PROPAGATION ALGORITHM

ว30263

การเขียนโปรแกรมปัญญาประดิษฐ์

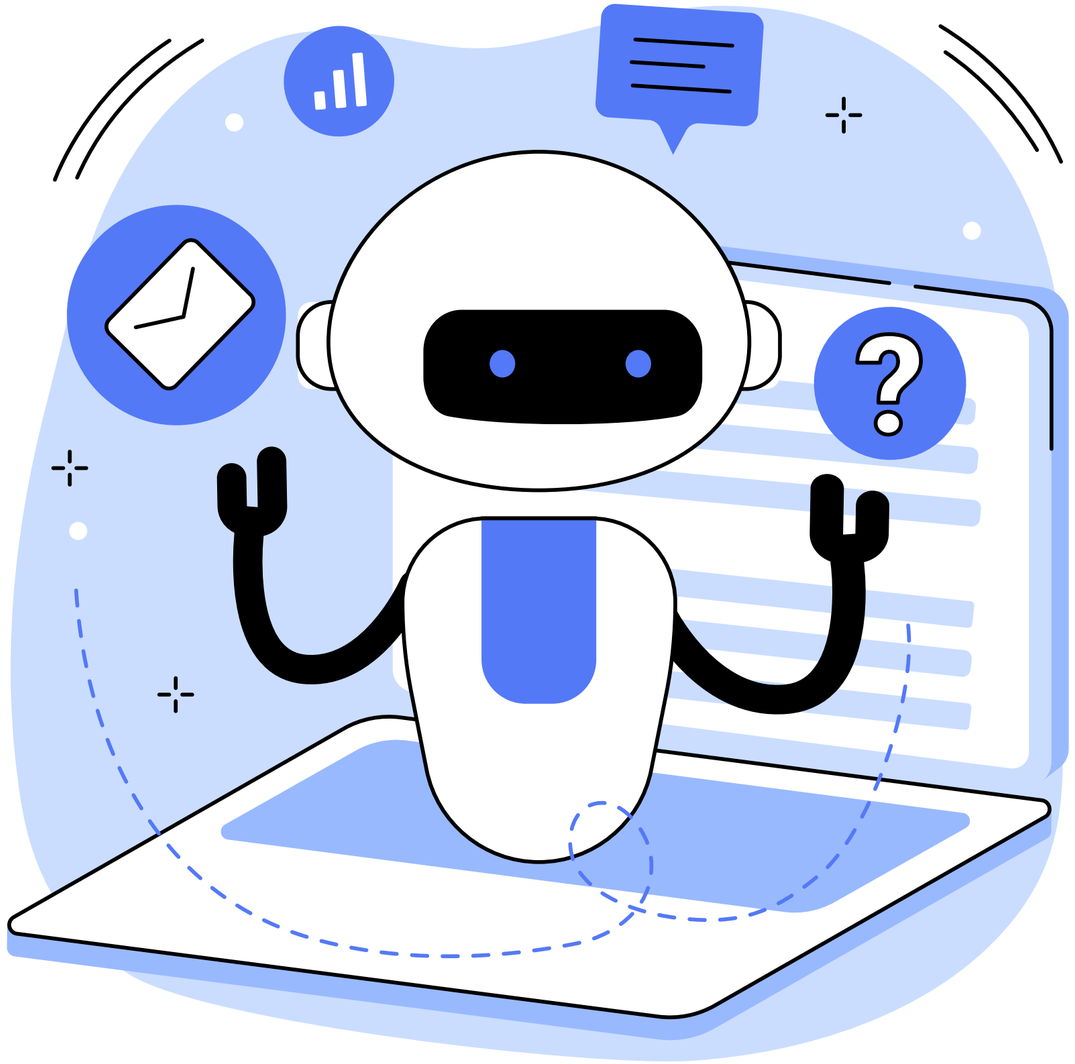
CONTENTS

1 Forward propagation

2 Backward propagation

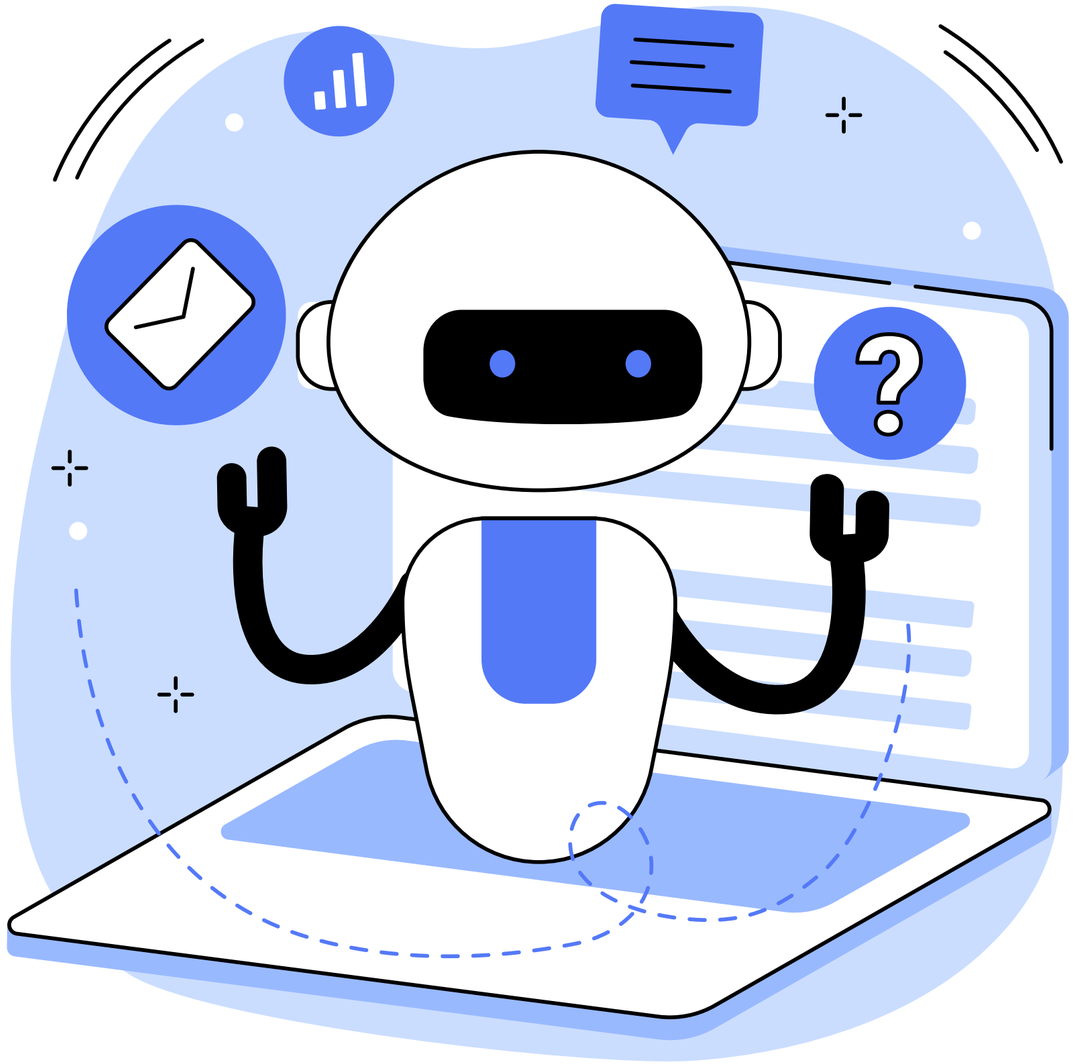
FORWARD PROPAGATION

Algorithm มี Parameter w และ b ที่เป็นตัวแทนของข้อมูลเรียบร้อยแล้ว กระบวนการ Forward propagation คือการนำข้อมูล x เข้ามาประมวลผลร่วมกับ Parameter เหล่านั้นเป็นชั้นๆ จนได้คำตอบ



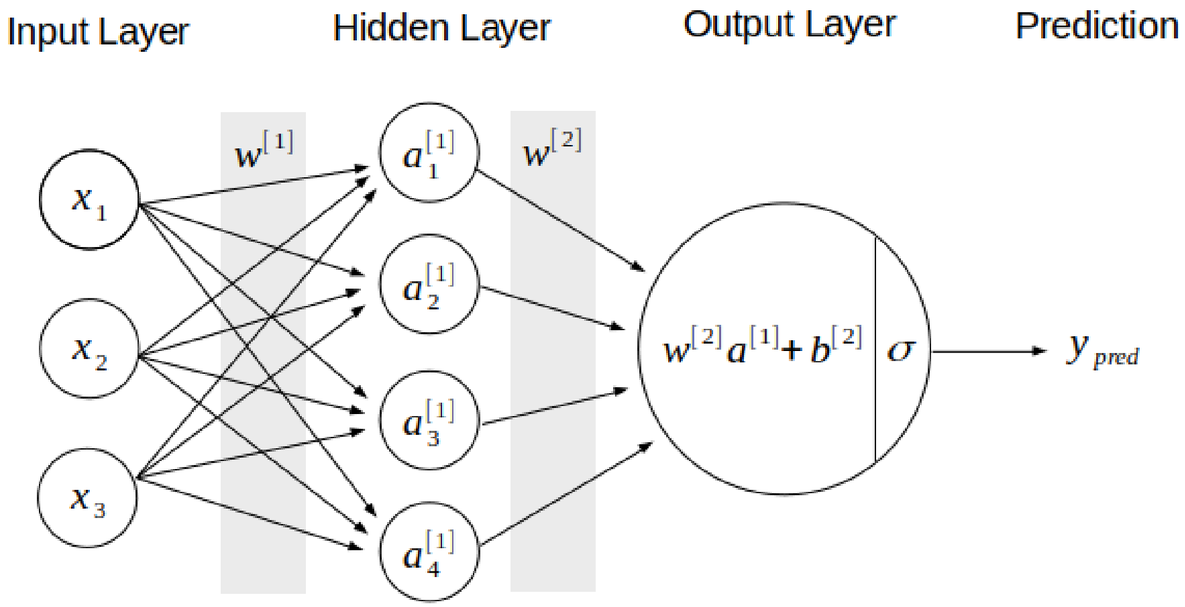
FORWARD PROPAGATION

ในตอนเริ่มต้น โมเดลจะยังไม่มี Parameter ที่ถูกต้อง เราจึงต้องสุ่มค่าเริ่มต้นของ Parameter ขึ้นมาก่อน เมื่อ Forward propagation ทำงานจบ 1 เที่ยว ก็จะเปรียบเทียบผลการพยากรณ์กับคำตอบที่รู้อยู่แล้ว จากนั้นโมเดลจะใช้กระบวนการตรงกันข้าม คือ Backward propagation ในการปรับค่า Parameter ให้สะท้อนข้อมูลใน Train set มากขึ้น ทำอย่างนี้หลายๆ รอบจนกระทั่งได้ความแม่นยำของโมเดลตามที่ต้องการ



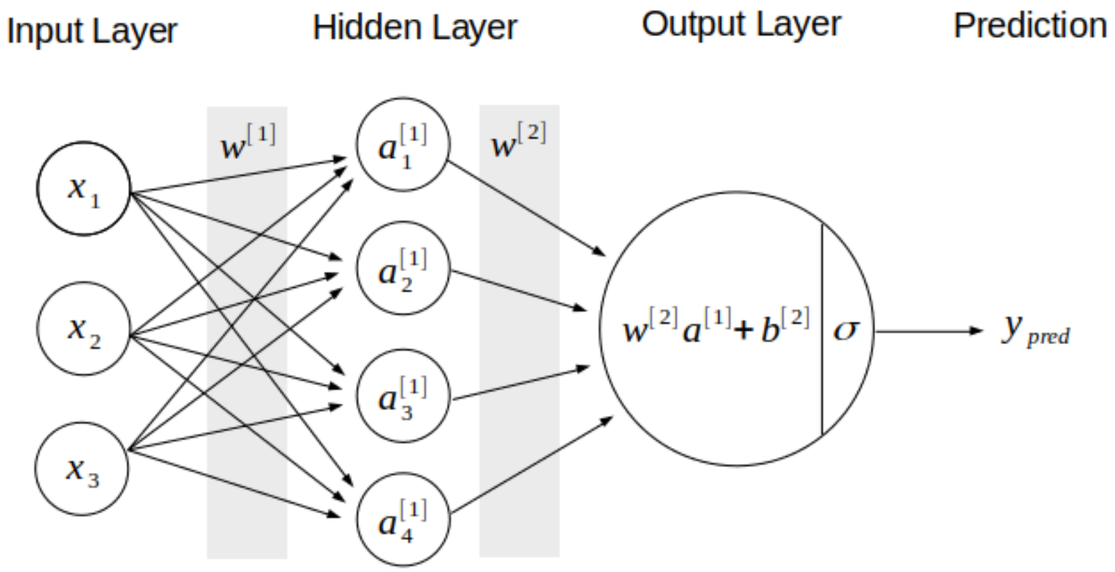
FORWARD PROPAGATION

Input layer คือข้อมูลขาเข้า ถ้าเป็นข้อมูลแบบมีโครงสร้าง x แต่ละตัวจะแทน Feature หรือคอลัมน์ของข้อมูล เช่น อายุ เพศ รายได้



FORWARD PROPAGATION

Hidden layer คือชั้นประมวลผลที่ซ่อนอยู่ ซึ่งมีได้หลายชั้น ใน Hidden layer แต่ละชั้นจะมีหน่วยประมวลผลที่เรียกว่า Neuron



FORWARD PROPAGATION

หน้าที่ของ Neuron แต่ละตัว คือการรับข้อมูล Input "ทุกตัว" จาก Layer ก่อนหน้า มาประมวลผลโดยใช้ Linear function ร่วมกับค่าน้ำหนัก w ของ Input แต่ละตัว ซึ่งให้ผล z แล้วนำ z ไปคำนวนใน Activation function g ซึ่งอาจจะเป็น Sigmoid, Tanh, หรือ RELU ก็ได้ คำตอบที่ได้เรียกว่า a สังเกตว่ากระบวนการนี้คล้ายกับขั้นตอน Classifier ของ ML algorithm

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = g(z^{[1](i)})$$

FORWARD PROPAGATION

- [1] ใน Brackets หมายถึงลำดับที่ของ Layer โดยนับ 1 ที่ Hidden layer แรก
- (i) ใน Parentheses หมายถึงลำดับรายการข้อมูลที่ i
- 1 ที่ห้อยด้านล่าง a คือลำดับที่ของ Neuron ใน Layer นั้นๆ
- นำสัญลักษณ์ทั้งหมดมารวมกัน ตัวอย่างเช่น a[1](4)3 หมายถึง Activation function ตัวที่ 3 ของ Layer ที่ 1

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = g(z^{[1](i)})$$

FORWARD PROPAGATION

สิ่งสำคัญที่ต้องรู้ในขั้นนี้ คือ Hidden layer สามารถมีได้หลายชั้น เช่นถ้าหากมีชั้นที่ 2 สมการที่ (1) ก็จะนำ a[1] มาแทน x และใช้ W และ b ของ Layer ที่สอง โดย [I] คือลำดับที่ของ Layer ซึ่งนับ 1 ที่ Hidden layer ที่ 1

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g(Z^{[l]})$$

FORWARD PROPAGATION

Output layer คือชั้นที่ประมวลผล Activation a ทั้งหมดจากชั้นก่อนหน้า โดยถือว่า a[1] คือ Input ร่วมกับค่าน้ำหนัก W[2] ของ a[1] ในชั้นก่อนหน้า (ไม่ใช่ของ x ในชั้นแรก) ได้ผลเป็น z[2] แล้วนำ z[2] ไปคำนวนใน Activation function เช่น Sigmoid function ได้ผลเป็น a[2]

$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

FORWARD PROPAGATION

ข้อมูลและตัวแปรแต่ละตัวใน Neural network มีหน้าตาอย่างไรในรูปแบบ Matrix และ Vector ข้อมูล Feature n0=4096 จำนวน m=10000 รายการ มิติของ Matrix X คือ (n0,m)= (4096,10000)

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(10000)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \cdots & x_2^{(10000)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} & \cdots & x_3^{(10000)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{4096}^{(1)} & x_{4096}^{(2)} & x_{4096}^{(3)} & \cdots & x_{4096}^{(10000)} \end{pmatrix}$$

FORWARD PROPAGATION

ข้อมูลชุดนี้ใน Neural network ขนาด Layer L=2 โดยมีจำนวน Neuron ใน Layer แรก n1=4 Neuron ส่วน Layer ที่สองเป็น Output layer ดังนั้นจึงมี n2=1ค่าน้ำหนัก W[1] จะต้องคูณแบบ Dot product กับ X ได้ ดังนั้นจึงมีมิติ (n1,n0)=(4,4096)

$$W^{[1]} = \begin{pmatrix} w_1^{[1](1)} & w_1^{[1](2)} & w_1^{[1](3)} & \cdots & w_1^{[1](4096)} \\ w_2^{[1](1)} & w_2^{[1](2)} & w_2^{[1](3)} & \cdots & w_2^{[1](4096)} \\ w_3^{[1](1)} & w_3^{[1](2)} & w_3^{[1](3)} & \cdots & w_3^{[1](4096)} \\ w_4^{[1](1)} & w_4^{[1](2)} & w_4^{[1](3)} & \cdots & w_4^{[1](4096)} \end{pmatrix}$$

FORWARD PROPAGATION

Matrix Representation of Dot Product

cuemath
THE MATH EXPERT

$$\vec{A}^T = \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} \qquad \vec{B} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

$$\begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} = A_1B_1 + A_2B_2 + A_3B_3 = \vec{A}.\vec{B}$$

Dot Product

$$\begin{bmatrix} a & b \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \bullet \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} aw + by & ax + bz \\ cw + dy & cx + dz \end{bmatrix}$$

FORWARD PROPAGATION

เมื่อ W[1]X อยู่ในมิติ (n1,n0)×(n0,m)=(n1,m)=(4,10000) ดังนั้น b[1] จึงจะต้องอยู่ในมิติ (n1,1)=(4,1) จึงจะสามารถบวกเข้าไปสมการได้

$$b^{[1]} = \begin{pmatrix} b_1^{[1](1)} \\ b_2^{[1](1)} \\ b_3^{[1](1)} \\ b_4^{[1](1)} \end{pmatrix}$$

FORWARD PROPAGATION

ส่วน A[1] ก็มีมิติ (n1,m) เหมือน Z[1] เพราะเป็นการนำ Z[1] มา Apply activation function ด้วยการ Broadcast ฟังก์ชันเข้าไปใน Matrix

$$A^{[1]} = \begin{pmatrix} g(z_1^{[1](1)}) & g(z_1^{[1](2)}) & g(z_1^{[1](3)}) & \cdots & g(z_1^{[1](10000)}) \\ g(z_2^{[1](1)}) & g(z_2^{[1](2)}) & g(z_2^{[1](3)}) & \cdots & g(z_2^{[1](10000)}) \\ g(z_3^{[1](1)}) & g(z_3^{[1](2)}) & g(z_3^{[1](3)}) & \cdots & g(z_3^{[1](10000)}) \\ g(z_4^{[1](1)}) & g(z_4^{[1](2)}) & g(z_4^{[1](3)}) & \cdots & g(z_4^{[1](10000)}) \end{pmatrix}$$

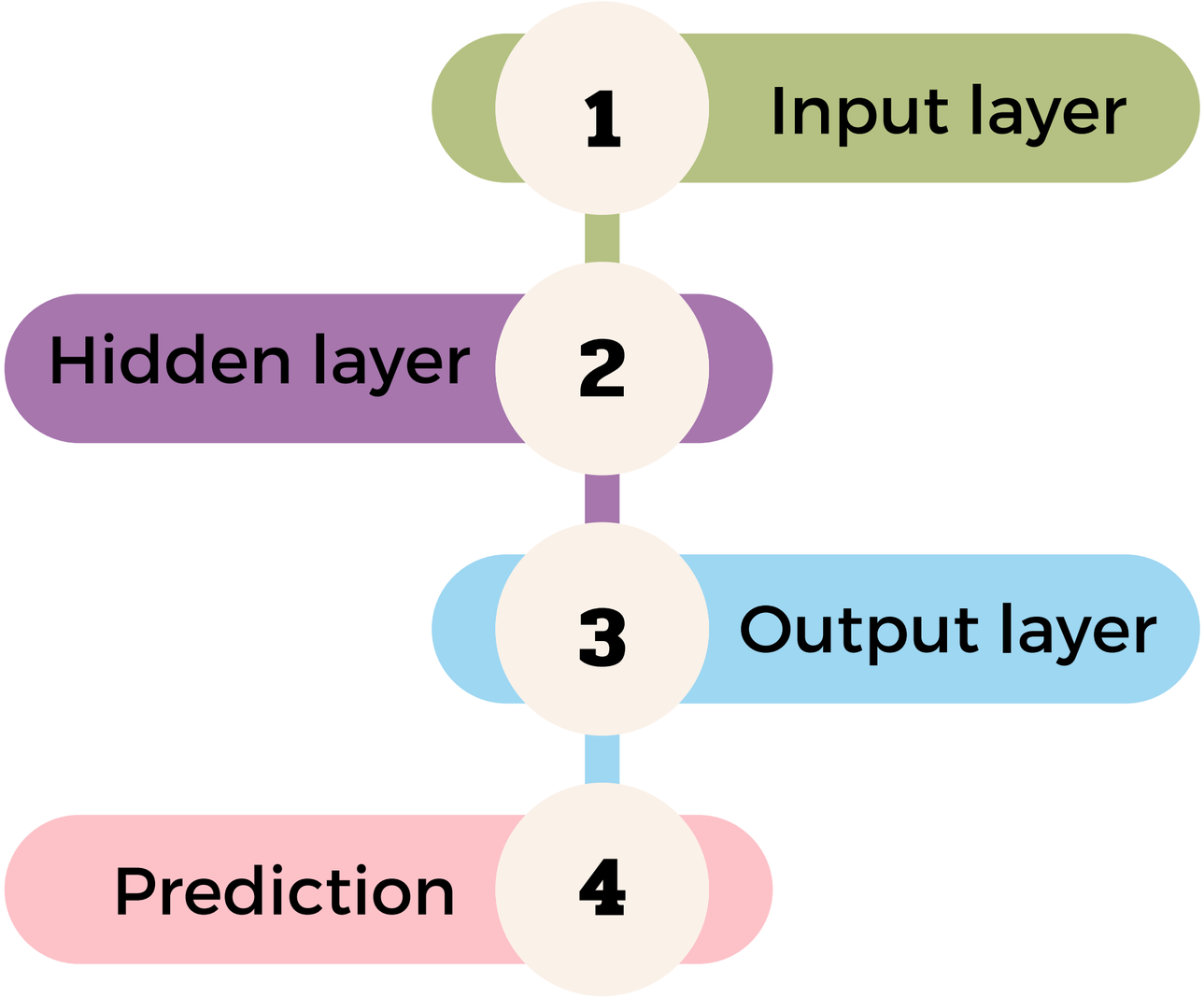
FORWARD PROPAGATION

Prediction นำเอาผล Activation a[2] ของ Output layer มาตัดสินใจใน Decision function เพื่อพยากรณ์ เขียนเป็นสมการได้

$$y^{(i)} = a^{[2](i)}$$

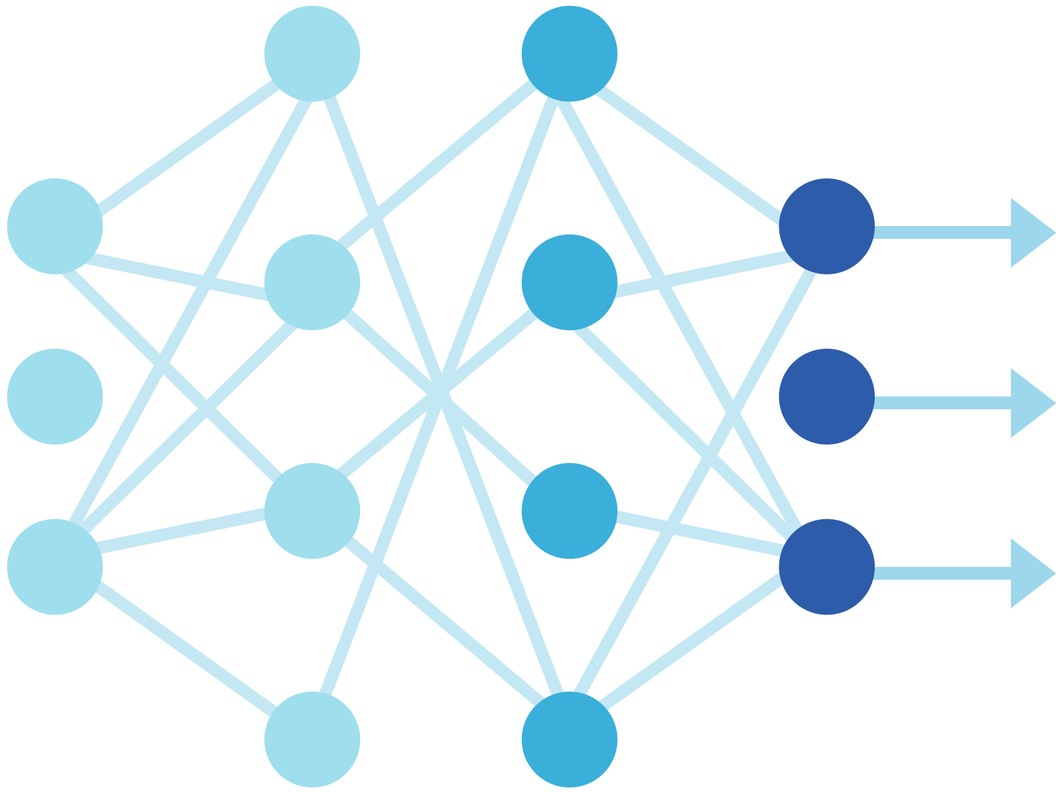
$$y^{(i)}_{predict} = \begin{cases} 1 & \text{if } a^{[2](i)} \geq 0.5 \\ 0 & \text{if } a^{[2](i)} < 0.5 \end{cases}$$

EXAMPLES OF ARTIFICIAL INTELLIGENCE



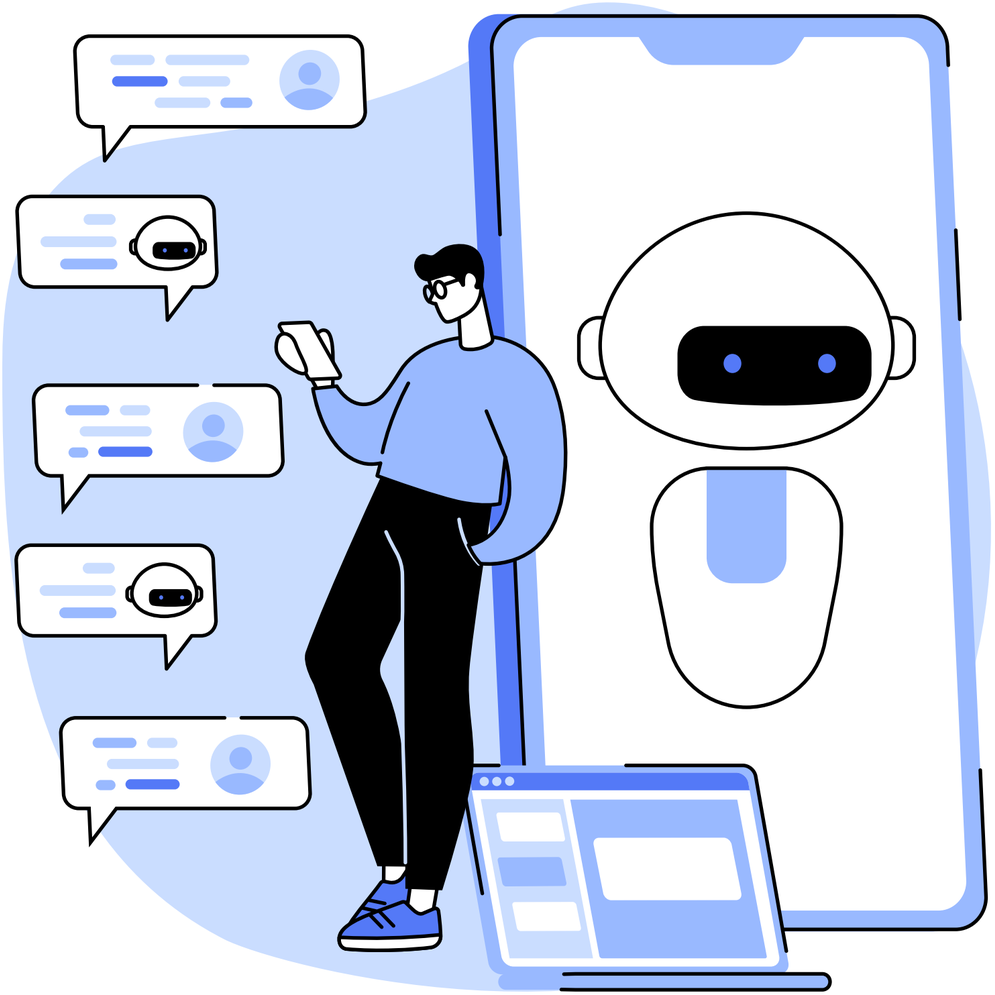
FORWARD PROPAGATION

กระบวนการทั้งหมดนี้ เรียกว่า Forward propagation ซึ่งจบลงด้วยการได้ค่าพยากรณ์ แต่แน่นอนว่าเมื่อยังไม่มี Parameter w และ b ที่ถูกต้อง ค่าที่พยากรณ์ได้ก็จะไม่ตรงกับความจริง ดังนั้นเราจะใช้กระบวนการ Backward propagation ในการปรับแต่ง Parameter ให้เป็นตัวแทนของข้อมูลได้เที่ยงตรงยิ่งขึ้น



BACKWARD PROPAGATION

Backward propagation คือส่วนที่ซับซ้อนและยากที่สุดของ Neural network algorithm



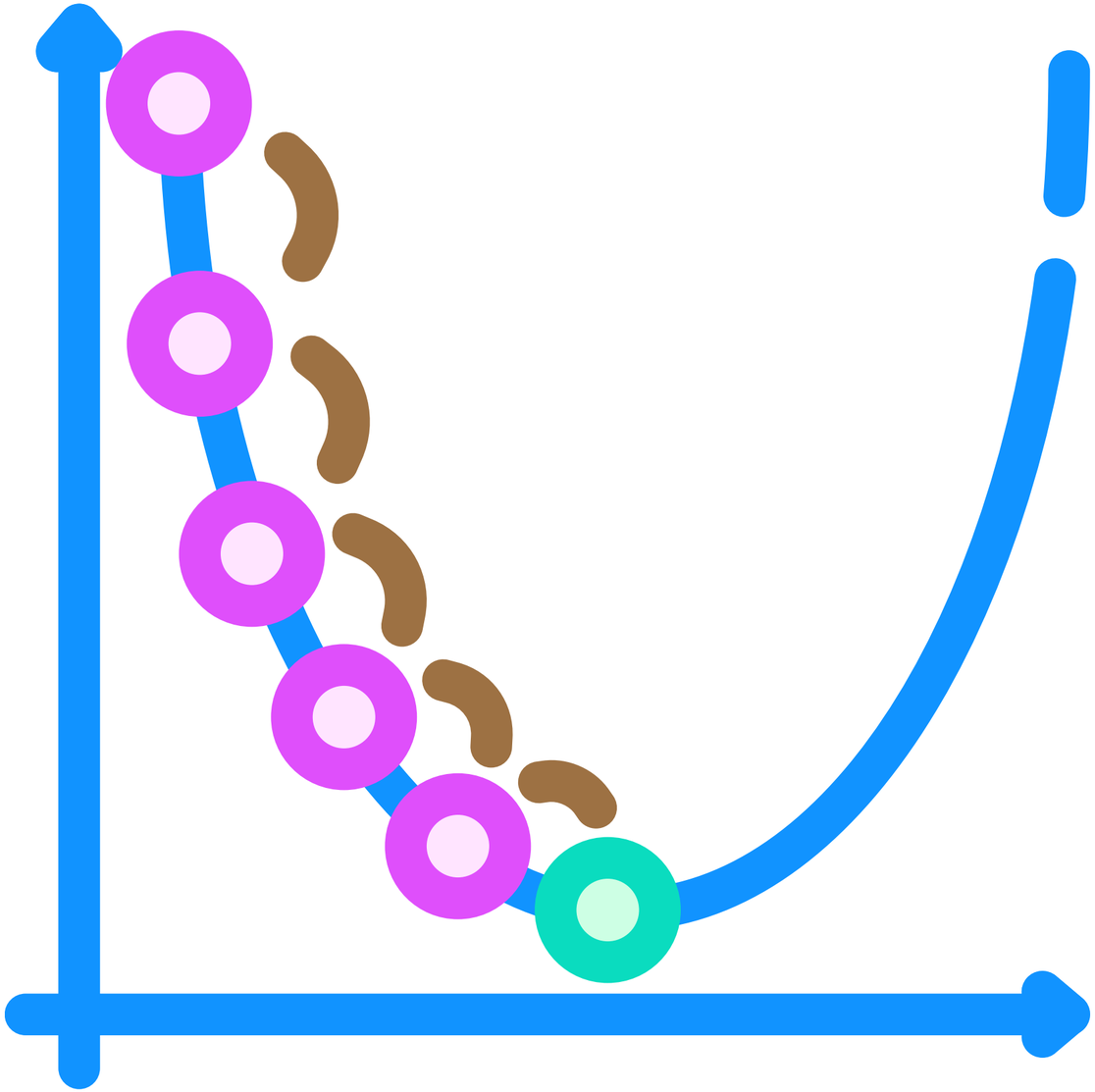
BACKWARD PROPAGATION

สามารถเขียนโค้ดของ **Cost function** โดยใช้ **numpy** ช่วย **Vectorise** เพื่อให้สามารถคำนวน **Cost function** ของรายการข้อมูลทั้งหมดได้อย่างรวดเร็วโดยไม่ต้องใช้ **For loop**

```
logprobs = np.multiply(np.log(A2),Y) + np.multiply(np.log(1-A2),1-Y)
cost = (-1/m)*np.sum(logprobs)
```

BACKWARD PROPAGATION

จะหาว่า Parameter ที่ทำให้ Cost function มีค่าต่ำที่สุด โดยใช้กระบวนการ Gradient descent ซึ่งมีหลักการคือ



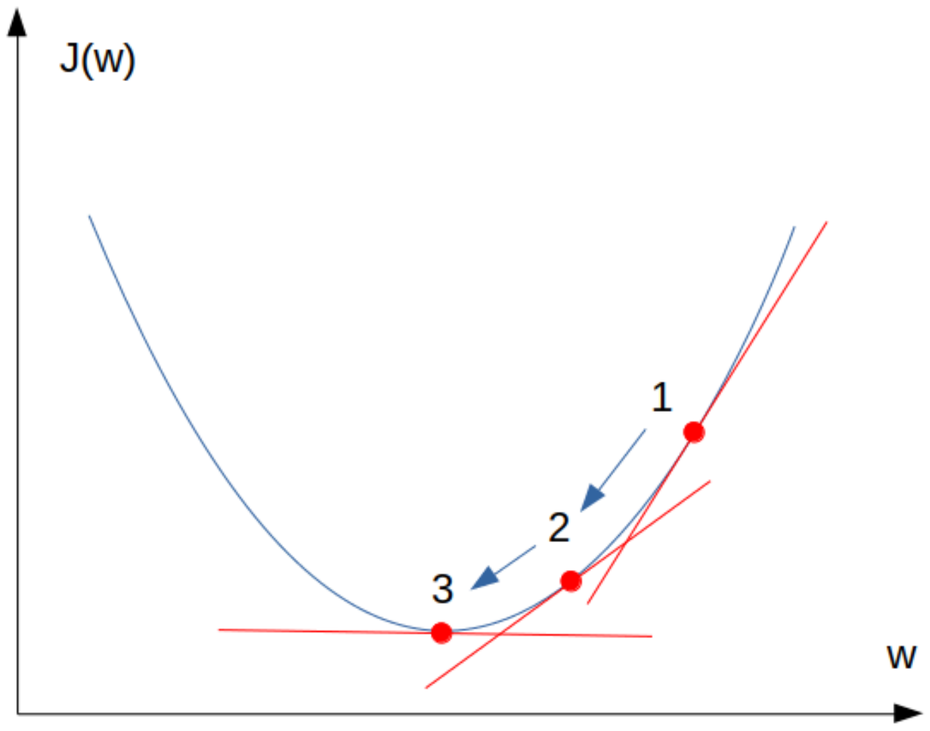
BACKWARD PROPAGATION

นำอนุพันธ์ที่ได้ไปลบออกจาก **Parameter** นั้น โดยควบคุมความเร็วในการลบด้วย **Learning rate** α แล้วนำ **Parameter** ใหม่ไปคำนวนใน **Forward propagation** จะได้ **Cost function** ที่มีค่าลดลง ทำซ้ำขั้นตอนนี้ไปเรื่อยๆ จน **Cost function** มีค่าต่ำที่สุดที่จะเป็นไปได้

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

BACKWARD PROPAGATION

เป็นภาพรวมของกระบวนการ แต่ใน **Neural network** เรามี **Parameter** หลายตัวและหลายชั้น ดังนั้นเราจึงต้องใช้หลายสมการในการหาอนุพันธ์ของ **Parameter** แต่ละตัว



BACKWARD PROPAGATION

โดย Parameter ที่ต้องหาอนุพันธ์ คือ Parameter ที่ส่งผลต่อค่าพยากรณ์ ได้แก่:

- z ของแต่ละ Layer ไม่ได้เป็นตัวแปรที่ต้องการ Optimise แต่จำเป็นต้องหาอนุพันธ์เพื่อจะได้ใช้ Chain rule หาอนุพันธ์ของ w และ b ได้
- w ของแต่ละ Layer เพราะเป็น Coefficient ของ x
- b ของแต่ละ Layer เพราะเป็น Intercept ของ Linear function z=wx+b

หาอนุพันธ์ของ z เช่น dz[2] และ dz[1] เพื่อเป็นอนุพันธ์ตั้งต้นให้หาอนุพันธ์ของ w และ b ได้ตามที่เขียนไว้

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]}$$

$$dW^{[1]} = dz^{[1]} x^{T}$$

$$db^{[1]} = dz^{[1]}$$

BACKWARD PROPAGATION

อนุพันธ์ทั้ง 6 ตัว (สำหรับโมเดลความลึก 2 ชั้น ถ้า 3 ชั้นก็ต้องเพิ่มอีก 3 ตัว) เราก็จะเอาอนุพันธ์ของ w และ b ไปอัปเดตค่าตัวแปรทั้งสอง

$$W^{[1]} := W^{[1]} - \alpha(dW^{[1]})$$

$$b^{[1]} := b^{[1]} - \alpha(db^{[1]})$$

$$W^{[2]} := W^{[2]} - \alpha(dW^{[2]})$$

$$b^{[2]} := b^{[2]} - \alpha(db^{[2]})$$

BACKWARD PROPAGATION

แล้วนำตัวแปรที่อัปเดตแล้วไปคำนวน **Forward propagation** ใน **Epoch** ใหม่ แล้วคิดอนุพันธ์ นำอนุพันธ์มาอัปเดตตัวแปร ทำอย่างนี้ซ้ำไปเรื่อยๆ จนถึงจุดที่ **Cost function** มีค่าต่ำที่สุด ก็จะได้โมเดลที่ฟิตกับ **Train set** ที่ดีที่สุดเท่าที่จะเป็นไปได้

การเพิ่ม NODE และ LAYER

งาน

- ให้เปรียบเทียบการทำงานโดยให้เพิ่ม **layer** เป็น **1,2,3** และ **layer** ละ **10,100,1000 node** โดยเปรียบเทียบ
  - **Accuracy**
  - ประสิทธิภาพ
    - จำนวน **epoch** ดูว่า คู่ใดต้องการ **epochs** น้อยกว่า ให้ถึงระดับ **accuracy** ที่กำหนด
    - ความเร็วใน ดูว่า คู่ไหนสามารถทำงานได้เร็วกว่า