ADL HW1 report

Q1. Data processing

a. How do you tokenize the data?

Intent classify:

將每個資料的 text 抽出來，並使用空格當作斷詞符號，並且記錄所有字出現的次數，取出最常出現的 vocab_size(default：10000)個字，接著利用 pre-trained embedding vector 獲得 text 中每個字對應到的 word embedding，並將其存下來作為後續訓練用的資料，若有些字並沒有出現在 pre-trained embedding vector 中的 tokens，那就幫他隨機創造一個 300 維的 word embedding vector（random()*2-1）。且為了訓練方便，將每個 sequence 加上 padding 直到長度達到 max_len（128）。

Label 的部分則是先把每種 intent 記錄下來，並轉成 0~149 的 index。

Slot tag:

將每個資料的 tags 與 tokens 抽出來，並且記錄 tags 的種類與所有 tokens 中出現的字的次數，取出最常出現的 vocab_size(default：10000)個字，接著利用 pre-trained embedding vector 獲得 token 中每個字對應到的 word embedding，並將其存下來作為後續訓練用的資料，若有些字並沒有出現在 pre-trained embedding vector 中的 tokens，那就幫他隨機創造一個 300 維的 word embedding vector（random()*2-1）。且為了訓練方便，將每個 sequence 加上 padding 直到長度達到 max_len（128）。

需要注意的是在處理 slot 的 label 的時候，把 padding 的部分 label 設為 10(因為此次的訓練資料中 tag 的種類只有 9 種)，讓後續在訓練的時候若看到 label 為 10，就會直接忽略此格的 loss 不予計算。

b. The pre-trained embedding you used.

選擇使用 GloVe 的 pre-trained word vectors，Common Crawl(840B tokens, 2.2M vocab, cased, 300d vectors, 2.03GB)的版本。

Q2. Describe your intent classification model

B 在以下代指 batch size，L 代指 sequence 的 max length，E 代指 word embedding dimension，N 代指 RNN-based layer 的層數。

a. Your model

Model 主要分成兩個部分，一是 RNN-based layer，另一則是 fully-connected layer，RNN-based layer 由 bi-LSTM 組成，input_size 為 embedding 的維度（300 維），num_layers 基本上設定成 2，hidden_size（512）、dropout（0.2）為超參數需要調整。而 fully-connected layer 為一層的 Linear layer，輸入的維度為 RNN-based layer 的輸出維度，輸出維度為 num_classes（150 維）。

令一個 batch 的輸入 sequence 為$X = \{x_0, x_1 \dots x_{\max_len}\}$，其維度為 B x L x E。$output$的維度為 B x L x E，$h_n$的維度為 2N x B x E。

$$output, h_n = biLSTM(X, (h_0, c_0))$$

但由於h_0與c_0都並未給定，因此會自動傳入零張量。

$$\bar{h} = concat(\overrightarrow{h_n}, \overleftarrow{h_n})$$

將隱藏層最後兩層合併成 1024 維的特徵張量。

$$\hat{y} = Linear(\bar{h})$$

將 1024 維的特徵張量轉換到 150 維來表示 label 的機率分佈。

b. Performance of your model

	Dev	Public test	Private test
Accuracy	0.9253	0.91377	0.912

vocab_size 使用 10000，因此會使用到全部的字，因為數量最多是 6489，token cover rate 為 5435/6491。

c. The loss function you used:

使用 Cross Entropy loss 與 L2 regularization， $\hat{y}$為 Linear layer 的輸出，y為 ground truth label，N 為 label 的數量，w_，λ為 regularization 的參數。

$$Loss = CrossEntropyLoss(\hat{y}, y)$$

$$= \sum_{n=1}^{N} -y_n \log\left(\frac{\exp(\hat{y}_n)}{\sum_{c=1}^{num_{class}} \exp(\hat{y}_n)}\right) + \lambda * ||w||_2^2$$

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

Optimizer: Adam

Learning rate: 0.001

Batch size: 128

Q3. Describe your slot tagging model

a. Your model

Model 主要分成兩個部分，一是 RNN-based layer，另一則是 fully-connected layer，RNN-based layer 由 bi-GRU 組成，input_size 為 embedding 的維度（300 維），num_layers 基本上設定成 2，hidden_size（512）、dropout（0.2）為超參數可以調整。而 fully-connected layer 為一層的 Linear layer，輸入的維度為 RNN-based layer 的輸出維度，輸出維度為 num_classes（9 維）。

令一個 batch 的輸入 sequence 為$X = \{x_0, x_1 \dots x_{\max_len}\}$，其維度為 B x L x E。$output$的維度為 B x L x E，$h_n$的維度為 2N x B x E。

$$output, h_n = biGRU(X, (h_0, c_0))$$

但由於h_0與c_0都並未給定，因此會自動傳入零張量。

$$\hat{y} = Linear(output)$$

將 1024 維的特徵張量轉換到 9 維來表示 label 的機率分佈。

b. Performance of your model

	Dev	Public test	Private test
Joint Accuracy	0.804	0.73351	

vocab size 為 10000，但最多只有 4115 種字，token cover rate 為 3000/4117。

c. The loss function you used:

使用 Cross Entropy loss 與 L2 regularization， $\hat{y}$為 Linear layer 的輸出，y為 ground truth label，N 為 label 的數量，w_，λ為 regularization 的參數，大致上與 Q2 相同，只有需要另外處理 padding 的 label，因為他們實際上不應該存在，因此在 cross entropy 的設定需要註明，若 label 為 10 的話將不予計算。

$$Loss = CrossEntropyLoss(\hat{y}, y)$$

$$= \sum_{n=1}^{N} -y_n \log\left(\frac{\exp(\hat{y}_n)}{\sum_{c=1}^{num_{class}} \exp(\hat{y}_n)}\right) + \lambda * ||w||_2^2 \ \{y_n \neq ignore\_index\}$$

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

Optimizer: Adam

Learning rate: 0.002

Batch size: 128

Q4. Sequence Tagging Evaluation

IOB2 指的是 tags 的 convention，I 指的是這個 token 在一個 chunk 內(非第一個)，O 指的是這個 token 不在 chunk 內，而 B 用在每個 chunk 的第一個 token 上。

以下為例:

```
y_true:[['B-MISC', 'O', 'B-MISC', 'I-MISC', 'B-MISC', 'O', 'O']]
y_pred:[['O', 'O', 'B-MISC', 'I-MISC', 'B-MISC', 'I-MISC', 'O']]
              precision    recall  f1-score   support

        MISC       0.50      0.33      0.40         3
```

y_true 中有三個 chunk 分別在 0,2-3,4 的位置，而 y_pred 中則只有兩個 chunk 分別在 2-3,4-5。

TP 指的是 true positive，FP 指的是 false positive，TN 指的是 true negative，FN 指的是 false negative。

TP=1，FP=1，FN=2，但直接把 TP+FP 當作該 tag 預測的數量，TP+FN 當作該 tag 實際的數量會更為直觀。

$$precision = \frac{TP}{TP + FP} = \frac{TP}{\textit{預測數量}} = \frac{1}{2}$$

$$recall = \frac{TP}{TP + FN} = \frac{TP}{\textit{實際數量}} = \frac{1}{3}$$

$$F1\ score = \frac{2 * precision * recall}{precision + recall} = \frac{2 * \frac{1}{6}}{\frac{5}{6}} = \frac{2}{5}$$

support 則為該 tag 實際的數量即 recall 的分母，support=3。

micro avg: 將所有 tag 的 TP 總和除以所有 tag 的預測數量或實際數量總和。

macro avg: 計算完所有 tag 的 precision 與 recall 後，取平均。

weighted avg: 計算完所有 tag 的 precision 與 recall 後，以 tag 的數量做 weighted 的平均。

	precision	recall	f1-score	support
date	0.82	0.82	0.82	206
first_name	0.96	0.86	0.91	102
last_name	0.85	0.67	0.75	78
people	0.73	0.73	0.73	238
time	0.83	0.82	0.83	218
micro avg	0.81	0.78	0.80	842
macro avg	0.84	0.78	0.81	842
weighted avg	0.82	0.78	0.80	842

$$Joint\ accuracy = \frac{num\ of\ completely\ correct\ sequences}{num\ of\ sequences} = \frac{0}{1} = 0$$

$$Token\ Accuracy\ = \frac{num\ of\ correct\ tokens}{num\ of\ tokens} = \frac{5}{7}$$

Joint Accuracy 與 Token Accuracy 的評分方式就與上面的有較大的差別，不會去考慮 chunk 而是直接考慮每一個 token 的正確與否，因此若每個 chunk 都錯一點，precision 或 recall 可能分數會很低，但 Token Accuracy 還是能夠很高，然而 Joint Accuracy 應該是最嚴格的，必須要所有的 tags 都需要正確才會算是正確。

Q5. Compare with different configurations

a. Vocab size

1. Intent classification

vocab_size	cover rate	Dev	public test	private test	best epoch
1000	0.9711	0.90433	0.89733	0.90844	70
2000	0.94006	0.919	0.91333	0.91511	44
3000	0.90773	0.926	0.92711	0.92622	70
full	0.83731	0.9253	0.91377	0.912	83

從上表可以發現 vocab_size 設為較為中等的數值，將一些 count 只有 1 的字刪除掉，可以讓 performance 表現好一點，推測是因為有些太偏門的字讓訓練 overfitting 到他們身上，但是也不能將 vocab_size 設的太少，因為會讓太多的字變成需要用 random 去生出 embedding，反而失去了原本使用 pre-trained embedding 的優勢。

2. Slot tagging

vocab_size	cover rate	Dev	public test	private test	best epoch
1000	0.8972	0.855	0.84289	0.8403	43
2000	0.8111	0.843	0.83324	0.85048	34
3000	0.7588	0.847	0.83163	0.84351	23
full	0.7286	0.804	0.73351	0.76902	58

這個結果就十分有趣了，因為將 vocab_size 降低以後，會發現 performance 得到非常大的提升，原因跟 intent 類似，但 vocab_size 降到很低之後，結果似乎沒有受到很大的變化，推測原因是因為有加了一層 dropout 在 embedding 層之後，因此就算用 random 去生出 embedding，但因為 dropout 的緣故還是會少掉一些 embedding，因此某種程度上來說，這個 task 似乎並不像 intent classification 如此依賴 pretrain 的 embedding，只要擁有最常出現的那些字的 embedding，而其他的 word embedding 都可以使用 random 或是 dropout 掉，其結果反而有可能會更好。

b. number of RNN-based layers

1. Intent classification

num of layers	Dev	public test	private test	best epoch
1	0.9247	0.90711	0.91066	49
2	0.9253	0.91377	0.912	83
3	0.9107	0.90488	0.89333	56

從上表可以明顯發現 num of RNN-based layers 設為 2 會最為剛好，太高會 overfitting，太小則是 performance 會稍微比不上 num 為 2 的情況。

2. Slot tagging

num of layers	Dev	public test	private test	best epoch
1	0.779	0.69765	0.73365	54
2	0.804	0.73351	0.76902	58
3	0.772	0.70884	0.73901	26

從上表可以明顯發現 num of RNN-based layers 設為 2 會最為剛好，太高會 overfitting，太小則會有一點 underfitting 的感覺。

c. hidden dimension in RNN-based layers

1. Intent classification

hidden dimension	Dev	public test	private test	best epoch
128	0.9123	0.89155	0.89555	85
256	0.924	0.91066	0.912	94
512	0.9253	0.91377	0.912	83

從上表可以發現 hidden dimension 對此任務的影響比較沒那麼大，當 hidden dim 為 256 與 512 時，表現其實十分接近，但如果太小的時候，模型的表現還是會不如預期。

2. Slot tagging

hidden dimension	Dev	public test	private test	best epoch
128	0.785	0.70026	0.73204	76
256	0.791	0.71849	0.74544	68
512	0.804	0.73351	0.76902	58

從上表可以發現 hidden dimension 對此任務的影響比較沒那麼大，表現其實都十分接近。

d. RNN-based layer type

1. Intent classification

RNN cell	Dev	public test	private test	best epoch
RNN	0.8683	0.85422	0.86177	50
LSTM	0.9253	0.91377	0.912	83
GRU	0.9287	0.916	0.91644	85

從上表可以發現 RNN 相較於另外兩個模型有著明顯的弱勢，不過 LSTM 與 GRU 本就是單純 RNN cell 的加強版，模型也較為複雜，結果較好是可以預期的，而 GRU 是稍微簡化過的 LSTM，因此在面對 overfitting 的問題的時候，GRU 反而能有更好的表現。

2. Slot tagging

RNN cell	Dev	public test	private test	best epoch
RNN	0.772	0.7067	0.73204	32
LSTM	0.8	0.72117	0.75562	96
GRU	0.804	0.73351	0.76902	58

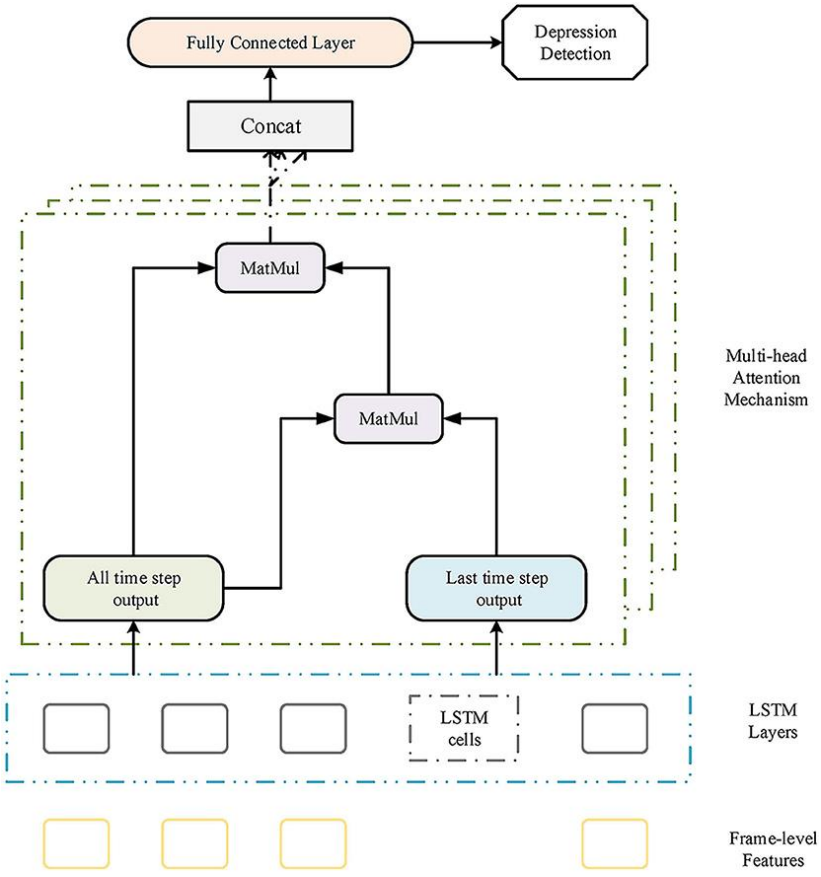
從上表可以發現 RNN 相較於另外兩個模型有著明顯的弱勢，不過 LSTM 與 GRU 本就是單純 RNN cell 的加強版，模型也較為複雜，結果較好是可以預期的，而 GRU 是稍微簡化過的 LSTM，因此在面對 overfitting 的問題的時候，GRU 反而能有更好的表現。

Bonus:

1. Intent classification

參考了”Multi-Head Attention-Based Long Short-Term Memory for Depression Detection From Speech”
(https://www.frontiersin.org/articles/10.3389/fnbot.2021.684037/full)的模型架構，將 LSTM 的 hidden state 與 out 做內積得到一個 weight matrix，接著再將 output 根據這個 weight matrix 取得一個 weighted output，達到 self-attention 的目的。最後將這個 weighted output 當作 fully-connected layer 的輸入，並輸出成 predicted label 的機率分布。

整體模型架構與下圖類似，惟 multi-head 的部分只使用一個 head。原本我們直接將 last hidden state 的資訊直接當作 fully-connected layer 的輸入，因此忽略了 RNN-based layer 的 output，但其實前面的每個 cell 的輸出仍應該與最後的 label 有十分重要的相關性，因此將前面的輸出與 last hidden state 做 attention，應能優化原本只看 last hidden state 的表現。



令一個 batch 的輸入 sequence 為$X = \{x_0, x_1 \dots x_{\max_len}\}$，其維度為 B x L x E。$output$的維度為 B x L x E，$h_n$的維度為 2N x B x E。

$$output, h_n = biLSTM(X, (h_0, c_0))$$

但由於h_0與c_0都並未給定，因此會自動傳入零張量。

$$\bar{h} = concat(\overrightarrow{h_n}, \overleftarrow{h_n})$$

將隱藏層最後兩層合併成 1024 維的特徵張量。$\bar{h}$的維度為 B x E x 1。

此時將 output 與$\bar{h}$做內積並做 softmax。soft attn hidden 的維度為 B x L x 1。

$$soft\ attn\ hidden = softmax(output \cdot \bar{h})$$

接著再將 output 轉成 B x E x L 與 soft attn hidden 做內積並做 softmax 以得到 weighted output。weighted output 的維度為 B x E x 1。

$$weighted\ output = softmax(output, soft\ attn\ hidden)$$

最後將 1024 維的特徵張量轉換到 150 維來表示 label 的機率分佈。

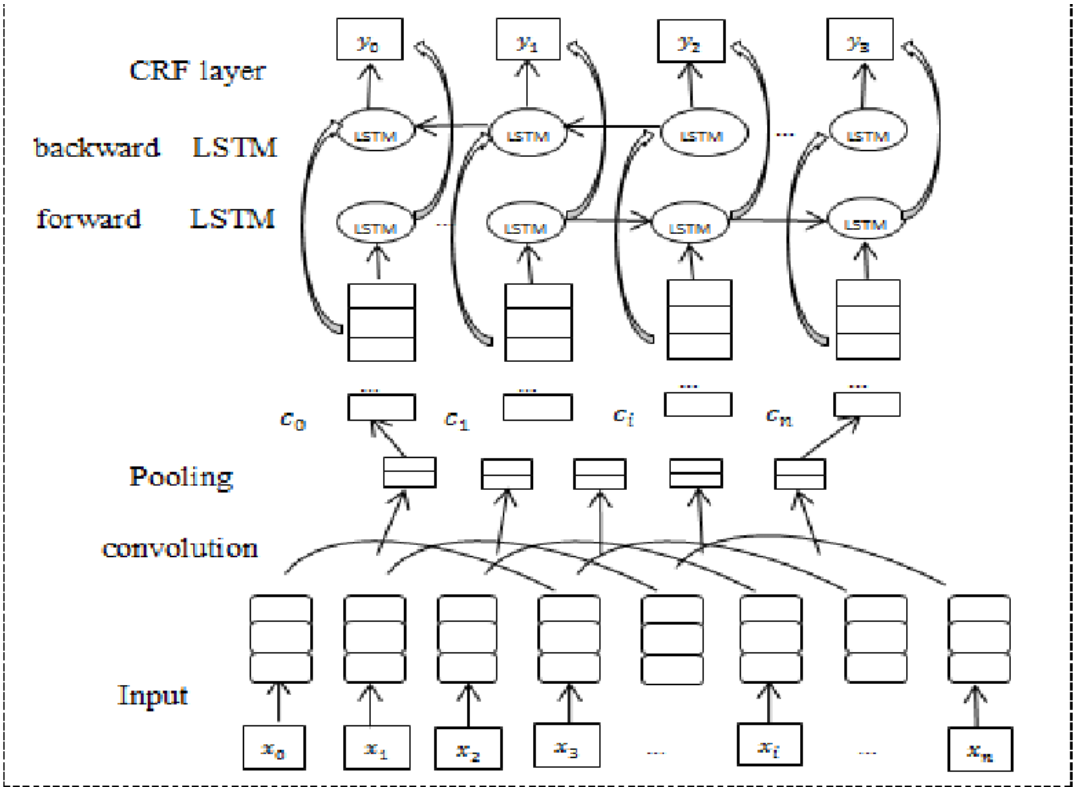
$$\hat{y} = Linear(\bar{h})$$

	Dev	public test	private test	best epoch
w/o attention	0.9253	0.91377	0.912	83
w/ attention	0.9316	0.91822	0.91244	78
w/ attention (tuned)	0.93	0.93155	0.93155	92

從上表可以發現，經過 attention 之後，不管是 validation 或是 test，其 performance 都有略微提高，因此最簡單的 dot-wise attention，仍能夠有效的讓 model 更專注在需要注意的 dimension 上，而自己測試的時候，幾乎所有的參數都能夠經過 attention 而提高模型表現，最後經過調整與使用 learning rate scheduler 後，得到最高分 public 與 private 都 0.93155。

2. Slot tagging

參考 Chinese Named Entity Recognition for Clothing Knowledge Graph Construction(https://www.researchgate.net/publication/336616104_Chinese_Named_Entity_Recognition_for_Clothing_Knowledge_Graph_Construction)的模型架構，如下圖，CRF model 的實作部分使用此連結(https://github.com/jidasheng/bi-lstm-crf/blob/master/bi_lstm_crf/model/crf.py)。



令一個 batch 的輸入 sequence 為$X = \{x_0, x_1 \dots x_{\max_len}\}$，其維度為 B x L x E。過一層 convolution(輸入 channel=輸出 channel=L)，抽取前後文的 feature，CNN ouput 的維度為 B x L x E。

$$cnn\ output = CNN(X)$$

接著將 cnn output 當作 RNN-based layer 的輸入，$output$的維度為 B x L x E，h_n 的維度為 2N x B x E。

$$output, h_n = biGRU(X, (h_0, c_0))$$

但由於h_0與c_0都並未給定，因此會自動傳入零張量。

接著要計算一些條件機率，我們定義 emission 與 transition 兩種 potentials，第 i 個字的 emission potential 來自 bi-LSTM timestep 為 i 的 hidden state，transition 則是存在一個|T| x |T|的矩陣 P 中，T 為 tags 組成的 set，其中$P_{j,k}$代表從 tag k transit 到 tag j 的分數。

$$P(y|output) = \frac{\exp\left(Score(output, y)\right)}{\Sigma_{y'}\exp\left(Score(output, y)\right)}$$

$$Score(output, y) = \Sigma_i log\varphi_{EMIT}(y_i \rightarrow x_i) + log\varphi_{TRANS}(y_i \rightarrow x_i)$$
$$= \Sigma_i h_i[y_i] + P_{y_i, y_{i-1}}$$

最後利用這個分數當作 loss function 來優化。

	valid	public test	private test	best epoch
bi-LSTM	0.804	0.73351	0.76902	58
CNN-bi-LSTM	0.832	0.80857	0.81511	34
bi-LSTM	0.839	0.78659	0.80171	44
CNN-bi-LSTM-CRF	0.819	0.80536	0.81022	68

上表的實驗都是在 vocab_size 為 full 的架構下進行測試，因為若直接調低，就相當於拿掉許多雜訊，就難以體現出此優化想要表達的結果。從上表可以發現，使用 CNN-bi-LSTM 可以先將 embedding 做一次卷積，而卷積可以萃取出附近的 local feature 與減噪的功效，所以某種程度上就類似於將 vocab_size 取小一點，搭配用 dropout 來進行減造的目的，而從結果來看，此方法雖然有效但還是稍微不如直接降低 vocab_size 來的直接有效，而搭配 crf 之後，其效果並沒有疊加起來的感覺，反而有點下降，可能是因為雜訊被消除之後，某種程度上就不太需要再考慮先驗機率，對模型的幫助就不大了，反而有可能造成反效果，而若只有使用 bi-LSTM+CTF，模型表現也有提升，不過可能需要再 tune 一些參數。

最後是透過調整 vocab_size 搭配 dropout，直接調參數調到 public: 0.84396, private: 0.84565。