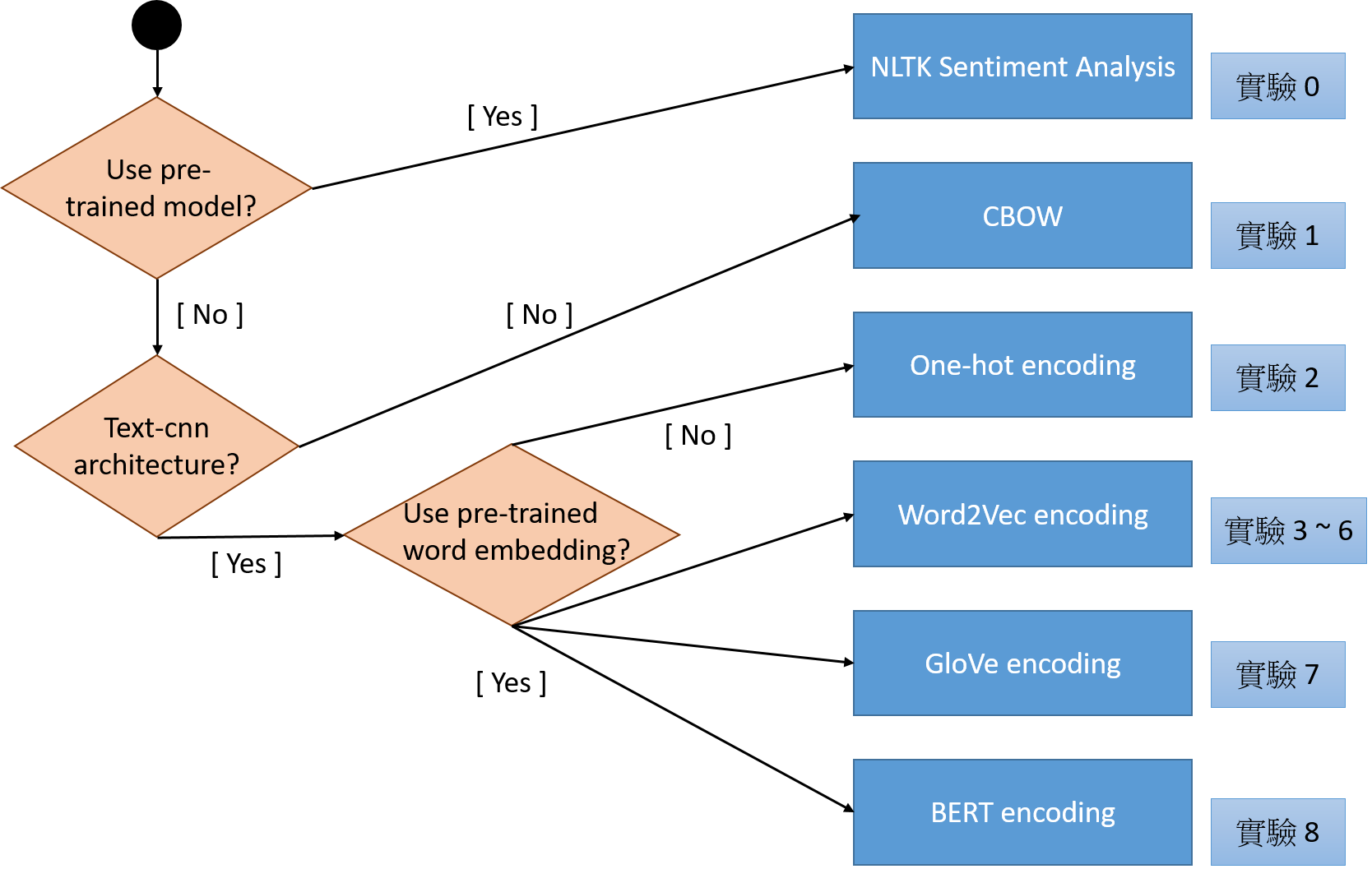
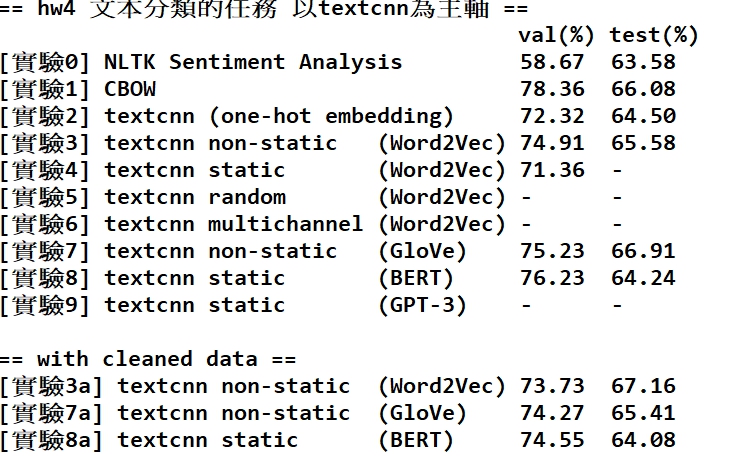
實驗主旨

由於上次論文是報告Kim Yoon的text-cnn，所以這次就拿他的實驗和他的迷行架構來做文本分類啦! 這次的主要任務有兩個，一是重現Kim的text-cnn模型，因為我在github或medium上看到的code都有點爛，沒有一篇code的可讀性和架構是100分的QQ，所以決定來重構囉 (都用keras寫了可讀性麼可以這麼差= =)；二是我想幫Kim補齊2014~2020間的pre-trained word embedding vectors，算是幫這篇論文畫上一個句點，我嘗試了Word2Vec，GloVe，BERT這三個vector space，可惜的是因為時間關係沒能嘗試GPT。

實驗流程



上圖的流程可對應至下方的實驗記錄，紅色框框是採用text-cnn模型架構的實驗。



實驗結果

我的Model Hyperparameter以及Training parameters皆與Kim的論文fine-tuned出的數據一樣，過程中我只會調整training epoches來得出最佳的模型。由上表的準確度來看，其實不管使用哪種word embedding其實結果都差不多，我從數據得出的結論如下:

* NLTK的pre-trained model無法對付本次資料，基本上是用猜的
* CBOW架構簡單，但test acc竟然是最高的，說明了2014的textcnn效能(準確度)不行了
* GloVe.6B.300d相較於Word2Vec.300d (1.6G版)好。(未嘗試過30G版的)
* 由於BERT可以處理oov和拆解字彙，理論上對文義的理解會比GloVe及Word2Vec好。但結果卻相似。

因此，此實驗推薦大家，比起套用更新更厲害的word embedding，不如採用state-of-art的模型架構，如seq2seq，r-cnn，attention等。

當初我會選擇textcnn模型為主軸，便是因為它的param極少，我的文書機都可以10s一個epoch。但是後來訓練fine-tune small BERT時使用Kaggle的GPU跑一個epoch竟然還要1分鐘。因此若是追求以效能會取時間，改進模型的應比改進word embedding合適。

模型與訓練

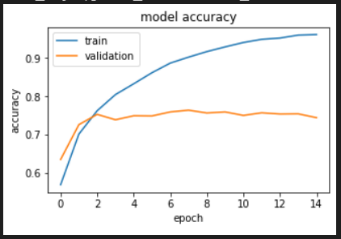
訓練流程:

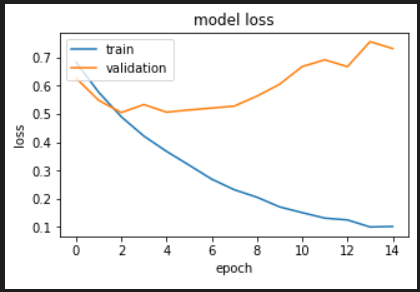
1. [Only for a-brench] a 分支實驗特別將can t don t等含有”n t”的字還原為can’t don’t，因為”n t”在word2vec和glove中屬於oov，還原後才能對應至該向量。
2. 前處理使用keras.Tokenizer完成texts\_to\_sequences。
3. 將embedding layer的權重從Tokenizer中提取出來，存放於embedding\_matrix。
4. 建立textcnn模型
5. 進行訓練，static-cnn不會更新embedding layer的權重，需大約5 epochs；non-static-cnn會更新embedding layer的權重，需大約15~25 epochs。

Ps: 接BERT的前處理不用做第二、三點。

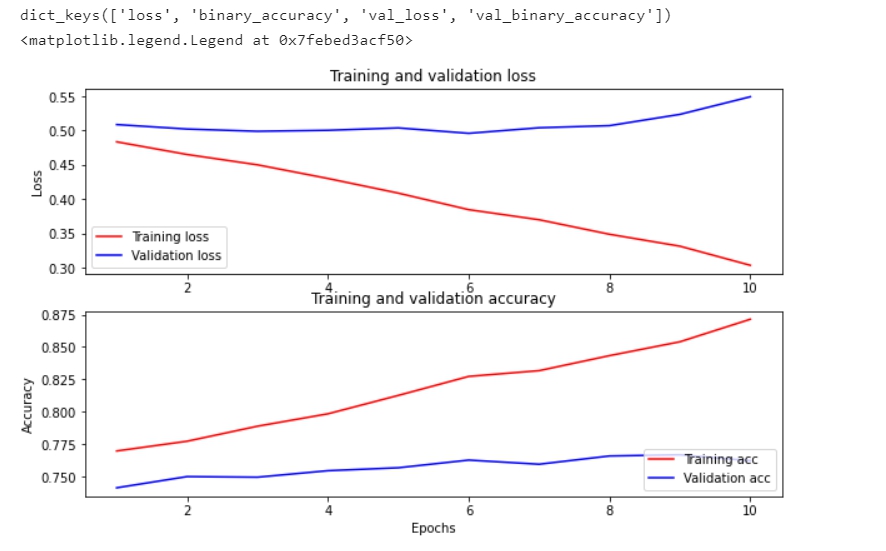
訓練過程:

1. 一般textcnn的訓練準確度如下，可見overfitting情形。(事後推測是沒有lr decay導致)





1. 而BERT的訓練過程圖相對完美，因使用了huggingface之admaw optimzer和超參數，與Google論文之數據相符。詳見繳交notebook的程式和參考連結。



模型架構圖:

紅框為embedding layer，綠框為Kim’s textcnn架構。

