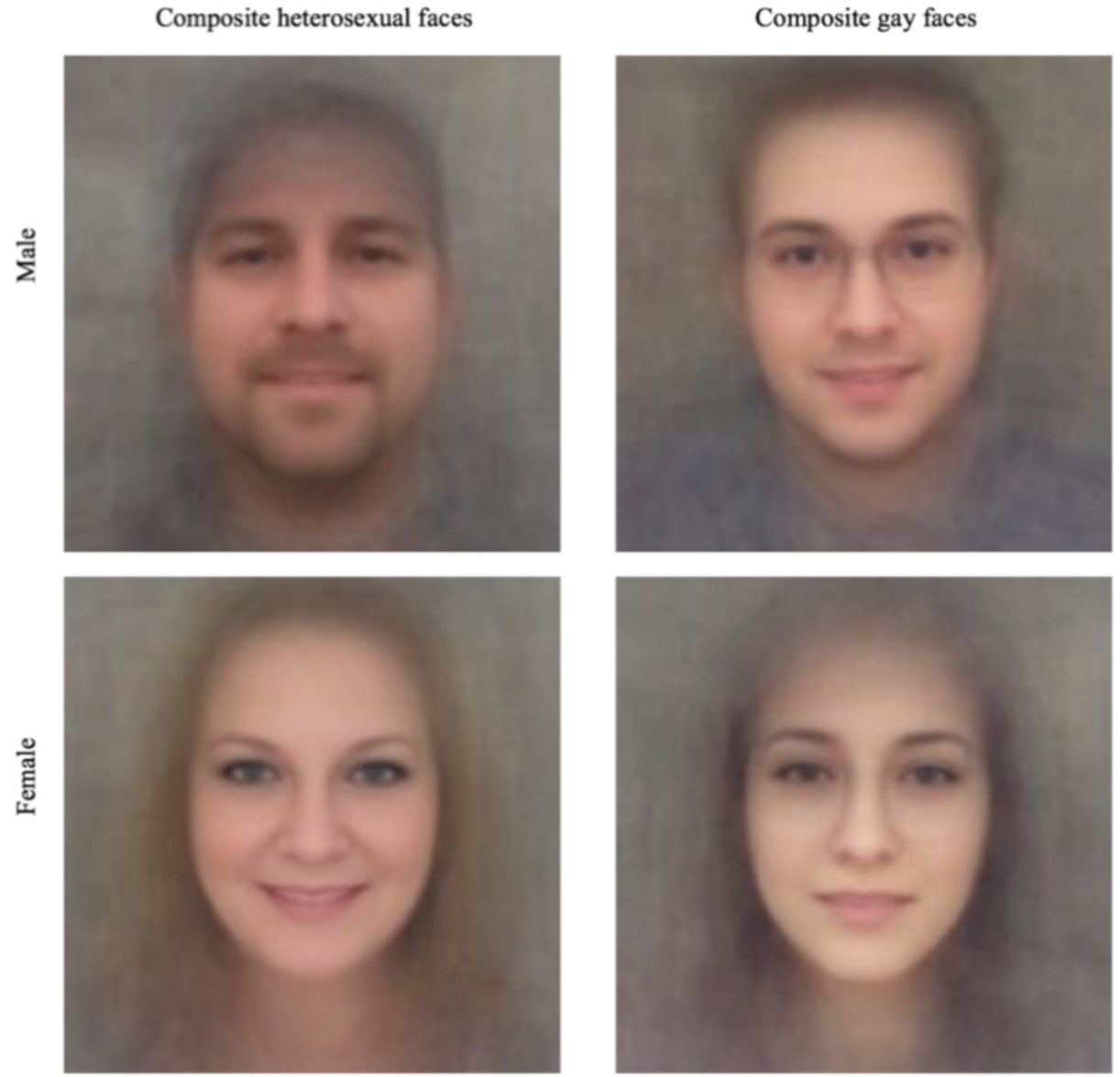
心理與神經資訊學
(Psychoinformatics & Neuroinformatics)

課號：Psy5261　　教室： 博雅 101

識別碼：227U9340　　時間： 四 234

Deep Learning Nets 的應用 (1/4)

利用 VGG-Face 來判斷是否為同志 (classification)



Deep Learning Nets 的應用 (2/4)

判斷誰可能犯罪 (classification)

(a) Three samples in criminal ID photo set S_c.

(b) Three samples in non-criminal ID photo set S_n

Figure 1. Sample ID photos in our data set.

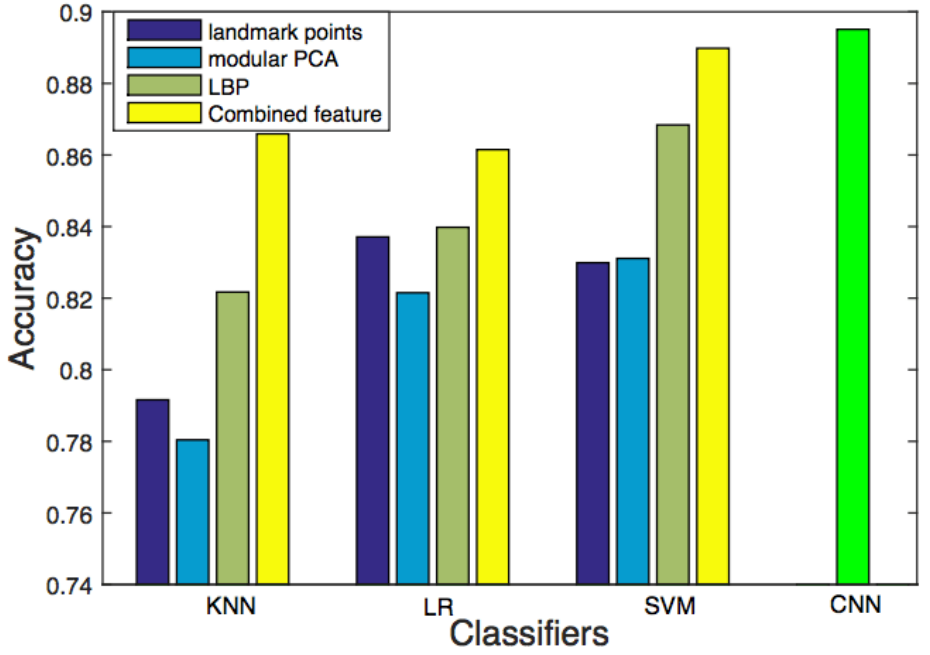


Figure 2. Accuracy of all four classifiers in all thirteen cases.

Deep Learning Nets 的應用 (3/4)

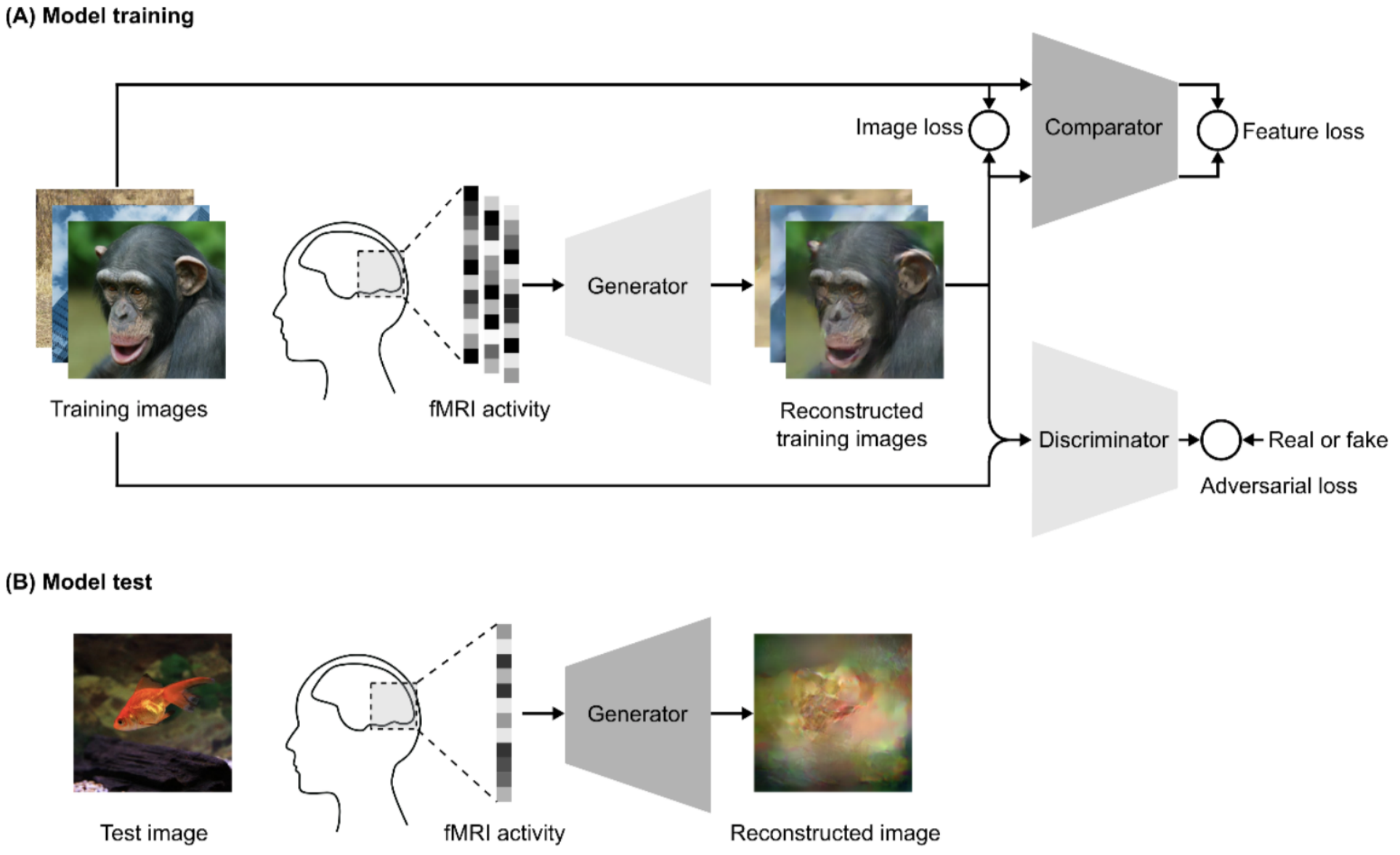
判斷誰比較美 (regression)

TABLE VI. CORRELATION COEFFICIENTS IN SINGLE NETWORK

Exp.	1	2	3	4	5	Average
PC	0.8509	0.8050	0.8112	0.7817	0.8446	0.8187

Deep Learning Nets 的應用 (4/4)

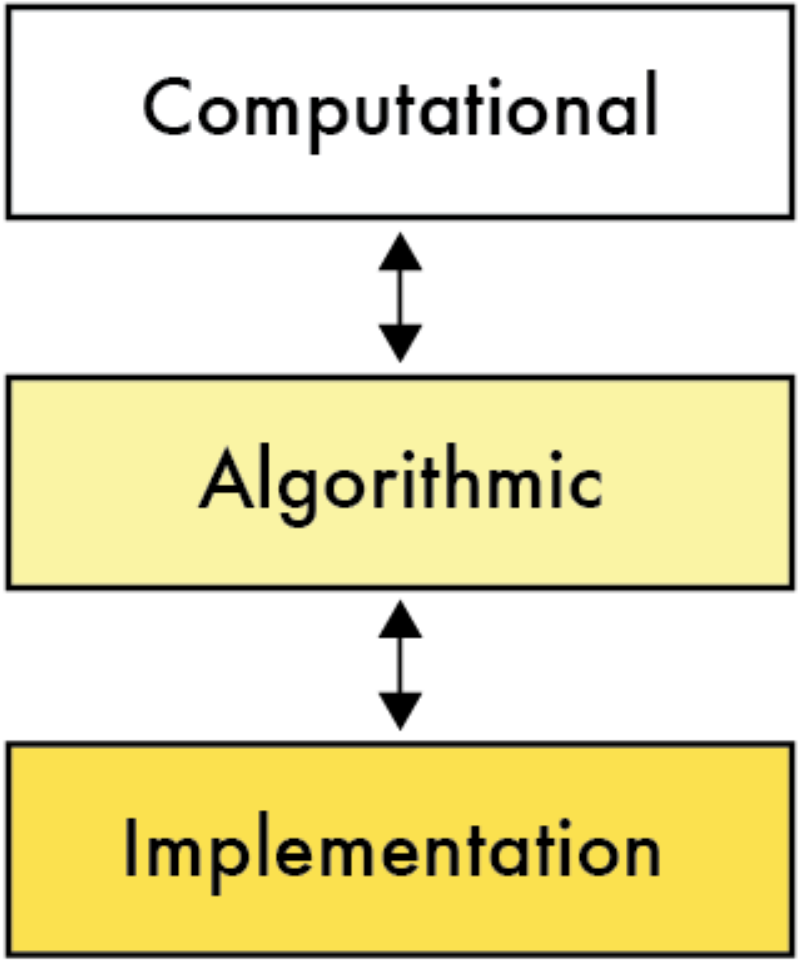
利用生成對抗網絡 (GAN) 來做 reconstruction



神經計算 & 類神經網路
(Neural Computation & Neural Networks)

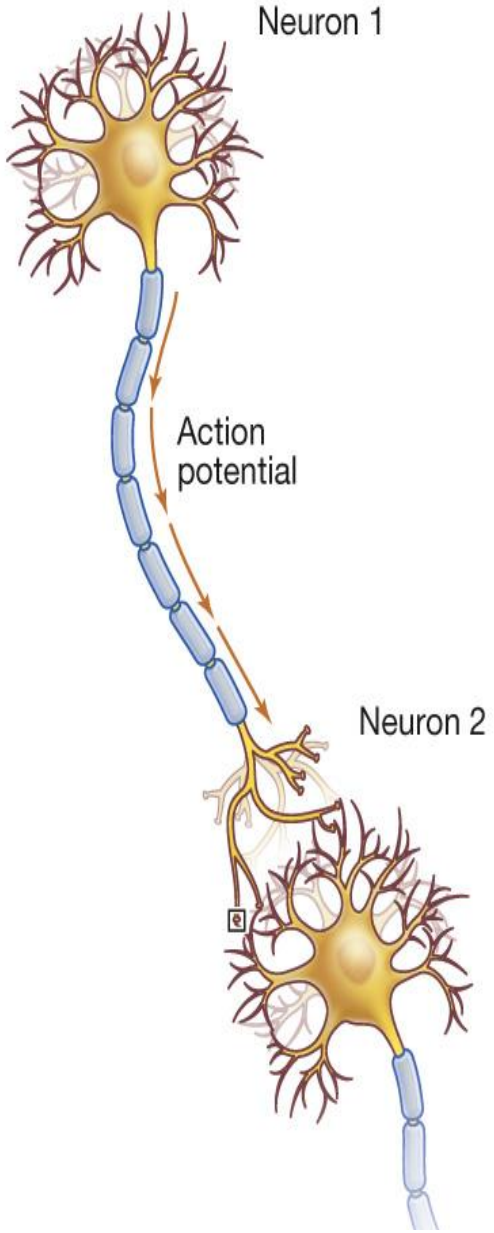
評估認知系統的特性與極限

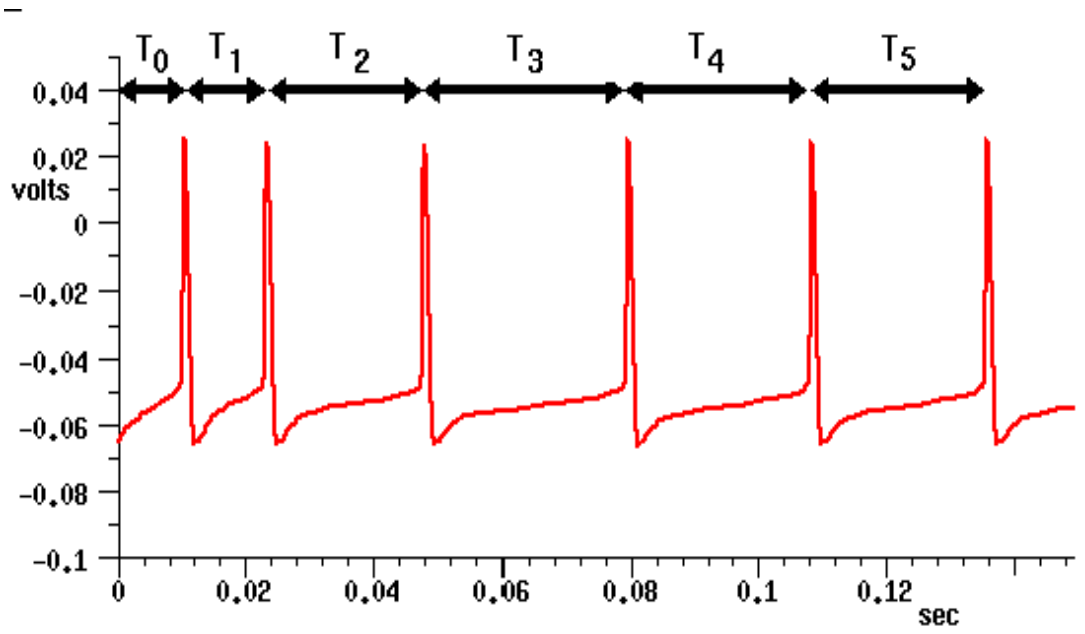
David Marr's 3 levels of analysis

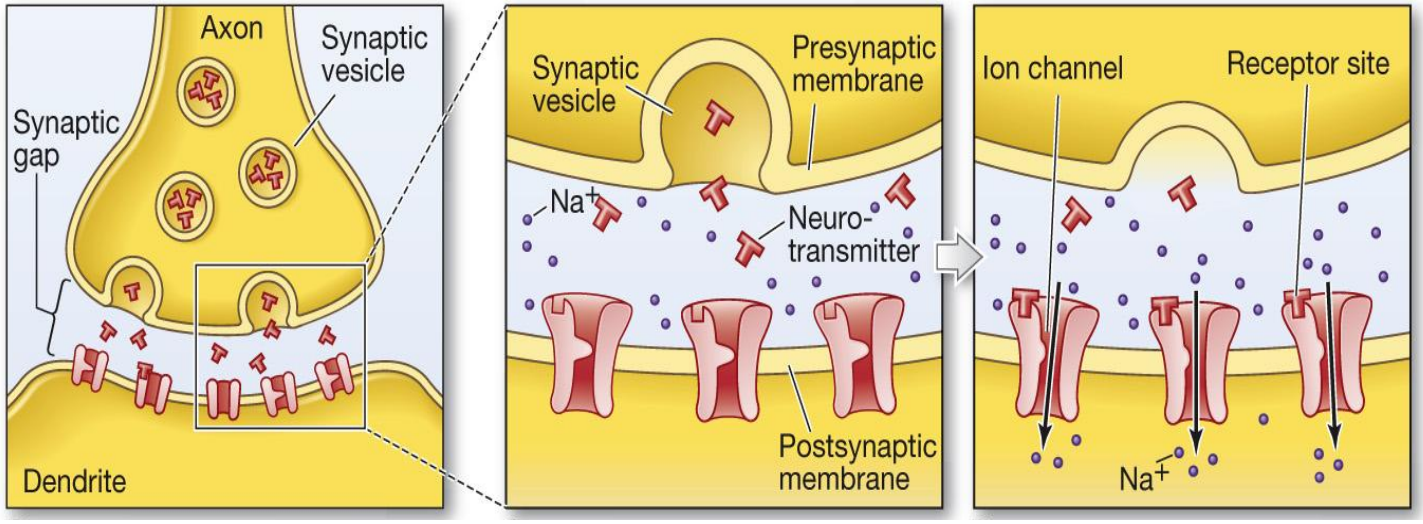


- 計算上的問題是什麼？
- 能夠用什麼演算法來解決？
- 要透過什麼硬體來實現算法？

Implementation: 有 0 與 1 的神經元

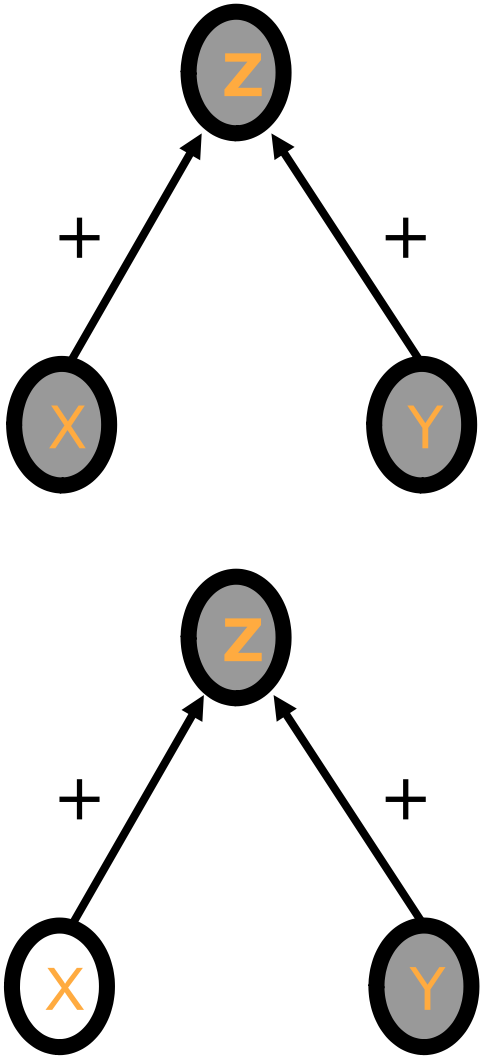






Implementation: 神經元做加法

如果接受刺激的神經元有個低閾值

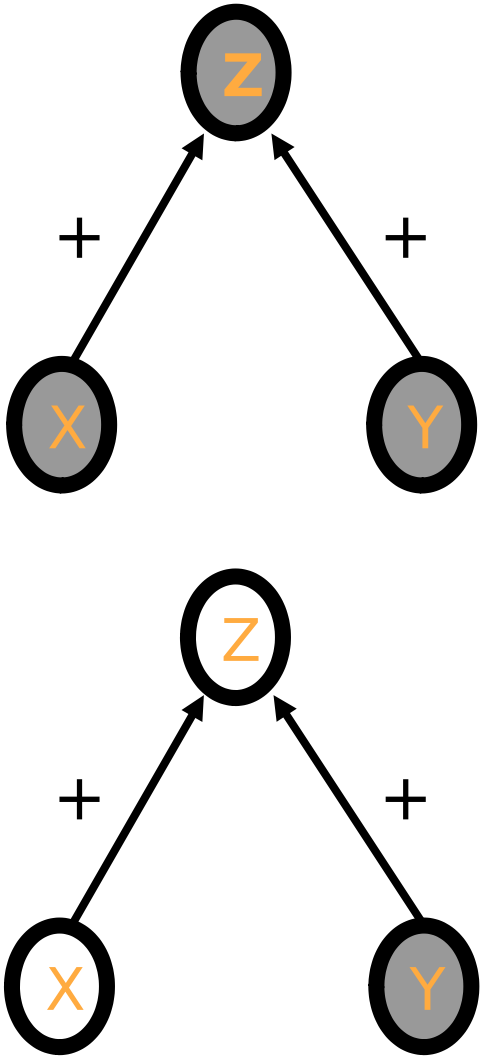


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Z=X+Y (OR)

Implementation: 神經元做乘法

如果接受刺激的神經元有個高閾值

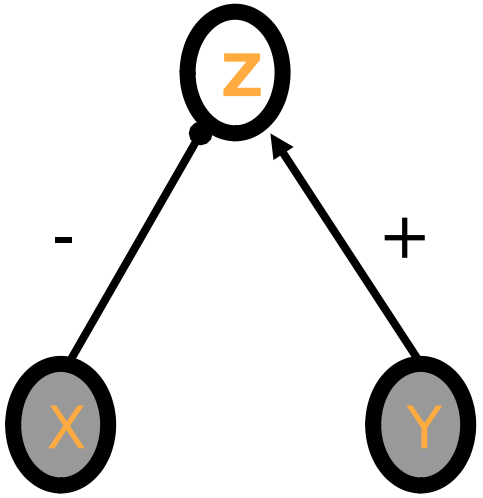


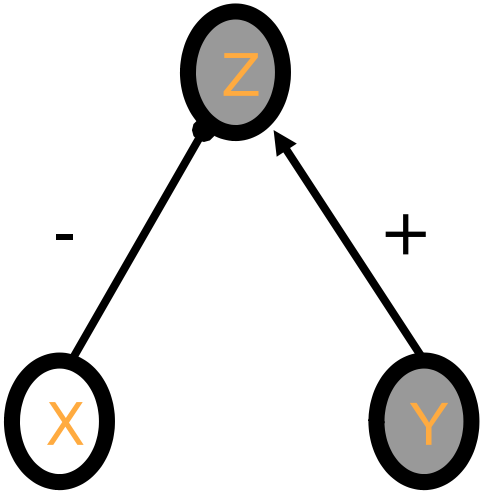
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Z=X*Y (AND)

Implementation: 神經元做減法

如果開始考慮神經元彼此的抑制關係



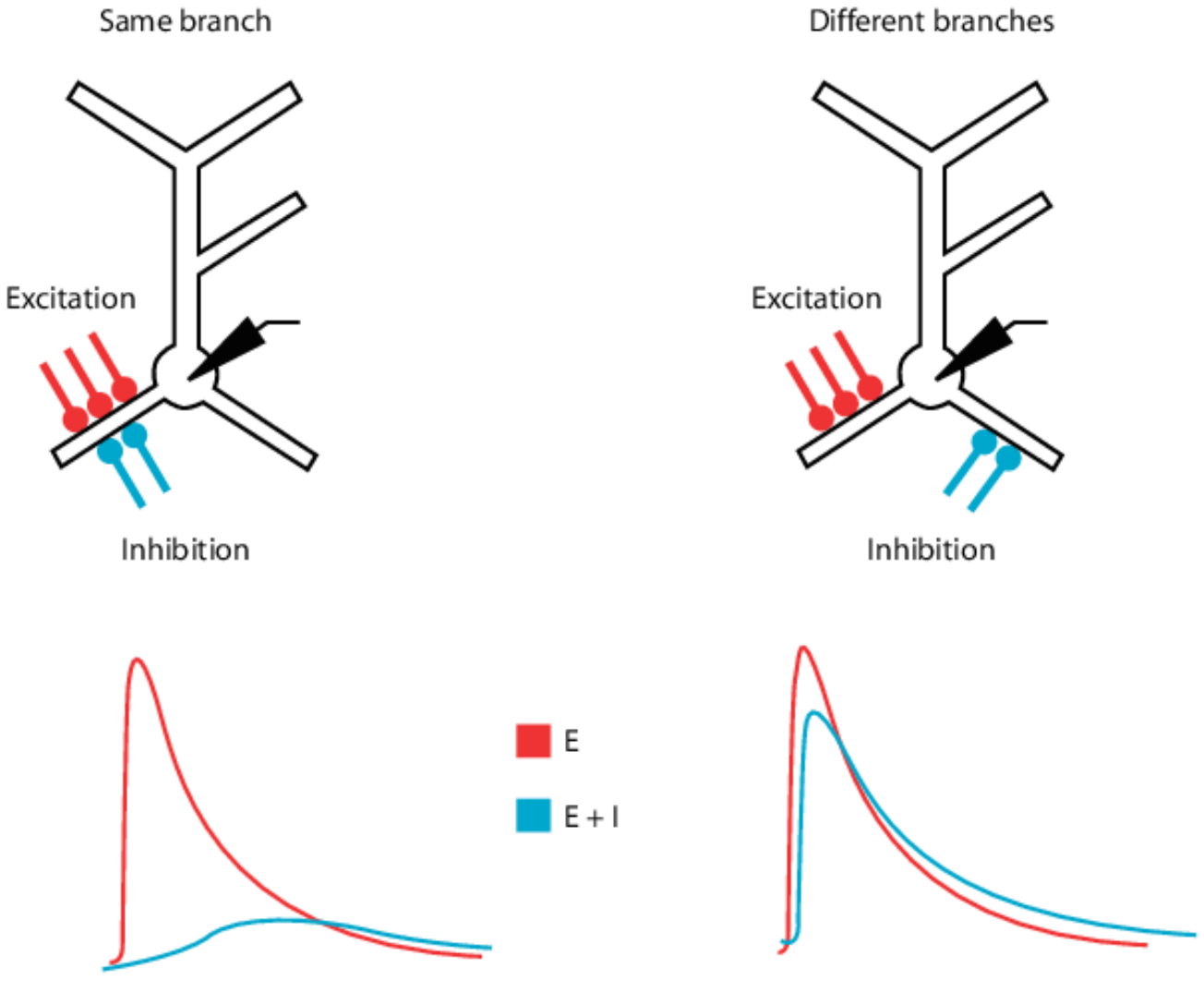


X	Z
0	1
1	0

Z=1-X (NOT)

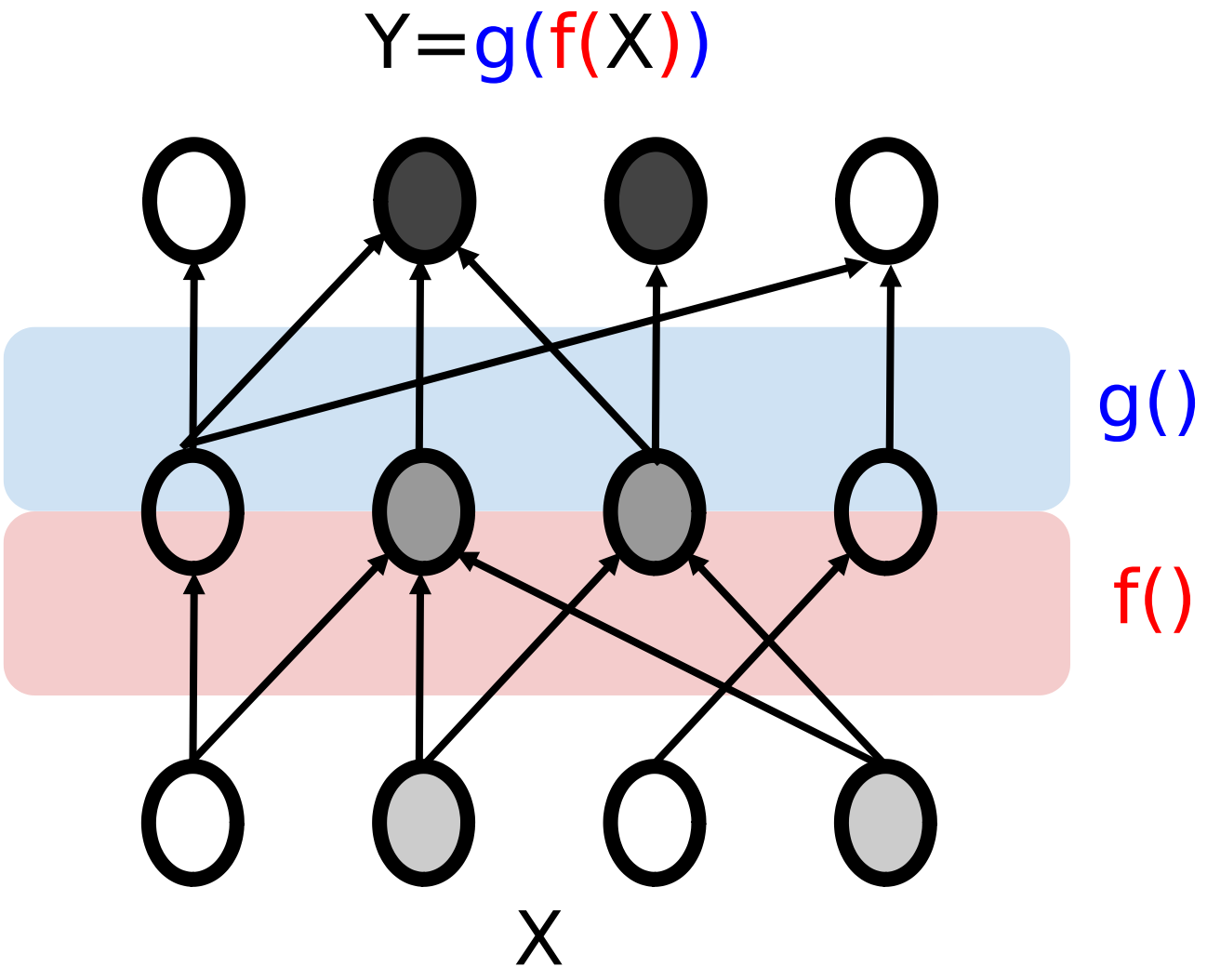
Implementation: 神經元做除法

神經元間的抑制可以導致減法或除法



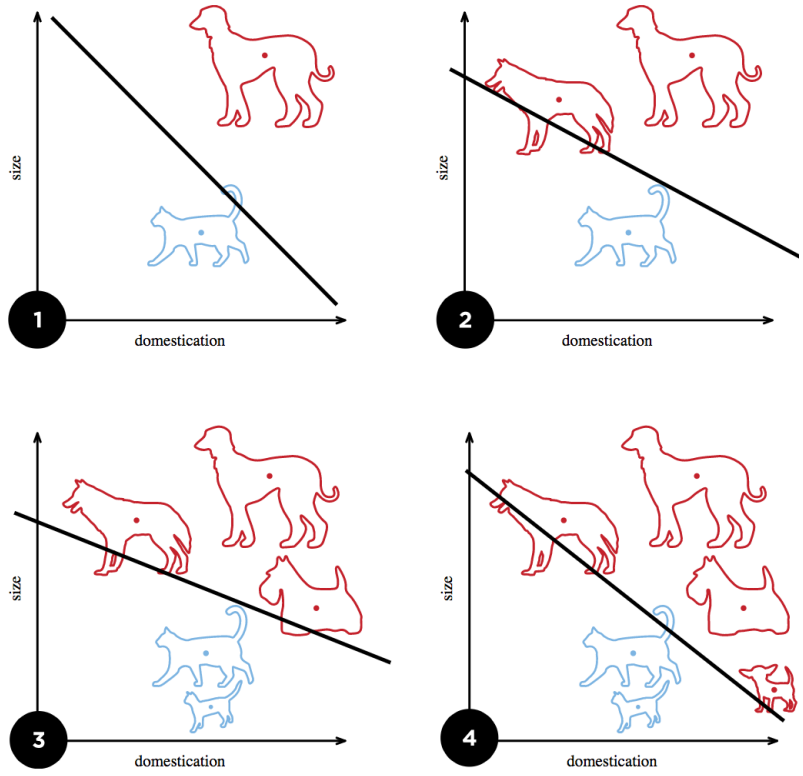
Algorithm: 基本計算的組合

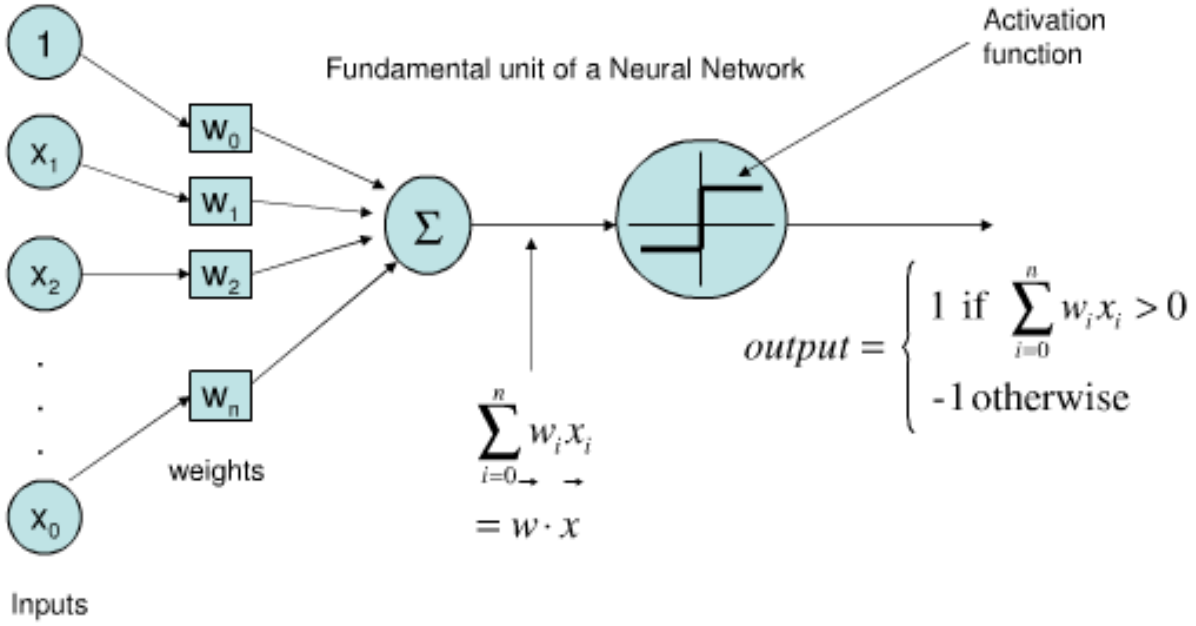
一個神經網路透過組合基本的計算來實現演算法



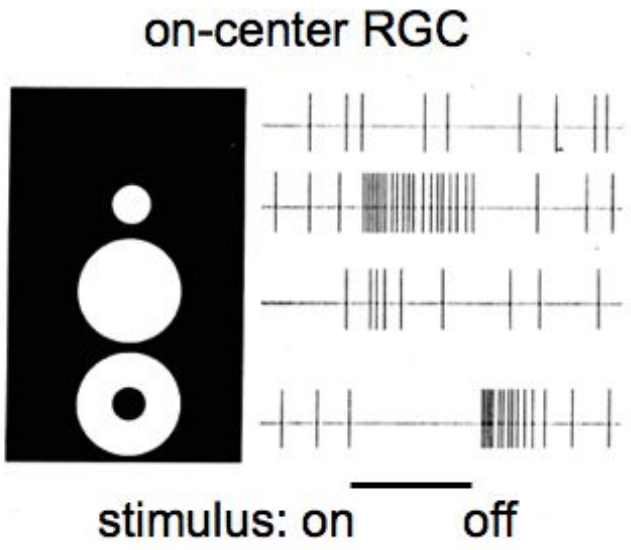
Computational Problem: 區分貓狗

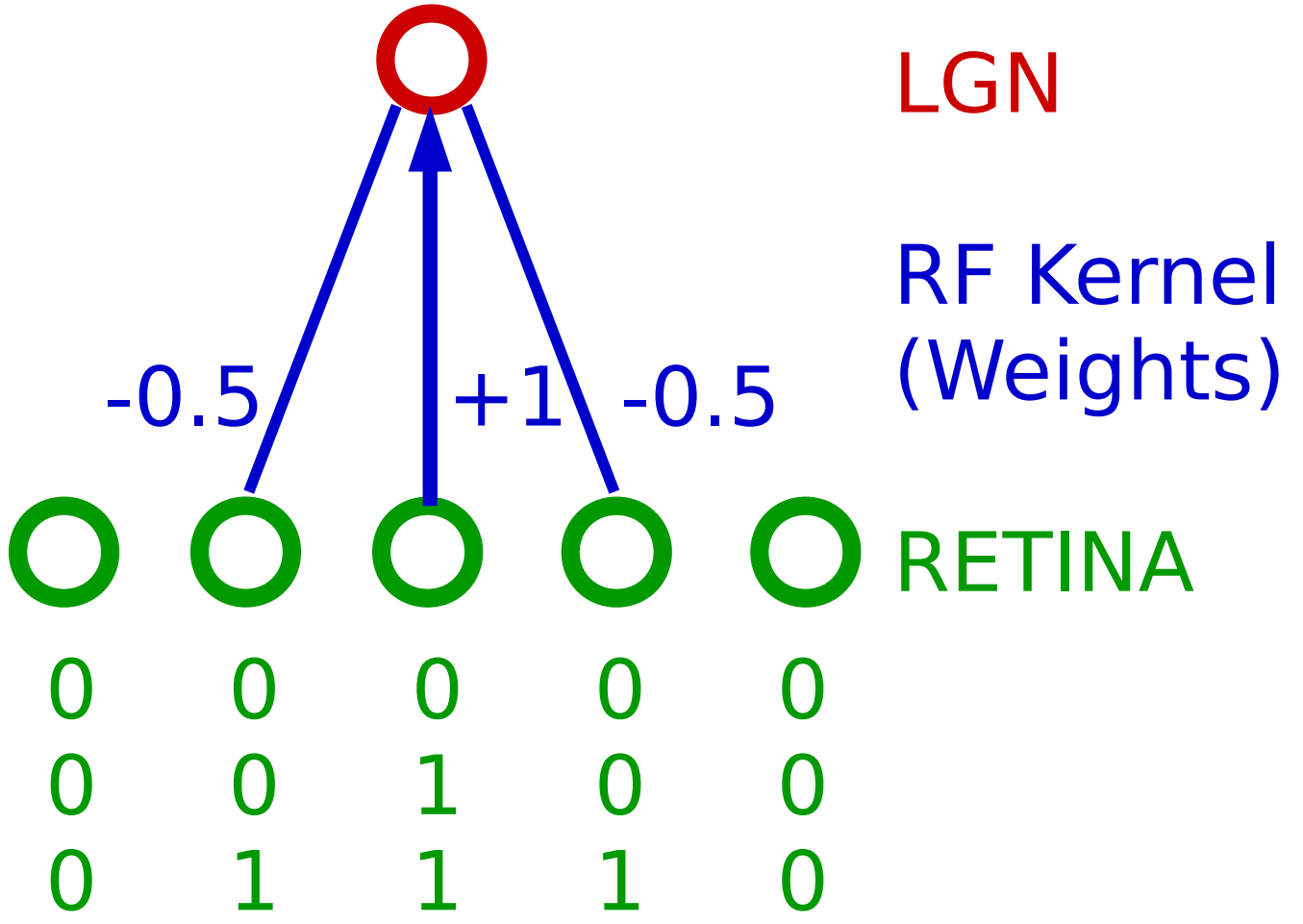
Perceptron (Rosenblatt, 1958)
Cognitron (Fukushima, 1975)





Neuron as a Pattern Recognizer






```
RETINA=matrix('0 0 0 0 0;0 0 1 0 0;0 1 1 1 0')
LGN_RF=matrix('0;-0.5;1;-0.5;0')
LGN_RESPONSE=RETINA*LGN_RF
```

類神經網路的學習
(Learning of Neural Networks)

Universal Approximation Theorem

3 層網路就可以逼近任何連續函數 (cf. 傅立葉分析)

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

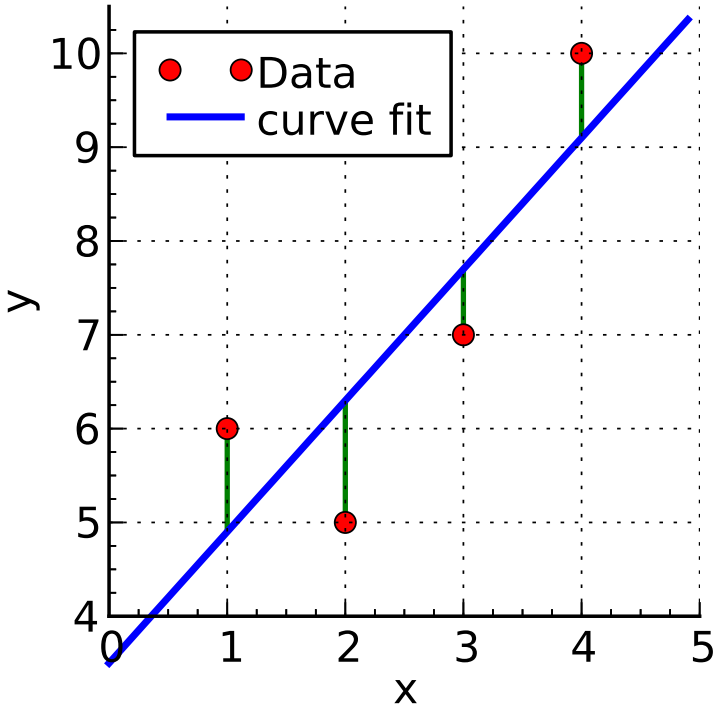
as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

Data Fitting

通常用最小平方法一次性地去最小化整體預測錯誤



$$y = \alpha + \beta x$$

$$y_i = \alpha + \beta x_i + \varepsilon_i$$

$$\hat{\varepsilon}_i = y_i - a - b x_i$$

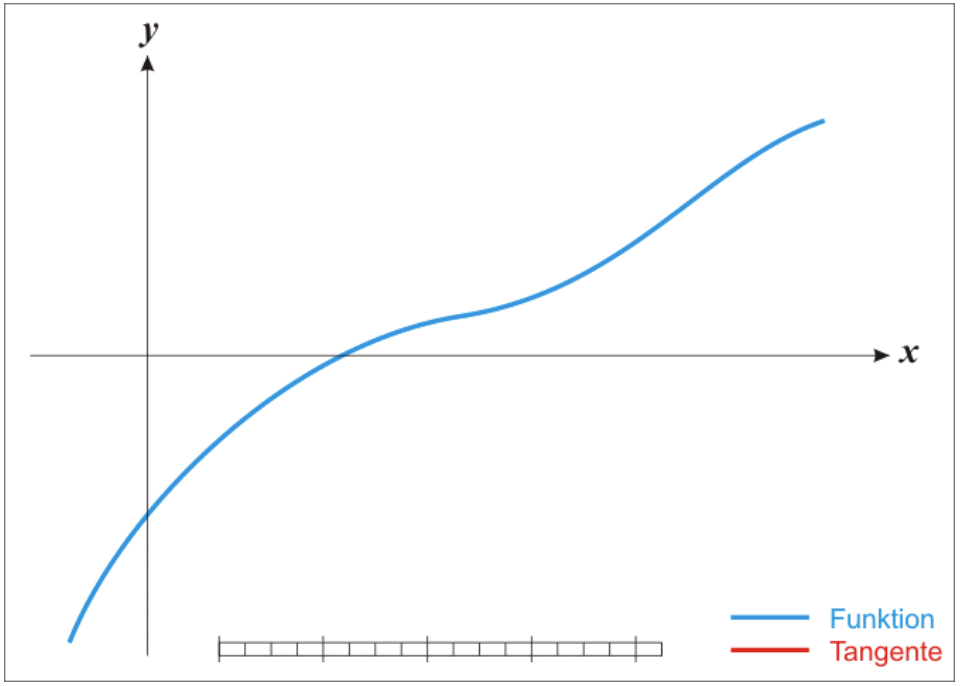
$$Q(a,b) = \sum_{i=1}^{n} \hat{\varepsilon}_i^{\,2} = \sum_{i=1}^{n} (y_i - a - b x_i)^2$$

$$\hat{\alpha} = \bar{y} - \hat{\beta}\,\bar{x},$$

$$\hat{\beta} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

Symbolic vs. Numerical Methods

數值法可以避免解 (多變量的聯立) 方程式



$$x : f(x) = 0$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

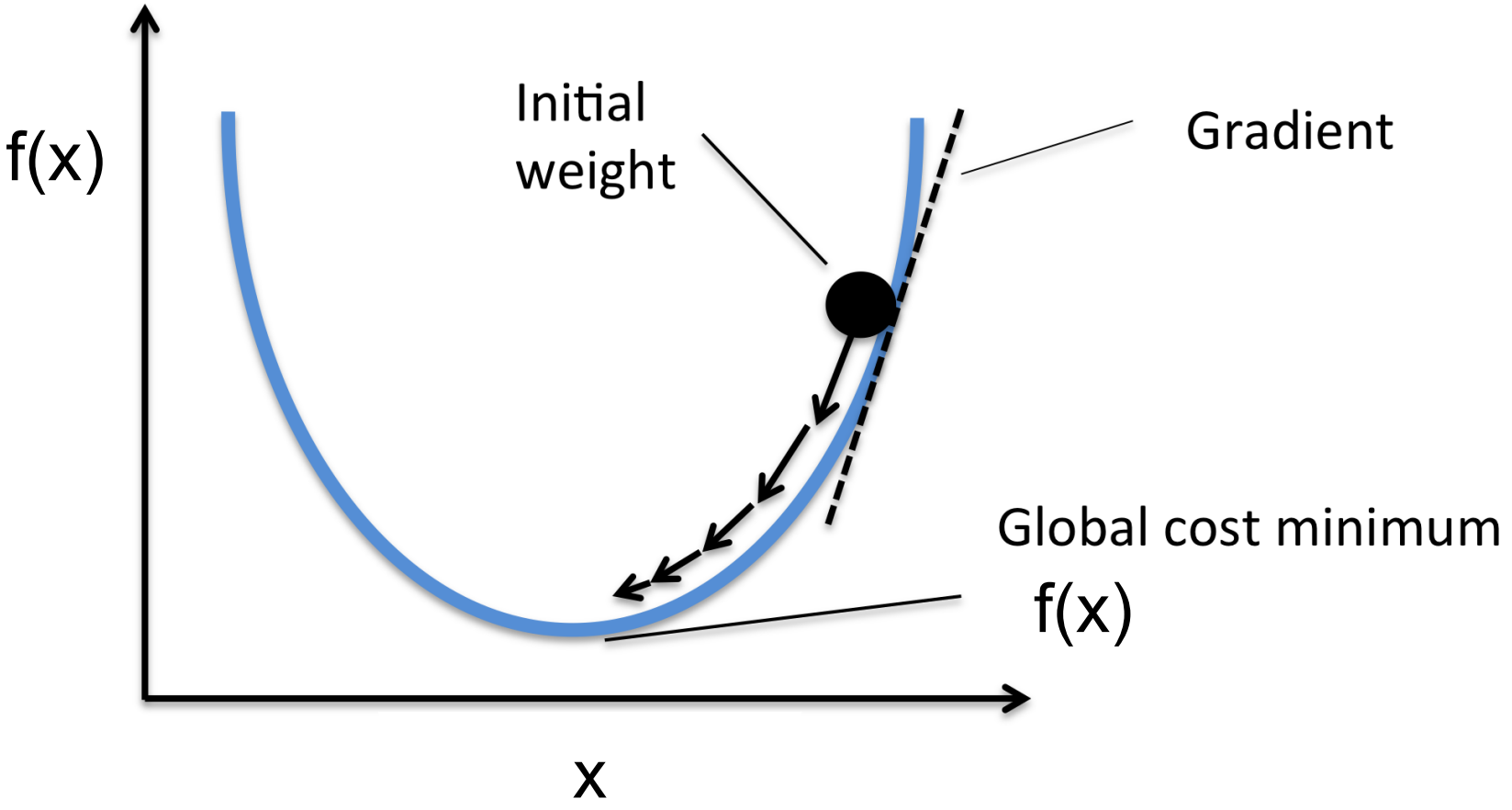
牛頓法透過多次調整 x_n 來解 $f(x_n)=0$

用 Gradient Descent 來最小化 Least Squares

局部最佳化：Gradient Descent

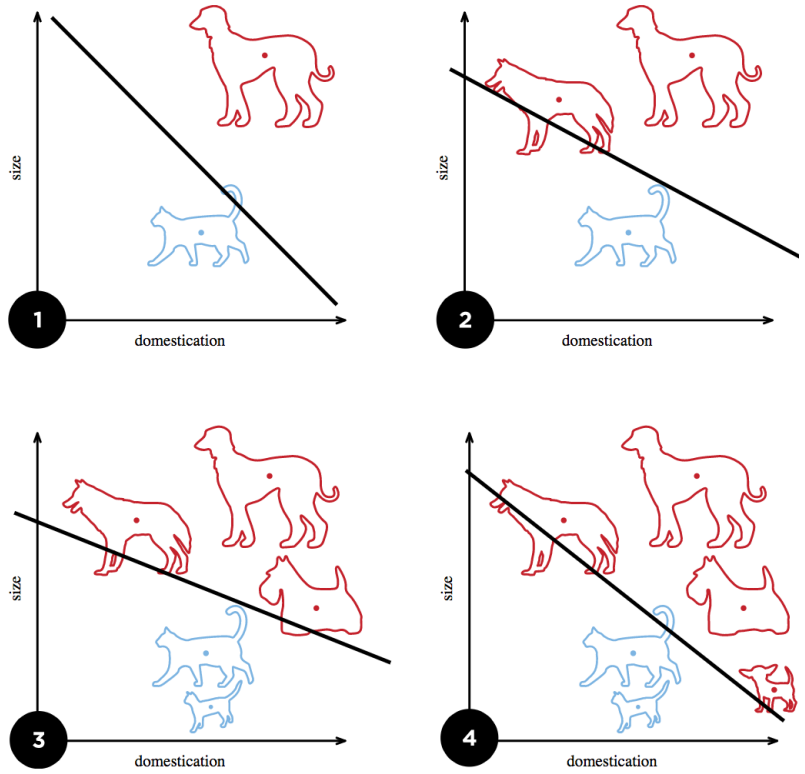
數次調整 x_t 使得 Error function $f(x_t)$ 能夠逐步最小

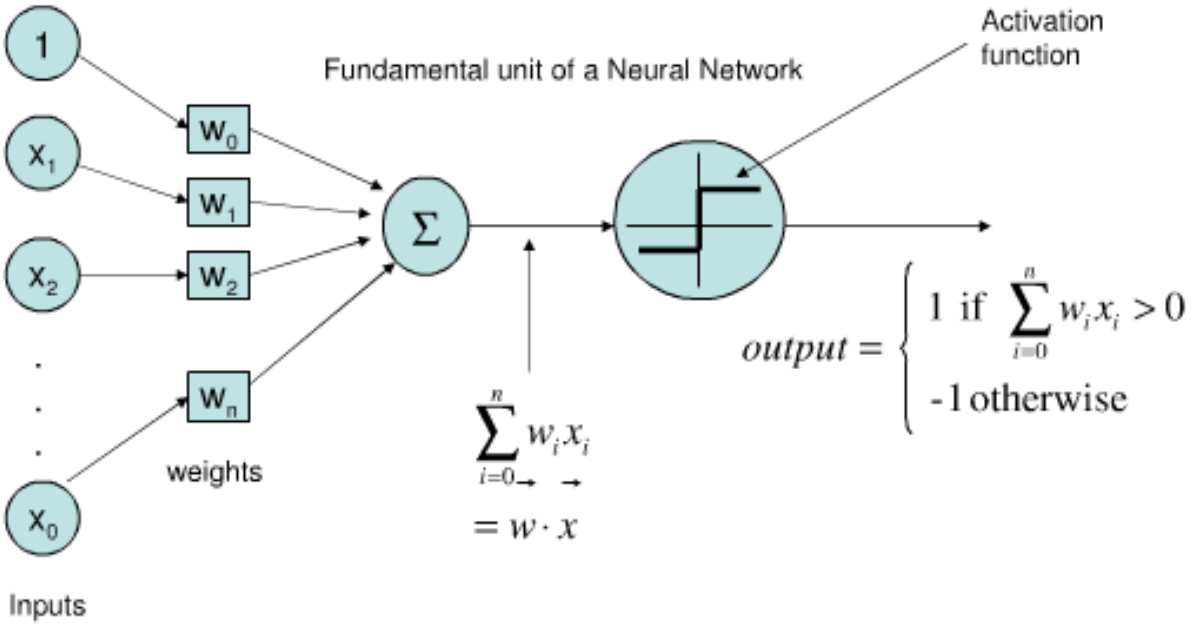
$$x_{t+1} = x_t - \alpha \nabla f(x_t) \Rightarrow f(x_0) \geq f(x_1) \geq f(x_2) \geq \ldots$$



Perceptron: 分類

學習規則可由 gradient descent 推導而來





$$E = \left(Y^p - sgn\left(\sum_i WX^p\right)\right)^2 \approx -\sum_p \left(WX^p\right) Y^p$$

$$W^{new} = W^{old} + \Delta W = W^{old} - \partial E(W)/\partial W = W^{old} + X^p Y^p$$

Delta Learning Rule

x_i驅使 y_j 產生的錯誤 $\delta_j=(t_j-y_j)$ 要透過修改 w_{ij} 來減小

$$y_j=\sum_i w_{ij}x_i$$

$$\Delta w_{ij}=x_i\delta_j=x_i(t_j-y_j)$$

上式只適用兩層且線性的網路

對任意的激發函數 f 則可由最小平方誤差法推得：

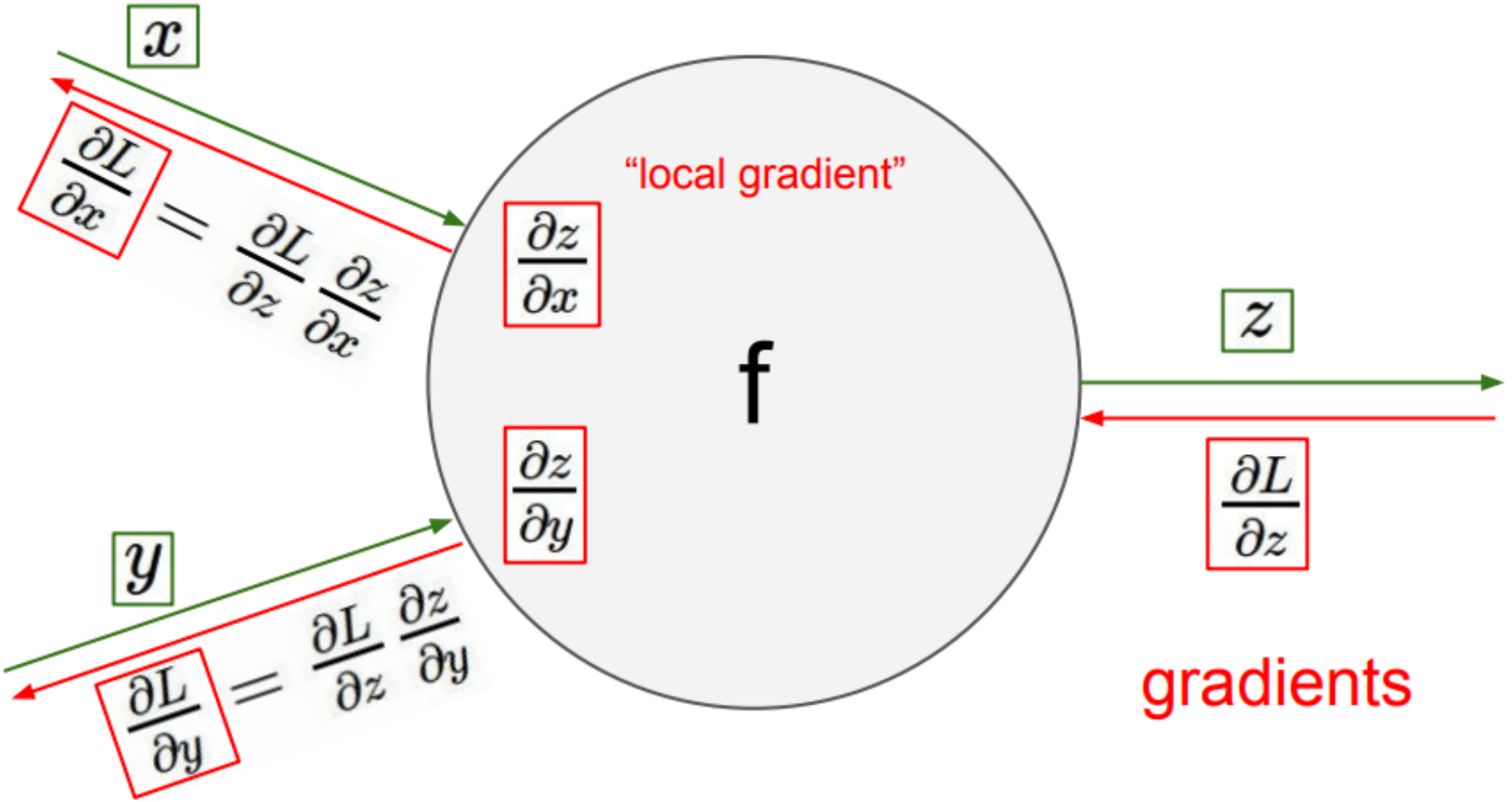
$$y_j=f\left(\sum_i w_{ij}x_i\right)$$

$$\Delta w_{ij}=x_i\delta_j f'(h_j)=x_i(t_j-y_j)f'(h_j)$$

Y=g(f(X)); dY/dX=dY/dg*dg/df*df/dX (連鎖律)

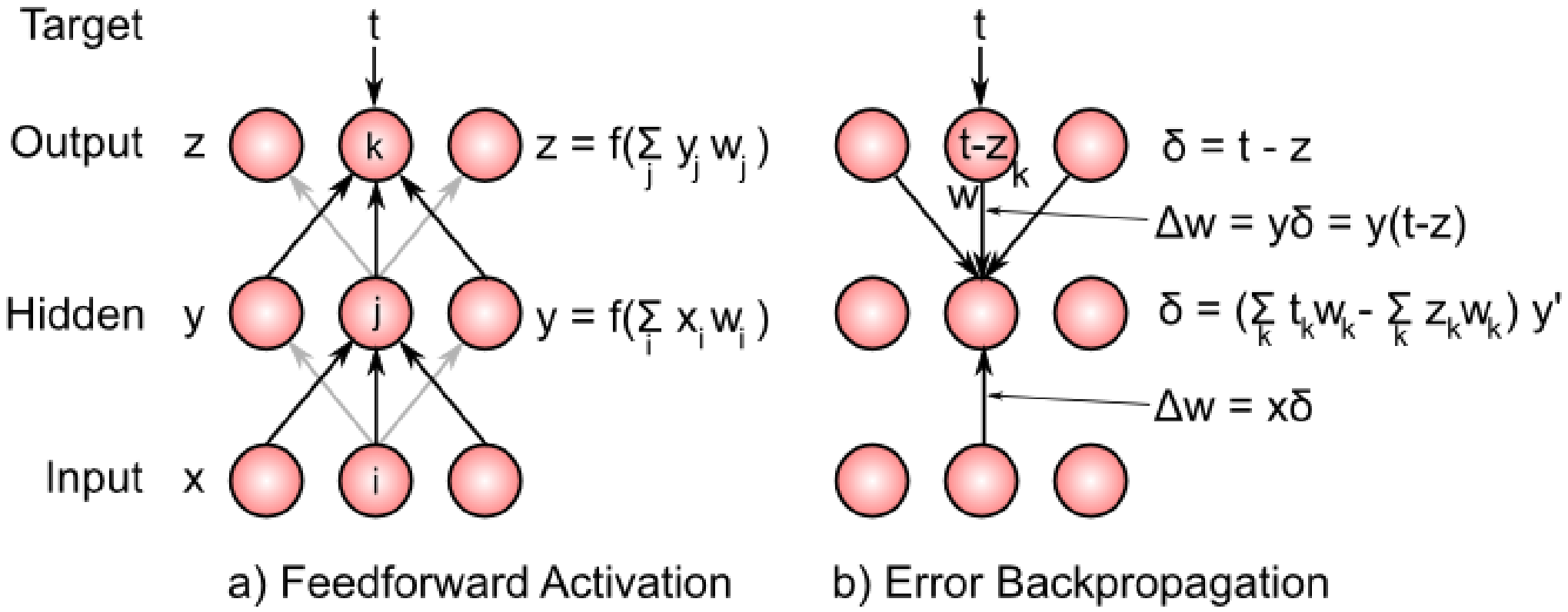
Backpropagation (1/2)

三層以上的 neural nets 常用的非生物性學習演算法



Backpropagation (2/2)

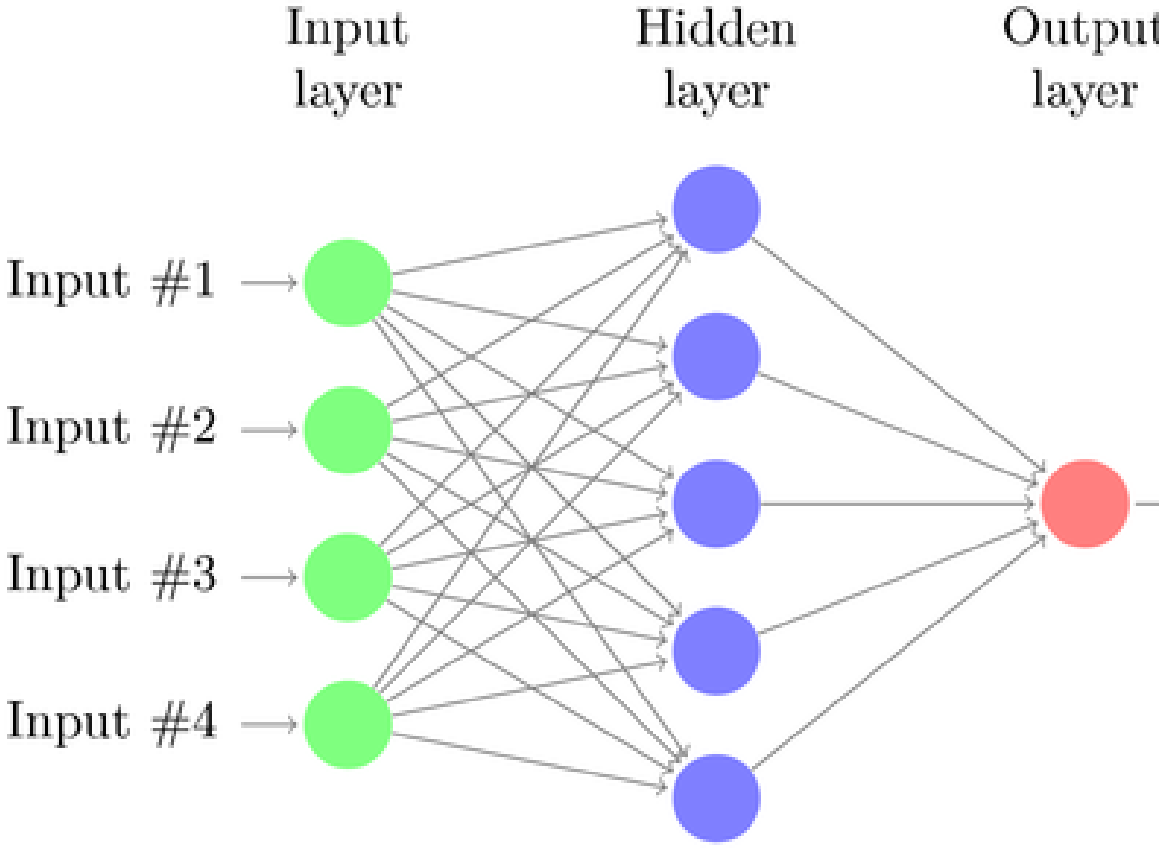
意義上是把 error signals 從 output 層往 input 層傳



在 deep neural nets 中離 output 層最遠的 $\delta \approx 0$

ANN as a Machine Learner

也會有各種歸納法 (induction) 所衍生出來的問題



Train

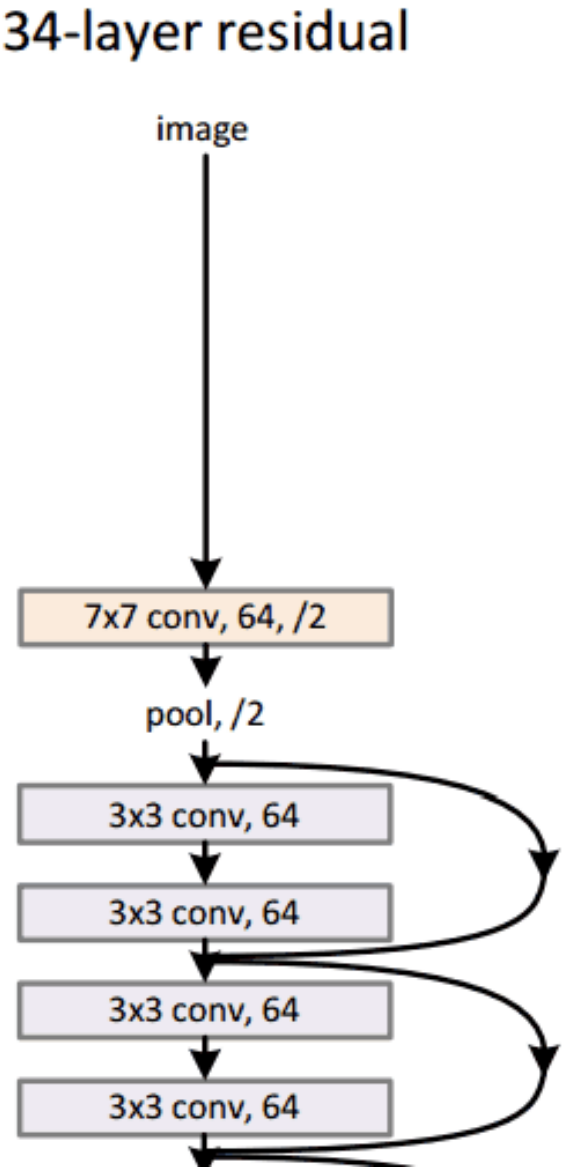
Test

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

深度學習神經網路
(Deep Learning Neural Networks)

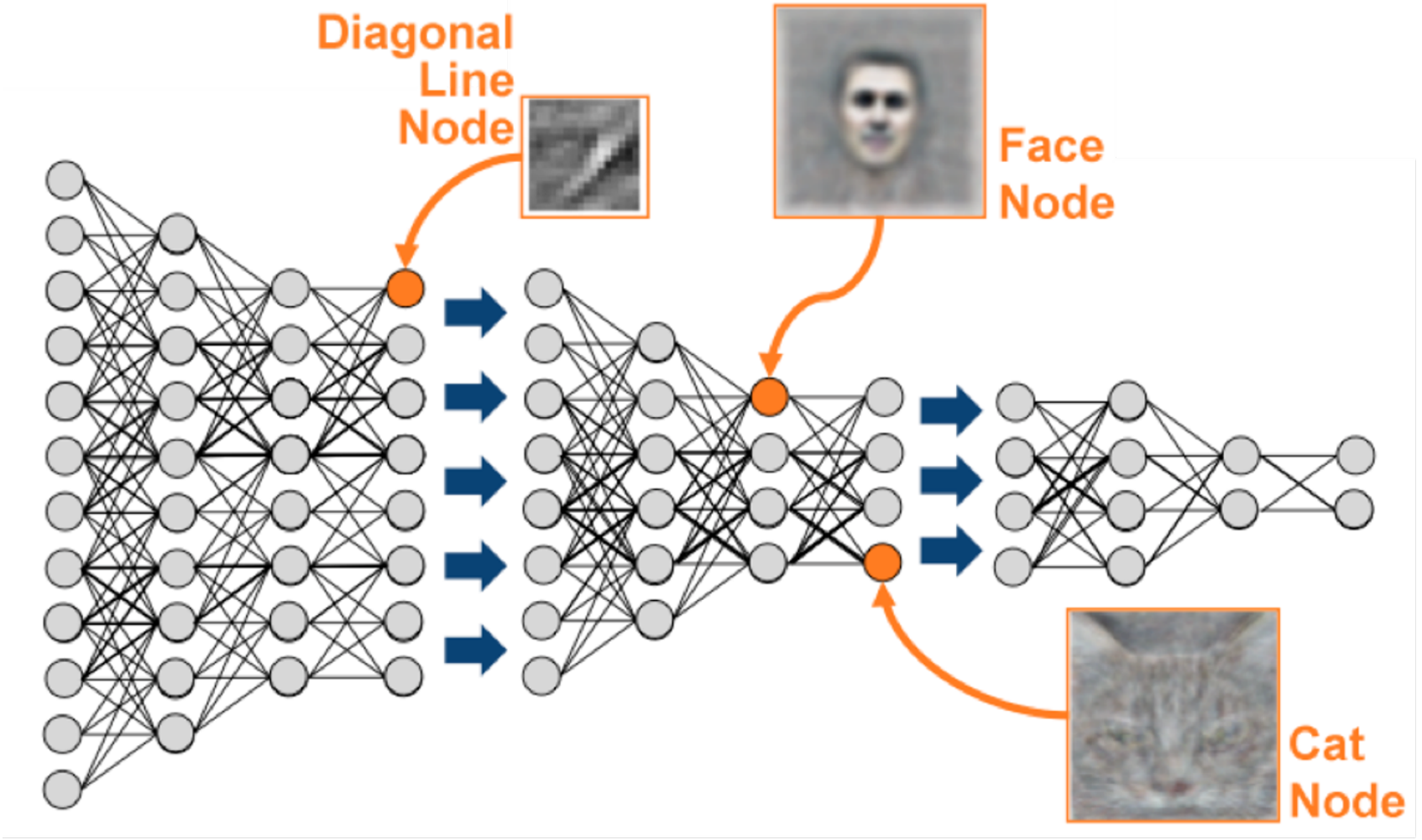
Deep Neural Network (1/2)

就是層數比較多的 Neural Net



Deep Neural Network (2/2)

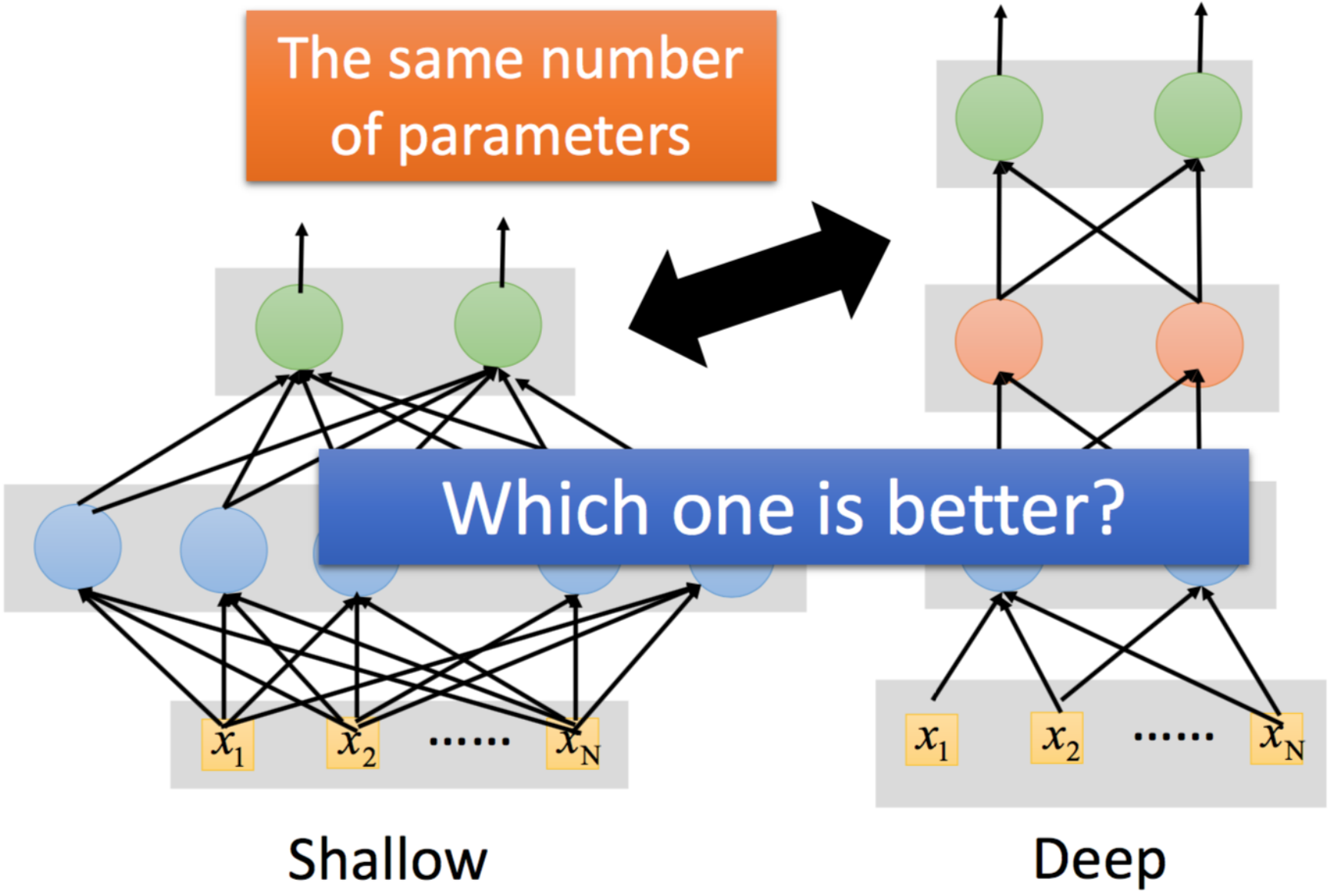
淺層偵測簡單物體；深層偵測複雜物體



解 perceptual problems w/o feature engineering

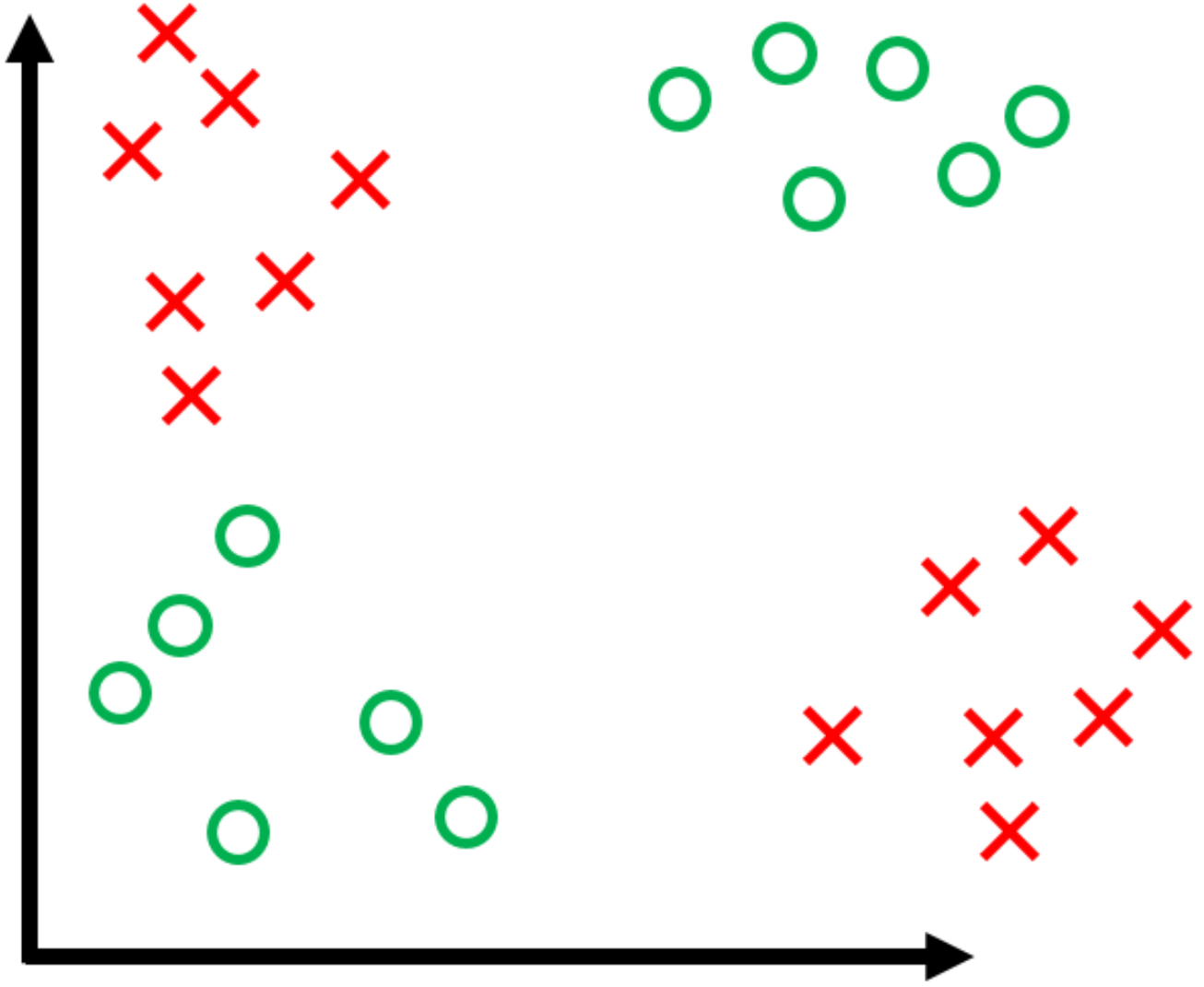
淺碟網路 vs. 深度網路：複雜度可一樣

但學習效率深度網路較好



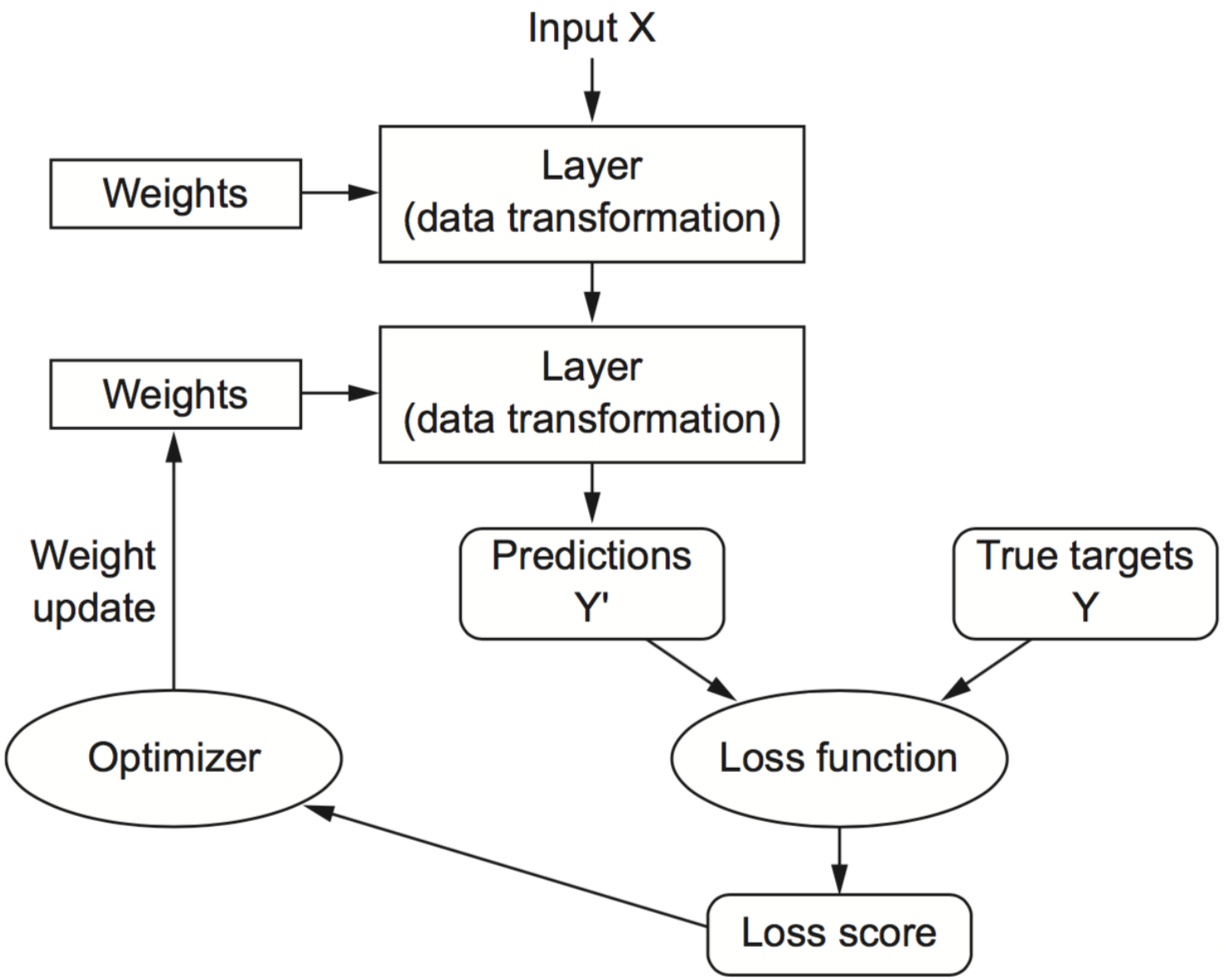
淺碟網路 vs. 深度網路：XOR Problem

DNN 的好處：先學子類別 & 減小 search space



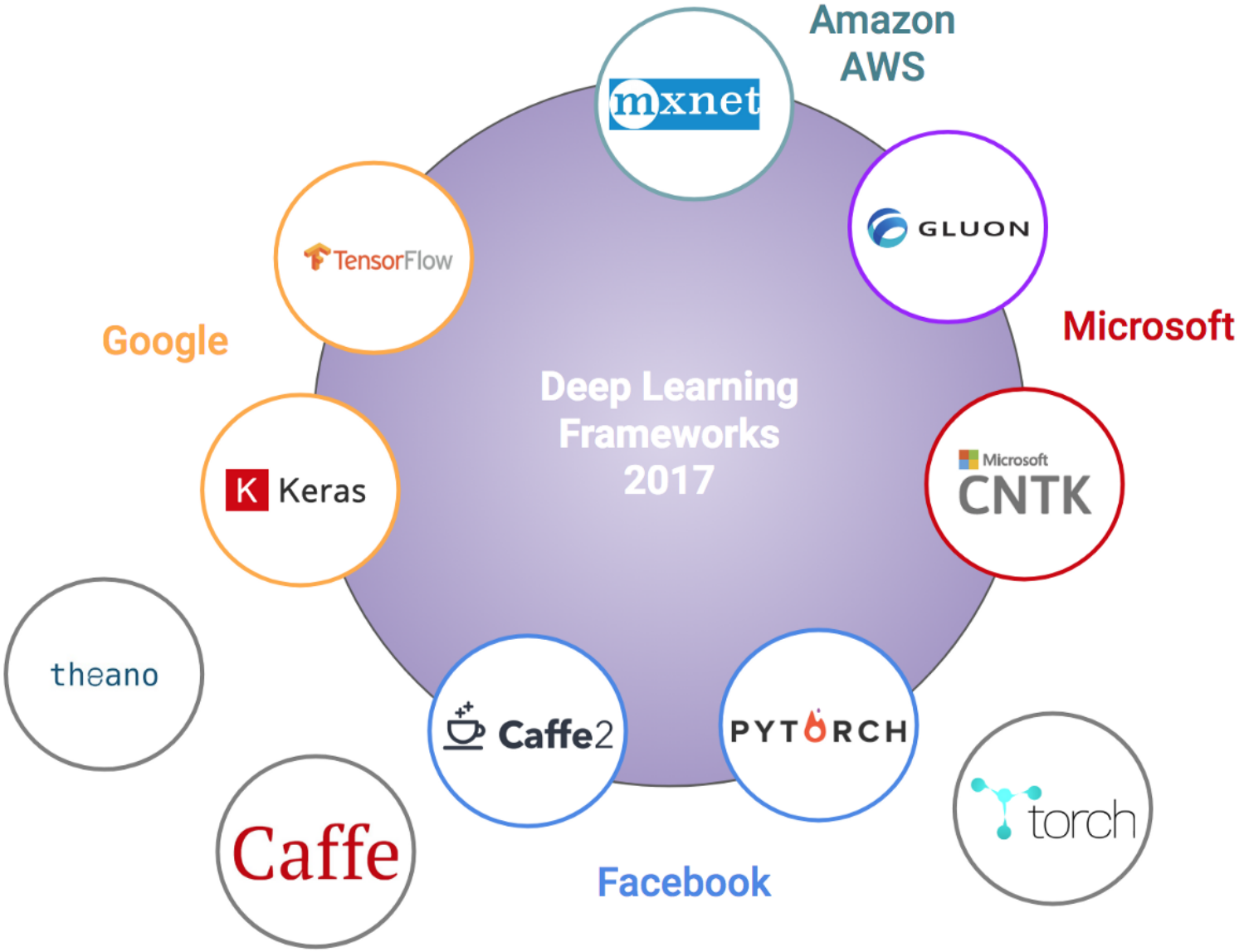
Deep Supervised Learning

和 Shallow Supervised Learning 程序一樣



Deep Learning Frameworks (1/3)

幾家大公司有各自的架構



Deep Learning Frameworks (2/3)

(中階的)PyTorch 比 TF 語法上更接近 NumPy

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```

TensorFlow

```
import numpy as np
np.random.seed(0)
import tensorflow as tf

N, D = 3, 4

with tf.device('/gpu:0'):
    x = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
    z = tf.placeholder(tf.float32)

    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tf.gradients(c, [x, y, z])

with tf.Session() as sess:
    values = {
        x: np.random.randn(N, D),
        y: np.random.randn(N, D),
        z: np.random.randn(N, D),
    }
    out = sess.run([c, grad_x, grad_y, grad_z],
                   feed_dict=values)
    c_val, grad_x_val, grad_y_val, grad_z_val = out
```

PyTorch

```
import torch
from torch.autograd import Variable

N, D = 3, 4

x = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
y = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)
z = Variable(torch.randn(N, D).cuda(),
             requires_grad=True)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

print(x.grad.data)
print(y.grad.data)
print(z.grad.data)
```

(高階的)Keras 是 (低階的)TensorFlow 之 wrapper

Deep Learning Frameworks (3/3)

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Game Over