NTU-ADL-HW1-Report (R11922A16 資工碩一 柳宇澤)

tags: *NTU* *ADL*

Q1: Data processing

我使用Sample Code進行修改來完成這次作業。我認為助教非常仁慈，資料的前處理基本上已經幫我們完成，第一手接觸的資料是已經處理過的 formatted json data，而且已經切好 (train, eval/dev, test)，因此我只需要看懂 Sample Code 的使用方式就好。

`preprocess.sh` 先下載了 Glove 的 pretrained embedding，由於投影片中提到不能使用trained model，使用 pretrained embedding 這種模糊地帶我就也沒碰了，就

直接使用 Glove 的 pretrained embedding (`embeddings.pt`)。

`util.py` 中包含重要的 `Vocab` class，preprocess 中會建立 vocab instaance 供日後 token 的 indexing 使用。另外也有一個 padding 的小工具 function。

`dataset.py` 定義訓練中需要使用的 data 與 function，preprocess也會產生 label index 的檔案 (`{intent,tag}2idx.json`)， `num_class` 有關於最後 FCN 的 Output feature size 就是在這邊 init dataset 時丟入 label index 檔案取得。

斷詞的部分， `intent` 資料集中我直接將 `text` 屬性用空格斷詞， `slot` 資料集中則已經斷好在 `tokens` 屬性。

Q2: Describe your intent classification model.

Model

1. Input layer
   A seqence with length `s` will be padding to a sequence with length `S'` .
2. Embedding layer
   Get pretrained glove embeddings of every token in the sequence. A vector's dimension is `D` . Vectors of a sequence's size will be `Vseq = [S', D]`
3. RNN(LSTM) layer
   I used 2 layered, dropout=0.1 bidirectional LSTM with hidden size `h` in intent classification task. LSTM will be equipped with `S'` hidden layer. The vector size of every hidden layer will be `V_l = [2h]` . I concatenated first-layered vector's last half(`V_0 = [h:]`) and last-layered vector's first half (`V_{S'-1} = [:h]`) as LSTM's output, `V_lstm = [2h]` .
4. Fully connected layer
   In this task, there are 50 label class. I transformed

`V_lstm` to 50 dimensioned vector, `V_class = [50]`, so each dimension value could represent the class' value.

5. (Optional) Softmax layer
   Convert `V_class` to `V_classprob = [50]` with softmax to represent intent class probability distribution of this sequence. But this step is not neccessary.

When calculating accuracy, we take the class with maximum probability as the prediction.

Public performance on Kaggle

190	r11922a16	0.90844	3	2d

Accuracy = 0.90844

Loss function

Binary Cross Entropy Loss: torch.nn.BCELoss

Optimizer, learning rate, batch size

- Optimizer: AdamW
- Learning rate: 0.001
- Batch size: 2048

Q3: Describe your slot tagging model.

Model

1. Input layer
   A seqence with length `s` will be padding to a sequence with length `S'`.
2. Embedding layer
   Get pretrained glove embeddings of every token in the sequence. A vector's dimension is `D`. Vectors of a sequence's size will be `Vseq = [S', D]`
3. RNN(LSTM) layer
   I used 3 layered, dropout=0.5 bidirectional LSTM with hidden size `h` in intent classification task. LSTM will be

samely equipped with `S'` hidden layer. The vector size of every hidden layer will be `V_l = [2h]`. Different from Intent model, we need to deal with tokens at every position in the sequence. Hence, I directly used hidden state vectors in each layer for representing each token. Finally, we got the feature vectors for every token, `V_lstmtks = stack(V_0, V_1, ..., V_{S'-1}) = [S', 2h]`

4. Fully connected layer
   In this task, there are 9 label class. I transformed `V_lstmtks` to `[S', 9]` vector, `V_tksclass`, so each 2nd dimension value could represent the class' value.
5. (Optional) Softmax layer
   Convert `V_tksclass` to `V_tksclassprob = [S', 9]` with softmax to represent slot class probability distribution of each token in the sequence. But this step is not neccessary.

When calculating accuracy, we firstly slice `V_tksclassprob` into `V_tksclassprob_origin` with original sequence length `S` (`[S', 9] → [S, 9]`). Then take the class with maximum probability as the prediction.

Public performance on Kaggle

246	**r11922a16**	0.74101	9	19h

Accuracy = 0.74101

Loss function

Binary Cross Entropy Loss: torch.nn.BCELoss
But since `S` differ for every sequence in this case, all sequence could not be aggregated into a matrix in a batch.
We firstly need to slice predictions & lables into original sequence length (`[S', 9] → [S, 9]`). Secondly, in a batch

with batch size `B`, we do resize to predictions & labels (`B*[S, 9] → [B*S, 9]`). Finally, we could calculate the batchwise BCE loss.

Optimizer, learning rate, batch size

- Optimizer: AdamW
- Learning rate: 0.01 with scheduler gamma = 0.5
- Batch size: 4096

Q4: Sequence Tagging Evaluation

```
              precision    recall  f1-score   support

        date       0.73      0.67      0.70       225
  first_name       0.76      0.86      0.81        91
   last_name       0.54      0.76      0.63        55
      people       0.70      0.68      0.69       244
        time       0.83      0.85      0.84       215

   micro avg       0.74      0.75      0.74       830
   macro avg       0.71      0.76      0.73       830
weighted avg       0.74      0.75      0.74       830
```

- Joint Accuracy:

    - Pred 與 Label 一整句要完全相同才算正確句。總預測正確句 / 總句數。
- Token Accuracy:
    - Pred 與 Label 只要一個字相同就算正確字。總預測正確字 / 總字數。
- Seqeval:
    - 在BIO標記中，**B-XXX**代表一個詞的開始，**I-XXX**則代表詞的其他字。根據這種斷詞，Seqeval以**B-XXX**為計算的單位，Precision=預測正確的**B-XXX**數量/Pred中**B-XXX**的數量，也就是預測的精準度；Recall=預測正確的**B-XXX**數量/Label中**B-XXX**的數量，也就是所有答案的取回程度；F1=2 * Precision * Recall / (Precision + Recall)，綜合Precision跟Recall的指標。

Q5: Compare with different configurations

1. Padding Size

In sample code, default variable `max_len` is set to `128`, which means each sequence will be padded to `128` length sequence. But according to the files, `./data/{slot, intent}/{train, test}`, it shows that:

1. Maximum sequence length of intent dataset = `28`
2. Maximum sequence length of slot dataset = `35`

In my opinion, we need to avoid increasing the padding size as much as possible because the precious information is in the raw sequences. Hence, I set `max_len` to `32` for intent dataset and `35` for slot dataset.

The Improvements on public score are:

1. Intent: `0.90311 → 0.90844`
2. Slot: `0.54209 → 0.65040`

2. Scheduler

When training the slot tagging model, I found that it's a harder task to train compared with intent classification. If I set `lr` to `0.0001`, it will learn slow. If I set `lr` to bigger

one `0.01` , it will learn well in early stage but fail to learn at late stage because of the large learning rate.
Hence, I try to add scheduler in my training process with applying initial `lr = 0.01` and reducing `lr` by the factor of `2` every `20` epochs.
It turns out that models successfully reduced the loss step by step.

3. Layers of LSTM & Dropout

I directly passed baseline of intent classification by using `2 layerd` LSTM. What's more, I think that `dropout` did help for generalization and avoding overfitting.
However, this setting didn't work for slot tagging. Hence, I try some settings:

# Layer	Dropout rate	Public score	Val loss
1	0.1	0.63538	-
2	0.1 (default)	0.65040	-
2	0.3	0.62734	-
3	0.5	**0.74101**	**0.0350**
3	0.8	-	0.0390
4	0.5	-	0.0810
4	0.8	0.52171	0.0600

Due to submission limit, I didn't upload results of every settings. For `Val loss` , I didn't start recording it in early try.
But from the table, we could observe that different layers are matches for different `dropout rates` . For example, 1 & 2 layered models are matches for `dropout = 0.1` , 3-layered model is a match for `dropout = 0.5` , and 4-layered model is a match for `dropout = 0.8` .

According to `public scores`, 3 layered model performs best while 4 layered model could loss too much information in high layers.
Let’s take a look at each layer’s performance trend, according to `val loss`, you would find that:

1. 1 & 2 layered models’ performance: `dropout = 0.1`
2. 3-layered model’s performance: `dropout 0.5 > 0.8`
3. 4-layered model’s performance: `dropout 0.8 > 0.5`

Roughly, **the more layers a model has, the more dropout the model needs**. In my opinion, it could caused by the fact that noise could probagate significantly in high layer.