Q1: Data processing:

a. How do you tokenize the data.

Intent classification:

首先把 training data 的 text 一個字一個字讀進來(split 空格)，然後做成一個 2 dim 的 list。 如下所示:

```
[['i', 'have', 'three', 'people', 'for', 'august', 'seventh'], ['do', 'you', 'have', 'highchairs', 'for', 'my', '4', 'kids'],
 'and', '4', 'children', 'please'], ["i'd", 'like', 'information', 'for', 'the', 'royal', 'george', 'strand', 'restaurant', '
 'have', 'a', 'booking', 'tomorrow', 'for', 'chara', 'conelly', 'at', '9pm'], ['can', 'i', 'have', 'the', 'phone', 'number',
 'for', 'restaurants', 'which', "don't", 'serve', 'hawaiian', 'food', 'is', 'not', 'a', 'tea', 'restaurant', 'and', 'the', 'p
 ['how', 'busy', 'are', 'you', 'at', '9ish'], ['my', 'current', 'reservation', 'is', 'at', '05:30pm'], ['last_name', 'last_na
 'want', 'a', 'restaurant', 'in', 'the', 'expensive', 'price', 'range', 'not', 'cheap'], ['our', 'party', 'will', 'consist',
 'cuisine', 'and', 'the', 'price', 'range', 'should', 'be', 'moderate', 'and', 'not', 'cheap'], ['i', 'want', 'an', 'outside'
  'of', 'march'], ['i', 'want', 'wondering', 'if', 'i', 'could', 'come', 'in', 'ten', 'minutes'], ['time', '20hr'], ['do', 'y
 'and', 'a', 'child', 'still', 'available'], ['thank', 'you', 'bye!'], ['11:30'], ['i', 'made', 'a', 'reservation', 'for', 'a
 ['can', 'i', 'book', 'it', 'for', '1:30pm'], ['i', 'want', 'to', 'make', 'a', '11:00am', 'reservation'], ['do', 'they', 'hav
 'under', 'the', 'name', 'melita', 'tscrious', 'for', '2', "it's", 'for', '8:00'], ['people', 'big', 'italian', 'family', 'he
```

由於要將 input size 固定，我將句子長度固定為 14 字，削長補短(補”<ED>”)，如下圖所示:

```
['do', 'you', 'know', 'any', 'fun', 'facts', 'about', 'shampoo', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'], ['can',
['go', 'into', 'whisper', 'mode', 'now', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'], ['can',
, ['could', 'you', 'tell', 'me', 'what', 'my', 'checking', 'account', 'balance', 'is', '<ED>', '<ED>', '<ED>', '<ED>'], ['
'<ED>', '<ED>'], ['can', 'you', 'set', 'up', 'a', 'pto', 'request', 'for', 'me', 'for', 'march', '20th', 'to', 'april'],
'<ED>'], ['i', 'need', 'a', 'pto', 'request', 'for', 'march', '3rd', 'and', '4th', '<ED>', '<ED>', '<ED>', '<ED>'], ['what
'<ED>'], ['how', 'can', 'i', 'rollover', 'my', '401k', 'at', 'my', 'new', 'job', '<ED>', '<ED>', '<ED>', '<ED>'], ['tell',
'<ED>'], ['how', 'long', 'should', 'i', 'wait', 'to', 'get', 'my', 'new', 'card', 'when', 'my', 'old', 'one'], ['please',
'<ED>'], ['set', 'reminder', 'to', 'feed', 'cat', 'tonight', 'at', '6pm', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'],
'<ED>'], ['not', 'really', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>']
'<ED>', '<ED>'], ["what's", 'the', 'correct', 'fuel', 'type', 'for', 'my', 'car', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>',
```

再將 training data 中有出現的字，寫成一個 dict ，dict 的 key 為 training data 中有出現的字，dict 的 value 為用 key 從 glove.840B.300d 中找出對應的 300 dim 向量，如果此字沒有在 glove.840B.300d 中出現過，則補 300 dim 的 zeros。如此一來我就從 glove.840B.300d 中做出，資料量較小的字典並存成 json 檔。

如下圖所示:


```
"i": [0.18733, 0.40595, -0.51174, -0.55482, 0.039716, 0.12887, 0.45137, -0.59149, 0.15591,
.27212, 1.6203, -0.24884, 0.1406, 0.33099, -0.018061, 0.15244, -0.26943, -0.27833, -0.052123,
84, 0.18262, -0.34541, 0.082611, 0.10024, -0.07955, -0.81721, 0.0065621, 0.080134, -0.39976,
 -0.088653, -0.29087, -0.047214, 0.046036, -0.017788, 0.06499, 0.088451, -0.31574, -0.58522,
1722, -0.55576, 0.088707, 0.1371, -0.0029873, -0.026256, 0.07733, 0.39199, 0.34507, -0.08013,
0.45012, 0.027179, 0.274, 0.14791, -0.0058324, 0.9591, -1.0129, 0.20699, 0.18237, -0.25234,
0.48327, 0.089523, -0.22373, -0.15654, 0.21578, 0.11673, 0.082006, -0.80735, 0.23903, -0.51304,
, 0.021233, 0.01335, -0.063938, -0.24957, -0.24938, 0.34812, -0.071321, 0.23375, -0.095384,
0.1576, -0.59125, 0.243, 0.63962, -0.09328, -0.27914, -0.066262, -0.06717, -0.40929, -3.03,
-0.34231, -0.63766, -0.36129, -0.059029, 0.1551, 0.044577, 0.23572, -0.17095, -0.22749,
, -0.31405, -0.085287, -0.33496, -0.097047, -0.14388, 0.11147, -0.45232, -0.24217, -0.18245,
9, 0.11817, 0.056851, -0.49151, 0.15496, 0.016425, 0.04165, -0.3499, -0.15979, 0.39705, 0.22963,
-0.78653, -0.061379, -0.37359, -0.11603, -0.2495, 0.10161, 0.033994, 0.1565, 0.21344, -0.11094,
239, 0.042514, 0.1185, -0.18337, -0.62865, -0.28021, 0.42351, 0.11277, 0.0012121, 0.1571,
0.29877, -0.012071, 0.28325, 0.10668, -0.18859, -0.41345, -0.3409, 0.047236, -0.38309, 0.43572,
, 0.19433, -0.1523, 0.42675, 0.28795, -0.55969, -0.13031, 0.089844, 0.42605, -0.19632, -0.071989,
3, -0.036548, -0.36739, -0.019819, 0.3213, 0.17479, 0.25175, -0.0076439, -0.093786, -0.37852,
, 0.20625, -0.037701, -0.122, -0.079253, -0.1029, 0.010558, 0.4988, 0.25382, 0.15526, 0.0017951,
16495, 0.18757, 0.53874], "need": [0.1206, 0.14264, -0.15579, -0.010927, -0.0035145, -0.08422,
6939, -0.43365, -0.11577, 0.091258, 1.8239, -0.44836, 0.35135, 0.079855, -0.26808, 0.20348,
1, -0.25098, 0.18022, 0.043457, 0.15591, 0.26119, 0.043879, 0.12919, 0.16836, -0.020581,
```


最後在把 text data input 到 deep learning model 中 train 時，要把每個文字都去查我做好的字典(json 檔)將每個字轉成 300 dim 的向量，因此最後送入 model 去訓練的 input data 維度為 [total training 資料筆數, 14(自設句子固定長度), 300] 的三維資料， 而在 y label data 的處理上，我將 training data 每筆的 intent 抓出來做成一個 dict，dict 的 key 為每個不相同的 intent(統計共 150 種)，dict 的 value 為 0~149 依序編碼。如下圖所示:

```
"tire_change": 0,
"replacement_card_duration": 1,
"pay_bill": 2,
"book_hotel": 3,
"roll_dice": 4,
"calendar": 5,
"report_lost_card": 6,
"reminder_update": 7,
"shopping_list": 8,
"smart_home": 9,
"w2": 10,
"goodbye": 11,
"traffic": 12,
"jump_start": 13,
"card_declined": 14,
"shopping_list_update": 15,
"tire_pressure": 16,
"accept_reservations": 17,
"transfer": 18,
"credit_score": 19,
"travel_notification": 20,
"improve_credit_score": 21
```


如果我今天設定 batch_size=16，則每個 step 送入 model 的 x_data 維度為[batch_size, 14, 300]，y_data 維度為[batch_size, 1]。

Slot tagging:

因為 input size 要一致，與 intent classification 做法不同這邊設置的句子長度為 training data 和 test data token 的最大字數，因為每個字都要 output 一個 tags，經由程式得知為 35，不足的補”<ED>”，如下圖所示:

```
['i', 'know', 'i', 'want', '9', 'people', 'to', 'be', 'here', '<ED>', '<ED>', '<
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '
 '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>',
<ED>', '<ED>', '<ED>'], ['any', 'inside', 'tables', '<ED>', '<ED>', '<ED>', '<ED
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'], ['i', 'need', 't
'the', 'name', 'noble', 'seipel', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'
['have', 'you', 'got', '7pm', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '
1:45', 'am', 'under', 'rudolf', 'helmich', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>
<ED>', '<ED>', '<ED>', '<ED>', '<ED>', '<ED>'], ['people', 'party', 'size', 'is.
```

在將這些字做成 dict，dict 的 key 為 training data 有出現過的字，dict 的 value 為 training data 的字透過查詢 glove.840B.300d 將其 300 維向取出，如果此 training data 字沒有在 glove.840B.300d 出現過，則用 300 dim 的 zeros 代替。做完如下圖所示:

```
 "<ED>": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
).0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], "do": [-0.014531,
.070334, -0.22503, 2.4932, -0.36484, -0.14024, 0.31678, 0.060765, -0.48319, -0.098549
389, 0.091197, -0.36923, 0.14988, -0.139, -0.21428, 0.67552, -0.29393, -0.043688,
16852, -0.12807, -0.46307, -0.41046, 0.22182, -0.01718, -0.066614, 0.1303, 0.063694,
55215, -0.15037, 0.48806, -0.093592, 0.19344, -0.13372, 0.22679, -0.058024, 0.046199,
```

此方法與 intent classification 做法相同，為了避免在 load glove.840B.300d 時因為檔案太大所造成耗時，因此做一個較小的且 training 有用到的 word embedding。而在 y label 上，我們多加了一個 class 叫”<ED>”為了使我們在 x data 中多補上”<ED>” tags output 也為”<ED>”， y 字典如下

```
{
    "I-time": 0,
    "B-people": 1,
    "O": 2,
    "B-last_name": 3,
    "B-first_name": 4,
    "B-time": 5,
    "I-people": 6,
    "I-date": 7,
    "B-date": 8,
    "<ED>": 9
}
```

最後 training 我們送入的 x data 維度為[total training 資料筆數, 35(max 長度), 300]，而每個 training step 維度為[batch_size, 35, 300]，y_data 維度為[batch_size, 35]。

b. The pre-trained embedding you used.

glove.840B.300d

Q2:Describe your intent classification model.

a. Your model

nn.LSTM(input_size=300, hidden_size=2048, num_layer=2, batch_first=True, bidirection=False)+nn.Linear(2048, 150)，把 input data [batch_size, 14, 300] 送入 LSTM model 出來的 output 為 [batch_size, 2048]，在將其送入一個 liner layer 做 classification 最後 output 為 [batch_size, 150]，150 為每個種類預測機率值，取 max 就會是 predict 出來的 class。

b. performance of your model.

Kaggle score: 91.33

c. the loss function you used.

```
loss_function = torch.nn.CrossEntropyLoss()
```

$$H = \sum_{c=1}^{C} \sum_{i=1}^{n} -y_{c,i} log_2(p_{c,i})$$

C 是類別數
n 是所有的資料數
yc,i 是 binary indicator (0 or 1) from one hot encode
pc,i 是第 i 筆資料屬於第 c 類預測出來的機率

d. The optimization algorithm (e.g. Adam), learning rate and batch size.

```
optimizer = torch.optim.AdamW(lstm.parameters(), lr=LR,
```

LR=0.001
Batch_size=64
Epoch=180

Q3:Describe your slot tagging model.

a. Your model

nn.LSTM(input_size=300, hidden_size=256, num_layer=2, batch_first=True, bidirection=True)+nn.Linear(256*2, 10)，把 input data [batch_size, 35, 300] 送入 LSTM model 出來的 output 為 [-1, 256*2]，在將其送入一個 liner layer 做 classification 最後 output 為 [-1, 10]，10 為每個種類預測機率值，取 max 就會是 predict 出來的 class，這裡多一種 class，是因為我將補”<ED>”湊成固定 35 長度 input size 多設一種類。

b. performance of your model.

Kaggle score: 81.98

c. the loss function you used.

```
loss_function = torch.nn.CrossEntropyLoss()
```

$$H = \sum_{c=1}^{C}\sum_{i=1}^{n} -y_{c,i}log_2(p_{c,i})$$

C 是類別數

n 是所有的資料數

yc,i 是 binary indicator (0 or 1) from one hot encode

pc,i 是第 i 筆資料屬於第 c 類預測出來的機率

d. The optimization algorithm (e.g. Adam), learning rate and batch size.

```
optimizer = torch.optim.AdamW(lstm.parameters(), lr=LR,
```

LR=0.001

Batch_size=32

Epoch=120

Q4: Sequence Tagging Evaluation

report classification_report(schema=IOB2)

	precision	recall	f1-score	support
date	0.77	0.75	0.76	206
first_name	0.98	0.93	0.95	102
last_name	0.95	0.79	0.87	78
people	0.73	0.75	0.74	238
time	0.86	0.83	0.85	218
micro avg	0.82	0.80	0.81	842
macro avg	0.86	0.81	0.83	842
weighted avg	0.82	0.80	0.81	842

	實際 YES	實際 NO
預測 YES	TP (True Positive)	FP (False Positive) Type I Error
預測 NO	FN (False Negative) Type II Error	TN (True Negative)

Accuracy = (TP+TN) / Tot. N

Precision = TP / (TP+FP)

Recall = TP / (TP+FN)

$$\textbf{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

F Measure

$$F_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

Join accuracy 要每句 predict 出來的 tags 與 ground truth label 完全吻合才算對，以句為單位，實現程式付在下方。

```
eval joint accuracy : 82.0%
```

```
#eval_accuracy

for i in range(len(data['eval'])):
    if ground_truth_dev_label[i]==pre_dev_token_label_2D_clear[i]:
        total_eval_correct+=1
eval_acc=total_eval_correct/len(data['eval']) *100
print(f"eval joint accuracy : {eval_acc}%")
```

Token accuracy 為以字為單位，只要 pred 出來的字與 ground truth label 相同就算正確，實現程式是用 seqeval.metrics.sequence_labeling 內的 accuracy_score 函式來計算。

```
Token accuracy: 96.86985172981878 %
```

```
print(f'Token accuracy: {sequence_labeling.accuracy_score(eval_groundTruth, eval_pred)*100} %')
```

EX:

```
y_true = [['O', 'O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
y_pred = [['O', 'O', 'B-MISC', 'I-MISC', 'I-MISC', 'I-MISC', 'O'], ['B-PER', 'I-PER', 'O']]
```

$$\text{Joint accuracy} = \frac{正確句子數量}{所有句子數量} = \frac{1}{2} = 0.5$$

$$\text{Token accuracy} = \frac{正確字數}{所有字數} = \frac{8}{(7+3)} = 0.8$$

Q5: Compare with different configurations (1% + Bonus 1%)

Task: slot tag
Batch size = 32
Epoch = 150
Hidden layer =256
Num layer = 2
Bidirectional=True

Three models have same architecture:

category	Training time	Eval acc	Kaggle acc
LSTM	12m30s	82.6	82.198
GRU	12m18s	82.1	81.8
RNN	13m7s	78.9	沒丟

Inference time on eval:

```
with torch.autograd.profiler.profile(enabled=True, use_cuda=True, record_shapes=False,profile_memory=False) as prof:
    dev_out=lstm(dev_torken_tensor.to('cuda'))
print(prof.table())
```

LSTM

```
Self CPU time total: 409.757ms
Self CUDA time total: 412.809ms
```

GRU

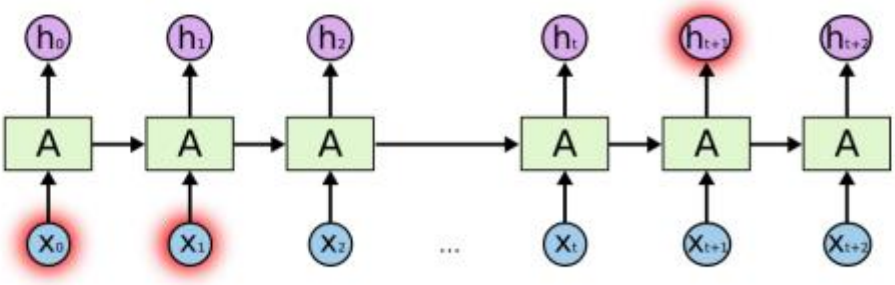
```
Self CPU time total: 398.212ms
Self CUDA time total: 400.497ms
```

RNN

```
Self CPU time total: 351.591ms
Self CUDA time total: 353.772ms
```

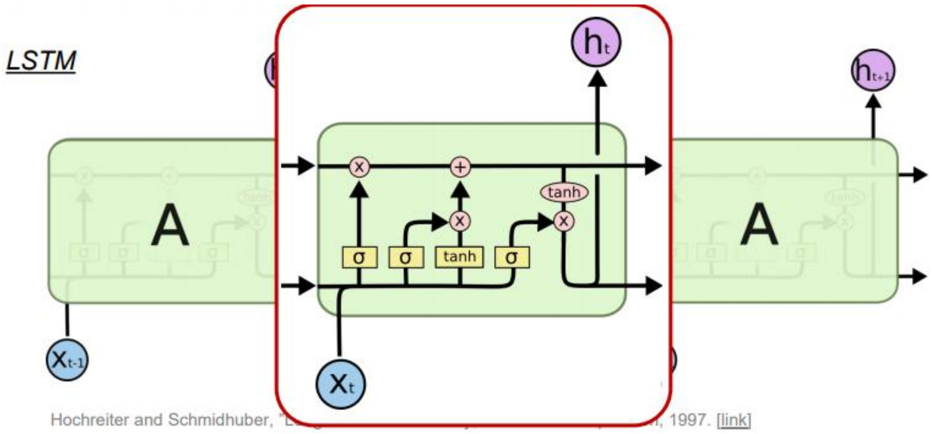
Result explanation:

RNN



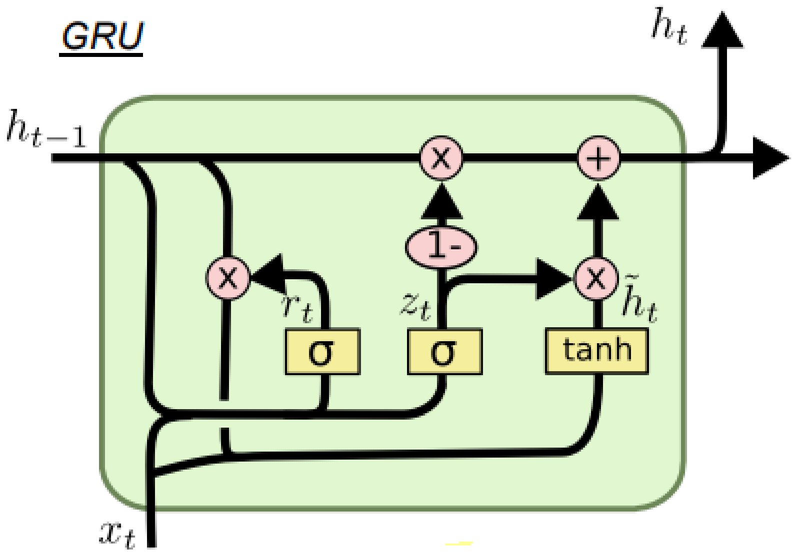
參數最少，inference time 最少，準確度最差因為會有 gradient exploding 或 vanishing 的情況發生→無法實際做到 long-term dependencies。

LSTM



多了 gating 的技術，可以控制如何處理資料，input gate、forget gate、 output gate 等等，可避免資料傳到後面消失但參數比 RNN 還多 inference time 比較常。

GRU



將 forget gate 與 input gate 合併成 update gate，使得參數比 LSTM 還要少，且也可以保留資料長距離的傳遞(避免發生 gradient vanishing 情況)。