Machine Learning 2017 Fall- Final Report

（零）題目與分工

- **Topic:** TV Conversation
- **Team:** NTU_b03902096_我用GPU你用deepQ
- **Members and teamwork**:
    - b03902016 周聖筌 Preprocessing
    - b03902017 曹峻寧 Preprocessing, Train and tune word2vec
    - b03902078 林書瑾 Skip-thoughts, RNN sentence similarity model
    - b03902096 陳柏屹 Preprocessing, Sentence vectors cosine similarity model

（一）前處理

對資料的前處理可以分為三大部分：首先要做完資料清理；接著，以 OpenCC 做繁體轉簡體的處理，並透過 Jieba 進行斷詞；最後，對於訓練資料我們會串接句子來增加其長度。另外，對於 testing data 則還需要在最開始先將原始檔案 parse 成方便我們進行預測的格式（我們將每個題目與每個選項各自做成一行，無論那個題目或選項包括幾句話，因此預測時讀的會是一行題目、六行答案交替的格式）。

以下介紹每個部分，以及每部分可以嘗試的變化。

1.資料清理

我們將資料中所有的符號（包括空格、換行）、英文以及數字去除，這部分其實很難做得乾淨，但我們檢查後確認的確是只留下了乾淨的中文字母無誤，完美地進行了資料清理。

這部分可以嘗試的是是否去除停用詞，我們做過一些嘗試後，去除停用詞對模型的表現造成的影響似乎不太一定，可能變好也可能更差，最後是決定統一都不去除停用詞。

2.繁轉簡、斷詞

我們嘗試過直接用繁體字進行斷詞（並將詞典替換成 github Idkrsi/jieba-zh_TW 的繁體詞典），或者透過 opencc-python (based on OpenCC) 將資料轉完簡體後用 jieba 本身的簡體辭典斷詞。效果比較可以參考實驗和討論的部分。整體而言，轉簡體再斷詞的效果還是會好一些。

另外，結巴斷詞時我們使用的是精確模式，此處也可以選擇全模式或搜尋引擎模式，但嘗試後發現還是精確模式最好。

3.串接句子

原始訓練資料會有很多過短的句子，平均而言，句子的長度甚至只有 4.x 個詞，對於訓練 word2vec 無疑是非常不利的。並且，其實訓練資料中許多時候要將前後幾句接起來才是真正完整的句子，例子如下：

```
不過 這個 妹妹
怎麼 跟 大姊 長 得 不太像
你 是 不 是 說 我們 雅信 笑臉 常開
他 妹妹 卻 是 臭臭 的 臉
我 也 搞 不 懂
同一 個 肚子 生出來 的
怎麼 會 那麼 不同
我 只 知道 我 在 生雅信 的 時候
心情 很 開心
但是 懷他 妹妹 的 時候
他 阿爹 身體 開始 不 好
我 心情 也 很 鬱悶
```

不難發現其中很多原始句子其實是沒有把話說完整，如果串接句子，資料看起來就好多了，且平均的句子長度也會變得比較合適，讓 word2vec 的 window 能有發揮空間。例子如下：

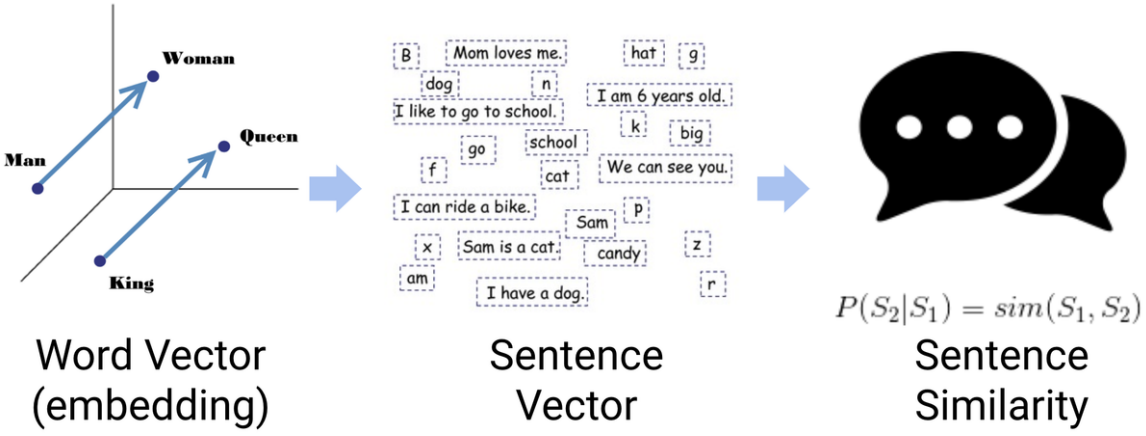
```
不過 這個 妹妹 怎麼 跟 大姊 長 得 不太像 你 是 不 是 說 我們 雅信 笑臉 常開 他 妹妹 卻 是 臭臭 的 臉
我 也 搞 不 懂 同一 個 肚子 生出來 的 怎麼 會 那麼 不同 我 只 知道 我 在 生雅信 的 時候
心情 很 開心 但是 懷他 妹妹 的 時候 他 阿爹 身體 開始 不 好 我 心情 也 很 鬱悶
```

這部分會牽扯到 3 個參數：minimum length (只看至少幾個詞以上的句子，測試資料中許多長度為 1 的句子如「來」、「好」等就可能在此時被消除)、concat window（要串接幾個句子形成一個新句）、stride（串接句子時每次移動幾行作為新的中心句，stride=1 時代表原始資料除頭尾外的每一句都會被當作中心句，也就是會有很多 overlap）。

關於這部分的參數有很多排列組合可以選擇、且彼此具備某些相關性。我們最後選擇 concat_window 7 為最佳參數（此時句子平均長度為 30 ），至於 minimum length 以及 stride 介於 1 ~ 3 時對結果的影響似乎不顯著（太大的時候訓練資料會銳減、結果則會明顯變差），因此可以隨意設置。

（二）模型

模型一：Sentence vectors + cosine similarity model



1. **Word Vector**
   透過前述的前處理得到訓練資料後，透過 gensim 的 word2vec 得到 word vectors。這部分有非常多 word2vec 的參數可以進行調整，我們留待實驗與討論進行詳述。此外，我們也嘗試過使用 Facebook 推出的 fasttext train word vectors，但看起來對模型表現影響不大，因此最後仍都使用 gensim。

2. **Sentence Vector**
   利用剛剛已經訓練完成的 word vector 來取得代表每一個 sentence 的 vector。

   最開始時，我們使用平均法，亦即sentence vector 是所有構成他的 word vector 平均而來。此時對於 OOV 我們嘗試兩種作法：直接忽略或是給 0 vector，但給 0 vector 其實就只是使 sentence vector 被依照 OOV 的數量被 scale down，對 cosine similarity 應無影響，故我們決定直接忽略。

   後來參考 A Simple but Tough-to-Beat Baseline for Sentence Embeddings (Sanjeev Arora et al, 2017) 之中的 Algorithm 1：將 target sentence 以其內每個 word 出現在 training data 中的頻率對 word vectors 做加權平均得到暫時的 sentence vector（注意此處有個 bias 項 α 非常重要）。接著對所有句子做 SVD 取 first singular vector $$u$$，並將每個 sentence vector * $$(1-uu^T)$$。利用此方法我們可以得到問句的 sentence vector 和每個回答選項的 sentence vector，這樣做出的 sentence vector 對模型表現有明顯幫助。整個 sentence vector 演算法簡單的 pseudo code 如下頁所示。

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

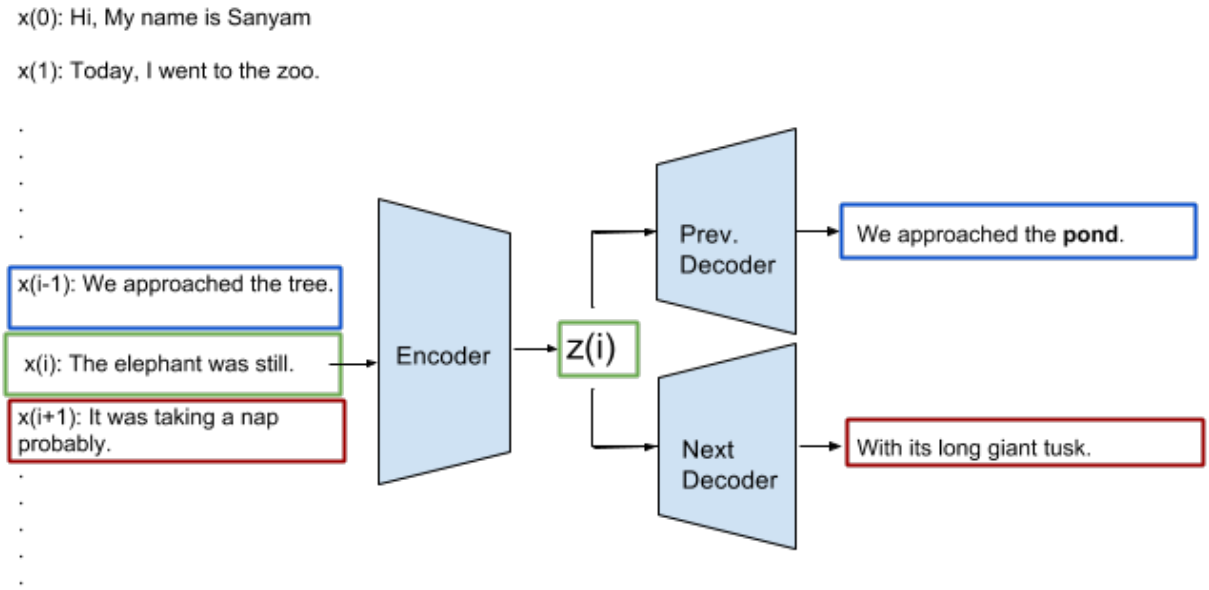
3. **Sentence Similarity**

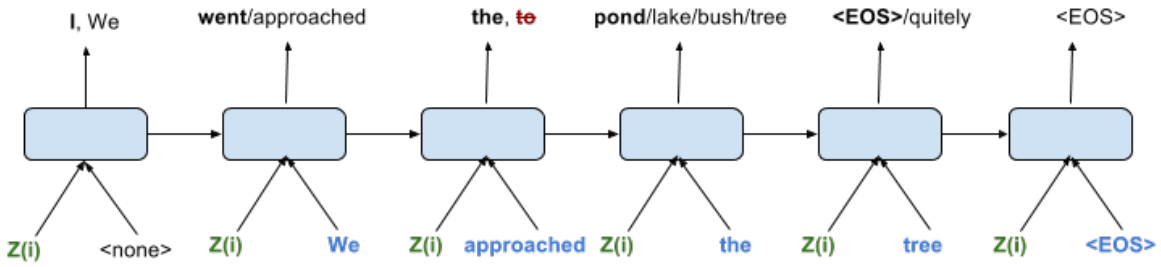
在得到 test data 中句子的 sentence vector 後，每個問句 - 選項配對都利用套件 SciPy 去計算兩個 sentence vectors 的 cosine similarity，最後選擇 cosine similarity 最大者為該題的答案。

我們另外使用了 ensemble 的技巧，也就是調整 gensim的參數生出不同的 word2vec models，並讓每個模型各自產出答案，最後再用一模型一票的方式票選出最終答案。我們也另外嘗試過取所有模型預測出的最高分、或取平均分數最高分當作答案，表現都不如投票。

模型二：Skip-thought Vectors

Skip-thought vectors 的作法與傳統不同，其著重考慮句子與句子間的關係，並且使用 homework 6 的 Autoencoder 技術，來學習 sentence embedding。





整體架構如上，先利用 RNN 實做 encoder，將句子中抽出向量 z(i) 當作 feature，而在 decoder 的部份，skip-thought 使用兩個 decoder，希望能藉由 z(i) 以及前一個字，一起放進不同 decoder 中，期望能夠 decode 出上一個或下一個句子。如圖中的第二塊 RNN cell，輸入為前一個 RNN 的 state 加上 z(i) 以及前一個字 We，期望能 decode 出 “approached” 這個單字，同時也是 objective function minimize 的對象，詳細公式請參見論文 formula (10).

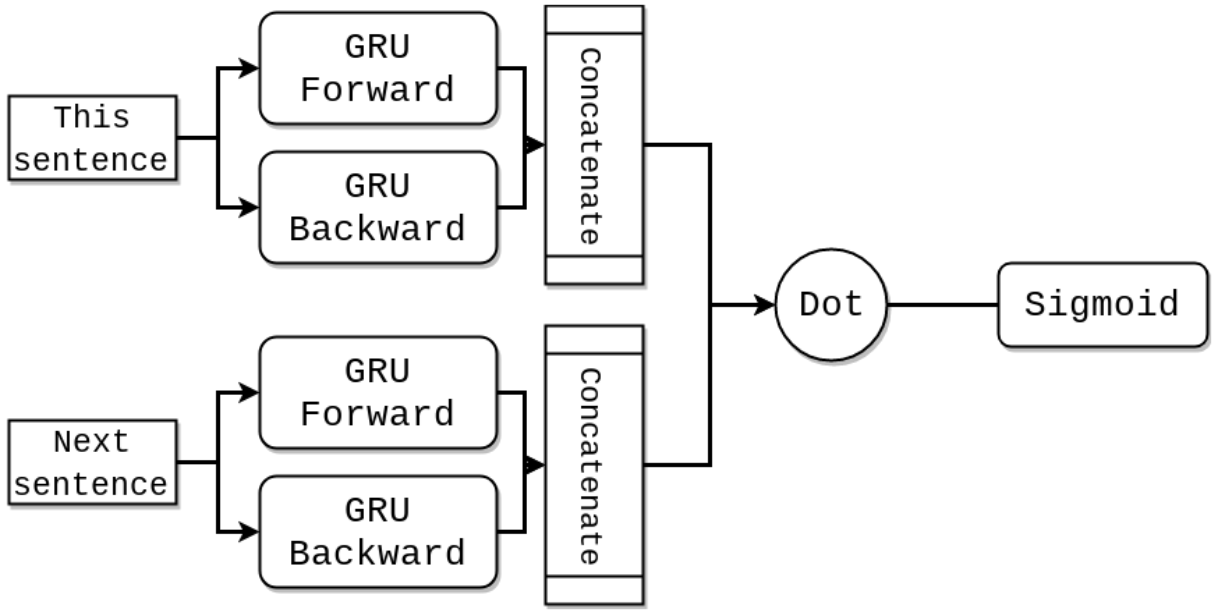
我們使用原論文作者所實做的 GitHub 原始碼來訓練 skip-thought vectors，修改以下步驟：

- 將原始的 training data 全部相接，並且只留下連續兩句話的 word >= 3 的句子，約有 30000 句
- 原實做有使用額外的資料做 vocabulary expansion，我們則使用從 training data 得到的 word2vec 代替
- dim_word 256, GRU 1024 units, , n_words 10000
- 使用 Adam optimizer，batch size 為 128，訓練 6 個 epoch
- 使用 GTX 750 約在數個小時內可以訓練完畢

下一個步驟是尋找 sentences 之間的關聯性，我們首先使用 cosine similarity來做，但成果不如預期，僅有 34%的準確率；因此我們嘗試用模型三來訓練前後句子向量之間的關係，然而 training data 卻沒有任何起色，準確率甚至與亂猜無異。因此，我們合理推測，skip thought 在這個 task 上成效並不佳，也許是因為 training data 性質的緣故。原論文作者使用的句子比較長，且是關於書籍的資料，而非對話，上下關聯性強，而我們的 data 上下關聯性不足，句子多數為口語短句，這讓 skip thought 難以 decode 出前後句，再加上有效的 training data 數量不足，testing data又有許多完全與 training data 無關的句子（尤其是問句會顯著影響結果），因此沒有辦法達到好的成效。雖然這個模型在此 task 上宣告失敗，但他的想法與概念非常新穎，直接用上下句來訓練 vector，值得學習。

模型三：RNN Sentence Similarity

訓練完畢 word embedding 後，我們想用 RNN 的方法找出 sentence similarity，設計模型如下：



- 將原始的 training data 全部相接，並且只留下連續兩句話的 word >= 3 的句子，約有 30000 句
  - 這些句子兩兩相鄰視為一組 label 1 的 training data
  - 將所有句子 shuffle 後，兩兩相鄰視為一組 label 0 的 training data
- 兩個句子分別經過 Bidirectional GRU，並設定 dropout 防止overfit
- 再將兩個代表 sentence 的向量內積，過 sigmoid
- Loss function 為 mean square error，optimizer 用 Adam
- 輸出為是否是上下句關係的機率

我們用 RNN 的特性來訓練句子的向量，然後再將兩者內積後過 sigmoid 輸出 0 ~ 1 之間的機率。這裡的假設是句子向量在高維度空間中會有特殊的關係，可以用非線性的 RNN 將各個 word embedding 轉換成這個空間中的向量，再透過內積區分上下句與非上下句的關聯。

在訓練過程中，抽取了 10% 的 shuffled training data 作為 validation data，然而訓練結果並不如預期，即使 training data 可以有效達到八成以上準確率， validation data 最高僅有六成，發生嚴重的 overfit，即使調高 dropout rate 或修改模型，幫助都非常有限。我們嘗試了加入更多 Dense 層、TimeDistributed 層試著做到 DNN 以及 Attention 的效果，但成效都沒有這個原始的 model 佳。

最後，最佳的模型便是上圖中設計的模型，該模型最後在正式的 testing data 上只獲得 35% 左右的準確率，並沒有達到很好的成效。

最佳模型簡介

我們最終在 Kaggle 上的最佳模型屬於模型一（Sentence vectors + cosine similarity model），採取 weighted sum 取得 sentence vectors，並透過投票 ensemble 3 個 word2vec models。

由於單個 word2vec model "SC_minlen1_concat7_str1_size64_window7_min2_sg" 在 Kaggle public set 就能取得 0.526 的準確率，我們以其為底微調 minlen 與 stride（分別設為 2）做出另外 2 個 models 進行 ensemble，最終在 Kaggle public set 能達到準確率 0.535。

由 model 名稱可以得知其對應的參數。概括而言，我們最佳模型採用了簡體字斷詞、串接 training data 前後 7 個句子形成新的 training data，word2vec 的 size (dimension) = 64, window = 7, min_count = 2，全都使用 Skip-Gram，其他 word2vec 參數則為預設值。

（三）實驗和討論

Model No.	單/多	繁簡	頻率加權	minlen	concat	stride	size	window	min	sg	kaggle
1	single model	繁	X	1	5	1	200	5	1	sg	0.4747
2	single model	繁	X	1	5	1	200	3	2	sg	0.45928
3	single model	繁	X	1	7	2	256	7	2	sg	0.48063
4	single model	繁	O	1	7	2	256	7	2	sg	0.50909
5	single model	繁	O	1	7	1	64	7	2	cbow	0.47667
6	single model	繁	O	1	7	1	64	7	2	sg	0.51818
7	single model	簡	O	1	7	2	256	7	2	sg	0.51383
8	single model	簡	O	1	7	2	256	7	2	cbow	0.46403
9	single model	簡	O	1	7	1	256	7	2	sg	0.50869
10	single model	簡	X	1	7	2	256	7	2	sg	0.49604
11	single model	簡	O	1	7	1	64	7	2	sg	0.52648
12	single model	簡	O	1	7	1	64	7	2	cbow	0.47667
13	single model	簡	X	1	7	1	64	7	2	sg	0.4992
14	single model	簡	O	1	7	1	64	7	1	sg	0.53122
15	single model	簡	O	1	7	1	64	3	2	sg	0.49723
16	ensembled model	簡	O	1	7	1	64	7	2	sg	0.53517
		簡	O	2	7	1	64	7	2	sg	
		簡	O	1	7	2	64	7	2	sg	

模型一實驗結果

我們的實驗主要著重在模型一。包括改變前處理、train word2vec 的參數以及生成 sentence vector 的方式。為求方便，以下用 (a, b) 表示 model 編號 a, b 的比較。

1.前處理

- 繁 v.s. 簡
  觀察上表 sg 的部分，包括 (3, 11), (4, 7) 都顯示出轉簡體再斷詞 (SC) 大約都會比繁體斷詞的準確率好 1% 左右。
- minlen (minimum length), concat (concat window), stride
  由 (7, 9) 來看，stride 2 稍優於 stride 1，亦即 training data overlap 的情形會較小（但 stride 太大時 training data 會過少）。至於 minlen 與 concat 從上表中不太能明確比較，但我們自己的觀察是 minlen = 1 ~ 3 時影響不大，concat 5 或 7 都可以。

2.word2vec

- CBOW v.s. Skip-Gram
  從 (5, 6), (7, 8), (11, 12) 都能看出 Skip-Gram 的準確率比 CBOW 好約 4 ~ 5%，亦即不論繁體或簡體，在這個工作上 Skip-Gram 會比 CBOW 適合。推測跟我們不去除停用詞、以及 training data 的性質有關。
- size
  size 即是 word vector 的 dimension，越大代表我們使用越多維度代表每個 word。(4, 6) 與 (9, 11) 顯示出 size 64 會比 256 好一些，256 維對於我們的 train data 可能包含太多冗餘資訊。（註：(4, 6) 雖然 stride 不同，但由 (7, 9) 已推測 stride 2 應優於 stride 1，model 6 卻仍然獲勝，故 size 64 仍應優於 size 256）
- window
  由 (1, 2), (1,3), (11, 15) 來看，較大的 window 似乎能提升表現。但是我們觀察 word2vec 出來的詞彙分佈發現，window 越大時，word2vec 每次訓練生成的模型隨機性似乎會越高，且 window > 7 時似乎 word2vec 會容易沒 train 好，因此最後都使用 window 7。
- min_count
  (11, 14) 顯示出 min_count 1 似乎會略好於 2，亦即比起把只出現一次的詞視為 OOV，給予他們某個向量仍然會較好。從 (1, 2) 來看也有類似趨勢（但也可能是由於 model 1 的 size 略大一點）。

3.sentence vector

- Average v.s. Weighted Sum

1. 由 (3, 4), (7, 10), (11, 13) 可發現對不同參數 weighted sum（介紹於模型一）都能對準確率給予 1 ~ 3% 的提升，是非常簡單有效的工具。

RNN 的表現

RNN 被使用在模型二與模型三中，兩者都用來做句子相關的訓練，但是我們發現兩者的表現都不如預期。我們歸納出兩個原因：

1. training data 的句子過短，造成上下文關聯性不足，也讓 RNN 在訓練時能進入 RNN cell 訓練的字非常少。根據我們的切詞統計，training data 中的平均句子長度僅有約 4 個詞。

2. training data 與 testing data 的歧異。在 testing data 中，除了有很多沒有出現在 training data 的相關句子，還有很多模稜兩可的題目，這讓 RNN 並不適合用在這個 task 上。

```
# 模稜兩可
1424,A:都是我太不小心了 才會弄髒你的衣服,A:請你以後小心一點 A:讓我來幫你處理吧 A:吃魚要小心噎到 A:衣服髒並不可恥 A:我一定會打掃得更乾淨 A:髒東西趕緊丟掉啦
2859,A:妳是誰啊 不要過來 B:別害怕 我不是壞人,A:我阿嬤可是開台始祖喔 A:如果月亮是圓的 A:不管妳嗑了什麼都給我來一點 A:爛機車發不動 A:慢著 妳先等等 A:下課十分鐘的戀愛

# 與訓練資料無關
1616,A:我相信大葉大學 絕對是你們最佳選擇 A:因為將來比較好就業,B:好啊 好餓喔 B:我想開車去啦 B:我也這麼覺得 B:哈哈 你嘴吧真甜 B:哇 你真的太猛了 B:人生就是這樣啊
3455,A:有什麼事 比自己的女朋友表演重要,B:交一個女朋友 B:樓上中肯 給推 B:一樓突破盲腸 B:我媽問我為甚麼哭著滑手機 B:柯粉又要出來護航了 B:五樓包莖
```

（四）參考資料與文獻

1. ldkrsi/jieba-zh_TW: 結巴中文斷詞台灣繁體版本。https://github.com/ldkrsi/jieba-zh_TW
2. Sanjeev Arora, Yingyu Liang, Tengyu Ma (2017) A Simple but Tough-to-Beat Baseline for Sentence Embeddings. https://openreview.net/pdf?id=SyK00v5xx
3. Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. In Advances in neural information processing systems (pp. 3294-3302).
4. Sanyam Agarwal (2017). My Thoughts On Skip Thoughts. https://sanyam5.github.io/my-thoughts-on-skip-thoughts/
5. NTU_r05525066_kaggle分身2 (2018). ML Final Project TV-conversation. https://drive.google.com/drive/folders/1Qfq-4EIFreMtIPQaMuE9_93leaHDMMcu
6. ryankiros/skip-thoughts.Sent2Vec encoder and training code from the paper "Skip-Thought Vectors" . https://github.com/ryankiros/skip-thoughts