Q1:Data processing

A:我使用的是 sample code，在 preprocess_intent 中，建立一 intents set 收集所有出現過的 intent，再將 intents 記錄成一個 dict,其在 intents 中的 index 為 value，intents 中的 value 為 dict 的 key，再將其轉成”intent2idx.json”儲存，完成對 intent 的 tokenize。
再來對 text 建立 vocab，建立一 words counter，每個 vocabulary 為一個 token，使用 words 紀錄 data 中所出現過的 token 及其出現次數，再依 token 出現次數多寡排順序存入 vocab，建立一 token_dict，token_dict 中 token 為 token_dict 的 key，token 的進入順序為在 token_dict 中的 value，設定 token_dict 中”Vocab.PAD”為 0，token_dict 中” Vocab.UNK”為 1，前者為用於 pad 的 value 後者為用於出現 unknown token 的 value，之後的數字才是用於對 token 進行 tokenize，再根據 token_dict 對 text 進行 tokenize。
在 preprocess_slot 中，建立一 tags set 收集所有出現過的 tag，再將 tags 記錄成一個 dict,其在 tags 中的 index 為 value，tags 中的 value 為 dict 的 key，再將其轉成”tagt2idx.json”儲存，完成對 tag 的 tokenize。
再來對 tokens 建立 vocab，建立一 words counter，每個 vocabulary 為一個 token，使用 words 紀錄 data 中所出現過的 token 及其出現次數，再依 token 出現次數多寡排順序存入 vocab，建立一 token_dict，token_dict 中 token 為 token_dict 的 key，token 的進入順序為在 token_dict 中的 value，設定 token_dict 中”Vocab.PAD”為 0，token_dict 中” Vocab.UNK”為 1，前者為用於 pad 的 value 後者為用於出現 unknown token 的 value，之後的數字才是用於對 token 進行 tokenize，再根據 token_dict 對 tokens 進行 tokenize。
我使用 glove 進行 pre-trained embedding。

Q2:intent classification model

a.

```
SeqClassifier(
  (embed): Embedding(6491, 300)
  (rnn): LSTM(300, 512, num_layers=2, batch_first=True, dropout=0.1, bidirectional=True)
  (lin): Linear(in_features=512, out_features=150, bias=True)
)
```

我的 model 首先將 data 放入 embedding 層,embedding 層是使用 glove 的 pre-trained weight 對 data 進行 embedding，data 中的每個 token 會轉換成 300 維輸出。再將轉換後的 data 輸入 LSTM 層，LSTM 層的 input_size 為 300 維，hidden_size 為 512 維，此為 hidden_state 的維度，其餘設定如上圖所示。

```
out, (h_n, c_n) = self.lstm(inputs, None)
```

上圖為 lstm 在 pytorch 中定義的模型，out 存有在各個時間 t 的 hidden_state 資訊，h_n 為在最後一個時間 t 的 hidden_state 資訊，c_n 為在最後一個時間 t 的 cell state 資訊。在我的 model 中取用 h_n 輸入 linear 層，期望得到最

後一個時間 t 的結果用以預測 intent，輸出為 150 維，相對應於 intent 的種類數量，其值代表猜測對應到此維度的 intent 的機率，我取用最大機率值的維度作為猜測結果。

b. 0.91733

c. loss function 為 crossentropy

d. optimization alg. 為 Adam，搭配 onecycleLR scheduler，learning rate 為 0.001，batch size 為 128，得到最低 loss 的 model 後，再對 model re-train，optimization alg. 為 Adam 搭配 ReduceLROnPlateau，learning rate 為 0.00001，batch size 為 64。

Q3：slot tagging model

a.

```
SeqTagger(
  (embed): Embedding(4117, 300)
  (rnn): LSTM(300, 512, num_layers=2, batch_first=True, dropout=0.1, bidirectional=True)
  (lin): Linear(in_features=1024, out_features=9, bias=True)
)
```

我的 model 首先將 data 放入 embedding 層，embedding 層是使用 glove 的 pre-trained weight 對 data 進行 embedding，data 中的每個 token 會轉換成 300 維輸出。再將轉換後的 data 輸入 LSTM 層，LSTM 層的 input_size 為 300 維，hidden_size 為 256 維，此為 hidden_state 的維度，其餘設定如上圖所示。

```
out, (h_n, c_n) = self.lstm(inputs, None)
```

上圖為 lstm 在 pytorch 中定義的模型，out 存有在各個時間 t 的 hidden_state 資訊，h_n 為在最後一個時間 t 的 hidden_state 資訊，c_n 為在最後一個時間 t 的 cell state 資訊。在我的 model 中取用 out 的部分內容輸入 linear 層，out 取用的部分為 data”tokens”中的 token 所相對應的 hidden_state，期望得到每個 token 對應到其時間 t 的結果用以預測 tags，即為取用前 l 個 hidden_state，l 是 data”tokens”的長度。Linear 層輸出為 9 維，相對應於 tags 的種類數量，其值代表猜測對應到此維度的 tags 的機率，我取用最大機率值的維度作為猜測結果。

b. 0.79034

c. loss function 為 crossentropy

d. optimization alg. 為 Adam 搭配 onecycleLR scheduler，learning rate 為 0.0001，batch size 為 64

Q4:sequence tagging evaluation

	precision	recall	f1-score	support
date	0.76	0.75	0.75	206
first_name	0.90	0.89	0.90	102
last_name	0.76	0.67	0.71	78
people	0.74	0.74	0.74	238
time	0.87	0.86	0.86	218
micro avg	0.80	0.78	0.79	842
macro avg	0.81	0.78	0.79	842
weighted avg	0.80	0.78	0.79	842

Token accuracy 為 True_pred_tokens/total_tokens，True_pred_tokens 為成功預測 token 所對應的 tag 的總數，total_tokens 為所有 Text 中 Token 的總數。

joint accuracy 為 True_pred_texts/Total_Test，True_pred_texts 為成功預測 text 中每個 token 所對應的 tag 的 text 的總數，Total_Test 為 Text 的總數。

Seqeval 將準確度分為三種指標，分別為 precision、recall、f1-score，將預測狀況分為三種，分別為 true positive(tp)、false positive(fp)、false negative(fn)，seqeval 視 entity 為單位，一個 entity 為一個從 B 開頭的 tag 開始(如:B-first_name、B-people、B-date)且相同類型(如：first_name、people、date) 的非 O 的連續 tags 序列，我們再根據不同類型的 entity 進行統計進而計算 metrics。

tp 代表預測出現且存在的 entity，fp 代表預測出現但錯誤的 entity。fn 代表預測未出現但存在的 entity。tp 總數+fn 總數即為 true_label 中的 entity 總數，tp 總數+fp 總數即為 pred_label 中的 entity 總數。

Precision 的公式為 tps/(tps+fps)，tps 代表 the number of tp，fps 代表 the number of fp。Recall 的公式為 tps/(tps+fns)，fns 代表 the number of fn。f1-score 的公式為 2*(precision*recall)/(precision+recall)。

Seqeval 的三種 metrics 相較於 token accuracy 及 joint accuracy 能更進一步了解到模型的準確度，我們能知道不同類型的 entity 的預測狀況。我們能透過 precision 了解到 model 是否能夠精準地預測到此類型的 entity，了解模型在預測 entity 時的精確度高低，誤判的情況是否常發生。能透過 recall 了解到 model 是否能夠找到 text 中所有存在的此類型的 entity。f1-score 則是代表 precision 及 recall 的綜合指標。

Seqeval 相較於 token accuracy 及 joint accuracy 能夠更準確地顯示 model 是否成功學習到一種類型的 entity，是否真正學到如何辨別一種類型的字詞(如:first name、people、date)。

Q5:different configurations

我選擇 Q2 的改善方法回答。

我先對 hyperparameter 設定預設值以利接下來做測試比較。

dropout=0.5、hidden_size=256、num_layers=2、lr=0.001、batch_size=128、num_epoch=50、optimizer=Adam、scheduler=None、max_len=128。

一開始我先比較 LSTM 及 RNN 兩者模型的效果，在相同的 hyperparameter 下，LSTM 相較於 RNN 需要更長的學習時間，但 LSTM 的收斂速度快於 RNN 且 LSTM 能得到的 best valid loss 相較於 RNN 更低。我選擇 valid loss 最低的 model 作為最佳的模型，在 valid_acc 上 LSTM 是 0.920333 ，RNN 是 0.90977，因此我選擇 LSTM 作為我的模型繼續調整 hyperparameter。

選擇好模型後，我先對 dropout 做調整，我選擇 0.1、0.2、0.5 做測試，我發現在不同 dropout 下會影響 train loss 的收斂速度，dropout 越大收斂速度越慢，因此在較大 dropout 下可降低 overfitting。在觀察 valid loss 及 train loss 的收斂速度後，我發現在 dropout=0.5 時兩者的收斂速度最接近，基於此因素我選擇 dropout=0.5 以降低 overfitting 的機率。

選擇好 dropout 後，我對 hidden size 做調整，我選擇 256、512 做測試，我發現在學習速度上 256 維快於 512 維，這是因為參數多寡的因素，而在 loss 上的表現 512 維的收斂速度慢於 256 維，但 512 維的 valid loss 與 train loss 在差距上小於 256 維，因此我打算之後對 256 及 512 維都進行測試。

之後的討論先以 256 維為基準，在 512 維上也會做一樣的測試 ，只在最後比較兩者的結果。

選擇 hidden_size 後，我對 batch size 進行測試，我選擇 32、64、128 做測試，在學習速度上 batch_size 越小其速度越慢，在 train loss 及 valid loss 的收斂速度也是 batch_size 越小則速度越慢，但在 valid_loss 上 32 收斂到三點多時已經很難再下降，64 則是收斂到二點多時已經很難再下降，128 則是收斂到零點多時還能繼續下降。基於此因素，我選擇 batch_size=128 進行訓練。

選擇好 batch size 後，我對 lr 進行測試，我選擇 0.01、0.001、0.0001 進行測試，在 0.001 及 0.001 時，兩者都會掉入 local optimum 而且跑不太出來，因此我選擇 lr=0.01 進行訓練。

選擇好以上的參數後，我分別對 hidden_size=256 及 hidden_size=512 進行訓練，hidden_size=256 得到了更低的 valid loss=0.9731，hidden_size=512 的 valid loss=1.073，我將此兩個 model 的 parameter 保存下來做 re-train。

基於以上兩個模型，我在 kaggle 上得到了 0.91066 及 0.90977。

我根據此兩個模型進行 re-train。

我將這兩個 model parameter 進行 re-train，設定 num_epoch=10、lr=0.0001 進行 re-train，希望能夠得到更低的 valid loss 使預測更加準確。這兩個模型在 kaggle 上得到了 0.91422 及 0.91777。

基於以上的設定，我加入了 scheduler 進行測試，在測試過程中，我分別測試了

MultiStepLR,OneCycleLR,StepLR,ReduceLROnPlateau,ExponentialLR,CosineAnnealingLR。

最後我選擇了 OneCycleLR 作為第一次訓練時的 scheduler，選擇 ReduceLROnPlateau 作為 re-train 的 scheduler。在加入 scheduler 訓練後我發現 loss 的收斂速度相較於沒有 shceduler 更慢且更加不平滑，這是基於 OneCyleLR 對學習率的調整策略，它會將 learning rate 以 cycle 為循環調整其高低，使 model 不會掉入 local optimum 也不致於 learning rate 過高學不到 optimum。我認為加入此 scheduler 能夠降低 overfitting，提高在 kaggle 的表現。

在 re-train 我選擇了 ReduceLROnPlateau，設定 patience=2，會在連續兩個 epoch 沒有降低 valid_loss 時降低學習率，我認為這個 scheduler 能夠幫助我再更進一步加強 re-train 帶來的效果。

基於以上兩個模型，我在 kaggle 上得到了 0.91866 及 0.91733。

在其他優化策略上，我測試過加入 warm-up 但並沒有得到更低的 valid_loss。