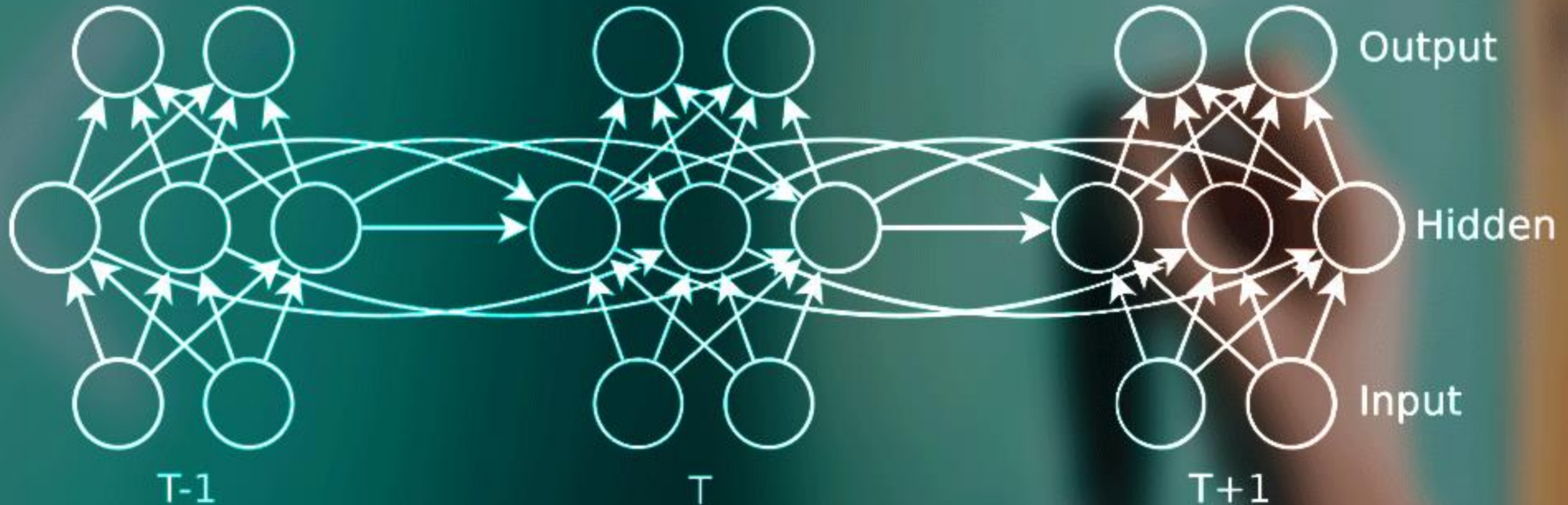


A blurred city street scene with a pink and blue banner overlay. The background shows a busy street with pedestrians and buildings. A pink banner with a blue triangle on the right side is positioned across the middle of the image. The text "反覆 持續 注意力..." is written in white on the pink banner.

反覆 持續 注意力...



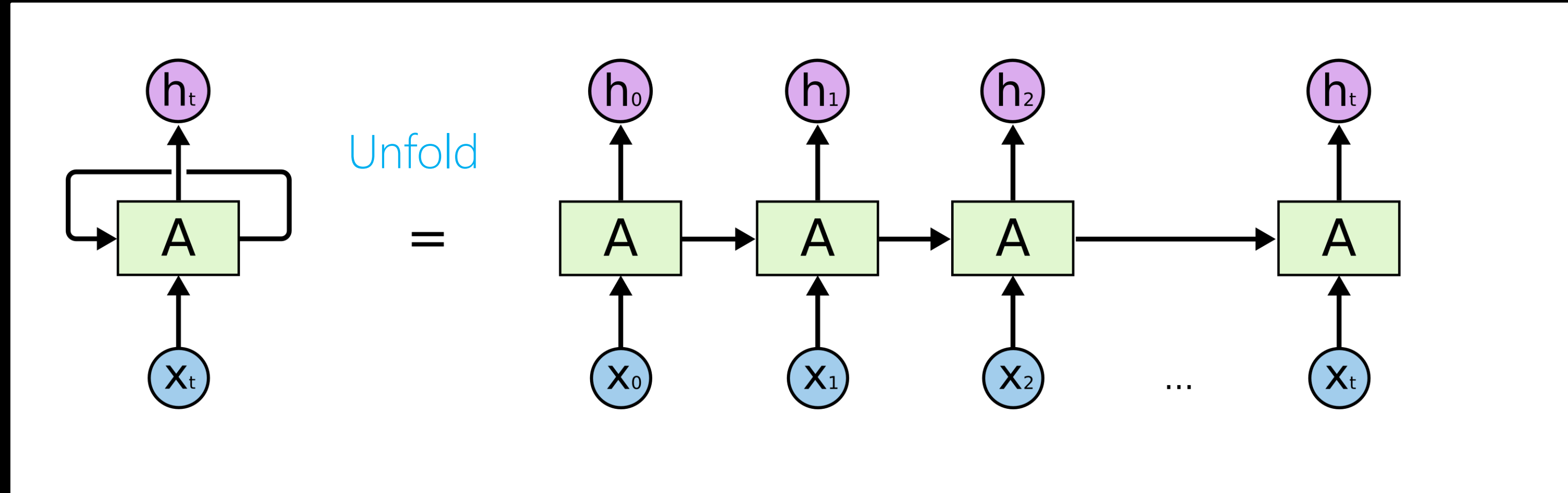
語言是一種序列資料  
序列資料的數據表示方式就是清單List  
`np.array([array1, array2, array3.....])`





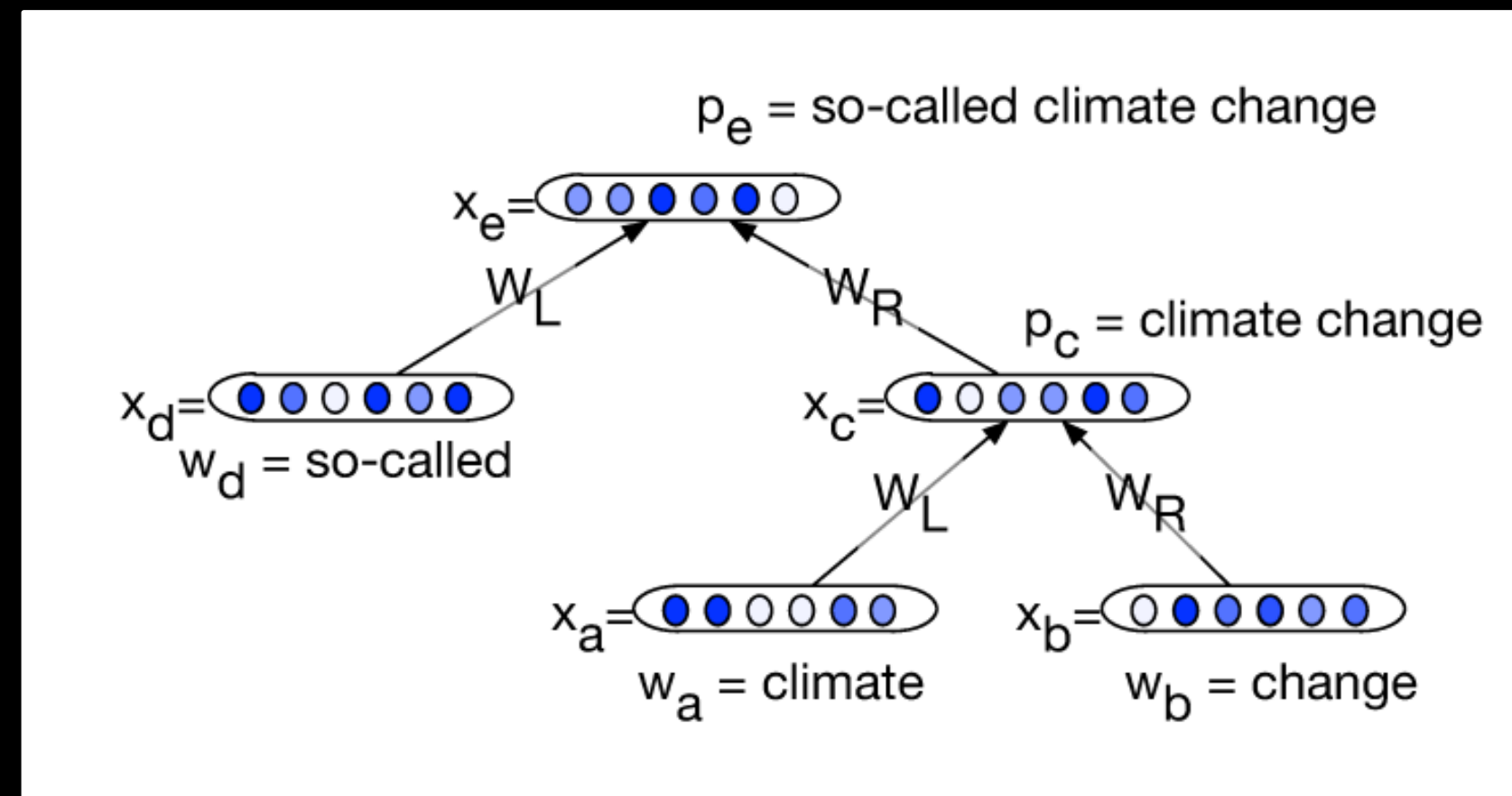
# 遞迴神經網路

RNN, Recurrent Neural Networks

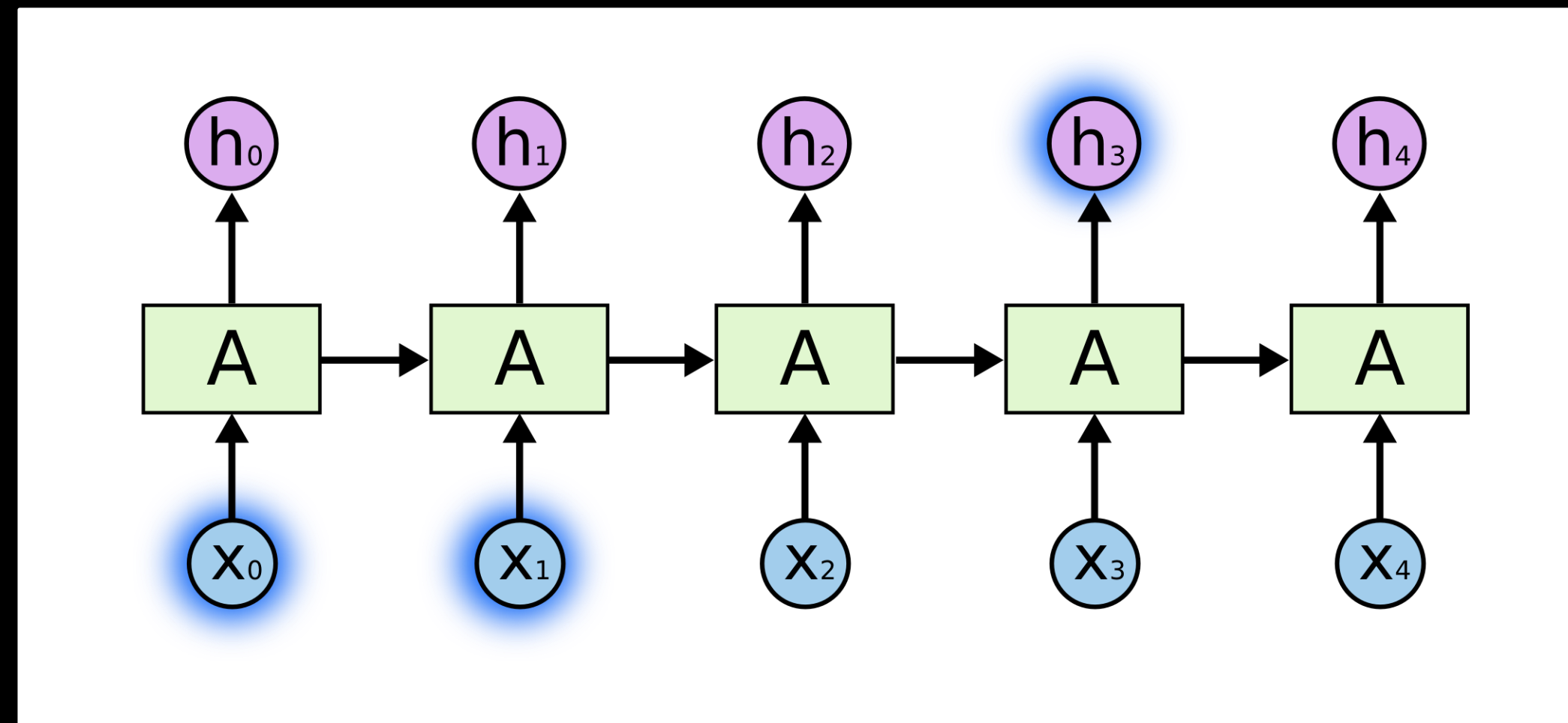


# 循環神經網路

RNN, Recursive Neural Networks



預測未來

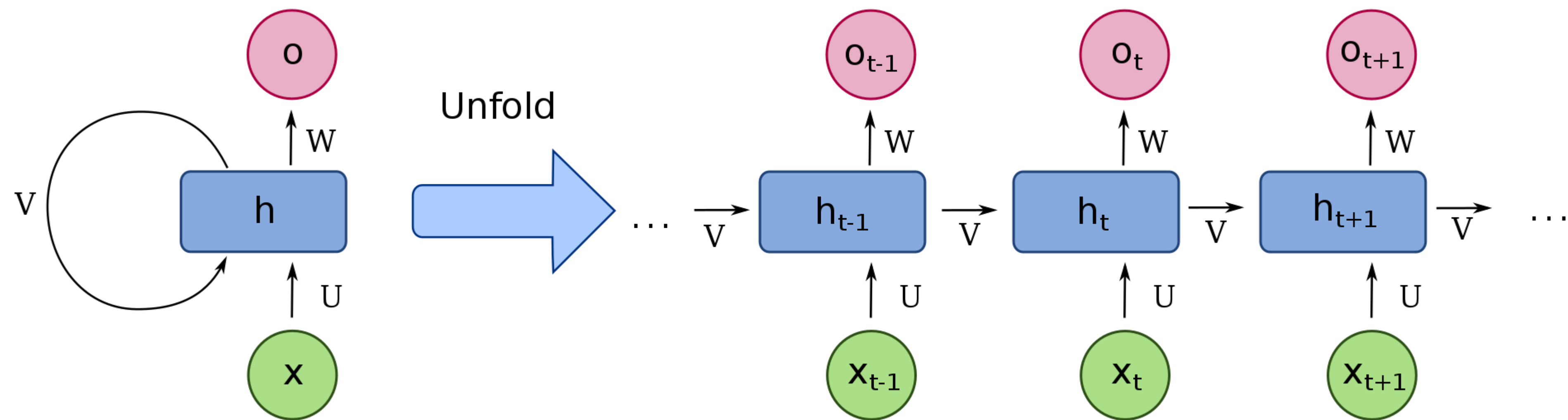


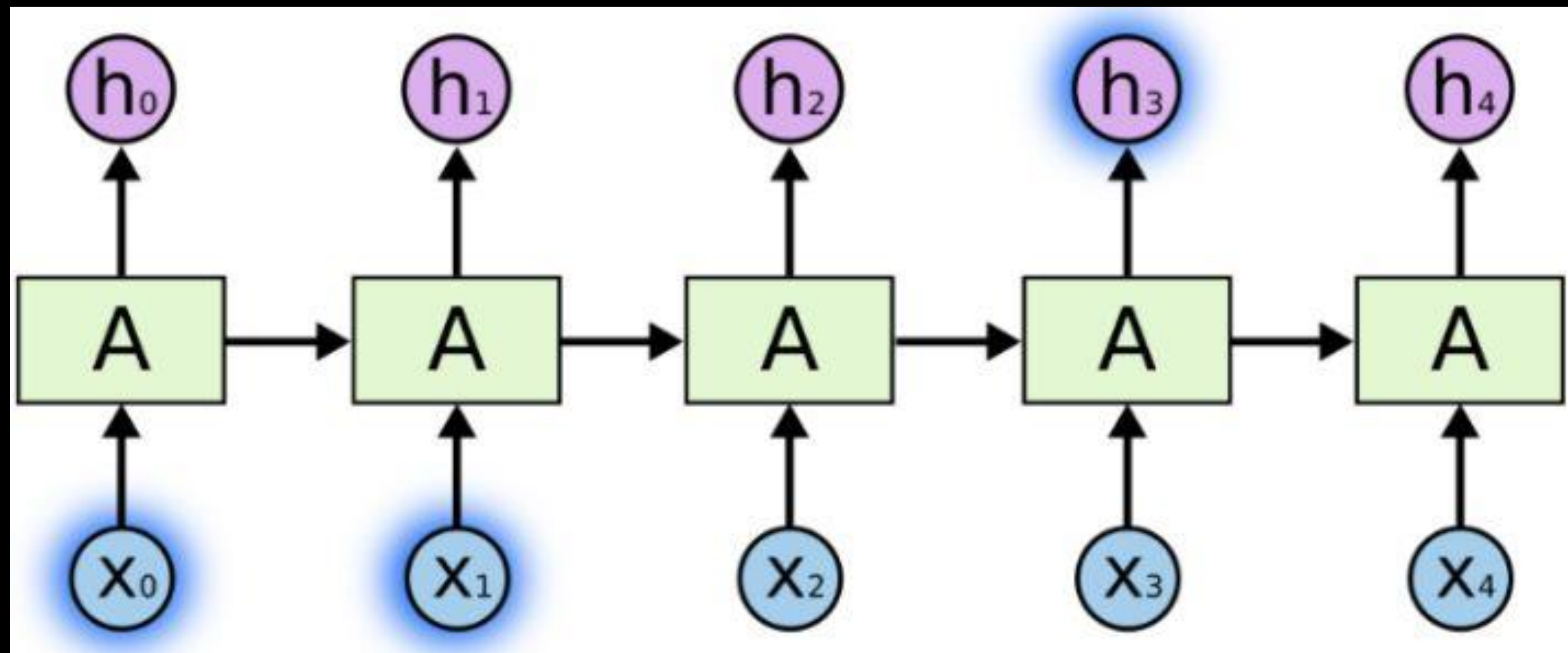
根據過去歷史

$0.99^{100}=0.3660$  梯度彌散

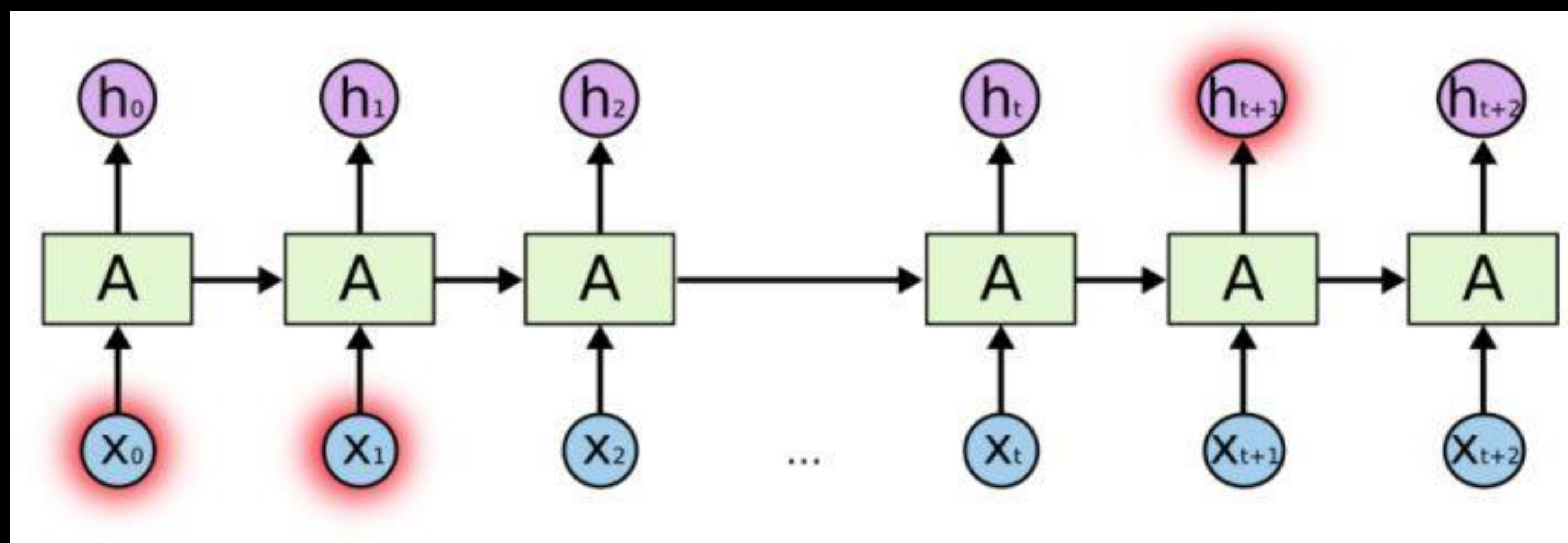
$1.01^{100}=2.7048$  梯度爆炸

# 遞迴神經網路



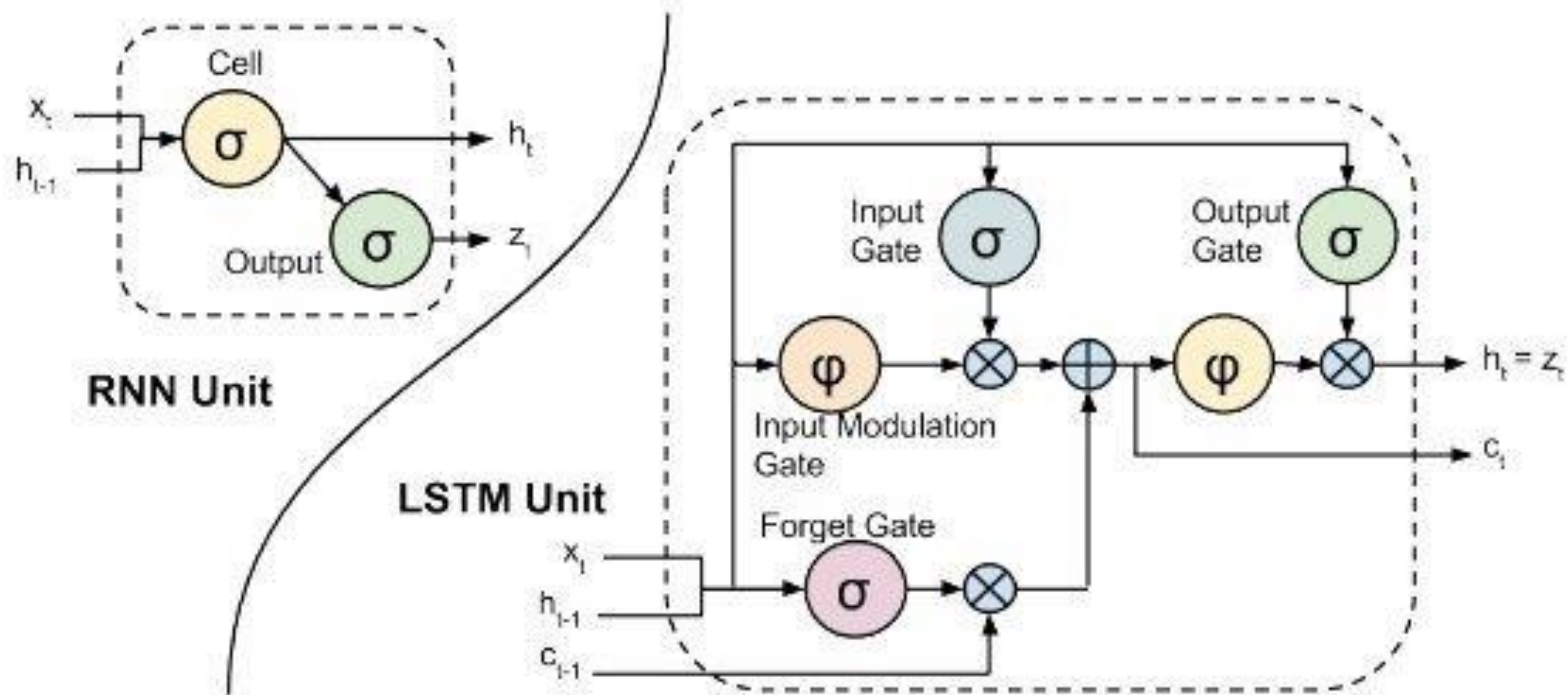


長期依賴 Long-Term Dependencies



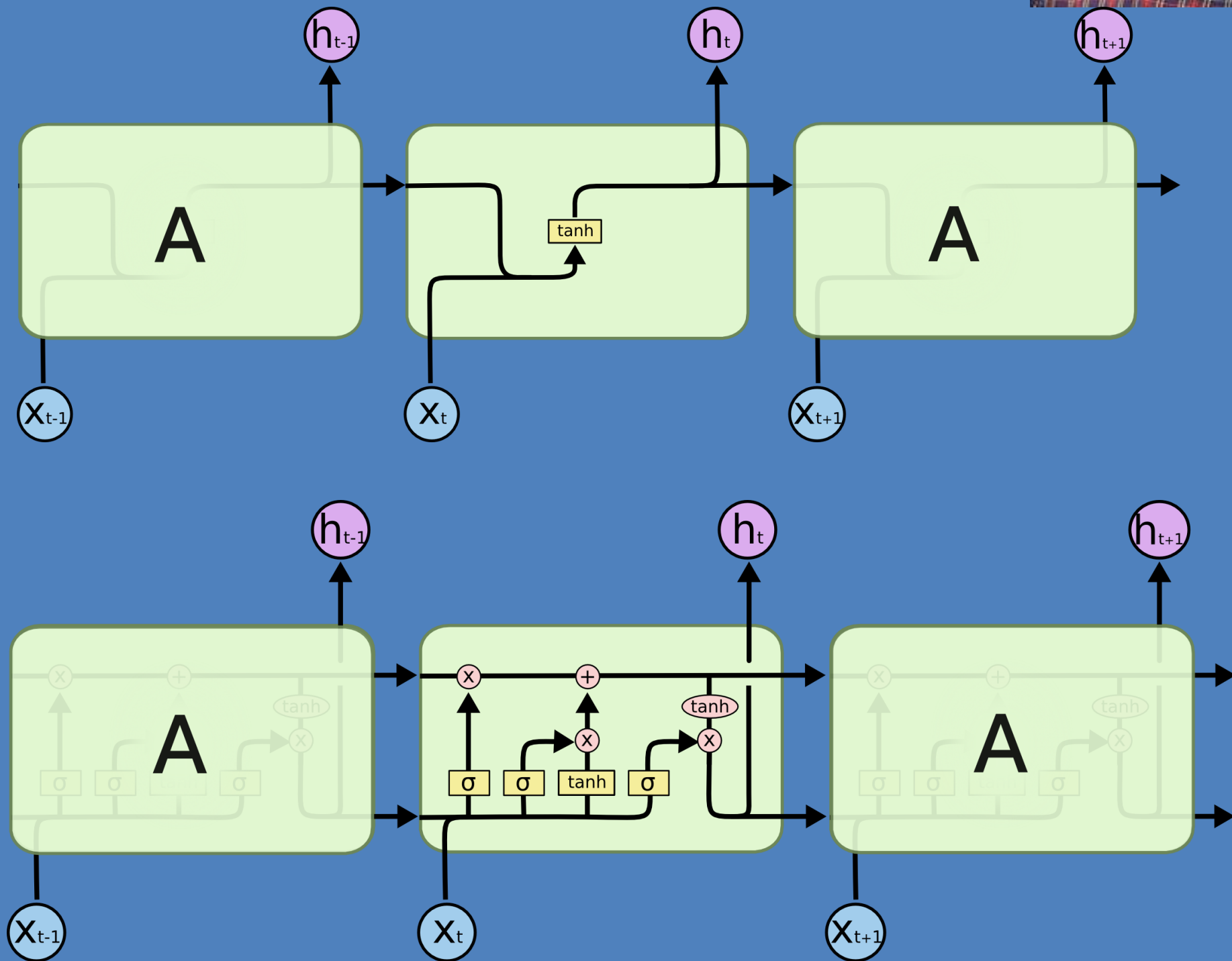


# Sequence Learning





Sepp Hochreiter  
1991發表原型

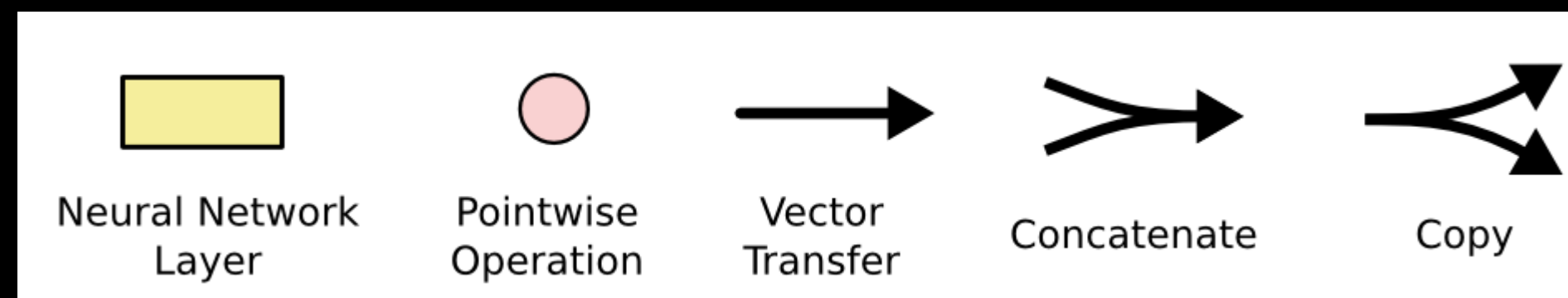
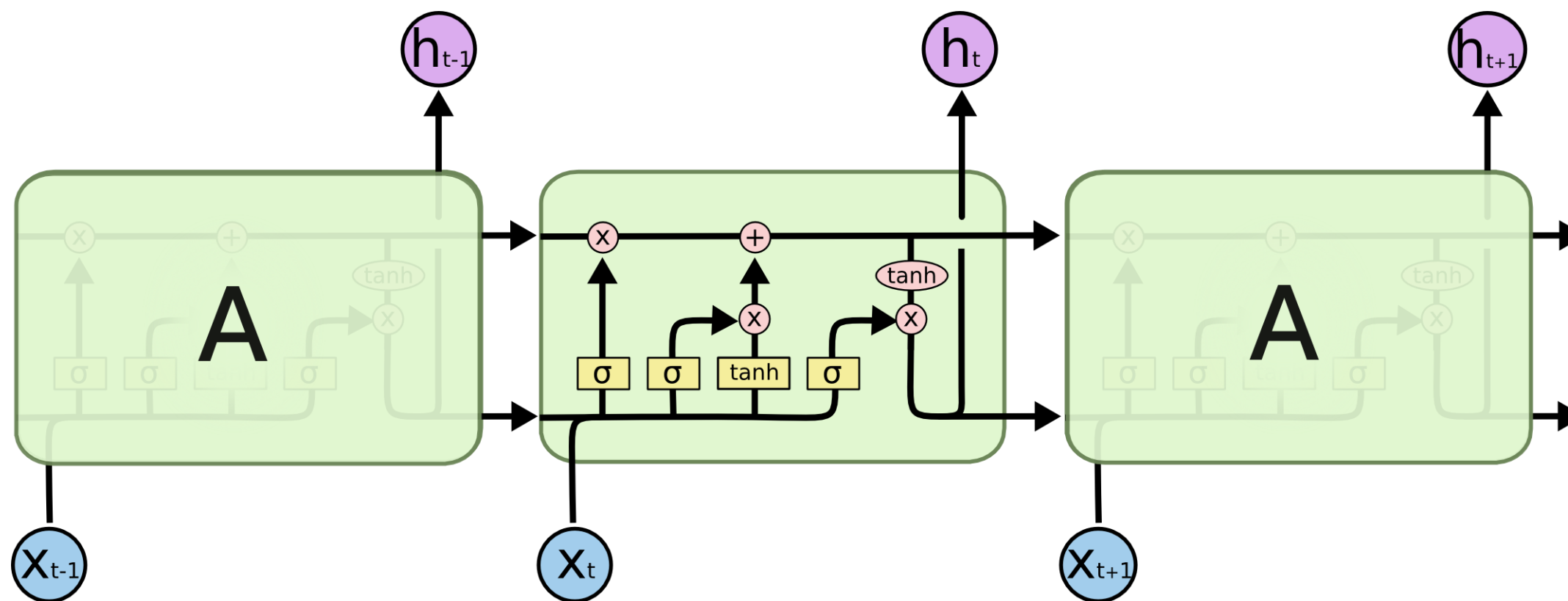


Jürgen Schmidhuber  
1997理論大功告成



# 長短記憶模型

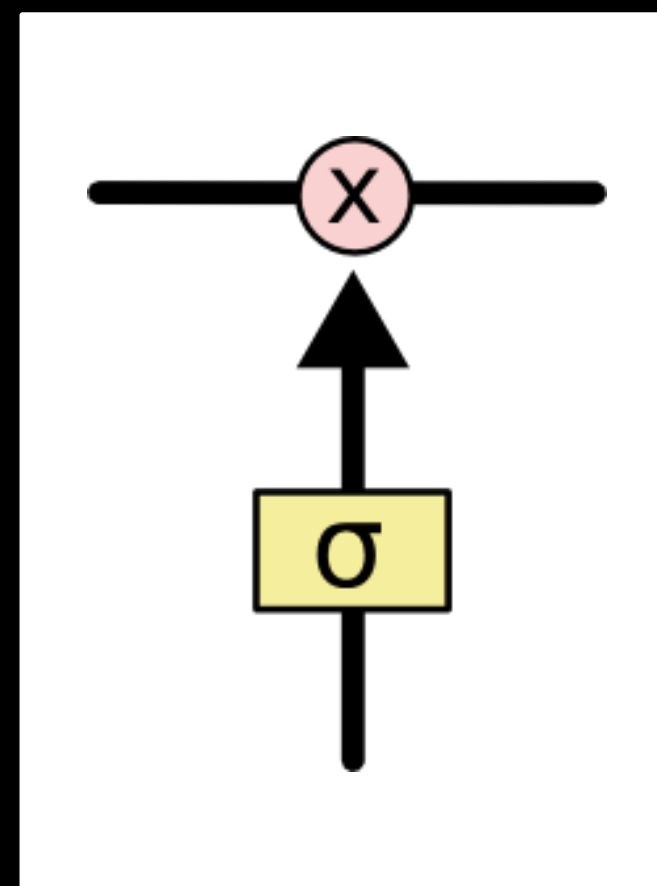
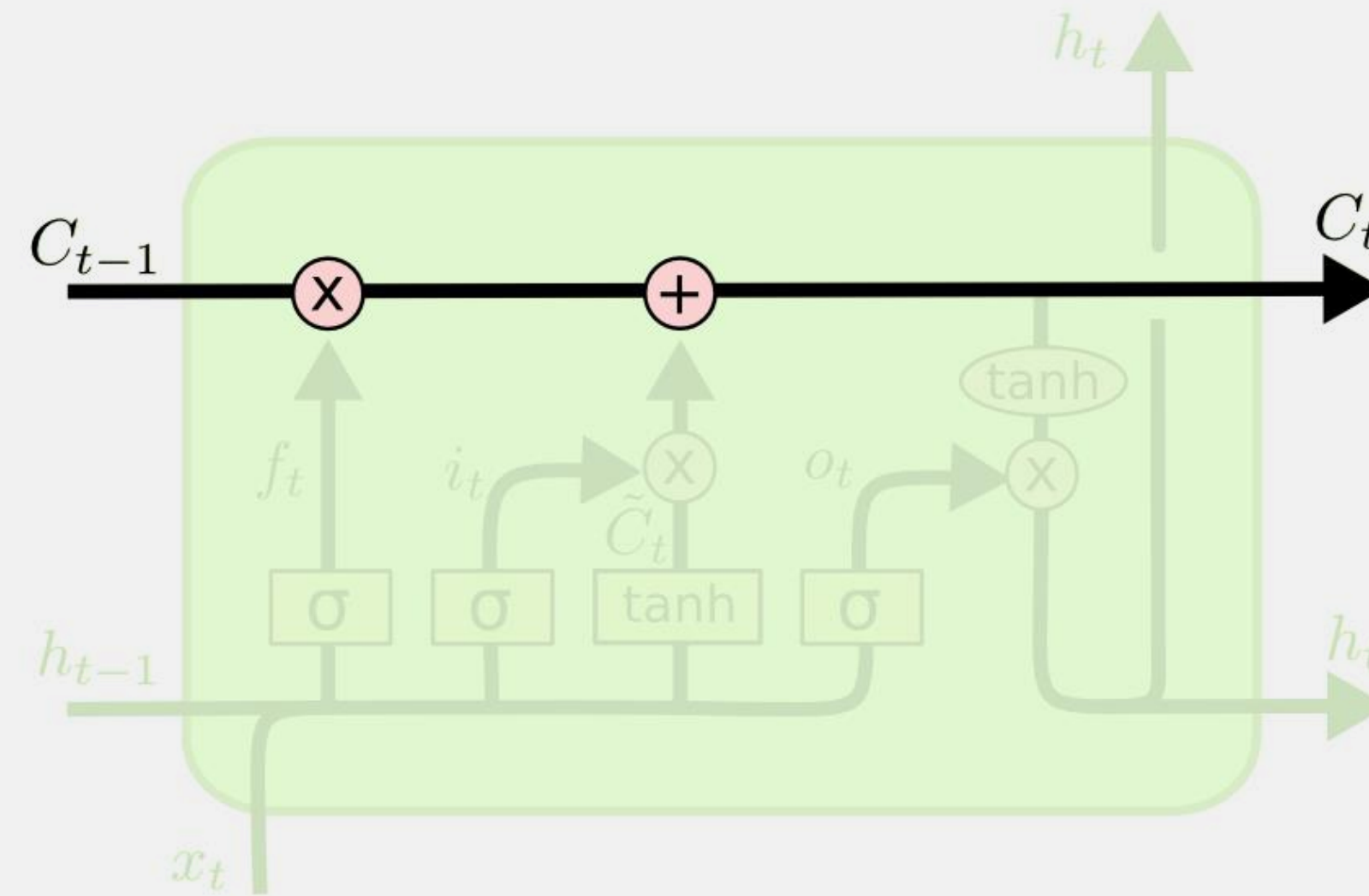
LSTM, Long Short Term Memory



神經層 單點計算 向量流動 匯流 複製

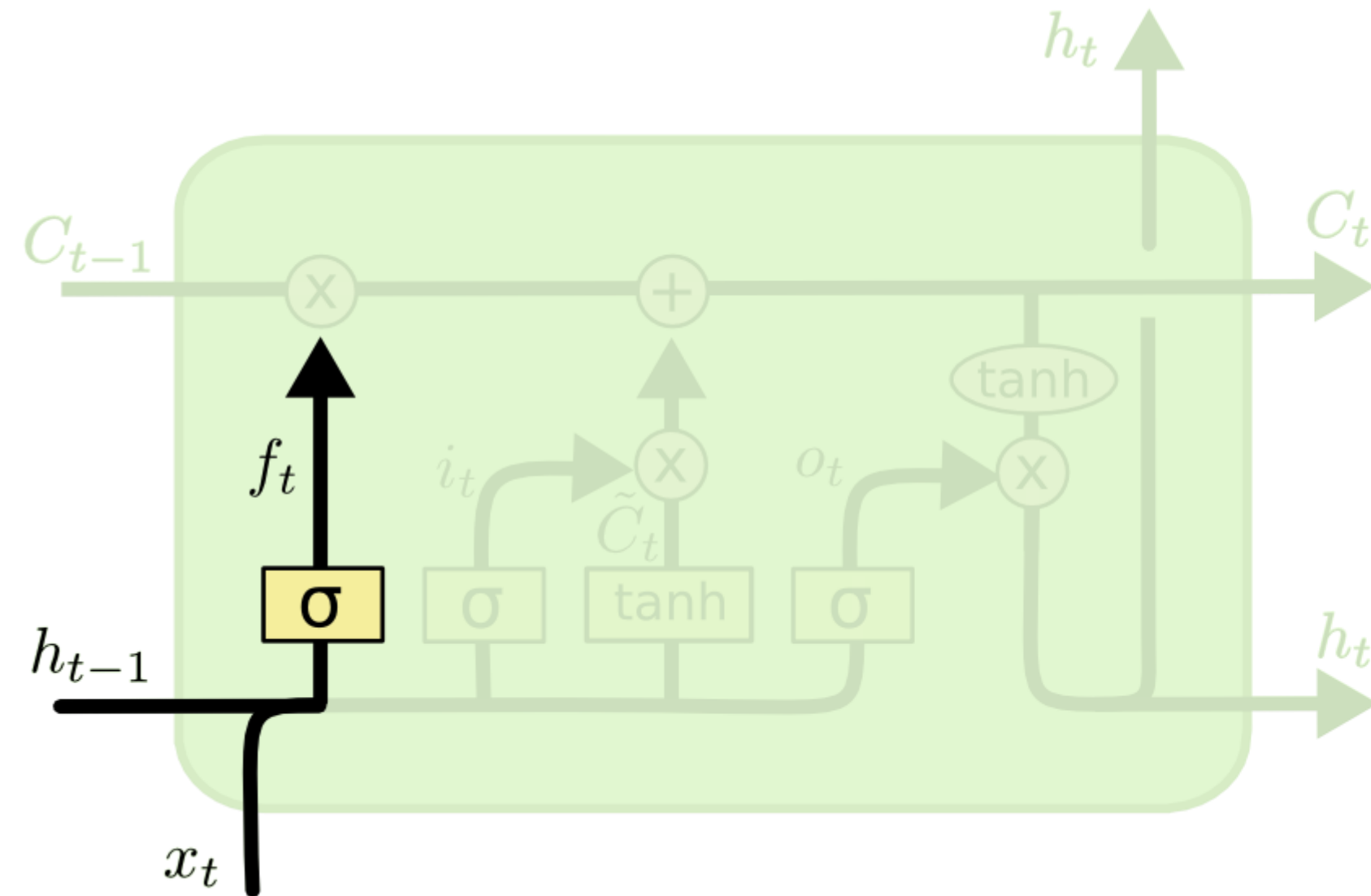


直接流過，沒有任何變化  
長期記憶



如何讓連續可微的函數能夠模擬不連續??

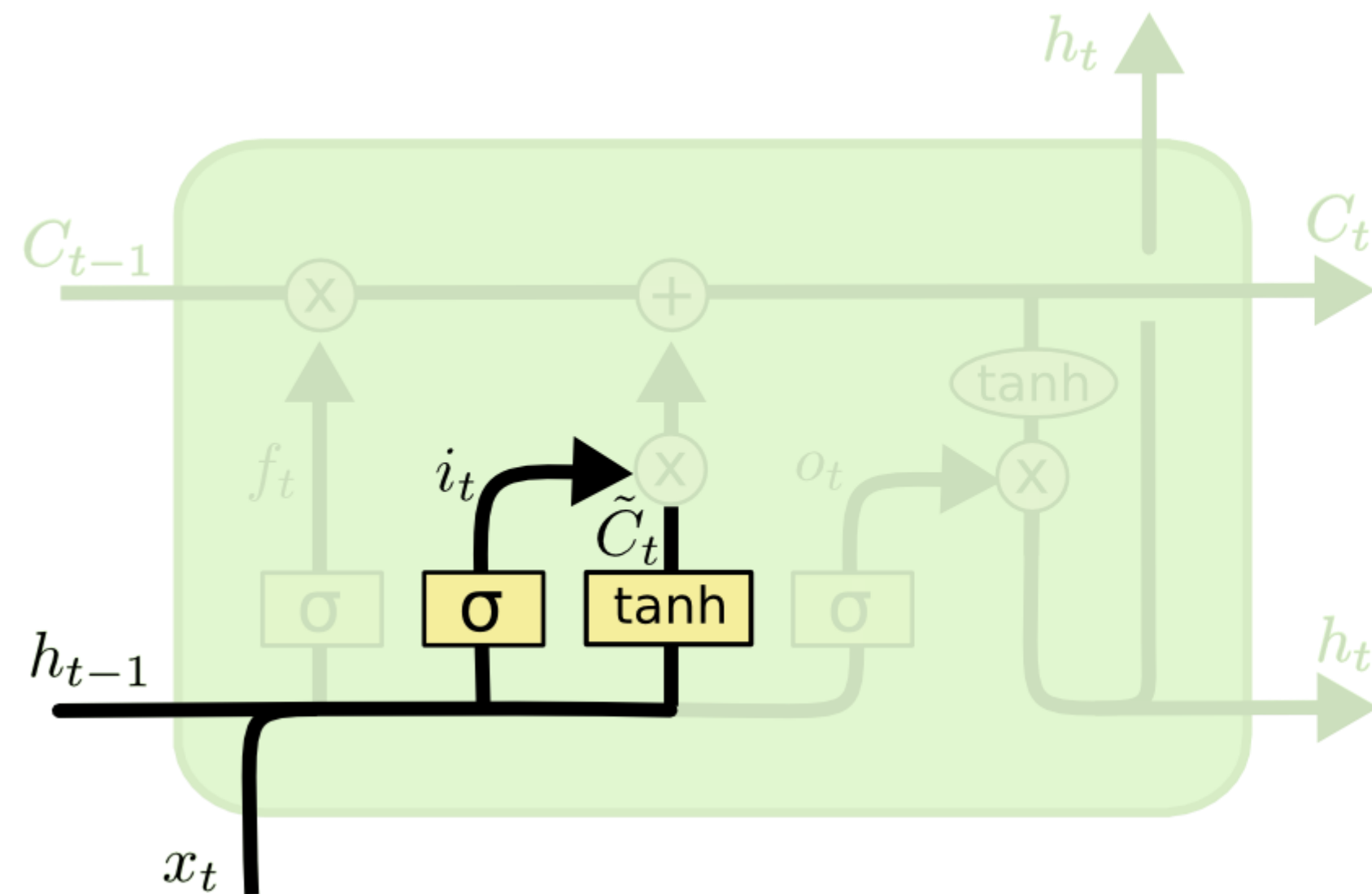




忘記門  
所獲取的資訊中那些該忘掉

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

遺忘向量



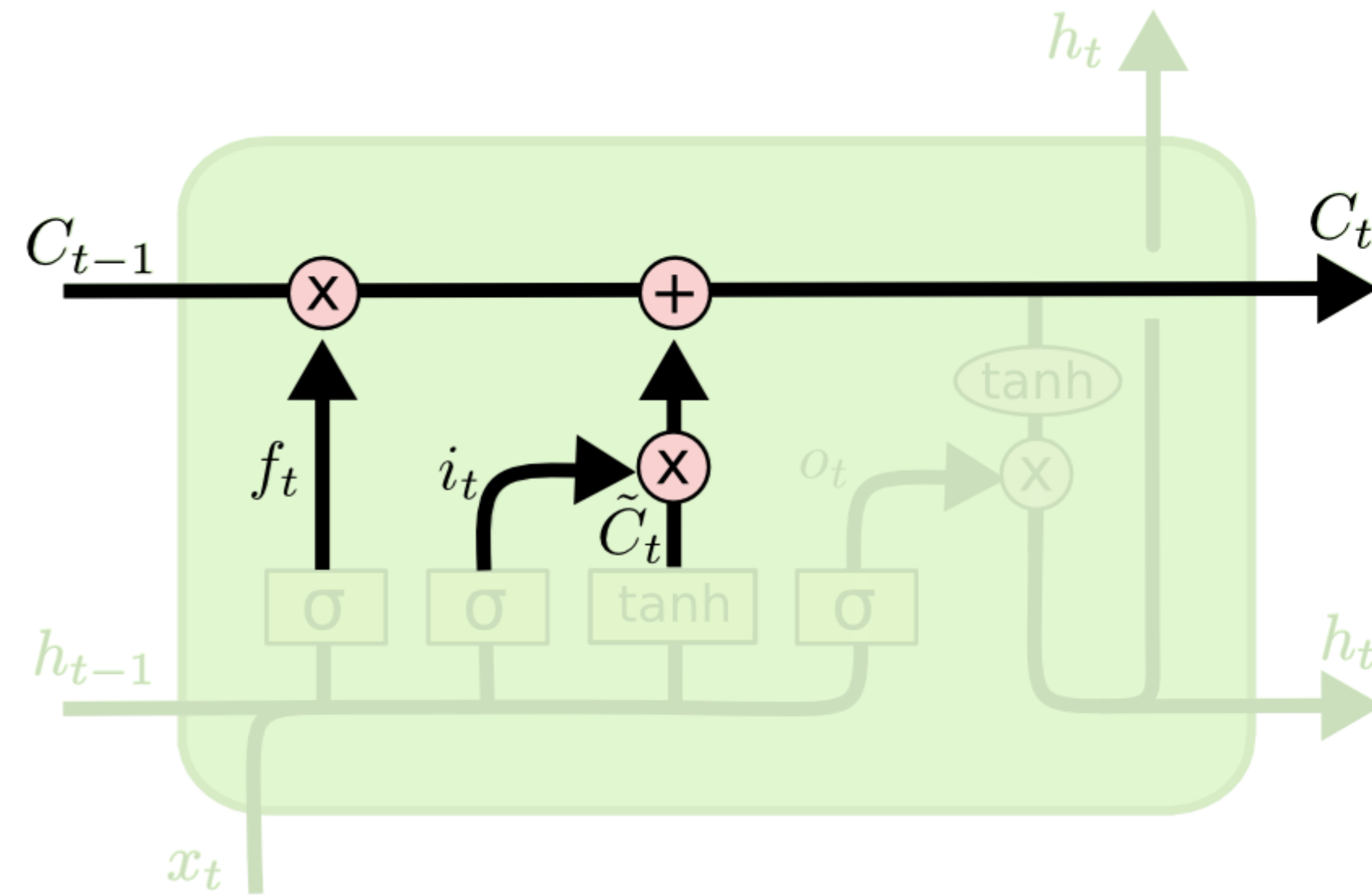
輸入門  
那些資訊應該以及如何被更新

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

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

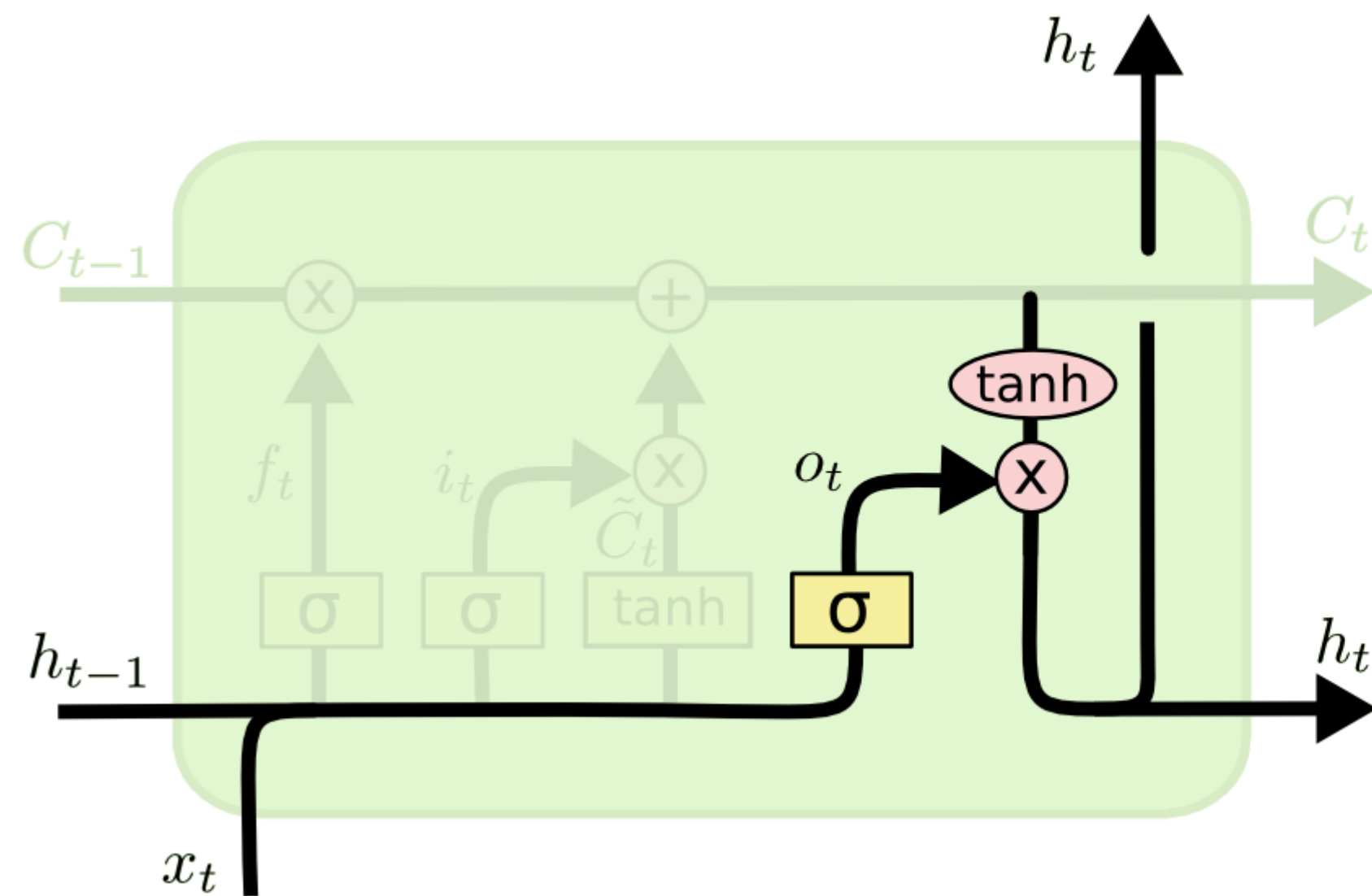


新舊狀態更迭



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

輸出門



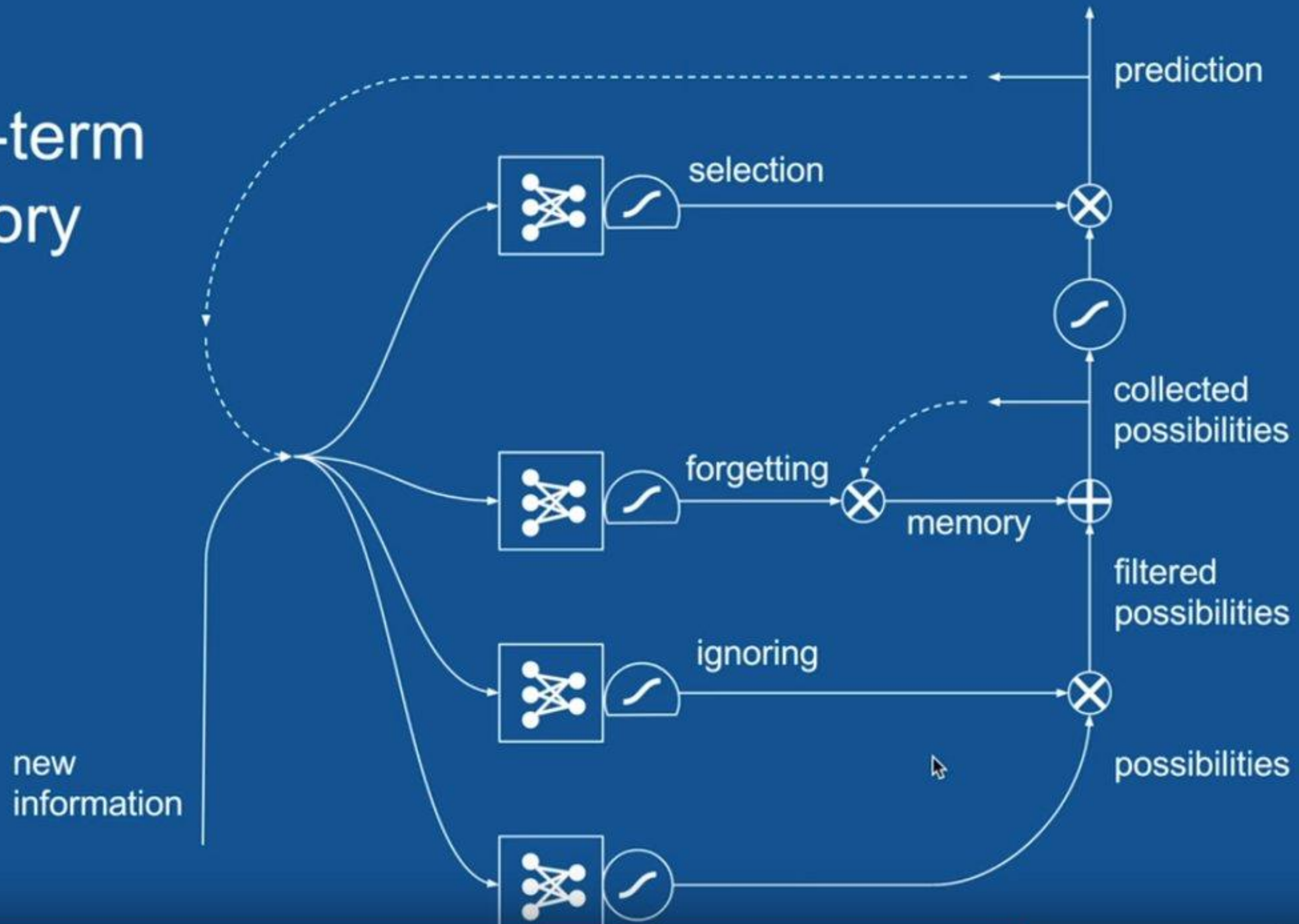
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

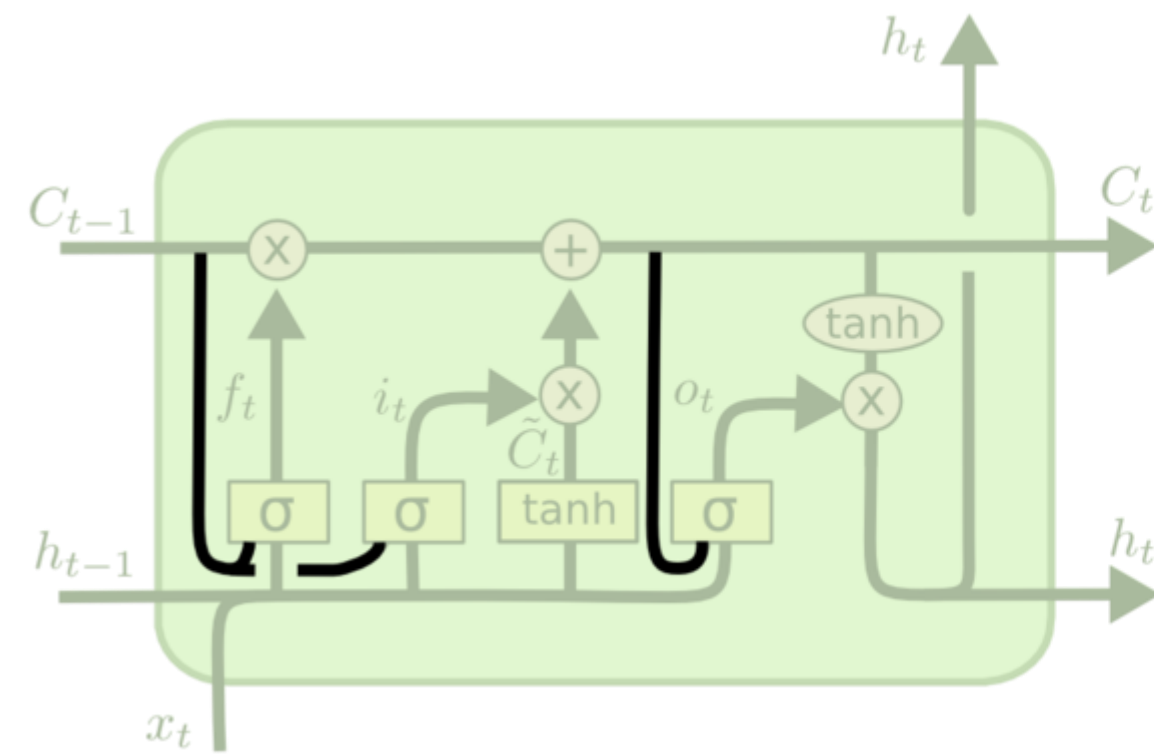
工作記憶



# long short-term memory



# 長短記憶模型的變體: Peephole



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

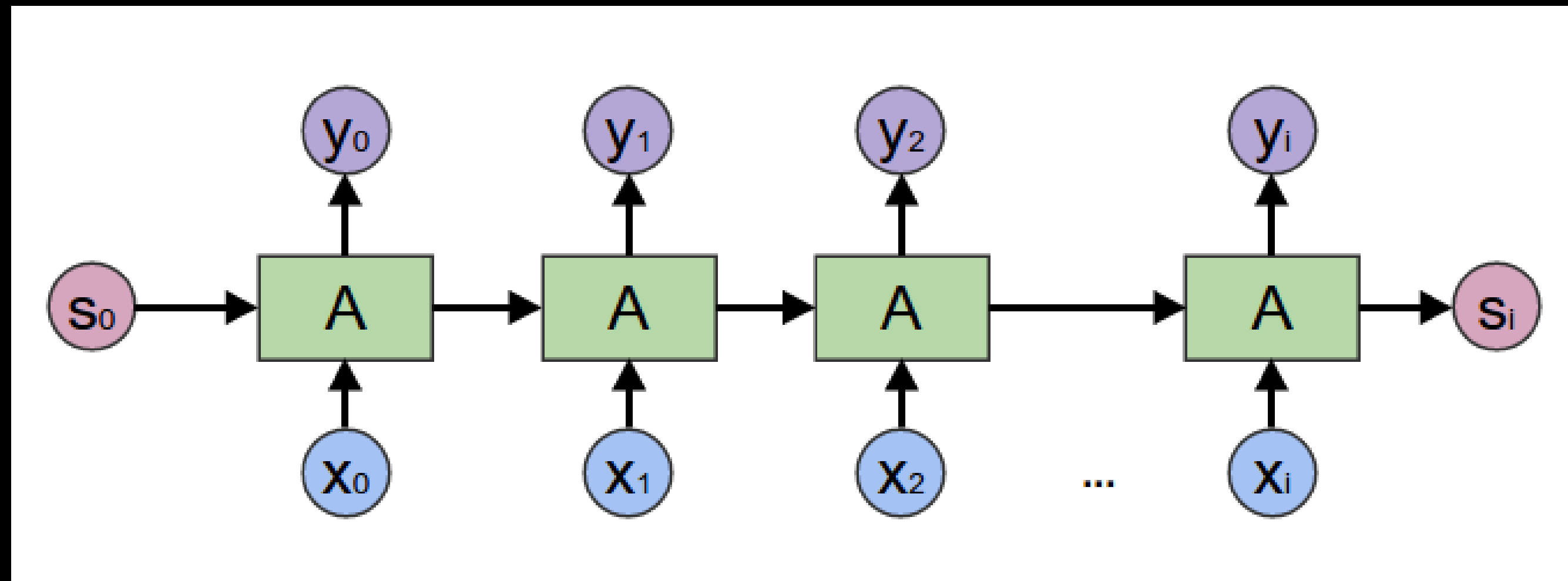
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

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



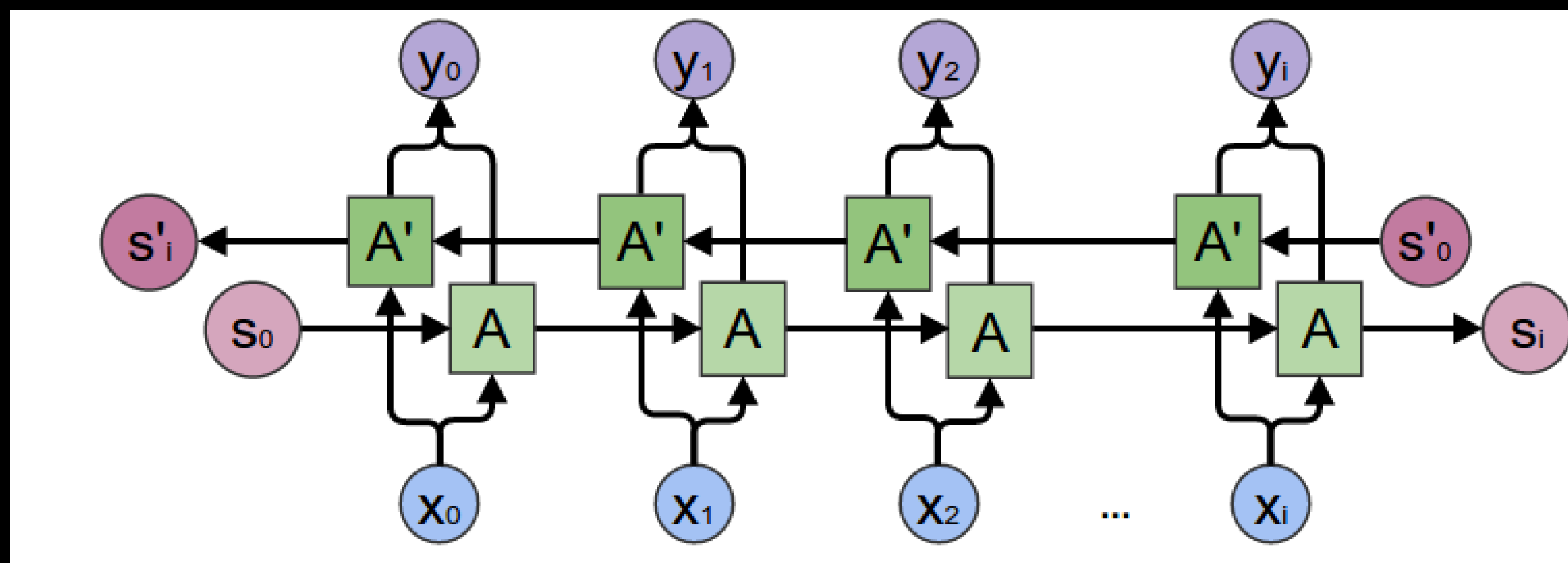


單層LSTM



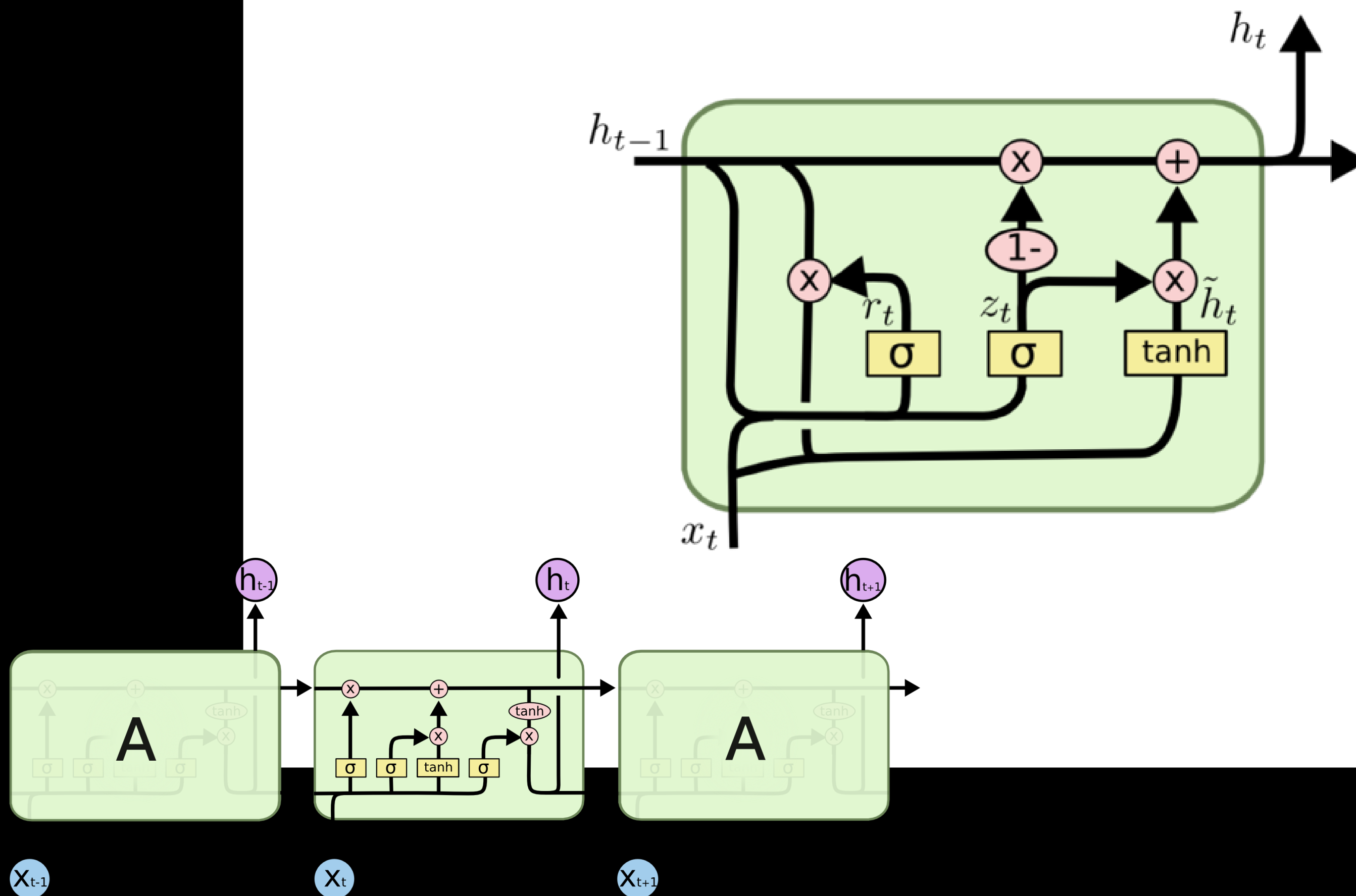
雙向LSTM

雙向LSTM等於是將一個句子正向輸入與反向輸入  
然後將兩個輸出向量concat在一起



# 長短記憶模型的變體: Gated Recurrent Unit (GRU)

忘記門與輸入門整合為更新門



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

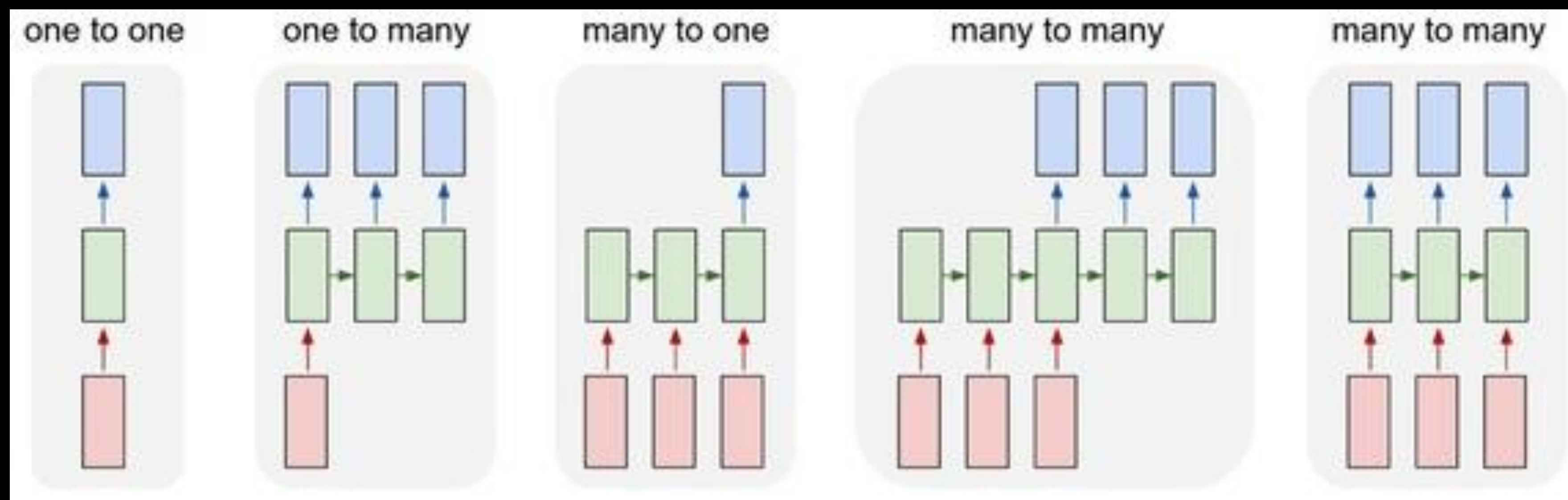
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



# 序列對序列的關係

## seq2seq



詞性標註

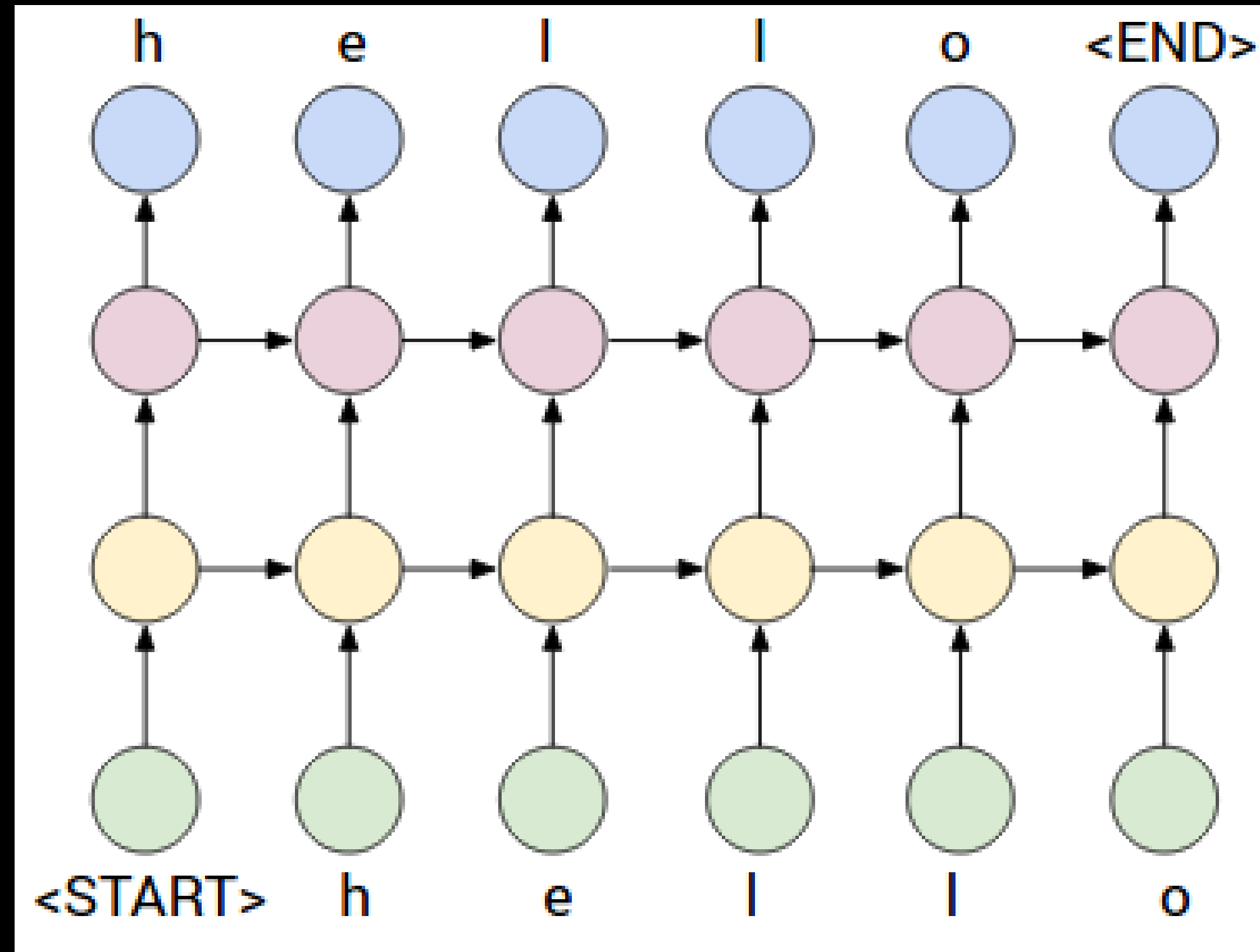
自動歌詞

序列分類

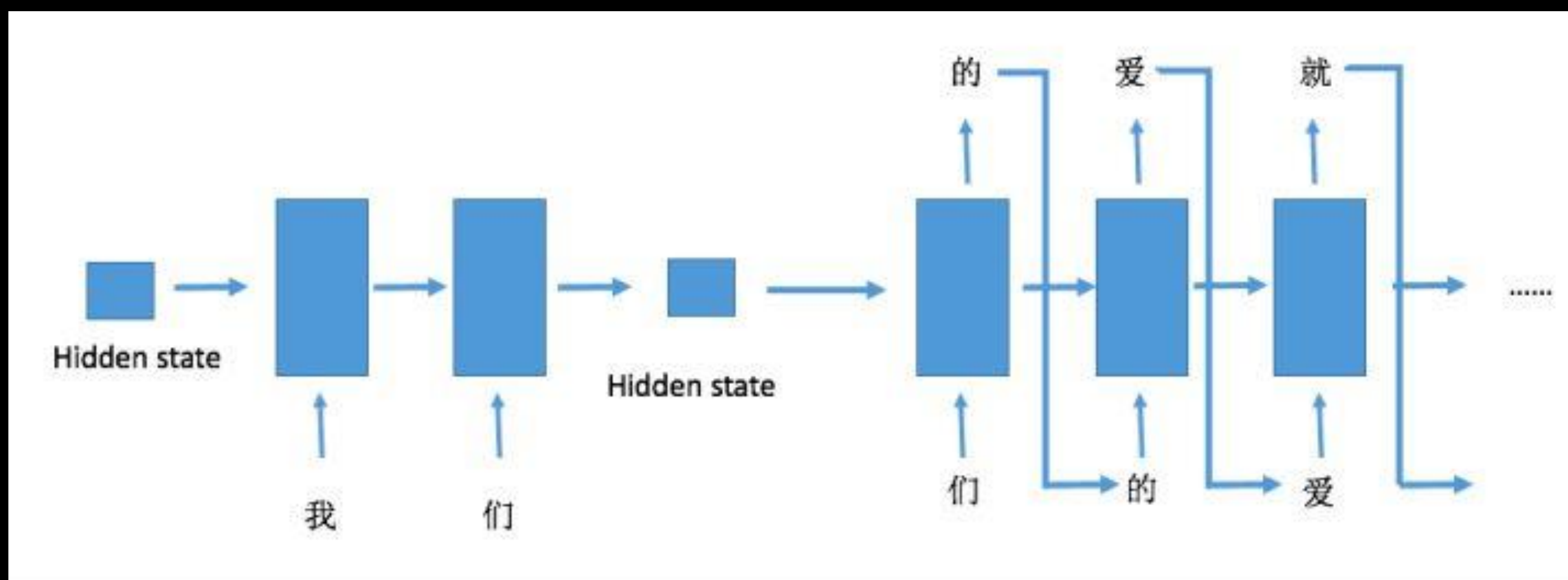
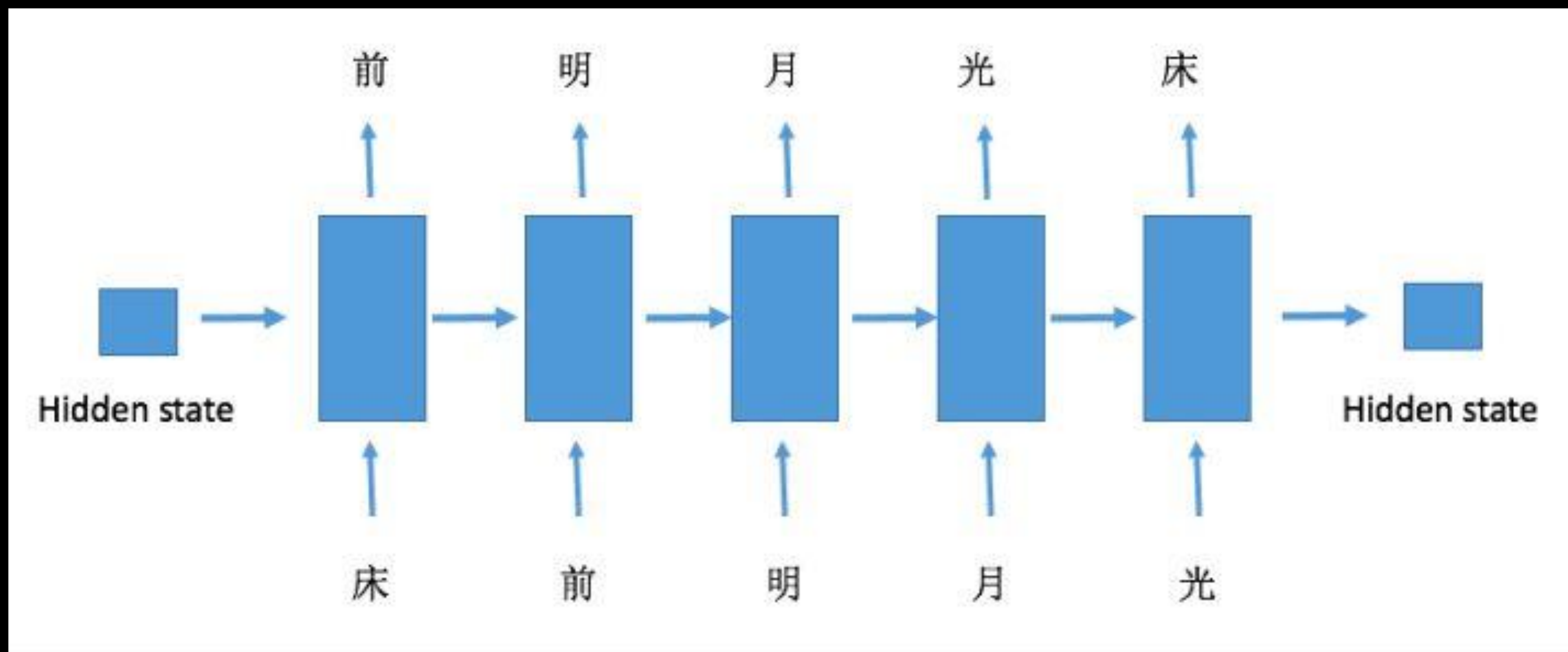
自動翻譯  
閱讀理解  
新聞摘要

# Char-RNN

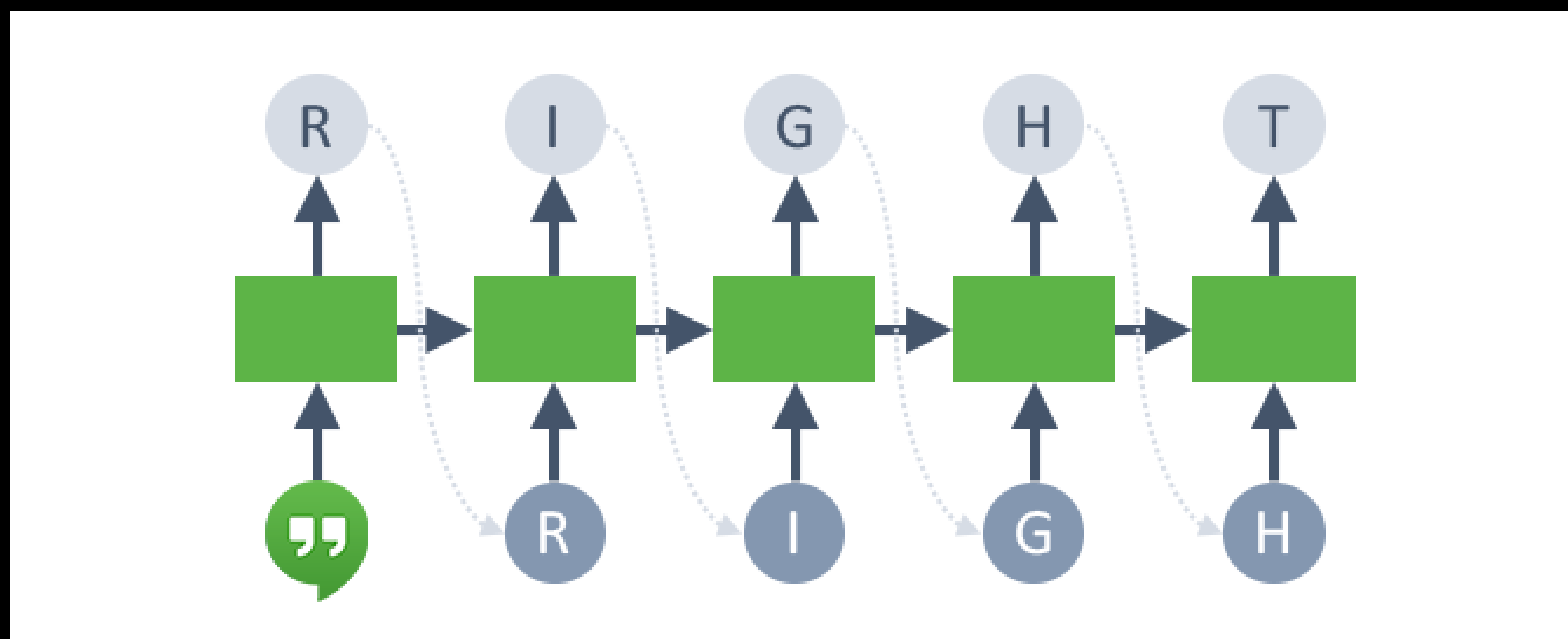
最簡單的1-to-1 Seq2Seq







天 空 很 藍 沒 有 雲  
↓ ↓ ↓ ↓ ↓ ↓ ↓  
上 很 藍 色 有 的 的



Teacher Forcing

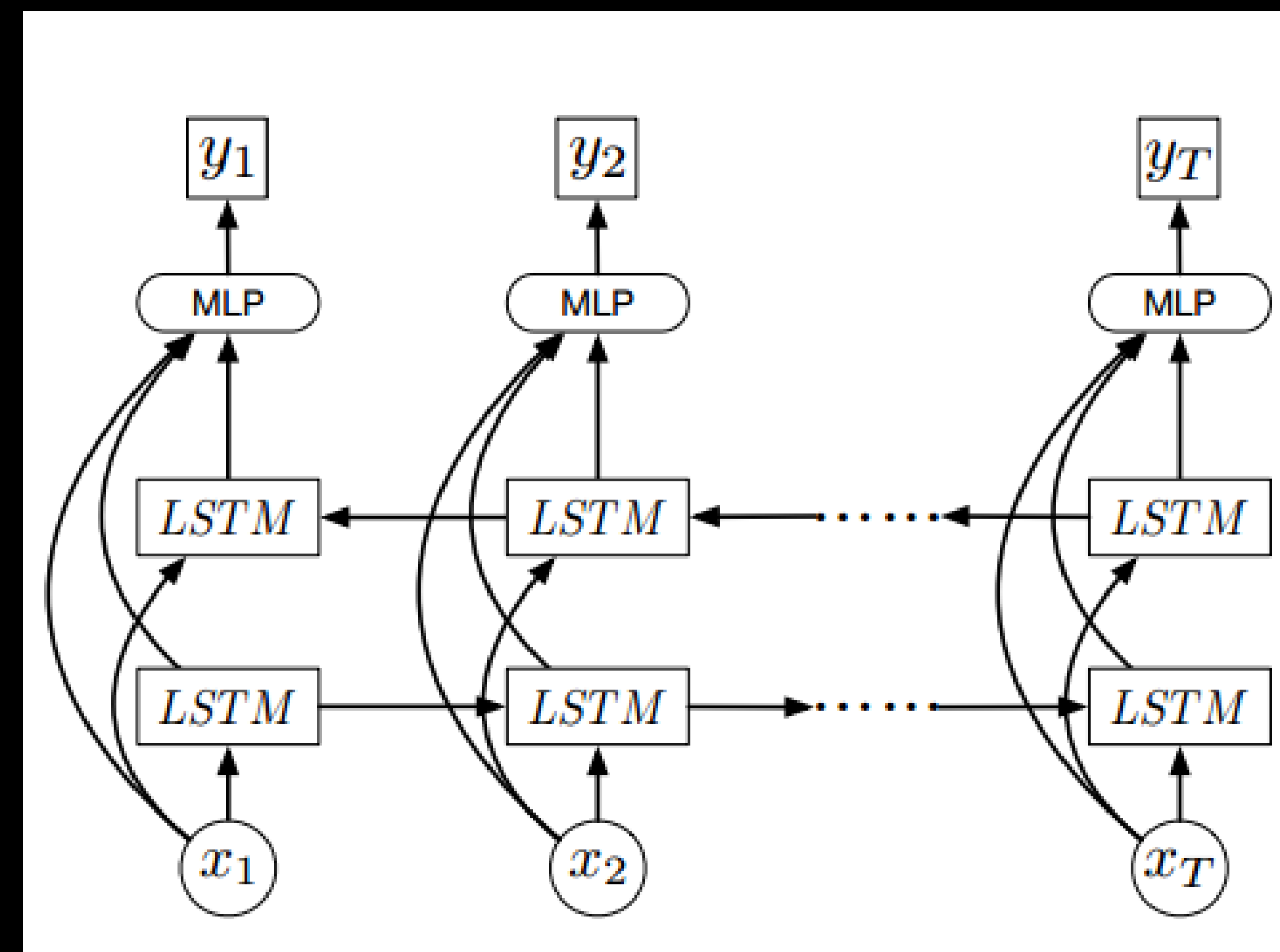


嵌入層

LSTM層stack

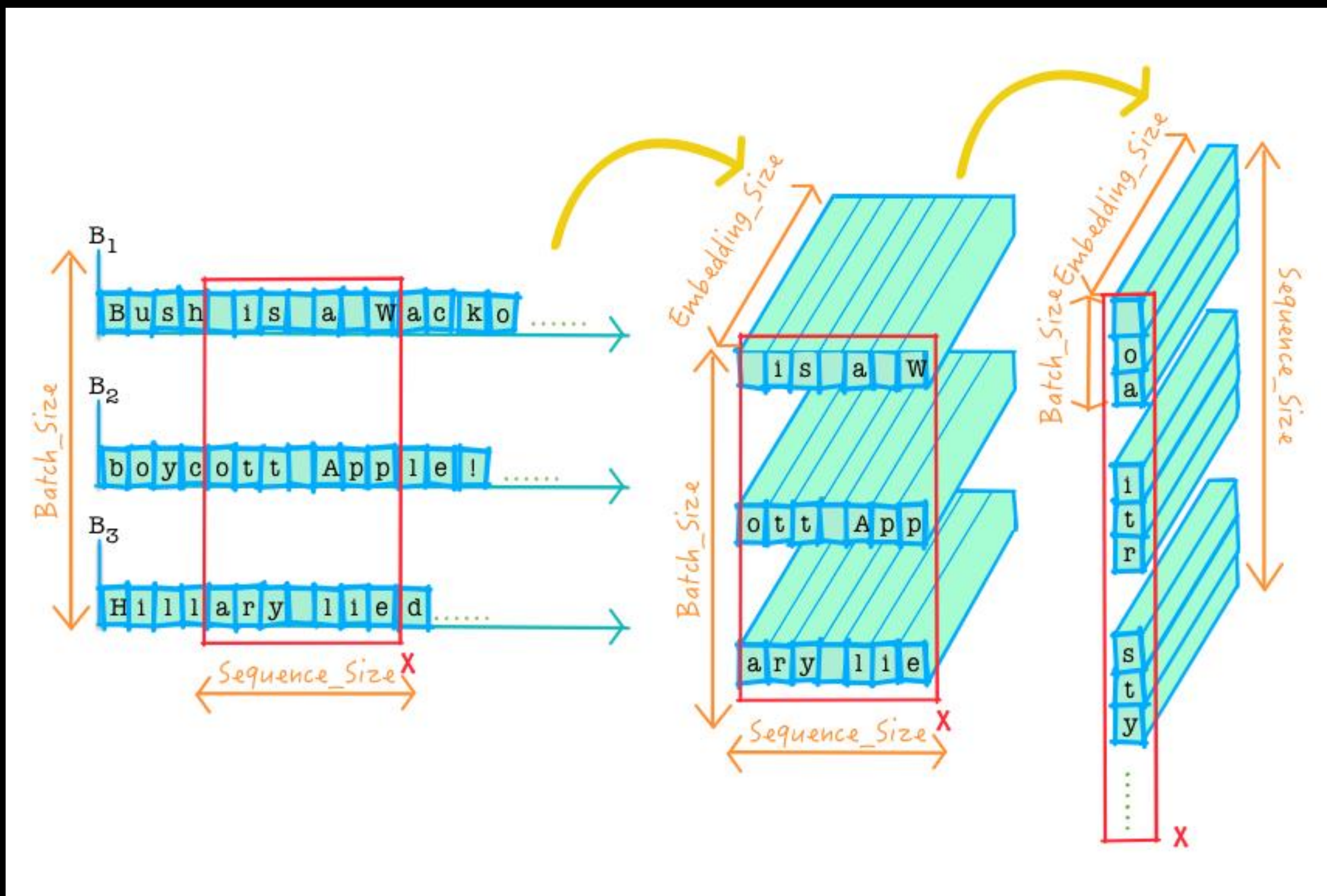
Dropout

Dense



# 嵌入層

Embedding( embedding\_size)





# 生成模型常見的困擾

## 如何解決柿子挑軟的吃的問題

```
Minibatch[ 341- 360]: loss = 5.936178 * 4515, metric = 90.92% * 4515;
Minibatch[ 361- 380]: loss = 5.850860 * 3881, metric = 89.51% * 3881;
Minibatch[ 381- 400]: loss = 5.821560 * 3832, metric = 88.99% * 3832;
【這句歌詞我會寫成...】
(原句:)想勸你 半邊天 多美滿

【用「天」發想歌詞...】
=>天你
Minibatch[ 401- 420]: loss = 5.882825 * 4265, metric = 90.95% * 4265;
Minibatch[ 421- 440]: loss = 5.914336 * 3745, metric = 89.29% * 3745;
Minibatch[ 441- 460]: loss = 6.038729 * 3823, metric = 90.79% * 3823;
Minibatch[ 461- 480]: loss = 5.975112 * 4475, metric = 90.61% * 4475;
Minibatch[ 481- 500]: loss = 5.884597 * 3927, metric = 90.04% * 3927;
【這句歌詞我會寫成...】
(原句:)如此一心一意 你未看起

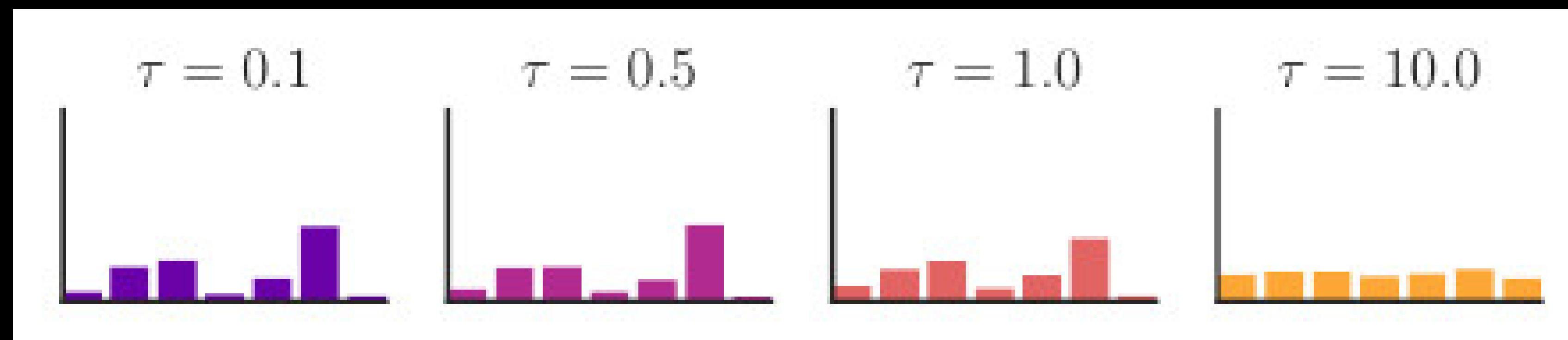
【用「夢」發想歌詞...】
=>夢你你你的
Minibatch[ 501- 520]: loss = 5.779777 * 3797, metric = 88.20% * 3797;
Minibatch[ 521- 540]: loss = 5.914313 * 4186, metric = 89.56% * 4186;
Minibatch[ 541- 560]: loss = 5.761072 * 4290, metric = 90.16% * 4290;
Minibatch[ 561- 580]: loss = 5.879596 * 4216, metric = 88.92% * 4216;
Minibatch[ 581- 600]: loss = 5.805179 * 4311, metric = 89.72% * 4311;
【這句歌詞我會寫成...】
(原句:)曾為你怎麼跌倒
```

模型尚未進入深層規則  
詞頻差異過大  
贏者全拿

# 改變文字的溫度

```
def apply_temp(p):  
    # apply temperature  
    p = np.power(p, (temperature))  
    # renormalize and return  
    return (p / np.sum(p))
```

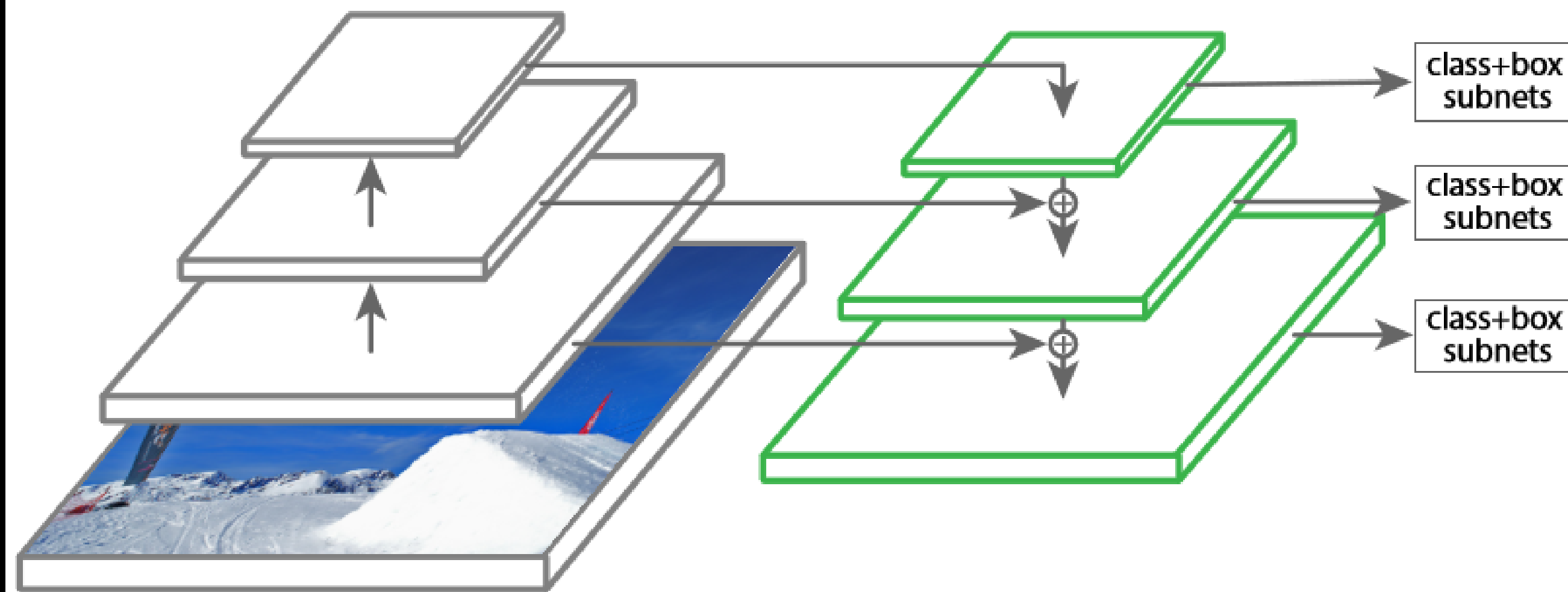
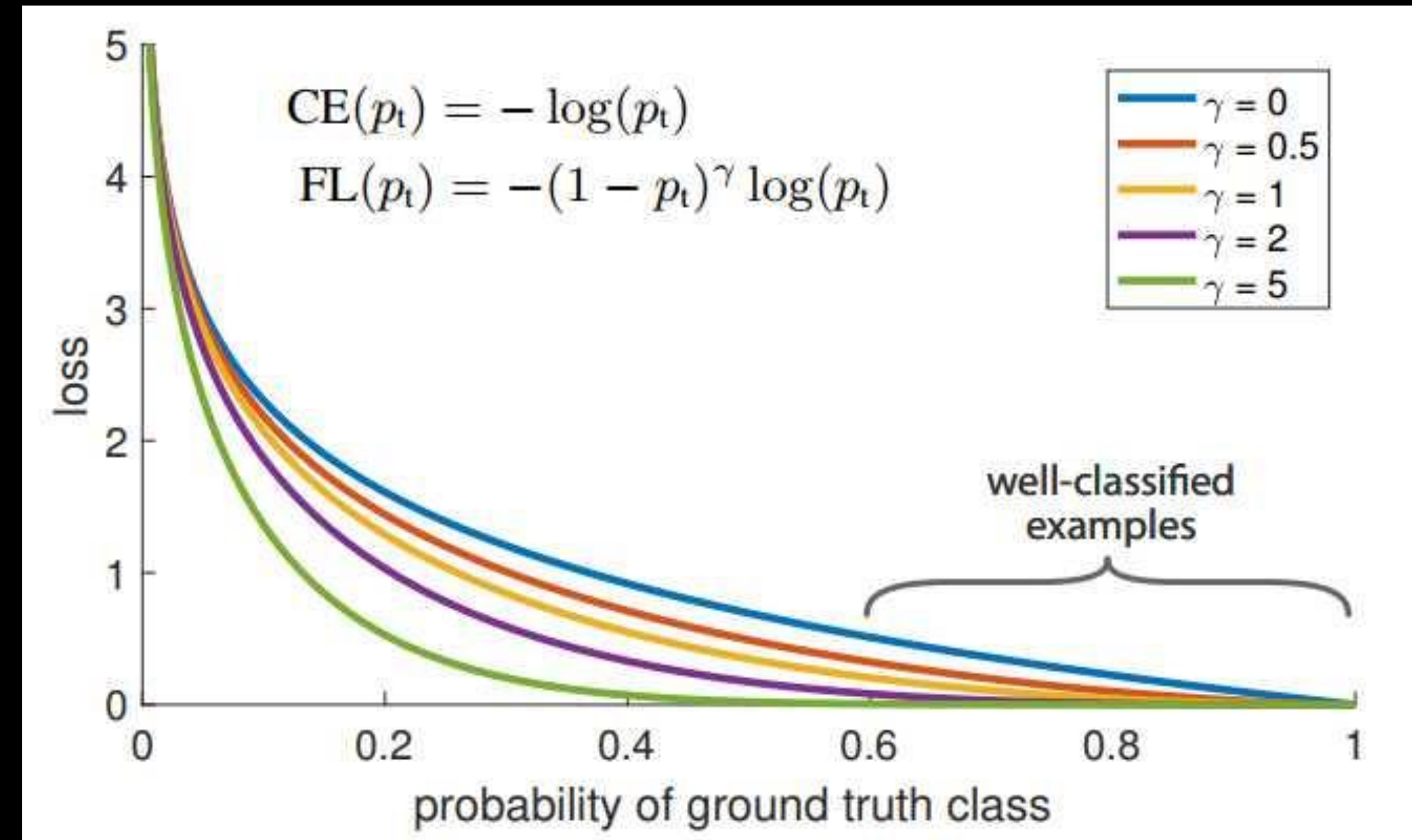
等於1時等於argmax  
等於無窮大時接近均等分佈





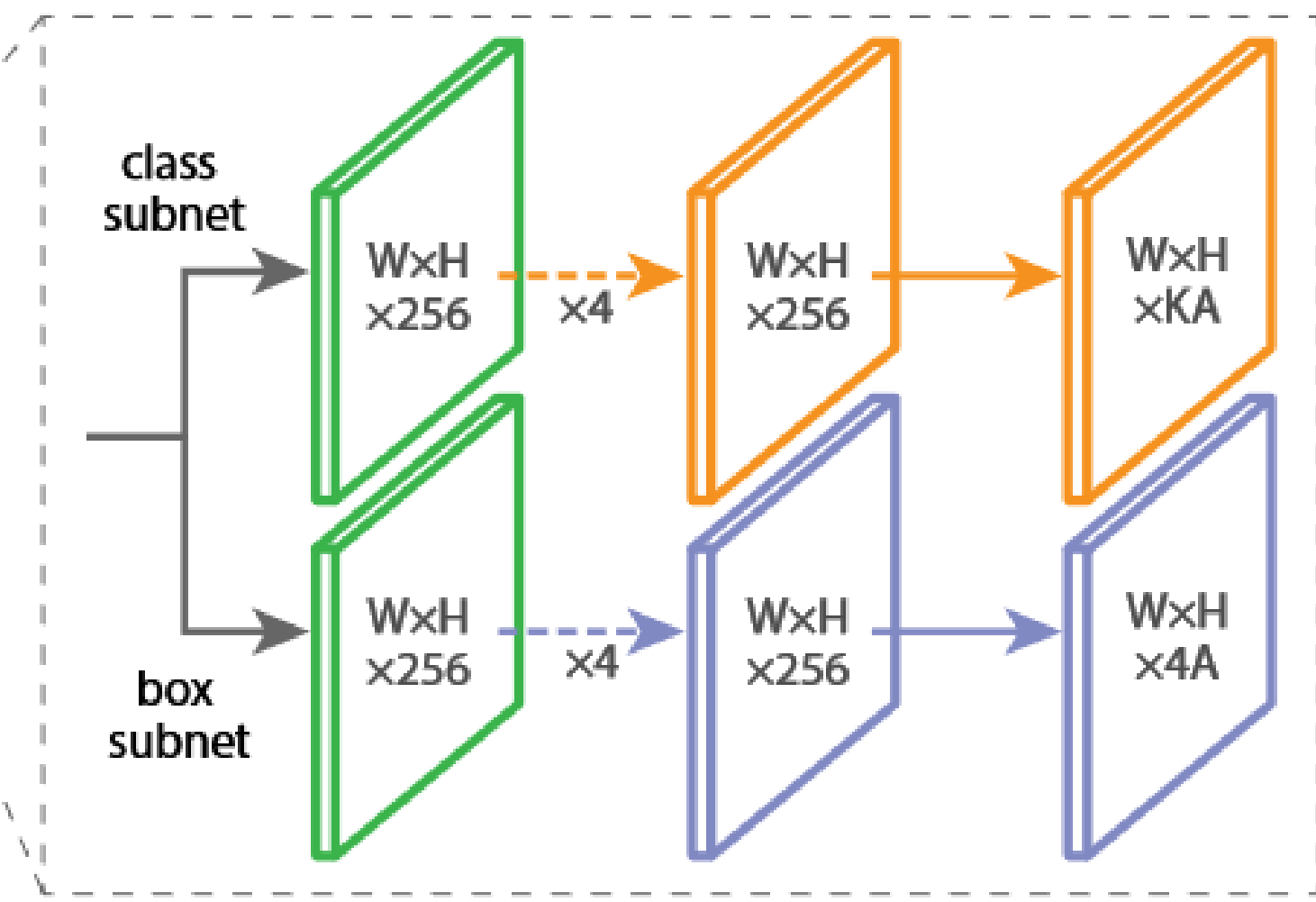
# Focal Loss

解決深度學習柿子挑軟的吃，造成中庸的準確率的問題



(a) ResNet

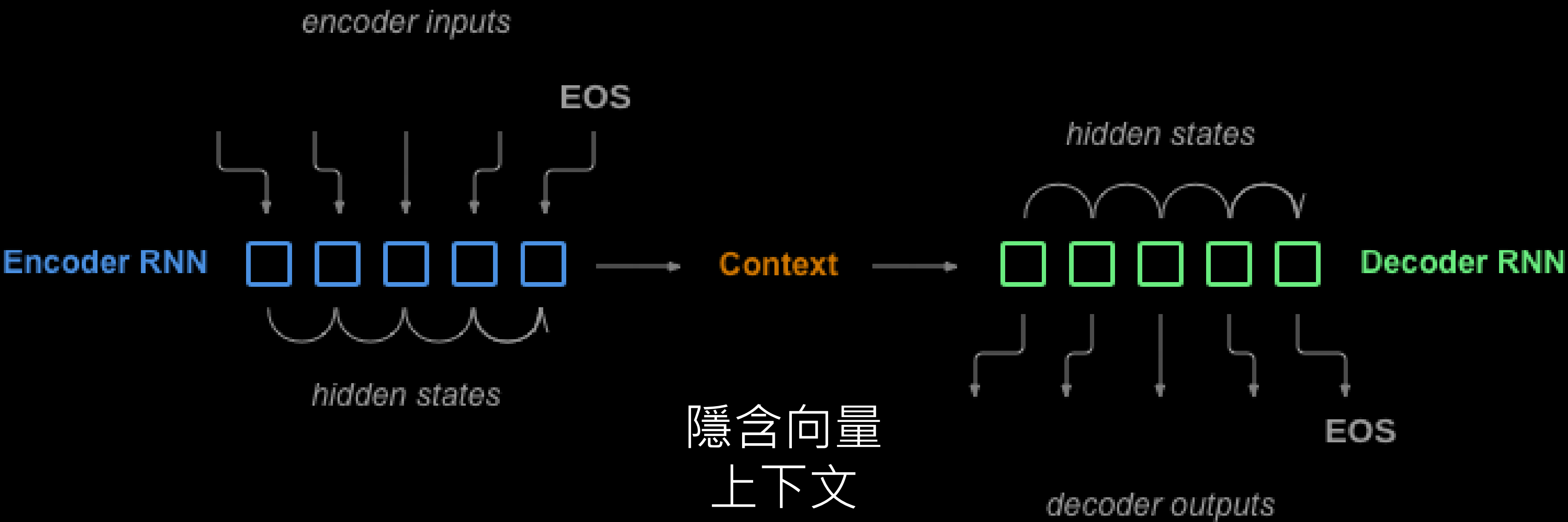
(b) feature pyramid net



(c) class subnet (top)

(d) box subnet (bottom)

將輸入編碼

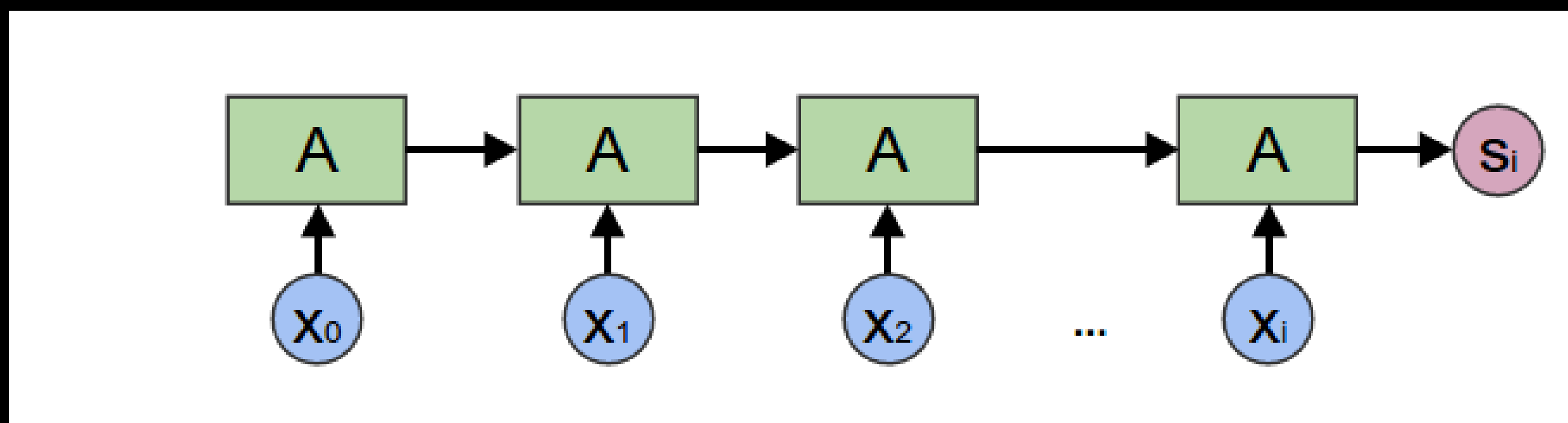


隱含向量  
上下文

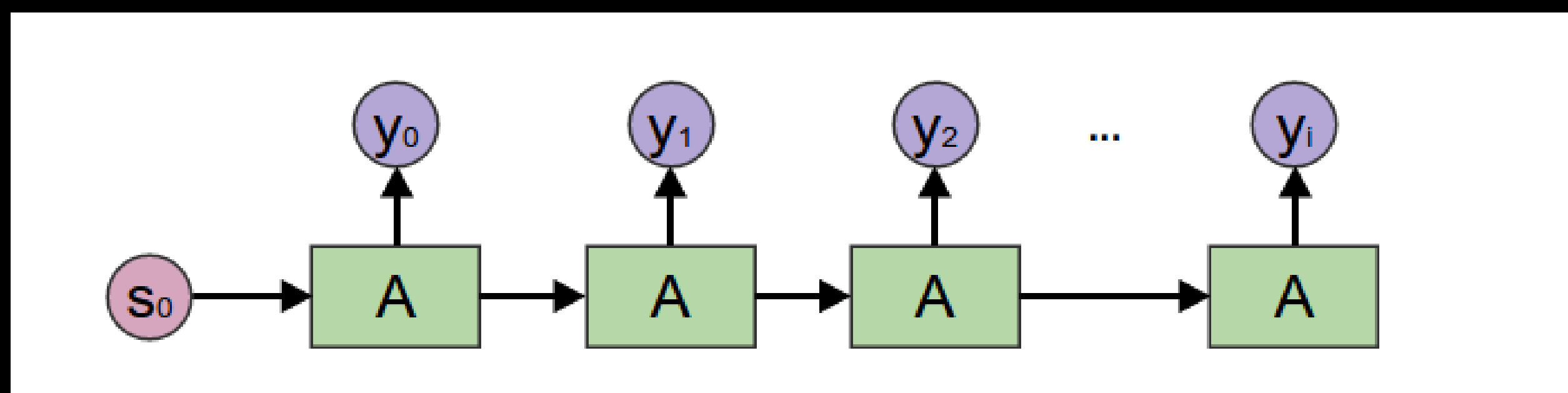
解碼為輸出



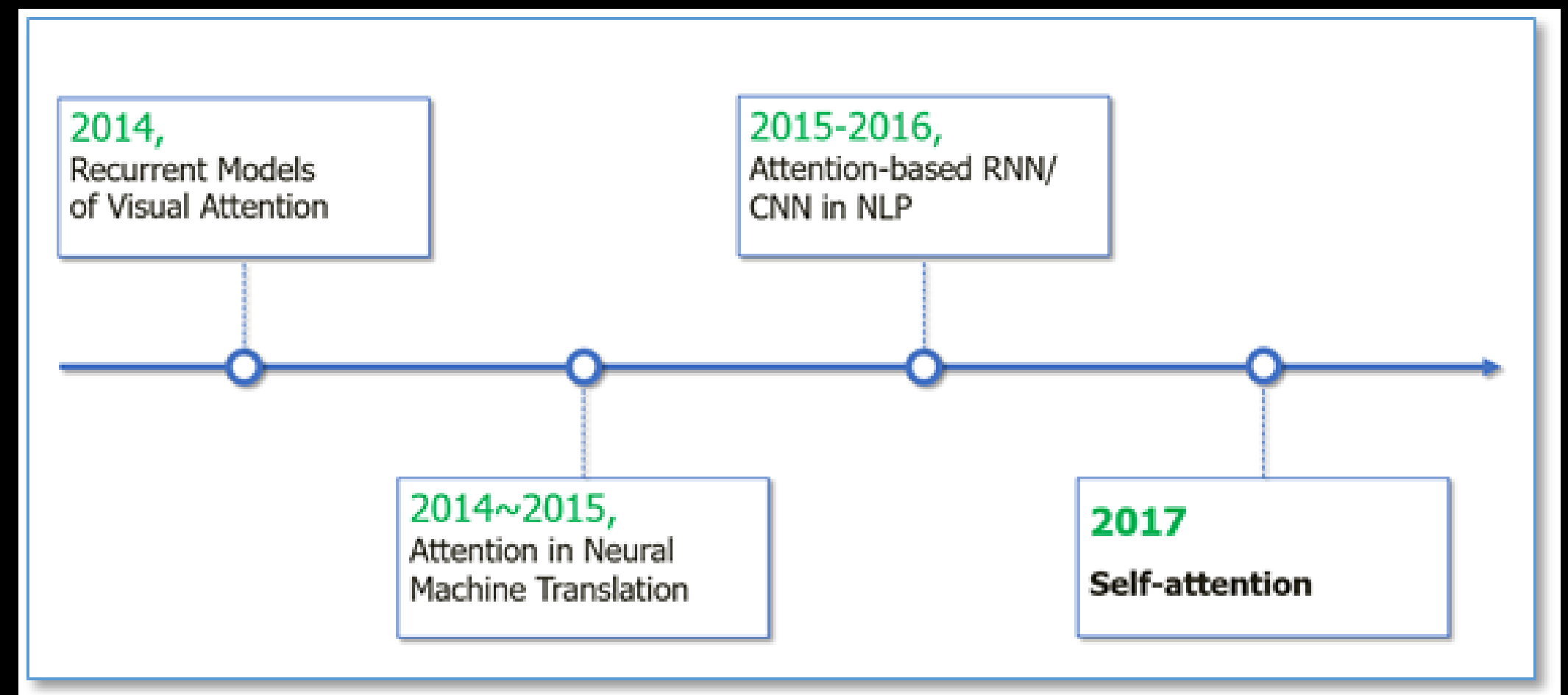
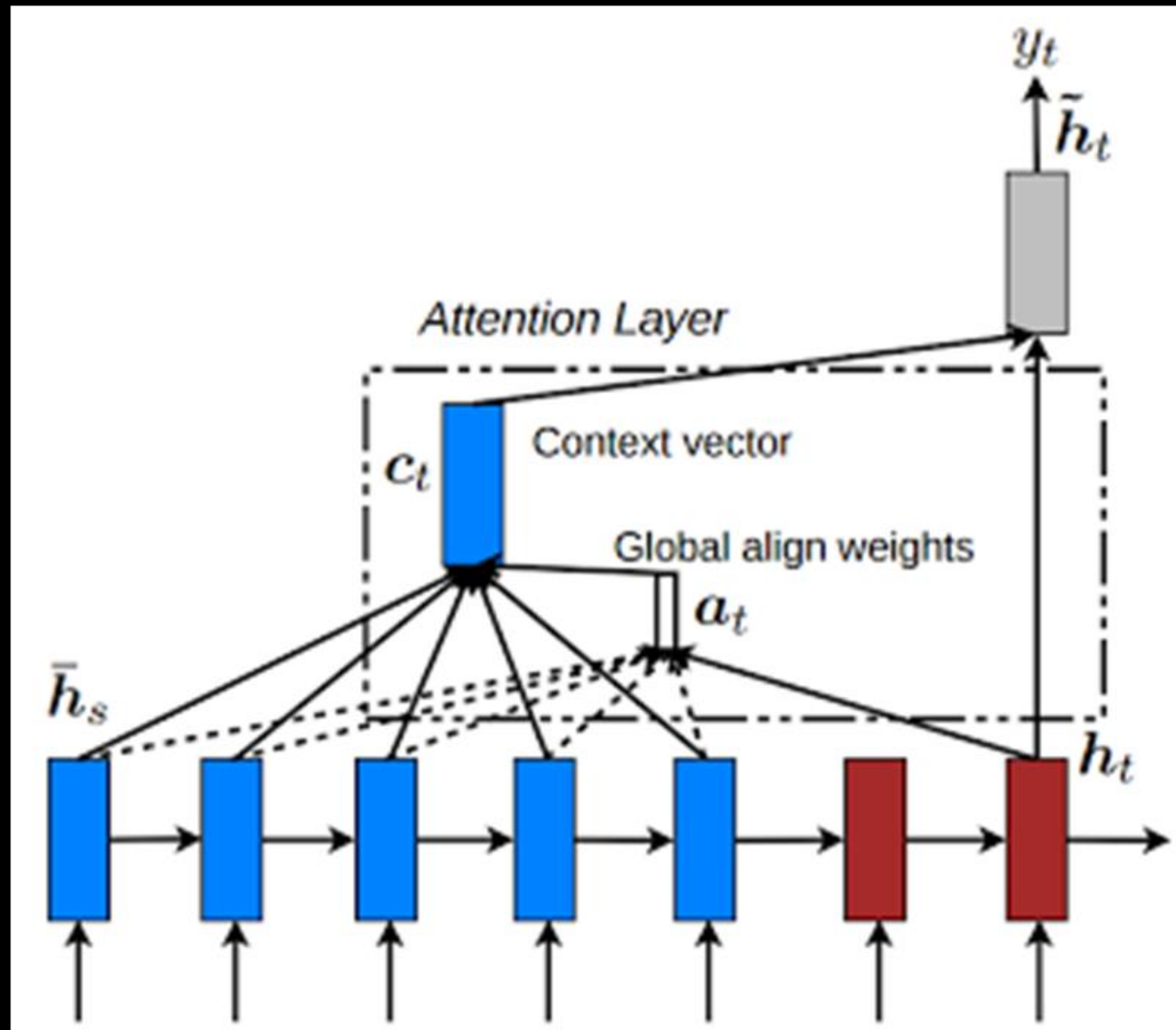
fold = RNN 編碼器



unfold = RNN 生成器



# 注意力機制 attention







A(0.98)



person(0.38)



is(0.38)



standing(0.28)



on(0.22)



a(0.26)



beach(0.32)



with(0.30)



a(0.20)



surfboard(0.33)



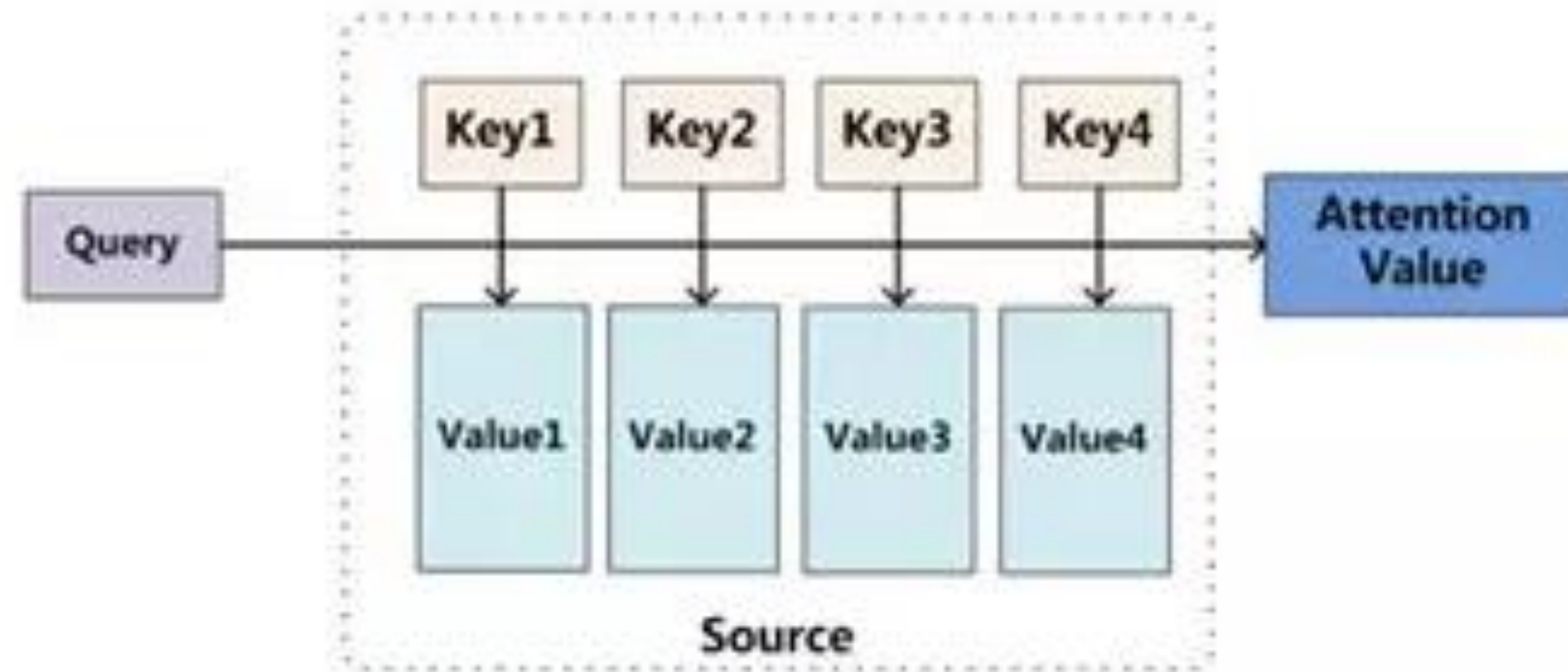
.(0.25)



(b) A person is standing on a beach with a surfboard.

注意力機制的啟發  
最早來自於視覺

注意力機制的原型可以視為對於對一系列key-value的查詢(query)  
查詢的結果，可以被視為模型的重要性權重



$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{L_s} \text{Similarity}(\text{Query}, \text{Key}_i) * \text{Value}_i$$

1. 計算每個key的相似性或是重要性
2. 使用softmax歸一化
3. 透過query查詢權重後求和

向量的乘法有兩種，一種是點乘，指的是對應位置成員彼此相乘，另一種則是與矩陣乘法相同，又稱之為哈達馬積(hadamard product)。下圖的左方就是標準的點乘，所以第一行第一列的成員只需要乘上另一個向量對應的一行第一列即可( $1*3$ )。至於完整的矩陣乘法，則是向量的第一列乘上另一個向量的第一行( $1*3+2*4=11$ )

1	2
3	4

3	2
4	1

1	2
3	4

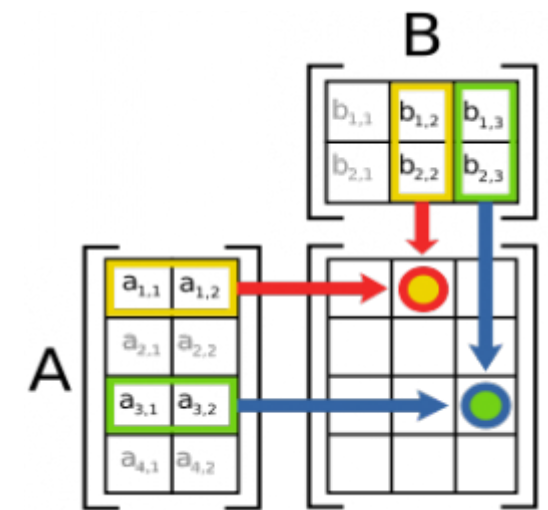
3	2
4	1

$1*3$	$2*2$
$3*4$	$4*1$

$1*3+2*4$	$1*2+2*1$
$3*3+4*4$	$3*2+4*1$

3	4
12	4

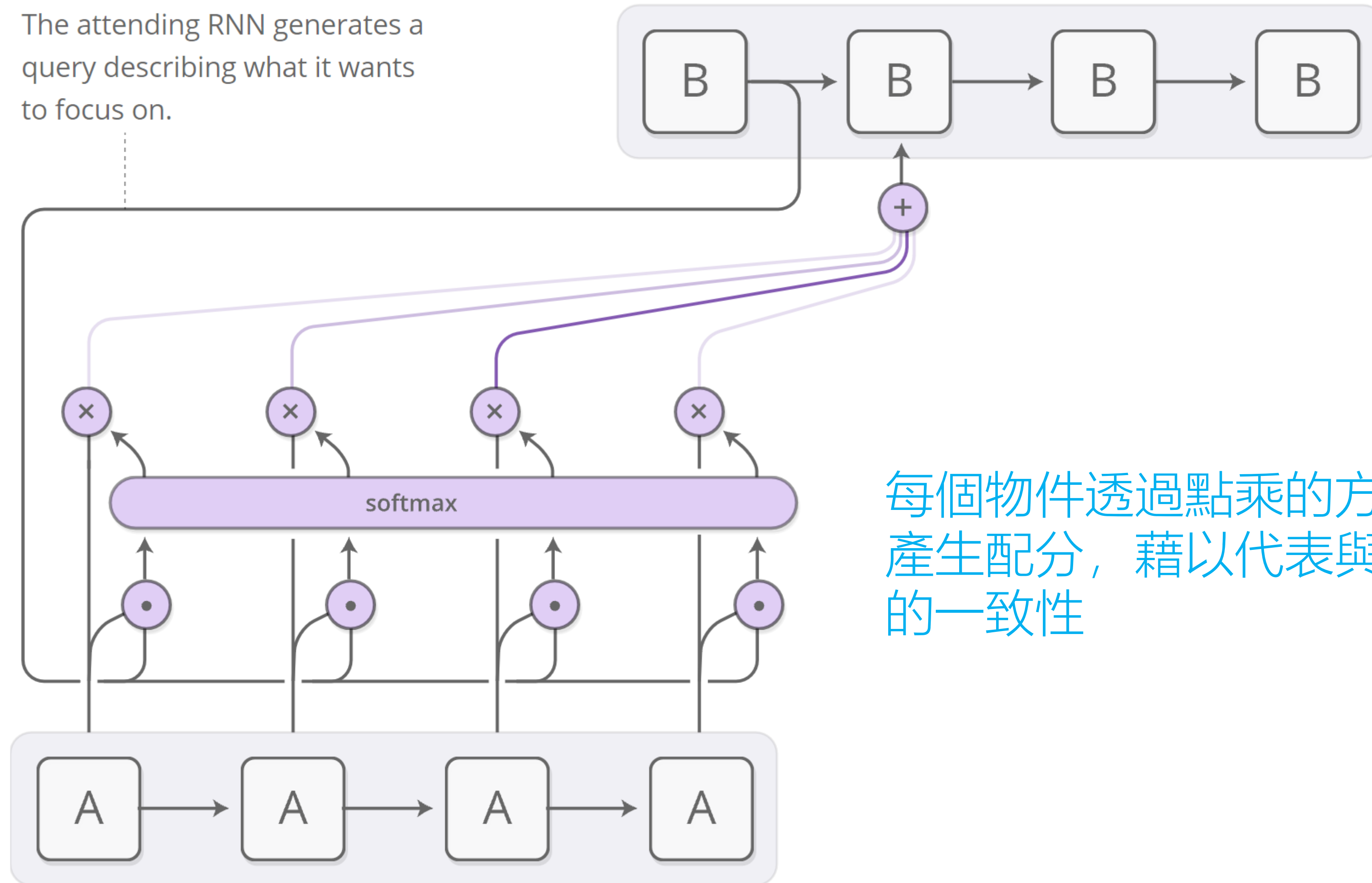
11	4
25	10





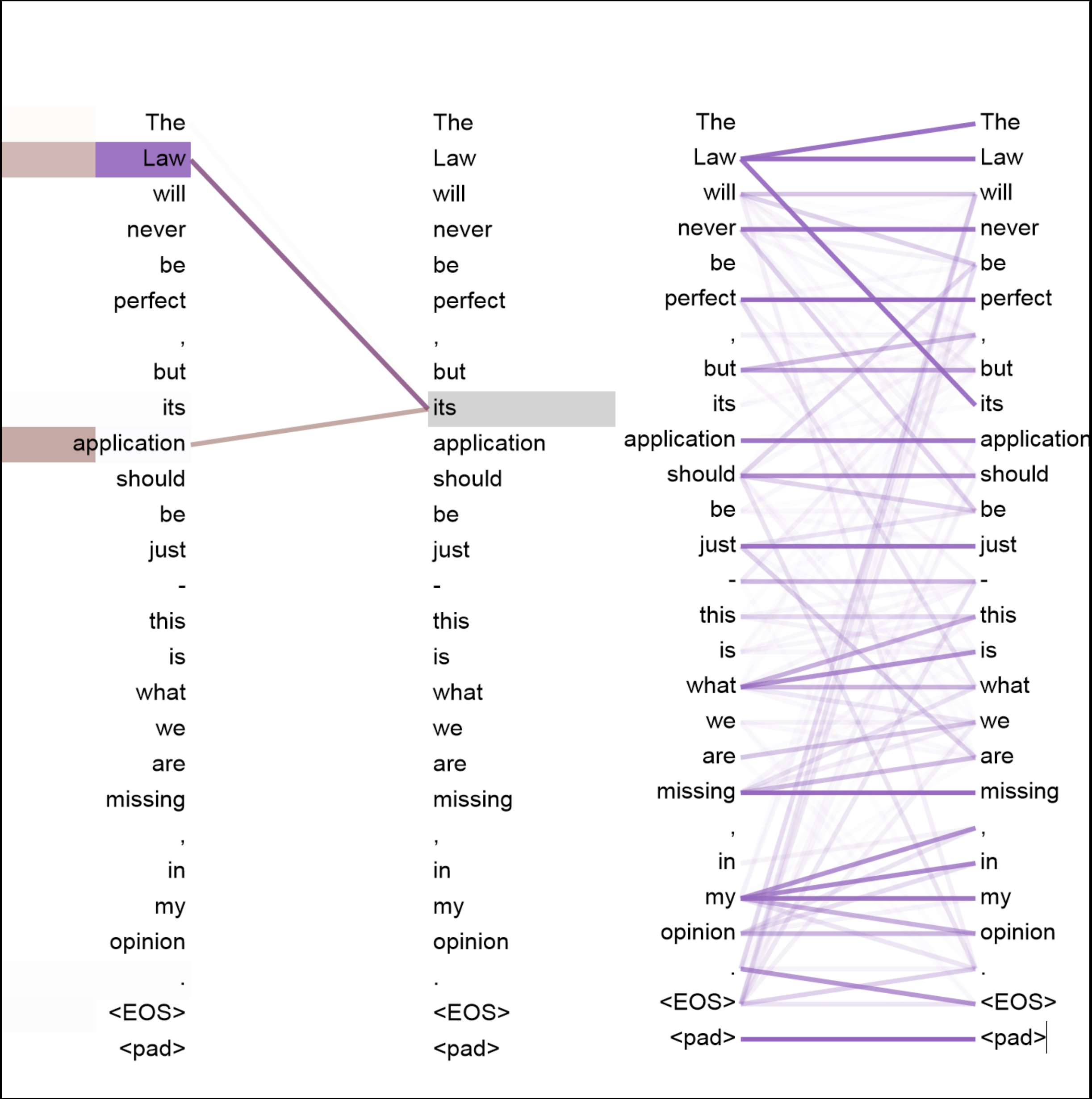
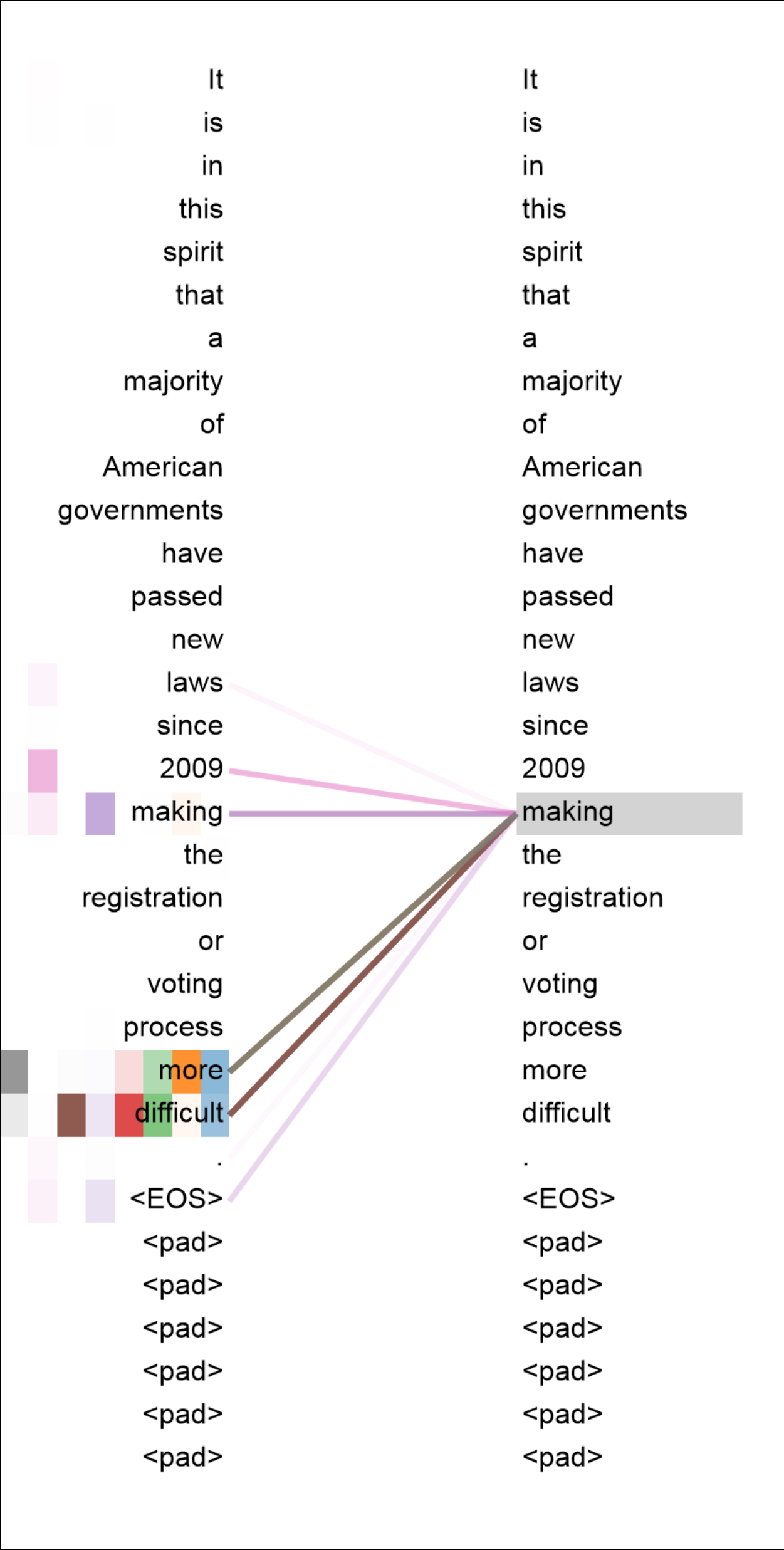
# RNN的注意力機制

The attending RNN generates a query describing what it wants to focus on.

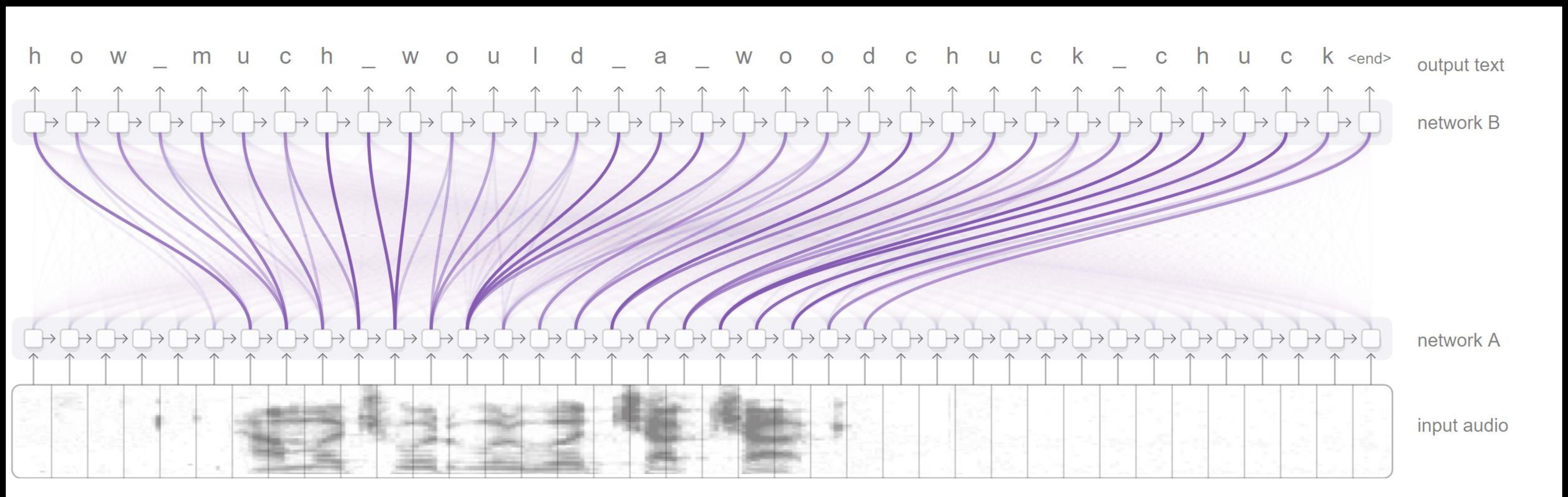
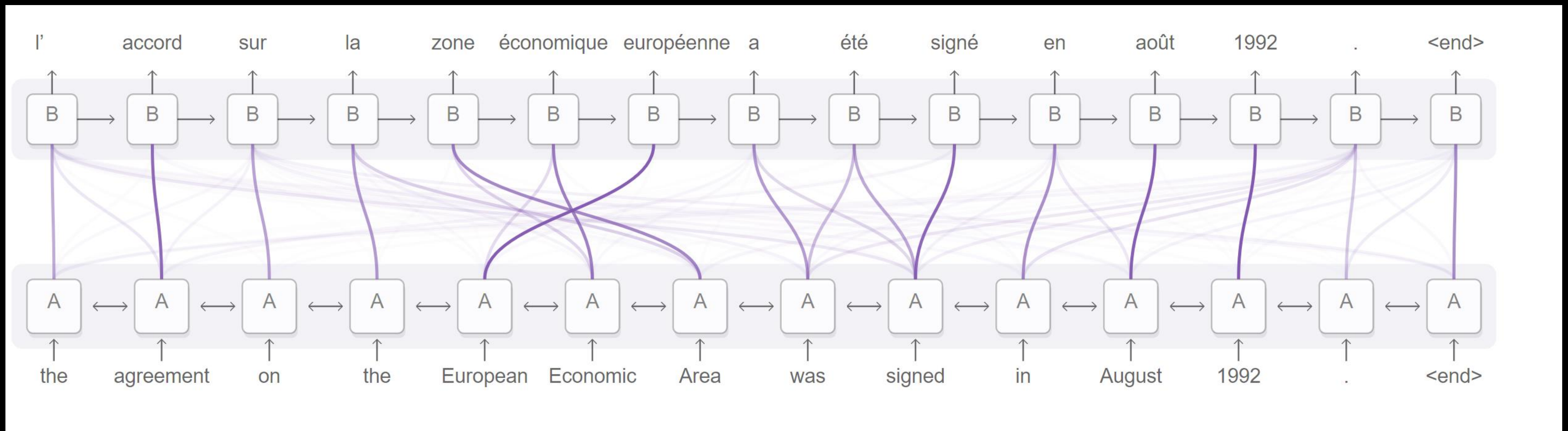


每個物件透過點乘的方式以產生配分，藉以代表與查詢的一致性

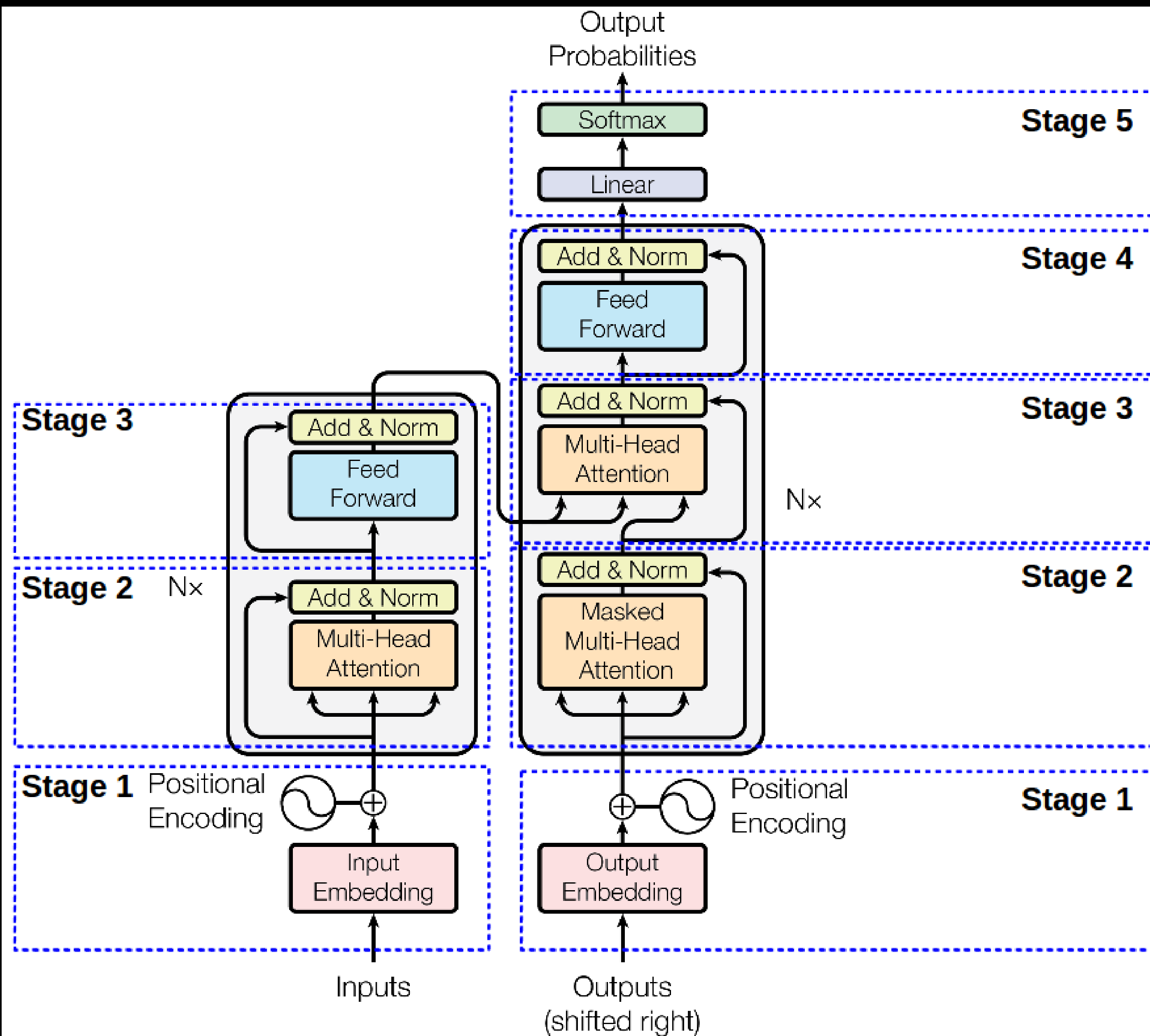
attention可以直接挖掘句子內部單詞與單詞的語義組合關係











Attention is all you need!  
Transformer

屬於編碼器-解碼器結構

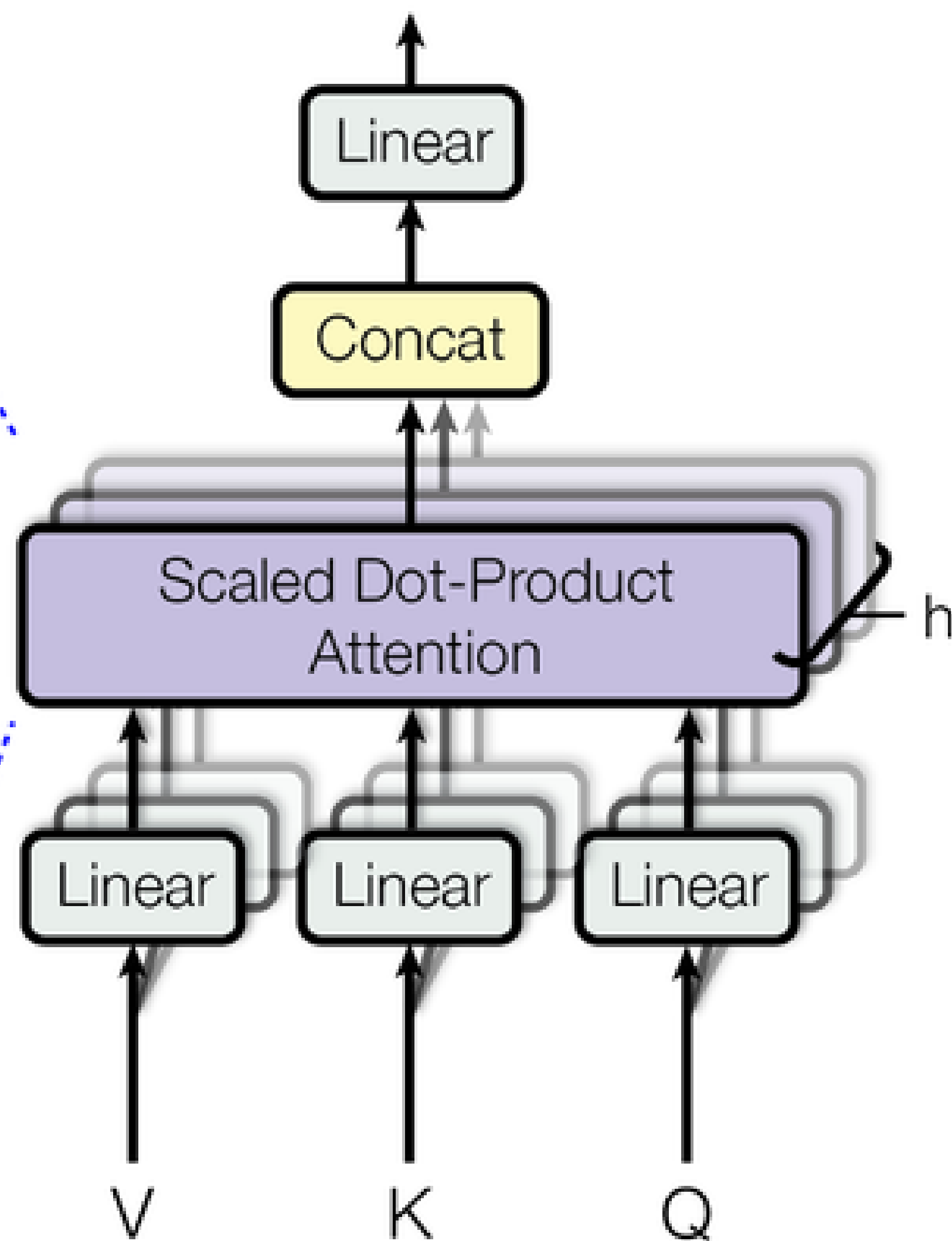
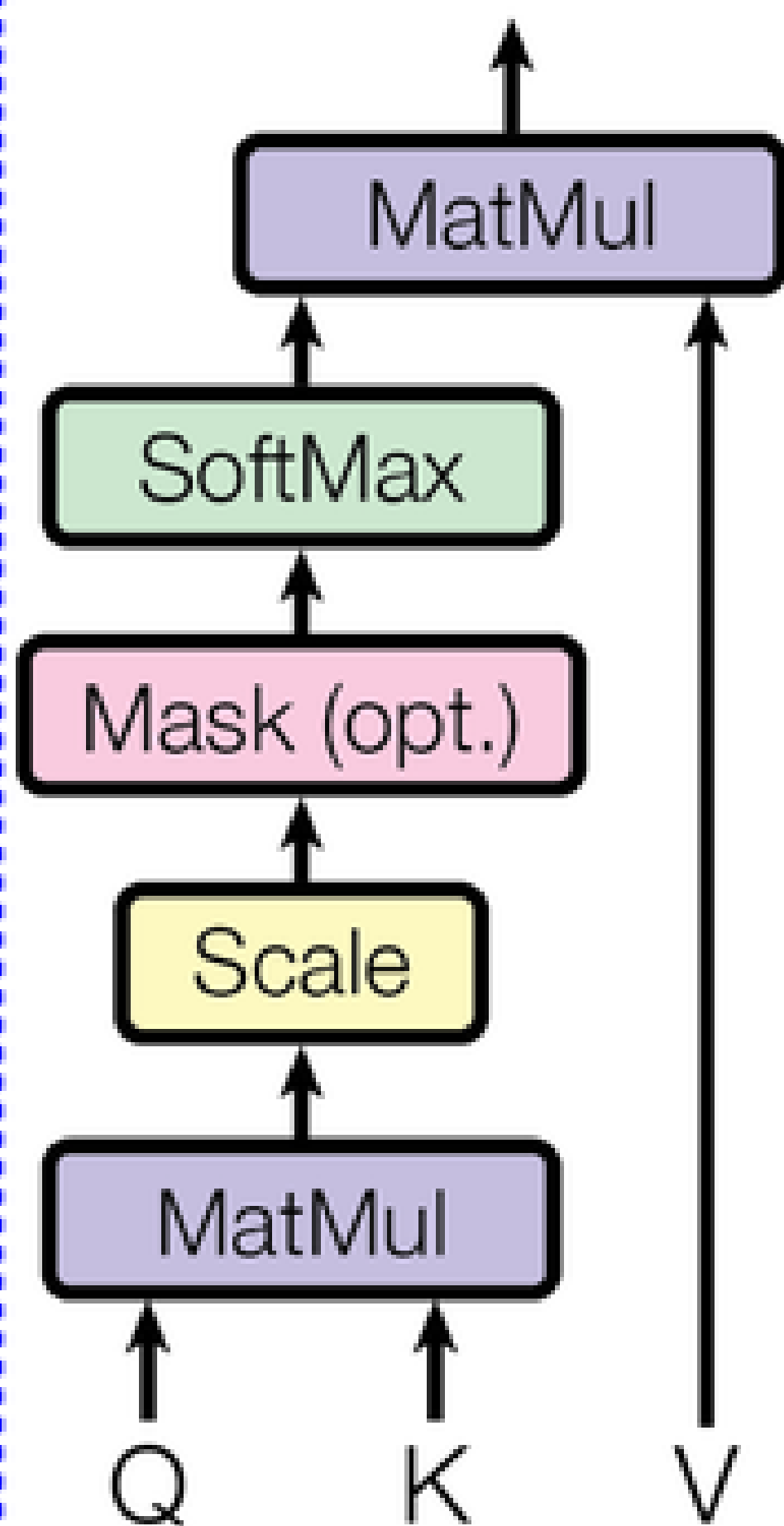
完全不使用RNN作為語言模型骨幹


首創多頭attention

使用了positional encoding

使用了殘差連結

### Scaled Dot-Product Attention





Q & A