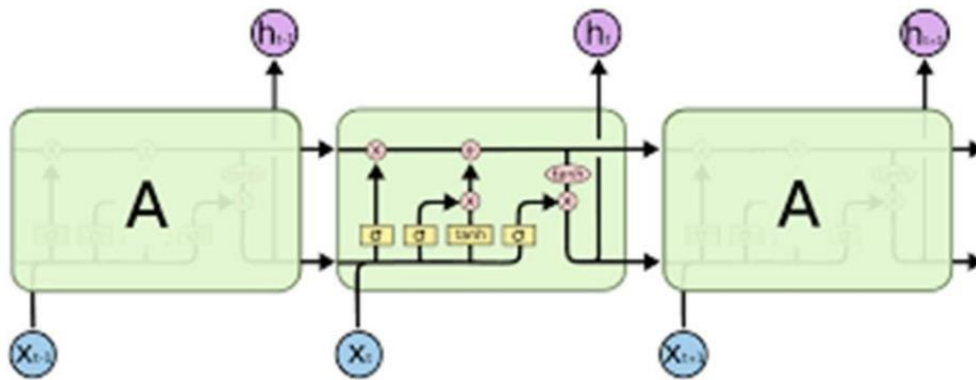


1. (30%) Given an LSTM model as shown in Figure 1, point out (i.e., write down the symbols) which is the input gate, output gate, and forget gate, respectively (10%). Then explain the function of each gate (10%). Suppose that we have an application that needs to predict an output y for a sequence of three inputs (x_1, x_2, x_3) (i.e., three time steps); draw an unfolded figure of LSTM for this application. (10%)

給定如圖 1 所示的 LSTM 模型，指出（即記下符號）分別是輸入門、輸出門和忘記門（10%）。

然後解釋每個門的功能（10%）。

假設我們有一個應用程式需要預測三個輸入序列（ x_1 、 x_2 、 x_3 ）（即三個時間步長）的輸出 y ；為此應用程式繪製 LSTM 的展開圖。（10%）



The repeating module in an LSTM contains four interacting layers.

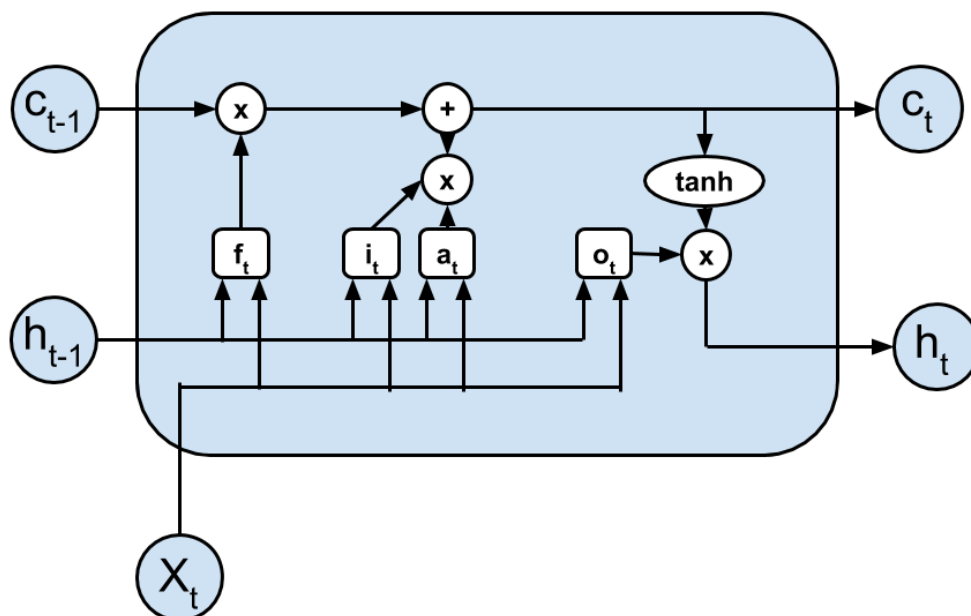


Figure 1. LSTM

2. (10%) What is the problem with a simple (or vanilla) RNN? How to solve it?

梯度消失和爆炸

- 梯度爆炸問題使 RNN 陷入不穩定狀態

由於 RNN 的長期組件不收斂被炸毀而導致的狀態。

- 當長期分量的梯度以指數速度快速變為零時，就會出現梯度消失問題，且模型為無法從暫時遙遠的事件中學習。（無法學習遙遠的事件或步驟）

梯度消失問題 (Vanishing Gradient Problem) :

在基礎 RNN 中，反向傳播過程中會計算損失函數相對於權重的梯度，這些梯度用來更新權重，當梯度通過多個時間步長向後傳播時，梯度會縮小。這種縮小會導致梯度變得非常小，最終接近於零。這就是所謂的梯度消失問題，使得模型難以學習長期依賴，因為與較早時間步長相關的權重無法有效更新。

梯度爆炸問題 (Exploding Gradient Problem) :

梯度在反向傳播過程中也可能會指數級地增大。這會導致權重更新過大，從而使網絡不穩定。雖然梯度爆炸問題相對較少見，但它會使訓練過程變得困難並導致模型無法收斂。

長期依賴問題 (Long-term Dependency Problem) :

由於梯度消失問題，基礎 RNN 難以捕捉長期依賴關係。這使得它們在處理需要長期上下文的任務時效果不佳。

解決方案

長短期記憶網絡（LSTM，Long Short-Term Memory）：

LSTM 網絡設計用來避免長期依賴問題。

記憶單元和輸入門、遺忘門和輸出門。

記憶單元：能夠長時間存儲訊息，

遺忘門（Forget Gate）：決定應該丟棄多少訊息。

輸入門（Input Gate）：決定應該存儲多少新訊息。

輸出門（Output Gate）：決定應該輸出多少訊息。

優勢：這種結構允許 LSTM 有效捕捉長期依賴，並減輕梯度消失問題。

門控循環單元（GRU，Gated Recurrent Unit）：

結構：

GRU 是 LSTM 的簡化版本，結合遺忘門（Forget Gate）和輸入門（Input Gate）成為一個更新門，並將記憶單元和隱藏狀態合併。

它包含兩個門：更新門和重置門。

運作：

更新門（Update Gate）：決定需要保留多少過去訊息。

重置門（Reset Gate）：決定應該忘記多少過去訊息。

優勢：GRU 的計算效率比 LSTM 更高，在許多任務中表現相似。

梯度截斷（Gradient Clipping）：

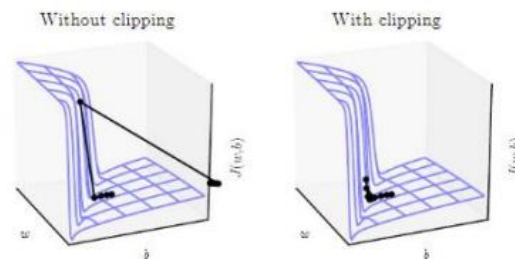
技術：

為了解決梯度爆炸問題，可以使用梯度截斷技術。這種技術涉及設置梯度的閾值，如果梯度超過這個閾值，則將其縮放到閾值範圍內。

To deal with the gradient exploding problem

1. Gradient clipping, where we threshold the maximum value a gradient can get

$$\text{if } \|g\| > \beta, \quad g \leftarrow \frac{\beta}{\|g\|} g$$



雙向 RNN (Bidirectional RNNs) :

這些網絡同時處理前向和後向數據，捕捉來自過去和未來的上下文訊息。

注意力機制 (Attention Mechanisms) :

注意力機制允許模型在預測每個輸出元素時關注輸入序列的特定部分，提高了長期依賴處理能力。

變壓器 (Transformers) :

變壓器完全依賴於注意力機制，摒棄了循環結構，在捕捉長期依賴和處理速度上表現優異，成為許多自然語言處理任務的最新技術。

總結

基礎 RNN 面臨梯度消失和梯度爆炸問題，這使得它們難以學習長期依賴。LSTM 和 GRU 等改進模型通過引入門機制有效地解決了這些問題。此外，梯度截斷、雙向 RNN、注意力機制和變壓器等技術進一步提升了模型捕捉長期依賴的能力和訓練的穩定性。

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

3. (10%) The following diagrams show a sequence-to-sequence translation application of an RNN with or without using attention. Point out which (Fig.2(a) or Fig.2(b)) is the RNN with attention model and which is not. Explain what is the difference between them. (10%) 下圖顯示了 RNN 的序列間翻譯應用程式，無論是否使用注意力。指出哪個（圖 2 (a) 或圖 2 (b)）是具有注意力模型的 RNN，哪個不是。解釋它們之間的區別

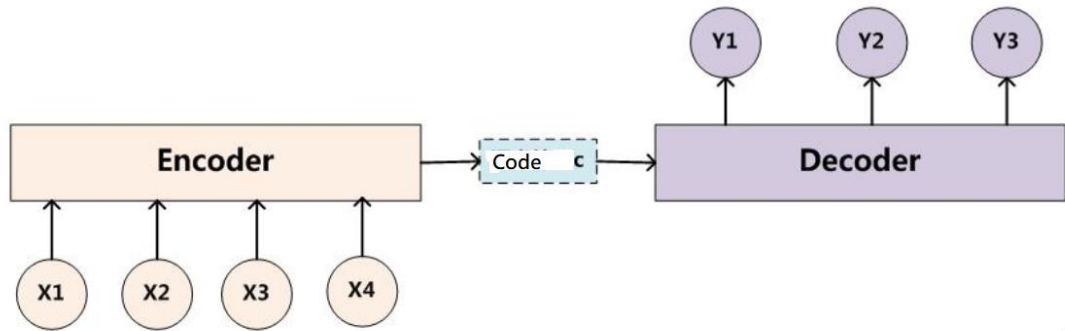
[Attention-Model.pdf](#)

- How to find c_i : weighted-sum of the component hidden state

$$C_{\text{汤姆}} = g(0.6 * f_2(\text{"Tom"}), 0.2 * f_2(\text{Chase}), 0.2 * f_2(\text{"Jerry"}))$$

$$C_{\text{追逐}} = g(0.2 * f_2(\text{"Tom"}), 0.7 * f_2(\text{Chase}), 0.1 * f_2(\text{"Jerry"}))$$

$$C_{\text{杰瑞}} = g(0.3 * f_2(\text{"Tom"}), 0.2 * f_2(\text{Chase}), 0.5 * f_2(\text{"Jerry"}))$$



4.

Fig. 2(a)

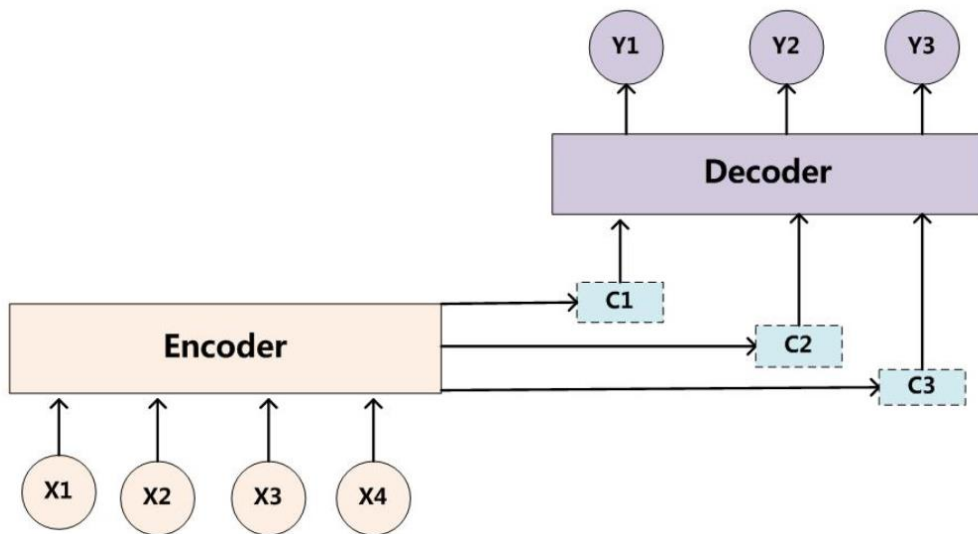


Fig. 2(b)

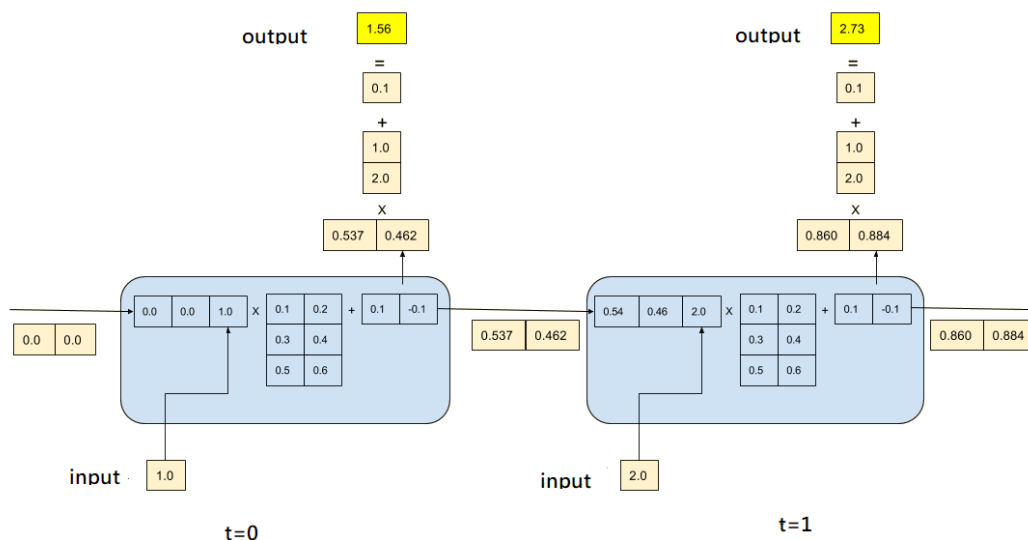
4. (20%) A simple RNN with an initial hidden state of $h_0=[0.0, 0.0]$, $U=[0.5,0.6]$, $V=[1.0,2.0]$, Hidden layer bias= $[1.0,-1.0]$, Output bias= $[0.1]$
 $W = [0.1,0.2]$
 $[0.3,0.4]$ 會在 code forwardrnn

Given the input vector $[2, 3]$, calculates the corresponding output of y .
For your reference, note that the activation function f is $\tanh()$.

$$h_1 = f(U \cdot x_1 + W \cdot h_0)$$

$$y_1 = g(V \cdot h_1)$$

An example of calculation is shown in the following figure.



5. (20%)The following piece of code defines a deep LSTM model for a time series prediction. It uses three time steps of historical time series values to predict the value of the next time step. (20%)以下代碼段定義了用於時間序列預測的深度 LSTM

模型。它使用歷史時間序列值的三個時間步長來預測下一個時間步的值。

```

1 # reshape from [samples, timesteps] into [samples, timesteps, features]
2 n_features = 1
3 X = X.reshape((X.shape[0], X.shape[1], n_features))
4 # define model
5 print("n_steps", n_steps, "n_features", n_features)
6 model = Sequential()
7 model.add(LSTM(30, activation='relu', return_sequences=True, \
8               input_shape=(n_steps, n_features)))
9 model.add(LSTM(30, activation='relu'))
10 model.add(Dense(1))
11 model.summary()

```

n_steps 3 n_features 1

Answer the following questions:

a How many parameters are there in the first LSTM layer?

b How many parameters are there in the second LSTM layer?

Hint: The input of the first layer is just a real number, and the input of the second layer is the hidden vector of the first layer, which has a dimension of 30.

第一個 LSTM 層有多少個參數？第二個 LSTM 層有多少個參數？提示：第一層的輸入只是一個實數，第二層的輸入是第一層的隱向量，維度為 30。

模型結構

第一個 LSTM 層 (lstm_2)

輸出形狀: (None, 3, 30)

None 表示批次大小不固定。

3 是時間步數 (n_steps)。

30 是 LSTM 層的記憶單元數。

參數數量: 3840

參數數量計算公式: $4 * (\text{units} * (\text{units} + \text{input_dim} + 1))$

units 是記憶單元數 (30)。

input_dim 是輸入特徵數 (1)。

公式細分: $4 * (30 * (30 + 1 + 1)) = 4 * 30 * 32 = 3840$

第二個 LSTM 層 (lstm_3)

輸出形狀: (None, 30)

None 表示批次大小不固定。

30 是 LSTM 層的記憶單元數。

參數數量: 7320

參數數量計算公式: $4 * (\text{units} * (\text{units} + \text{input_dim} + 1))$

units 是記憶單元數 (30)。

input_dim 是來自前一層的輸出維度 (30)。

公式細分: $4 * (30 * (30 + 30 + 1)) = 4 * 30 * 61 = 7320$

全連接層 (dense_1)

輸出形狀: (None, 1)

None 表示批次大小不固定。

1 是輸出特徵數。

參數數量: 31

參數數量計算公式: $\text{units} * (\text{input_dim} + 1)$

units 是輸出特徵數 (1)。

input_dim 是來自前一層的輸出維度 (30)。

公式細分: $1 * (30 + 1) = 31$

總參數數量

總參數數量: 11191

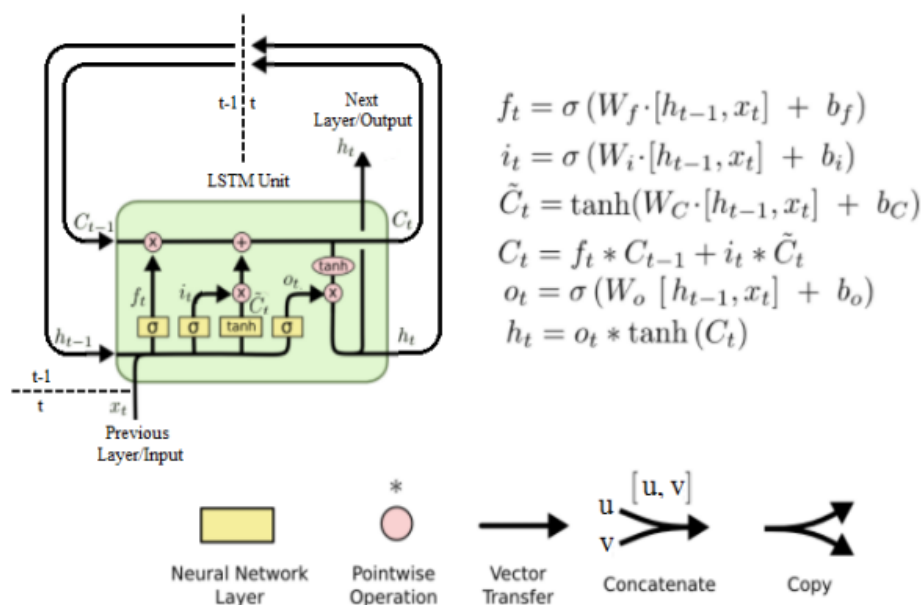
這是所有層參數數量的總和，即 $3840 + 7320 + 31 = 11191$

模型摘要

這個摘要展示了模型的各層結構，包括輸入和輸出形狀、每層的參數數量，以及總參數數量。這有助於了解模型的複雜性和計算需求。

這個模型由兩層 LSTM 層和一層全連接層組成，適用於時間序列預測任務，特別是預測下一個時間步的值。每層的參數數量顯示了模型需要訓練的權重和偏置數量，反映了模型的學習能力和複雜度。

Understanding LSTM Networks

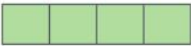
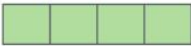

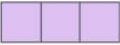
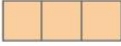
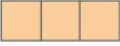
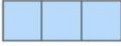
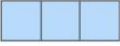


Bf 是 bias

6. (10%) Transformer is the foundation of the many recent large pre-trained language models, such as BERT and ChatGPT. The concept of self-attention is the core of the Transformer. Based on the following figure, please explain the relationships among query, key, value and the resulting representation Z . Please answer this question by considering only one-head attention. *Specifically, please answer how Z 's first row is derived from v_1 and v_2 .*

6. (10%) Transformer 是最近許多大型預訓練語言模型的基礎，例如 BERT 和 ChatGPT。自我關注的概念是變形金剛的核心。根據下圖，請解釋查詢、鍵、值和結果表示 Z 之間的關係。請只考慮單頭注意力來回答這個問題。具體來說，請回答 Z 的第一行是如何從 v_1 和 v_2 派生的。

老師可能會給我們三個字，叫我們計算 hidden layer

Input	Thinking	Machines
Embedding	x_1 	x_2 
Queries	q_1 	q_2 
Keys	k_1 	k_2 
Values	v_1 	v_2 
Score	$q_1 \cdot k_1 = 112$	$q_2 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12



在 transformer 裡面

<..\W13 0513\Understanding Transformer Architecture Using Simple Math.docx>

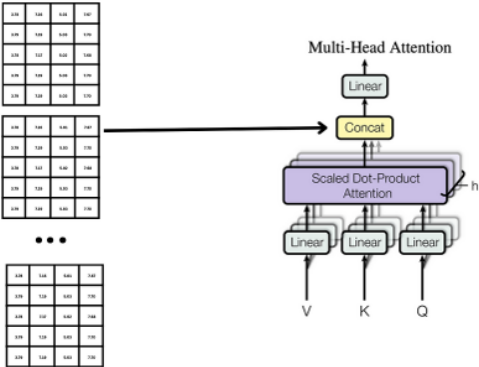
<..\W13 0513\corrections on some examples.pptx>

題目也有 GAN

<Information, entropy, cross entropy, KL-Divergence.pdf>

https://blog.luckylucy.live/2023/08/28/Aigc_Stable%20Diffusion%E5%8E%9F%E7%90%86/

3.78	7.16	5.01	7.67
3.79	7.19	5.03	7.70
3.78	7.17	5.02	7.68
3.79	7.19	5.03	7.70
3.79	7.19	5.03	7.70



3.78	7.16	5.01	7.67	3.78	7.16	5.01	7.67	3.78	7.16	5.01	7.67
3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70
3.78	7.17	5.02	7.68	3.78	7.17	5.02	7.68	3.78	7.17	5.02	7.68
3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70
3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70	3.79	7.19	5.03	7.70

5x12

X

0.85	0.21	0.6	0.37
0.12	0.54	0.78	0.33
0.67	0.28	0.91	0.19
0.35	0.62	0.3	0.97
0.67	0.34	0.32	0.76
0.12	0.54	0.26	0.79
0.99	0.87	0.61	0.56
0.45	0.34	0.23	0.12
0.76	0.38	0.74	0.89
0.77	0.16	0.20	0.48
0.26	0.86	0.70	0.29
0.34	0.79	0.30	0.69

12 X 4

=

10.11	10.81	12.79	12.15	14.34	12.08
10.14	10.86	12.84	12.19	14.39	12.13
10.12	10.83	12.81	12.16	14.35	12.10
10.14	10.86	12.84	12.19	14.39	12.13
10.14	10.86	12.84	12.19	14.39	12.13

5x6

Multi-Head At