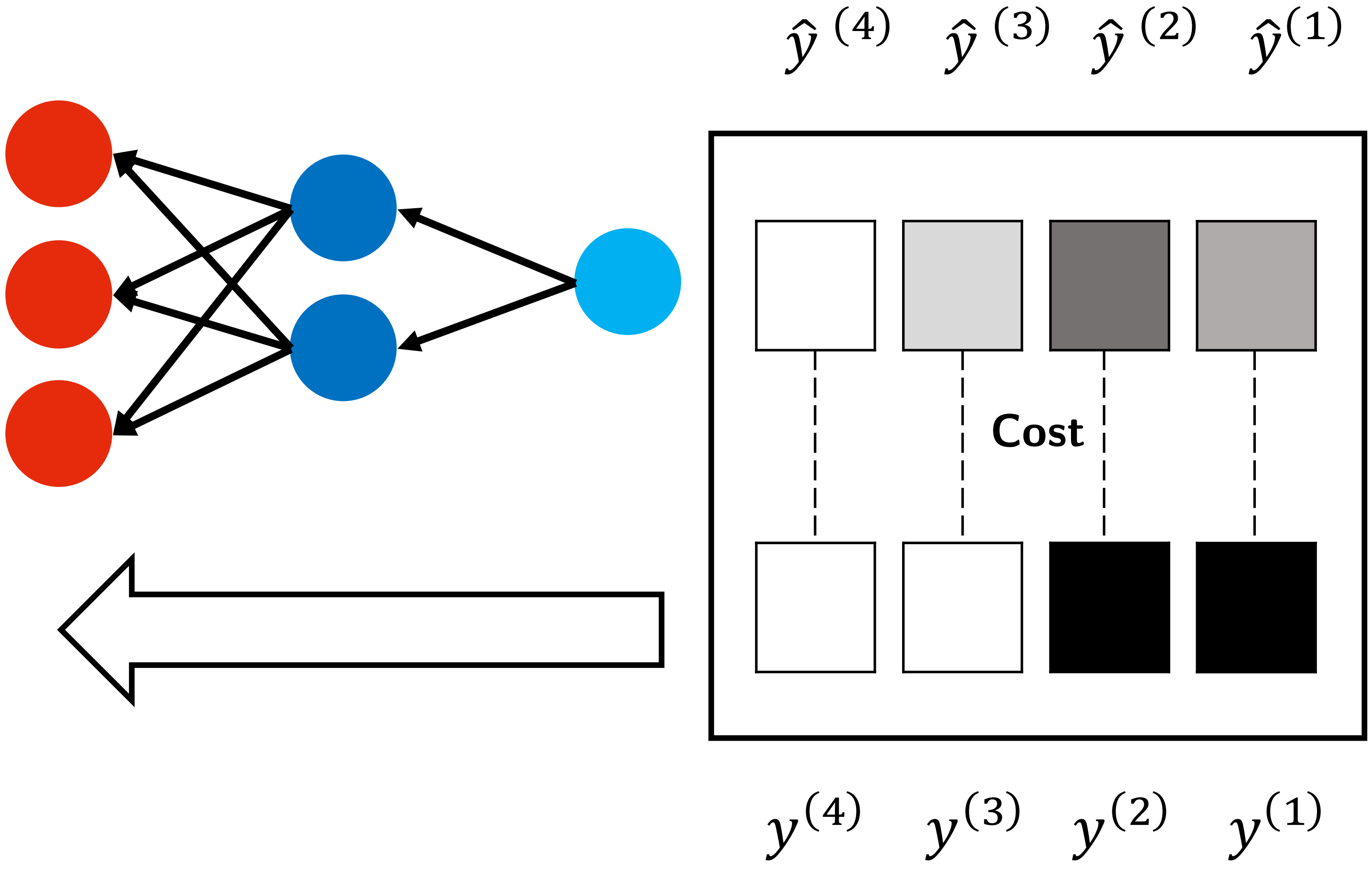
$\dim(W_2) =$ $(2,1)$

$\dim(b_2) =$ $(1,1)$

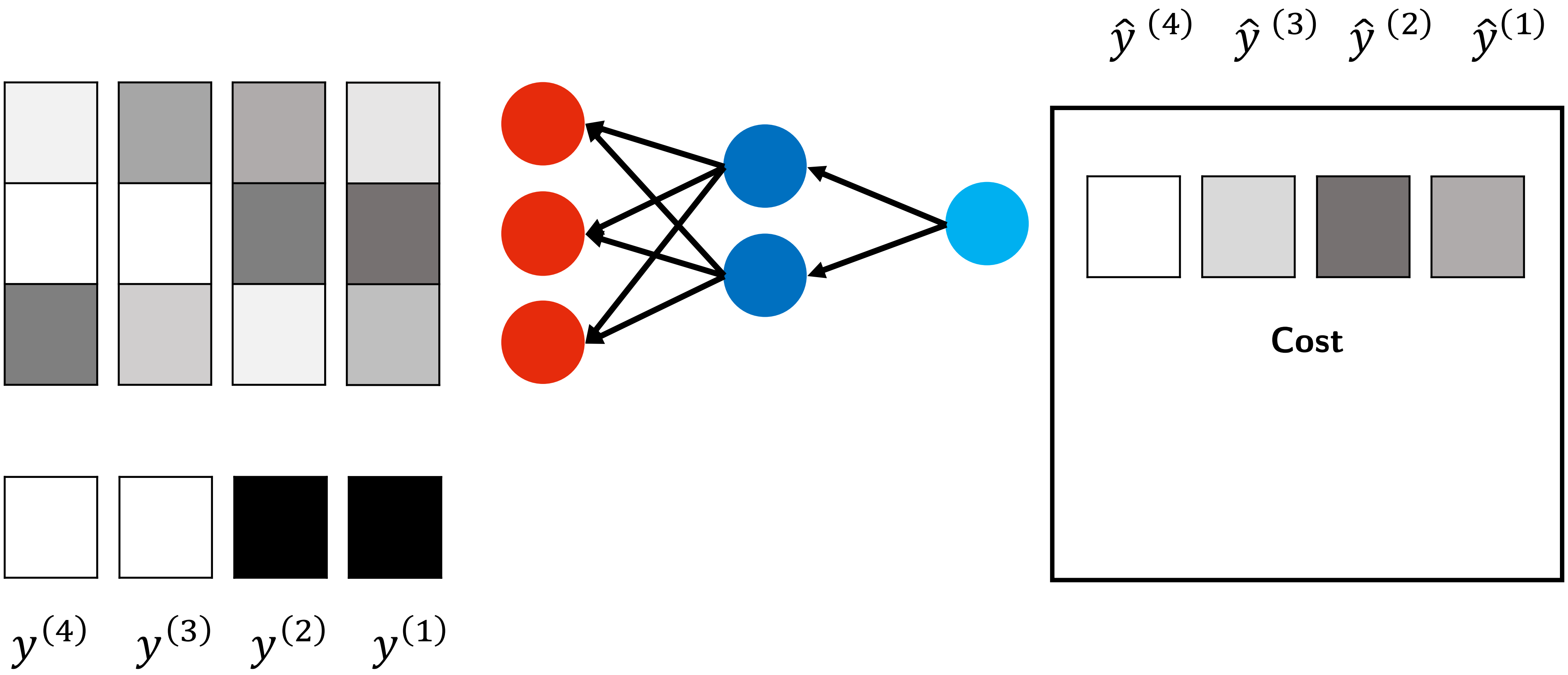




Backward Pass ←



Backward Pass ←



Gradient Descent

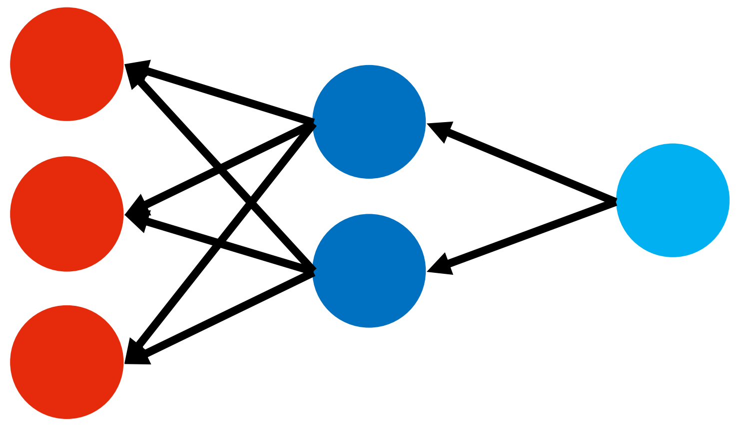
Parameter Update

$$b_2 \leftarrow b_2 - \alpha \frac{\partial}{\partial b_2} Cost(\hat{y}, y)$$

$$W_2 \leftarrow W_2 - \alpha \frac{\partial}{\partial W_2} Cost(\hat{y}, y)$$

$$b_1 \leftarrow b_1 - \alpha \frac{\partial}{\partial b_1} Cost(\hat{y}, y)$$

$$W_1 \leftarrow W_1 - \alpha \frac{\partial}{\partial W_1} Cost(\hat{y}, y)$$



x Ilma Aliya Fiddien

Mathematics in Deep Learning

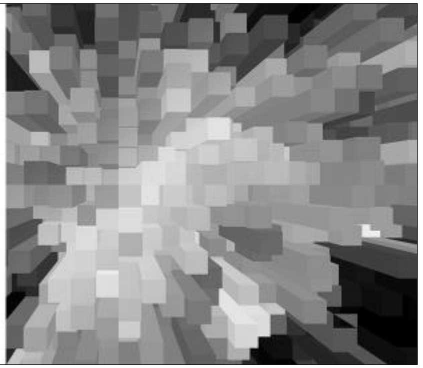
Forward Pass

in Feedforward Neural Network

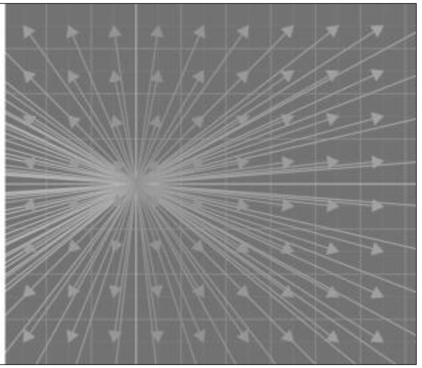
Outline

Feedforward Neural Network

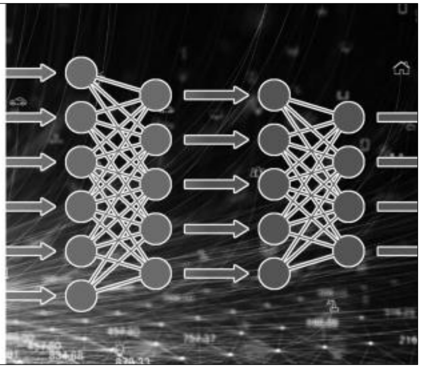
Intro



Linear Algebra



Forward Pass

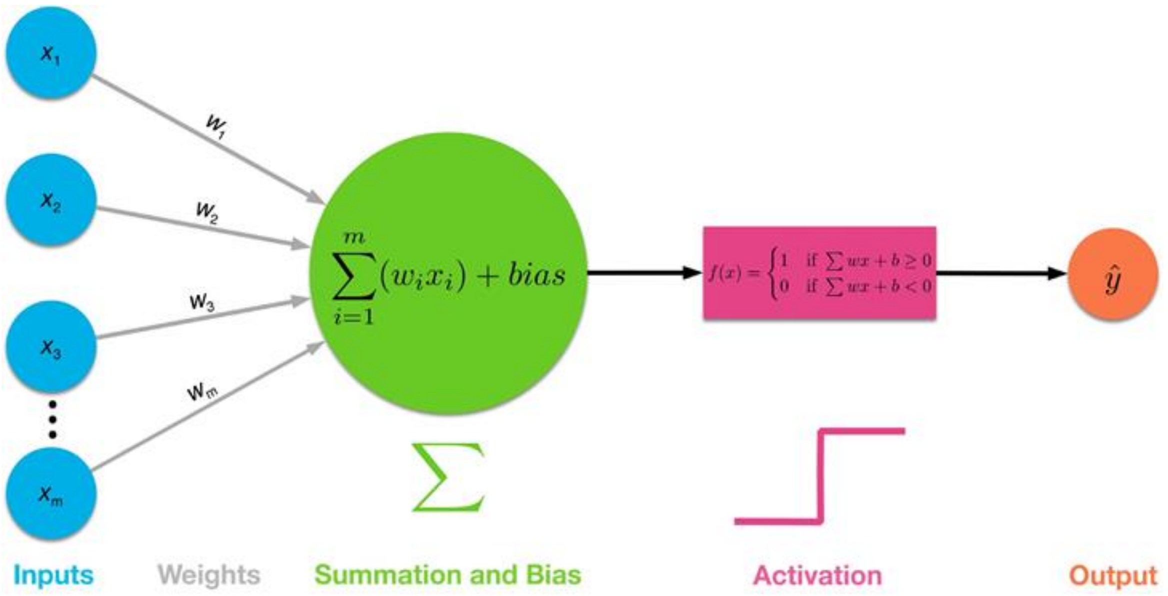


Feedforward Neural Network

Intro

Neuron & Perceptron

$$X = \begin{bmatrix} x_1 \\ x_2 \\ .. \\ x_3 \end{bmatrix}$$

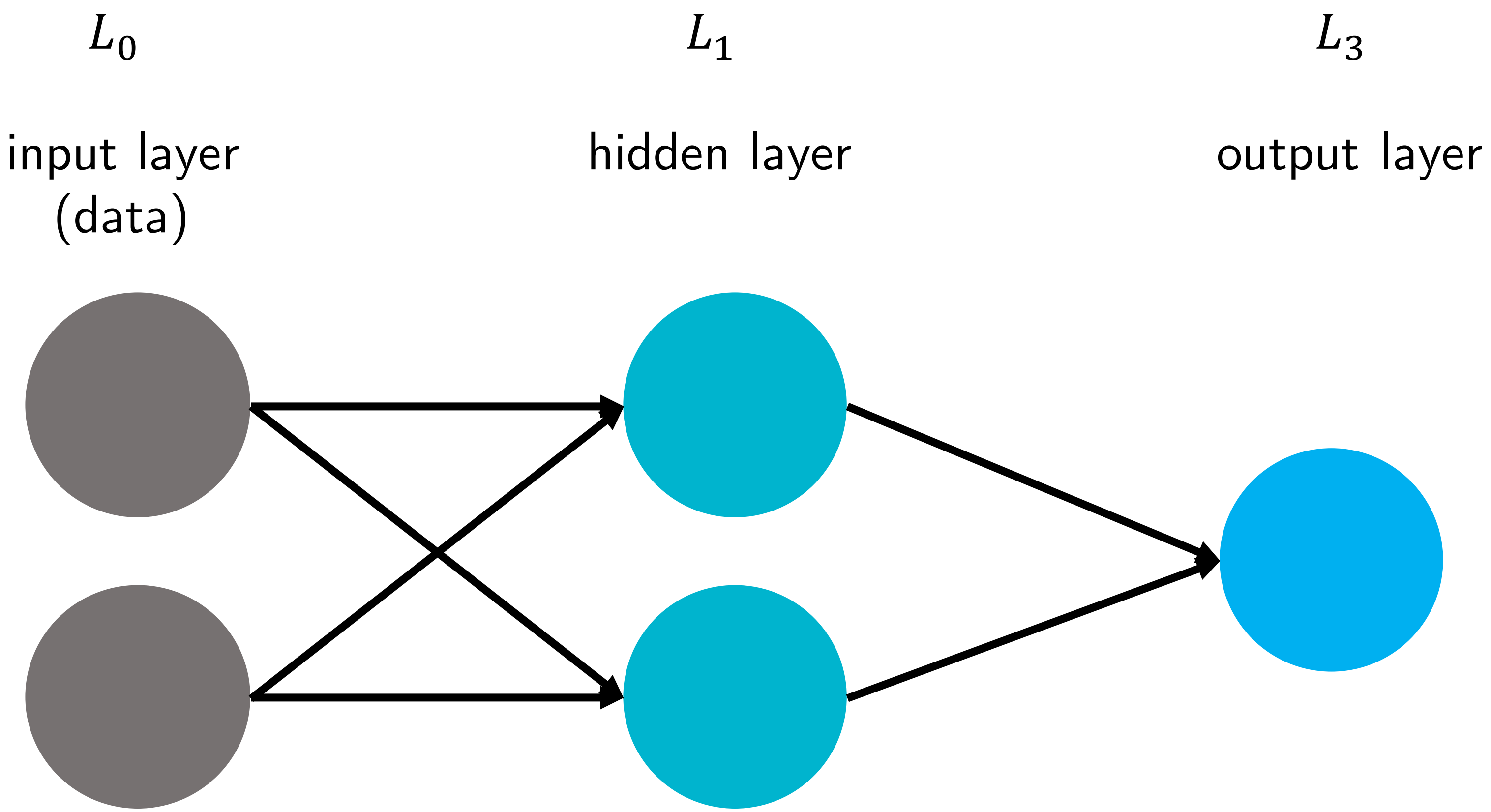


$$W = \begin{bmatrix} w_1 & .. \\ w_2 & .. \\ .. & .. \\ w_m & .. \end{bmatrix}$$

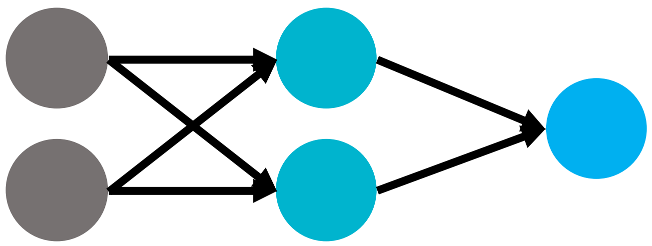
$$\hat{y} = f\left(\sum_{i=1}^{m}(w_i x_i) + bias\right)$$

$$\hat{y} = f(WX + b)$$

Multi-layer perceptron



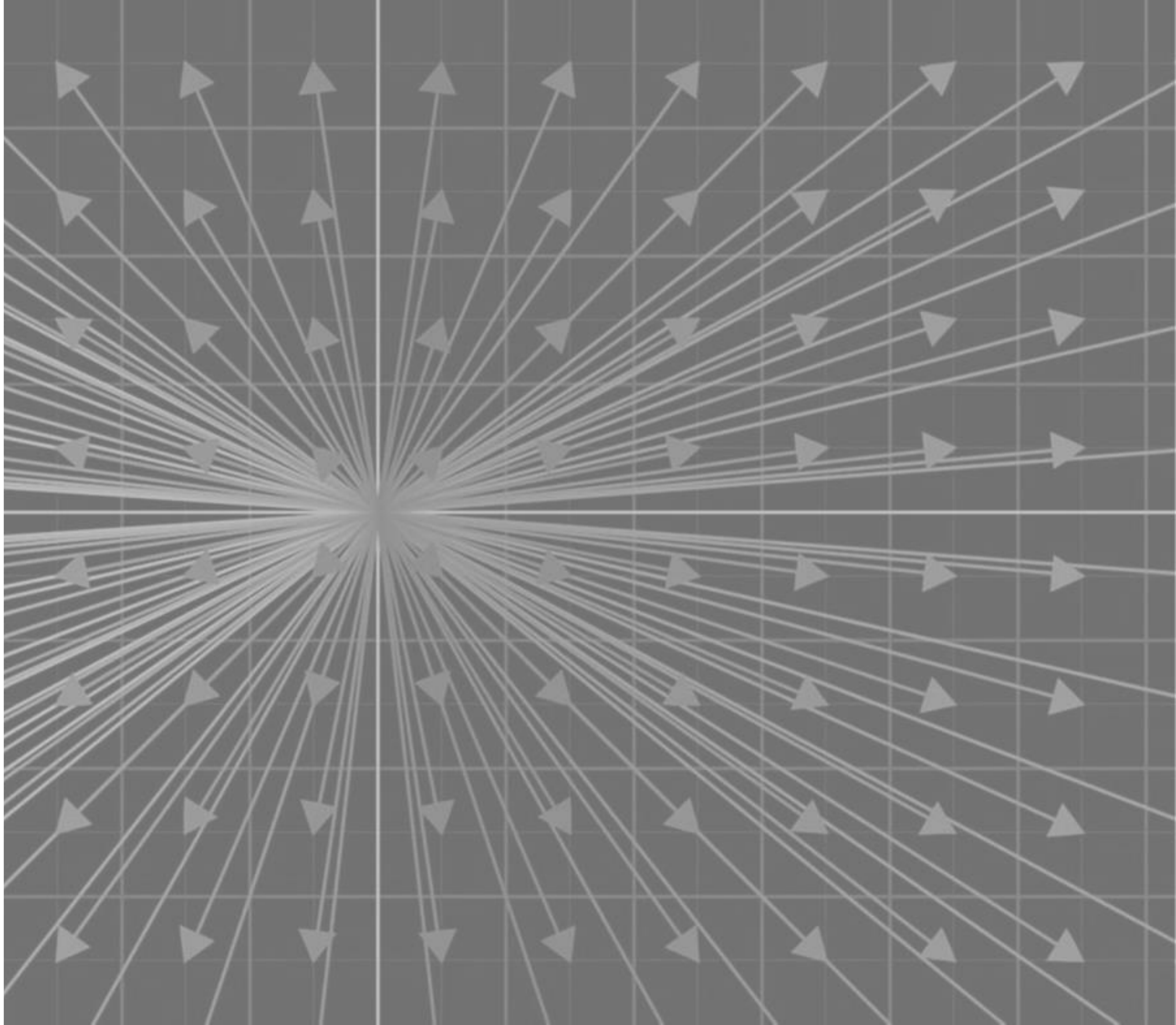
Multi-layer perceptron



Applied to the XOR problem

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

Linear Algebra



Scalars, Vectors, Matrices & Tensors

Scalars = Single Value = 0-dimensional Tensor

$$s = 66$$
$$a = 849$$

Vectors/Array = 1-Dimensional Tensors

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Matrix = 2-Dimensional Tensor

$$\boldsymbol{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

Multi-Dimensional matrix/Multi-Dimensional array/ ndarray = n-Dimensional **Tensor**

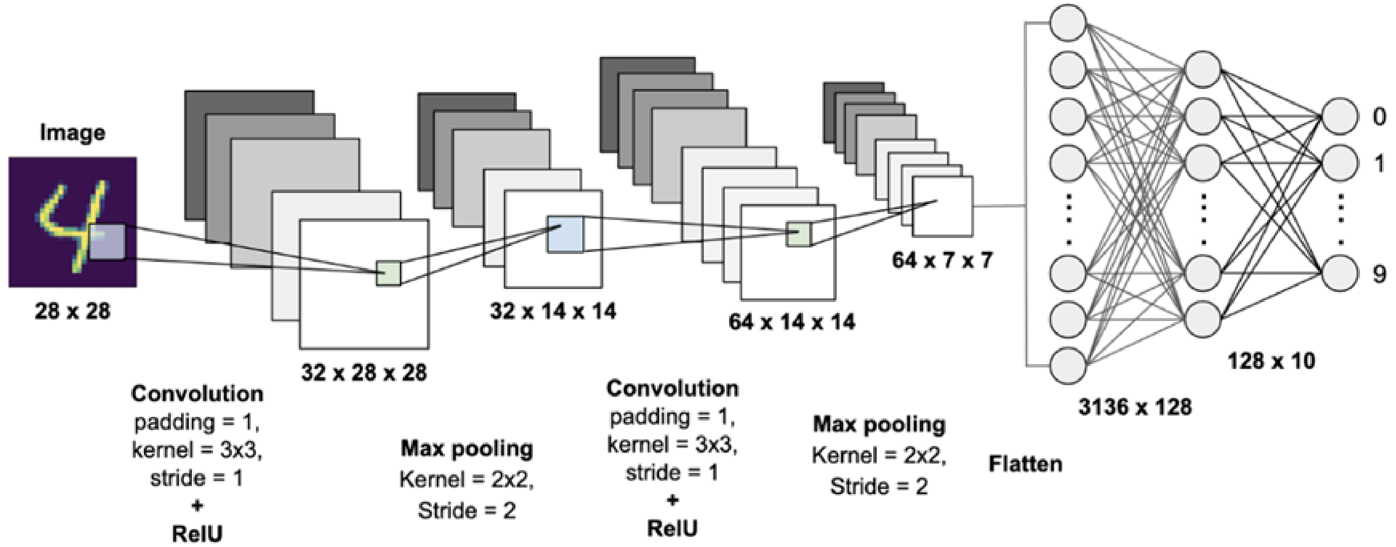
```
>>> s = 66
>>> s
66
```

```
>>> import numpy as np
>>> x = np.array([1,2,3])
>>> print(x)
[1 2 3]
>>> print(x.reshape((3,1)))
[[1]
 [2]
 [3]]
```

```
>>> A = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> print(A)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
>>> t = np.arange(36).reshape((3,3,4))
>>> print(t)
[[[ 0 1 2 3]
  [ 4 5 6 7]
  [ 8 9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]

 [[24 25 26 27]
  [28 29 30 31]
  [32 33 34 35]]]
```

Why Tensors?



- Imagine there are 28 x 28 x 32 x 28 x 28 x 32 x 14 x 14 x 64 x 14 x 14 x 64 x 7 x 7 x 3136 x 128 x 128 x 10 operations
- Tensors "wrap" the multidimentional data in a single container

Operation with Tensors

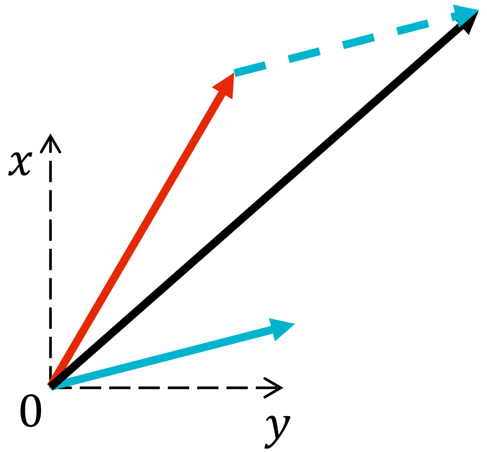
Addition and Substraction

Element-wise operation – operate on the same element position

$$\boldsymbol{A} + \boldsymbol{B} = \boldsymbol{C}$$

$$m \times n \qquad m \times n \qquad m \times n$$

dengan $A_{i,j} + B_{i,j} = C_{i,j}$



Contoh:

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

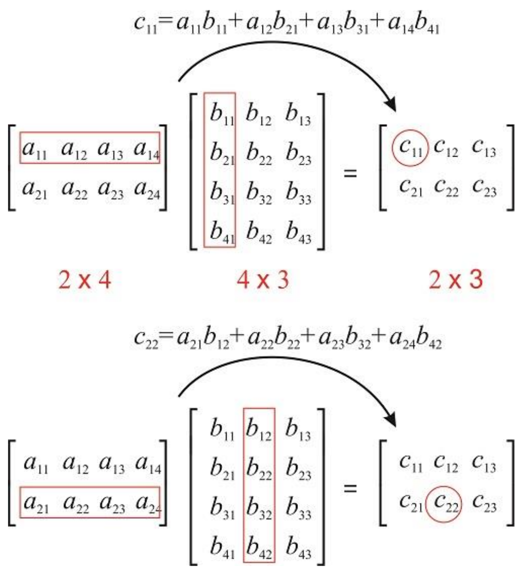
Operation with Tensors

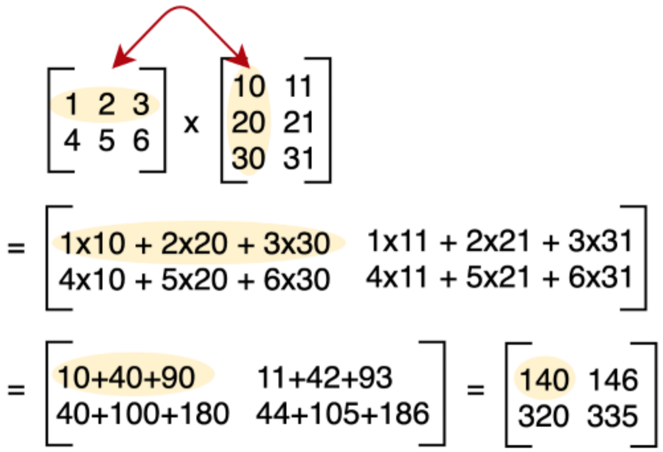
Multiplication

Matrix product A dan B akan menghasilkan C, dengan syarat, dimensi A adalah $m \times n$ dan dimensi adalah $n \times p$ yang menghasilkan C dengan dimensi $m \times p$

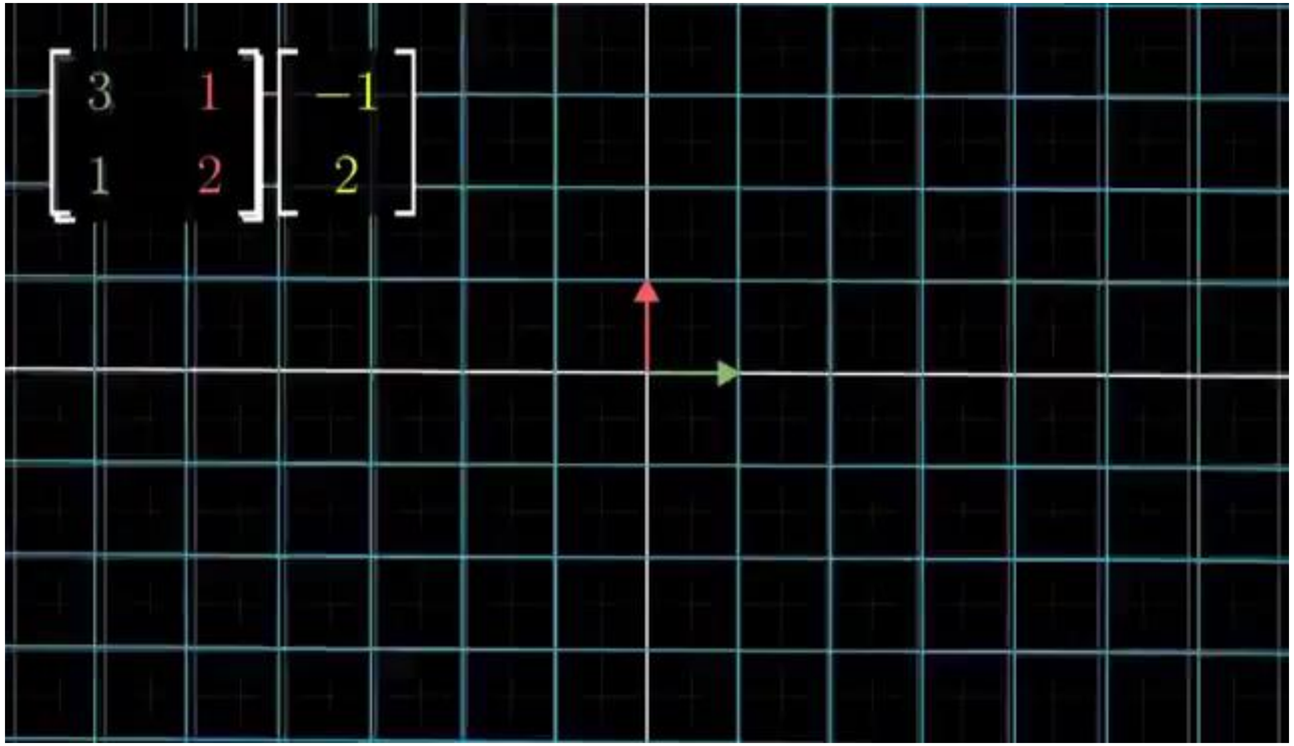
$$\underset{m \times n}{\boldsymbol{A}}\ \underset{n \times p}{\boldsymbol{B}} = \underset{m \times p}{\boldsymbol{C}}$$

$$\sum_k A_{i,k} B_{k,j} = C_{i,j}$$





Operation with Tensors



Sumber animasi: 3Brown1Blue, *Linear transformations and matrices | Essence of linear algebra, chapter 3*

Operation with Tensors

Transpose

Transpose adalah hasil cermin Tensor terhadap suatu garis main diagonal. Transpose dari $\boldsymbol{A}$ adalah $\boldsymbol{A}^T$, dengan

$$(\boldsymbol{A}^\top)_{i,j} = A_{j,i}.$$

$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Identity ($\boldsymbol{I_n}$) and Inverse ($\boldsymbol{A^{-1}}$)

$$\boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x} \qquad \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n$$

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

$$\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

$$\boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

Operation with Tensors

Properties

- Transpose dari skalar adalah skalar itu sendiri

$$a = a^\top$$

- Scalar bisa dikalikan dan atau ditambahkan pada matriks

$$\boldsymbol{D} = a \cdot \boldsymbol{B} + c$$

dengan

$$D_{i,j} = a \cdot B_{i,j} + c.$$

- Sifat perkalian matriks (*matrix product*)

A. Distributif

$$\boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C}$$

B. Asosiatif

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}$$

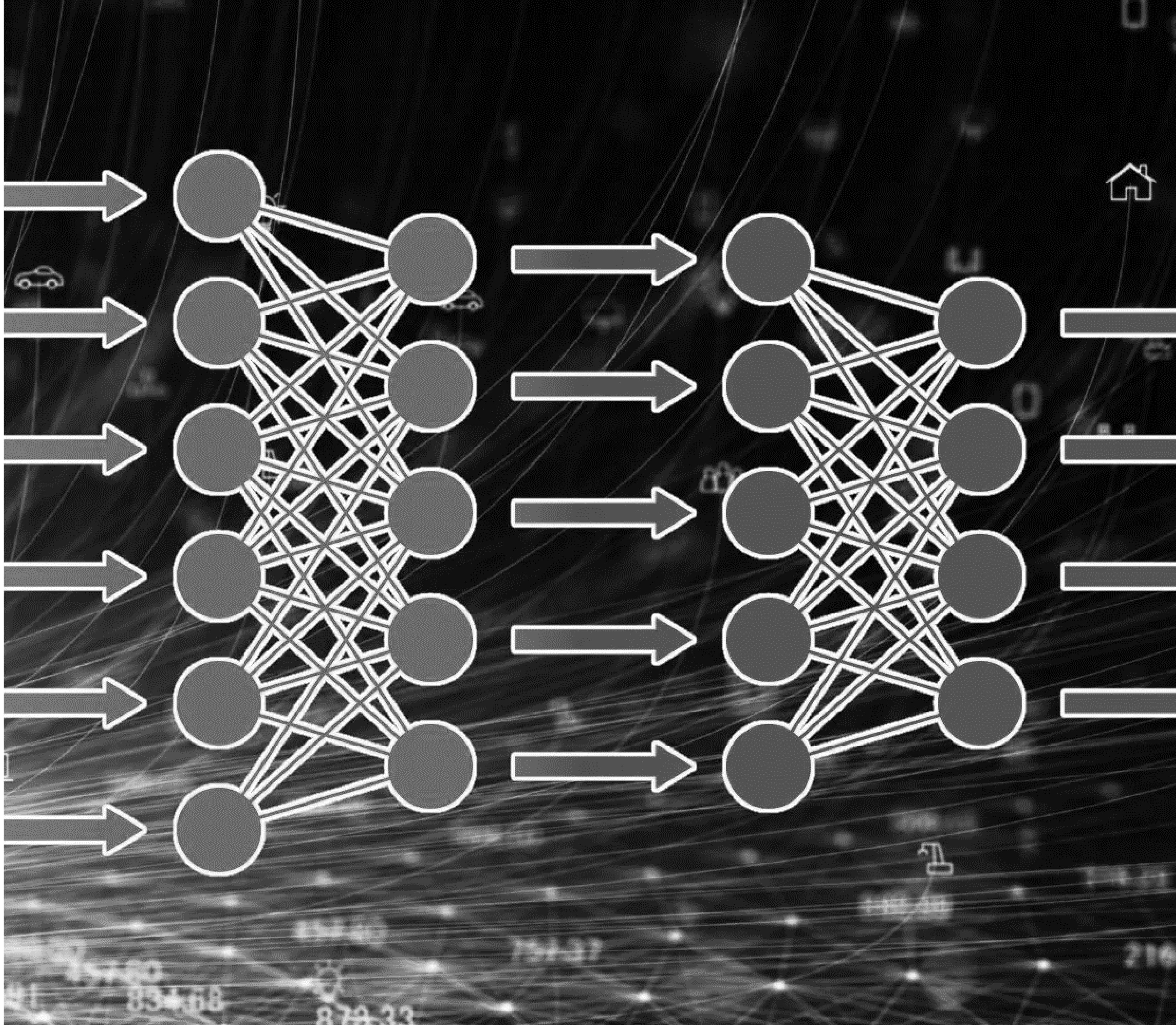
C. Tidak komutatif

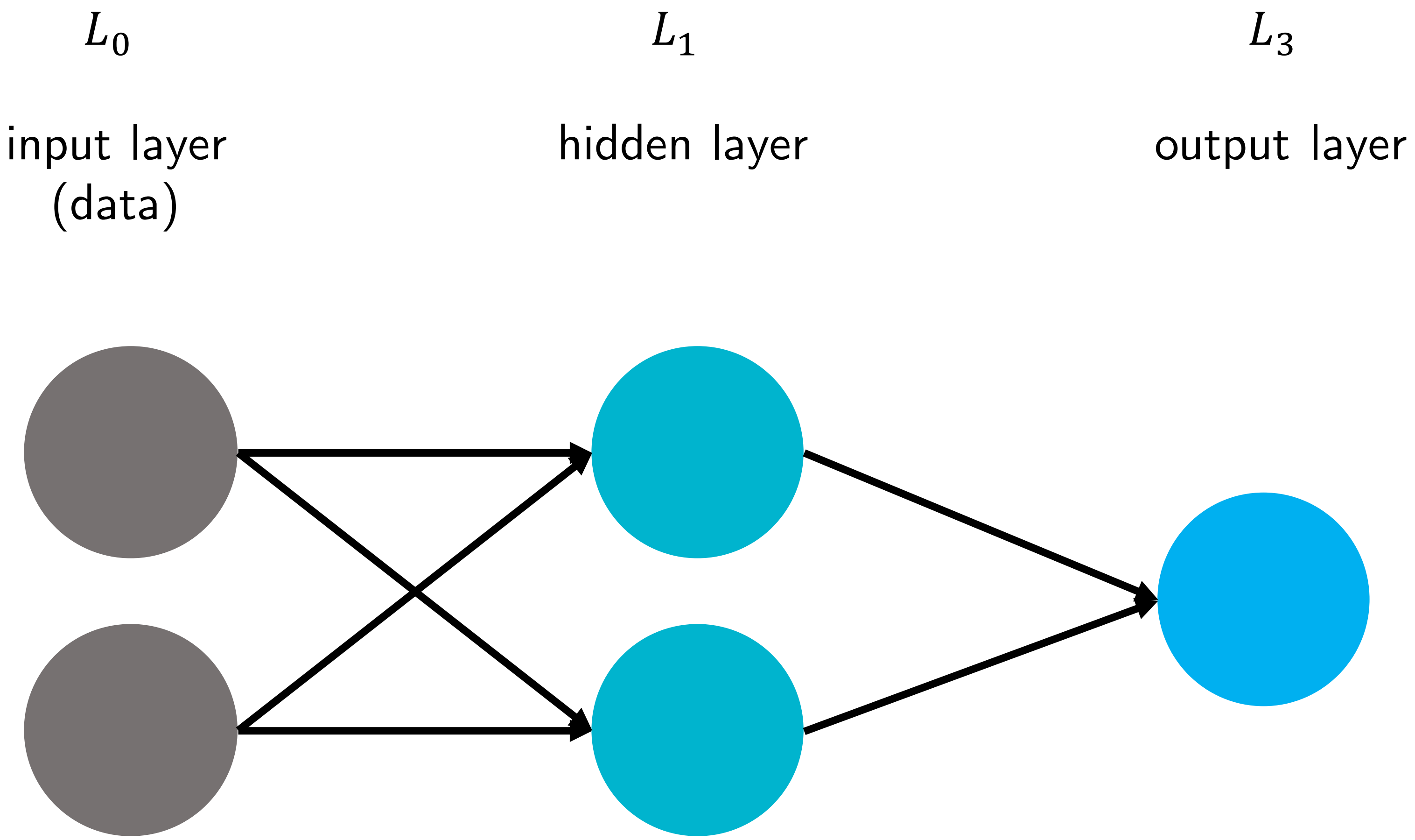
$$(\boldsymbol{AB})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top$$

D. Bentuk transpose dari *matrix product*

$$\boldsymbol{x}^\top \boldsymbol{y} = \left(\boldsymbol{x}^\top \boldsymbol{y}\right)^\top = \boldsymbol{y}^\top \boldsymbol{x}$$

Forward Pass



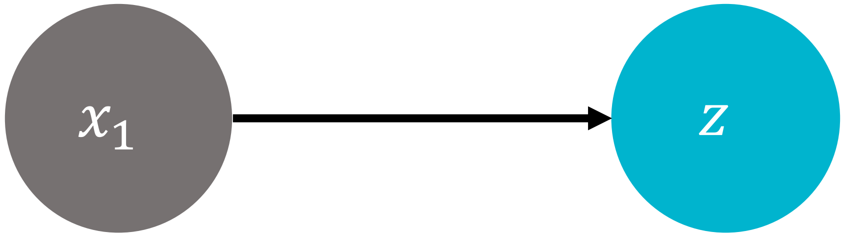

L_0
input layer
(data)
L_1
hidden layer
L_3
output layer

L_0

input layer

L_1

hidden layer

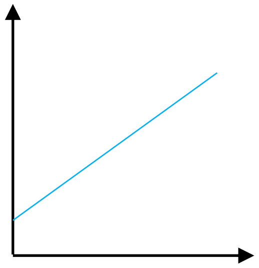


parameter model

weight

bias

$$= x_1 w + b$$
$$= (1)2 + 3$$
$$= 5$$



$x_1 = 1$

$w = 2$

$b = 3$

```
import numpy as np

x = np.array([1])
# parameter Layer-1
w = 2
b = 3
# hidden Layer-1
z = x*w+b
z
```

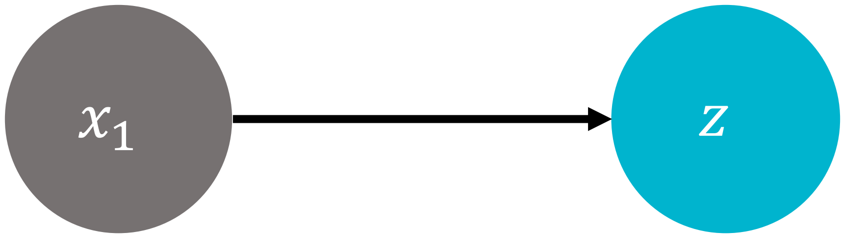
```
array([5])
```

L_0

input layer

L_1

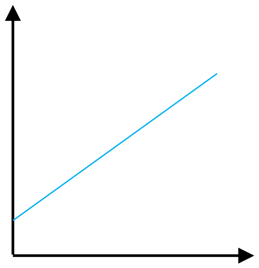
hidden layer



parameter model

weight bias

$$= x_1 w + b$$
$$= (1)2 + 3$$
$$= 5$$



$$x_1 = 1$$

$$w = 2$$
$$b = 3$$

```python
import numpy as np

x = np.array([1])
# parameter Layer-1
w = 2
b = 3
# hidden Layer-1
z = x*w+b
z
```

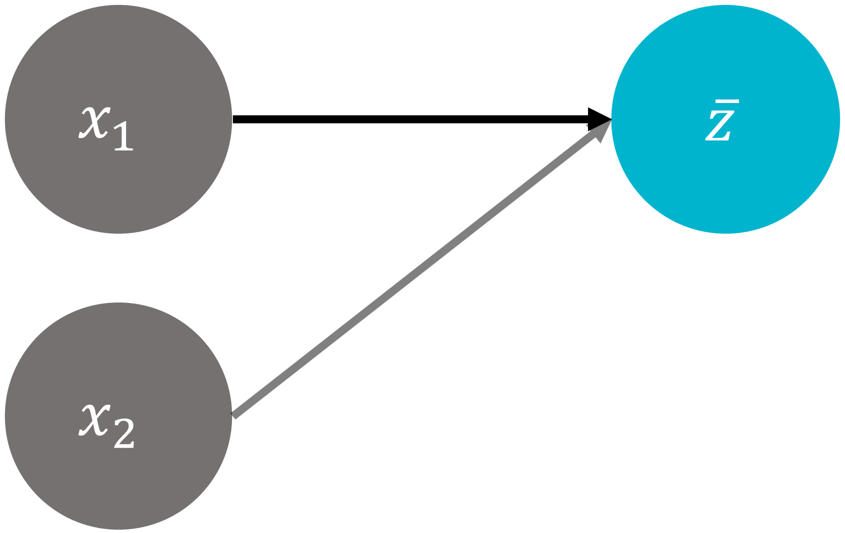
```
array([5])
```

L_0

input layer

L_1

hidden layer



$$= \bar{x} \cdot \overline{w} + b = x_1 w_1 + x_2 w_2 + b$$

$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 3 = 5$$

$$\bar{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$w = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

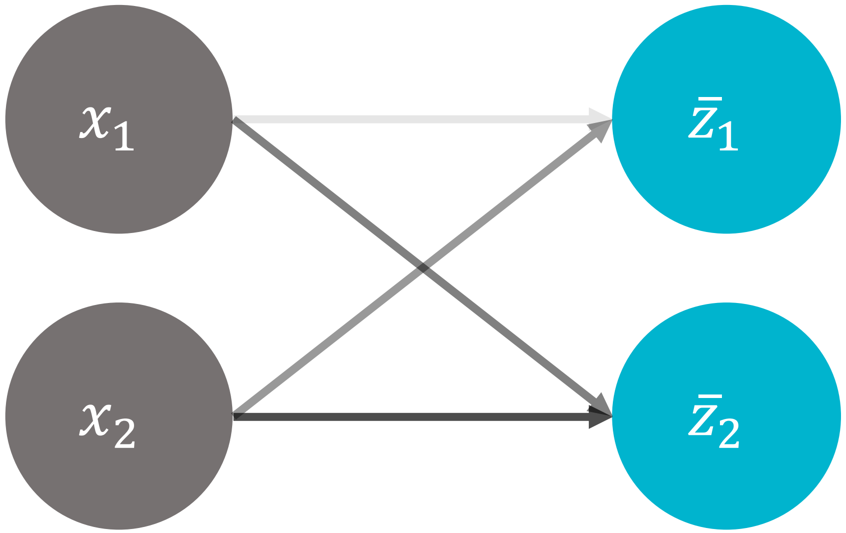
$$b = 3$$

```
import numpy as np

x = np.array([[1, 0]])
# parameter Layer-1
w1 = np.array([2, 1])
b1 = 3
# hidden Layer-1
z1 = x@w1 + b1
z1
```

```
array([5])
```

L_0 input layer

L_1 hidden layer



$$= \bar{x} \cdot \bar{w}_1 + b_1$$

$$= \bar{x} \cdot \bar{w}_2 + b_2$$

$$\bar{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \qquad w_2 = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

$$b_1 = 3 \qquad b_2 = 5$$

```
import numpy as np

x = np.array([[1, 0]])
# parameter Layer-1
w1 = np.array([2, 1])
b1 = 3
w2 = np.array([4, 3])
b2 = 5
# hidden Layer-1
z1 = x@w1 + b1
z2 = x@w2 + b2
z1, z2
```

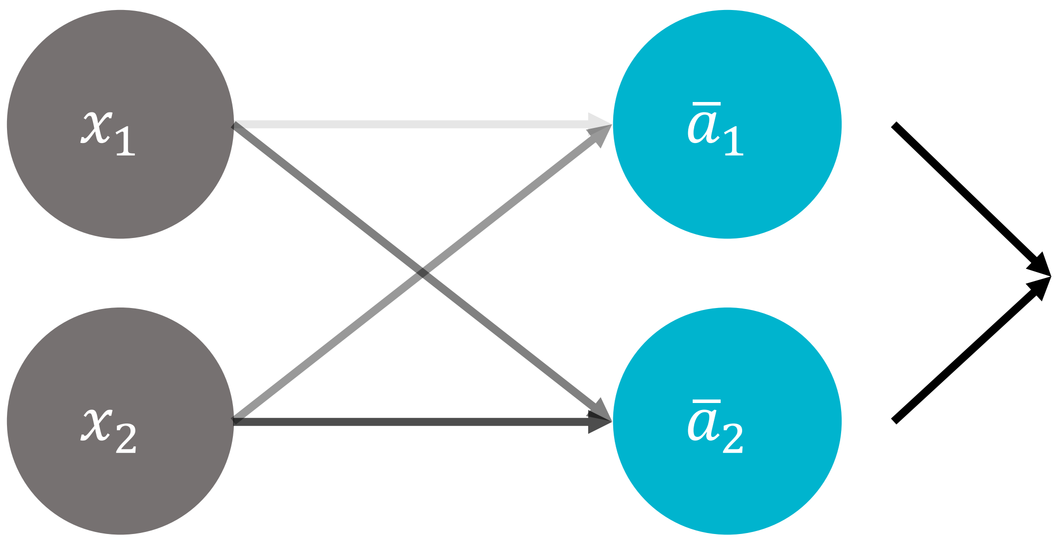
```
(array([5]), array([9]))
```

L_0 input layer

L_1 hidden layer

Aktivasi!

(tambahkan nonlinearitas!)



$$A_1 = \sigma(\bar{x} \cdot W_1 + b_1)$$

$$\bar{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} 3 & 5 \end{bmatrix}$$

```
import numpy as np

# activation function: sigmoid
a = lambda z: 1/(1+np.exp(-z))

x = np.array([[1, 0]])
# parameter di Layer-1
W1 = np.array([[2, 4],
               [1, 3]])
b1 = np.array([3, 5])
# hidden Layer-1
Z1 = x@W1.T + b1
A1 = a(Z1)
A1
```

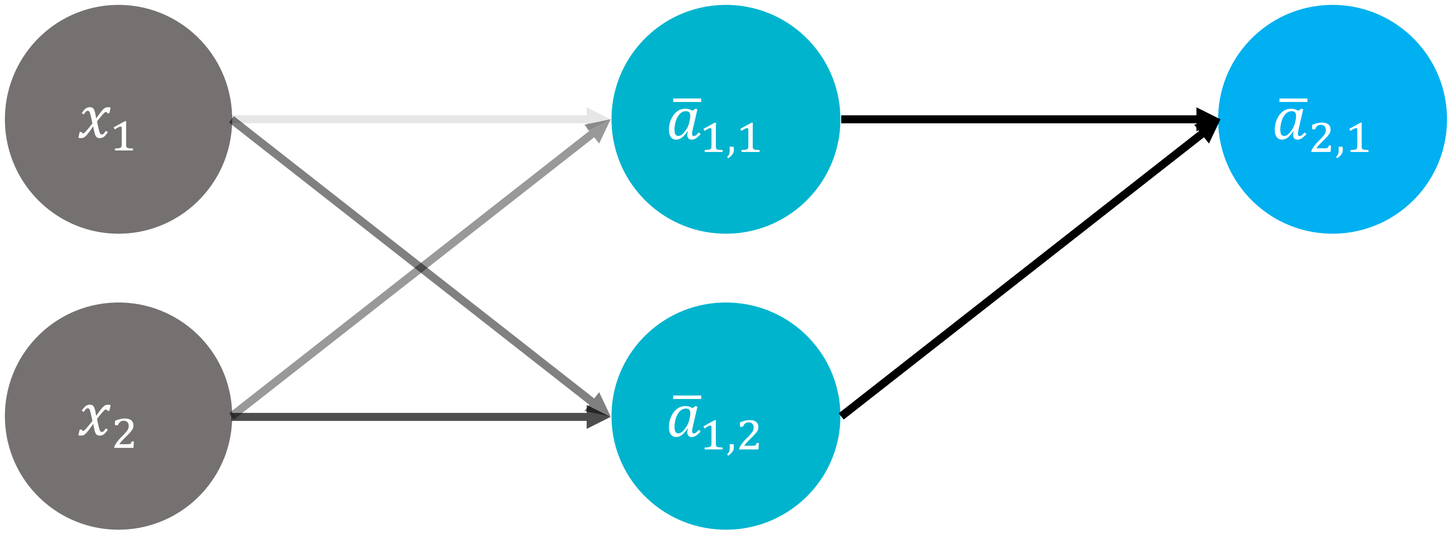
```
array([[0.99330715, 0.99752738]])
```

L_0

input layer

L_1

hidden layer



$$= \sigma(A_1 \cdot W_2 + b_2) = A_2$$

$$\bar{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} 3 & 5 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 6 \\ 7 \end{bmatrix}$$

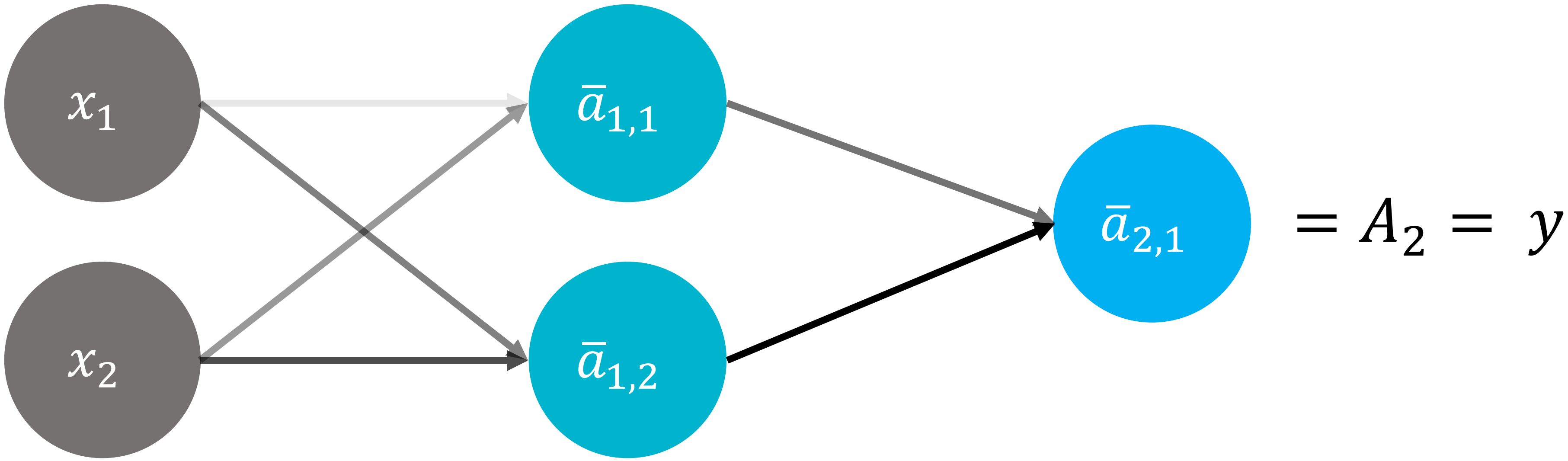
$$b_2 = 0$$

```
import numpy as np

# activation function: sigmoid
a = lambda z: 1/(1+np.exp(-z))

x = np.array([[1, 0]])
# parameter di Layer-1
W1 = np.array([[2, 4],
               [1, 3]])
b1 = np.array([3, 5])
# parameter di Layer-2
W2 = np.array([6, 7])
b2 = 0
# hidden Layer-1
Z1 = x@W1.T + b1
A1 = a(Z1)
# output layer
Z2 = A1@W2.T + b2
A2 = a(Z2)
A2

array([0.99999761])
```

L_0 input layer

L_1 hidden layer

L_3 output layer



$\bar{x} = \begin{bmatrix} 1 & 0 \end{bmatrix}$

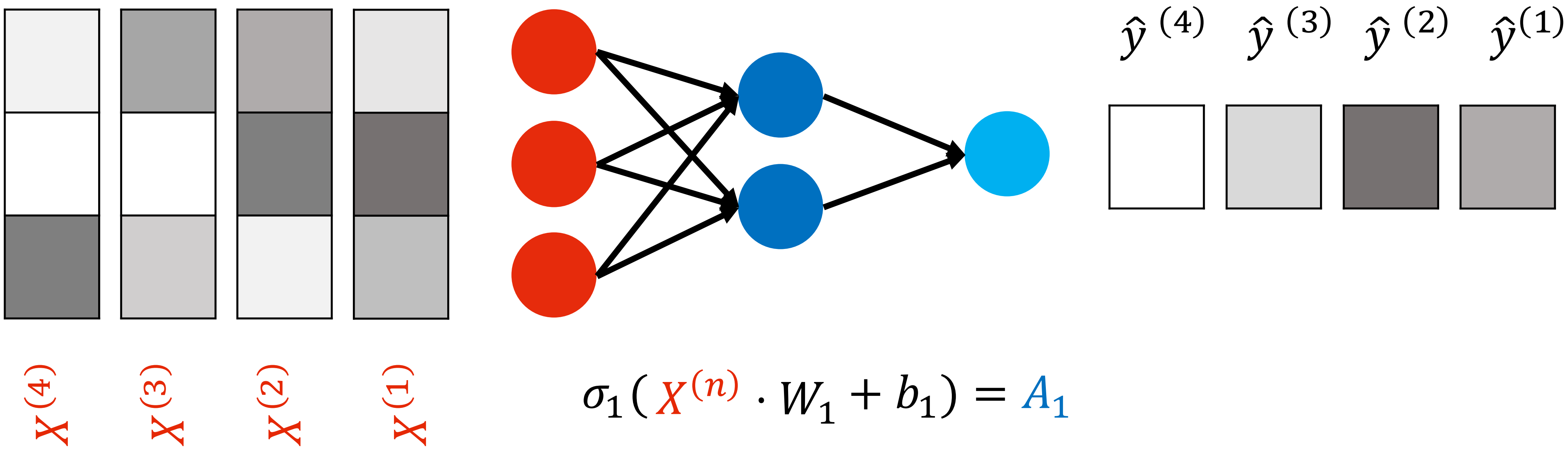
$W_1 = \begin{bmatrix} 2 & 4 \\ 1 & 3 \end{bmatrix}$

$b_1 = \begin{bmatrix} 3 & 5 \end{bmatrix}$

$W_2 = \begin{bmatrix} 6 \\ 7 \end{bmatrix}$

$b_2 = 0$

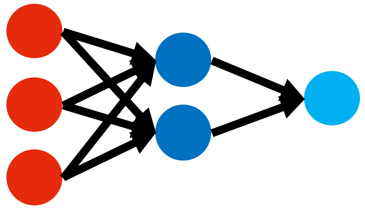
Forward Pass →



$$\sigma_1(X^{(n)} \cdot W_1 + b_1) = A_1$$

$$\sigma_2(A_1 \cdot W_2 + b_2) = A_2 = \hat{y}^{(n)}$$

Tensor Operations

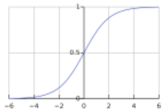


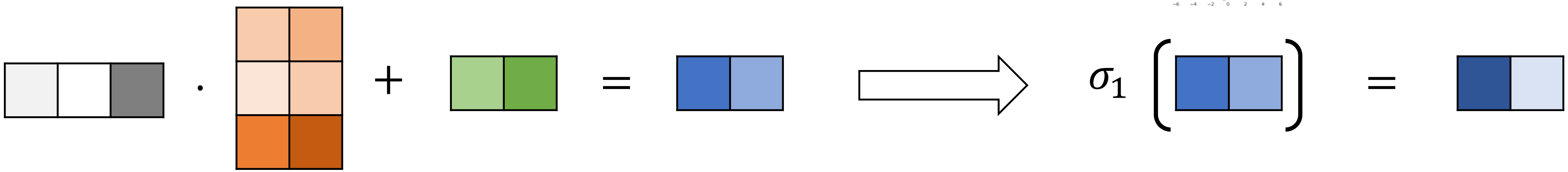
$$a_1(X^{(n)} \cdot W_1 + b_1) = A_1$$

$\dim(X^{(4)}) =$
$(1, 3)$

$\dim(W_1) =$
$(3,2)$

$\dim(b_1) =$
$(1,2)$



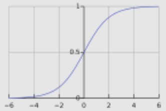


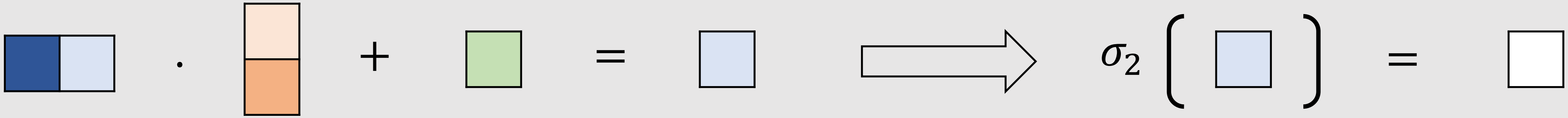
$$a_2(A_1 \cdot W_2 + b_2) = A_2 = \hat{y}^{(n)}$$

$\dim(A_1) =$
$(1, 2)$

$\dim(W_2) =$
$(2,1)$

$\dim(b_2) =$
$(1,1)$



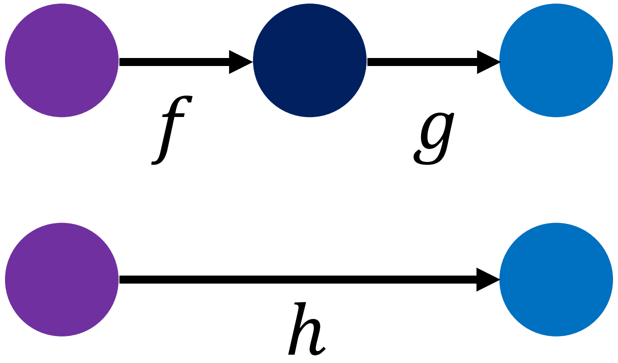


Kenapa butuh *activation function*?

Tumpukan persamaan linear adalah persamaan linear

$$f(x) = \mathrm{a}x + \mathrm{b}$$
$$g(x) = \mathrm{c}x + \mathrm{d}$$
$$\begin{aligned} f\big(g(x)\big) &= \mathrm{c}g(x) + \mathrm{d} \\ &= \mathrm{c}(\mathrm{a}x + \mathrm{b}) + \mathrm{d} \\ &= \mathrm{ac}x + \mathrm{cb} + \mathrm{d} \\ &= \mathrm{p}x + \mathrm{q} = h(x) \leftarrow \text{persamaan linear!} \end{aligned}$$

dengan $\mathrm{p} = \mathrm{ac}, \mathrm{q} = \mathrm{cb} + \mathrm{d}$.

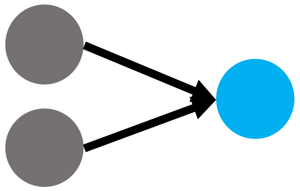


Kenapa butuh *activation function*?

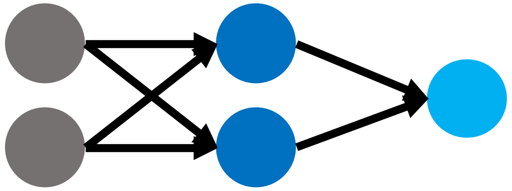
Contoh: Masalah XOR

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

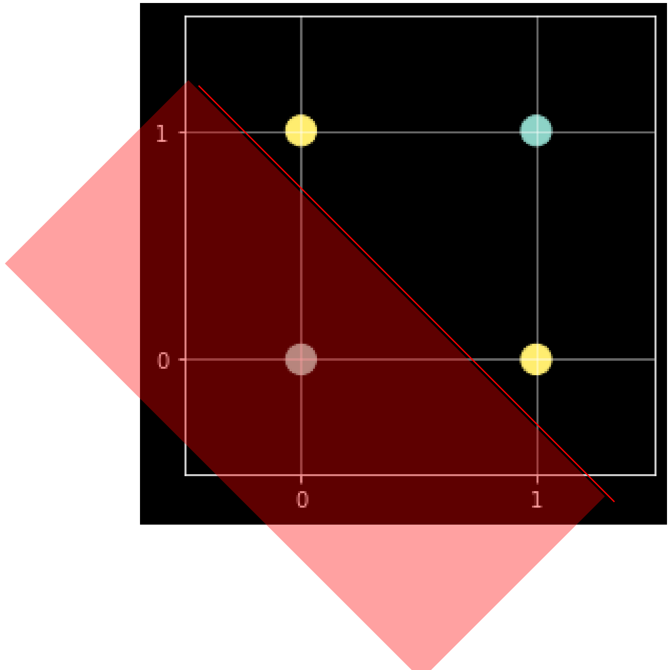
Kita butuh memasukkan sifat
“nonlinearitas” ke dalam model.

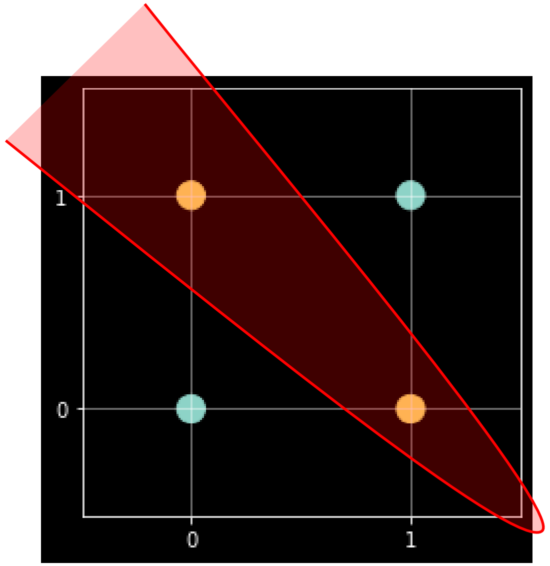


Model linear



Model nonlinear





Next...

- Differential Calculus
- Gradient Descent
- Backward Pass

Futher learning...

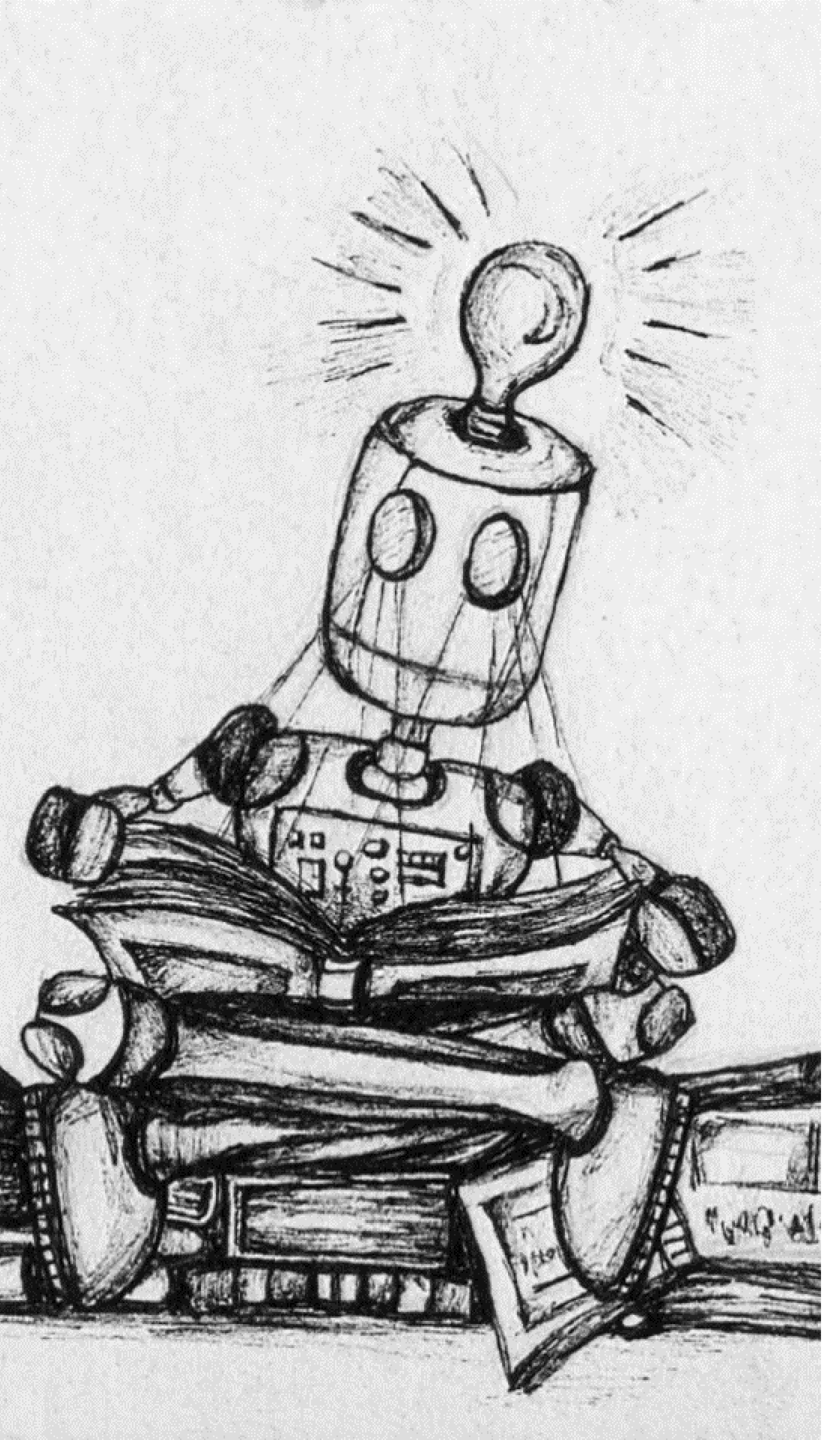
- **Deep Learning Book (Goodfellow et. al., 2016)**

  https://www.deeplearningbook.org/

- **Dive into Deep Learning:**

  Appendix: Mathematics for Deep Learning

  https://www.d2l.ai/chapter_appendix-mathematics-for-deep-learning/index.html



Thank you!