

文字資料的深度學習

ch 1 1



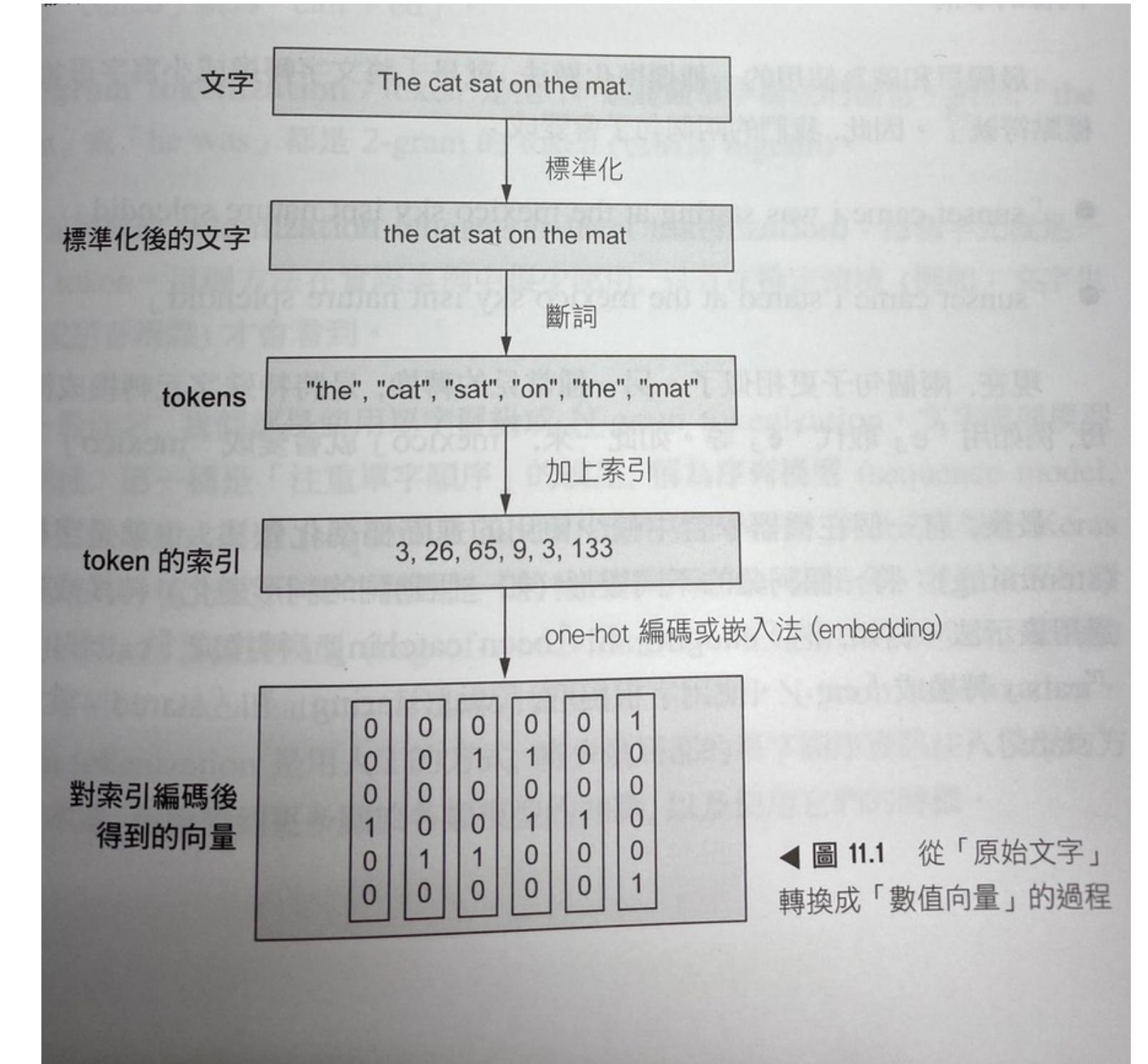
概述自然語言處理

natural language processing, NLP

- 人類的語言
- 在演化過程中逐漸形成
- 機器語言v.s自然語言
- 現代的NLP

(文字分類、內容過濾、情感分析、語言模型、翻譯、文章摘要)

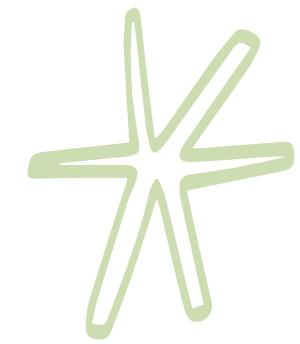
準備文字資料



文字標準化

- 特徵工程的基本形式
- [ɪ] v.s [i]、[isnt] v.s [isn't]

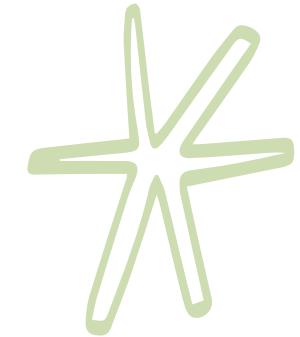
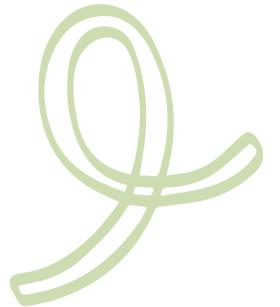
- 「sunset came i was staring at the mexico sky isnt nature splendid」
- 「sunset came i stared at the méxico sky isnt nature splendid」



斷詞

- 單字層級的斷詞
 - **called=call+ed**
- N-gram 斷詞->**詞袋**
 - n個連續單字構成之組合
- 字元層級的斷詞
 - 使用於特定情境

詞袋



- 處理token，而不是串列or序列
- 用於較淺的處理模型

這裡有一個簡單的例子。我們來看看「the cat sat on the mat.」這個句子，它拆解成以下這組 2-gram 的集合：

```
{"the", "the cat", "cat", "cat sat", "sat",  
"sat on", "on", "on the", "the mat", "mat"}
```

以上句子也可拆解成以下這組 3-gram 的集合：

```
{"the", "the cat", "cat", "cat sat", "the cat sat",  
"sat", "sat on", "on", "cat sat on", "on the",  
"sat on the", "the mat", "mat", "on the mat"}
```



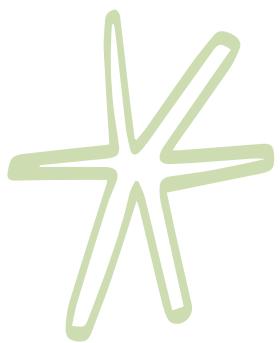
建立索引

- 雜湊法
- 只將常見的單字放進詞彙表

```
vocabulary = {}
for text in dataset:
    text = standardize(text) ← 將文字標準化
    tokens = tokenize(text) ← 將文字分解成 token
    for token in tokens:
        if token not in vocabulary:
            vocabulary[token] = len(vocabulary)] ← 若當前的 token 不在 vocabulary 中，則添加一個新項目，key 為當前的 token，value 則為當前 vocabulary 的長度
```

接著，把整數轉換成可由神經網路處理的向量，如 one-hot 向量：

```
def one_hot_encode_token(token):
    vector = np.zeros((len(vocabulary),)) ← 向量的長度為 vocabulary 的長度
    token_index = vocabulary[token] ← 取得特定 token 對應到的整數
    vector[token_index] = 1 ← 將該整數對應到的位置值設為 1
    return vector
```



使用TextVectorization層

1. 放於tf.data中(效率高)

```
int_sequence_dataset = string_dataset.map(  
    text_vectorization,  
    num_parallel_calls=4) ← 在多個 CPU 核心上平行化 map() 呼叫
```

← string_dataset 是一個會生成字串張量的 Dataset 物件



使用TextVectorization層

2. 變成模型的一部分(效率差)

```
text_input = keras.Input(shape=(), dtype="string") ←  
vectorized_text = text_vectorization(text_input) ← 創建一個預期接收字串的 Input 物件  
embedded_input = keras.layers.Embedding(...)(vectorized_text) ← 將 Input 物件輸入神經層  
output = ...  
model = keras.Model(text_input, output)  
  
這兩者之間有一個重要的一點
```

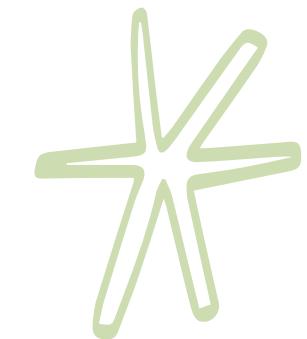
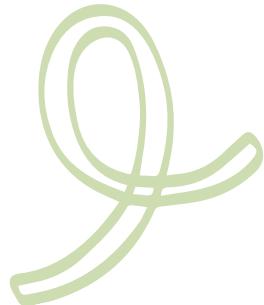
你可以在模型頂部不斷接上新的層，就像一般的函數式 API 模型一樣

表示單字的兩種方法

1. 集合->詞袋法(bag-of-words)

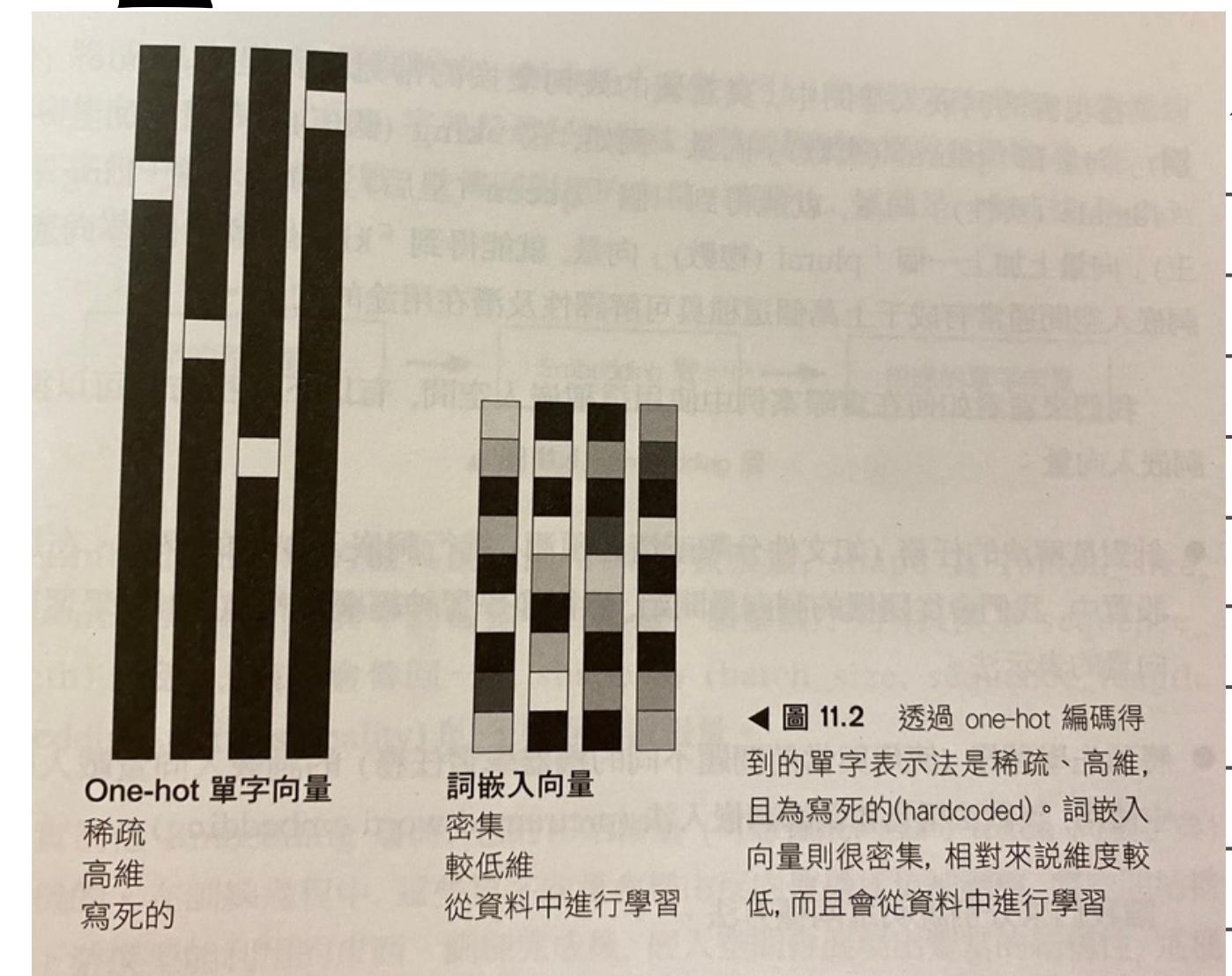
- 抛棄順序
- 利用TextVectorization層
- 二元編碼的單一單字(unigram)
- 二元編碼的bigram(N -gram)
- 使用TF-IDF編碼的bigram
- TF-IDF正規化

表示單字的兩種方法



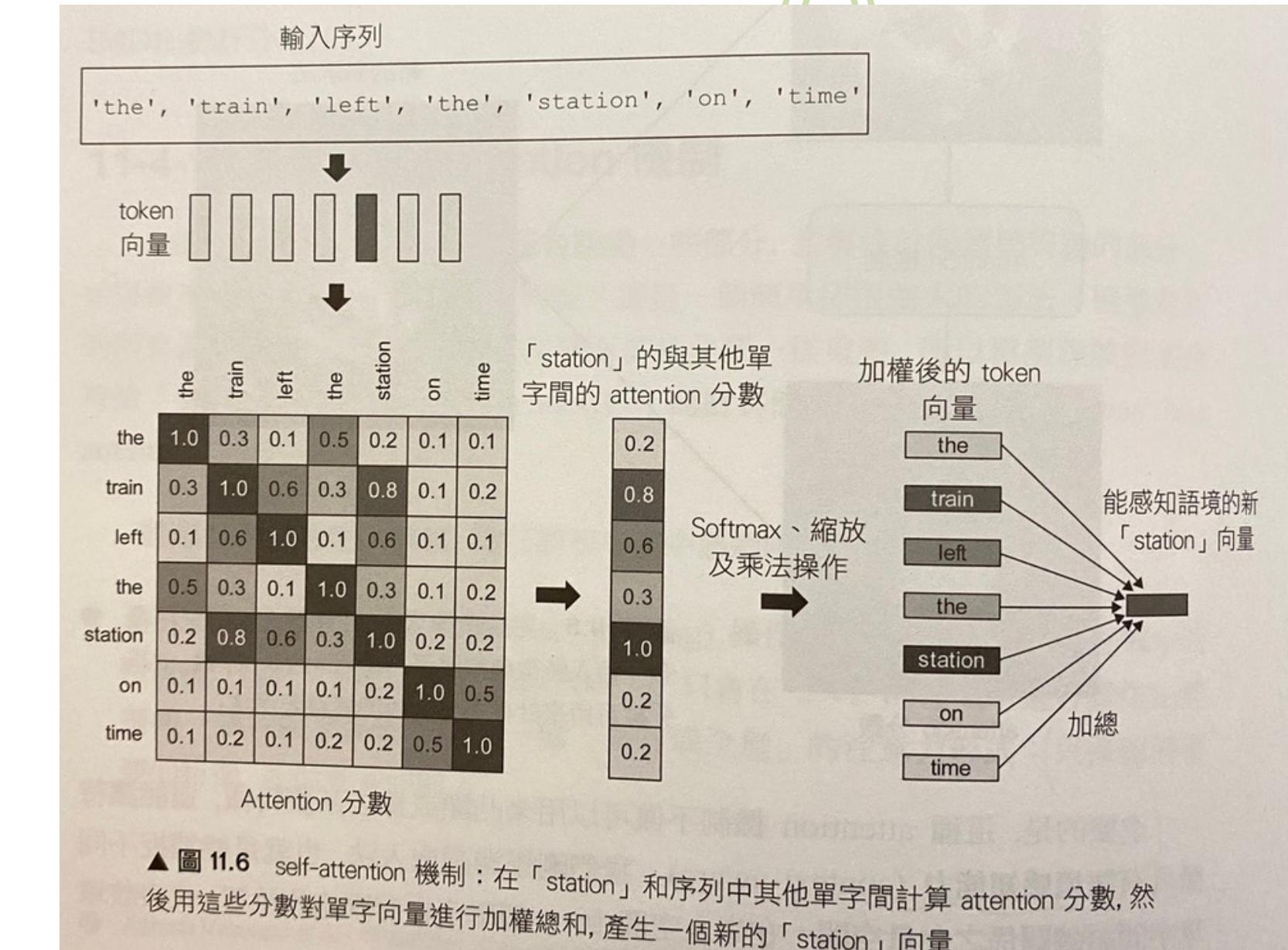
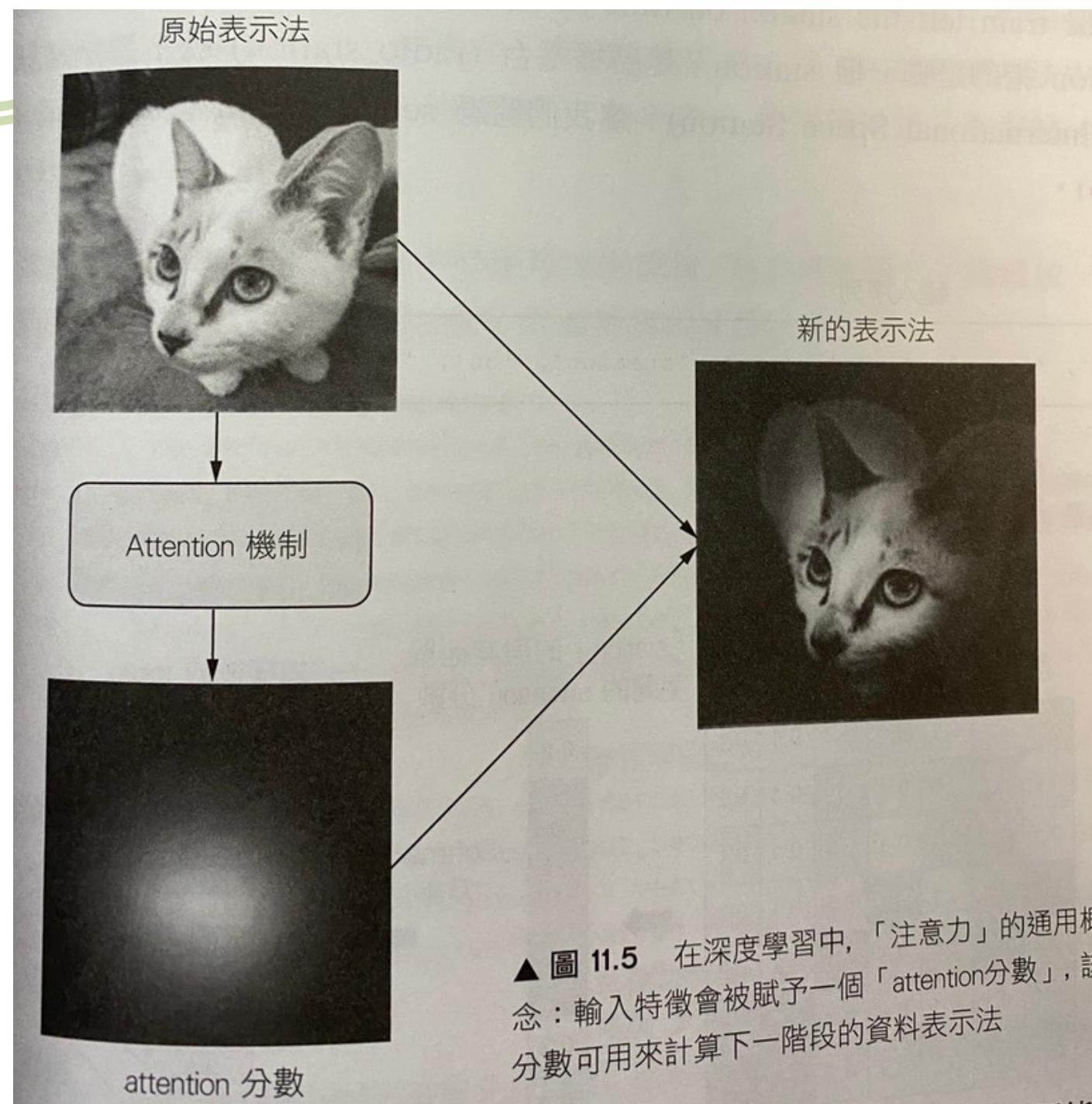
2. 序列->序列模型(sequence model)

- 用one-hot編碼將單字轉為向量
- 詞嵌入法
 - 針對預解決的任務
 - 使用預先學習好的詞



Transformer 架構

- self-attention 機制
 - 卷積神經網路中的最大池化
 - TF-IDF 正規化

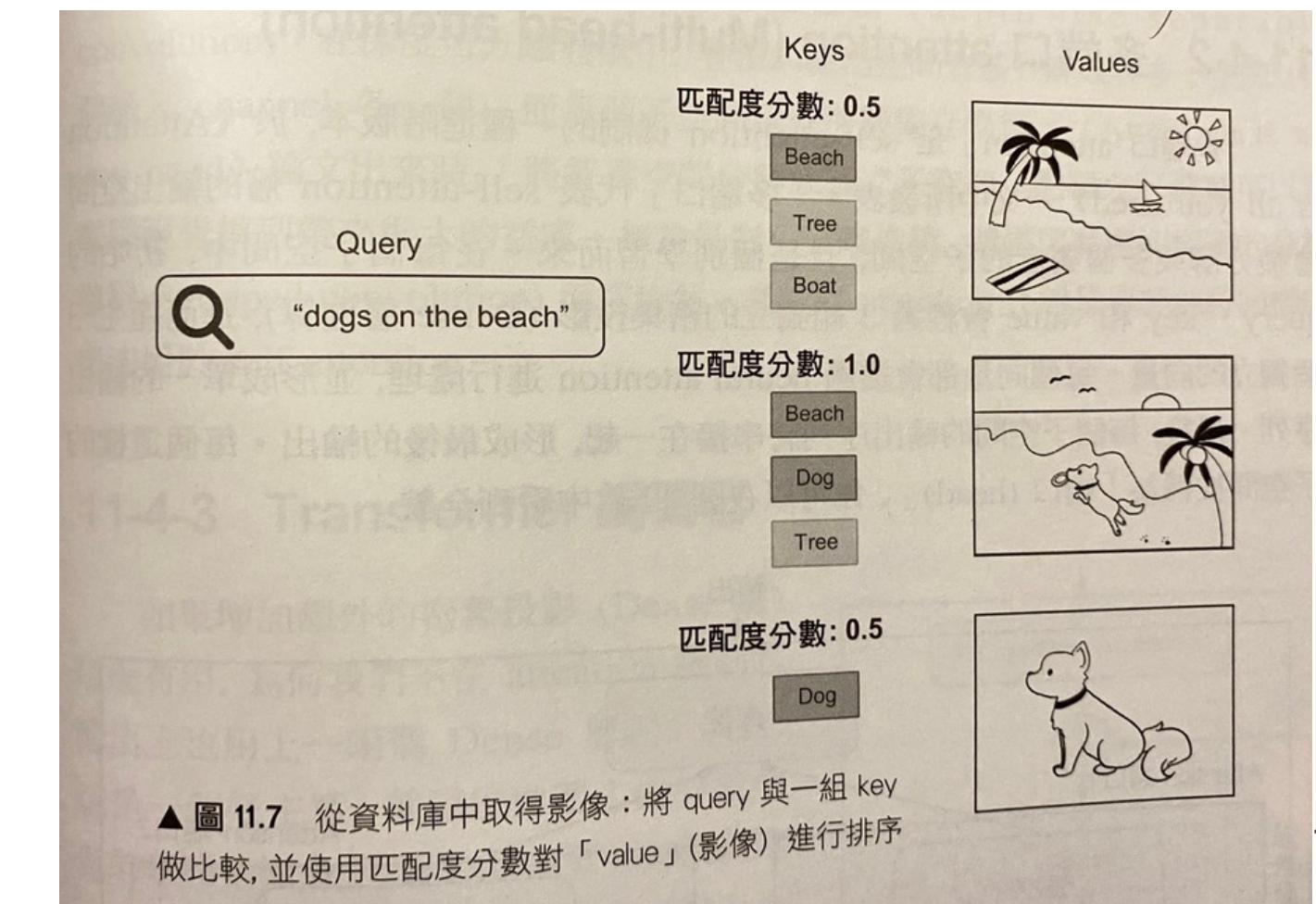


Transformer架構

- 普適化的self-attention:query-key-value模型

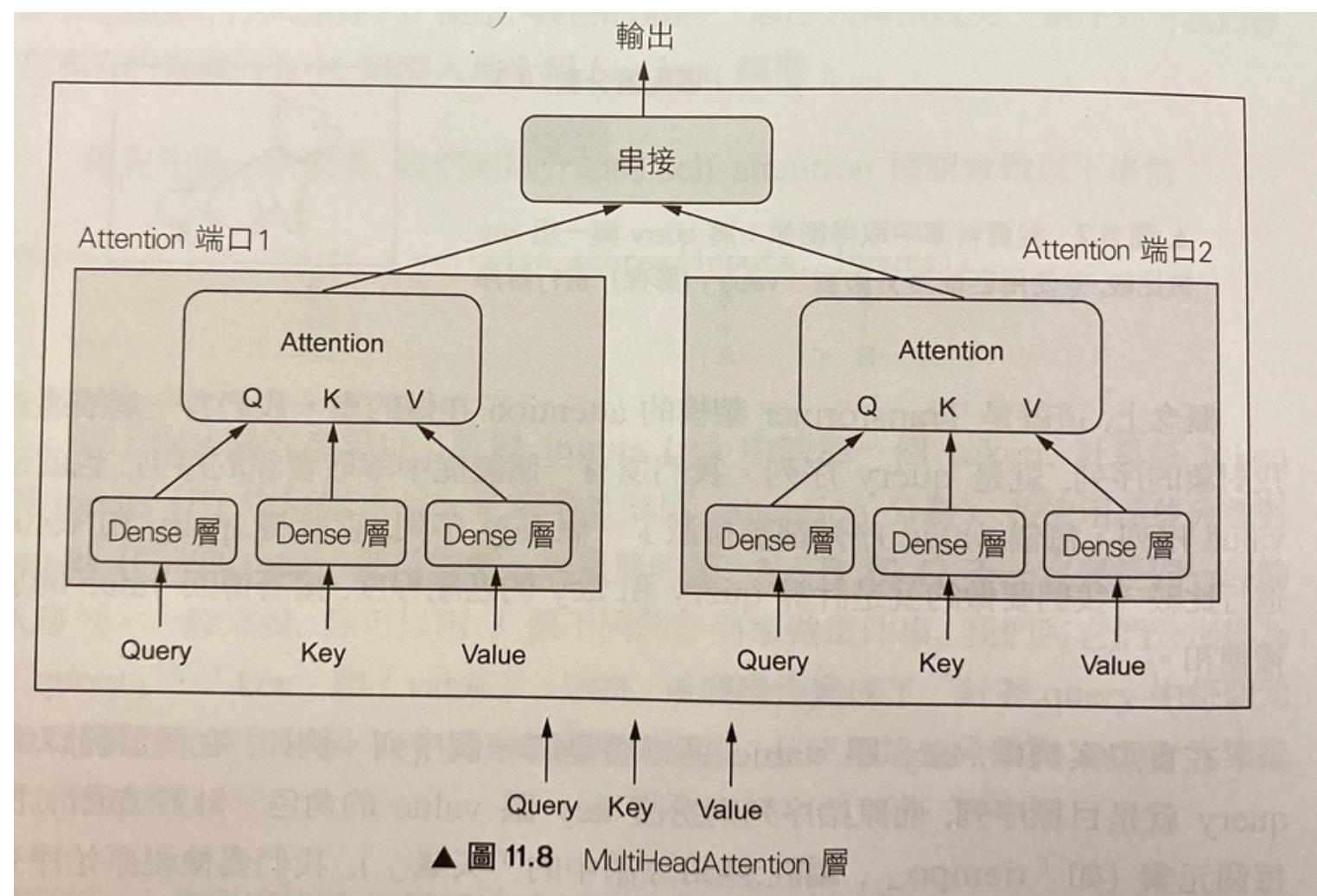
```
outputs = sum(inputs * pairwise_scores(inputs, inputs))  
          ↑ C  
          ↑ A      ↑ B
```

```
outputs = sum(values * pairwise_scores(query, keys))
```



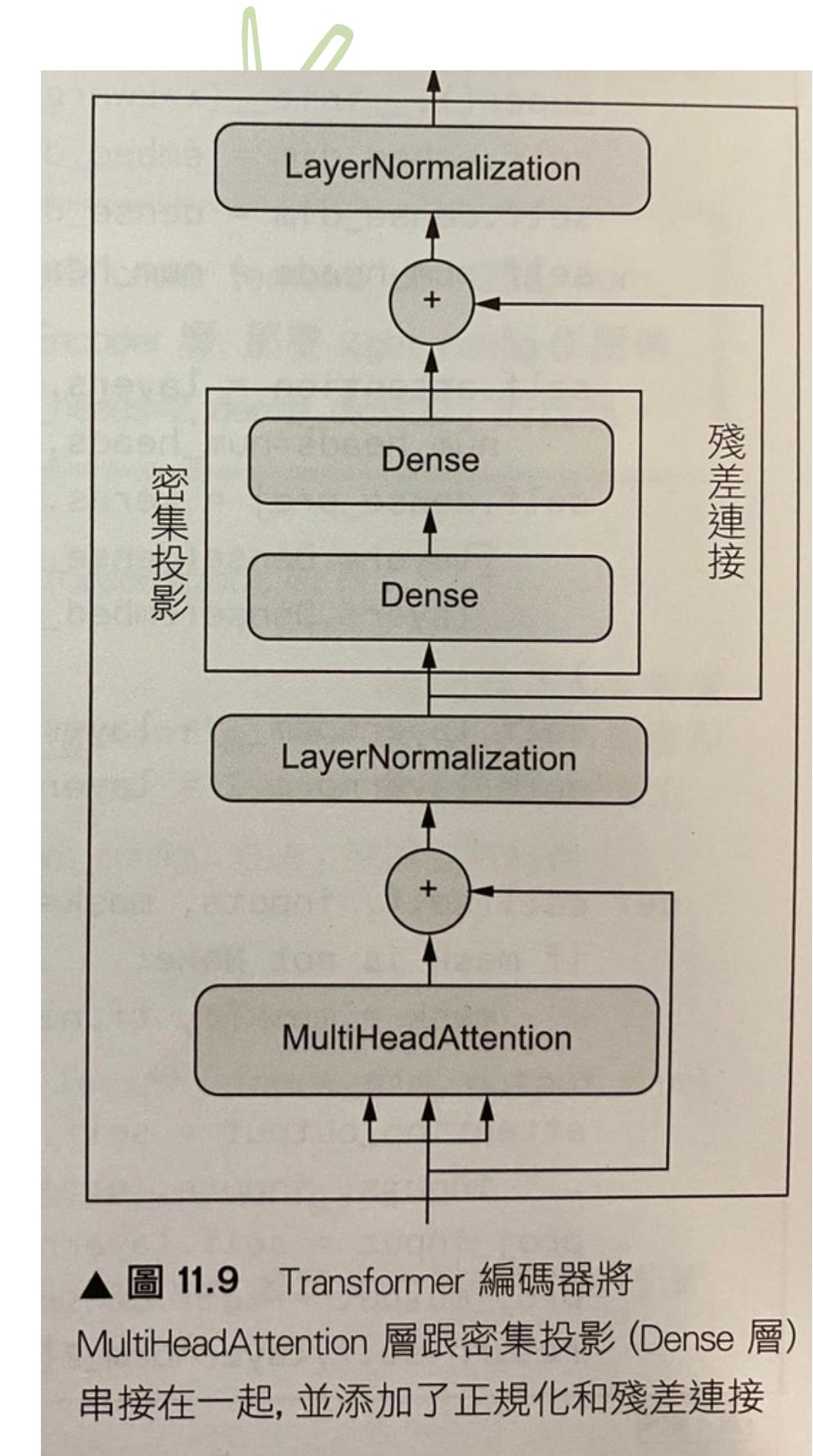
多端口 attention

- 是self-attention機制的一種進階版本



Transformer編碼器

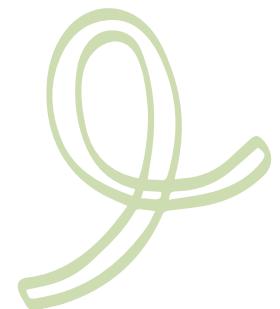
- 多端口層
- dense層
- 正規化層
- 殘差連接



▲ 圖 11.9 Transformer 編碼器將
MultiHeadAttention 層跟密集投影 (Dense 層)
串接在一起, 並添加了正規化和殘差連接

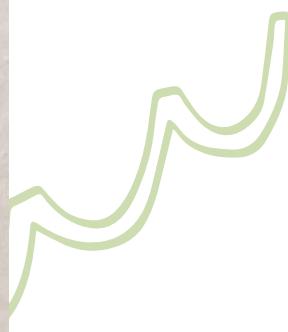
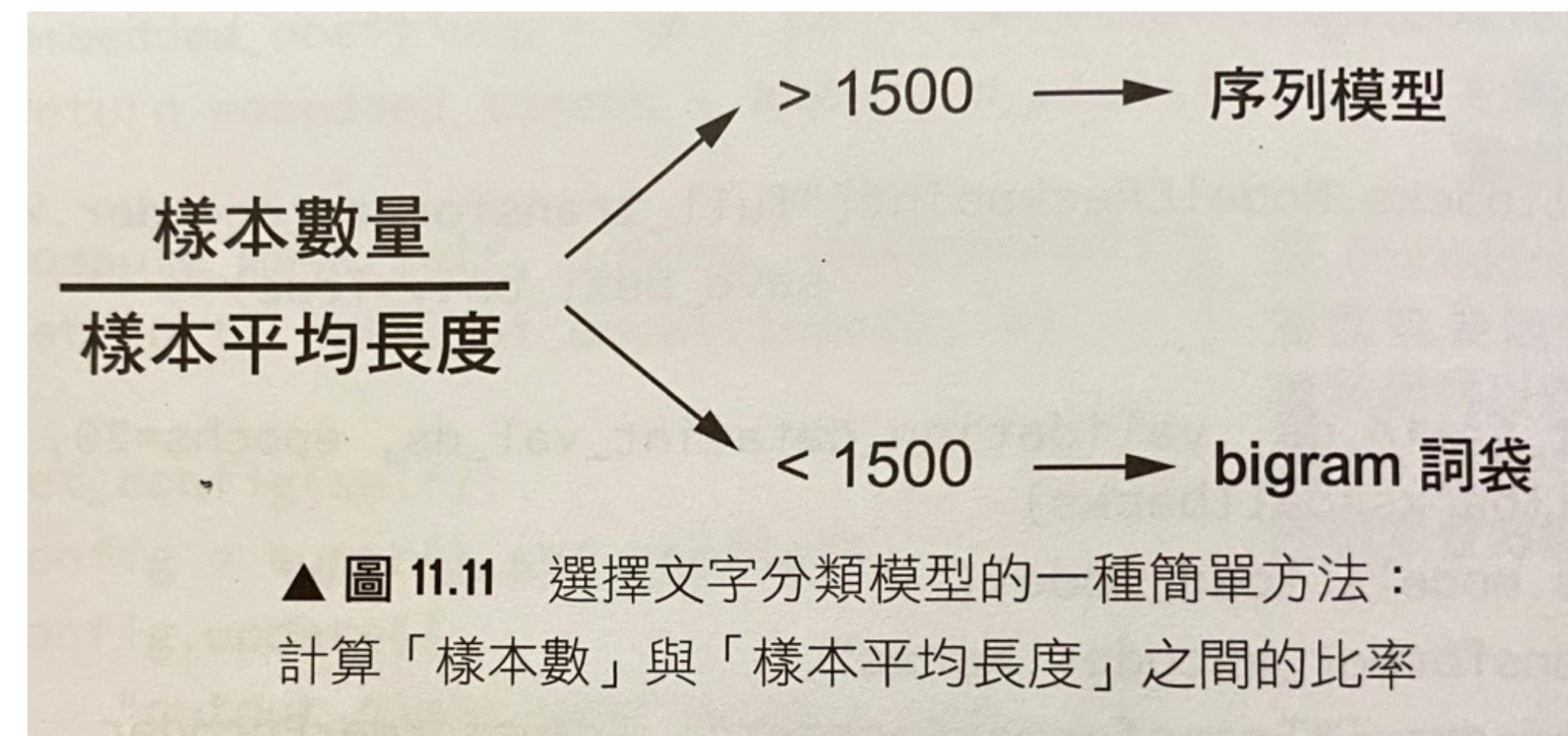
位置編碼

- 在每個單字的嵌入向量中，加入它在句子中的位置
- 詞的嵌入向量由兩部分組成
 - 一般的詞向量
 - 一個位置向量
- 位置嵌入法



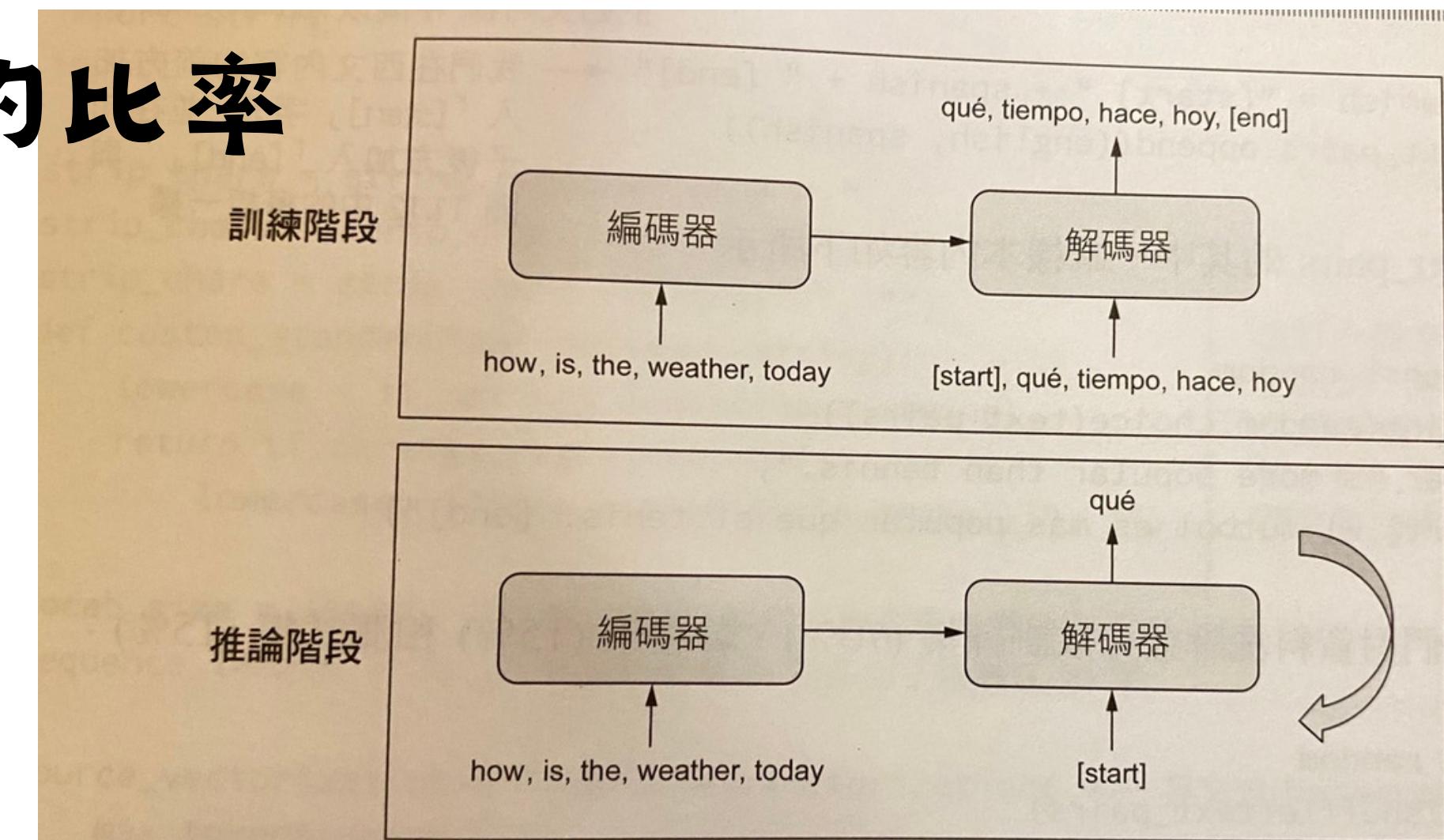
何時該選擇序列模型，而非詞袋模型

- 訓練「資料中的樣本數」與「每個樣本的平均字數」之間的比率



文字分類以外的任務-Seq2seq模型

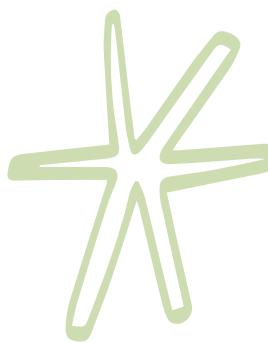
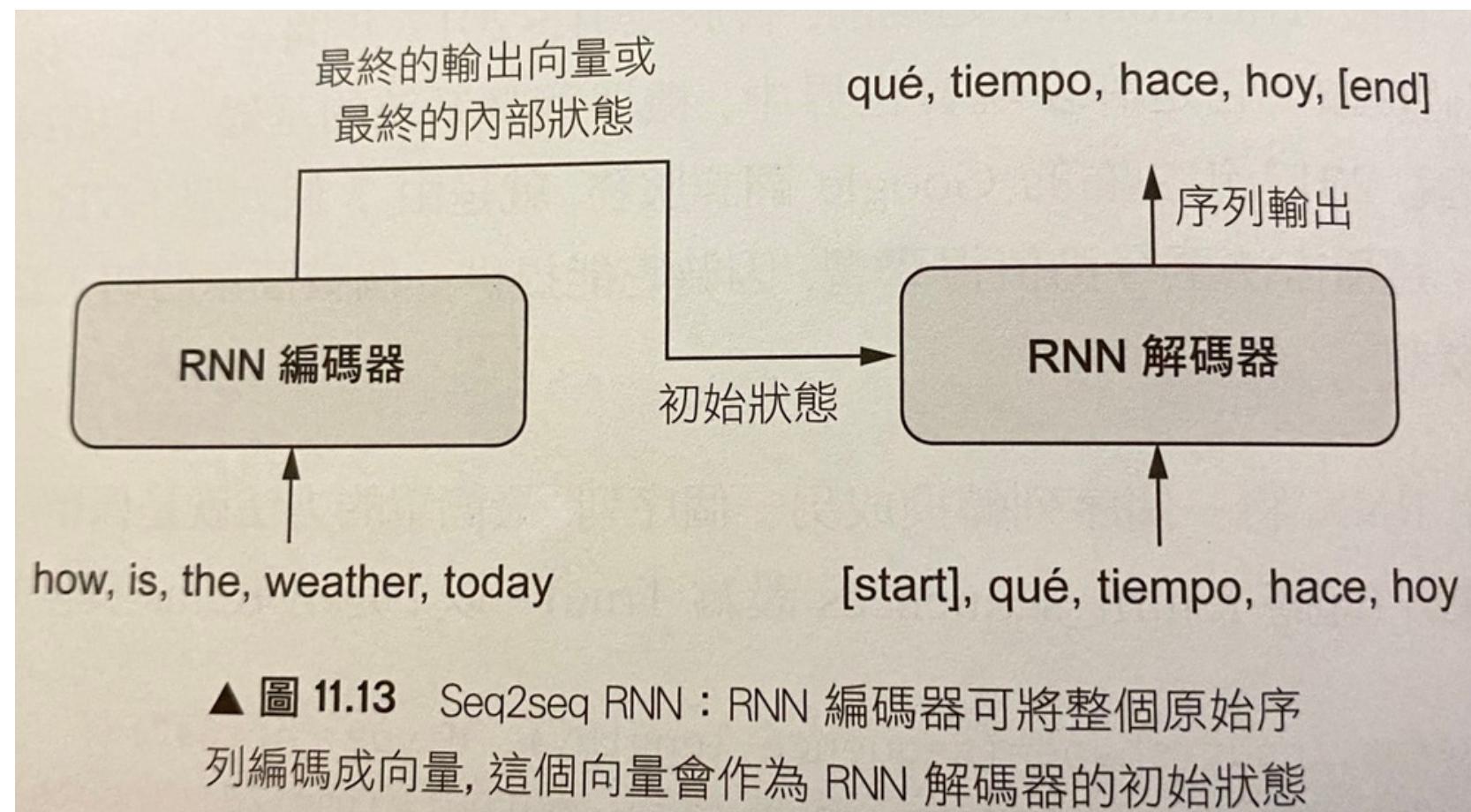
- 訓練「資料中的樣本數」與「每個樣本的平均字數」之間的比率



▲ 圖 11.12 Seq2seq 的學習：編碼器會先處理原始序列以輸出編碼向量，接著將編碼向量送到解碼器中。解碼器會檢視當前的目標序列，並預測目標序列中下一步的 token。在推論過程中，我們會一次生成一個目標 token，並將其送回解碼器中以生成下一個 token

用RNN進行Seq2seq的學習

- 有兩個主要的問題
 - 目標序列和原始序列的長度必須相同
 - 「逐一處理時步」特性



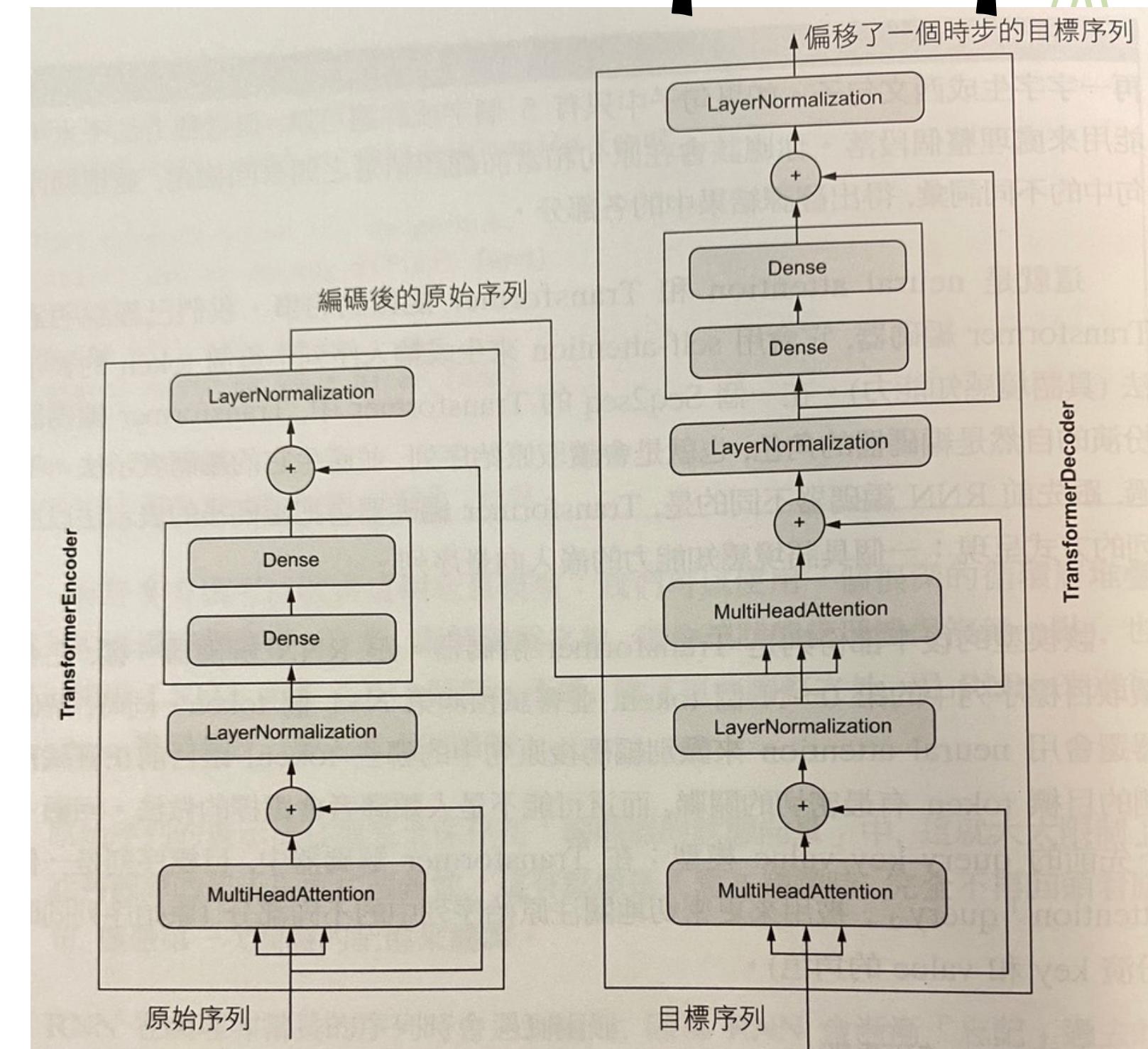
用RNN進行Seq2seq的學習

- 先天上的限制
 - 原始序列的表示法必須保存在「編碼器的狀態向量」中
 - 在處理非常長的序列時會遇到困難
- 利用Transformer解決以上限制

用Transformer進行Seq2seq的學習

- 比RNN能處理更長、更複雜的序列
- 具語境感知能力的嵌入向量序列

用Transformer進行Seq2seq的學習



▲ 圖 11.14 TransformerDecoder 與 TransformerEncoder 類似，只是多了一個額外的 attention 區塊，其中的 key 跟 value 是由經過 TransformerEncoder 編碼的原始序列。編碼器和解碼器組合在一起，便構成了一個端到端的 Transformer

Thank ~