ADL HW1

tags: `write-up`

Q1: Data processing

intent part

tokenize the data

1. data: 從 sample 中取出 "text" 以空白切割，將字詞以函數 (function) utils.Vocab.encode_batch 做操作，此函數可以將 token 映射成 id，並且可以將所有 tokens (aka. text) padding 成相同的長度。(為了 model 將此變數的型別轉成 LongTensor)
2. lable: 從 sample 中取出 "intent"，將其直接以函數 label2idx 從 label 映射成 id。(為了 model 將此變數的型別轉成 LongTensor)
3. id: 從 sample 中取出 "id" 並且儲存。

pre-trained embedding (vocab)

1. 對於 vocab 的製作，根據 most_common: vocab_size 的設定，從 `.json` 取出最常見的字詞，與其在 glove 檔案內儲存的高維空間向量。
2. 對於我們的資料集完成 embeddings 的製作，如果字詞出現在 vocab 之中擇取其 300 維的向量，反之則用亂數產生此字詞在高維 (300 維) 空間的向量。
3. 如此一來，字詞與其向量的對應表就順利完成了。因為使用了 glove 可以使得我們訓練模型更有效率。
4. 最後產出 6491 個字詞，每個字詞使用 300 維的向量來表示。

slot part

tokenize the data

> 首先，為了 model 中的功能，以 sample 中的 "tokens" 長度為依據，對所有 sample 做重新排序 (in descending order)。

1. data: 從 sample 中取出 "tokens"，將每個字詞以函數 (function) utils.Vocab.encode_batch 做操作，此函數可以將 token 映射成 id，並且可以將所有 tokens (aka. text) padding 成相同的長度；此外，為了要能夠使得模型預測得更好，將所有 tokens padding 至 max_len。(為了 model 將此變數的型別轉成 LongTensor)
2. lable: 從 sample 中取出 "tags"，將其直接以函數 label2idx 從 label 映射成 id；同上，為了要能夠使得模型預測得更好，將所有 tags padding 至 max_len，並且以 `0` 做為 padding 的 tag。(為了 model 將此變數的型別轉成 LongTensor)

3. id: 從 sample 中取出 "id" 並且儲存。
4. length: 另外因為並非所有 tokens 都具有相同長度，所以紀錄每個 tokens 各自的長度。

pre-trained embedding (vocab)

1. 與 intent part 在 pre-trained embedding (vocab) 相同，所以不再贅述。
2. 但是對於 most_common: vocab_size 設定有做出額外調整，請參見 Q5。
3. 最後產出 1002 個字詞，每個字詞使用 300 維的向量來表示。

Q2: Describe your intent classification model

model

> 說明順序與 forward 順序相同，公式說明請見下一小節。

文字說明

首先，設定 pre-build: Embedding layer by embeddings

1. `(batch_size, embedding_dim) = Embedding(batch)`
2. `(batch_size, sequence_length, encoder_output_size) = LSTM(batch_size, sequence_length, embedding_dim)`
   - LSTM: hidden_size (= 512), num_layers (= 2), dropout (= 0.1) 皆是參考 args 中的設定。
3. `Dropout(dropout = 0.1)`
4. 取 3-dim tensor 中 second-dim 的 最後一筆資料，aka. `[:, -1, :]` 操作。 (為了從 LSTM output 取重要的 feature)
5. `(batch_size, 150) = Linear(batch_size, encoder_output_size)`
   - Linear: in_features (= self.encoder_output_size), out_features (self.num_class = len(intent2idx) = 150) 皆是參考 self 中的設定。

公式說明

$$
\begin{aligned}
H_0 &= Embedding(x) \\
H_1, c_1 &= LSTM(H_0) \\
H_1 &= H_1[:, -1, :] \\
H_2 &= Dropout(H_1) \\
\hat{y} &= Linear(H_2)
\end{aligned}
$$

performance (on kaggle)

	Score	Placement
Public	0.91333	153
Private	0.90800	175

loss function

文字說明

`CrossEntropyLoss(label_smoothing = 0.2)` ，使用 label_smoothing 可以稍微使得 Score 稍微提升，請參見 Q5。

公式說明

$$Loss = CrossEntropyLoss(\hat{y}, y)$$

optimization , learning rate and batch size

使用 Adam，learning rate 一樣使用預設值 `1e-3` ，batch_size 一樣使用預設值 `128` 。

Q3: Describe your slot tagging model

model

> 說明順序與 forward 順序相同，公式說明請見下一小節。

文字說明

首先，設定 pre-build: Embedding layer by embeddings

1. `(batch_size, embedding_dim) = Embedding(batch)`
2. `utils.rnn.pack_padded_sequence(batch_size, pad_len, encoder_output_size)`
   - 將 Tensor 包裝成 PackedSequence。
3. `(batch_size, pad_len, encoder_output_size) = LSTM(batch_size, pad_len, embedding_dim)`
   - LSTM: hidden_size (= 512), num_layers (= 2), dropout (= 0.1) 皆是參考 args 中的設定。
4. `utils.rnn.pad_packed_sequence(batch_size, pad_len, encoder_output_size)`
   - 將 PackedSequence 還原成 Tensor 的 dim。
5. `Dropout(dropout = 0.1)`
6. `(batch_size, pad_len, sequence_length) = Linear(batch_size, pad_len, encoder_output_size)`
   - Linear: in_features (= self.encoder_output_size), out_features (self.num_class = len(intent2idx) = 150), pad_len (= max_len = 128) 皆是參考 self 中的設定。

7. 最後為了 loss function 的計算，將 3-dim 中 second-dim and third-dim 對調，aka. `permute(0, 2, 1)` 。

公式說明

$$
\begin{aligned}
H_0 &= Embedding(x) \\
PackedSequence_1 &= packpaddedsequence(H_0) \\
PackedSequence_2, c_2 &= LSTM(PackedSequence_1) \\
H_3, c_3 &= padpackedsequence(PackedSequence_2) \\
H_4 &= Dropout(H_3) \\
\hat{y} &= Linear(H_4) \\
\hat{y} &= \hat{y}.permute(0, 2, 1)
\end{aligned}
$$

performance (on kaggle)

	Score	Placement
Public	0.82841	17
Private	0.83815	5

loss function

文字說明

`CrossEntropyLoss(label_smoothing = 0.2)` ，使用 label_smoothing 可以稍微使得 Score 稍微提升，請參見 Q5。

公式說明

$$
Loss = CrossEntropyLoss(\hat{y}, y)
$$

optimization , learning rate and batch size

使用 Adam，learning rate 一樣使用預設值 `1e-3` ，batch_size 一樣使用預設值 `128` 。

Q4: Sequence Tagging Evaluation

my result

```
Train: token acc = 0.99899, joint acc = 0.99337
Eval : token acc = 0.97212, joint acc = 0.83800
              precision    recall  f1-score   support

        date       0.81      0.79      0.80       206
  first_name       0.98      0.98      0.98       102
   last_name       0.99      0.90      0.94        78
      people       0.76      0.76      0.76       238
        time       0.85      0.85      0.85       218

   micro avg       0.84      0.83      0.84       842
   macro avg       0.88      0.86      0.87       842
weighted avg       0.84      0.83      0.84       842
```

```
Train 100 / 100 : loss = 0.83779, token acc = 0.99899, joint acc = 0.99337
Eval 100 / 100 : loss = 0.11812, token acc = 0.97212, joint acc = 0.83800
Save model was done.

              precision    recall  f1-score   support

        date       0.81      0.79      0.80       206
  first_name       0.98      0.98      0.98       102
   last_name       0.99      0.90      0.94        78
      people       0.76      0.76      0.76       238
        time       0.85      0.85      0.85       218

   micro avg       0.84      0.83      0.84       842
   macro avg       0.88      0.86      0.87       842
weighted avg       0.84      0.83      0.84       842
```

名詞解釋

TP: True Positive FP: False Positive TN: True Negative FN: False Negative

accuracy: 準確度，所有猜測中預測正確的比例，介於 0~1 之間，且越高越好。

$$\frac{TP+TN}{TP+FP+FN+TN}$$

precision: 精準度 (精確度)，所有猜 Positive 中預測正確的比例，介於 0~1 之間，且越高越好。

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

recall: 召回率，所有為 Positive 的資料預測正確的比例，介於 0~1 之間，且越高越好。

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

f1-score: is the harmonic mean of the precision and recall，介於 0~1 之間，且越高越好。

$$\frac{2 \times precision \times recall}{precision + recall}$$

support: y_true 的數量 (sum of true)

micro avg: 用別的計算方式，納入 TP & FN & FP。 macro avg: 直接對對應欄位取平均數。 (分子各自皆為 1、分母為 5 in this case) weighted avg: 直接對對應欄位取加權平均數。 (加權數為各個 tag 的比重，分子各自為每種 tag 的個數、分母為 842 in this case)

官方解答

> 避免我的語意模糊不清，順便提供官方解答。

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.

accuracy 計算方式

token acc: $\frac{\text{猜對的}tag\text{總數}}{tag\text{數量}}$

joint acc: $\frac{\text{猜對的整句}tags\text{總數}}{\text{整句}tags\text{數量}}$

Q5: Compare with different configurations

intent part

> pad size = 10000 (default, constant) smooth stand for label_smoothing

RNN type	Linear	ReLU	Dropout	smooth	Score
RNN	1	0	1	x	bad
LSTM	1	0	0	x	0.89244
LSTM	1	0	1	0.25	0.91333
LSTM	2	0	1	x	0.89511
LSTM	2	0	1	0.30	0.90222
LSTM	2	0	1	0.25	0.90622
LSTM	2	0	1	0.20	0.90400
LSTM	2	0	1	0.15	0.90088
LSTM	2	0	1	0.10	0.90000
LSTM	2	0	1	0.05	0.90488
LSTM	3	1	2	x	0.87422
LSTM	3	0	2	x	0.88177
LSTM	2	1	2	x	0.88888
GRU	2	0	1	x	0.88711

> pad size variety smooth stand for label_smoothing

RNN type	Linear	Dropout	smooth	pad size	Score
LSTM	1	1	0.20	10000	0.91066
LSTM	1	1	0.20	1000	0.90400

小結

嘗試了各種組合，不過我也不是 ML 大師，不知道其他更 fancy 的技巧，所以只在我所熟知的範圍多加嘗試，增加 layer、改變 RNN type、使用 activation layer，發現對 loss_fn 增加 smooth 可以稍微增加正確率，不過沒有 slot 這題明顯，且這時我做的測試沒有固定 SEED，所以不能夠保證一定是 loss_fn 的改動，才造成正確率的上升。

做完 slot 發現一個大事 (見下方說明)，沒想到不能夠套用在 intent 之中，反而還使得正確率下降 (此時已經固定 SEED)。

slot part

> pad size = 10000 (default, constant) smooth stand for label_smoothing

RNN type	Linear	PReLU	Dropout	smooth	Score
LSTM	1	1	1	0.20	0.73083
LSTM	1	0	1	0.20	0.71957
LSTM	1	0	1	0.15	0.73619
LSTM	1	0	1	0.05	0.72439
LSTM	2	0	1	0.10	0.71420
LSTM	2	0	1	0.05	0.71420
GRU	2	0	2	0.15	0.70402
LSTM	2	0	2	0.15	0.71313

> pad size variety smooth stand for label_smoothing

RNN type	Linear	Dropout	smooth	pad size	Score
LSTM	1	1	0.20	10000	0.71957
LSTM	1	1	0.20	3000	0.81554
LSTM	1	1	0.20	1000	0.82841

小結

跟 intent 小題一樣，我嘗試了各種組合 (前期測試時沒有固定 SEED)，但是正確率十分悽慘，大多都是在 Eval 階段的 token 正確率還算 OK，但是 joint 正確率卻十分糟糕，大多都只有稍微超過 baseline 一些些，與 intent 一樣試過各種 layer 的組合，可是正確率還是很低。(當初還不知道原來 private 是 public 子集時超級緊張)

後來與朋友討論發現 preprocess 做的事情，也是其中一個可以調整的變因；另外因為有些在 slot 中出現的字詞，雖然有出現但是次數非常少，如果直接使用 glove 的結果，可能會使得 model 無法學習到 glove train 好的 vector (也有可能是我的 model 太菜了)；(此時有固定 SEED) 所以直接降低 most_common: vocab_size，從 10000 -> 3000 -> 1000，好在正確率有與預期一樣大幅度的提升。