Applied Deep Learning - Homework 1

資工碩一 R10922005 李澤諺

March 22, 2022

Problem 1

我使用了 Gensim 的 glove-wiki-gigaword-200 作爲 pretrained word embedding，並使用了 spaCy 的 en_core_web_lg 進行 tokenization。首先，我在 Gensim 的 glove-wiki-gigaword-200 中再加入了兩個 word vector，分別對應到 <PAD> 和 <UNK>，兩者皆 inialize 爲 200 維的 zero vector。接著，我會將 input sequence 使用 spaCy 的 en_core_web_lg 進行 tokenization 與 lemmatization，接著，在 intent classification 中，我會將所有 input sequence 的長度皆轉爲 32，意即，若 input sequence 的長度超過 32，則只保留該 input sequence 的前 32 個 token，而若 input sequence 的長度不足 32，則會在該 input sequence 後補上 <PAD> 直到其長度到達 32，而在 slot tagging 之中，則是會在所有的 input sequence 後補上 <PAD> 直到其長度到達 40。最後，我會將每一個 token 使用 glove-wiki-gigaword-200 進行查找，若其在 glove-wiki-gigaword-200 中有對應的 word vector，就將該 token 轉爲其在 glove-wiki-gigaword-200 中對應的 word vector 的 index，反之則將該 token 轉爲 <UNK> 在 glove-wiki-gigaword-200 中對應的 word vector 的 index，以此進行 data preprocesing。

Problem 2

我所實作的 model 其 forward 流程如下：

(1) 將 input tensor 記爲 x，其中 x 的大小爲 $(batch\_size,\ seq\_len)$。

(2) 首先將 x 通過 embedding layer，即 $w = embedding(x)$，其中 w 爲通過 embedding layer 之後所得到的 word vector，其大小爲 $(batch\_size,\ embedding\_dim)$，此外，embedding layer 的 parameter 是使用 Gensim 的 glove-wiki-gigaword-200 來進行 initialization，並會隨著 model 的訓練過程不斷更新。

(3) 接著會將 w 通過一層 linear layer，將其 project 到 512 維，即 $v = linear(w)$，其中 $linear$ 的 input dimension 爲 200，即 word vector 的 dimension，而 output dimension 爲 512。

(4) 接著會將 v 通過 transformer encoder，即 $h = transformer\_encoder(v)$，其中 transformer encoder 中共有 2 層 self-attention layer，而每一層 self-attention layer 的 input dimension 皆爲 512，並皆有 4 個 head。

(5) h 的大小爲 $(batch\_size,\ seq\_len,\ hidden\_dim)$，在得到 h 之後會將其沿著 seq_len 的 dimension 方向做 mean-pooling、min-pooling、max-

pooling，並將所得到的三個 tensor 進行 concatenate，得到一個大小爲 $(batch\_size,\ 3 \times hidden\_dim)$ 的 tensor，將其記爲 $\hat{h}$。

(6) 最後會將 $\hat{h}$ 通過 fully connected feedforward network，即 $y = fc(\hat{h})$，其中 fc 的第一層爲 linear - batch normalization - ReLU - dropout 的架構，其 input dimension 爲 1536，hidden dimension 爲 2048，dropout rate 爲 0.5，而 fc 的第二層只有一層 linear layer，即爲整個 model 的 output layer，故其 output dimension 即爲 class 的數目。

我使用了 Adam 作爲 optimizer，並使用 cross entropy 作爲 loss function，以此訓練 model，其中 batch size 爲 128，learning rate 皆爲 0.0001，訓練了 25 個 epoch，以此得到最後的 model，其在 Kaggle public leaderboard 上所得到的 accuracy 爲 0.93777。

Problem 3

我所實作的 model 其 forward 流程如下：

(1) 將 input tensor 記爲 x，其中 x 的大小爲 $(batch\_size,\ seq\_len)$。

(2) 首先將 x 通過 embedding layer，即 $w = embedding(x)$，其中 w 爲通過 embedding layer 之後所得到的 word vector，其大小爲 $(batch\_size,\ embedding\_dim)$，此外，embedding layer 的 parameter 是使用 Gensim 的 glove-wiki-gigaword-200 來進行 initialization，並會隨著 model 的訓練過程不斷更新。

(3) 接著會將 w 通過一層 linear layer，將其 project 到 512 維，即 $v = linear(w)$，其中 $linear$ 的 input dimension 爲 200，即 word vector 的 dimension，而 output dimension 爲 512。

(4) 接著會將 v 通過 bidirectional LSTM，即 $h,\ c = LSTM(v)$，其中 LSTM 的 input dimension 爲 512，hidden dimension 爲 1024，並且共有 4 層 hidden layer。

(5) 最後會將 h 通過 fully connected feedforward network，即 $y = fc(h)$，其中 fc 的第一層爲 linear - batch normalization - ReLU - dropout 的架構，其 input dimension 爲 2048，hidden dimension 爲 2048，dropout rate 爲 0.5，而 fc 的第二層只有一層 linear layer，即爲整個 model 的 output layer，故其 output dimension 即爲 tag 的數目。

我使用了 Adam 作爲 optimizer，並使用 cross entropy 作爲 loss function (計算 loss 時會忽略 <PAD>)，以此訓練 model，其中 batch size 爲 64，learning rate 皆爲 0.0001，訓練了 50 個 epoch，以此方式訓練了 5 個 model 並進行 ensemble，其在 Kaggle public leaderboard 上所得到的 accuracy 爲 0.80536。

Problem 4

下表爲我於 slot tagging 中訓練出來的 model 其經過 seqeval 所得到的 evaluation 的結果：

	precision	recall	f1-score	support
date	0.79	0.74	0.77	206
first_name	0.91	0.87	0.89	102
last_name	0.86	0.85	0.85	78
people	0.73	0.71	0.72	238
time	0.86	0.88	0.87	218
micro avg	0.81	0.79	0.80	842
macro avg	0.83	0.81	0.82	842
weighted avg	0.81	0.79	0.80	842

Token accuracy 和 joint accuracy 的差別在於，token accuracy 是以 token 爲單位來計算 accuracy，用整個 dataset 之中被正確預測的 token 數量除以整個 dataset 之中所有的 token 數量而得，而 joint accuracy 則是以 sequence 爲單位來計算 acuracy，當一個 sequence 中所有的 token 都被正確預測時，才稱該 sequence 被正確預測，joint accuracy 是將整個 dataset 之中被正確預測的 sequence 數量除以整個 dataset 之中所有的 sequence 數量而得。

Problem 5

在 slot tagging 中，如果只使用 LSTM 對 input sequence 的每個 token 進行預測，可能會產生忽略 tag 之間相關性的問題，例如，若輸入的 input sequence 爲 ['I', 'saw', 'Vivian', 'Chen', 'yesterday']，而 LSTM 的預測結果爲 ['O', 'O', 'B-last_name', 'B-first_name', 'O']，雖然 LSTM 可以根據 input sequence 的上下文正確辨識出'Vivian' 和'Chen' 是屬於人名的一部份，因而將其分別預測爲'B-last_name' 和'B-first_name'，然而 LSTM 的 input 之中並不會有 tag sequence，因此 LSTM 無法學到 tag 前後之間也會有相關性，像是在同一個人名之中'B-last_name' 一定只會出現在'B-first_name' 之後，因此將'Vivian' 和'Chen' 分別預測爲'B-last_name' 和'B-first_name' 其實是錯的。爲了解決這個問題，一個可能的做法是使用LSTM-CRF：在原本的 LSTM 後面再加上 CRF layer，CRF 可以根據 input 的 tag sequence 學到 tag 之間的相關性，進而 decode 出更爲合理的 tag sequence 作爲預測結果。我使用了 pytorch-crf 套件所實作的 CRF layer，將其加入我於 Problem 3 所實作的 model 之中，作爲最後的 output layer，爲了與 Problem 3 做出公平的比較，其它的 data preprocessing、hyper-parameter 等等皆與 Problem 3 相同維持不變，並且同樣訓練了 5 個 model 以進行 ensemble，其最後在 Kaggle public leaderboard 上所得到的 accuracy 從原本的 0.80536 上升到了 0.80857，由此可見 CRF layer 確實對 slot tagging 有所幫助。