ADL Homework 1 Report

Name： Liu Chia Ming **Student Number：** R10H41007 **Date：** October 17 2022

A. Data Processing：

The sample code given is used for data preprocessing.

a. [Intent Classification]

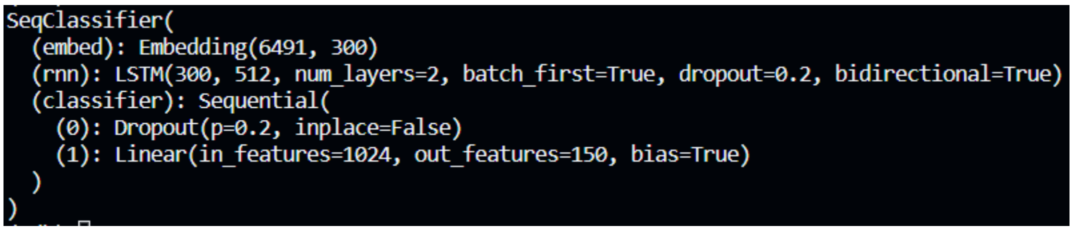
1. loading data from train/eval.json
   - tagging unique intent label (Y) with index, and save to intent2idx.json
   - tagging unique token with index, and save to vocab.pkl
2. load pre-trained glove embedding (840B tokens, 2.2M vocab, cased, 300d vectors)
3. create each token's embedding, each token will convert to 300d vector
   - Total Unique Token：6491
   - match token with glove：5435
   - Coverage：83.73%
   - Not match token would be insert random number (sample code)
4. save embedding output as embeddings.pt

b. [Slot Tagging]

1. loading data from train/eval.json
   - tagging unique tag(Y) with index, and save tag2index.json
   - tagging unique token with index, and save to vocab.pkl
2. load pre-trained glove embedding (840B tokens, 2.2M vocab, cased, 300d vectors)
3. create each token's embedding, each token will convert to 300d vector
   - Total Unique Token：4117
   - match token with glove：3000
   - Coverage：72.86%
   - Not match token would be insert random number (sample code)
4. save embedding output as embeddings.pt

B. Describe your intent classification model：

a. My Baseline model



```
SeqClassifier(
  (embed): Embedding(6491, 300)
  (rnn): LSTM(300, 512, num_layers=2, batch_first=True, dropout=0.2, bidirectional=True)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=1024, out_features=150, bias=True)
  )
)
```

This model combined with Embedding、Bi-LSTM、Linear Layer, model flow showed as below.

[Model Flow]

1. Embedding layer：convert input sequence token (S) to 300d embedding (W)
2. Bi-LSTM layer：2 layers with hidden size 512, and dropout is 0.2
3. Concat hidden state：I use h[-1]、max(h) as my baseline model
4. Linear Layer：1 layer convert concat state to 150d intent output vector
5. Get Pred_Label：argmax with Linear layer output

[Formulation]

$$W = Embedding(S)$$

$$h_t, c_t = BiLSTM(w_t, h_0, c_0)$$

$$\hat{h} = concat(h_t, \max(H))$$

$$P = Linear(\hat{h})$$

$$Y_{pred} = argmax(P)$$

S：The input sequence token

w_t：The t-th word vector in a sentence which is a 300 length vector.

W：$[w_1, \dots., w_t]$

h_t：hidden state of LSTM layer at timestamp t

H：$[h_0, \dots., h_t]$

P：$[p_1, \dots., p_{150}]$, the 150d intent output vector

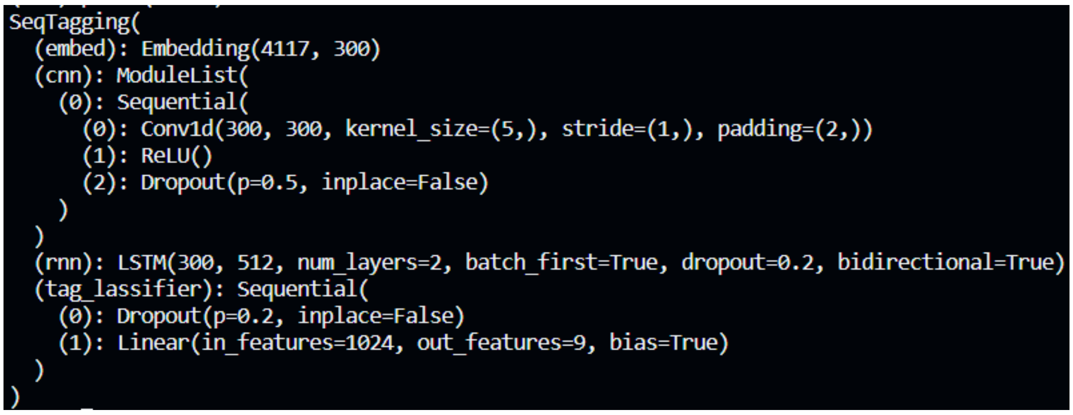
b. **Kaggle public score**：0.89466

c. **Loss Function**：Cross Entropy

d. **(Optimizer, Learning rate, Batch size)**：(Adam, 1e-3, 128)

C. Describe your intent classification model：

a. My Baseline model



```
SeqTagging(
  (embed): Embedding(4117, 300)
  (cnn): ModuleList(
    (0): Sequential(
      (0): Conv1d(300, 300, kernel_size=(5,), stride=(1,), padding=(2,))
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
    )
  )
  (rnn): LSTM(300, 512, num_layers=2, batch_first=True, dropout=0.2, bidirectional=True)
  (tag_lassifier): Sequential(
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=1024, out_features=9, bias=True)
  )
)
```

This model combined with Embedding、CNN、Bi-LSTM、Linear Layer, model flow showed as below.

[Model Flow]

1. Embedding layer：convert input sequence token (S) to 300d embedding (W)
2. Conv Layer：1 layer with (kerner=5, stride=1, padding=2) to remove noise in the sentence
3. Bi-LSTM layer：2 layers with hidden size 512, and dropout is 0.2
4. Linear Layer：1 layer convert concat state to 9d output vector
5. Get Pred_Label：argmax with Linear layer output

[Formulation]

$$W = Embedding(S)$$
$$\widehat{W} = Conv1D(W)$$
$$M, (h_t, c_t) = BiLSTM(\widehat{w_t}, h_0, c_0)$$
$$P = Linear(M)$$
$$Y_{pred} = argmax(P)$$

S：The input sequence token

$\widehat{w_t}$：The t-th word vector in a sentence which is a 300 length vector.

$\widehat{W}$：$[\widehat{w_1}, \dots.., \widehat{w_t}]$, transform by W using 1 layer CNN, like weight sum of neighbor

W：$[w_1, \dots.., w_t]$

h_t：hidden state of LSTM layer at timestamp t

M：$[M_1, \dots.., M_t]$, new sequence embedding after BI-LSTM

P：$[p_1, \dots.., p_9]$, the 9d output vector

b. **Kaggle public score**：0.81608

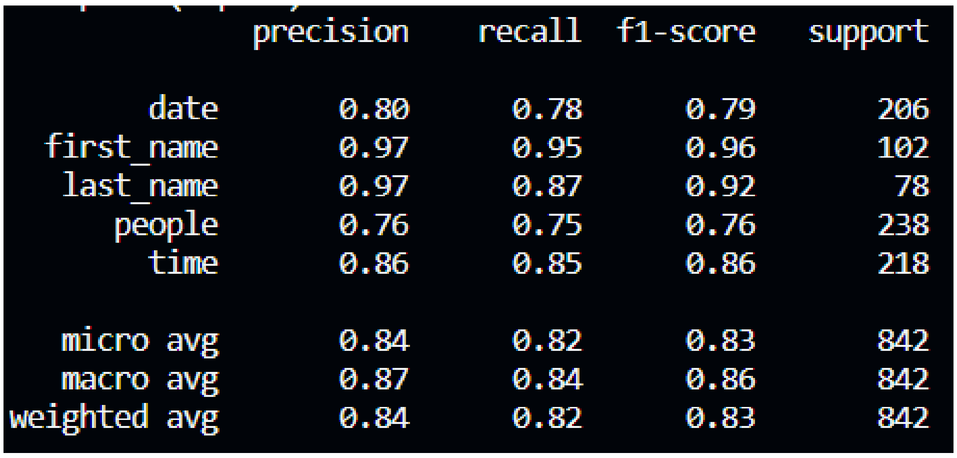
c. **Loss Function**：Cross Entropy

d. **(Optimizer, Learning rate, Batch size)**：(Adam, 5e-4, 128)

D. Sequence Tagging Evaluation ：

[Seqeval]

a. Report：



```
              precision    recall  f1-score   support

        date       0.80      0.78      0.79       206
  first_name       0.97      0.95      0.96       102
   last_name       0.97      0.87      0.92        78
      people       0.76      0.75      0.76       238
        time       0.86      0.85      0.86       218

   micro avg       0.84      0.82      0.83       842
   macro avg       0.87      0.84      0.86       842
weighted avg       0.84      0.82      0.83       842
```

b. **Joint Acc:** 0.8260 (826/1000)

c. **Token Acc:** 0.9696 (7651/7891)

d. **Explanation**：

- The reason for “token acc > join acc” are that the join acc need whole sequence predicted successfully, so it’s normal.
- The report show that “Precision > recall”, which means that our model are good at make prediction, but will ignore some correct token.

[Formulate for each indicator]

$$Joint\ acc = \frac{number\ of\ whold\ sequence\ successfully\ predicted}{number\ of\ whold\ sequence}$$

$$Token\ acc = \frac{number\ of\ token\ successfully\ predicted}{number\ of\ all\ token}$$

TP, FN, TN, FP

- TP (true positive) : we say it's positive and we are right.
- FN (false negitive) : we say it's negitive and we are wrong.
- TN (true negitive) : we say it's negitive and we are right.
- FP (false positive) : we say it's positive and we are wrong

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

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

E. Compare with different configurations ：

[Intent Classification]

➢ Kaggle Final Score：(Public, Private) = (0.91911, 0.91466)

a. Different Hidden state concat method

- In the Q2 (baseline)，I use $\hat{h} = concat(h_t, \max(H))$，and take it as input for linear layer, but I find that I can use different method for it. And I get an improvement for my model
- We can find that the method of $\hat{h} = concat(h_t, h_{t-1})$ are the best, I think the reason is that this method will not ignore any thing on the last hidden state (2-layer-bidirectional), while others not.

Different Hidden State Concat Method ($\hat{h}$)		
Concat Method	Eval Set	Kaggle Public
$\hat{h} = concat(h_t, \max(H))$	0.9020	0.89466
$\hat{h} = concat(h_t, h_{t-1})$	0.9323	0.91911
$\hat{h} = concat(h_t, \max(H), mean(H))$	0.9090	0.90044

b. Different hidden size for 2-layers LSTM

- after choose our concat method, I test on the different hidden size (256/512/1024), the experiment result are showed below.
- The table showed that 512 hidden size are still the best. I think that the 256 size might too small for this task, while 1024 size might too big.

Different Hidden Size for 2-layers Bi-LSTM		
Hidden size	Eval Set	Kaggle Public
256	0.8924	0.90577
512	0.9323	0.91911
1024	0.9200	0.91333

[Slot Tagging]

➢ Kaggle Final Score：(Public, Private) = (0.81608, 0.82743)

c. Different layer-depth for CNN part

- I find that the performance between train/eval set are obvious, I think that the reason maybe are the poor information of neighbor, so I try to adjust different CNN layer to improve it.
- The table showed below, I try 0/1/2 CNN-layer. However, it seemed that the performance of model are not improved, so we finally keep this architecture (CNN-layer=1).

Different CNN layer-depth			
Bi-LSTM Layer	Conv1D Layer	Eval Set	Kaggle Public
2	0	0.7860	0.7201
2	1	0.8260	0.81608
2	2	0.8210	0.78820
2	3	0.7970	0.72654