目次

- GitHub 專案連結
  https://github.com/telunyang/python_nlp

Module 1. 何謂自然語言處理(Natural Language Processing)

簡介

自然語言處理（Natural Language Processing，NLP）是人工智慧領域的一個重要分支，專注於讓電腦理解和操作人類語言。NLP 結合了資訊科學、人工智慧和語言學的元素，旨在填補人類自然語言和電腦之間的差距。

NLP 的主要任務包括自然語言理解和自然語言生成等。自然語言理解讓電腦能夠「理解」和解釋人類語言的意義，自然語言生成的目標是從一些資料中「生成」人類可以理解的自然語言。NLP 有許多應用，包括機器翻譯、情感分析、文本摘要、自動客服系統等。隨著深度學習等先進技術的出現，NLP 的發展也日新月異，使得電腦能夠更好地理解和生成自然語言。

然而，NLP 也面臨許多挑戰，如語義歧義、俚語和諺語的理解、文本的歧義和不確定性等。此外，不同語言之間的差異也給 NLP 帶來了很大的困難。但隨著技術的進步，相信這些問題將得到解決，使得電腦能夠更好地理解和使用人類語言。

文字的特性

在自然語言處理（NLP）中，文字的特性包括以下幾點：

- **不定長度**：
  每個句子或段落的長度可能會有所不同，這使得處理文字比處理固定長度的資料（如圖像）更為複雜。
- **順序性**：
  文字的順序對於其意義具有重要影響。例如，「狗追貓」和「貓追狗」雖然包含相同的詞，但意義卻完全不同。
- **語義多樣性**：
  一個詞在不同的語境中可能具有不同的意義，這就需要 NLP 技術理解和考慮語境。例如，「蘋果」可能指的是水果，也可能是電腦公司；「bank」可能指的是銀行，也可能是河岸邊(河畔)。
- **語言特性**：
  不同的語言具有不同的結構和規則。例如，英語中的單詞是由空格分隔的，而中文則沒有明顯的詞之間的分隔符號。
- **文化和語境因素**：
  文字所傳達的意義經常與其文化和語境有關。這使得 NLP 需要理解文化和

語境，才能準確地處理和理解文字。

常見用途

自然語言處理（NLP）在許多領域中都有著廣泛的應用，以下列舉幾個常見的例子：

- **機器翻譯**：
  例如 Google 翻譯，能自動將一種語言的文字翻譯成另一種語言。
- **情感分析**：
  用來判定文字內容中隱含的情感傾向，例如正面、負面或中性。這在社交媒體互動、品牌管理等領域都有廣泛的應用。
- **聊天機器人**：
  需要理解用戶所輸入的語言文字，並生成適當的文字回應。這在客戶服務、智慧型助手等領域都有廣泛的應用。
- **文本生成**：
  可以用來自動生成文章、報告、詩歌等。例如，新聞公司可以自動撰寫財經報告。(可參考 https://www.youtube.com/watch?v=c3fHRQonqlM)
- **文本摘要**：
  將長篇文章自動壓縮成短小的摘要，以方便使用者快速獲取所需資訊。
- **問答系統**：
  像是 Google 的搜尋引擎，或是 Microsoft Edge 的 Bing，利用了 NLP 來理解用戶的問題並提供相對應的答案。

安裝 conda 的 nlp 虛擬環境
conda create --name nlp python=3.10 notebook ipykernel

(非必要)安裝 kernel
python -m ipykernel install --user --name k_nlp --display-name "python3@nlp"

刪除 kernel
jupyter kernelspec list jupyter kernelspec uninstall k_nlp Remove 1 kernel specs [y/N]: y 按下 enter

刪除 nlp 虛擬環境

```
conda remove --name nlp --all
```

Module 2. 文字預處理

流程說明

不同的語言在自然語言處理的預處理階段會有些不同；這只是預處理的基本步驟，根據不同的應用和資料集，可能需要調整或增加更多的步驟：

英文 NLP 預處理

- **文本清洗**：
  去除文本中的無關內容，如 HTML 標籤、特殊字符等。
- **分詞**（Tokenization）：
  將句子分割為單詞或詞語。
- **轉換為小寫**：
  將所有單詞轉換為小寫，這有助於統一詞彙。
- **去除停用詞**：
  去除一些在語句中出現頻率高，但對於語義分析沒有多大貢獻的詞，如“is”，“and”，“the”等。
- **詞幹提取(Stemming) & 詞性還原(Lemmatization)**：
  將單詞轉換為其基本形式。例如，詞幹提取可能將"eating"、"eats"轉換為"eat"；詞性還原可能將"running"、"ran"在指定詞性(例如動詞verb)後，轉換為"run"。
- **詞袋(bag-of-word, BoW)**或**詞嵌入(Word Embedding)**：
  將文字轉換為模型能理解的數值或向量。

中文 NLP 預處理

- **文本清洗**：
  同樣需要去除文本中的無關內容。
- **分詞**：
  中文的分詞較為複雜，因為詞與詞之間沒有明顯的分隔符。常見的中文分詞工具包括 jieba 等。
- **去除停用詞**：
  與英文相似，中文也有一些常出現但意義較小的詞需要去除，例如「你、我、他、的」這些經常出現在文章段落中出現的字詞。
- 轉換為**詞向量(Word Vector)**：

這可以使用詞袋模型(Bag-of-word, BoW)，也可以使用詞嵌入(Word Embedding)技術如 Word2Vec。

英文字預處理

NLTK（Natural Language Toolkit）是一個 Python 函式庫，用於處理自然語言處理（NLP）的相關任務。它提供了各種工具和資源，包括語料庫、語法分析器、詞性標註器、詞幹提取器等，可用於文本處理和分析。

備註: 記得安裝套件

```
!pip install nltk numpy svgling scikit-learn
```

2_牛刀小試

```
# 用來進行 tokenize, part_of_speech 和 named entity recognition 的 文本 (這邊使用英文歌詞)
lyrics = '''
When I was young,
I'd listen to the radio,
Waitin' for my favorite songs,
When they played I'd sing along,
It made me smile.
'''

# 匯入套件
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk import ne_chunk

# ne_chunk 會用到 numpy
import numpy as np

# 下載文句分詞器 (sentence tokenizer)
nltk.download('punkt')

# 下載詞性標註器 (tagger)
nltk.download('averaged_perceptron_tagger')
```

```
# 下載命名實體辨識工具
nltk.download('maxent_ne_chunker')
nltk.download('words')

# 分詞
tokens = word_tokenize(lyrics); tokens

# 詞性標註 (part-of-speech)
pos_tags = pos_tag(tokens); pos_tags

# 命名實體辨識
ne_chunks = ne_chunk(pos_tags); ne_chunks
```

2_英文字預處理

```
# 匯入套件
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.feature_extraction.text import CountVectorizer

# 下載分詞器
nltk.download('punkt')

# 下載停用詞
nltk.download('stopwords')

# 下載已經定義字詞-語義關係來尋找上下位關係的英文詞典，也包含了同義詞、時態、名詞單複數等資訊
nltk.download('wordnet')

# 輸入文本
text = '''Hey Jude, don't make it bad. Take a sad song and make it better. Remember to let her into your heart. Then you can start to make it better. Hey Jude, don't be afraid. You were made to go out
```

```
and get her. The minute you let her under your skin. Then you begin
to make it better.'''

# 文本清洗，去除標點符號與數字等非字母的字符
text = ''.join(c for c in text if c.isalpha() or c.isspace()); text

# 分詞
tokens = word_tokenize(text); tokens

# 轉換為小寫
tokens = [word.lower() for word in tokens]; tokens

# 去除停用詞
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]; tokens

# 詞幹提取
ps = PorterStemmer()
tokens_stem = [ps.stem(word) for word in tokens]; tokens_stem

# 詞形還原
lemmatizer = WordNetLemmatizer()
tokens_lemm = [lemmatizer.lemmatize(word) for word in tokens];
tokens_lemm
```

中文字預處理

備註: 記得安裝套件

```
!pip install jieba
```

2_中文字預處理

```
# 匯入套件
import jieba

# 輸入文本
text = "這是使用 Jieba 和 sklearn 進行中文字預處理的範例。"
```

```
# 文本清洗，去除標點符號與數字等非字母的字符
text = ''.join(c for c in text if c.isalpha())

# 分詞 (cut 是用於 iteration 的斷詞函式，lcut 是將斷詞以 list 格式回傳)
tokens = jieba.lcut(text)

# 去除停用詞，這邊為了簡化，我們預先定義一個停用詞列表
stop_words = ['和', '的', '是', '這']
tokens = [word for word in tokens if word not in stop_words]; tokens

# 讀取自定義的字典
# jieba.load_userdict('./dict.txt')

# 自定義字詞
jieba.add_word('網球史上')
jieba.add_word('網球運動員')
jieba.add_word('單打冠軍')
jieba.add_word('冠軍')
jieba.add_word('大滿貫')
jieba.add_word('總決賽')
jieba.add_word('大師賽')

# 輸入文本
text = '''羅傑費德勒，已退役的瑞士男子職業網球運動員，費德勒總共贏得 20 座大滿貫冠軍，單打世界排名第一累計 310 周，其中包括連續 237 周世界排名第一的男子網壇紀錄，為網球史上最佳的男子選手之一。費德勒生涯贏得 103 個 ATP 單打冠軍，含 20 座大滿貫冠軍和 6 座 ATP 年終總決賽冠軍，以及 28 座大師賽冠軍。'''

# 分詞 (cut 是用於回傳 generator 的斷詞函式，lcut 是將斷詞以 list 格式回傳)
tokens = jieba.lcut(text)

# 去除停用詞，這邊為了簡化，我們預先定義一個停用詞列表
stop_words = ['和', '的', '是', '這']
tokens = [word for word in tokens if word not in stop_words]; tokens
```

補充：詞雲(Word Cloud)

Word Cloud（又稱文字雲、字雲、詞）是一種視覺化工具，用於表示一組文本資料中的詞彙頻率(很常用在特定文章中，檢視哪些用字遣詞比較常見，藉此了解特定議題的聲量)。在 Word Cloud 中，每個字詞的大小代表它在文本中出現的頻率或重要性。出現頻率較高的字詞會以較大的字體呈現，而出現頻率較低的字詞則以較小的字體呈現。這種方式可以快速地向讀者呈現文本的主要主題。

備註: 記得安裝套件

```
!pip install wordcloud
```

2_詞雲(Word Cloud)

```
!pip install wordcloud

'''
基本用法
'''
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# 假設我們已經有了一個字詞和它的頻率的字典
word_freq = {'Python': 50, '資料科學':40, '機器學習': 30, '深度學習':
20, '自然語言處理': 5}

# 建立一個文字雲物件
wc = WordCloud(
    font_path='./fonts/NotoSansTC-Regular.otf',
    background_color='white',
    width=1024,
    height=768
)

# 生成文字雲 (wc.generate_from_frequencies 用於 dict)
wc.generate_from_frequencies(word_freq)

# 顯示文字雲
```

```
plt.figure(figsize=(10.24, 7.68), dpi=100)
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()

'''
使用自訂圖片來作為詞雲顯示範圍
'''
from wordcloud import WordCloud, ImageColorGenerator
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import jieba

# 讀取圖片
image = Image.open('./images/doraemon_01.png')  # 替換為你的圖片路徑
mask = np.array(image)

# 這是我們的原始文本資料，這裡只是一個例子，你可以使用任何你想要分析的文本
text = '''
謝 天

…幾年來自己的奔波，作了一些研究，寫了幾篇學術文章，真正做了一些小貢獻以後，
才有了一種新的覺悟：即是無論什麼事，得之於人者太多，出之於己者太少。

因為需要感謝的人太多了，就感謝天罷…
'''

# 使用 jieba 進行分詞
seg_list = jieba.lcut(text)

# 將分詞後的結果組合成一個長字串，詞與詞之間以空格相隔，這是 wordcloud 需要
的格式
seg_str = ' '.join(seg_list)

# 建立一個詞雲物件
wc = WordCloud(
```

```
    font_path='./fonts/NotoSansTC-Regular.otf',
    background_color='white',
    mask=mask,
    width=820,
    height=1349,
    random_state=42
)

# 生成詞雲 (wc.generate 用於純文字)
wc.generate(seg_str)

# 從圖片中生成顏色
image_colors = ImageColorGenerator(mask)

# 使用從圖片中生成的顏色
wc.recolor(color_func=image_colors)

# 顯示文字雲
plt.figure(figsize=(8.2, 12.49), dpi=100)
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.show()
```

補充：OpenCC

OpenCC 是一個開源的中文簡繁轉換工具，它的特點是利用詞庫進行轉換，可以處理一些詞組翻譯上的問題，例如"鼠标"轉換為"滑鼠"而不是"鼠標"，達到與人工翻譯相媲美的效果。它支持多種轉換模式，包括簡體到繁體、繁體到簡體，以及港澳繁體到台灣繁體等等。

備註: 記得安裝套件

```
!pip install opencc-python-reimplemented
```

2_OpenCC

```
!pip install opencc-python-reimplemented


from opencc import OpenCC
```

```
# 初始化 OpenCC，設定轉換的模式，這裡設定為's2tw'（簡體到臺灣使用的繁體）
cc = OpenCC('s2tw')

# 進行轉換
to_convert = '简体中文'
converted = cc.convert(to_convert)

print(converted)  # 輸出轉換後的結果，應該會看到'简体中文'被轉換為'簡體中文'
```

Module 3. 量化文字的常見方式

One-Hot Encoding

一種表示「類別」變數的方法。在這種方法中，每一個類別都被表示為一個二進制向量，向量中只有一個元素是 1（表示該類別），其餘元素都是 0。這些向量對於不同的詞都是正交的(Orthogonal)，換句話說，One-hot 編碼無法捕捉到詞與詞之間的相似性。

	字典檔
	["人", "帥", "真好"]
人	[1, 0, 0]
帥	[0, 1, 0]
真好	[0, 0, 1]

圖：One-Hot Encoding 的範例

3_One-Hot Encoding

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import jieba

# 完整的句子
sentence = "我喜歡閱讀書籍，也喜歡使用電腦來學習新的知識"
```

```
# 使用 jieba 進行斷詞
words = list(jieba.cut(sentence)); words

# 轉換為 OneHotEncoder 能接受的二維陣列形式
words_array = np.array(words).reshape(-1, 1); words_array

# 建立 OneHotEncoder 物件
onehot_encoder = OneHotEncoder(sparse_output=False)

# 進行 one-hot encoding
onehot_encoded = onehot_encoder.fit_transform(words_array)

# 輸出結果
# ['我', '喜歡', '閱讀', '書籍', '，', '也', '喜歡', '使用', '電腦來', '學習', '新', '的', '知識']
print(onehot_encoder.categories_)
print(onehot_encoded)

# 找出「我」的 One-Hot Encoding
print(onehot_encoded[:, 0])

# 找出「喜歡」的 One-Hot Encoding
print(onehot_encoded[:, 2])
```

詞袋模型(Bag-of-Words model, BoW)

BoW 模型將每個單詞視為詞彙表中的一個單獨條目，並將每個文件（或句子）表示為詞彙表中各個詞頻的向量。BoW 模型忽略了詞與詞之間的順序，只考慮了詞的出現頻率。這對於一些任務可能足夠，但對於需要考慮語境的任務可能不夠，例如「我喜歡你」和「你喜歡我」在詞袋模型中是相同的。

英文部分

```
# 安裝套件
!pip install scikit-learn pandas

# 匯入套件
import pandas as pd
```

```
from sklearn.feature_extraction.text import CountVectorizer

# 為了預覽 array 結果，匯入必要的套件，以及設定環境
import sys
import numpy
numpy.set_printoptions(threshold=sys.maxsize)

# 詞袋模型
docs = [
    "Hey Jude, don't make it bad. Take a sad song and make it better. Remember to let her into your heart. Then you can start to make it better.",
    "Hey Jude, don't be afraid. You were made to go out and get her. The minute you let her under your skin. Then you begin to make it better."
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs)
print(vectorizer.get_feature_names_out())
print(X.toarray())

# 透過 pandas 來預覽結果
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out()); df
```

中文部分

```
# 安裝套件
!pip install jieba

# 匯入套件
import jieba

# 自定義字詞
jieba.add_word('網球史上')
jieba.add_word('網球運動員')
jieba.add_word('單打冠軍')
jieba.add_word('冠軍')
```

```
jieba.add_word('大滿貫')
jieba.add_word('總決賽')
jieba.add_word('大師賽')

# 輸入文本
text = '''羅傑費德勒，已退役的瑞士男子職業網球運動員，費德勒總共贏得 20 座大滿貫冠軍，單打世界排名第一累計 310 周，其中包括連續 237 周世界排名第一的男子網壇紀錄，為網球史上最佳的男子選手之一。費德勒生涯贏得 103 個 ATP 單打冠軍，含 20 座大滿貫冠軍和 6 座 ATP 年終總決賽冠軍，以及 28 座大師賽冠軍。'''

# 分詞 (cut 是用於回傳 generator 的斷詞函式，lcut 是將斷詞以 list 格式回傳)
tokens = jieba.lcut(text)

# 去除停用詞，這邊為了簡化，我們預先定義一個停用詞列表
stop_words = ['和', '的', '是', '這']
tokens = [word for word in tokens if word not in stop_words]; tokens

# 轉換為詞向量，這裡使用詞袋模型
vectorizer = CountVectorizer()
X = vectorizer.fit_transform([' '.join(tokens)])
print(vectorizer.get_feature_names_out())
print(X.toarray())

# 透過 pandas 來預覽結果
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out()); df
```

TF-IDF(Term Frequency-Inverse Document Frequency)

TF-IDF 是一種統計方法，用來評估一個「詞」對於一個文件集或一個語料庫中的其中一份文件的「重要程度」。它的值越大，則詞對文件的重要性越高。

TF-IDF 是 Term Frequency（詞頻） 和 Inverse Document Frequency（逆向文件頻率）兩個詞的組合，也就是詞頻（Term

Frequency，TF）和逆向文件頻率（Inverse Document Frequency，IDF）的「乘積」(TF * IDF)。

這種表示方法的特點，是可以反映出詞語對於特定文件的專屬性，比詞袋模型多了考慮字詞的稀有程度，進而能將重要的關鍵詞突顯出來。

英文部分

```
# 匯入套件
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# 假設我們有這些英文文本
docs = [
    'This is the first document.',
    'This document is the second document.',
    'And this is the third one.',
    'Is this the first document?',
]

# 創建 TF-IDF 範例
vectorizer = TfidfVectorizer()

# 進行 TF-IDF 轉換
X = vectorizer.fit_transform(docs)

# 顯示結果
print(vectorizer.get_feature_names_out())
print(X.toarray())

# 透過 pandas 來預覽結果
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out()); df
```

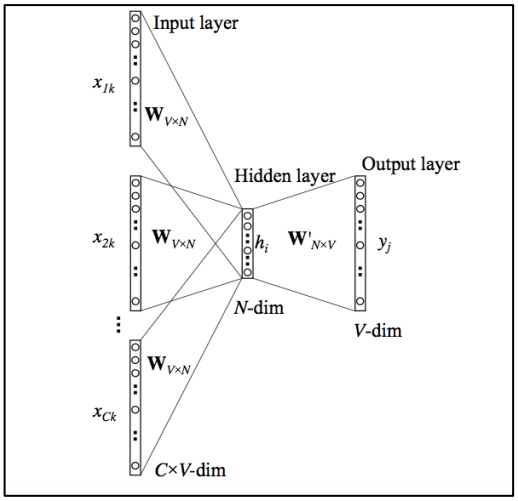
中文部分

```
import jieba
from sklearn.feature_extraction.text import TfidfVectorizer

# 假設我們有這些中文文本
```

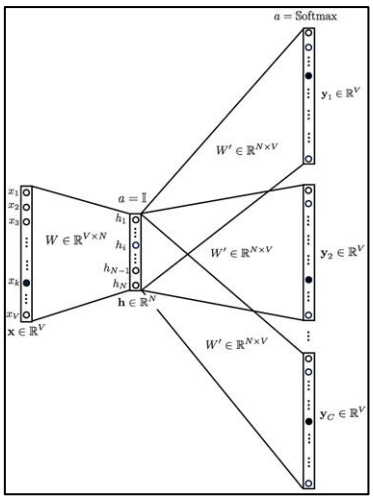
```
documents = [
    '這是第一份文件',
    '這份文件是第二份文件',
    '而這是第三個',
    '這是第一份文件嗎'
]

# 先進行分詞
documents = [' '.join(jieba.cut(doc)) for doc in documents]

# 創建 TF-IDF 範例
vectorizer = TfidfVectorizer()

# 進行 TF-IDF 轉換
X = vectorizer.fit_transform(documents)

# 顯示結果
print(vectorizer.get_feature_names_out())
print(X.toarray())

# 透過 pandas 來預覽結果
df = pd.DataFrame(X.toarray(),
columns=vectorizer.get_feature_names_out()); df
```

詞向量(Word2Vec)

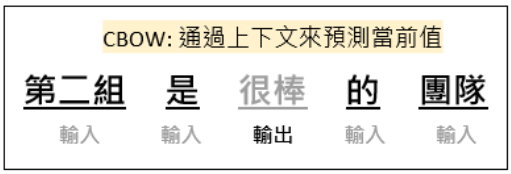
詞嵌入(Word Embedding)是一種更進階的「詞向量化」方法，它可以捕捉詞與詞之間的相關性和語義資訊。詞嵌入通常使用深度學習方法（如 Word2Vec 或 GloVe）來學習一個高維度的詞向量(Word Vector)，這個詞向量能將語義相似的字詞映射(map)到向量空間(Vector Space)的相近位置。因此，詞嵌入能比 BoW 更好地保留語言的語義結構。

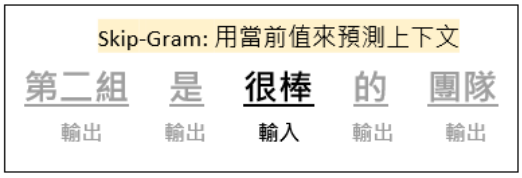


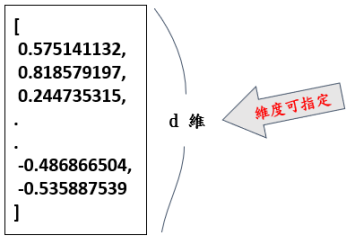
圖： Continuous Bag-of-Words，CBOW



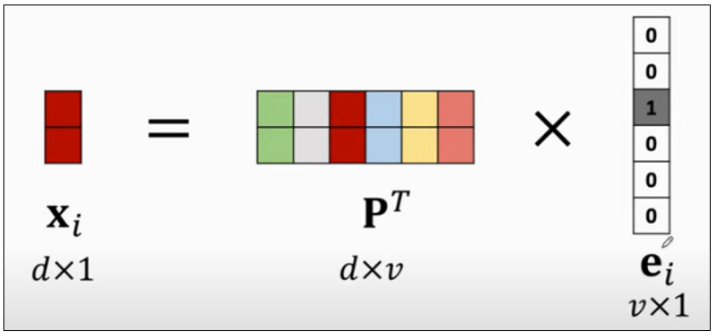
圖： Skip-Gram







說明：
- 上面的嵌入代表一個「字詞」(Word)
- 一個字詞擁有一個獨立的嵌入 (Embedding)



圖：將 one hot encoding 降低維度的方式；取自 https://www.youtube.com/watch?v=6_2_2CPB97s

3_詞向量(Word2Vec)

```
# 安裝 gensim
!pip install gensim


# 匯入套件
from gensim.models import Word2Vec
import jieba

# 句子 (也可以是一篇文章)
sentences = [
    "我喜歡閱讀書籍",
```

```
    "我也喜歡使用電腦來學習新的知識",
    "閱讀可以開闊我們的視野",
    "電腦是現代學習的重要工具"
]

# 使用 jieba 進行斷詞
sentences = [jieba.lcut(sentence) for sentence in sentences]

# 設定參數
sg = 0 # sg=1 -> skip-gram, sg=0 -> cbow

# 向前看幾個字或向後看幾個字
window_size = 2

# 向量維度
vector_size = 100

# 訓練幾回
epochs = 20

# 最少多少個字才會被使用
min_count = 1

# seed
seed = 42

# 建立 Word2Vec 模型
model = Word2Vec(
    sentences,
    vector_size=vector_size,
    window=window_size,
    sg=sg,
    min_count=1,
    seed=seed,
    epochs=epochs)

# 取得 "閱讀" 這個詞的詞向量
```

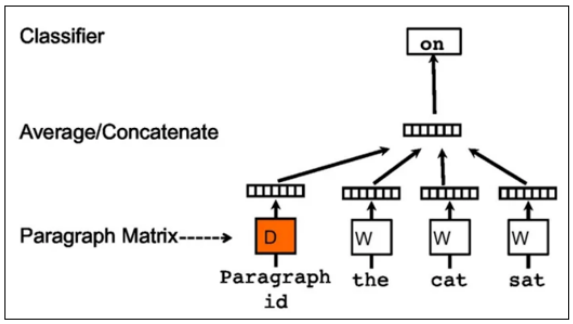
```
vector = model.wv['閱讀']

# 輸出 "閱讀" 的詞向量
print(vector)

# 儲存模型
model.save('word2vec.model')

# 讀取模型
loaded_model = Word2Vec.load("word2vec.model")

# 尋找相近的字詞
loaded_model.wv.most_similar('閱讀', topn=10)

# 計算相近度
loaded_model.wv.similarity('書籍', '知識')
```

補充：文件向量(Doc2Vec)

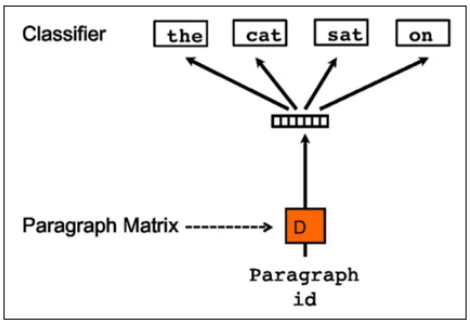
Doc2Vec 是一種將文件 (document) 轉換成數字向量的技術，這種技術能夠將文件中的字詞順序和上下文資訊編碼進向量當中。這是 Word2Vec 方法的延伸，Word2Vec 是一種將字詞轉換成數字向量的技術，它同樣能夠編碼字詞的上下文資訊。

Doc2Vec 包含兩種模型，分別是 Paragraph Vector-Distributed Memory (PV-DM) 模型和 Paragraph Vector-Distributed Bag of Words (PV-DBOW) 模型。DM 模型預測目標詞語，給定上下文字詞和文件向量，而 DBOW 模型則忽略上下文詞語，只用文件向量預測目標字詞。

Doc2Vec 的主要優點是它能夠學習到文件的抽象表示，這些表示能夠反映文件的主題和風格。這些表示可用於各種自然語言處理任務，例如文本分類、情感分析和文件相似性比較等。Doc2Vec 是一種學習文件的數字表示方法，這種表示能夠捕捉到文件的上下文和語義資訊，並且能夠用於自然語言處理的各種任務。



圖： PV-DM (Paragraph Vector-Distributed Memory)



圖： Paragraph Vector-Distributed Bag of Words

參考資料：https://zhuanlan.zhihu.com/p/336921474

3_文件向量(Doc2Vec)

```
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument
import jieba

# 句子 (也可以是一篇文章)
documents = [
    "我喜歡閱讀書籍",
    "我也喜歡使用電腦來學習新的知識",
    "閱讀可以開闊我們的視野",
    "電腦是現代學習的重要工具"
]

# 使用 jieba 進行斷詞
documents = [jieba.lcut(doc) for doc in documents]

# 將文本標記為 TaggedDocument 格式
documents = [TaggedDocument(doc, [index]) for index, doc in
enumerate(documents)]

# 建立 Doc2Vec 模型
model = Doc2Vec(
    documents,
    vector_size=300,
    window=2,
    min_count=1,
    epochs=20
```

```
)

# 查找索引為 0 的文檔向量
vector = model.dv[0]

# 輸出索引為 0 的文檔向量
print(vector)

# 輸出索引為 0 的文檔向量
print(vector)
```

Module 4. 統計語言模型

簡介

統計語言模型（Statistical Language Model, SLM）是用來描述和計算一段文字在某種語言中出現機率的模型。該模型的主要目標是：給定一個詞的序列（也就是一段文字），計算這個序列出現的概率。

統計語言模型的核心是「條件機率」，也就是在一些詞已知的情況下，預測下一個詞的機率。例如，假設我們有一個詞的序列 "I am going to the"，統計語言模型的目標就是計算各種可能的下一個詞的概率，例如 "store"、"park"、"office" 等。

可以嘗試在 Google 搜尋欄位中，逐一輸入文字，它會預測使用者可以選取的文字或關鍵字。

統計語言模型有很多種類，包括以下幾種：

- **N-gram 模型**：
  這種模型假設每個詞的出現只與前面的 N-1 個詞相關。例如，在 bi-gram 模型（N=2）中，我們假設每個詞的出現只與前面的一個詞相關。
- **隱藏式馬可夫模型**（Hidden Markov Model, HMM）：
  這種模型假設文字是由一個隱藏的馬可夫鏈 (Markov chain) 生成的，每個詞的出現只與前面的狀態 (state) 相關。
- **主題模型**（Topic Model）：
  這種模型假設一段文字中的每個詞都與某種「主題」相關，並試圖找出這些主題。

這些模型都有各自的假設和限制，並且在不同的情況下有不同的表現。在近年來，隨著深度學習的發展，基於神經網絡的語言模型（例如 Transformer 的 BERT、GPT 等）也越來越流行。這些模型能得更複雜的語言結構和語境資訊，從而得到更好的結果。

N-gram 模型

簡單用法

4_N-gram 模型

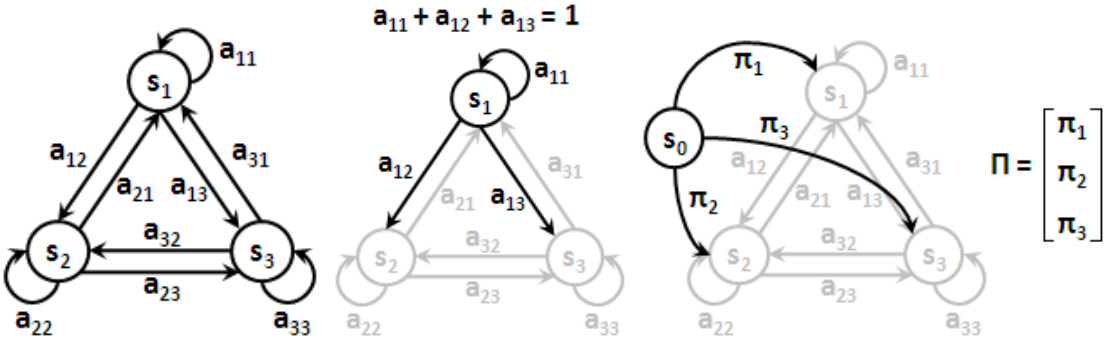
```
import jieba

def generate_ngrams(words, n):
    # 建立空的 n-grams 列表
    ngrams = []

    # 迭代詞彙列表中的每個詞
    for i in range(len(words) - n + 1):
        # 新增下一個 n-gram
        ngrams.append(words[i:i+n])

    return ngrams

# 範例輸入
text = "我喜歡閱讀書籍，也喜歡使用電腦來學習新的知識"

# 使用 jieba 進行斷詞
words = jieba.lcut(text)

# 產生 bi-grams
bigrams = generate_ngrams(words, 2)

# 輸出結果
for bigram in bigrams:
    print(bigram)
```

計算下一個字出現的機率

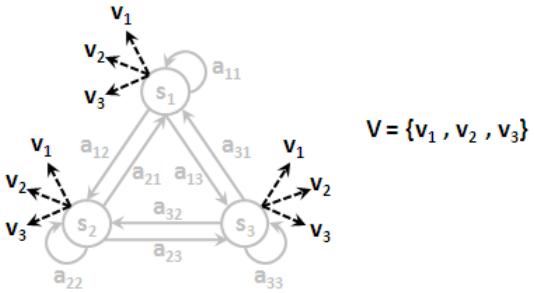
4_N-gram 模型

```
from collections import defaultdict, Counter

# 假設我們已經有了一個已斷詞的文本列表
tokens = ['我', '愛', '吃', '蘋果', '，', '我', '也', '愛', '吃', '橘子
']

# 建立一個預設為空列表的詞典
ngram_dict = defaultdict(Counter)

# 指定我們要使用的 N-gram 的 N 值
N = 2

# 遍歷所有的詞彙
for i in range(len(tokens)-N):
    # 得到 N-gram 和它的下一個詞彙
    ngram = tuple(tokens[i:i + N])
    next_token = tokens[i + N]
    # 更新詞典
    ngram_dict[ngram][next_token] += 1

# 轉換次數為機率
for ngram, next_tokens in ngram_dict.items():
    total_count = sum(next_tokens.values())
    for next_token, count in next_tokens.items():
        ngram_dict[ngram][next_token] = count / total_count

# 假設我們要預測 "我愛" 的下一個詞彙
ngram = ('愛', '吃')

# 從詞典中取得所有可能的下一個詞彙和它們的機率
next_tokens_probs = ngram_dict[ngram]

# 將它們按照機率排序，取前 k 個
k = 2
top_k_next_tokens = sorted(next_tokens_probs.items(), key=lambda x:
x[1], reverse=True)[:k]
print(top_k_next_tokens)
```

隱藏式馬可夫模型(Hidden Markov Model, HMM)



圖： 馬可夫模型



圖： 隱藏式馬可夫模型

主題模型(Topic Model)

4_主題模型(Topic Model)

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import jieba

# 假設我們有以下的文本資料，這些文本資料可能是從檔案讀取，也可能是從網路爬蟲取得
documents = [
    '這是第一個文本的內容，主要在談論科技產品如電腦、手機等等。',
    '這是第二個文本的內容，也在談論科技產品，但更偏重在智能家居。',
    '這是第三個文本的內容，主要在談論食物如披薩和漢堡等等。',
    '這是第四個文本的內容，也在談論食物，但更偏重在烘焙製品。',
    # 更多的文本資料...
]
```

```
# 使用 jieba 進行斷詞，並將斷詞的結果以空格分隔
documents = [' '.join(jieba.cut(doc)) for doc in documents]

# 使用 CountVectorizer 進行詞頻統計，並建立詞頻矩陣
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

# 使用 LDA 進行主題模型訓練
lda = LatentDirichletAllocation(n_components=2, random_state=42)
lda.fit(X)

# 印出每一個主題下權重較高的詞語
feature_names = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda.components_):
    print(f"Topic #{topic_idx + 1}:")
    print(" ".join([feature_names[i] for i in topic.argsort()[:-10 - 1:-1]]))
```

Module 5. 深度學習與 NLP

人工神經網路基礎

人工神經網路（Artificial Neural Networks，ANNs）是機器學習中的一種運算方式，這種運算方式的靈感，來自於生物的神經網路，用於模仿人腦進行思考和學習的過程。

以下是對於人工神經網路在自然語言處理（NLP）中的基礎說明：

- 神經元（Neuron）：
  神經元是神經網路中的基本單元，每個神經元接收一些輸入，對輸入進行加權計算（每個輸入都有相應的權重），然後通過一個激活函數（如 ReLU、tanh、sigmoid 等）產生輸出。
- 層（Layer）：
  神經網路由多個層組成。第一層稱為輸入層，最後一層稱為輸出層，中間的層稱為隱藏層。在 NLP 任務中，輸入通常是詞語的向量表示（如詞嵌入），輸出則取決於具體的任務（例如，在文本分類中，輸出可能是各個類別的機率）。
- 前向傳播（Forward Propagation）：

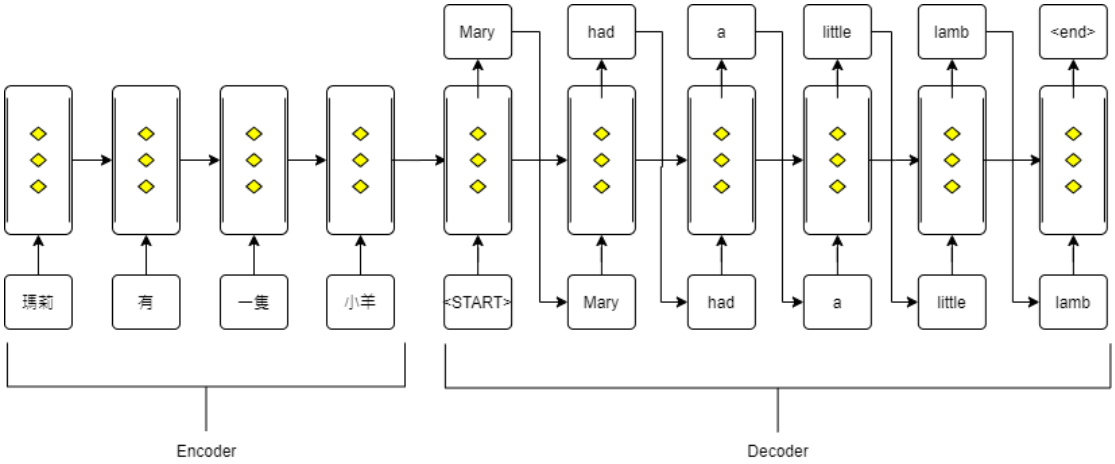
神經網路的學習過程由兩個步驟組成：前向傳播和反向傳播。在前向傳播中，輸入資料通過網路向前移動，每一層都進行運算並將結果傳遞給下一層，直到輸出層。

- 反向傳播（Back Propagation）和梯度下降（Gradient Descent）：反向傳播是一種計算梯度的方法，它是根據輸出結果和真實答案的差異（即損失函數），計算每個權重對損失的影響，然後透過梯度下降來更新權重以最小化損失。

序列到序列(Seq2Seq)的概念

序列到序列（Sequence-to-Sequence，簡稱 Seq2Seq）模型是一種用於處理序列資料的深度學習架構。如其名稱所示，Seq2Seq 模型的目的是將一個序列映射到另一個序列，並且兩個序列的長度不一定相等。這使得它非常適合於如機器翻譯、語音辨識、文本摘要等任務。

Seq2Seq 模型主要由兩部分組成：編碼器（Encoder）和解碼器（Decoder）。每一部分都是一個循環神經網路（RNN）或其他一些類型的神經網路。在機器翻譯的例子中，編碼器會接受一個語言（如英語）的句子作為輸入，並將其轉換為一種內部表示（稱為上下文向量）。然後，解碼器將此上下文向量作為輸入，並產生另一種語言（如法語）的句子作為輸出。



圖： Seq2Seq 架構

一個重要的概念是「注意力機制」(Attention Mechanism)。在基本的 Seq2Seq 模型中，編碼器需要將所有的輸入資訊壓縮到一個固定大小的上下文向量中，這可能會造成資訊的丟失。為了解決這個問題，注意力機制允許解碼器在生成每個詞的時候，都能查看編碼器的所有隱藏狀態（而非僅看最後一個輸出），並從中選擇對當前位置最有用的狀態。這種方式顯著提高了 Seq2Seq

模型在長序列上的表現。

Seq2Seq 模型的另一個重要應用是對話系統或聊天機器人。在這種情況下，編碼器的輸入可能是一個問題，解碼器的輸出則是對該問題的回答。

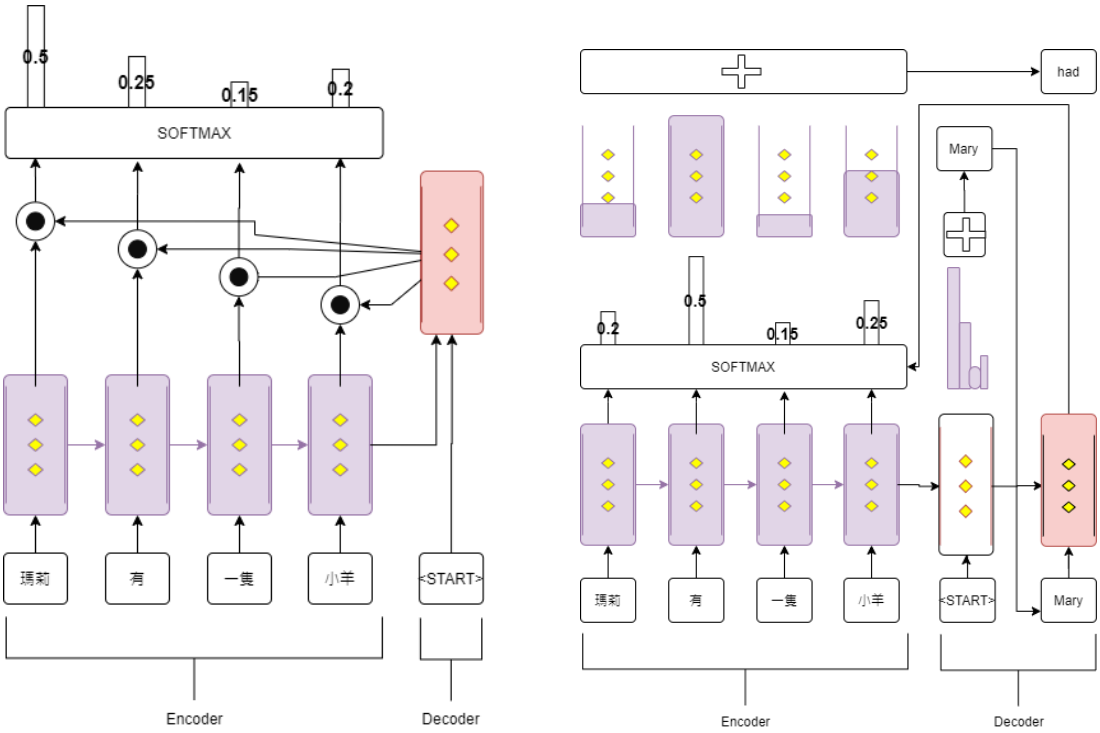
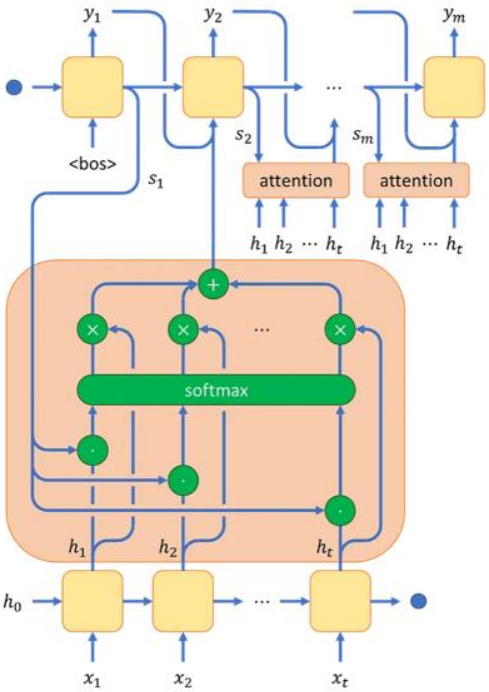


圖: 整合 attention 機制



$$H = [h_1 \quad h_2 \quad \cdots \quad h_t]^T$$

$$\text{Attention}(s_i, H) = \text{softmax}\left(\frac{s_i H^T}{\sqrt{d_h}}\right) \cdot H$$

圖： 更詳細的 attention 流程

RNN 與 LSTM 介紹與應用

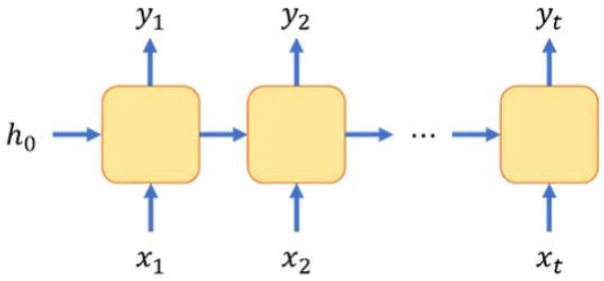
循環神經網路（Recurrent Neural Network，簡稱 RNN）

是一種特殊的人工神經網路，專門設計來處理有順序性的輸入資料。它具有記憶性，可以記住前面已處理過的資訊，並將這些資訊應用在後續的處理過程中。

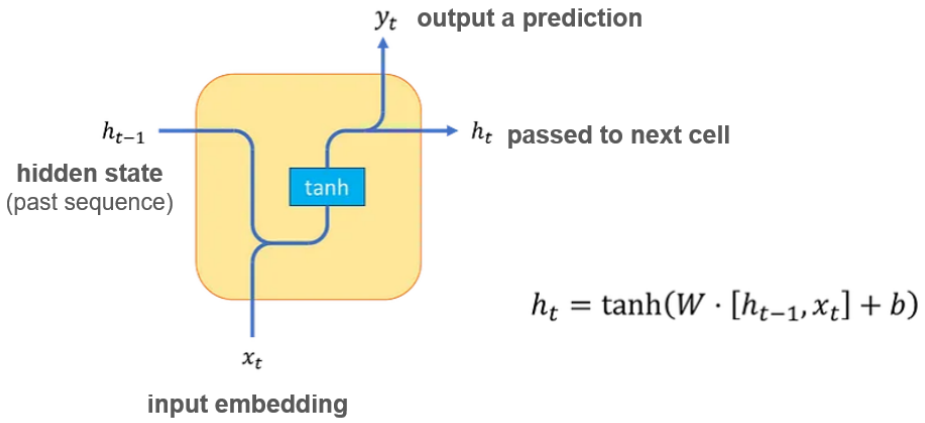
RNN 的主要結構是一個將隱藏狀態（hidden state）反饋（feed back）到自身的循環結構，這讓 RNN 能夠保存和利用過去的資訊。在每一個時間步，RNN 會接收一個新的輸入和前一時間步的隱藏狀態，然後產生一個新的隱藏狀態，這個新的隱藏狀態將會在下一時間步被用作輸入。

因此，RNN 的一個重要特性就是它具有「記憶」的能力，並且能夠使用這些記憶來處理和預測序列中的未來元素。這使得 RNN 非常適合於處理時間序列資料，如股票價格、天氣預報，或是自然語言處理中的文字和語音資料等。

然而，傳統的 RNN 存在所謂的「梯度消失（Vanishing Gradient）」和「梯度爆炸（Exploding Gradient）」問題，這使得 RNN 難以學習和理解過長的序列。為了解決這些問題，學者們提出了一些 RNN 的變體，如長短期記憶網路（Long Short-Term Memory, LSTM）和門控循環單元（Gated Recurrent Unit, GRU），它們在許多序列相關的任務中已經取得了顯著的效果。



圖： RNN 基本架構



圖：RNN 基於前一個字來理解下一個字的方式

LSTM（Long Short-Term Memory）

是一種特殊的循環神經網路（RNN）結構，被設計出來解決傳統 RNN 在學習長期依賴性時遇到的所謂梯度消失（Vanishing Gradient）問題。一個 LSTM 單元由三個主要部分組成，這些部分協同工作，可以讓 LSTM 單元記住或遺忘資訊：

- **遺忘門（Forget Gate）：**
  遺忘門決定 LSTM 單元應該遺忘多少過去的資訊。它使用 sigmoid 函數，可以輸出一個介於 0（忘記全部）和 1（全部保留）之間的數字。
- **輸入門（Input Gate）：**
  輸入門決定 LSTM 單元應該記住多少新的資訊。就像遺忘門一樣，它也使用 sigmoid 函數。然而，此外，LSTM 單元還將新的候選狀態（用 tanh 函數生成）與輸入門的輸出相乘，來決定哪些新的資訊應該被記住。
- **輸出門（Output Gate）：**
  輸出門決定 LSTM 單元的輸出應該是什麼。它使用 sigmoid 函數來決定哪些狀態應該被輸出，然後將這些狀態與單元狀態的 tanh 的輸出相乘，生成 LSTM 單元的最終輸出。

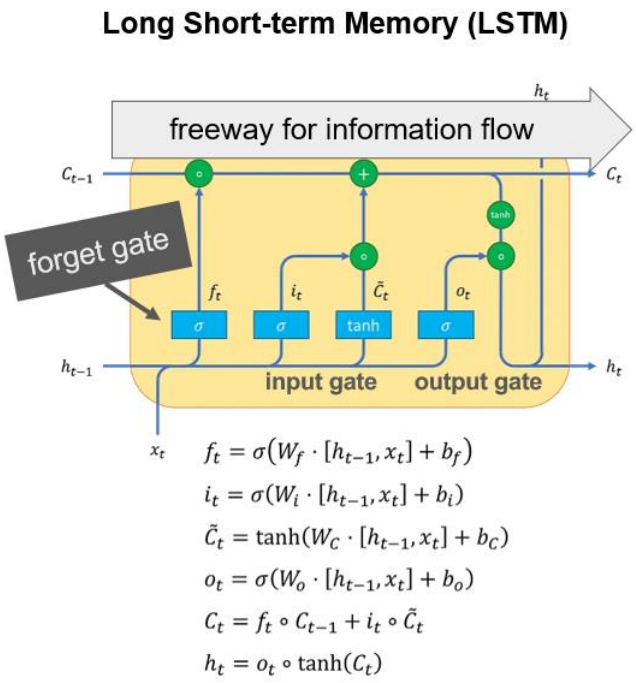


圖: LSTM 透過三個主要部分來決定哪些東西需要記得或忘記

這三個門的功能使 LSTM 能夠在處理長序列資料時，根據需要記住或遺忘資訊，從而有效地學習長期依賴關係。因此，LSTM 已經被廣泛地用在語音識別、語言模型、機器翻譯等自然語言處理的任務中。

LSTM 雖然能有效地解決梯度消失問題，但是仍然存在梯度爆炸問題，所以在實際操作時，還需要結合梯度裁剪（Gradient Clipping）等技巧使用。

Module 6. Transformer 神經網路模型

簡介

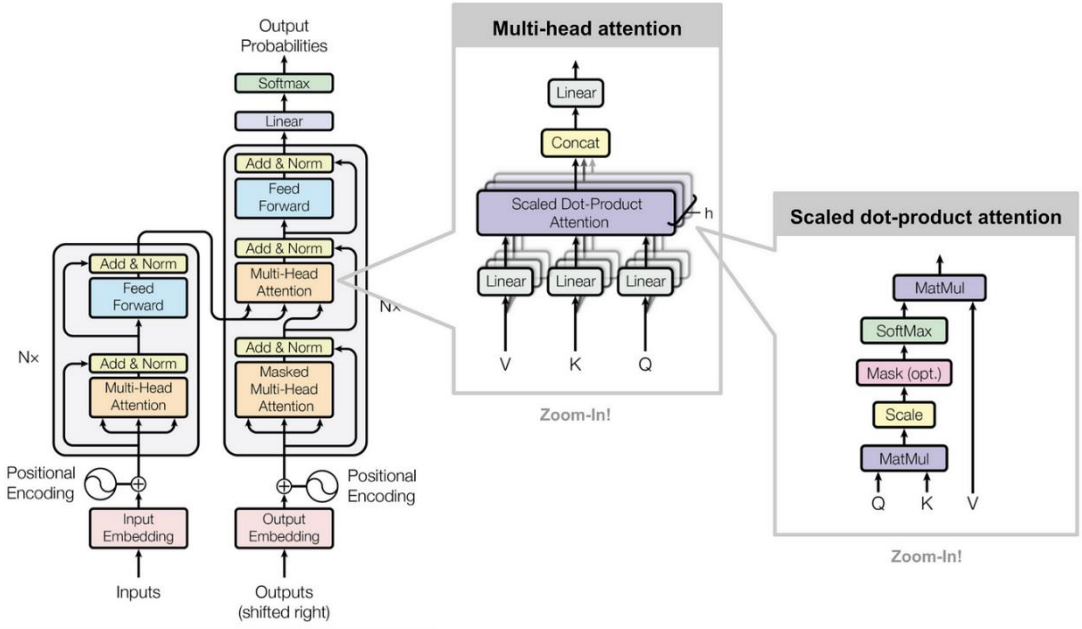
Transformer 是一種在自然語言處理領域中廣泛使用的深度學習模型，尤其在機器翻譯和文字生成等任務上表現優異。它的特點在於完全放棄了過去常見的循環神經網路（RNN）架構，改用全新的方法處理序列資料，這使得它能夠更有效地處理長距離依賴（long-distance dependencies）問題，並且具有很高的平行運算能力。

主要組成部分是 "自注意力機制"（Self-Attention Mechanism）或稱 "Scaled Dot-Product Attention"，它使得模型能夠權衡序列中的所有元素，並根據每個元素與其他元素的關係賦予不同的權重。這樣，模型就能夠在確定一個元素的表示時考慮到所有的上下文資訊。

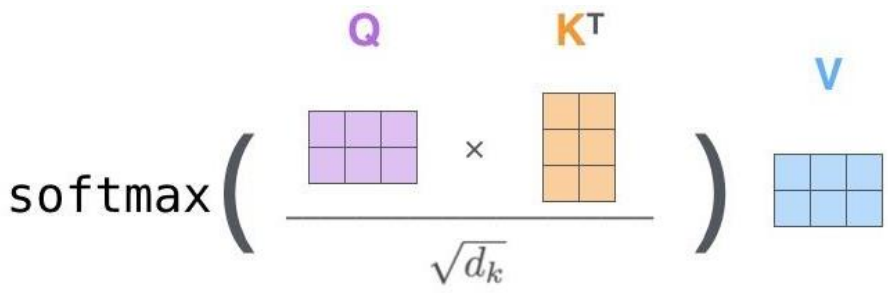
另一個主要的組成部分是 "位置編碼"（Positional Encoding），因為 Transformer 本身並無法處理序列的順序資訊，因此需要通過加入位置編碼來提供該信息。

基本結構由兩部分組成：編碼器（Encoder）和解碼器（Decoder）。編碼器由多個相同的層疊加在一起形成，每層包含一個自注意力子層和一個全連接的前饋網路子層。解碼器也由多個層組成，每層還多了一個注意力子層來關注編碼器的輸出。

最著名的應用可能就是 "BERT"（Bidirectional Encoder Representations from Transformers）和 "GPT"（Generative Pretrained Transformer）等預訓練語言模型，這些模型在各種自然語言處理任務上都取得了當時最好的效果。



圖：Transformer 的神經網路架構



圖：Attention 機制

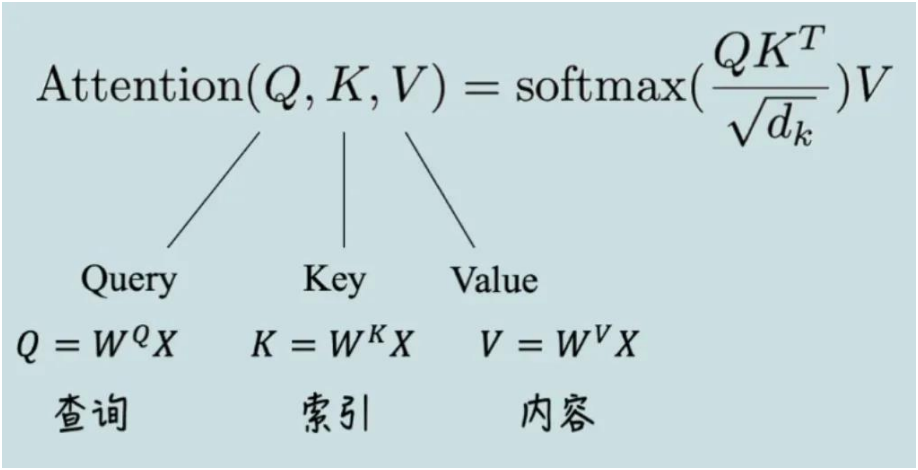


圖: 實際的 Attention 計算方式

BERT

BERT (Bidirectional Encoder Representations from Transformers) 是由 Google 在 2018 年提出的一種預訓練的深度學習模型，主要用於自然語言處理 (NLP) 任務。它具有很好的傳遞語境訊息的能力，並且在多項 NLP 任務中都取得了當時的最好效果。

以下是關於 BERT 的一些重要特性和概念：

- 雙向訓練：
  在傳統的語言模型中，只考慮上下文或者下文中的單詞（例如，左到右或右到左）。但是，BERT 使用的訓練方法是雙向的，它一次性考慮了上下文中的所有單詞。這對於理解語境中的每個單詞的意義有很大幫助。
- Transformer 架構：
  BERT 以 Transformer 架構為基礎，將原本用於處理序列到序列問題的 Transformer 模型調整為只包含其編碼器部分。這種架構可以處理長距離的依賴關係，並且能夠平行化處理，使得訓練更加高效。
- Masked Language Model：
  BERT 的訓練方法之一是 Masked Language Model (MLM)，其中一部分輸入的單詞會被隨機遮蔽，然後模型預測這些被遮蔽的單詞。
- Next Sentence Prediction：
  BERT 的另一種訓練方法是 Next Sentence Prediction (NSP)，模型預測第二個句子是否在原文中緊接在第一個句子之後。這種訓練方法使得 BERT 能夠理解句子之間的關係。

這些特性使得 BERT 可以用於多種 NLP 任務，例如問答系統、命名實體

識別、情感分析等。尤其是在微調（Fine-tuning）後，BERT 通常能夠在這些任務中達到出色的效果。

GPT

GPT (Generative Pre-training Transformer) 是由 OpenAI 研發的語言模型，主要被應用於自然語言處理 (NLP) 的任務。

以下是 GPT 的一些關鍵特性：

- Transformers 架構：
  GPT 基於 Transformer 的架構，這種架構最初被設計為處理序列到序列的問題。但在 GPT 中，只使用了 Transformer 的解碼部分。
- 語言模型預訓練：
  GPT 是一種生成式的預訓練模型，這意味著它在大量的文本資料上進行非監督式學習，嘗試預測下一個單詞，以此學習語言的模式。然後，在特定的下游任務（如文本分類、問答等）中，通過微調(fine-tune)來適應該任務。
- 自回歸性質：
  GPT 使用了一種叫做自回歸(auto regression)的方式來生成文本。這種方式是一種自左向右的方式，也就是說，每一個單詞都只能看到它前面的單詞，不能看到後面的單詞。
- 遷移學習：
  與傳統的 NLP 模型相比，GPT 的一個主要特點是它的轉移學習能力。這種模型可以在一種任務上進行預訓練，然後在另一種不同的任務上進行微調，從而節省了大量的計算資源。
- 規模化：
  GPT 模型尤其以其規模大而著名。GPT-3，最新版的 GPT，有 1750 億個模型參數，這使得它能夠生成極為自然並符合語境的文本。

在多種 NLP 任務中，GPT 都展示了出色的性能，包括文本生成、文本分類、翻譯、摘要生成等。

參考資料

[1] NLTK 初學指南(一)：簡單易上手的自然語言工具箱－探索篇
https://medium.com/pyladies-taiwan/nltk-初學指南-一-簡單易上手的

自然語言工具箱-探索篇-2010fd7c7540
[2] NLP 的基本執行步驟(II) — Bag of Words 詞袋語言模型
https://medium.com/@derekliao_62575/nlp 的基本執行步驟-ii-bag-of-words-詞袋語言模型-3b670a0c7009
[3] [Day4] 語言模型(一)-N-gram
https://ithelp.ithome.com.tw/articles/10259725
[4] [Day5] 語言模型(二)-N-gram 實作
https://ithelp.ithome.com.tw/articles/10259982
[5] 直觀理解 LDA (Latent Dirichlet Allocation) 與文件主題模型
https://tengyuanchang.medium.com/直觀理解-lda-latent-dirichlet-allocation-與文件主題模型-ab4f26c27184
[6] Day 7 把文字裝成一袋？Bag of Word (BoW) & TF-IDF 在 NLP 中的應用
https://ithelp.ithome.com.tw/articles/10288695
[7] Extract Entities From Text In Natural Language Processing
https://medium.com/@nutanbhogendrasharma/extract-entities-from-text-in-natural-language-processing-66c77169973d
[8] NLTK 初學指南(三)：基於 WordNet 的語義關係表示法 — 上下位詞結構篇
https://medium.com/pyladies-taiwan/nltk-初學指南-三-基於-wordnet-的語義關係表示法-上下位詞結構篇-4874fb9b167a
[9] 真 IT 小叮噹深度學習：自然語言處理（二）文本處理流程
https://kknews.cc/tech/xlmxzmo.html
[10] jieba
https://github.com/fxsjy/jieba
[11] 中文斷詞
https://blog.maxkit.com.tw/2020/08/blog-post.html
[12] 彙整中文與英文的詞性標註代號：結巴斷詞器與 FastTag / Identify the Part of Speech in Chinese and English
https://blog.pulipuli.info/2017/11/fasttag-identify-part-of-speech-in.html
[13] POS tagging 词性标注 之 武林外传版
http://codewithzhangyi.com/2019/03/12/pos/
[14] [機器學習] MultinomialNB 貝氏(貝葉氏)分類-理論篇
https://medium.com/@d101201007/機器學習-multinomialnb-貝氏-貝葉氏-分類-df1d59b6fd9d
[15] 樸素貝葉斯分類-實戰篇-如何進行文字分類

https://www.gushiciku.cn/pl/pjta/zh-tw
[16] gensim 中 word2vec 使用方法记录
https://blog.csdn.net/MarkAustralia/article/details/128466581
[17] gensim 函数库中 Word2Vec 函数 size，iter 参数错误解决（ __init__() got an unexpected keyword argument ‘size‘）
https://blog.csdn.net/lcy6239/article/details/115786432
[18] [Day 12] 時間都去哪了？資料前處理：抓住那個欠錢不還的傢伙－詞性標註之 HMM 的應用
https://ithelp.ithome.com.tw/m/articles/10298368
[19] 何謂 Artificial Neural Network?
https://r23456999.medium.com/%E4%BD%95%E8%AC%82-artificial-neural-netwrok-33c546c94794
[20] 25. 簡介 LSTM 與 GRU
https://medium.com/programming-with-data/25-簡介 lstm-與 gru-3e0eaa100d29
[21] 28. 注意力機制（Attention mechanism）
https://medium.com/programming-with-data/28-注意力機制-attention-mechanism-f3937f289023
[22] WordCloud for Python documentation
https://amueller.github.io/word_cloud/index.html
[23] GitHub: word_cloud
https://github.com/amueller/word_cloud
[23] Beautifully Illustrated: NLP Models from RNN to Transformer
https://towardsdatascience.com/beautifully-illustrated-nlp-models-from-rnn-to-transformer-80d69faf2109
[24] Hidden Markov Model
https://web.ntnu.edu.tw/~algo/HiddenMarkovModel.html