ADL HW1

R11922A11 高涵毅

Q1: Data processing

這題我使用了 sample code 的 *preprocess.sh* 進行前處理，以下說明它的作用。

a. How do you tokenize the data

這次作業兩個題目的語言皆為英文，因此在程式中可以透過空格切出多個 tokens。先將訓練與驗證集的句子都看過一遍，計算每個 token 出現的次數，最後使用 *vocab_size* 這個參數，篩出最常出現的前 N 個 token，並在字典中額外加入<PAD>跟<UNK>，分別用來處理 batch 中不同長度句子的補齊，以及代表不收錄在字典的其他 tokens。

b. The pre-trained embedding you used

我使用的 GloVe 是由 Stanford 的實驗室所研發，透過非監督式學習方式學習大量文本中不同單字的關聯，藉由這個方式可以快速獲得有意義的 feature representation，值得注意當有些 token 沒有出現在 GloVe，那麼程式會隨機初始化一向量代表該 token，而<PAD>與<UNK>在 GloVe 已有對應的特徵表達。

Q2: Describe your intent classification model

a. your model

這題 baseline model 使用的是 2-layer bidirectional GRU，可以寫成以下表達式：

$$o_t, h_t = \mathrm{GRU}(w_t, h_{t-1})$$

其中w_t是第 t 個 time stamp 的 token 所對應之 word embedding，h_t是第 t 個 time stamp 的 hidden state，o_t是第 t 個 time stamp 的 output，超參數的設定如下：

```
hidden_size = 512
dropout = 0.1
```

因為前面使用 bidirectional 的模型，h_t的輸出維度會是原先 hidden_size 的 2 倍，之後再接上一個 Sequential 的模型輸出結果，下面分別呈現表達式與 FC 模型設定：

$$h_t^* = concat(h_t[-2,\dots], h_t[-1,\dots]),$$
$$result = FC(h_t^*)$$

```
torch.nn.Sequential(
    torch.nn.Dropout(dropout),
    torch.nn.ReLU(),
    torch.nn.BatchNorm1d(hidden_size),
    torch.nn.Linear(hidden_size, num_class)
)
```

這裡 dropout 機率與前面 GRU 所採用的一樣都是 0.1。

b. performance of your model

0.88933 (public baseline: 0.86933)

c. the loss function you used

使用 CrossEntropyLoss 作為 loss function，每一個輸入只會計算一次 loss，並加入 L2

penalty ($\lambda = \frac{1}{1e-5}$)控制模型的權重。

d. The optimization algorithm, learning rate and batch size

Optimizer 使用 Adam；learning rate 初始為0.001，並使用 CosineAnnealingLR 的調整方法讓 learning rate 逐漸變小到$1e-5$；batch size 設為128。

Q3: Describe your slot tagging model

a. your model

這題 baseline model 使用的是 2-layer bidirectional GRU，可以寫成以下表達式：

$$o_t, h_t = \mathrm{GRU}(w_t, h_{t-1})$$

其中w_t是第 t 個 time stamp 的 token 所對應之 word embedding，h_t是第 t 個 time stamp 的 hidden state，o_t是第 t 個 time stamp 的 output，超參數的設定如下：

```
hidden_size = 512
dropout = 0.1
```

因為前面使用 bidirectional 的模型，h_t的輸出維度會是原先 hidden_size 的 2 倍，之後再接上一個 Sequential 的模型輸出結果，下面分別呈現表達式與 FC block 模型設定：

$$result = FC(h_t)$$

```
torch.nn.Sequential(
    torch.nn.Dropout(dropout),
    torch.nn.ReLU(),
    torch.nn.LayerNorm(hidden_size),
    torch.nn.Linear(hidden_size, num_class)
)
```

這裡 dropout 機率與前面 GRU 所採用的一樣都是 0.1。

b. performance of your model

0.78377 (public baseline: 0.71689)

c. the loss function you used

使用 CrossEntropyLoss 作為 loss function，每一個輸入會有多個 tokens，每個 token 都會計算一次 loss，並加入 L2 penalty ($\lambda = \frac{1}{1e-4}$)控制模型的權重。

d. The optimization algorithm, learning rate and batch size

Optimizer 使用 Adam；learning rate 初始為0.001，並使用 CosineAnnealingLR 的調整方法讓 learning rate 逐漸變小到$1e-5$；batch size 設為256。

Q4: Sequence Tagging Evaluation

seqeval packages evaluation

	precision	recall	f1-score	support
date	0.79	0.76	0.77	206
first_name	0.99	0.98	0.99	102
last_name	0.91	0.96	0.94	78
people	0.76	0.73	0.74	238
time	0.85	0.81	0.83	218
micro avg	0.83	0.81	0.82	842
macro avg	0.86	0.85	0.85	842
weighted avg	0.83	0.81	0.82	842

differences between the evaluation method

token accuracy: 依每個 token 為單位，將所有句子串在一起，計算整體準確率。

$$\text{token\_acc} = \frac{\text{\# correctly predicted tokens}}{\text{sum of \# tokens in data}}$$

joint accuracy: 依句子為單位，計算一個句子中所有 token 都對，占所有句子比例。

$$\text{joint\_acc} = \frac{\text{\# correctly predicted sentences}}{\text{\# sentences in data}}$$

seqeval:

在知名的 scikit-learn 套件中，針對多類別的問題同樣會使用 classification report，背後邏輯不外乎是根據混淆矩陣，計算出 TP, FP, TN, FN 這四個區域的數目，就可以推出上圖的 precision, recall 及 F1-score。但在 NER 或是 IOB 問題，一個 entity 可能就包含多個 token，如果其中有幾個 token 標錯，就應該視為整個 entity 標錯才合理，基於這樣的邏輯，seqeval 可以更客觀的評斷模型好壞，而 strict mode 限制不僅標註的 entity 要相同，前方的標籤如 I 或 B 也要一致才算對。

Q5: Compare with different configurations

Intent classification

部分的超參數依照範例程式所設置，其中預設不變的超參數如下：

```
hidden_size = 512
num_layers = 2
bidirectional = True
```

下面接著比較不同超參數，對模型準確度的影響。須注意 pretrained embedding 在模型訓練過程，內部權重會隨著迭代做更新。

max_ len	model type	drop- out	lr	L2	batch size	normal- ize	# of FC blocks	epoch (best)	public score	private score
192	GRU	0.1	1e-3	1e-5	128	batch	1[X]	50	0.88933[+]	0.89022
256	GRU	0.2	1e-3	1e-5	1024	batch	2	50	0.88800	0.88355
192	GRU	0.1	1e-3	1e-4	128	batch	1	50	0.90355	0.91777
192	GRU	0.2	1e-3	1e-4	128	batch	1	50	**0.92577**[*]	**0.92577**
192	GRU	0.2	1e-3	1e-4	256	batch	1	50(15)	0.92311	0.91822
128	GRU	0.3	1e-3	3e-4	256	layer	1	50(48)	0.91866	**0.92577**
128	GRU	0.3	1e-3	3e-4	256	batch	1	100(54)	0.92355	0.92222
128	GRU	0.3	1e-3	3e-4	512	batch	1	50(50)	0.92533[*]	**0.92577**

[X]表示沒有使用 dropout；[*]表示最終評分模型，但只有 max_len = 128 的模型可重現

一開始訓練就將 max_len 設為 192, 256，是因為設定 128 會出現異常 index error 的問題，接著調整的項目如 FC block 的個數、L2 penalty 與 dropout 比例，結果意外的訓練出最佳模型，而後續試驗才關注收斂的問題，不過已從結果發現可能存在 overfitting。若從最後三次模型結果進行分析，可以得出較大的 batch size 會比較晚收斂，不過有機會得到較好的 score；而在 FC block 中使用 Layer Normalization 雖然在驗證集與 public data 表現較差，但在 private data 並列最高分。

Slot tagging

部分的超參數依照範例程式所設置，其中預設不變的超參數如下：

```
num_layers = 2
dropout = 0.1
bidirectional = True
lr = 1e-3
L2 = 1e-4
```

下面接著比較不同超參數，對模型準確度的影響。須注意 pretrained embedding 在模型訓練過程，內部權重會隨著迭代做更新。

max_ len	hidden_ size	model type	drop- out	batch size	normal- ize	# of FC blocks	epoch (best)	public score	private score
192	512	GRU	0.1	256	layer	1	50	0.78337[+]	0.78188
128	512	RNN	0.1	128	layer	1	50(43)	0.82144[*]	0.82422
128	512	LSTM	0.1	128	layer	1	50(19)	**0.82788**[*]	**0.83011**
128	512	GRU	0.1	128	layer	1	50(17)	0.81340	0.81832

[+]代表 baseline model，vocab_size 設為 10000，且未 pack sequence；[*]表示最終評分模型

訓練完 baseline model 後，我找到 Reference 中針對可變長度處理的手法，在 pytorch 的工具箱中有 pack_padded_sequence 與 pad_packed_sequence，可以大幅提升訓練時間，同樣的 epoch 個數，baseline model 訓練將近 1 個小時，而後續 model 僅需 10 分鐘時間。

就在理解兩個 function 的運作邏輯，我意外探索到 Q1 中 preprocessing 可能帶來的影響，相較 intent classification 來說，slot tagging 的文本量與輸出類別皆小於前者，採用相同 vocab_size 會發現字典中有較多意義不明的 token，後續 vocab_size 縮小至 1000，可以明顯看到準確度皆明顯上升，不過 GRU 模型反而較不適合此問題。

Reference

- GloVe: Global Vectors for Word Representation
- Pytorch 調整可變長度序列批次
- LSTM tutorial and a few suggestions
- How to get final hidden state of bidirectional 2-layers GRU in pytorch