ML Final Report

組別：r06921066_讚嘆逸霖師父簽PHD

題目：TV-conversation

組長：r06921066 劉宇閎

組員：r06921010 蕭羽庭

r06921016 簡雅慧

r06921017 詹少宏

1. 工作分配

r06921066 劉宇閎 : report撰寫, seq2seq模型訓練

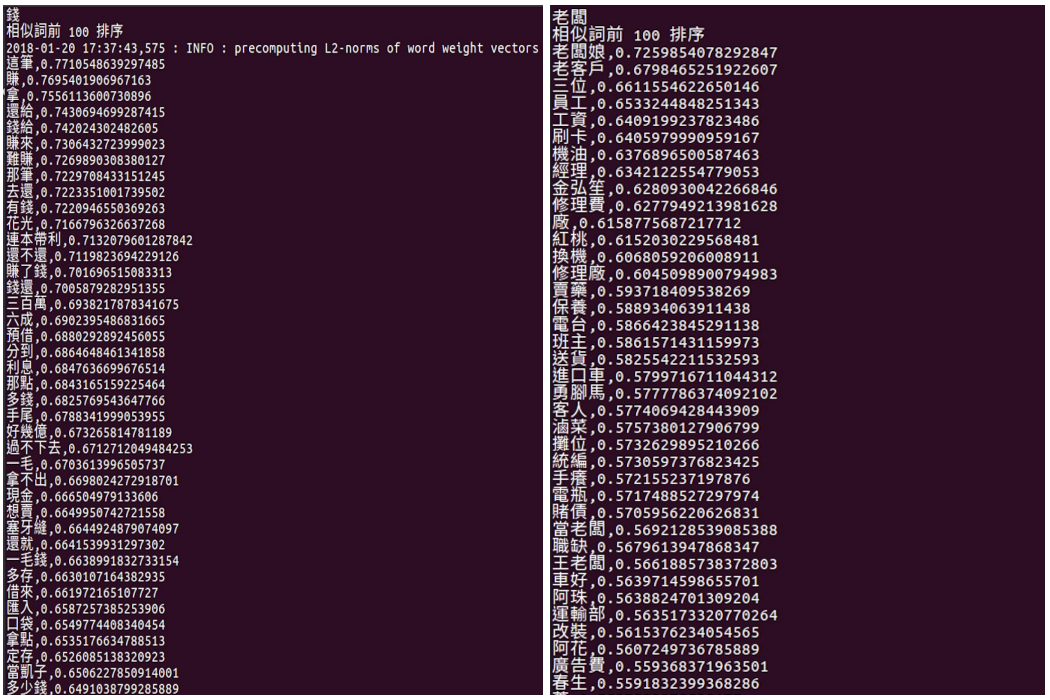
r06921010 蕭羽庭 : Train word vector , seq2seq模型訓練

r06921016 簡雅慧 : Train word vector , word vector相似性評估

r06921017 詹少宏 : report撰寫, word vector相似性評估, dnn 實驗

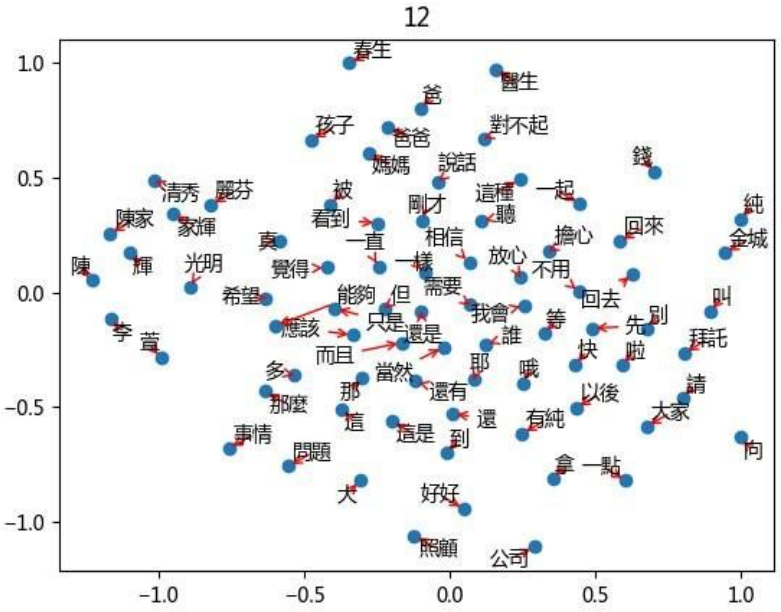
2. Preprocessing

首先由於中文並不像英文依樣有清楚的斷句（space）來做單字的區隔，所以我們使用jieba套件來做中文字的切割。在切割的單字出來後，我們將已切割的單字丟入gensim來將單字或單詞轉換為word vector但因為其中有許多的參數可以做調整，為了使整體最佳化我們在網路上找了一個檢測word vector準確度的套件，使用方法為輸入一個詞，程式會在整個word vector中找出cosine similarity最大的前100個詞彙輸出並且秀出他的相似度，透過這種方法我們簡單挑選出了幾組最佳的參數並且使用這幾組word vector來將我們的題目作轉換。測試結果如下圖一所示：



圖一、word2vetor model 相似度實測結果

除了使用這方法外，我們也參考了作業六的Visualization產生了下圖二用以檢視我們Word2Vec的成果：



圖二、Word2Vec 視覺化結果

然而，在中文中具有許多的雜訊，例如：「你、我、他、的」等詞彙。這些雜訊並沒有辦法完整地讓我們得知整體句子想表達的方向，以及究竟該句想表達的意思，所以我們另外加入了stop word，當偵測的stop word時就將其視為雜訊並將他濾掉。此外，也發現有些標點符號在其中也有許多可能被視為雜訊，如『』“。等，因此，我們也將其加入stop word中，而stop word的使用時機為testing，當偵測到題目有stop word時會自動跳過該字元，透過這種方式可以讓整體的雜訊更加被抑制且較不會影響到預測的結果。

3. gensim 參數介紹

在使用gensim套件前為了使我們的調整對整體有實際意義的影響，所以我們稍微查詢了一下gensim套件參數的意義。紅字部分為我們覺得調整後有實際影響的：

- **sentences**：可以是一個list，對較大的dataset，建議使用BrownCorpus, Text8Corpus或lineSentence建構。
- **sg**： 用於設置訓練算法，default 值為0，對應CBOW算法；sg=1則採用skip-gram算法。
- **window**：當前詞與預測出來的詞在一個句子中的最大距離為多少。
- **alpha**: learning rate
- **seed**：initial state
- **min_count:**可以將字典大小做一些簡易的限制. 當整個dataset中詞出現的次數小魚min_count時系統會自動將其濾除，可以減少雜訊對整個字典的影響，default值為5。
- **max_vocab_size:** 用來設定在詞向量training期間的RAM限制。當有過多的獨立單詞出現在字典中且數量已超過max_vocab_size時就會自動將出現次數最低的濾除掉可以防止資料量太大時導致暫存器不足。另

外經過計算後會發現每一千萬個詞需要約1GB的暫存大小，故在此我們並沒有將其做特別的限制，採用default值 None。

- **sample:** 高頻詞匯的隨機採樣使用的threshold ，default值為1e-3 。
- **negative:** 如果>0,则会采用negativesampling，用於设置noise words。
- **cbow_mean:** 若為0，則採用上下文詞向量的和，如果為1則採用平均值。default 值為1，此參數只有使用CBOW的时候才起作用。
- **hashfxn：** hash函數来初始化整個weight。Default為使用python的hash函数。
- **iter**： 迭代次數，default值為5。在我們最好的結果中，iter 為120。
- **trim_rule：**用來設定詞彙表的整理規則，可以拿來選定哪些單詞要被保留下或是被刪除。Default值為None。
- **sorted_vocab：** 如果為1（default），則在分配word index 時會先對詞基於頻率做降序的排序。
- **batch_words：**每次training的batch size，即單字數，default為10000。
- **workers:** 訓練參數時所使用的thread 數目。

4. Model Description

- **Word Embedding:**

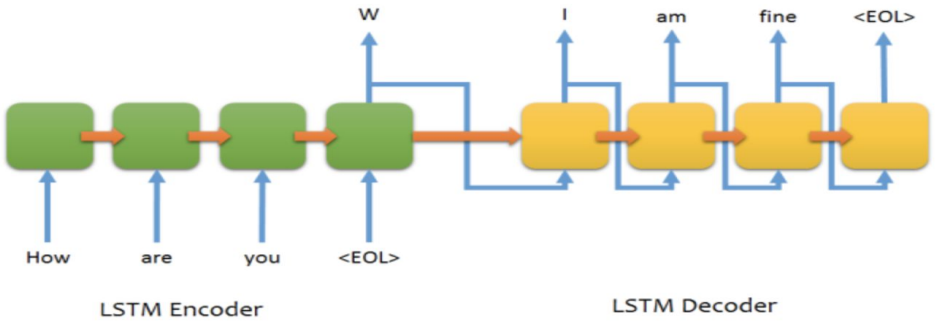
  此種實驗為單純的使用gensim套件來做word vector的training，故在此不多做太多的介紹，而經過多次實驗後目前得到最佳分數所使用的參數為如下表一所示：

Parameters	Values
alpha	0.01
size	64
window	5
iter	120
min_count	5
sample	1e-4
sg	1
hs	0
negative	5
workers	6

表一、word2vec 參數表

- **Word Embedding + Seq2Seq**

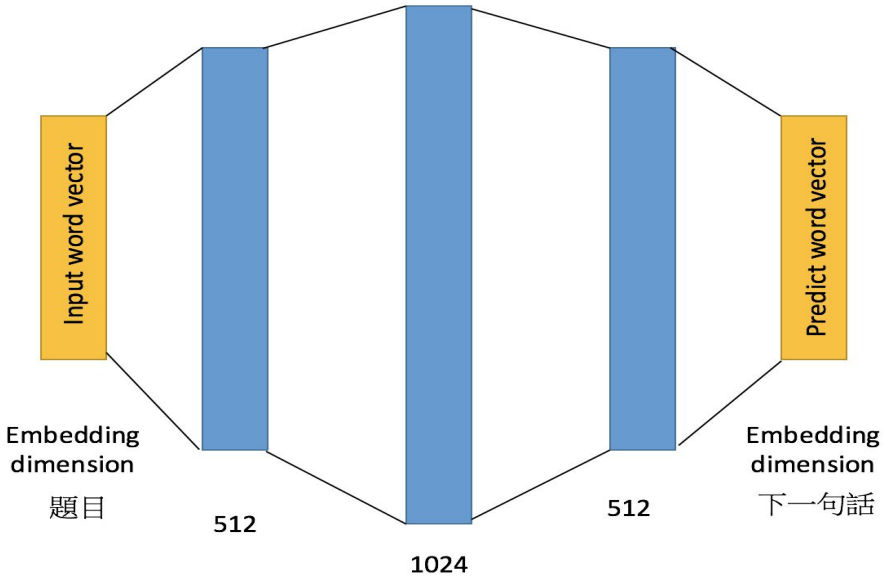
我們參考了網路上Seq2Seq的各種教學，由於先前在另一份Final Listen and translate中也有使用過Seq2Seq來做training所以我們選擇了其中一種最容易實現的架構如圖三：



圖三、seq2seq model 示意圖

- **Word Embedding + DNN**

會使用這種看似較特別做法的原因為，我們目前kaggle的最佳值是透過第一種model來產生的，而比對法則是採用詞向量去做比對，所以我們猜測依照常理我們原本的model就很好，如果我們在中間加入一層DNN並且以cosine similarity當作loss function的話應該可以得到相似的結果甚至較原本的更優。而我們使用的模型架構如下：



圖四、Word Embedding + DNN model

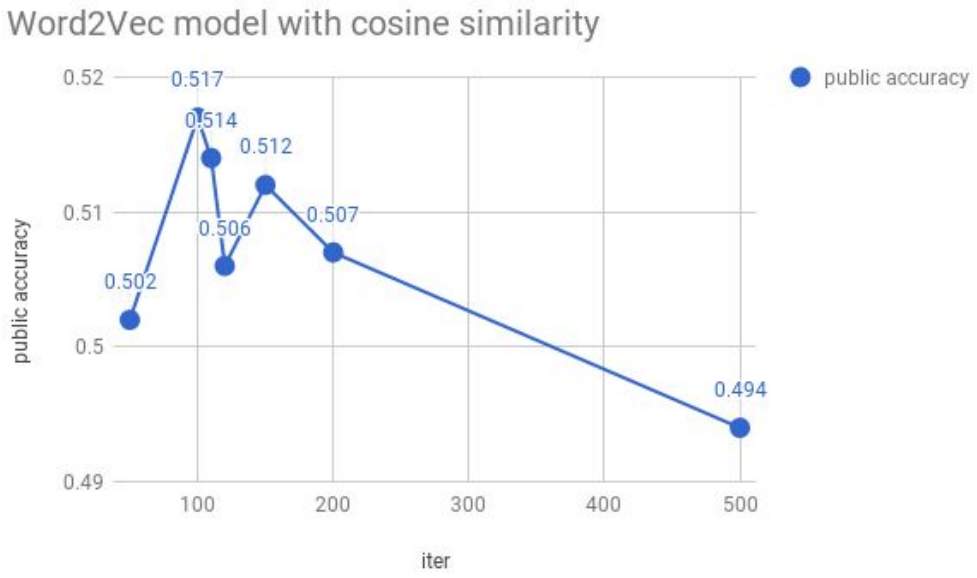
5. Experiments and Discussion

a. Word Embedding

在第一種model中，我們嘗試過許多種的方式進行測試，首先一開始是使用簡易的Word2Vec的gensim套件，並且調整其中的參數size(64,128,256,512)分別得到以下圖五的結果。起初在測試做實驗前我們根據網路上查到的資料做了簡單的推測：size若越大，則可以在整體效果上得到更佳的表現。但在實驗後發現embedding dimension若過大反而會

使分數降低。也因此我們做了簡單且合理的聯想：第一種為dataset不夠大，雖然總計出的句子約有75萬句且經jeiba 做切割後出來的單詞基本上也打破百萬的程度，但其中可能有許多單詞以重複出現過數次，所以整體的量可能還是不夠，也因此size調大反而對整個結果展現出更不好的效果。而另外一種聯想則是因為整個dimension大幅上升但使得word vector training過程變得更加困難，且我們iteration的次數並沒有因應地做調整所以才會得到較差的效果。

此外，我們也針對 word2vec的 iter 與accuracy 作圖，發現若iter過高，很有可能發生overfitting 的現象，但最根本的影響仍是initial seed。



圖五、accuracy-iter 圖

我們也在實驗過程1中發現了一件有趣的事情，若將word vector應用在Testing data上時，分別比較了兩種作法，**一種為將題目句子的單字逐個逐個與選項的單字逐個做比較相似性並且累加**，將數值最大的是為正解，而會有這種想法的原因為，經由觀測題目我們發現很有趣的事情為，由於規定所以答案的內容會有很大一部分是由劇本內部的話句所組成，但是非答案的選項則幾乎都是和劇本完全不相關的句字。例如：

Question

A:婚事不能再拖下去了

Answer

B:噓 寶寶熟睡了

B:先打個草稿吧

B:什麼時候大家又陷入如此著迷的狀態了

B:對不起 是我自作主張的

B:我也想趕緊辦婚事

B:環境的步調太快 已經跟不上了

由於gensim在做word to vector時不再劇本內的字是會被無視(跳過不做處理)所以起初我們猜測 當選項內和題目相似類型的字數越多則該選項為答案的機率越高，故我們才會使用逐字逐字比對法。

另外一種作法為將題目的**每個單詞的向量累加起來並取平均，將其視為「句向量」，對選項也用相同的方法並且比較最終的cosine similarity**，越大的選項就是最終的答案，這兩種方法雖然都是使用相同的word vector但是卻得到的完全相異的結果，第一種在kaggle上得到的F1-score為 0.394但是若是使用第二種的詞向量來做預測的話卻可以得到0.517，關於這部分我們其中一種推測是一句話中仍然有許多會誤導我們架構預測的雜訊詞句，若是使用詞向量時可以抑制這種影響，所以可以讓我們的預測更加的準確，反之若是單字逐步比對得話，每一個單字都會影響我們最終預測的結果，如果今天有一個錯誤選項中剛好被放入很多跟題目相同的字時，就會導致整個預測失準。句向量的範例如下：

Question

A:妳想用這種方式還我三千萬 別作夢了

B:三千萬耶 就算妳一輩子在這裡做苦工 都賺不到這筆錢

Answer

C:一輩子要怎麼賺到一億 0

C:晚餐要吃什麼 1

C:糟糕 來不及倒垃圾了 2

C:我自有辦法 不需要你們操心 3

C:廚餘請丟在廚餘桶 4

C:洗碗很好玩 我最喜歡洗碗 5

在這題中可以很清楚的知道答案應為3但是在第一種做法中預測出的答案為0，所以可以看出他對於整體誤導性的嚴重程度。

- **Word Embedding + Seq2Seq**

在實驗二中我們使用的Seq2Seq架構是直接由Listen and Translate仿製過來：

```
model = Sequential()
model.add(Seq2Seq(input_length = 10 ,input_dim  = Embedding_dimension ,hidden_dim = 50, output_length = word_len, output_dim = Embedding_dimension, depth = 3))
model.summary()
model.compile(optimizer = adam, loss = cos_sim)
model.fit(sound, vec, batch_size = batch_size, epochs = epochs, validation_split = 0.1,callbacks = [checkpoint])
```

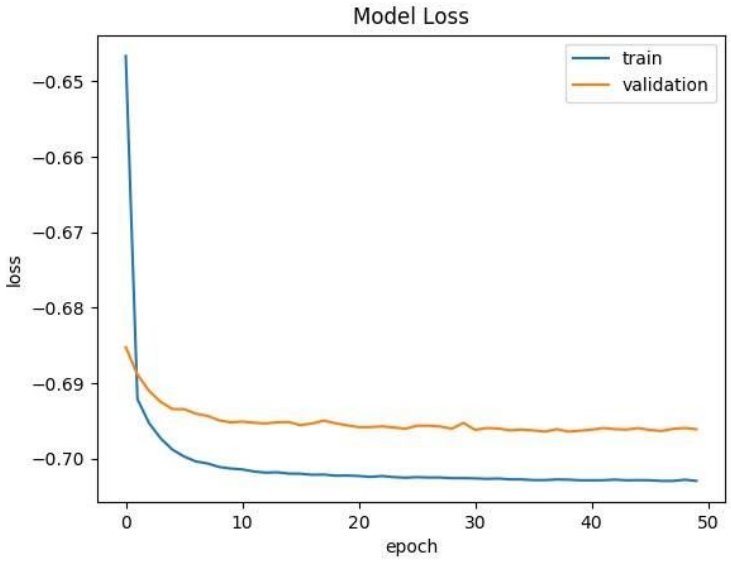
圖六、Seq2Seq code

在實驗中雖然我們的loss逐步下降，但把train好的model使用到testing上時得到的效果卻非常的差，此結果在Listen and Translate中也有出現相同的問題，雖然不確定是否為程式上的錯誤但目前有幾個簡單的推測，第一個為LSTM在資訊傳遞時總共會有4個gate，但我們當初在程式撰寫時並沒有對其中的參數做調整所以導致結果不佳，第二個為因LSTM中每個Block的參數是共用的，但是因為句子長短不一導致其中有很多我們會以補0來填滿整個input的time step但這樣的行為可能會使在中間某一

段backpropagation時無法訓練到，甚至導致梯度消失也因此整個model等於沒有做到完成的訓練。

- **Word Embedding + DNN**

  鑑於使用Seq2Seq成果不佳，故改成以詞向量為出發點做training，最終第三個實驗裡我們在訓練過程中也有loss逐步下降的趨勢(此處的loss為cosine proximity)。然由於時間有限，只針對1_train.txt的資料做訊練，因此testing的結果為 0.39。



圖七、Word Embedding + DNN model training curve