HW1 Report

姓名：黃浩恩

學號：f05921120

Coworker：賴棹沅、謝忱、嚴聲揚

Q1 Data processing

Q1.1 Describe how do you use the data for rnn.sh, attention.sh, best.sh:

Q1.1.a How do you tokenize the data.

透過使用 nltk Library，並指用 nltk.word_tokenize(sentence)，可以將輸入的句子，將其拆解並輸出 1 個 list，內包含各 token。

Q1.1.b Number of negative samples used to train your model.

目前使用 1 個 Positive sample 跟 4 個 Negative samples。

Q1.1.c Truncation length of the utterances and the options.

目前 utterances 使用 180 字做為訓練長度，options 為 50 字。

Q1.1.d The pre-trained embedding you used.

使用 FastText 的 crawl-300d-2M.vec，含有 2 million word vectors trained on Common Crawl (600B tokens)。

Q2 Describe your RNN w/o attention model

Q2.1 Describe

Q2.1.a your RNN without attention model

將 Context 與各個 Option 丟入 GRU 內，出來的結果再過一個 Linear 與 tanh，最後將 Context 與各個 Option 透過 torch.cat 來做結合，最後再過兩個 Linear，torch.stack 每個選項後輸出。

Q2.1.b performance of your model. (on the public leaderboard, at least)

於 Public Score 為 9.72666，Private Score 為 9.79714。

Q2.1.c the loss function you used.

使用 torch.nn.BCEWithLogitsLoss()。

Q2.1.d The optimization algorithm (e.g. Adam), learning rate and batch size.

Optimization 使用 Adam，learning rate 為 1e-3，batch size 為 16。

Q3 Describe your RNN w/ attention model

Q3.1 Describe

Q3.1.a your RNN with attention model

將 Context 與各個 Option 丟入 GRU 內，出來的結果來做 Attention，取得 Context 與 Option 之關係後，回傳 Attention

Weight，透過 torch.mul 將 Option 與 Attention Weight 融合，再丟入 GRU 內，最後再經過兩層 Linear，torch.stack 每個選項後輸出。

Q3.1.b performance of your model. (on the public leaderboard, at least)

於 Public Score 為 9.59333，Private Score 為 9.61142。

Q3.1.c the loss function you used.

使用 torch.nn.BCEWithLogitsLoss()。

Q3.1.d The optimization algorithm (e.g. Adam), learning rate and batch size.

Optimization 使用 Adam，learning rate 為 1e-3，batch size 為 16。

Q4 Describe your best model

Q4.1 Describe

Q4.1.a your RNN with attention model

將 Context 與各個 Option 丟入 GRU 內，出來的結果來做 Attention，取得 Context 與 Option 之關係後，回傳 Attention Weight，透過 torch.mul 將 Option 與 Attention Weight 融合，再丟入 GRU 內，最後出來的結果再使用 torch.cat 把 Context 整合，再經過兩層 Linear，torch.stack 每個選項後輸出。

Q4.1.b performance of your model (on the public leaderboard, at least)

於 Public Score 為 9.58666，Private Score 為 9.62571。

Q4.1.c the loss function you used.

使用 torch.nn.BCEWithLogitsLoss()。

Q4.1.d The optimization algorithm (e.g. Adam), learning rate and batch size.

Optimization 使用 Adam，learning rate 為 1e-3，batch size 為 32。

Q4.2 Describe the reason you think why your best model is better than your RNN w/ and w/o attention model.

因為透過 attention 後的結果，只找到 attention weight 對應 option 的觀測加權值，而 Best.py 是再將這樣的結果透過最後出來的結果使用 torch.cat 把 Context 整合，把出來的結果跟 Context 做一次的 interaction，故分數有所上升。

Q5 Compare GRU and LSTM

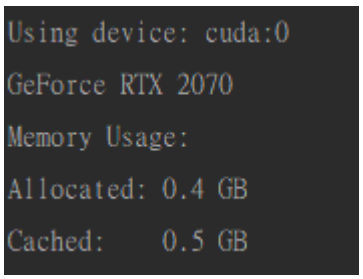
Q5.1 Compare GRU and LSTM models for the following properties

Q5.1.a the recall@10 score (on validation set or public leaderboard).

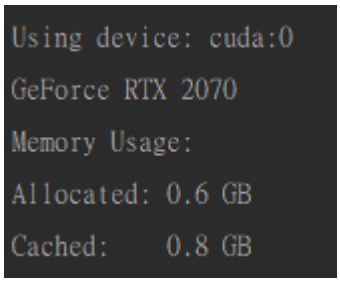
同樣為 Best.py 之程式架構，(因為 Kaggle 次數限制，故採取 Validation data 的 Recall 來做評分)，GRU 的 Recall 分數為：0.61，LSTM 的 Recall 分數為：0.61。由上可知，LSTM 於 Validation data 的 Recall 之成長，沒有明顯的差別。

Q5.1.b required GPU memory.

倘若於 Best.py 中使用 GRU 的模型，Memory 使用情形如下：



若使用 LSTM，Memory 使用情形如下：



如上，可以知道 GRU 較省 GPU 記憶體資源。

Q5.1.c training, testing speed.

倘若使用 GRU 模型，training 速度為 18.72it/s，若使用 LSTM，則

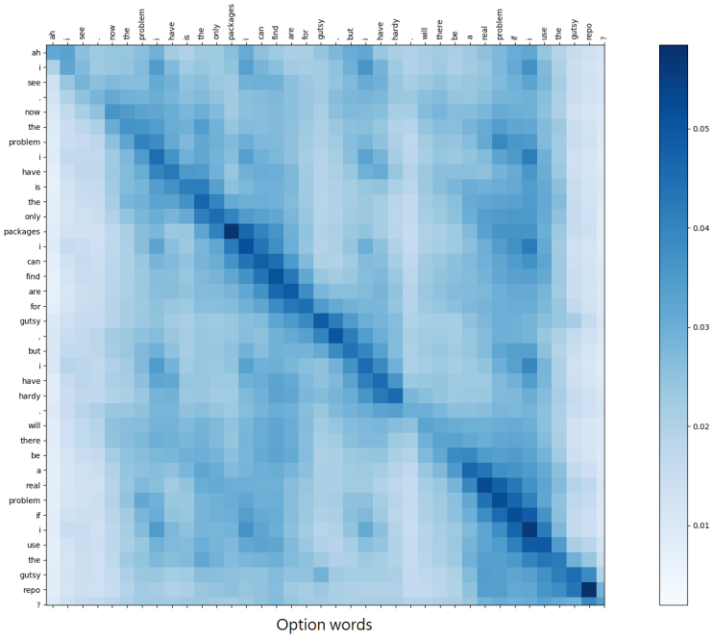
速度/使用模型	GRU	LSTM
training	18.72it/s	12.72it/s
testing	1.18s/it	3.57s/it

Q6 Visualize the attention weights

Q6.1 ake one example in the validation set and visualize the attention weights (after softmax)

Q6.1.a Readable text on the two axises.

Q6.1.b Colors that indicate the value.



Q6.2 Describe your findings.

由圖可以發現，於 option 取出來的字，彼此與鄰近間有區塊被歸類，但是也可以明顯的看出，attention 於此貌似 train 的不太好，沒有像助教一樣那麼明顯的變化。

Q7 Compare different settings.

Q7.1 Compare training with different settings:

Q7.1.a different reasonable loss functions

於本次實作中，實驗了 RMS，BCEWithLogitsLoss，BCE，其中發現 BCEWithLogitsLoss 內建 Sigmoid，讓一開始時做很卡，查了 pytorch 官方網站才發現。

但是使用 RMS 與 BCE 效果都沒有很好，加上最後是 0 跟 1 比較，所以考慮 BCEWithLogitsLoss。

Q7.1.b different number of negative samples

一開始實作使用 1 對 4 錯，但是發現雖然 Recall 很差，於是使用 1 比 9，效果有提升。

Q7.1.c different number of utterances in a dialog

原本將所有的字句丟入，但最後僅採最後 3 句話，效率提升。

Q7.1.d different pre-trained word embeddings

有使用 SPACY，但是一開始因為不熟悉，沒有將不必要的功能關閉，使得 train 過於緩慢，之後更換 FastText，快了許多。