勞動部產業新尖兵計畫

人工智慧金融應用與實務培訓班

課程模組： AI 金融科技課程 - AI 程式設計

5.CNN 實務 - 時間序列處理

葉建華 (Yeh, Jian-hua)

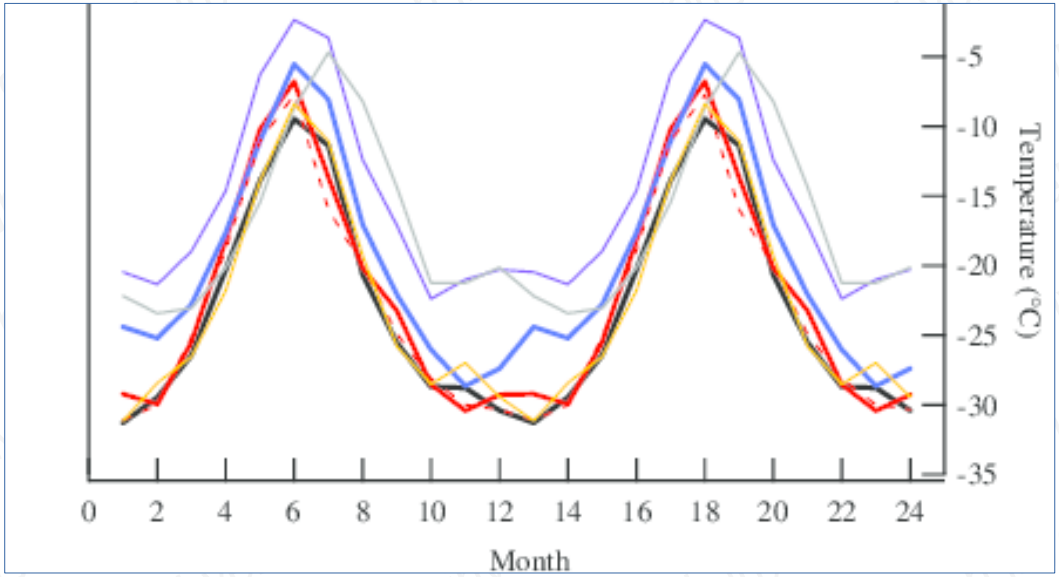
tdi.jhyeh@tdi.edu.tw
au4290@gmail.com

講次內容

- 什麼是時間序列預測問題
- 單變量 CNN 模型
- 多變量 CNN 模型
- 多時階 CNN 模型

時間序列預測?

- 格陵蘭 24 個月的氣溫觀測，你看出了什麼?



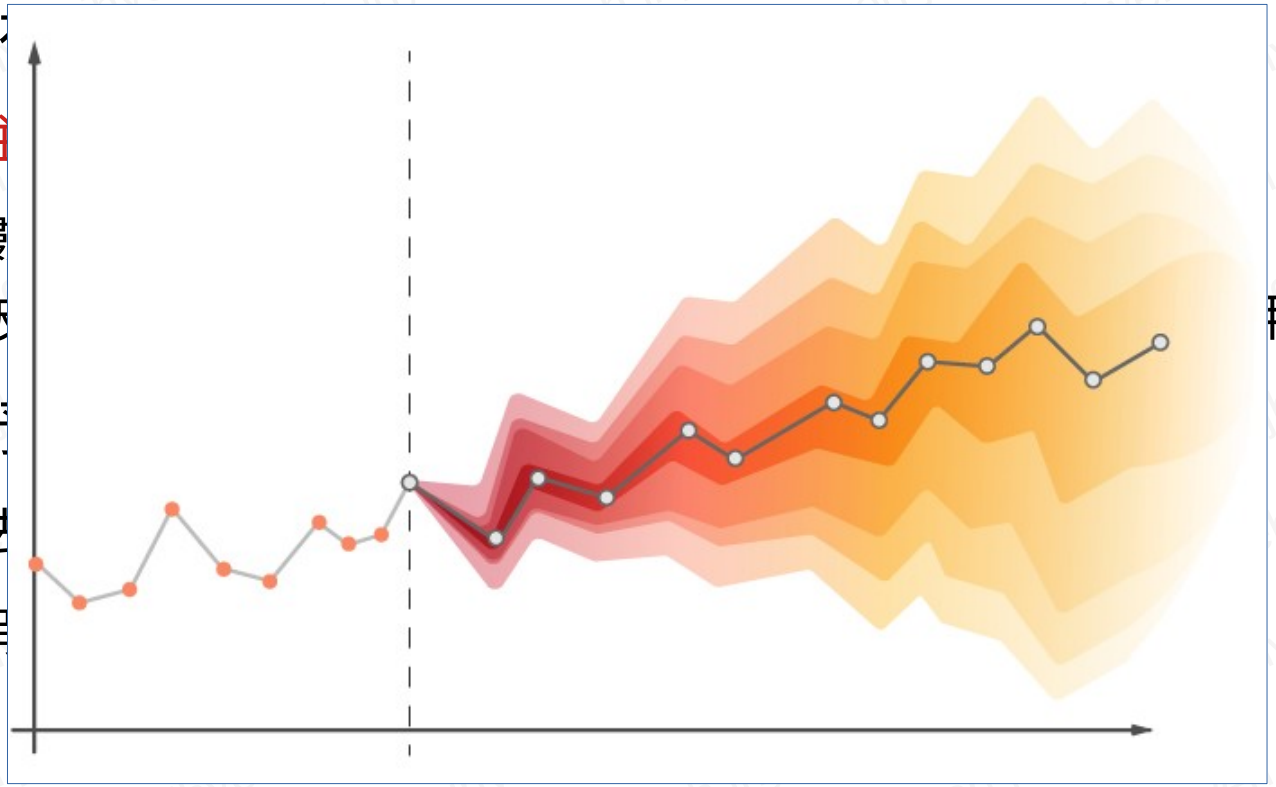
如果要你再畫接下來的 12 個月，你要怎麼畫?

時間序列預測問題

- 測量主體在一定時間範圍內的每一時點上的量測值
  - 重點：主體、時間、測量值
  - 應用：數理統計、信號處理、樣式識別、計量經濟、量化金融、天氣地震、生物電波、控制工程、通信功能，所有涉及時間數據測量的應用
- 時間序列預測
  - 根據過去的變化趨勢預測未來的發展
  - 前提：假定事物的過去延續到未來（就是歷史會重演！）

時間序列預測問題

- 測量主體
  - 重點：主
  - 應用：數
    天氣地震、
    生物電波
    用
- 時間序列
  - 根據過去
  - 前提：假



講次內容

- 什麼是時間序列預測問題
- 單變量 CNN 模型
- 多變量 CNN 模型
- 多時階 CNN 模型

單變量 CNN 模型

- 就是單一數據的時間序例
- 應用於預測「下一步」

單變量 CNN 模型

- 資料準備：切出（題目，答案）配對
- 想想看處理邏輯應該是什麼?

10, 20, 30, 40, 50, 60, 70, 80, 90 =>

X,	y
10, 20, 30	40
20, 30, 40	50
30, 40, 50	60
40, 50, 60	70
50, 60, 70	80
60, 70, 80	90

- 資料準備：
- 想想看處理

10, 20, 30, 40, 50,

```
# 時間序列資料，串列前後元素有時序的先後順序
list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# 我們希望資料變成一段一段小的時序資料
#  X,                  y
# 10, 20, 30          40
# 20, 30, 40          50
# 30, 40, 50          60
# ...

def splitSequence(seq, steps):
    x, y = [], []
    for i in range(len(seq)):
        # 找到本次取樣的尾端索引
        end_ix = i + steps
        # 檢查是否該結束了
        if end_ix > len(seq)-1:
            break
        # 切出兩部份!
        seq_x, seq_y = seq[i:end_ix], seq[end_ix]
        x.append(seq_x)
        y.append(seq_y)
    return x, y

n_steps = 3
listX, listY = splitSequence(list1, n_steps)

for x, y in zip(listX, listY):
    print(x, y)
```

（題目，答案）配對

```
[10, 20, 30] 40
[20, 30, 40] 50
[30, 40, 50] 60
[40, 50, 60] 70
[50, 60, 70] 80
[60, 70, 80] 90
```

80
90

單變量 CNN 模型

- 特徵處理
  - 使用 Keras 來製作 CNN 模型
  - 針對 Keras API Conv1D 所需調整資料維度

```
import numpy as np

arrayX = np.array(listX)
arrayY = np.array(listY)
print(arrayX.shape)
print(arrayY.shape)

n_features = 1
arrayX = arrayX.reshape((arrayX.shape[0], arrayX.shape[1], n_features))
print(arrayX.shape)
```

```
(6, 3)
(6,)
(6, 3, 1)
```

單變量 CNN 模型

- 預測模式
  - 「下一步」預測， one-step prediction
  - 使用 Keras API 的 1D CNN 模型

單變量 CNN 模型

- 預測模式

```
from tensorflow.keras.layers import Activation, Conv1D, MaxPooling1D, Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))

print(model.summary())
model.compile(optimizer=Adam(lr=0.001), loss='mse')
```

單變量 CNN 模型

- 預測模式

```
from tensorflow.keras.layer
from tensorflow.keras.model
from tensorflow.keras.optim

model = Sequential()

model.add(Conv1D(filters=64
model.add(MaxPooling1D(pool
model.add(Flatten())
model.add(Dense(50, activat
model.add(Dense(1))

print(model.summary())
model.compile(optimizer=Ada
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_5 (Conv1D)            (None, 2, 64)             192
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 1, 64)             0
_________________________________________________________________
flatten_4 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 50)                3250
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 51
=================================================================
Total params: 3,493
Trainable params: 3,493
Non-trainable params: 0
_________________________________________________________________
None
```

談一下 mse

- 就是均方差！ $$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$
  - 預測值和實際觀測值間差的平方的均值
  - 只考慮誤差的平均大小，不考慮其方向
  - 又被稱為 L2 損失 或 L2 範數損失
  - 以後看到 L2 norm 就是指這個啦！

為什麼要平方?

方向是什麼意思?

那 **L1 norm** 是什麼?

單變量 CNN 模型

- 訓練 train 資料集
  - model.fit() 又出現了!

```
import matplotlib.pyplot as plt

train_history = model.fit(arrayX, arrayY, epochs=1000, verbose=1)

plt.plot(train_history.history['loss'], label='loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

單變量 CNN 模型

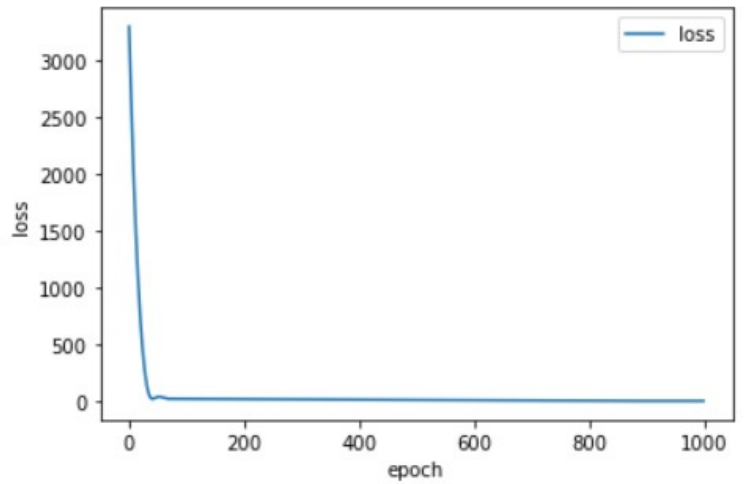
- 訓練 train 資料集
  - model.fit() 又

```
import matplotlib.pyplot as

train_history = model.fit(ar

plt.plot(train_history.histo
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

```
Epoch 998/1000
1/1 [==============================] - 0s 6ms/step - loss: 1.5347e-08
Epoch 999/1000
1/1 [==============================] - 0s 10ms/step - loss: 1.2961e-08
Epoch 1000/1000
1/1 [==============================] - 0s 7ms/step - loss: 1.2621e-08
```



單變量 CNN 模型

- 自製 test 資料做測試
  - 使用 model.predict()

```
x_input = np.array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)

x_input = np.array([7, 8, 9])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)

x_input = np.array([75, 85, 95])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)
```

你觀察到了什麼?

```
[[101.25561]]
[[12.900721]]
[[106.892365]]
```

講次內容

- 什麼是時間序列預測問題
- 單變量 CNN 模型
- 多變量 CNN 模型
- 多時階 CNN 模型

多變量 CNN 模型

- 就是多個來源數據的時間序例
- 應用於預測「下一步」

多變量 CNN 模型

- 資料準備：切出（題目，答案）配對
- 想想看處理邏輯應該是什麼?

10, 20, 30, 40, 50, 60, 70, 80, 90
15, 25, 35, 45, 55, 65, 75, 85, 95

=>

X,	y
[10 15]	
[20 25]	65
[30 35]	
[20 25]	
[30 35]	85
[40 45]	
...	

- 資料準備：
- 想想看處理

10, 20, 30, 40, 50,

```python
import numpy as np

# 資料來源: 兩個串列
list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90]
list2 = [15, 25, 35, 45, 55, 65, 75, 85, 95]
# list3是預計用來輸出的資料
list3 = [list1[i]+list2[i] for i in range(len(list1))]

# 轉換串列成numpy陣列
inSeq1 = np.array(list1)
inSeq1 = inSeq1.reshape((len(list1), 1))
inSeq2 = np.array(list2)
inSeq2 = inSeq2.reshape((len(list2), 1))
outSeq = np.array(list3)
outSeq = outSeq.reshape((len(list3), 1))

# 縱向堆疊成為資料集
dataset = np.hstack((inSeq1, inSeq2, outSeq))
print(dataset)

def split_sequences(seq, steps):
    x, y = [], []
    for i in range(len(seq)):
        # 找到本次取樣的尾端索引
        end_ix = i + steps
        # 檢查是否該結束了
        if end_ix > len(seq):
            break
        # 切出兩部份!
        seq_x, seq_y = seq[i:end_ix, :-1], seq[end_ix-1, -1]
        # gather input and output parts of the pattern
        seq_x, seq_y = seq[i:end_ix, :-1], seq[end_ix-1, -1]
        x.append(seq_x)
        y.append(seq_y)
    return x, y

n_steps = 3

# 轉出訓練模型所需要的資料形式
listX, listY = split_sequences(dataset, n_steps)
for x, y in zip(listX, listY):
    print(x, y)
```

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
[[10 15]
 [20 25]
 [30 35]] 65
[[20 25]
 [30 35]
 [40 45]] 85
[[30 35]
 [40 45]
 [50 55]] 105
[[40 45]
 [50 55]
 [60 65]] 125
[[50 55]
 [60 65]
 [70 75]] 145
[[60 65]
 [70 75]
 [80 85]] 165
[[70 75]
 [80 85]
 [90 95]] 185
```

(題目，答案)
配對

多變量 CNN 模型

- 特徵處理
  - 使用 Keras 來製作 CNN 模型
  - 針對 Keras API Conv1D 所需調整資料維度

```
import numpy as np

# [樣本, 輸出時階] 轉成 [樣本,  輸出時階, 特徵數]
n_features = 1
arrayX = np.array(listX)
arrayY = np.array(listY)
arrayX = arrayX.reshape((arrayX.shape[0], arrayX.shape[1], n_features))
print(arrayX.shape)
```

```
(5, 3, 1)
(5, 2)
```

多變量 CNN 模型

- 預測模式
  - 「下一步」預測， one-step prediction
  - 使用 Keras API 的 1D CNN 模型

多變量 CNN 模型

- 預測模式

```
from tensorflow.keras.layers import Activation, Conv1D, MaxPooling1D, Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))

print(model.summary())
model.compile(optimizer=Adam(lr=0.001), loss='mse')
```

多變量 CNN 模型

- 預測模式

```
from tensorflow.keras.laye
from tensorflow.keras.mode
from tensorflow.keras.opti

model = Sequential()

model.add(Conv1D(filters=6
model.add(MaxPooling1D(poo
model.add(Flatten())
model.add(Dense(50, activa
model.add(Dense(1))

print(model.summary())
model.compile(optimizer=Ad
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_2 (Conv1D)            (None, 2, 64)             320
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 1, 64)             0
_________________________________________________________________
flatten_2 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 50)                3250
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51
=================================================================
Total params: 3,621
Trainable params: 3,621
Non-trainable params: 0
_________________________________________________________________
None
```

多變量 CNN 模型

- 訓練 train 資料集
  - model.fit() 又出現了!

```
import matplotlib.pyplot as plt

train_history = model.fit(arrayX, arrayY, epochs=1500, verbose=1)

plt.plot(train_history.history['loss'], label='loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

多變量 CNN 模型

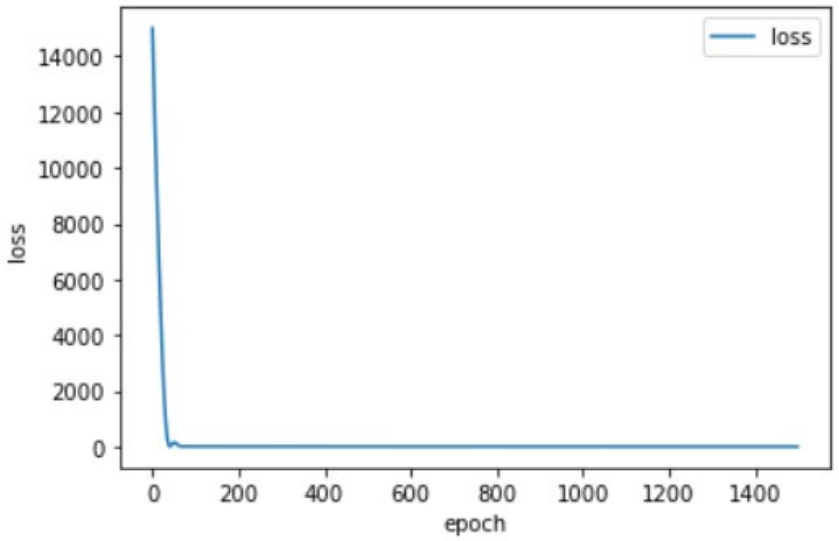
- 訓練 train 資料集
  - model.fit()

```
import matplotlib.pyplo

train_history = model.

plt.plot(train_history
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

```
Epoch 1498/1500
1/1 [==============================] - 0s 5ms/step - loss: 0.3926
Epoch 1499/1500
1/1 [==============================] - 0s 8ms/step - loss: 0.3900
Epoch 1500/1500
1/1 [==============================] - 0s 4ms/step - loss: 0.3874
```



多變量 CNN 模型

- 自製 test 資料做測試
  - 使用 model.predict()

```
x_input = np.array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)
```

你觀察到了什麼?

```
[[205.97539]]
```

講次內容

- 什麼是時間序列預測問題
- 單變量 CNN 模型
- 多變量 CNN 模型
- 多時階 CNN 模型

多時階 CNN 模型

- 就是單一來源數據的時間序例
- 應用於預測「下幾步」

多時階 CNN 模型

- 資料準備：切出（題目，答案）配對
- 想想看處理邏輯應該是什麼?

10, 20, 30, 40, 50, 60, 70, 80, 90
15, 25, 35, 45, 55, 65, 75, 85, 95

=>

X,	y
10, 20, 30	40, 50
20, 30, 40	50, 60
30, 40, 50	60, 70
40, 50, 60	70, 80
50, 60, 70	80, 90

- 資料
- 想想

10, 20,
15, 25,

```python
# 時間序列資料，串列前後元素有時序的先後順序
list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90]

# 我們希望資料變成一段一段小的時序資料
#  X,  ->  ->  ->  y
# 10, 20, 30  ->  40, 50
# 20, 30, 40  ->  50, 60
# 30, 40, 50  ->  60, 70
# ...

def splitSequence(seq, steps_in, steps_out):
    x, y = [], []
    for i in range(len(seq)):
        # 找到本次取樣的尾端索引
        end_ix = i + steps_in
        out_end_ix = end_ix + steps_out
        # 檢查是否該結束了
        if out_end_ix > len(seq):
            break
        # 切出兩部份!
        seq_x, seq_y = seq[i:end_ix], seq[end_ix:out_end_ix]
        x.append(seq_x)
        y.append(seq_y)
    return x, y

n_steps_in = 3
n_steps_out = 2
listX, listY = splitSequence(list1, n_steps_in, n_steps_out)

for x, y in zip(listX, listY):
    print(x, y)
```

（題目，答案）

y　　配對

```
[10, 20, 30] [40, 50]
[20, 30, 40] [50, 60]
[30, 40, 50] [60, 70]
[40, 50, 60] [70, 80]
[50, 60, 70] [80, 90]
```

多時階 CNN 模型

- 特徵處理
  - 使用 Keras 來製作 CNN 模型
  - 針對 Keras API Conv1D 所需調整資料維度

```
arrayX = np.array(listX)
arrayY = np.array(listY)
print(arrayX.shape)
print(arrayY.shape)

n_features = arrayX.shape[2]
```

```
(7, 3, 2)
(7,)
```

多時階 CNN 模型

- 預測模式
  - 「下幾步」預測， multi-step prediction
  - 使用 Keras API 的 1D CNN 模型

多時階 CNN 模型

- 預測模式

注意
input_shape!

```
from tensorflow.keras.layers import Activation, Conv1D, MaxPooling1D, Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(n_steps_out))

print(model.summary())
model.compile(optimizer=Adam(lr=0.001), loss='mse')
```

多時階 CNN 模型

- 預測模式

```
from tensorflow.keras.lay
from tensorflow.keras.mod
from tensorflow.keras.opt

model = Sequential()

model.add(Conv1D(filters=
model.add(MaxPooling1D(po
model.add(Flatten())
model.add(Dense(50, activ
model.add(Dense(n_steps_

print(model.summary())
model.compile(optimizer=A
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_6 (Conv1D)            (None, 2, 64)             192
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 1, 64)             0
_________________________________________________________________
flatten_6 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_12 (Dense)             (None, 50)                3250
_________________________________________________________________
dense_13 (Dense)             (None, 2)                 102
=================================================================
Total params: 3,544
Trainable params: 3,544
Non-trainable params: 0
_________________________________________________________________
None
```

多時階 CNN 模型

- 訓練 train 資料集
  - model.fit() 又出現了!

```
import matplotlib.pyplot as plt

train_history = model.fit(arrayX, arrayY, epochs=1500)

plt.plot(train_history.history['loss'], label='loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

多時階 CNN 模型

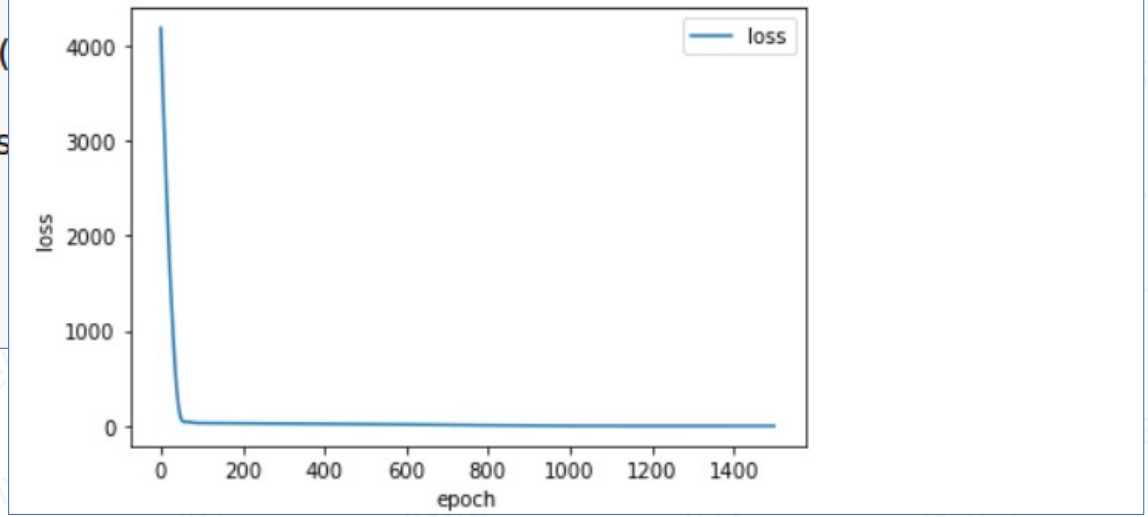
- 訓練 train 資料集
  - model.fit()

```
import matplotlib.pyplot a

train_history = model.fit(

plt.plot(train_history.his
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='best')
plt.show()
```

```
Epoch 1498/1500
1/1 [==============================] - 0s 6ms/step - loss: 0.0056
Epoch 1499/1500
1/1 [==============================] - 0s 7ms/step - loss: 0.0056
Epoch 1500/1500
1/1 [==============================] - 0s 5ms/step - loss: 0.0055
```



多時階 CNN 模型

- 自製 test 資料做測試
  - 使用 model.predict()

```
x_input = np.array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)

x_input = np.array([7, 8, 9])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)

x_input = np.array([75, 85, 95])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input)
print(yhat)
```

你觀察到了什麼?

```
[[102.859566 114.774254]]
[[13.286224 16.519444]]
[[108.54503 120.97099]]
```

CNN 時間序列預測討論

- 你知道輸入端資料製作的意義嗎?
- 使用 Conv1D 的意義在於?
- 單變量、多變量、多時階的差異
- 有沒有看出模型預測的特性?
- 什麼? 還有更多?！

這個講次中，你應該學到了 ...

- 時間序列預測的意義
- 如何使用 CNN 進行單變量時間序列預測
- 如何使用 CNN 進行多變量時間序列預測
- 如何使用 CNN 進行多時階時間序列預測