Introduction

1. Model description

- 每一筆 testing data 中的句子，是由三個部分所構成。第一部分是空格之前的詞 (上文)，第二部分是句子中被挖掉的詞，也就是空格，第三部分是空格之後的詞 (下文)。因此如果能用某種形式的 token 來分別代表這三個部分，在找出這三個部分相互的關係，那便有機會預測缺失的字。
- 我們首先用 gensim 以及 training data 來 train 每個字的 word vector，接著要以 train 好的 word vector 來表示上下文。根據 word2vec paper 裡的說法，word vector 具有加法性質，把某兩個 word 的 vector 相加所得到的 vector，可以一定程度的代表這兩個 word 合併起來的語意。
- 我們把這個概念推展到 sub-sentence，試著用一個 vector 來表示上文，一個 vector 來表示空格，一個 vector 來表示下文。上文及下文的 vector，就是分別把上文下文中的每一個 word 的 vector 相加。
- 在 training 階段，我們會把 training data 中的每一個句子的前半句當成上文，最中間的字當成空格，後半句當成下文，因此每一句話都能得到三個 vector。我們希望用前兩個 vector 來預測第三個 vector，因此 RNN 的 input 是有兩個 timestamps 的 sequence，每個 timestamp 會輸出一個 vector，我們希望在第二個 timestamp 的 output vector 能和第三個 vector 越像越好。
- 在 testing 階段，我們會把五個選項分別填進空格，因此每一筆 testing data 會重複用上文的 vector 以及各個選項的 vector 來預測下文的 vector，看哪個選項所 output 出來的 vector 與下文的 vector 最相似，就把那個選項當答案。

2. Improvements

- 一開始的 model 是用句子中前面的詞來預測句尾的字，然後在 testing 時把五個選項分別填入空格，看看預測出來的字跟句尾的字有多高的相似度。但實作這個 model 時發現，五個選項丟進去 output 出來的 vector 其實都長得差不多，跟最後一個字做 cosine similarity 的結果差距都非常小，當句子比較長，空格的位置離句尾很遠時，差距會更小。如果改成預測空格後的下一個字，結果也比 baseline 低，因此才想到用上文及空格來預測下文的 model。
- 我們有先算過在 testing data 中最長的句子 (len=34) 以及最短句子 (len=5) 的長度，在切 training data 的句子時，我們只挑選長度介於這兩個值之間的句子。我們也發現 testing data 中空格的平均位置 (9.9) 差不多是句子平均長度 (19.35) 的一半，因此才把句子長度一半的位置當作空格，空格前半當上文，空格後半當下文。
- 在算上下文的 vector 時，我們發現如果沒把 stopword 濾掉，這個 model 會完全 train 不起來，因為我們不是把句子中 word 一個一個丟進 RNN，而是把很多詞合併成一個 vector，由於 stopwords 出現頻率很高，因此句子中的 stopwords 會讓各個句子的區別性降低，導致 performance 很差。(我們把 corpus 中出現次數最高的前幾名當成 stopwords)

- 同理，在整個 corpus 中出現次數過少的字，也必須濾掉，這個步驟會讓結果更好。原因可能是出現次數太少的字有可能是 misspelling，在 word vector 訓練過程中，這些 misspelling 的字跟原本 correct spelling 的字會被 embedded 到完全不同的地方。
- 在 testing 階段，一開始是把上文 vector，選項 vector，下文 vector 丟進 RNN，看哪個選項到的 loss 最低，就選擇那個答案。後來發現直接拿五個選項的 output vector，跟下文的 vector 做 cosine similarity，挑 similarity 最大的當答案，這樣會有 2% 左右的差距。
- 我們從 training set 中切出了大小為 batch_size * 10 的 validation set。如果 RNN 的 output 與預期的 vector，它們的 cosine similarity >= 0.4，就當成預測正確。我們試過了許多參數，包括：layer 數一層到五層，RNN cell 要選 LSTM 還是 GRU，dropout keep probability 從 0.5 到 0.9，loss function 試過 nce loss、cross entropy、cosine similarity，optimizer 試過 Adam、AdaDelta、SGD，最後根據 validation 上表現比較好的幾個設置上傳 kaggle。

3. Experiment settings and results

以下是根據 validation set 得出的結果中，上傳 kaggle 後準確率最高的設定

- *Experiment results*
  - epochs = 2
  - learning_rate = 0.000018
  - batch_size = 256
  - RNN model: 1 layer LSTM (hidden layer size = 800)
  - no dropout
  - word vector size: 300
  - loss function: tf.losses.cosine_distance
  - optimizer: tf.train.GradientDescentOptimizer
  - training time: 1 hr
  - max accuracy: 36.5%
- *Environments*
  - OS: CentOS Linux release 7.3.1611 (Core)
  - CPU: Intel(R) Xeon(R) CPU E3-1230 v3 @ 3.30GHz
  - GPU: GeForce GTX 1070
  - Memory: 16GB DDR3
  - Python2.7.5 (for training) & Python3.5.0 (for parsing)
  - Libraries:
    - numpy 1.12.0
    - scipy 0.18.1
    - tensorflow 1.0.1
    - tensorflow-gpu 1.0.1
    - gensim 1.0.1