# Machine Learning Final Project
# Report

**Selected Problem (1%)**：TV Conversation

**Team Information (1%)**：

(a) Team Name：NTU_r06942074_DeepMind亞洲分部

(b) Members：
R06942074 李宇哲、R06942033 顏嘉緯、R06921001 簡碩辰、R06922082 蔡瑋倫

(c) Work Division：

| | |
|---|---|
| R06942074 李宇哲 | Scripts writing for training and testing, Presentation, Report writing: Model Description |
| R06942033 顏嘉緯 | Hyperparameters tuning, Ensembling effective answers |
| R06921001 簡碩辰 | Quality testing, Report writing: experiments |
| R06922082 蔡瑋倫 | Initial code construction, Hyperparameters tuning |

## A. Preprocessing / Feature Engineering (3%)

將 training data 全部讀進後，經過中研院的繁體 jieba 斷詞，將每三句話連成一句新的話，例如：(xxx), (yyy), (zzz), (iii), (jjj) 連成 (xxx yyy zzz), (yyy zzz iii), (zzz iii jjj)，每次如果只看一句話的話，資訊略嫌不足，可能會少掉上下句的相關性等等，因此我們最後實驗出把三句話連成一句新的話再丟入 Word2Vec 進行訓練得到的結果是比較好的。另外，由於在斷詞後的相鄰句子間有很大的機率出現重複的詞，會導致在訓練 Word2Vec 時，可能會有 bias，於是我們將相同、連在一起的詞刪除，例如：（我愛黑澀會 黑澀會 黑澀會）→（我愛黑澀會）。

## B. Model Description (At least two different models) (7%)

### B.1 Methodology

我們的方法主要是先將拿到的句子用 jieba 去切成每一個字詞，然後去訓練每個字的 word to vector，然後再用訓練的 word2vec 建立每一個句子的 vector (i.e. sen2vec)，然後比較題目與選項兩個句子間的 cosine similarity，最後選出相似度最高的選項作為最後的預測結果，當然若想改用其他的方法來預測結果也是可以，例如 L2 norm 等。

Jieba → Word2Vec → Sen2Vec → Cosine Similarity

### B.1.1 Jieba

Jieba 分詞結合了基於規則和基於統計兩類方法。

中文分詞的模型實現主要分類兩大類：基於規則和基於統計。

1. 基於規則

基於規則是指根據一個已有的詞典，採用前向最大匹配、後向最大匹配、雙向最大匹配等人工設定的規則來進行分詞。

例如對於「上海自來水來自海上」這句話，使用前向最大匹配，即從前向後掃描，使分出來的詞存在於詞典中並且盡可能長，則可以得到「上海/自來水/來自/海上」。這類方法思想簡單且易於實現，對數據量的要求也不高。

當然，分詞所使用的規則可以設計得更覆雜，從而使分詞效果更理想。但是由於中文博大精深、語法千變萬化，很難設計足夠全面而通用的規則，並且具體的上下文語境、詞語之間的搭配組合也都會影響到最終的分詞結果，這些挑戰都使得基於規則的分詞模型愈發力不從心。

2. 基於統計

基於統計是從大量人工標註語料中總結詞的概率分布以及詞之間的常用搭配，使用有監督學習訓練分詞模型。

對於「上海自來水來自海上」這句話，一個最簡單的統計分詞想法是，嘗試所有可能的分詞方案，因為任何兩個字之間，要嘛需要切分，要嘛無需切分。

對於全部可能的分詞方案，根據語料統計每種方案出現的概率，然後保留概率最大的一種。很顯然，「上海/自來水/來自/海上」的出現概率比「上海自/來水/來自/海上」更高，因為「上海」和「自來水」在標註語料中出現的次數比「上海自」和「來水」更多。

而我們所使用的套件是：

結巴中文斷詞台灣繁體版本
https://github.com/ldkrsi/jieba-zh_TW

因為此次的資料都是繁體字，故使用繁體斷字相對於內建的簡體 jieba 斷字應該會來得好上許多。

### B.1.2 Word2Vec

通常有兩大知名演算法去實作此功能，分別是 Skip-Gram 與 CBOW。

#### 1. Skip-Gram

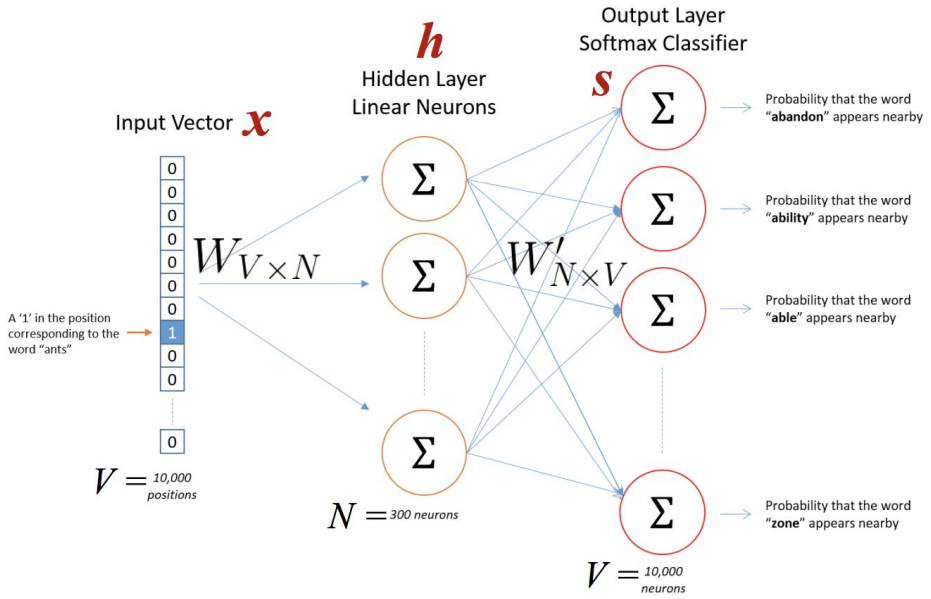
主要概念就是從給定的某個字去預測附近會出現哪些字，因此 objective function 就是要 maximize 給定一個中間的字時 window size 裡面字的機率。

$$w_1, w_2, \cdots, \underbrace{w_{t-m}, \cdots, w_{t-1}}_{w_I}, w_t, \underbrace{w_{t+1}, \cdots, w_{t+m}}_{w_O}, \cdots, w_{T-1}, w_T$$

context window

$$p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) = \prod_{C} p(w_{O,c} \mid w_I)$$

因此，為了要 maximize 這樣的 objective function，需要有一個 neural network 去訓練這些字詞。通常 word2vec 都會有一層 hidden layer，然後還有 output layer。




(上圖取自陳縕儂老師的課堂投影片)

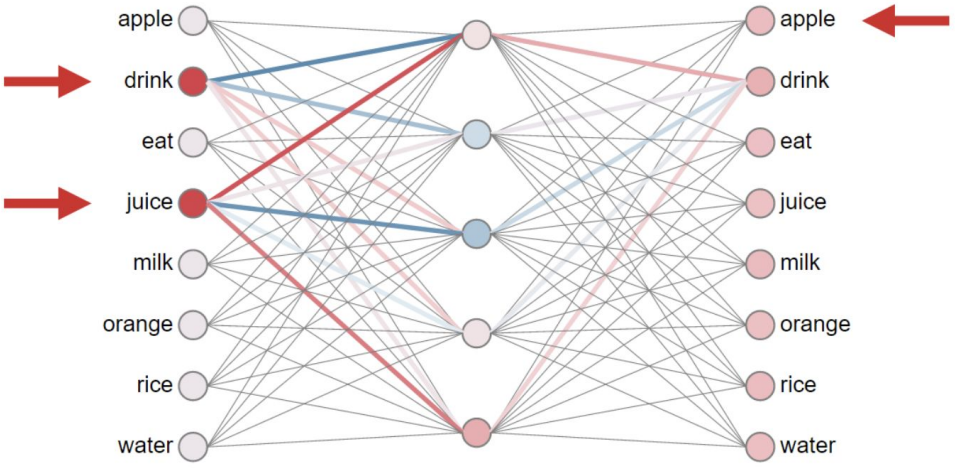
Loss Function:
給定一個 target word Wi：

$$\begin{aligned} C(\theta) &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) \\ &= -\log \prod_{c=1}^{C} \frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})} \\ &= -\sum_{c=1}^{C} s_{j_c} + C \log \sum_{j'=1}^{V} \exp(s_{j'}) \end{aligned}$$

2. CBOW

CBOW 的方法跟 skip-gram 不同的是，CBOW 主要是給 surrounding words 然後去 predict 中間的 word，基本上概念跟 skip-gram 相反。

$$p(w_t \mid w_{t-m}, \cdots w_{t-1}, w_{t+1}, \cdots, w_{t+m})$$




## B.1.3 Sentence to Vector

Sentence to vector 就是第三個步驟，將一句話的每一個字用 word2vec 產生的向量加起來取平均。例如一個句子裡所有詞向量取平均，可以粗略的作為句子的向量，句子之間可以用向量來計算相似度。假設對話中的主題概念大致上都很相似，以下示範比賽題目中，使用詞向量平均找出答案句的簡單快攻法。

```
emb_cnt = 0
avg_dlg_emb = np.zeros((dim,))
# jieba.cut 會把dialogue作分詞
# 對於有在word_vecs裡面的詞我們才把它取出
# 最後詞向量加總取平均，作為句子的向量表示
for word in jieba.cut(dialogue):
  if word in word_vecs:
    avg_dlg_emb += word_vecs[word]
    emb_cnt += 1
avg_dlg_emb /= emb_cnt
```

### B.1.4 Cosine Similarity

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

將答案與每一個選項做 cosine similarity 然後找到相似度最大的選項。

```
for idx,ans in enumerate(answers):
  avg_ans_emb = np.zeros((dim,))
  for word in jieba.cut(ans):
    if word in word_vecs:
      avg_ans_emb += word_vecs[word]
      emb_cnt += 1
  sim = np.dot(avg_dlg_emb, avg_ans_emb) / np.linalg.norm(avg_dlg_emb) / np.linalg.norm(avg_ans_emb)
  print("Ans#%d: %f" % (idx, sim))
  if sim > max_sim:
    max_idx = idx
    max_sim = sim

print("Answer:%d" % max_idx)
```

## B.2 Model Description

1. 主架構：

   Step1：使用繁中版本的 jieba。(https://github.com/ldkrsi/jieba-zh_TW)

   Step2：結合連續三個句子當成一個新句子。

   Step3：對新句子進行 **SG / CBOW** Word2Vec Training，並使用以下參數。
   <size=64, min_count=0, sg=1, window=unlimited>

   Step4：使用訓練出來的 word2vec mapping model 對每個字詞做 mapping，並對每個句子算出平均向量。

   Step5：使用餘弦相似法，比較題目以及各個選項的平均向量，選出角度最相近的選為最後答案。

2. 兩種模型的差別：

   模型一：在 Step3 使用 Skip-Gram 演算法。
   模型二：在 Step3 使用 CBOW 演算法。

## C. Experiments and Discussion (8%)

1. 實驗數據分析：

我們查閱了許多文獻、試了許多不同 Word2Vec 參數，將其列為以下表格。其中：
> Unit 代表我們將連續 Unit 行文字結合成一句話，增加字詞之間的關聯性。
> Feature size 為將字詞映射到幾維的特徵向量。
> Accuracy 為最後 Kaggle 準確率。

部分實驗數據整理：

| Unit | Feature Size | SG Accuracy | CBOW Accuracy |
|---|---|---|---|
| 1 | 384 | 0.42411~0.43 | 0.40039 |
| 1 | 256 | 0.44150~0.45 | 0.42985 |
| 2 | 384 | 0.48102~0.49 | 0.45067 |
| 3 | 384 | 0.48735~0.49 | 0.45296 |
| 4 | 384 | 0.47707~0.48 | 0.43560 |
| 5 | 384 | 0.47075~0.48 | 0.43450 |
| 3 | 128 | 0.50513~0.51 | 0.44678 |
| 3 | 64 | 0.51146~0.52 | 0.44940 |

最後採用前三好的 SG Word2Vec 模型來進行 ensemble，得到結果為 0.54822。

2. 實驗討論：

(1) Feature Size 的調整
之前看網路上文獻想說都建議 300 維是較為足夠，結果沒想到這次的實驗結果反而顯示 64 維是為最好參數，我想是因為這次資料只提供 5 個劇本，雖然字數很多，但其實同一份劇本中的內容算是相近的，也就是說如果提供過多的 feature size 給所有字詞去 fit，到最後就會趨近於一個字佔一個 feature，意即 one-hot encoding。

(2) Unit 的調整
劇本中的文字大多數都是對話呈現，常出現如「對啊」、「對了」、「好」、「好了」、「是啊」、「不」等短詞，如果直接當成一個句子便跟其他對話連結不起來，因此將多行合成一句有助於語意理解。然而當一個句子所包含的行數太多，加上 window length 沒有限制的條件下，就會變成把某些互相沒有關係的句子也學進去，如此一來對語意理解也有莫大影響，因此必須挑一個適當的數量進行合併。

(3) 隨機性
我們發現演算法在做 Word to Vector 時，可能會有隨機性，也就是每次做出來的 Mapping Function 可能會不一樣。這也對我們實驗觀測造成一大影響，原因應該是計算 Word2Vec 有使用到 HMM (Hidden Markov Model) ，而這牽涉到機率分布，算是一種不可避免的現象。經實驗，每次的結果丟上 Kaggle 誤差不超過 0.02。

(4) Skip-Gram 與 CBOW 的比較
Skip-Gram 是給定 input word 來預測上下文，但 CBOW 剛好與之相反，是給定上下文來預測 input word。Skip-Gram 一次只會輸出一個向量，但 CBOW 會把周圍的字詞都變成向量，例如「今天 | 是 | 晴天」，假設 input word 是「是」，Skip-Gram 會透過演算法得到「是」這個字的向量，而 CBOW 會去得到「今天」和「晴天」的向量，大部分的 Word2Vec 都是使用 CBOW 居多，因為會比較想去得到這個字與其他周圍字詞的相關性，進而去達到預測的目的，不過這次的資料前後上下句不見得有太大相關性，甚至可能完全無關，所以使用 Skip-Gram 反而可以達到比較好的表現。