Part 0

選用的方法: Lemmatization

```
# Lemmatization

lm = WordNetLemmatizer()
s_list=[]
for s in sentences:
    w = s.split(' ')
    for i in range(len(w)):
        lemma = lm.lemmatize(w[i],'v')
        if lemma == w[i]:
            lemma = lm.lemmatize(w[i],'n')
        s_list.append(lemma)

filtered_token = [token for token in s_list if token.lower() not in stop_word_list]
lemmatization_text = ' '.join(filtered_token)
```

先將 sentence 中的字 split 出個別詞彙，運用 Lemmatization 將 split 分別根據動詞(v)或名詞(n)提煉出他們的詞幹 lemma。最後再將提煉出來的詞幹透過 stop_word_list 來篩選。

Part 1

- Perplexity

$$perplexity(S) = p(w_1, w_2, w_3, \cdots, w_m)^{-1/m}$$
$$= \sqrt[m]{\frac{1}{p(w_1, w_2, w_3, \cdots, w_m)}}$$
$$= \sqrt[m]{\prod_{i=1}^{m} \frac{1}{p(w_i | w_1, \cdots, w_{i-1})}}$$

用來衡量語言模型的指標。所以如果 Perplexity 越小，句子的機率就越高，而越通順的語句在語言模型中便會給出越高的機率。因此 Perplexity 越小代表我們的 model 的表現越好。

- Screenshot:

Without preprocess

```
Perplexity of ngram: 116.26046015880357
F1 score: 0.3662, Precision: 0.6233, Recall: 0.5108
```

With remove_stop_word

```
Perplexity of ngram: 195.43245350997685
F1 score: 0.3371, Precision: 0.5895, Recall: 0.501
```

With Lemmatization

```
Perplexity of ngram: 186.29833910794946
F1 score: 0.342, Precision: 0.6325, Recall: 0.5029
```

在沒有 preprocess 的情況下，含有一些連接詞主詞等等，比較貼近通順的語句，perplexity 比較小。
而在 remove_stop_word 跟 lemmatization 的 perplexity 比較大，但整體來說兩者沒有太大的差別。

Part 2

- Pre-training steps

  Masked token prediction

  將 input 取 tokens 後 randomly masked，並決定要用 special tokn 或是隨機挑字來蓋住。之後將輸入的向量做 linear 和 softmax 後來預測結果。這部分就是在 pretrain 中，Bert 要訓練成功預測自己蓋住文字的類別。

  Next sentence prediction

  將包括分段符號的句子頭加入[CLS]放入模型中，將[CLS]的輸出乘上 linear transform 來做二元分類，預測這句子間是否有相接（輸出 yes/no）。

- 4 Bert application scenarios

  Case 1

  Input 為 sequence，output 為 class。例如：Sentiment analysis，將句子 tokens 加上[CLS]丟入後，預測整句是 positive 還是 negative (class).

  Case 2

  Input 為 sequence，output 和 input 一樣都為 sequence，長度也相同。例如：POS tagging。將句子輸入後的 token，預測每個字為哪個詞性的類別。

  Case 3

  Input 為 two sequence，output 為 class。例如：NLI，給機器兩個句子（前提跟假設），機器預測前提是否能推導出假設。

  Case 4

  有限制的問答系統，input 為文章跟問題，機器 output 出 integer (s,e) 來代表正確答案會對應文章中第 s 到 e 的字。

- Diff between Bert and distilBERT

  DistilBERT 在 pretrain 階段對比 Bert 會透過 knowledge distillation 減少 40% 模型的大小。除此之外會保留 Bert 的 language understanding abilities，比 Bert 快 60%。

  DistilBERT is cheaper, lighter and faster than Bert.

- ScreenShot

```
5000it [28:06,  2.96it/s]
10000it [01:41, 98.90it/s]
Epoch: 1, F1 score: 0.9234, Precision: 0.9235, Recall: 0.9234, Loss: 0.2747
```

Part 3

- Diff between RNN and LSTM

  RNN 沒有 cell 來記憶。因此訓練中 RNN 的隱藏層訊息來自於當前輸入和上一層的訊息，當輸入較長時會有長期依賴、梯度消失的問題。

  LSTM 有 cell 的幫助，用了 input, output 和 forget gate 來保存、處理訊息，forget gate 可以用來捨棄前一層不需要的舊訊息，cell 可以運用 input, forget gate 整理保存訊息，進而產生訓練有記憶的功能，可以處理長期和短期的依賴問題。

- Model meaning

```
self.embeddings = nn.Embedding(n_vocab, embedding_dim)
self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers, dropout = 0.2)
self.hidden2out = nn.Linear(hidden_dim, num_labels)
# END YOUR CODE
```

  1. Embedding layers

     Input:

     first dim: Sequence length

     second dim: word length

     Output:

     First dim: Sequence length

     Second dim: embedding numbers of features

  2. LSTM layers

     Input:

     First dim: Sequence length

     Second dim: embedding numbers of features

     Output:

     First dim: Sequence length

Second dim: output features number

3. Linear

   Input:

   First dim: Sequence length

   Second dim: output features number

   Output:

   First dim: Sequence length

   Second dim: the result class

- Screenshot

```
Epoch: 9, F1 score: 0.7592, Precision: 0.7607, Recall: 0.7595, Loss: 0.5478
5000it [00:33, 150.95it/s]
10000it [00:21, 472.85it/s]
Epoch: 10, F1 score: 0.7631, Precision: 0.7635, Recall: 0.7632, Loss: 0.5355
```

Discussion

- Innovation of the NLP field &Why ngram -> LSTM -> BERT.

  Innovation:

  NLP 一直都會將 textual data 轉換至 embedding，近幾年會用詞彙出現的 frequency 來找他們相對應的 vectorized representation。

  Why ngram -> LSTM -> BERT

  我覺得為什麼會有這樣的進化是因為一開始 ngram 只是單純透過移動窗口的大小來算詞彙會出現的機率，但當詞彙量更大的時候，這種比較沒效率的方法就會產生計算量跟參數過多的問題。因此後來出現了 LSTM，可以透過記憶的 gate 跟 cell 狀態保持裡面的參數跟調整新的狀態，但主要的問題是對於 model 訓練上一定要有資料的 label，所以後來便演進成 BERT 能 self supervised learning 的模型。

- Problems & solutions

  一開始有出現 RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking argument for argument index in method wrapper_CUDA__index_select)

  後來發現是 test_input 跟模型不在同樣的 device 上，只要 test_input.to(‘cuda’)就解決了。