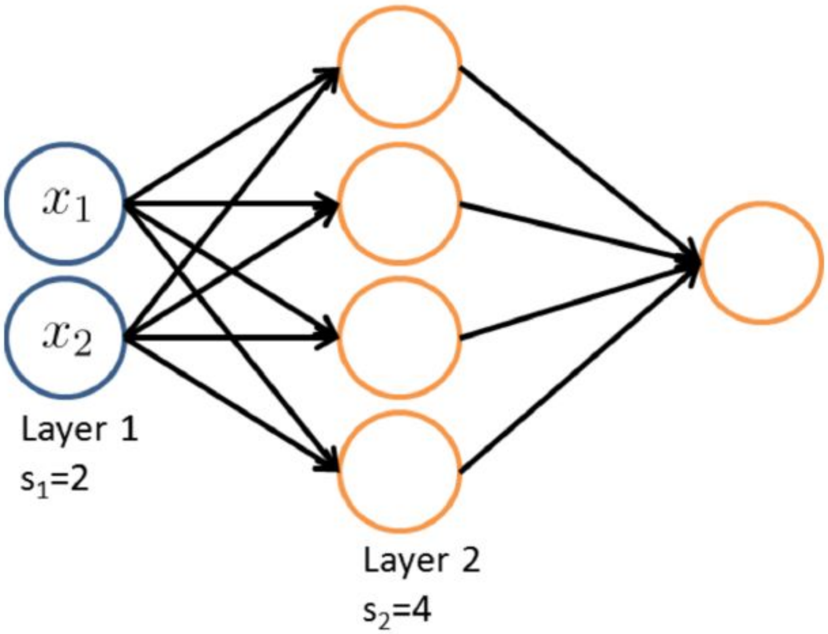
# 8.2 Neural Network (Part 3)

## Neural Network : Learning

Krittameth Teachasrisaksakul

# Feedforward Propagation



# Feedforward Propagation

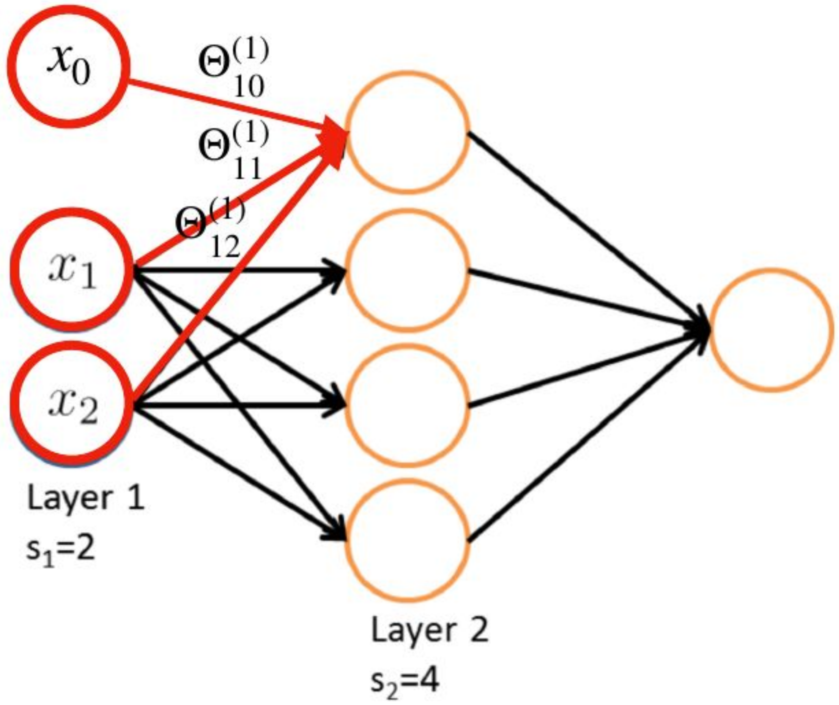
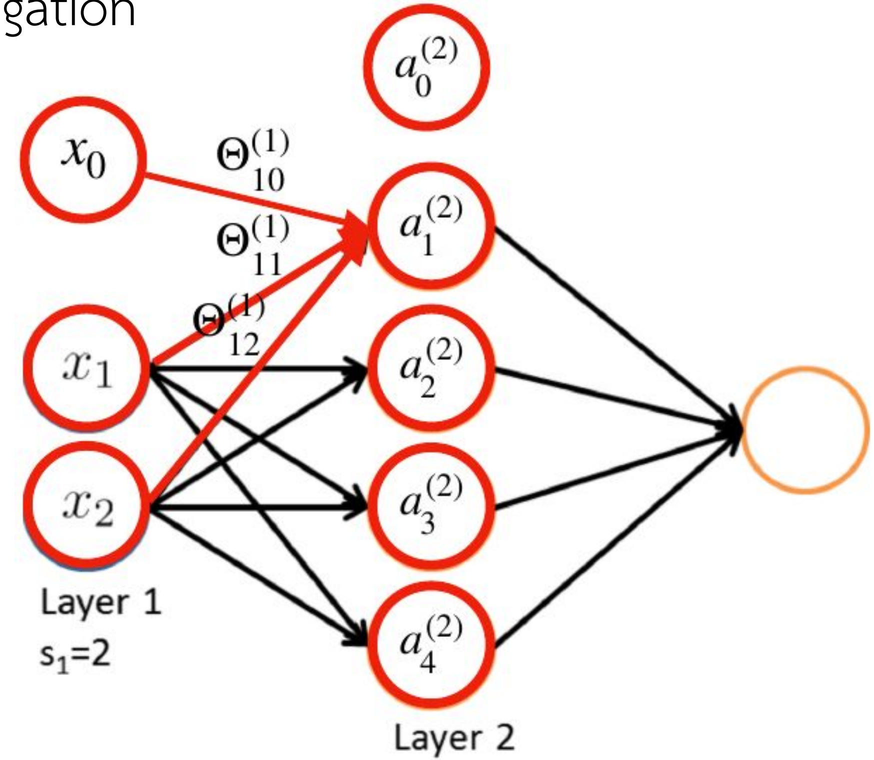



*Figure source: Andrew Ng's Machine Learning course*

# Feedforward Propagation




$s_2$=4

*Figure source: Andrew Ng's Machine Learning course*

# Feedforward Propagation

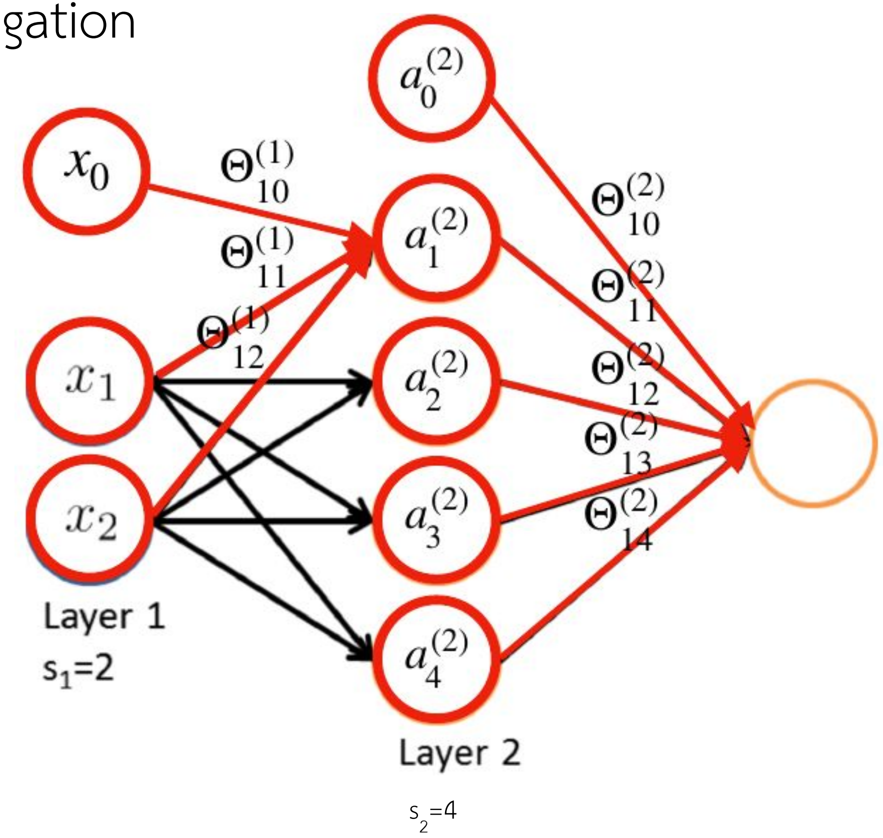
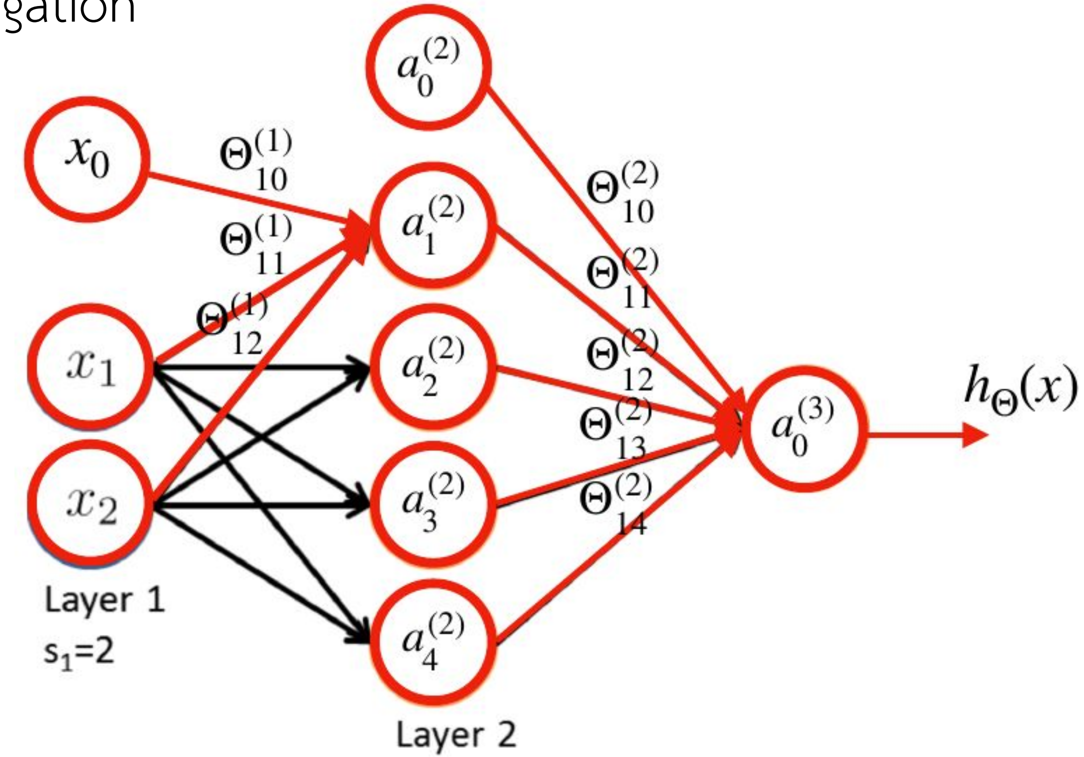



*Figure source: Andrew Ng's Machine Learning course*

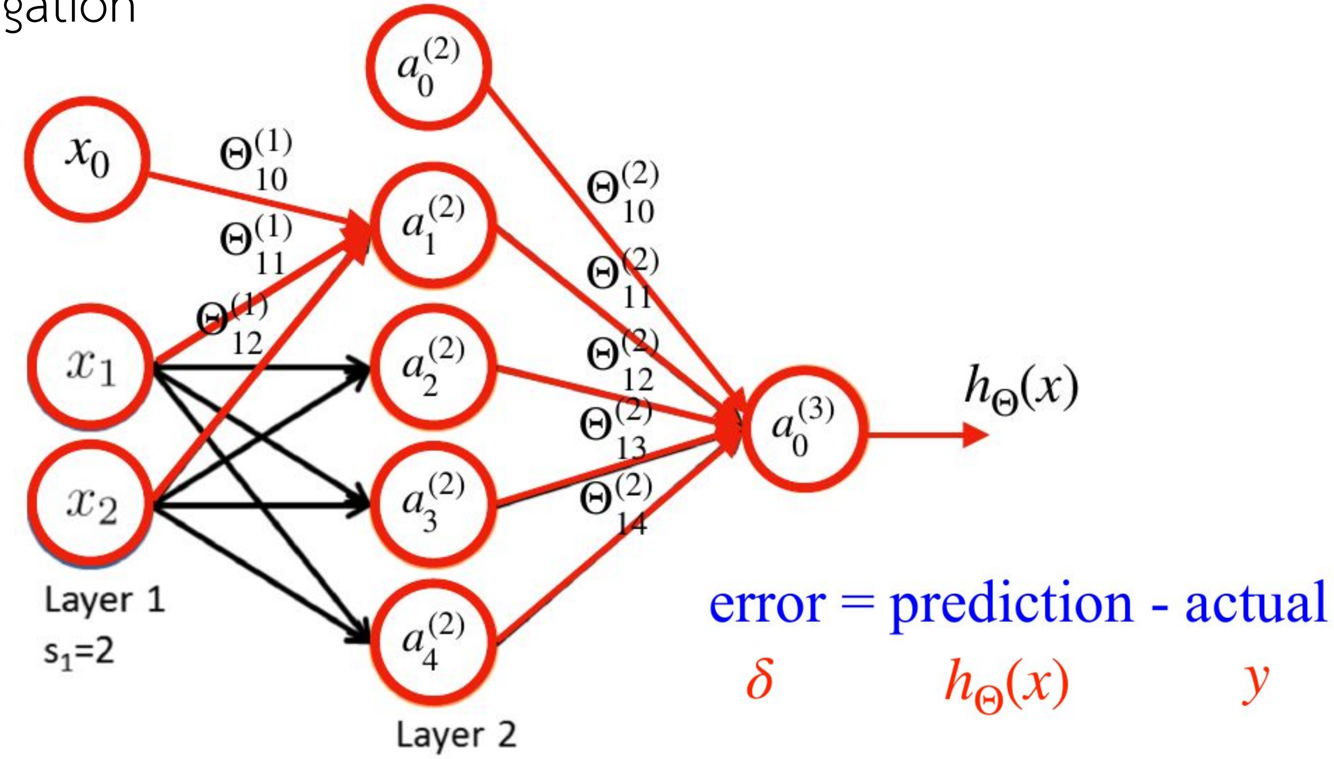
# Feedforward Propagation




$s_2$=4

*Figure source: Andrew Ng's Machine Learning course*

# Feedforward Propagation




$s_2$=4

Figure source: Andrew Ng's Machine Learning course

# Neural Network : Learning

## Learning via Backpropagation

(การเรียนรู้ด้วย Backpropagation)

Krittameth Teachasrisaksakul

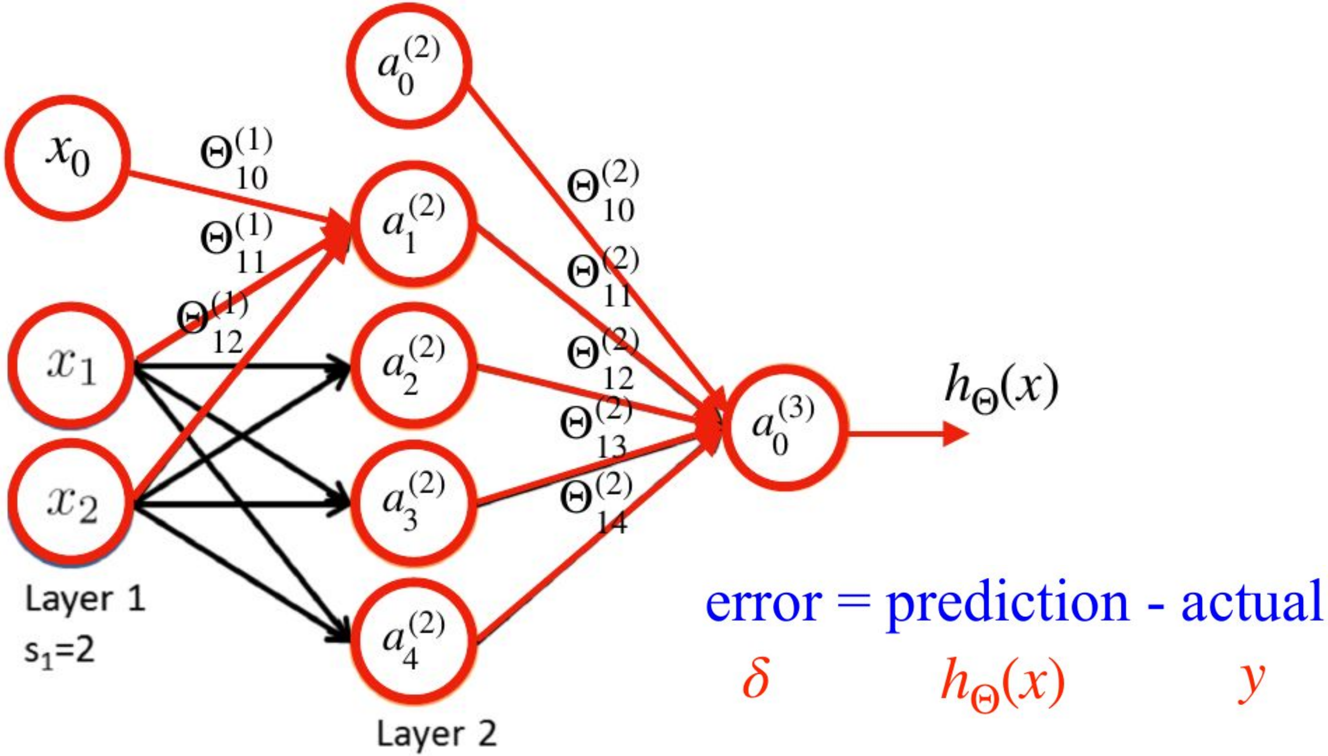
# Backpropagation

- Backpropagation ย่อมาจาก ‘the backward propagation of errors’ (การถ่ายทอด error จากหลังไปหน้า) เพราะ error ถูกคำนวณที่ output และถูกถ่ายทอด จากหลังไปหน้าจนครบทุก layer ของ network
- ถูกคิดค้นโดยนักวิจัยหลายคน ในช่วงต้นยุค 1960 และถูกเสนอครั้งแรกโดย Werbos เพื่อใช้ใน neural network ในวิทยานิพนธ์ปริญญาเอกของเขา (ค.ศ. 1974)
- เกี่ยวข้องอย่างใกล้ชิดกับ *Gauss-Newton* algorithm และเป็นส่วนหนึ่งของการวิจัยที่ต่อเนื่องในเรื่อง neural backpropagation
- ในบริบทของการเรียนรู้ (learning) backpropagation ถูกใช้ทั่วไปโดย gradient descent optimization algorithm เพื่อปรับค่า weight ของ neuron โดยคำนวณ gradient ของ cost function

Paul Werbos

*Figure source: Andrew Ng’s Machine Learning course*

# Backpropagation




$s_2$=4

*Figure source: Andrew Ng's Machine Learning course*

# Backpropagation

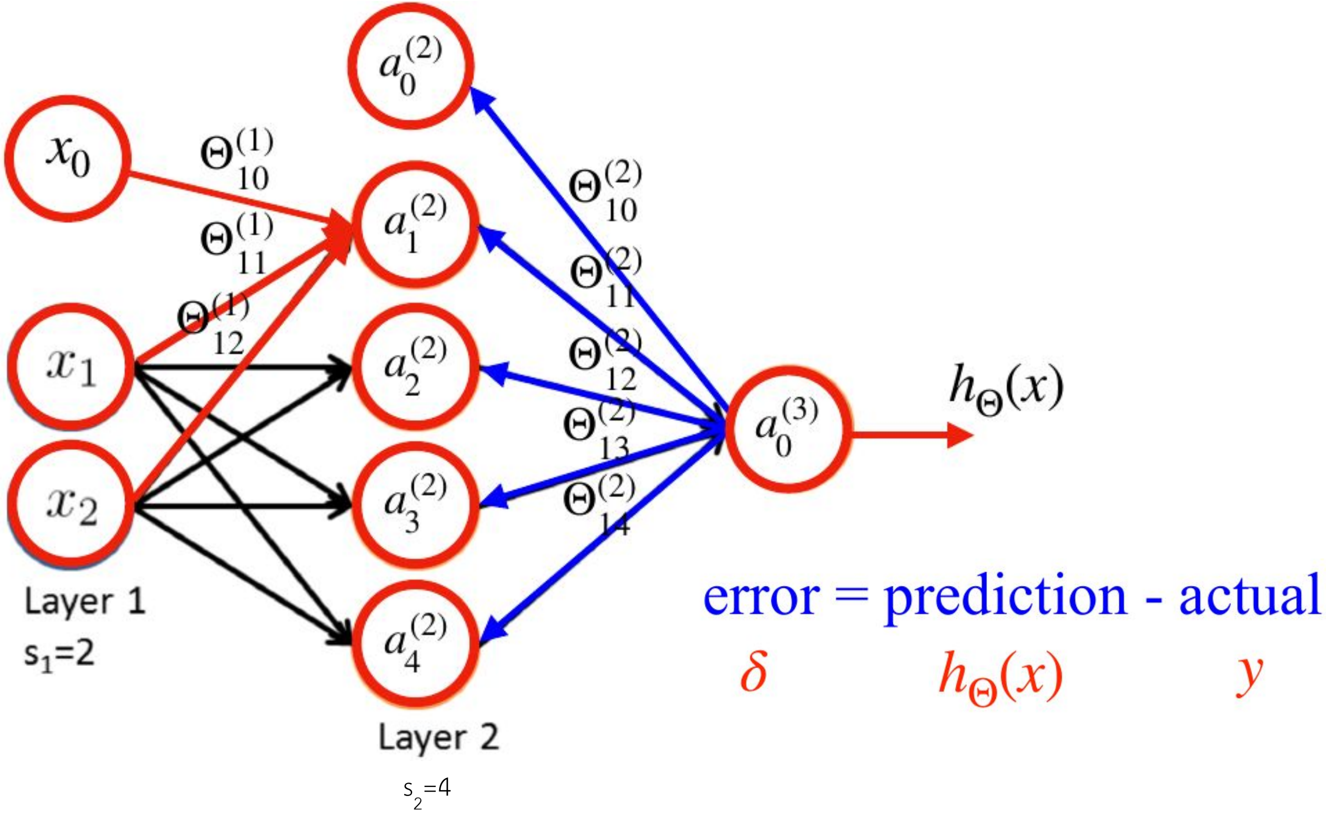
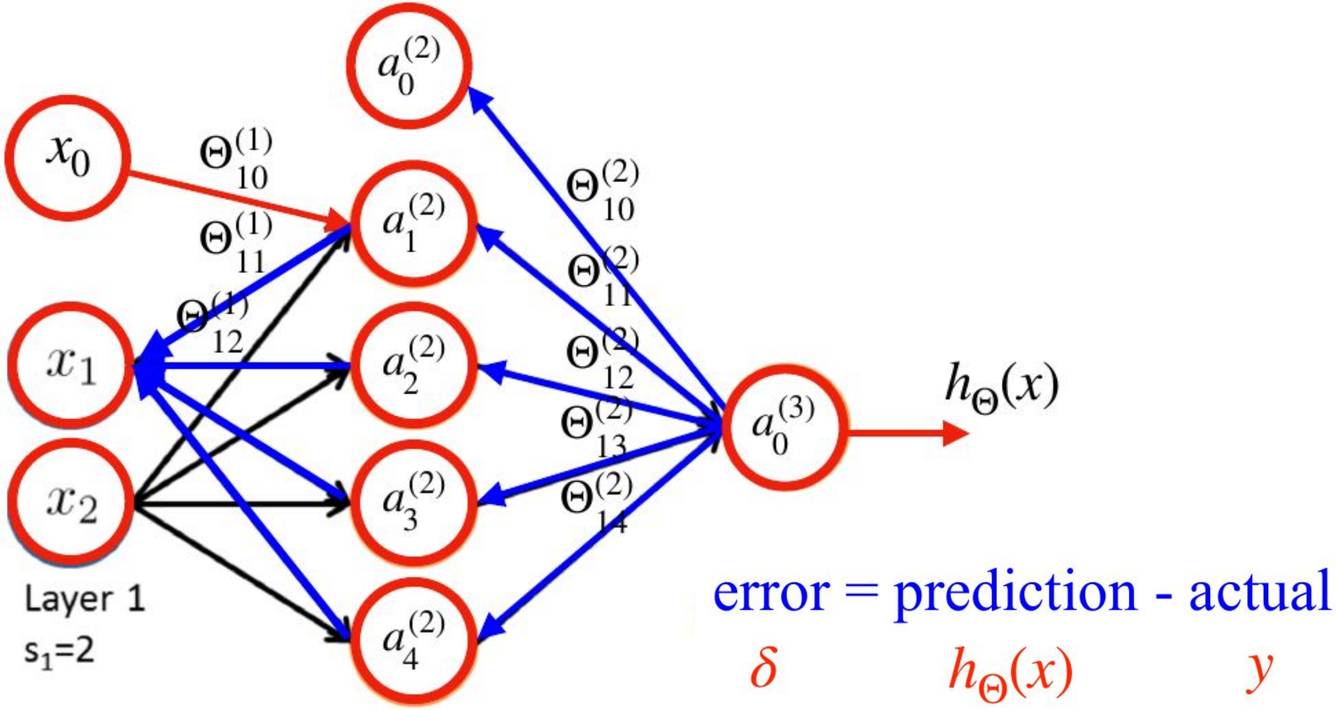



*Figure source: Andrew Ng's Machine Learning course*

# Backpropagation




Layer 2

$s_2=4$

*Figure source: Andrew Ng's Machine Learning course*

# Question เฉลย

สมมติเรามี training examples 2 ตัว $(x^{(1)}, y^{(1)})$ และ $(x^{(2)}, y^{(2)})$

ข้อใดต่อไปนี้ เป็นลำดับที่ถูกต้องของ operation สำหรับคำนวณ gradient ?

(FP = forward propagation, BP = back propagation, $\longrightarrow$ = ตามด้วย)

(i) FP โดยใช้ $x^{(1)}$ $\longrightarrow$ FP โดยใช้ $x^{(2)}$ แล้ว BP โดยใช้ $y^{(1)}$ $\longrightarrow$ BP โดยใช้ $y^{(2)}$

(ii) FP โดยใช้ $x^{(1)}$ $\longrightarrow$ BP โดยใช้ $y^{(2)}$ แล้ว FP โดยใช้ $x^{(2)}$ $\longrightarrow$ BP โดยใช้ $y^{(1)}$

(iii) BP โดยใช้ $y^{(1)}$ $\longrightarrow$ FP โดยใช้ $x^{(1)}$ แล้ว BP โดยใช้ $y^{(2)}$ $\longrightarrow$ FP โดยใช้ $x^{(2)}$

(iv) FP โดยใช้ $x^{(1)}$ $\longrightarrow$ BP โดยใช้ $y^{(1)}$ แล้ว FP โดยใช้ $x^{(2)}$ $\longrightarrow$ BP โดยใช้ $y^{(2)}$

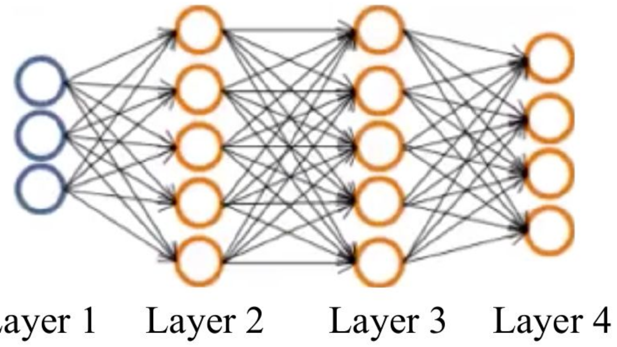
# ตัวอย่าง: Running Example

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

$L$ = จำนวน layer ทั้งหมด ใน network เช่น $L = 4$

$s_l$ = จำนวน units ใน layer $l$ (ไม่นับรวม bias unit)

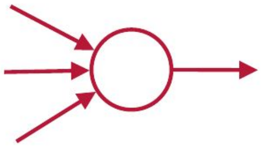
Neural network สำหรับ classification




**ประเภท 1: Binary classification**

(การแยกประเภท 2 ประเภท)

$y \in \{0,1\}$ ก็คือ 1 output unit



$y = \begin{cases} 1 \\ 0 \end{cases}$

**ประเภท 2: $K$-class classification**

(การแยกประเภท มากกว่า 2 ประเภท)

$y \in \mathbb{R}^K$ ก็คือ $K$ output unit เช่น

$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ pedestrian, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ car, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ motorcycle, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ truck

$L$ = จำนวน layer ทั้งหมด ใน network

$s_l$ = จำนวน units ใน layer $l$ (ไม่นับรวม bias unit)

$K$ = จำนวน output unit หรือ class

# Cost Function

**Logistic Regression:** (Regularized logistic regression)

$$J(\theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\Sigma_{j=1}^{n}\theta_j^2$$

เป็น generalization (รูปแบบทั่วไป) ของ ....

**Neural Network สำหรับ Classification (การแยกประเภท):**

$$J(\Theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m}\Sigma_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)})\log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\Sigma_{l=1}^{L-1}\Sigma_{i=1}^{s_l}\Sigma_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

เมื่อ $h_\Theta(x) \in \mathbb{R}^K$ เพื่อทำให้ $(h_\Theta(x))_k =$ hypothesis ที่ให้ผลเป็น output ตัวที่ $k$

## Cost Function

$L$ = จำนวน layer ทั้งหมด ใน network

$s_l$ = จำนวน units ใน layer $l$ (ไม่นับรวม bias unit)

$K=$ จำนวน output unit หรือ class

**Logistic Regression:** (Regularized logistic regression)

$$J(\theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\Sigma_{j=1}^{n}\theta_j^2$$

เป็น generalization (รูปแบบทั่วไป) ของ ....

**Neural Network สำหรับ Classification (การแยกประเภท):**

$$J(\Theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m}\boxed{\Sigma_{k=1}^{K}} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$$

(output node แต่ละอัน)

$$+\boxed{\frac{\lambda}{2m}\Sigma_{l=1}^{L-1}\Sigma_{i=1}^{s_l}\Sigma_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}$$

**ส่วน regularization :** รวม ค่ากำลังสองของ matrix $\Theta$ หลายตัว จากทุก layer ใน network

เมื่อ $h_\Theta(x) \in \mathbb{R}^K$ เพื่อทำให้ $(h_\Theta(x))_k =$ hypothesis ที่ให้ผลเป็น output ตัวที่ $k$

# Training in Neural Network (การฝึกโมเดล Neural Network)

**Cost Function:**

$$J(\Theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m}\Sigma_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)})\log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\Sigma_{l=1}^{L-1}\Sigma_{i=1}^{s_l}\Sigma_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

**Goal (เป้าหมาย):** $\min_{\Theta} J(\Theta)$

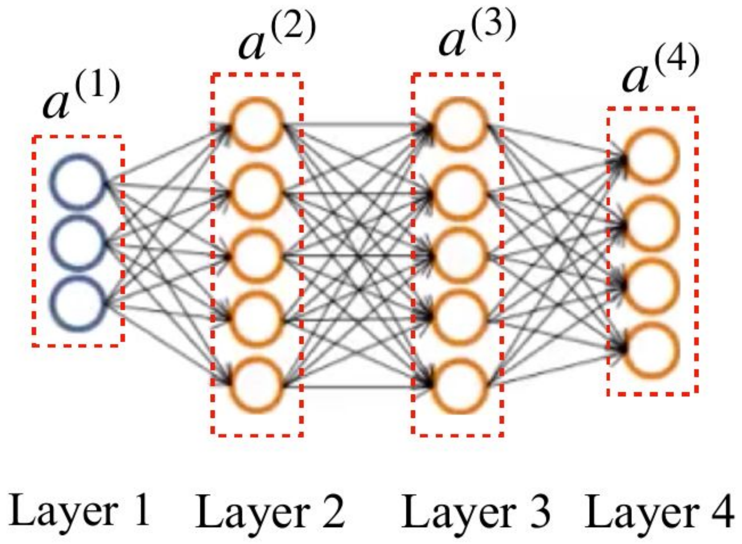
**Backpropagation ของ neural network** มีเป้าหมายเดียวกับ gradient descent ใน logistic และ linear regression ก็คือ **ทำให้ cost function น้อยที่สุด**

**โดย** ปรับค่า กลุ่มของ parameters ใน $\Theta$ ที่ เหมาะสม (optimal)

**การคำนวณ gradient (Gradient Computation)** ต้องคำนวณ:

- $J(\Theta)$
- $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$ where $\Theta_{ij}^{(l)} \in \mathbb{R}$

# ขั้นแรก : ใช้ Feedforward




สมมติ มีtraining example 1 ตัว $(x, y)$:

ใช้ forward propagation

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$

# Backpropagation (Idea / แนวคิด)

**ความเข้าใจพื้นฐาน:** $\delta_j^{(l)}$ แทน error ของ node $j$ ใน layer $l$

ก็คือ error ที่เราอยากเก็บไว้ในตัวแปร $\delta_j^{(l)}$

เพื่อเก็บความเข้าใจพื้นฐานนั้น

สำหรับแต่ละ output unit (layer $L = 4$):

$$\boxed{\delta_j^{(4)} = a_j^{(4)} - y_i = (h_\Theta(x))_j - y_j}$$

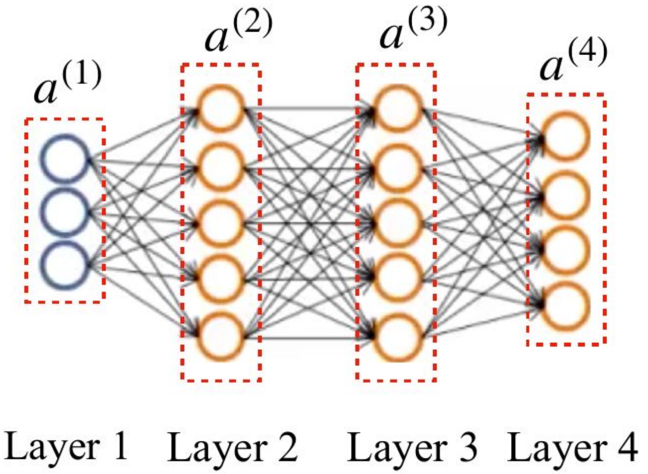
หรือ vectorized implementation

(การดำเนินการในรูป vector):

$$\delta^{(4)} = a^{(4)} - y$$

$L$ = จำนวน layers ทั้งหมด

$a^{(L)}$ = vector ของ output ของ activation unit ของ layer สุดท้าย




error ของ layer สุดท้าย = ผลทำนายจาก layer สุดท้าย - output ที่ถูกต้อง $(y)$

# Backpropagation (Idea / แนวคิด)

**ความเข้าใจพื้นฐาน:** $\delta_j^{(l)}$ แทน error ของ node $j$ ใน layer $l$

ก็คือ error ที่เราอยากเก็บไว้ในตัวแปร $\delta_j^{(l)}$

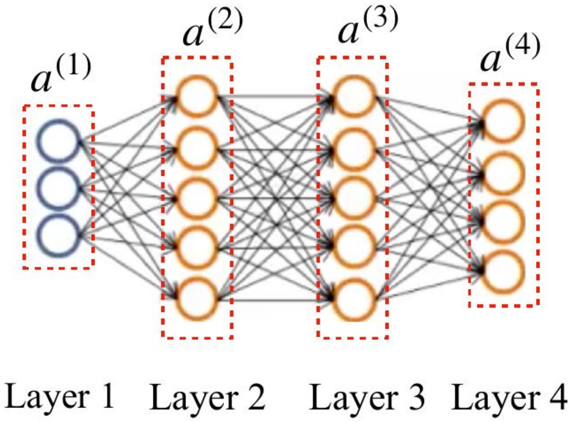
เพื่อเก็บความเข้าใจพื้นฐานนั้น

สำหรับแต่ละ output unit (layer $L = 4$):

$$\delta_j^{(4)} = a_j^{(4)} - y_i = (h_\Theta(x))_j - y_j$$

หรือ vectorized implementation (การทำในรูปแบบ vector):

$$\delta^{(4)} = a^{(4)} - y$$




คำนวณ error ใน layer ก่อนหน้า

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

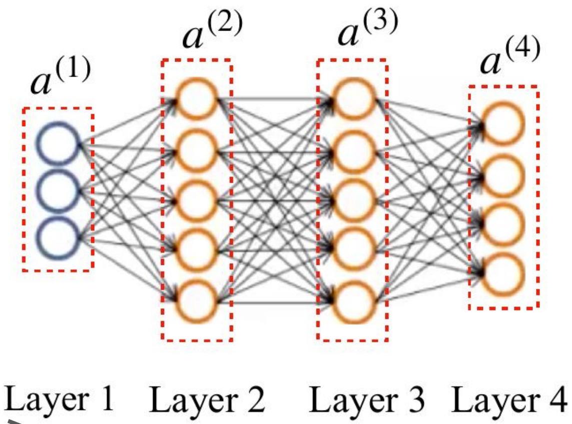
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

เมื่อ .* แทน element-wise multiplication (การคูณสมาชิกแต่ละตัวเข้าด้วยกัน) และ

$$g'(z^{(i)}) = a^{(i)} .* (1 - a^{(i)})$$

$g'(z^{(i)})$: derivative ของ activation function $g$ ที่มี input เป็น $z^{(i)}$

# Backpropagation (Idea / แนวคิด)




**ความเข้าใจพื้นฐาน:** $\delta_j^{(l)}$ แทน error ของ node $j$ ใน layer $l$

ก็คือ error ที่เราอยากเก็บไว้ในตัวแปร $\delta_j^{(l)}$

เพื่อเก็บความเข้าใจพื้นฐานนั้น

สำหรับแต่ละ output unit (layer $L = 4$):

$$\delta_j^{(4)} = a_j^{(4)} - y_i = (h_\Theta(x))_j - y_j$$

หรือ vectorized implementation (การทำในรูป vector):

$$\delta^{(4)} = a^{(4)} - y$$

คูณสมาชิกทีละตัวด้วย function $g'$

คำนวณ error ใน layer ก่อนหน้า

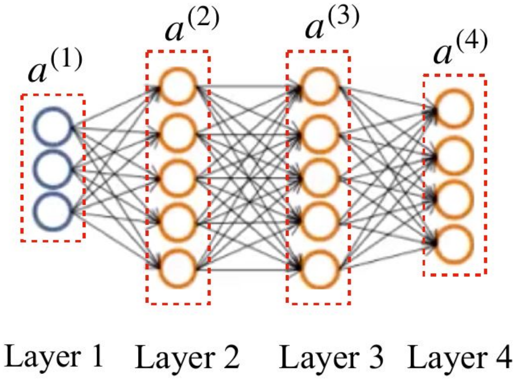
$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} \boxed{.* g'(z^{(3)})}$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

เมื่อ .* แทน element-wise multiplication (การคูณสมาชิกแต่ละตัวเข้าด้วยกัน) และ

$$\boxed{g'(z^{(i)})} = a^{(i)} .* (1 - a^{(i)})$$

# Backpropagation (Idea / แนวคิด)




**ความเข้าใจพื้นฐาน:** $\delta_j^{(l)}$ แทน error ของ node $j$ ใน layer $l$

ก็คือ error ที่เราอยากเก็บไว้ในตัวแปร $\delta_j^{(i)}$

เพื่อเก็บความเข้าใจพื้นฐานนั้น

ไม่มี $\delta^{(1)}$ เพราะเป็น input layer

สำหรับแต่ละ output unit (layer $L = 4$):

$$\delta_j^{(4)} = a_j^{(4)} - y_i = (h_\Theta(x))_j - y_j$$

หรือ vectorized implementation (การทำ[illegible]ในรูป vector):

$$\delta^{(4)} = a^{(4)} - y$$

คำนวณ error ใน layer ก่อนหน้า

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

เมื่อ .* แทน element-wise multiplication (การคูณสมาชิกแต่ละตัวเข้าด้วยกัน) และ

$$g'(z^{(i)}) = a^{(i)} .* (1 - a^{(i)})$$

# Gradient Computation : การคำนวณ Gradient

ชุดข้อมูล Training set: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

ตั้งค่า: $\Delta_{ij}^{(l)} := 0$ (for all $l, i, j$)

For $i = 1$ to $m$ :

ตั้งค่า $a^{(1)} := x^{(i)}$

ทำ forward propagation เพื่อคำนวณ $a^{(l)}$ เมื่อ $l = 2, 3, \ldots, L$

ใช้ $y(i)$ : คำนวณ $\delta^{(L)} = a^{(L)} - y^{(i)}$

คำนวณ $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ (vectorized implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$)

ท้ายสุด คำนวณ gradient โดย

- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ if $j = 0$

(**Note:** $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = D_{ij}^{(l)}$)

# Gradient Computation : การคำนวณ Gradient

ชุดข้อมูล Training set: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

ตั้งค่า: $\Delta_{ij}^{(l)} := 0$ (for all $l, i, j$)

*กำหนดค่า สมาชิกทุกตัวของ matrix เป็น 0*

For $i = 1$ to $m$:

*index ของ training example จากชุดข้อมูล training set*

ตั้งค่า $a^{(1)} := x^{(i)}$

ทำ forward propagation เพื่อคำนวณ $a^{(l)}$ เมื่อ $l = 2, 3, \ldots, L$ (*l* = index ของ unit ใน layer l)

ใช้ $y(i)$ : คำนวณ $\delta^{(L)} = a^{(L)} - y^{(i)}$

คำนวณ $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l+1)}$ (vectorized implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$)

ท้ายสุด คำนวณ gradient โดย

- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ if $j = 0$

$\left(\textbf{Note: } \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = D_{ij}^{(l)}\right)$

# Gradient Computation : การคำนวณ Gradient

ชุดข้อมูล Training set: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

คำนวณ $\delta$ ของทุก layer ยกเว้นอันสุดท้าย : เริ่มจาก layer ขวาไปซ้าย (จาก layer $L$-1 ไป $L$-2, จาก layer $L$-2 ไป $L$-3, ...)

ตั้งค่า: $\Delta_{ij}^{(l)} := 0$ (for all $l, i, j$)

*กำหนดค่า สมาชิกทุกตัวของ matrix เป็น 0*

For $i = 1$ to $m$:

*index ของ training example จากชุดข้อมูล training set*

ตั้งค่า $a^{(1)} := x^{(i)}$

ทำ forward propagation เพื่อคำนวณ $a^{(l)}$ เมื่อ $l = 2, 3, \ldots, L$ ($l$ = index ของ unit ใน layer l)

ใช้ $y(i)$ : คำนวณ $\delta^{(L)} = a^{(L)} - y^{(i)}$

คำนวณ $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ (vectorized implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$)

*[เขียนในรูป vector]*

ท้ายสุด คำนวณ gradient โดย

- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ if $j = 0$

(**Note:** $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = D_{ij}^{(l)}$)

# Gradient Computation : การคำนวณ Gradient

ชุดข้อมูล Training set: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

คำนวณ $\delta$ ของทุก layer ยกเว้นอันสุดท้าย : เริ่มจาก layer ขวาไปซ้าย (จาก layer $L$-1 ไป $L$-2, จาก layer $L$-2 ไป $L$-3, ...)

ตั้งค่า: $\Delta_{ij}^{(l)} := 0$ (for all $l, i, j$)

*กำหนดค่า สมาชิกทุกตัวของ matrix เป็น 0*

For $i = 1$ to $m$:

*index ของ training example จากชุดข้อมูล training set*

ตั้งค่า $a^{(1)} := x^{(i)}$

ทำ forward propagation เพื่อคำนวณ $a^{(l)}$ เมื่อ $l = 2, 3, \ldots, L$ ($l$ = index ของ unit ใน layer l)

ใช้ $y(i)$ : คำนวณ $\delta^{(L)} = a^{(L)} - y^{(i)}$

คำนวณ $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ (vectorized implementation $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$)

*[เขียนในรูป vector]*

ท้ายสุด คำนวณ gradient (ใช้ matrix $D$ สะสมรวมค่า เพื่อคำนวณ partial derivative) โดย

- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

(**Note:** $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = D_{ij}^{(l)}$)

$i$ = index ของ unit ใน layer ถัดไป

$j$ = index ของ unit ใน layer ปัจจุบัน

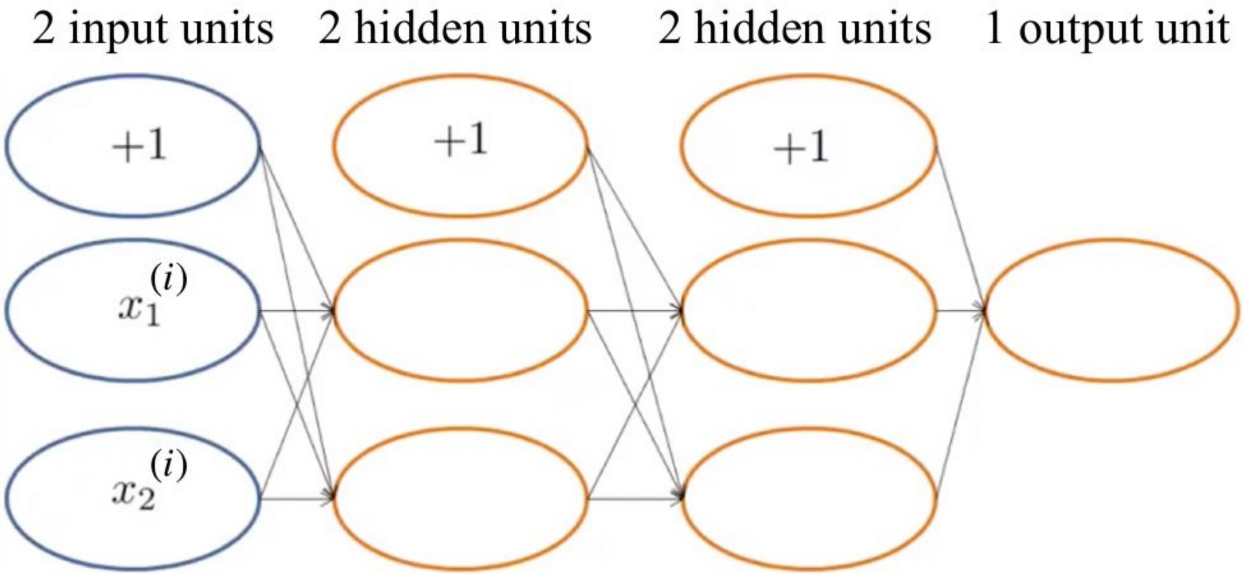
# Neural Network : Learning

## Feedforward +

## Backpropagation (Backprop)

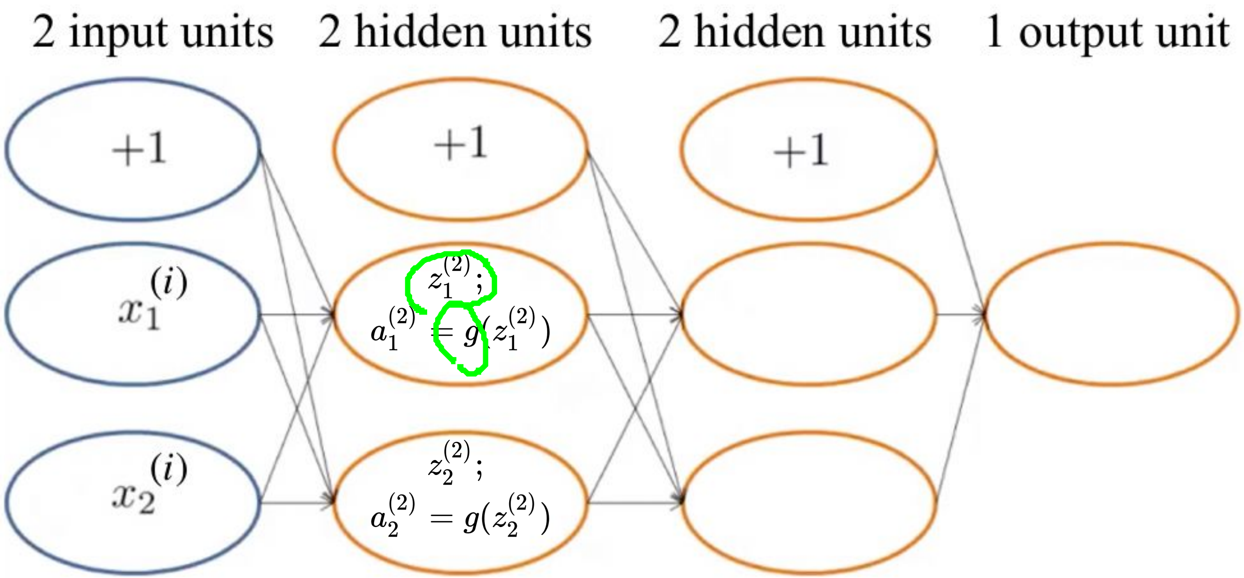
Krittameth Teachasrisaksakul

# Forward Propagation



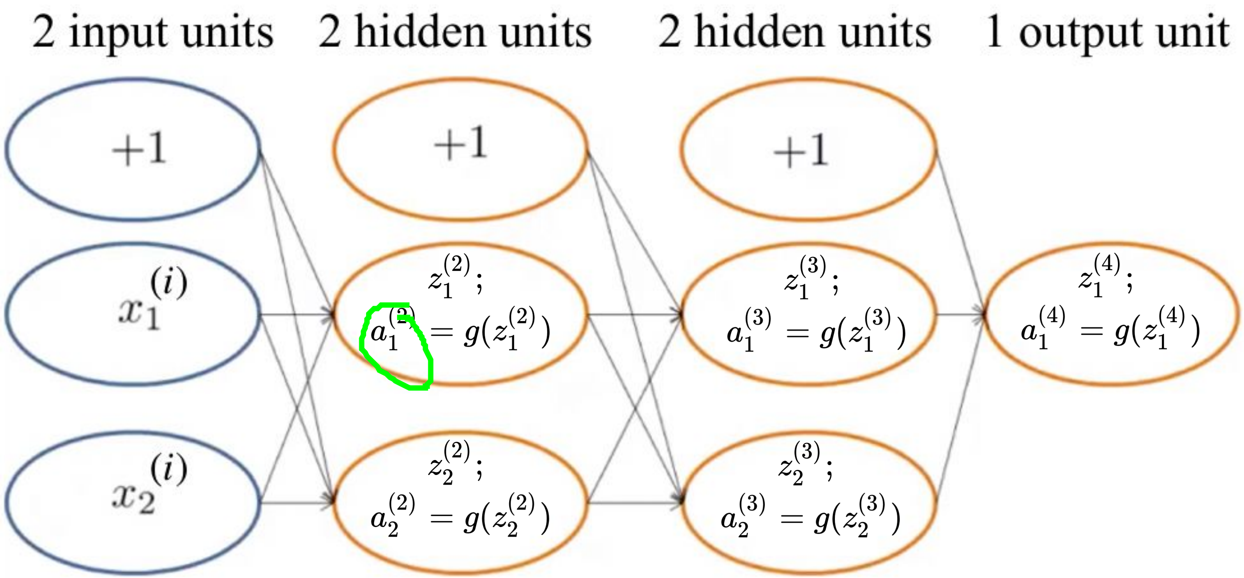

**ชุดข้อมูล Training Set:** $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

# Forward Propagation



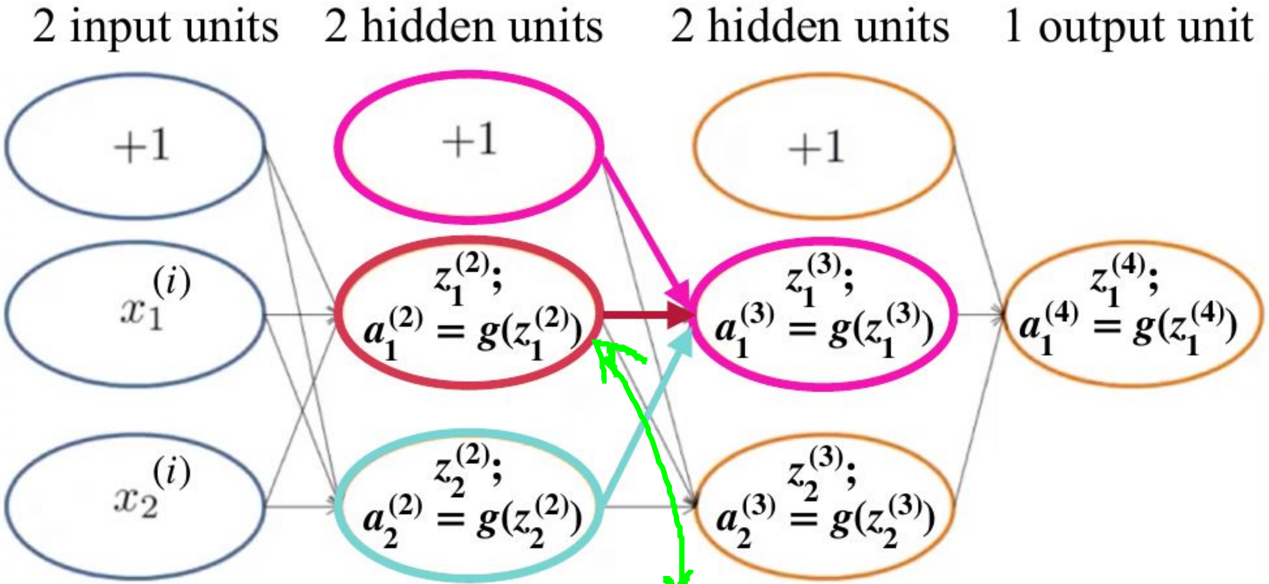

**ชุดข้อมูล Training Set:** $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

# Forward Propagation




**ชุดข้อมูล Training Set:** $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

# Forward Propagation




$$z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} \times a_1^{(2)} + +\Theta_{12}^{(2)} \times a_2^{(2)}$$

# Backpropagation

**เพื่อเข้าใจว่ามันทำอะไร** : พิจารณากรณีนี้:

- มี 1 output unit
- ไม่ทำ regularization ก็คือ $\lambda = 0$
- มี logistic sigmoid output 1 ตัว ก็คือ $m = 1$

Cost function:

$$J(\Theta) = -\frac{1}{m}\left[\Sigma_{i=1}^{m} \boxed{y^{(i)}\log(h_\Theta(x^{(i)})) + (1 - y^{(i)})\log(1 - (h_\Theta(x^{(i)}))}\right]$$

$$\cancel{+\frac{\lambda}{2m}\Sigma_{l=1}^{L-1}\Sigma_{i=1}^{s_l}\Sigma_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}$$

$$\Downarrow$$

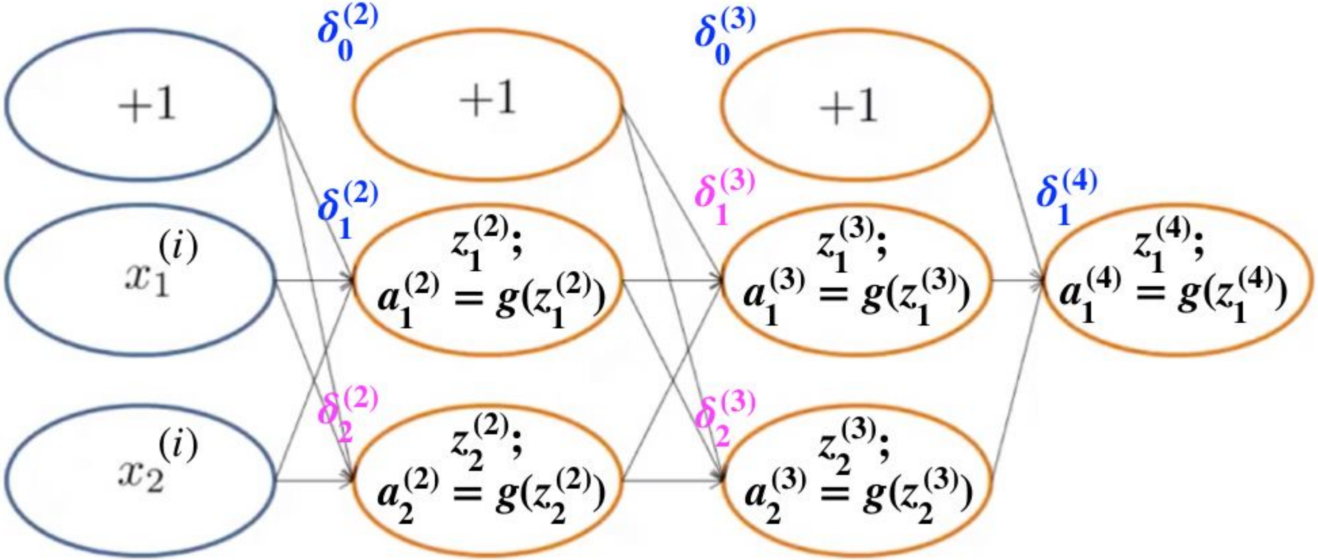
$$\textbf{cost}(i) = y^{(i)}\log(h_\Theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\Theta(x^{(i)}))$$

เราสามารถมอง $\textbf{cost}(i)$ เป็นว่า network ทำงานได้ดีแค่ไหนเมื่อใช้กับ example $i$?

# Backpropagation

***ทบทวน:*** derivative เป็นความชันของเส้นสัมผัสของ cost function

ดังนั้น ยิ่งความชันชันมาก ยิ่งผิดมาก (ความผิดพลาดสูง)



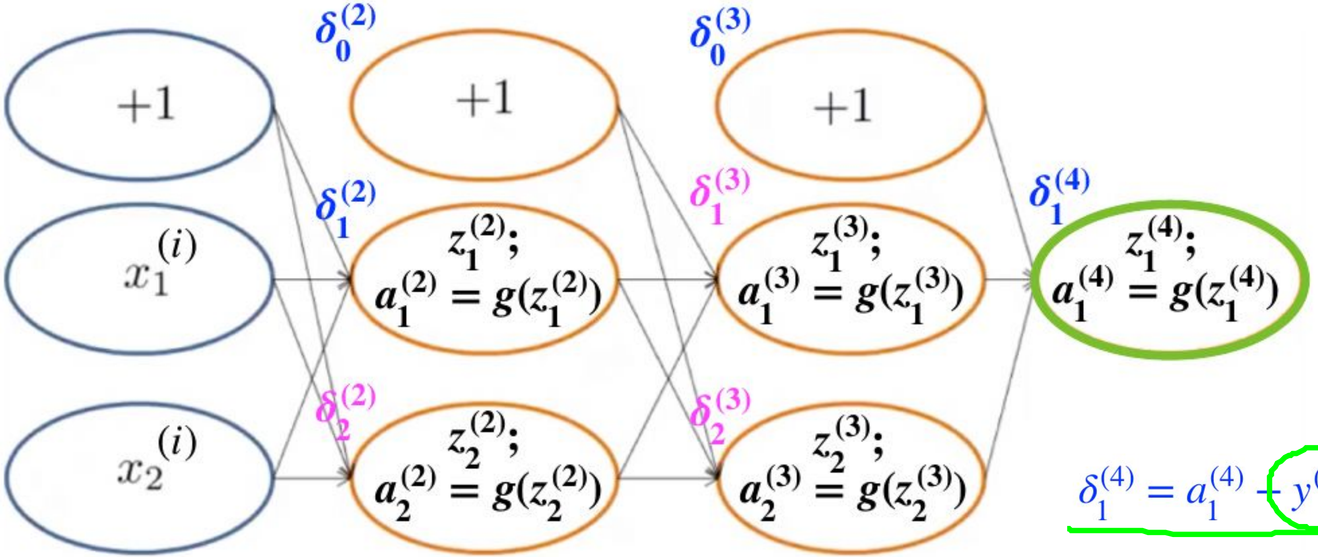

$\delta_j^{(l)}$ แทน error สำหรับ cost สำหรับ $a_j^{(l)}$ (unit $j$ ใน layer $l$)

หลังจากนี้ จะแสดงให้เห็นว่า $\delta_j^{(i)} = \boxed{\frac{\partial}{\partial z_j^{(l)}} \textbf{cost}(i)}$ ับ $j \geq 0$

# Backpropagation

***ทบทวน:*** derivative เป็นความชันของเส้นสัมผัสของ cost function

ดังนั้น ยิ่งความชันชันมาก ยิ่งผิดมาก (ความผิดพลาดสูง)




$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$

$\delta_j^{(l)}$ แทน error สำหรับ cost สำหรับ $a_j^{(l)}$ (unit $j$ ใน layer $l$)

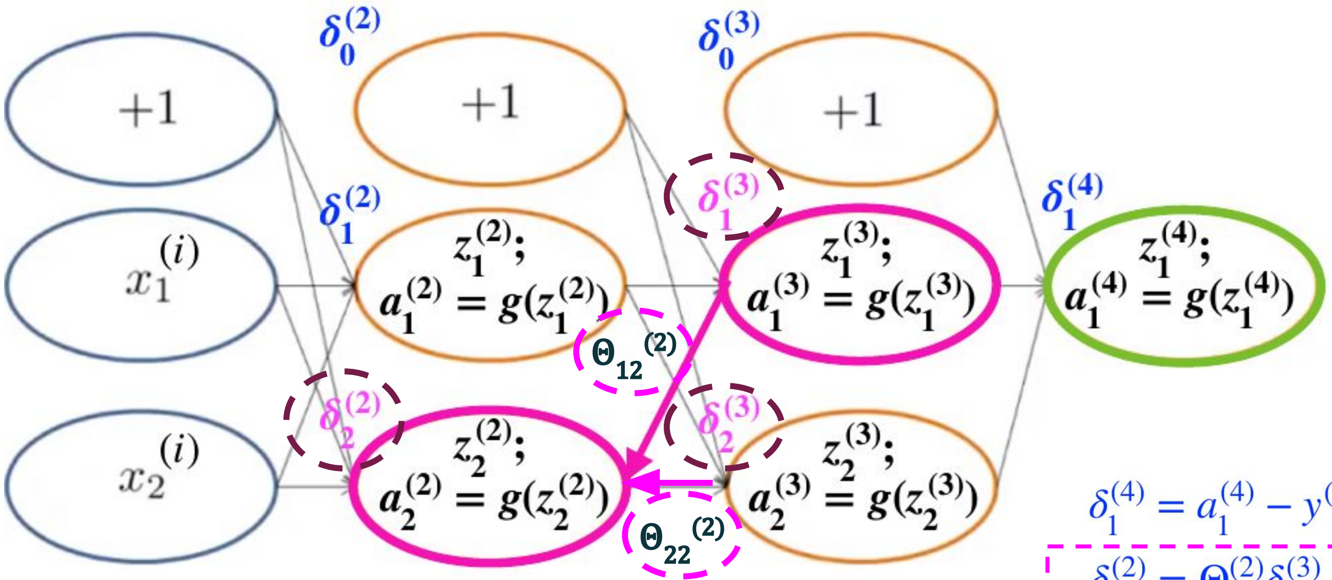
หลังจากนี้ จะแสดงให้เห็นว่า $\delta_j^{(i)} = \boxed{\frac{\partial}{\partial z_j^{(l)}} \textbf{cost}(i)}$ ับ $j \geq 0$

วิธีคำนวณ $\delta_j^{(l)}$

# Backpropagation

(1) มองลูกศรเป็น $\Theta_{ij}^{(l)}$ (2) ทำจากด้านขวาไปซ้ายของ network

(3) คำนวณ $\delta_j^{(l)}$ = ผลรวมของ [ $\Theta_{ij}^{(l)} \times \delta_j^{(l)}$ ที่อยู่ด้านขวาของ $\Theta_{ij}^{(l)}$ ]




$$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$$

$$\delta_2^{(2)} = \Theta_{12}^{(2)}\delta_1^{(3)} + \Theta_{22}^{(2)}\delta_2^{(3)}$$

$\delta_j^{(l)}$ แทน error สำหรับ cost สำหรับ $a_j^{(l)}$ (unit $j$ ใน layer $l$)

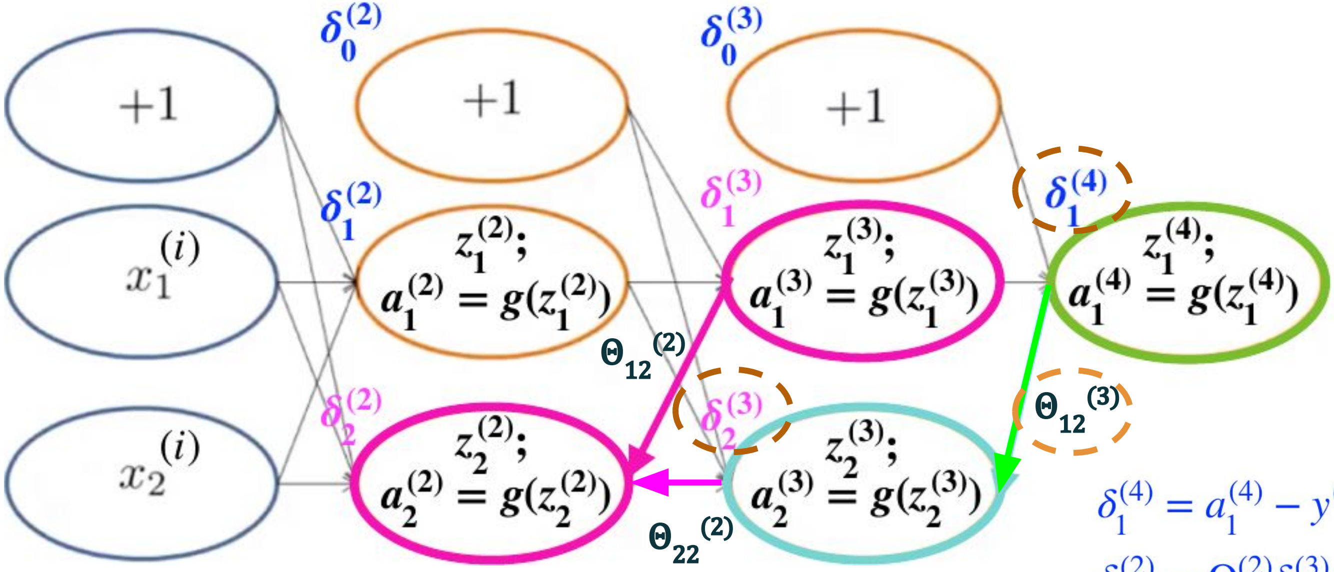
หลังจากนี้ จะแสดงให้เห็นว่า $\delta_j^{(i)} = \frac{\partial}{\partial z_j^{(l)}} \textbf{cost}(i)$ ับ $j \geq 0$

วิธีคำนวณ $\delta_j^{(l)}$

# Backpropagation

(1) มองลูกศรเป็น $\Theta_{ij}^{(l)}$ (2) ทำจากด้านขวาไปซ้ายของ network

(3) คำนวณ $\delta_j^{(l)}$ = ผลรวมของ [ $\Theta_{ij}^{(l)} \times \delta_j^{(l)}$ ที่อยู่ด้านขวาของ $\Theta_{ij}^{(l)}$ ]




$$\delta_1^{(4)} = a_1^{(4)} - y^{(i)}$$

$$\delta_2^{(2)} = \Theta_{12}^{(2)}\delta_1^{(3)} + \Theta_{22}^{(2)}\delta_2^{(3)}$$

$$\delta_2^{(3)} = \Theta_{12}^{(3)}\delta_1^{(4)}$$

$\delta_j^{(l)}$ แทน error สำหรับ cost สำหรับ $a_j^{(l)}$ (unit $j$ ใน layer $l$)

หลังจากนี้ จะแสดงให้เห็นว่า $\delta_j^{(i)} = \frac{\partial}{\partial z_j^{(l)}}\textbf{cost}(i)$ บ $j \geq 0$

# Question

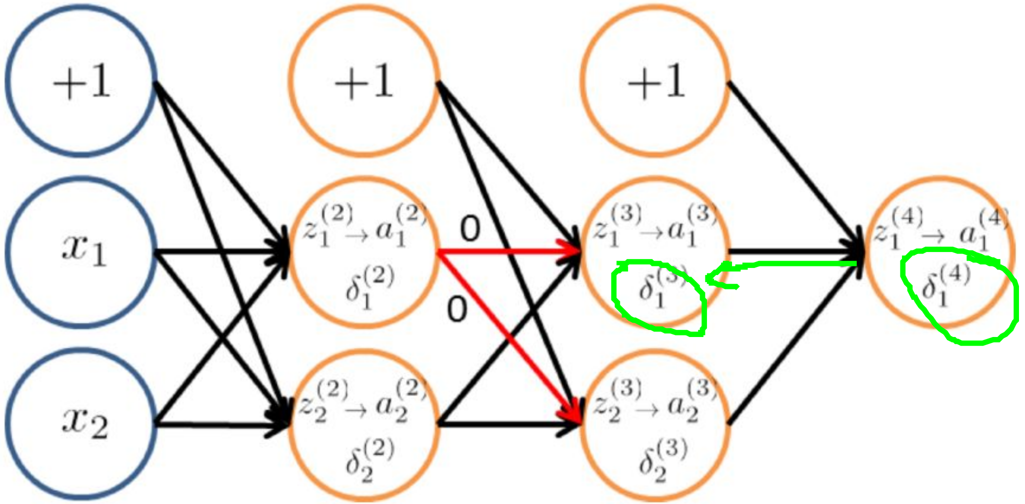
พิจารณา neural network ต่อไปนี้

สมมติ ค่า weights ทั้งสองค่า ที่เป็นสีแดง

($\Theta_{11}{}^{(2)}$ และ $\Theta_{21}{}^{(2)}$) เท่ากับ 0

หลังจาก run back propagation

ข้อใดเป็นจริงเกี่ยวกับค่าของ $\delta_1{}^{(3)}$ ?




(i) $\delta_1{}^{(3)} > 0$

(ii) $\delta_1{}^{(3)} = 0$ ก็ต่อเมื่อ $\delta_1{}^{(2)} = \delta_2{}^{(2)} = 0$ เท่านั้น

(iii) $\delta_1{}^{(3)} \leq 0$ โดยไม่ขึ้นอยู่กับค่าของ $\delta_1{}^{(2)}$ และ $\delta_2{}^{(2)}$

(iv) มีข้อมูลไม่เพียงพอที่จะบอกได้

# Neural Network : Learning

## Backpropagation
## w.r.t. mathematical sense

Krittameth Teachasrisaksakul

# การปรับค่า Parameter (Parameter Update)

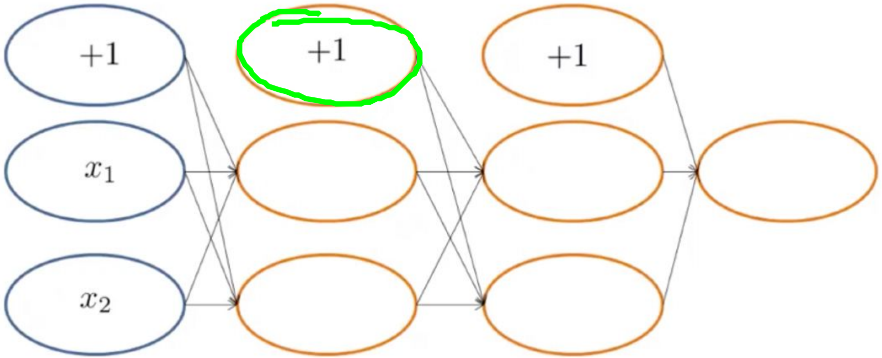
- ในกรณีที่ output layer มี logistic sigmoid หน่วยเดียว
- หลังจากทำ forward propagation : จะได้ค่าที่ถูกทำนาย (predicted value) $\hat{y}$ (หรือ $h_{\theta}(x)$ )
- กฎการปรับค่า parameter ของ neural network บ่อยครั้งจะทำ ในรูปแบบของการ backpropagating error หรือ loss
- ถ้าเรามี objective function เช่น maximum likelihood : สามารถแปลงเป็น loss โดย negating มัน
- ดังนั้น จะได้ log loss function สำหรับ network ที่มี output เป็น logistic sigmoid หน่วยเดียว:

- สังเกตว่า มันง่ายที่จะทำแบบเ $\mathscr{L}(\hat{y}, y) = -\left[y \log \hat{y} + (1 - y)\log(1 - \hat{y})\right]$ n สำหรับ $y$) หรือ softmax output (การกระจายตัวแบบ multinomial สำหรับ $y$) [1]

[1] อ้างอิง Goodfellow et al. (2016) Section 6.2.2.4 เกี่ยวกับ output ประเภทอื่น

# การปรับค่า Parameter (Parameter Update)

เพื่อปรับค่า parameter ใน layer $l$ : จะปรับโดยใช้ทำ gradient descent กับ log loss:




$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$$

$$\mathbf{b}_0^{(l)} := \mathbf{b}_0^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}_0^{(l)}}$$

หมายเหตุ : เขียน $W^{(l)}$ แทน $\Theta^{(l)}$ และ $b^{(l)}$ แทน $\theta_0^{(l)}$

# การปรับค่า Parameter (Parameter Update)

ขั้นแรก พิจารณาค่า weight ที่ output layer จะได้ว่า:

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(4)}} = -\frac{\partial}{\partial \mathbf{W}^{(4)}}\left((1-y)\log(1-\hat{y}) + y\log\hat{y}\right)$$

$$= -(1-y)\frac{\partial}{\partial \mathbf{W}^{(4)}}\log\left(1-g(\mathbf{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})\right)$$

$$-y\frac{\partial}{\partial \boldsymbol{W}^{(4)}}\log\left(g(\boldsymbol{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})\right)$$

$$= \frac{(1-y)g'(\boldsymbol{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})\boldsymbol{a}^{(3)^T}}{1-g(\boldsymbol{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})} - \frac{yg'(\boldsymbol{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})\boldsymbol{a}^{(3)^T}}{g(\boldsymbol{W}^{(4)}\boldsymbol{a}^{(3)}+\boldsymbol{b}^{(4)})}$$

# การปรับค่า Parameter (Parameter Update)

ต่อจากนั้น สังเกตว่า ใน model นี้ $g(z)$ เป็น logistic sigmoid

แทนที่ $g(z)$ ด้วย $\sigma(z)$ และแทน $g'(z)$ ด้วย $\sigma'(z)$ เพื่อทำให้เข้าใจง่ายขึ้น

เพราะ $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ ทำให้สามารถทำให้ expression (นิพจน์) ง่ายขึ้น:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(4)}} &= \ldots \\
&= (1 - y)\sigma(\mathbf{W}^{(4)}\mathbf{a}^{(3)} + \mathbf{b}^{(4)})\mathbf{a}^{(3)T} - y\left(1 - \sigma(\mathbf{W}^{(4)}\mathbf{a}^{(3)} + \mathbf{b}^{(4)})\right)\mathbf{a}^{(3)T} \\
&= (1 - y)\mathbf{a}^{(4)}\mathbf{a}^{(3)T} - y(1 - \mathbf{a}^{(4)})\mathbf{a}^{(3)T} \\
&= (\mathbf{a}^{(4)} - y)\mathbf{a}^{(3)T}
\end{aligned}
$$

# การปรับค่า Parameter (Parameter Update)

แล้วทำอย่างไรกับ weights ของ layer 3 ?

$$a^{4} = g(z^{(4)})$$

เราสามารถใช้ chain rule (กฎลูกโซ่) จากเรื่องแคลคูลัส (calculus)

เมื่อเรามี function $f(z)$ เมื่อ $z = g(x)$ เราสามารถเขียน

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

ในกรณีของเรา จะได้ว่า

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(3)}} = \frac{\partial \mathscr{L}}{\partial \mathbf{a}^{(4)}}\frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}}\frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}}\frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}}\frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(3)}}$$

# การปรับค่า Parameter (Parameter Update)

เพื่อประเมินค่า expression นี้ พยายามใช้สิ่งที่เรารู้เกี่ยวกับ $\frac{\partial \mathscr{L}}{\partial W^{(4)}}$ ก่อน

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(4)}} = \boxed{\frac{\partial \mathscr{L}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}}} \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{W}^{(4)}} = \boxed{(\mathbf{a}^{(4)} - y)}\mathbf{a}^{(3)}$$

?

เราสามารถใช้ส่วนนี้ได้

$$\frac{\partial \mathscr{L}}{\partial \mathbf{z}^{(4)}} = \frac{\partial \mathscr{L}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}} = \boxed{\mathbf{a}^{(4)} - y}$$

สำหรับ พจน์ที่เหลือ จะได้ว่า

$$\frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}^{(3)}} = \mathbf{W}^{(4)} \qquad \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{z}^{(3)}} = g'(\mathbf{z}^{(3)}) \qquad \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(3)}} = \mathbf{a}^{(2)}$$

# การปรับค่า Parameter (Parameter Update)

เอาพจน์จาก chain rule มาใส่ ในลำดับที่เหมาะกับ การคำนวณโดยใช้ vector เราจะได้

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(3)}} = \mathbf{diag}\left(g'(\mathbf{z}^{(3)})\right)\mathbf{W}^{(4)}(\mathbf{a}^{(4)} - y)\mathbf{a}^{(2)^T}$$

การคำนวณ คล้ายกับ การคำนวณของ bias weight ยกเว้นว่าเรามี 1 แทนที่ $\mathbf{a}^{(2)}$

# การปรับค่า Parameter (Parameter Update)

แล้วทำอย่างไรกับ weights ของ layer 2 ?

เราอยากได้

$$\frac{\partial \mathscr{L}}{\partial w_{ij}^{(2)}}$$

เราจะเห็นได้ทันทีว่า $w_{\mathrm{ij}}^{(2)}$ มีผลกระทบต่อ activation $\mathbf{a}^{(3)}$ ทุกตัว ใน layer ที่ 3

ในกรณีนี้ chain rule ที่อยู่ในรูปทั่วไปมากขึ้น เมื่อ $y = f(\mathrm{u})$ และ $\mathrm{u} = g(\mathrm{x})$ ก็คือ

$$\frac{\partial y}{\partial x_i} = \Sigma_j \frac{\partial y}{\partial u_j} \frac{\partial u_j}{\partial x_i}$$

# การปรับค่า Parameter (Parameter Update)

ในกรณีของเรา : จาก generalized chain rule (ในรูปทั่วไป) จะได้

$$\frac{\partial \mathscr{L}}{\partial w_{ij}^{(2)}} = \Sigma_k \frac{\partial \mathscr{L}}{\partial \mathbf{a}_k^{(3)}} \frac{\partial \mathbf{a}_k^{(3)}}{\partial w_{ij}^{2)}}$$

ขยายพจน์ 2 พจน์ในผลรวม จะได้

$$\frac{\partial \mathscr{L}}{\partial w_{ij}^{(2)}} = \Sigma_k \frac{\partial \mathscr{L}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}} \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}_k^{(3)}} \frac{\partial \mathbf{a}_k^{(3)}}{\partial \mathbf{z}_k^{(3)}} \frac{\partial \mathbf{z}_k^{(3)}}{\partial \mathbf{a}_j^{(2)}} \frac{\partial \mathbf{a}_j^{(2)}}{\partial \mathbf{z}_j^{(2)}} \frac{\partial \mathbf{z}_j^{(2)}}{\partial w_{ij}^{(2)}}$$

# การปรับค่า Parameter (Parameter Update)

เริ่มซับซ้อนมากขึ้น

การแก้ปัญหาอย่างมีประสิทธิภาพ คือ รู้ว่าพจน์

$$\frac{\partial \mathscr{L}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{z}^{(4)}} \frac{\partial \mathbf{z}^{(4)}}{\partial \mathbf{a}_k^{(3)}} \frac{\partial \mathbf{a}_k^{(3)}}{\partial \mathbf{z}_k^{(3)}}$$

ได้ถูกคำนวณแล้ว ในกระบวนการหาค่า

$$\frac{\partial \mathscr{L}}{\partial w_{jk}^{(3)}}$$

(กลับไป และดู slide # 40)

# การปรับค่า Parameter (Parameter Update)

เราอยากใช้ซ้ำพจน์ที่คล้ายๆกับ $$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i^{(l)}}$$

นิยาม: $$\delta_i^{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_i^{(l)}}$$

เราจะได้ backpropagation algorithm สำหรับ neural network

# Fully-connected Network

ถ้ามี example $(x, y)$

$$\mathbf{a}^{(1)} := \boldsymbol{x}$$

for $l = 2 \ldots L$ do

$$\mathbf{z}^{(l)} := \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} := g^{(l)}(\mathbf{z}^{(l)})$$

$$\delta^{(L)} := \frac{\partial \mathscr{L}}{\partial \mathbf{z}^{(L)}}$$

for $l = L \ldots 2$ do

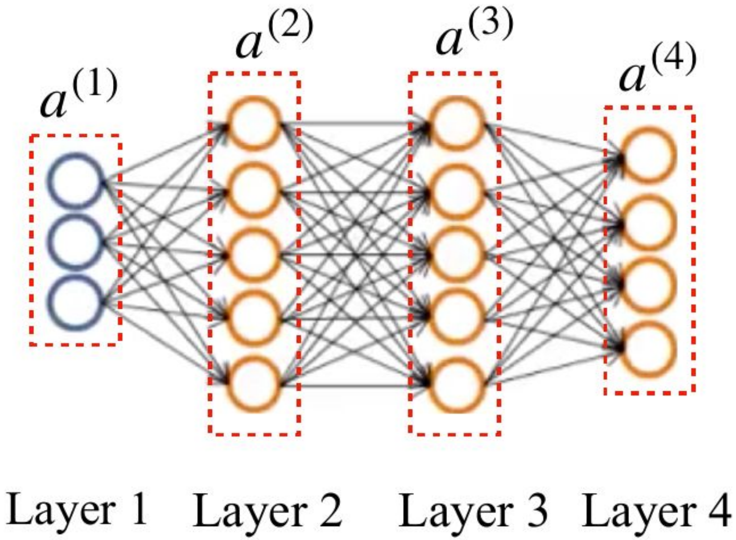
$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}^{(l)}} := \delta^{(l)} \mathbf{a}^{(l-1)^T}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{b}^{(l)}} := \delta^{(l)}$$

if $l > 2$ then

$$\delta^{(l-1)} := \mathbf{diag}\left(g'(\mathbf{z}^{(l-1)})\right)\mathbf{W}^{(l)}\delta^{(l)}$$

Recall that $\mathscr{L}(\hat{y}, y) = -\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right]$



# Neural Networks ในทางปฏิบัติ

Krittameth Teachasrisaksakul

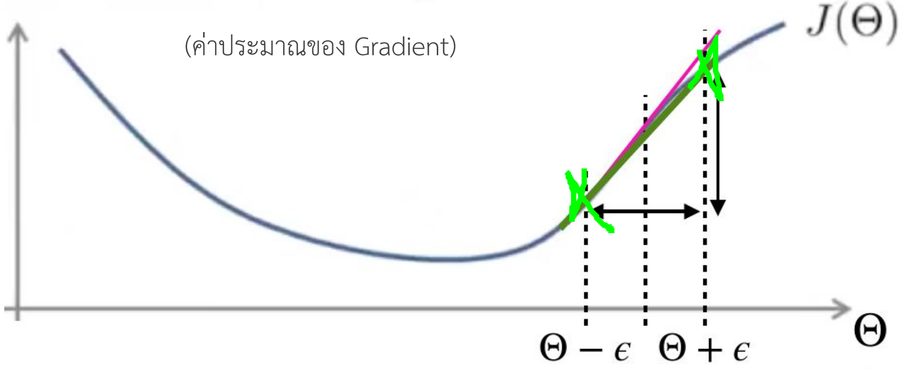
# การตรวจสอบ Gradient (Gradient Checking)

เพื่อทำให้แน่ใจว่า การคำนวณ gradient (backprop) ถูก implement อย่างถูกต้อง โดยยืนยันความถูกต้องว่า

$$\frac{J(\Theta+\epsilon)-J(\Theta-\epsilon)}{2\epsilon} \approx \text{backprop}$$

**Numerical estimation of gradients**

(ค่าประมาณของ Gradient)




$(\Theta \in \mathbb{R})$

ประมาณค่า derivative (อนุพันธ์) ของ cost function ด้วย

$$\frac{d}{d\Theta}J(\Theta) \approx \frac{J(\Theta+\epsilon)-J(\Theta-\epsilon)}{2\epsilon}$$

$$e.g.\ \ \epsilon = 10^{-4}$$

ค่าที่น้อยของ $\epsilon$ เช่น $\epsilon = 10^{-4}$ รับรองว่า backpropagation ทำงานถูกต้อง
แต่ถ้าค่าของ $\epsilon$ น้อยเกินไป อาจเกิด numerical problems

# Question

ให้ $J(\theta) = \theta^3$, $\theta = 1$, $\varepsilon = 0.01$ และเราใช้สูตร

$$\frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

เพื่อประมาณค่าอนุพันธ์ (derivative) ค่าใดที่เราจะได้ ถ้าใช้การประมาณค่า (approximation) นี้ ?

(เมื่อ $\theta = 1$ ค่า derivative ที่แท้จริงและแม่นยำ คือ

$$\frac{d}{d\theta} J(\theta) = 3$$

(i) 3.0000

(ii) 3.0001

(iii) 3.0301

(iv) 6.0002

# Question

ให้ $J(\theta) = \theta^3$, $\theta = 1$, $\varepsilon = 0.01$ และเราใช้สูตร

$$\frac{J(\theta+\epsilon) - J(\theta-\epsilon)}{2\epsilon}$$

เพื่อประมาณค่าอนุพันธ์ (derivative) ค่าใดที่เราจะได้ ถ้าใช้การประมาณค่า (approximation) นี้ ?

(เมื่อ $\theta = 1$ ค่า derivative ที่แท้จริงและแม่นยำ คือ

$$\frac{d}{d\theta}J(\theta) = 3$$

(i) 3.0000

(ii) 3.0001

(iii) 3.0301

(iv) 6.0002

$\theta+\varepsilon = 1+0.01 = 1.01$

$J(\theta+\varepsilon) = (\theta+\varepsilon)^3 = (1.01)^3$

$\theta-\varepsilon = 1-0.01 = 0.99$

$J(\theta-\varepsilon) = (\theta-\varepsilon)^3 = (0.99)^3$

$$\frac{d}{d\Theta}J(\Theta) \approx \frac{J(\Theta+\epsilon) - J(\Theta-\epsilon)}{2\epsilon}$$

$= [\,(1.01)^3 - (0.99)^3\,] \,/\, (2 \times 0.01)$

$= 3.001$

# แนวทาง Implementation (การทำให้เกิดผล / การเขียนโปรแกรม)

- Implement backprop เพื่อคำนวณ $\delta^{(4)}, \delta^{(3)}, \delta^{(2)}$
- Implement numerical gradient checking (การตรวจสอบ gradient เชิงตัวเลข)
- ตรวจว่ามันได้ค่าที่ใกล้เคียงกัน
- ปิด gradient checking (การตรวจสอบ gradient) และใช้ back prop เพื่อเรียนรู้ (learning)

** อย่าลืม ปิด code ที่ทำ gradient checking ก่อน train classifier

เพราะถ้า run การคำนวณ gradient ในทุก iteration ของ gradient descent ⟶ code จะช้ามาก

# Question

สาเหตุใดที่เราควรใช้ backpropagation algorithm มากกว่า วิธีการคำนวณ gradient ระหว่างการเรียนรู้ (learning) ?

(i) วิธีการคำนวณ gradient implement ได้ยากกว่ามาก

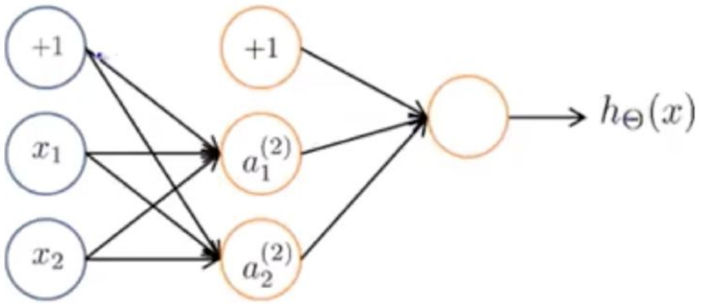
(ii) Numerical gradient algorithm (algorithm ที่คำนวณ gradient) ช้ามาก

(iii) ตอนทำ Backpropagation ไม่จำเป็นต้องตั้งค่า parameter $\boldsymbol{\varepsilon}$

(iv) ไม่มีข้อใดถูกต้อง

# การตั้งค่าเริ่มต้นของ Parameter (Parameter Initialization)

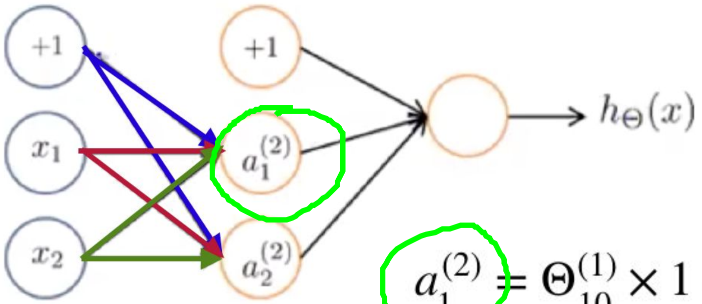
อย่าตั้งค่า parameter ทุกตัวเป็น 0 !        (ทำไมควรหลีกเลี่ยงที่จะทำแบบนี้ ?)




$\Theta_{ij}^{(l)} = 0$ สำหรับ $i, j, l$ ทุกค่า

# การตั้งค่าเริ่มต้นของ Parameter (Parameter Initialization)

อย่าตั้งค่า parameter ทุกตัวเป็น 0 !        (ทำไมควรหลีกเลี่ยงที่จะทำแบบนี้ ?)




$\Theta_{ij}^{(l)} = 0$ สำหรับ $i, j, l$ ทุกค่า

$$\left.\begin{aligned} a_1^{(2)} &= \Theta_{10}^{(1)} \times 1 + \Theta_{11}^{(1)} \times x_1 + \Theta_{12}^{(1)} \times x_2 \\ a_2^{(2)} &= \Theta_{20}^{(1)} \times 1 + \Theta_{21}^{(1)} \times x_1 + \Theta_{22}^{(1)} \times x_2 \end{aligned}\right\} \therefore a_1^{(2)} = a_2^{(2)} \text{ and } \delta_1^{(2)} = \delta_2^{(2)}$$

ดังนั้น $$\frac{\partial}{\partial \Theta_{10}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{20}^{(1)}} J(\Theta)$$ ซึ่งหมายความว่า $\Theta_{10}^{(1)} = \Theta_{20}^{(1)}$

# การตั้งค่าเริ่มต้นของ Parameter (Parameter Initialization)

การ**ตั้งค่าเริ่มต้น** weights (matrix $\Theta$) ของ neural network **เป็น 0 (ศูนย์) ทั้งหมด $\longrightarrow$ ไม่ดี**

เพราะเมื่อทำ backpropagation : output ของทุกๆ unit/node ในแต่ละ layer จะเหมือนกัน

และ gradients ที่ถูก backpropagate ในภายหลัง จะเหมือนกัน

# การตั้งค่าเริ่มต้นของ Parameter (Parameter Initialization)

การ**ตั้งค่าเริ่มต้น** weights (matrix $\Theta$) ของ neural network **เป็น 0 (ศูนย์) ทั้งหมด** $\longrightarrow$ **ไม่ดี**

เพราะเมื่อทำ backpropagation : output ของทุกๆ unit/node ในแต่ละ layer จะเหมือนกัน

และ gradients ที่ถูก backpropagate ในภายหลัง จะเหมือนกัน

**วิธีแก้ไข:** ตั้งค่าเริ่มต้นของแต่ละ แบบสุ่ม $\Theta_{ij}^{(l)}$ ง $[-\varepsilon, \varepsilon]$ ซึ่งควรมีค่าน้อย

เช่น มีค่าใกล้ 0: $\Theta_{jk}^{(i)} \sim \mathcal{N}(0,0.1)$

**วิธีที่ดีกว่า ที่ใช้จริง** (ในทางปฏิบัติ) คือ Xavier/He initialization

$$\Theta_{jk}^{(i)} \sim \mathcal{N}\left(0,\sqrt{\frac{2}{n^{(l)} + n^{(l-1)}}}\right)$$

เมื่อ $n^{(l)}$ เป็นจำนวน units ใน layer $l$ ซึ่งจะสนับสนุนให้ variance (ความแปรผัน) ของ output ของ layer ใกล้เคียงกับ variance ของ input

# การตั้งค่าเริ่มต้นของ Parameter (Parameter Initialization)

สำหรับ ReLU hidden unit : แนะนำว่า ให้ตั้งค่าเริ่มต้นของ bias weights เป็นค่าที่เป็นบวก (หรือ คงที่) ที่มีค่าน้อย

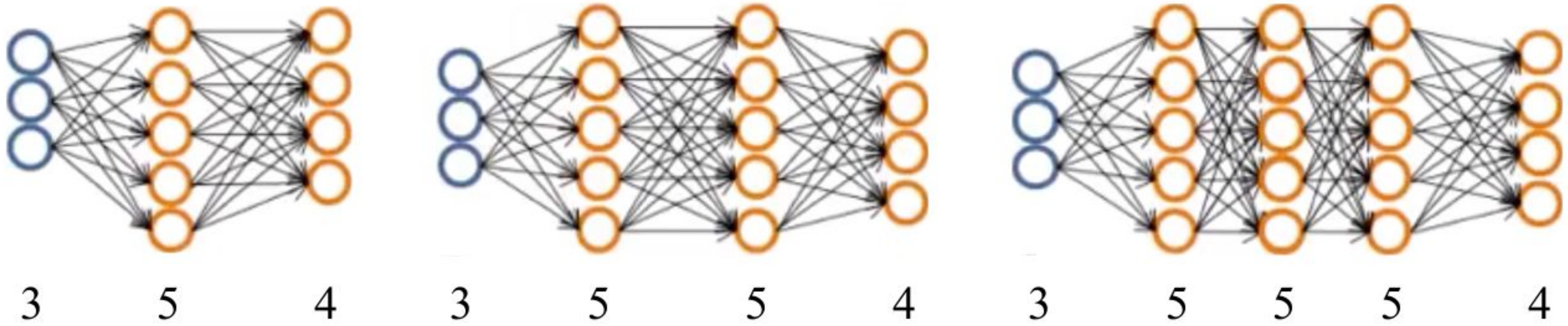
ซึ่งจะทำให้แน่ใจว่า output ของ unit เป็นบวก สำหรับ training example ส่วนมาก

# Neural Networks ในทางปฏิบัติ (2)

Krittameth Teachasrisaksakul

# 1. เลือก network architecture

Network architecture (สถาปัตยกรรมของโครงข่าย) หมายถึง โครงสร้าง หรือ connectivity pattern (แบบแผนการเชื่อมต่อ) ระหว่าง neuron เช่น

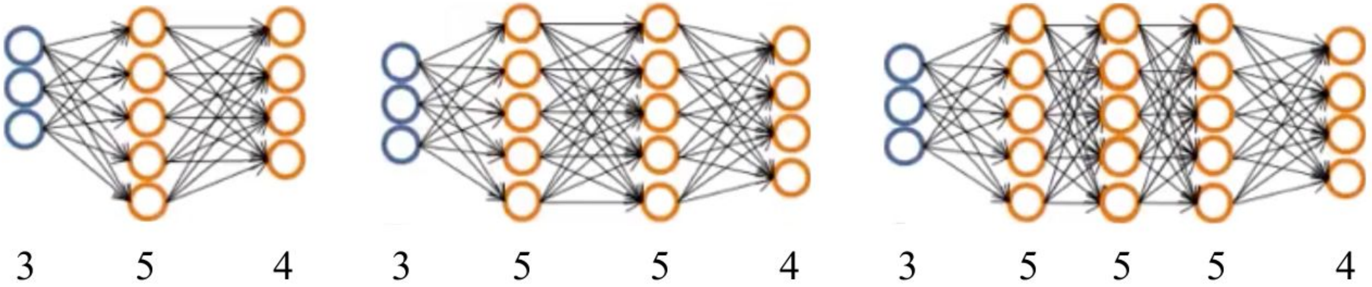



**Architecture choices (ตัวเลือกของสถาปัตยกรรม):**

- มีกี่ unit ใน hidden layer ?
- มีกี่ hidden layer ใน neural network ?

# 1. เลือก network architecture

Network architecture (สถาปัตยกรรมของโครงข่าย) หมายถึง connectivity pattern (แบบแผนการเชื่อมต่อ) ระหว่าง neuron เช่น




ค่อนข้างตรงไปตรงมา

จำนวน input unit = dimension ของ features $x^{(i)}$

จำนวน output unit = จำนวน class / ประเภท (ถ้าเราแก้ปัญหา classification)

จำนวน hidden layer = 1 (ค่า default / ค่าที่ถูกเลือกโดยอัตโนมัติ)

ถ้ามีมากกว่า 1 hidden layer แนะนำว่าให้ทุกๆ hidden layer มีจำนวน unit เท่ากัน

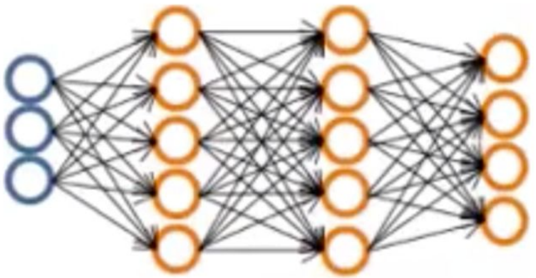
ส่วนมาก ยิ่งจำนวน hidden unit มาก ยิ่งดี (คือ ยิ่งหา function ที่ fit / เข้ากับ training data set ได้ดี)

แต่ต้องสมดุลกับ cost of computation เพราะมันจะเพิ่มขึ้น ถ้ามี hidden unit มากขึ้น

# 2. Training neural network

1. ตั้งค่าเริ่มต้นของ weights โดยสุ่ม
2. Implement forward propagation เพื่อหาค่า $h_\Theta(x^{(i)})$ สำหรับค่า $x^{(i)}$ ใดๆ
3. Implement code เพื่อคำนวณ cost function $J(\Theta)$
4. Implement backpropagation เพื่อคำนวณ partial derivative

$$\frac{\partial J(\Theta)}{\partial \Theta_{jk}^{(l)}}$$



**Forward propagation & Backpropagation:**

for i = 1 to m:

ทำ forward propagation และ backpropagation โดยใช้ example $(x^{(i)}, y^{(i)})$

ก็คือ หาค่า activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

( ที่ใช้คำนวณ $\frac{\partial}{\partial \Theta_{jk}^{(}l)} J(\Theta))$

# 2. Training neural network

5. **ใช้ gradient checking เพื่อยืนยันว่า backpropagation ทำงานถูกต้อง**

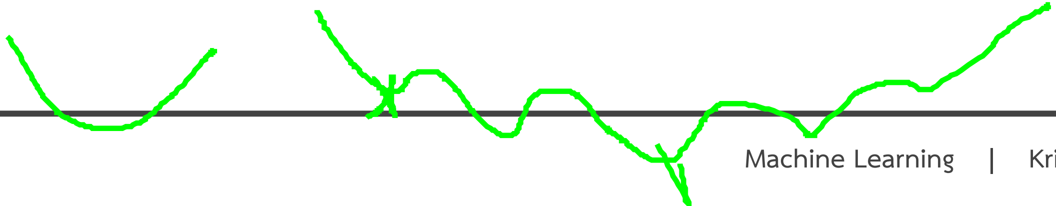
   ก็คือ เปรียบเทียบค่า $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ คำนวณด้วย backpropagation

   เทียบกับ ค่าประมาณ (numerical estimate) ของ gradient $J(\Theta)$

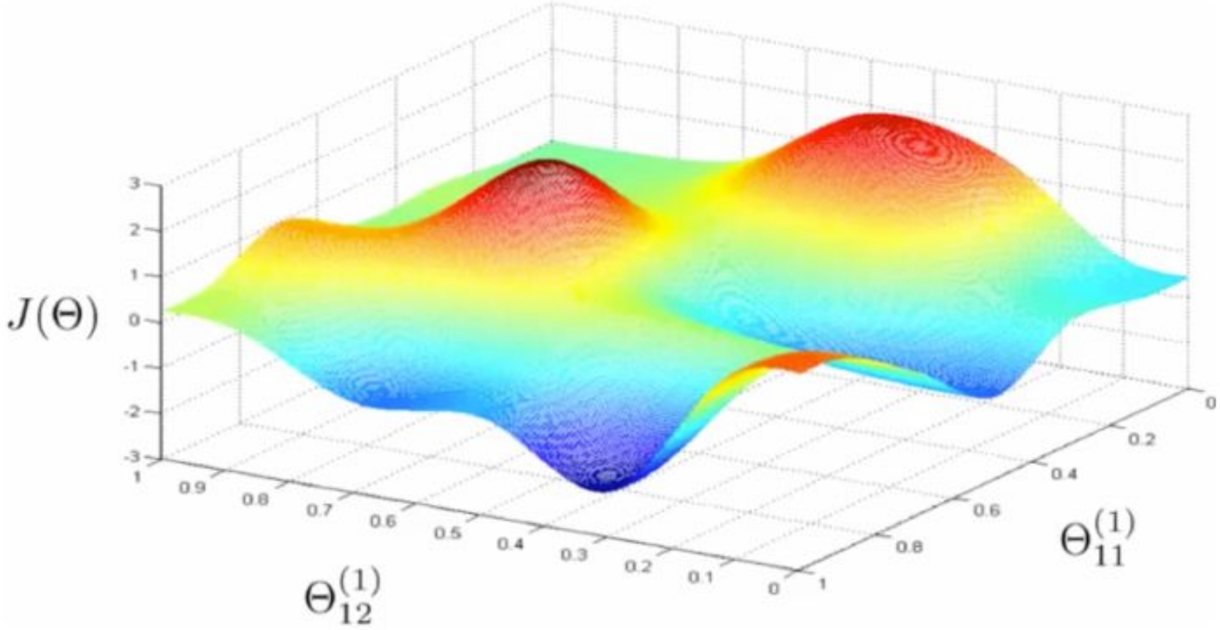
   แล้วปิดการใช้ code ของ gradient checking

6. **ใช้ gradient descent** (หรือ built-in optimization function) กับ backpropagation **เพื่อทำให้ cost function $J(\Theta)$ น้อยที่สุด** (ซึ่งเป็น function ของ parameter weight $\Theta$)

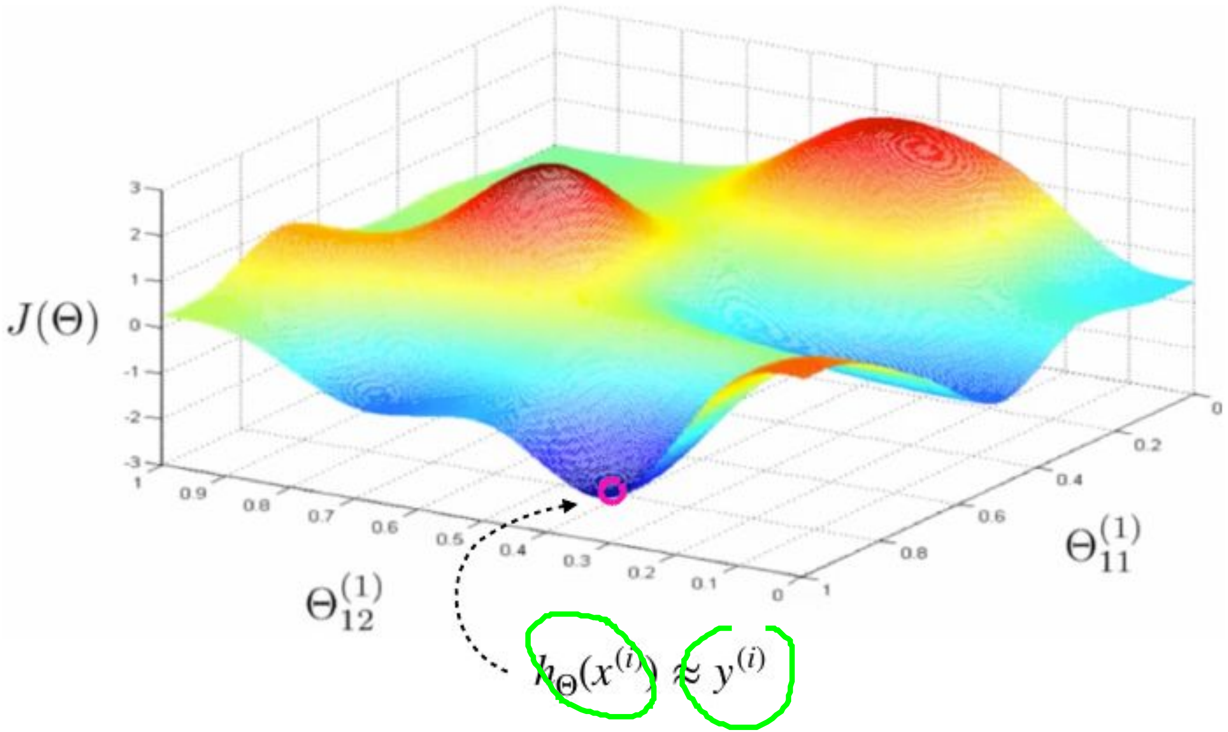
หมายเหตุ: $J(\Theta)$ เป็นแบบ non-convex ก็คือ ใช้ (batch) gradient descent algorithm สามารถติดอยู่ที่ค่า local optima (ค่าที่ดีที่สุดเฉพาะที่)



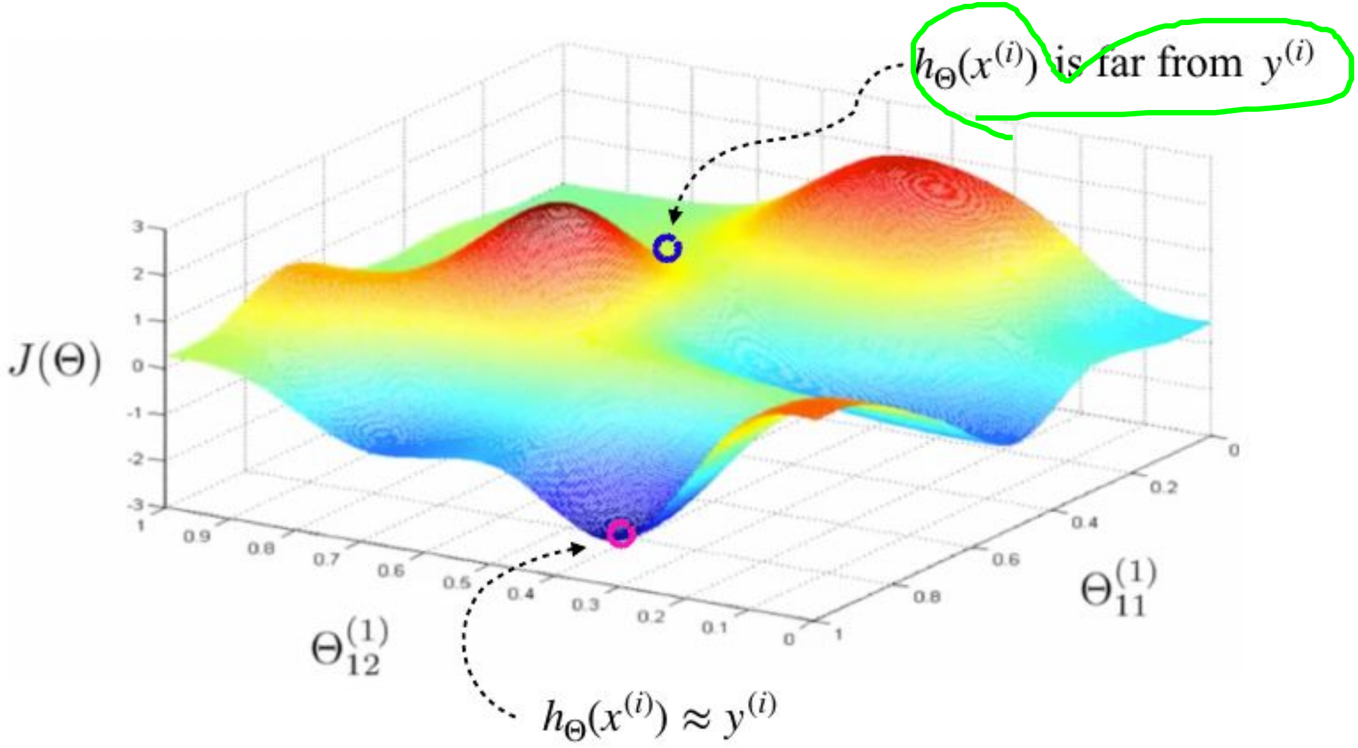
# Backprop with GD (Gradient Descent)



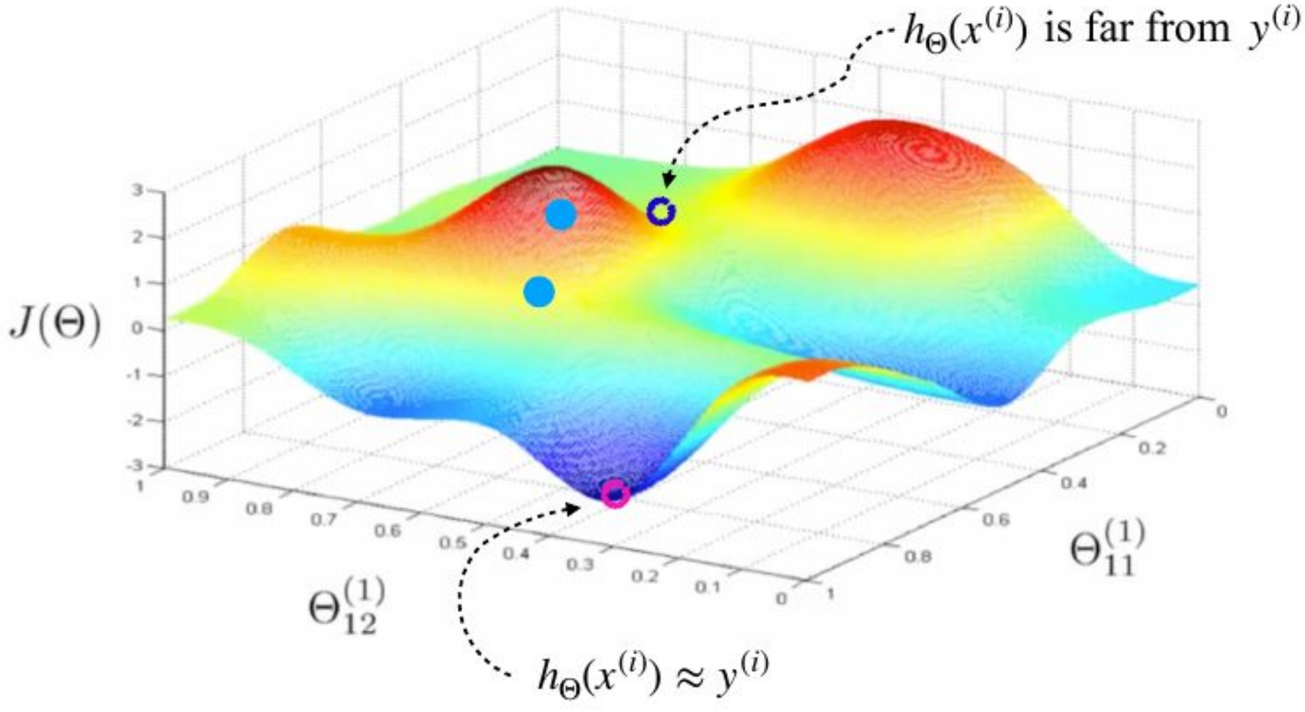
# Backprop with GD (Gradient Descent)



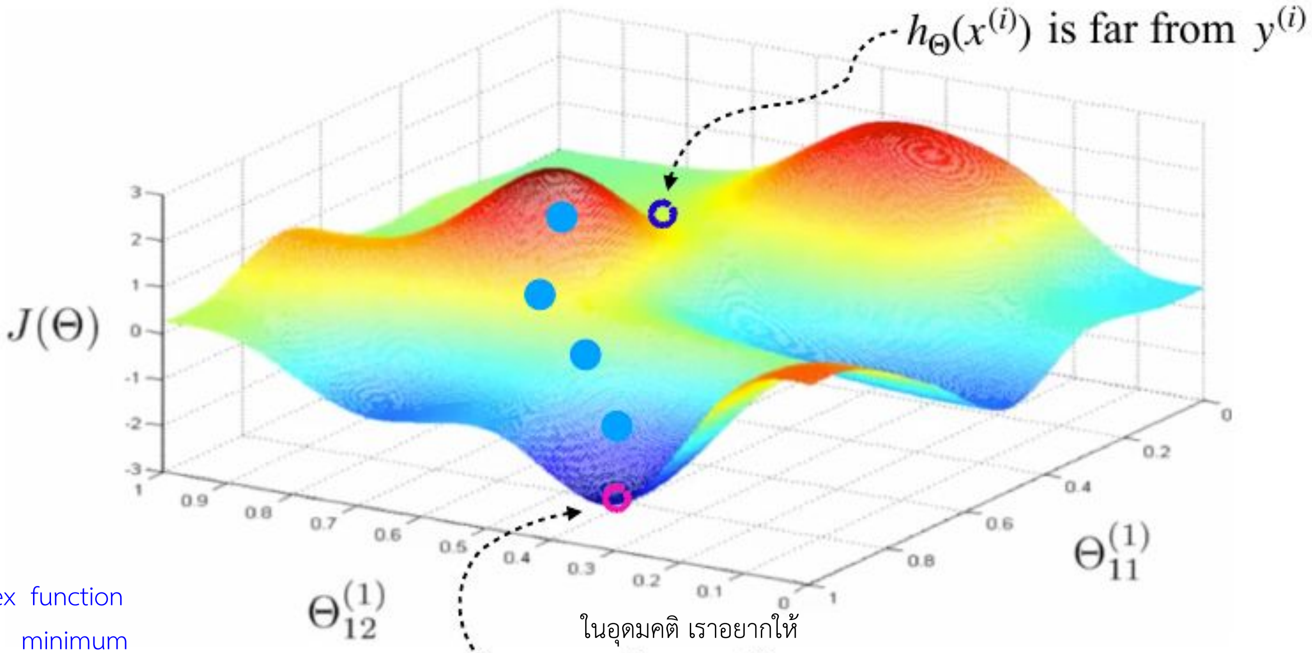
# Backprop with GD (Gradient Descent)



# Backprop with GD (Gradient Descent)



# Backprop with GD (Gradient Descent)




แต่ $J(\Theta)$ ไม่ใช่ convex function
และอาจไปหยุดที่ local minimum
แทน

ในอุดมคติ เราอยากให้

$h_\Theta(x^{(i)}) \approx y^{(i)}$

เพราะจะทำให้ cost function น้อยที่สุด (minimize cost function)

# Question

สมมติใช้ gradient descent กับ backpropagation เพื่อพยายาม minimize (ทำให้น้อยที่สุด) $J(\Theta)$
ข้อใดต่อไปนี้เป็นขั้นตอนที่มีประโยชน์กับการยืนยันว่า learning algorithm ทำงานถูกต้อง?

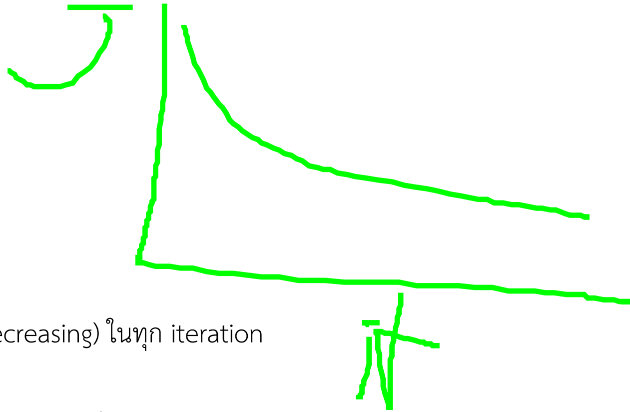
(i) Plot $J(\Theta)$ เป็น function ของ $\Theta$ เพื่อทำให้แน่ใจว่า gradient descent กำลังทำให้ค่า loss / cost ลดลง

(ii) Plot $J(\Theta)$ เป็น function ของจำนวน iterations และทำให้แน่ใจว่ามันเพิ่มขึ้น (หรือ อย่างน้อย ไม่ลดลง / non-decreasing) ในทุก iteration

(iii) Plot $J(\Theta)$ เป็น function ของจำนวน iterations และทำให้แน่ใจว่ามันลดลง (หรือ อย่างน้อย ไม่เพิ่มขึ้น / non-increasing) ในทุก iteration

(iv) Plot $J(\Theta)$ เป็น function ของจำนวน iterations เพื่อทำให้แน่ใจว่าค่า parameter ทำให้ classification accuracy ดีขึ้น

# Question



สมมติใช้ gradient descent กับ backpropagation เพื่อพยายาม minimize (ทำให้น้อยที่สุด) $J(\Theta)$
ข้อใดต่อไปนี้เป็นขั้นตอนที่มีประโยชน์กับการยืนยันว่า learning algorithm ทำงานถูกต้อง?

(i) Plot $J(\Theta)$ เป็น function ของ $\Theta$ เพื่อทำให้แน่ใจว่า gradient descent กำลังทำให้ค่า loss / cost ลดลง

(ii) Plot $J(\Theta)$ เป็น function ของจำนวน iterations และทำให้แน่ใจว่ามันเพิ่มขึ้น (หรือ อย่างน้อย ไม่ลดลง / non-decreasing) ในทุก iteration

(iii) Plot $J(\Theta)$ เป็น function ของจำนวน iterations และทำให้แน่ใจว่ามันลดลง (หรือ อย่างน้อย ไม่เพิ่มขึ้น / non-increasing) ในทุก iteration

(iv) Plot $J(\Theta)$ เป็น function ของจำนวน iterations เพื่อทำให้แน่ใจว่าค่า parameter ทำให้ classification accuracy ดีขึ้น

# ขั้นตอนการ Optimization (Optimization Procedure)

Krittameth Teachasrisaksakul

# ทบทวน: Stochastic vs. Batch

จนถึงบัดนี้ derivation (สิ่งที่หามาได้) ใช้ได้กับตัวอย่าง $(x, y)$ คู่เดียวเท่านั้น

สำหรับ batch gradient descent ต้องใช้กฎ:

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \alpha \frac{\partial J}{\partial \mathbf{W}^{(l)}}$$

เมื่อ $J$ เป็น cost function:

$$J = \frac{1}{m} \Sigma_{i=1}^{m} \mathscr{L}^{(i)}$$

และ $\mathscr{L}^{(i)}$ เป็น loss สำหรับ 1 example (ตัวอย่าง)

Stochastic gradient descent จะ noisy มากกว่า แต่จะ converge เร็วกว่า batch gradient descent

# ทบทวน: Mini-batch

เพราะ batch gradient descent แม่นยำมากกว่า แต่ทำงานช้ากว่า stochastic gradient descent

จุดกึ่งกลางโดยทั่วไป คือ mini-batch gradient descent (GD)

อย่างที่เคยอธิบาย : mini-batch GD เป็น จุดกึ่งกลางระหว่าง ความแม่นยำของ batch GD และความเร็วของ stochastic GD

ก็คือ แบ่งชุดข้อมูลเป็น partitions $B_1, B_2, \ldots$ [หรือ สุ่มตัวอย่าง จำนวนเท่ากัน ซ้ำๆ (repeatedly sample uniformly) จากชุดข้อมูล training set เต็มชุด] และให้:

$$J_i = \frac{1}{|B_i|} \Sigma_{j \in B_i} \mathscr{L}^{(i)}$$

# Momentum

Another common optimization เรียกว่า momentum

ปัญหา คือ บางครั้งข้อมูลที่ noisy ทำให้เรากระโดดกลับไปมาระหว่าง แอ่ง (valley) หลายๆอัน ใน loss function ทำให้ converge ช้า

โดยใช้ค่า momentum : เราจะเก็บค่าที่เปลี่ยนแปลงไป (update) ของ parameter แต่ละตัว และใช้มันคำนวณค่า moving average ของ gradient ในช่วงเวลาหนึ่ง

เราใช้กฎการปรับค่า (update rule):

$$\mathbf{V}_{d\mathbf{W}^{(l)}} := \beta \mathbf{V}_{d\mathbf{W}^{(l)}} + (1-\beta)\frac{\partial J}{\partial \mathbf{W}^{(l)}}$$

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \alpha \mathbf{V}_{d\mathbf{W}^{(l)}}$$

พจน์ momentum $\beta$ จะส่งเสริมให้ optimization เร่งขึ้นที่

# Neural Network : Learning

## การหลีกเลี่ยง Overfitting

Krittameth Teachasrisaksakul

# Overfitting

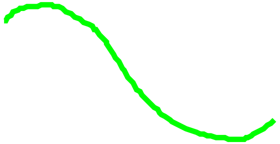
Neural network เป็น approximator (ตัวประมาณค่า) ที่ universal (สามัญ, ใช้ได้ทั่วไป)

อธิบายคร่าวๆ: ถ้ามีข้อมูล training เพียงพอ และ model complexity (ความซับซ้อนของ model) เพียงพอ : backpropagation สามารถเรียนรู้ function ใดๆ เพื่อให้ได้ความแม่นยำ (accuracy) ที่ระดับใดก็ได้

แต่เมื่อ model complexity มากเกินไป สำหรับขนาดข้อมูล training set เราจะพบการ overfitting ก็คือ accuracy (ความแม่นยำ) ของ training set สูง แต่ accuracy ของ test set ต่ำกว่ามาก

**วิธีแก้ไข:**

1. ลด model complexity (เช่น ลดจำนวน hidden unit / layer)
2. เก็บ training data มากขึ้น
3. ใช้ regularization



# Regularization

ให้ w แทน vector 1 ตัวที่เก็บชุดของ parameter ทั้งหมดใน model ของเรา (ก็คือ และ ทุกตัว) และ ให้ J $w_{ij}^{(l)}$ t functic $b_i^{(l)}$ model

L2 regularization จะเพิ่มพจน์ใหม่ให้กับ cost function:

$$J_{L2} = J + \frac{\lambda}{2}\|\mathbf{w}\|^2 = J + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

ทำไมจึงทำแบบนี้?

- $\lambda = 0$ ทำให้เกิดการเรียนรู้ parameter แบบไม่ใช้ regularization
- ค่า $\lambda$ ที่สูง ส่งเสริมให้เกิด solution (คำตอบ) ที่มี w ต่ำ โดยเฉพาะ มีสมาชิกเป็น 0 จำนวนมากที่สุด เท่าที่เป็นไปได้

การบังคับให้ parameter ที่มีประโยชน์น้อยกว่าเป็น 0 จะช่วยลด model complexity และลด overfitting

# Regularization

Regularizer จะมีผลกระทบต่อ learning rule (กฎการเรียนรู้) อย่างไร?

ก่อนหน้านี้ เรามี

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial J}{\partial \mathbf{w}}$$

ตอนนี้ เรามี

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial J}{\partial \mathbf{w}} - \alpha \frac{\lambda}{2} \frac{\partial \mathbf{w}^T \mathbf{w}}{\partial \mathbf{w}}$$

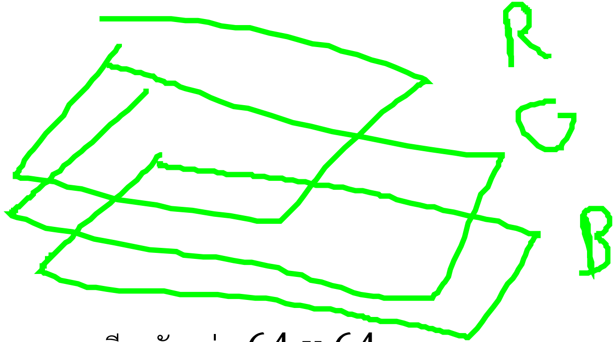
$$\Longleftrightarrow (1 - \alpha\lambda)\mathbf{w} - \alpha \frac{\partial J}{\partial \mathbf{w}}$$

นี่หมายความว่า $\alpha$ และ $\lambda$ เป็น จำนวนจริงบวกที่น้อย

ในแต่ละครั้งที่ปรับค่า (update) : ทำให้ขนาด (magnitude) ของ weight น้อยลงเล็กน้อย

เพียงแค่ค่า weight ที่สำคัญที่สุด จะคงเหลืออยู่

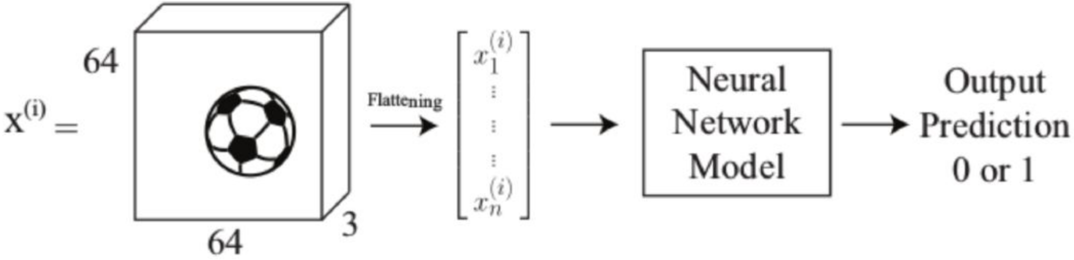
# Parameter Sharing (การใช้ Parameter ร่วมกัน)




ใช้ตัวอย่างของการแยกประเภท (classify) รูปภาพว่า มีลูกฟุตบอล หรือไม่มี

**ขั้นตอน:**

1. Scale (ปรับขนาด) รูปภาพให้เป็นขนาดมาตรฐานเดียวกัน เช่น $64 \times 64$
2. Flatten (แปลง) input ขนาด $64 \times 64 \times 3$ สมาชิก เป็น vector ขนาด $12{,}288$ สมาชิก และส่งให้ neural network



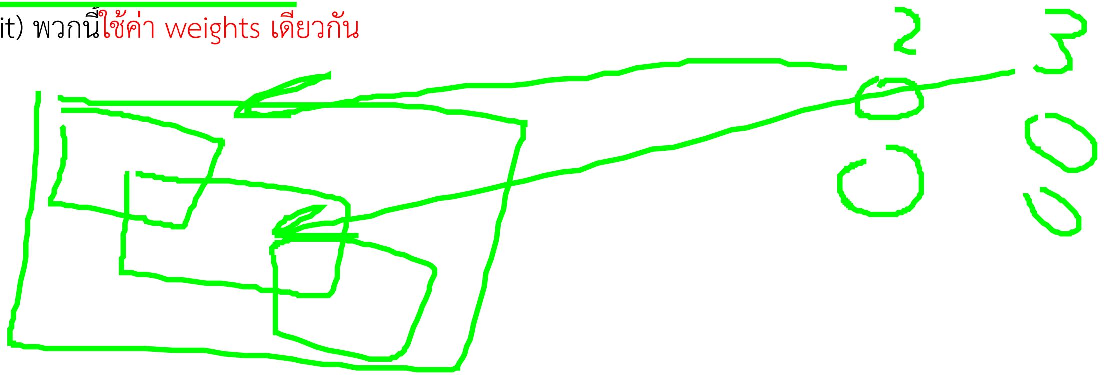

(from Ng (2017), CS 229 Lecture note on Deep Learning)

# Parameter Sharing

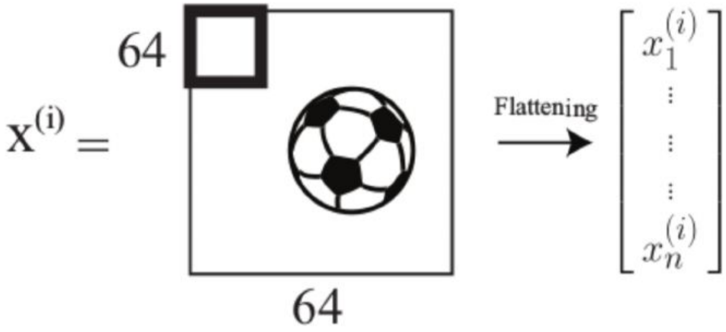
ถ้าใช้ logistic regression model สำหรับ parameter $12,288\ (64 \times 64 \times 3)$ ตัว ของ model : จะต้อง train model ด้วยรูปภาพที่มีลูกบอล อยู่ใน ทุกตำแหน่งที่เป็นไปได้ในภาพ

Model ดังกล่าวจะล้มเหลว ถ้ามันเจอกับรูป ที่มีลูกบอลตรงตำแหน่ง ที่มันไม่เคยเจอมาก่อน ตอน training

วิธีแก้วิธีหนึ่ง คือ parameter sharing คือ แต่ละ unit ใน hidden layer จะตรวจพื้นที่ย่อย (sub-region) ที่ต่างกัน และซ้อนกัน (overlapping) ของรูปภาพ แต่หน่วย (unit) พวกนี้ใช้ค่า weights เดียวกัน



# Parameter Sharing



(from Ng (2017), CS 229 Lecture note on Deep Learning)

เราอาจดึงค่า $\theta$ จาก $\mathbb{R}^{4\times4\times3}$ หมายความว่า จะมีค่า weight สำหรับ intensity (ค่าความเข้มสี) ของ pixel R, G, B ในส่วนพื้นที่ (region) ขนาด 4x4 pixel

# Parameter Sharing

เพื่อเรียนรู้ $\theta$: เราอาจ

- slide region of interest (บริเวณที่เราสนใจ ซึ่งเป็นบริเวณใดๆ ภายในภาพ) บนตำแหน่งแต่ละตำแหน่งของรูปใน training set เพื่อสร้าง training vector ที่มีสมาชิก 48 ตัว หรือ
- เขียนแทนสมาชิกแต่ละตัวใน convolution เป็นหน่วยที่แยกกัน แต่ผูกค่า weight ของพวกมันไว้ด้วยกัน ระหว่างการทำ backpropagation

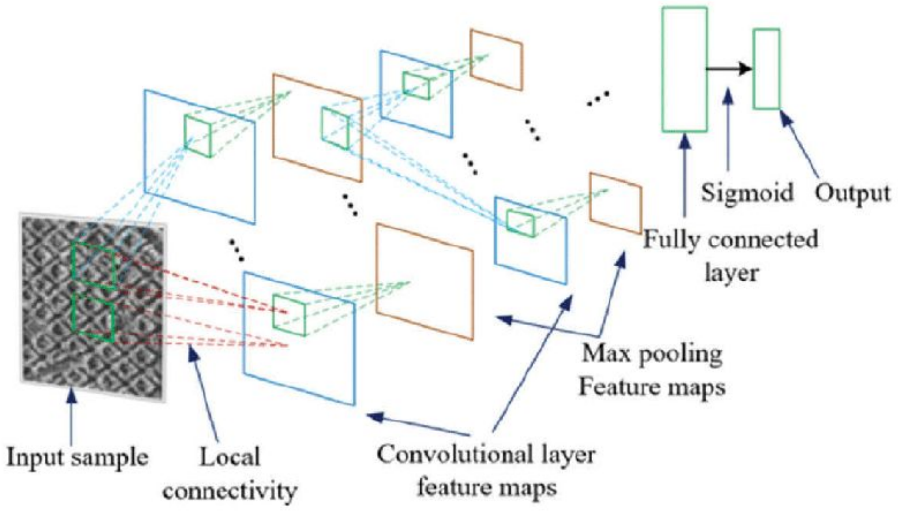
ลักษณะการทำ convolution ของ weight sharing เป็นแนวคิดพื้นฐานของ convolution neural network (CNN):

- ได้รับแรงบันดาลใจจาก model โดย Hubel และ Weisel ซึ่งเป็นการตอบสนอง (response) ของเซลล์ประสาท (neuron) ใน visual cortex (เปลือกสมองส่วนการเห็น) ของแมว ต่อสิ่งกระตุ้นทางการมองเห็น (visual stimuli)
- ได้รับแรงบันดาลใจจาก Neocognitron โดย Fukushima (1988)

# Parameter Sharing

Convolution Neural Networks (CNNs):

- ถูกประยุกต์ใช้สำเร็จกับการจดจำอักขระที่เป็นลายมือ (handwritten character recognition) โดย LeCun และผู้ร่วมงาน (LeNet 5, 1988)
- ชนะการแข่งขัน ImageNet Large Scale Visual Recognition Challenge ในปี 2012 และทุกการแข่งขันของ ImageNet ตั้งแต่นั้นมา
- ทำให้เกิดความตื่นตัว (ได้รับความสนใจ) ในแวดวง machine learning และ AI




(from DOI: 10.1080/2150704X.2017.1335906)

# สรุป

- เราได้เรียนรู้บทนำโดยย่อเกี่ยวกับ deep learning
- หวังว่าจะได้สาระหลัก และรู้ว่าจะขยายผลไปสู่การแก้ปัญหาในโลกแห่งความจริง (real-world problems) อย่างไร
- Further reading (แหล่งการอ่านเพิ่มเติม):
    - เริ่มจากอ่านงานวิจัยเกี่ยวกับ AlexNet แล้วจึงอ่านต่อเรื่องอื่นๆ

# References

1. Andrew Ng, Machine Learning, Coursera.
2. Teeradaj Racharak, AI Practical Development Bootcamp.