

NLP Assignment 1 - Named Entity Recognition

109504502 資電三 陳紓嫻

I. Data preprocessing

引用的函式庫

```
import pickle
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
import numpy as np
import preprocessing as p
import re
```



i. 資料集統計

```
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file = '/home/shuxian109504502/NLP_HW2/NLP-1/glove.twitter.27B.100d.txt'
word2vec_output_file = '/home/shuxian109504502/NLP_HW2/NLP-1/glove.twitter.27B.100d.word2vec.txt'
glove2word2vec(glove_input_file, word2vec_output_file)
glove_model = KeyedVectors.load('/home/shuxian109504502/NLP_HW2/NLP-1/glove.twitter.27B.100d.model')
```

一開始先載入 glove 的預訓練模型，初步了解資料集的狀況，像是 unknown word 的計算等等，因為此資料集從 twitter 收集，因此採用 glove-twitter 的預訓練詞向量。

載入資料集後，會發現有很多網址或是 hashtag 等等，會導致 glove 無法產生詞向量，因此在接下來的處理就會以這些東西為主要處理的目標。

ii. 詞向量模型矩陣預處理匯出

```
# #insert '<pad>' and '<unk>' tokens at start of vocab_npa.
vocab_npa = np.append(vocab_npa, '<pad>')
vocab_npa = np.append(vocab_npa, '<unk>')
vocab_npa = np.append(vocab_npa, '<STA>')
vocab_npa = np.append(vocab_npa, '<END>')
print(vocab_npa[-10:])

pad_emb_npa = np.zeros((1, embs_npa.shape[1])) #embedding for '<pad>' token.
unk_emb_npa = np.mean(embs_npa, axis=0, keepdims=True) #embedding for '<unk>' token.
sta_emb_npa = np.arange(0, 1, 0.01, dtype="float64")
end_emb_npa = np.arange(-1, 0, 0.01, dtype="float64")
```

由於詞向量本身並無<unk>,<pad>,<STA>,<END>等單字，因此將詞向量載入後，加入新的詞向量以及新的單字到字典裡，將詞向量已單單自存入 npy 檔方便之後直接使用。

iii. 單字詞典預處理及匯出

由於單字矩陣無法將之後轉為數字的 text，轉換回來因此一樣先將單字矩陣轉換為字典類型，方便之後轉換，儲存到 pickle 檔，方便之後直接匯出。

iv. 資料集預處理及匯入

使用 tweet-preprocessor 將一些網址、hashtag、reserved word 做轉換，方便將單字轉換為 glove 可使用的字詞，並且對資料集做 padding，還有篩選掉句子長度太短的資料，並且在美劇前後開頭加入標籤<STA>、<END>。

```
def textprocessor(word):
    if re.search("\$HASHTAG\$",word):
        word = re.sub("\$HASHTAG\$", "<hashtag>", word)
    if re.search("\$MENTION\$",word):
        word = re.sub("\$MENTION\$", "<user>", word)
    if re.search("\$RESERVED\$",word):
        word = re.sub("\$RESERVED\$", "rt", word)
    if re.search("\$URL\$",word):
        word = re.sub("\$URL\$", "<url>", word)
    if re.search("\$SMILEY\$",word):
        word = re.sub("\$SMILEY\$", "<smile>", word)
    if re.search("\$d+(K|k*)",word):
        word = re.sub("\$d+(K|k*)", "price",word)
    return word
```

II. Model Architecture

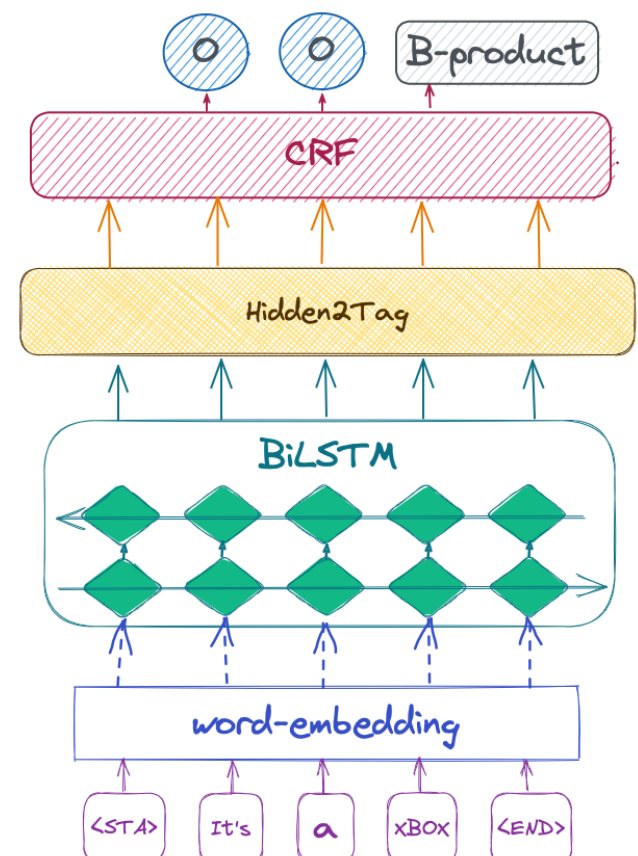
```
self.drop = nn.Dropout(dropout)
#embedding
self.word_embeds = nn.Embedding.from_pretrained(torch.from_numpy(self.embs_vec).float())
#lstm
self.lstm = nn.LSTM(embedding_dim, hidden_dim//2, num_layers = self.num_layer, bidirectional = True)
#LSTM的輸出對應tag空間 (tag space)
self.hidden2tag = nn.Linear(hidden_dim, self.tagset_size)
#CRF
self.crf = CRF(self.tagset_size)
```

本次作業的模型主要使用 Bi-LSTM 架構為主體，一共分為四層

- i. Word-Embedding:
Glove，預訓練詞向量為 twitter-27B-100d
- ii. Bi-LSTM:
num_layer = 1
Hidden_size = 1024*2
- iii. Hidden2Tag:
Hidden_size * Tag_size(21)
- iv. CRF: 使用 pytorch-crf 的函式庫，損失函數在此計算

原先在 word-embedding 及 bilstm 的輸出都做 dropout，但是測試完後發現只對 bilstm 的輸出做 dropout 結果表現會比較好，而 dropout = 0.5 時，效果也會比 dropout = 0.25 好。

而一開始模型的 hidden size 設定為 128，做出的結果發現模型只能偵測兩個種類的 label，之後調到 512*2，發現可偵測的 label 更多了，所以最後調到 1024*2 到表現最好，參數再調上去表現就會下降了。



III. Training process

```
best_f1 = 0
for epoch in range(5):
    train_loader = DataLoader(train_set, batch_size= BATCH_SIZE, shuffle= True)
    score_ = train(epoch, model, optimizer)
    if score_ > best_f1:
        best_model = copy.deepcopy(model.state_dict())
        best_f1 = score_

torch.save(best_model, "/home/shuxian109504502/NLP_HW2/NLP-1/best_model_batch_4.pkl")
```

每跑完一次 epoch 集測試一次 dev 資料集，預先設定 epoch = 10，但發現 epoch=5 時會最好，train_loader 的部分有事先打亂，一開始只在初始化打亂 train-loader，但之後嘗試在每個 epoch 都重新打亂資料集，發現結果比之前更好，每次 epoch 完後會比較 f1-score 較好的模型會留下，做後續 test 預測的部分，optimizer 總共採用兩種，Adam 及 SGD，但 Adam 的效果整體較佳，lr 為 0.001。

IV. Evaluation score

測試方法採用 conlleval.py 執行，由於在 colab 上做執行，無法跑此檔案（跑不出來 QQ），因此更準確的 f1-score 會在模型跑完之後另外下載到本機執行測試，而在模型中先計算 accuracy 和用 flscore 的函式庫做結果的評估，每個 epoch 跑完後都會評估，而整個模型都跑完後會進行最後一次的總評估。

最後測試集分數最好的參數結構為：

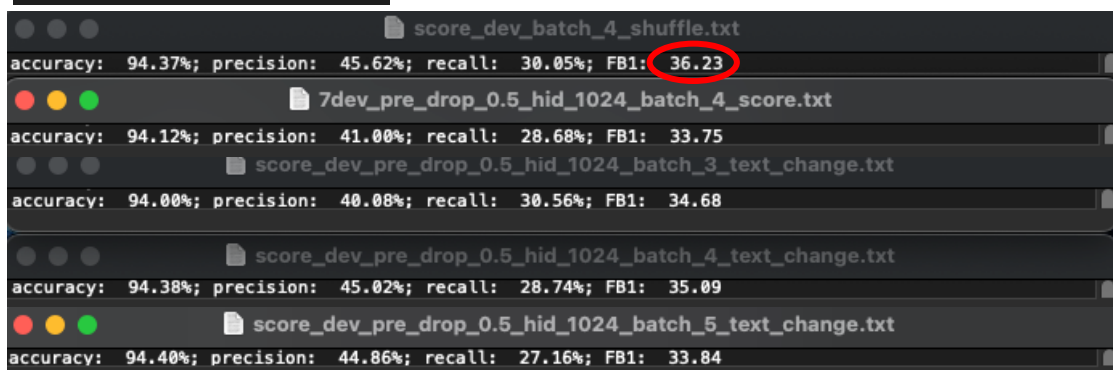
```
HIDDEN_SIZE = 1024*2
EMBEDDING_DIM = 100
BATCH_SIZE = 4
NUM_LAYERS = 1
DROPOUT = 0.5
```

f1-score 可以到達 36.23

accuracy 為 94.37%

precision 為 45.62%

recall 為 30.65%



File Name	accuracy	precision	recall	FB1
score_dev_batch_4_shuffle.txt	94.37%	45.62%	30.05%	36.23
7dev_pre_drop_0.5_hid_1024_batch_4_score.txt	94.12%	41.00%	28.68%	33.75
score_dev_pre_drop_0.5_hid_1024_batch_3_text_change.txt	94.00%	40.08%	30.56%	34.68
score_dev_pre_drop_0.5_hid_1024_batch_4_text_change.txt	94.38%	45.02%	28.74%	35.09
score_dev_pre_drop_0.5_hid_1024_batch_5_text_change.txt	94.40%	44.86%	27.16%	33.84