

# Machine Learning (2017, Fall)

## Final Project – TV Conversation

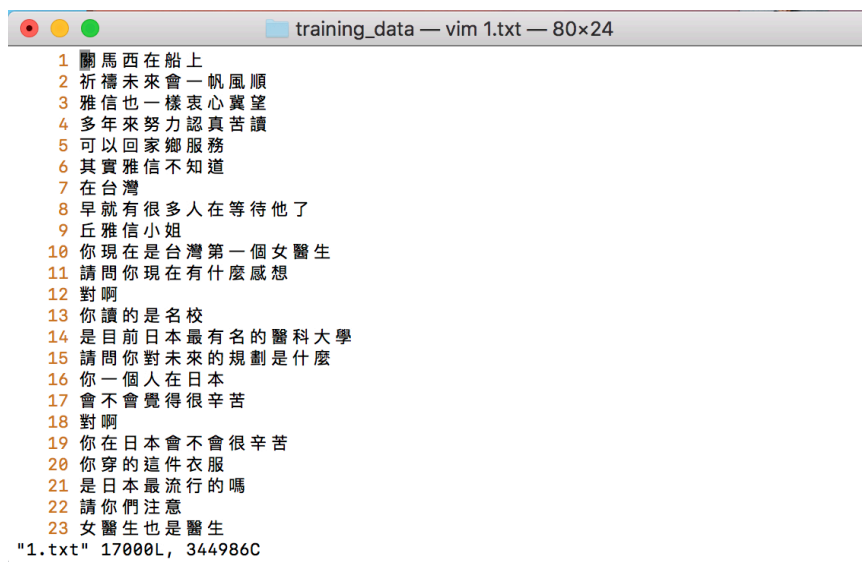
b03901086 電機四 楊正彥 b03901124 電機四 李昂軒

b03901130 電機四 林奕廷 b03901145 電機四 郭恆成

### 1 Task Description

給定數句由一個人或多個人所組成的中文對話，並從提供的選項中選擇出最有可能接在前面對話之後的語句，而訓練資料僅能使用提供的台灣電視劇台詞字幕，不可使用其他的資料或者語言模型。

### 2 Preprocessing/Feature Engineering



```
1 關馬西在船上
2 祈禱未來會一帆風順
3 雅信也一樣衷心冀望
4 多年來努力認真苦讀
5 可以回家鄉服務
6 其實雅信不知道
7 在台灣
8 早就有很多人在等待他了
9 丘雅信小姐
10 你現在是台灣第一個女醫生
11 請問你現在有什麼感想
12 對啊
13 你讀的是名校
14 是目前日本最有名的醫科大學
15 請問你對未來的規劃是什麼
16 你一個人在日本
17 會不會覺得很辛苦
18 對啊
19 你在日本會不會很辛苦
20 你穿的這件衣服
21 是日本最流行的嗎
22 請你們注意
23 女醫生也是醫生
"1.txt" 17000L, 344986C
```

Fig 1. Raw training data

訓練資料總共有 757000 行，而平均每行由 jieba.txt.big 斷詞之後每句會有 4.62 個字詞，但很明顯的這樣的訓練資料所訓練出來的 word2vec 效果是非常差的，為了改善這點，我們分別將訓練資料中的前後幾句話相連接，以達到變相增加訓練資料的效果。

|                   |       |       |       |       |       |       |       |
|-------------------|-------|-------|-------|-------|-------|-------|-------|
| 前後連接句數 $k$        | 0     | 1     | 2     | 3     | 4     | 5     | 6     |
| 平均長度 $l$          | 4.62  | 9.24  | 13.9  | 18.5  | 23.1  | 27.7  | 32.3  |
| word2Vec 訓練時間 $t$ | 84.72 | 178.6 | 253.8 | 312.5 | 357.4 | 440.3 | 520.0 |

Table 1. word2Vec training comparison

Preprocessing (平均單詞個數 - word embedding 訓練時間)

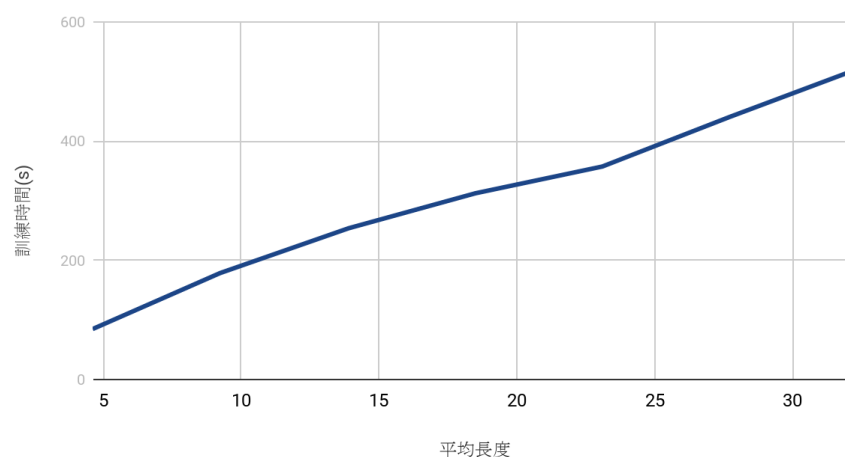


Fig 2. Average word per sentences to word embedding training time

從上圖可以看出每個句子所包含的單詞數量越多，訓練時間也越長，成長關係約為正比，不過我們比較在乎的是連接越長是否能得到越佳的 word2Vec。

Preprocessing (平均單詞個數 - accuracy)

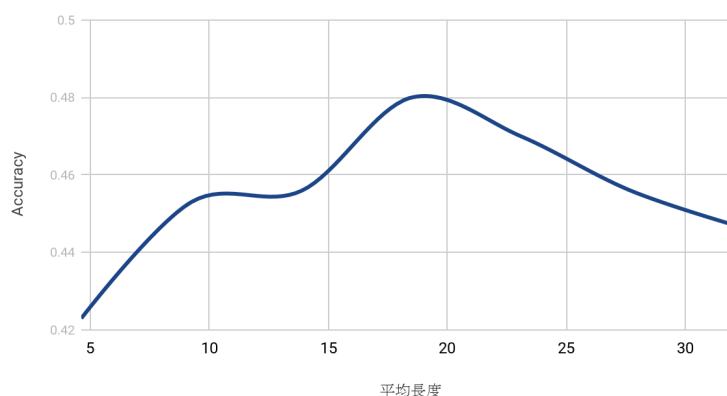


Fig 3. Average word per sentences to kaggle public score

上圖是在維度為 200，window 為 5，min\_count 為 2 的參數環境下去做 word2Vec，可以看出來，平均長度在 18 附近的時候，kaggle public score 會是峰值，可見說延長我們在訓練過程當中每個句子的長度，雖然可能會有大量重複的字句，但對於訓練 word embedding 的效果反而會越好。我們同時也觀察到，延長的句子以 3、4 句左右為佳，過長反而會發生退步的情況。

### 3 Model Architectures

#### 3.1 Paired-word Similarity

Paired-word similarity 是我們這組最早嘗試的方法，假設經過 preprocess 後的問題標為  $q$ ，其中每個字經過 word2Vec 標為  $q_n$ ， $n$  為題目的總長度；而選項則標為  $a^k$ ，其中每個字的 vector 標為  $a_m^k$ ， $m$  為該選項的總長度。而每個選項成為題目的下一句的 likelihood 則量化為：

$$sim_k = \sum \frac{\text{cosine\_similarity}(q_i, a_j^k)}{n \times m}$$

因為我們預期題目跟選項很容易提到 word embedding 相近的詞組，故比較題目與選項中各字詞的 cosine similarity 並且進行一個類似加權的手段，以期能到選出正確的選項。

#### 3.2 Siamese Manhattan LSTM

使用論文 Siamese Recurrent Architectures for Learning Sentence Similarity 中所提出的方法，其架構如下，主要概念為把問題與選項餵進同一個 LSTM layer，然後把 LSTM layer 輸出的結果經由自定義的 exponential similarity function 後的得到一個介於 0 與 1 的值。

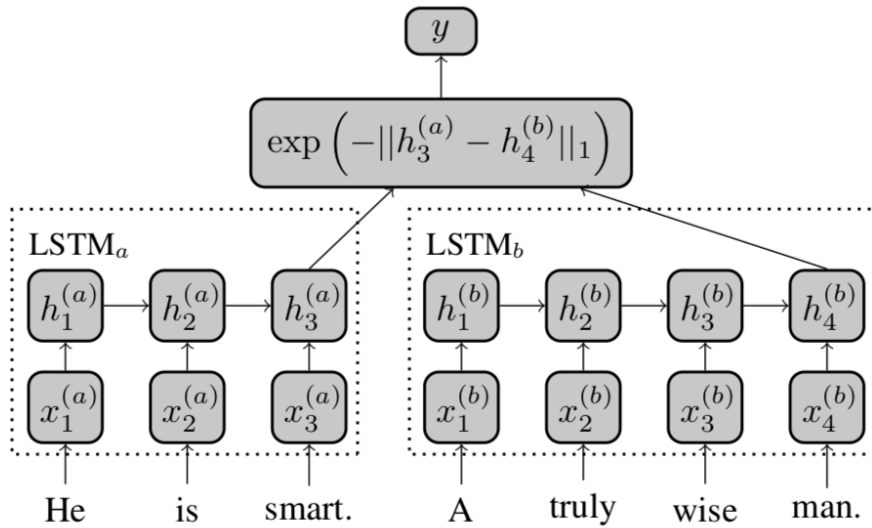


Fig 4. Model configuration of Siamese Manhattan LSTM

### 3.3 Modified Siamese Manhattan LSTM

根據論文 Siamese Recurrent Architectures for Learning Sentence Similarity 中所提出的模型，我們進行了一些架構上的變化以符合我們這次的 tasks。首先我們認為前後兩句話使用同一個 LSTM 是比較不合理的，因此我們題目以及答案所丟進的 LSTM layer 為兩個獨立的 layer，接著才在又餵進同一個 LSTM layer 中，其次我們所使用的 loss function 為參考論文所訂的 malstm\_distance 的 mse。而訓練相關參數為：optimizer=Adadelta(lr=0.001)、batch\_size=512、epoch=250。

```
# The visible layer
left_input = Input(shape=(max_length,), dtype='float32')
right_input = Input(shape=(max_length,), dtype='float32')

embedding_layer = Embedding(53597, 500, weights=[embedding_matrix], input_length=max_length, trainable=False)

# Embedded version of the inputs
encoded_left = embedding_layer(left_input)
encoded_right = embedding_layer(right_input)

# Since this is a siamese network, both sides share the same LSTM
shared_lstm = LSTM(n_hidden)

left_output = shared_lstm(encoded_left)
right_output = shared_lstm(encoded_right)

# Calculates the distance as defined by the MalSTM model
malstm_distance = Merge(mode=lambda x: exponent_neg_manhattan_distance(x[0], x[1]), output_shape=lambda x: (x[0][0], 1))([left_output, right_output])

# Pack it all up into a model
malstm = Model([left_input, right_input], [malstm_distance])

# Adadelta optimizer, with gradient clipping by norm
optimizer = Adadelta(clipnorm=gradient_clipping_norm)

malstm.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['accuracy'])

# Start training

callbacks = [EarlyStopping('val_loss', patience=10, verbose=1),
             ModelCheckpoint('model/current_malstm_weights.h5', save_best_only=True, save_weights_only=True, verbose=1)]

malstm_trained = malstm.fit([x_train_q, x_train_a], Y_train, batch_size=batch_size, nb_epoch=n_epoch,
                           validation_data=([x_val_q, x_val_a], Y_val), callbacks=callbacks)
```

Fig 5. Model configuration of Modified Siamese Manhattan LSTM

### 3.4 Sentence Embedding Similarity

由於這次的 testing 有大量題目都是完完全全的 oov，因此使用 LSTM 組成的 RNN 的表現都非常的不如預期。因此最後我們改回從簡單的 word embedding 下手，將每個 sentence 的 vector  $s$  定義為其中所有不是 oov 的 vector 總和平均：

$$s = \sum \frac{v_i}{n}$$

接下來只需要單純比較題目與選項經過 sentence embedding 之後得到的 vectors 的 cosine similarity 即可。

## 4 Experiments and Results

我們主要針對了 word2Vec 函式裡面的四個參數去做變化，dimension size、window size、minimum count 以及 sg(skip-gram)。下面的關係圖由於每一次在訓練 word embedding 的時候都有隨機的參數在裡面，因此準確率都只是個多次實驗下來的估計值。

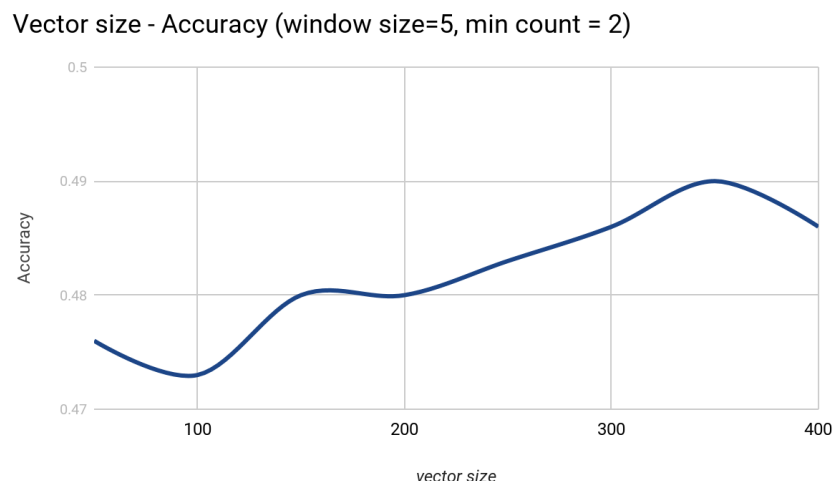


Fig 6. Word-embedding vector size comparison

先看到維度對於準確率的影響，可以看到說在 200 維度到 400 維度之間會有著較好的準確率，但是其 trade-off 則是時間的花費，在 150 維度以下，基本上是 2 分鐘之內可以跑完的，不過到了 200 維度到 400 維度之間，平均每多 50 個維度會多花 50 秒的訓練時間，因此考慮到比較合理的結果，選擇 200~300 維度是個不錯的選擇。

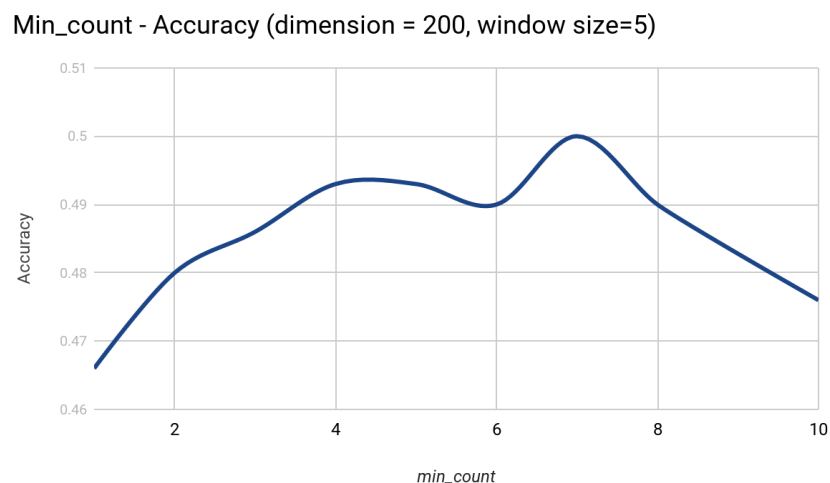


Fig 7. Word-embedding min\_count comparison

這個參數在 4 至 8 之間會有比較好的效果，且此參數與時間無相關性，因此可以在適當的範圍內以及其他參數變化之下去做小幅度的微調。

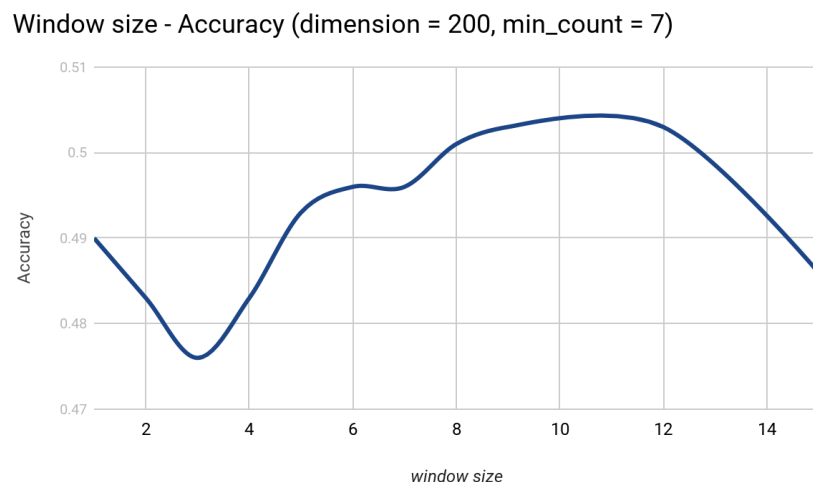


Fig 8. Word-embedding window size comparison

這個則是在訓練 word embedding 的時候，前後會看幾個單詞的相關訓練參數，當我們看得越多，理論上來說也會得到較佳的結果，但是過多的時候又會失去某個單詞跟前後文的重要性。可以看的出來在 5~12 之間會有比較好的準確性，但是這個參數跟訓練時間有著正向關係，平均每多一個 window size，會多花 20~30 秒的運算時間，因此這也是一個需要去做實驗衡量出最適當的參數。

### skip-gram - Accuracy

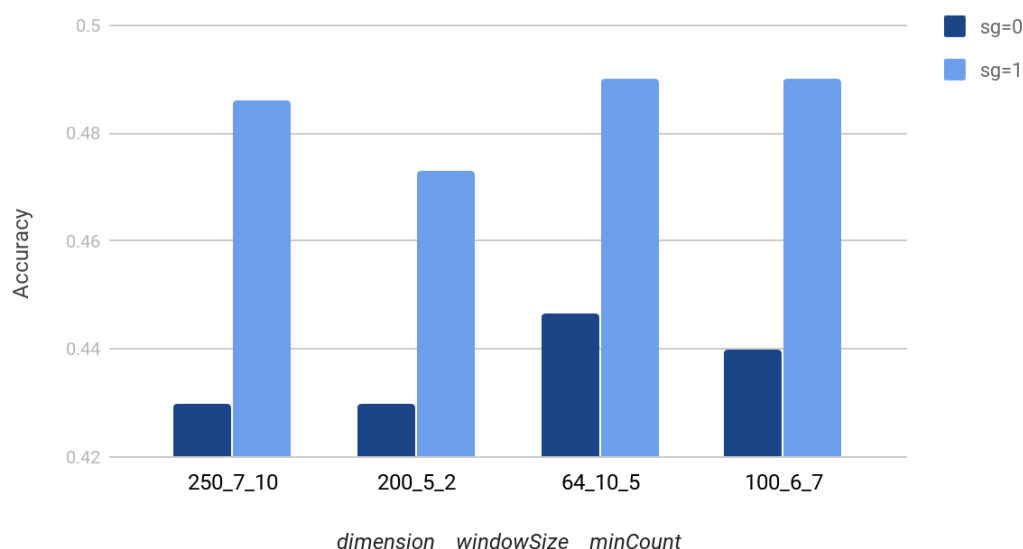


Fig 9. Word-embedding training method comparison

最後我們更改了 word embedding 的 training 方法，sg=1 的時候，word2vec 會使用 skip-gram 演算法去訓練 word embedding，我們隨機抽取幾組參數出來，會發現說使用此演算法的確會有著顯著的進步，但是其代價則是所花費的時間，平均下來有使用 skip gram 的訓練過程會多花 4 倍的運算時間，不過相比準確率如此大幅度的提升，這樣的選擇是值得的。

下表為我們在前一章節中所介紹四個 model 分別在這次 final 中所得到的在 kaggle 上最高的分數（為求統一皆使用未經 ensemble 的模型比較）：

|                                 |         |
|---------------------------------|---------|
| Paired-word Similarity          | 0.42806 |
| Siamese Manhattan LSTM          | 0.34347 |
| Modified Siamese Manhattan LSTM | 0.38379 |
| Sentence Embedding Similarity   | 0.50197 |

Table 2. Different models' best performance via kaggle

這次 final 之所以複雜的 RNN 表現比起單純的 embedding 要來得差其實是很值得討論的現象，我們推測有以下幾個原因：因為我們是提取訓練資料中前後兩句作為一個 pair 標記為 1 當 positive training data，random 抓取兩個句子做一個 pair 標記為 0 當 negative training data。但是 testing 的題目以及選項有很大量不是從 training data 中所出的題目，造成 RNN 的 train 法根本對模型完全沒有幫助，即是在 training 過程中達到了 60%左右的 val acc，在 testing 時還是無法精準答對出那種整題都是 oov 的狀況。

## 5 Team Work Division

|     |                          |
|-----|--------------------------|
| 楊正彥 | Coding、Model tuning、撰寫報告 |
| 李昂軒 | Coding、Model tuning、撰寫報告 |
| 林奕廷 | Preprocessing、撰寫報告       |
| 郭恆成 | Coding、Model tuning、撰寫報告 |

## 6 Reference

- [1] Chen, Z., Zhang, H., Zhang, X., & Zhao, L. Quora Question Pairs.
- [2] Jonas Mueller and Aditya yagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. In Proceedings of the irtieth AAAI Conference on Arti cial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. 2786–2792.
- [3] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In ICLR.
- [4] W. Yih, X. He, C. Meek. 2014. Semantic Parsing for Single-Relation Question Answering. In Proceedings of ACL 2014.
- [5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In ICLR Workshop Papers.