

# Conversations in TV shows

隊名：隊名還在想 2

隊長：電機四 陳咸嘉 B03901149

組員一：電機四 孫裕修 B03901085

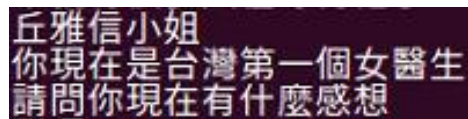
組員二：電機四 王允成 B03901087

## 一、分工表


姓名	分工內容
陳咸嘉	將資料做 preprocessing，並且利用 gensim 套件的 word2vec 和 GloVe 套件來實作詞向量模型。整合 Report 和程式碼。
孫裕修	將資料做 preprocessing，並且利用 gensim 套件的 word2vec 以及 RNN 做詞向量模型，並用 ensemble 提升模型準確率。
王允成	將資料做 preprocessing，使用 gensim 套件的 word2vec 和 tf-idf、auto-encoder 來實作詞向量模型。

## 二、Preprocessing and Feature Engineering

Training data 的形式為一行一句。第一步先透過 jieba 斷詞後，再去除 stop words，避免重複太多次的字影響詞向量的訓練(如：你、我、的)。第二步為將上下一句的句子串接在一起，使每一行皆由 3 句語句構成，各個詞皆用空格隔開。(如下所示，上圖為原始 training\_data，下圖為經過預處理的結果)



丘雅信小姐  
你現在是台灣第一個女醫生  
請問你現在有什麼感想



丘雅信 小姐 現在 台灣 第一個 女醫生 請問 現在 感想

Testing data 的形式為 id 加上問題與選項，首先先用逗號隔開 id、問題、選項，再由空格切出 6 個候選答案，並將問題與答案以 list 的方式存起來。再由 jieba 斷詞、最後去除 stop words。

### 三、Model Description

#### (一) Word2Vec Model (以 cosine 相似度為基礎)

我們在 kaggle 上有最好表現的模型是 word2vec。在 training 的模型中，我們用到以下文件：

1. provideData/training\_data/1~5\_train.txt
2. stopwords.txt (見來源[1])
3. dict.txt.big (見來源[2])

先將 training data 做上述的預處理之後，使用 word2vec 的模型，設定參數為：size = 100, min\_count = 5, window = 7, sg = 1。其中 size 代表詞向量維度，min\_count 代表該詞要在文本中出現一定次數才會被訓練，window 代表前後參考的詞的個數，sg = 1 代表使用 skip-gram，0 則使用 cbow。

此外，有無使用 stop words 與串接語句數皆為我們此次 project 的參數之一。詳細內容會在實驗與討論(一)做比較，主要以 cosine similarity 為基礎。

#### (二) Word2Vec Model (以 word vectors 平均為基礎)

加總所有的 word vectors 來組成 sentence vector。一種產生 sentence vector 最直觀的方式就是直接加總句子中所有的 word vectors。

$$sentence\_vector(S) = \sum_{w \in S} word\_vector(w)$$

這樣的方法就可以將所有 word 的概念都包含在了一個 vector 之中。我們使用 cosine similarity 來作為判斷兩句子相似程度的 metric。

cosine similarity 代表的是兩個 vector 之間夾角的大小，夾角越小代表相似程度越大，然而在這樣的一個標準下，和 Euclidean distance 不同的是，vector 本身的長度並不重要，因此使用了這個評斷方式也很有效的幫助我們稍微的處理了 OOV 的問題，因為在有 OOV 的問題時，我們不好判斷該怎麼處理句子的長度，因此 sentence vector 的長度是不具有任何物理意義的，而 cosine similarity 可以幫助我們避開這個問題。

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

### (三) TF-IDF Model

以 tf-idf score 來做為權重，以 weighted sum 的方式以 word vectors 組成 sentence vector。

然而在一個句子裡面，每個 word 的重要性很明顯地會不同，例如：“我很快樂”，這個句子的重點會是在傳達“快樂”這個概念，因此“我”跟“很”明顯的相對不重要，因此以單純直接相加的結果也許對句子的 concept 會產生偏差。

而 tf-idf score 是用來判斷一個 word 在整個 corpus 之中的重要性的一個指標，這個 word 越常出現，且出現在越少的 document 之中，代表這個 word 在整個 corpus 之中是更加獨特而且重要的，tf-idf score 分成了兩個部分，第一個部分 tf score 代表的是在這個 document 中，某個詞語出現的頻率，而 idf score 代表了在所有的 document 中，這個 word 總共出現在了幾個 document 之中，最終的 tf-idf score = tf-score \* idf-score (idf-score 是一個反向的分數)。

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad idf_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|} \quad tfidf_{i,j} = tf_{i,j} \times idf_i$$

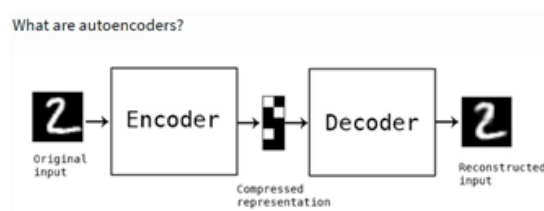
我將 idf-score train 在了所有的 provide data 之上，而 tf-score 則是計算在了當句題目或是解答的句子，這樣的作法其實不盡理想，而結果也確實十分差勁，會在之後的單元討論到。

### (四) RNN Model I

以 RNN auto-encoder 來產生 sentence vector

以上的幾種方法都沒有考慮到句子中詞語的順序性，例如：“我愛台妹”跟“台妹愛我”包含了完全相同的 word，因此在前面兩個方法之中，會產生出完全相同的 sentence vector，然而很明顯的兩句想表達的概念卻是大相逕庭，因此一個 RNN auto-encoder 可以幫助我們解決這個問題。

auto-encoder 是一種 unsupervised learning，可以透過 encoder 將原本高維度的 raw data 轉化為 context vector，而在透過 decoder 還原為原來高維度的 raw data 並使 distortion 越少越好，而 RNN auto-encoder 尤其適合用於 NLP 方面的加密，因為詞語的順序對於句子十分重要。



在 train RNN 的 auto-encoder 時，我們會把句子中所有的 word vector 疊起來，並且 padding 到 50 個字，而 OOV 則直接忽略，使用 auto-encoder 其實還有一個好處是能夠 assign 一個<unknown>的 tag 給他，讓他能夠自己處理 OOV 的詞語，然而這部分因為我們覺得我們 OOV 的詞語的詞義相差太大，可能並不適合使用這樣的方法。

我們使用了 200000 句的句子當作 training data，input shape = (50, 400)產生出 256 維的 sentence vector。

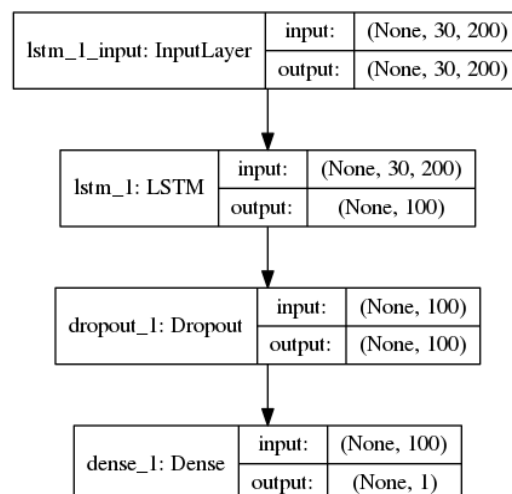
## (五) RNN Model II

我們另外提出了一種以 RNN 為基礎的模型。RNN 模型的優點在於它的模型設計中包含順序，因此在分析句子的成分時，理論上是非常適合的。與第四個模型中的模型不同的是，在這裡我們利用 supervised learning 嘗試達成相同的任務。

因為測試資料的問題中都是一個六選一的選擇題，我們要從中挑出一個正確的答案，使上下句構成的句子成為一個通順的句子，在這裡如果我們可以讓 RNN 模型學出"通順"與"不通順"句子的區別的話，那經由比較由六個選項所對應到的六個句子的"通順度"，那就可以選出答案。

既然要讓 RNN 模型學出"通順"與"不通順"概念，我們將原本被斷句斷好的 corpus，將上下句接起來，這樣就可以產生出很多組"通順的句子"。相反的我們就在原本的 corpus 中隨機選取兩句接在一起就可以產生"不通順的句子"

在訓練時，我們一樣利用 gensim 的 word2vec，將一個句子用一組向量表示，並將通順句給予標籤 1 反之則為 0。我們所建構出的 RNN 模型如下圖表示：



## 四、Experiment and Discussion

### (一) Word2Vec Model (以 cosine 相似度為基礎)

我們利用了 word2vec 模型，調整不同參數做了比較，大部分的預測方式如下：假設題目中有 M 個詞，答案中有 N 個詞，分別從題目和答案中取出一個詞計算 cosine similarity，總共 M\*N 個組合，忽略包含 OOV 的配對。

表格參數說明：D 代表維度(size)，M 代表 min\_count，W 代表 window，sg 代表 skip-gram，N 代表前後參考語句數，S 代表使用 stopwords(1 代表有使用)，Other 註明其他 testing 判斷方式。

D	M	W	sg	N	S	Other	kaggle
100	5	10	0	1	0	對每個 pair 的相似度加總平均	0.40474
200	5	10	0	1	0	對每個 pair 的相似度加總平均	0.42055
250	5	10	0	1	0	對每個 pair 的相似度加總平均	0.41264
400	5	10	0	1	0	對每個 pair 的相似度加總平均	0.41857
400	5	10	0	1	1	對每個 pair 的相似度加總平均	0.31146
400	5	10	0	1	1	對每個 pair 的相似度絕對值 加總平均	0.35494
400	5	10	0	2	0	對每個 pair 的相似度加總平均	0.34071
200	5	10	0	1	0	對每個 pair 的相似度加總平均 不考慮 OOV 的次數	0.37786
200	5	10	0	1	0	對每個 pair 的相似度加總平均 若遇到相同字，則給相似度 0.8	0.42687
200	1	10	0	1	0	對每個 pair 的相似度加總平均 若遇到相同字，則給相似度 0.8	0.39802
100	5	10	0	1	0	對每個 pair 的相似度加總平均 Jieba cut all 設為 true	0.37826
						將上述模型做 ensemble	0.40039

100	1	10	0	3	1	使用 glove 套件 對每個 pair 的相似度加總平均	0.42648
200	1	7	0	5	1	對每個 pair 的相似度加總平均 其中相似度大於 0.7 的乘以 2	0.43320
200	1	7	0	5	1	同上，但透過不同的相似度比較(不一定是 0.7)來一層層篩選，依次篩選掉 1/2/2 個。	0.43557
100	5	7	1	3	1	對每個 pair 的相似度加總平均	0.47549
100	5	7	1	3	1	對每個 pair 的相似度加總平均 只加總相似度大於 0.2 的	0.47865

由上表可知，維度和 stop words 有時候對於準確率並沒有太大的效果，反而是更改來了訓練模式(由 cbow 改成 **skip-gram**)，使準確率大幅提升約 4%。去了結了一下架構，skip-gram 是 input 一個字，預測上下文；cbow 是 input 上下文，預測一個字。由直觀來看，skip-gram 應該會比 cbow 來準確，因為我們注重的是答案和題目間的關聯性，由一個字去和另一個字做相似的程度比較。

## (二) 比較 Model 主題提到的(二)、(三)、(四)、(五)模型：

我們利用了三種不同產生 sentence vector 的方式來 predict 我們的結果。

Method	Sum Weighted	Sum with Tf-Idf score	RNN auto-encoder	RNN II
Kaggle Public Accuracy	0.44466	0.28893	0.32648	0.13833

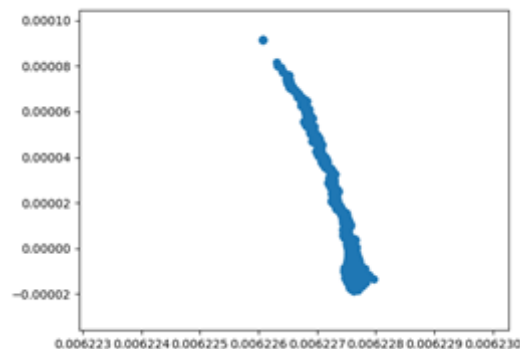
出乎意料的，竟然是最直觀的方法結果最好，以下討論一下 tf-idf 跟 RNN auto-encoder 失敗的原因。

### Why tf-idf fails?

tf-idf 其實很在意的是 corpus 的大小，corpus 夠大才有辦法展現出 tf-idf 的統計特性，雖然訓練 idf 時的 corpus 夠大，但是在計算 tf-score 時只用了單一句子去算明顯有問題，應該將同一個劇本裡的所有台詞當作 corpus 來計算 tf-score，而且加入了 tf-idf vectorizer 又大大增加了 OOV 的問題。

## Why RNN auto-encoder I fails?

下面這張圖是所有 training data embedded 到 256 維之後再降維到 2 維平面上的圖，可以看到，sentence vector 之間的 cosine similarity 確實會不一樣，然而實際上差距卻是很小，可以推斷我們 input 的 raw data 之間可能本身就太像了，因為我們將字數 padding 到了 50 字，然而有很大一部分的句子只有 3~4 個字，因此造成了這些 raw data embedded 之後十分接近，然而又很難縮短 padding 的長度，因為仍然有許多長度較長的句子，因此像對話這種有長度不一的句子，可能較不適合使用 RNN 訓練。



## why RNN II fails?

利用 RNN 模型判斷句子通順程度的方法，在實驗上所達到的準確率非常低。在訓練上 RNN 模型在判斷通順句子的準確率以及判斷不通順句子的準確率大約為 80%，但是因為我們所要面對的問題是在六個句子中判斷出正確的句子是哪個，因此在判斷句子正確度的準確率只有 80% 的情況下，整體的正確率只會有大約 20%，又因為測試資料的不正確答案並不是一組從 corpus 隨機抓出來的句子，因此正確率會下降的更低。如果要用這種方法達成超過 strong baseline，在判斷句子通不通順的準確率必須要超過 90%，在運算資源以及訓練資料不足的狀況下，用這種方法是不切實際的。

## 五、參考資料：

[1] stop words

<https://github.com/tomlinNTUB/Machine-Learning/blob/master/data/%E5%81%9C%E7%94%A8%E8%A9%9E-%E7%B9%81%E9%AB%94%E4%B8%AD%E6%96%87.txt>

[2] jieba dictionary

[https://github.com/fxsjy/jieba/blob/master/extra\\_dict/dict.txt.big](https://github.com/fxsjy/jieba/blob/master/extra_dict/dict.txt.big)

[3] gensim tutorial

<https://radimrehurek.com/gensim/models/word2vec.html>

[4] skip-gram & cbow

<http://zongsoftwarenote.blogspot.tw/2017/04/word2vec-model-introduction-skip-gram.html>