Applied Deep Learning - HW1

資工四 b06902024 黃秉迦

Q1: Data processing

(a) Tokenization

處理 intent 的 dataset 時，利用空白將每筆 data 的 text 切開形成 tokens，最後在 tokens 的最前面加上 "[BOS]"，並在最後面加上 "[EOS]"

處理 slot 的 dataset 時，直接使用每筆 data 的 tokens，在 tokens 的最前面加上 "[BOS]"，並在最後面加上 "[EOS]"

(b) Pre-trained embedding

取得所有 data 的 tokens 後，找出最常出現的前 10000 種 tokens，並依序處理。

若 token 出現在 glove 的 pre-trained weight (glove.840B.300d.txt) 中，則將其對應的 weight 當作該 token 的 embedding weight

若 token 沒出現在 glove 的 pre-trained weight (glove.840B.300d.txt) 中，則 random 出一個長度為 300 且值域為 (1, -1) 的 vector 當作該 token 的 embedding weight

最後將 embedding weight 存起來

Q2: Intent classification model

(a) Model

$$
\begin{aligned}
&\text{let} \begin{cases} w_t = \text{word embedding of the t-th token} \\ h_t = \text{hidden state after t-th token as input} \end{cases} \\
\Rightarrow\ & h_{1..t} = 2 - Layer\_Bidirectional\_GRU(w_{1..t}, h_{0..t-1}, dropout = 0.3) \\
\Rightarrow\ & h_{1..t}^{Dropout} = Dropout(h_{1..t}, 0.3) \\
\Rightarrow\ & x^{BN} = BatchNorm(h_t^{Dropout}) \\
\Rightarrow\ & x = FC(x^{BN}) \\
\Rightarrow\ & y = Softmax(x)
\end{aligned}
$$

(b) Performance

0.90666

(c) Loss function

CrossEntropyLoss

(d) Optimization algorithm, learning rate and batch size

- optimizer: AdamW with weight decay 1e-2
- learning rate: 1e-4
- batch size: 128

Q3: Slot tagging model

(a) Model

$$
\begin{aligned}
&\text{let} \begin{cases} w_t = \text{word embedding of the t-th token} \\ h_t = \text{hidden state after t-th token as input} \end{cases} \\
\Rightarrow\ & h_{1..t} = 2 - Layer\_Bidirectional\_GRU(w_{1..t}, h_{0..t-1}, dropout = 0.3) \\
\Rightarrow\ & h_{1..t}^{Dropout} = Dropout(h_{1..t}, 0.3) \\
\Rightarrow\ & x_{1..t}^{BN} = BatchNorm(h_{1..t}^{Dropout}) \\
\Rightarrow\ & x_{1..t} = FC(x_{1..t}^{BN}) \\
\Rightarrow\ & y_{1..t} = Softmax(x_{1..t})
\end{aligned}
$$

(b) Performance

0.75710

(c) Loss function

CrossEntropyLoss

(d) Optimization algorithm, learning rate and batch size

- optimizer: AdamW with weight decay 1e-2
- learning rate: 1e-4
- batch size: 128

Q4: Sequence Tagging Evaluation

Seqeval

	precision	recall	f1-score	support
date	0.77	0.74	0.75	206
first_name	0.94	0.88	0.91	102
last_name	0.85	0.71	0.77	78
people	0.70	0.70	0.70	238
time	0.83	0.84	0.84	218
micro avg	0.79	0.77	0.78	842
macro avg	0.82	0.77	0.79	842
weighted avg	0.79	0.77	0.78	842

Token accuracy

7594 / 7891 = 0.9623621847674566

Joint accuracy

782 / 1000 = 0.782

Comparison

- 透過 seqeval 可知 model 做得最好的是 first_name，而做的最差的是 people。觀察 data 後發現這應該與其難度有關。基本上只要遇到 model 沒看過的字就很有可能是 first_name，因此辨認 first_name 相對簡單。而辨認 people 時，有時 "[數字] peolple" 中的 people 對應的 token 為 people，有時其對應的 token 卻是 O，因此較為困難
- 由於計算 joint accuracy 時正確的條件較 token accuracy 嚴格許多，因此相對於 token accuracy，joint accuracy 有明顯的差距

- seqeval 會提供每種 token (B-XXXX 和 I-XXXX 視為一種) 的 performance，特別關注某些 token 或某些 token 的 cost 比較大的時候特別適合使用
- token accuracy 只關心 token 的正確率，token 十分 inbalance 時可能其實做得很好但 performance 看起來很棒
- joint accuracy 關心整句話的 accuracy，要求整句話都要正確的時候特別適用

Q5: Compare with different configurations

Intent

Table 1

orthogonal initialization	val acc
X	0.9270
O	0.9273

- orthogonal initailization: 表示 GRU 中的 gates 是否使用 orthogonal initialization

Table 2

setting: with orthogonal initialization

num layers	hidden size	bidirectional	val acc
1	128	True	0.9070
1	128	False	0.9090
1	256	True	0.9200
1	256	False	0.9270
1	512	True	**0.9383**
1	512	False	0.9257
2	128	True	0.9187
2	128	False	0.9070
2	256	True	0.9270
2	256	False	0.9207
2	512	True	0.9270
2	512	False	0.9353
3	128	True	0.9107
3	128	False	0.9020
3	256	True	0.9227
3	256	False	0.9227
3	512	True	0.9373
3	512	False	0.9360

Conclusion

- 雖然 你在训练RNN的时候有哪些特殊的trick？ 一文中表示 orthogonal initialization 能有效提升 performance，但從 Table 1 中可看出差異並不明顯。
- 從 Table 2 中可以發現
    1. num layers 對 performance 的影響不明顯
    2. hidden size 對 performance 的影響很明顯越大越好，但幅度並不大
    3. bidirectional 能夠提升 performance，但是效果十分微弱且不明顯
- 從以上實驗中可知，1 layer + 512 hidden size + bidirectional 的效果最好，所以其實並不需要太複雜的架構

Slot

Table 1

orthogonal initialization	label	replace	val joint acc
X	default	X	0.7280
X	default	O	0.7070
X	ignore	X	0.7560
X	ignore	O	0.7290
X	add label	X	0.7600
X	add label	O	0.7540
O	default	X	0.6970
O	default	O	0.6940
O	ignore	X	**0.7820**
O	ignore	O	0.7620
O	add label	X	**0.7820**
O	add label	O	0.7750

- orthogonal initailization: 表示 GRU 中的 gates 是否使用 orthogonal initialization。
- label:
    - default: 表示 training 時所有特殊 token 的 label 為 "O"
    - ignore: 表示 training 時所有特殊 token 的 label 為 -100，意即 training 時不關心特殊 token 的表現
    - add label: 表示 training 時特殊 token 的 label 與該 token 一致，如 "[BOS]" 的 label 為 "[BOS]"
- replace: 表示是否將所有 "B-XXXX" label 換成 "I-XXXX"

Table 2

setting: orthogonal initialization + add label

num layers	hidden size	bidirectional	val joint acc
1	256	True	0.7760
1	256	False	0.7690
1	512	True	0.7800
1	512	False	0.7640
1	1024	True	0.7720
1	1024	False	0.7730
2	256	True	0.7830
2	256	False	0.7700
2	512	True	0.7800
2	512	False	0.7700
2	1024	True	0.7780
2	1024	False	0.7860
3	256	True	0.7710
3	256	False	0.7770
3	512	True	0.7810
3	512	False	0.7840
3	1024	True	**0.7950**
3	1024	False	0.7810

Conclusion

- 你在训练RNN的时候有哪些特殊的trick？ 一文中表示 orthogonal initialization 能有效提升 performance，從 Table 1 中可看出的確有些許的幫助。
- 從 Table 1 中可發現 label 使用 add label 的方法是最好的，一方面是這種方式可以使 model 學到更多資訊，另一方面可能是因為 task 變難了，所以 overfitting 的現象減弱了
- replace 原本的目的是希望 model 可以專注在學習 token 的種類而非位置，然而或許是因為 task 變簡單了，所以 overfitting 的現象增強了，也有可能是因為位置的資訊也是十分重要的
- 從 Table 2 中可以發現
    1. num layers 能夠稍微提升 performance
    2. hidden size 能夠提升 performance，但是效果十分微弱且不明顯
    3. bidirectional 能夠提升 performance，但是效果十分微弱且不明顯
- 從以上實驗中可知，雖然 3 layer + 1024 hidden size + bidirectional 的效果最好，但其實 model 間 performance 的差異不大，所以其實可能也不需要太複雜的架構