

AI



機器學習

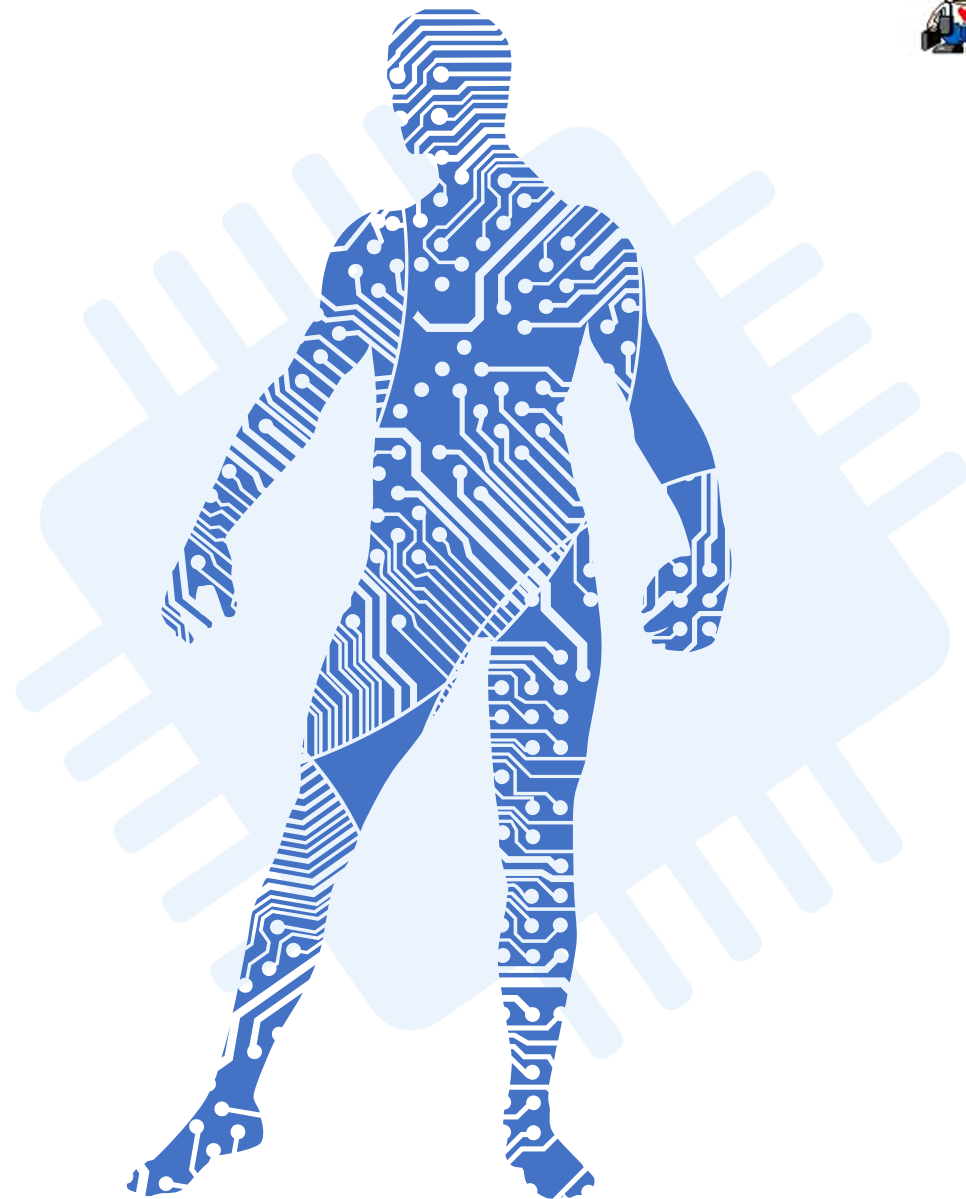
第 15 章 神經網路 (Neural Networks)

講師：紀俊男



本章大綱

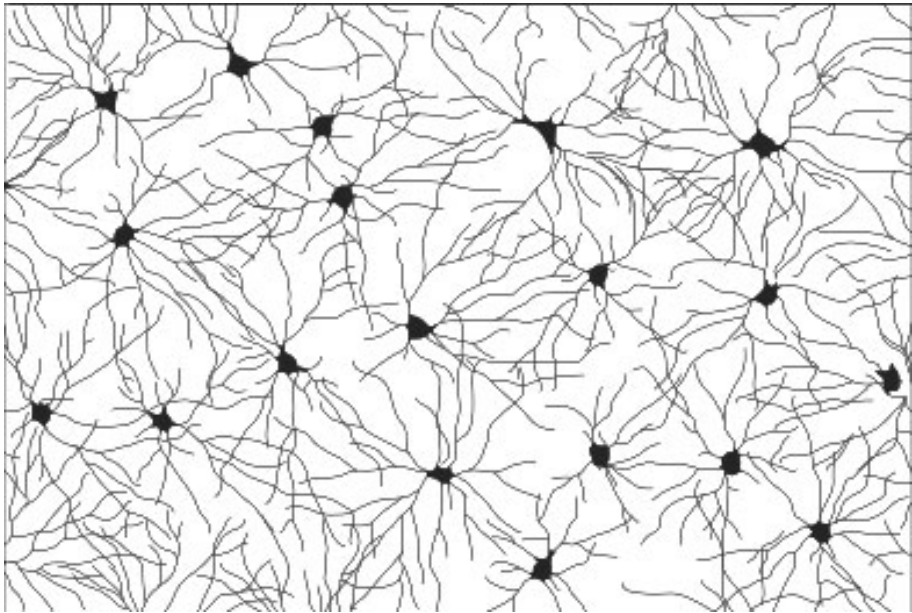
- 感知器原理解說
- 人工神經網路原理解說
- 環境安裝
- 二選一分類問題：蘑菇可吃嗎
- 多選一分類問題：鳶尾花
- 迴歸問題：50 家企業投資
- 本章總結



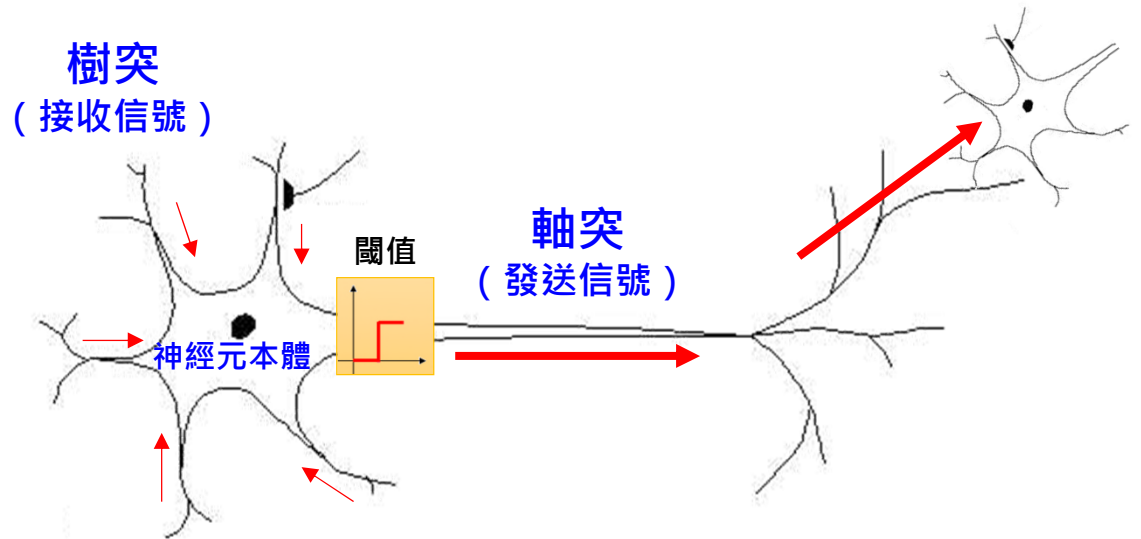


AI

感知器原理解説



生物神經網路概觀圖

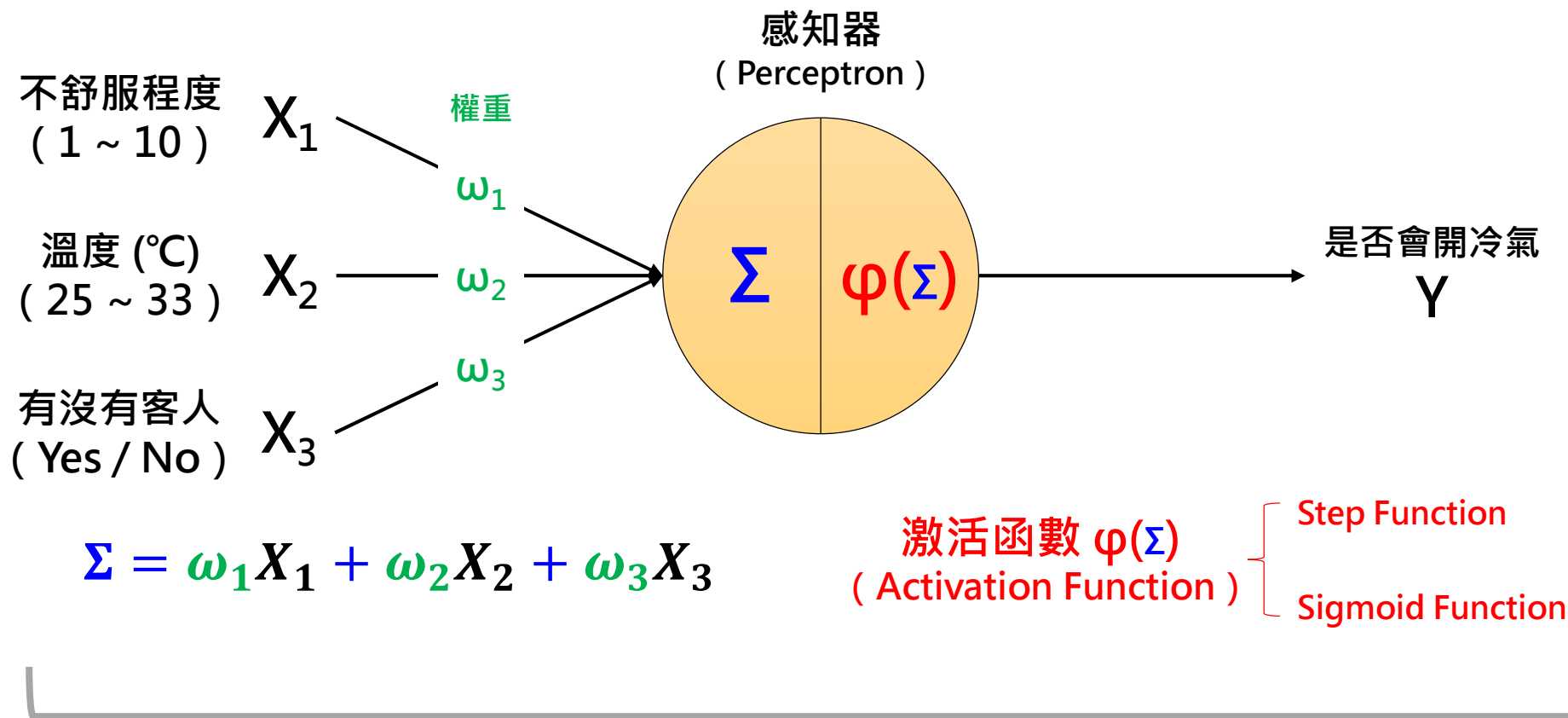


生物神經元

人工模擬出來的「神經元」



• 感知器 (Perceptron)



= 線性邏輯迴歸分類器

「感知器」範例



平常需要做
「特徵縮放」

5	8	4
28	29	32
1	0	0

不舒服程度
(1 ~ 10)

X_1

溫度 (°C)
(25 ~ 33)

X_2

有沒有客人
(Yes / No)

X_3

權重

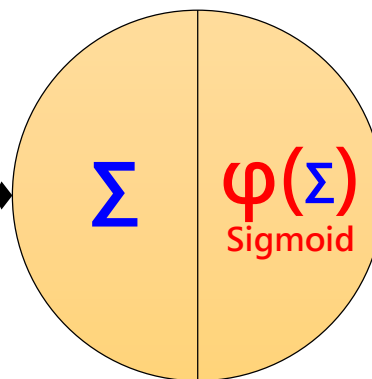
init = "uniform"

0.1

0.1

0.1

感知器
(Perceptron)



是否會開冷氣
 \hat{Y}

不舒服	溫度	客人	開冷氣
4	32	No	No
8	29	No	Yes
5	28	Yes	Yes

正向傳播 (計算損失函數)

損失函數

$$C = \frac{1}{2} (\hat{Y} - Y)^2$$

一樣本點修正一次：隨機梯度下降

K 樣本點修正一次：批次隨機梯度下降

全體樣本點修正一次：一般梯度下降

全體樣本點訓練一次 = 一期 (Epoch)

$$\Sigma = 4 * 0.1 + 32 * 0.1 + 0 * 0.1 = \underline{3.6}$$

$$\hat{Y} = \varphi(3.6) = \frac{1}{1 + e^{-3.6}} = 0.9734 = \text{Yes}$$

$$\frac{1}{2} (1 - 0)^2 = 0.5$$

$$\Sigma = 8 * 0.1 + 29 * 0.1 + 0 * 0.1 = \underline{3.7}$$

$$\hat{Y} = \varphi(3.7) = \frac{1}{1 + e^{-3.7}} = 0.9758 = \text{Yes}$$

$$\frac{1}{2} (1 - 1)^2 = 0.0$$

$$\Sigma = 5 * 0.1 + 28 * 0.1 + 1 * 0.1 = \underline{3.4}$$

$$\hat{Y} = \varphi(3.4) = \frac{1}{1 + e^{-3.4}} = 0.9677 = \text{Yes}$$

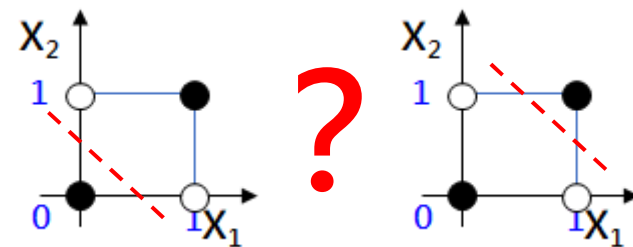
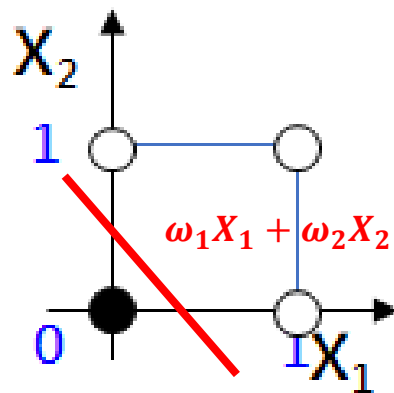
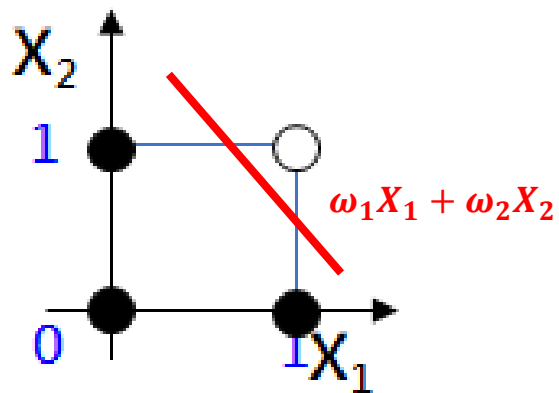
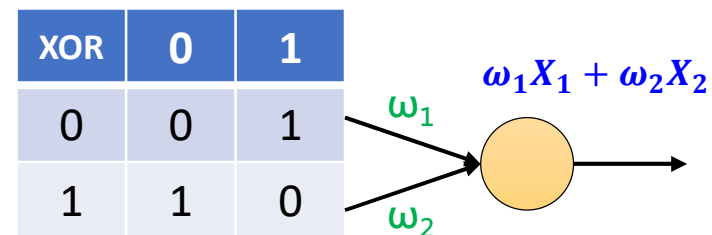
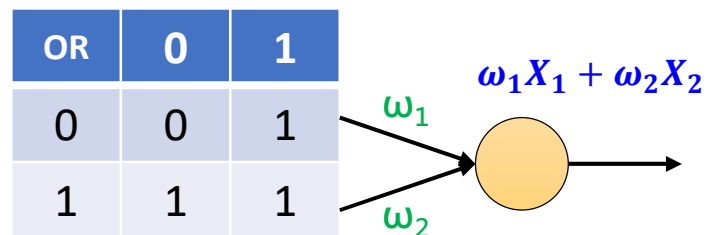
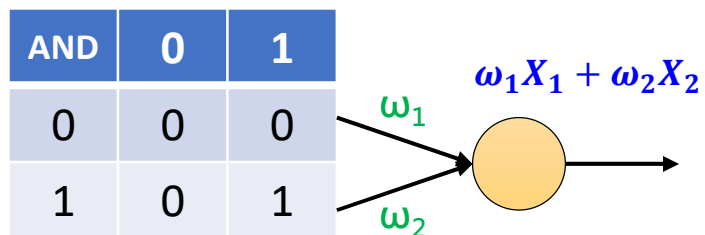
$$\frac{1}{2} (1 - 1)^2 = 0.0$$

反向傳播 (修正權重，讓損失函數有最小 (偏微分))

感知器的致命傷



- 只能用於「**線性可分**」的問題

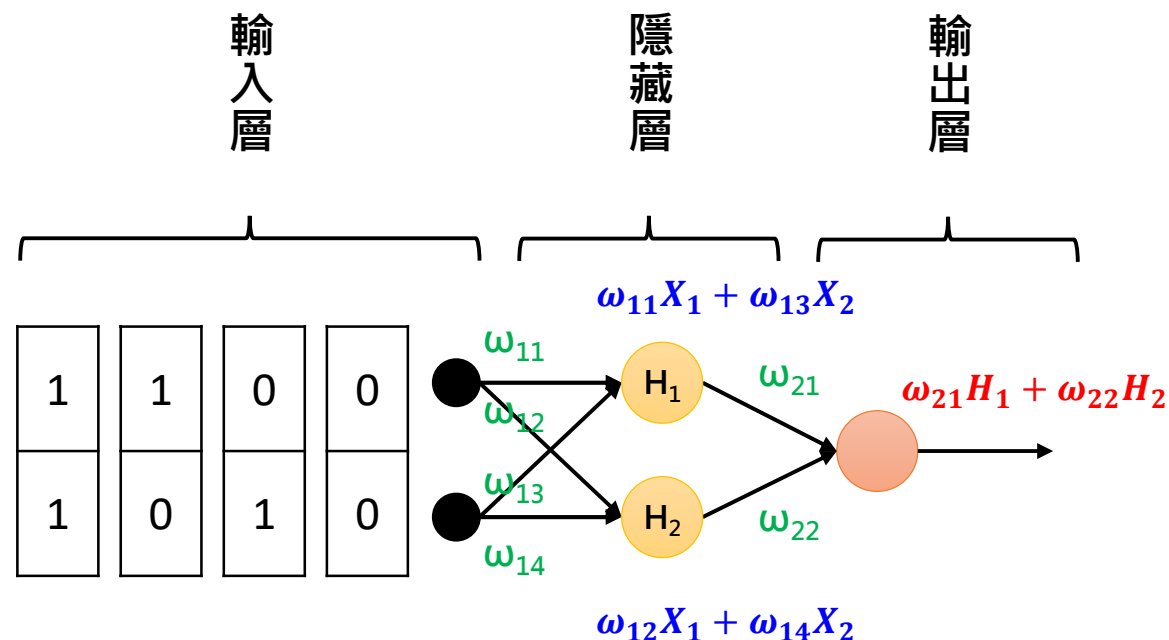
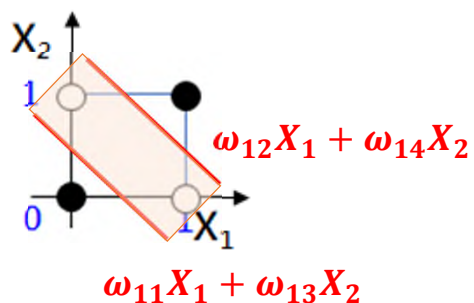


感知器致命傷的解法



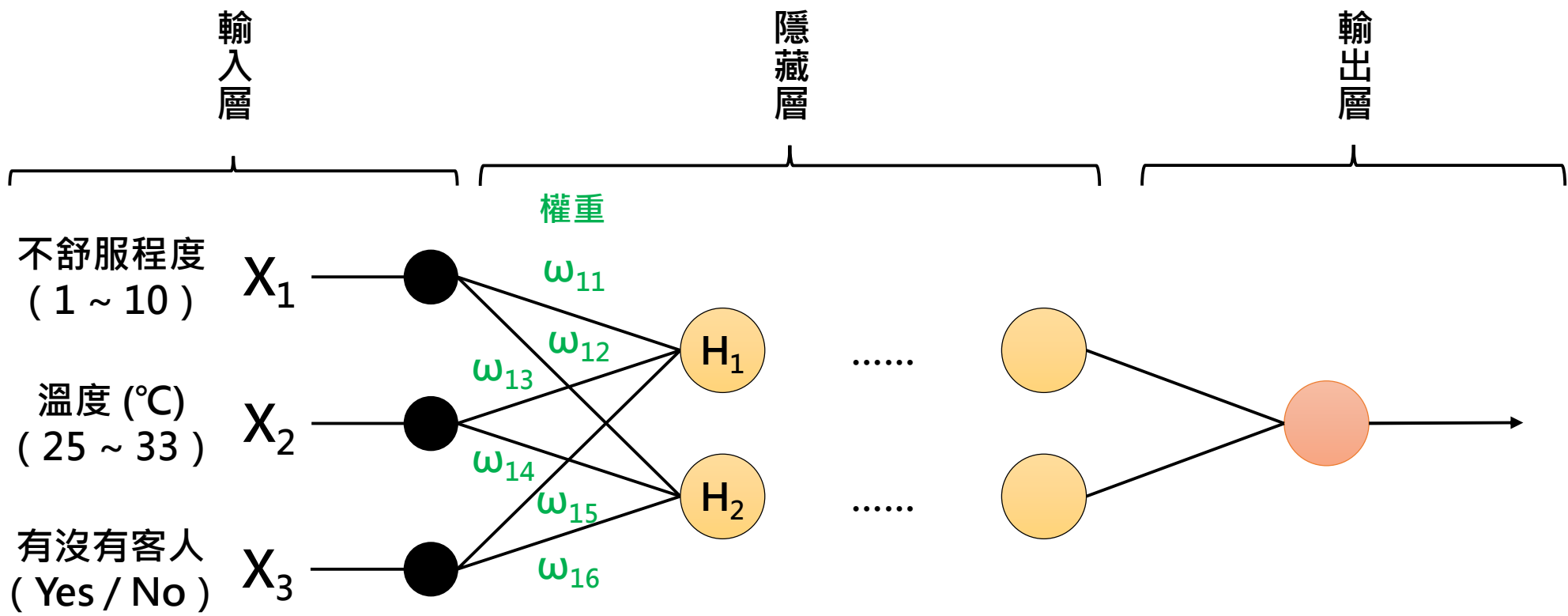
- 多層次感知器 (Multi-Layers Perceptron)
= 人工神經網路 (Artificial Neural Networks, ANN)

XOR	0	1
0	0	1
1	1	0





人工神經網路 原理解說



隱藏層作用

- 增加抽象概念 (H_1 : 以不舒服程度為主。 H_2 : 以客人有無為主)
- 將模型提昇至「能解決線性不可分問題」的等級

- 沒有定論！常用的公式如下：

公式一：算數平均數

$$\frac{\text{上一層節點數} + \text{下一層節點數}}{2}$$

公式二：經驗法則公式

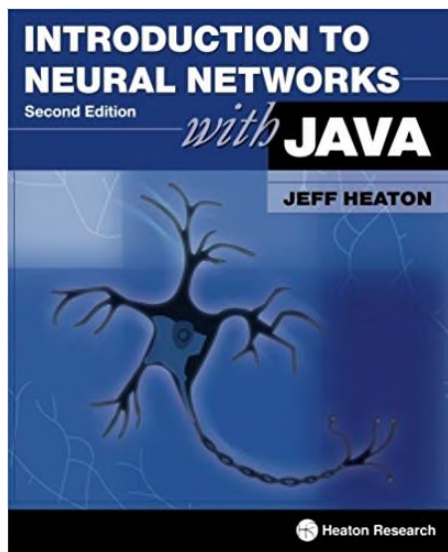
$$\frac{\text{樣本點個數}}{\alpha \times (\text{上一層節點數} + \text{下一層節點數})}$$

$\alpha=2\sim10$ (Scaling Factor)
(=2可防止過擬合，一般=5)

隱藏層該有多少層



- **沒有標準**！但依據**經驗法則**，**不需太多層**就能解大部分的問題！



- **0 層**
 - 任何「線性可分」的題目都能解。
 - 等同「線性邏輯迴歸分類器」
- **1 層**
 - 任何「有限定義域」映射至「有限值域」的函數可分的都能解。
- **2 層**
 - 任何數學函數可分的都能解。

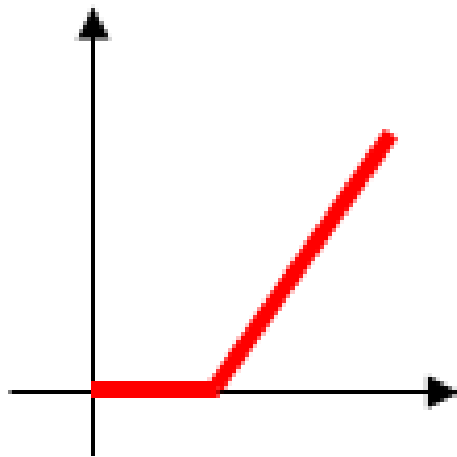
Introduction to Neural Networks
with Java (2nd Ed.)

隱藏層使用的「激活函數」



- 大多使用「**線性整流函數**」 (Rectifier Linear Unit, **ReLU**)

$$\text{ReLU} = f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



線性整流函數 (ReLU) 的好處：

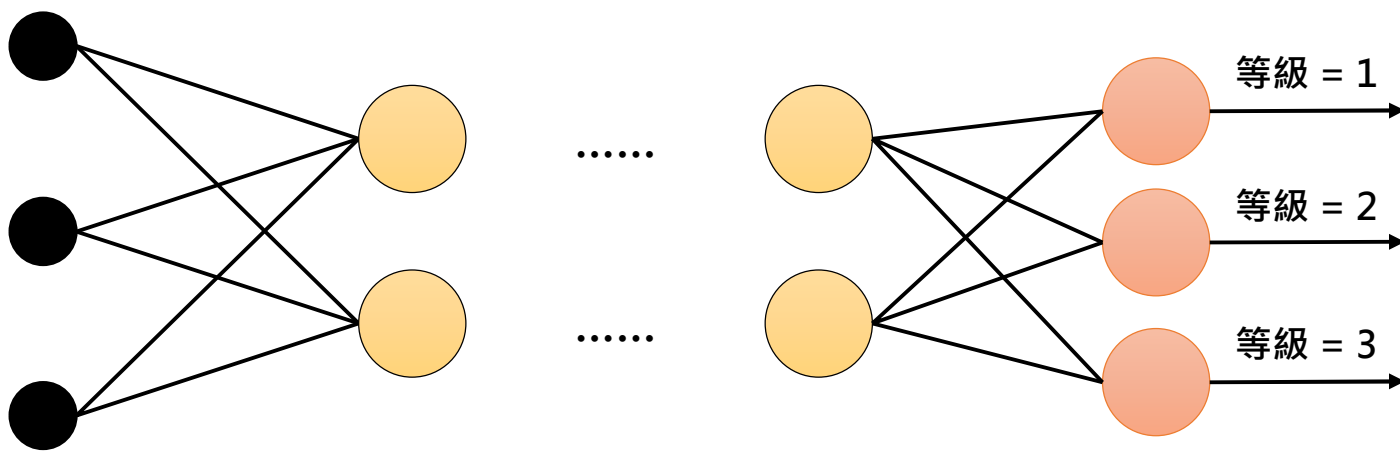
- 解決「梯度消失問題」
 - Sigmoid 的兩端是漸進線，會有「X 前進很多，Y 不太動」的問題。
 - 這會導致收斂末期，會收斂得很慢。稱為「梯度消失問題」。
- 直接切斷貢獻度小的神經元
 - 一超過「閾值」，ReLU 會直接讓它變成 0。
 - Sigmoid 會「接近 0，但不等於 0」，仍殘留一點值，拖慢運算效能。
- 較貼近生物神經元「全有或全無」的特性
 - 真正的生物神經元，低於「閾值」會直接不反應，ReLU 比較像。
 - Sigmoid 低於「閾值」，仍會保留很微弱的「殘值」。
- 節省計算量
 - ReLU 的計算量比 Sigmoid 省，不必算 e^{-x} ，效果又相近。

輸出層答案是「多選一」時



- 輸出層會超過一個節點，且改用 **Softmax 激活函數**

「紅酒評等」資料集



Softmax 函數

$$P(Y = j | X) = \frac{e^{x^T w_j}}{\sum_{k=1}^n e^{x^T w_k}}$$

- X : 特定自變數。如 (35歲, 男性)
- $P(Y=j|...)$: Y 分出來的答案是 1, 2, 3... 的機率
- $x^T w$: 所有自變數 x 所有權重
- $e^{x^T w}$: 把 e^{-x} 從 $\frac{1}{1+e^{-x}}$ 簡化出來，用以代表 $x^T w_j$ ($j = 1, 2, 3...$) 發生之機率
- 假設 $P(Y=1 | X)$ 、 $P(Y=2 | X)$ 、 $P(Y=3 | X)$... $P(Y=2 | X)$ 機率最高，則 $Y=2$ 就該被激活。



AI

環境安裝



Keras

- 基於 TensorFlow 的**上層套件**。
- 程式碼好寫，但客製化彈性小。
- 目前併入 TensorFlow 2.x 函式庫。

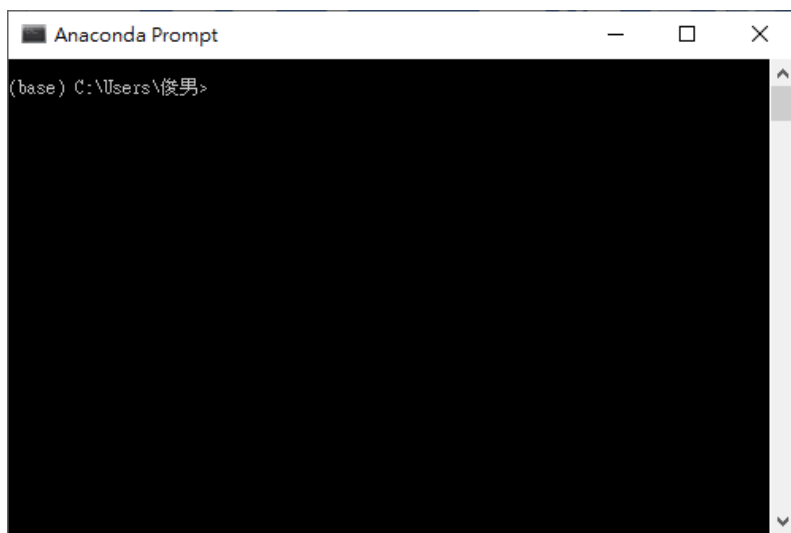


TensorFlow

- 神經網路、深度學習的**底層套件**。
- 客製化彈性大，程式碼繁瑣不好寫。



Anaconda Prompt



- 輸入下列指令安裝

- conda **install tensorflow**
 - 讓系統自由選擇安裝預設版本。
- conda **install tensorflow==2.1**
 - 如果想強迫系統安裝特定版本，可用此法。
- conda **update --all**
 - 將已安裝的所有套件，全都**更新**到最新版本。
 - 可做、可不做。安裝後發生**莫名錯誤**時可試試看。

- 若您的機器沒安裝 **Visual Studio**，**可能**會出現下列錯誤訊息。
- 不影響運作，**無視**即可！

```
C:\Users\俊男>if defined platform (set "VSREGKEY=HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\14.0" ) ELSE (set "VSREGKEY=HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\14.0" )

C:\Users\俊男>for /F "skip=2 tokens=2,*" %A in ('reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\VisualStudio\14.0" /v InstallDir') do SET "VSINSTALLDIR=%B"
錯誤：系統找不到指定的登錄機碼或值。

C:\Users\俊男>if "" == "" (set "VSINSTALLDIR=" )

C:\Users\俊男>if "" == "" (
ECHO "WARNING: Did not find VS in registry or in VS140COMNTOOLS env var - your compiler may not work"
GOTO End
)
"WARNING: Did not find VS in registry or in VS140COMNTOOLS env var - your compiler may not work"
系統無法找到指定的批次標籤 - End
```



- 進入 **Spyder** 後，在 **IPython** 輸入下列指令：

```
In [16]: import tensorflow as tf
```

← 引入 tensorflow 函式庫

```
In [17]: tf.__version__
```

← 顯示 tensorflow 當前版本

```
Out[17]: '2.1.0'
```

```
In [17]:
```

```
In [18]: import tensorflow.keras as K
```

← 引入 keras 函式庫

```
In [19]: K.__version__
```

← 顯示 keras 當前版本

```
Out[19]: '2.2.4-tf'
```



- 請先打開 Anaconda Prompt，依序輸入下列指令：
 - conda **install tensorflow** (或 conda **install tensorflow==2.1**)
 - conda **update --all** (選做)
- 用下列方法，測試 TensorFlow 與 Keras 安裝是否成功：
 - import **tensorflow** as tf
 - **tf.__version__**
 - import **tensorflow.keras** as K
 - **K.__version__**



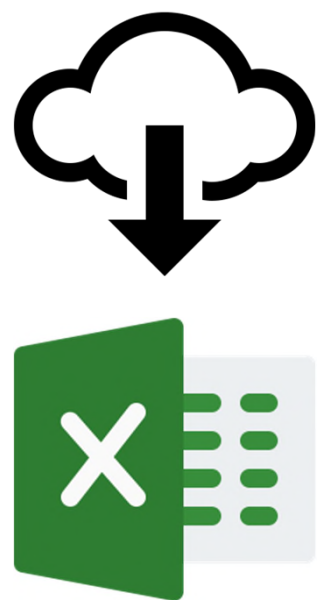


分類問題（1）： 蘑菇可以吃嗎？

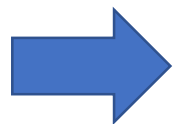
「二選一」輸出問題



- 依照講師指示，**下載**並**瀏覽**資料集



Mushroom.csv



	A	B	C	D
1	class	cap-shape	cap-surface	cap-color
2	p	x	s	n
3	e	x	s	y
4	e	b	s	w
5	p	x	y	w
6	e	x	s	g
7	e	x	y	y
8	e	b	s	w
9	e	b	y	w
10	p	x	y	w

目的：利用「**傘帽**、**表皮**、**傘色...**」
--> 推算「**香菇是否有毒**」

- class** : p-有毒、e-可吃
- cap-shape** : **傘帽**
b-鐘形、c-錐形、x-圓形、f-扁形、k-把手形、s-凹陷
- cap-surface** : **傘皮**
f-纖維感、g-凹洞、s-鱗片
s-光滑
- cap-color** : **傘色**
n-棕、b-淺黃、c-肉桂、g-灰、r-綠、p-粉紅、u-紫
e-紅、w-白、y-黃
- bruises** : **瘀傷** (t-有/f-無)
- odor** : **氣味**。a-杏仁、l-茴香
c-雜酚油味、y-魚腥味、f-臭
m-霉、n-無、p-辛、s-辣
-
- 完整列表：
<https://bit.ly/3kiiCs7>

撰寫程式碼

```
1 import HappyML.preprocessor as pp
2
3 # Load Data
4 dataset = pp.dataset(file="Mushrooms.csv")
5
6 # Decomposition
7 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(1, 23)], y_columns=[0])
8
9 # Dummy Variables
10 X = pp.onehot_encoder(X, columns=[i for i in range(22)], remove_trap=True)
11 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
12
13 # Feature Selection
14 from HappyML.preprocessor import KBestSelector
15 selector = KBestSelector(best_k="auto")
16 X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
17
18 # Split Training / TEsting Set
19 X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
```

- 資料前處理流程：
1. 載入資料
 2. 切分自變數、應變數
 3. 處理缺失資料
(無缺失資料)
 4. 類別資料數位化
(+ 移除虛擬變數陷阱)
 5. 特徵選擇
 6. 切分訓練集、測試集
 7. 特徵縮放



隨堂練習：資料前處理



- 請撰寫下列程式碼，並予以執行，完成「**資料前處理**」的步驟：

```
1 import HappyML.preprocessor as pp
2
3 # Load Data
4 dataset = pp.dataset(file="Mushrooms.csv")
5
6 # Decomposition
7 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(1, 23)], y_columns=[0])
8
9 # Dummy Variables
10 X = pp.onehot_encoder(X, columns=[i for i in range(22)], remove_trap=True)
11 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
12
13 # Feature Selection
14 from HappyML.preprocessor import KBestSelector
15 selector = KBestSelector(best_k="auto")
16 X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
17
18 # Split Training / TEsting Set
19 X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
```





- `zeros`、`ones`：全為 0、全為 1。
- `random_uniform`：從 0~1 間的 uniform 分布挑一個值
- `random_normal`：從 $\mu=0, \sigma=0.05$ 的常態分布挑數值
- `glorot_uniform`：用扇入扇出節點計算出來的 uniform 分布版本（收斂快）
- `glorot_normal`：用扇入扇出節點計算出來的常態分布版本（收斂快）

```
1 # In[] Neural Networks without HappyML's Class
2 from tensorflow.keras.models import Sequential ←負責建造「整個」神經網路
3 from tensorflow.keras.layers import Dense ←負責建造神經網路的「一層」
4
5 # Initialize the whole Neural Networks
6 classifier = Sequential()
7
8 # Add the Input & First Hidden Layer 輸入層個數 第一隱藏層個數
9 classifier.add(Dense(input_dim=X_train.shape[1], units=45, kernel_initializer="random_normal", activation="relu"))
10
11 # Add the Second Hidden Layer 第二隱藏層個數
12 classifier.add(Dense(units=23, kernel_initializer="random_normal", activation="relu"))
13
14 # Add the Output Layer 輸出層個數 輸出層激活函數
15 classifier.add(Dense(units=1, kernel_initializer="random_normal", activation="sigmoid"))
```

- `relu`：常用於「隱藏層」。
- `linear`：常用於「迴歸問題」的輸出層。
- `sigmoid`：常用於「二選一」分類問題的輸出層。
- `softmax`：常用於「多選一」分類問題的輸出層。

- **sgd** : Stochastic Gradient Descent (隨機梯度下降) 使用偏微分，逐步尋找「導數 = 0」那一點的方法。
- **adam** : Adaptive Moment Estimation 。學習速率全自動調整，收斂速度快，適用於大部分情況。
- **adadelta** : Adaptive Delta 。一種無需決定學習速率的演算法。

收斂演算法

- **binary_crossentropy** : 用於輸出為「二選一」的場合。
- **categorical_crossentropy** : 用於輸出為「多選一」的場合。
- **mse** : 用於輸出為「連續值」(迴歸) 的場合。

損失函數

- **accuracy**、**recall....** : 用於分類問題。
- **mse** : 用於迴歸問題。

效能評估指標

```
1 # Compile the whole Neural Networks
2 classifier.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
3
4 # Fit
5 classifier.fit(x=X_train, y=Y_train, batch_size=10, epochs=100)
6
7 # Predict
8 import pandas as pd
9 Y_pred = classifier.predict(x=X_test).astype(int)
10 Y_pred = pd.DataFrame(Y_pred, index=Y_test.index, columns=Y_test.columns)
```

每多少樣本點，就做一次反向傳播
「期數」(Epochs)
所有樣本點要重取樣幾次
(以求訓練/驗證取樣公平性)

隨堂練習：神經網路建造與訓練



- 請撰寫、並執行前兩頁的**程式碼**。
- 用肉眼比較看看 **Y_test** (真實值) vs. **Y_pred** (預測值) 的差別。





效能評估



```
1 from HappyML.performance import ClassificationPerformance
2
3 pfm = ClassificationPerformance(Y_test, Y_pred)
4
5 print("Confusion Matrix:\n", pfm.confusion_matrix())
6 print("Accuracy: {:.2%}".format(pfm.accuracy()))
7 print("Recall: {:.2%}".format(pfm.recall()))
8 print("Precision: {:.2%}".format(pfm.precision()))
9 print("F1-score: {:.2%}".format(pfm.f_score()))
```



- 請撰寫前一頁「效能評估」的程式碼，並執行看看：

```
1 from HappyML.performance import ClassificationPerformance
2
3 pfm = ClassificationPerformance(Y_test, Y_pred)
4
5 print("Confusion Matrix:\n", pfm.confusion_matrix())
6 print("Accuracy: {:.2%}".format(pfm.accuracy()))
7 print("Recall: {:.2%}".format(pfm.recall()))
8 print("Precision: {:.2%}".format(pfm.precision()))
9 print("F1-score: {:.2%}".format(pfm.f_score()))
```





分類問題（2）： 鳶尾花

「多選一」分類問題

載入 Iris 資料集 →

切分自變數 X、應變數 Y
並取得 Y 代碼對應名稱
(0=setosa, 1=versicolor...etc.)

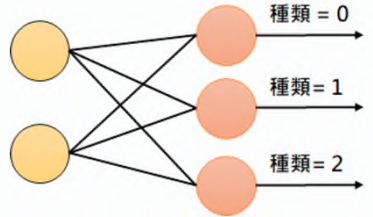
Y 做「獨熱編碼」，
以符合神經網路輸出層

用卡方檢定法，
挑選顯著性高的特徵

切分訓練集、測試集 →

特徵縮放 →

```
1 from sklearn.datasets import load_iris
2 import HappyML.preprocessor as pp
3
4 # Load Data
5 dataset = load_iris()
6
7 # X, Y
8 import pandas as pd
9 X = pd.DataFrame(dataset.data, columns=dataset.feature_names)
10 Y = pd.DataFrame(dataset.target, columns=["Iris_Type"])
11 Y_name = dataset.target_names.tolist()
12
13 # One_hot incoder for Y
14 Y = pp.onehot_encoder(ary=Y, columns=[0])
15
16 # Feature Selection
17 from HappyML.preprocessor import KBestSelector
18 selector = KBestSelector(best_k="auto")
19 X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
20
21 # Split Training / Testing Set
22 X_train, X_test, Y_train, Y_test = pp.split_train_test(x_ary=X, y_ary=Y)
23
24 # Feature Scaling
25 X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```



	0
48	0
49	0
50	1
51	1

Index	Iris_Type_0	Iris_Type_1	Iris_Type_2
48	1	0	0
49	1	0	0
50	0	1	0
51	0	1	0



建構神經網路



```
1 # In[] Neural Networks without HappyML's Class
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense
4
5 # Initialize the whole Neural Networks
6 classifier = Sequential()
7
8 # Add the Input & First Hidden Layer
9 classifier.add(Dense(input_dim=X_train.shape[1], units=20, kernel_initializer="random_normal", activation="relu"))
10
11 # Add the Second Hidden Layer
12 classifier.add(Dense(units=10, kernel_initializer="random_normal", activation="relu"))
13
14 # Add the Output Layer 「三選一」的輸出層 「三選一」的激活函數
15 classifier.add(Dense(units=3, kernel_initializer="random_normal", activation="softmax"))
```


編譯、訓練

```
1 # Compile the whole Neural Networks
2 classifier.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
3
4 # Fit
5 classifier.fit(x=X_train, y=Y_train, batch_size=10, epochs=100)
6
7 # Predict
8 import pandas as pd
9 Y_pred = classifier.predict(x=X_test)
10 Y_pred = (Y_pred > 0.5).astype(int)
11 Y_pred = pd.DataFrame(Y_pred, index=Y_test.index).idxmax(axis=1)
12
13 Y_test.columns = [0, 1, 2]
14 Y_test = Y_test.idxmax(axis=1)
```

預測答案
(改回單欄)

	0	1	2
0	0.997379	0.00262133	2.35755e-08
1	2.12966e-06	0.0732935	0.926704

真實答案
(也改回單欄)

	0	1	2
0	1	0	0
1	0	0	1

idxmax: Index Max
傳回最大值所在之索引

Index	0
27	0
137	2

Index	Iris_Type_0	Iris_Type_1	Iris_Type_2
27	1	0	0
137	0	0	1

Index	0	1	2
27	1	0	0
137	0	0	1

Index	0
27	0
137	2

```
1 from HappyML.performance import ClassificationPerformance
2
3 pfm = ClassificationPerformance(Y_test, Y_pred)
4
5 print("Confusion Matrix:\n", pfm.confusion_matrix())
6 print("Accuracy: {:.2%}".format(pfm.accuracy()))
7 print("Recall: {:.2%}".format(pfm.recall()))
8 print("Precision: {:.2%}".format(pfm.precision()))
9 print("F1-score: {:.2%}".format(pfm.f_score()))
```



- 請撰寫前幾頁程式碼，並執行看看。

```
The Significant Level: 0.05
```

```
--- The p-values of Feature Importance ---
```

```
TRUE <0.05 5.53397228e-26 (petal length (cm))
```

```
TRUE <0.05 2.75824965e-15 (petal width (cm))
```

```
TRUE <0.05 4.47651499e-03 (sepal length (cm))
```

```
FALSE >0.05 1.56395980e-01 (sepal width (cm))
```

```
Number of Features Selected: 3
```

```
Confusion Matrix:
```

```
[[10  0  0]
```

```
 [ 0 15  0]
```

```
 [ 0  0 13]]
```

```
Accuracy: 100.00%
```

```
Recall: 100.00%
```

```
Precision: 100.00%
```

```
F1-score: 100.00%
```





迴歸問題：
50 家企業投資

	[0]	[1]	[2]	[3]	[4]
1	R&D Spend	Administratio	Marketing Sp	State	Profit
2	165349.2	136897.8	471784.1	New York	192261.83
3	162597.7	151377.59	443898.53	California	191792.06
4	153441.51	101145.55	407934.54	Florida	191050.39

載入資料集 →

```
1 import HappyML.preprocessor as pp
2
3 # Dataset Loading
4 dataset = pp.dataset("50_Startups.csv")
5
```

切分自變數、
應變數 →

```
6 # Independent/Dependent Variables Decomposition
7 X, Y = pp.decomposition(dataset, [0, 1, 2, 3], [4])
8
```

所在州名
獨熱編碼 →

```
9 # Apply One Hot Encoder to Column[3]
10 X = pp.onehot_encoder(X, columns=[3], remove_trap=True)
11
```

切分訓練集、
測試集 →

```
12 # Split Training vs. Testing Set
13 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y, train_size=0.8)
14
```

特徵縮放 →

```
15 # Feature Scaling (optional)
16 X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
17 Y_train, Y_test = pp.feature_scaling(fit_ary=Y_train, transform_arys=(Y_train, Y_test))
```



建構神經網路



```
1 # In[] Neural Networks without HappyML's Class
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense
4
5 # Initialize the whole Neural Networks
6 regressor = Sequential()
7
8 # Add the Input & First Hidden Layer
9 regressor.add(Dense(input_dim=X_train.shape[1], units=3, kernel_initializer="normal", activation="relu"))
10
11 # Add the Second Hidden Layer
12 regressor.add(Dense(units=2, kernel_initializer="normal", activation="relu"))
13
14 # Add the Output Layer
15 regressor.add(Dense(units=1, kernel_initializer="normal", activation="linear"))
```

迴歸的激活函數

迴歸的損失函數 迴歸的效能評估

```
1 # Compile the whole Neural Networks
2 regressor.compile(optimizer="adam", loss="mse", metrics=["mse"])
3
4 # Fit
5 regressor.fit(x=X_train, y=Y_train, batch_size=5, epochs=50)
6
7 # Predict
8 import pandas as pd
9 Y_pred = pd.DataFrame(regressor.predict(x=X_test), index=Y_test.index, columns=Y_test.columns)
```

```
1 # In[] Performance with RMSE
2 from HappyML.performance import rmse
3
4 print("The RMSE of Neural Networks: {:.4f}".format(rmse(Y_test, Y_pred)))
```




- 請撰寫前幾頁程式碼，並執行看看。

```
Epoch 44/50  
40/40 [=====] - 0s 275us/step - loss: 0.7057 - mean_squared_error: 0.7057  
Epoch 45/50  
40/40 [=====] - 0s 275us/step - loss: 0.6957 - mean_squared_error: 0.6957  
Epoch 46/50  
40/40 [=====] - 0s 250us/step - loss: 0.6821 - mean_squared_error: 0.6821  
Epoch 47/50  
40/40 [=====] - 0s 275us/step - loss: 0.6718 - mean_squared_error: 0.6718  
Epoch 48/50  
40/40 [=====] - 0s 275us/step - loss: 0.6604 - mean_squared_error: 0.6604  
Epoch 49/50  
40/40 [=====] - 0s 275us/step - loss: 0.6499 - mean_squared_error: 0.6499  
Epoch 50/50  
40/40 [=====] - 0s 275us/step - loss: 0.6380 - mean_squared_error: 0.6380
```

The RMSE of Neural Networks: 1.0488





課後作業：挽留銀行客戶



- 說明
 - 某歐洲銀行調查了一萬名客戶的往來記錄，以及到目前為止是否還留在本銀行。全部存放在 `Churn_Modelling.csv` 檔內（原始來源：<https://is.gd/oWXEsw>）。
 - 該資料集的自變數 X 欄位如下：
 - `RowNumber`：流水號，我們不需要。
 - `CustomerId`：客戶編號，我們不需要。
 - `Surname`：客戶的姓，我們不需要。
 - `CreditScore`：信用分數。存款多寡、還款能力...等綜合分數。
 - `Geography`：國別。
 - `Gender`：性別。
 - `Age`：年齡。
 - `Tenure`：帳號持有年份。
 - ...
 - 該資料集的應變數 Y 如下：
 - `Exited`：客戶是否離開本銀行（1=是，0=否）



- 要求
 - 請使用**神經網路**，設計一款分類器，可以用來預測客戶是否可能離開本銀行。
 - 請自行選用一款「**特徵選擇工具**」（Kbest、PCA），去除不必要的特徵。
 - 請印出「**混淆矩陣、確度、廣度、精度、F 值**」等指標，以說明你模型的效能。

- 輸出

The Significant Level: 0.05

```
--- The p-values of Feature Importance ---
TRUE <0.05 0.00000000e+00 (Age)
TRUE <0.05 0.00000000e+00 (Balance)
TRUE <0.05 0.00000000e+00 (EstimatedSalary)
TRUE <0.05 5.81457176e-51 (Geography_Germany)
TRUE <0.05 1.56803624e-27 (IsActiveMember)
TRUE <0.05 9.96353608e-25 (CreditScore)
TRUE <0.05 7.01557451e-13 (Gender_Male)
TRUE <0.05 4.92250487e-06 (Geography_Spain)
TRUE <0.05 2.45493956e-02 (NumOfProducts)
FALSE >0.05 7.05344899e-02 (Tenure)
FALSE >0.05 6.98496209e-01 (HasCrCard)
```

Number of Features Selected: 9

Confusion Matrix:

```
[[1932  74]
 [ 288 206]]
```

Accuracy: 85.52%

Recall: 69.01%

Precision: 80.30%

F1-score: 72.33%



- 將訓練好的神經網路模型存起來：
 - classifier.save("my_model.h5")

```
1 8948 4446 0d0a 1a0a 0000 0000 0008 0800
2 0400 1000 0000 0000 0000 0000 0000 0000
3 ffff ffff ffff ffff 9825 0100 0000 0000
4 ffff ffff ffff ffff 0000 0000 0000 0000
5 6000 0000 0000 0000 0100 0000 0000 0000
```

HDF5 格式

- Hierarchical Data Format 5.0
- 用來儲存大量資料的一種檔案格式。
- 1987 年開發於美國國家超級計算應用中心。

- 載入一個神經網路模型檔：
 - from tensorflow.keras.models import load_model
 - model = load_model("my_model.h5")



- 感知器 (Perceptron)

- 結構：輸入值、權重、感知器本身、激活函數、輸出值。
- 正向傳播：計算「損失函數」值。
- 反向傳播：修正「權重」，以降低「損失函數」值。
- 感知器缺點：只能做「線性可分」的問題。

- 人工神經網路 (Artificial Neural Networks, ANN)

- 隱藏層：替模型增加「抽象概念」與「非線性要素」。
- 隱藏層節點計算：
 - $(\text{上層} + \text{下層}) / 2$
 - $\text{樣本點} / (a * (\text{上層} + \text{下層}))$ ， $a = 2 \sim 10$ 。
- 隱藏層層數：
 - 除了影像辨識、集群...等特定問題外，1~3 層就已足夠。
- 輸出層：
 - 「二選一」分類問題：1 節點
 - 「多選一」分類問題：節點數 = 應變數 Y 所有變化數
 - 迴歸問題：1 節點



- 環境安裝

- 需要的函式庫：[TensorFlow](#)、[Keras](#)
- 安裝方法：
 - `conda install tensorflow`
 - `conda update --all` (選做)

- 常見的激活函數 (Activation Functions)

- 線性整流函數 ([ReLU](#))：常用於「隱藏層」中。
- [Linear](#) 函數：常用於迴歸問題的「輸出層」中。
- [Sigmoid](#) 函數：常用於「二選一」分類問題的「輸出層」中。
- [Softmax](#) 函數：常用於「多選一」分類問題的「輸出層」中。

- 常見的權重初始化演算法 (Kernel Initializers)

- [zeros](#)：將所有權重，通通初始化為 0。
- [ones](#)：將所有權重，通通初始化為 1。
- [random_uniform](#)：從 0~1 間的 uniform 分布挑一個值。
- [random_normal](#)：從 $\mu=0, \sigma=0.05$ 的常態分布挑數值。
- [glorot_uniform](#)：用扇入扇出節點計算出來的 uniform 分布版本。
- [glorot_normal](#)：用扇入扇出節點計算出來的常態分布版本。





本章總結

• 常見的收斂演算法 (Optimizers)

- `sgd` : 隨機梯度下降法 (Stochastic Gradient Descendent) 。使用偏微分，逐步尋找「導數 = 0」那一點的方法。
- `adam` : Adaptive Moment Estimation 。學習速率全自動調整，收斂速度快，適用於大部分情況。
- `adadelta` : Adaptive Delta 。一種無需決定學習速率的演算法。

• 常見的損失函數 (Loss Functions)

- `binary_crossentropy` : 用於輸出為「二選一」の場合。
- `categorical_crossentropy` : 用於輸出為「多選一」の場合。
- `mse` : 用於輸出為「連續值」(迴歸) の場合。

• 常見的效能評估指標 (Performance Metrics)

- `accuracy`、`recall`.... : 用於分類問題。
- `mse` : 用於迴歸問題。

• 存載神經網路模型

- 儲存 : `classifier.save("my_model.h5")`
- 載入 :
 - `from tensorflow.keras.models import load_model`
 - `model = load_model("my_model.h5")`

