SCC.222: Artificial Intelligence Concepts

Week 8: Multi-Layer Perceptron and Basic Neural Networks

Jun Liu

j.liu81@lancaster.ac.uk

Jun Liu

- Professor, School of Computing and Communications
- Research Area: *Digital Health, Computer Vision, Machine Learning*
- Email: *j.liu81@lancaster.ac.uk*
- Office: *#C51, InfoLab21*

Biological Neural Network

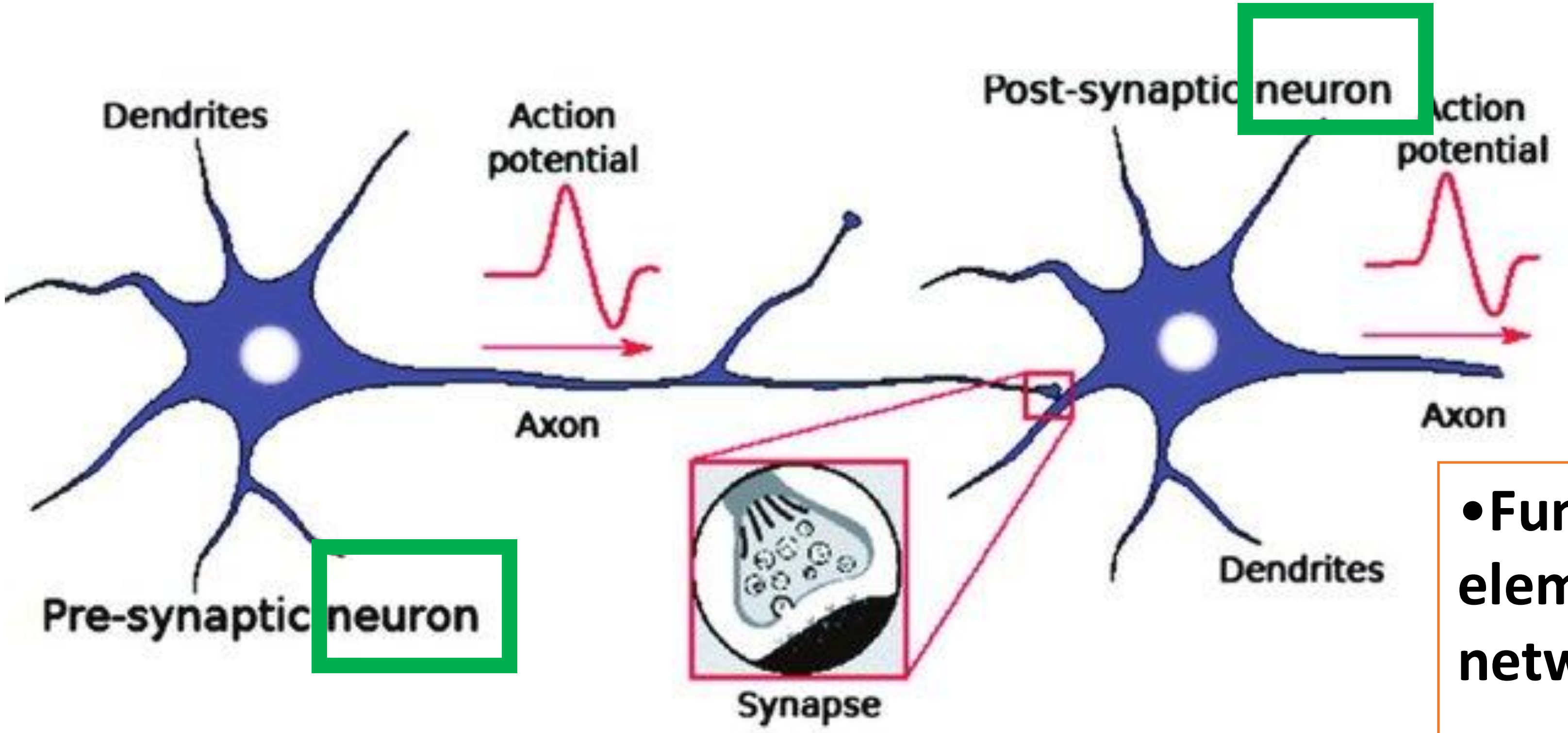


Image from https://www.intechopen.com/chapters/56580, A. Huang, X. Zhang, R. Li and Y. Chi

- **Fundamental processing elements of the biological neural network are neurons.**
- **A honeybee brain has one million neurons.**
- **A human brain has 86 billion neurons.**

Credit (adapted from): Jalal Mahmud, Hyung-Yeon, Gu

Biological Neural Network

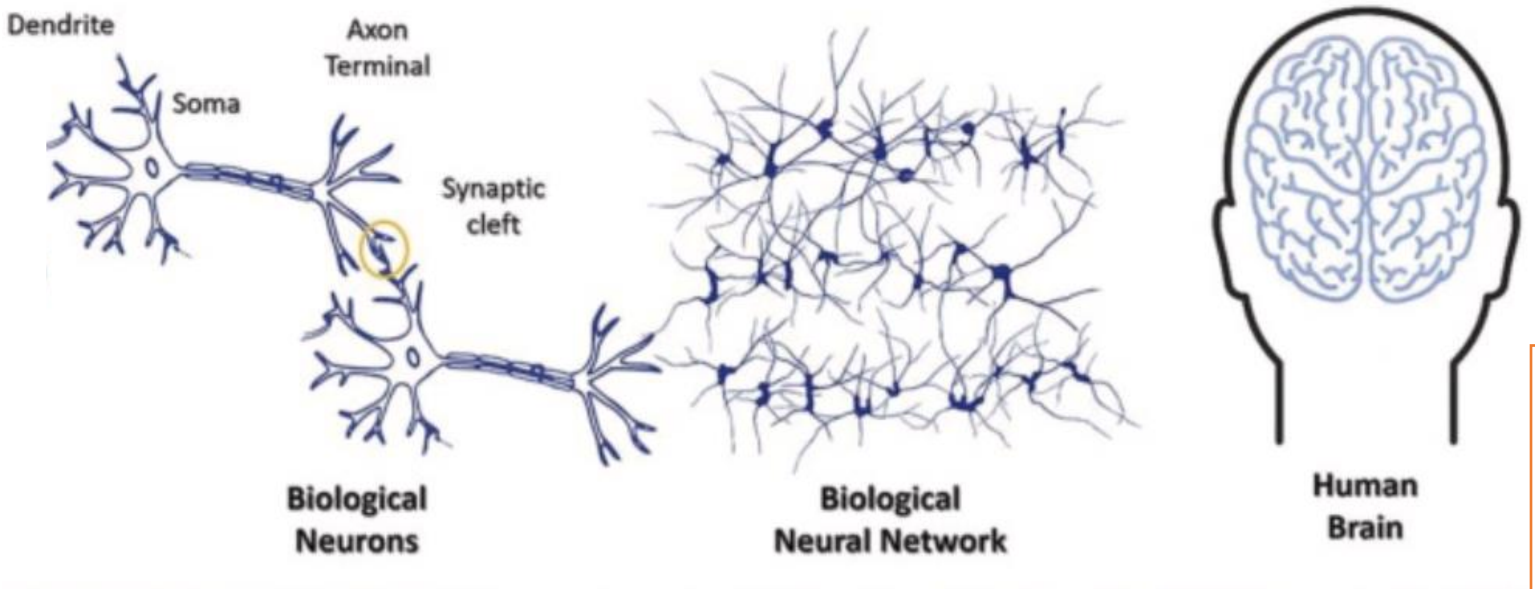


Image from https://doi.org/10.1016/j.memori.2023.100088 (T. Ahmed)

- **Fundamental processing elements of the biological neural network are neurons.**
- **A honeybee brain has one million neurons.**
- **A human brain has 86 billion neurons.**

Credit (adapted from): Jalal Mahmud, Hyung-Yeon, Gu

Biological Neural Network - > Artificial Neural Network

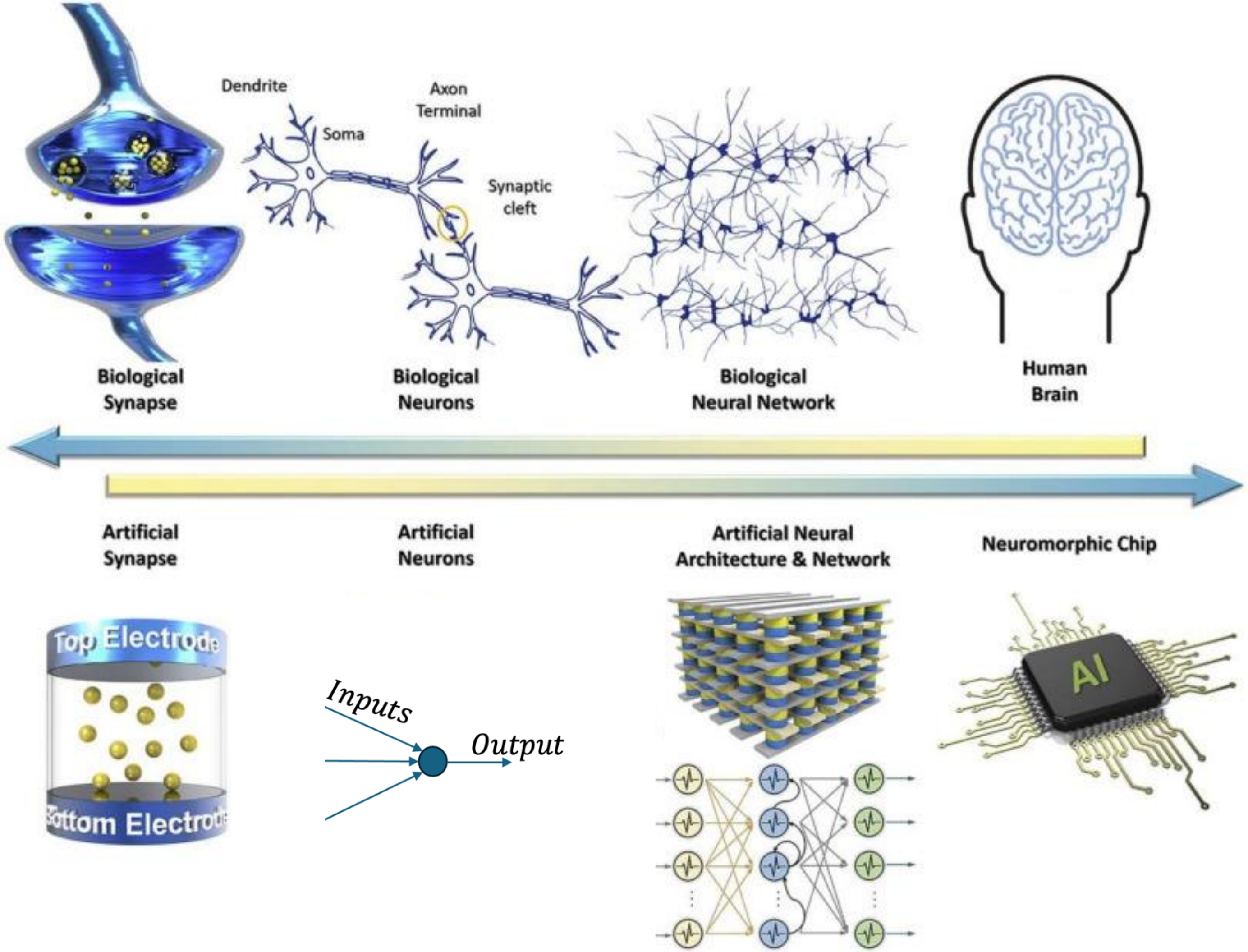


Image adapted from https://doi.org/10.1016/j.memori.2023.100088 (T. Ahmed)

- **Neural Network (NN) is a biologically motivated approach to machine learning.**
- **NN's similarities *w.r.t* biological neural network:**
  - NN's fundamental processing elements are also neurons
  - NN also receives inputs from other sources, then combines them in someway, and finally outputs the final result.

Credit (adapted from): Jalal Mahmud, Hyung-Yeon, Gu

Biological Neural Network - > Artificial Neural Network

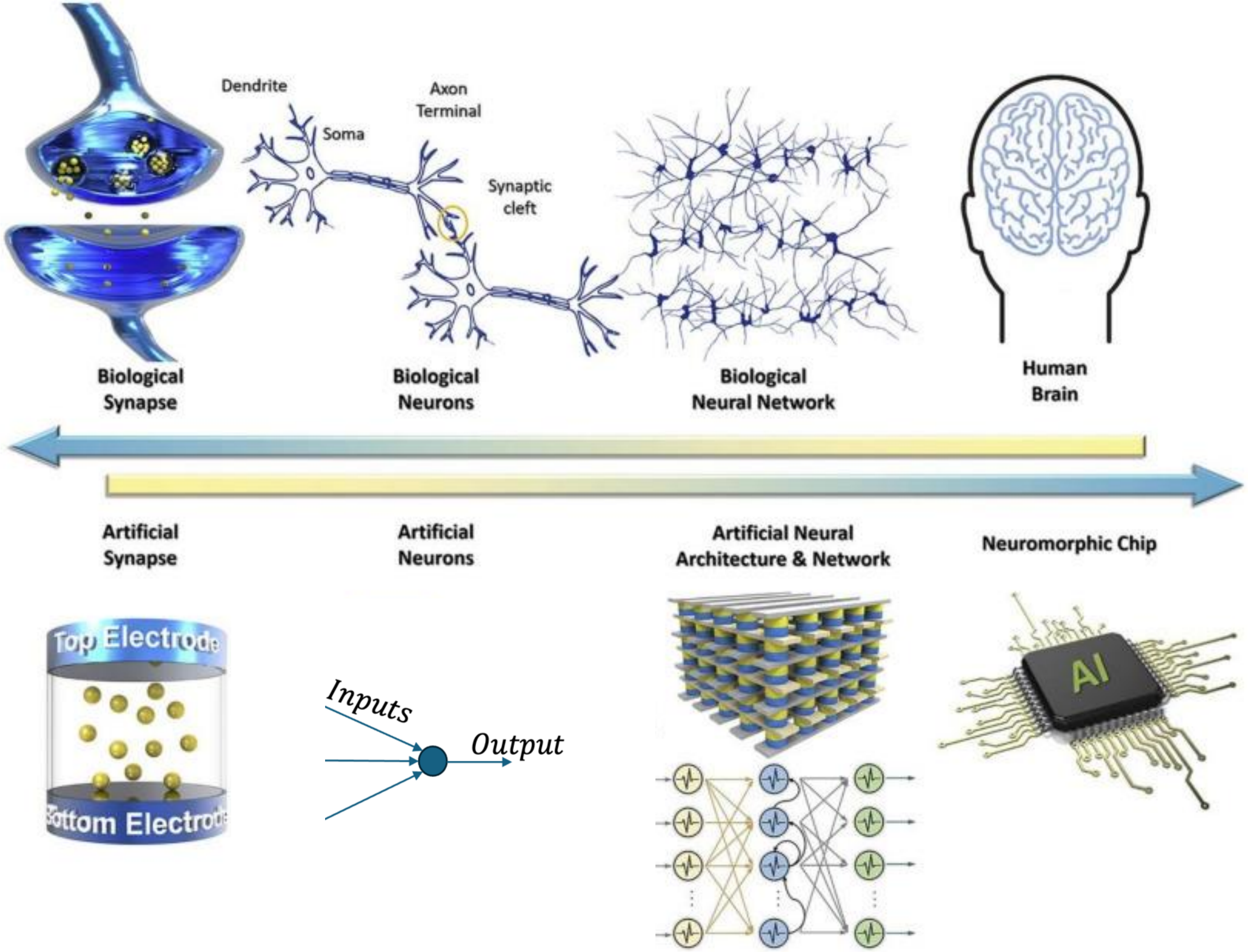
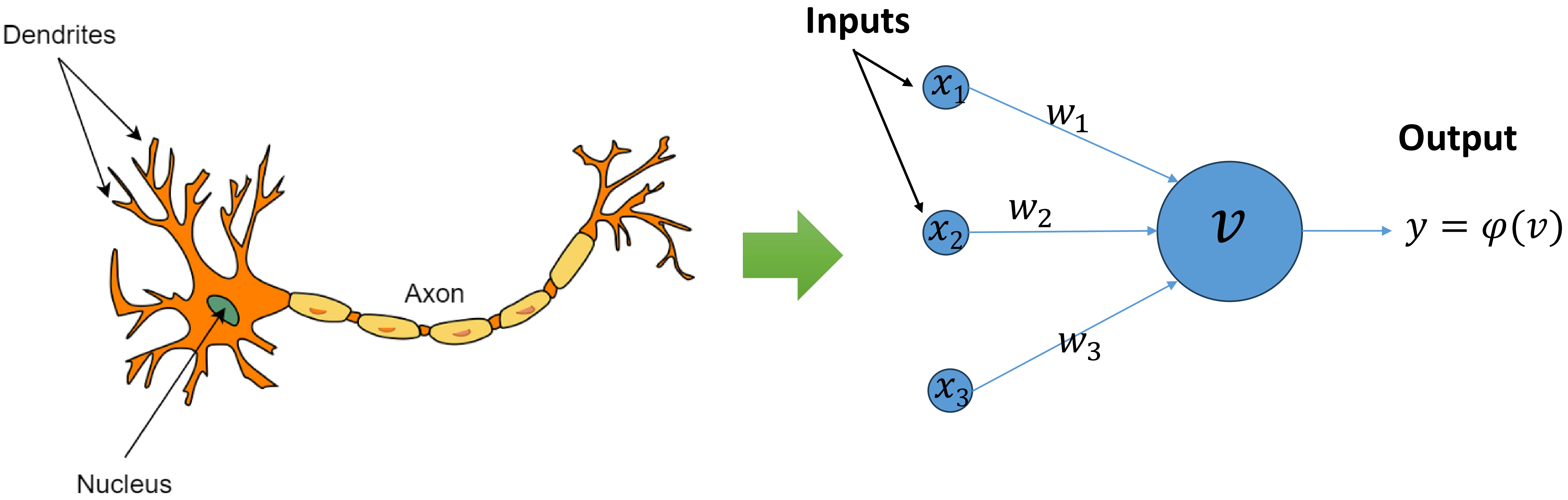


Image adapted from https://doi.org/10.1016/j.memori.2023.100088 (T. Ahmed)

- NN tries to mimic human brains.
- NNs are quite hot in 80s and early 90s; Popularity diminished in late 90s. *(Do you know why?)*
- Resurgence in the past 10 years: state-of-the-art techniques for many applications.
- NNs are not nearly as complex as the actual human brain structure.

Credit: Andrew Ng

Neuron

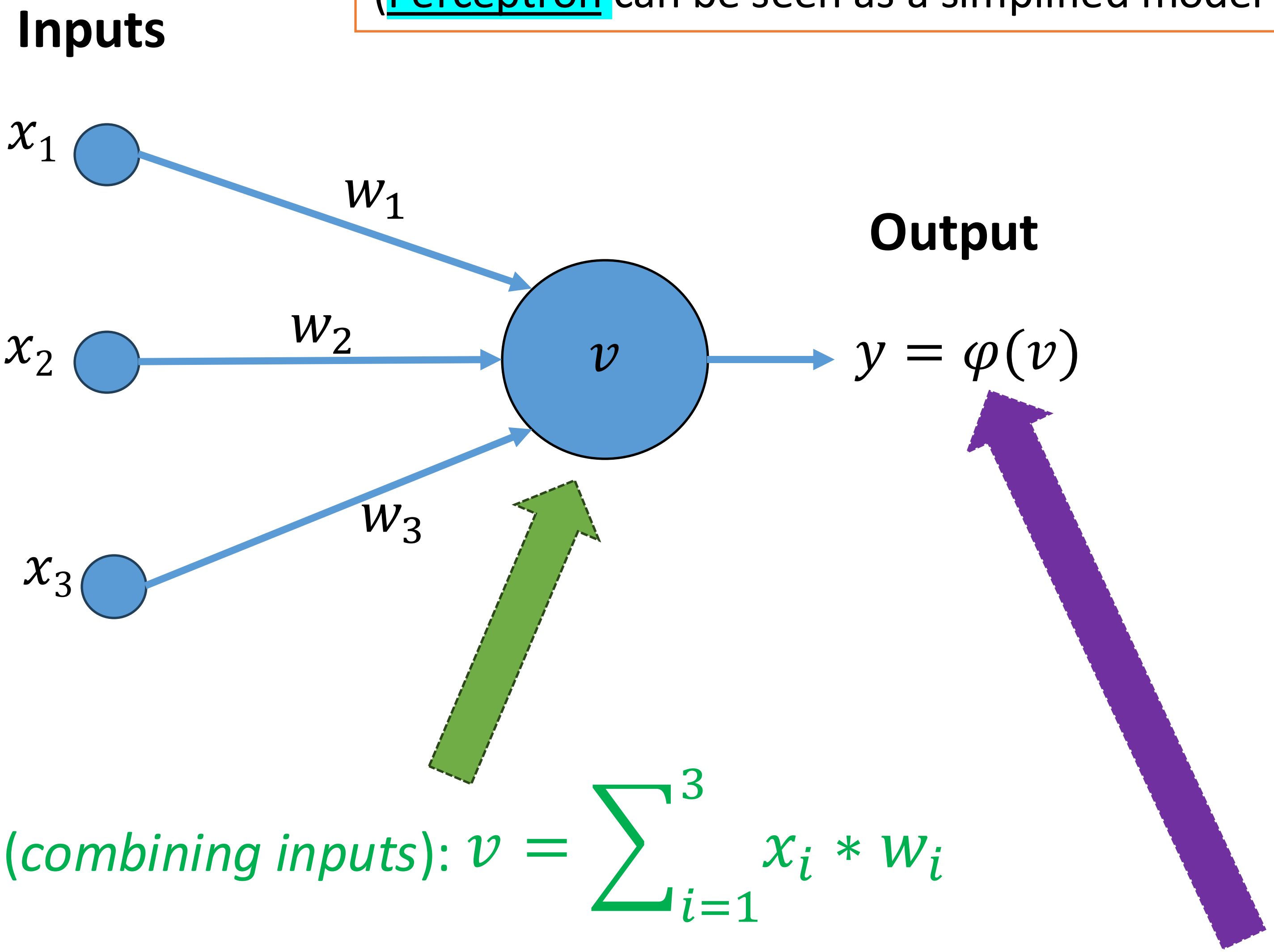


NN’s fundamental processing elements are also neurons.

Image adapted from https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc (R. Pramoditha)

Perceptron

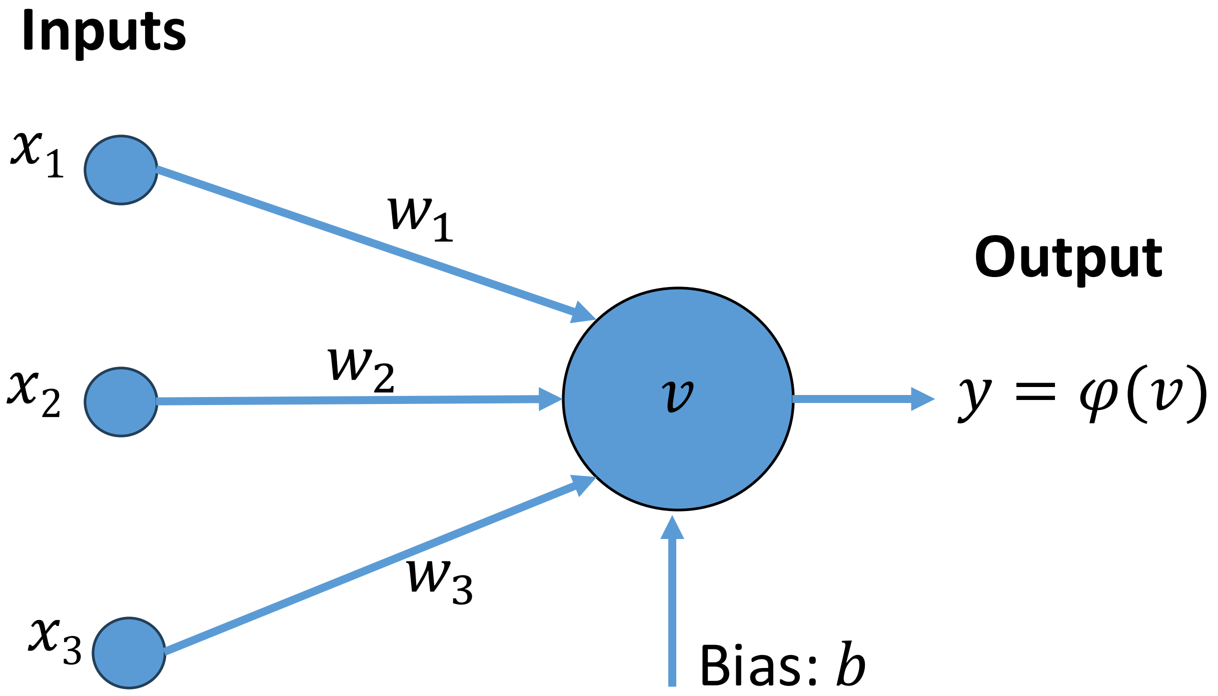
- **In machine learning, we call the neuron as perceptron.** (Perceptron can be seen as a simplified model of a biological neuron).



Linear Function (*combining inputs*): $v = \sum_{i=1}^{3} x_i * w_i$

Activation Function (*doing some additional operations*): $\varphi(v)$

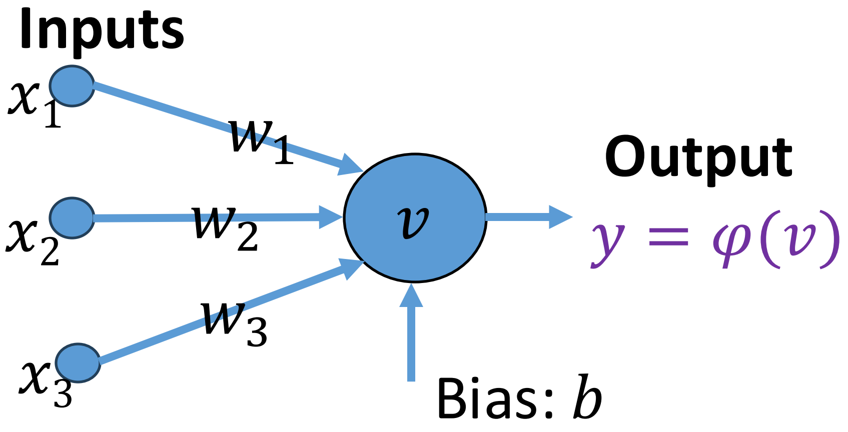
Perceptron



w_1, w_2, w_3 and b are called ***network weights*** **(or *network parameters*).**

Linear Function: $v = \sum_{i=1}^{3} x_i * w_i + b$

Activation Function $\varphi(v)$



We can choose different types of activation functions, based on application requirements.

- **Linear**

$$Linear(x) = x$$

1. Please plot the function

2. What is the range of output

- **Rectified Linear Unit (ReLU)**

$$Relu(x) = \begin{cases} x, x > 0 \\ 0, x \le 0 \end{cases}$$

1. Please plot the function

2. What is the range of output

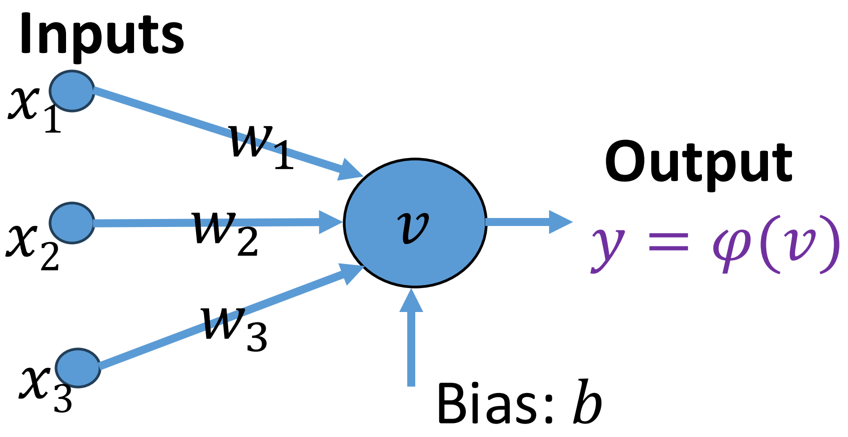
- **Hard Threshold (Step Function)**

$$Step(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$$

1. Please plot the function

2. What is the range of output

Activation Function $\varphi(v)$

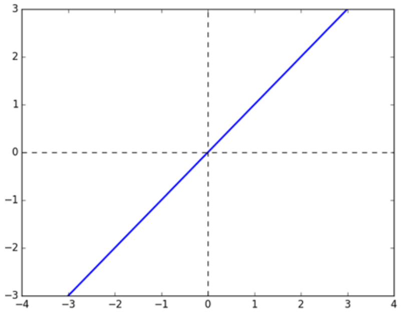


We can choose different types of activation functions, based on application requirements.

➢Linear

$$Linear(x) = x$$

1. Please plot the function



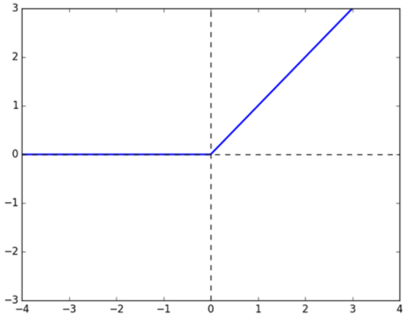
2. What is the range of output

$(-\infty, +\infty)$

➢Rectified Linear Unit (ReLU)

$$Relu(x) = \begin{cases} x, x > 0 \\ 0, x \le 0 \end{cases}$$

1. Please plot the function



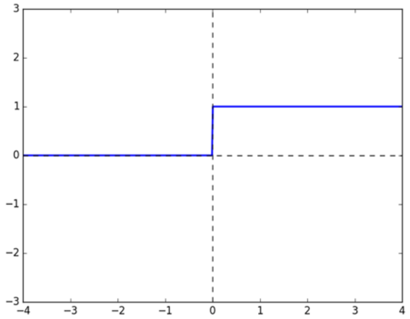
2. What is the range of output

$[0, +\infty)$

➢Hard Threshold (Step Function)

$$Step(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$$

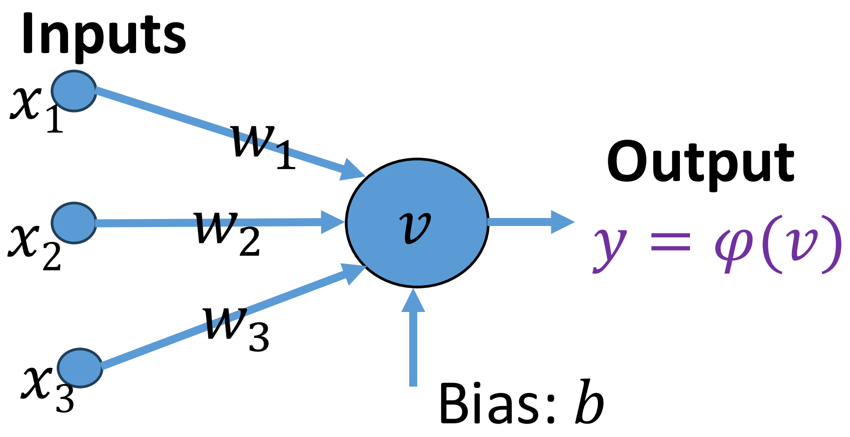
1. Please plot the function



2. What is the range of output

0 or 1

Activation Function $\varphi(v)$



We can choose different types of activation functions, based on application requirements.

➢Sigmoid (σ)

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

1. Please plot the function

2. What is the range of output

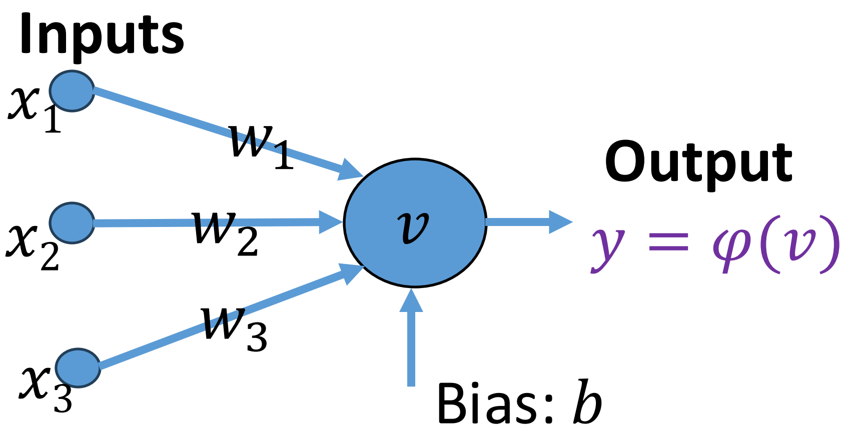
➢Tanh

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

1. Please plot the function

2. What is the range of output

Activation Function $\varphi(v)$



Inputs

Output

$y = \varphi(v)$

Bias: b

We can choose different types of activation functions, based on application requirements.

➢Sigmoid (σ)

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

1. Please plot the function

2. What is the range of output

$(0, 1)$

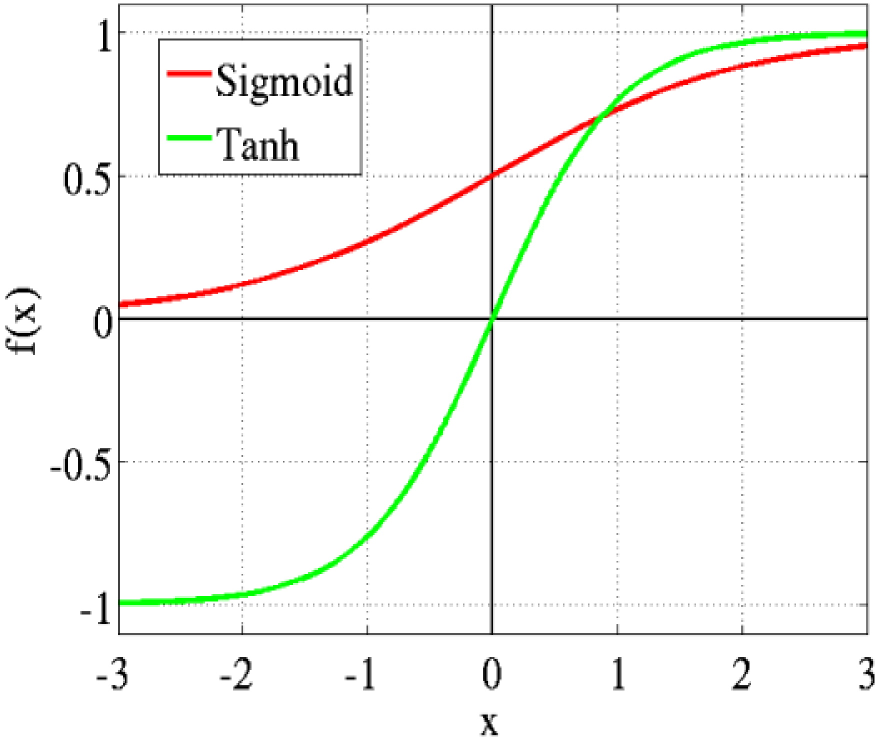
➢Tanh

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

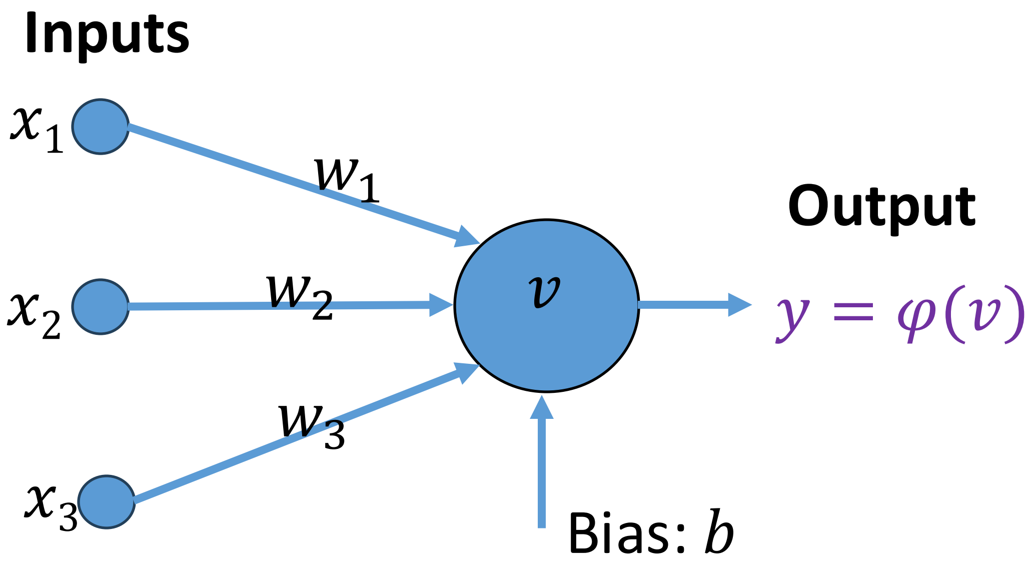
1. Please plot the function

2. What is the range of output

$(-1, 1)$

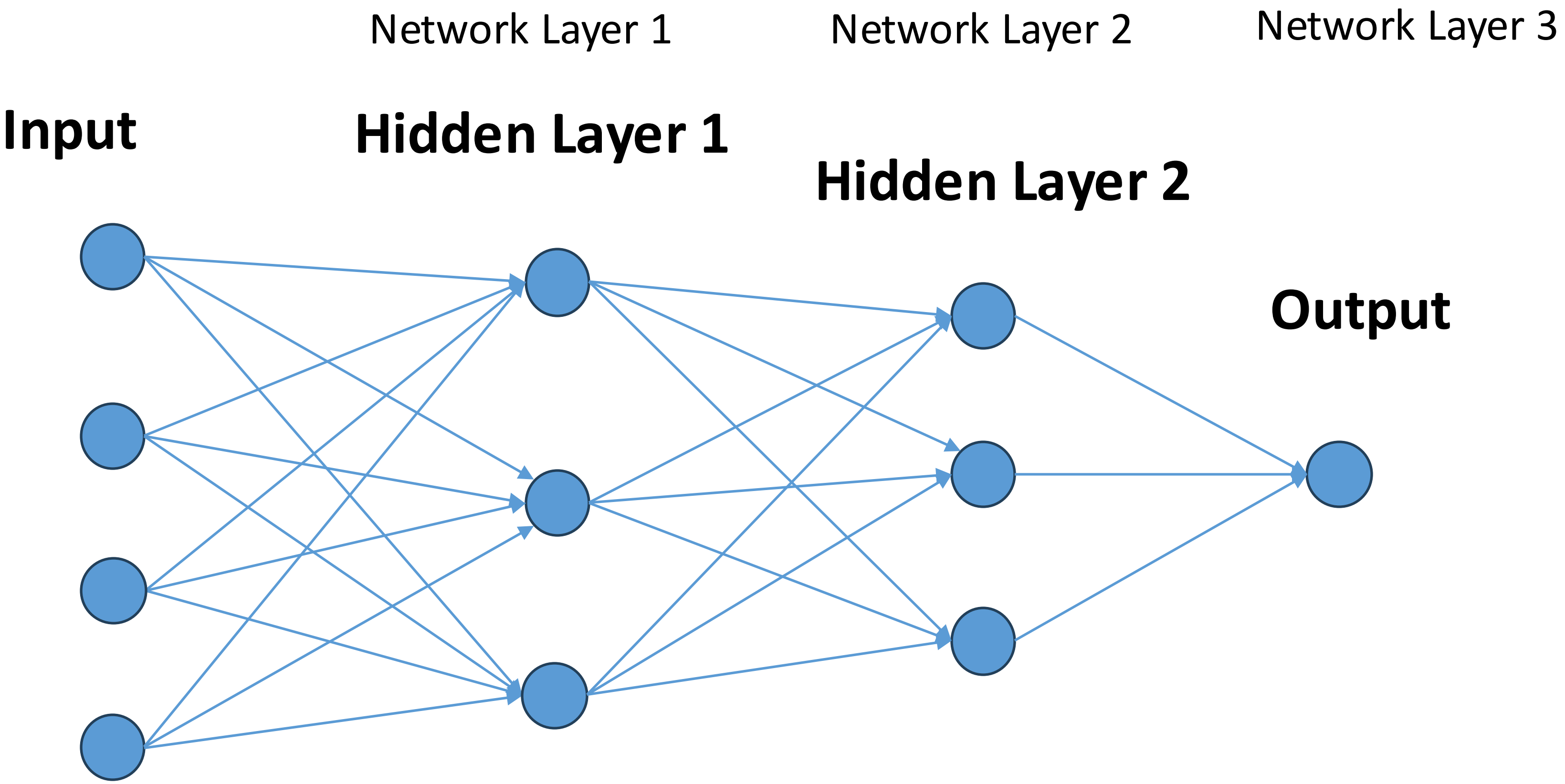


Activation Function $\varphi(v)$



- Please list some applications, where it is good to use ***Step function*** (or ***Sigmoid function***) as the activation function.

Multi-Layer Perceptron (MLP)



Network Layer 1

Network Layer 2

Network Layer 3

Input

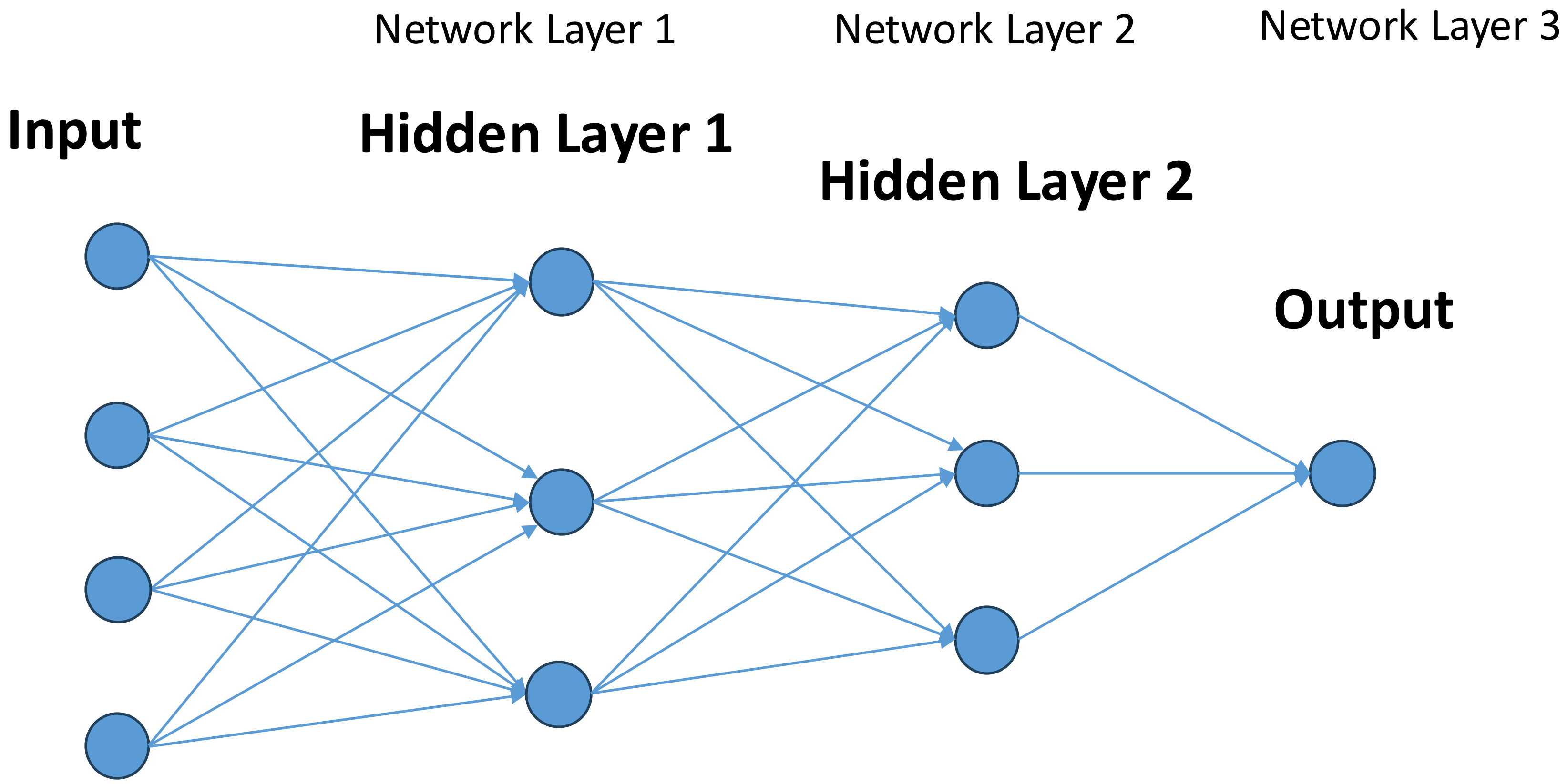
Hidden Layer 1

Hidden Layer 2

Output

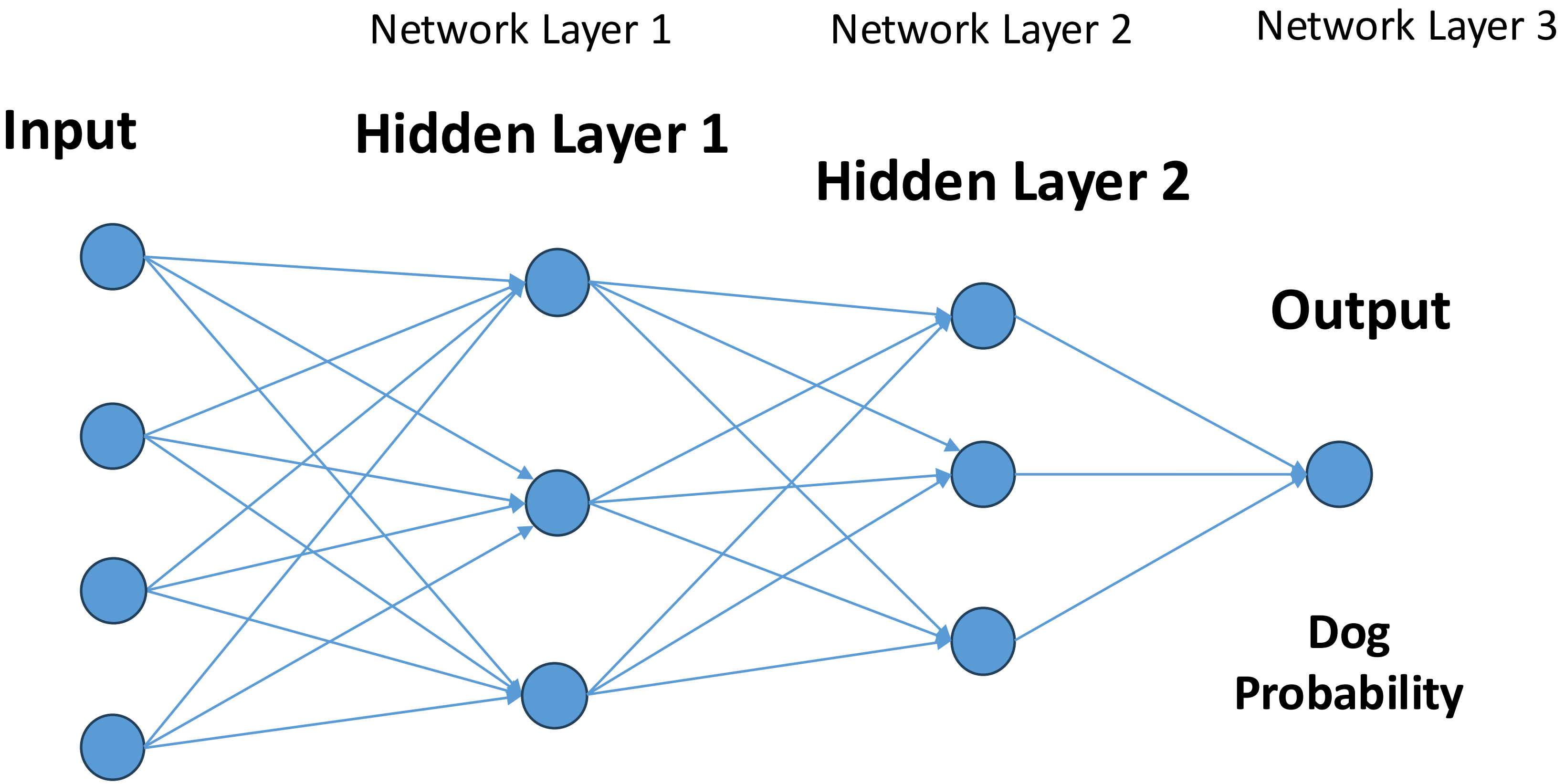
Stacking multiple layers of perceptrons (neurons), we get the multi-layer perceptron (MLP), which has stronger expressive capability, compared to a single perceptron. *(Question: why stronger?)*

Multi-Layer Perceptron (MLP)



Question: How many parameters are in this MLP?

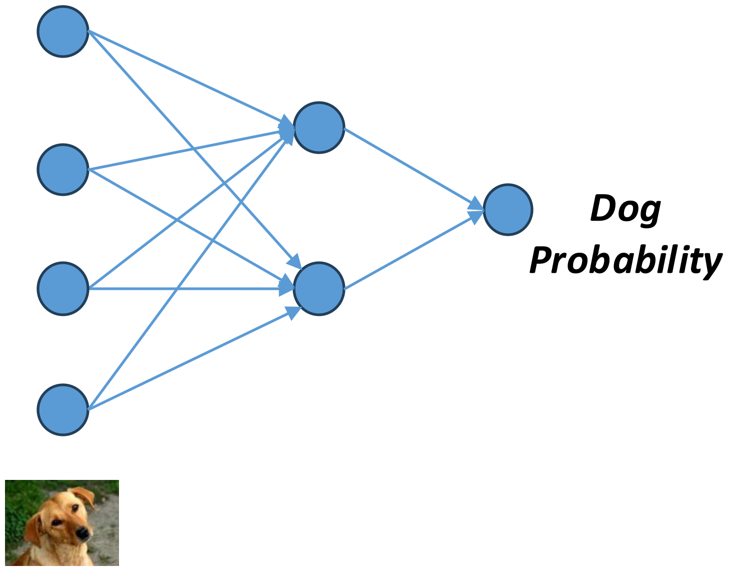
Single output



This is a binary classification problem

Credit: Andrew Ng

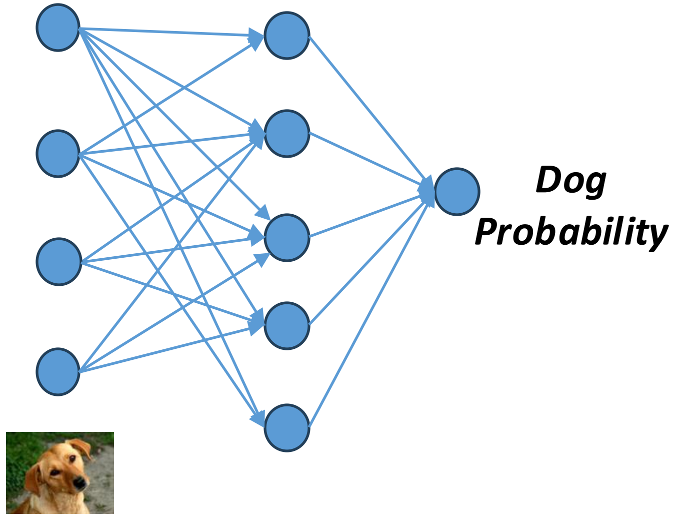
Single output



(a)

How many network layers?

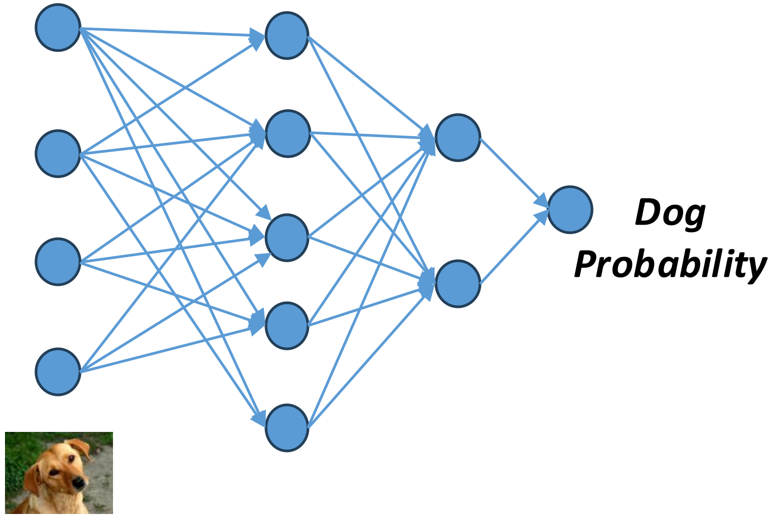
How many parameters?



(b)

How many network layers?

How many parameters?



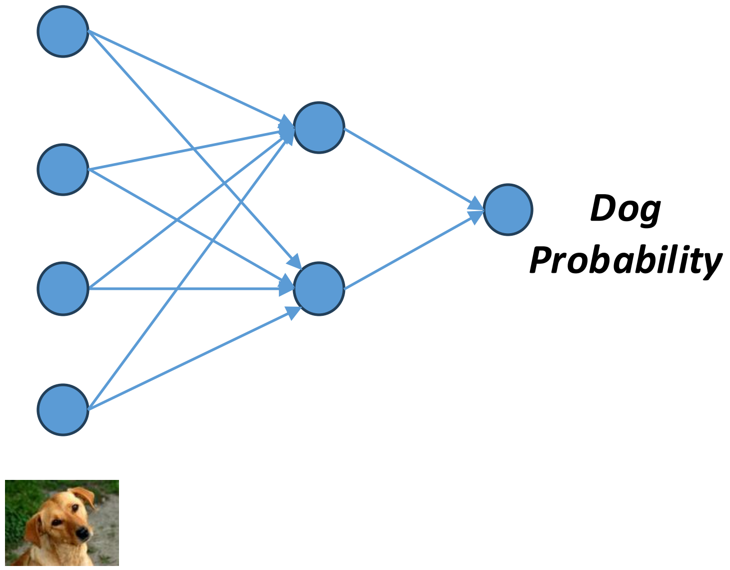
(c)

How many network layers?

How many parameters?

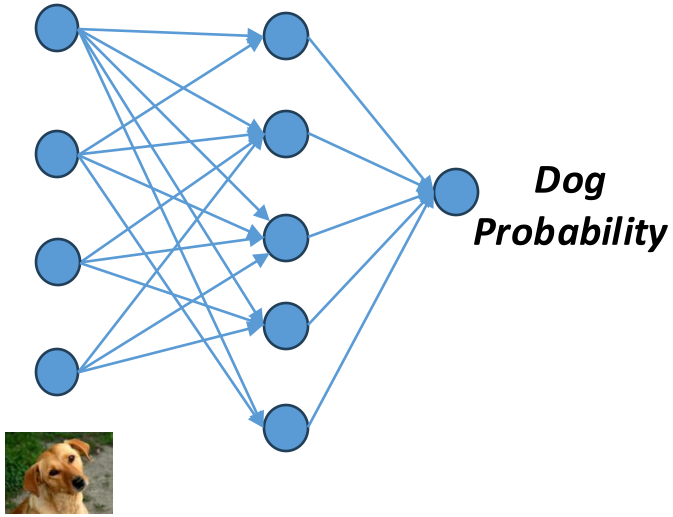
Credit: Andrew Ng

Single output



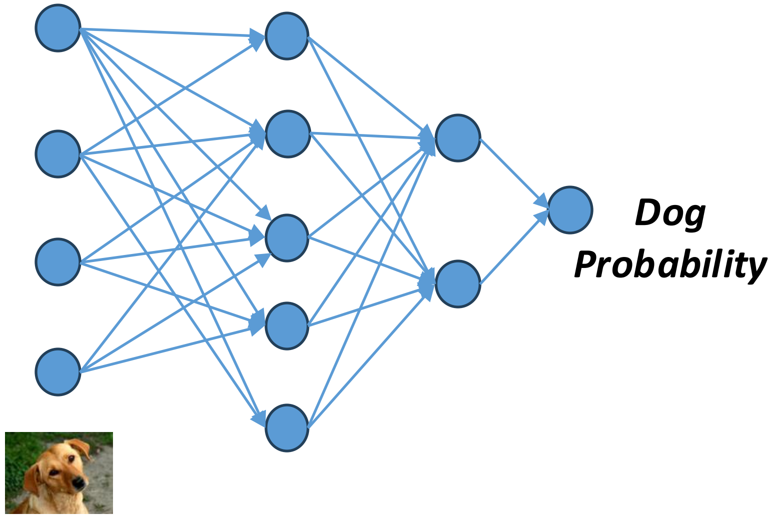
(a)

How many network layers?
2
How many parameters?
10



(b)

How many network layers?
2
How many parameters?
25

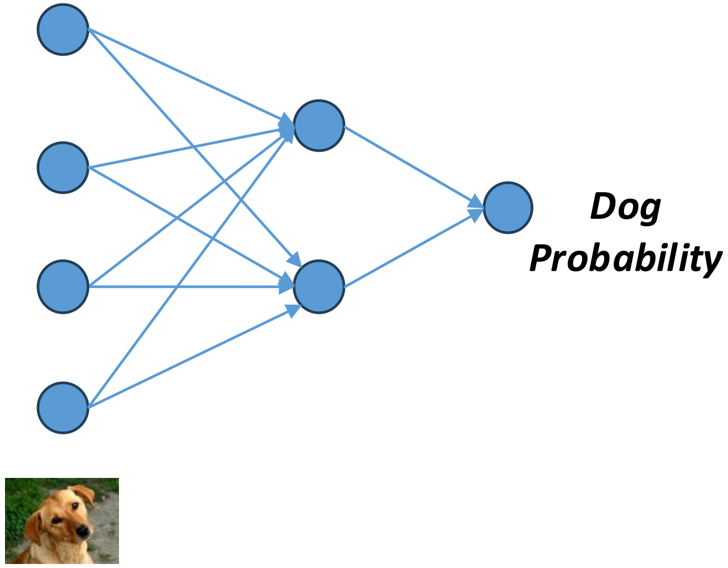


(c)

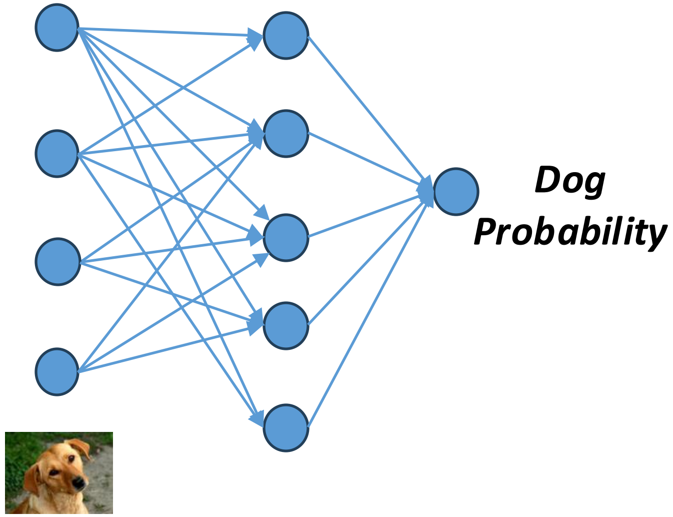
How many network layers?
3
How many parameters?
32

Credit: Andrew Ng

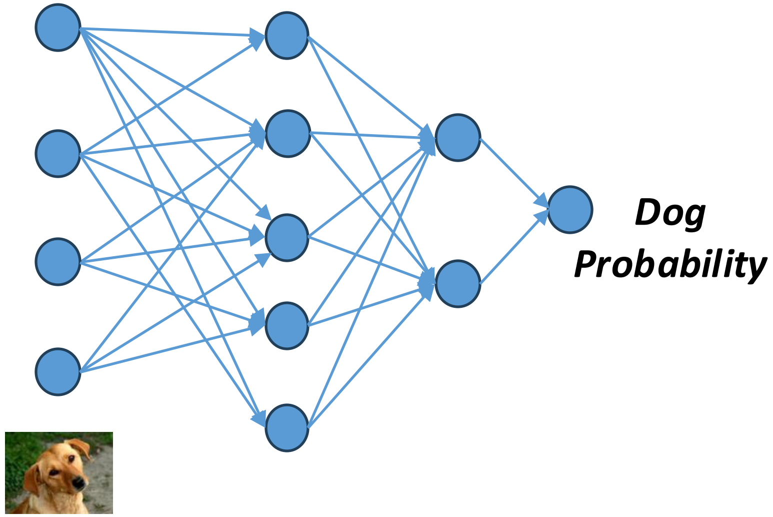
Single output



(a)



(b)

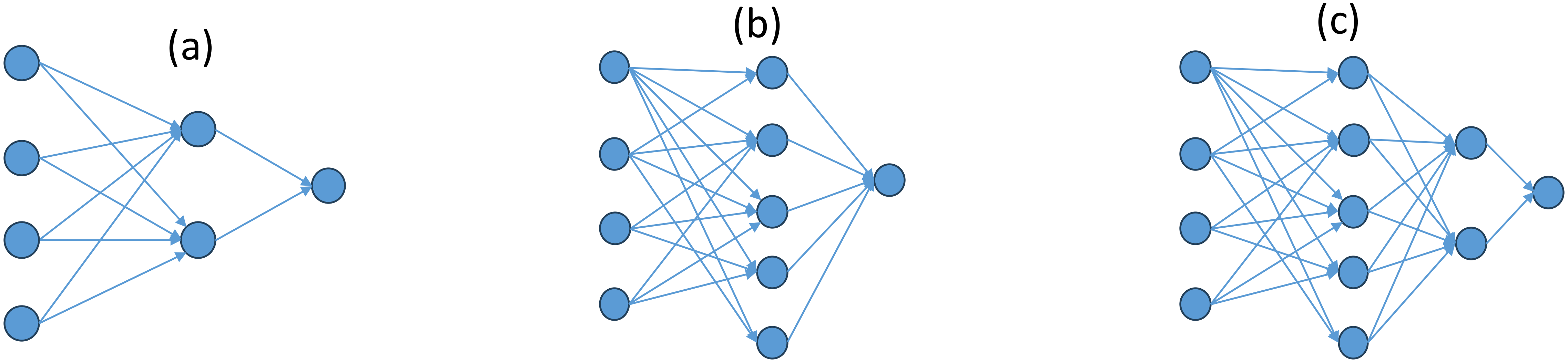


(c)

Which network architecture is the best?

Credit: Andrew Ng

Single output



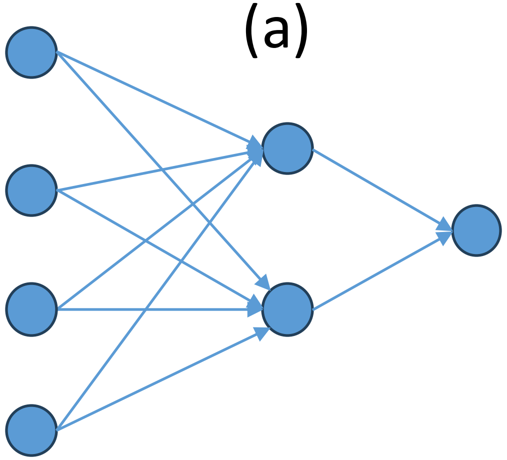
Which network architecture is the best?

➢ Hard to say which is the best.

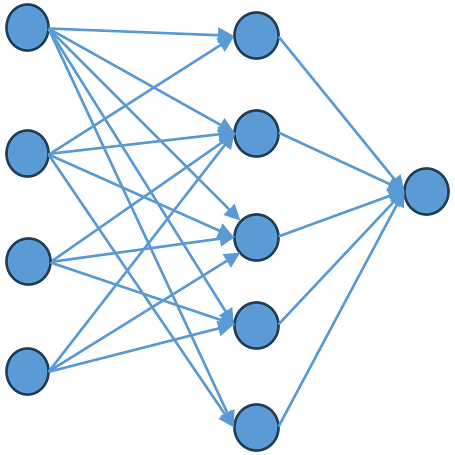
- Network (a) has less parameters, so computation cost is the smallest, but its representation capability is relatively weak.
- Network (c) has more layers and more parameters, so computation cost is the biggest, but representation capability is the strongest (can handle more complex problems)

Single output

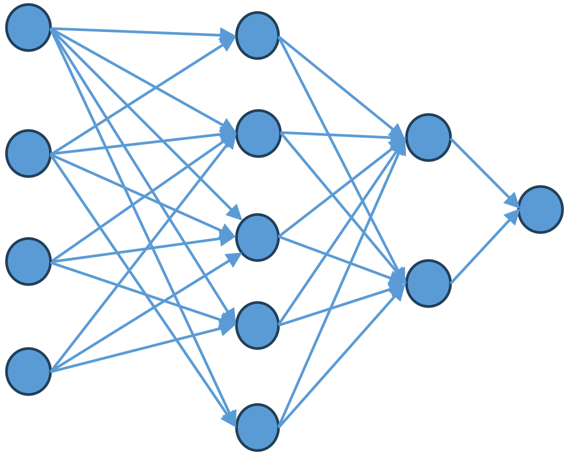
(a)



(b)



(c)



Which network architecture is the best?

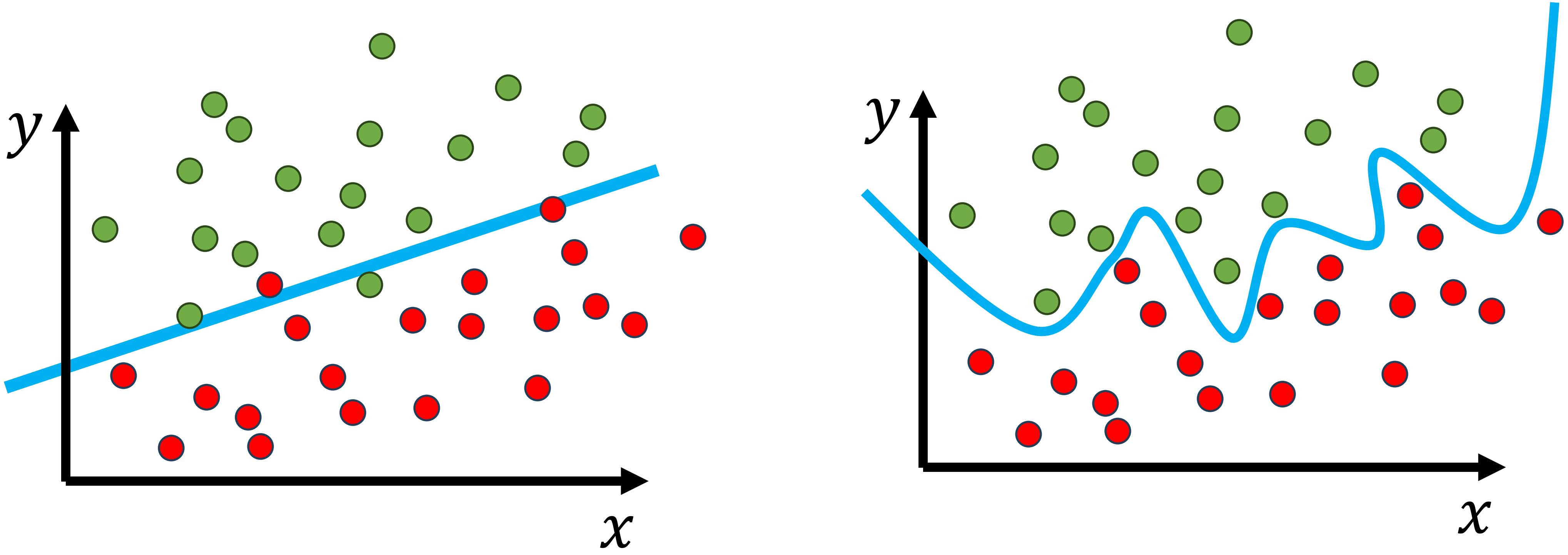
➢ Hard to say which is the best.

- Network (a) has less parameters, so computation cost is the smallest, but its representation capability is relatively weak.
- Network (c) has more layers and more parameters, so computation cost is the biggest, but representation capability is the strongest (can handle more complex problems)

Question: if we do not care about computation cost, is it good to always use the most complex network (c)?

No. If the network is too complex (with too many layers and too many parameters), it is easily to have the **overfitting** problem.

Overfitting

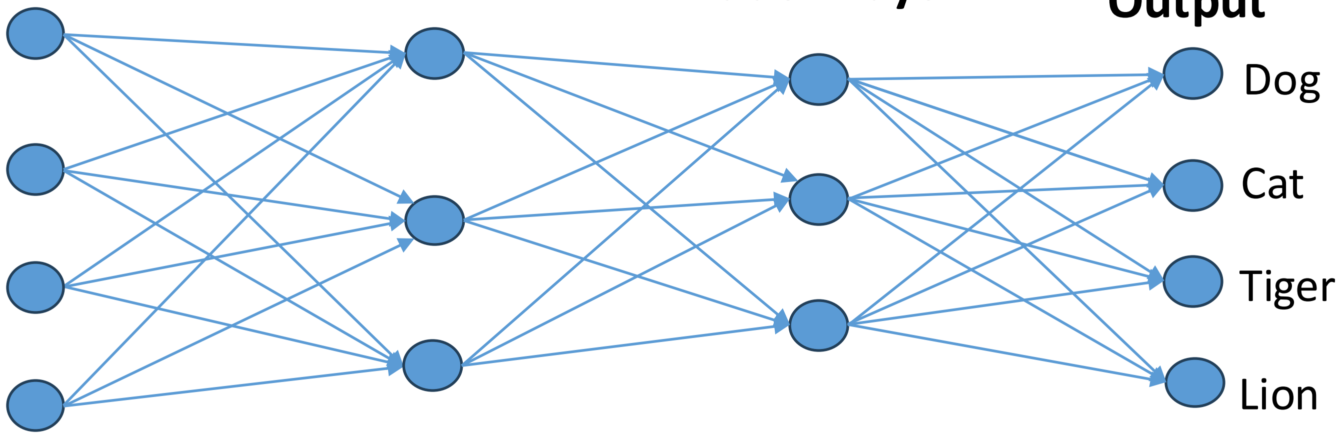


If we use a complex network to handle a simple problem ……..

Multiple outputs

Network Layer 1 *Network Layer 2* *Network Layer 3*

Input **Hidden Layer 1** **Hidden Layer 2** **Output**



This is a multi-class classification problem

We want to get the output:

$$y \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

for:

$$y \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

for:

$$y \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

for:

$$y \approx \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

for:

Credit: Andrew Ng

Please design a Perceptron to calculate AND

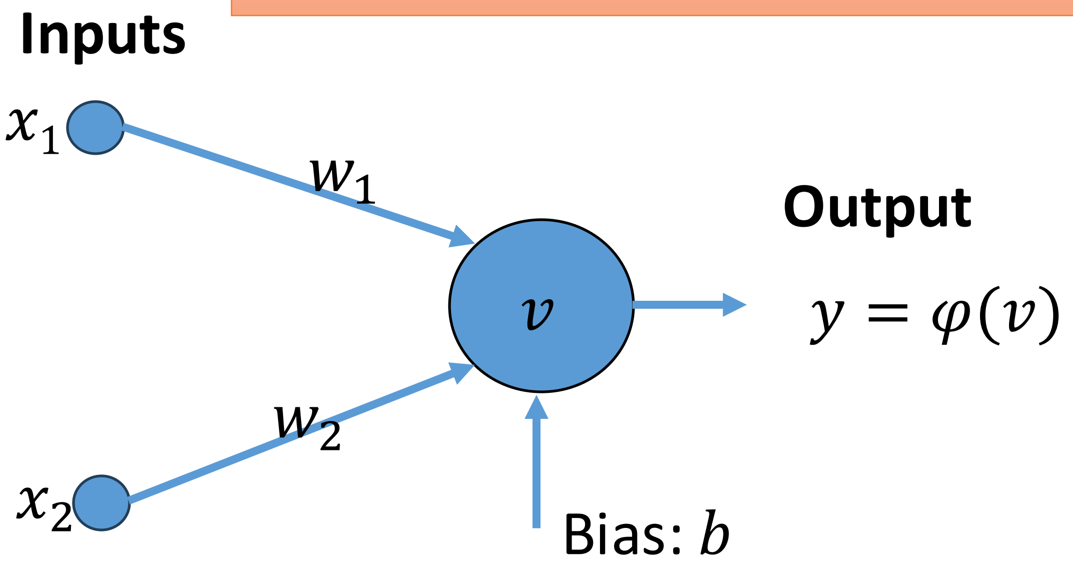
The AND operation:

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$

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

To design the MLP:

- At the first step, we need to identify: (1) how many inputs? (2) how many outputs? Then we can get the design below.



Please design a Perceptron to calculate AND

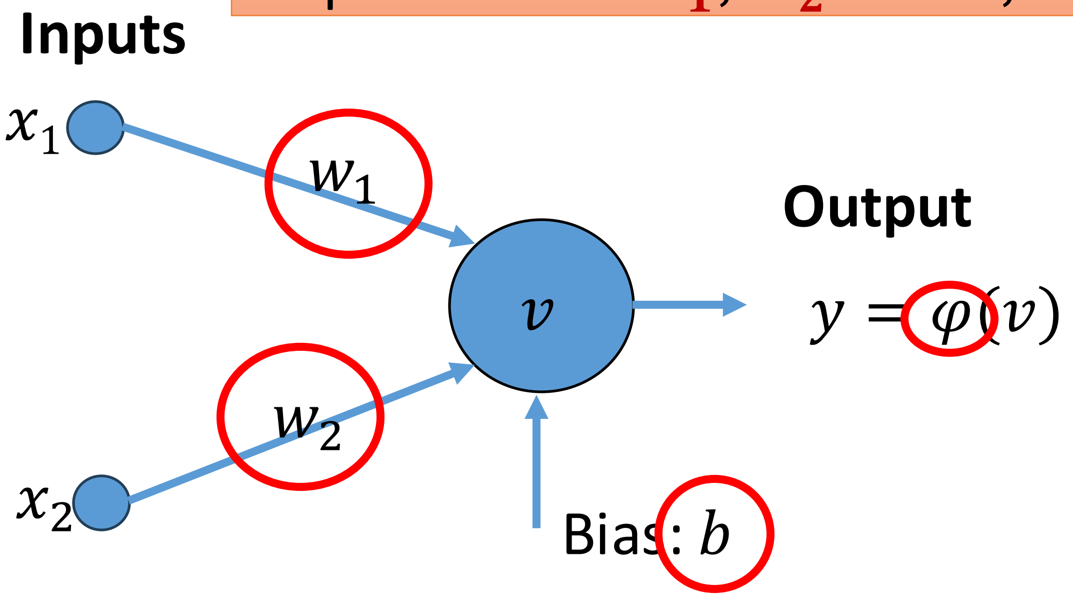
The AND operation:

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$

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

To design the MLP:

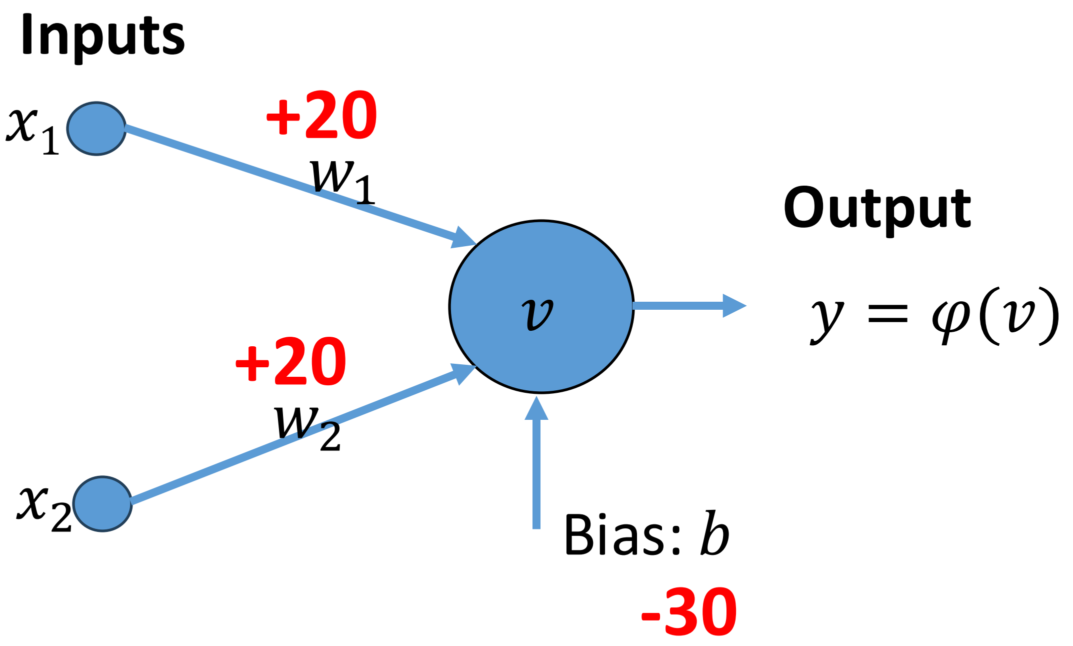
- At the first step, we need to identify: (1) how many inputs? (2) how many outputs? Then we can get the design below.
- At the second step, we need to decide: (1) what activation φ to use? (2) how to choose the network parameters w_1, w_2 and b,

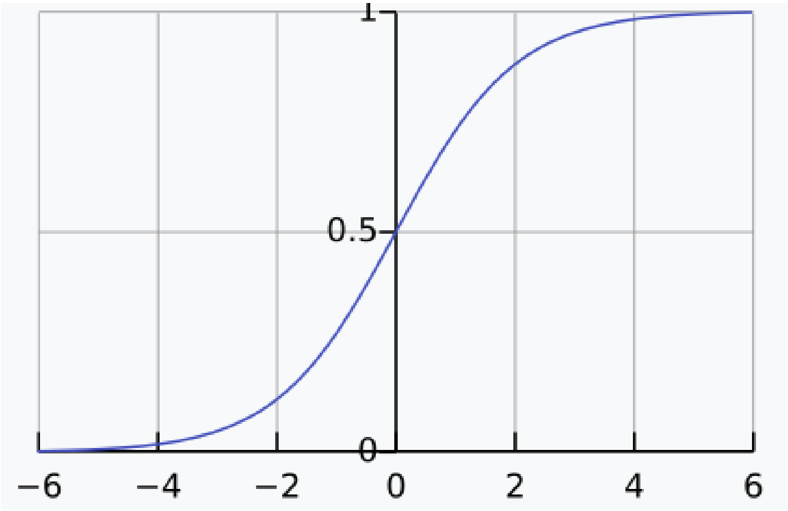


Please design a Perceptron to calculate AND

The AND operation:

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



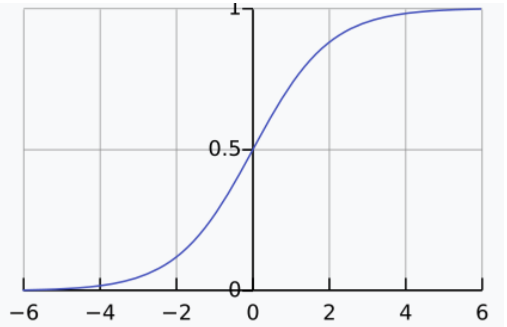


Sigmoid activation function

x_1	x_2	$y = \varphi(v)$
0	0	$\varphi(-30) \approx 0$
0	1	$\varphi(-10) \approx 0$
1	0	$\varphi(-10) \approx 0$
1	1	$\varphi(10) \approx 1$

Credit: Andrew Ng

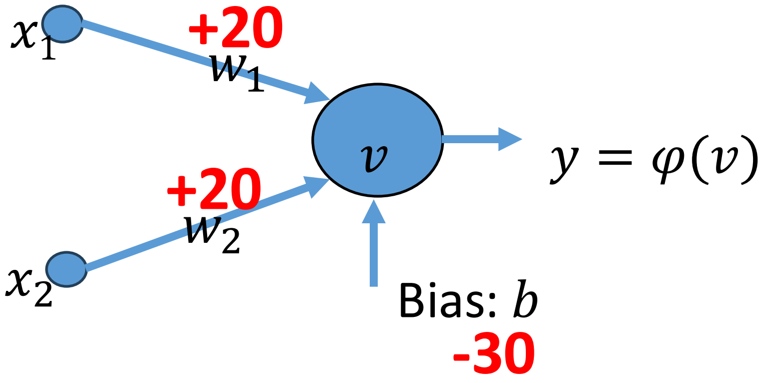
Perceptron for Calculating AND, OR, NOT



Sigmoid activation function

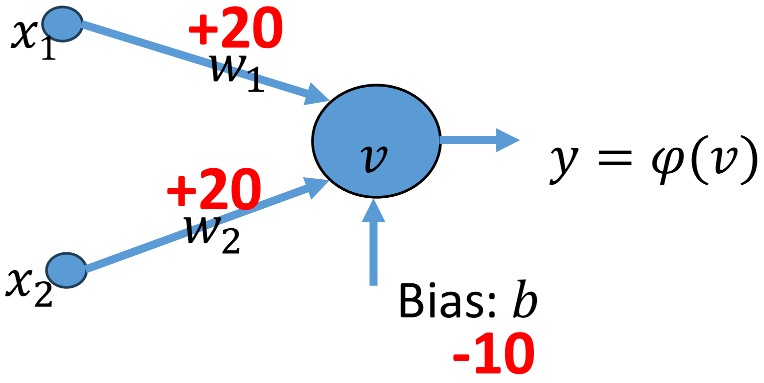
AND

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



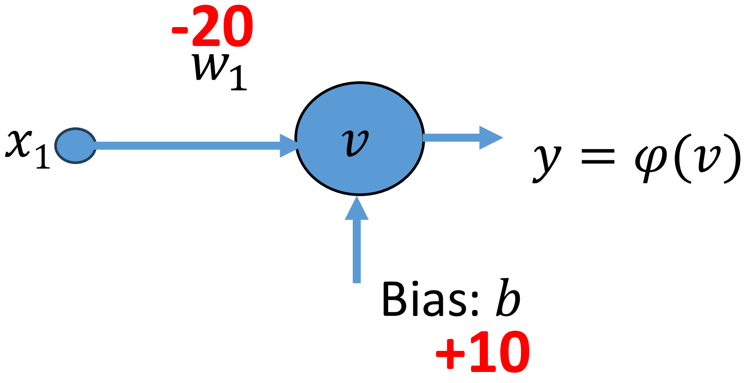
OR

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



NOT

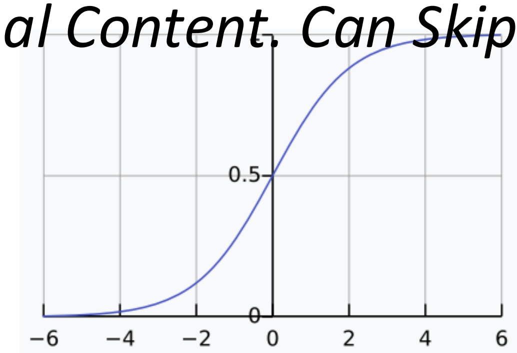
x_1	y
0	1
1	0



Credit: Andrew Ng

(Optional Content, Can Skip)

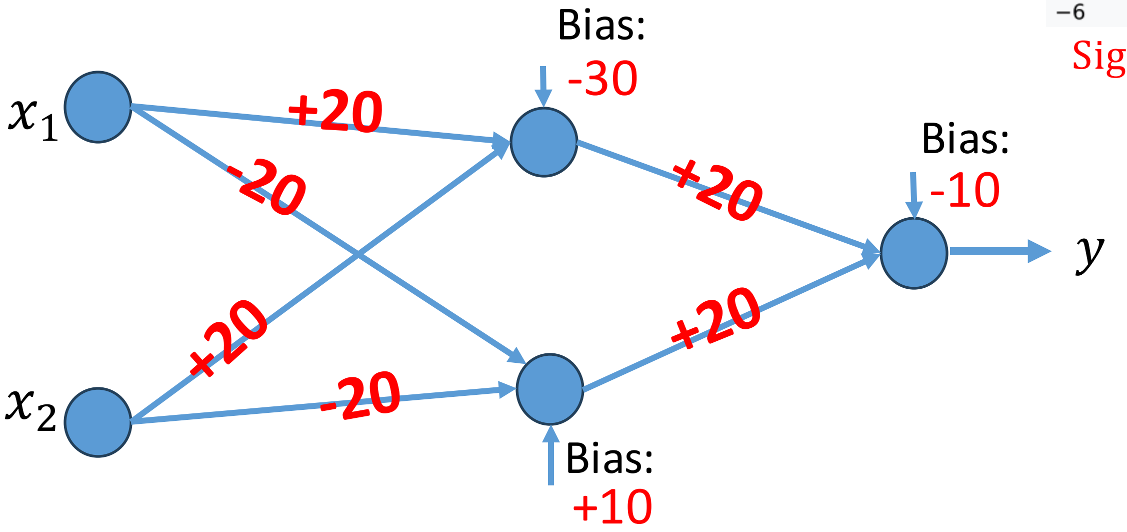
MLP for Calculating not(XOR)



Sigmoid activation function

not(XOR)

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

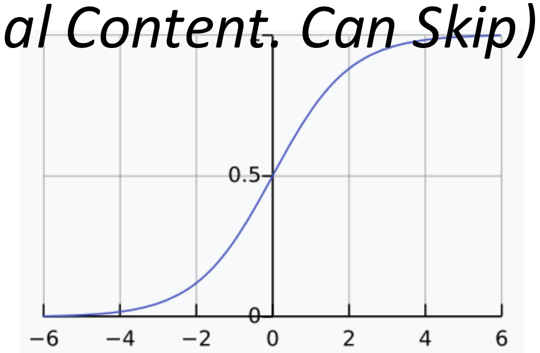


- Stacking multiple layers, we carefully designed the MLP that can handle the complex not(XOR) task.

Credit: Andrew Ng

(Optional Content, Can Skip)

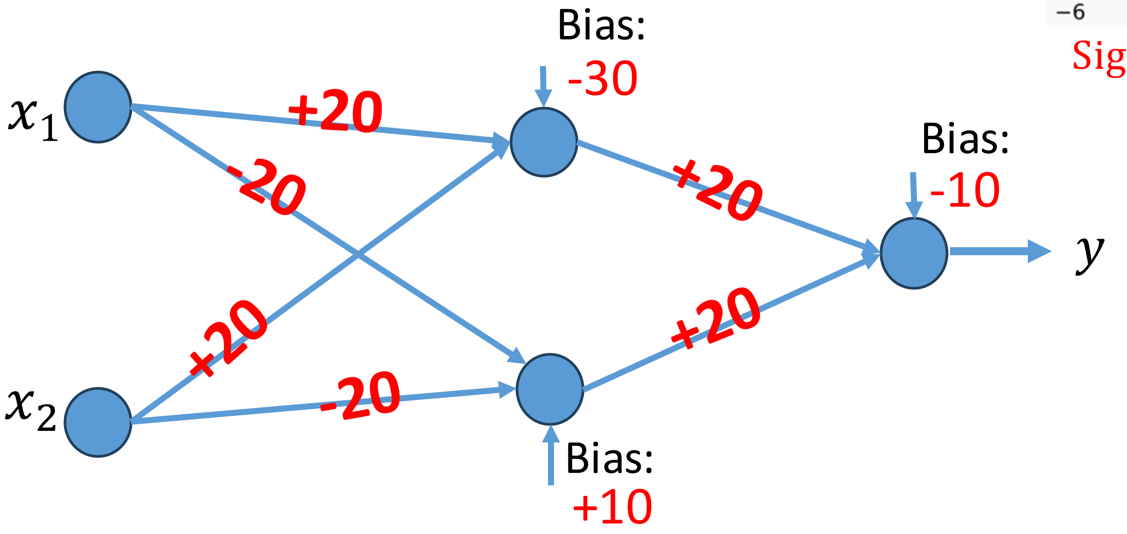
MLP for Calculating $\mathbf{not(XOR)}$



Sigmoid activation function

$\mathbf{not(XOR)}$

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



- Are you struggling in designing such a complex MLP (*finding it hard to manually design parameters*)?
  - ✓ No worries, in machine learning, we do not manually craft the parameters. Instead, we let the network automatically learn the parameters (*using the back-propagation algorithm*).

Credit: Andrew Ng

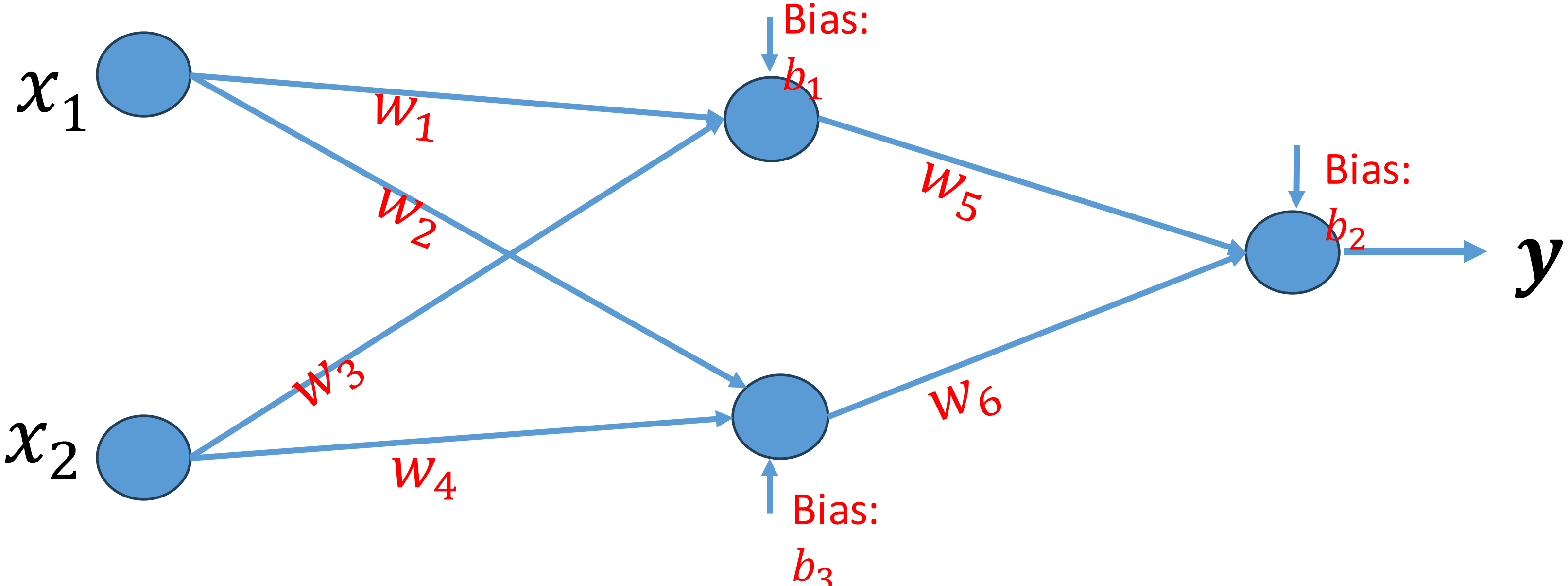
(Optional Content. Can Skip)

Learning Parameters via *Back-Propagation*

not(XOR)

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

1. Initialize all the parameters ($w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3$) to random values.
2. Forward pass: input data samples (e.g., $x_1 = 0$, $x_2 = 0$) to the neural network, and then get the output $\hat{y}$.
3. Compare $\hat{y}$ with the desired output y, and the difference between them is regarded as the error: $error = \hat{y} - y$.



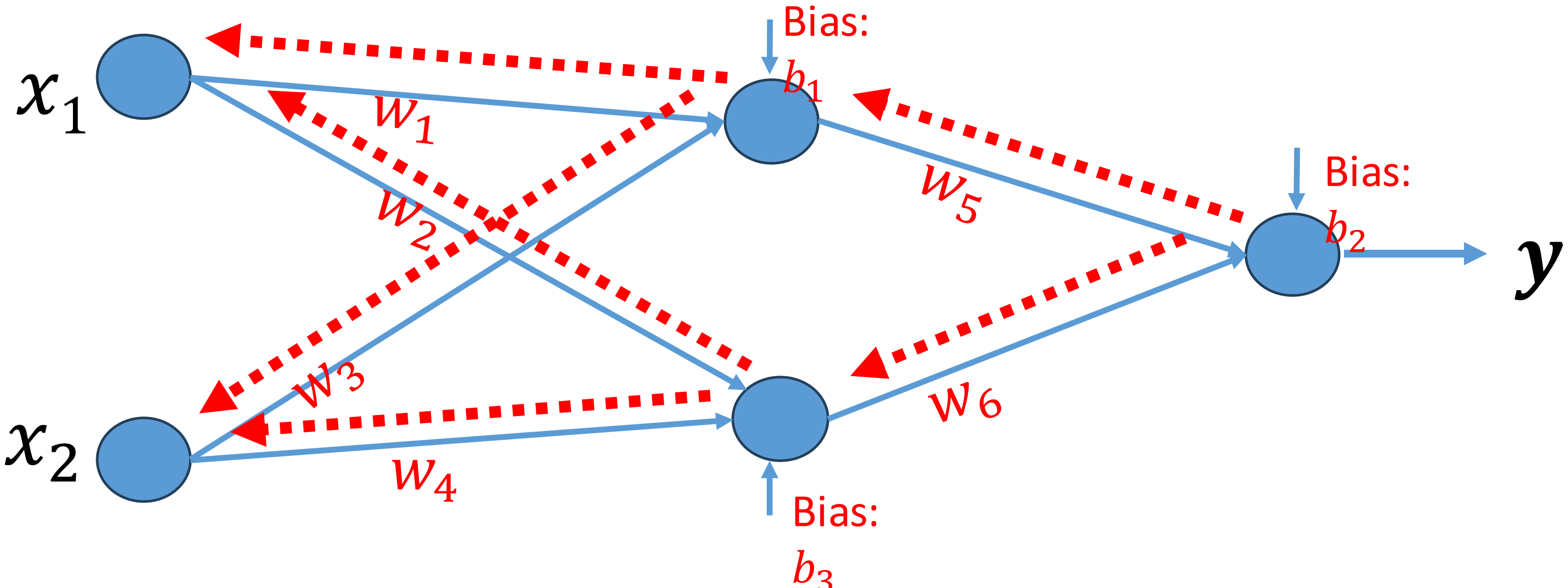
(Optional Content. Can Skip)

Learning Parameters via *Back-Propagation*

not(XOR)

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

1. Initialize all the parameters ($w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3$) to random values.
2. Forward pass: input data samples (e.g., $x_1 = 0, x_2 = 0$) to the neural network, and then get the output $\hat{y}$.
3. Compare $\hat{y}$ with the desired output y, and the difference between them is regarded as the error: $error = \hat{y} - y$.
4. **Back-propagate the error to the beginning of the network. During back-propagation, we adjust the parameters (e.g., if we find error is caused by that w_5 is too big, then we decrease w_5).**



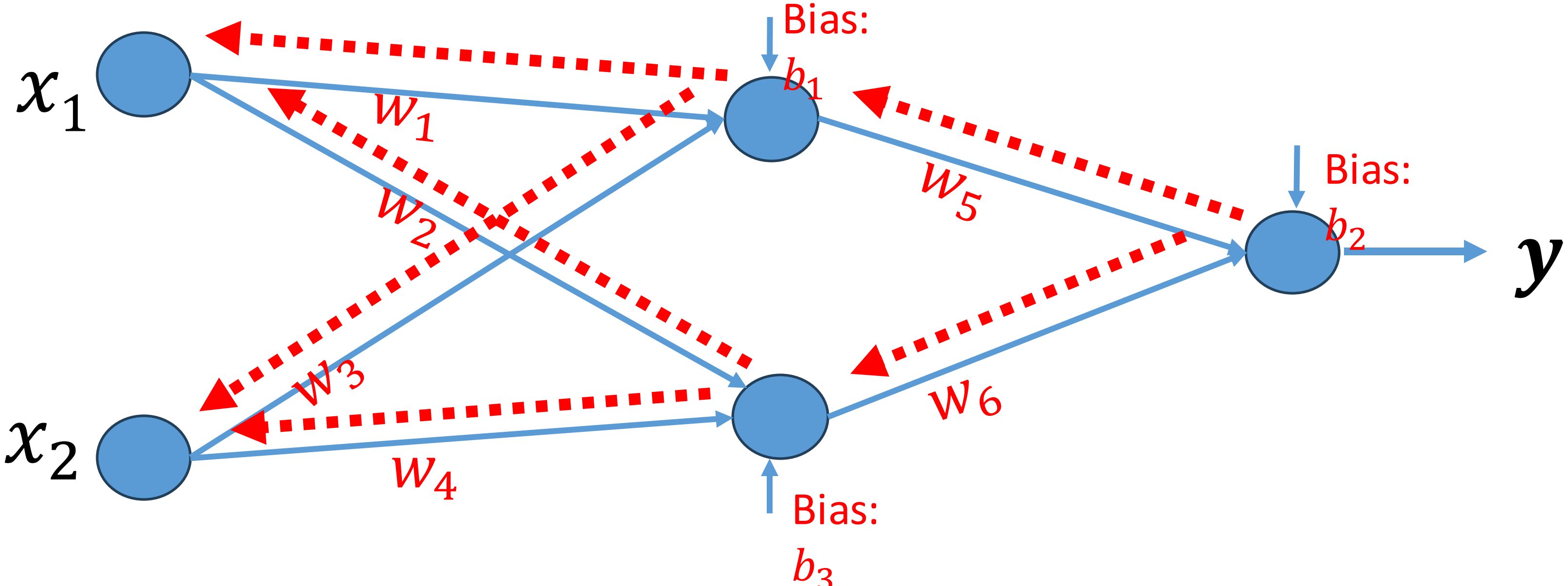
(Optional Content. Can Skip)

Learning Parameters via *Back-Propagation*

not(XOR)

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

1. Initialize all the parameters ($w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3$) to random values.
2. Forward pass: input data samples (e.g., $x_1 = 0, x_2 = 0$) to the neural network, and then get the output $\hat{y}$.
3. Compare $\hat{y}$ with the desired output y, and the difference between them is regarded as the error: $error = \hat{y} - y$.
4. Back-propagate the error to the beginning of the network. During back-propagation, we adjust the parameters (e.g., if we find error is caused by that w_5 is too big, we decrease w_5).
5. **Repeat Steps 2 and 5 multiple times. Finally, we get optimal parameters.**



(Optional Content. Can Skip)

Learning Parameters via Back-Propagation

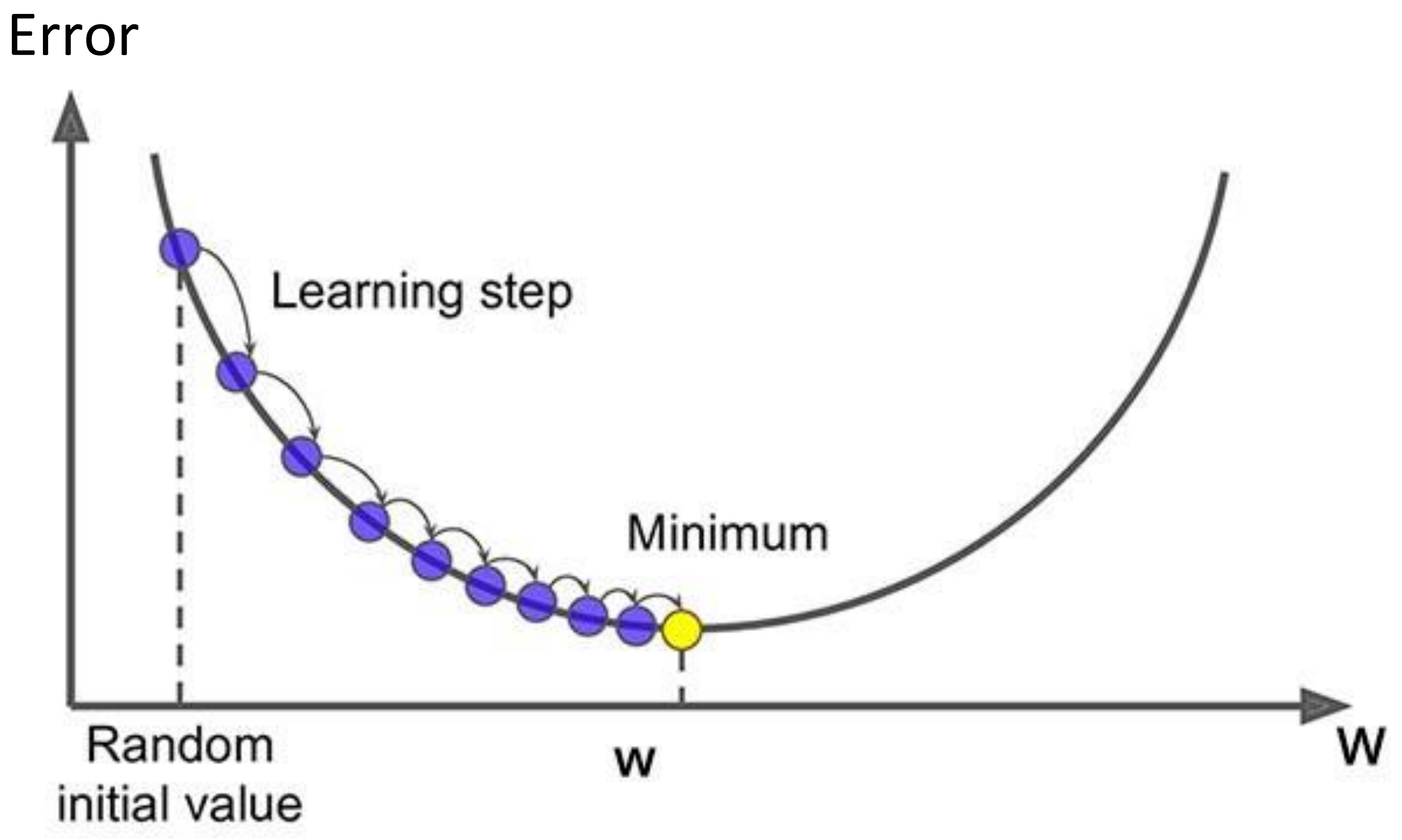


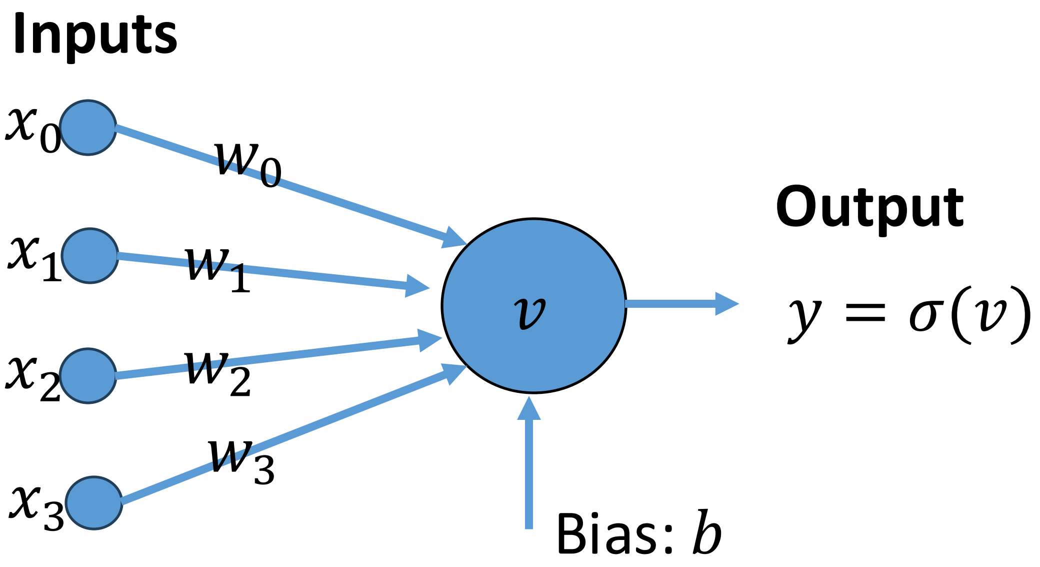
Image from https://mlpills.dev/machine-learning/gradient-descent/ (http://openaccess.uoc.edu/)

(Optional Content. Can Skip)

Learning Parameters via Back-Propagation

- Assume we have a simple network (only one layer, and the activation function is the $Sigmoid$ function).
- How can we learn the parameters w_0, w_1, w_2, w_3 and b, such that based on inputs (x_0, x_1, x_2, x_3), it can generate the expected output (y).

x_0	x_1	x_2	x_3	y
1.0	2.0	2.1	1.2	1.0
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…



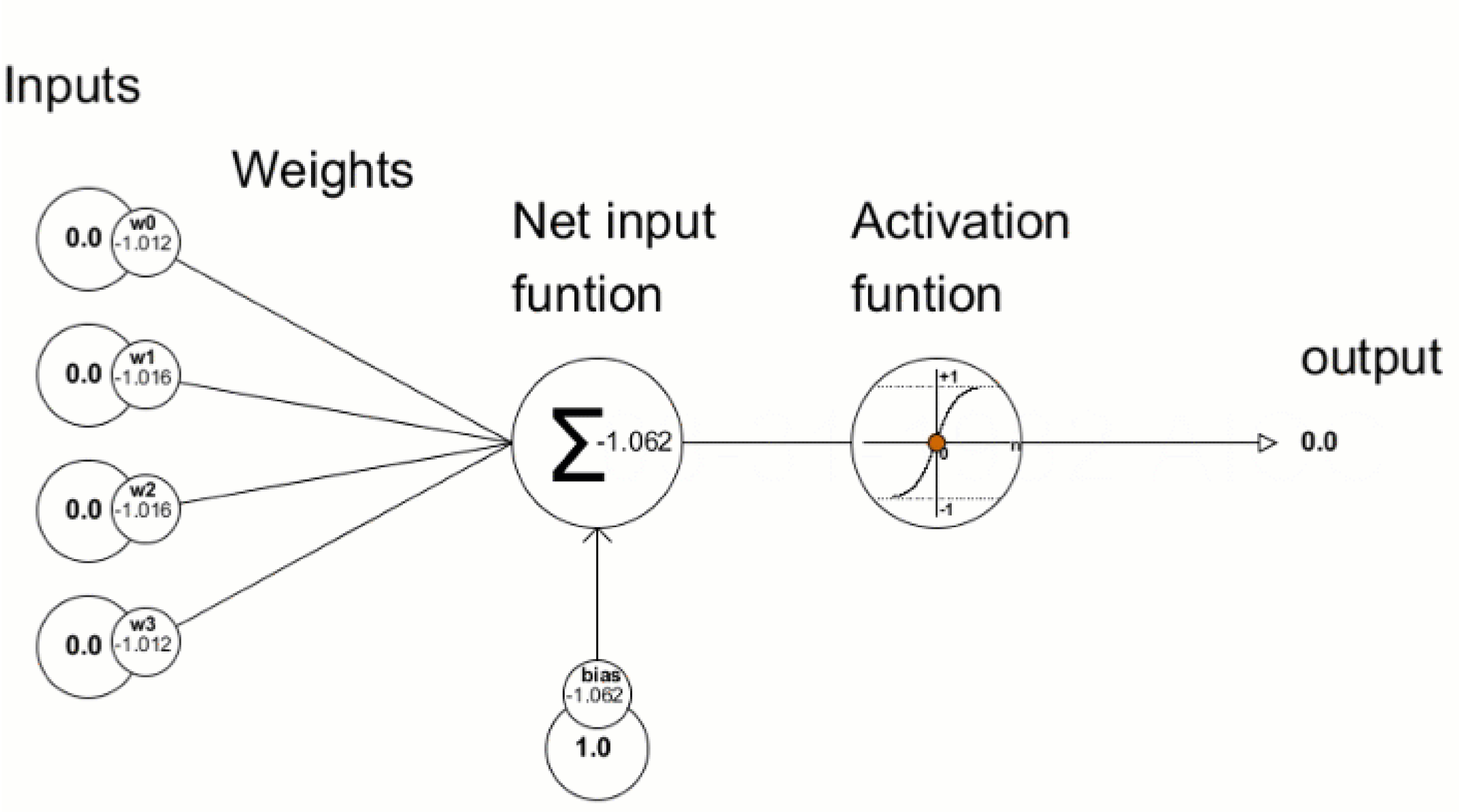
$$v = x_0 * w_0 + x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + b$$

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

(Optional Content. Can Skip)

Learning Parameters via Back-Propagation

An intuitive example



Gif from https://dotnettutorials.net/lesson/how-artificial-neural-network-work/

References

- *J. Mahmud, H.-Y. Gu, Presentation on Neural Network, CSE 634 Data Mining Techniques*
- *https://www.seas.upenn.edu/~cis5190/*