# 自然語言處理
# RNN&LSTM

Instructor: 馬豪尚

# 深度神經網路(DNN)的架構

› 輸入層接收資料後，傳給隱藏層的每個神經元進行運算，最後由輸出層輸出運算完的結果

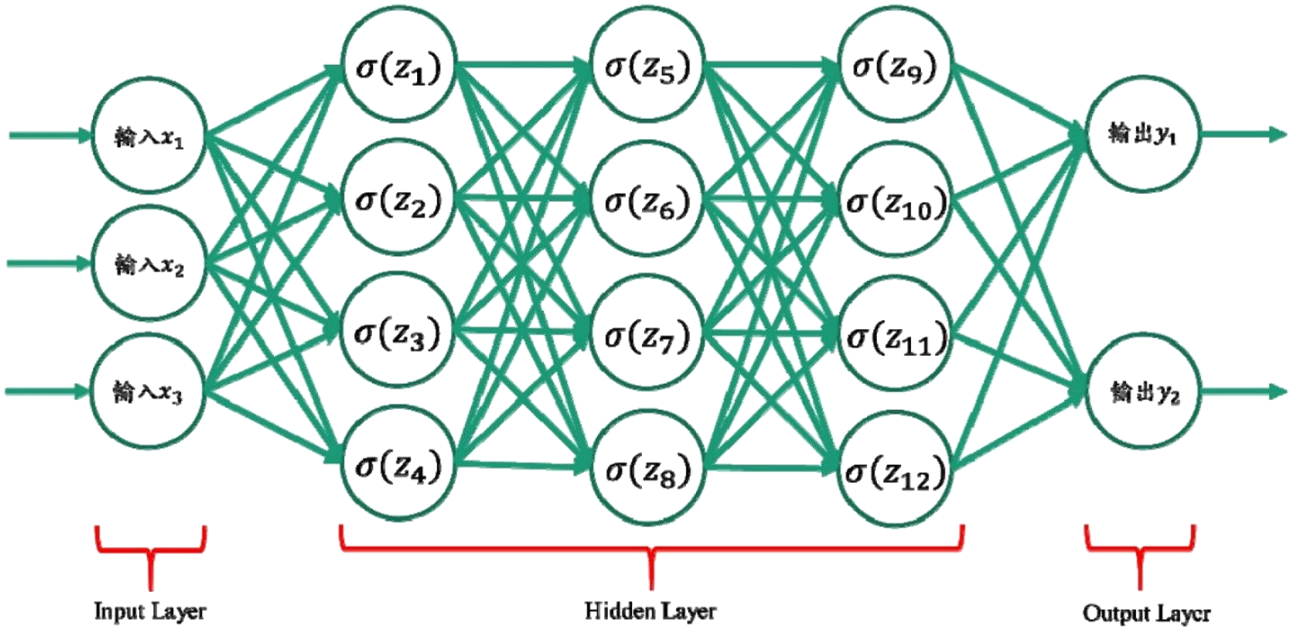



圖 4-13　神經網路架構

# Keras和Tensorflow

- Sequential API
  - 宣告一個模型以Sequential的方式來搭建
  - Sequential為一層一層有序的加入神經網路
  - 在每一層使用 Dense Layer並設定神經元個數以及Activation

```python
#  匯入必要的庫
import  keras
from  keras.models  import  Sequential
from  keras.layers  import  Dense

#  創建一個Sequential模型
model  =  Sequential()

#  添加輸入層（假設輸入特徵有10個）
model.add(Dense(units=64,  input_dim=10,  activation='relu'))

#  添加第一個隱藏層
model.add(Dense(units=32,  activation='relu'))

#  添加第二個隱藏層
model.add(Dense(units=16,  activation='relu'))

#  添加輸出層（假設有2個輸出類別）
model.add(Dense(units=2,  activation='softmax'))

#  編譯模型，指定損失函數、優化器和評估指標
model.compile(loss='categorical_crossentropy',  optimizer='adam',  metrics=['accuracy'])

#  打印模型結構
model.summary()
```

# Keras

- Model Functional API
  - 利用model函式更靈活的建構神經網路架構
  - 自定義每一層的神經元數和activation函數
  - 自訂每層的輸入為某一層的輸出

# Keras Model Functional API

```
from  tensorflow.keras  import  layers
from  tensorflow.keras.models  import  Model

#自訂義模型
def  build_model():
    #模型輸入層(層參數)
    model_input  =  layers.Input(shape=64)
    #第一層隱藏層(層參數)(輸入)
    x  =  layers.Dense(16,activation='relu')(model_input)
    #第二層隱藏層(層參數)(輸入)
    x  =  layers.Dense(16,activation='relu')(x)
    #輸出層(層參數)(輸入)
    model_output  =  layers.Dense(3,activation='softmax')(x)
    #使用Model函式建立模型(輸入,  輸出)
    return  Model(model_input  ,model_output)
model  =  build_model()
model.summary()
```

# Keras 建立模型二元分類實例

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

# 創建一個小的玩具資料集
data = np.random.rand(1000, 5)    # 特徵
labels = (data.sum(axis=1) > 2.5).astype(int)    # 標籤
# 將資料切成訓練和測試
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
# 建立一個簡單的DNN模型
model = Sequential()
model.add(Dense(32, input_dim=5, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# 編譯模型
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# 訓練模型
model.fit(X_train, y_train, epochs=50, batch_size=64, verbose=1)
# 評估模型
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
# 使用模型來預測
predictions = model.predict(X_test)
# 印出前10個結果
for i in range(10):
        print(f"Sample {i+1}: Predicted={predictions[i][0]:.4f}, Actual={y_test[i]}")
```

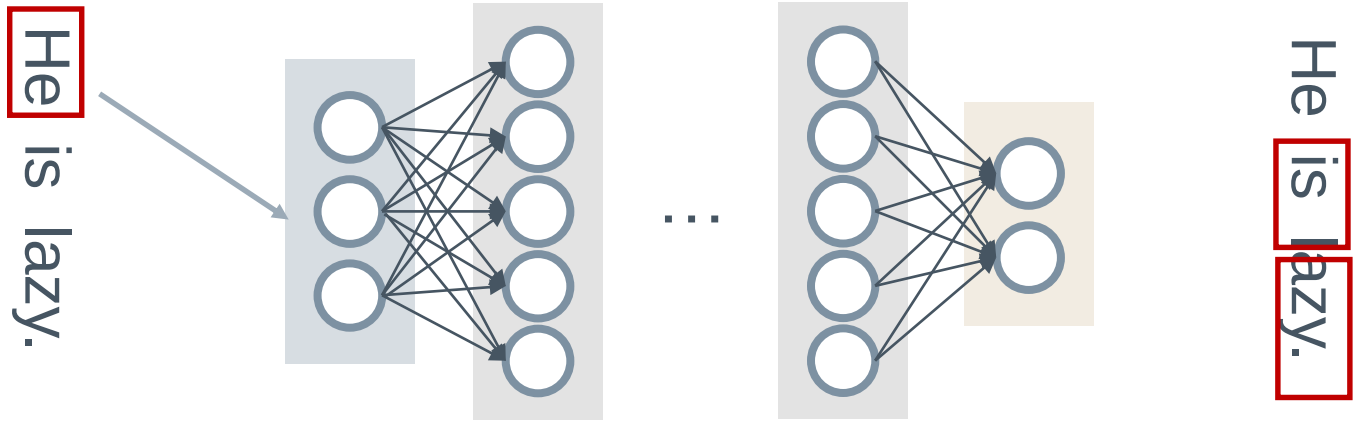
# 練習

- 使用Keras建立一個深度學習的多元分類模型
- 使用Sklearn的wine資料集當作輸入
  - 切割成訓練和測試資料集
- 訓練一個wine的分類模型並測試其效果
  - 顯示分類準確度

# Recurrent Neural Networks

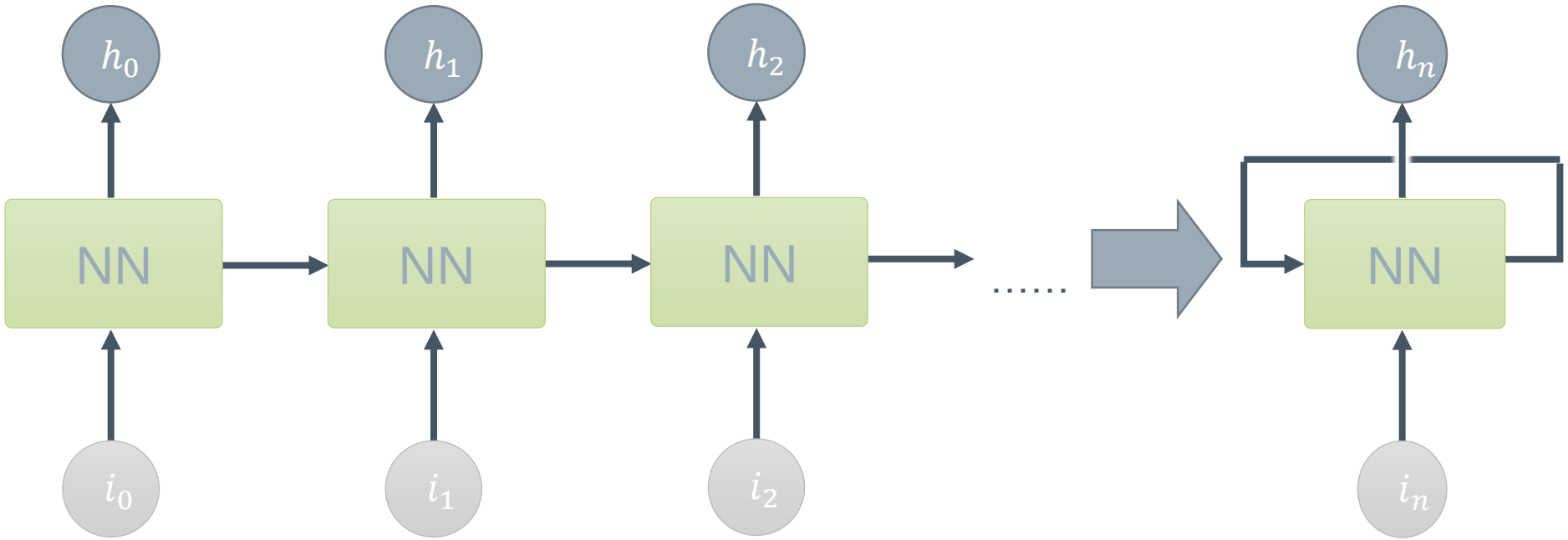
# 語言學習模型

- 人類是如何了解一個自然語言中的句子?
  - 每當下個詞彙映入眼中，你腦中的處理都會跟以下兩者相關：
    - 前面所有已讀的詞彙
    - 目前腦中的記憶狀態



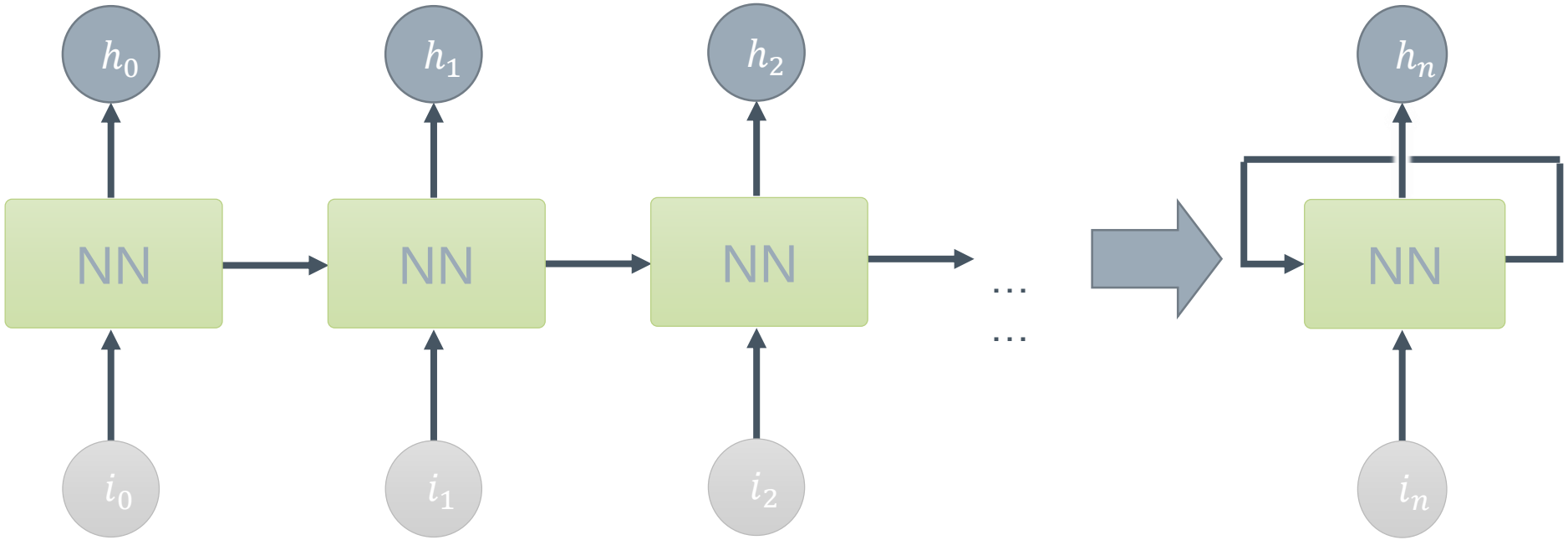
# Recurrent Neural Networks

有記憶的循環神經網路



# Recurrent Neural Networks

› 想像有一個輸入序列 $I = [i_0, i_1 \cdots i_n]$



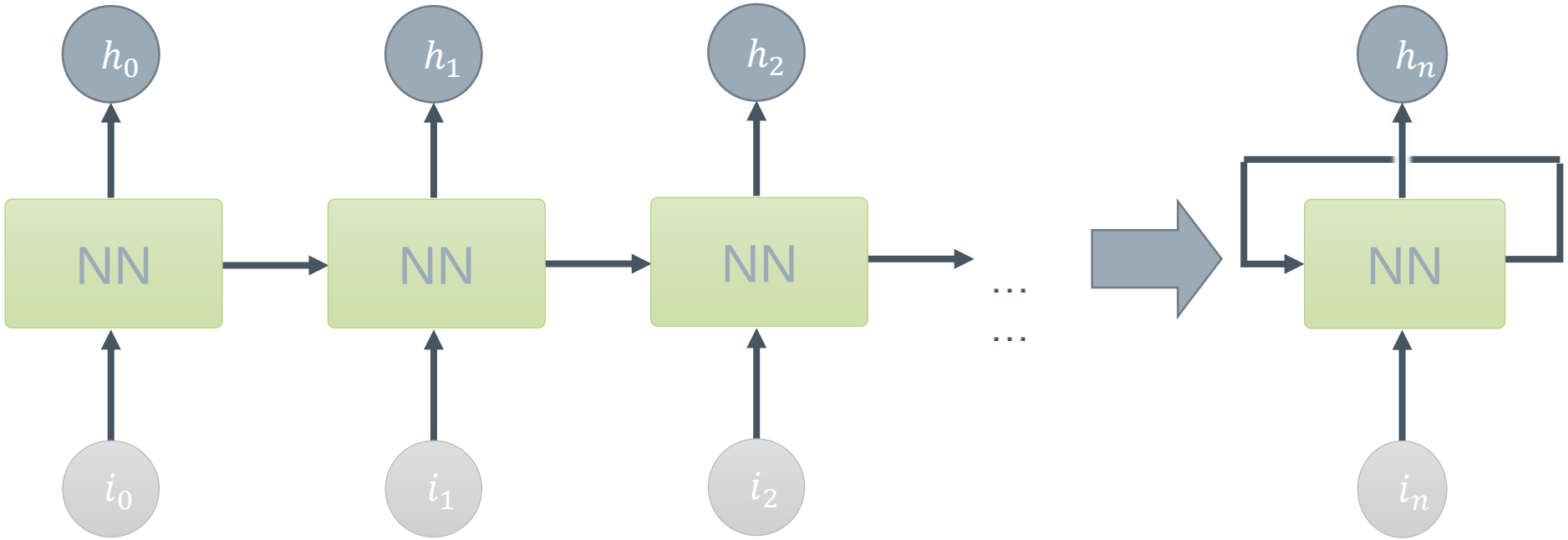

› RNN 在第一個時間點 $t_0$並不會直接把整個序列 I 讀入

› 在第一個時間點$t_0$ ，它只將該序列中的第一個元素 $i_0$ 輸入一個類神經網路中

› 類神經網路則會針對 $i_0$做些處理以後，更新自己的「狀態」並輸出第一個結果 $h_0$

# Recurrent Neural Networks

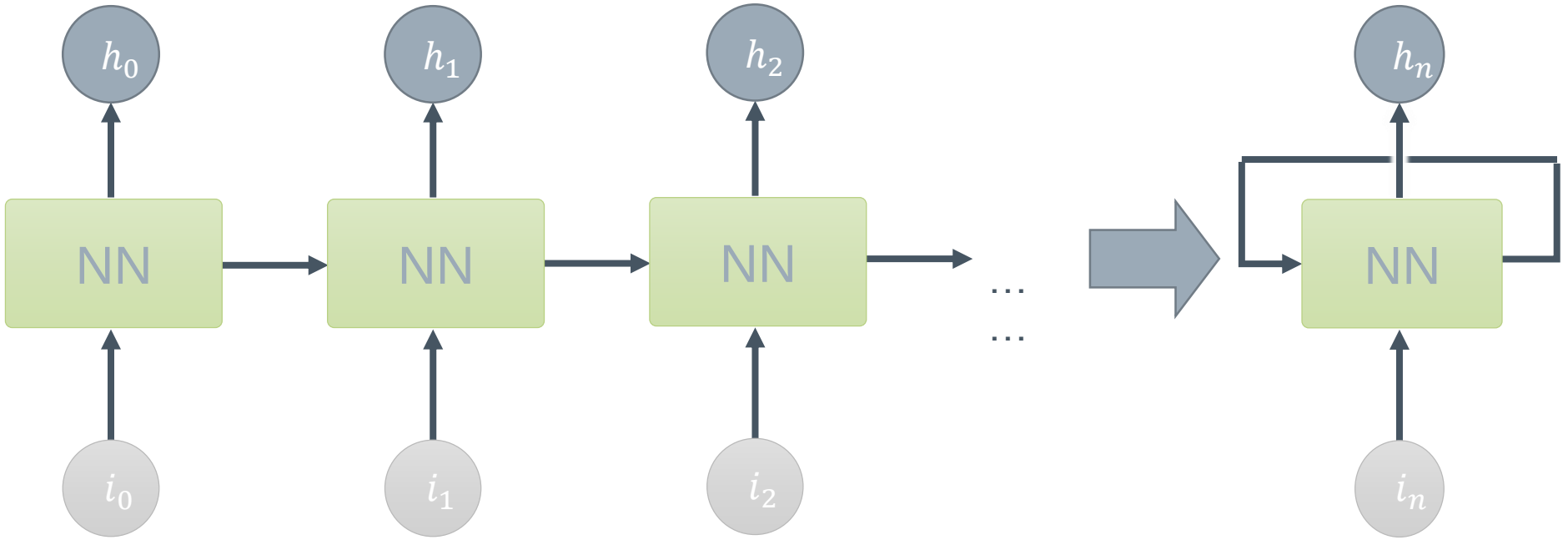
› 下個時間點 $t_1$ ，RNN 如法炮製



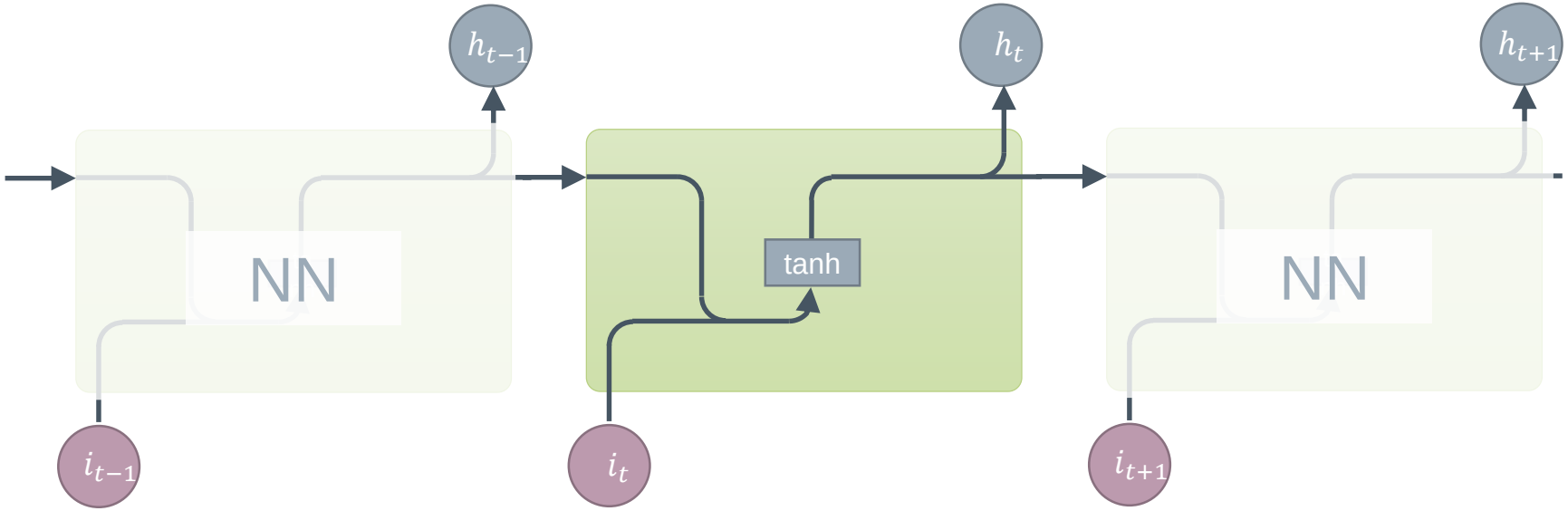

› 讀入序列 I中的下一個元素$i_1$ ，並利用剛剛處理完 $i_0$ 得到的網路狀態，處理 $i_1$ 並更新自己的狀態（也被稱為記憶），接著輸出另個結果 $h_1$

# Recurrent Neural Networks

› 將 RNN 以左邊的形式表示的話，你可以很清楚地了解，當輸入序列越長，向右展開的 RNN 也就越長。（模型也就需要訓練更久時間）



# RNN的內部結構

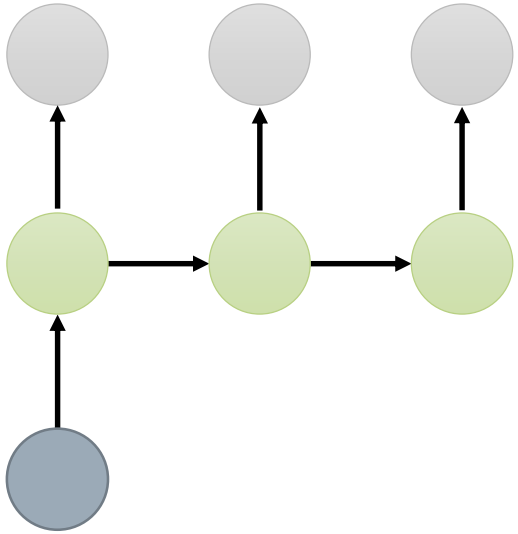


# 不同類型的RNN

- 以輸入和輸出的數量來區分
  - 一對多
  - 多對一
  - 多對多
    - 同步數量
    - 不同步數量

# 一個輸入對多個輸出

- 每個時間點輸入的資料都是一樣的
- 多個輸出
  - 輸出的序列是有順序關係的

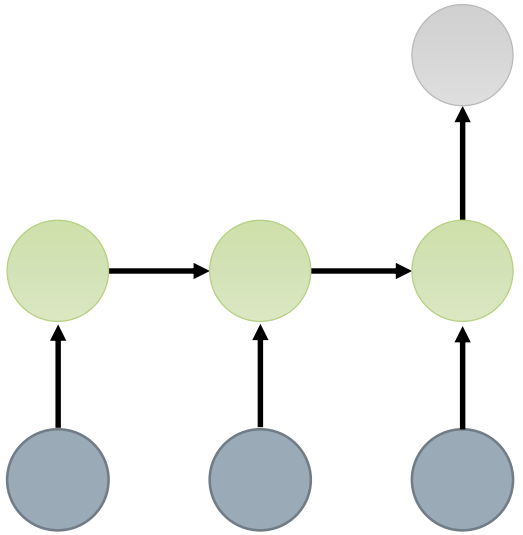


- E.g.) 圖片的自然語言描述

# 多個輸入對一個輸出

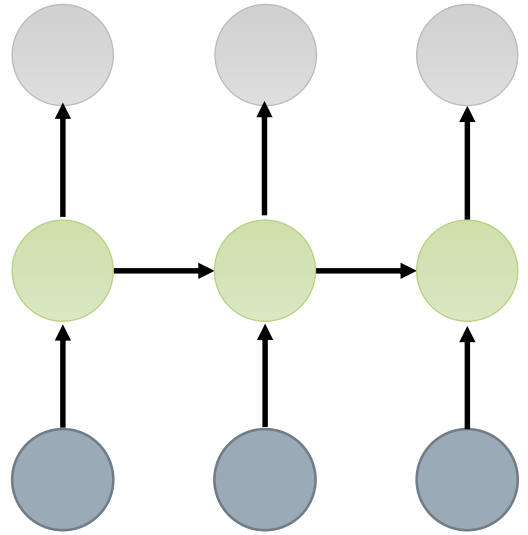
- 多個輸入
  - 輸入的資料是有順序關係
- 一個輸出

- E.g.) 情感分析

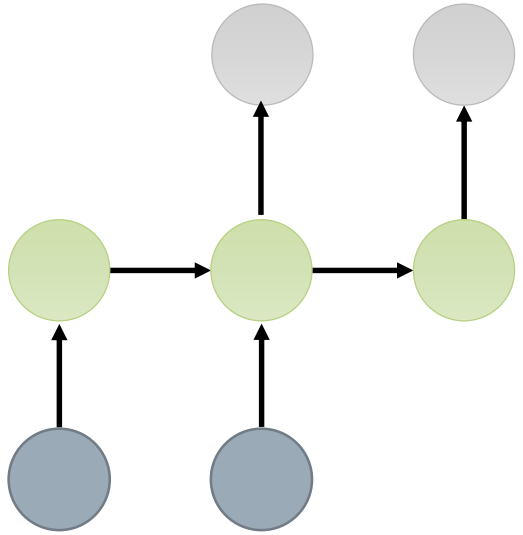


# 多個輸入對多個輸出

- 輸入和輸出數量同步
  - 輸入和輸出的資料是有順序關係
  - 且一個輸入對應到一個輸出
  - E.g.) 預測下一個字、詞性標註



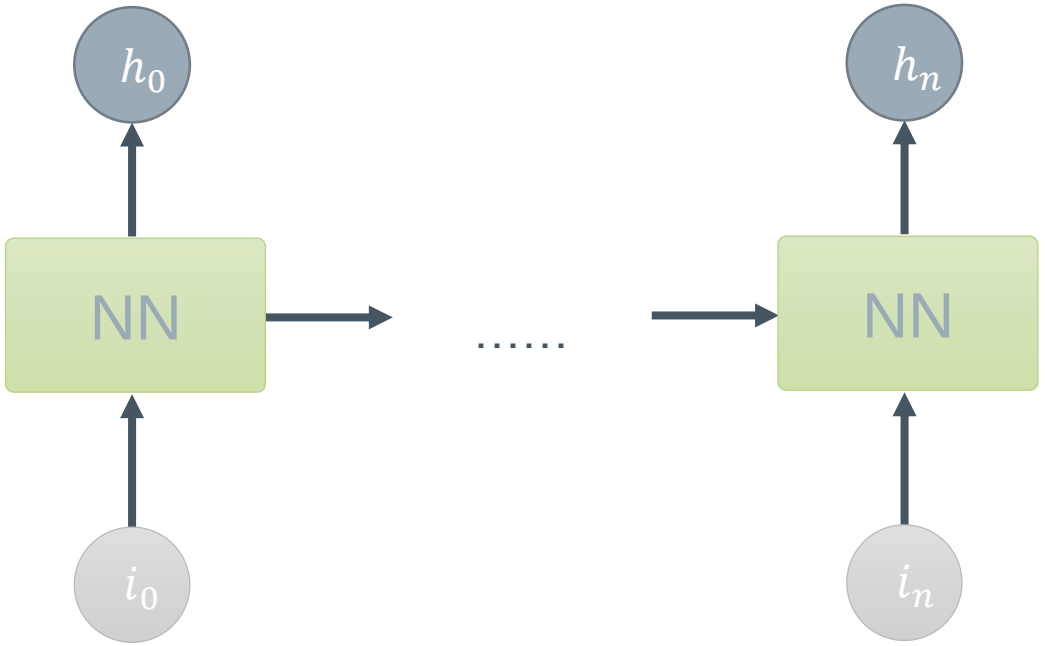
- 輸入和輸出數量不同
  - 輸入和輸出的資料是有順序關係
  - 輸入的長度可以和輸出的長度不同
  - E.g.) 機器翻譯



# Long Short Term Memory

# Long Short Term Memory

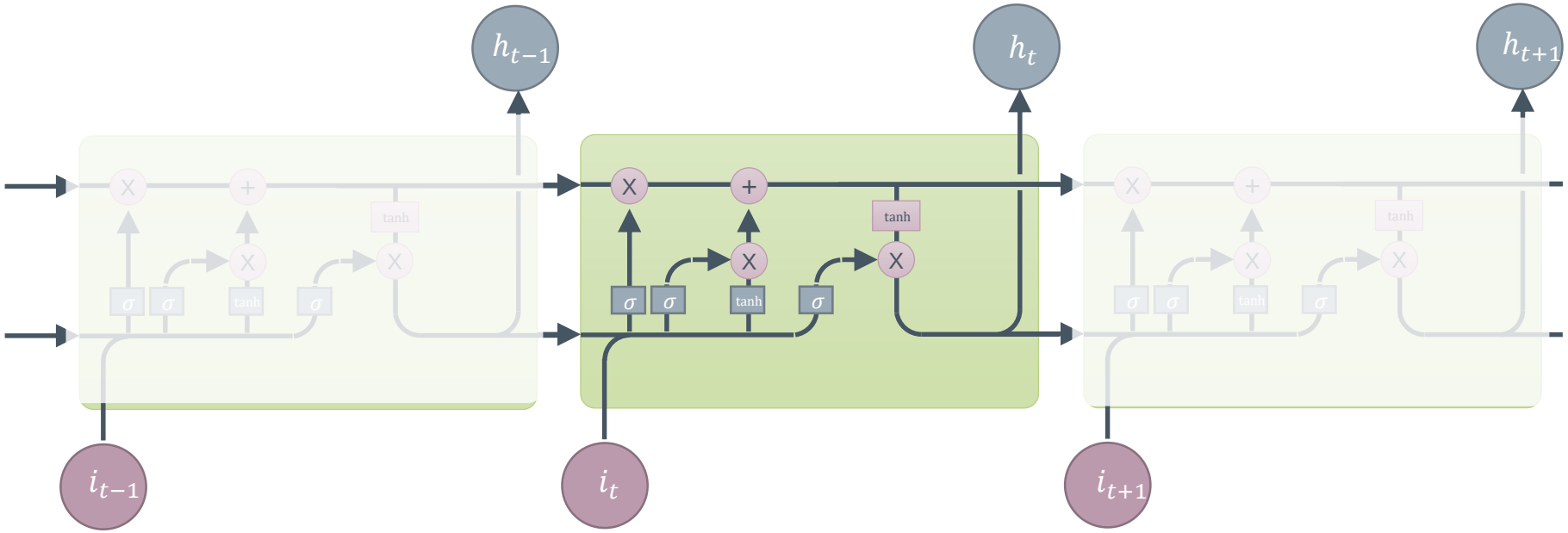
- RNN的問題?
  - 雖然可利用先前的資訊，但是只會記得前一個狀態的資訊
  - 經過多次的遞迴後，較早的狀態已經被遺忘



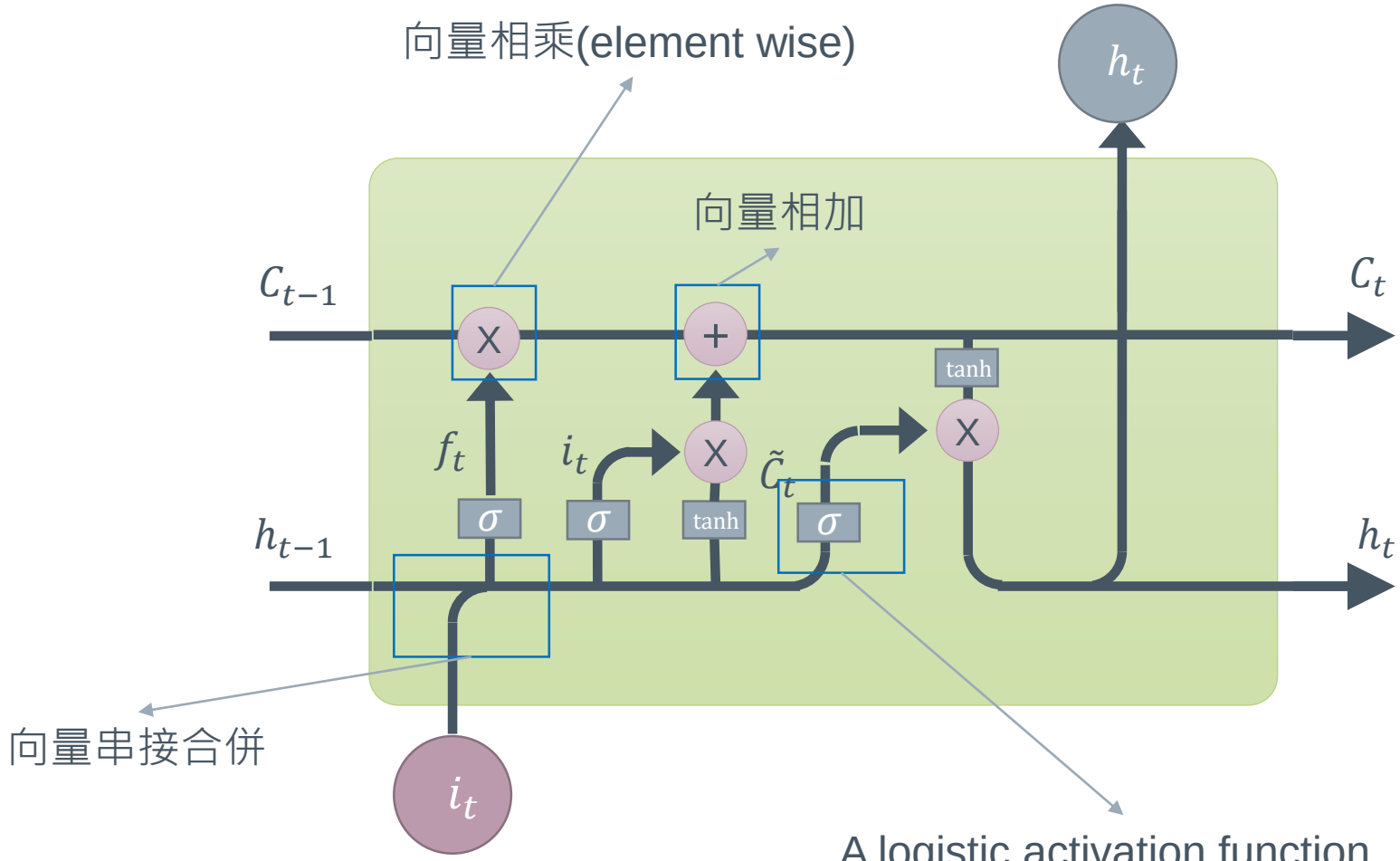
# 記憶力比較好的LSTM

- 短期記憶
  - 記憶最近一個狀態的資訊
  - 就像一個一般的RNN
- 長期記憶
  - 記憶較長時間以前的狀態資訊
  - 透過遺忘機制來決定是否被記住

# The Structure of LSTM

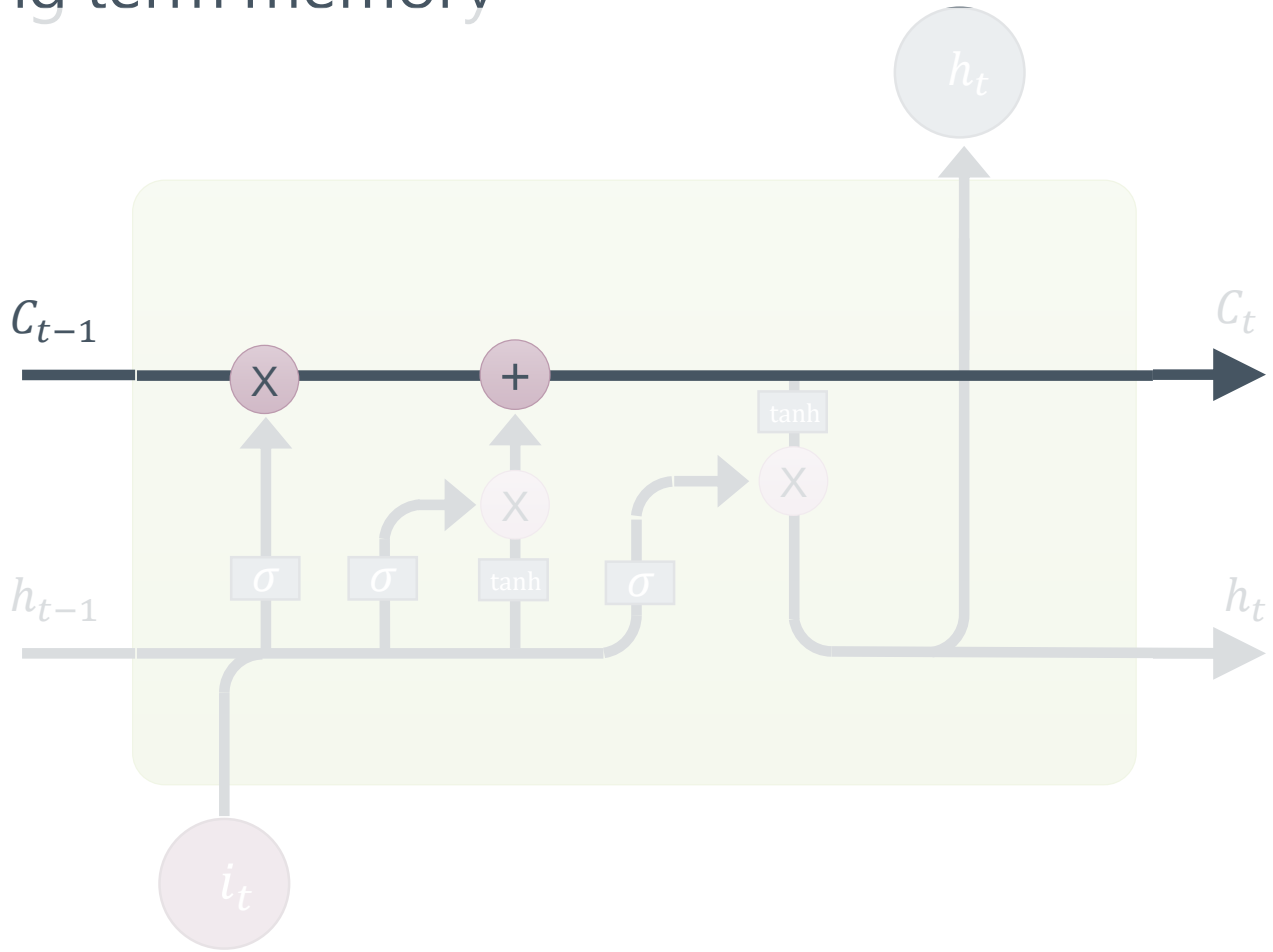


# LSTM內部架構和單元



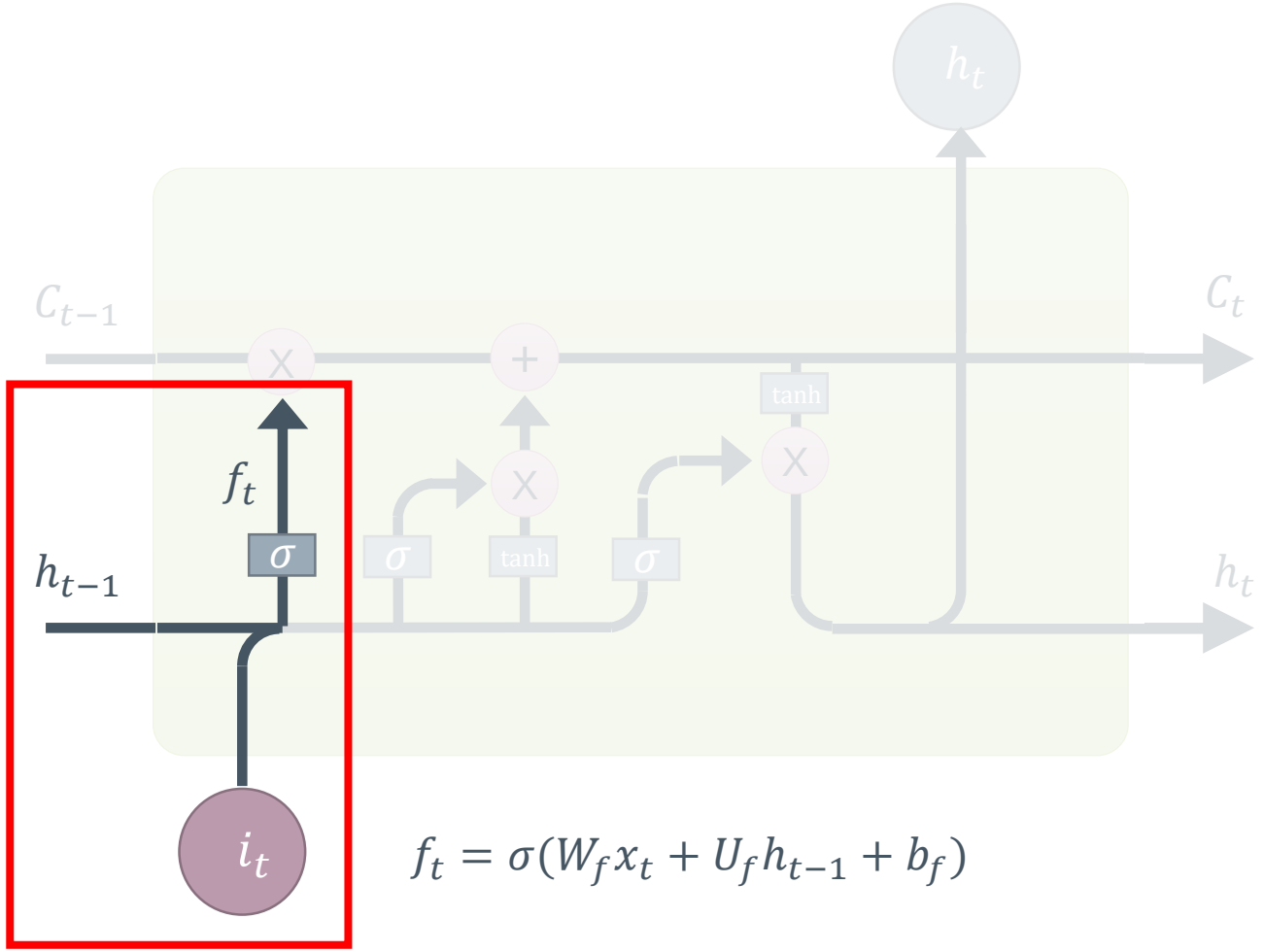
# The Memory in LSTM

- Record the previous outputs
  - Long term memory



# Forget Gate

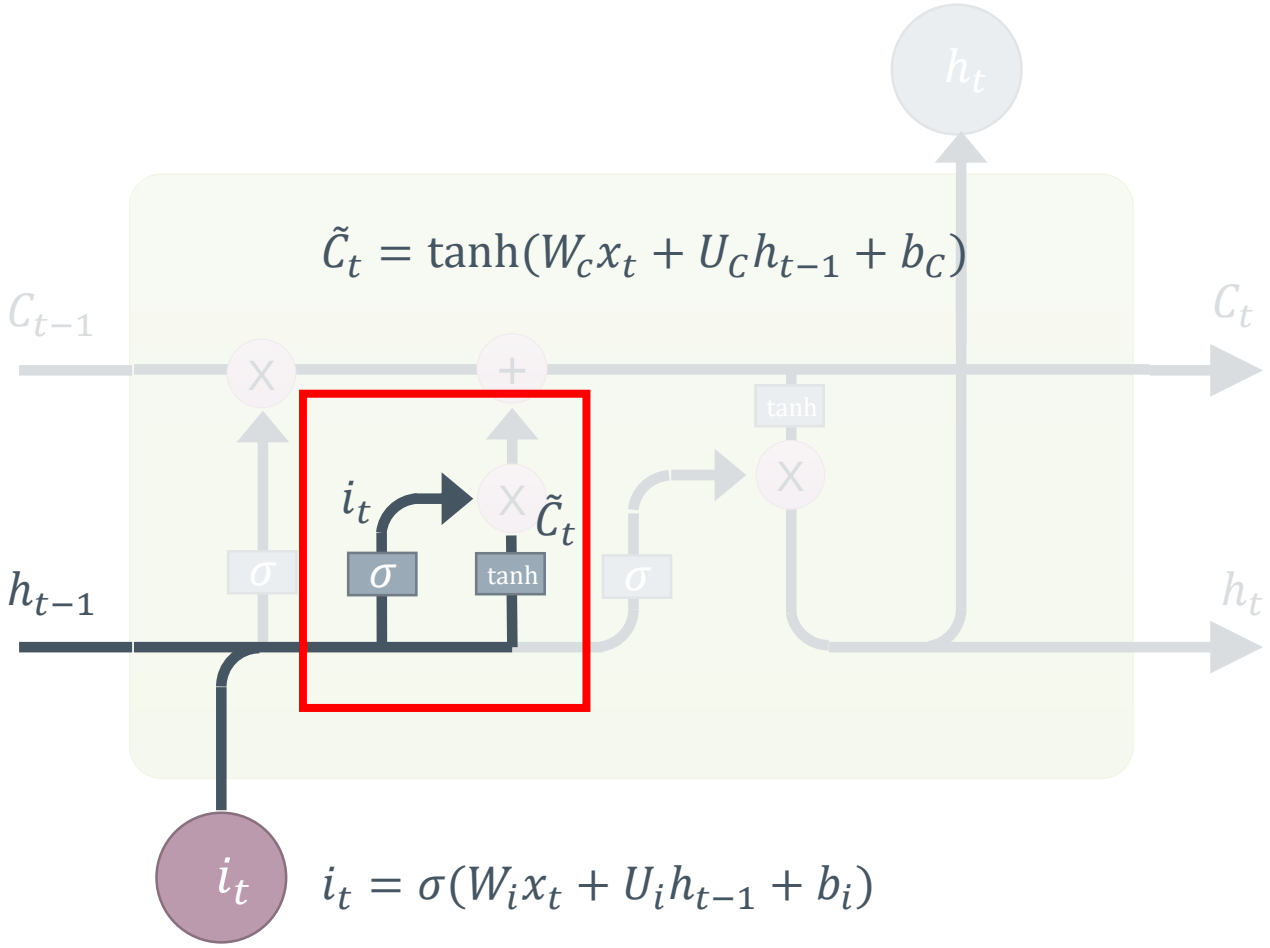
› 遺忘機制: 決定是否要遺忘目前的記憶狀態




$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

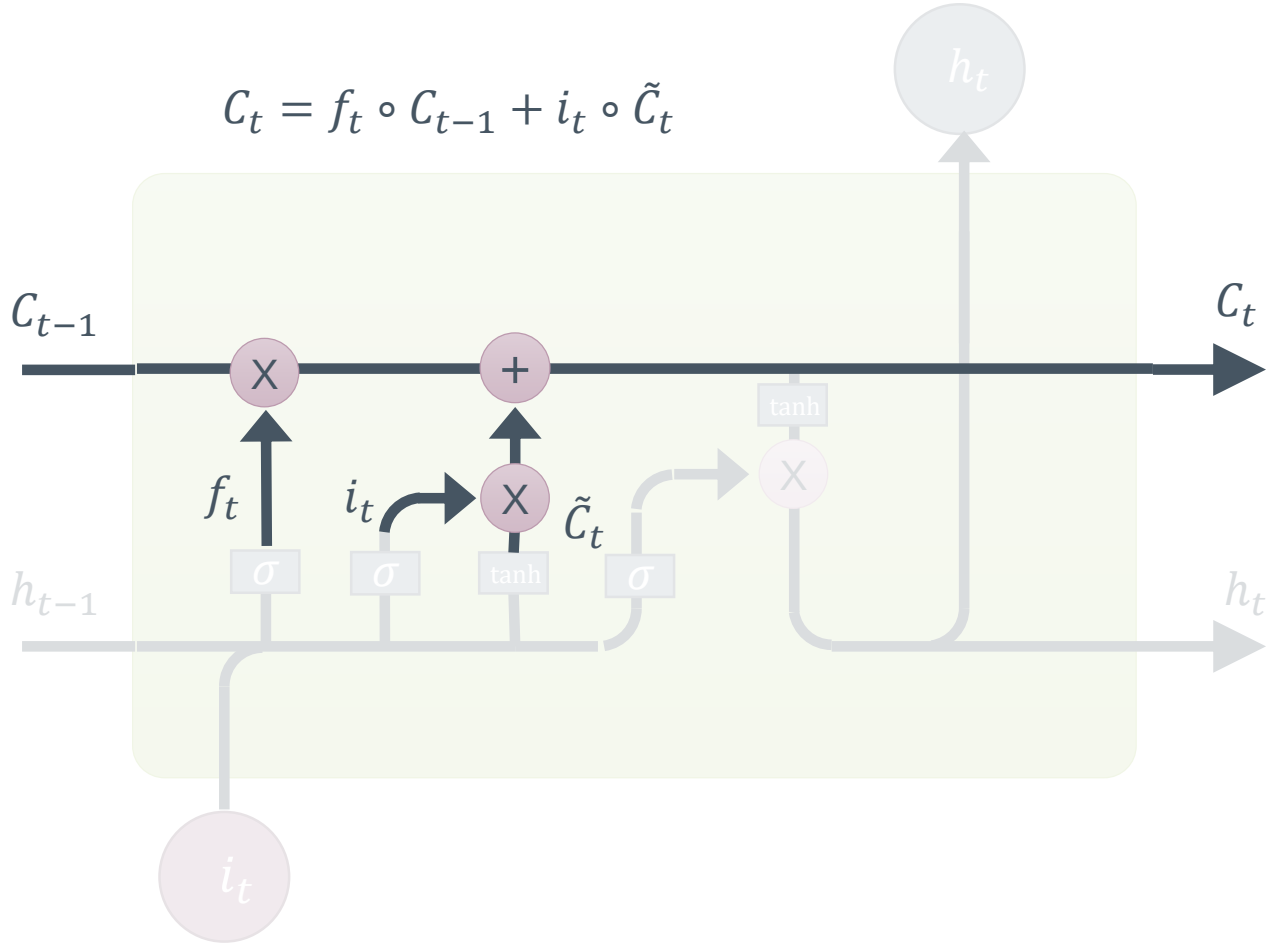
# Input Gate

› 決定輸入值是否重要需要被處理以及有多少資訊是需要被記住



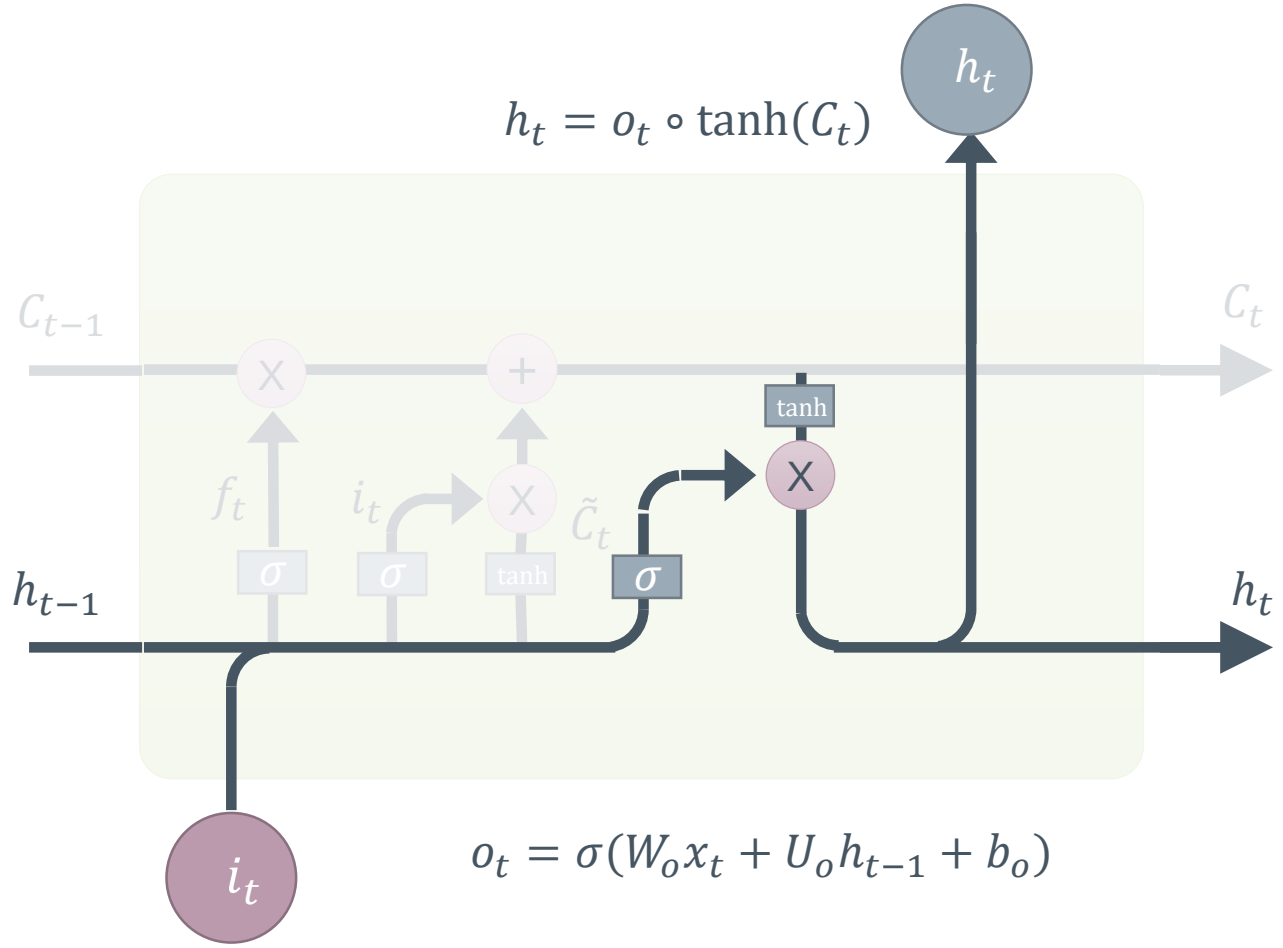
# 更新長期記憶

- 經由模型判定近期狀態和輸入值是否需要被記憶來更新長期記憶



# Output Gate

› 決定更新後的記憶狀態有多少要輸出



# LSTM的實作

```
import keras
from keras.models import Sequential
from keras.layers import LSTM, Dense

# 創建一個Sequential模型
model = Sequential()

# 添加LSTM層
model.add(LSTM(units=64, input_shape=(10, 1), activation='tanh', return_sequences=False))

# 添加全連接層（輸出層，假設有2個輸出類別）
model.add(Dense(units=2, activation='softmax'))

# 編譯模型，指定損失函數、優化器和評估指標
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 打印模型結構
model.summary()
```