Neural Networks

อ. ปรัชญ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

Today

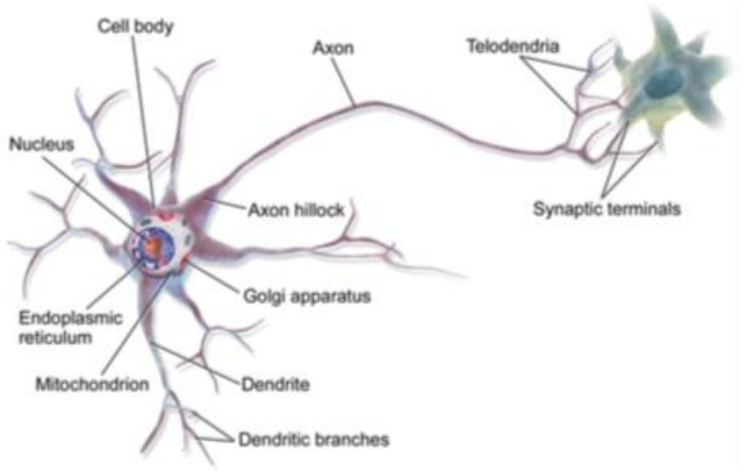
- Artificial Neural Networks
  - Brief History of ANN
  - Applications of ANN
  - Main components of basic ANN model
    - Neuron
    - Weight & Bias
    - Activation Function
    - Layer of Neurons
    - Number of Parameters
  - Training algorithms: Gradient Descent & Backprop
  - Activation Functions & Non-linearity
  - Softmax Layer
  - Feedforward Neural Networks
  - Lab: MNIST with TF/Keras
  - (ไม่ออกสอบ) Historical ANN models: Perceptron, MLP

Artificial Neural Network (โครงข่ายประสาทเทียม)

- เป็น supervised learning
  - ใช้ทำ classification เป็นหลัก (สามารถทำ regression ได้)
  - เป็นวิธีแบบ parametric (มีโมเดล/สมมติฐานเกี่ยวกับข้อมูล)
- ข้อดี
  - ความแม่นยำสูงมาก
  - สามารถนำไปประยุกต์ใช้กับปัญหาที่มี**ขนาดใหญ่**และ**ซับซ้อนมาก**ได้
  - เหมาะกับงานประมวลผลภาพ ภาษา
- ข้อเสีย
  - ต้องใช้ข้อมูลจำนวนมหาศาล, ใช้ทรัพยากรในการประมวลผลสูง (GPU)
  - cost function ไม่ convex (มี minima ได้หลายจุด) ดังนั้นการเลือก initialize ค่าพารามิเตอร์เริ่มต้นจึงมีผลมากต่อคำตอบสุดท้ายของ gradient descent
  - เป็น black box model ตีความ/อธิบายที่มาที่ไปของผลลัพธ์ได้ยาก (ตรงข้ามกับ decision tree)

Origin of Artificial Neural Network

- ได้แรงบันดาลใจมาจาก Biological neurons
- ในช่วงปี 1960s เชื่อกันว่าการสร้าง ANN เพื่อจำลองการคำนวณที่ซับซ้อนของสมองจะนำไปสู่ AI ที่ฉลาดจริง (Connectionism)
- แต่ก็ล้มเหลวไปด้วยข้อจำกัดเหล่านี้
  - Lack of computation power
  - Lack of data
  - Lack of efficient training algorithms
- ปัจจุบัน ANN (ในนาม "Deep Learning") กลับมา dominate สาขา ML อีกครั้ง เนื่องจากเกิด
  - Fast CPU, GPU
  - Big Data
  - Algorithms: Backpropagation, SGD, dropout, Adam

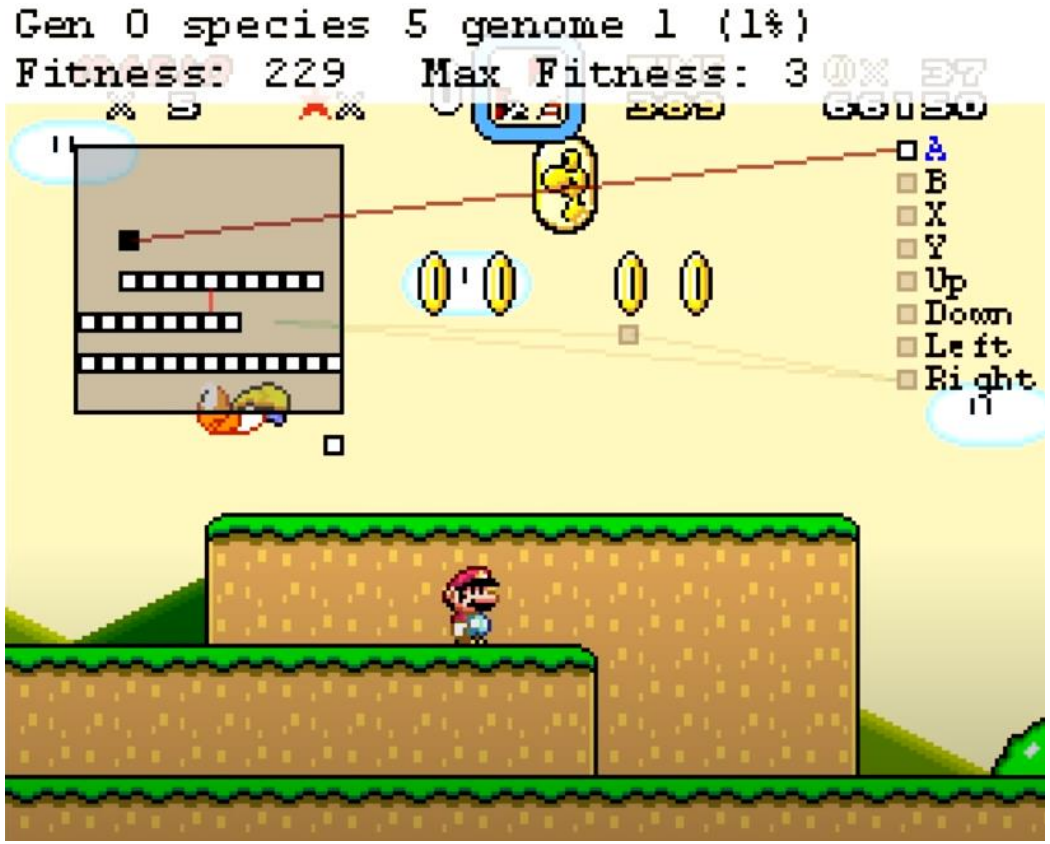


https://en.wikipedia.org/wiki/Neuron

Some Applications of ANN

- Image Classification การจำแนกประเภทรูปภาพ
  - AlexNet: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- Speech Recognition การรู้จำเสียงพูด
  - Apple's Siri: https://machinelearning.apple.com/2017/10/01/hey-siri.html
- Video Recommendation ระบบแนะนำวิดีโอ
  - YouTube: https://ai.google/research/pubs/pub45530
- Games
  - DeepMind's AlphaGo: https://deepmind.com/documents/119/agz_unformatted_nature.pdf
  - Marl/O: https://www.youtube.com/watch?v=qv6UVOQ0F44
  - Atari game: https://www.youtube.com/watch?v=V1eYniJ0Rnk
- Drug Discovery
  - https://lips.cs.princeton.edu/pdfs/duvenaud2015fingerprints.pdf

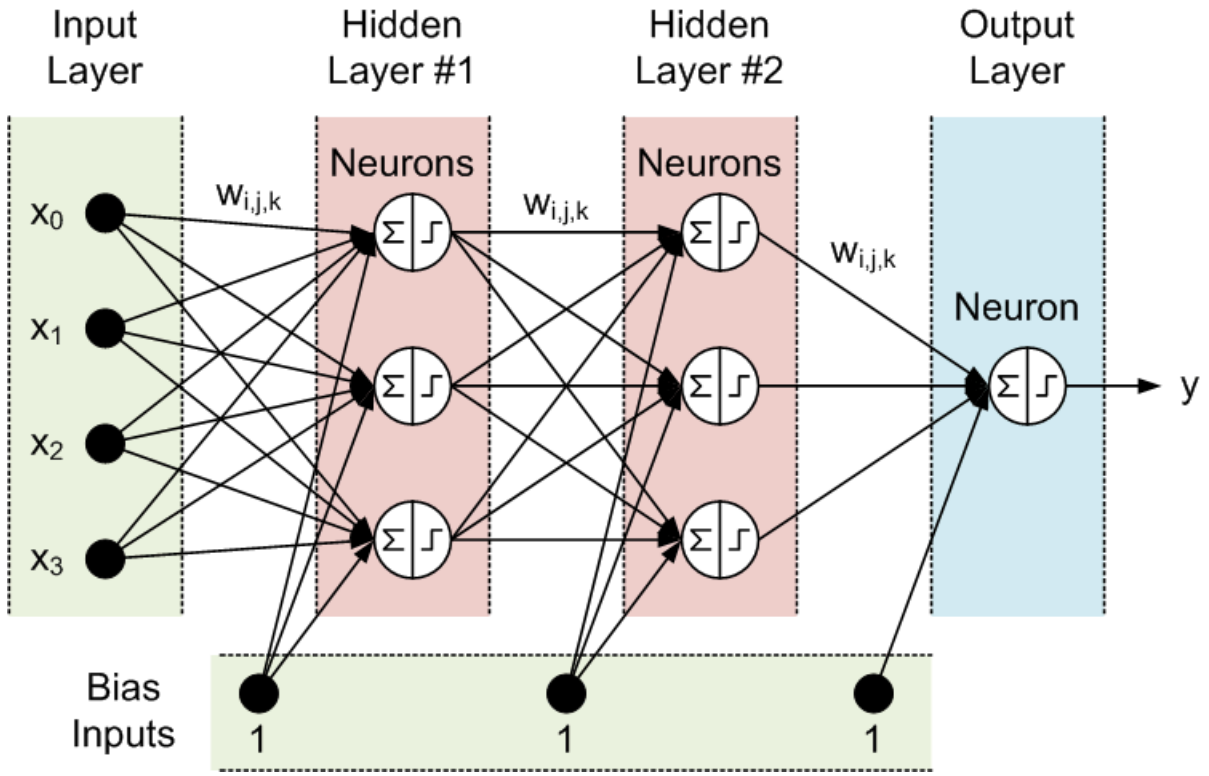
MarI/O



https://www.youtube.com/watch?v=qv6UVOQ0F44

Main components of basic ANN models

- ANN (หรือเรียกสั้นๆ ว่า NN) เป็น model ที่ประกอบด้วย neuron จำนวนมากที่ทำงานร่วมกัน
- คำศัพท์เกี่ยวกับองค์ประกอบของ NN ที่ต้องรู้:
  - Neuron
  - Weight & Bias
  - Activation Function
  - Activation
  - Layer (Input/Hidden/Output)

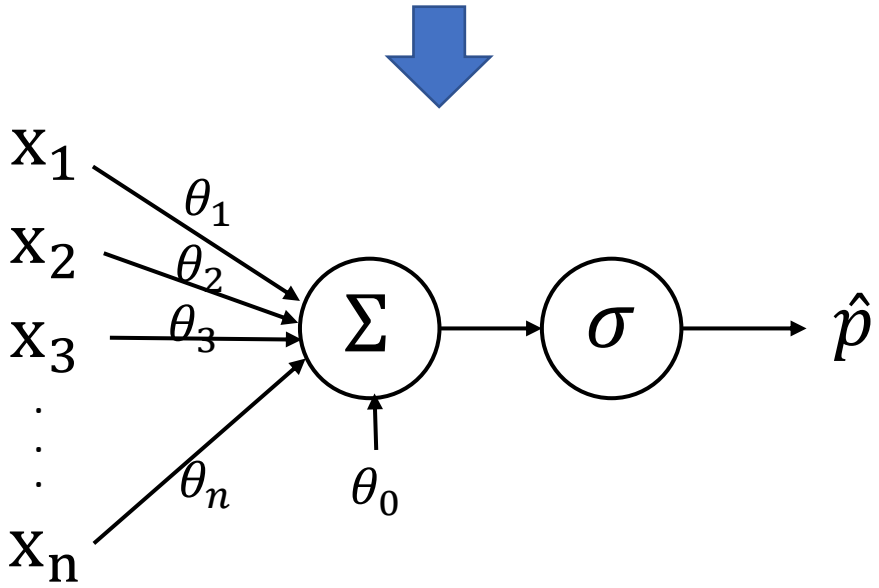


https://stackoverflow.com/questions/28288489/neural-networks-does-the-input-layer-consist-of-neurons

ทบทวน: Logistic Regression as Computation Graph

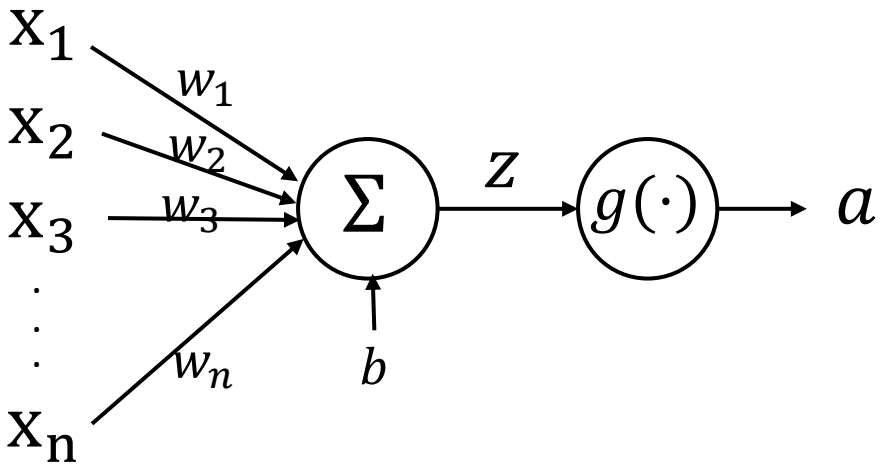
- เราสามารถมองโมเดล **Logistic Regression** เป็น กราฟการคำนวณ ได้

$$\hat{p} = \sigma(\theta^T \mathrm{x})$$
$$= \sigma(\theta_0 + \theta_1 \mathrm{x}_1 + \theta_2 \mathrm{x}_2 + \cdots + \theta_n \mathrm{x}_\mathrm{n})$$



Neuron

- Neuron เป็นหน่วยคำนวณพื้นฐานใน Neural Network (NN)
- Neuron ประกอบด้วย
  - **input**: $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{n}$
  - **weights**: $w_1, w_2, \ldots, w_n$
  - **bias**: b
  - **activation function**: $g(\cdot)$

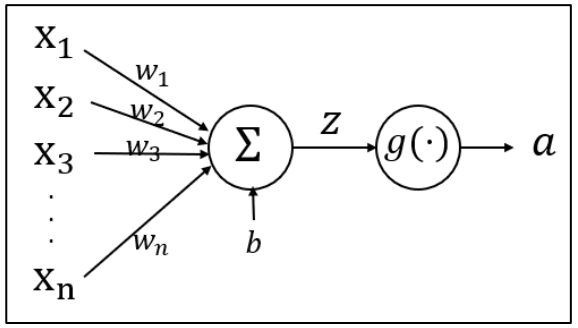


- Neuron จะทำการคำนวณ:
  - $z = w \cdot \mathrm{x} + \mathrm{b} = \mathrm{w}_1\mathrm{x}_1 + \cdots + \mathrm{w}_\mathrm{n}\mathrm{x}_\mathrm{n} + \mathrm{b}$ (z = linear combination of x)
  - $a = g(z) = g(w \cdot \mathrm{x} + b)$ (a = activation of neuron)

- Neuron สามารถทำหน้าที่เป็น binary linear classifier (เหมือน logistic regression)

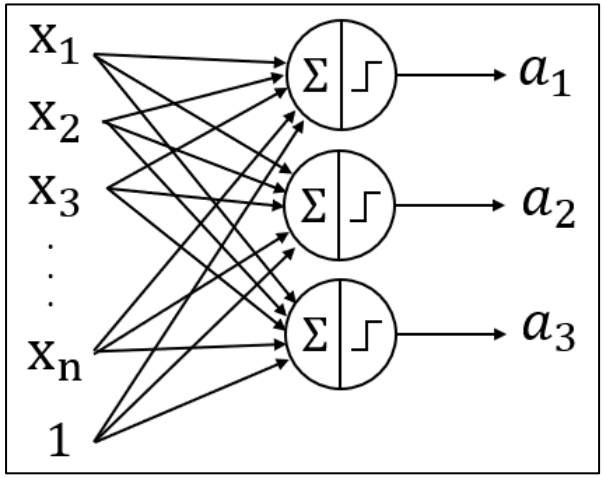
Layer of Neurons

- Layer คือ ชั้นของ NN ที่ประกอบด้วย Neuron หลายๆ ตัว
  - ใน 1 layer อาจมี 3-4 neurons หรือ >1,000,000 neurons ก็ได้
  - แต่ละ Neuron มีค่า weight เป็นของตัวเอง อิสระจากกัน
  - ใช้หลาย neurons ดีอย่างไร?
    - ทำให้ NN สามารถทำงาน multi-class classification ได้
    - แต่ละ neuron ทำหน้าที่ในการตรวจจับ pattern ที่ต่างกันไปในข้อมูลได้



single neuron

x 3

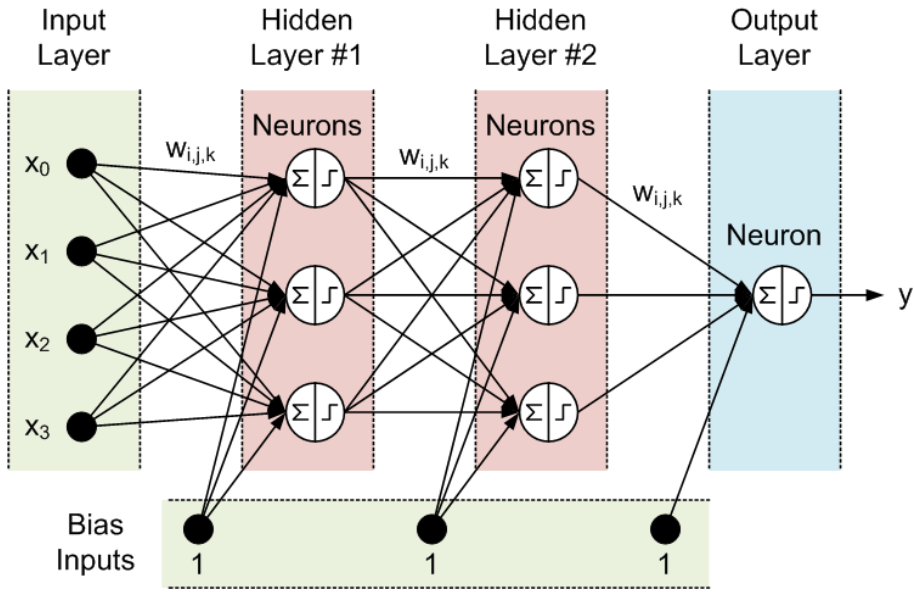


layer of 3 neurons

Trick: เวลานับจำนวน layer ให้นับที่ชั้นเส้นเชื่อม

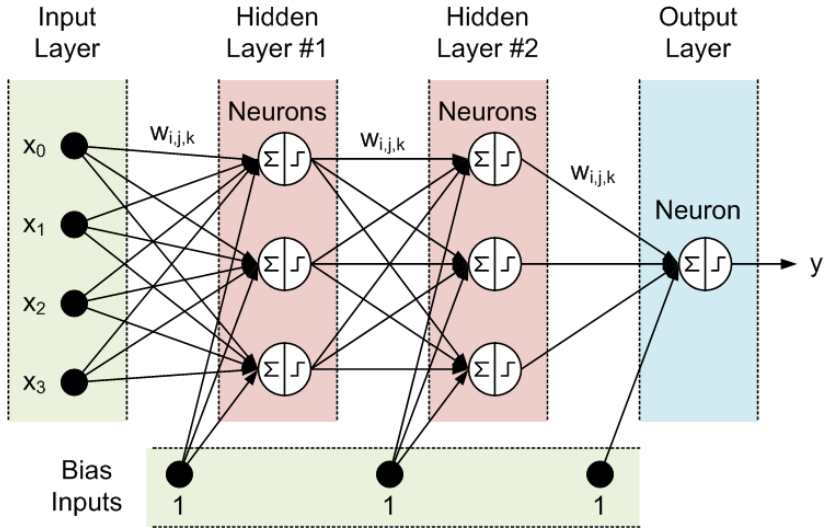
Stacking multiple layers

- NN สามารถประกอบด้วยหลายๆ layers ที่ถูกนำมาพ่วงต่อกัน (stack) ได้
  - ในภาพตัวอย่าง คือ NN ที่มี 3 layers (2 hidden + 1 output, ไม่นับ Input)
  - สังเกตว่า activation ของ layer l ➔ input ของ layer $l+1$
- Layer 3 แบบ:
  - Input Layer คือ layer แรกสุดที่ใช้นำข้อมูลเข้ามา (ไม่มีการคำนวณใดๆ)
  - Output Layer คือ layer สุดท้ายที่ตอบค่า $\hat{y}$ ที่เราต้องการ
  - Hidden layer(s) คือ layer ที่อยู่ระหว่าง input/output layer
- ทุก NN ต้องมี input layer และ output layer เป็นอย่างน้อย
- ลักษณะการเชื่อมต่อเป็นแบบ Fully-connected
  - มีเส้นเชื่อมครบระหว่างทุกคู่ neuron ใน 2 layer ที่ติดกัน



https://stackoverflow.com/questions/28288489/neural-networks-does-the-input-layer-consist-of-neurons

Number of Parameters



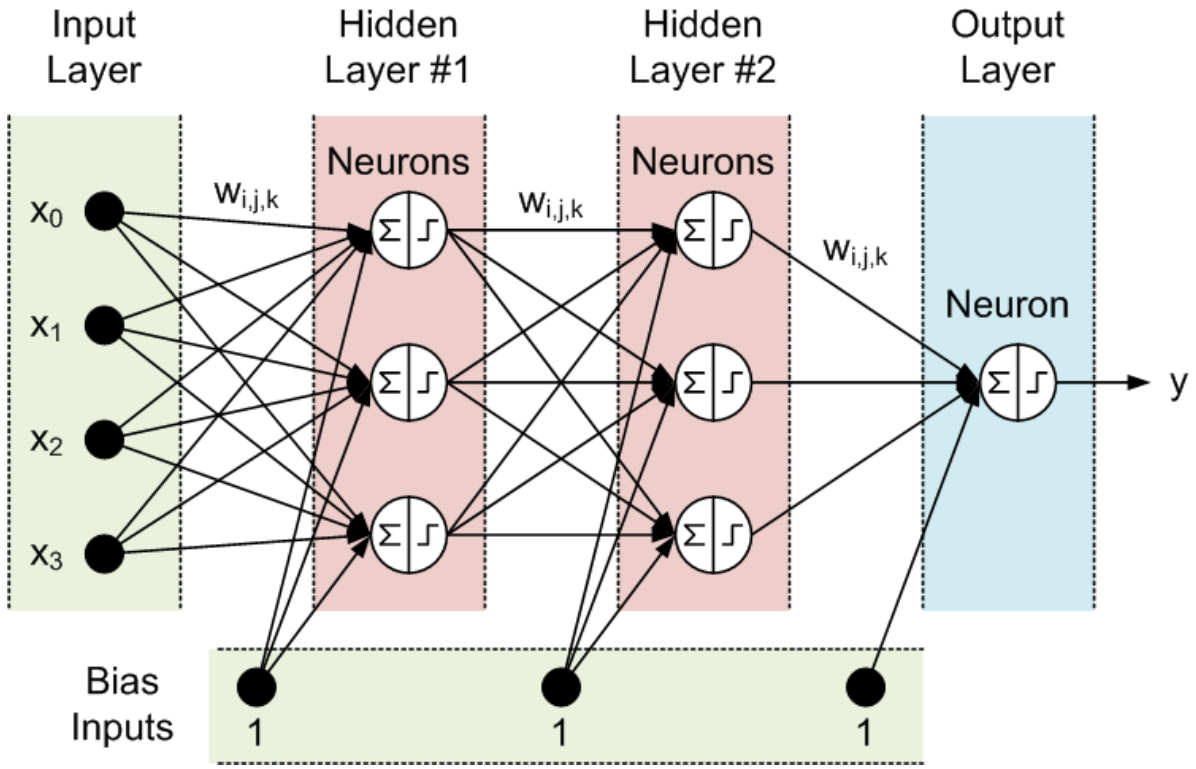
- เราสามารถมอง **NN model** เป็นสมการในรูป
  - $\hat{y} = f(\bar{x})$
  - ซึ่งภายในนั้นมีค่า **weight/bias** จำนวนมาก ที่นำมาใช้คูณ/บวกกับ **input**

- ค่า **weight/bias** คือ **parameter** ของ **NN model**
  - 1 เส้นเชื่อมในภาพ = 1 **parameter** ที่ต้องปรับในระหว่างการ **train**

- ขนาดของ **model** (เล็ก/ใหญ่) วัดจากจำนวน **parameter**
  - เช่น LLaMA (language model ของ Meta AI) มีหลายขนาดให้เลือกใช้
  - 7B, 13B, 33B, 65B

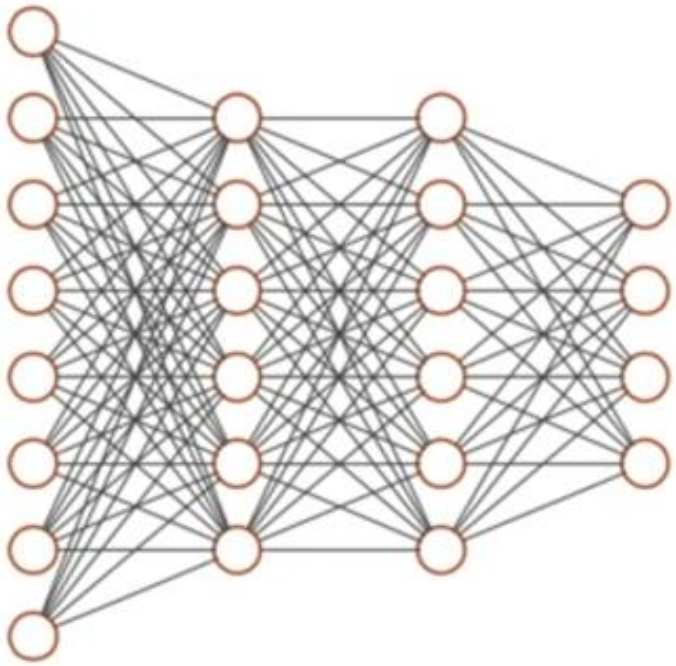
Number of Parameters

- Recap: จำนวน parameter ของ NN คือจำนวนเส้น weight + จำนวน biases
- Exercise 1: จงนับจำนวน parameter ของ Neural Network ในภาพ



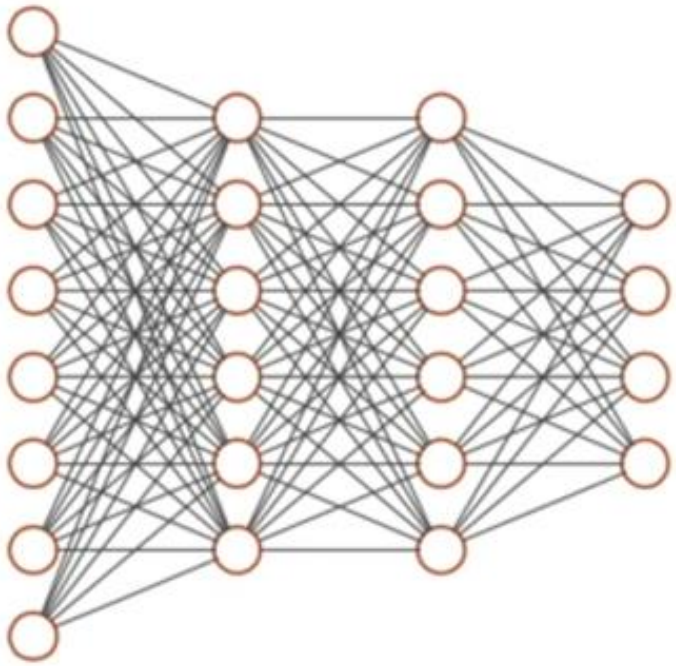
Number of Parameters

- Exercise 2: จงนับจำนวน parameter ของ Neural Network ในภาพ



Number of Parameters

- Exercise 2: จงนับจำนวน parameter ของ Neural Network ในภาพ



จำนวนเส้นเชื่อม = 8*6 + 6*6 + 6*4 = 108

จำนวน bias = 6 + 6 + 4 = 16 (เฉพาะ hidden/output layers)

total params = 108 + 16 = 124

How to train neural networks?

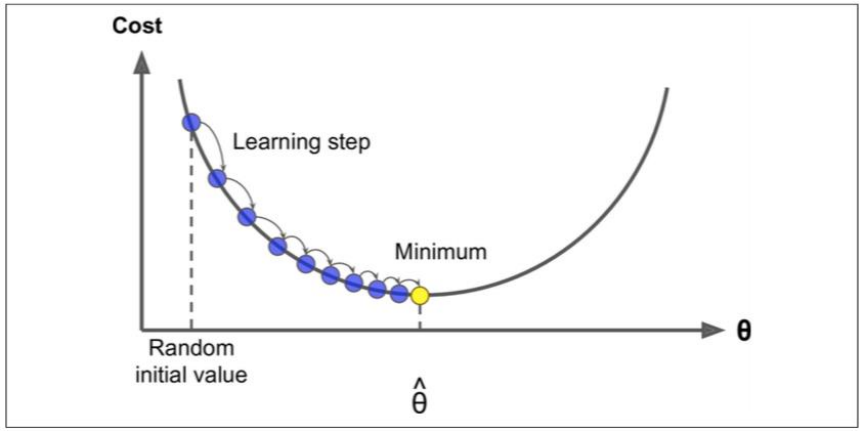
- ทบทวน:
  - ML model จะเก่ง/ไม่เก่งขึ้นกับค่าของ parameter
  - การ train ML model คือการหาค่าของ parameter ทุกตัวที่ทำให้ model ตอบได้แม่นยำที่สุด (= มีค่า cost/loss $\mathcal{L}(\hat{y}, y)$ ต่ำที่สุด)

- เราสามารถ train NN ด้วยวิธี _____ ?

How to train neural networks?

- ทบทวน:
  - ML model จะเก่ง/ไม่เก่งขึ้นกับค่าของ parameter
  - การ train ML model คือการหาค่าของ parameter ทุกตัวที่ทำให้ model ตอบได้แม่นยำที่สุด (= มีค่า cost/loss $\mathcal{L}(\hat{y}, y)$ ต่ำที่สุด)

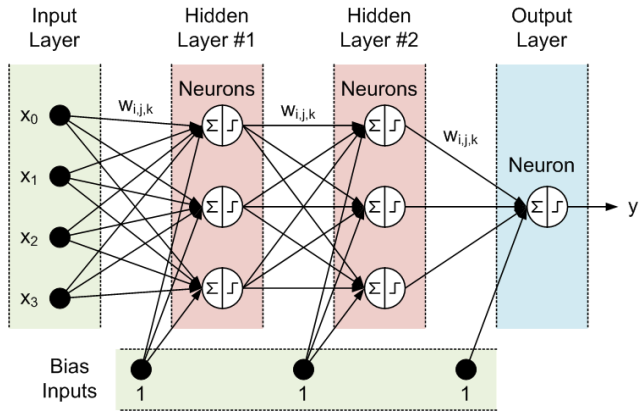
- เราสามารถ train NN ด้วยวิธี gradient descent ได้ ☺
  - เริ่มจาก randomize ค่า **w, b**
  - ค่อยๆ วนปรับ **w, b** ด้วย update rule:
    - $\boldsymbol{w}' = \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})$
    - $\boldsymbol{b}' = \boldsymbol{b} - \eta \nabla_{\boldsymbol{b}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})$
  - จนกว่า cost จะต่ำพอ



What makes training NN difficult?

- ปัญหาที่ทำให้ **gradient descent** ทำได้ยากสำหรับ **NN**
  - ค่า **gradient** $\nabla_w \mathcal{L}(w, b)$ คำนวณยาก
    - เพราะ **loss** $\mathcal{L}$ คำนวณจาก $\hat{y} = f(w, b)$ ซึ่ง f เป็นฟังก์ชันที่ซับซ้อนมากๆ
    - ถ้าคำนวณ **diff** แบบตรงๆ จะใช้เวลานานมาก ☹
  - **Cost function** ของ **NN** เป็นแบบ **non-convex** (มี **minima** ได้หลายจุด)
    - ทำให้ได้คำตอบที่เป็น **local minima** (ไม่ค่อยเก่ง) ☹
  - มีจำนวน **parameter** w, b ที่ต้องปรับเยอะมากๆ
    - ใช้เวลา **train** นาน ☹

ฟังก์ชัน f

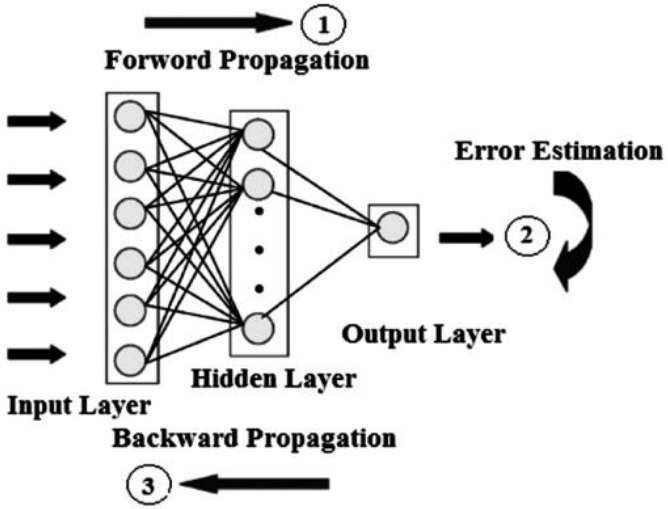


What makes training NN difficult? -- Solutions

- ปัญหาที่ทำให้ gradient descent ทำได้ยากสำหรับ NN
  - ค่า gradient $\nabla_w \mathcal{L}(w, b)$ คำนวณยาก
    - **Solution:** อัลกอริทึม Backprop
  - Cost function ของ NN เป็นแบบ non-convex (มี minima ได้หลายจุด)
    - **Solution:** เป็น advanced topic ไม่ออกสอบ
  - มีจำนวน parameter w, b ที่ต้องปรับเยอะมากๆ
    - **Solution:** parallelize, ใช้ hardware แรงๆ

Backpropagation

- อัลกอริทึม Backprop ใช้เพื่อคำนวณหา $\nabla_W \mathcal{L}$ (gradient ของ loss function w.r.t w)
  - เทคนิค: ใช้ chain rule ใน calculus ทำให้การคำนวณเร็วขึ้นมาก



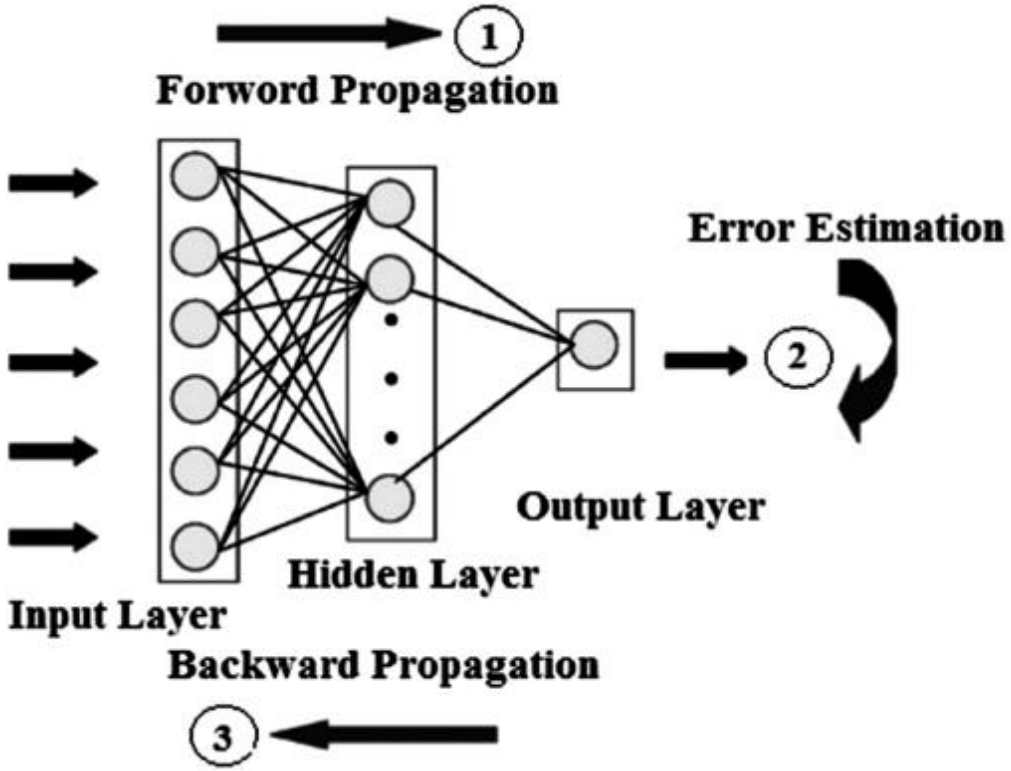
- รายละเอียดการทำงานของ backprop ไม่ออกสอบ หากต้องการเข้าใจ แนะนำให้ดูวิดีโอของ 3Blue1Brown
  - Part 1: https://www.youtube.com/watch?v=Ilg3gGewQ5U
  - Part 2: https://www.youtube.com/watch?v=tIeHLnjs5U8

(ไม่ออกสอบ) Backpropagation Overview

Goal: ต้องการหา matrix $\nabla_W \mathcal{L}$ (ซึ่งก็คือ หา $\frac{\partial \mathcal{L}}{\partial w_i}$ สำหรับทุก w_i)

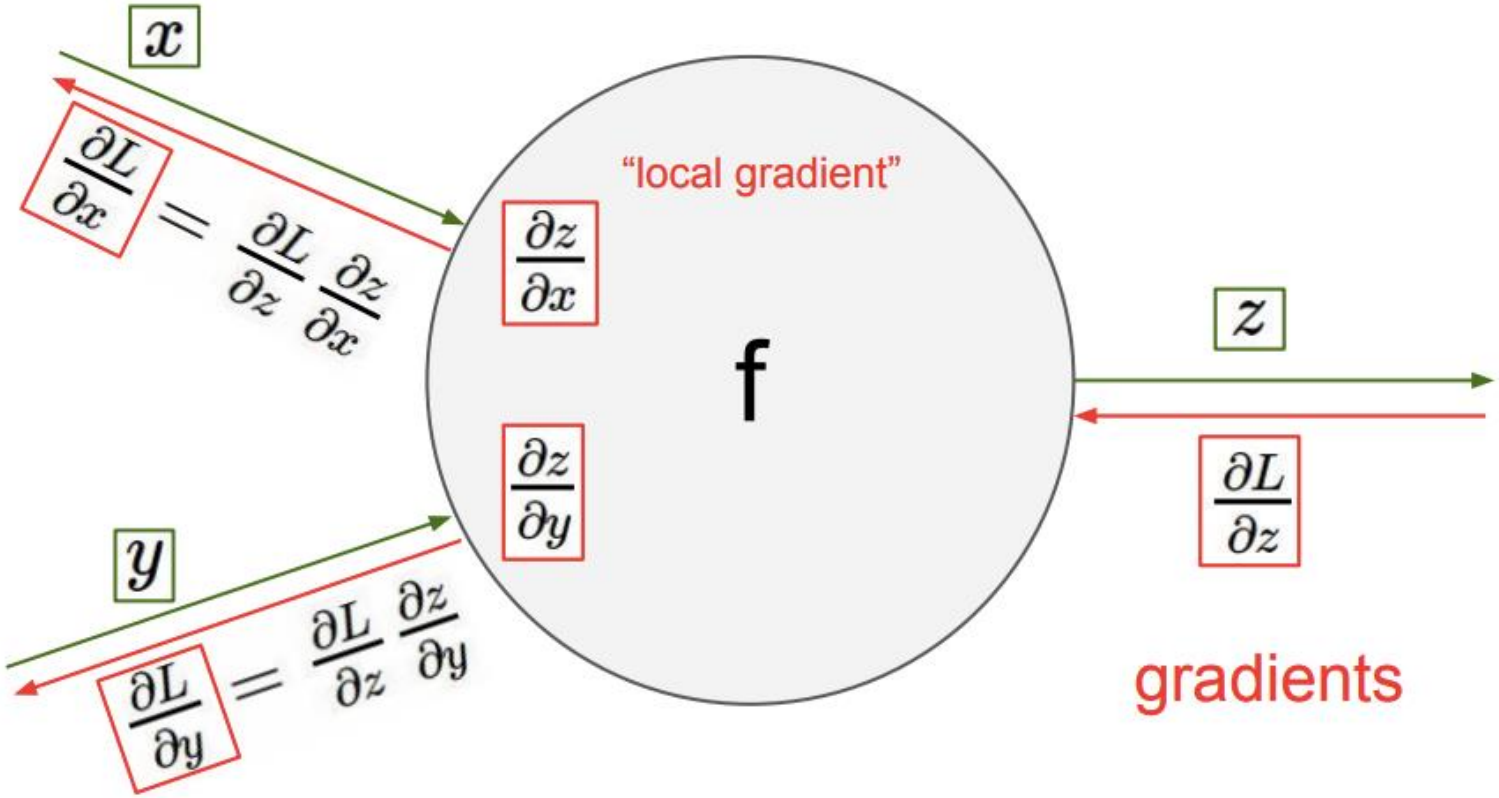
ขั้นตอนวิธีคร่าว ๆ

- for each training instance,
  - make predictions $\hat{y}$ (forward pass)
  - measure loss $\mathcal{L}(\hat{y}, y)$
  - propagate $\frac{\partial \mathcal{L}}{\partial w_i}$ in reverse (apply chain rule) (backward pass)
  - use the computed $\frac{\partial \mathcal{L}}{\partial w_i}$ to adjust w_i (gradient descent)



(ไม่ออกสอบ) Backward Pass

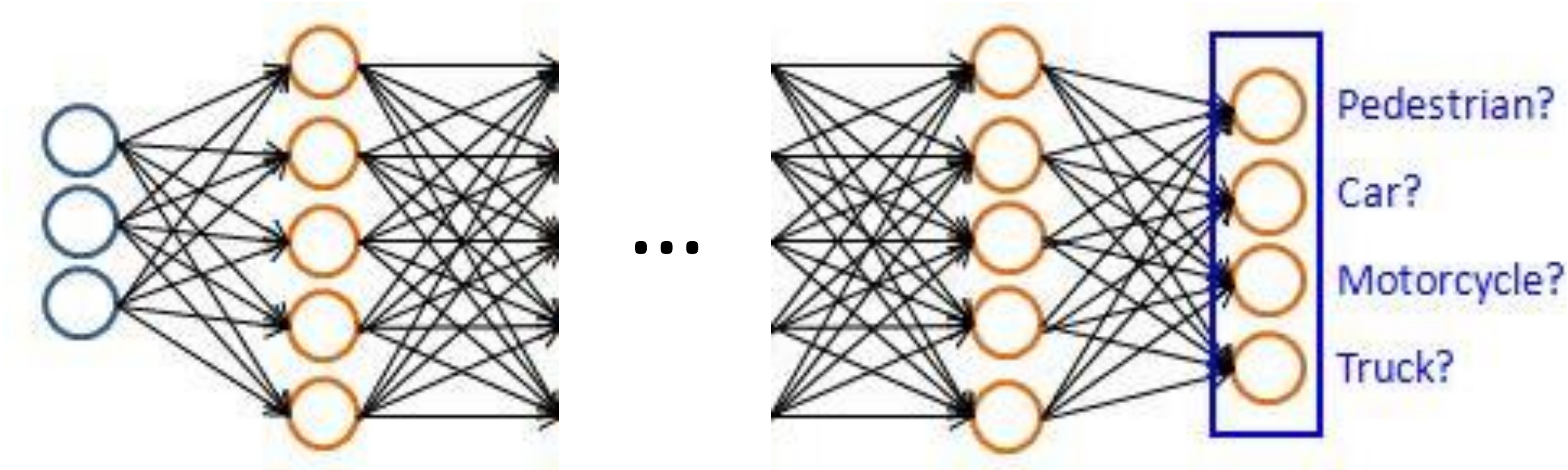
- การคำนวณ **gradient** ใน **backward pass** ด้วย **chain rule**



หากต้องการเข้าใจคณิตศาสตร์โดยละเอียด ให้ดูวิดิโอนี้ https://www.youtube.com/watch?v=Ilg3gGewQ5U

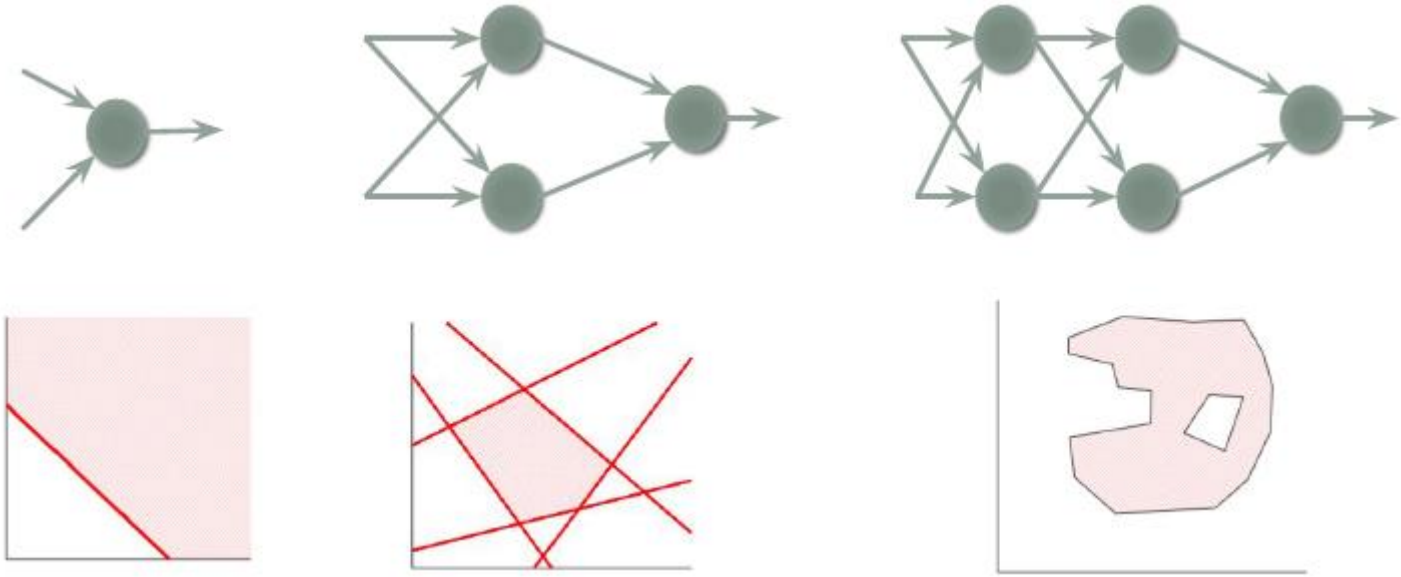
Adding more layers => Deep Neural Networks

Input layer hidden layer 1 hidden layer n output layer



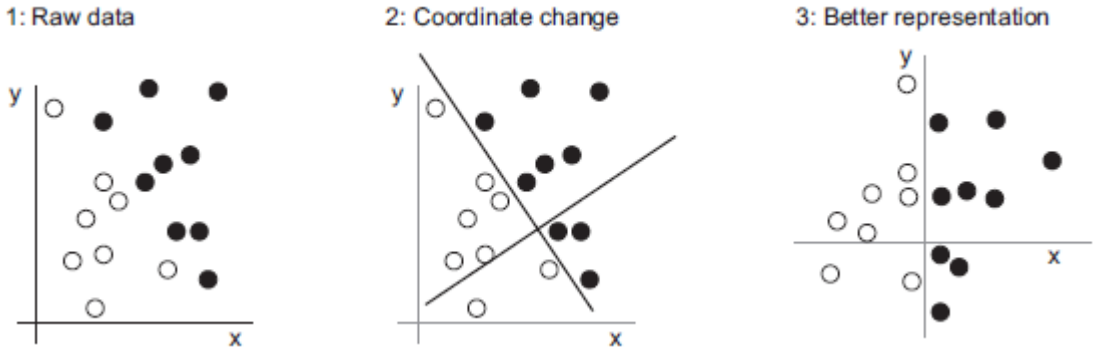
Effect of deep network

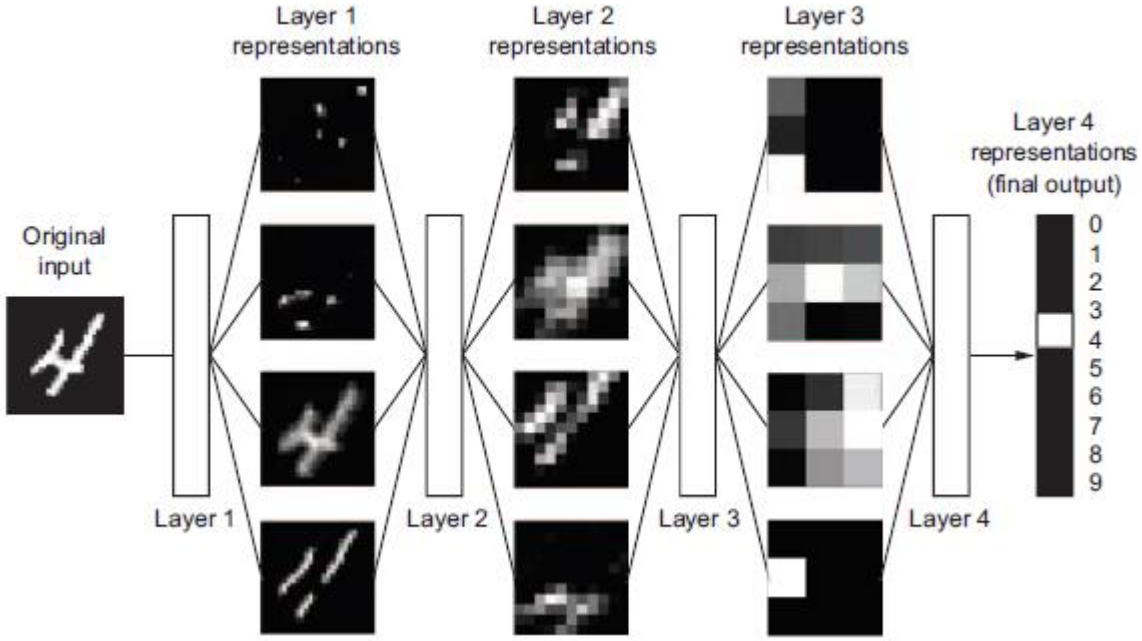
- มองว่าแต่ละ **neuron** จะทำให้เกิด **hyperplane** ที่ผ่าแบ่ง **feature space**
- การที่เรานำ **neuron** หลายๆ **layer** มาซ้อนกัน จะทำให้เกิด **decision boundary** ที่มีความซับซ้อนมากขึ้นได้
  - Powerful, but prone to overfitting (นึกถึง 1-NN)



https://github.com/ekapolc/pattern_course18/blob/master/slides/L08-NeuralNetworks.pdf

(ไม่ออกสอบ) Representation Learning



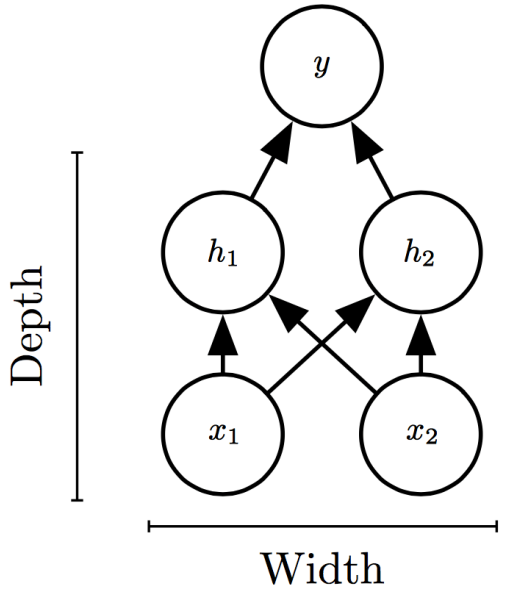


3Blue1Brown Explaining how NN works:
https://www.youtube.com/watch?v=aircAruvnKk

(ไม่ออกสอบ) Universal Approximation Theorem

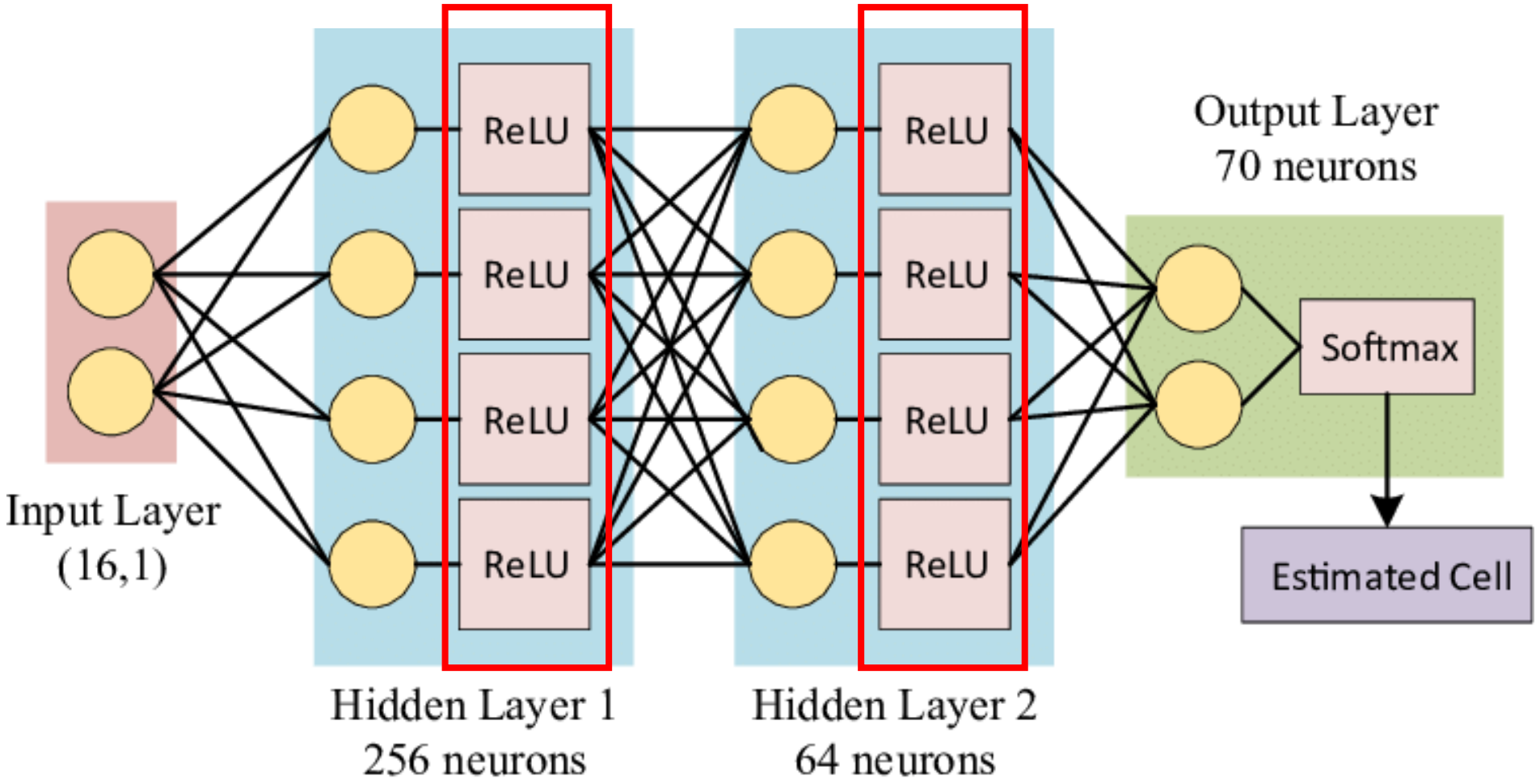
- Think of Neural Network as function approximation.
  - $Y = f(x) + \epsilon$
  - $Y = \hat{f}(x) + \epsilon$
  - NN: $\Longrightarrow \hat{f}(x)$

- One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more



https://harvard-iacs.github.io/2019-CS109B/category/lectures.html

(Nonlinear) Activation Function



A Belmonte-Hernández et al., SWiBluX: Multi-Sensor Deep Learning Fingerprint for Precise Real-Time Indoor Tracking

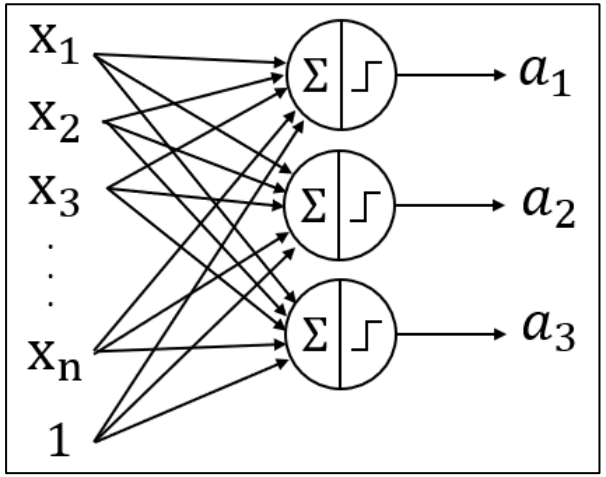
Viewing a layer as matrix multiplication

- ในการที่จะเข้าใจ activation function ได้นั้น เราต้องเข้าใจก่อนว่า:
- การคำนวณของ layer มองเป็นการนำ weight matrix W มาคูณกับ input vector $\bar{x}$ ได้ ดังนี้

$$\bar{a} = \sigma(\bar{z}) = \sigma\left(W\bar{x} + \bar{b}\right)$$

- เช่น layer 3 neurons ในภาพทำการคำนวณดังนี้:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \sigma\left(\begin{bmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & W_{2n} \\ W_{31} & W_{32} & \dots & W_{3n} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$



layer of 3 neurons

Problem

- พิจารณา NN ที่มี 2 layers โดยที่
  - W, b_1 คือ weight matrix, bias ของ layer 1
  - V, b_2 คือ weight matrix, bias ของ layer 2
- ถ้าไม่มี nonlinearity $\sigma(\cdot)$ คั่นระหว่าง layer จะได้ว่า

$$\begin{aligned} a &= V(W\mathrm{x} + \mathrm{b}_1) + \mathrm{b}_2 \\ &= VW\mathrm{x} + V\mathrm{b}_1 + \mathrm{b}_2 \\ &= (VW)\mathrm{x} + (V\mathrm{b}_1 + \mathrm{b}_2) \\ &= UX + b \end{aligned}$$

- ผลคือ: ทุก layer ยุบรวมเหลือ layer เดียว
  - ทำให้ NN กลายเป็นแค่ Linear Regression model ธรรมดา ☹

Solution: Nonlinearity

- ทางออกคือ เพิ่ม activation function g ที่เป็น non-linear ให้กับทุก layer
  - $a = g(Vg(W\mathrm{x} + b_1) + b_2)$
  - ไม่สามารถยุบ layer ได้ ☺
  - ช่วยให้ NN สามารถตรวจจับความสัมพันธ์แบบ non-linear ได้

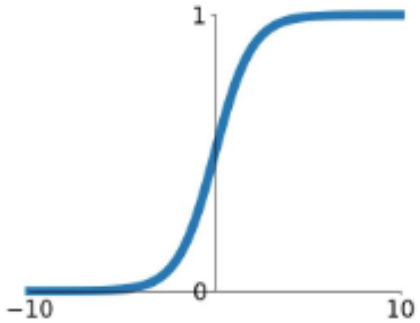
- activation function ที่ดีควรเป็นอย่างไร ?
  - สามารถหาอนุพันธ์ได้ (differentiable)
  - คำนวณได้เร็ว
  - ได้ค่า gradient ที่มากพอควรในทุกๆ layer (กันปัญหา vanishing gradient)

- Common choices: ReLU, Sigmoid, tanh, swish, leaky ReLU, MaxOut

ตัวอย่าง Activation Functions

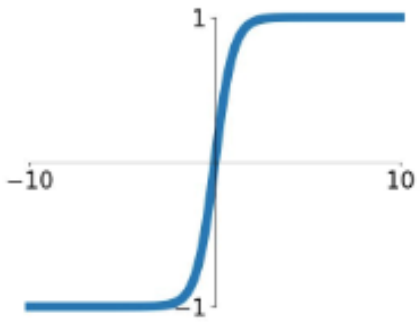
Sigmoid

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



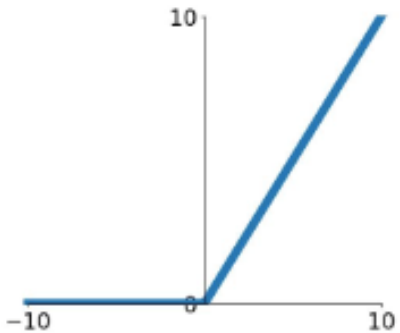
tanh

$$\tanh(x)$$



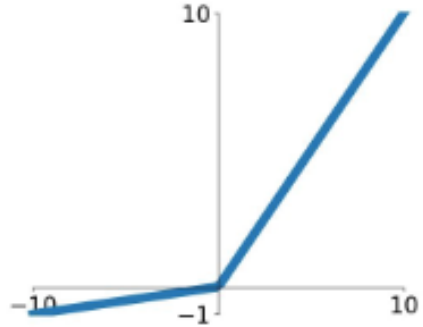
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

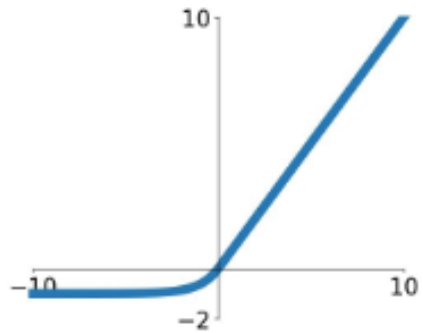


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

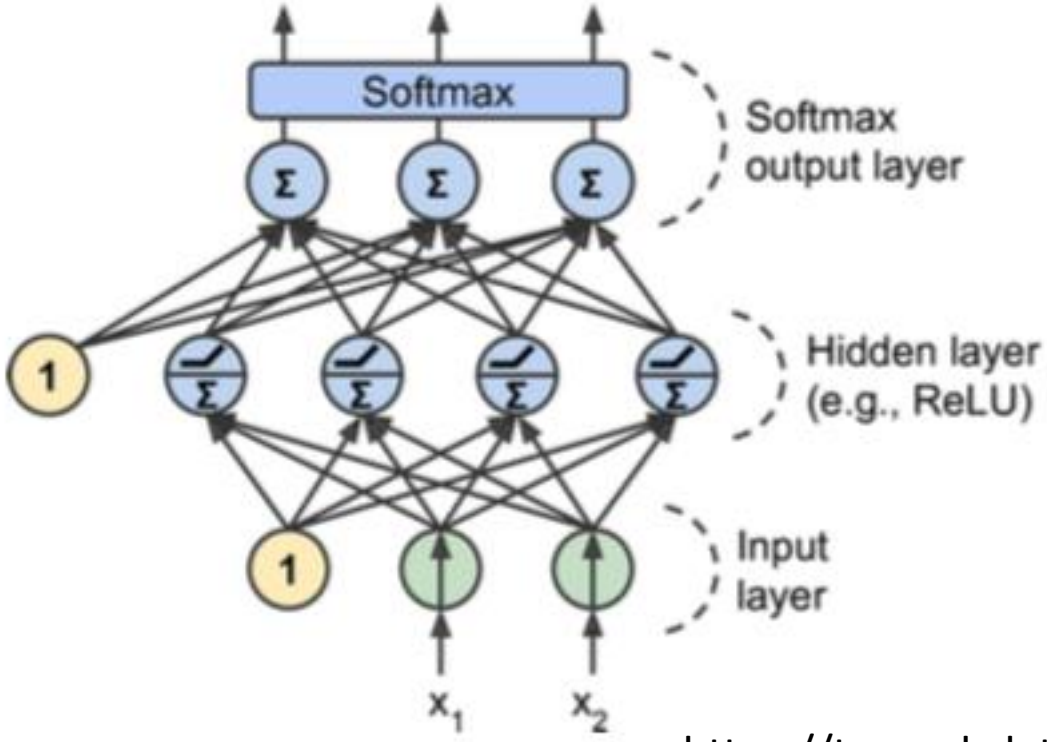
ELU

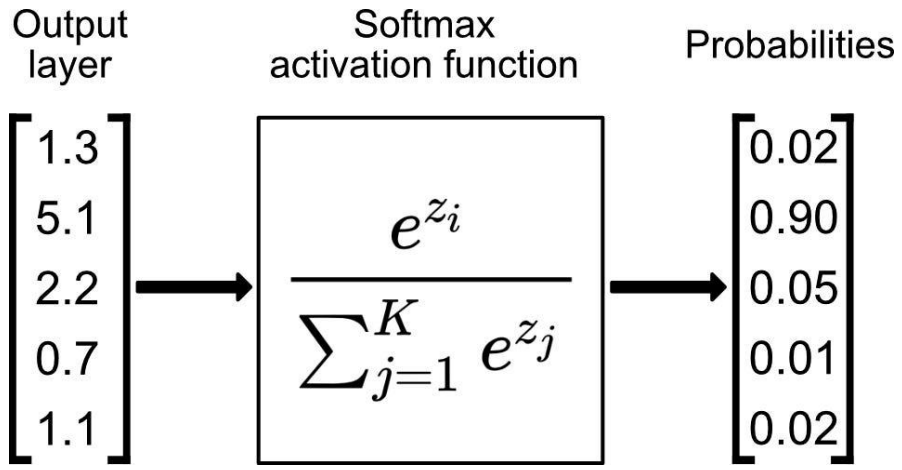
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Softmax Layer

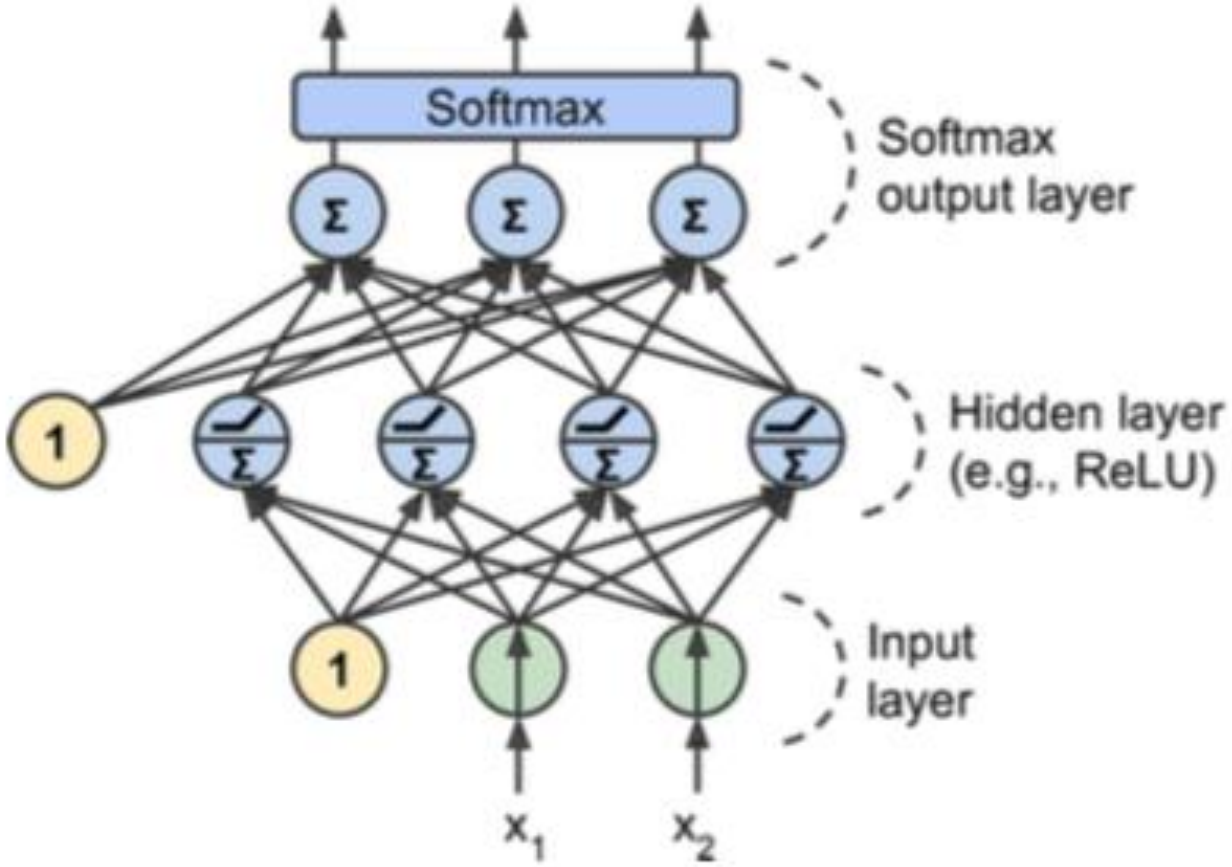
- สำหรับงาน classification เพื่อให้ output ของ network คำนวณค่า**ความน่าจะเป็น**ในช่วง 0-1 ของแต่ละคลาส จึงนิยมให้ layer สุดท้ายเป็นแบบ Softmax (generalized/multiclass sigmoid)





https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

Feedforward Neural Network

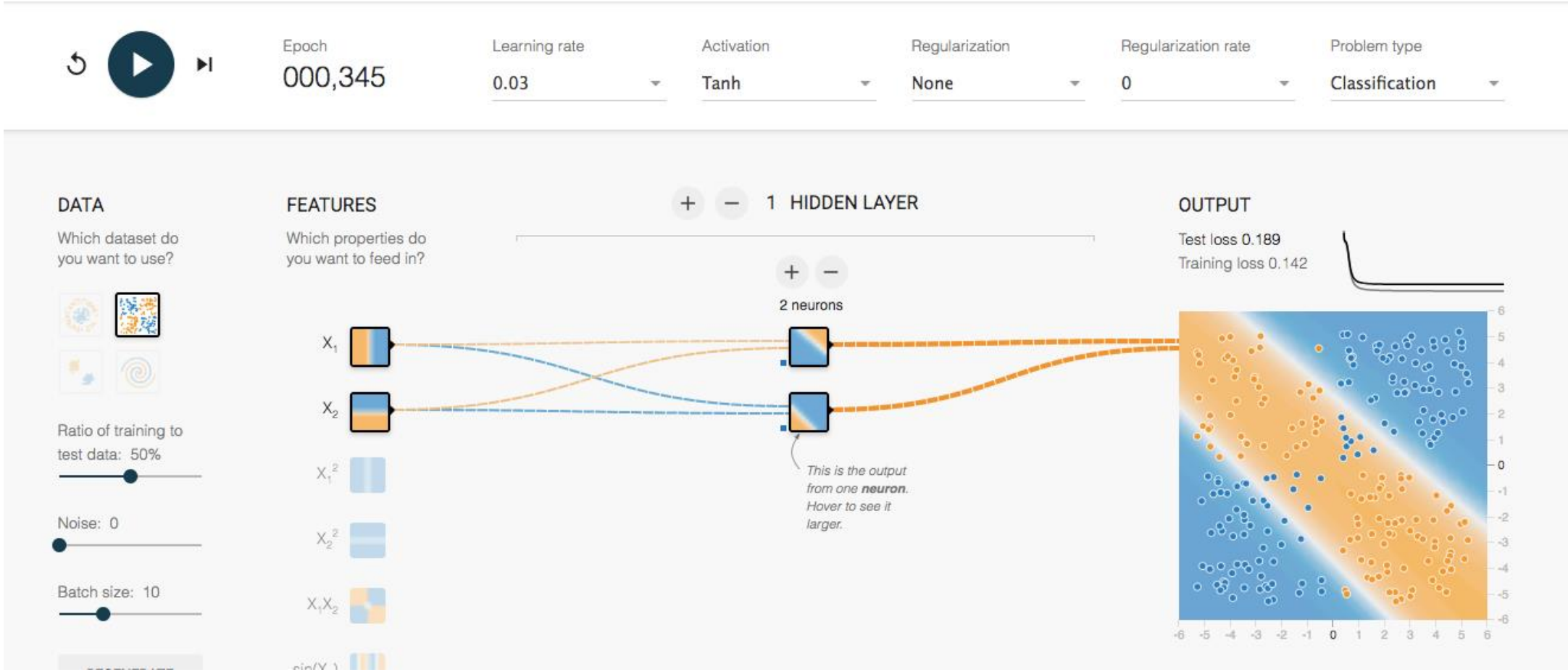


เราเรียก architecture ของ ANN ที่ไหลไปในทางเดียว จาก input ไปยัง output ดังภาพนี้ว่า

"Feedforward Neural Network"

Visualizing NN with Tensorflow Playground

- https://playground.tensorflow.org



Advanced topics on NN

- (ไม่ออกสอบ)
  - Vanishing Gradient Problem
  - Weight Initialization
  - Learning Rate Scheduling
  - Dealing with overfitting: Dropout, Batch Norm

- (Next week) Specialized NN: CNN, RNN

Lab: MNIST with Neural Network

Lab: MNIST with Neural Network

- Popular open-source deep learning libraries

Table 9-1. Open source Deep Learning libraries (not an exhaustive list)

Library	API	Platforms	Started by	Year
Caffe	Python, C++, Matlab	Linux, macOS, Windows	Y. Jia, UC Berkeley (BVLC)	2013
Deeplearning4j	Java, Scala, Clojure	Linux, macOS, Windows, Android	A. Gibson, J. Patterson	2014
H2O	Python, R	Linux, macOS, Windows	H2O.ai	2014
MXNet	Python, C++, others	Linux, macOS, Windows, iOS, Android	DMLC	2015
TensorFlow	Python, C++	Linux, macOS, Windows, iOS, Android	Google	2015
Theano	Python	Linux, macOS, iOS	University of Montreal	2010
Torch	C++, Lua	Linux, macOS, iOS, Android	R. Collobert, K. Kavukcuoglu, C. Farabet	2002

Lab: MNIST with Neural Network

- **Keras** เป็น **high-level API** ที่สามารถนำไปรันบน **tensorflow, pytorch** ได้
- ใช้งานง่าย เหมาะกับการเรียนรู้ครั้งแรก
- https://keras.io/

Useful Links to Keras Docs

- Sequential Model: https://keras.io/guides/sequential_model/
- Dense Layer: https://keras.io/api/layers/core_layers/dense/
- compile & fit: https://keras.io/api/models/model_training_apis/
- (*) Example codes: https://keras.io/examples/

MNIST with Keras – Import dataset

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()

x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

MNIST with Keras – Create NN model

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

MNIST with Keras – Fit/Evaluate model

```
model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy", metrics=['accuracy'])
model.fit(x_train, y_train, epoch=3)
val_loss, val_acc = model.evaluate(x_test, y_test)
```

```
model.predict([x_test])
```

Experimenting

- ทดลองใช้ NN ที่มีเพียง input/output layer โดยไม่มี hidden layer แล้วสังเกตว่า test accuracy เปลี่ยนไปอย่างไร
- ทดลองใช้ hidden layer ที่ไม่มีการทำ activation (นำโค้ดส่วน activation=... ออก) แล้วสังเกตค่า test accuracy
- ทดลองเปลี่ยนค่า epochs, optimizer, จำนวน neuron ใน layer, จำนวน hidden layer
- ทำความเข้าใจข้อมูลที่ model.summary() แสดง
  - จำนวน parameter ของโมเดลคือเท่าใด
  - เลข 784, 10 ใน input_size คืออะไร
  - เลข 32 ใน input_size คืออะไร

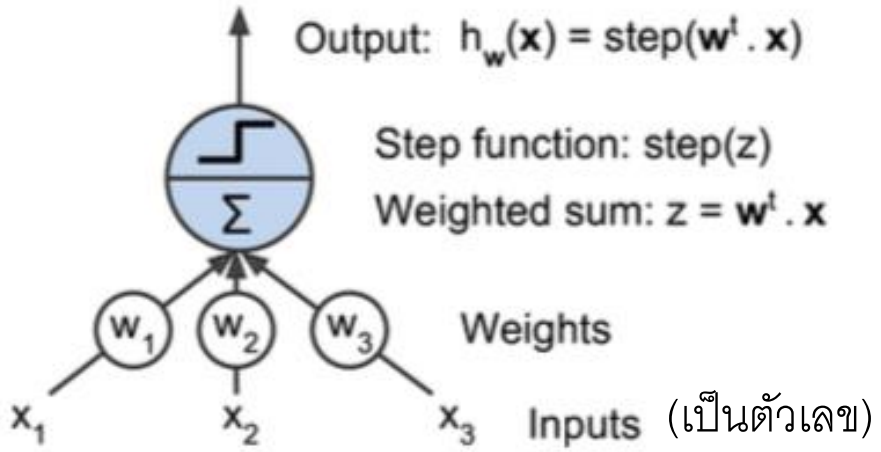
คำศัพท์ที่ควรรู้เกี่ยวกับการ implement NN

- batch size
  - จำนวนข้อมูลที่ส่งเข้าไปในแต่ละ iteration (default: 32) สำหรับทำ mini-batch SGD
- iteration
  - การส่งข้อมูล 1 batch เข้าไปใน NN เพื่อทำการ train ปรับค่า param (backprop)
- epoch
  - การส่งข้อมูล training set เข้าไปใน NN เพื่อทำการ train ครบทั้ง training set 1 รอบ
- **ตัวอย่าง:**
  - จำนวน training data: 50,000 ภาพ
  - batch_size: 32 (default) ➔ แสดงว่าในการ train ส่งข้อมูล iteration ละ 32 ภาพ
  - ดังนั้น ใน 1 epoch จึงต้องทำทั้งหมด 50,000/32 = 1563 iterations
    - จึงจะส่งข้อมูลให้ NN ครบทั้ง 50,000 ภาพ
  - ปกติเราอาจต้อง train หลาย epoch จึงจะได้ loss ที่พอใจ

(ไม่ออกสอบ) Historical ANN models

Perceptron

- เป็น ANN ที่เรียบง่าย คิดค้นโดย Rosenblatt ปี 1957
- ประกอบด้วย Neuron ที่เป็น Linear Threshold Unit (LTU) ดังรูป

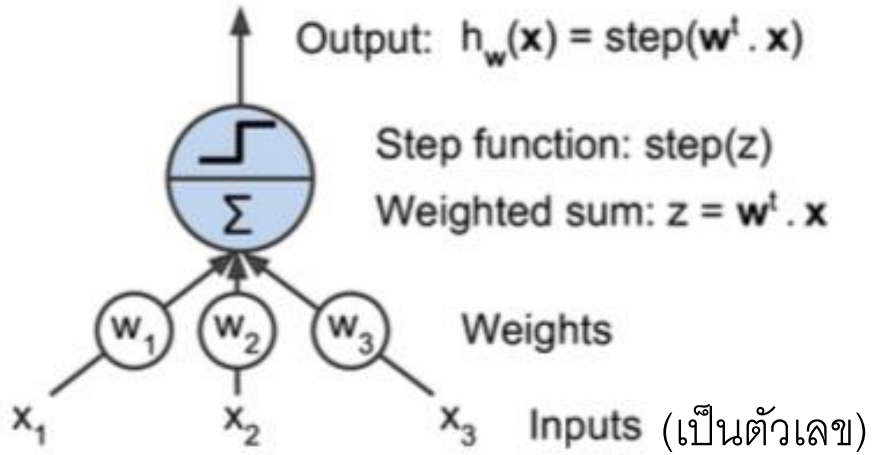


$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

- สำหรับ step function อาจใช้เป็นฟังก์ชัน heaviside หรือ sign

Perceptron

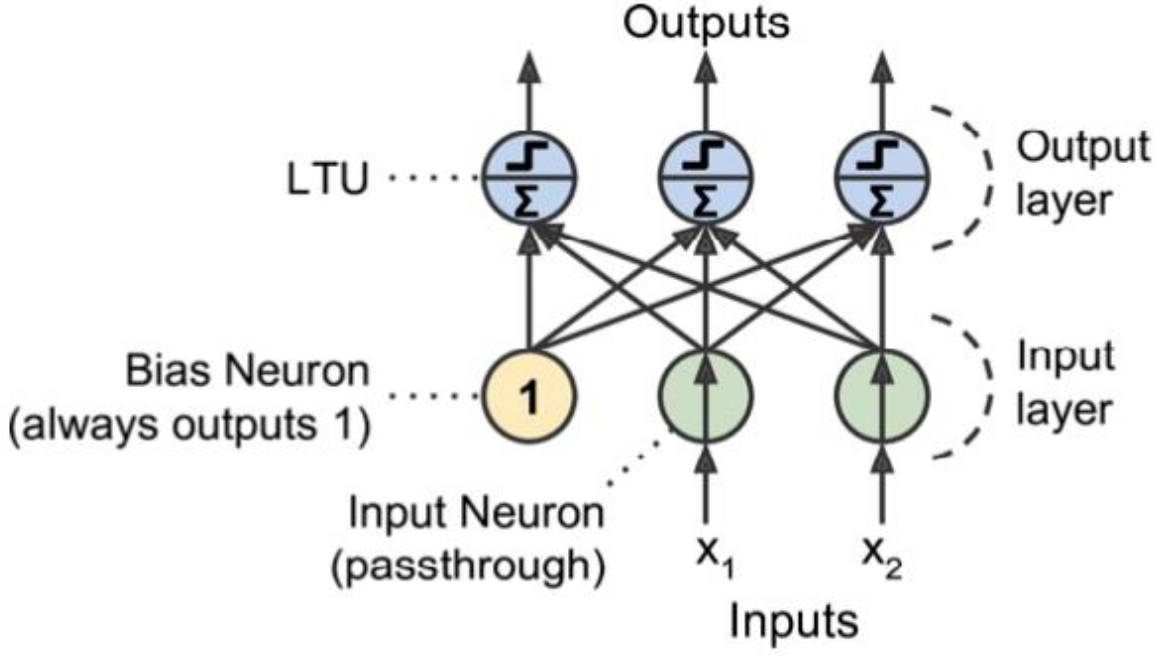
- Single LTU สามารถใช้ทำ binary classification (นึกถึง Iris) ได้
- สังเกตว่าโมเดลของ LTU คือการหา linear combination ของ input แล้วนำมา threshold เพื่อให้ output ทำนายออกมาว่าเป็นคลาส + positive หรือ – negative
- คล้าย Logistic Regression



$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Perceptron

- Perceptron ประกอบด้วย layer ของ LTU เพียง 1 layer
- มี Bias input node ที่มีค่าเป็น 1 เสมอ
  (เพื่อให้ decision boundary ไม่จำเป็นต้องผ่านจุด origin)
- ในภาพเป็น Perceptron ที่มี 3 LTU สามารถใช้ classify ได้ 3 คลาส



Perceptron - Training

- การ train Perceptron คือการหาค่า weight ที่ทำให้โมเดลสามารถทำนายได้แม่นที่สุด
- Rosenblatt ใช้ Hebb's rule: "Cells that fire together, wire together."
- กล่าวคือ ต้องการให้ weight มีค่าสูง หาก neuron คู่นั้น output ค่าเดียวกันบ่อย ๆ
- จึงเกิดเป็น Perceptron learning rule ดังนี้

Equation 10-2. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

- สังเกตว่า weight จะถูก update ก็ต่อเมื่อ $\hat{y} \neq y$ (ทำนายพลาด) เท่านั้น

Perceptron Code

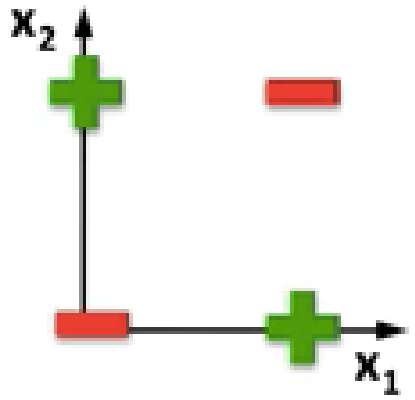
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype(np.int) # Iris Setosa?
per_clf = Perceptron(random_state=42)
per_clf.fit(X, y)
y_pred = per_clf.predict([[2, 0.5]])
```

คล้าย Logistic Regression แต่ต่างกันที่ output ของ Perceptron ไม่ใช่ class probability

Perceptron - Limitations

- ข้อจำกัดของ Perceptron คือไม่สามารถทำ non-linear classification ได้
  - training data ต้องสามารถถูกแบ่งแดนด้วย hyperplane ตรงได้เท่านั้น (linearly separable)
  - เช่น ไม่สามารถแก้ XOR problem ได้

Linear classifiers
cannot solve this

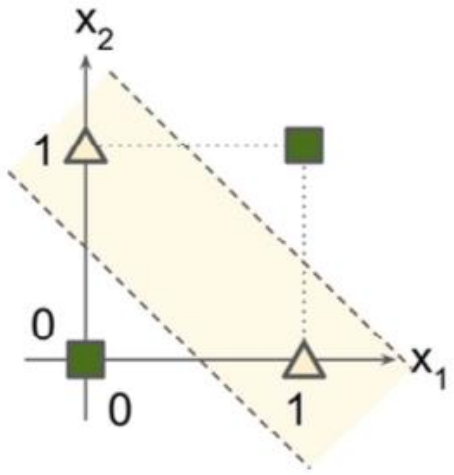


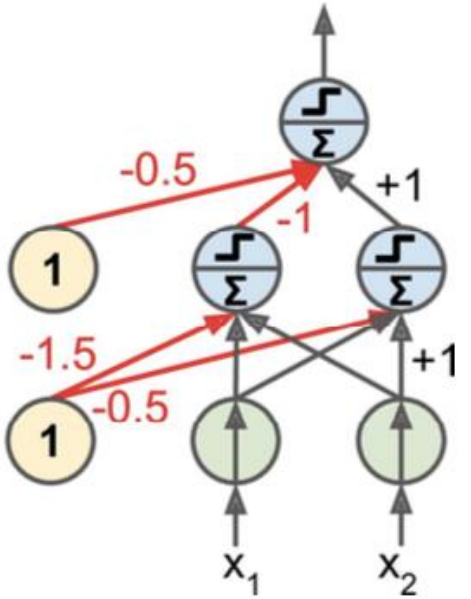
?

Perceptron - Limitations

- ข้อจำกัดของ Perceptron คือไม่สามารถทำ non-linear classification ได้
  - training data ต้องสามารถถูกแบ่งแดนด้วย hyperplane ตรงได้เท่านั้น (linearly separable)
  - เช่น ไม่สามารถแก้ XOR problem ได้

solution: เพิ่มจำนวน layer (Multilayer Perceptron)





Multilayer Perceptron (MLP)

- เราเรียก ANN มี 2 hidden layer ขึ้นไปว่า Deep Neural Network
- ในยุคนั้น นักวิจัยตันกับการหาวิธี train MLP อยู่หลายปี
- จนกระทั่งปี 1986 Rumelhart ได้ตีพิมพ์อัลกอริทึม ที่ปฏิวัติวงการ ชื่อ **Backpropagation**