ADL HW1

B06902104 資工三吳由由

0.1 Q1

- 我使用spacy的en_core_web_sm module進行tokenize, 因為觀察rouge算分數時的預處理會把英文字和數字以外的符號轉變成空格，我也對input做一樣的事，只保留英文字和數字，其他符號忽略，並且依已預處理好的word2index.json來把tokenize好的text, summary改成數字表示，沒有在字典裡的字改以unk表示。每筆testing data以一個dictionary存，dictionary的key有id, text, summary(如果有summary的話)。

- 每次collate_fn準備batch時先得到整個batch中text最長的一筆資料，把資料長度記下來作為這個batch的pad length, 其他同batch中較短的資料則truncate到此batch的pad length，實做方法是在後面全部補0。summary也是和text以一樣的方式truncate。

- 我使用的pre-trained embedding是glove.840B.300d.txt使用gensim讀檔，讀檔前要先用指令python -m gensim.scripts.glove2word2vec –input glove.840B.300d.txt –output glove.840B.300d.w2vformat.txt來轉成gensim能讀的format。我使用gensim將這次訓練有用到的字轉成wordEmbedding.npy，之後訓練預測時可直接取得。

0.2 Q2 Extractive Summarization Model

- Model

  $tag_i = sigmoid(L(tanh(L(LSTM(w_i, h_{i-1}, layer = 5, bidirection, hiddensize = 12)))))$

  where L is linear layer, w_i is the word embedding of token i, h_{i-1} is the hidden state of LSTM on $i-1$ state, tag_i is the probability of token i for binary classification. If $tag_i \geq 0.5$, $tag_i = 1$, else $tag_i = 0$

- Performance on validation set

	rouge-1	rouge-2	rouge-3
mean	0.186	0.028	0.129
std	0.079	0.041	0.057

- Loss function: BCEWithLogitsLoss with pos weight = 6.8145
-
  - Optimization algorithm: Adam
  - learning rate: 0.001
  - Batch size: 32
- Post-processing strategy: 一句當中算1的tag佔整句字數的比例，取ratio最大者為答案，所有同分且最大的句子都選

$$Ratio_s = \frac{\sum_i^{n_s}(tag_i == 1)}{n_s}$$
$$selectSentence = \max_{0 \le s \le m} Ratio_s$$

where n_s is the number of token in sentence s and m is number of sentences in one train/valid data. $Ratio_s$ indicate the ratio of token which is 1 in sentence s. tag_i is the tag which is predicted by the model.

```
ExtractiveTagger(
  (word_embedding): Embedding(97743, 300)
  (lstm): LSTM(300, 12, num_layers=4, batch_first=True, bidirectional=True)
  (l): Linear(in_features=24, out_features=8, bias=True)
  (classifer): Linear(in_features=8, out_features=1, bias=True)
)
```

0.3 Q3 Seq2Seq + Attention Model

- Model
  - Encoder

$$out_i, h_i = GRU(Drop(w_i), h_{i-1}, hiddensize = 250, bidirection)$$
$$hidden = tanh(L(h_{d1}, h_{d2}))$$

where w_i is word embedding for training token i, h_i is hidden state of GRU, $Drop$ is dropout for word embedding, h_{d1}, h_{d2} indicate

last step of two direction of bidirection GRU hidden state. out_i is output of GRU at time i. L indicate linear layer，他將兩個250維共500維的h_{d1} h_{d2}變成250維過activation function tanh後輸出。

- Decoder

$$\begin{aligned} out_i, h_i &= GRU(Drop(w_i), h_{i-1}, hiddensize = 250) \\ attentionWeight &= Attention(h_i, encoderOut) \\ context &= attentionWeight \times encoderOut \\ predict &= L(out_i, context) \end{aligned}$$

where w_i is word embedding for training token i, h_i is hidden state of GRU, $Drop$ is dropout for word embedding. out_i is output of GRU at time i. $attentionWeight$是進入$Attention$計算出後的weight,和encoder out做矩陣乘法得到$context$，幫助deocder model以weight決定輸出predict時要注意在哪些地方上。L indicate linear layer，他將out_i和$context$結合起來投影到corpus總字數維，代表在在這個時間i下所有可能字輸出的機率分佈

- Attention

$$weight = softmax(L(tanh(L(hidden, encoderOut))))$$

where $weight$ is attention weight, L is linear layer, $tanh$ is activation function. $hidden$是decoder的hidden state as query, $encoderOut$是encoder的output as value, 我使用hidden state query value以訓練出好的attention weight讓model注意力放在更合適的字上

- Seq2Seq

$$\begin{aligned} encoderOut, encoderHide &= Encoder(sentence) \\ h_1 &= encoderHide \\ s_1 &=< s > \\ s_i, h_i &= Decoder(s_{i-1}, h_{i-1}, encoderOut) \end{aligned}$$

$sentence$是要訓練的整段文字，送入encoder得到encoder output與encoder最後一刻的hidden state, decoder的hidden state以encoder最後一步的hidden state initialize, 預測出的第i個字為s_i

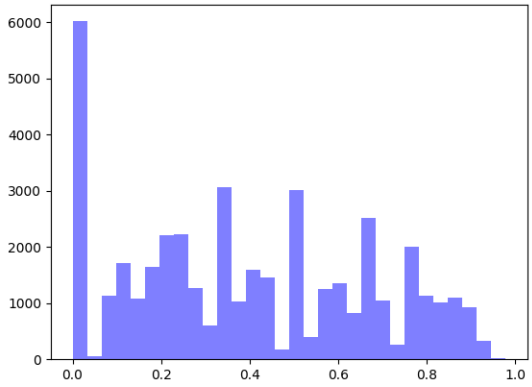
```
Seq2Seq(
  (encoder): Encoder(
    (word_embedding): Embedding(97743, 300)
    (gru): GRU(300, 128, batch_first=True, bidirectional=True)
    (l): Linear(in_features=256, out_features=128, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (decoder): AttDecoder(
    (attention): Attention(
      (attn): Linear(in_features=384, out_features=128, bias=True)
      (v): Linear(in_features=128, out_features=1, bias=False)
    )
    (word_embedding): Embedding(97743, 300)
    (gru): GRU(300, 128, batch_first=True)
    (l1): Linear(in_features=384, out_features=97743, bias=True)
    (dropout): Dropout(p=0.2, inplace=False)
  )
)
```

- Performance on validation set

	rouge-1	rouge-2	rouge-3
mean	0.256	0.071	0.209
std	0.129	0.100	0.120

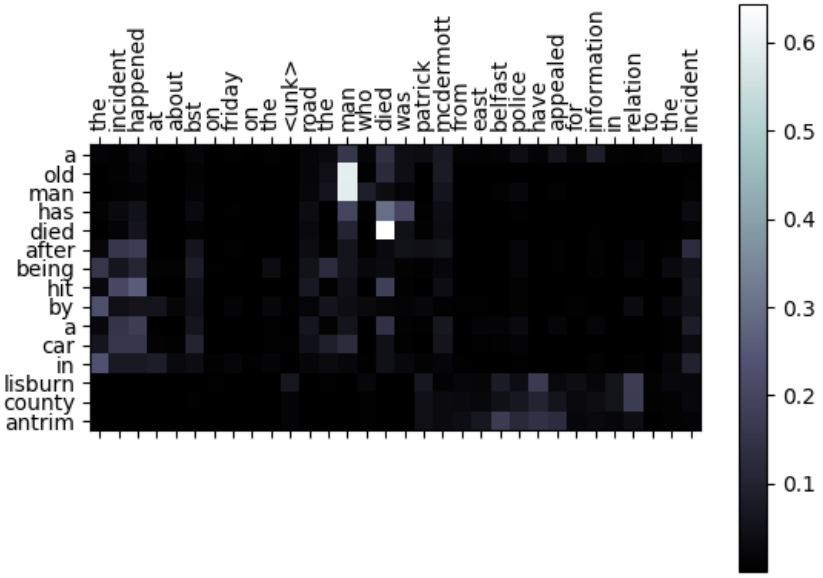
- Loss function: cross entropy loss
-
  - Optimization algorithm: Adam
  - learning rate: 0.001
  - Batch size: 32

0.4 Q4 Distribution of relative location



我發現在句首被預測出為extractive summary的比例比其他位置明顯高很多，而在文章尾端被預測成summary的句子相對很少。很有可能是因為大部分英文文章的開頭就會開門見山的說出重點，中間文句為文章主旨做補充，所以model選為重點時只看開頭跟中間就已經能達到不錯的文意，不會選到文章後面做summary

0.5 Q5 Attention weight



這張圖的橫軸為原文，縱軸為summary,可以發現很明顯的當summary出man這個字時，對應到原文man的attention weight很高，summary出died時原文對應到died的attention weight也很高，所以model基本上抓住原文最關鍵的summary(man died)。原文提到的是incident happened on road而model自動產生的summary是hit by a car雖然原文沒提及此事，但model預測出的句子很合理，是個很不錯的預測。此外，原文的unk出現時，得到的attention weight全黑，也代表model學到不要把注意力放在unk上

0.6 Q6 Rouge-L

Rouge L計算的是longest common subsequence, 使用DP方法計算最長共同子序列，首先先把dp table初始成0, row的字代表正確答案reference，column的字代表預測出的summary candidate

```
for i in range(1, N):
 for j in range(1, M):
  if reference[i-1] == candidate[j-1]:
```

```
    table[i][j] = table[i-1][j-1]+1
  else:
    table[i][j] = max(table[i-1][j], table[i][j-1])
longestLength = table[-1][-1]
precision = longestLength/len(prediction token)
recall = longestLength/len(target)
fscore = 2*precision*recall/(precision+recall)
return fscore
```

當字match到，就是現在有match到的最長值加1
當字沒match到，就選目前為止match到的最長值
longest common subsequence的答案為表格最右下角的值
precision代表預測出來的字中有多少是正確的
recall代表正確答案中有多少被預測出來
得到fscore最終答案