2017FALL Machine Learning
Final Report

- Report Overview:
  - Task description
  - Team name, members and work division
  - Data Preprocessing and Feature Engineering
  - Model Implementation
  - Experiments, Results and Discussion
  - References

- Task description:
  專題題目：Conversation in TV shows
  題目簡介：給定一個對話，在一組答案中選出最佳回應

- Team name, members and work division
  隊伍名稱：NTU_b04901025_梁書嘖嘖稱奇
  組員：b04901019 梁書哲、b04901138 張景程
  b04901025 陳鴻智、b04901118 王克安
  隊伍分工：
  b04901019 梁書哲：
  Training Data 處理(讀檔、斷詞、繁轉簡、串接)、w2v 模型訓練。
  b04901025 陳鴻智：
  Training Data 處理(讀檔、斷詞、繁轉簡、串接)、sentence reweight 演算法實作、ensemble。
  b04901118 王克安：
  w2v 模型訓練、addition experiments implementation、撰寫 report。
  b04901138 張景程：
  Training Data 處理(讀檔、斷詞、繁轉簡、串接)、model 架構設計、predict cosine similarity、siamese network、w2v 模型訓練。

- Data preprocessing and Feature Engineering:
  - jieba 斷詞
    首先要先將 Training Data 中的句子斷成個別的字或詞，才能使用 Word2Vec 轉換成詞向量。在做英文的斷詞中通常是用空白來做斷詞，但是中文卻不能這樣去做斷詞，因此我們使用了 jieba 的中文斷字詞典。
    由於 jieba 是由中國開發的套件，其繁體字典並沒有如簡體字典來的完整，我們另外使用了 OpenCC 將所有 Training Data 轉換

為簡體字，並且使用 jieba 預設的簡體字典進行斷詞。

(Jieba 的詞典有三個模式：精確模式、全模式、搜尋引擎模式，經過實驗我們最後選擇使用精確模式。)

- genism Word2Vec

  從官方文件中可以看到有許多可調的參數，其中我們對於 size、window、min_count、sg、iter 有做調整，各參數意義如下

  Size: Embedding Dimension

```
class gensim.models.word2vec.Word2Vec(sentences=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=<built-in function hash>, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False) ¶
```

  Window: maximum distance between the current and predicted word within a sentence.

  Min_count: ignore all words with total frequency lower than this

  Sg: skip-gram is employed.

  Iter: number of iterations (epochs) over the corpus.

  經過實驗後發現，以下參數有較好的表現：

參數名稱	數值
size	64~128
window	7
min_count	1
sg	True
iter	10~30

  此外，我們參考 presentation 組別的做法，將 jieba 切詞完後的句子把每三句 data append 在一起送進 gensim 的 word2vec 中將字詞轉換成為一個 n 維度的向量。

- Tokenizer(for Siamese)

  我們首先嘗試了 Keras 內建的 Tokenizer，但是發現套用在中文在斷詞上會發生問題，因此我們改用 Python Standard Library 的 Dictionary 資料結構自行編寫一個 Tokenizer，從 training data 的第一句開始，利用 jieba 做斷詞，再依序檢查每一個詞是否已經存在於 dictionary 中，若沒有的話，則將該字詞當作 key 新增至 dictionary 中，並將其 value 設為 dictionary 的 size，藉由此方法能將一句話轉換為一個 Scalar vector，再放入後續訓練的模型架構。

- Model Implementation
  模型架構的部分我們採取了兩種不同的做法，以下針對兩種做法做詳細的說明:
  - Sentence Similarity
    在這個模型中我們採取的手法很簡單，即將所有單字經過我們 pre-train 好的 word-to-vector 字典轉換成一個 vector，並將一句話裡所含有的單字 vector 做平均，來代表一個句子的 embedding vector。再將題目與所有的選項去計算 cosine similarity，最後的解答即是相似度最高的選項。

  - Siamese Manhattan LSTM
    Siamese Manhattan LSTM 是我們在尋找參考資料的時候找到的，是一個適合用來決定句子間的相似度的一個網路。這個網路可以分成三個部分來做說明：
    - Siamese Network:
      Siamese 網路指的是包含兩個或以上相同子網路的架構，其中兩個子網路的架構、參數都是相同的。因為子網路中的參數是共享的，這樣就減少了訓練所需要的 data 量以及 overfitting 的可能性。這種類型的網路經常被用來比較兩種輸入間的關聯或相似性。

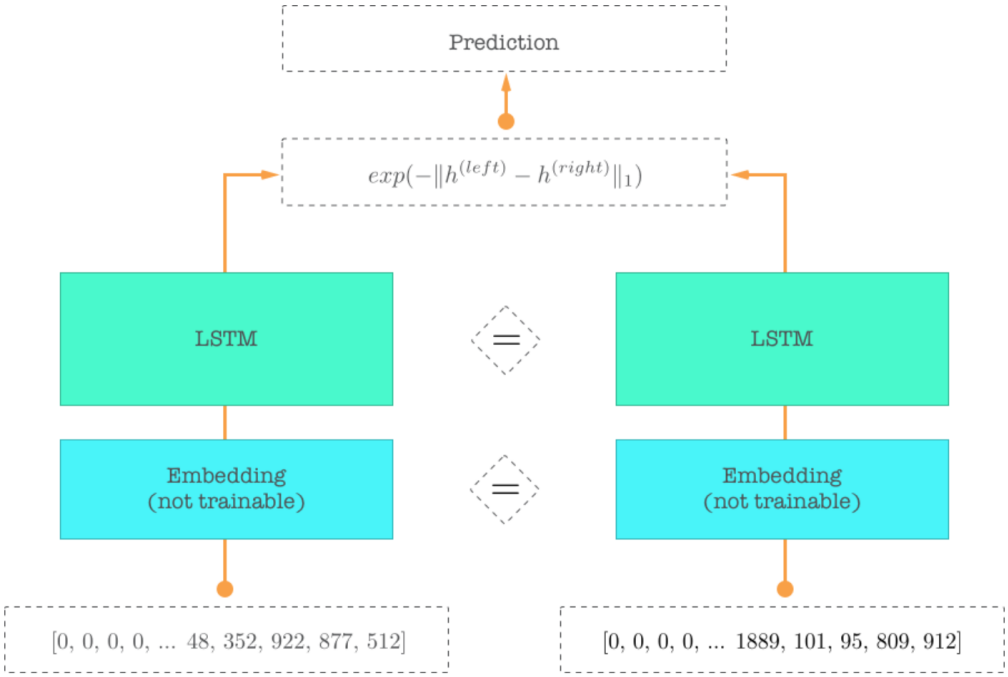
    - Preprocess:
      呈上述的資料架構，在這個 network 當中，我們對先前處理好的 data 做更進一步的處理：首先將 tokenize 完的句子 pad 成 max_length=15，並去掉原本長度 < 3 的句子來做訓練。由於這個 model 需要同時將兩個句子丟進去 train，也就是 question 和 answer 的 pair，因此我們將原對話內容每一句的"下一句話"來當作該句話的 answer。
      我們將這樣的 training pair 的 label 標記為 1，同時對每個 question 隨機從 data set 當中 assigned 一句話當做 answer，label 標記為 0。

    - Manhattan distance:
      Manhattan 在這邊指的是網路的 similarity function 是取決於兩個句子之間的 Manhattan distance，而並非常見的 cosine distance。其公式如下：

      $$exp(-\|h^{(left)} - h^{(right)}\|_1)$$

      MaLSTM similarity function

模型的部分我們是參考這個模型架構來做設計：



上圖中藍色的 Embedding layer 是我們用 w2v pre-train 好的，而要 train 的是綠色 LSTM 的部分。

下圖是我們實作的模型架構：

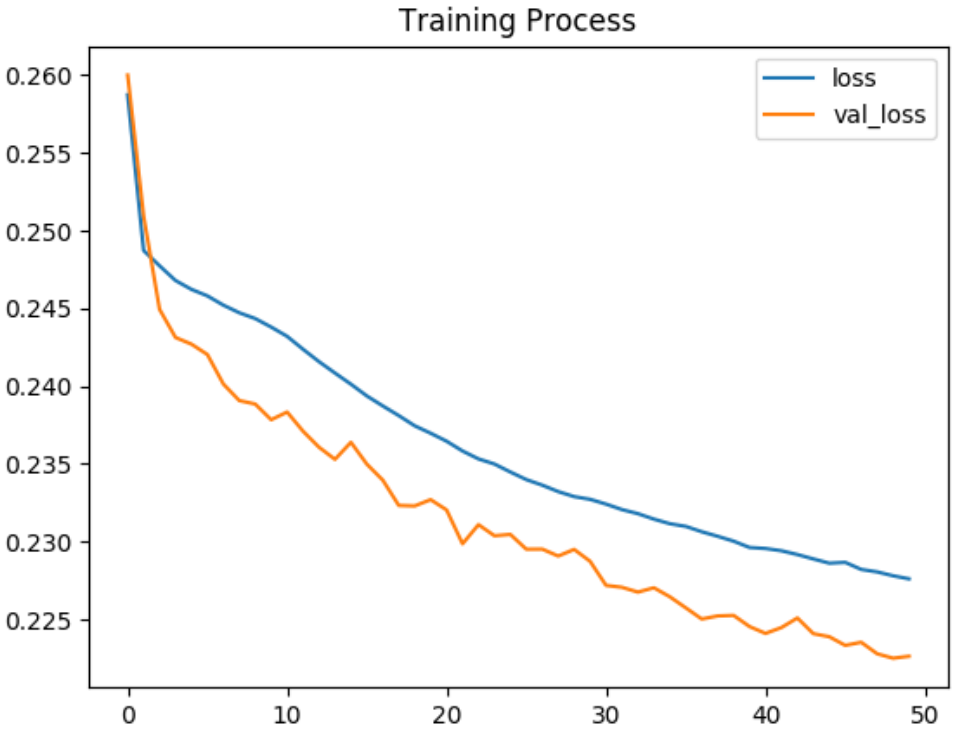
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 50)	0	
input_2 (InputLayer)	(None, 50)	0	
embedding_1 (Embedding)	(None, 50, 512)	10240000	input_1[0][0] input_2[0][0]
lstm_1 (LSTM)	(None, 50, 256)	787456	embedding_1[0][0]
lstm_2 (LSTM)	(None, 50, 256)	787456	embedding_1[1][0]
lstm_3 (LSTM)	(None, 128)	197120	lstm_1[0][0] lstm_2[0][0]
merge_1 (Merge)	(None, 1)	0	lstm_3[0][0] lstm_3[1][0]

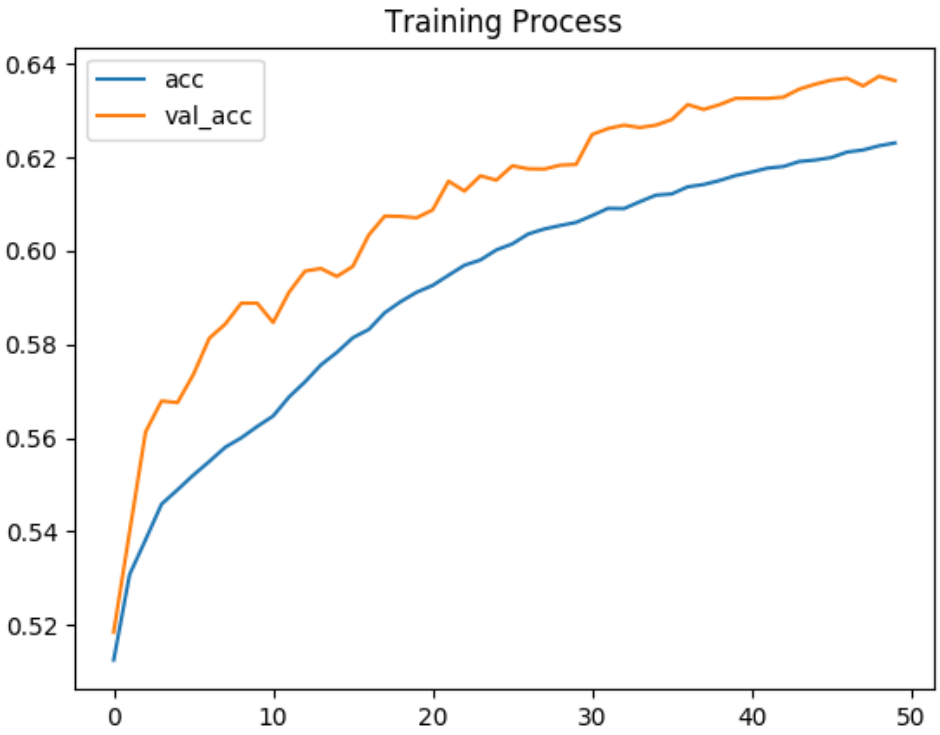
Total params: 12,012,032
Trainable params: 12,012,032
Non-trainable params: 0

其中 embedding layer 的部分是用我們自己 pre-train 的 w2v embedding matrix。

Optimizer : Adadelta(clipnorm=1.25)
loss function = mean_square_error
epochs = 50

下圖是我們的 training 過程：





- Experiments, Results and discussion:
  - Sentence Similarity

    前幾次的實驗準確率都落在 0.36~0.38 左右，若將 Word2vec 從 CBOW 改為 skip_gram，可以大幅提升至 0.45，ensemble 完大約為 0.50，但是若將 jieba 的切字詞典換成簡體中文版本的話，在同樣的參數設定下 ensemble 的準確率可以提升到 0.51，於是我們往後的實驗都是 based on 簡體中文的 jieba 切字詞典來進行。

    首先在 train word embedding 的步驟有兩個最重要的參數可以進行調整，分別是 Embedding dimension 和 window size。

    Embedding dimension=m 是指說以一個 m 維度的向量來代表這個字，而 window size=n 指的是說在 embed 某個字的當下還需考慮這個字前後 n 個字，有點類似 n-gram 的概念。我們主要是調整這兩個參數來 train 我們的 word embedding。

    下表是我們嘗試過的幾組不同 word embedding 的組合：

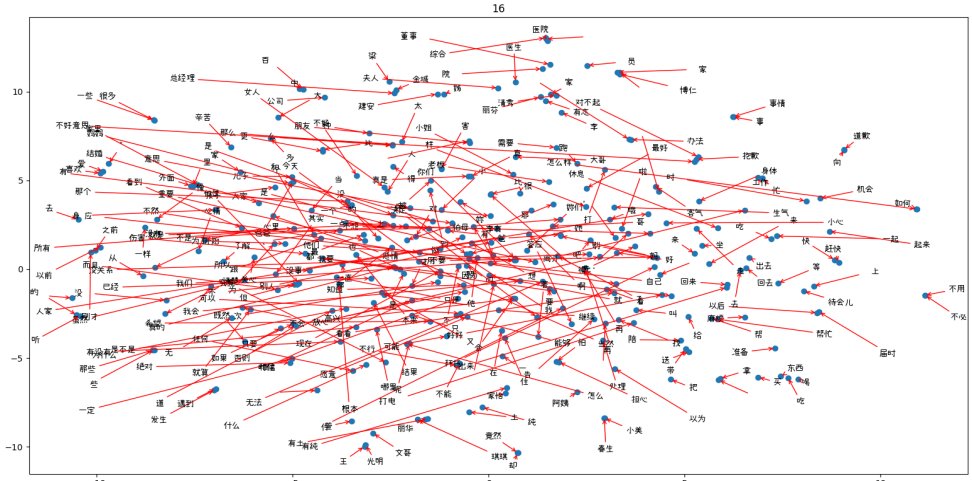
Exp NO.	Embedding dimension	Context window size	Accuracy
1	32, 64, 128（繁體）	3, 5, 7	0.50869
2	32, 64, 128, 256	3, 5, 7	0.51304
3	32, 64, 128	3, 5, 7	0.51146
4	64, 128	3, 5, 7	0.51343
5	64, 128	5, 7	0.51225

    從實驗 4,5 中可以觀察到少了 dim32 的 embedding 後準確率上升了不少，推測是在這個 dataset 之下 32 個維度無法完整包含字與字之間的關係，導致準確率下降。

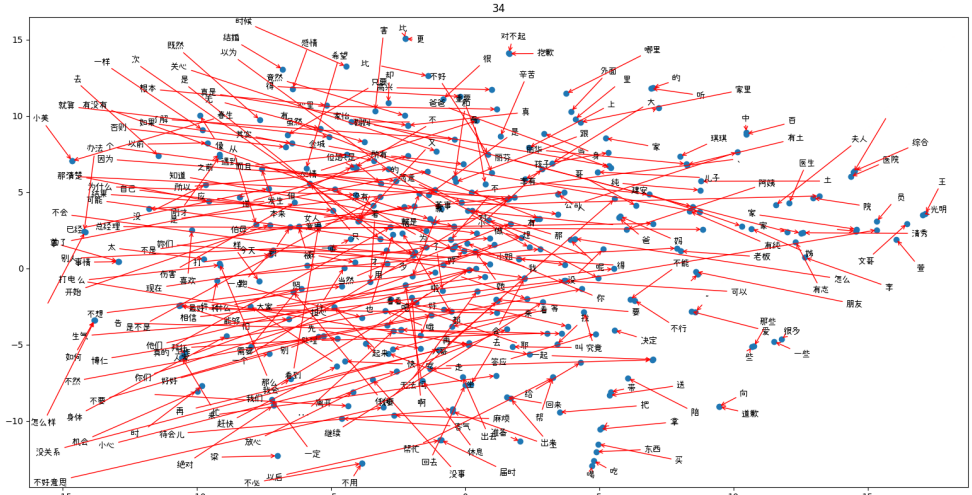
    下面幾張圖是我們針對準確率最高的幾組 model 來做圖，作法跟 hw6 一樣是先將 w2v 中的字詞以 TSNE 降到兩維之後再作圖：

    以下三招圖分別是
    dim = 128, win = 7
    dim = 100, win = 7
    dim = 64 , win = 7

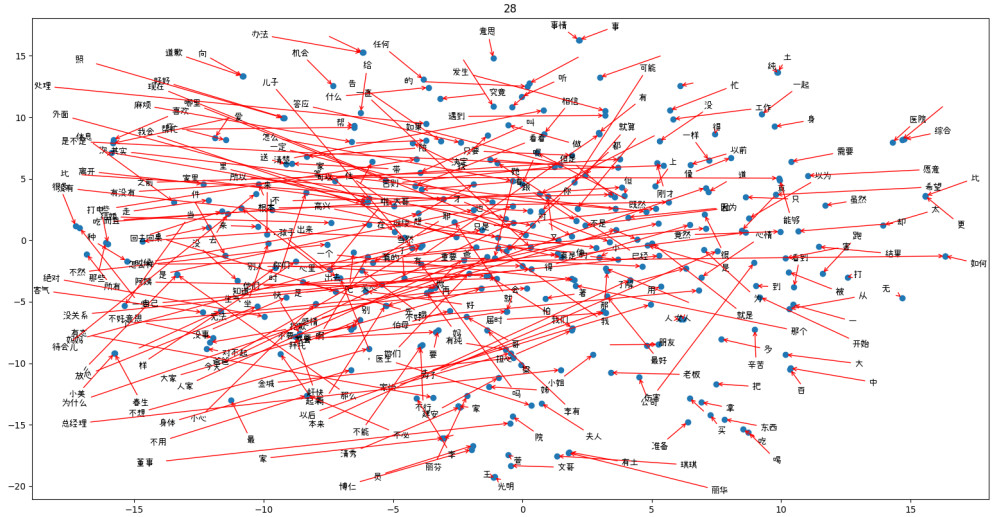
16



34



28



接下來我們針對考慮 stopwords 跟考慮 OOV(out of vocabulary)的狀況來進行實驗。

■ stopword

在做英文的 word embedding 時會過濾掉一些常用但是意義不大　的字，例如 you, are, we, am, is 等等，這些叫做 stopwords。這些字在句子中常常佔有很大的份量卻包含很少的資訊，所以通常會把這些字拿掉。我們在 github 上找到一份中文 stopword 列表，於是將 training data 跟 testing data 中含有 stopwords 的字詞過濾掉並進行實驗，以下是實驗結果：

	filter stopwords	normal
version 1	0.4484	0.4907
version 2	0.4497	0.4854
version 3	0.4417	0.4801

從實驗結果發現準確率下降了，與一般過濾掉 stopword 準確率會上升的情況相反。我們猜測原因可能是因為 training data 和 testing data 中包含很多在 stopword list 中的字。實驗時我們發現會有許多 question 跟 answer 再過濾掉 stopword 後都是空白的情形，導致結果準確率下降，與我們的猜測相同。分析我們的狀況與一般狀況的差別，應該是因為我們的 dataset 是由同學所提供的，所以裡面的字詞都比較口語化，也就會有比較多的冗言贅字，所以有些問題跟答案相較於 real world 的 dataset 是比較沒有意義的，而 real world 的 dataset 可能包含比較豐富的資訊，而且包含特定 domian 的關鍵字，因此在過濾掉 stopwords 之後準確率才會上升。

■ OOV

Out-Of-Vocabulary 的狀況指的是說在 testing 的階段 model 遇 到在 training 時沒遇過的字詞，也就是說把這個字拿給 w2v 進行 query 時會出現 word not in dictionary 的問題。我們這邊的做法是在 w2v 中新增一個 unk 的選項，當發生 OOV 時就將該字詞設為 unk，這樣在 query 時這個 OOV 就會被 assign 成一個 unk token，並回傳一個 unk 的 word vector。我們的結果如下：

	handle OOV	normal
version 1	0.4060	0.4907
version 2	0.4219	0.4854
version 3	0.3875	0.4801

如何 handle OOV 的問題一直是 nlp 中一個很大的問題。在處理 OOV 時我們決定指定一個預先設好的向量來做這個 Unk 的值。從結果來看準確率仍是下降的。理論上在 unk 很少的狀況下任意 assign 一個向量給這個 unk 是不會影響實驗結果太多的，但在我們這個 dataset 之下 unk 經常發生，所以在任意指定的情況下會讓句意變的不夠清楚。我們有嚐試 implement facebook 最新處理 OOV 的 fastext，但是效果並沒有比較好。撇除技術上的問題，我們的結論是 dataset 中的資料跟真實世界的資料有不小的差距導致。

■ hybrid

集合上述兩種作法，我們進行了第三次實驗，其結果如下表：

	OOV + stopword	normal
version 1	0.3227	0.4907
version 2	0.3306	0.4854
version 3	0.3187	0.4801

準確率變低在我們的預料之中，畢竟上述兩種方法都會降低準確率。但是在我們 review 過的 paper 當中幾乎都有用這兩種方法。或許是我們 implement 的方法不對，但是我們仍覺得一部分的原因是 dataset 的問題。

- Siamese Manhattan LSTM：

在兩組 LSTM 的架構及參數都相同的情況下我們 train 出來的結果並不是很理想，準確率是：0.2466

經過分析我們覺得準確率低不是因為 overfit 的結果，於是決定讓兩組 LSTM 的參數不一樣。結果準確率雖有上升至 0.35573，但依然不是一個非常理想的成果。

	Accuracy
Shared weights	0.24466
Unshared weights	0.35573

討論其準確率低我們推測主要是因為 training set 和 testing set 分佈差異過大，除了產生許多 OOV 的問題外，training 和 testing 的句子長度也非常不同，造成 neural network 無法準確套用在 testing data 上。

- Sentence reweighting:

  原先從 word embedding 轉為 sentence embedding 時，只是單純的相加取平均。而根據 A Simple but Tough-to-Beat Baseline for Sentence Embeddings (Sanjeev Arora, Yingyu Liang, Tengyu Ma, 2017)這篇 paper 裡面的做法，使用 smooth inverse frequency (SIF) ，根據句子的長度和每個詞出現的頻率去重新計算每個詞的 weight。因為在 training data 裡面其實有意義的詞並不多，如果我們能夠找到比較關鍵的詞，給他更大的 weight，之後要從題目找關鍵詞的 similiarity 也會比較準確。演算法如下：

Algorithm 1 Sentence Embedding

Input: Word embeddings $\{v_w : w \in \mathcal{V}\}$, a set of sentences $\mathcal{S}$, parameter a and estimated probabilities $\{p(w) : w \in \mathcal{V}\}$ of the words.
Output: Sentence embeddings $\{v_s : s \in \mathcal{S}\}$

for all sentence s in $\mathcal{S}$ **do**
  $v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a+p(w)} v_w$
end for
Form a matrix X whose columns are $\{v_s : s \in \mathcal{S}\}$, and let u be its first singular vector
for all sentence s in $\mathcal{S}$ **do**
  $v_s \leftarrow v_s - uu^\top v_s$
end for

  a 是任意的參數，p(w)是 w 這個詞在 training data 中出現的機率。

  這個做法在我們的 case 中是比較行得通的作法。經過討論我們覺得這個方法會比 stopwords 的表現還要好的原因是在這個 dataset 當中若直接把 stopwords 拿掉會讓句子比較短的問題或答案資訊量不足，但是若用 sentence re-weighting 的話可以將那些字詞留下但給予比較低的 weighting 而給重要的字比較高的 weighting。這樣的做法在這次 project 中為效果最好的做法。

- Ensemble

  最後，我們將表現良好的 model 集合 ensemble。方法為各 model 預測最有可能的選項，以投票表決的方式決定每一題最終的答案，使用 ensemble 後達到 Kaggle Public Score: 0.53122 的成績。而在 ensemble 的過程中我們也發現並不是將單一 model 分數最高的 model ensemble 再一起的最後成績就會最高，而是要透過實驗來試出哪些 model 組合在一起會有最高的分數。

- Final Thoughts

  總結這次 final project 的實作，Training Data 與 Testing Data 的差異性過大，導致在 Training Data 上所做的訓練無法完全的表現在 Testing Data 上。不過使用 Sentence similarity 基本的演算法

搭配上細心的參數調整，還是能夠有不錯的表現，讓我們從最初通過 simple baseline 的 0.39 進步到最終的 0.53。唯一比較可惜的是不論是用 NN 或是 feature engineering 都沒辦法超越單純 sentence similarity 的分數，也讓我們體會到要利用真實世界的 data 來 train 一個夠 robust 的 NN，就算是一個相對簡單的 task，也是十分困難的。

以下是組員做完這次 final project 的心得：

王克安：

英文的 word embedding 中會去過濾掉一些常用但是沒有意義的字，例如 you, are, we, am, is 等等，這些叫做 stopwords。這些字在句子中常常佔有很大的份量卻包含很少的資訊，所以通常會把這些字拿掉。中文目前還沒有一個完整的 stopword 列表，所以結果做出來成效不會很好。

陳鴻智：

這次的 dataset 裡面，因為是演員的台詞，有太多無意義的句子。而且斷句的位子通常是語氣停頓的位子，而不是語意結束的位子，導致 training data 的長度和 testing 的問題長度對不起來，如果有去掉 stopword 的話會更慘。所以大家跑出來的結果都不算太高。

梁書哲：

在完成這份 project 的過程中嘗試了非常多種的 Model，沒有想到最終 sentence similarity 的做法達到了最高分。在實作的過程中也發現中文斷詞及字典的困難性，相較於英文有更多樣的變化，需要更加細心地去調整 Preprocessing 方法及參數，常常細小的調整就會有巨大的變化，讓我學習及實作的過程充滿驚奇。

張景程：

這次 final 本質上算是一個很棒的題目，讓機器從六個選項中選出一句最適合的作為回答，可以算是一個較簡單的 chatbot，只可惜這次給的 corpus 不是很理想，導致 NN 的 performance 沒有預期來得高。 在實作過程中我們體會到中文和英文的差異，從切詞開始就遇到了麻煩，到後來集中火力在 train w2v 的 model 也發現到每次訓練即使給定同樣的參數，準確率也會有所差別，多少含有些許運氣成份，細微的參數變化也可能會產生巨大的影響

- References:
https://github.com/fxsjy/jieba
https://github.com/yichen0831/opencc-python
https://github.com/eliorc/Medium/blob/master/MaLSTM.ipynb
https://medium.com/mlreview/implementing-malstm-on-kaggles-quora-question-pairs-competition-8b31b0b16a07