第 9 週 -機器學習-時間序列模型

大 綱
Outlines

01 時間序列問題

02 Recurrent Neural Network(RNN)

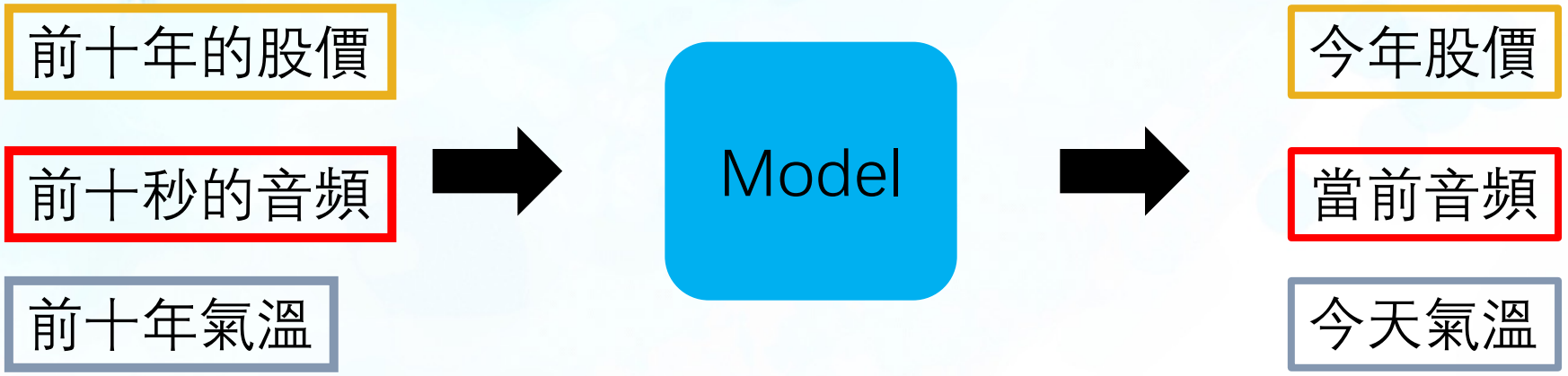
03 Gated recurrent unit(GRU)

04 實作

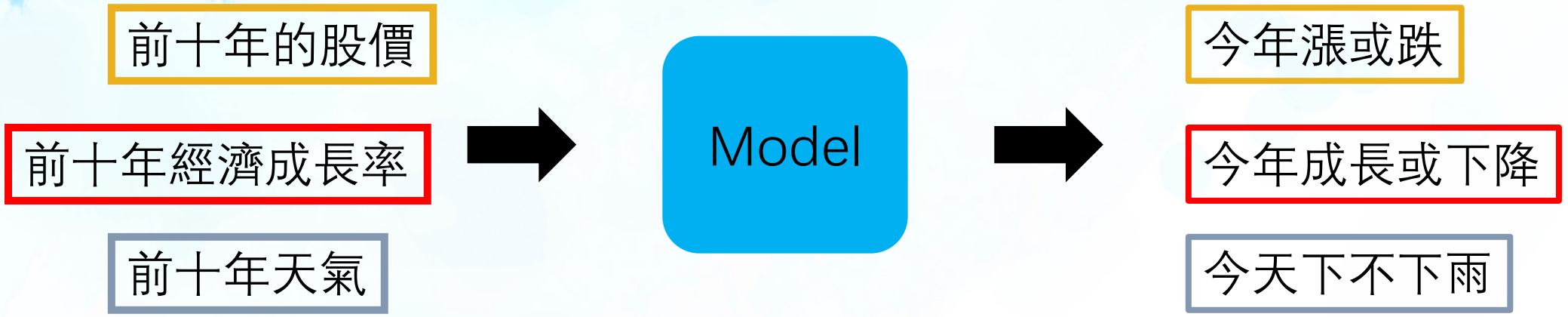
01

時間序列問題

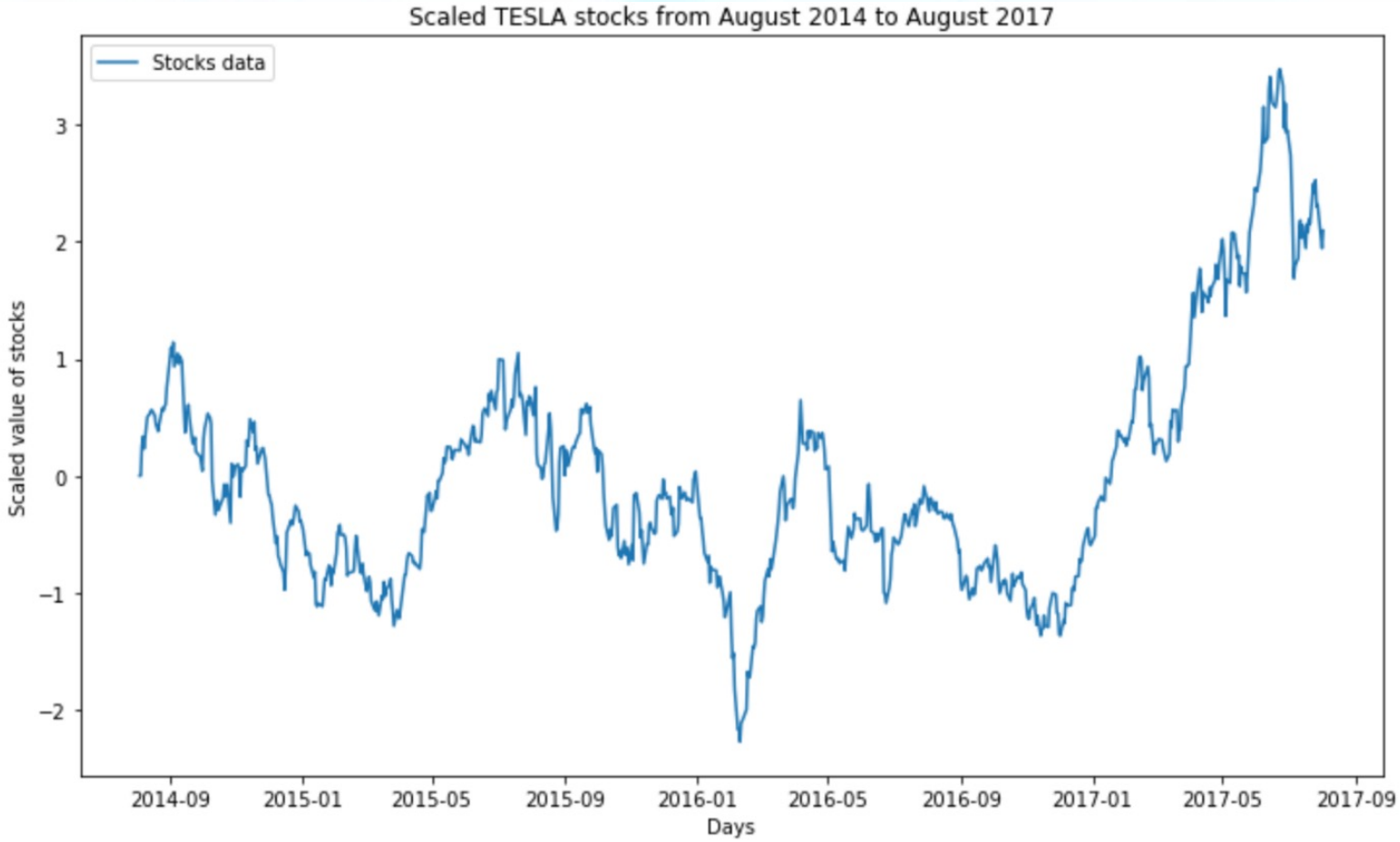
時間序列問題（回歸）



時間序列問題（分類）

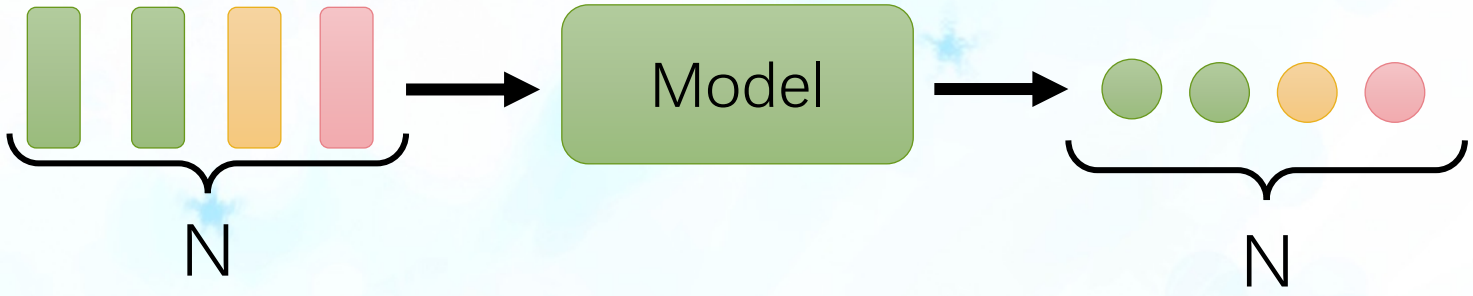


時間序列的樣子

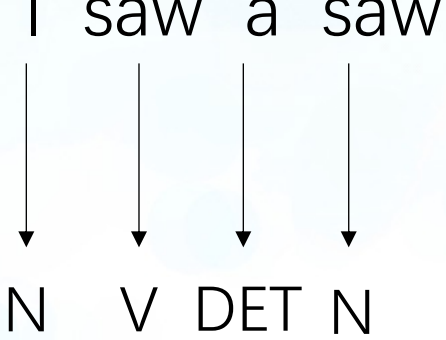

Scaled TESLA stocks from August 2014 to August 2017
Stocks data
Scaled value of stocks
3
2
1
0
−1
−2
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
Days

What is the output?

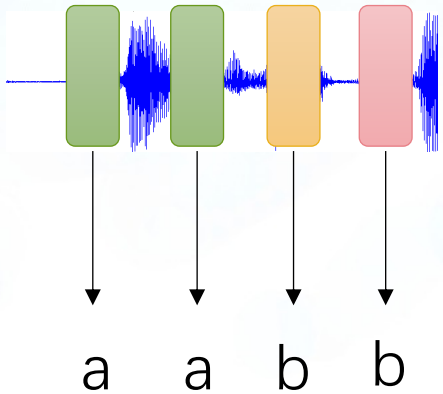
- Each vector has a label.



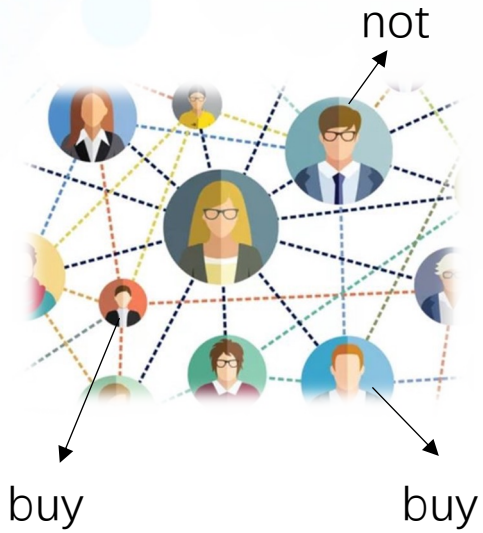
Example Applications



POS tagging

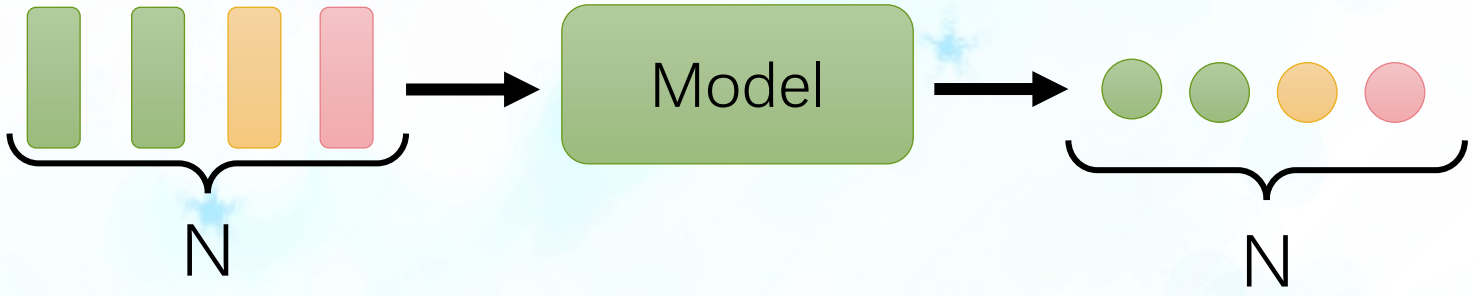


HW2

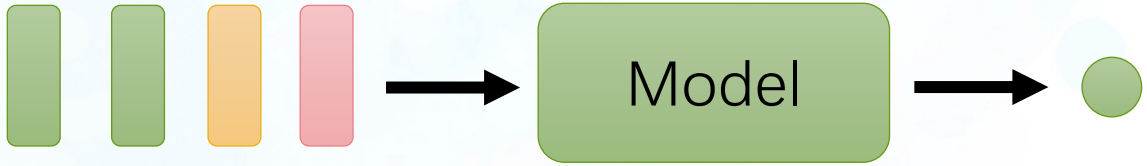


What is the output?

- Each vector has a label.



- The whole sequence has a label.

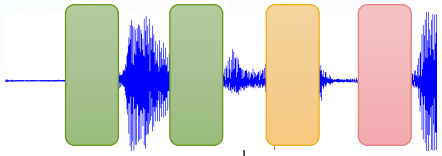


Example Applications

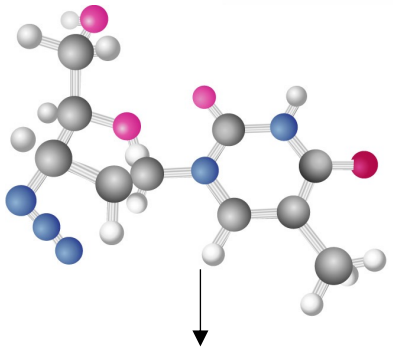
this is good

Sentiment analysis

positive



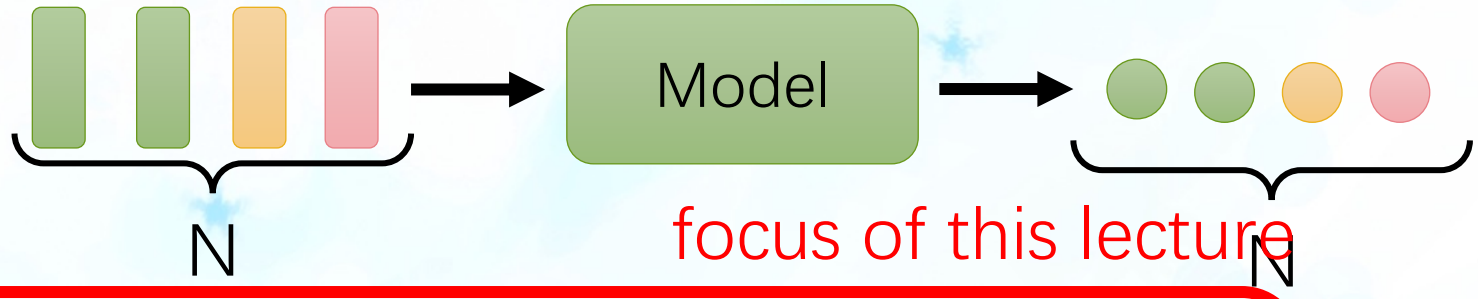
speaker



hydrophilicity

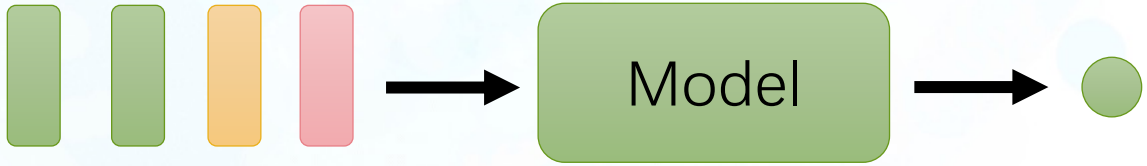
What is the output?

- Each vector has a label.



focus of this lecture

- The whole sequence has a label.

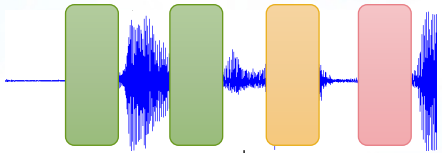


Example Applications

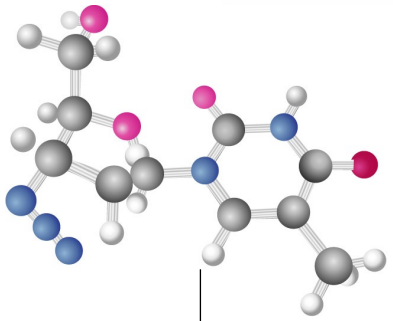
this is good

Sentiment analysis

positive



speaker



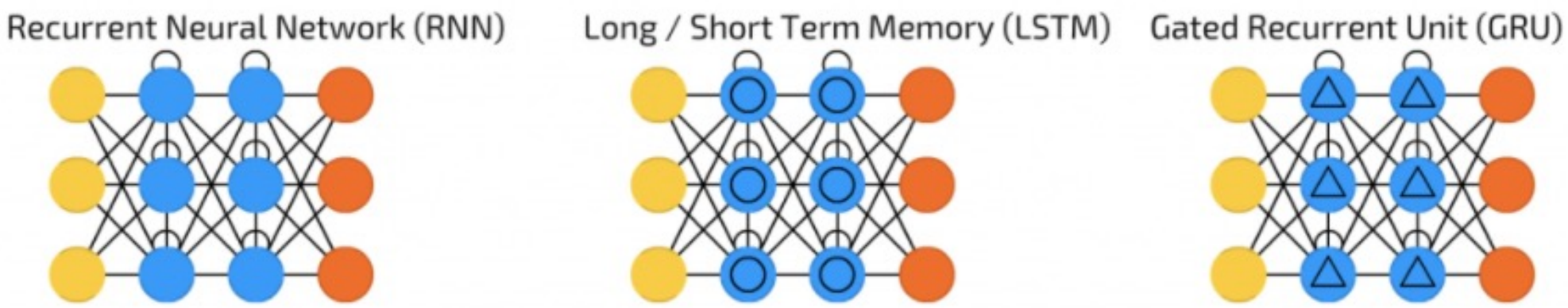
hydrophilicity

02

Recurrent Neural networks(RNN)

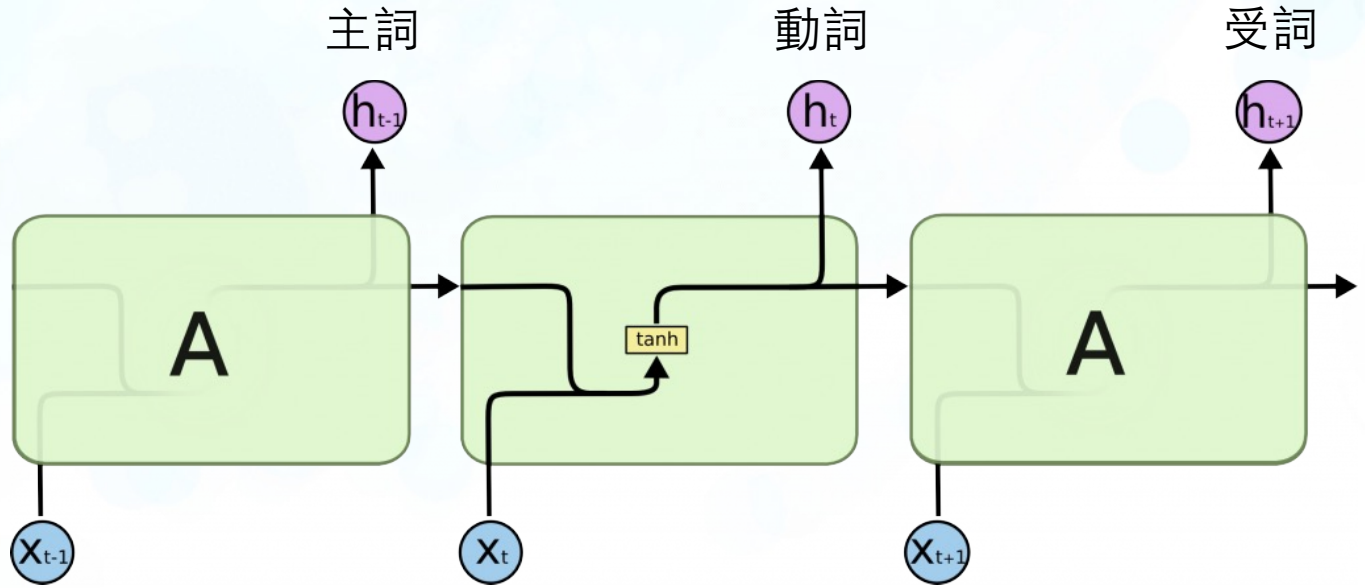
Recurrent Neural Network（RNN）

- Recurrent Neural Network（RNN）是神經網絡的一種，常應用在處理時間、空間序列上有強關聯的訊息，尤其在 NLP（Natural Language Processing，自然語言處理）領域上，RNN 更是參與了重要的一角，有趣的應用如語音識別、翻譯、描述照片、作曲等等。



Recurrent Neural Network（RNN）

- 基於我們在閱讀文章時，是根據對上下文來理解文章意義，RNN的概念在於將狀態在自身網絡中循環傳遞，因此可以接受更廣泛的時間序列結構輸入，允許訊息持續存在。



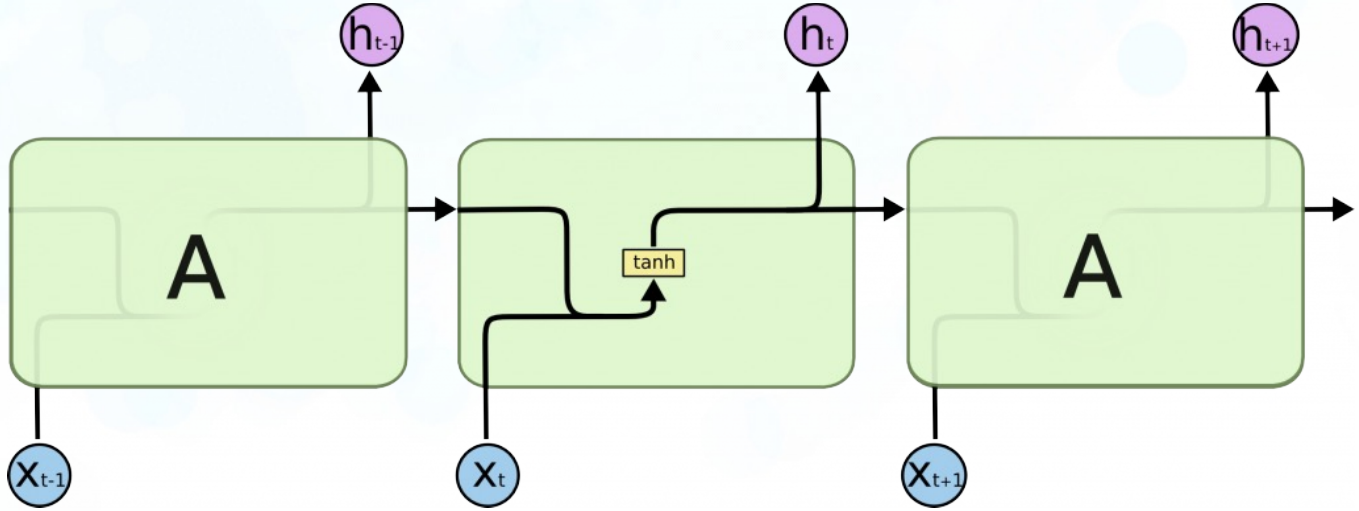
many to many 問題

Recurrent Neural Network（RNN）

- 基於我們在閱讀文章時，是根據對上下文來理解文章意義，RNN的概念在於將狀態在自身網絡中循環傳遞，因此可以接受更廣泛的時間序列結構輸入，允許訊息持續存在。

19度

output：第四天溫度

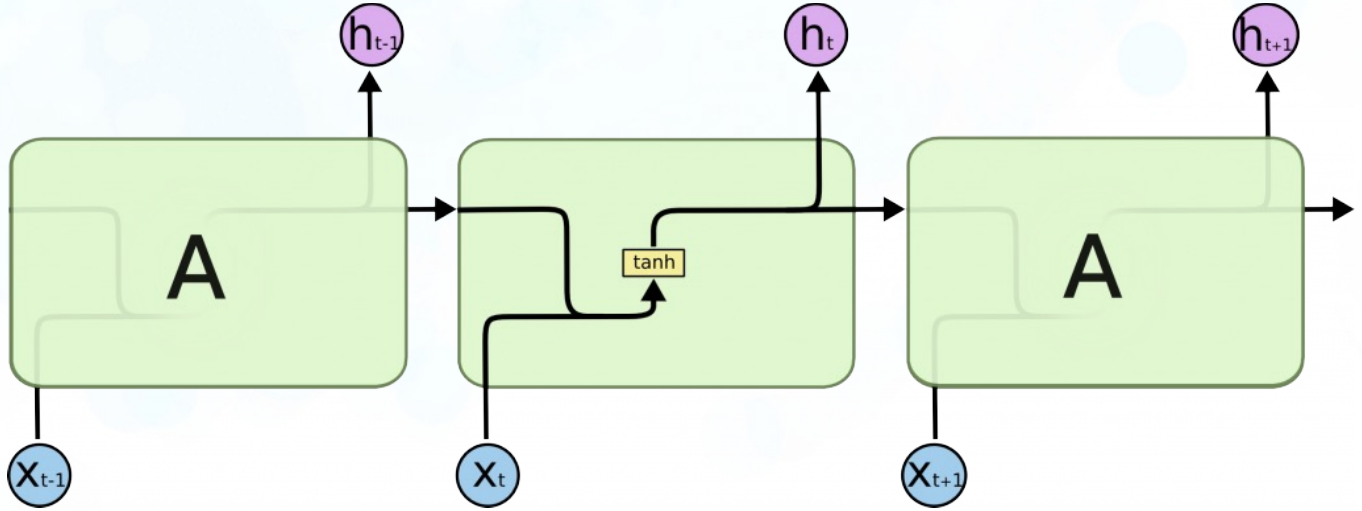


many to one 問題

intput： 前幾天氣溫　16度　17度　18度

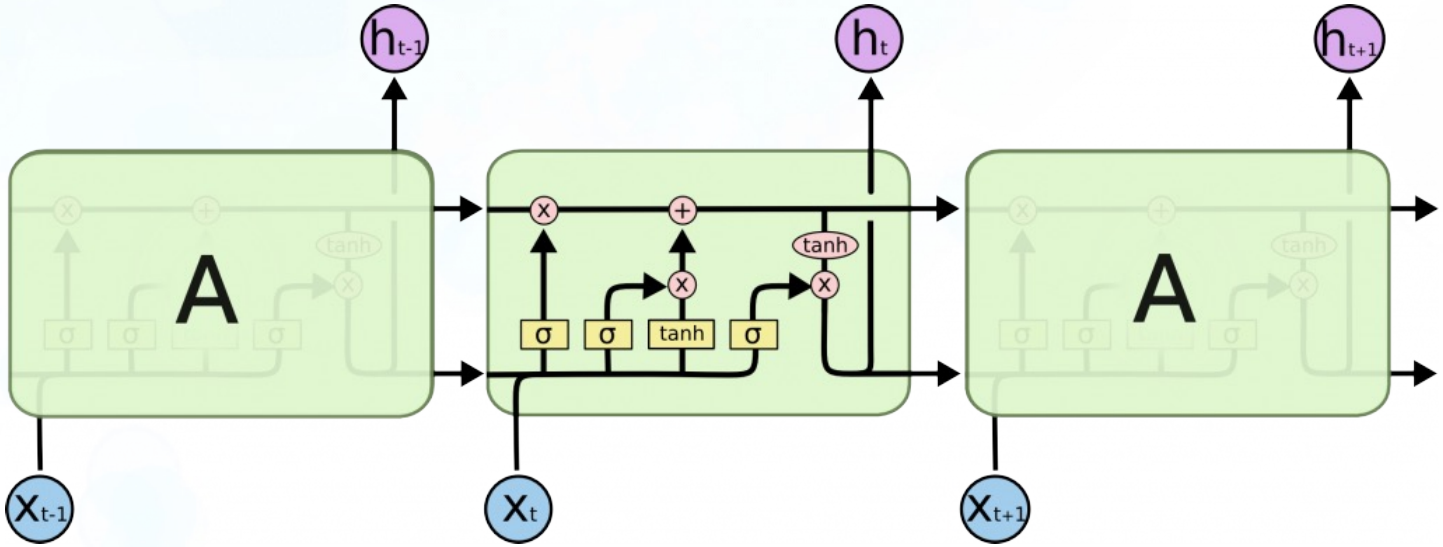
Recurrent Neural Network（RNN）

- 基於我們在閱讀文章時，是根據對上下文來理解文章意義，RNN的概念在於將狀態在自身網絡中循環傳遞，因此可以接受更廣泛的時間序列結構輸入，允許訊息持續存在。



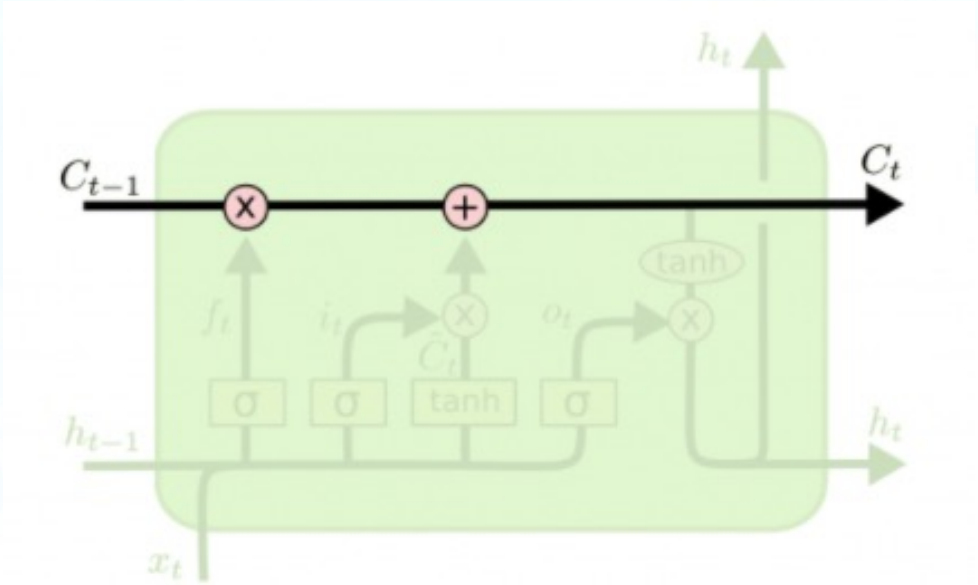
Long/Short Term Memory（LSTM）

- 基於上述 RNN 的限制，可以透過 RNN 的變形，也就是 LSTM 來解決。
- LSTM 的特色是能夠學習長距離的依賴關係（Long-Term Dependencies），它不同於 RNN 有個單一的神經網絡層（tanh），而是有四個層，以特別的方式進行溝通，如圖：



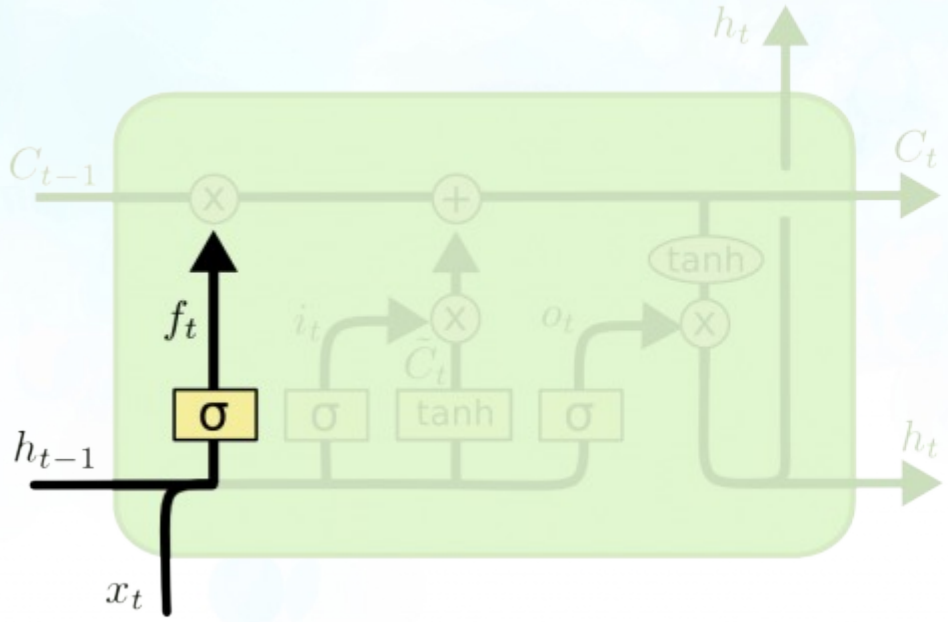
Long/Short Term Memory（LSTM）

- LSTM 核心關鍵在於 Cell State（單元狀態），沿著整個鏈運行，如下圖：



Forget gate layer

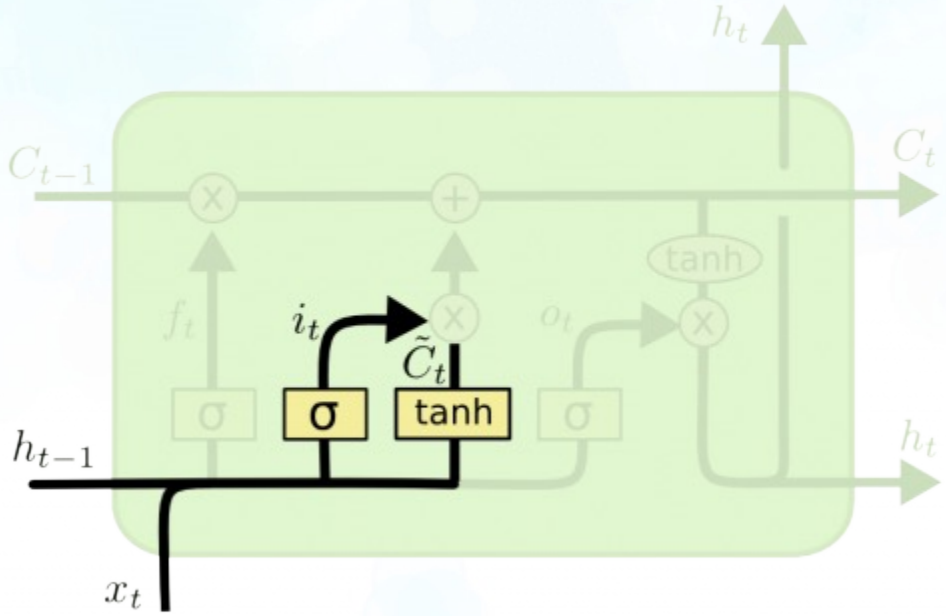
- 第一步：決定要從單元狀態中丟掉什麼訊息，所以稱為遺忘門，由激活函數sigmoid決定。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Input gate layer

- 第二步：決定要在單元狀態中儲存哪些新的訊息，又分成兩部分：
- 激活函數sigmoid決定更新哪些值。
- 激活函數tanh創建新的向量可添至單元狀態。

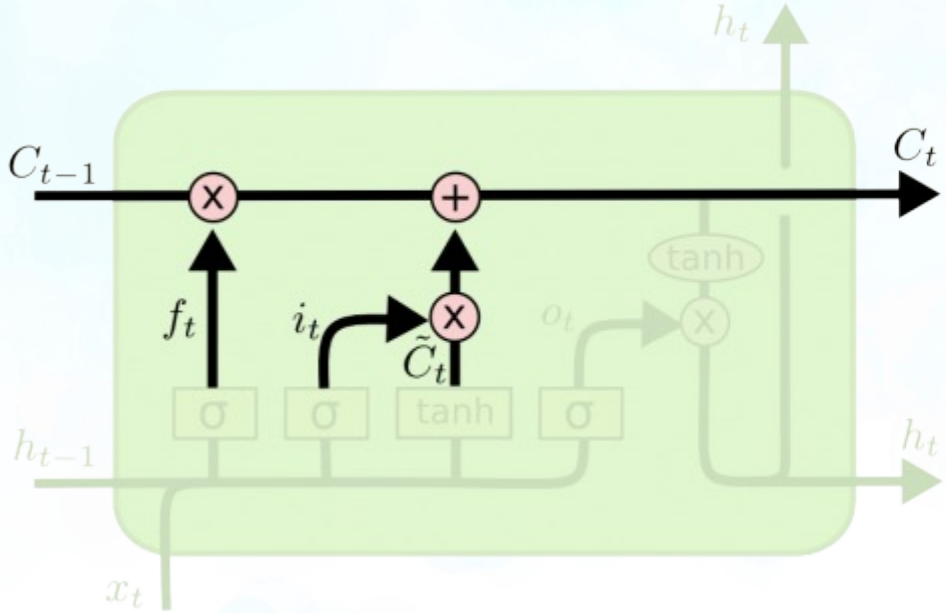


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Input gate layer

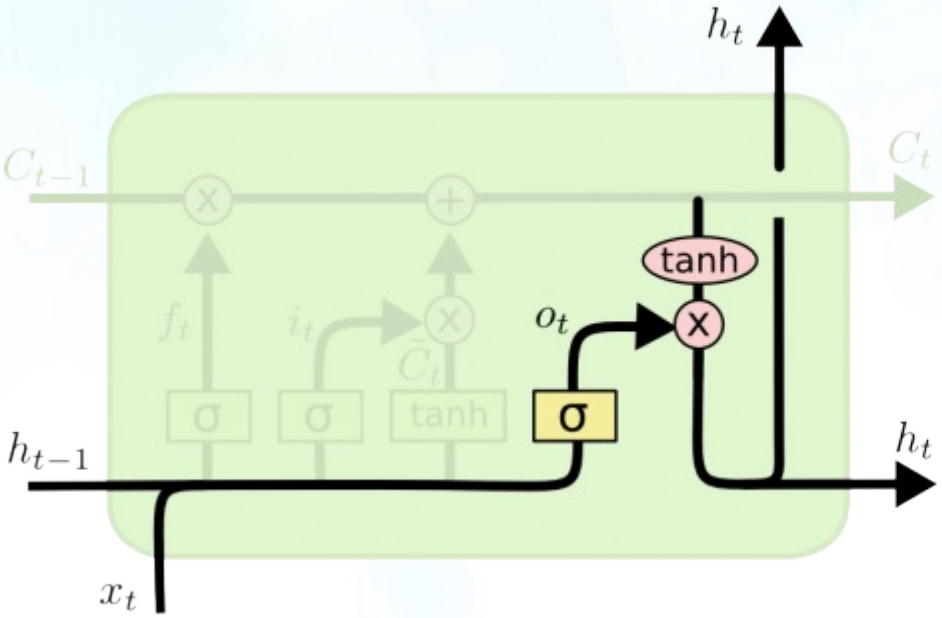
- 接下來結合這兩部分來進行狀態更新。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate layer

- 最後一步：決定輸出的內容。
- 首先透過sigmoid層決定要輸出單元狀態的哪些部分，然後透過tanh函數（把值轉換為 [-1,1] 區間），把它的單元狀態與sigmoid的輸出相乘，因此決定最後輸出的部分。



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Pytorch Model - lstm

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two GRUs together to form a *stacked GRU*, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each GRU layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional GRU. Default: `False`

https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Pytorch Model - lstm

Inputs: input, (h_0, c_0)

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.
- **c_0**: tensor of shape $(D * \text{num\_layers}, H_{cell})$ for unbatched input or $(D * \text{num\_layers}, N, H_{cell})$ containing the initial cell state for each element in the input sequence. Defaults to zeros if (h_0, c_0) is not provided.

where:

$$
\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{cell} &= \text{hidden\_size} \\
H_{out} &= \text{proj\_size if proj\_size} > 0 \text{ otherwise hidden\_size}
\end{aligned}
$$

https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Pytorch Model - lstm

```
>>> rnn = nn.LSTM(input_size=10, hidden_size=20, num_layers=2)
>>> # input=(sequence length, batch_size, input_size)
>>> input = torch.randn(5, 3, 10)
>>> # h0=(D * num_layers, batch_size, H_out) input_size)
>>> h0 = torch.randn(2, 3, 20)
>>> # c0=(D * num_layers, batch_size, H_out) input_size)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Pytorch Model - lstm

```
# Recurrent neural network (many-to-one)
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, num_classes):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        # Set initial hidden and cell states
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))  # out: tensor of shape (batch_size, seq_length, hidden_size)

        # Decode the hidden state of the last time step
        out = self.fc(out[:, -1, :])
        return out

model = RNN(input_size, hidden_size, num_layers, num_classes).to(device)
```

實作一： lstm 模型

- https://colab.research.google.com/drive/1AIxkbrUKLS2pEYd4z5MRivAIW2gXBWxN?usp=sharing

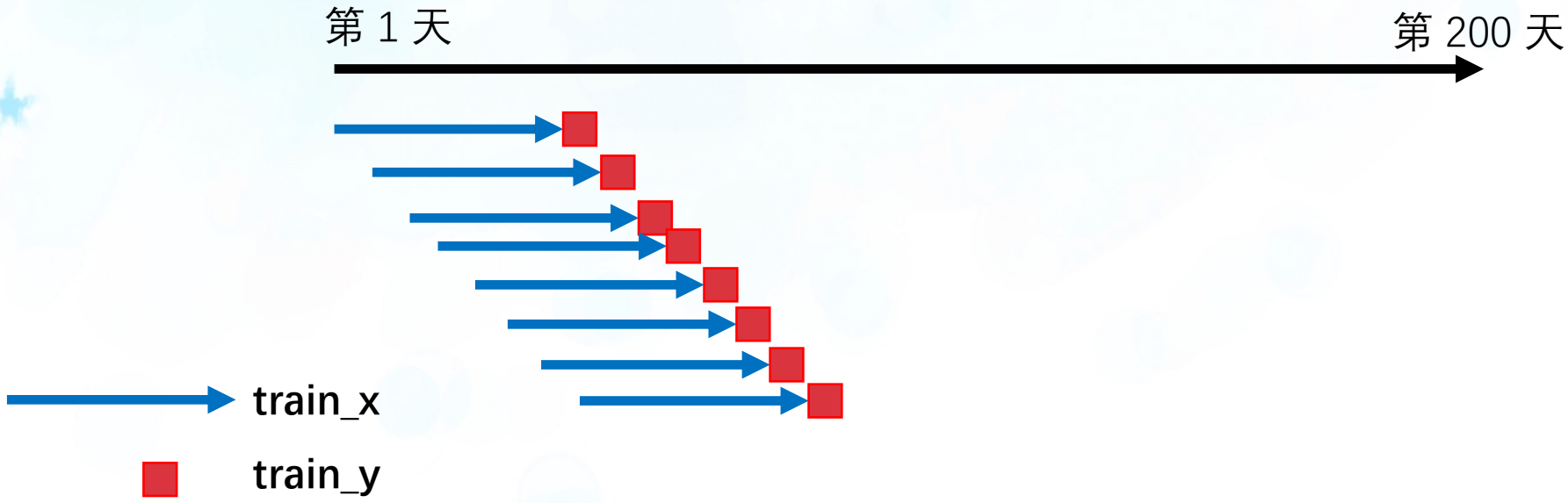
```
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        # Write your code here.

    def forward(self, x):
        # Write your code here.

        return out
```

股票預測

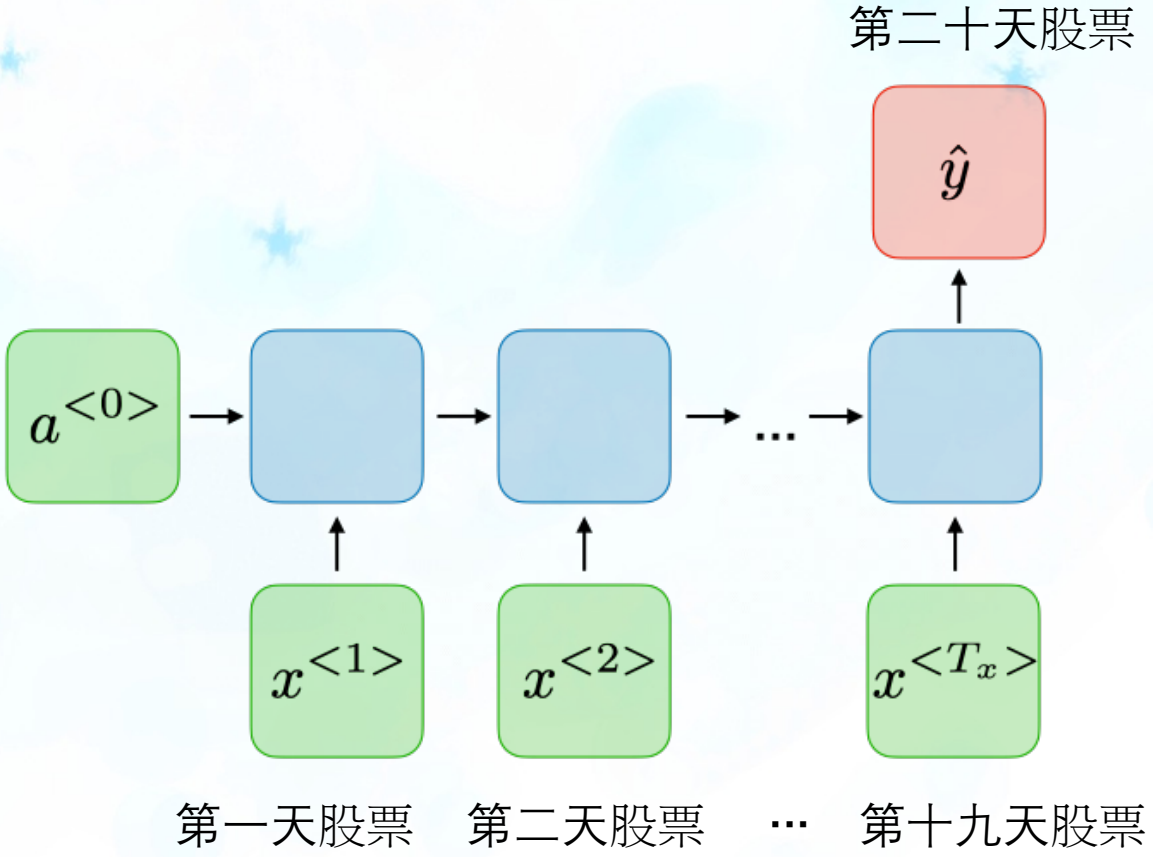

Amazon Stock Price
Close Price (USD)
1200
1000
800
600
400
200
0
2006-01-03
2007-12-28
2009-12-22
2011-12-16
2013-12-13
2015-12-09
2017-12-04
Date

股票預測

- 每一筆 train x 都有 19 個值，代表第 1 天到第 19 天股價。
- 每一筆 train y 有 1 個值，代表第 20 天的股價。
- 透過前 19 天的股價預測第 20 天的股價。



股票預測

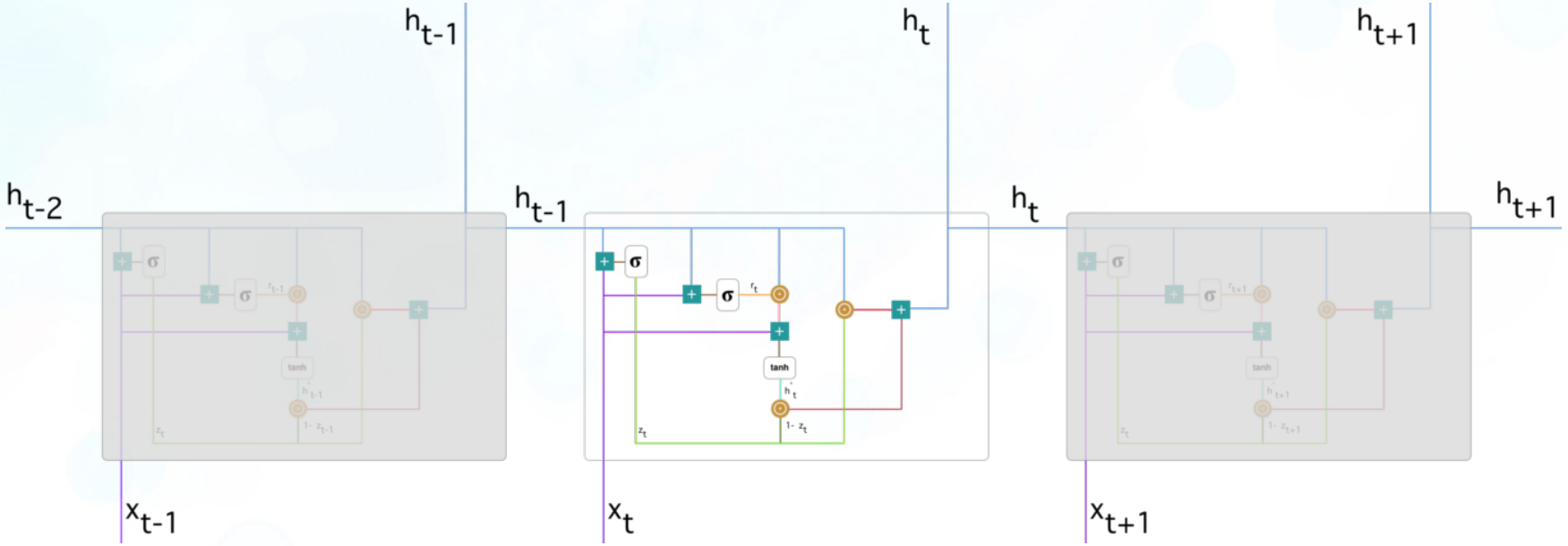


03

Gated recurrent unit(GRU)

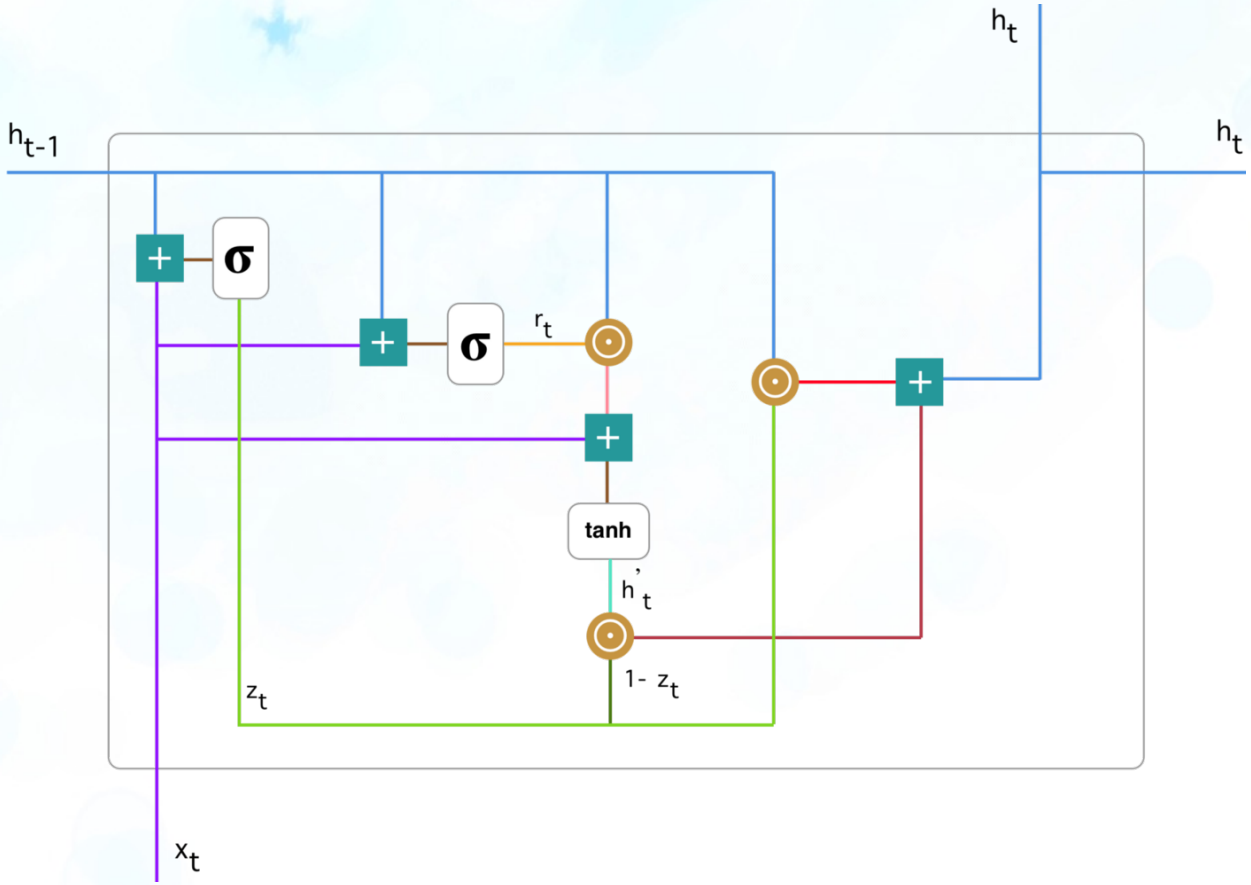
Gated Recurrent Unit（GRU）

- 上面介紹標準的 LSTM，不過不是所有的 LSTM 結構相同。
- 實際上與 LSTM 相關的論文都有採用微小的變形，如目前比較流行的 GRU：



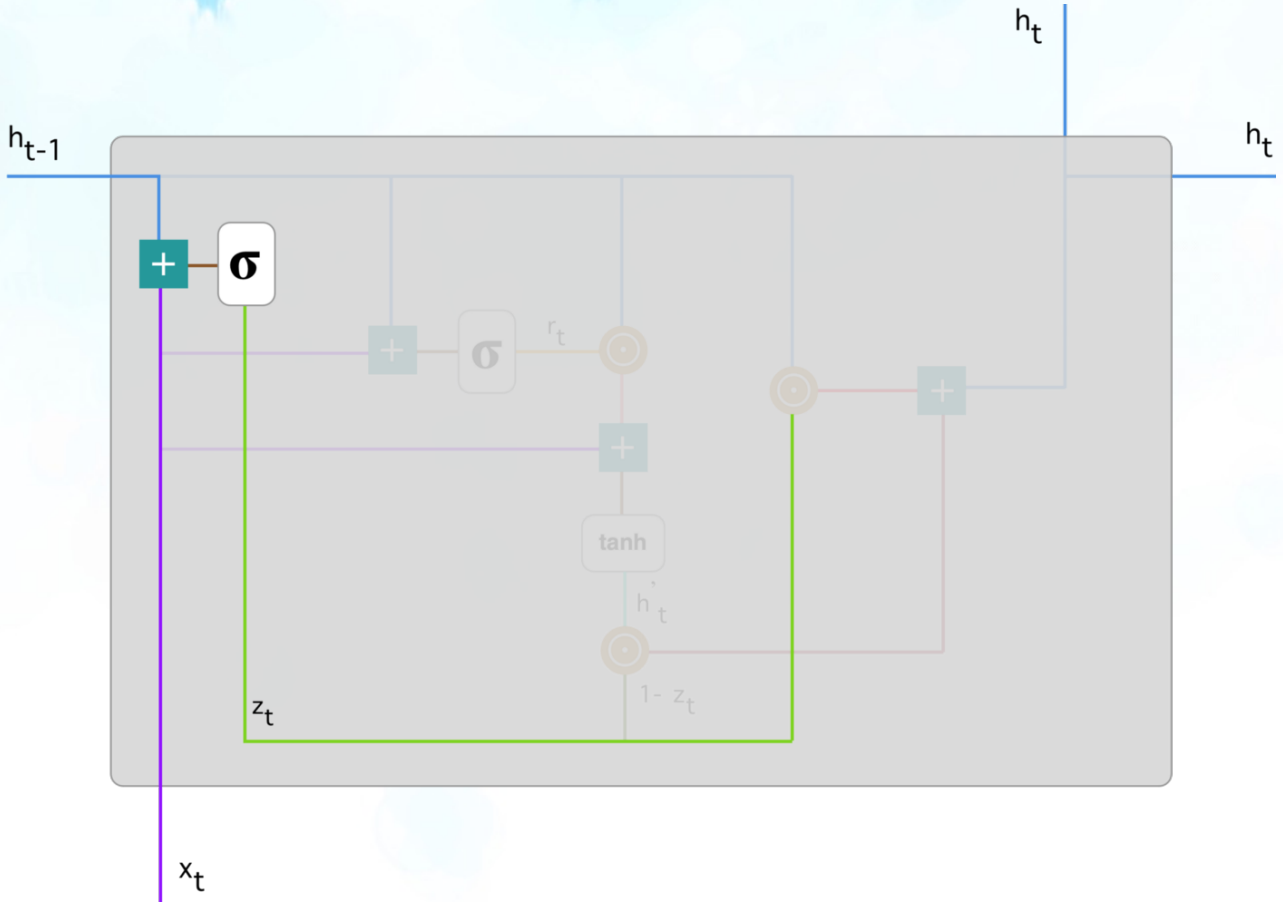
Gated Recurrent Unit（GRU）

- 整體 GRU 架構



Gated Recurrent Unit（GRU）

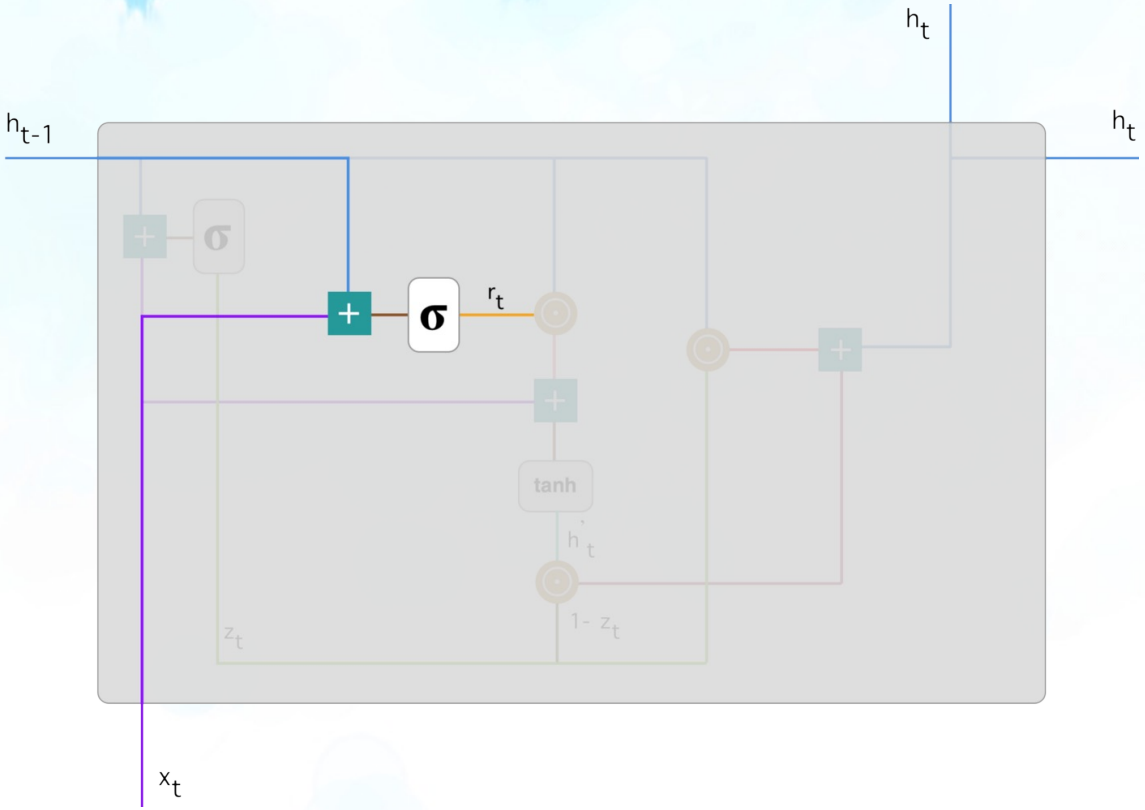
- 第一步 Update gate：計算第 t 個時間點的 update gate z_t



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Gated Recurrent Unit（GRU）

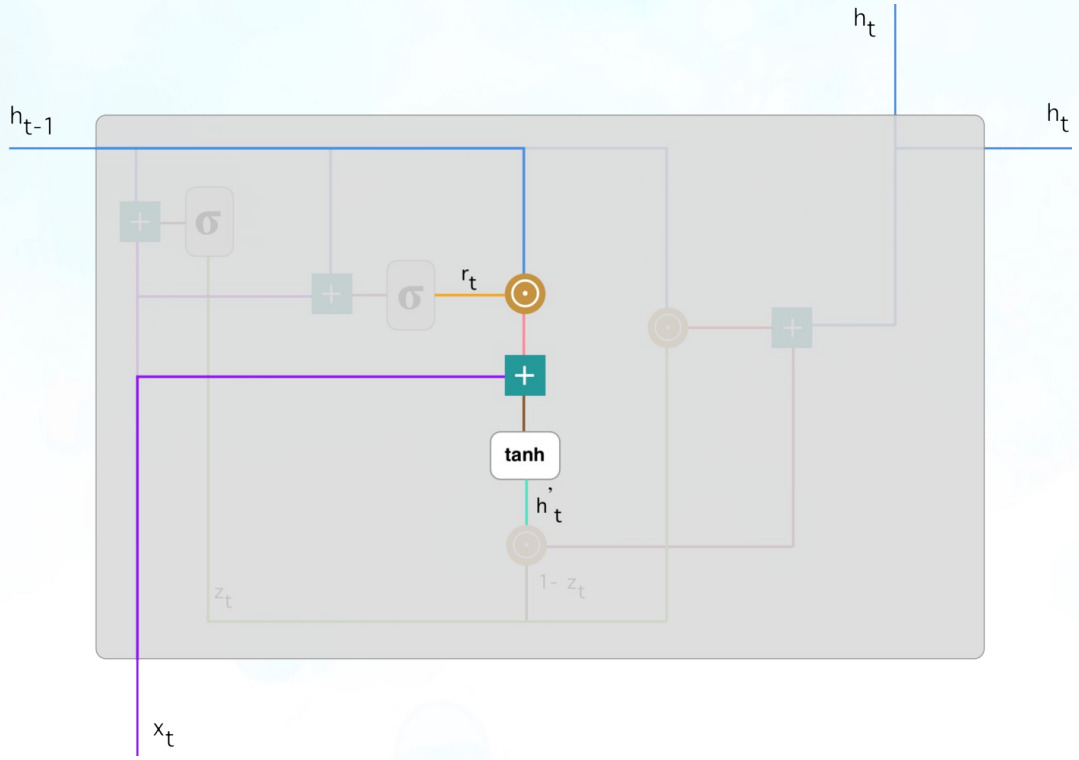
- 第二步 Reset gate：這步決定有多少過去的資訊要被遺忘



$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Gated Recurrent Unit（GRU）

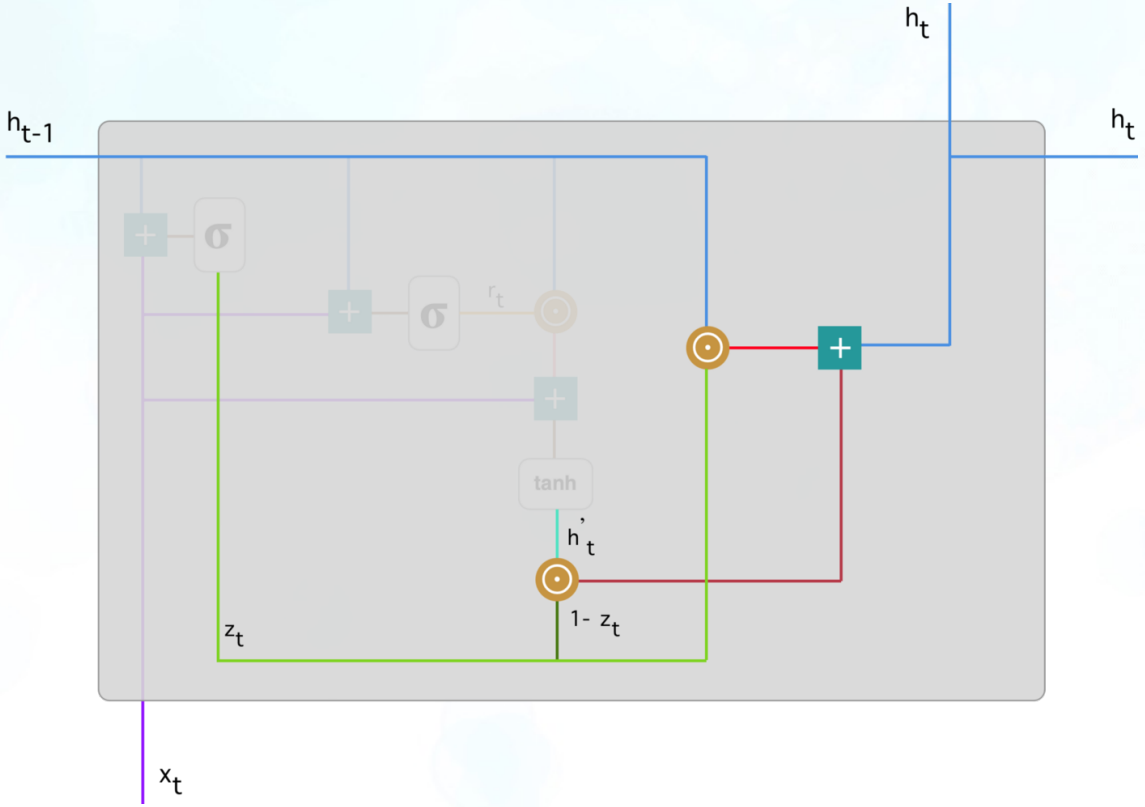
- 第三步 Current memory content：使用 reset gate ，決定要從過去取得多少資訊



$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

Gated Recurrent Unit（GRU）

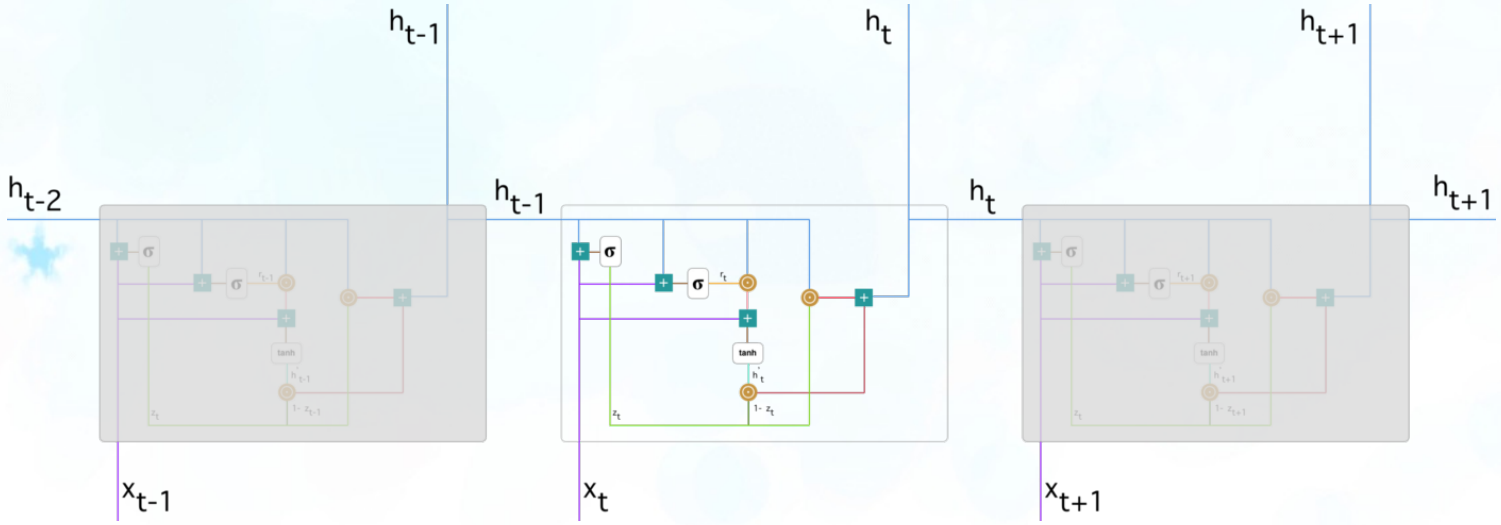
- 第四步 Final memory：最後計算 output ，要取得多少當前的資訊以及要取得多少過去的資訊。



$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

Gated Recurrent Unit（GRU）

- 整體演算法



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

Pytorch Model - GRU

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two GRUs together to form a *stacked GRU*, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each GRU layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional GRU. Default: `False`

https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

Pytorch Model - GRU

Inputs: input, h_0

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D$ torch.nn.utils.rnn.pack_sequence $)$ or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence. Defaults to zeros if not provided.

where:

$$
\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{out} &= \text{hidden\_size}
\end{aligned}
$$

https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

Pytorch Model - GRU

```
>>> rnn = nn.GRU(input_size=10, hidden_size=20, num_layers=2)
>>> # input=(sequence length, batch_size, input_size)
>>> input = torch.randn(5, 3, 10)
>>> # h0=(D * num_layers, batch_size, H_out)
>>> h0 = torch.randn(2, 3, 20)
>>> output, hn = rnn(input, h0)
```

https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

實作二： gru 模型

- https://colab.research.google.com/drive/1AIxkbrUKLS2pEYd4z5MRivAIW2gXBWxN?usp=sharing

```
class GRU(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(GRU, self).__init__()
        # Write your code here.

    def forward(self, x):
        # Write your code here.

        return out
```

04

實作

實作 - RNN

- 題目：股票價格預測(AMZN)
- 利用範例程式，撰寫 LSTM 以及 GRU 模型。

程式碼：

https://colab.research.google.com/drive/1AIxkbrUKLS2pEYd4z5MRivAIW2gXBWxN?usp=sharing

繳交方式

- 將檔案壓成 zip 檔並上傳至 Week 9 作業繳交
  - 繳交 zip 檔，須包含 **ipynb** 檔 以及 **docx**檔。
  - ipynb 須包含模型程式碼
  - docx 須包含模型程式碼截圖以及兩種模型之預測結果截圖

Reference

- https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be