

1. (1%) 請說明你實作的 RNN 的模型架構、word embedding 方法、訓練過程 (learning curve) 和準確率為何？(盡量是過 public strong baseline 的 model)

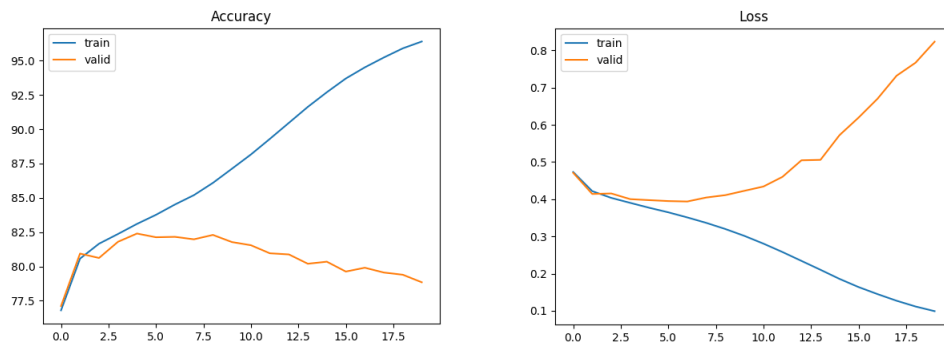
由於這次的模型實在不是很好 train，暴調一波參數還是沒什麼太大的進步，勉強強過 strong baseline，測試過的方式有：(粗體為最終選用的)

Categories	Setting	Selected	Tried	Discussion
Network structure	Different RNN model	GRU	LSTM	GRU 是簡化後的 RNN 模型，比 LSTM 的參數更少，因此會相對較好收斂，實做起來的結果略好於 LSTM
	Different RNN hidden layer dimension	200	250 150 100	一層的 neuron 太少的話，參數量就會少，model 的涵蓋的函式範圍就會較小，但太多參數也會導致不好收斂找到極值，測試下來 200 剛好 (基於 word vector 250 的情況下)
	Different RNN layer	2 layers	1 layer /more than 3 layers	兩層的效果剛好，太深太淺效果都不會比較好
	Different fully connected layer	Shallow (1layer)	Deeper (2~3layer)	實驗下來的結果有點像是前面 RNN 出來後就大致底定了，深淺的 FC 對結果沒有太大幫助，因此選擇淺的，較易收斂
	Different dropout rate	50%	0%or 90%	有 dropout 可以避免 overfitting training set, 50% 剛剛好，太多無義，反而會影響收斂
Optimizer	Different optimizer	Adam with lr=0.0002	Adadelta with lr=1	理論上 Adadelta 在某些情況下收斂的速度非常快效果很好，不過在這個 case 上兩者速度差不多，結果也差不多，adam 略好一點

Training parameters	Different batch size	Small (16/32)	Large (128~1024)	Batch size 會影響到算 gradient 時是一次多少筆 data 一起算 average 的 gradient，在 noisy data 的時候小的 batch size 通常有助於幫助收斂，但運算速度較慢，因為 GPU 算一次矩陣運算的時間不會因為矩陣的 size 變慢或快太多，本次作業的句子分析就屬於比較 noisy 的 data。
	Different sentence segment	Full	Random sample sub-sentence	有鑒於這次即使加了 dropout 還是很容易 overfit training set，因此想仿照 CV 的做法，做 data augmentation 但如果取一個 size 的 window 左右隨機滑動擷取 input，很容易結到全部都是 padding，效果不好，還是都固定從頭開始取，能讓 padding 的影響最小
Word embedding	Different sentence len	32	20 or 39	全部句子中最長的長度是 39，而 32 幾乎涵蓋了大多數的句子長，但又不致於會補太多 padding
	Different padding method	Add <PAD>	Repeat the last word	原本想說與其加一個無意義的<PAD>項不如試試重複字尾看看有沒有效，結果是沒用，會誤導語意，效果更差。
	Different vector dimension	250	200 or 300	根據 gensim 的 word2vec 通常轉換成 200~300 dimension 之間的 word vector 效果會最好
	Different embedding method	Skip-gram	CBOW	兩種不同的 Perdition based 的 word vector

				context，經實驗 sg 較好
	Different window size	5	3 or 7	Window size 也是適中即可，5 剛好。
Semi-supervised (self-training)	Different threshold	10/90	20/80 50/50	使用 self-training 的 hard label，把根據 labeled training data 訓練好的模型拿來預測 unlabel，取分數>0.9 的 label 為 1，<0.1 的 label 為 0。

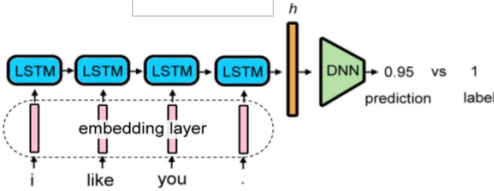
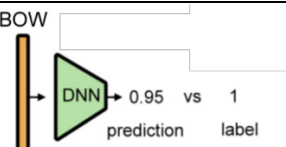
總的來說流程是，使用上述架構和參數先訓練了一個 based on labeled training data (190000/200000 as train, rest as valid)，挑 epoch5 為第一個 model (acc=82.4):



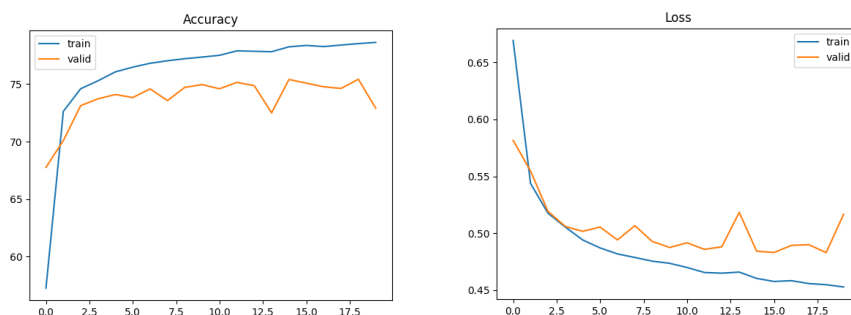
然後再根據這個 model，其他條件一樣把 lr 從 0.0002 降到 0.00005 再 train，挑 epoch2 為第二個 model (acc=82.43)，然後再拿第二個 model 來做 self-learning 得到 93510 比 labeled data，併入，再基於第二個 model 再 train，1~190000&200000~293510 as train，190000~200000 as valid，挑 epoch1 為第三個 model (acc=82.380)，即為最後的 best model acc > strong baseline。

Model	Valid accuracy	Kaggle public acc	
Model 1 (ep5)	82.400	82.165	Train on labeled lr = 0.0002
Model 2 (ep2)	82.430	82.254	Train on labeled lr = 0.00005
Model 3 (ep1) (best model)	82.380	82.321	Train on labeled+new data lr = 0.00005

2. (2%) 請比較 BOW+DNN 與 RNN 兩種不同 model 對於"today is a good day, but it is hot"與"today is hot, but it is a good day"這兩句的分數(過 softmax 後的數值)，並討論造成差異的原因。

Model	Structure & Parameters	today is a good day, but it is hot	today is hot, but it is a good day
RNN based Best model (Acc=82.38 on valid)	 <p>parameter total:6686601, trainable:512601</p>	0.2460	0.8226
BOW+DNN based (Acc=75.42 on valid)		0.7088	0.7000

兩個 model 一個來自第一題的 best model 另外一個為其他參數皆不變，只抽換掉 RNN 的層改為三層的 fully connected layer (故總共四層)，在 labeled training set 上 training 20 個 epoch 後的結果如下：



其實還不會太差，我們選 epoch 作為最終的 model，然後預測上面表格的句子，由結果可知，雖然此 model 也有不錯的 accuracy 但是因為其使用的方法是 DNN，只是胡亂把句子串起來，整串丟進去做 fully connected 算 weight，有點像是一次同時看全部然後下結論，而 RNN based 的卻一個一個 word 餵進 RNN，並有 memory cell 彼此前後影響的，有點像是按照順序從頭到尾看過後才下結論，因此會對文字出現的順序有鑑別度。像上面的實驗就證明了這個現象，先出現 good 再出現 but，就會被 RNN 判斷為 negative，而先有 but 再有 good 則會是 positive，而在 DNN 眼中，因為所有文字都是獨立的，可能 dataset 中有關於 good 的都會是 positive 的，讓他把 good 看得比 but 和 hot 重要，最終分數傾向 positive，但由於 but 和 hot 的扣分，也沒有像 RNN 的 positive 分數那麼高(0.7 v.s 0.8)。不過這個實驗也說明了只抓關鍵字不看語法還是對於這個正面反面語意判斷很有用的，準確度還是有一定的水準。

3. (1%) 請敘述你如何 improve performance (preprocess、embedding、架構等等)，並解釋為何這些做法可以使模型進步，並列出準確率與 improve 前的差異。(semi supervised 的部分請在下題回答)

由第一題我們給的實驗結果可以看出，這次作業主要的問題是 model 很容易 overfitting training data，5 個 epoch 之後 valid 的 loss 就開始變大，acc 變小，因此除了第一題提到的那些調整來幫助 model 更易收斂外，如何減少 overfit 的改動其實才是對 accuracy improve 最多的部分，包含 dropout 和 semi-supervised learning。而關於如何 improve performance 的做法都在第一題的第一個表格中皆有提到，而最後 best model 採用的都是對 accuracy 有 improve 的方法，第一題的第二個表格和下方表格則可看出 improve 前後量化的差異，詳參照第一題。

Model	Valid accuracy	Kaggle public acc
LSTM / 1layer / 150hidden	82.000	82.010
GRU / 2layer / 150hidden	82.225	82.022
GRU / 2layer / 200hidden (model2)	82.430	82.254

4. (2%) 請描述你的 semi-supervised 方法是如何標記 label，並比較有無 semi-supervised training 對準確率的影響並試著探討原因 (因為 semi-supervise learning 在 labeled training data 數量較少時，比較能夠發揮作用，所以在實作本題時，建議把有 label 的 training data 從 20 萬筆減少到 2 萬筆以下，在這樣的實驗設定下，比較容易觀察到 semi-supervise learning 所帶來的幫助)。

在這次作業中我們有使用到 semi-supervised 的方法來得到最終的 best model，詳見第一題的描述與表格，具體的作法我們是使用 self-learning 中的 hard label，先訓練好一個不錯的 model (model2)，用這個 model 來幫 unlabeled data 標記，分數超過 0.9 或低於 0.1 則直接 label 為 1 或 0，然後再把這些新的 labeled data 併入 training data，基於之前那個還不錯的 model (model2) 的參數上再下去訓練，雖然因為 hard label 導致 distribution 更兩極化和 model 之前看過的資料不同，因此 valid loss 會上升，valid acc 也會下降，但加入新的 data 能有效解決 overfit 於 training data 的問題，因此我們可以在 kaggle public 上看到 acc 有明顯的 improve。

而若是在 data 更懸殊的情況，結果如下：(batch size=1024, 其他同第一題)

	Training data	Valid data	Valid accuracy
1. Model train w/o self-learn data	0~20000 labeled data	190000~200000 dabeled data	76.084
2. Model train w/ self-learn data	0~20000 labeled data + 200000~283408 new labeled data	190000~200000 dabeled data	67.197
3. Model train w/ self-learn data based on 1's pretrain	0~20000 labeled data + 200000~283408 new labeled data	190000~200000 dabeled data	69.648

Training Data 的增幅達 500% (20000 -> 103408)，但 self-training 的方法在 validation 上還是沒有很理想，可能是 data 亮的差距還不夠大，也可能是這樣的方法在這個問題上效果不好，可能要採取其他 clustering 的方式來幫 unlabeled data 做更好的分類和 labeling。

