Applied Deep Learning Homework 1

B07902037 蔡沛勳

Q1. Data processing.

a. Data tokenization:

For intent_cls, I split each sentence into words with the space character. As for slot_tag, the sentence was tokenized initially.

To make the length of each sentence the same. If a sentence's length is more than 128 tokens, I used only the former 128 tokens to train the model. Otherwise, I will add some special token <pad> for these sentences to make it contain 128 tokens.

b. Pre-trained embeddings:

For both intent_cls and slot_tag, I used GloVe as the pre-trained embedding. For every tokens in each dataset, I converted the tokens into the corresponding index, and used Glove as my initial embedding layer to generate embeddings.

Q2. Describe your intent model.

a. Model:

```
SeqClassifier(
    (embed): Embedding(num_embeddings = 6491,
                       embedding_dim = 300
    )
    (lstm): LSTM(input_size = 300,
                 hidden_size = 256,
                 num_layers = 2,
                 dropout = 0.5,
                 batch_first = True,
                 bidirectional = True
    )
    (fc): Sequential(
        (0): Dropout(p = 0.5, inplace = False)
        (1): Linear(in_features = 512, out_features = 150, bias = True)
    )
)
```

Get the embeddings of data with embedding_dim = 300:

$$x = Embedding(data)$$

Feed the data into bidirectional LSTM with hidden_size = 256:

$$h_i, c_i = LSTM(w_i, h_{i-1}, c_{i-1})$$

Feed the first word's hidden state into a Linear classifier with out_features = 150:

$$pred = Linear(Dropout(h_0, p = 0.5))$$

b. Performance:

Evaluation Data	Kaggle Public Score	Kaggle Private Score
0.91333	0.90933	0.90533

c. Loss function: CrossEntropyLoss

d. Optimizer: Adam, learning rate = 0.001, batch size = 256

Q3. Describe your slot tagging model.

a. Model:

```
TagClassifier(
    (embed): Embedding(num_embeddings = 4117,
                       embedding_dim = 300
    )
    (lstm): LSTM(input_size = 300,
                 hidden_size = 256,
                 num_layers = 2,
                 dropout = 0.1,
                 batch_first = True,
                 bidirectional = True
    )
    (fc): Sequential(
        (0): Linear(in_features = 512, out_features = 9, bias = True)
    )
)
```

Get the embeddings of data with embedding_dim = 300:

$$x = Embedding(data)$$

Feed the data into bidirectional LSTM with hidden_size = 256:

$$h_i, c_i = LSTM(w_i, h_{i-1}, c_{i-1})$$

Feed each word's hidden state into a Linear classifier with out_features = 9:

$$pred_i = Linear(h_i)$$

b. Performance:

Evaluation Data	Kaggle Public Score	Kaggle Private Score
0.803	0.71796	0.74758

c. Loss function: CrossEntropyLoss(ignore_index = -1), which ignores the padding token <pad>.

d. Optimizer: Adam, learning rate = 0.001, batch size = 128

Q4. Sequence Tagging Evaluation

Classification report:

	precision	recall	f1-score	support
date	0.79	0.73	0.76	206
first-name	0.96	0.79	0.87	102
last-name	0.81	0.85	0.83	78
people	0.73	0.72	0.73	238
time	0.83	0.84	0.84	218
micro avg	0.80	0.78	0.79	842
macro avg	0.82	0.79	0.80	842
weighted avg	0.81	0.78	0.79	842

Evaluation:

我們可以觀察出 model 較擅長對 first-name、last-name 及 time 做出正確的 tagging, 尤其是 first-name 的 precision score 高達 0.96, 代表機器預測出的 entity 為正確預測的機率非常高。而對於 date 及 people 的 tagging 則較低。可能的原因為兩種 name 的 entity 不像剩餘三種會發生連續的 tag 一起出現, 所以不會因多標或少標連續的同一 tag 而產生錯誤。

Difference between the evaluation methods:

Joint accuracy: 整句分類正確數 / 總句數

Token accuracy: 預測正確的 Token 數 / Token 總數

Seqeval accuracy score: 同 Token accuracy

Seqeval precision score: 正確預測的 entity 數 / 預測的 entity 總數

Seqeval recall score: 正確預測的 entity 數 / 原始標註的 entity 總數

Seqeval f1 score: 2 * precision score * recall score / (precision score + recall score)

Q5. Compare with different configurations

I tried to improve my slot_tag model on 3 aspects: learning rate / hidden size / num layer

Learning rate: (hidden size = 256, num layers = 2)

	0.1	0.01	0.001	0.0001
Train acc	0.504	0.937	0.965	0.952
Train loss	0.376	0.024	0.023	0.022
Eval acc	0.532	0.762	0.778	0.730
Eval loss	0.346	0.183	0.147	0.178

撇除表現較差的 lr = 0.1, 剩餘三者的 performance 相差無幾, 不過訓練時間上 lr = 0.0001 訓練了 50 個 epoch 才達到穩定, 訓練時間為其他的五倍以上, 而 lr = 0.1 及 lr = 0.01 則訓練了 10 個 epoch 就趨於穩定。這邊使用效率高且在 eval accuracy 表現最好的 lr = 0.001 進入下一測試。

Hidden size: (learning rate = 0.001, num layers = 2)

	64	128	256	512	1024
Train acc	0.883	0.919	0.965	0.940	0.937
Train loss	0.052	0.033	0.023	0.021	0.021
Eval acc	0.730	0.741	0.778	0.779	0.772
Eval loss	0.164	0.156	0.147	0.157	0.158

在訓練時間上, 訓練時間與 hidden size 大小成正相關, 同樣跑 10 個 epoch hidden size 為 256、512、1024 的訓練時間依序為 248s、510s、1317s。這邊這邊使用效率高且 eval accuracy 表現差不多的 hidden size = 256 進入下一測試。

Num layers: (learning rate = 0.001, hidden size = 256)

	1	2	3
Train acc	0.927	0.965	0.983
Train loss	0.032	0.023	0.006
Eval acc	0.772	0.778	0.758
Eval loss	0.135	0.147	0.231

由圖表看出在訓練 > 2 層的 num layers 布景會需要更多 epoch 才穩定, 且會加重 model overfitting 的程度, 所以這邊使用 num layers = 2 來做訓練。

最後, 我觀察到此 model 會有嚴重的 overfitting, 我在 Adam 的參數加上 weight decay = 1e-5 及在 Linear classifer 中加上 Dropout(p = 0.5) 來嘗試降低該現象。

Final result: (learning rate = 0.001, hidden size = 256, num layers = 2)

	Train Accuracy	Eval Accuracy	Kaggle Public	Kaggle Private
Original	0.971	0.803	0.718	0.746
Improved	0.917	0.765	0.751	0.754

可以看可以看出新的 model 較原本 model 產生的結果穩定，在 public 及 private 的分數皆超過 0.75，而原先在 eval accuracy 得到 0.803 的表現可能是由 overfittinig 導致的結果, 才會跟 kaggle 上的成績落差這麼大。