國立臺灣大學
National Taiwan University

指導教授：陳縕儂
學生 ： 黃泰誠
學號：R09921083

Applied Deep Learning Homework 1

Q1: Data processing:

Preprocess:

用 wget 將 glove 的檔案載下，glove 檔內有數兆個英文字且各用一組 300 維的向量去表示，因此我們需要藉由 glove 來建立一個適合本次專題的 dictionary，因此我將 intent 的"train.json"檔載下並擷取他的 "text"，將 texts 內的字抓下並將每個存在於 glove 裡的字用 glove 的向量去表示，來建構一個新的 dictionary。

<Intent_cls.sh>

Step1: 建立了一個 sentence word list 將 "train.json"的 "text" 載入，未了建立 input data 所以每一句話的字數需等長，用事前建立的 dictionary 將所有個字代換成 300 維的向量，用來訓練的 input data 就完成了，input data 的 shape 是 (15000, 12, 300)。 備註：字數限定為 12 個字

Step2: 建立 output data 將"train.json"的"intent"載入，再用作業內給的資料"intent2idx.json"轉換成數字，得到一個 shape 為(15000,)的 output。

Step3: 丟入 RNN model 做訓練

Step4: 如同 Step1 將 test data 載入並將她轉換成(4500, 12, 300) 的 input data 送入訓練好的 RNN model 再將得到的數字透過"intent2idx.json"轉換回文字，並建立成 csv 檔以便上傳至 kaggle。

<slot_tag.sh>

Step1: 建立 train input array，將 slot_tags 檔案中標示"tokens"的所有字用 300 維的向量表示，使得 input shape 為(7244, 35, 300)，7244 是因為 slot tags 的 train data 中有 7244 句話，又因為將每一句整理下來後發現最長的一句有 35 個字所以為了將資料送入 model 訓練所以需要將每句話都擴充到 35 個字如果不到 35 個字的一句話就比須將他補上 9 代表這句話已經結束，而 300 是 embedding 進去每個字的向量，同時 test input data 和 eval input data 也是都如此。

Step2:建立 train output 也是一樣的方式，開啟 cache 檔案中的"tags2idx.json"將每個 label 轉換為數字，使得 output shape 為(7244, 35)，eval output 的作法也是如此。

Step3:丟入 RNN model 做訓練。

Step4:將輸出所得到的數字用"tags2idx.json"轉換回文字並將每一列的字 join 起來丟入 csv 檔。

Q2: Describe your intent classification model

a. My model:

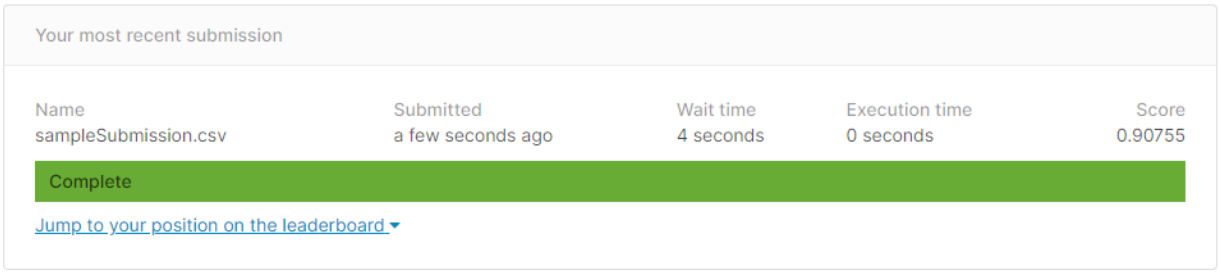
說明：為了將 LSTM 的 OUTPUT 轉化成 150 個 class 好讓我最後能輸出正確的字詞，所以在 LSTM 後加了一層 Linear Layer 將輸出資料輸出成 150 個。

```
class RNN(nn.Module):
    def __init__(self):
        super(RNN, self).__init__()

        self.rnn = nn.LSTM(
            input_size=INPUT_SIZE,
            hidden_size=850,
            num_layers=1,
            batch_first=True,
        )

        self.out = nn.Linear(850, 150)

    def forward(self, x):
        r_out, (h_n, h_c) = self.rnn(x, None)  # x (batch, time_step, input_size)
        out = self.out(r_out[:, -1, :])  # (batch, time_step, input_size)
        return out
```

```
RNN(
  (rnn): LSTM(300, 850, batch_first=True)
  (out): Linear(in_features=850, out_features=150, bias=True)
)
```

b. Performance of my model:

Accuracy:90.755%



c. The loss function I used:

CrossEntropyLoss：

$$cross\_entropy = -\sum_{k=1}^{N}(p_k * \log q_k)$$

d. The optimization algorithm, learning rate and batch size:

Optimization algorithm: Adam
Epoch: 40
Learning rate: 0.001
Batch size: 16

Q3: Describe your slot tagging model.

a. My model:

```python
class RNN(nn.Module):
    def __init__(self):
        super(RNN, self).__init__()

        # self.fc1 = nn.Linear(300, 300)

        self.rnn = torch.nn.LSTM(
            input_size=INPUT_SIZE,
            hidden_size=128,
            num_layers=2,
            batch_first=True,
            bidirectional=True,
        )

        # self.relu = nn.ReLU()
        self.h1 = nn.Linear(128*2, 10)
        # self.h2 = nn.Linear(64, 10)

    def forward(self, x):
        r_out, h_state = self.rnn(x, None)  # x (batch, time_step, input_size)
        out = self.h1(r_out)  # (batch, time_step, input_size)
        # out = self.h2(out)
        return out
```

```
RNN(
  (rnn): LSTM(300, 128, num_layers=2, batch_first=True, bidirectional=True)
  (h1): Linear(in_features=256, out_features=10, bias=True)
)
```

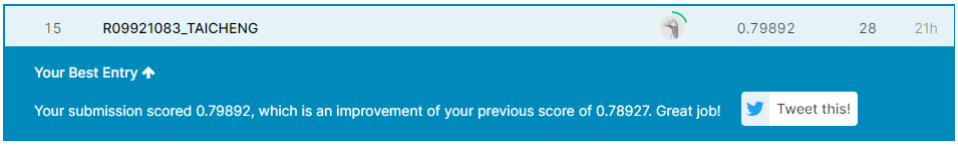
b. Performance of my model:

Accuracy:79.892%

c. the loss function I used:

CrossEntropyLoss：

$$cross\_entropy = -\sum_{k=1}^{N}(p_k * \log q_k)$$



d. The optimization algorithm, learning rate and batch size:

Optimization algorithm: Adam

Learning rate: 0.001

Batch size: 12

Q4: Sequence Tagging Evaluation

```
joint Accuracy = 762 / 1000
Token Accuracy = 7560 / 7891
0.7577565632458234
              precision    recall  f1-score   support

        date       0.75      0.75      0.75       205
  first_name       0.88      0.85      0.87       106
   last_name       0.82      0.85      0.84        75
      people       0.68      0.70      0.69       233
        time       0.75      0.76      0.76       215

   micro avg       0.75      0.76      0.76       834
   macro avg       0.78      0.78      0.78       834
weighted avg       0.75      0.76      0.76       834
```

metrics	description
accuracy_score(y_true, y_pred)	Compute the accuracy.
precision_score(y_true, y_pred)	Compute the precision.
recall_score(y_true, y_pred)	Compute the recall.
f1_score(y_true, y_pred)	Compute the F1 score, also known as balanced F-score or F-measure.
classification_report(y_true, y_pred, digits=2)	Build a text report showing the main classification metrics. ^{} is number of digits for formatting output floating point values. Default value is ^{}.

Evaluation method in seqeval: 可以針對 y_true 和 y_pred 字詞的類別分別給出預測的評分，這樣比如上的圖示就可以看到她在 people 的部分只有 0.68，這樣一來就可以很準確的知道我所訓練的模型它的缺點在哪裡。

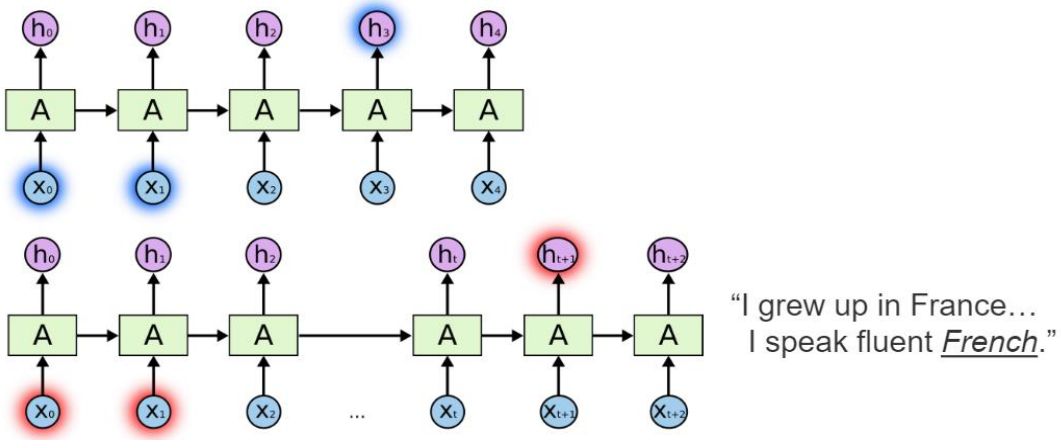
Token accuracy: 比對 y_true 和 y_pred 的字，如果 y_pred 所出現的字和 y_true 一模一樣的話就 +1，如果不一樣的話就不計算，最後將所計算出來的數字除以總共的字數所得到的結果。

Joint accuracy: 比對 y_true 和 y_pred 的每一句話，如果 y_pred 所預測出的一整句話和 y_true 一模 一樣，則 joint accuracy 就+1，最後將統計下來的數字除以 y_pred 總共的句子個數，得到的數字就是 final prediction 的準確率。

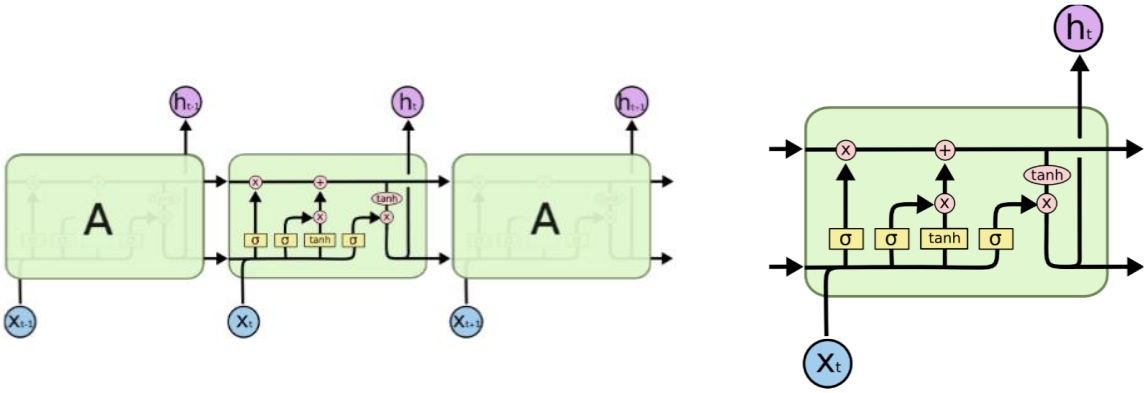
Q5: Compare with different configurations

	LSTM	RNN	GRU
Epoch	80	80	80
Batch size	12	12	12
Best accuracy	79.892%	74.7%	76.7%
Hidden layers	3	3	3
Hidden_size	128	128	128
Bidirectional	True	True	True
Num_layers	2	2	2
CUDA time	1109.3ms	950.22ms	1009.34ms

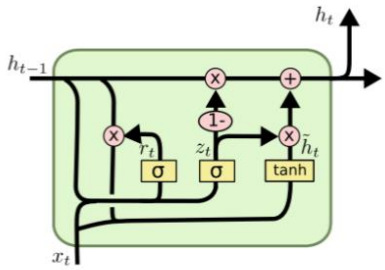
RNN：對長期的依賴關係不敏感（因為梯度消失了），其實就是對長期記憶的丟失



LSTM：分別採用兩大策略來解決上述的缺點。首先，針對梯度消失問題，採用 gate 來解決，效果非常好。而對於短期記憶覆蓋長期記憶的問題， LSTM 採用一個 cell state 來保存長期記憶， 再配合門機制對信息進行過濾，從而達到對長期記憶的控制。



GRU:是 RNN 的另一種變體，其與 LSTM 都採用了門機制來解決上述問題，不同的是 GRU 可以視作是 LSTM 的一種簡化版本，主要繼承了 gate 的機制，可以說 GRU 比較簡化且參數相對較少。



結論：比較各個模型在運行 slot_tags 的效能，發現 LSTM 的準確率最高，其實和 GRU 的差別並不大但，但是在 GPU 上運行的時間相對比較久，而 RNN 的準確率相較來說比較低，但是運行的時間是最快速的。