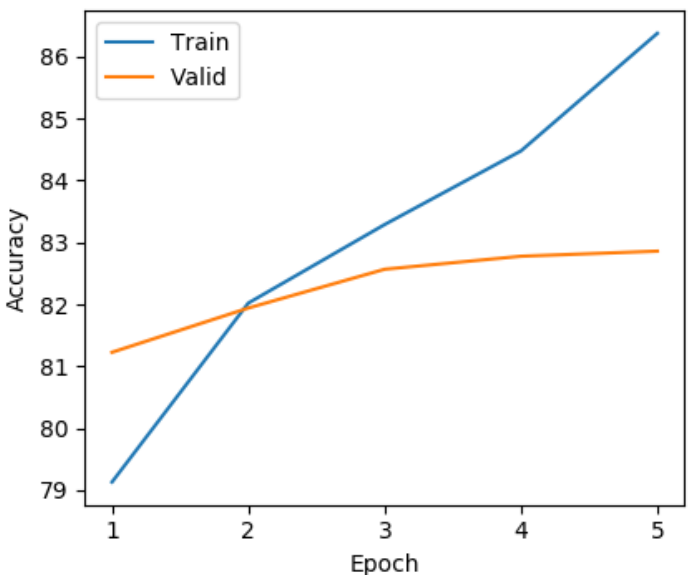
學號： B05902010　系級：資工四　　姓名：張頌平

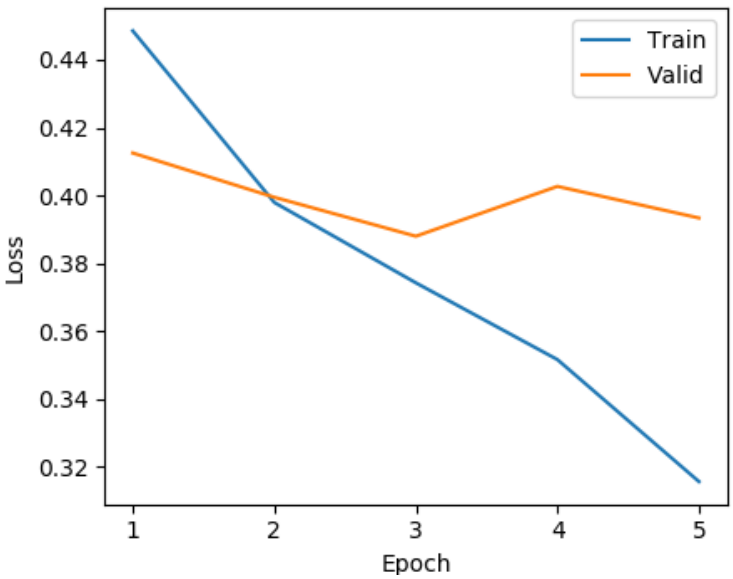
1. (1%) 請說明你實作的RNN的模型架構、word embedding 方法、訓練過程 (learning curve)和準確率為何？ (盡量是過public strong baseline的model)

我使用的模型架構是RNN based的 ***LSTM***，得到output後取最後一個時間點的output當成representation，並且在後面接一層linear classifier和sigmoid後即得到分類的分數；word embedding的方法是將所有training data(包含no_label的data)和testing data都輸入至 ***genism*** 來訓練 word2vec 而後得到word embedding；
訓練參數細節如下：

- training data : validation data = 4:1
- learning rate: 1e-3
- batch size: 32
- dropout: 0.2
- num_layers: 3
- optimizer: first 4 epoch: Adam, last epoch: SGD(lr: 1e-3, cosine annealing scheduler)
- word2vec min count: 8
- word2vec iter: 20
- hidden_dim: 128
- embedding_dim: 128
- Validation Acc: 0.82860 ; Public test set Acc: 0.82941

訓練過程如下圖





***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。**

BOW+DNN Score:
sen1: 0.6405
sen2: 0.6604

RNN Score:
sen1: 0.1212
sen2: 0.9936

從實驗結果發現，RNN 很準確地將兩個句子評出截然不同的分數，但用BOW+DNN算出的兩句子分數十分接近，原因是因為BOW(Bag of words)的表現方式不考慮文法以及詞的順序，只考慮句子中出現了哪些詞和詞的次數，因此在兩句子用的詞一模一樣的情況下，分數自然也就非常接近了；而RNN based的model(這裡使用的是LSTM)在訓練過程中，會將每一個時間點學習到的資訊，再輸入至下一個時間點，在最後一個點的輸出時，是有考慮到從前到後每一個詞的意思和他們之間的前後關係，因此最後結果也比BOW+DNN好上許多。

***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）**

preprocess:
因為每個batch的sequence長度會不同，所以我們在每次訓練時都會做padding的動作，但如果padding的數量太多，就會影響到model在學習時學到不重要的資訊，而如果決定切斷一些內容，可能會導致文意的部分喪失。因此我利用pytorch的pack_padded_sequence，首先先將每個sequence padding成整個batch最長sequence的長度，透過將每個batch的sequence排序過後，並且提供裝有sequence length的list，可以讓model在學習時忽略被padding的部分，這樣的改進讓模型的準確率進步不少。

- Before(Max length: 32): Validation Acc: 0.82445 ; Public test set Acc: 0.82622
- Using pack_padded_sequence: Validation Acc: 0.82765 ; Public test set Acc: 0.82898

word_embedding:
預設的min_count=5, iter=10，因為本次的training data來自twitter，會有滿多錯字，可以預設一些出現比較少的字不太會有特定的含義，因此我將min_count調整為8, 並且調整iter為20(訓練更久)，實際實驗兩者的結果差異不大，可能代表儘管有些typo，但在原先word2vec的訓練當中仍然有學習到其代表的意思。

- Previous word embedding: Validation Acc: 0.82765 ; Public test set Acc: 0.82898
- Now word embedding: Validation Acc: 0.82860 ; Public test set Acc: 0.82941

架構:

架構方面並沒有做太多的改進，只有把LSTM改成Bi-LSTM，透過雙向的訓練能夠讓模型學到更多資訊，而實際訓練後結果也有進步一點點～

- LSTM: Validation Acc: 0.82860 ; Public test set Acc: 0.82941
- BiLSTM: Validation Acc: 0.82927 ; Public test set Acc: 0.83048

Ensemble:

最後有將使用不同架構或是不同data的模型做ensemble (包括LSTM、Bi-LSTM、有加入semi-supervised的LSTM和Bi-LSTM、使用不同的 training/validation的data所訓練的BiLSTM)，ensemble使用的方法為weighted voting，最後上傳至kaggle的accuracy為 ***0.83523*** & 0.83477。

***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）。**

我使用的方法是老師上課有提到的self-training和entropy-based regularization

- self-training: 方法是先將training data訓練過後，對unlabeled data進行預測，如果分數高於0.8的就當作是正面，低於0.2當成負面，並且把其當成training data再訓練一遍，進行3次後輸出test data的答案。使用self-training的方法最後結果並沒有進步，有調整threshold和對於unlabeled data預測的次數，不過最後結果都呈現少許退步，估計原因是像助教所說labeled training data的數量比較多，已經學到夠多內容了，也因為這樣的方式容易使得model更容易overfit(因為那些新增的data本來就是已經預測很好的了)，因此最後結果進步較不顯著(應該說沒進步QQ)。

  準確率比較:
  - Before self-training: Validation Acc: 0.82765 ; Public test set Acc: 0.82898
  - After self-training: Validation Acc: 0.82695 ; Public test set Acc: 0.82874

- entropy-based regularization: 我的方法是將約500000的unlabeled data加入到訓練過程中，並且將他們的entropy加入到每個iteration(每個iteration配3個unlabeled data entropy)的loss當中，對於unlabeled data predict出的結果，若是其分類很明確，entropy會較低(資訊量清楚，loss低)，反之則較高(資訊不清楚，loss高)。weight有嘗試過0.05~0.5，最後以0.1為最佳，在validation set 和public test set 皆有微小進步。

  準確率比較:
  - Before entropy-based regularization:
    Validation Acc: 0.82860 ; Public test set Acc: 0.82941

- After entropy-based regularization:
  Validation Acc: 0.82965 ; Public test set Acc: 0.82990