ADL HW1

No collaborators

Q1: Data processing

I simply use the given sample code.

a. I tokenized the data by the space in text. The preprocess_<task>.py builds a mapping between the index and intent/tag and calculates the number of occurrences of the word (token) in the dataset to construct dictionary of the top common words.

b. glove.840b.300d.txt: 840B tokens, 2.2M vocab, cased, 300d vectors

Q2: Intent Classification

a. Embedding layer transforms the token to 300-dimension vector.

$$e_t = \text{Embedding}(w_t)$$

- w_t: the t-th token
- e_t: embedding of the t-th token

Feature extractor: 2-layer bidirectional GRU with 0.1 dropout

$$o_t, h_t = \text{GRU}(e_t, h_{t-1})$$

- h_t: hidden state of the t-th token
- o_t: output of the t-th token

Classifier: 0.1 dropout layer following by a linear layer

$$l = \text{Linear}(h_T[-1, -2])$$

- $h_T[-1, -2]$: the final 2 layers of h_T (hidden state of the final token)
- l: logit

```
SeqClassifier(
  (embedding): Embedding(6491, 300)
  (recurrent): GRU(300, 512, num_layers=2, batch_first=True, dropout=0.1, bidirectional=True)
  (fc): Sequential(
    (0): Dropout(p=0.1, inplace=False)
    (1): Linear(in_features=1024, out_features=150, bias=True)
  )
)
```

b. 0.92533

c. CrossEntropyLoss()

d. Optimizer: Adam; learning rate: 1e-3; batch size: 128

Q3: Slot Tagging

a. Embedding layer transforms the token to 300-dimension vector.

$$e_t = \text{Embedding}(w_t)$$

- w_t: the t-th token
- e_t: embedding of the t-th token

Feature extractor: 3-layer bidirectional LSTM with 0.1 dropout

$$o_t, (h_t, c_t) = \text{LSTM}\big(e_t, (h_{t-1}, c_{t-1})\big)$$

- h_t: hidden state of the t-th token
- c_t: cell state of the t-th token

- o_t: output of the t-th token

Classifier: 0.1 dropout layer following by a linear layer

$$l_t = \text{Linear}(o_t)$$

- l_t: logit of the t-th token

```
SlotTagger(
  (embedding): Embedding(4117, 300)
  (recurrent): LSTM(300, 512, num_layers=3, batch_first=True, dropout=0.1, bidirectional=True)
  (fc): Sequential(
    (0): Dropout(p=0.1, inplace=False)
    (1): Linear(in_features=1024, out_features=9, bias=True)
  )
)
```

b. 0.75013

c. CrossEntropyLoss()

d. Optimizer: Adam; learning rate: 1e-3; batch size: 128

Q4: Sequence Tagging Evaluation

Seqeval

	precision	recall	f1-score	support
date	0.78	0.78	0.78	206
first_name	0.89	0.92	0.90	102
last_name	0.79	0.69	0.74	78
people	0.77	0.76	0.77	238
time	0.84	0.86	0.85	218
micro avg	0.81	0.80	0.81	842
macro avg	0.81	0.80	0.81	842
weighted avg	0.81	0.80	0.80	842

The metrics mentioned in the followings are precision, recall, and F-score.

- Micro avg: Calculate metrics globally by counting the total true positives, false negatives and false positives.
- Macro avg: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- Weighted avg: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

$$\text{token\_accuracy} = \frac{\#\ of\ correct\ slot\ prediction\ of\ each\ token}{total\ \#\ of\ token\ (word)} = \frac{7642}{7891} = 0.9684$$

$$\text{joint\_accuracy} = \frac{\#\ of\ sequence\ in\ which\ all\ slots\ are\ predicted\ correctly}{total\ \#\ of\ sequence\ (text)} = \frac{815}{1000} =$$

0.815

Q5: Compare with different configurations

The evaluations of the following configurations are from Kaggle public score.

1. Intent Classification

Baseline model: 2-layer bidirectional LSTM with 512 hidden state and 0.1

dropout

a. Recurrent structure

RNN	Baseline (LSTM)	GRU
0.91155	0.90400	0.92533

b. Hidden dimension

256	Baseline (512)	1024
0.91422	0.90400	0.90933

c. Number of layers

1	Baseline (2)	3
0.92266	0.90400	0.90533

From the result shown above, I think the task is so simple that it easily overfits the training data as we increase the parameters of the model.

2. Slot Tagging

Baseline model: 2-layer bidirectional LSTM with 512 hidden state and 0.1 dropout

a. Recurrent structure

RNN	Baseline (LSTM)	GRU	CNN-BiLSTM (kernel size = 3)
0.70616	0.71313	0.71045	0.70134

LSTM and GRU perform better than RNN since they are able to carry more information of the previous tokens (or time step).

b. Hidden dimension

256	Baseline (512)	1024
0.71367	0.71313	0.72922

c. Number of layers

1	Baseline (2)	3
0.72171	0.71313	0.75013

The seq2seq tagging task is more complicate than the intent classification. According to the experiments above, a bit more parameters will fit the data better.

d. Loss function

Baseline (CrossEntropyLoss)	FocalLoss(γ=2)
0.71313	0.72546

Since I observe that the numbers of tags are quite imbalance (O>> B, I), I apply Focal loss which can handle this problem.

e. (BONUS) CNN-BiLSTM kernel size

3	5
0.70134	0.70187

The model with the bigger kernel size contains more context information, so it can perform a bit better than the one with the small kernel size.

Reference

1. GloVe: Global Vectors for Word Representation (stanford.edu)
2. [PyTorch] 如何使用 pad_packed_sequence 和 pack_padded_sequence 調整可變長度序列批次 - Clay-Technology World (clay-atlas.com)
3. Text Classification Pytorch | Build Text Classification Model (analyticsvidhya.com)
4. clcarwin/focal_loss_pytorch: A PyTorch Implementation of Focal Loss. (github.com)
5. [1511.08308v5] Named Entity Recognition with Bidirectional LSTM-CNNs (arxiv.org)