Reading Circle #1
Chapter2 Perceptron

2018.05.14

M1 Kouki OKADA

Index

- What's perceptron?
- Implementation of perceptron (AND/NAND/OR)
- Limitation of perceptron (XOR)
- Summary

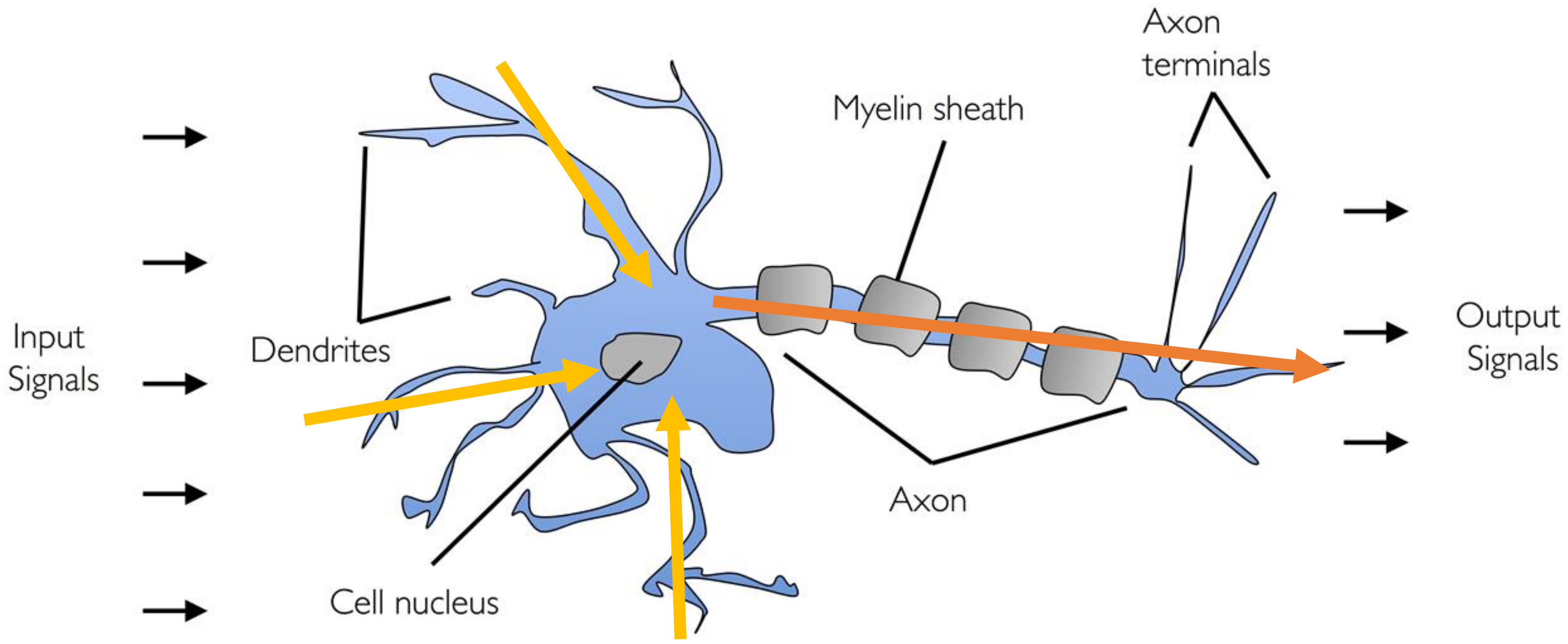
What’s perceptron?

Early machine learning

- In 1943, Warren McCulloch and Walter Pitts presented formal neuron (形式ニューロン).
  This algorithm is the artificial neuron for the first time and based on brain neuron.
  cf.) *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 5(4):115-133, 1943

- Frank Rosenblatt invented perceptron based on formal neuron in 1957.
  cf.) *A Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review 65(6): 386-408, 1958.

Neuron

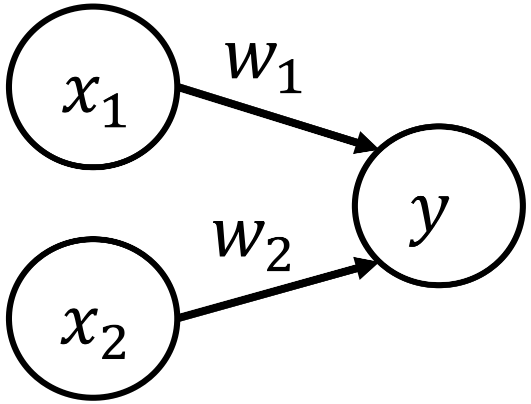


① Some current signals conducting from dendrites (樹状突起) and combine at cell body (細胞体).

② If the combined signal's voltage suppress threshold value, current signal conducts to the axon.

③ The signal conducts from axon terminal to another neuron.

Perceptron (two inputs)



$$y = \begin{cases} 0 \ (w_1x_1 + w_2x_2 \leq \theta) \\ 1 \ (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

- Multiple input, one output signal (0 or 1)
- x : inputs / y : output / w : weights
- Each “○” is called “neuron” or “node”
- Θ : threshold value

- Algorithm

  ① Calculate $w_1x_1 + w_2x_2$ in the neuron

  ② Compare the summation with θ

  If $w_1x_1 + w_2x_2 > \theta$ $then$ $y = 1$

  → “the neuron fires”

  $otherwise$ $y = 0$

Implementation of perceptron

Logic circuit

- AND gate

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

- OR gate

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

- NAND gate

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

- One of parameters satisfy these table
  - AND: $(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$
  - NAND: $(w_1, w_2, \theta) = (-0.5, -0.5, -0.7)$
  - OR: $(w_1, w_2, \theta) = (0.5, 0.5, 0.2)$

Simple implementation(AND)

```
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    tmp = w1*x1 + w2*x2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1
```

- $(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$
- Calculate $w_1 x_1 + w_2 x_2$
- If $w_1 x_1 + w_2 x_2 > \theta$ *then*

  $y = 1$

  otherwise

  $y = 0$

```
print(AND(0, 0))
print(AND(0, 1))
print(AND(1, 0))
print(AND(1, 1))
```

```
0
0
0
1
```

Bias

$$y = \begin{cases} 0 \ (w_1 x_1 + w_2 x_2 \le \theta) \\ 1 \ (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$$y = \begin{cases} 0 \ (-\theta + w_1 x_1 + w_2 x_2 \le 0) \\ 1 \ (-\theta + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

$$y = \begin{cases} 0 \ (b + w_1 x_1 + w_2 x_2 \le \theta) \\ 1 \ (b + w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

- $b = -\theta$
- w_1, w_2: weight
  - Weight controls the importance of input signals.
- b: bias
  - Bias controls the possibility of firing of neuron.

Implementation using bias and weight

```python
import numpy as np

def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5]) # w1=0.5, w2=0.5
    b = -0.7 # theta=0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    elif tmp > 0:
        return 1
```

$$
\left[
\begin{aligned}
&[x_1, x_2] * [w_1, w_2] \\
&\Rightarrow [x_1 w_1, x_2 w_2] \\
&\ np.sum(w * x) \\
&\ \Rightarrow x_1 w_1 + x_2 w_2
\end{aligned}
\right.
$$

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

```python
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5,-0.5]) # w1=-0.5, w2=-0.5
    b = 0.7 # theta=-0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    elif tmp > 0:
        return 1
```

x1	x2	y
0	0	1
0	1	1
1	0	1
1	1	0

Implementation using bias and weight 2

```python
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5]) # w1=0.5, w2=0.5
    b = -0.2 # theta=0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    elif tmp > 0:
        return 1
```

```
x1  x2 | y
----------
 0   0 | 0
 0   1 | 1
 1   0 | 1
 1   1 | 1
```

```python
def run_logic_circuit(logic):
    print('x1 x2 | y')
    print('----------')
    print(' 0  0 |', logic(0,0))
    print(' 0  1 |', logic(0,1))
    print(' 1  0 |', logic(1,0))
    print(' 1  1 |', logic(1,1))
```

```python
run_logic_circuit(AND)
```

```python
run_logic_circuit(NAND)
```

```python
run_logic_circuit(OR)
```

What is “learning”?

- In this implementations, we (human) decided the parameters of perceptron (w1, w2, b).
- In the problems of machine learning, computer decides the parameters automatically.

- “Learning” is a process to decide optimal parameter value.

Limitation of perceptron

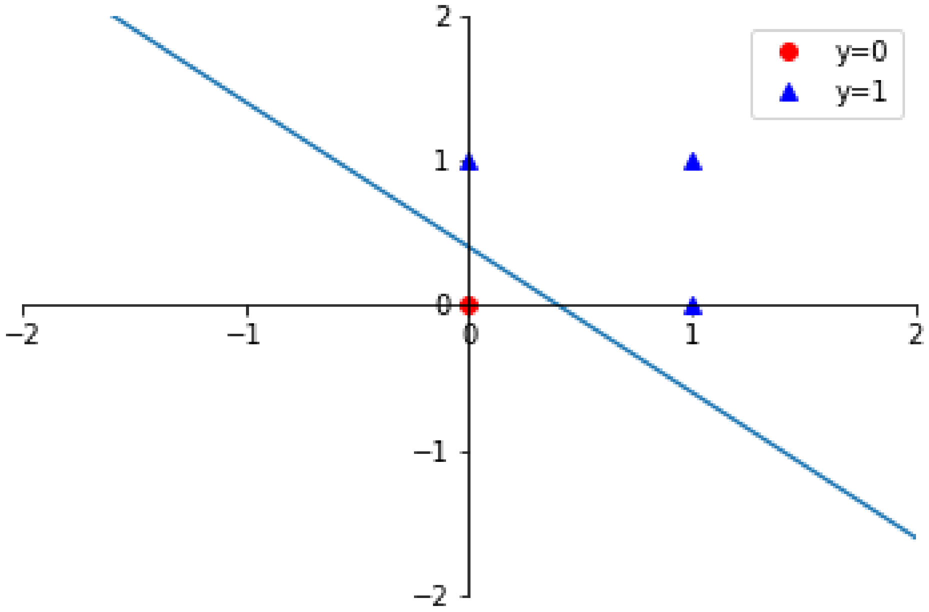
Visual image of OR perceptron

$$(w_1, w_2, \theta) = (0.5, 0.5, 0.2)$$

$$y = \begin{cases} 0 \ (-0.2 + 0.5x_1 + 0.5x_2 \leq \theta) \\ 1 \ (-0.2 + 0.5x_1 + 0.5x_2 > \theta) \end{cases}$$

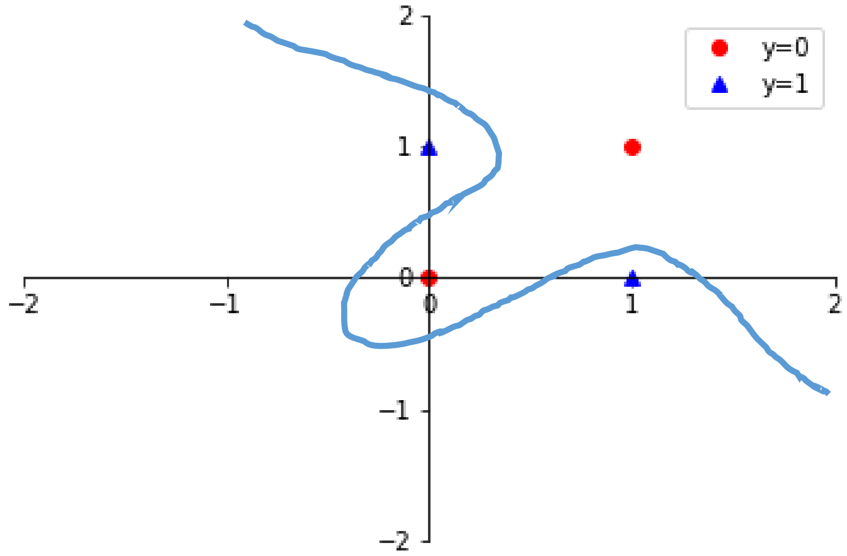
It needs to be separated ○ and △ by straight line, to make OR gate.

- ➢ It is well separated.



XOR (Exclusive OR) gate

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



- How can we separate these points by straight line?

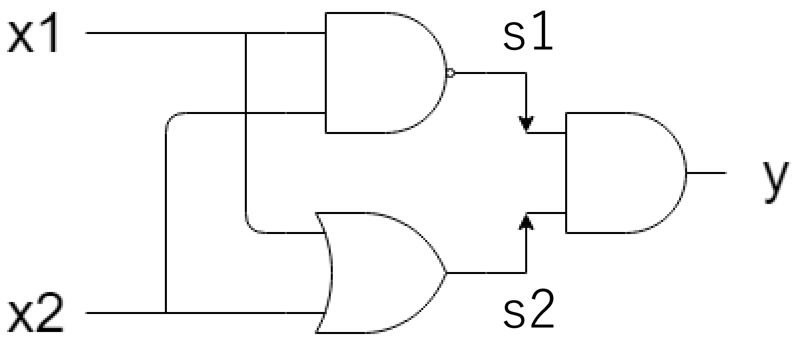
Impossible!

↑ Limitation of simple perceptron

➢ But It is possible to separate by the curve.

- The region separated by straight line: linear region
- The region separated by curve: nonlinear region

How can we implement XOR gate?

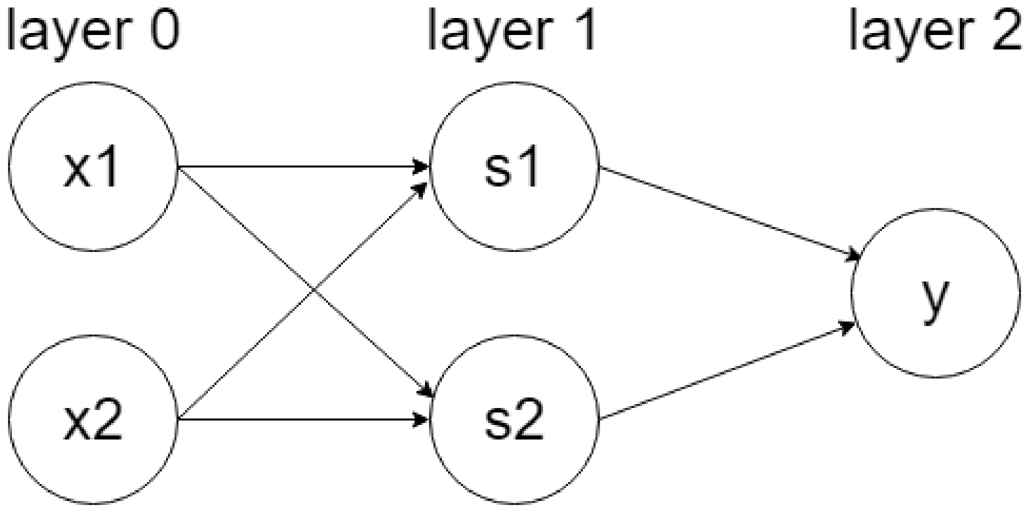


- Express XOR using AND, NAND, OR

```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Notation of XOR by perceptron



- Two layer perceptron. Different from AND, NAND, OR
- Multi-layered perceptron: The perceptron consists of multiple layers.
- The perceptron makes it possible to express more flexibly by deepening the layer.

From NAND to the computer

- Multilayer perceptron can make more complex circuit.
  E.g.) adder, encoder converting binary to decimal
- The computer can make from combination of NAND gate.
  NAND gate can make by perceptron.
  - ➢The computer can make by perceptron.
- How many layers does it need to make computer?
- In theory, only two layers
  - ➢Because it was proved that any functions can make by two layers perceptron.

Summary

- Perceptron algorithm has inputs and output. When a certain input is given, a fixed value is output.
- Perceptron uses “weight” and “bias” as parameters.
- Perceptron can express logic circuit.
- Simple (single layer) perceptron can’t express XOR.
- While simple perceptron can only express linear region, multilayer perceptron can express nonlinear region.
- Multilayer perceptron can express computer.