遞迴神經網路與序列模型

蔡炎龍&教研處

「版權聲明頁」

本投影片已經獲得作者授權台灣人工智慧學校得以使用於教學用途，如需取得重製權以及公開傳輸權需要透過台灣人工智慧學校取得著作人同意；如果需要修改本投影片著作，則需要取得改作權；另外，如果有需要以光碟或紙本等實體的方式傳播，則需要取得人工智慧學校散佈權。

– 台灣人工智慧學校

課程內容

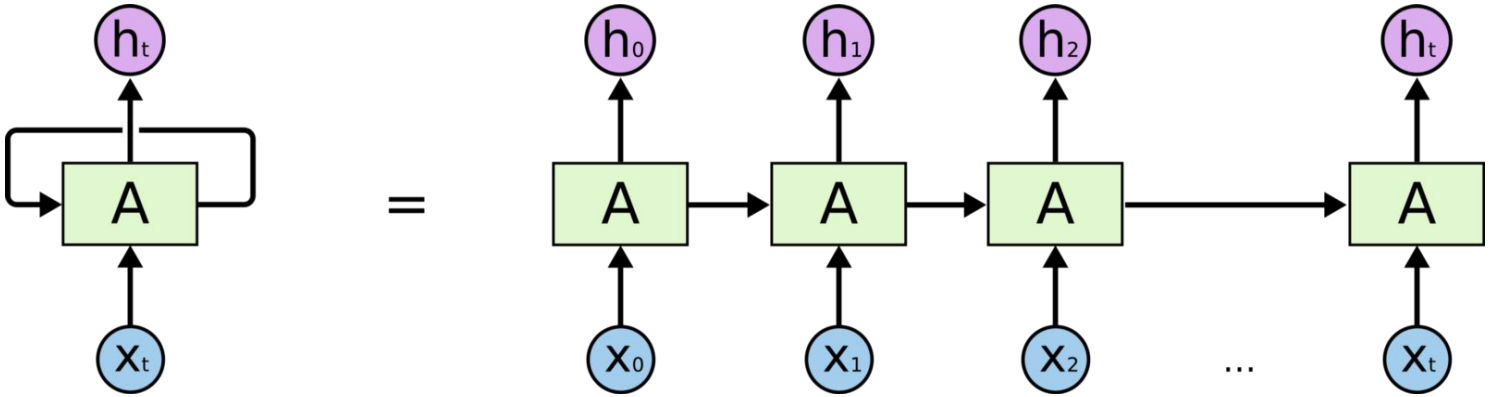
1. DNN 與 RNN 理論講解
2. 快速回顧 RNN & LSTM
3. Tensorflow 實作 RNN

快速回顧遞迴神經網路 (RNN)

RNN

- recurrent neural network
- 有記憶的神經網路
- 有隱藏狀態(hidden state)當作輸入, 隨著時間序列依序傳進RNN
- 公式:h_t = activation(W·concat(x_t, h_{t-1}) + b)

圖解



但有多少人看得懂...

用簡單的矩陣運算操作一遍！

- 假設現在有一筆時間序列的資料 shape = (3, 6)
- 有三個時段, 每個時段有六個 featrue

$$X\_feature = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 2 & 1 & 3 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 \end{pmatrix} \begin{matrix} \rightarrow t1 \\ \rightarrow t2 \\ \rightarrow t3 \end{matrix}$$

設定RNN要輸出的大小

- 就跟普通的 dense layer 一樣, RNN 網路也是要把n維空間的資料縮放到m維空間, 這裡假設我們要把每一個時段有六維的 feature mapping 到三維的空間

- 先初始化weight, weight shape = (n, m+n) = (3, 9)
  n 代表要轉換的維度, m 代表原本的feature大小

$$\text{RNN\_weight} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 & 2 & 0 & 0 \end{pmatrix}$$

決定initial state

- RNN的輸入必須有隱藏狀態 h, 因此我們在第一步的時候需要給他一個初始的 h0(通常裡面的數值都是給0), initial state 的長度跟 RNN 的輸出一樣。

$$h0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

可以來計算了！

- z1 = weight · concat(x_t1, h0) (為了計算方便省去bias)
- h1 = activation(z1) (這裡我們用Relu)

$$h1 = z1 = \begin{pmatrix} 3 \\ 1 \\ 5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

x_t

hidden state

可以來計算了！

- z2 = weight · concat(x_t2, h1)
- h2 = activation(z2)

$$h2 = z2 = \begin{pmatrix} 8 \\ 9 \\ 12 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 1 \\ 3 \\ 0 \\ 0 \\ 3 \\ 1 \\ 5 \end{pmatrix}$$

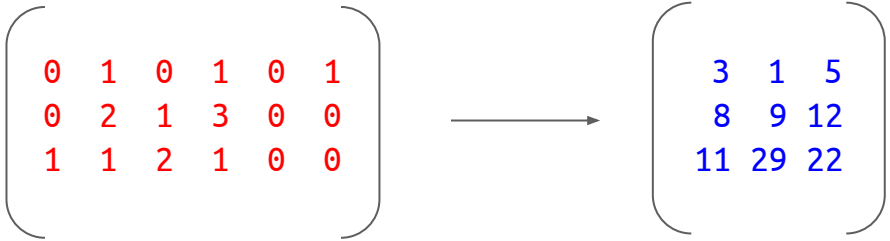
可以來計算了！

- z3 = weight · concat(x_t3, h2)
- output = h3 = activation(z3)

$$h3 = z3 = \begin{bmatrix} 11 \\ 29 \\ 22 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 0 \\ 0 \\ 8 \\ 9 \\ 12 \end{bmatrix}$$

That's it!

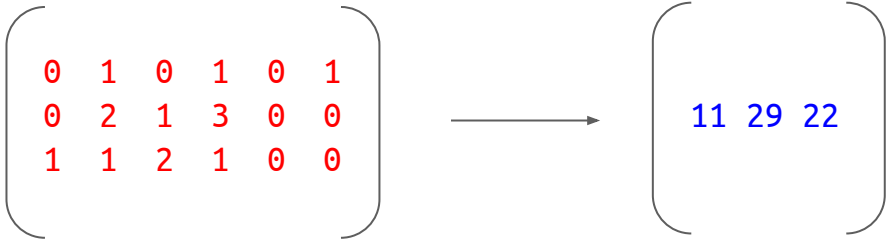
- 我們成功的把時間長度為三, 每個單位時間有六個維度的資料轉換成轉換成單位時間有三維的資料
- 如果有多個 sample, 算法也是一樣的
- shape 的變化:
  (batch, time_step, feature_m) -> (batch, time_step, feature_n)



That's it!

- 在一些 RNN 的應用, 最有用的資訊是最後一個時間點的輸出(網路看過所有資料的輸出), 因此為了降低維度, 會捨棄掉其他時間點的輸出當結果。

- shape 的變化:
  (batch, time_step, feature_m) -> (batch, feature_n)



變數量的變化

- 為什麼面對時間序列的問題, 是選擇用 RNN 而不是 DNN？
- 當然也可以使用 DNN！
- 使用 DNN, 以剛剛的例子, shape (3, 6) -> shape (3), 需要的變數量為 3 * 6 (攤開時間序列) * 3 = 54
- 剛剛的 RNN weight, 變數量只有 3 * 9 = 27, 是DNN的一半。
- 如果資料有 1000 個時段, RNN weight 還是 27 個, 但 DNN weight 就會有 1000 * 6 * 3 = 18000！

補充

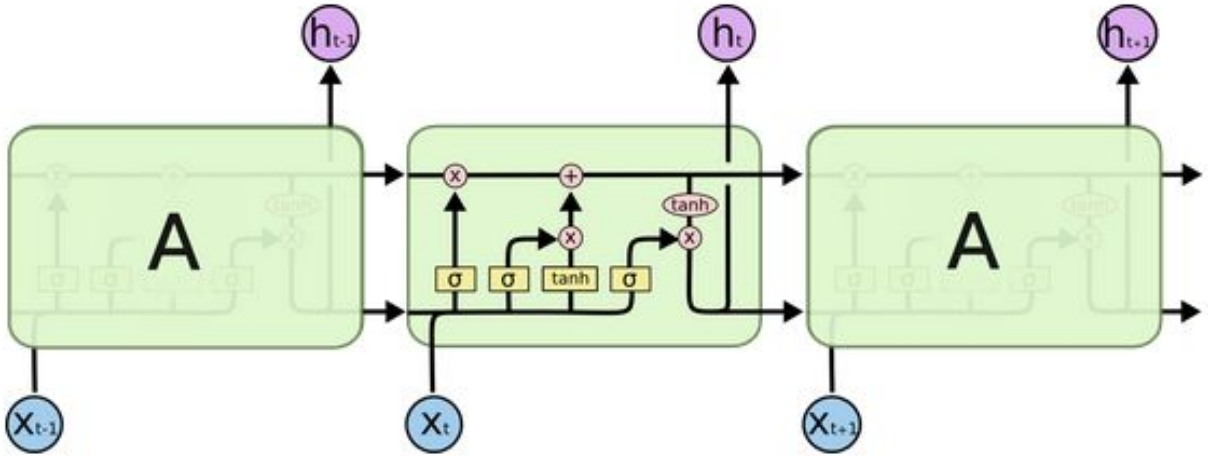
- $h_t = \phi(W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b)$

$$W \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = W_{xh} \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} + W_{hh} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Introduce LSTM

LSTM

$$
\begin{aligned}
i_t &= sigmoid(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= sigmoid(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= sigmoid(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$



The repeating module in an LSTM contains four interacting layers.

LSTM

- Long Short Term Memory network

- $$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

- 從一組weight變成四組weight，其中三組的矩陣運算結果分別代表 forget gate、input gate、output gate。

- 除了原本要算的 hidden state h_t，現在又多了一個 cell memory c_t 要傳到下一個時間點當輸入。

gate 的用途

- 在 LSTM 中，網路建構了 3 個 gate 控制信息流通量：
  - 輸入門 i_t：多少信息可以流入 memory cell c_t
  - 遺忘門 f_t：上一時刻 memory cell c_{t-1} 中的信息可以累積到當前 memory cell c(t) 中
  - 輸出門 o_t：當前時刻的 memory cell c_t 有多少可以流入當前隱藏狀態 h_t 中

Why we use LSTM

- RNN的特點是使用同樣的weight重複對時間序列做計算
- 100 time steps = 100 layers
- Vanishing Gradient，先輸入的資訊重要性變低
- 利用三種gate使網路比較不會遺忘先前的輸入

梯度爆炸

- LSTM 只解決了梯度消失問題，但並沒有解決梯度爆炸問題。
- 實作上利用 gradient clipping 來防止梯度爆炸。

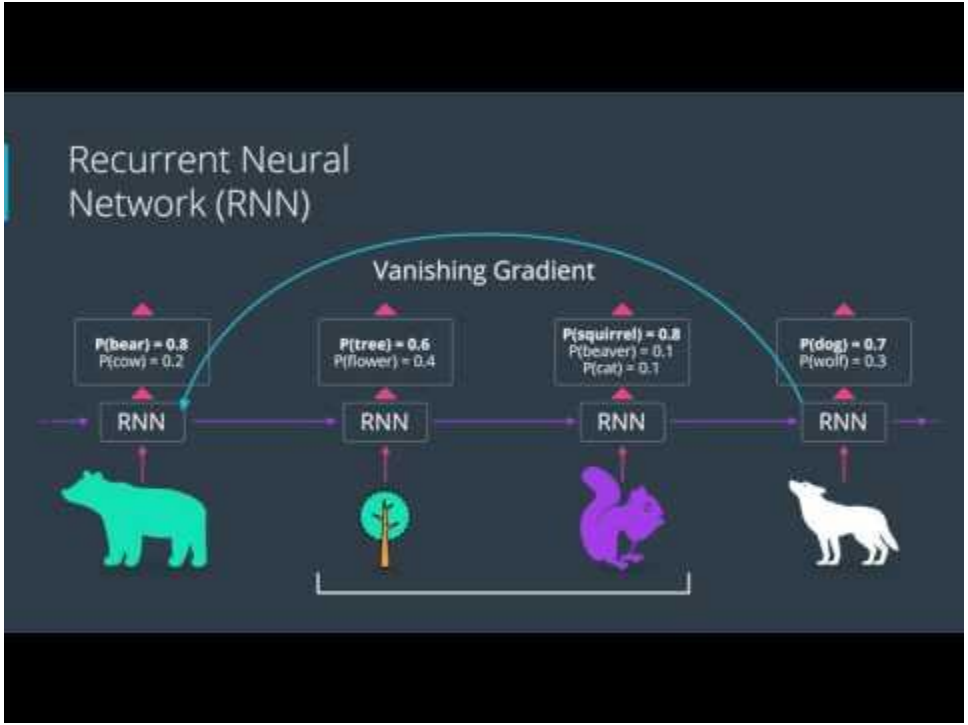
FNN vs. CNN vs. RNN

- Feedforward Neural Network
  - 自己學習做 feature engineering
- Convolutional Neural Network
  - 學習如何 “觀察” (特徵偵測)
- Recurrent Neural Network
  - 學習如何 “聽取” (訊息過濾)

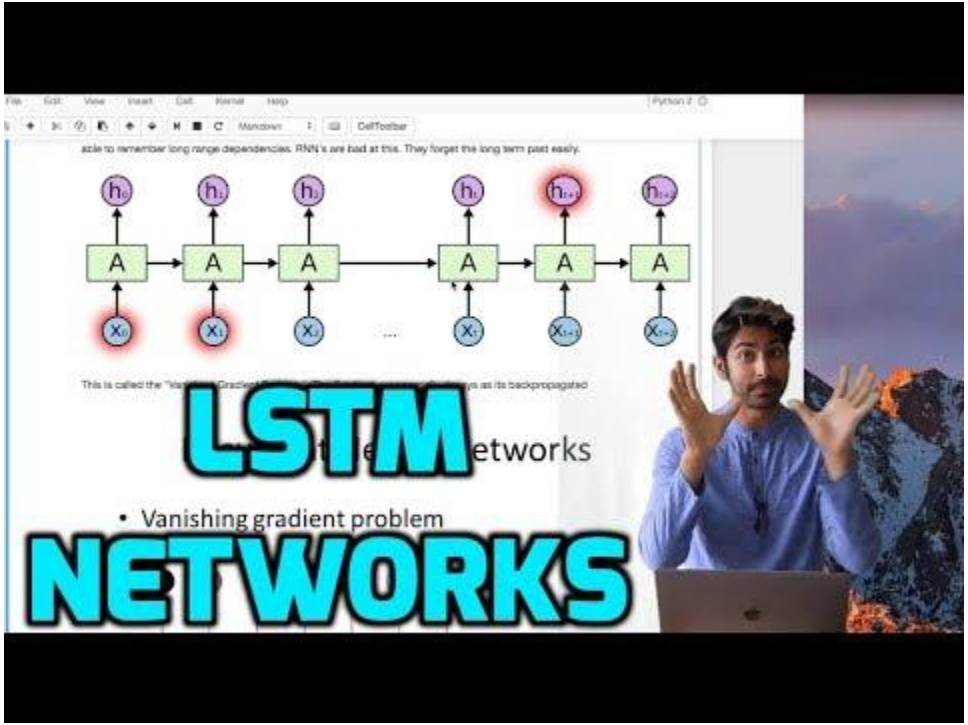
Related Reference

- Andrej Karpathy
- Understanding LSTM Networks
- 有趣的機器學習 LSTM
- CNN、RNN、DNN的內部網路結構有什麼區別

Udacity RNN: quick introduction



numpy 版 LSTM !!! (optional)



Tensorflow 實作 RNN

從 RNN cell 開始

- `tf.contrib.rnn.BasicRNNCell` 會傳一個最基本的 RNN 運算物件
  - 用法：

```
simple_cell = tf.nn.rnn_cell.BasicRNNCell(10)
output, h_state = simple_cell(input, init_state)
```

- `simple_cell` 的輸入是一個 time step 的 feature，然後會返回計算出的 output 和 hidden state。但在 basic rnn 的例子裡，output 和 hidden state的數值是一樣的。

- cell 只能做 RNN 一步的運算而已。

tf.nn.dynamic_rnn

- 由於一個 RNN cell 只能做一步的運算，而 RNN 要進行的是多步 time step 的計算，所以需要 tf.nn.dynamic_rnn 的幫忙。

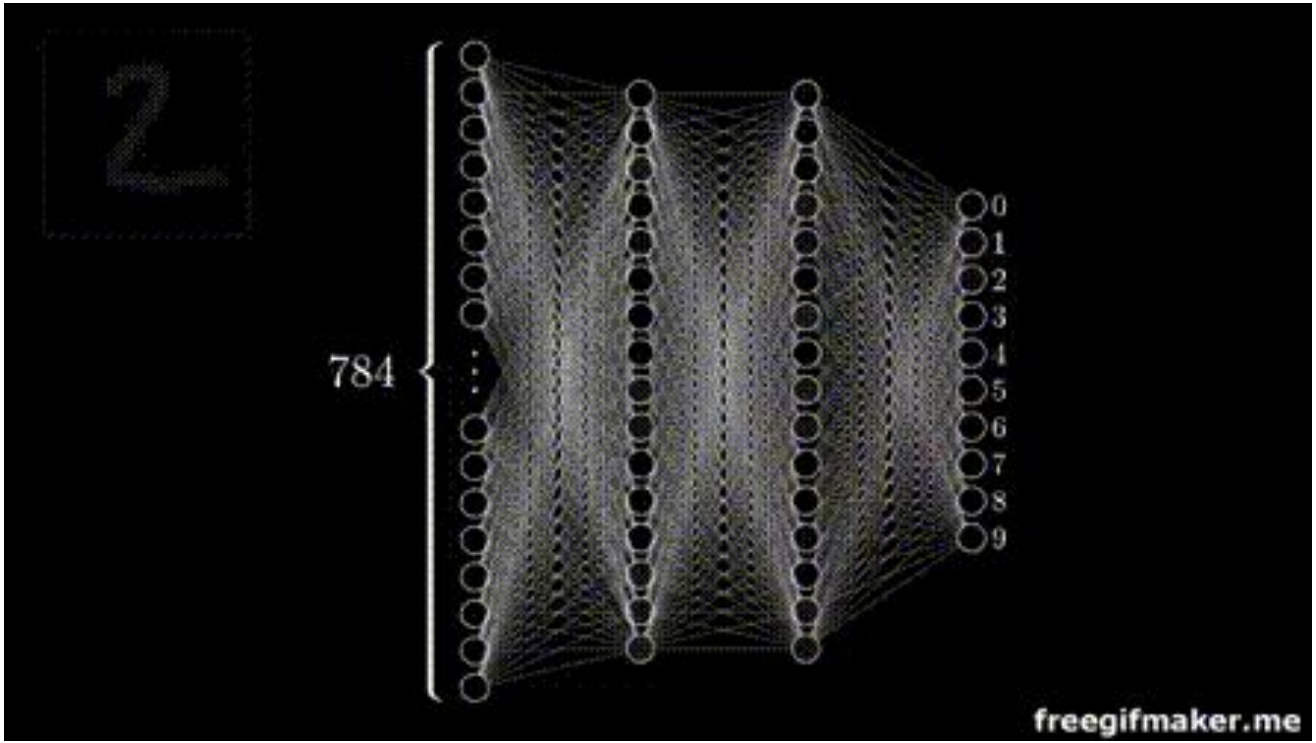
- tf.nn.dynamic_rnn 自動將資料重複通過 RNN cell，回傳出每一個 time step 的結果以及最後一個 time step 的 hidden state。
  - ```
    outputs, final_state = tf.nn.dynamic_rnn(simple_cell,
                                             input_data,
                                             init_state)
    ```


    ```

手寫數字辨識(28x28)

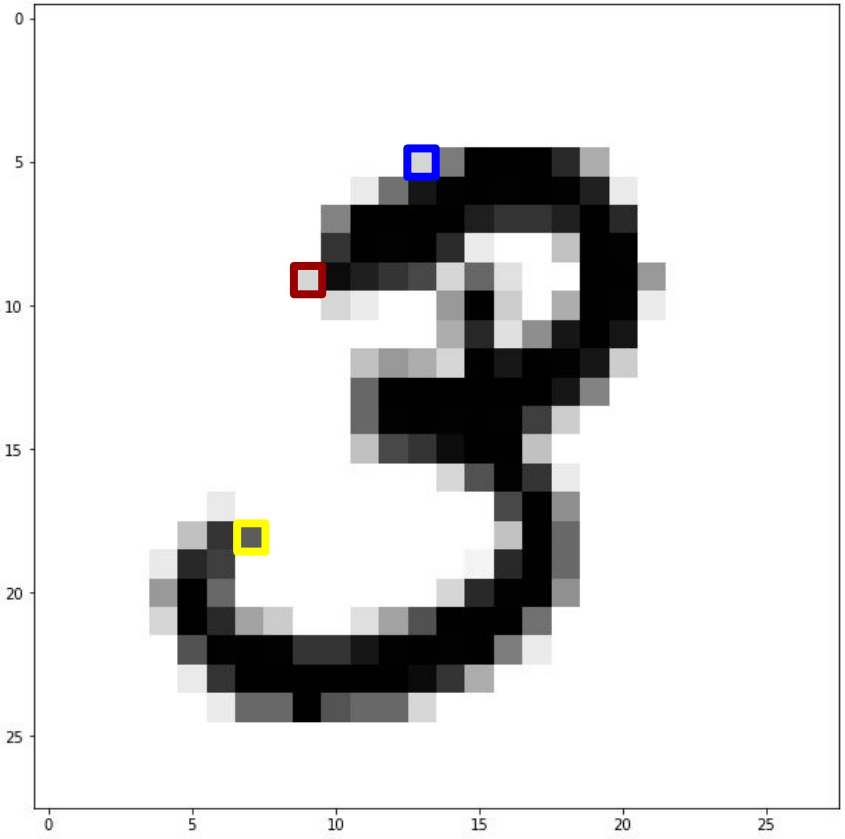
- 同樣為手寫數字辨識的問題，我們可以把它轉化成：
  - 普通的DNN神經網路，784個pixel每一個都是獨立的feature
  - 圖像分析的CNN神經網路，用多個filter檢視固定大小的區域取得圖特有的紋路
  - 時序模型的RNN神經網路，28個row代表28個time step，每一個time step輸入28個pixels，模型會記憶不同時段的資料，預測出結果

For DNN



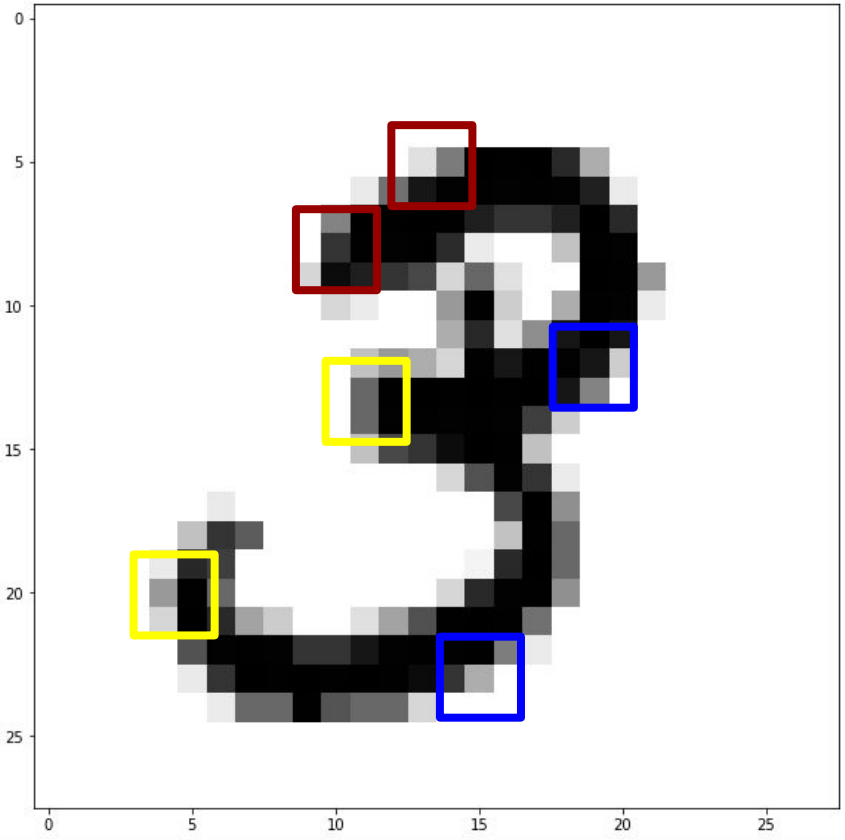
https://www.youtube.com/watch?v=Ilg3gGewQ5U

For DNN



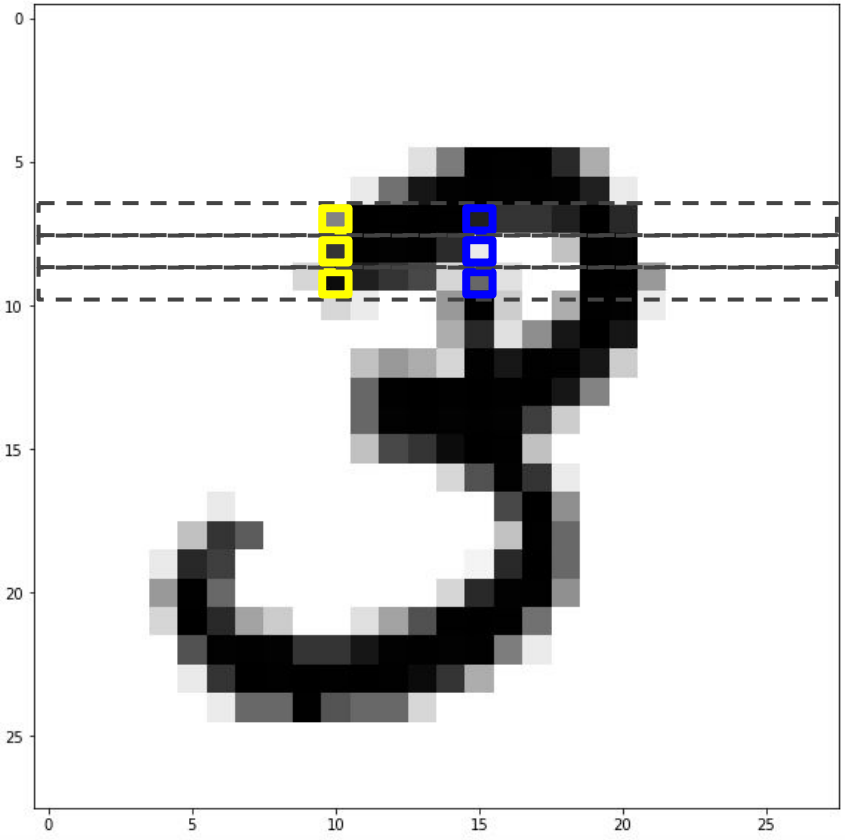
三個點各代表不同的三個feature。

For CNN



三種filter掃過整張圖,
取出三個feature map。

For RNN



RNN一次讀取一個row，feature數只有28個。（兩個黃色代表同樣對應的feature，只是不同時間輸進RNN）

關於 RNN 的輸出

- 在手寫數字辨識的例子中，假設我們 batch size = 128，輸入 RNN 的資料形狀 data.shape = (batch_size=128, time_step=28, feature=28)。

- 假設要將圖片轉換成32維，那輸出結果的形狀就會是 outputs.shape = (batch_size=128, time_step=28, newfeature=32)。

  其中 outputs[:, 0, :] 代表神經網路看過一個 row 所產生的新的32維，
  而神經網路不可能只讀取一行 pixel 就能判斷圖片的種類，一定是看完整張圖片才能獲得最詳盡的資訊，因此我們通常會取最後一個 time step 的結果接到下一層。
  也就是 last_output = output[:, -1, :]

練習時間

- 將普通的 RNN 改成 LSTM 或者 GRU，觀察訓練效果如何。
- 把手寫數字每兩張合併成一張大圖輸入到 model，嘗試讓模型能一次預測兩個數字。

*實際上 RNN 不擅長識別圖像，這個章節主要是簡單介紹用 tensorflow 建立 RNN model