



Hw1 report

B12902007 林映辰

我在 kaggle 手動選擇了提交我最後一份訓練出的模型，但是系統自動選擇了另一份模型進行評分。而我最後的 report 以及所有的 model、code 都是來自於我手動提交的模型，也就是 private score/public score 為 0.81811/0.80116 的這份模型。

| Submission and Description | Private Score ⓘ | Public Score ⓘ | Selected |
|--|-----------------|----------------|-------------------------------------|
|  results.csv Complete · 2d ago · ADL-hw1-10 | 0.81296 | 0.81754 | <input type="checkbox"/> |
|  results.csv Complete · 2d ago · ADL-hw1-11 | 0.81811 | 0.80116 | <input checked="" type="checkbox"/> |

Q1: Data processing

1. Tokenization

我使用了 AutoTokenizer 加載了 luhua/chinese_pretrain_mrc_macbert_large，它是基於 BERT 的 tokenizer。一開始先將句子進行預處理，如移除某些字符、大小寫處理等，接著將每個字詞在 vocab.txt 查找，如果有找到就把該字詞視為一個 token，沒找到就把字詞拆解成更小的單位。填充符號、未知符號、句子開頭、句子結尾、分隔符號、隱藏字詞則會用 [PAD], [UNK], [CLS], [SEP], [MASK] 等 token 來處理。接著在訓練 Multiple Choice 模型時，會先把 first_sentences, second_sentences 變成一維陣列，進行 tokenization 並重組成 tokenized_inputs。

2. Answer Span

在訓練 Extractive QA 模型時，tokenizer 在做 tokenization 時會生成 offset_mapping，代表每個 token 對應句子的開始和結束的位置，接著遍歷 offset_mapping，用 start_char 和 end_char 標示句子的開始及結束位置，接著用 while 迴圈找出對應的 token_start_index 和 token_end_index，也就是句子的 token 開始和結束位置。

接著，利用 token 預測出機率分佈後，找出 n_best_size (在此為 20) 個機率最高的開始和結束位置，接著過濾不合理的輸出，像是長度太大、結束在開始位置前、位置不在 context 中，選擇合理答案中最好的答案，即為 final start/end position。

Q2: Modeling with BERTs and their variants

1. 最後使用的 pre-trained model 為

luhua/chinese_pretrain_mrc_macbert_large。

performance: 0.9714 for train_mc, 0.8425 for train_qa, 0.81811 for kaggle private score, 0.80116 for kaggle public score

loss function: Cross Entropy Loss

optimization algorithm: AdamW

learning rate: 2e-5 for train_mc, 2e-5 for train_qa

batch size: 1 for train_mc, 1 for train_qa

epoch: 2 for train_mc, 3 for train_qa

模型：我分別訓練了 train_mc 和 train_qa 兩個模型完成任務。

train_mc 負責在四個選項中選擇與問題最相關的選項，先加載 pre-trained model 和 tokenizer，接著進行 fine-tuning，首先對四個選項和問題進行 preprocess，first_sentences 放入問題四次，second_sentences 放入每個選項的文字，接著放入模型訓練，每個 epoch 會加載 preprocessed data 並計算 loss 並透過 optimizer 來更新模型最後進行 evaluation。

train_qa 負責在一段文字中擷取與問題最相關的片段。先加載 pre-trained model 和 tokenizer，接著進行 fine-tuning，首先對問題和要擷取的文字進行 preprocess，question 放入問題，context 放入要擷取的文字，接著放進模型進行訓練，每個 epoch 會加載 preprocessed data 並計算 loss 並透過 optimizer 來更新模型最後進行 evaluation，順便在過程中把每 1000 steps 的 loss 和 Exact Match 都紀錄在 learning_curve_data.json，最後進行 post processing 將 token 預測出的機率分佈轉成要擷取的文字。

test.py 則是用於 testing，用 evaluation 的同樣步驟來進行預測最終結果，最後輸出成 prediction.csv。

2. 我也使用了 bert-base-chinese 做為 pre-trained model 來訓練。

model 與上面的結構完全相同，只差在 pre-trained model 和 tokenizer。

pre-trained model 的不同：bert-base-chinese 有進行 random input masking，是基於 BERT 的 Fill-Mask Model，有 12 個 num_hidden_layers、vocab_size 為 21128。luhua/chinese_pretrain_mrc_macbert_large 則是用網路上收集的大量中文 MRC 資料進行訓練，並進行了資料清洗、數據標注、無答案數據構造。

performance: 0.9608 for train_mc, 0.8006 for train_qa, 0.78055 for kaggle private score, 0.77309 for kaggle public score

loss function: Cross Entropy Loss

optimization algorithm: AdamW

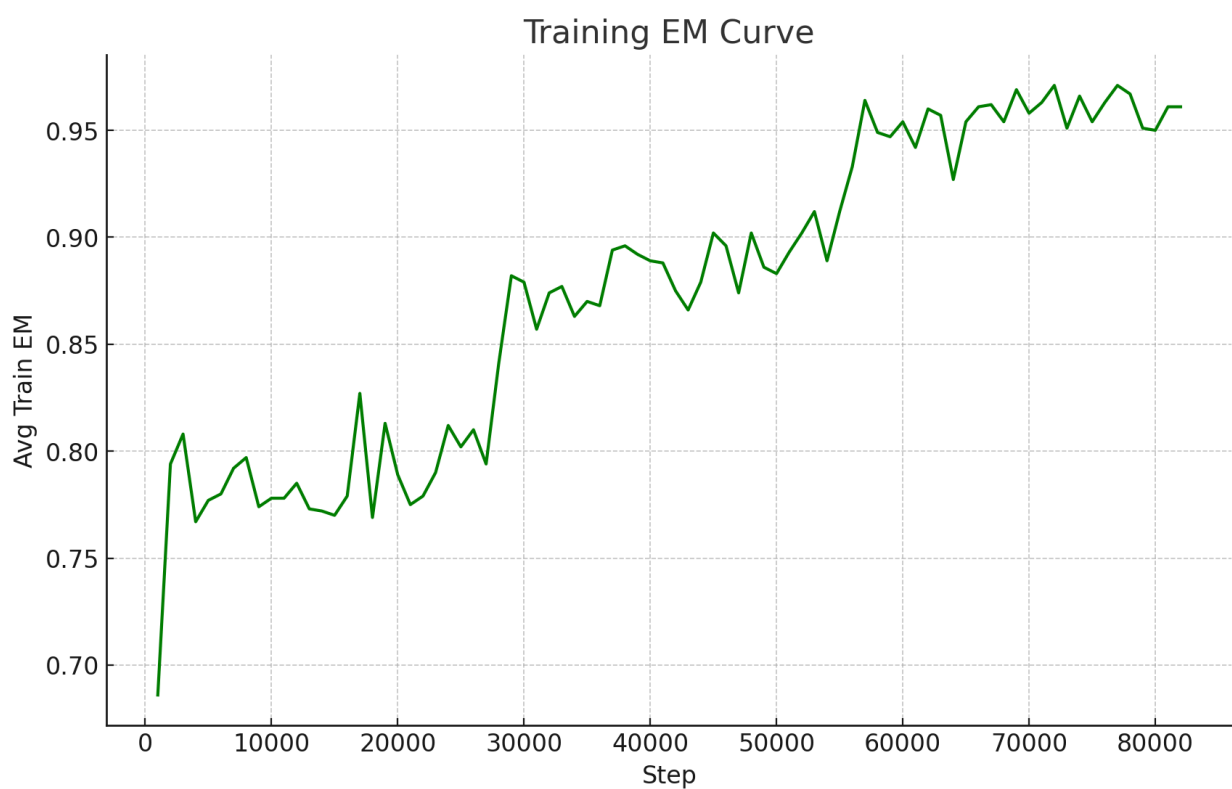
learning rate: $2e-5$ for train_mc, $2e-5$ for train_qa

batch size: 1 for train_mc, 1 for train_qa

epoch: 2 for train_mc, 3 for train_qa

Q3: Curves

我使用了 avg_loss 和 avg_em 來紀錄每 1000 steps 時 loss value 和 Exact Match metric 的平均，最後寫入 learning_curve_data.json 中。



Q4: Pre-trained vs Not Pre-trained

我在 paragraph selection 任務中訓練了 non pre-trained model 的模型。設定如下：

```
config = BertConfig(  
    num_hidden_layers = 6,  
    hidden_size = 384,  
    num_attention_heads = 6,  
    intermediate_size = 1000,  
    max_position_embeddings = 512,  
)
```

learning rate: 2e-5

batch size: 1

epoch: 2

最後的 performance 為 0.5231，表現與 pre-trained model 的 performance 0.9714 相差很多。

Q5: Bonus

基於 Q2-1 的 model，我刪除了 train_mc 的部分，並使用與 train_qa 幾乎一樣的方法來訓練模型，唯一的差別是 context 不再是 train_mc 選出的句子，而是直接把四個句子接在一起直接來訓練。

```
context = [''.join([context_file[pid] for pid in pids]) for pids  
in paragraphs_ids]
```

loss function: Cross Entropy Loss

optimization algorithm: AdamW

learning rate: 2e-5

batch size: 1

但是 performance 相比於原本較不理想，推測是因為沒有換成可以處理較長 input 的模型。