

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%）

LSTM (長短期記憶)

LSTM 網絡設計用於記住長期依賴，讓模型可以長期的計算每個輸入的影響程度，並由一系列單元組成，每個單元包含三個門：遺忘門、輸入門和輸出門。

1. 遺忘門 f_t : 決定從前一個單元狀態中應該丟棄或保留的信息。使用 Sigmoid 函數輸出 0 到 1 之間的值。
2. 輸入門 i_t : 決定將哪些新信息存儲在單元狀態中。結合輸入數據和之前的隱藏狀態 hidden state 來更新單元狀態。
3. 輸出門 o_t : 控制輸出，決定單元狀態的哪一部分應作為下一時間步的隱藏狀態輸出。
4. AT: 以前稱為 gate gate, 是一種 \tanh 激活函數，用於將輸出入資料控制在 -1 到 +1 之間。

門控循環單元 (GRU, Gated Recurrent Unit):

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。

GRU 有兩個門：更新門和重置門。

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。

主要差異: LSTM 有三個門 (遺忘門、輸入門、輸出門)，而 GRU 有兩個 (重置門、更新門)。LSTM 更複雜，可以捕捉更多複雜的模式，而 GRU 更簡單，訓練速度更快。

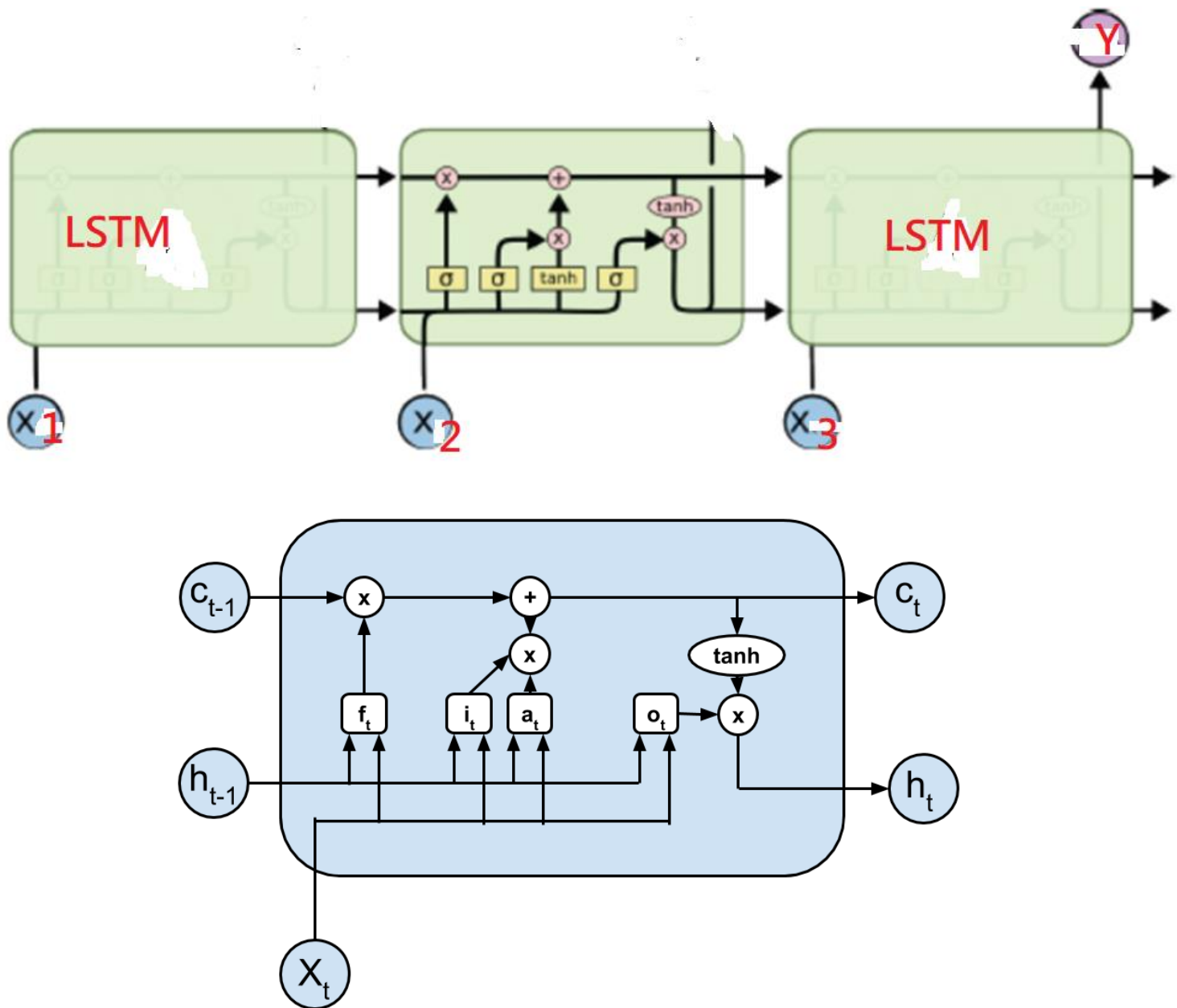


Figure 1. LSTM

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

可能會導致 **gradient vanishing or grading exploding problems** 梯度消失或分級爆炸問題。使用「**梯度裁剪**」求解梯度爆炸問題，使用 **LSTM 或 GRU** 求解梯度消失問題。

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 的長期不收斂被導致資訊量過多的狀態。

- 當長期梯度以指數速度快速變為零時，就會出現梯度消失問題，且模型暫時無法從遙遠的事件中學習。

總結

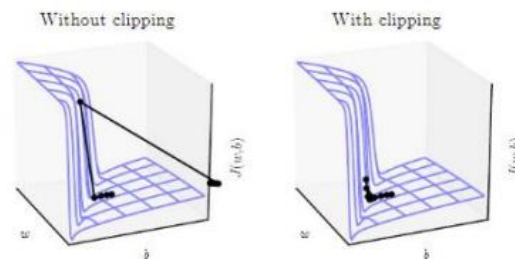
基礎 RNN 面臨梯度消失和梯度爆炸問題，這使得模型難以學習長期依賴。LSTM 和 GRU 等改進模型通過引入門機制有效地解決了這些問題。此外，梯度截斷 Gradient Clipping、雙向 RNN (Bidirectional RNNs)、注意力機制(Attention Mechanisms)和 transformer 等技術進一步提升了模型捕捉長期依賴的能力和訓練的穩定性。

<u>梯度消失問題 (Vanishing Gradient Problem) :</u>
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。
<u>梯度爆炸問題 (Exploding Gradient Problem) :</u>
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
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

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



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。

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)：

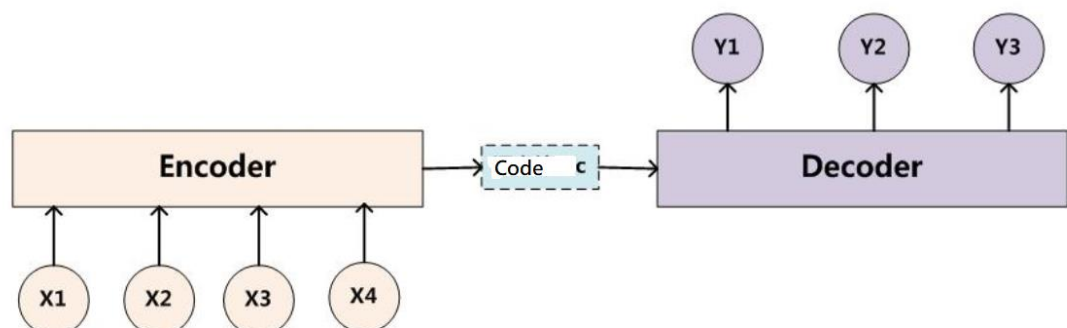
變壓器完全依賴於注意力機制，摒棄了循環結構，對於資訊的長期依賴性佳和處理速度快。

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](#)

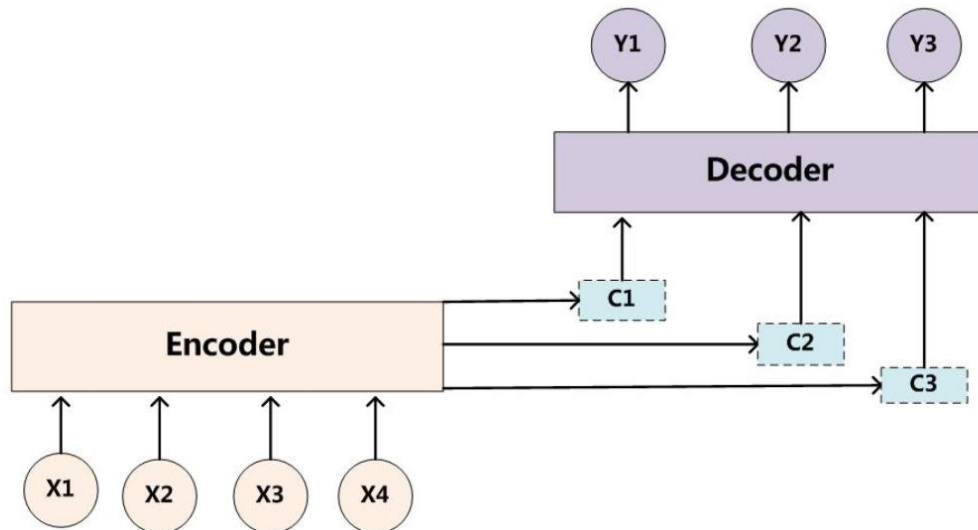
注意力機制 (attention model)：

注意力機制允許模型關注輸入序列的特定部分，讓模型得以對句子(序列)中每個字對個別字的相關性和上下文影響做判斷。



4.

Fig. 2(a)



✓ Fig. 2(b)

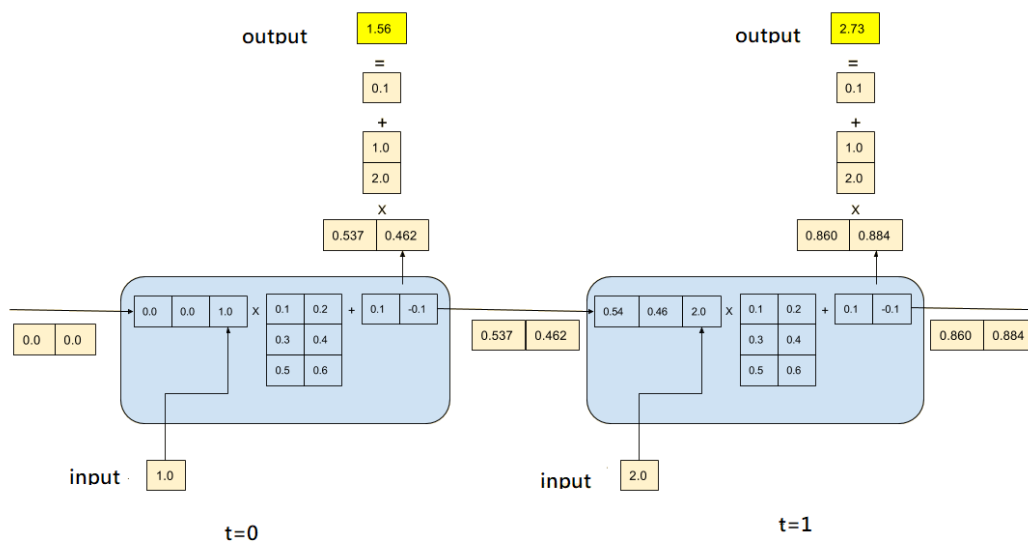
4. (20%) A simple RNN with an initial hidden state of $h_0=[0.0, 0.0]$, $U=[0.5,0.6]$, $V=[1.0,2.0]$, Hidden layer bias= $[1.0,-1.0]$, Output bias= $[0.1]$
 $W = [0.1,0.2]$
 $[0.3,0.4]$ 會在 code forwardrnn

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

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

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

- A. 將 h_0 和 $\text{input_vector}(x_1)$ 合成一個向量
- B. 乘上 W 、 U 權重，做 matrixs 矩陣的乘法計算，橫乘直
- C. 加上 hidden layer bias
- D. Tanh，成為下一層 h_1 的 hidden layer，並同時繼續運算
- E. 乘上 V 權重
- F. 加上 output bias 得到 output y_1



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%)

```
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:

- How many parameters are there in the first LSTM layer?
- 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.

Layer parameters 公式

$$4 \text{ gates} * \text{units(記憶單元數量)} * [\text{units} + (\text{layer1: n_feature}) + 1] + (\text{layer2: iuput 上層輸出參數})$$

第一個 LSTM 層 (lstm_2) parameters

$$4 * [\text{units} * (\text{units} + \text{n_feature} + 1)] = 4 * (30 * (30 + 1 + 1)) = 3840$$

第二個 LSTM 層 (lstm_3) parameters

$$4 * [\text{units} * (\text{units} + \text{iuput_dim} + 1)] = 4 * (30 * (30 + 30 + 1)) = 7320$$

全連接層 (dense_1)公式

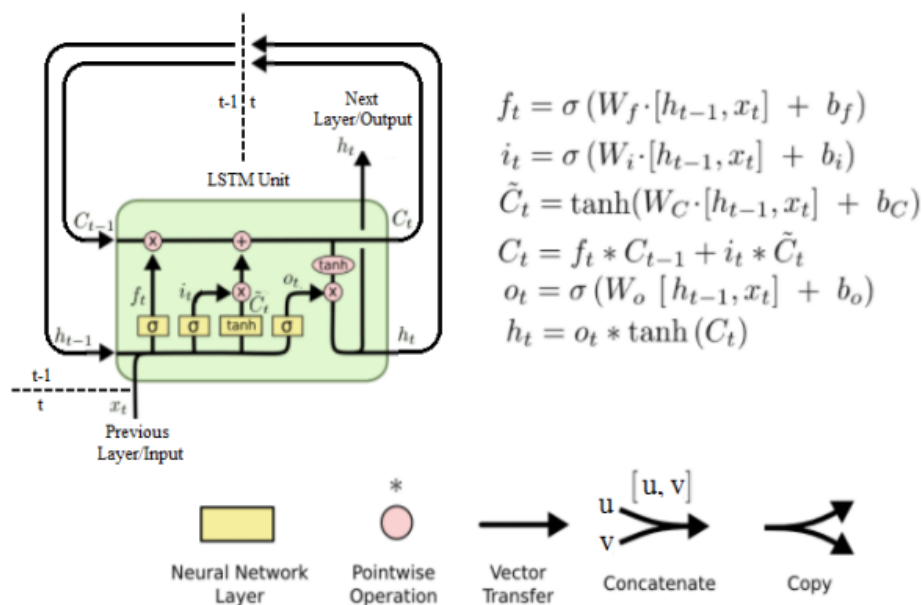
$$\text{Parameter} = (\text{input_dim} * \text{neuron}) + \text{neurons} \\ (\text{model.add.dense(A)})$$

全連階層參數量受上一層輸出 units 和密集層自身神經元數影響

$$\text{Parameter} : (30*1)+1=31$$

$$\text{總參數數量} = 3840 + 7320 + 31 = 11191$$

Understanding LSTM Networks

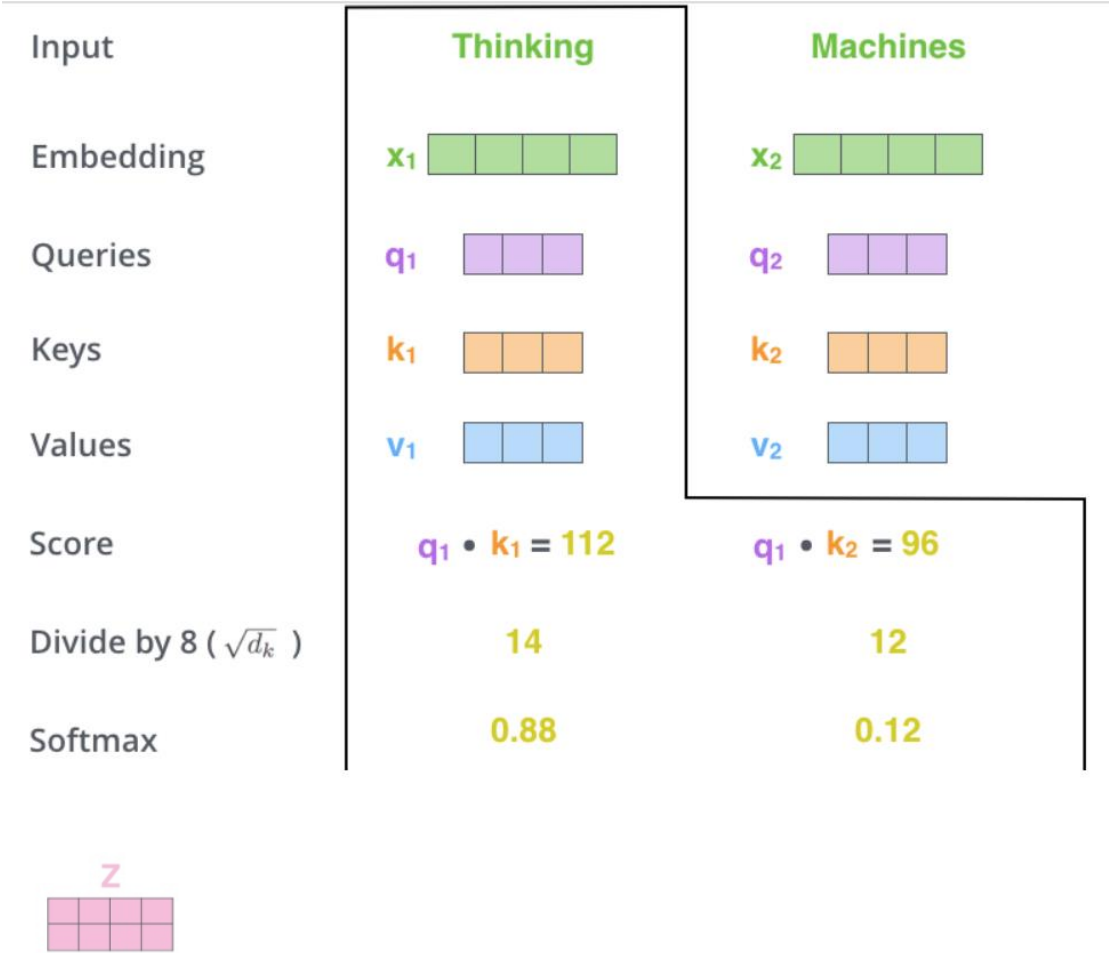


Bf 是 bias

5. (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 v1 and v2.*

在 Transformer 模型中，自注意力機制的核心概念如下：



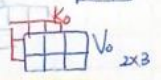
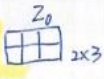
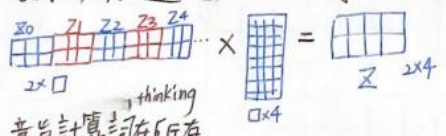
1. **Query (Q)**：用於搜尋相關訊息的查詢向量矩陣。
2. **Key (K)**：用於配對查詢的鍵向量矩陣。
3. **Value (V)**：包含實際訊息的值向量矩陣。
4. **表示 Z**：自注意力機制的輸出，是基於注意力得分的值的加權和。



The answer is : $Z_1 = 0.88 * v_1 + 0.12 * v_2$

$$Q6 = (0.88 + \frac{112}{V1}) \times (0.12 + \frac{96}{V2}) = 10,850.0256 \quad \text{by } V1 \text{ 和 } V2 \text{ 衍生}$$

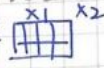
QKV的關係和Z的關係，用單頭注意力解釋，並說明Z是怎麼derived

1. input sequences thinking
machine
 2. 嵌入向量
embed each words 
 3. 計算單頭注意力 
 4. 得自注意力 
 5. $\text{softmax}(\frac{Q \cdot K}{\sqrt{d_k}}) \cdot V$ 得 Z 
 6. 把多頭Z連起來 $\times W^o$ 得 Z

- 意旨計算詞在所有單頭自注意力中的Value值，成多頭的Z值。

透過將one-head attention的結果

彙總產生 multi-self attention,

是為了得到多個 self attention 間不同詞間相關性和上下文意完整理解。

透過將詞嵌入向量得 

並將詞向量 x_1, x_2 分別乘上 W^k, W^v

得到 Head 0 的 Q_0, K_0, V_0 self-attention

→ 再 $\text{softmax}(\frac{Q \cdot K}{\sqrt{d_k}}) \times V$ 得 Z 。

→ 把所有單頭 self-attention 的 Z 合起來 $\times W^o$ 得 Z ，就可以計算所有詞在不同 self-attention 的值中多頭

$$\text{故 } Z_1 = V_1 \cdot \text{softmax}(0.88) + V_2 \cdot \text{softmax}(1.12)$$

= 詞1 (thinking) 在 thinking 和 machine 中產生的相關性和上下文意都被考慮。

Transformer Single Embedding 嵌入.

前由 input \rightarrow input embedding \rightarrow encoding 編碼

1. Input Embedding 嵌入: 將單詞轉換為一系列數列 (向量), 並將向量做為處理和理解語言的輸入 (稱為編碼器). Dimensional 維度是自設的.

input Embedding \rightarrow Cat is sleeping on the mat

.54	.76	.23	.89	.1	.67
.31	.93	.68	.45	.82	.17
.40	.56	.72	.64	.20	.95
.72	.89	.18	.31	.52	.39
.62	.87	.04	.98	.34	.52

} 預設5維 / 5 dim

2. Positional Encoding 位置編碼: 讓 Model 知道每個詞的順序, Transformer 是一整句一起理解, 並會考慮詞 & 詞互相影響, (似上下文)

* PE 有 2 公式 $\begin{matrix} \text{For even position} \\ \text{For odd position} \end{matrix}$, 不過應該題目會給. 相關程度

3. Adding "Word Embedding" & "Positional Embedding"

input embedding		positional embedding
for	+	
Cat is sleeping on the mat.		Cat is sleeping on the mat.
.54 .76 .23 .89 .1 .67		.0 .891 .90 .41 .95 .95
\vdots		\vdots
= Cat is sleeping on the mat *		
.54 1.60 1.13 1.031 -0.65 -0.28		
\vdots		

4. Multi-self attention: Matrix 相乘 QKV 考慮上文、詞間互相影響.

計算 / 考慮句子中某個詞與其他部分的相關程度.

4. Multi-self attention

所有的 Word Embedding + Positional Embedding 都相同, 不同的是 QKV 的 linear weights.

疊乘 Weight 再相加

Cat is sleeping on the mat

5x6 Matrix

linear Weights

6x4 Matrix

* Query 5x4 Matrix

5x4 Matrix

1 Query 計算

直向相加

$$\begin{aligned}
 & .54 \times (.67, .14, .71, .03) \\
 & + 1.601 \times (.85, .31, .14, .95) \\
 & + 1.13 \times (.24, .57, .76, .93) \\
 & + 1.031 \times (.39, .95, .59, .38) \\
 & + (-0.65) \times (.64, .94, .77, .51) \\
 & + (-0.28) \times (.94, .73, .17, .02) \\
 & = \begin{pmatrix} .362 & .076 & .383 & .016 \\ 1.361 & 0.50 & 2.24 & 1.521 \\ 0.271 & .644 & .860 & 1.051 \\ -.402 & .980 & .610 & .392 \\ -.0416 & -.0611 & .501 & -.033 \\ -.263 & -.204 & -.048 & -.0056 \end{pmatrix} = \text{Query} \begin{pmatrix} 1.71 & 1.38 & 1.52 & 2.64 \end{pmatrix}
 \end{aligned}$$

2 Key 計算

只算第一行

$$\begin{aligned}
 & .54 \times (.50, .76, .28, .12) \\
 & + 1.601 \times (.35, .33, .63, .25) \\
 & + 1.13 \times (.79, .99, .04, .95) \\
 & + 1.031 \times (.03, .09, .23, .09) \\
 & - 0.65 \times (.85, .04, .09, .93) \\
 & - 0.28 \times (.14, .63, .30, .21) \\
 & = \begin{pmatrix} .27 & .41 & .151 & 0.065 \\ .56 & .53 & 1.01 & .40 \\ -.90 & 1.12 & 0.05 & 1.07 \\ .03 & .09 & .24 & .09 \\ -.055 & -.003 & -.01 & -.28 \\ .604 & .18 & -.09 & -.06 \end{pmatrix} = \text{Key} \begin{pmatrix} 1.17 & 1.94 & 1.35 & 1.285 \\ 1.16 & & & 1.29 \end{pmatrix}
 \end{aligned}$$

3. Value 計算

老師給的公式錯誤.

$$\begin{aligned}
 & .54 \times (.50, .76, .28, .12) \\
 & + 1.601 \times (.35, .33, .63, .25) \\
 & + 1.13 \times (.79, .99, .04, .95) \\
 & + 1.031 \times (.03, .09, .23, .09) \\
 & - 0.65 \times (.85, .04, .09, .93) \\
 & - 0.28 \times (.14, .63, .30, .21) \\
 & = \begin{pmatrix} .27 & .410 & .151 & 0.065 \\ .0560 & 0.528 & 1.009 & 0.400 \\ .892 & 1.119 & 0.045 & 1.074 \\ .0031 & 0.093 & 0.237 & 0.093 \\ -.553 & -.026 & -.00585 & -.280 \\ .039 & .176 & -.006 & .059 \end{pmatrix} \\
 & \quad .084
 \end{aligned}$$

反正得出的 Query / Key / Value

$$5. \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

成4x5

Query 5x4

Key 5x4

Key 矩陣要轉置 Transpose

1.71 1.38 1.52 2.64
5.27 5.13 4.62 4.72
2.19 2.13 1.60 1.58
5.76 5.12 4.53 4.22
4.32 8.11 5.54 8.05

1.16 1.94 1.35 1.29
4.15 3.97 2.22 3.18
1.29 1.87 0.93 1.21
3.95 4.10 2.43 2.83
5.45 5.65 4.52 4.50

1.16 4.15 1.29 3.95 5.45
1.94 3.97 1.87 4.10 5.65
1.35 1.87 0.93 2.43 4.52
1.29 4.10 2.43 4.52 4.50

只求第一行

得5x5 Matrix

$$\begin{pmatrix} 1.71 \times (1.16, 1.94, 1.35, 1.29) \\ 1.38 \times (4.15, 3.97, 2.22, 3.18) \\ 1.52 \times (1.29, 1.87, 0.93, 1.21) \\ 2.64 \times (3.95, 4.10, 2.43, 2.83) \end{pmatrix} + \begin{pmatrix} 1.71 \times (1.16, 4.15, 1.29, 3.95, 5.45) \\ 1.38 \times (1.94, 3.97, 1.87, 4.10, 5.65) \\ 1.52 \times (1.35, 1.87, 0.93, 2.43, 4.52) \\ 2.64 \times (1.29, 4.10, 2.43, 4.52, 4.50) \end{pmatrix} = \begin{pmatrix} 10.1 & 29.3 & 9.39 & 23.5 & 35.8 \\ 28.3 & 67.5 & 26.3 & 66.9 & 99.8 \\ 12.8 & 26.1 & 10.2 & 25.7 & 38.3 \\ 28.1 & 67.7 & 26.3 & 66.6 & 99.7 \\ 44.4 & 102.7 & 42.0 & 102.3 & 157.8 \end{pmatrix}$$

* Scaling 縮放, 將上述 QxK 轉置結果的 5x5 $\div \sqrt{d_K} \rightarrow$ dimension 5

$$\begin{pmatrix} 10.1 & 29.3 & 9.39 & 23.5 & 35.8 \\ 28.3 & 67.5 & 26.3 & 66.9 & 99.8 \\ 12.8 & 26.1 & 10.2 & 25.7 & 38.3 \\ 28.1 & 67.7 & 26.3 & 66.6 & 99.7 \\ 44.4 & 102.7 & 42.0 & 102.3 & 157.8 \end{pmatrix} \div \sqrt{5} = \begin{pmatrix} 4.52 & 10.89 & 4.20 & 10.51 & 16.01 \\ 12.66 & 30.19 & 11.76 & 29.70 & 44.63 \\ 5.23 & 11.67 & 4.56 & 11.99 & 17.13 \\ 12.57 & 30.28 & 11.76 & 29.78 & 44.59 \\ 19.86 & 48.61 & 18.82 & 47.59 & 70.57 \end{pmatrix}$$

可以想成一條一個頭

$$\sqrt{d_K} < \sqrt{5}$$

$$\text{softmax} = \frac{e^{4.52}}{e^{4.52} + e^{10.89} + e^{4.20} + e^{10.51} + e^{16.01}} = 0.0001$$

橫全相加

每個重複 softmax 得 5x5 Matrix

$\text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)$ 5x5 Matrix

Value 5x4 = softmax 值 x V = Z

$$\begin{pmatrix} 0.0001 & 0.0054 & 0.0007 & 0.0004 & 0.9994 \\ 0 & 0.0005 & 0 & 0.0003 & 0.9999 \\ 0.00045 & 0.004 & 0.0033 & 0.0036 & 0.9922 \\ 0 & 0.005 & 0 & 0.0037 & 0.9999 \\ 0 & 0.0003 & 0 & 0.0001 & 1 \end{pmatrix} \times \begin{pmatrix} 1.20 & 1.80 & 0.32 & 1.90 \\ 2.95 & 4.52 & 3.45 & 4.99 \\ 0.89 & 1.58 & 1.14 & 1.83 \\ 3.17 & 4.64 & 3.85 & 5.17 \\ 3.09 & 7.19 & 5.03 & 7.70 \end{pmatrix}$$

只算一行

得 5x4 Matrix

一個單頭注意力

$$\begin{pmatrix} 0.0001 \times (1.20, 1.80, 0.32, 1.90) \\ 0.0054 \times (2.95, 4.52, 3.45, 4.99) \\ 0.0007 \times (0.89, 1.58, 1.14, 1.83) \\ 0.0004 \times (3.17, 4.64, 3.85, 5.17) \\ 0.9994 \times (3.09, 7.19, 5.03, 7.70) \end{pmatrix} = \begin{pmatrix} 3.78 & 7.16 & 5.01 & 7.67 \\ 3.79 & 7.19 & 5.03 & 7.70 \\ 2.78 & 7.17 & 5.02 & 7.68 \\ 3.79 & 7.19 & 5.03 & 7.70 \\ 3.79 & 7.19 & 5.03 & 7.70 \end{pmatrix}$$

得 5x4 Matrix

三注意力

得 3x4 Matrix

= 5x12 Matrix

Transformer: 序列輸入 序列輸出 模型結構 \rightarrow input \rightarrow ^(有時 hidden state) encoder \rightarrow ^(vector) context \rightarrow ^{output} decoder

based on RNN, 但透過「self-attention」自注意力機制考慮句子中所有單字相關性和上下文, 可以同時理解句子的所有部分, 對整理語意更精準分析, 克服 RNN 中長期依賴和計算效率不佳的限制
long-term dependency

* Transformer 由 編碼器 encoder 和 解碼器 decoder 組成

* Encoder 編碼器 \rightarrow 將輸入序列轉為連續形式, 以獲得上下文信息
輸出編碼器一系列嵌入向量, 稱「隱藏狀態」或「上下文」。

* Decoder 解碼器 \rightarrow 將 Encoder 輸出的隱藏狀態反饋計算成 token
可能有三種 Type 的 Transformer

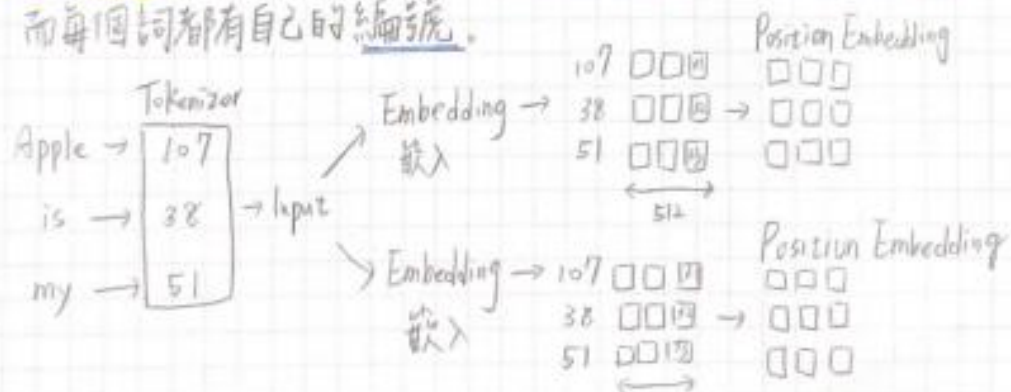
☆ 1. Encoder-only Transformers \rightarrow 強在可深入理解和解釋文本, 透過將輸入序列轉為各種數位形式, ex BERT / RoBERTa / DistilBERT
使用雙向注意力 \rightarrow 考慮前 / 後內容。

☆ 2. Decoder-only Transformer 此 Model 強在強大的輸出、表述能力, 透過反饋計算預測下一個 P_{max} 的單詞來完成輸出序列, GPT。
而此 Model 採 Autoregressive attention, 僅考慮已輸出的上文。

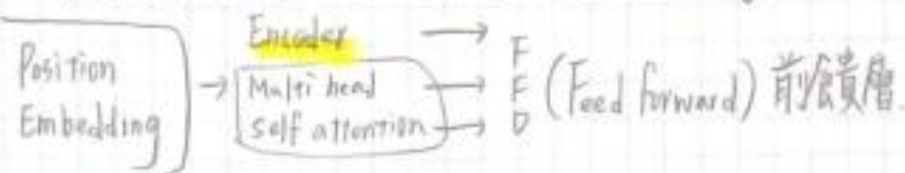
☆ 3. Encoder-decoder Transformer \rightarrow 同時擁有輸入文本、理解語意、輸出回應的能力, 如 T5 / BART, 可用於翻譯和產生摘要。

★ 標記化文本

☆ Tokenizer 分詞器 \rightarrow 使用 Tokenizer 將單詞轉為數位, 因機器只懂數位, 而每個詞都有自己的編號。



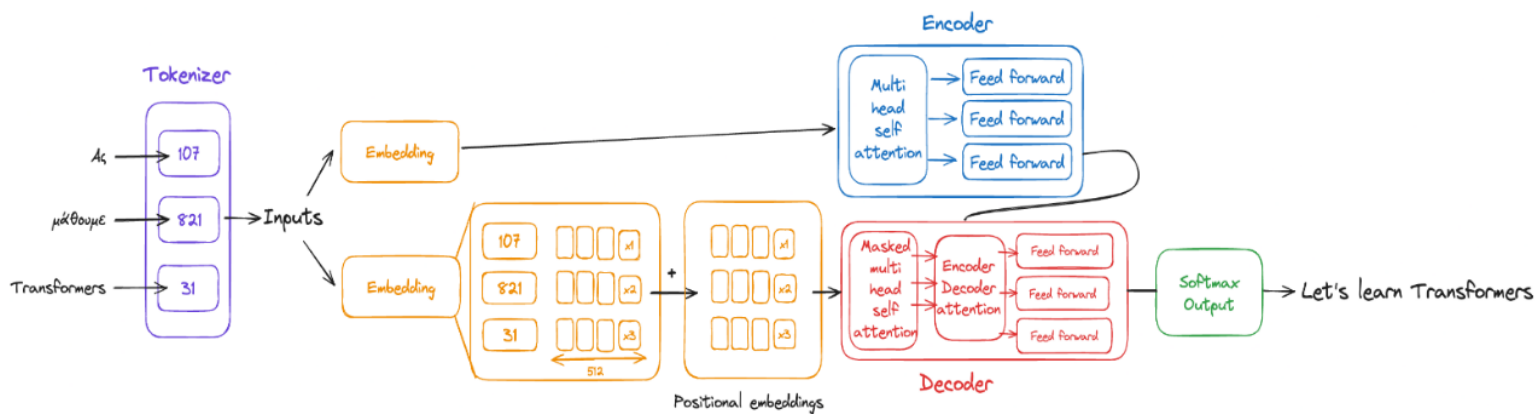
- ★ Embedding 嵌入 → 負責將被標記成數詞編號的詞轉成向量，一個標記為一個向量，而這些向量被存在一個「向量嵌入空間」，同時也會將單詞的位置記錄下來 (Position Embedding)，方便文意理解。



- ★ Encoder 編碼器 → 嵌入序列會 input 到編碼器中，這些 input 先經過 Multi head self-attention 負責注意機制，再到 Feed Forward，並輸入向量 (logits vector)。
 (就是把 self-attention 做多次，連接起來，並把這個向量 $\times W^0$ 得 x 。)
- ★ Multi head self-attention → 句子中各個詞會互相影響，要用不同角度去分析語意，所以是「Multi head 多頭」。
- ★ Feed-Forward 前饋 → 由兩層全連接神經網路組成，單獨處理嵌入的序列，並由 Transformer 最終輸出層生成 logit，通常 FFD 使用 Gelu 做為激活函數 (Gelu 讓數據可在線性和非線性中轉換)。
- ★ Decoder 解碼器 → 和編碼器相同，通常有多個 Decoder 組成，Decoder 同樣將嵌入序列的 input 放到 Multi head self-attention 再將 input 傳到 FFD，Decoder 和 Encoder 主要差別在於，Decoder 解碼器有 2 個注意子層。
Masked multi-head self-attention 蒙面多頭注意力機制：避免偷看答案，確保生成的 token 僅基於「過去輸出 (已產生的內容) 和 當前預測」的 token。各頭同樣代表詞間互相影響，並讓 Model 考慮上下文意。
- ★ Encoder-decoder attention → 讓 Model 專注在原文和譯文之間的轉換，此 decoder 會在輸出時考慮上下文以及目前為止輸入的內容，並輸出此時最相關的 Output (Token) (P Max)

★ Attention 機制：幫助機器提升翻譯質量

★ Transformer = 讓模型訓練並行，提升訓練效率。

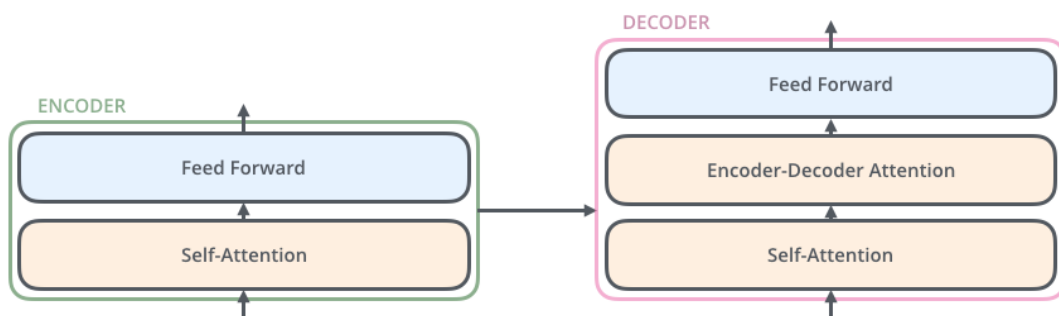


老師可能會給我們三個字，叫我們計算 hidden layer

Transformer——一種利用注意力來提高模型訓練速度的模型

<https://zjuturtle.com/2020/01/25/transformer/>

其中透過編碼器 encoder 和解碼器 decoder 來進行翻譯的動作，將輸入的句子語言進行編碼，並透過解碼器解碼成另一個語言



編碼器的輸入首先流經自注意力層 self attentions layer，該層幫助編碼器在對特定單字進行編碼時查看輸入句子中的其他單字(看需要注意的前後文)。

自注意力層的輸出被饋送到前饋神經網路 FFD。完全相同的前饋網路獨立應用於每個位置。

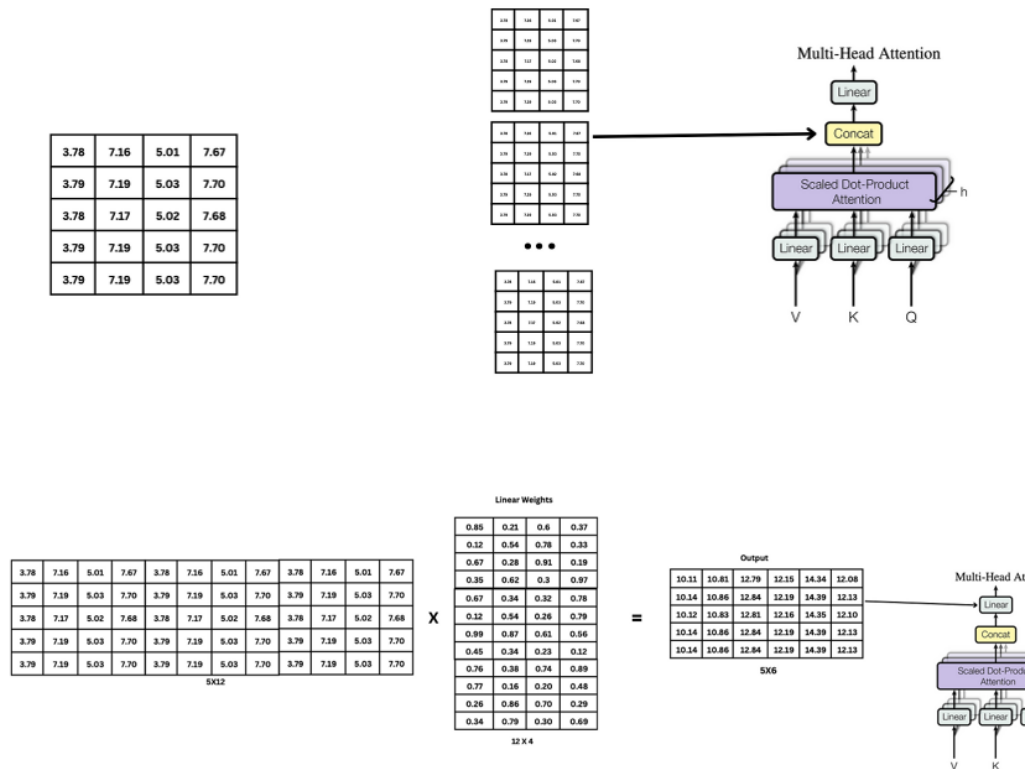
解碼器 decoder 具有這兩個層，但它們之間是一個注意力層，幫助解碼器專注於輸入句子的相關部分（類似於 seq2seq 模型中注意力的作用）。

題目也有 GAN

[Information, entropy, cross entropy, KL-Divergence.pdf](#)

<https://jianjiesun.medium.com/dl-ml%E7%AD%86%E8%A8%98-%E4%B8%89-kl-divergence-cross-entropy-4b48810e0e90>

[stable diffusion](#)



RNN

生成對抗網絡（簡稱 GAN）

深度神經網路架構，包括兩個互相競爭的網絡：**生成器（Generator）**和**判別器（Discriminator）**，GAN 通過對抗訓練，來讓模型生成與真實數據分佈相似數據的神經網絡架構。它由兩個主要組件構成：

1. 生成器（G）：

生成器接受一個**隨機噪聲向量**作為輸入，並生成**很像真實的假數據**。目標是生成能夠**騙過判別器的假數據**，使判別器無法區分這些數據是真是假。

2. 判別器（D）：

判別器接受一個數據樣本（可以是真實數據，也可以是生成數據）作為輸入，並區分輸入數據是真是假。目標是**讓模型判斷真假數據的準確度最大化**。

GAN 的訓練過程是生成器 G 和判別器 D 兩者的對抗過程：

1. **初始化權重**：隨機初始化 G 和 D 的權重。
 2. **訓練判別器**：使用一批真實數據和生成數據來訓練判別器，使其能夠正確區分真實數據和假數據。
 3. **訓練生成器**：只更新生成器 G 的權重，使生成器生成的數據能夠騙過判別器。
 4. **反覆進行**：重複上述步驟，在多個迭代中交替訓練生成器和判別器，直到生成器能夠生成非常逼真的數據。
- GAN 透過生成器 Generator 和判別器 **Discriminator** 的對抗訓練，學習並生成與真實數據分佈相似的數據，並在
 - 生成逼真的圖像
 - 藝術風格轉換
 - 數據增強
 - 圖像修復和超分辨率

等多個領域展現了其強大的應用潛力。

成本函數和最佳化

- 生成器的損失函數：

$$L_G = -\log(D(G(z)))$$

其中 D 是判別器，G 是生成器，z 是隨機噪聲向量。

- 判別器的損失函數：

$$L_D = -(\log(D(x)) + \log(1 - D(G(z))))$$

其中 x 是真實數據。