

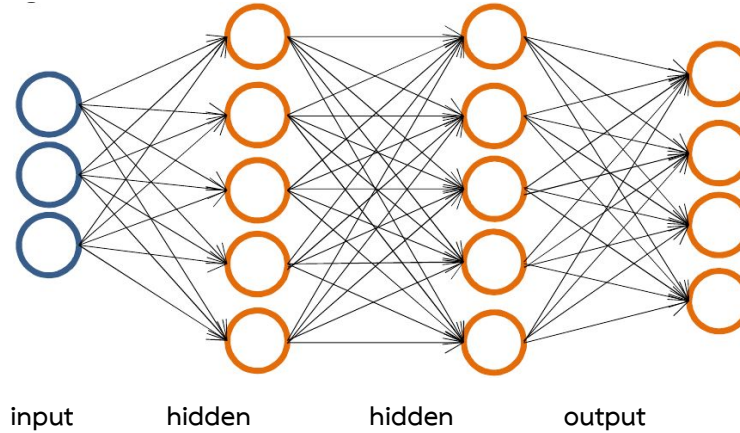
## 8. Neural Network (Part 2)

### Neural Network : Feedforward

Krittameth Teachasrisaksakul

# Recap

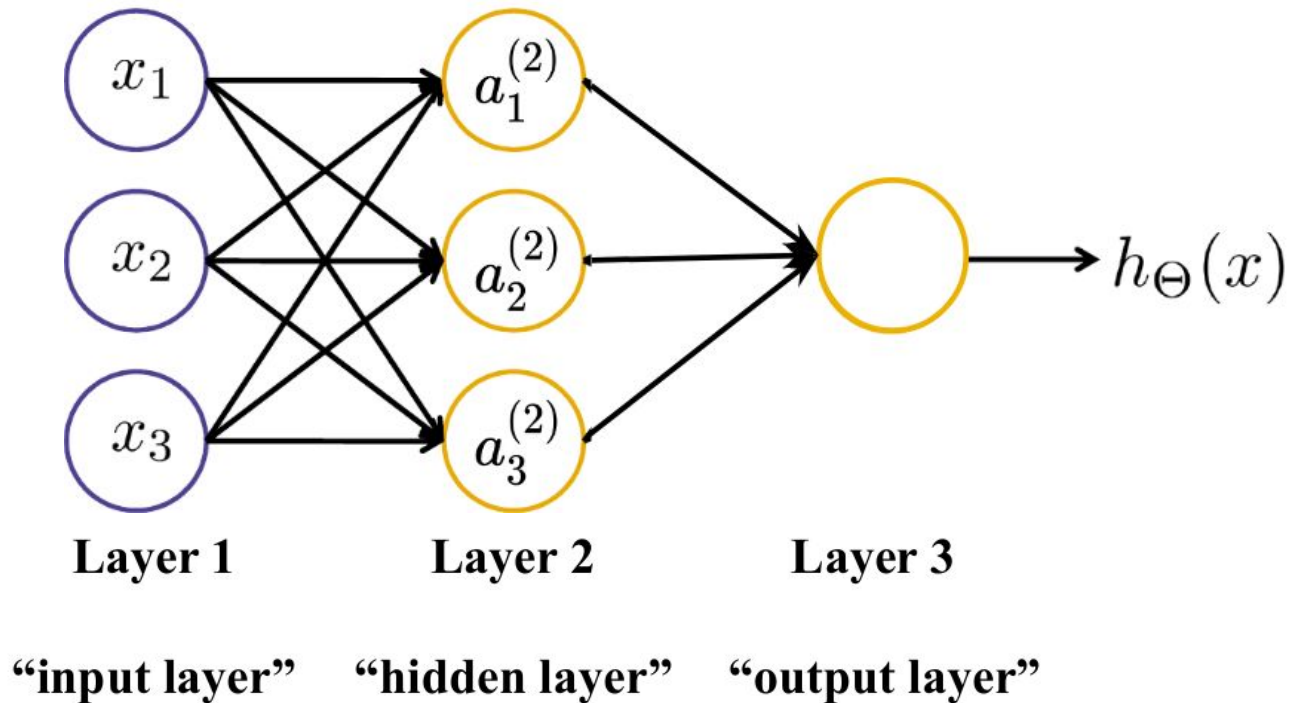
- ถ้ามี dataset (ชุดข้อมูล)  $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$  : เป้าหมายของ neural network คือ หา intermediate features ที่จะทำนาย  $\mathbf{y}^{(i)}$  แต่ละตัว ได้ดีที่สุด โดยใช้ (จาก)  $\mathbf{x}^{(i)}$  ที่สอดคล้องกัน
- บางครั้ง neural network ถูกเรียกว่า black box เพราะมันยากที่จะเข้าใจความหมายของ intermediate features ที่ network หาได้



# Neural Network: สัญลักษณ์ (Notation)

- สมมติ เรามี input layer ที่ประกอบด้วย features  $X_1, X_2, \dots$
- ใช้สัญลักษณ์ (notation)  $a_i^{(j)}$  แทน **activation** ของ หน่วย / unit ที่  $i$  ในชั้น layer ที่  $j$  เช่น
  - $a_1^{(2)}$  แทน output ของ hidden unit ที่ 1 ใน hidden layer ที่ 1
  - $a_1^{(3)}$  แทน output ของ unit ที่ 1 ใน hidden layer ที่ 2 (หรือ output layer ถ้าเป็น network ที่มีเพียง 1 hidden layer)
- เพื่อรวมสัญลักษณ์: ให้  $a_i^{(1)} = x_i$  ก็คือ มอง input เป็น layer 1

## Neural Network: สัญลักษณ์ (Notation)



# Activation Function

เราเคยเห็น ReLU network แบบง่าย แล้ว

เราสามารถมอง logistic regression เป็น neural network อย่างง่ายที่มี 1 output unit และ ไม่มี hidden unit ก็คือ

$$g(\mathbf{x}) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

สามารถเขียนสมการนี้ในรูปแบบ neural network ด้วย 2 ขั้นตอน:

1. คำนวณ linear response

2. คำนวณ activation function

$$z = \theta_0 + \sum_{i=1}^n \theta_i x_i$$

เมื่อ

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

# Activation Function

ส่วนมาก  $g(z)$  ไม่เป็นเชิงเส้น (non-linear)

activation function ที่พบทั่วไปมากที่สุด ได้แก่

1. Sigmoid :

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

2. ReLU (activation function ที่ถูกเลือกอัตโนมัติ / เป็นค่าเริ่มต้น ใน neural network สมัยใหม่):

$$g(z) = \max(z, 0)$$

3. Hyperbolic tangent:

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

# Forward Propagation (การถ่ายทอด/ส่งไปข้างหน้า)

- การคำนวณเต็มรูปแบบ มีขั้นตอน ดังนี้
- สำหรับ hidden unit แรก ใน hidden layer แรก  $\rightarrow$  คำนวณ

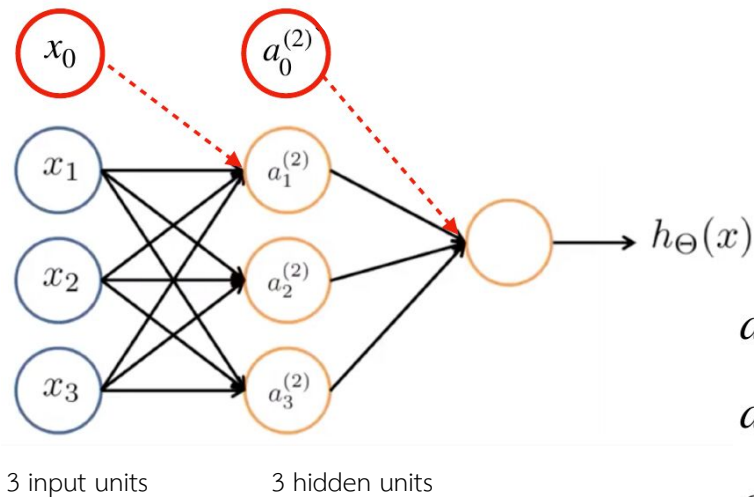
$$z_1^{(2)} = \Theta^{(1)T} \mathbf{x} + \theta_0^{(1)} \rightarrow \text{ทำหน้าที่เป็น bias ที่ให้ค่า output} \\ = 1 \text{ เสมอ}$$

และ

$$a_1^{(2)} = g(z_1^{(2)})$$

- เมื่อ  $\Theta^{(j)}$  เป็น matrix ของ parameters หรือ weights ที่ควบคุม function ที่ map (เชื่อมโยง) จาก layer  $j$  ไปยัง layer  $j+1$
- ทำขั้นตอนนี้ซ้ำกับแต่ละ unit ในแต่ละ layer เพื่อหาค่าของ output layer สุดท้าย

## Forward Propagation (การถ่ายทอด/ส่งไปข้างหน้า)



$a_i^{(j)}$  = 'activation' of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = matrix of weights controlling function mapping from layer  $j$  to layer  $j + 1$

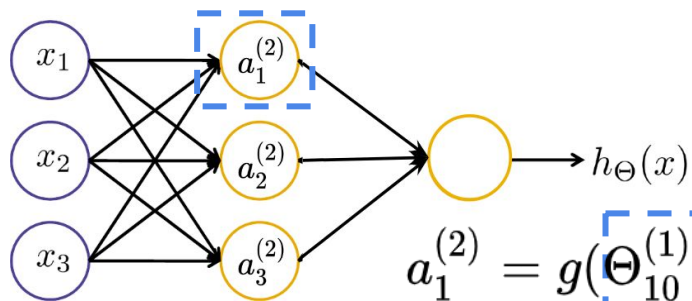
$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \end{aligned} \quad \Theta^{(i)} \in \mathbb{R}^{3 \times 4}$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

ถ้าใน network : layer  $j$  มี  $s_j$  unit (หน่วย) และ layer  $j+1$  มี  $s_{j+1}$  unit  
 แล้ว  $\Theta^{(j)}$  จะมี dimension (มิติ) เป็น  $s_{j+1} \times (s_j + 1)$



# Neural Network



$a_i^{(j)}$  = "activation" ของ unit (หน่วย)  $i$  ใน layer (ชั้น)  $j$

$\Theta^{(j)}$  = matrix ของ weights ที่ควบคุม function mapping from layer  $j$  to layer  $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$



หาค่า activation unit 1 unit ใน hidden layer โดย

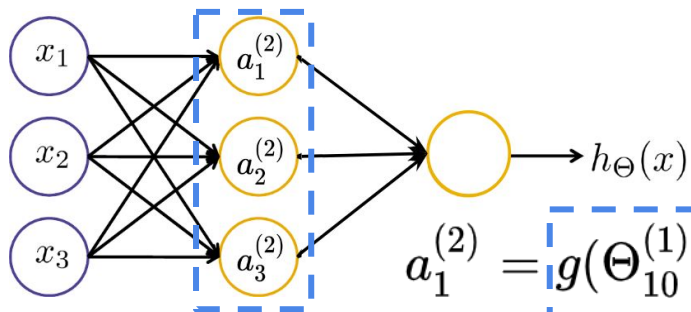
1. คูณ 1 แถวของ matrix parameters  $\Theta^{(1)}$  กับ input vector  $\mathbf{x}$

$$\begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$= \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

2. ใช้ logistic function  $g(\cdot)$  กับ ผลจาก ข้อ 1

# Neural Network



$a_i^{(j)}$  = "activation" ของ unit (หน่วย)  $i$  ใน layer (ชั้น)  $j$

$\Theta^{(j)}$  = matrix ของ weights ที่ควบคุม function mapping from layer  $j$  to layer  $j+1$

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \end{aligned}$$

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$$

- แต่ละ layer (ชั้น) จะมี matrix ของ weights  $\Theta^{(j)}$  ของมันเอง

- ถ้าใน network

- layer  $j$  มี  $s_j$  unit (หน่วย) และ
- layer  $j+1$  มี  $s_{j+1}$  unit
- แล้ว  $\Theta^{(j)}$  จะมี dimension (มิติ) เป็น  $s_{j+1} \times (s_j + 1)$

layer 1 มี 3 unit, layer 2 มี 3 unit

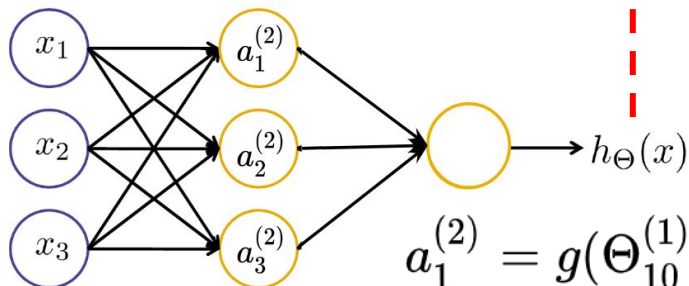
—> dimension ของ  $\Theta^{(j)}$

$$= s_{j+1} \times (s_j + 1) = 3 \times (3+1) = 3 \times 4$$

input nodes นับรวม bias node  $x_0$  ด้วย

hypothesis output

# Neural Network



$a_i^{(j)}$  = "activation" ของ unit (หน่วย)  $i$  ใน layer (ชั้น)  $j$

$\Theta^{(j)}$  = matrix ของ weights ที่ควบคุม function mapping from layer  $j$  to layer  $j+1$

layer 2 มี 3 unit, layer 3 มี 1 unit

dimension ของ  $\Theta^{(j)}$

$$= s_{j+1} \times (s_j + 1) = 1 \times (3+1) = 1 \times 4$$

$$\Theta^{(2)} \in \mathbb{R}^{1 \times 4}$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_{\Theta}^{(x)} = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$$\begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}$$

$$\begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

ค่าจาก activation nodes

Weight matrix ของ layer 2

## Question

จาก neural network ในรูป

dimension ของ  $\Theta^{(1)}$  จะเป็นเท่าไร?

$j = 1$

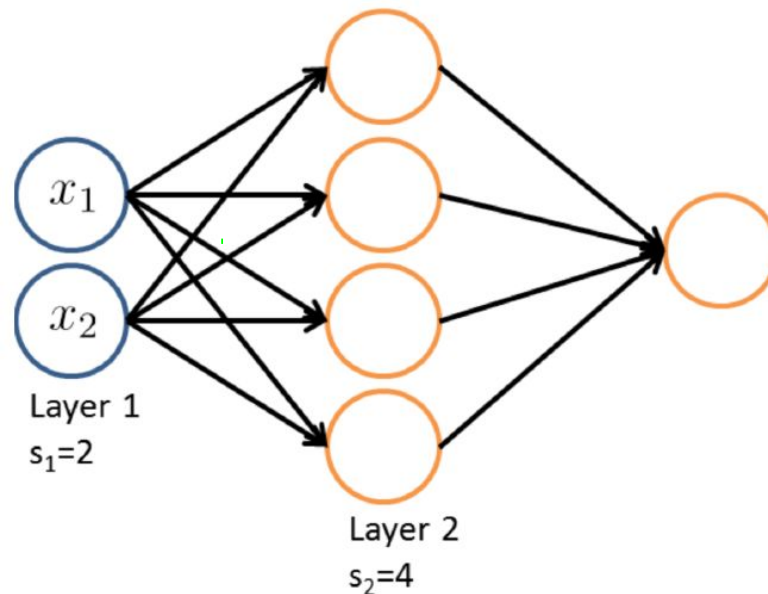
(i)  $2 \times 4$

(ii)  $4 \times 2$

(iii)  $3 \times 4$

(iv)  $4 \times 3$

$4 \times (2 -$



## Question

จาก neural network ในรูป

dimension ของ  $\Theta^{(1)}$  จะเป็นเท่าไร?

$j=1$

(i)  $2 \times 4$

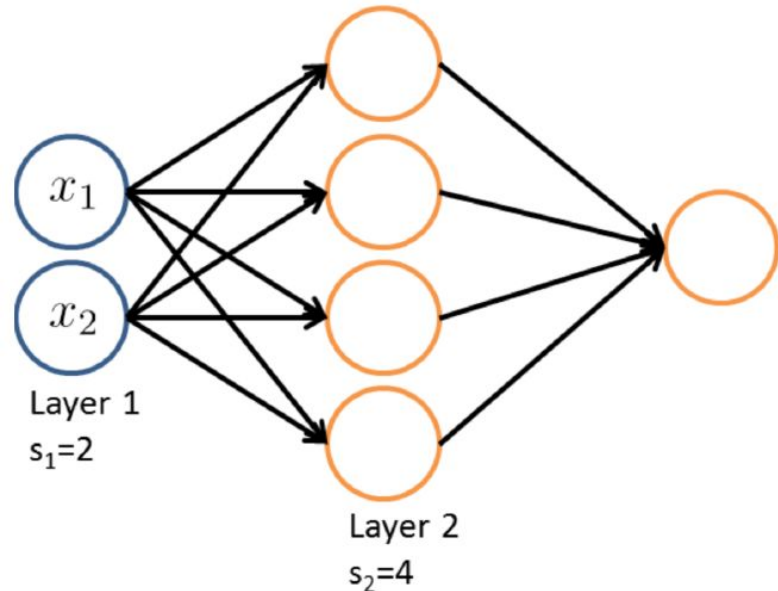
layer 1 มี 2 unit, layer 2 มี 4 unit

(ii)  $4 \times 2$

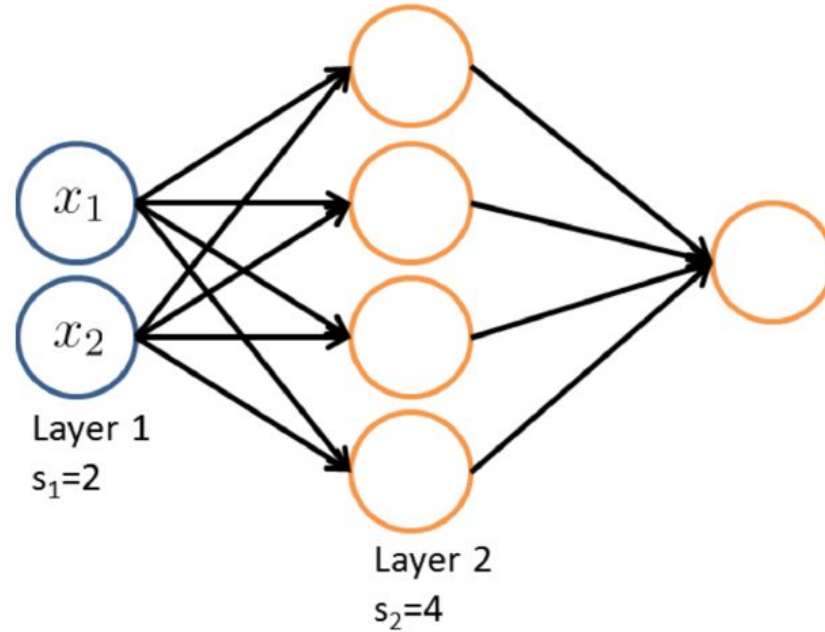
(iii)  $3 \times 4$

dimension ของ  $\Theta^{(j)}$   
 $= s_{j+1} \times (s_j + 1)$   
 $= 4 \times (2 + 1)$   
 $= 4 \times 3$

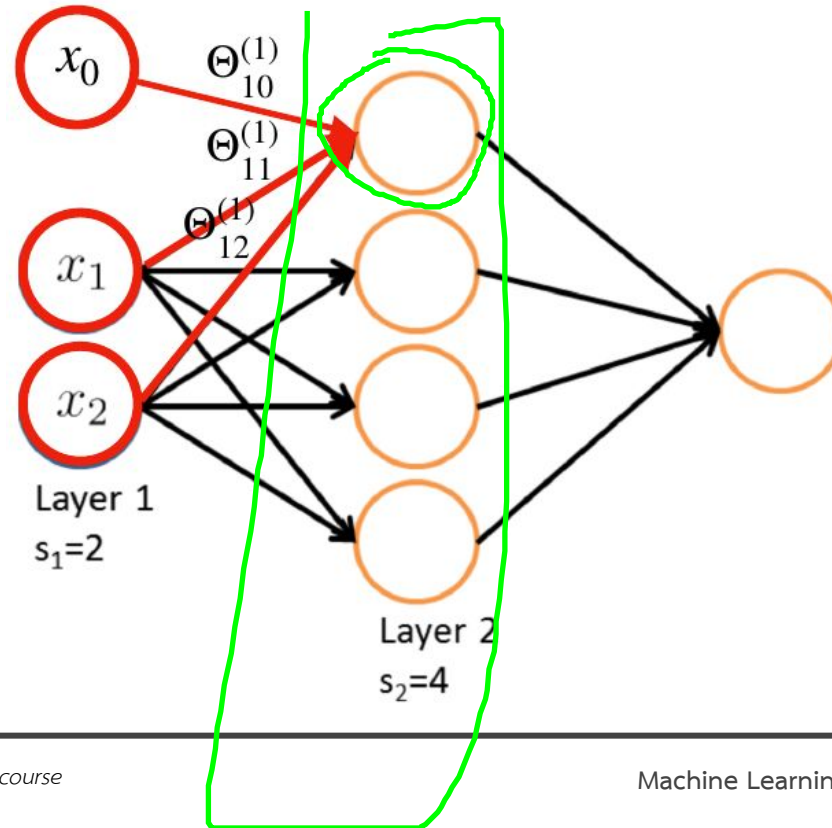
(iv)  $4 \times 3$



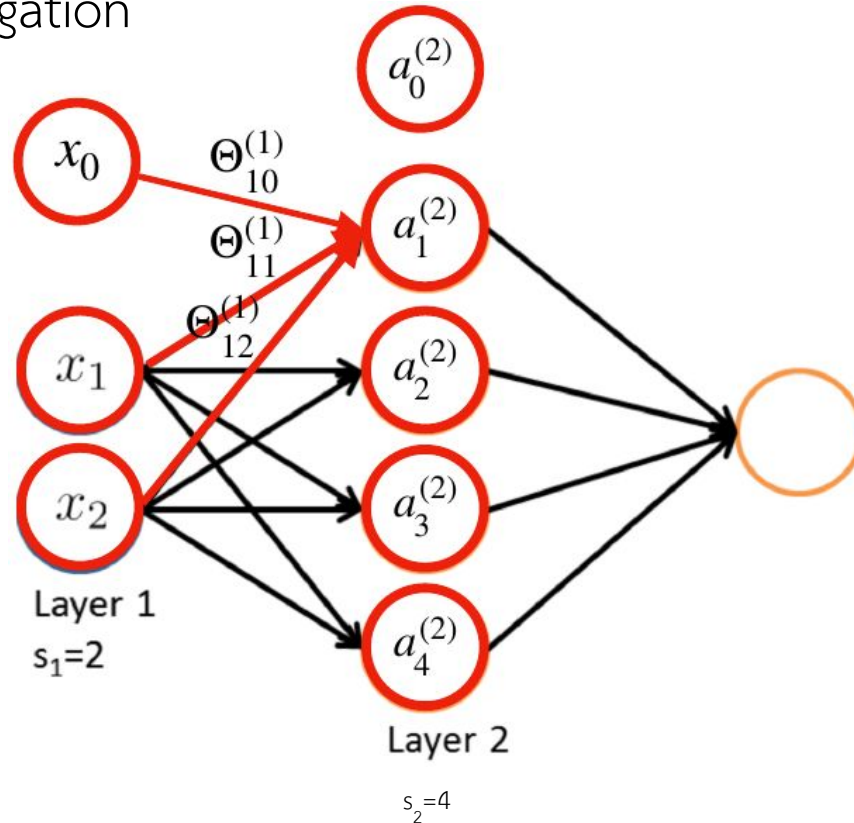
# Feedforward Propagation



# Feedforward Propagation

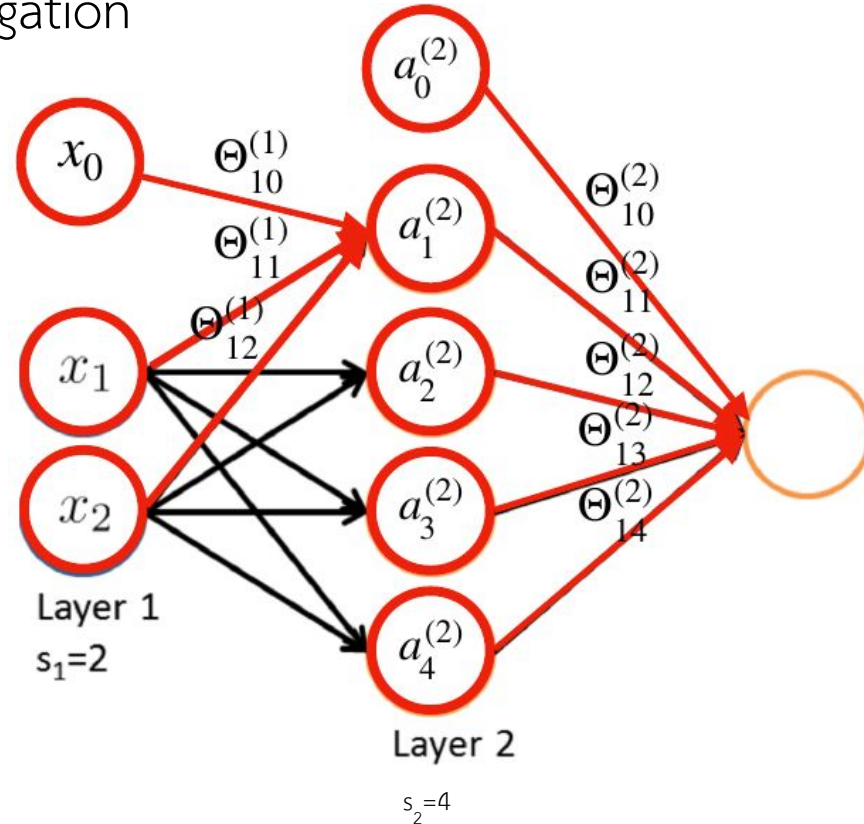


# Feedforward Propagation

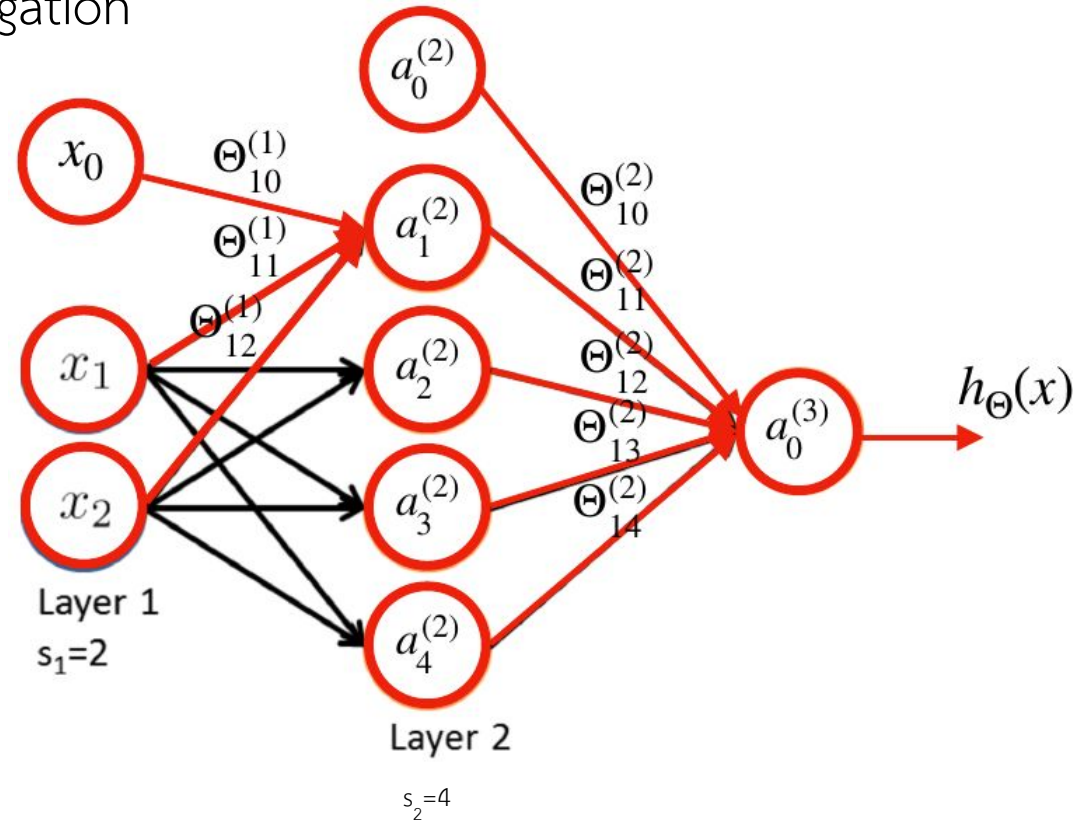




# Feedforward Propagation



# Feedforward Propagation



# Forward Propagation

## หัวข้อที่เหลือ

ตอนนี้ เราเข้าใจ การคำนวณแบบ feed-forward ของ neural network แล้วเราจะมาเรียนเรื่อง ดังนี้

- การกระทำ (execution) ที่มีประสิทธิภาพของการคำนวณแบบ feed-forward
- กระบวนการ **gradient descent** ที่ใช้เรียนรู้ **weights**  $\Theta$

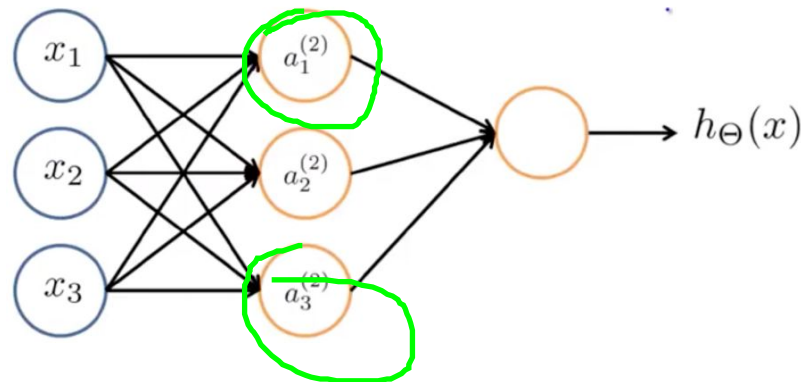
หมายเหตุ: Ng ใช้สัญลักษณ์  $\Theta$  แทน weights แต่แหล่งอ้างอิงอื่นๆบางแหล่งใช้  $W$

ดังนั้น slide นี้จะใช้ทั้งสองตัวสลับไปมา

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

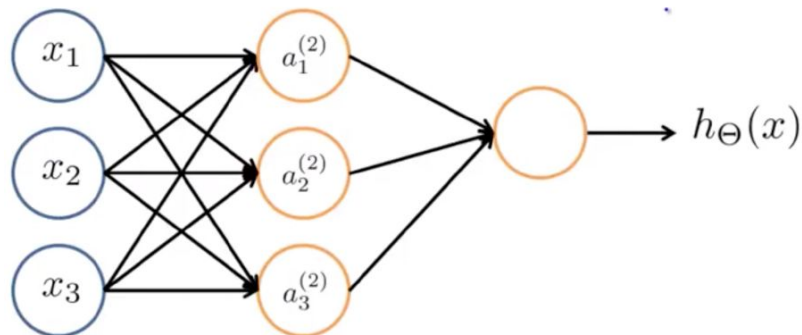
พิจารณาการคำนวณ activation ของ hidden unit จะได้ว่า

$$\begin{aligned} \mathbf{z}_1^{(2)} &= \mathbf{W}_1^{(1)T} \mathbf{x} + \mathbf{b}_1^{(1)} & \text{and} & & \mathbf{a}_1^{(2)} &= g(\mathbf{z}_1^{(2)}) \\ &\vdots & & & \vdots & \\ \mathbf{z}_4^{(2)} &= \mathbf{W}_4^{(1)T} \mathbf{x} + \mathbf{b}_4^{(1)} & \text{and} & & \mathbf{a}_4^{(2)} &= g(\mathbf{z}_4^{(2)}) \end{aligned}$$



## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

พิจารณาการคำนวณ activation ของ hidden unit จะได้



ขึ้นอยู่กับ ‘deepness’ (ความลึก) ของ architecture, สำหรับ 1 input เราอาจทำการคำนวณของ activation ของ 100 หรือ 1,000 unit

Code ที่ทำการคำนวณนี้ใช้ ‘for’ loop และสิ่งที่คล้ายกันจะ run **อย่างช้ามาก** โดยเฉพาะถ้า implement (เขียนโปรแกรม) เป็น bytecode ด้วยภาษา Python หรือ Java

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

สิ่งที่ต้องมี คือ ความสามารถที่จะทำ matrix algebra (พีชคณิตเมทริกซ์) ด้วยการเรียก (call) หรือคำสั่ง library 1 อัน ที่ highly optimized โดยใช้คำสั่ง CPU (CPU instruction) ที่ถูกนำมาเพื่อ vector operation

BLAS เป็น library ที่โด่งดัง ซึ่งทำสิ่งนี้ และ numpy ก็สร้างโดยใช้ library นี้

อีกวิธีหนึ่ง คือ อาจ parallelize computation (ทำการคำนวณขนาน/พร้อมกัน) โดยใช้ GPU resources (ทรัพยากร) (GPU : graphics processing unit = หน่วยประมวลผลกราฟฟิกส์)

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

ถ้าจะทำการคำนวณในรูปแบบ vector (vectorized computation) สำหรับทั้ง layer → ต้องทำ operation ใน operation เดียว:

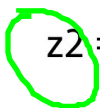
$$\begin{aligned}a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)\end{aligned}$$

สามารถ implement ได้ด้วย Python



```
W1 = np.matrix(np.random.normal(0, 1, (3, 4)))
```

```
x = np.matrix(np.random.normal(0, 1, (4, 1)))
```



```
z2 = np.dot(W1, x)
```



คำสั่งชุดนี้จะ run เร็วกว่า for loop  
ที่ซ้อน 2 ชั้นมาก

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

ถ้าจะทำการคำนวณในรูปแบบ vector (vectorized computation) สำหรับทั้ง layer  $\rightarrow$  ต้องทำ operation ใน operation เดียว:

$$\mathbf{z}_1^{(2)} = \mathbf{w}_1^{(1)T} \mathbf{x} + \mathbf{b}_1^{(1)}$$

สามารถ implement ได้ด้วย Python statement เดียว

```
W1 = np.matrix(np.random.normal(0, 1, (3, 4)))  
x = np.matrix(np.random.normal(0, 1, (4, 1)))  
z2 = np.dot(W1, x)
```

คำสั่งชุดนี้จะ run เร็วกว่า for loop  
ที่ซ้อน 2 ชั้นมาก



## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

เพื่อคำนวณ  $a^{(2)}$  ด้วย vector operation เราสามารถใช้ vectorized functions (function ที่อยู่ในรูป vector) ใน implementation ของเรา

ตัวอย่าง: ถ้าเรามี sigmoid function เรา implement:

เป็น

$$g(z^{(2)}) = \frac{1}{1 + e^{-z^{(2)}}}$$

```
def g(z):  
    return 1 / (1 + exp(-z))
```

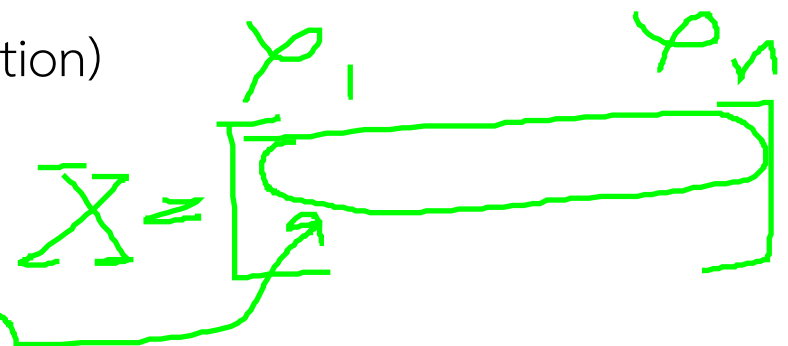
```
z2 = np.matrix([[1, 2, 3]]).T  
a2 = g(z2)
```

นี้จะ run ได้เร็วกว่า executing `exp()` function ใน  
Python loop มาก

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

คำนวณโดยใช้ training examples หลายๆ ตัว

ใช้  $\mathbf{x}^{(i)}$  ของแต่ละ training example  $i$ :

$$\mathbf{z}_1^{(2)} = \mathbf{W}_1^{(1)T} \mathbf{x}^{(i)} + \mathbf{b}_1^{(1)}$$


ถ้าทำ operation นี้ใน loop ซ้ำๆ จะช้ากว่าใช้ vectorized operation ( $X$  เป็น vector):

$$\mathbf{z}_1^{(2)} = \mathbf{W}_1^{(1)T} X + \mathbf{b}_1^{(1)}$$

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

ภาษาบางภาษา เช่น Python จะอนุญาตให้ **broadcast** การบวก (addition operation) ในแนวนอน:

(broadcast = ทำสิ่งเดิมซ้ำๆ กับสมาชิกทุกตัว)

```
A = np.matrix([[1, 2, 3], [2, 3, 4]])
```

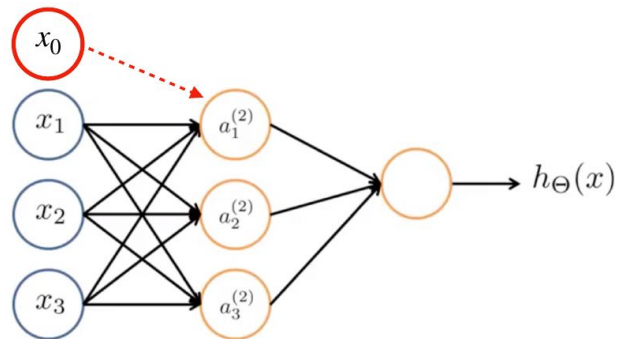
```
b = np.matrix([[2, 3]]).T
```

```
>> A + b
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 5 \\ 5 & 6 & 7 \end{bmatrix}$$

ด้วย broadcasting การคำนวณจะมีประสิทธิภาพมากกว่า for loop มาก

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

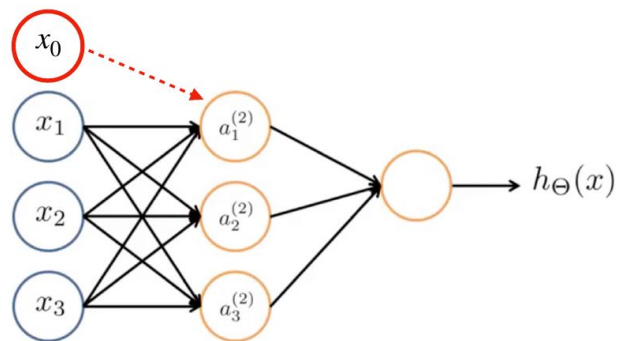


$$a_1^{(2)} = g(z_1^{(2)})$$

$z_i^{(j)}$  : ผลรวมเชิงเส้นแบบถ่วงน้ำหนัก (weighted linear combination) ของ inputs ที่ถูกส่งไปที่ neuron:

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \rightarrow z_1^{(2)} \\ a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \rightarrow z_2^{(2)} \\ a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \rightarrow z_3^{(2)} \\ h_{\Theta}(x) &= a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}) \end{aligned}$$

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)

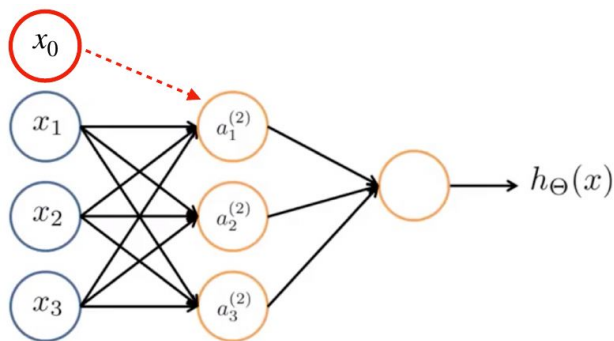


เช่น  $a_1^{(2)} = g(z_1^{(2)})$

$z_i^{(j)}$  : ผลรวมเชิงเส้นแบบถ่วงน้ำหนัก (weighted linear combination) ของ inputs ที่ถูกส่งไปที่ neuron:

$$\left. \begin{aligned} z_1^{(2)} &= \Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3 \\ z_2^{(2)} &= \Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3 \\ z_3^{(2)} &= \Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3 \end{aligned} \right\} \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

## การคำนวณที่มีประสิทธิภาพ (Efficient computation)



$$a_0^{(2)} = 1 \Rightarrow a^{(2)} \in \mathbb{R}^4$$

$$\therefore z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$$

เช่น  $a_1^{(2)} = g(z_1^{(2)})$

$z_i^{(j)}$  : ผลรวมเชิงเส้นแบบถ่วงน้ำหนัก (weighted linear combination) ของ inputs ที่ถูกส่งไปที่ neuron:

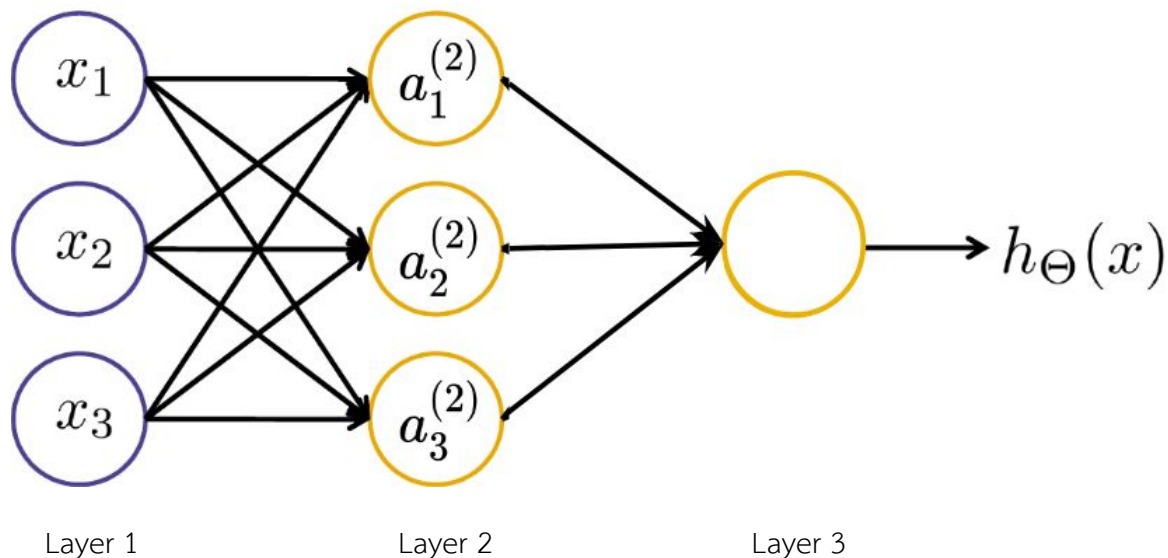
$$\left. \begin{aligned} z_1^{(2)} &= \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \\ z_2^{(2)} &= \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \\ z_3^{(2)} &= \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \end{aligned} \right\} \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

i.e.  $z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$

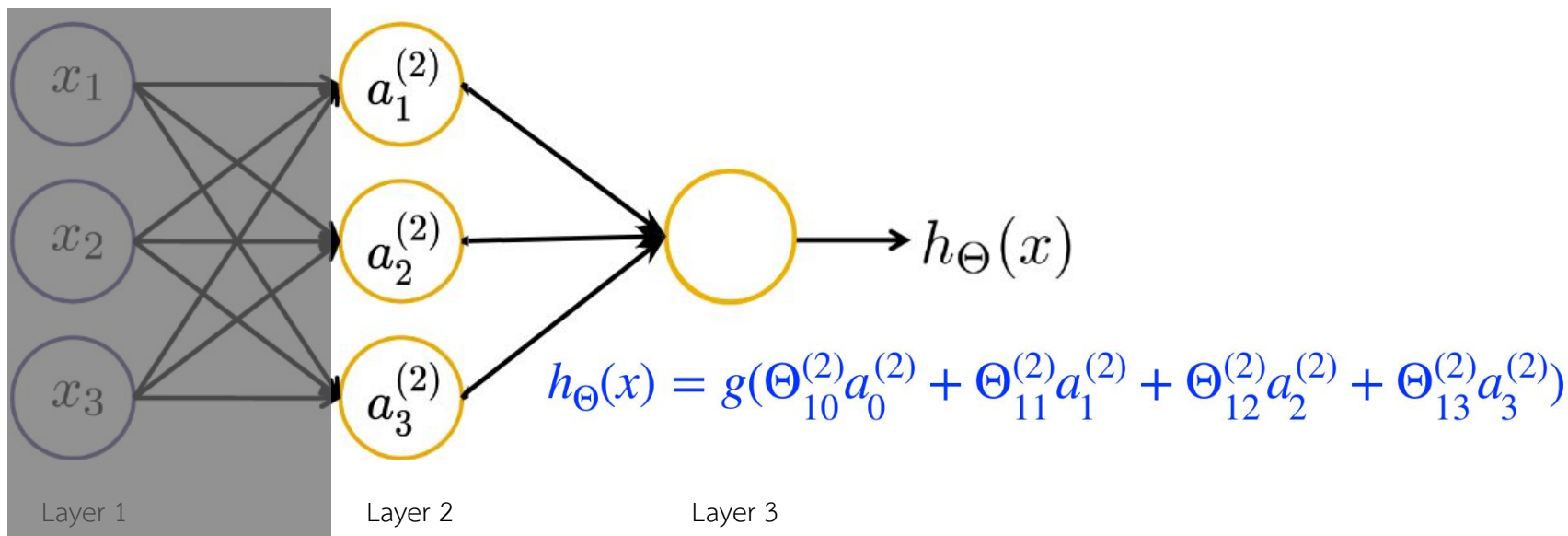
# Forward Propagation

ดังกล่าวข้างต้น ถ้า network ไม่มี hidden unit และขึ้นอยู่กับชนิด activation function : network อาจกลายเป็นการทำ regression



# Forward Propagation

ดังกล่าวก่อนหน้านี้ ถ้า network ไม่มี hidden unit และขึ้นอยู่กับชนิด activation function : network อาจกลายเป็นการทำ regression

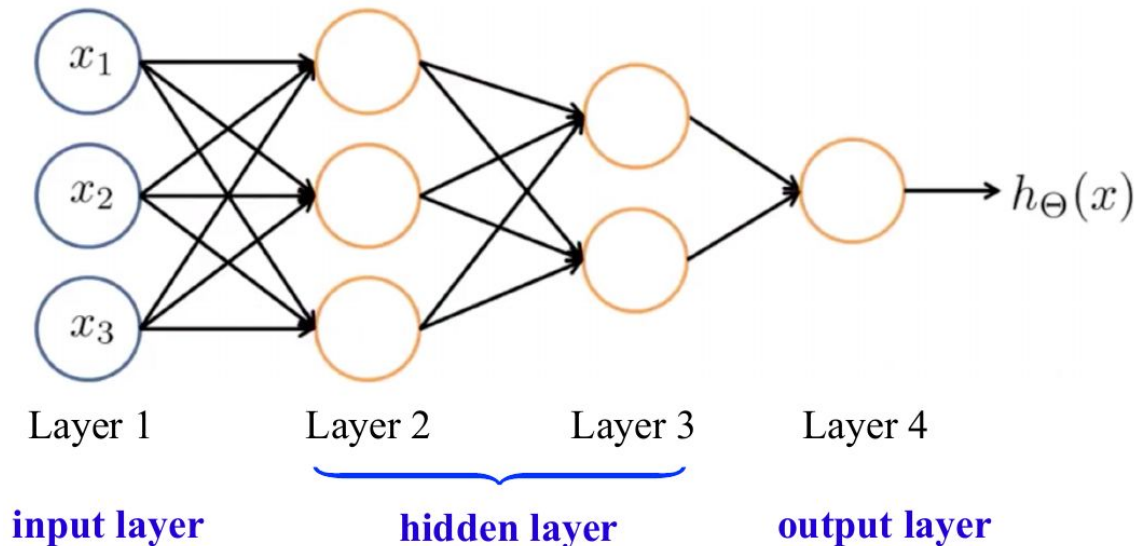




# Network Architecture อื่น

คำว่า **architecture** เกี่ยวกับการที่ neuron ต่างๆ ต่อเข้าด้วยกันอย่างไร

(เช่น จำนวน layer, จำนวน unit, ชนิดของ unit, connectivity ระหว่าง unit)



## Question

พิจารณา network ให้  $a^{(1)} = x \in \mathbb{R}^{n+1}$  แทน input เมื่อ  $a_0^{(1)} = 1$

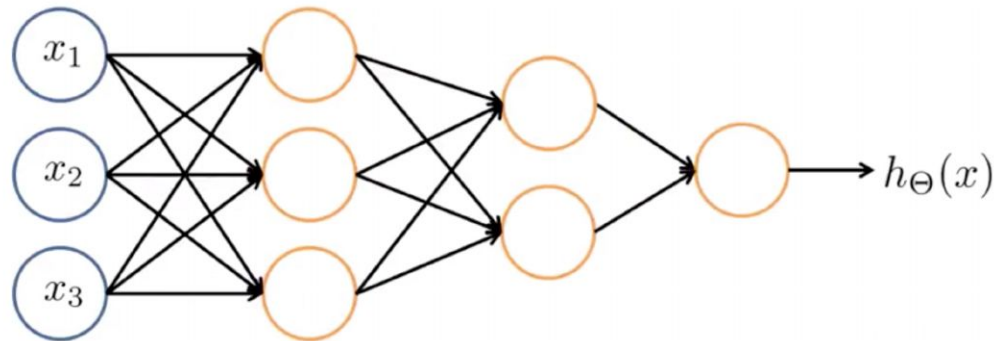
จะคำนวณ  $a^{(2)}$  ได้อย่างไร?

(i)  $a^{(2)} = \Theta^1 a^{(1)}$

(ii)  $z^{(2)} = \Theta^2 a^{(1)}; a^{(2)} = g(z^{(2)})$

(iii)  $z^{(2)} = \Theta^1 a^{(1)}; a^{(2)} = g(z^{(2)})$

(iv)  $z^{(2)} = \Theta^2 g(a^{(1)}); a^{(2)} = g(z^{(2)})$

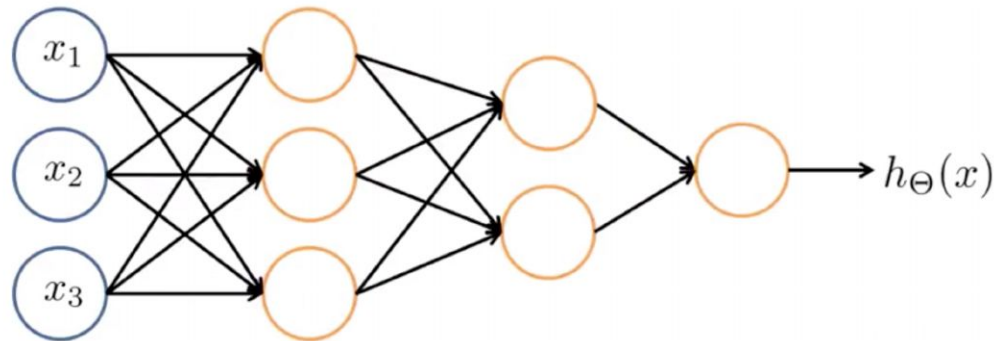


## Question

พิจารณา network ให้  $a^{(1)} = x \in \mathbb{R}^{n+1}$  แทน input เมื่อ  $a_0^{(1)} = 1$

จะคำนวณ  $a^{(2)}$  ได้อย่างไร?

- (i)  $a^{(2)} = \Theta^1 a^{(1)}$
- (ii)  $z^{(2)} = \Theta^2 a^{(1)}; a^{(2)} = g(z^{(2)})$
- (iii)  $z^{(2)} = \Theta^1 a^{(1)}; a^{(2)} = g(z^{(2)})$
- (iv)  $z^{(2)} = \Theta^2 g(a^{(1)}); a^{(2)} = g(z^{(2)})$



# References

1. Andrew Ng, Machine Learning, Coursera.
2. Teeradaj Racharak, AI Practical Development Bootcamp.
3. What is Machine Learning?, <https://www.digitalskill.org/contents/5>