NLP觀念介紹

黃翔 – 9/26

前言

- **各模型的意涵/差異?**

此外，我們進一步針對新聞內容之情緒進行分析，透過比較字典法、深度學習演算法（Bi-LSTM），以及大型語言模型（包含 BERT、FinBERT 與額外預訓練之 FinBERT-ESGECT 模型）在句子情緒標籤的準確度，並分析新聞內容中，不同情緒特徵對累積異常報酬之影響，包含語氣、情緒力，以及文章內容的模糊程度與前瞻性論述的程度。結果顯示，大型語言模型的標籤正確率最佳，其中又以 FinBERT 和 FinBERT-ESGECT 的標籤正確率最高，其次依序為 BERT、Bi-LSTM、以及字典法；然而，在情緒特徵對累積異常報酬的影響方面，尤以模糊程度和語氣對累積異

文本轉換為數字

文字 -> 向量 -> 分析 / 建模

人類能夠通過上下文和語境理解句子的含義，識別語言中的各種語義、語法規則和抽象概念。對電腦來說，文字本質上只是一串符號，它無法直接理解其中的含義。因此，需要將「文字」轉化「數值」，讓計算機能夠處理和分析它們。

- **詞袋 (Bag of Words, BOW)**

詞袋模型將文本表示為「詞的集合」，忽略詞的順序與語法結構，只關注詞的出現頻率。在詞袋模型中，每個詞的順序並不重要，而是將文本視作一個詞的「袋子」。 作法：**BOW 作法：分詞 -> 前處理 -> 去除停用字 -> 字詞出現次數統計。**

1. **分詞 (Tokenization)**：將文章中的每個詞彙切開，整理成文字表。 I / love / natural / language / processing
2. **前置處理 (Preprocessing)**：詞形還原 (單字標準化)、轉換小寫等等。

```
#  1. 動詞 "running" 的詞根是 "run"。
#  2. 名詞 "better" 的詞根是 "good"。
```

3. **去除停用詞 (Stop Words)**：be動詞、助動詞、代名詞等等..不具特殊意義的詞彙，將他們刪除。
4. **詞彙出現次數統計**：計算詞彙在文章中出現的次數，並由高排到低。

字典法：分類問題

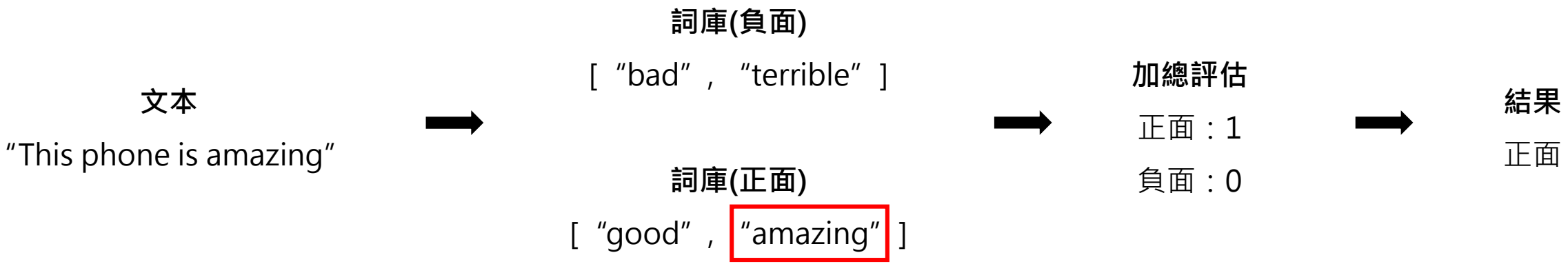
- **詞庫字典法**

詞庫是為了讓程式理解某種「特定文字」內容，事先蒐集並設計相關詞庫，再透過分析文句中相關詞句「出現次數」來進一步理解該段文句。

1. 建立詞庫：專家手動構建一個詞庫，這個詞庫包含了許多詞語，每個詞語都與一個特定的語義、情感或屬性相關聯。在情緒分析中，詞庫可能包含像 "good"、"excellent" 這樣的正面詞，以及 "bad"、"terrible" 這樣的負面詞。

2. 文本匹配：將文本中的詞與詞庫中的詞進行匹配。如果文本中的某個詞存在於詞庫中，就可以提取該詞對應的語義或情緒標籤。

3. 加總評估：根據匹配到的詞語及其對應的標籤，對整段文本進行加總評估。例如一篇評論中出現了很多正面詞語，那麼整體上可以判斷這篇評論是正面的。



改進BOW模型

- **TF-IDF (Term Frequency - inverse Document Frequency)**

BOW 模型的問題是某些常見的詞，雖然不是停用詞，但也會在每個文檔中頻繁出現，且對區分文檔的重要性很低。所以提出 TF-IDF 演算法，他會針對「跨文件常出現」的詞彙給較低的分數 (如果 only 在每個文章都有出現，TF-IDF的分數就較低)

1. **字詞頻率 (Term Frequency, TF)**：衡量一個詞在一個文檔中的出現次數，出現頻率越高，TF 值越大。

2. **逆文件頻率(Inverse Document Frequency, IDF)**：衡量一個詞在整個語料庫中的普遍性。那些在很多文檔中都出現的詞（如“olny”）被賦予較低的權重，而只在「少數文檔」中出現的詞應該賦予較高的權重。

不同文檔

doc_1: "I love eating an apple every day“

doc_2: "She bought an apple and an orange"

doc_3: "Apple pie is delicious“

doc_4: "He prefers bananas to guavas"

➡

計算 “apple” TF(詞頻)

doc_1: 1/6

doc_2: 1/7

doc_3: 1/4

doc_4: 0/5

➡

計算 “apple” IDF

文章總數：4

DF(包含”apple”文章)：3

IDF = log(4/3) = 0.1249

➡

計算 TF-IDF

doc_1: 1/6 * 0.1249

doc_2: 1/7 * 0.1249

doc_3: 1/4 * 0.1249

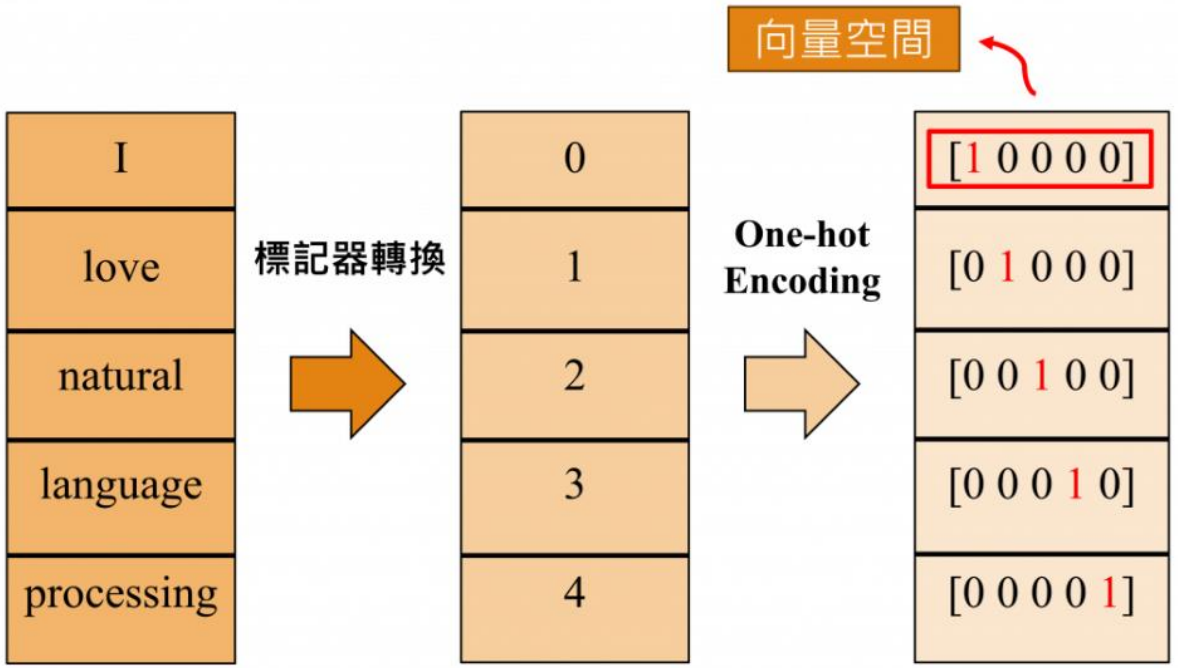
doc_4: 0/5 * 0.1249

語義關係

- **詞嵌入（Word Embedding）**

BoW 和 TF-IDF 問題是忽略了「語義關係」。例如「cat」和「dog」會被視為完全不相關的詞，因為它們只是單純的符號，沒有考慮詞語的語義相似性。詞嵌入將詞語表示為「低維的稠密向量」，這些向量不僅表示詞語的出現頻率，還能捕捉到詞語之間的**語義關係**。相比於BOW生成的高維稀疏向量，詞嵌入技術生成的向量是**低維稠密向量**，且具有語義上的結構性。

稀疏矩陣舉例：BOW 的 One-Hot Encoding



低維稠密向量舉例：詞嵌入生成的向量

假設用3 維空間來表示詞語 (實際常 100 ~ 300 維)

“apple" 被表示為 [1, 0.5, 0.7]

“fruit” 被表示為 [1, 0.5, 0.8]

1維表示「食物與否」，如果是食物，值接近1。

2維表示「固體與否」，固體食物可能接近0.5，液體食物接近0。

3維表示「甜味」，值越大表示越甜。

考慮詞的順序

- **上下文**

  BOW 將文本視為無序的詞袋，無法考慮詞與詞之間的**順序**。而 Word Embedding 雖然解決了詞向量的表示問題，能夠捕捉詞與詞之間的語義關係，但它本質上依然無法處理詞序問題。例如，處理「貓追狗」和「狗追貓」時，它無法理解詞語之間的**順序差異**。也無法根據上下文變化來**調整詞語的語義**。例如，詞「bank」可以表示「銀行」或「河岸」，但在 Word Embedding 中，它的向量是固定的，不能隨著上下文的不同改變其含義。

- **記憶力**

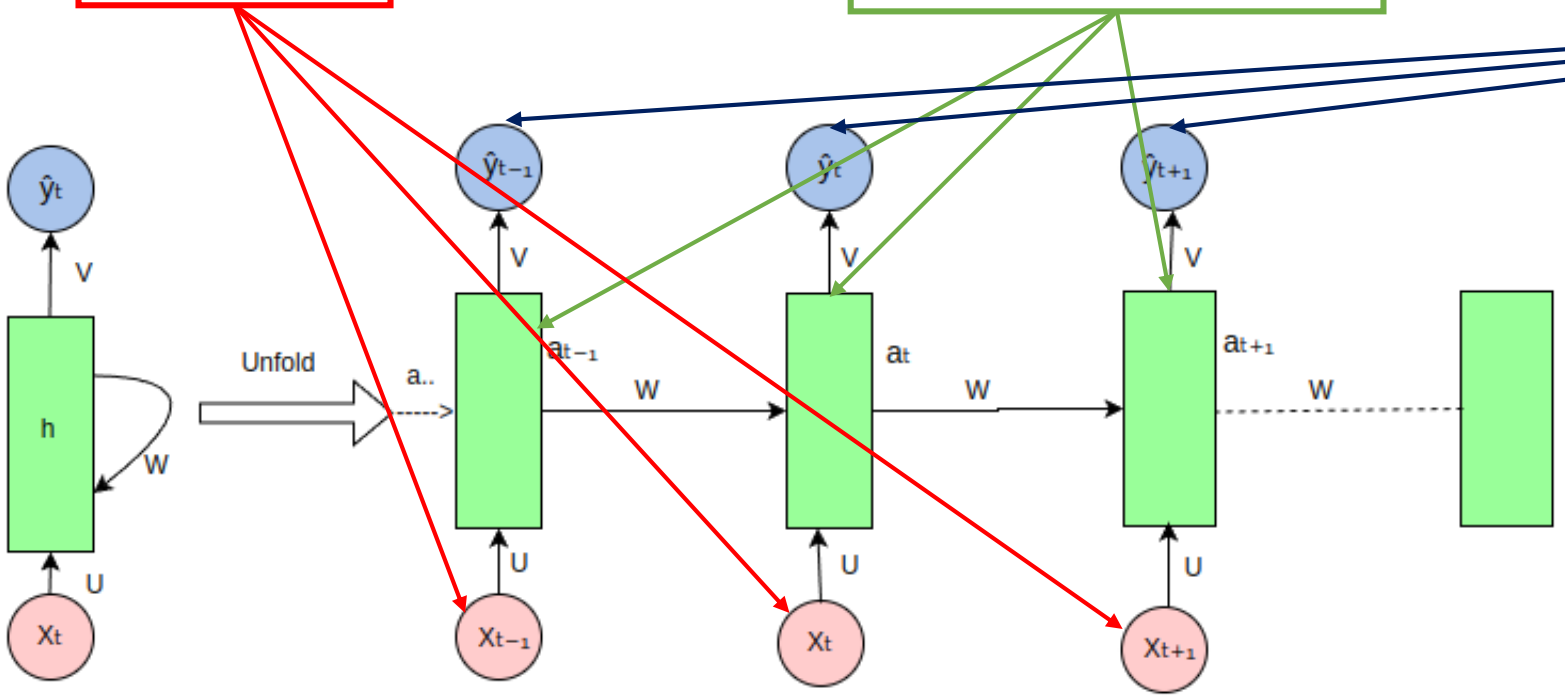
  自然語言的推斷除了要考慮上下文的關係以外，還要考慮**「記憶力」**。例如：「小明今天很累，他打算早點睡覺。」，我們能理解「他」指的是「小名」，就是因為**「記憶力」**，或稱**時間相關問題**，就是當下的答案會受過去的答案影響，而且也會影響未來的答案的一種特殊情況。

考慮詞的順序

- **循環神經網絡（Recurrent Neural Networks, RNN）**

  RNN 是為了解決序列數據而設計的。RNN 能夠記住前面的信息並將其應用於後面的計算中。RNN 引入了一個「循環」的概念，即它會在處理序列中的每一個詞時，將前一個詞的信息傳遞給下一個詞，從而考慮到句子的上文。

RNN 具有一個「隱藏層(Hidden State)」，這個隱藏層負責保留序列數據中的歷史信息。在每個時間步，RNN 都會接收一個輸入，然後基於當前輸入和前一個時間步的隱藏狀態來更新新的隱藏狀態，並產生當前的輸出。



權重 * 上時的隱藏狀態 bias

$$h_t = W * h_{t-1} + Ux_t + b$$

此時隱藏狀態 權重 * 此時的輸入

Activation Function

$$y_t = g(V * h_t)$$

此時輸出 (權重 * 此時隱藏狀態)

RNN問題：長期依賴問題、梯度消失

- **長期依賴性（long-term dependency）**

長期依賴性（long-term dependency）是指網路在很長的序列中，例如：“The dog that I saw yesterday was running fast”，處理到“running”時，會需要「主語」這一信息，但由於“dog”在很久之前就出現了。網路無法有效地記住。

- **梯度消失(Vanishing Gradient problem)**

梯度消失是指當反向傳播的梯度在傳遞過程中變得越來越小，最終接近於零，這會導致網路的權重更新非常緩慢，甚至無法更新。在 RNN 的運算中，每個時間在用上一個時間隱藏狀態(ht-1)來更新當前的隱藏狀態(ht)，這涉及到激活函數(例如Sigmoid函數導數在0~1)，它們的導數通常小於 1。當梯度隨著時間步不斷地進行反向傳播時，這些小於 1 的導數會被不斷相乘，導致梯度變得越來越小。

例如：序列長度(t) = 4, 使用 Sigmoid激活函數, 權重矩陣 W = 0.5

h1 = Sigmoid (0.5 * h0)

h2 = Sigmoid (0.5 * h1)

h3 = Sigmoid (0.5 * h2)

h4 = Sigmoid (0.5 * h3)

1. **從 h4 反向傳播：∂L/∂h4 = (算得出來，假設為1)**，因為已知輸出y4，所以可以計算出L，因此，可以對L的h4微分，得知h4對L的影響程度
2. **∂L/∂h3** = 1(已算出的∂L/h4) * 0.5 * 0.25(假設sigmoid的導數) = 0.125
3. **∂L/∂h2** => 0.125(∂L/∂h3) * 0.5 * 0.25 = 0.015625 (之後省略)

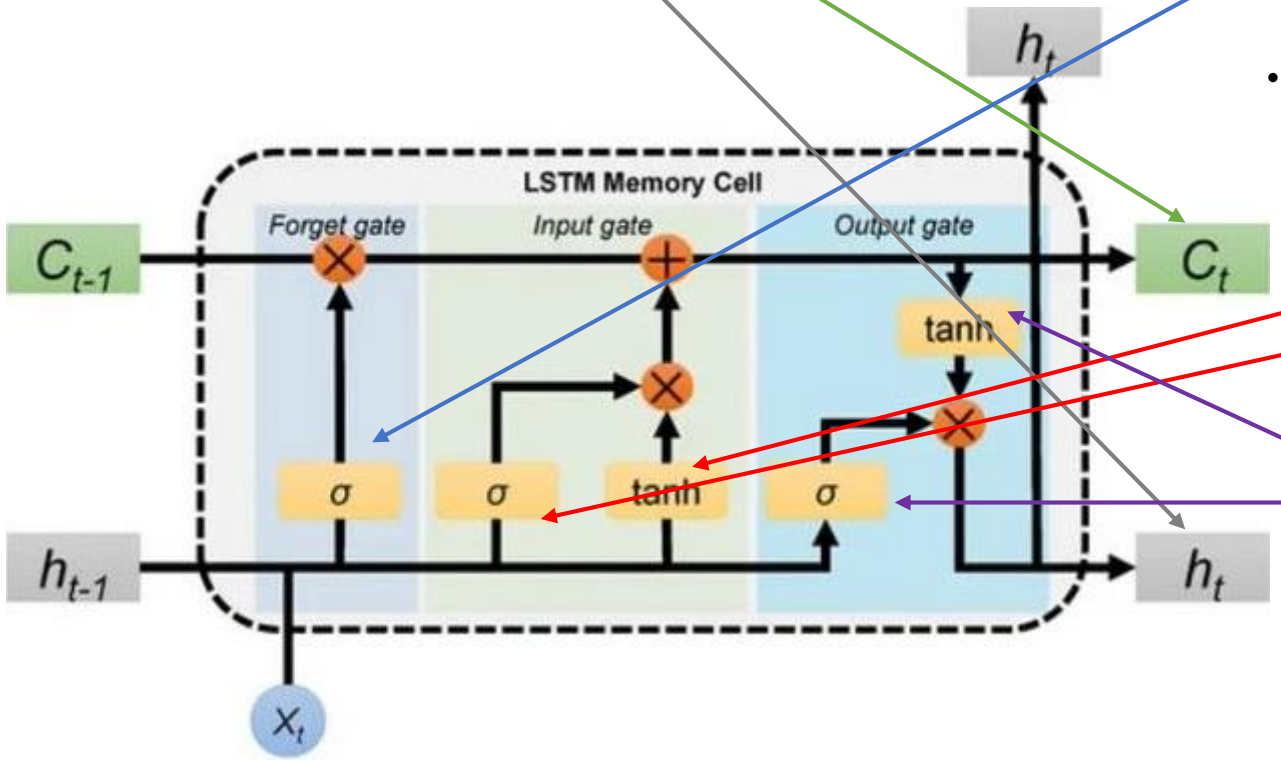
超小的梯度幾乎無法更新網路的權重，導致 RNN 無法記住早期的輸入(特別是當序列很長時)。早期的關鍵信息會被「忘記」

解決長期依賴問題

- **長短期記憶網絡（Long Short-Term Memory, LSTM）**

LSTM 專門設計來解決 RNN 的梯度消失問題。LSTM 通過引入記憶細胞（Memory Cell），能夠選擇性地記住和遺忘信息，從而有效保留序列中的長距上下文信息。以及三種控制信息流動的門(GATE)。

- **狀態保存層(Cell State, C_t)**：用來保存長期記憶。
- **隱藏狀態(Hidden State, h_t)**：用來處理和保存當前時間步以及過去幾個時間步的短期記憶。



- **遺忘門(Forget Gate)**：遺忘門根據當前的輸入和上一個時間步的隱藏狀態，決定應該忘記多少來自上一時間步的長期記憶(Cell State)。通過 sigmoid 函數 σ，輸出一個0到1之間的數值，0表示完全忘記，1表示完全保留。

- **輸入門(Input Gate)**：輸入門控制應該將多少當前輸入加到Cell state。分為兩部分：一個是控制引入多少新信息，另一個是生成這些新信息。
  - **Candidate Cell State**：反映了當前輸入和上下文結合後的新信息，「候選」的記憶更新內容，表示當前時刻可以引入的新信息
  - **Input gate**：決定Candidate Cell State 中的多少內容應該引入多少。也是0-1之間。

- **輸出門(Output Gate)**：決定當前輸入是否加到「隱藏狀態的輸出」，這個門也是Sigmoid函數，表示要加入與否。
  - 最後針對長期記憶**(Cell state)**是否加到隱藏狀態的輸出(Output)，通常使用tanh函數，數值會落在[-1, 1]之間，-1表移除長期記憶。

解決長期依賴問題

- **遺忘門(Forget Gate)**：遺忘門決定 LSTM 在上一時間步狀態保存層 Ct-1 需要「忘記」多少的記憶。通過 sigmoid 函數 σ，輸出一個0到1之間的數值，0表示完全忘記，1表示完全保留。

sigmoid　　上一個時間步的ht和當前入，接成一個向量

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

Ct保留比例　　線性轉換　　bias

- **輸入門(Input Gate)**：輸入門控制應該將多少當前輸入加到Cell state。分為兩部分：一個是控制引入多少新信息，另一個是生成這些新信息。
  - **Candidate Cell State**：反映了當前輸入和上下文結合後的新信息，「候選」的記憶更新內容，表示當前時刻可以引入的新信息
  - **Input gate**：決定Candidate Cell State 中的多少內容應該引入多少。也是0-1之間。

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

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

更新

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

上階段忘不忘?　　(共同作用)　　這階段記不記?

- **輸出門(Output Gate)**：決定當前輸入是否加到「隱藏狀態的輸出」，這個門也是Sigmoid函數，表示要加入與否。
  - 最後針對長期記憶(**Cell state**)是否加到隱藏狀態的輸出(Output)，通常使用tanh函數，數值會落在[-1, 1]之間，-1表移除長期記憶。

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

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

當前輸入貢獻　　長期記憶貢獻

最終輸出

「雙向」LSTM

- **BiLSTM (Bi-directional LSTM)**

  BiLSTM是由前向 LSTM 和後向 LSTM 组合而成的「雙向」LSTM。在一個序列的輸入中，一個從序列的前向（從左到右）處理數據，另一個從序列的反向（從右到左）處理數據。因此，雙向 LSTM 可以同時考慮數據的**前後文**，即能夠利用整個序列中的信息，不僅僅是當前時間步及之前的資訊。

- **前向 LSTM（Forward LSTM）**：從左到右，計算每個時間步的隱藏狀態。

用 t-1 計算 t

$$\overrightarrow{h}_t = \mathrm{LSTM}(x_t, \overrightarrow{h}_{t-1})$$

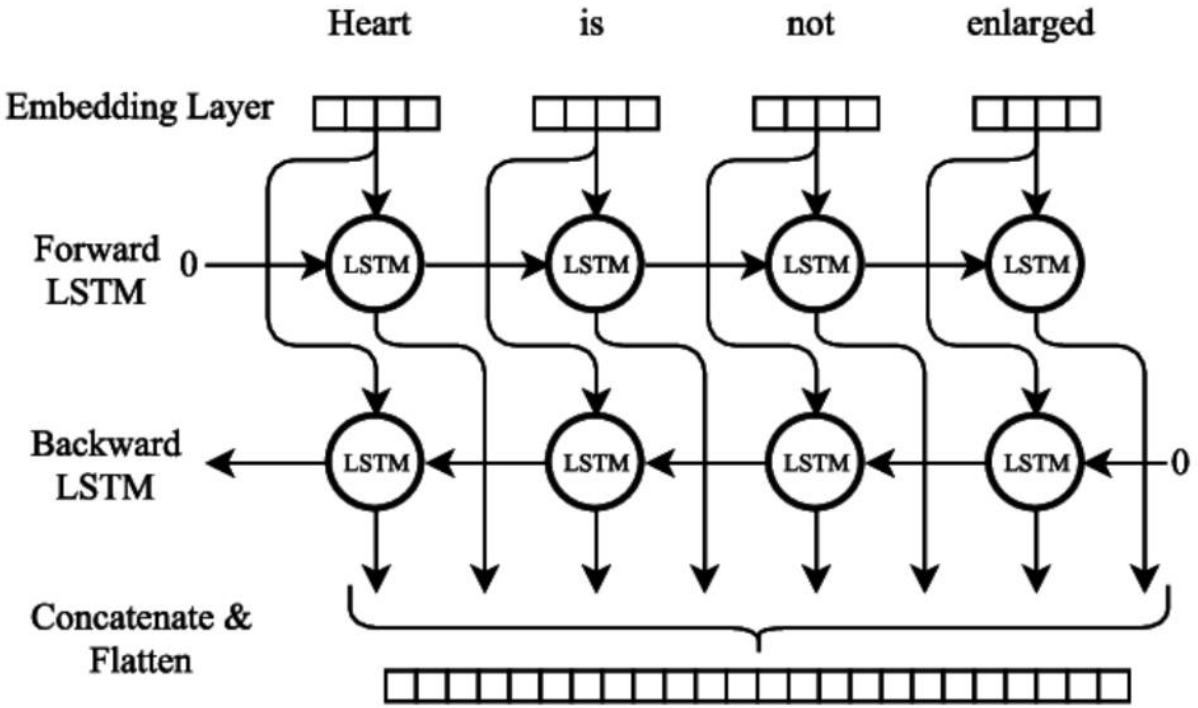
- **反向 LSTM（Backward LSTM）**：從左到右，計算每個時間步的隱藏狀態。

用 t+1 計算 t

$$\overleftarrow{h}_t = \mathrm{LSTM}(x_t, \overleftarrow{h}_{t+1})$$

- **合併隱藏狀態**：將前向 LSTM 和反向 LSTM 的隱藏狀態拼接在一起。最終的隱藏狀態 ht同時包含了來自前文和後文的信息。

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$$



「雙向」LSTM

- **前向 LSTM（Forward LSTM）**：從左到右，計算每個時間步的隱藏狀態。

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

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

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

$$\overrightarrow{c_t} = \overrightarrow{f_t} \circ \overrightarrow{c_{t-1}} + \overrightarrow{i_t} \circ \overrightarrow{\tilde{c}_t}$$

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

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

- **反向 LSTM（Backward LSTM）**：從左到右，計算每個時間步的隱藏狀態。

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

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

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

$$\overleftarrow{c_t} = \overleftarrow{f_t} \circ \overleftarrow{c_{t+1}} + \overleftarrow{i_t} \circ \overleftarrow{\tilde{c}_t}$$

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

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

- **合併隱藏狀態**：將前向 LSTM 和反向 LSTM 的隱藏狀態拼接在一起。最終的隱藏狀態 ht同時包含了來自前文和後文的信息。

$$h_t = \left[\overrightarrow{h}_t; \overleftarrow{h}_t\right]$$

水平拼接：例如

$\overrightarrow{h}_t$ = [0.2, 0.5, 0.4]

$\overleftarrow{h}_t$ = [0.7, 0.2, 0.3]

$\left[\overrightarrow{h}_t; \overleftarrow{h}_t\right]$ = [0.2, 0.5, 0.4, 0.7, 0.2, 0.3]

Transformer 發展背景

- **LSTM 的局限性**

LSTM 通過引入「門」，能夠在一定程度上解決 RNN 的「梯度消失」和「長期依賴」。但 LSTM 仍然存在局限：

- **序列性**：LSTM 必須按時間步逐一處理輸入，無法並行計算，導致訓練速度較慢。
- **長距依賴**：儘管 LSTM 能夠捕捉長期依賴，但隨著序列長度的增加，模型的能力仍然有限。
- **記憶瓶頸**：LSTM 依賴於隱藏狀態(ht)來傳遞信息，可能會導致信息在長時間步中逐漸遺失。

- **注意力機制**

為了解決上述問題，**注意力機制（Attention Mechanism）** 被引入到序列模型中。注意力機制允許模型在處理當前時間步時，「動態」的選擇並關注輸入序列中「最相關」的部分。

- **Transformer 誕生**

為了徹底解決序列性計算的限制，Vaswani 等人在 2017 年提出了 Transformer 模型，發表了題為 "Attention is All You Need" 的論文。Transformer是專門用於語言類別的一種架構。完全基於注意力機制，能夠並行處理序列數據，大大提高了模型的訓練效率和性能。

Transformer

- **Transformer 架構**

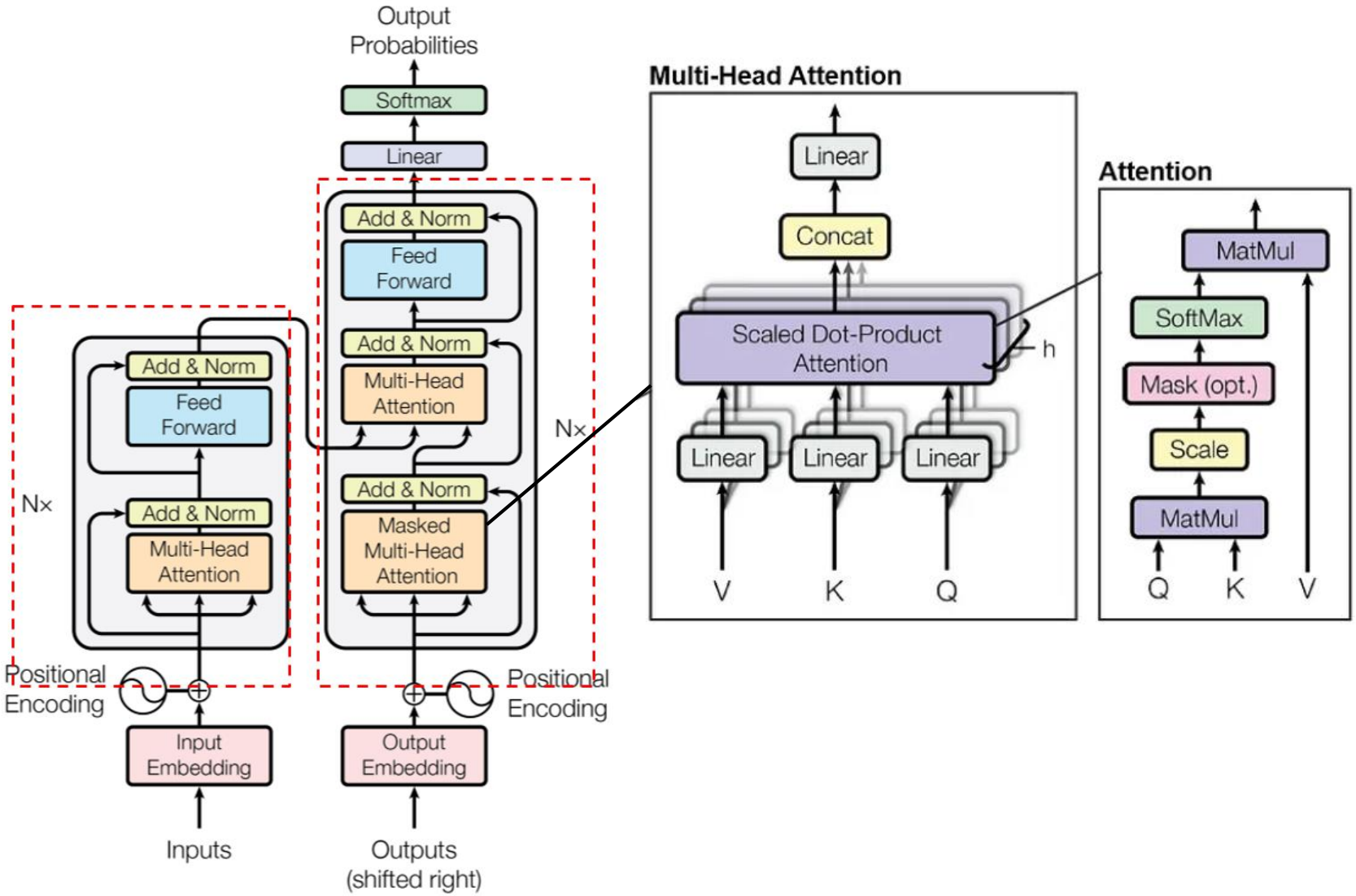
1. 模型的輸入與輸出
2. 位置編碼 (Position Encoding)
3. 編碼器-解碼器結構 (Encoder-Decoder)
4. 線性層 (Linear)、softmax層

- **Encoder 結構**

1. 多頭注意力機制 (Multi-Head-Attention)
2. 前饋神經網路（Feed Forward）
3. 殘差連接和標準化（Add & Norm）

- **Decoder 結構**

1. 遮蔽多頭注意力機制
2. 前饋神經網路
3. 殘差連接和標準化



Transformer

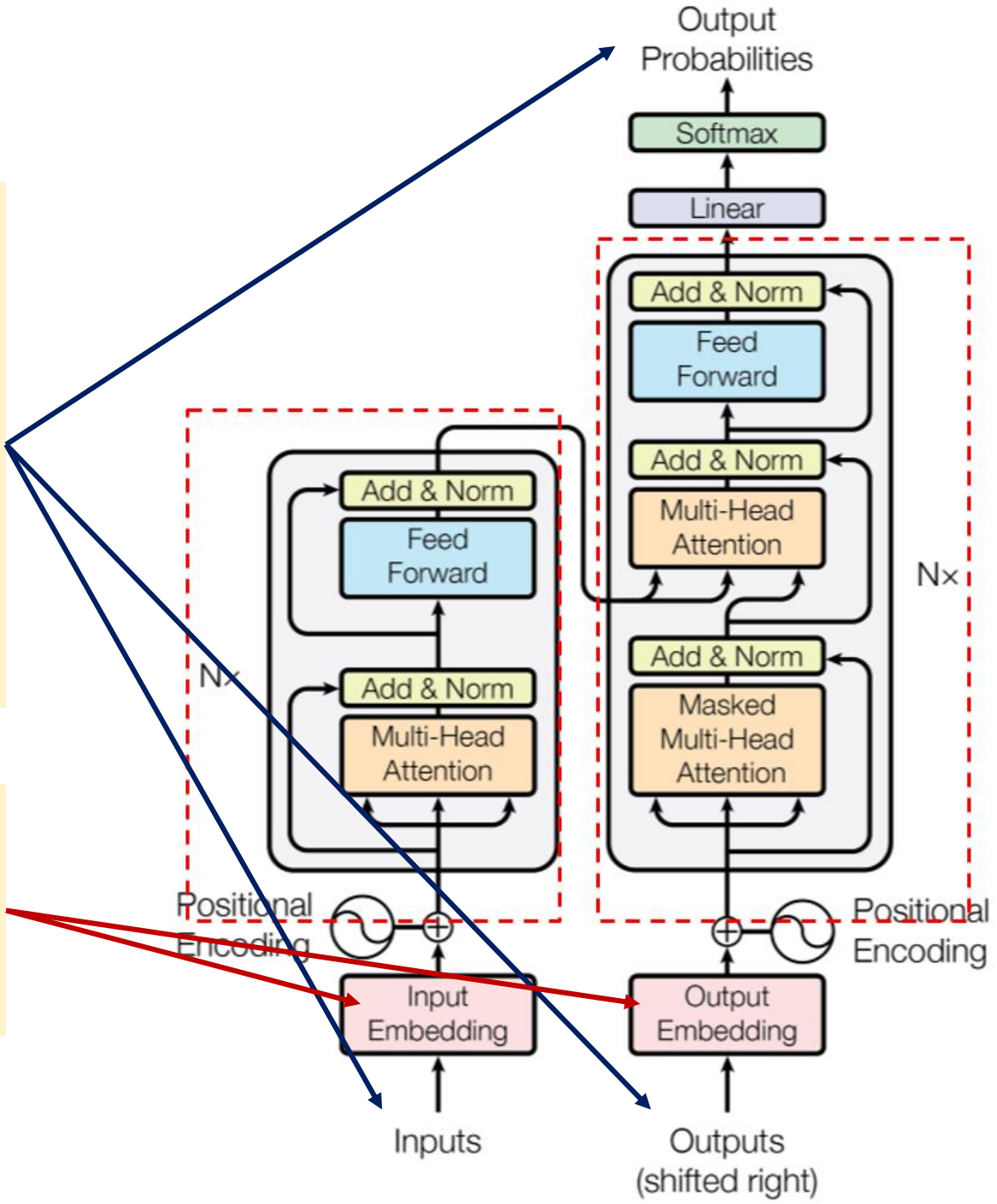
- **模型的輸入與輸出**

1. 輸入是下方的 input 和 output
2. 輸出是右上方的output probabilities

- **Input**：訓練模型時從 input 輸入英文數據。
- **Output**：從 output 輸入「翻譯後的中文」標註數據。
- **模型的翻譯預測結果，會從右上方的output位置輸出。**

- **詞向量層**

在文字進入模型後，會被 transformer 的第一個組件**詞向量層**進行處理，單詞會被轉換為向量矩陣。



Transformer

- **位置編碼 (Position Encoding)**

- 對模型來說，假設輸入 ” Are you ok?” 時，與輸入 you Are ok?、you ? Are ok ... 之間是沒有任何區別的，因為在轉換成向量矩陣之後，單詞之間是沒有順序性的，模型感受不到詞與之間的順序。

- 位置編碼會用**正弦函數**和**餘弦函數**的計算來找出單字的正確位置。

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

- **pos**：表示單字在句子中的位置。(Are->0, You->1, ok->2, ?->3)
- **i**：是維度索引。(根據word embedding的維度，去進行同維度的編碼)
- **dmodel**：總維度數。這裡假設為4

單詞詞向量矩陣

Are	0.4002	0.9882	0.6261	0.1502
you	0.3413	0.9284	0.9349	0.8958
OK	0.5347	0.2969	0.7019	0.6293
?	0.8704	0.4956	0.3794	0.5646

+

位置編碼矩陣

0.0000	1.0000	0.0000	1.0000
0.8415	0.9950	0.0100	0.9999
0.9093	0.9801	0.0200	0.9999
0.1411	0.9553	0.0300	0.9999

偶數維度（索引為 $2i$）：使用 正弦函數（sin）
奇數維度（索引為 $2i$+1）：使用 餘弦函數（cos）

用 Are you ok? 中的 you 為例，you 所處的位置為1，而 you 這個單字有四個維度，因此需要計算PE(1,0)、PE(1,1)、PE(1,2)、PE(1,3)等四個維度，然後與剛剛在詞向量層中轉換的詞向量矩陣相加

單詞詞向量矩陣

you	0.3413	0.9284	0.9349	0.8958

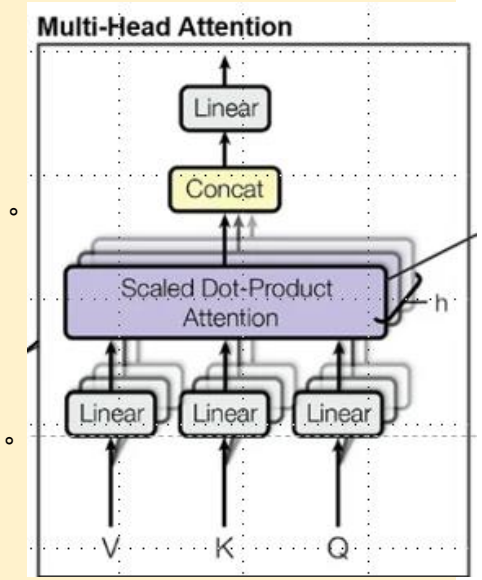
+

位置編碼矩陣

0.8415	0.9950	0.0100	0.9999

Transformer

- **編碼器-解碼器結構(Encoder-Decoder)**
- **編碼器(Encoder)**：將輸入序列轉換為一個隱藏表示，這個表示會用於 Decoder 生成輸出。 Encoder 主要由**「多頭自注意力機制」**與**「前向全連接層」**組成。
- **多頭自注意力機制 (Multi-Head Self-Attention)**：假設輸入序列為：$X = [x_1, x_2, ..., x_n]$
  - 每個 Xi 是剛剛處理完的向量。
  - dmodel = 4 (假設)
- **查詢（Query）、鍵（Key）、值（Value）**：每個向量會通過線性變換生成三矩陣。
  1. **Query矩陣**：表示當前詞希望「查詢」句子中其他詞的相關性。
  2. **Key矩陣**：表示句子中其他詞的特徵，用來跟 Query 詞進行匹配。
  3. **Value矩陣**：代表具體的語義，有了注意力權重(從QK相似度計算而來) 後，將根據這些權重來加權計算結果。



- **注意力得分計算**：注意力得分是通過查詢與鍵的點積來計算，並經過縮放和 softmax 歸一化。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

$\frac{1}{\sqrt{d_k}}$ 是縮放因子，用來防止內積過大。

- **多頭注意力機制**：將 Q、K、V 分為 h 個頭，每個頭執行自己的注意力計算，最後，將這些頭的輸出拼接在一起。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O$$

W^O 是線性轉換的矩陣。

Transformer

- **編碼器-解碼器結構(Encoder-Decoder)**
- **Layer Normalization**：在多頭自注意力機制的輸出之後，會將其與輸入進行殘差連接。可以穩定數值分佈，提升訓練效率。

$$z_{\text{attn}} = \text{LayerNorm}(X + \text{MultiHead}(Q, K, V))$$

- **前向全連接層（Feed-Forward Neural Network, FFN）**：每個位置的輸出向量 z_{attn} 會通過一個前向全連接層，這是一個兩層的 MLP（多層感知機），中間包含一個 ReLU 激活函數：

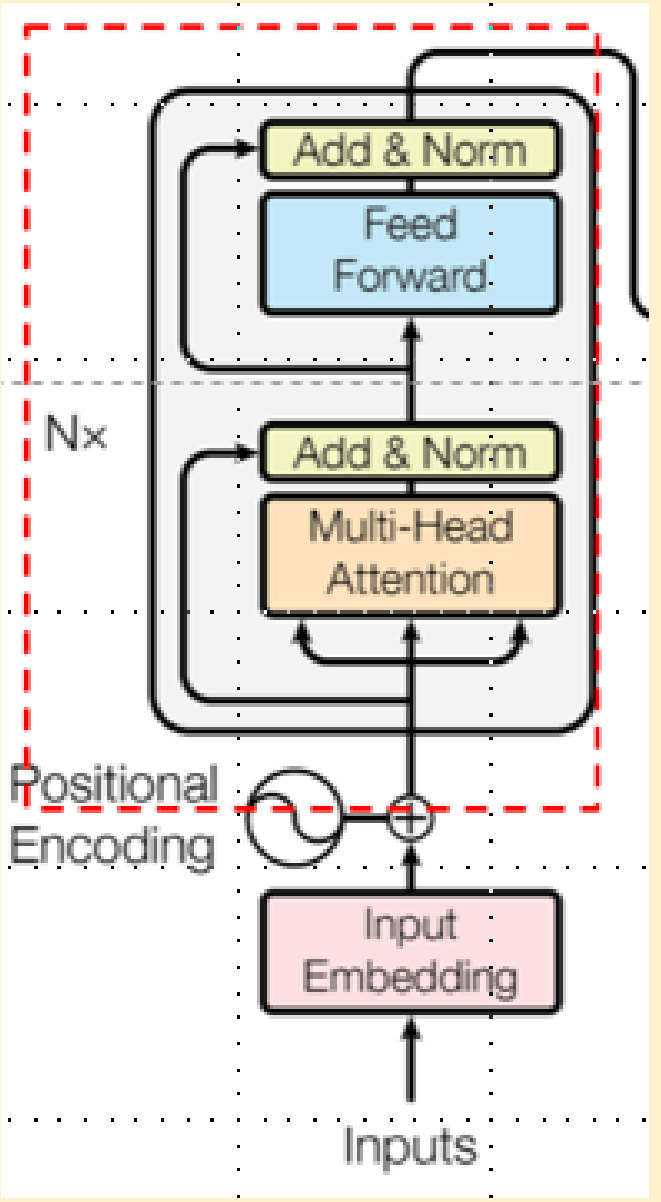
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- **第二次 LayerNorm**：

$$z_{\text{ffn}} = \text{LayerNorm}(z_{\text{attn}} + \text{FFN}(z_{\text{attn}}))$$

- **Encoder 輸出**：經過多層（典型為6層）這樣的處理後，編碼器會生成一個隱藏狀態序列，這些隱藏向量將會輸入給 Decoder 使用。

$$\mathbf{H} = (h_1, h_2, \ldots, h_n)$$



Transformer

- **編碼器-解碼器結構(Encoder-Decoder)**
- **解碼器(Decoder)**：Decoder的任務是根據 Encoder 生成的隱藏表示，逐步生成最終的輸出序列。Decoder 的每一層與 Encoder 類似，但額外包含了一個編碼器-解碼器注意力機制（Encoder-Decoder Attention），使 Decoder 能夠關注到 Encoder 的輸出。

- **自注意力機制 (Self-Attention)**：與 Encoder 中的自注意力機制類似，Decoder 會首先對自己生成的序列進行自注意力處理。但在這裡，會進行遮蔽（masking），以確保每個位置僅能關注到它之前的單詞，避免模型「偷看」未來的信息。遮蔽操作是通過將未來的位置賦予一個非常大的負值，然後在 softmax 步驟中將這些位置的概率壓為 0。

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + \text{mask}\right)V$$

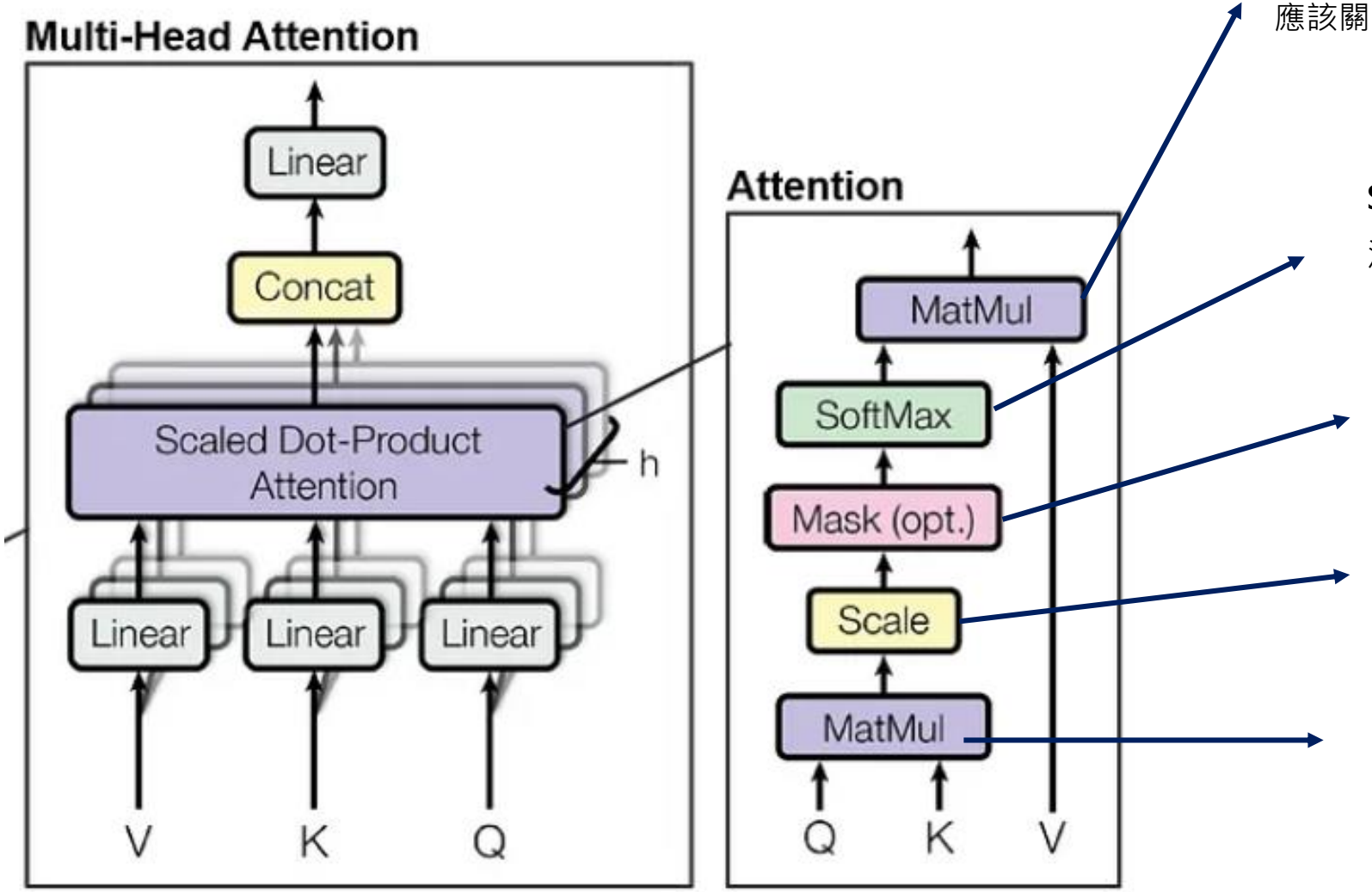
mask 是一個矩陣，對於未來時間步給予非常大的負值。

- **編碼器-解碼器注意力機制 (Encoder-Decoder Attention)**：Decoder 會根據自己生成的序列與 Encoder 輸出的隱藏表示進行注意力計算。

$$\text{Attention}(Q_{\text{decoder}}, K_{\text{encoder}}, V_{\text{encoder}}) = \text{softmax}\left(\frac{Q_{\text{decoder}} K_{\text{encoder}}^\top}{\sqrt{d_k}}\right)V_{\text{encoder}}$$

- **前向全連接層（Feed-Forward Network, FFN）**：和 Encoder 一樣，Decoder 每層也包含一個前向全連接層，經過殘差連接與層歸一化進行進一步的處理。

Transformer



矩陣乘法：將計算出的注意力權重與值向量 V 相乘，從而得到最終的注意力輸出。這樣，每個輸出的向量是基於這個單詞應該關注哪些其他單詞以及每個單詞的值向量來計算的。

Softmax：計算注意力權重，表示每個單詞應該關注其他單詞的程度。

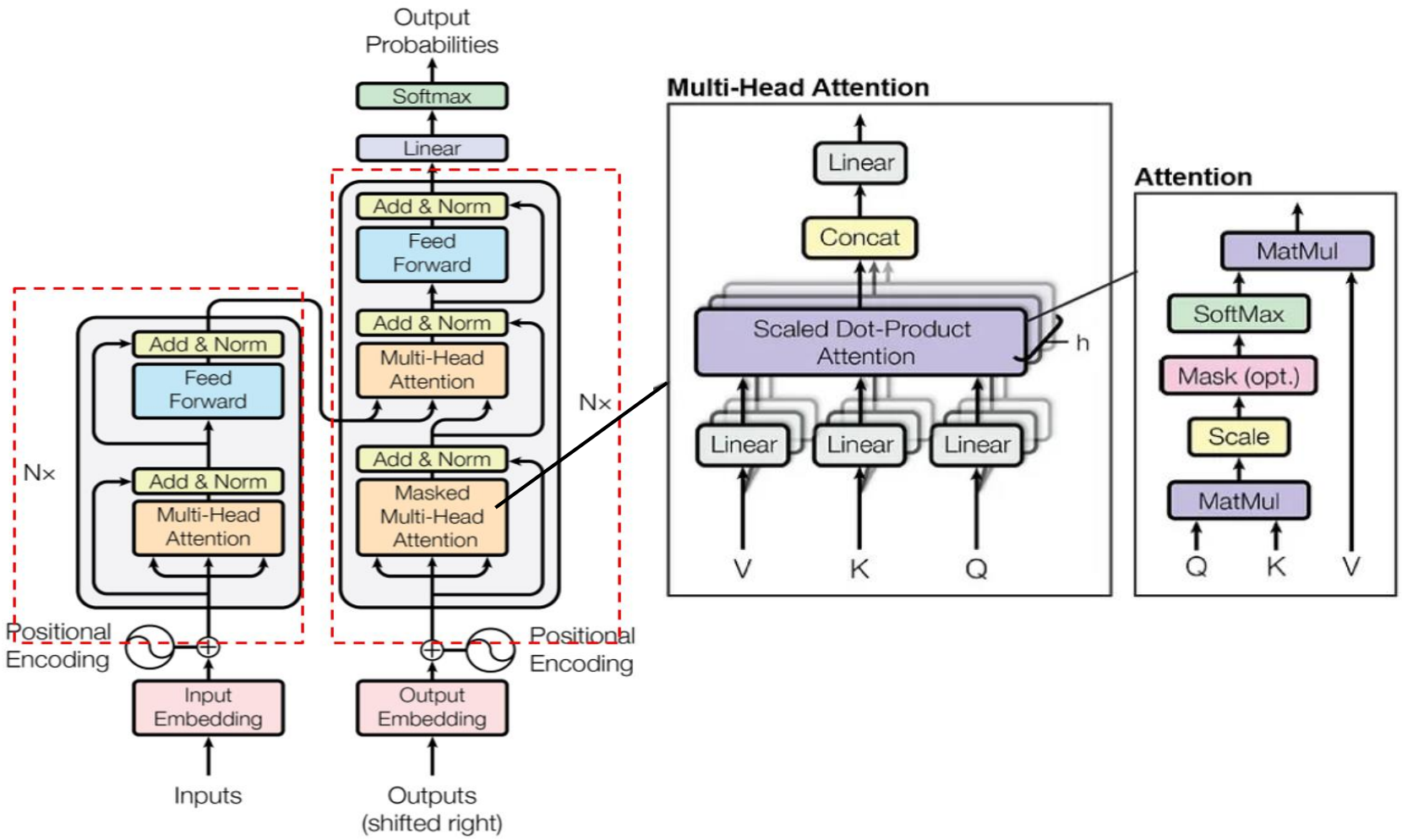
遮蔽：阻止模型看到未來的單詞。

縮放：除以 $\frac{1}{\sqrt{d_k}}$

矩陣乘法(點積)：計算查詢向量和鍵向量的點積。(計算相似性)

Transformer

Transformer 模型的創新在於它**摒棄了傳統 RNN 和 LSTM 的順序計算**，依賴於**全局的自注意力機制**來並行處理序列中的每個位置，從而大幅提高了計算效率。這使得 Transformer 能夠在捕捉長距依賴關係上表現出色，解決了 LSTM 隨著序列長度增長而面臨的「長期依賴問題」和計算瓶頸。



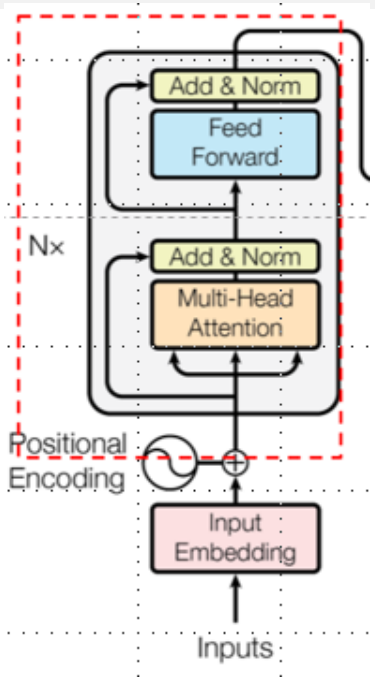
BERT 

- **BERT（Bidirectional Encoder Representations from Transformers）**

Google 在 2018 年提出的 NLP 模型，BERT 完全基於 Transformer 編碼器（Encoder），而不使用解碼器 (Decoder)部分。Transformer 編碼器可以並行處理整個序列，並通過自注意力機制捕捉每個單詞與其他單詞的關聯性。這使得 BERT 能夠非常高效地處理長距依賴關係，以及能夠同時考慮上下文，通過雙向自注意力機制，BERT 能夠完整地捕捉句子的全局信息。

- **輸入表示（Input Representation）**

在 BERT 中，輸入是一系列 Token 並且以 Embedding 的型態表示，在 Transformer Block 中進行處理與學習， 輸出是一個大小為 H 的 Embedding Sequence，其中每個 Embedding 對應到的就是他輸入文字經過轉換後的 Representation。 $\mathbf{H} = (h_1, h_2, ..., h_n)$



- **BERT 的輸入包含三部分：單詞嵌入、段落嵌入、位置嵌入。**

1. **單詞嵌入（Token Embedding）**：每個單詞經過分詞器（WordPiece Tokenizer）後被映射為一個向量表示，這使得模型能夠處理未知詞彙。

2. **段落嵌入（Segment Embedding）**：BERT 可以同時處理兩個句子，因此需要標記每個單詞屬於句子 A 還是句子 B。

3. **位置嵌入（Position Embedding）**：與 Transformer 一樣，BERT 也需要位置編碼來標識每個單詞在序列中的位置，因為模型本身並沒有內建的序列信息。

BERT 

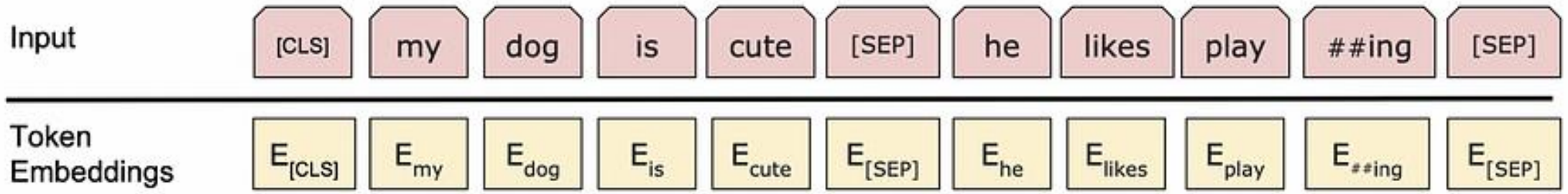
單詞嵌入（Token Embedding）：

每個單詞經過**分詞器(WordPiece Tokenizer)**後被映射為一個向量表示，這使得模型能夠處理未知詞彙。

- **WordPiece 分詞：**

為了解決詞彙表過大或單詞過於罕見的問題，BERT 不直接使用整個單詞作為輸入，而是將單詞切分為更小的子單位（子詞）。例如「unhappiness」會被切分為「un」「##happy」「##ness」，這樣模型可以學習到子詞的語義信息，並且能處理罕見詞或未知詞。

- **[CLS] 與 [SEP] 標記：**
  - [CLS]：每個輸入序列的第一個位置都是一個特殊的標記 [CLS]，表示句子的開始。
  - [SEP]：當 BERT 處理兩個句子時，會用 [SEP] 標記分隔兩個句子。



BERT 

段落嵌入（Segment Embedding）：

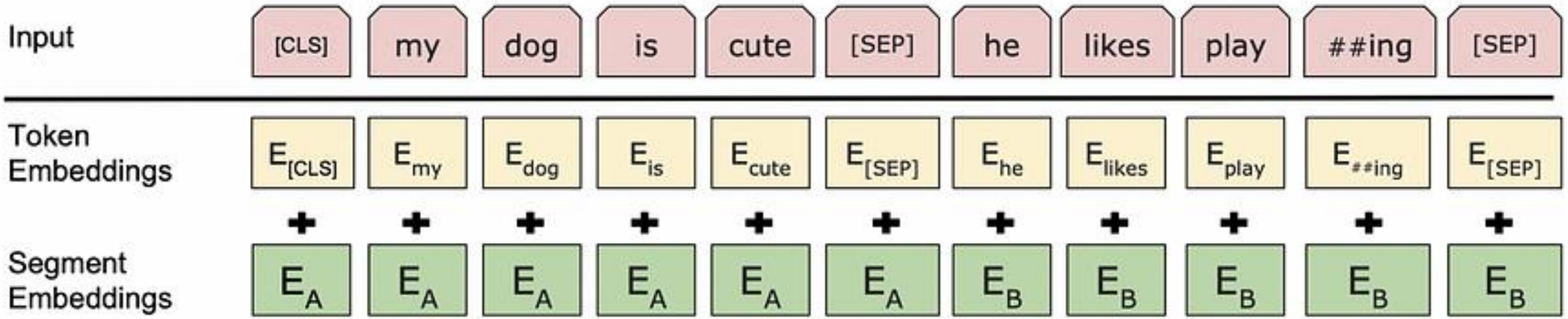
BERT 可以同時處理兩個句子，因此需要標記每個單詞屬於句子 A 還是句子 B。這對於理解句子之間的**關聯性**非常重要。

- **Segment A 和 Segment B：**

在處理兩個句子(句子對)時，第一個句子中的所有單詞的 Segment Embedding 被標記為 Segment A（通常表示為 0）。

第二個句子中的所有單詞則被標記為 Segment B（通常表示為 1）。

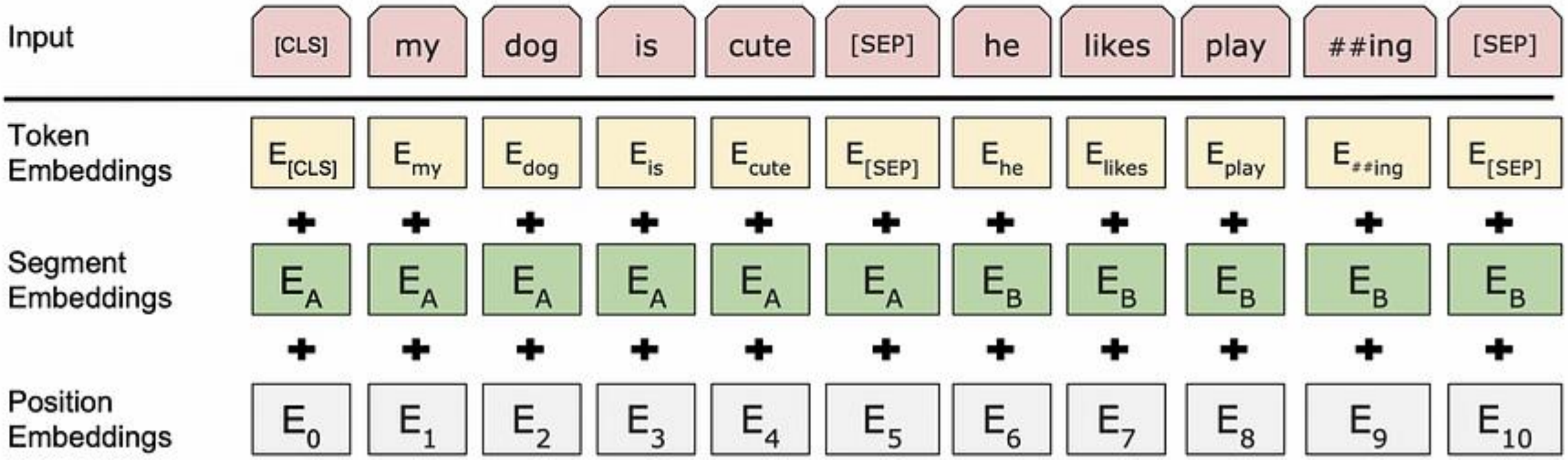
這樣，模型可以區分哪個單詞屬於句子 A，哪個單詞屬於句子 B。



BERT 

位置嵌入（Position Embedding）：

與 Transformer 一樣，BERT 也不具備對序列順序的內建感知能力，因此需要使用 Position Embedding 來顯示每個單詞在句子中的位置。



BERT 

- **預訓練任務**

BERT 的成功一大部分來自於其創新的預訓練任務。這些任務讓 BERT 能夠通過**「大量無標註的文本」**學習語言的結構，並且能夠通過微調快速適應下游任務。

- **Masked Language Model（MLM）**

傳統的語言模型是通過從左到右或從右到左預測下個單詞來進行預訓練的，這意味著模型在預測單詞時只能看到部分上下文。BERT 則採用了 遮蔽語言模型（MLM），通過隨機遮蔽一些單詞，然後要求模型根據句子的雙向上下文來預測被遮蔽的單詞。

- **遮蔽機制：**

在訓練過程中，BERT 會隨機選擇輸入序列中的 15% 的單詞進行遮蔽。這些被遮蔽的單詞會被替換為特殊標記 [MASK]。然後，模型需要通過上下文來預測這些被遮蔽的單詞是什麼。

- **運作模式：**

模型的任務是根據其前後的上下文來預測被遮蔽 [MASK]的位置應該填入哪個單詞。BERT 模型會從一個 30,000 字的詞彙表（vocabulary）中挑選出可能的單詞。

BERT 

- **運算過程：為了完成這個預測任務，BERT 的處理步驟如下**

1. **Encoder Output：**經過 BERT 的多層 Encoder 後，每個位置（包括 [MASK]）都會生成一個隱藏表示（Encoder Output)。這個隱藏表示包含了上下文的語義信息，能幫助模型預測被遮蔽的位置應該是什麼單詞。

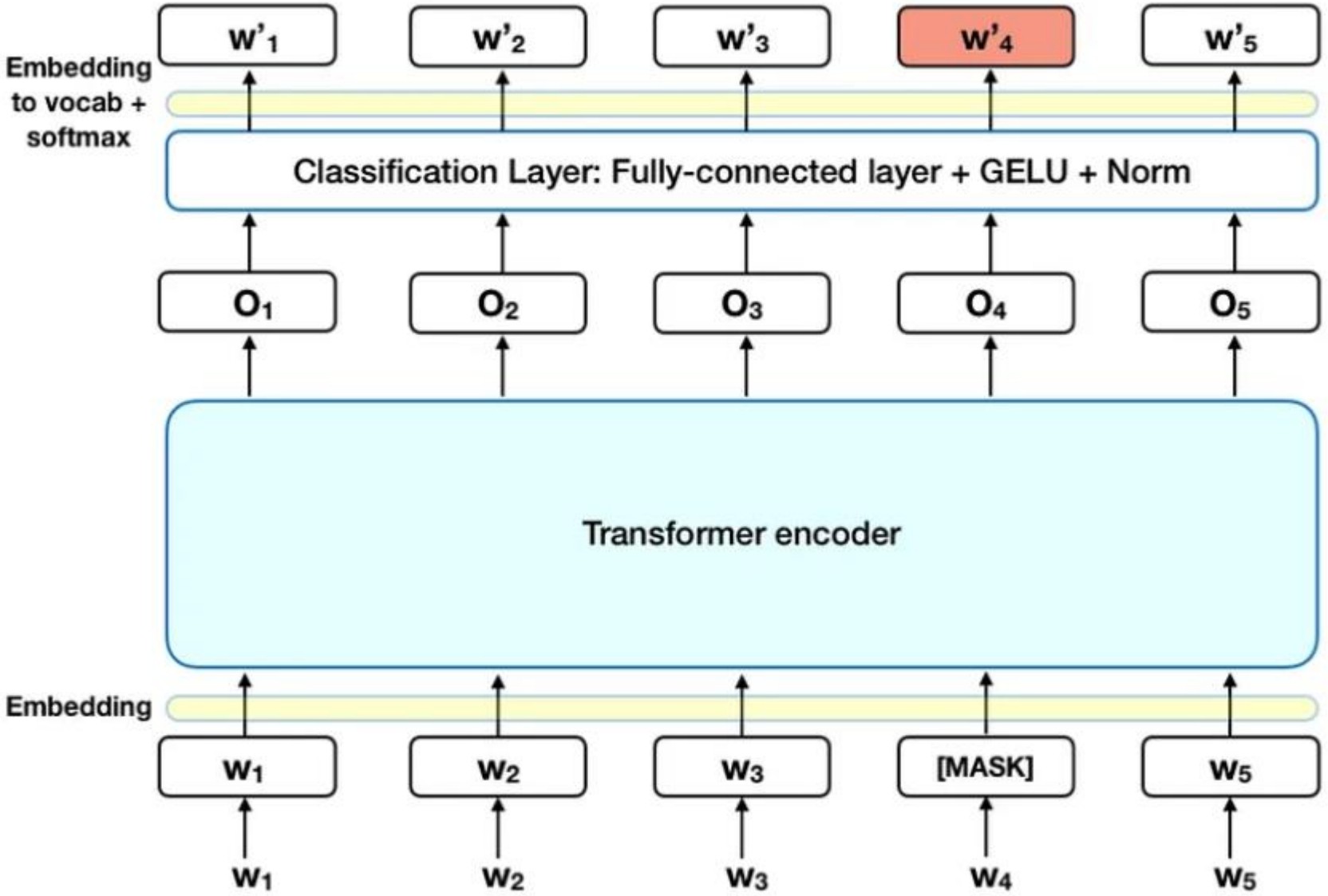
2. **Classification Layer：**這個 Encoder Output 會進入一個分類層（Classification Layer）。分類層的作用是將隱藏向量轉換成與詞彙表大小相同的向量維度。在 BERT 中，詞彙表的大小是 30,000，因此分類層的輸出向量的維度是 30,000。

3. **Softmax：**分類層的輸出會進一步通過 softmax 函數，這個函數將分類層的輸出轉換為一個概率分佈。對於每個被遮蔽的單詞，softmax 會為詞彙表中的每個詞彙分配一個概率，表示該詞彙作為預測結果的可能性。

4. **最終預測：**模型選擇概率最大的詞作為預測結果，即從 30,000 個候選詞彙中挑選出概率最高的那個詞。

5. **計算 Cross Entropy Loss：**交叉熵損失函數（Cross Entropy Loss），用來衡量模型預測分佈和真實分佈之間的差異。如果模型預測的詞與真實詞的差異越大，交叉熵損失就越大，反之亦然。損失值會被用來反向傳播，更新 BERT 模型的權重參數，讓模型在未來的預測中更加準確。

BERT


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

MLM

BERT 

- **Next Sentence Prediction（NSP）**

是 BERT 模型用來學習兩個句子之間關係的預訓練任務，這對於許多需要理解句子間關係的 NLP 任務（如問答系統 QA ）非常重要。為了讓模型更好地理解這種句子間的邏輯關係，NSP 訓練通過讓模型判斷兩個句子是否連續來提升其表現。

- **NSP 運作概念：**

NSP 的目的是讓模型能夠判斷句子 B 是否是句子 A 的「下一句」。也就是是在問模型：句子 B 接著句子 A 出現是合理的嗎？

- **IsNext**：句子 B 是句子 A 的連續句，兩者是緊密相關的。例如：「他走進商店。買了牛奶。」
- **NotNext**：句子 B 和句子 A 是隨機組合的，兩者沒有關聯。例如：「他走進商店。自然語言處理好難。」

- **運作模式：**

在 BERT 的訓練數據中，有 50% 的句子對是連續的，這樣的句子對會被標記為 IsNext，而另外 50% 的句子對是隨機組合的，標記為 NotNext。BERT 會將句子 A 和句子 B 放在一起，並在它們之間加上 [CLS] 和 [SEP] 這些特殊標記來分隔兩個句子。

- **[CLS]**：表示序列的開始，BERT 會用它來做最終的判斷。
- **[SEP]**：分隔兩個句子的標記。

例如：**[CLS]** the man went to **[MASK]** store **[SEP]** he bought a gallon **[MASK]** milk **[SEP]** → Label = IsNext

BERT 

- **模型判斷：**

1. BERT 會處理整個句子的序列，經過多層 Transformer 後，生成每個單詞的隱藏表示。模型會特別關注 [CLS] 這個標記位置的輸出，因為它代表整個句子的總結信息。

2. 模型將 [CLS] 標記的隱藏向量輸入一個分類層，分類層會生成兩個值，分別對應 IsNext 和 NotNext。

3. 這兩個值會經過 Softmax 函數，轉換為機率，表示句子連續和不連續的可能性。模型會選擇機率大的作為結果。

BERT 

- **Fine-Tuning**

微調就是將特定的下游任務（如分類、問答、命名實體識別等）的數據傳入 BERT，讓模型基於這些數據進行調整。

- **[CLS]**

在微調過程中，**[CLS]** 標記是用來進行句子級別的任務，比如文本分類。每次 BERT 模型處理完輸入句子後，**[CLS]** 標記的位置會產生一個隱藏向量，這個向量是對整個輸入的總結或摘要。這個向量會傳入一個新的**輸出層**，根據任務的需求進行分類。

- **Token Representation**

除了 [CLS] 用於句子級別的任務外，BERT 也會生成每個單詞（token）的表示向量，這些表示可以用來執行詞彙級別的任務。

FinBERT-19 (2020) : First Financial Sentiment Analysis Model

1. **Initialization:** Utilizes the general-domain BERT PLM with 3.3 billion tokens.
2. **Continual Pre-training:** Focuses on a financial-domain corpus to introduce domain-specific knowledge.
3. **Fine-Tuning:** Tailored to financial NLP tasks to optimize performance.

FinBERT-20 (2021)

1. **Pre-trained on Finance**: Utilizes a financial communication corpus with 4.9 billion tokens.
2. **FinVocab Release:** Includes both uncased and cased vocabularies, mirroring the token size of the original BERT model.
3. **Application in Sentiment Analysis:** Conducted fine-tuning experiments on the same dataset used by FinBERT-19.

FinBERT-21 (2021)

1. **Development:** Trained on both general and financial corpora.
2. **Multi-task Learning:** Implements six self-supervised pre-training tasks to enhance language understanding and semantic capture.
3. **Applications:** Conducted experiments in Sentiment Analysis, Sentence Boundary Detection, and Question Answering.