ADL21 HW1

Q1 - Data processing

使用 sample code，先透過 preprocess.sh 下載 glove embeddings，再透過 preprocess_intent 和 preprocess_slot 來做前處理。

a. How do you tokenize the data.

Intent：

將 train, eval json 檔案讀入，紀錄出現過的 intent，把 text 以空白斷詞，並計算每個字詞的出現次數。 紀錄完所有出現過的 intent 後，輸出 intent2idx.json。

取出前 vocab_size 斷詞後最常出現的字(common words)，將他們存到 vocab class 中。 vocab class 會將這些常出現的字從 2 開始做 idx，前兩項0為 padding、 1 為 Unknown。 將 vocab class 輸出成 vocab.pkl，方便後續在處理 token 轉 id，並且做 padding。

讀入下載的 glove embedding，glove 內每行第一字為 word，後面接著為 embedding vector。這邊只處理 common words。

把 common words 內的字轉成 embedding，如果該 word 沒有出現在 glove 中就以亂數取代。輸出成 embeddings.pt。

Slot:

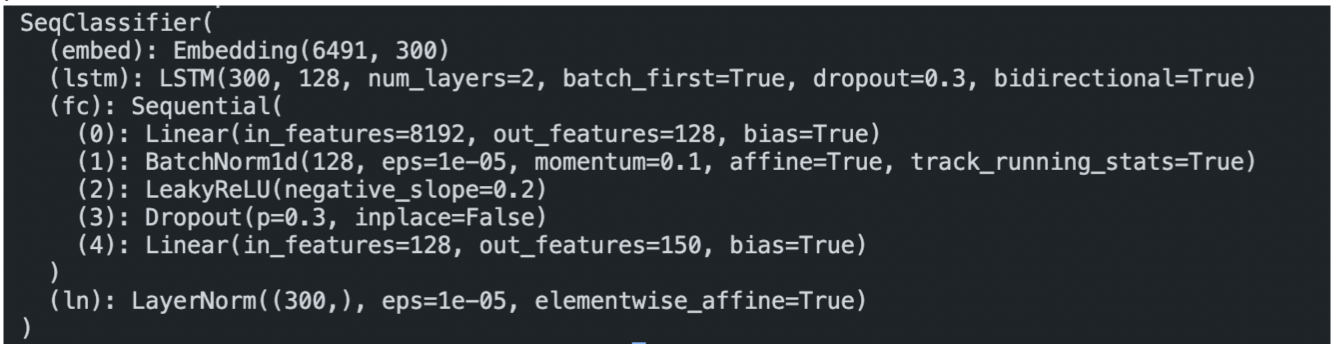
slot 與 intent 的流程相同，只差在 tokens 不需要做空白斷詞。

b. The pre-trained embedding you used.

使用 GloVe（global vectors for word representation.）。 由 stanford 開發的一種非監督學習法，通過聚合來自語料庫的global word-word co-occurrence matrix 來生成 embeddings。

Q2 - Describe your intent classification model

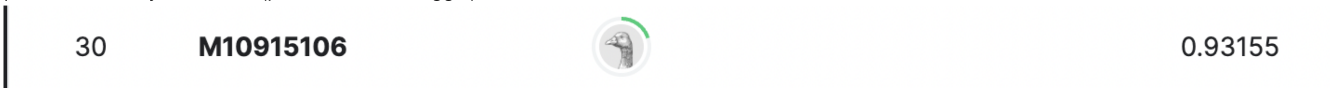
1. your model



```
SeqClassifier(
  (embed): Embedding(6491, 300)
  (lstm): LSTM(300, 128, num_layers=2, batch_first=True, dropout=0.3, bidirectional=True)
  (fc): Sequential(
    (0): Linear(in_features=8192, out_features=128, bias=True)
    (1): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Dropout(p=0.3, inplace=False)
    (4): Linear(in_features=128, out_features=150, bias=True)
  )
  (ln): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
)
```

   word embedding = $Embedding(ti)$, where ti is sentence with token index.
   word embedding = $LayerNorm(emb)$, where emb is the word embeddings of $Embedding(ti)$.
   $out, h_t, c_t = LSTM(w_t, h_{t-1}, c_{t-1})$, where w_t is the word embedding of the t-th token.
   $out = fc(out)$, where fc is a sequential container which modules will be added in the order of Linear layer, BatchNorm1d, LeakyReLU, Dropout, and another Linear layer, out is the output of linear layer $fc1$ and normalized by $BatchNorm1d$.
   model outputs "out", where out is the output of fc.

2. performance of your model. (public score on kaggle)



```
30    M10915106    0.93155
```

3. the loss function you used.

   使用 Cross Entropy loss.

   Cross Entropy 會去學習目標分佈，讓預測分佈越來越靠近目標分佈。 對於大多數分類問題，Cross Entropy Loss 都能很好用。

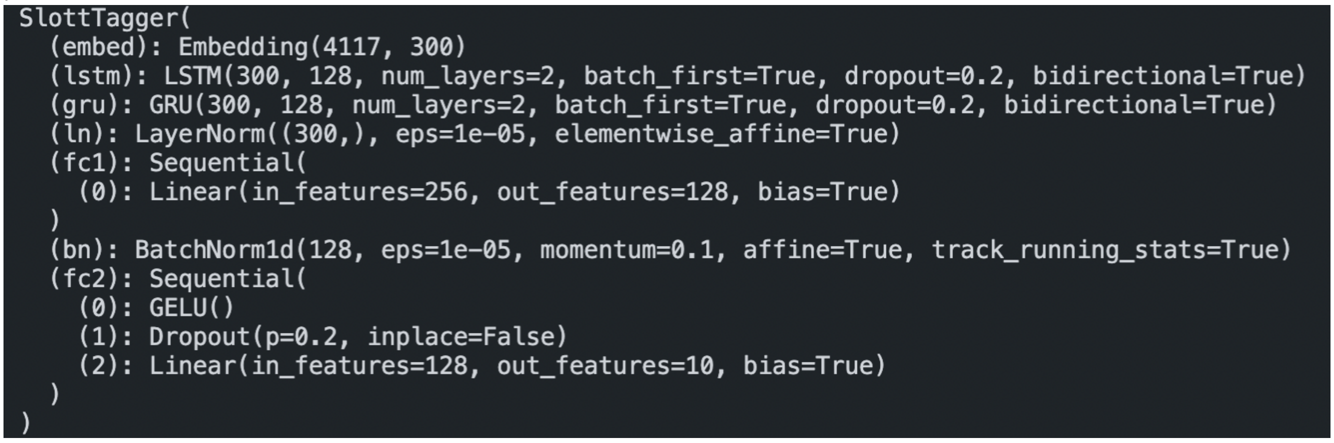
4. The optimization algorithm (e.g. Adam), learning rate and batch size.

   - Optimization Algorithm: Adam
     - 搭配 learning rate scheduler，因為模型收斂很快，希望 epoch 越後面 lr 越小。
   - Learning Rate: 1e-3
   - Batch Size: 512
     - 因為使用 BatchNorm 所以把 BatchSize 開一定大小。

~~~
```python=
optimizer = torch.optim.Adam(model.parameters(),lr=args.lr)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)
```
~~~

Q3 - Describe your slot tagging model

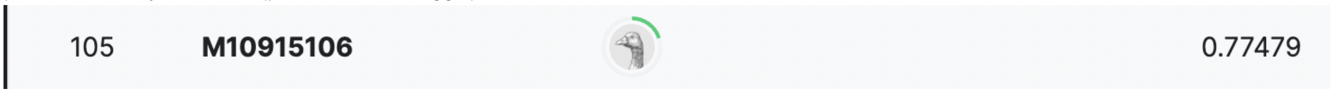
1. your model



```
SlottTagger(
  (embed): Embedding(4117, 300)
  (lstm): LSTM(300, 128, num_layers=2, batch_first=True, dropout=0.2, bidirectional=True)
  (gru): GRU(300, 128, num_layers=2, batch_first=True, dropout=0.2, bidirectional=True)
  (ln): LayerNorm((300,), eps=1e-05, elementwise_affine=True)
  (fc1): Sequential(
    (0): Linear(in_features=256, out_features=128, bias=True)
  )
  (bn): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Sequential(
    (0): GELU()
    (1): Dropout(p=0.2, inplace=False)
    (2): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

上圖中 GRU 最後並未使用。
word embedding = $Embedding(ti)$, where ti is sentence with token index.
word embedding = $LayerNorm(emb)$, where emb is the word embeddings of $Embedding(ti)$.
$out, h_t, c_t = LSTM(w_t, h_{t-1}, c_{t-1})$, where w_t is the word embedding of the t-th token.
$fc_{out} = fc1(out)$, $fc1$ is sequential container which contains a singel linear layer, where out is the output features of LSTM.
$bn = BatchNorm1d(fc_{out})$, where fc_{out} is the output of linear layer.
$out = fc2(out)$, $fc2$ is a sequential container which modules will be added in the order of GELU activation, Dropout, and Linear layer, where out is the output of linear layer $fc1$ and normalized by $BatchNorm1d$.
model outputs "out", where out is the output of $fc2$.

2. performance of your model. (public score on kaggle)



105 M10915106 0.77479

3. the loss function you used.

使用 Cross Entropy loss. 透過 weight 來處理 data imbalance 問題，class weight 為：全部資料 / (該類別數量 * 總類別)。 label_smoothing 來避免模型 overconfident 問題。

```
criterion = torch.nn.CrossEntropyLoss(
    weight=class_weight, label_smoothing=0.01)
```

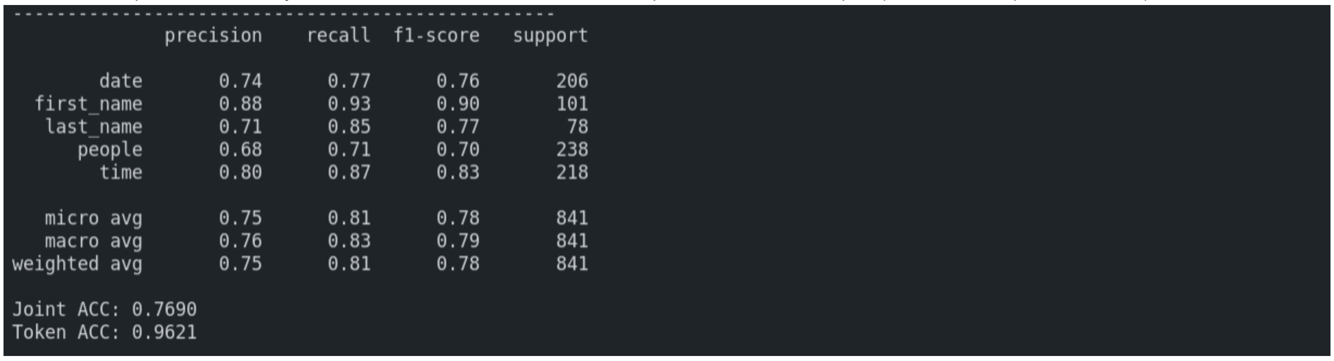
4. The optimization algorithm (e.g. Adam), learning rate and batch size.

    - Optimization Algorithm: Adam
        - 使用 weight_decay 做 L2 penalty。
    - Learning Rate: 3e-4
    - Batch Size: 512
        - 因為使用 BatchNorm 所以把 BatchSize 開一定大小。

```
optimizer = torch.optim.Adam(
    model.parameters(), lr=args.lr, weight_decay=1e-8)
```

Q4 - Sequence Tagging Evaluation

- Please use seqeval to evaluate your model in Q3 on validation set and report classification_report(scheme=IOB2, mode='strict').



```
-----------------------------------------------------
              precision    recall  f1-score   support

        date       0.74      0.77      0.76       206
  first_name       0.88      0.93      0.90       101
   last_name       0.71      0.85      0.77        78
      people       0.68      0.71      0.70       238
        time       0.80      0.87      0.83       218

   micro avg       0.75      0.81      0.78       841
   macro avg       0.76      0.83      0.79       841
weighted avg       0.75      0.81      0.78       841

Joint ACC: 0.7690
Token ACC: 0.9621
```

- Explain the differences between the evaluation method in seqeval, token accuracy, and joint accuracy.
    - joint accuract: $\frac{sentence_{correct}}{sentence_{all}}$
        - 只計算整句正確才算對，分母為句子數，分子為整句正確的句子數
    - token accuracy: $\frac{token_{correct}}{token_{all}}$
        - token 有對就有分，分母為總 token 數，分子為正確的 token 數
    - seqeval:
        - precision: $\frac{TP}{TP+FP}$
        - recall: $\frac{TP}{TP+FN}$
        - f1-score: $2 * \frac{precision*recall}{precision+recall}$
        - FP,TP,FN 的計算都是以 sentence 為單位。
        - TP: 預測和label 完全正確， FP: 對於 X 類別，預測出 X 類別但 label 不是，FN: 對於 X 類別，預測出非 X 類別但 label 是。

Q5 - Compare with different configurations

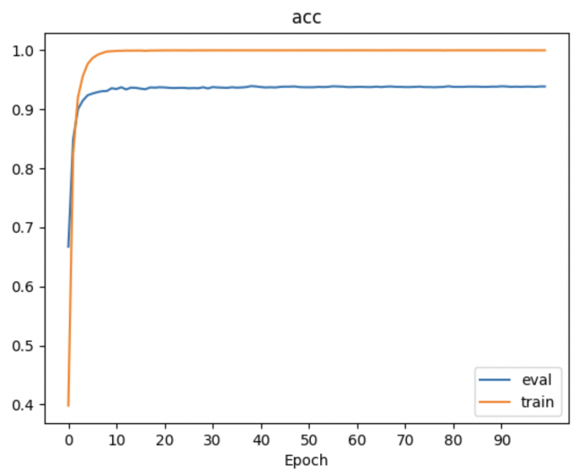
- Please try to improve your baseline method (in Q2 or Q3) with different configuration (includes but not limited to different number of layers, hidden dimension, GRU/LSTM/RNN) and EXPLAIN how does this affects your performance / speed of convergence / ...

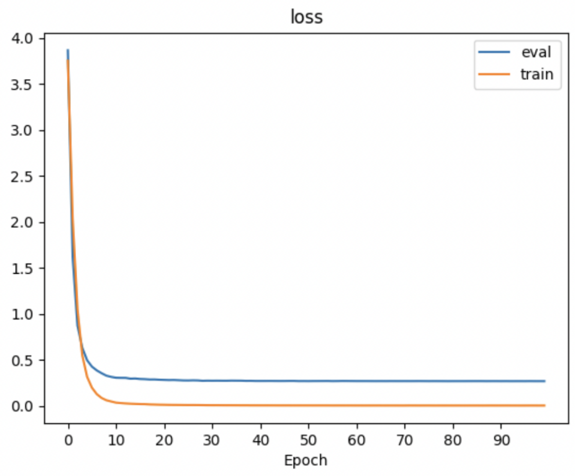
1. 針對 Intent Classification 我一開始 hidden_size 1024， batch_size 256，並且沒有加上 dropout_layer 只使用一層 Linear。
2. 加一層 dropout 後，valid set 效果有好一些，證明 dropout 有用，並且可以稍微避免 overfitting。
3. 加到兩層 fc 並在中間加上 BatchNorm1d, ReLU 和 Droput，把 hidden_size 降到 64，超越 Baseline。
4. 因為 Loss Curve 看起來有些 overfitting，所以加上 Weight Decay = 1e-6，效果沒有比較好。
5. 把 ReLU 換成 LeakyReLU 效果更好。
6. 加上 Scheduler 因為收斂的太快。

最終： 使用 StepLRSchduler

```
self.ln = nn.LayerNorm(input_size)
self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,
                            dropout=dropout, bidirectional=bidirectional)
self.fc = nn.Sequential(
    nn.Linear(self.encoder_output_size, hidden_size),
    nn.BatchNorm1d(hidden_size),
    # nn.ReLU(),
    nn.LeakyReLU(0.2),
    nn.Dropout(dropout),
    nn.Linear(hidden_size, num_class)
)
```

```
# data
parser.add_argument("--max_len", type=int, default=32)
# model
parser.add_argument("--hidden_size", type=int, default=128)
parser.add_argument("--num_layers", type=int, default=2)
parser.add_argument("--dropout", type=float, default=0.2)
parser.add_argument("--bidirectional", type=bool, default=True)
# optimizer
parser.add_argument("--lr", type=float, default=1e-3)
# data loader
parser.add_argument("--batch_size", type=int, default=512)
# training
parser.add_argument(
    "--device", type=torch.device, help="cpu, cuda, cuda:0, cuda:1", default="cuda"
)
parser.add_argument("--num_epoch", type=int, default=100)
```





Intent 實驗紀錄

name	val_acc	lr	hidden_size	num_layers	max_len	dropout	batch_size	num_epoch	dropout_layer	others
bset_0.pt	0.83	1e-3	1024	2	32	0.2	256	100	not added	
best_3.pt	0.8923	1e-3	1024	2	32	0.2	64	50	before fc	
best_4.pt	0.92	1e-3	64	2	32	0.2	512	100	between 2 fc	two layer fc with BN ReLU (PASS BASELINE)
best_4.pt	0.929	1e-3	64	2	32	0.2	512	100	between 2 fc	BN LReLU, wd=0.000001
best_5.pt	0.938	1e-3	128	2	32	0.2	512	100	between 2 fc	tow layer fc with BN and LeakyReLU
best_6.pt	0.943	1e-3	128	2	32	0.3	512	100	between 2 fc	BN LReLU(0.1), scheduler(step10, 0.5)
best_7.pt	0.939	1e-3	128	2	32	0.2	512	100	between 2 fc	BN LReLU(0.2), scheduler(step10, 0.5), (BEST RESULT)