ADL HW1

r11922a05 資工AI一 林聖硯

Q1: Data processing

a. How do you tokenize the data.

For intent classification

- Remove any repeated punctuation marks, like “!”, “@”, “>”, etc
- Lower every word
- Split the sentence with space (’ ') into tokens

For slot tagging

- Remove **'s** in XXX’s, remove **'m** in I’m, remove **'d** in I’d (since this task has a large number of tokens for which we don’t have any embeddings, missing pretrained embedding for a token alone is likely a useful signal in this task. Thus I exclude all such tokens from vocabulary so they’d all get mapped to [UNK])
- Remove any repeated punctuation marks, like “!”, “@”, “>”, etc.

b. The pre-trained embedding you used
I use the following two embedding for my experiments

- Fasttext(Facebook pre-trained word vectors): crawl-300d-2M.vec.zip. There are 2 million word vectors trained on Common Crawl (600B tokens).
  - Ref: **https://fasttext.cc/docs/en/english-vectors.html** (https://fasttext.cc/docs/en/english-vectors.html)
- GloVe pre-trained word vectors: Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download)
  - Ref: **https://nlp.stanford.edu/projects/glove/** (https://nlp.stanford.edu/projects/glove/)

Since we couldn’t use their python packages, I write a function to parse them into word embeddings.

Q2: Describe your intent classification model.

a. your model

- backbone: 2 layer bi-LSTM, dropout rate = 0.1, hidden size = 512
  the computation of each component in LSTM showed in the following equations. (Ref: **pytorch, LSTM module** (https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html))
- head: 2-layer MLP with dropout and ReLU, use the last hidden state as MLP input

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

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

- public accuracy: 0.89244

c. the loss function you used.

- nn.CrossEntropyLoss

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

- optimizer: Adam
- leaning rate: 1e-3
- batch size: 128

Q3. Describe your slot tagging model

a. your model

- backbone: 2 layer bi-LSTM, dropout rate = 0.1, hidden size = 512
  b. performance of your model. (public score on kaggle)
- joint accuracy: 0.77479
  c. the loss function you used.
- nn.CrossEntropyLoss
  d. The optimization algorithm (e.g. Adam), learning rate and batch size.
- optimizer: Adam
- leaning rate: 1e-3
- batch size: 16

Q4. Sequence Tagging Evaluation (2%)

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

```
(adl-hw1) r11922a05@cuda3:~/ADL21-HW1$ bash slot_tag.sh data/slot/eval.json pred.eval.slot.csv
Accuracy 0.792000 (792/1000)
              precision    recall  f1-score   support

        date       0.76      0.78      0.77       206
  first_name       0.96      0.89      0.92       102
   last_name       0.88      0.78      0.83        78
      people       0.72      0.72      0.72       238
        time       0.82      0.82      0.82       218

   micro avg       0.80      0.79      0.79       842
   macro avg       0.83      0.80      0.81       842
weighted avg       0.80      0.79      0.79       842
```

b. Explain the differences between the evaluation method in seqeval, token accuracy, and joint accuracy.

Seqeval have the two evaluation modes. Default method and strict method. In strict mode, the inputs are evaluated according to the specified schema, IOB2 means evaluate the prediction with IOB format except that the B- tag is used in the beginning of every chunk (i.e. all chunks start with the B- tag).

The formulas of precision, recall and f1-score are as follows.

- precision = $\frac{TP}{TP+FP}$
- recall = $\frac{TP}{TP+FN}$
- f1-score = $2 \times \frac{precision \times recall}{precision + recall}$

Precision, recall and f1-score should be 1 (high) for a good classifier. If precision is high in a certain class, that means your model doesn't predict other classes' data as this cass (low FP). If recall is high in a certain class, that means your model doesn't predict this class data as other classes.

Micro averaging computes a global average precision/recall/F1-score by counting the sums of the True Positives (TP), False Negatives (FN), and False Positives (FP). We first sum the respective TP, FP, and FN values across all classes and then plug them into the precision/recall/F1-score equation.

The macro-averaged score is computed using the arithmetic mean (aka unweighted mean) of all the per-class scores. It is the most straightforward among the numerous averaging

methods.
The weighted-averaged score is calculated by taking the mean of all per-class scores while considering each class's support, which is the number of actual occurrences of the class in the dataset.
In my opinion, I think weighted-averaged score or micro averaging score could reflect the model's performance more effectively since macro-averaged scores tend to "hide" the low performance of certain class by averaging it with other high performance class.
Token accuracy means computing token-wise accuracy. That is, the accuracy increases if the model predict one token right. Joint accuracy means computing sentence-wise accuracy, which means the accuracy increases if the tags of **whole sentence** are correct.

Q5. Compare with different configurations

For Q2, I try several methods to improve my performance.

- Use cross validation to select best hyperparmeters
- Adjust LSTM layer
- GloVe embedding v.s. Concatenate Glove embedding with FastText embedding
- lower learning rate in embedding layer (regular learning rate times 0.1) v.s. same learning rate across all layer
- Use last hidden state as MLP input v.s. concatenate with average hidden state and concatenate with maximum hidden state (the following picture shows the implementation)

```python
def forward(self, batch) -> Dict[str, torch.Tensor]:
    x = self.embed(batch['token_ids'])  # (bs, seq, embed_size)
    h, c = self.encoder(x)  # h = (bs, seq, enc_size)

    # pool last state, avg and max pool of hidden states to fc layer
    h_last = h[:, -1, :]
    h_avg = h.mean(1)
    h_max, _ = h.max(1)
    x = torch.cat([h_last, h_avg, h_max], 1)  # (bs, pooled_size)

    return self.head(x)  # (bs, num_class)
```

The following table shows my experiment result.

LSTM layer	lower learning rate in embbeding layeror not	Glove / Glove + FastText	last hidden state / concat(last, avg, max)	Valid acc	Public acc
2	No	Glove	last hidden state	0.80673	0.82943
3	No	Glove	last hidden state	0.79137	0.77551
4	No	Glove	last hidden state	0.78924	0.78398
2	No	Glove	Concat	0.85912	0.87112
3	No	Glove	Concat	0.84132	0.85555
4	No	Glove	Concat	0.83912	0.84142
2	Yes	Glove	Concat	0.86819	0.87723
2	No	Glove + FastText	Concat	0.87023	0.86942
3	No	Glove + FastText	Concat	0.86997	0.85926
3	Yes	Glove + FastText	Concat	0.90344	0.88542
2	No	Glove + FastText	Concat	0.87111	0.89245
2	Yes	Glove + FastText	Concat	0.93272	0.92844

For Q3, I try several methods to improve my performance.

- Use cross validation to select best hyperparmeters
- Adjust batch size
- lower learning rate in embedding layer (regular learning rate times 0.1) v.s. same learning rate across all layer
- number of layer in LSTM

- GloVe embedding v.s. Concatenate Glove embedding with FastText embedding
- lower learning rate in embedding layer

The following table shows my experiment result.

Batch size	LSTM layer	lower learning rate in embbeding layeror not	Glove / Glove + FastText	Valid acc	Public acc
16	2	No	Glove	0.74015	0.84943
32	2	No	Glove	0.73122	0.73551
64	2	No	Glove	0.72776	0.72398
128	2	No	Glove	0.71912	0.71524
256	2	No	Glove	0.69821	0.70425
16	3	No	Glove	0.73010	0.72786
16	2	Yes	Glove	0.76819	0.77111
16	2	No	Glove + FastText	0.76023	0.76622
16	3	No	Glove + FastText	0.75997	0.75985
16	3	Yes	Glove + FastText	0.78344	0.76542
16	2	No	Glove + FastText	0.77333	0.76107
16	2	Yes	Glove + FastText	0.79348	0.77479