Q1

前處理的部分我使用助教提供preprocess.sh進行處理

1. 首先將intent及slot兩個任務所對應的train及eval資料所出現的token做成以2開始能一對一對應的dictionary(0為padding, 1則為unknown), 並存成vocab.pkl;label的部分也是將所有出現過的類別作成能夠一對一對應的dictionary, 並分別存成intent2idx.json及tag2idx.json。

2. 將各個任務的vocab.pkl與sample code 下載的glove預訓練詞向量(300維)進行對照, 有出現過則使用glove預訓練詞向量, 沒出現過則隨機生成, 處理完畢後各自存成embeddings.pt。

Q2

a. Model:

i. 前處理之後以Batch為單位(以x為例)輸入進模型, 剛進模型會以第一題敘說的方式每個token轉成300維詞向量：
x -> (Batch size, Max Seq Len)
W_embeddings -> (Vocab size, 300)
x = W_embeddings * x -> (Batch size, Max Seq Len, 300)

ii. 接著LayerNorm後進入雙向雙層GRU:
output_{t}, hidden_state_{t} = GRU(x_{t}, hidden_state_{t-1});
init_hidden_state為pytorch內建初始化權重矩陣
output -> (Batch size, Max Seq Len, 2 * Hidden size)
hidden_state -> (2 * 2, Batch size, Hidden size)

iii. 因為是文本分類問題輸出取hidden_state最後一層並合併雙向GRU權重矩陣成(Batch, 2 * Hidden_size)放入分類器：
hidden_state_{last layer} -> (Batch size, 2 * Hidden size)

iv. 進入分類器後先進行LayerNorm，再進行Dropout防止過擬合，最後經過線性層輸出分到各個類別的機率:
hidden_state_{last layer} -> (Batch size, 2 * Hidden size)
W_linear -> (2 * Hidden size, Num class)
W_linear * hidden_state_{last layer} -> (Batch size, Num class)

```
SeqClassifier(
  (embed): Sequential(
    (0): Embedding(6491, 300, padding_idx=0)
    (1): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
  )
  (model): GRU(300, 512, num_layers=2, batch_first=True, dropout=0.4, bidirectional=True)
  (classifier): Sequential(
    (0): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (1): Dropout(p=0.4, inplace=False)
    (2): Linear(in_features=1024, out_features=150, bias=True)
  )
)
```

b. Performance:

Val : 0.943

Test : 0.928

c. & d. Loss Function & Hyperparameters :

Loss : CrossEntropy
Batch Size : 128
Hidden size : 512
Optimization : AdamW
Lr : 1e-3
Scheduler : CosineAnnealingLR(0.1倍率衰減)
Dropout : 0.4
Initial Weight : Normal(mean=0, std=0.02)
Model : Bidirectional double layers GRU
Max Seq Len : 28
Num Layers : 2
Epoch : 20

Q3

a. Model :

i. 前處理之後以Batch為單位(以x為例)輸入進模型，剛進模型會以第一題敘說的方式每個token轉成300維詞向量：
x -> (Batch size, Max Seq Len)
W_embeddings -> (Vocab size, 300)
x = W_embeddings * x -> (Batch size, Max Seq Len, 300)

ii. 接著LayerNorm後進入雙向雙層GRU:
output_{t}, hidden_state_{t} = GRU(x_{t}, hidden_state_{t-1})；
init_hidden_state為pytorch內建初始化權重矩陣
output -> (Batch size, Max Seq Len, 2 * Hidden size)
hidden_state -> (2 * 2, Batch size, Hidden size)

iii. 因為是詞性標註問題輸出取output每一層(t_0~t_{max_seq_len})放入分類器，進入分類器後先進行Dropout防止過擬合，再經過LayerNorm，最後經過線性層輸出分到各個類別的機率: :
output -> (Batch size, Max Seq Len, 2 * Hidden size)
W_linear -> (2 * Hidden size, Num class)
W_linear * output -> (Batch size, Max Seq Len, Num class)

```
SeqTagger(
  (embed): Sequential(
    (0): Embedding(4117, 300, padding_idx=0)
    (1): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
  )
  (model): GRU(300, 512, num_layers=2, batch_first=True, dropout=0.4, bidirectional=True)
  (classifier): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (2): Linear(in_features=1024, out_features=9, bias=True)
  )
```

b. Performance:

Val : 0.805
Test : 0.789

c. & d. Loss Function & Hyperparameters :

Loss : CrossEntropy(reduction=sum)
Batch Size : 128
Hidden size : 512

Optimization : AdamW
Lr : 1e-3
Scheduler : ReduceLROnPlateau(0.1倍率衰減)
Dropout : 0.4
Initial Weight : Normal(mean=0, std=0.02)
Model : Bidirectional double layers GRU
Max Seq Len : 35
Num Layers : 2
Epoch : 20

Q4

```
Joint Accuracy = 0.805
Token Accuracy = 0.968
F1 score = 0.80
              precision    recall  f1-score   support

        date       0.75      0.74      0.75       206
  first_name       0.92      0.94      0.93       102
   last_name       0.88      0.74      0.81        78
      people       0.74      0.74      0.74       238
        time       0.85      0.86      0.86       218

   micro avg       0.81      0.80      0.80       842
   macro avg       0.83      0.81      0.82       842
weighted avg       0.81      0.80      0.80       842
```

{'O': 0.982502322700526 5, 'B-people': 0.9201688672268907, 'I-people': 0.8917748917748918, 'B-time': 0.9174311926605505, 'B-date': 0.9271844660194175, 'I-date': 0.9172413793103448, 'I-time': 0.8142857142857143, 'B-first name': 0.9411764705882353, 'B-last name': 0.7435897435897436}

1. token_accuracy和joint_accuracy差別在於token是以每個單字為單位，joint則以每個句子為單位算準確率，joint要整個句子詞性都對才算對，因此從圖中可以明顯看出joint_accuracy比token_accuracy低不少。
2. report中precision代表true positive / predict positive，recall代表true positive / actual positive，而為了比較model間performance更加公平，f1-score將precision及recall取harmonic mean，結果會更傾向於值較小的指標，公式為2PR ／（P+R)，最後micro avg結果會更受數量多類別影響，macro avg則能較好弭平兩者間的差距。
3. 從report中可以看出模型在people與date類別的詞性標註預測得不是很好，但深入探究其token詞性類別準確率其實都高達9成，因此可以推測如果對模型做一些小修正joint_accuracy應該能有所提升；此外在last_name中precision及recall有比較大的差距，推測原因可能是資料比較少導致。

Q5

1. Intent Classification :
    a. 使用Data Argumentation : 生成一個與batch_token相同大小的標準常態分配亂數mask矩陣, 並將亂數值絕對值大於1.5的index換成unknown, 亂數絕對值大於2.5的index隨機替換成字典中其他字, 最後隨機替換每筆data的token成wordnet中的同義字
    b. 調整Batch size : 有嘗試將大小調整為64及32, 但表現不及128來得好
    c. 調整Initial Weight : 原本預設是Identity配relu但效果反而是最差, 之後又嘗試xavier normal 及 kaiming normal發現kaiming normal相較normal差了一點, xavier normal則跟normal表現差不多, 因此最後決定還是保留使用normal
    d. 調整Dropout rate及Model : 嘗試將Dropout rate調低及Model使用LSTM發現有過擬和的情形, 測出來的val acc不理想
    e. 增加Epoch : 觀察Training過程val loss有持續下降的趨勢, 因此決定將Epoch增加為 40觀察結果, 確實表現是有進步的
    f. Performance :
    Val : 0.950
    Test : 0.943
2. Slot Tagging :
    a. 使用MultitaskNet : 把兩個模型合併起來一起Train使用相同的兩層GRU只差在Embeddings與Classifier使用不同, 將兩個任務所計算的loss相加進行反向傳播
    b. 使用CRF Layer : 利用pytorch-crf套件在GRU之後再加一層CRF, 但結果測下來表現反而退步
    c. 使用Data Argumentation : 隨機替換同義字及相同詞性類別字典中其他字, 但結果測下來表現沒有變好
    d. 調整Batch size : 有嘗試將大小調整為64及32, 與intent classification不同, Batch調小模型表現變得更好, 其中又以64最為出色
    e. 增加Epoch : 因為intent classification的先例, 也將Epoch增加為40觀察結果, 確實表現是有進步的
    f. **Performance :**
    Val : 0.830
    Test : 0.826