Report

前言：

這次遇到一個很奇怪的情況，明明在 train 和 validation 的分數都有達到 70% 以上，但是經過兩個 task 之後所產生出來的 output file 傳上去卻只有 60% 附近，而且每次產生出來的檔案都一模一樣（照理講應該不太可能），也不是助教後來在公告新增的提醒，因為我有自己換過了。想了很久還是想不出問題出在哪裡，沒有亂動 huggingface example 的架構，作法也是先將我們的 Dataset 調整成和他們一樣的儲存方式，多加一些自己想要看到的數據，例如算出不同地方的Accuracy、Loss 等等，最後還是沒能在時限以前找到出問題的地方（目前推測是可能 Dataloader 拿資料的時候怪怪的，為我有讓他額外 return 一些資料讓後續可以和答案配對），所以很遺憾沒能通過 baseline，會再找時間把問題搞懂，下次作業希望能有好結果。

Q1：Data processing

1. Tokenizer： BertTokenizer
    - tokenize 之後會產生幾種不同的東西：
        1. input_ids：和 HW1 一樣會有一個辭典裡面收錄所有字的編號，把目前的文字內容加上需要的分句或分詞標記後，再根據辭典全部換成編號。
        2. token_type_ids：由 0 或 1 組成，0 代表對應的 token 在第一句， 1 代表對應的 token 在第二句
        3. attention_mask：由 0 或 1 組成，避免將 padding 用於計算 attention （1 是要計算 attention ，0 是不是要計算 attention ）
        4. offset_mapping：因為文章可能會很長，因此可能會被切成數個段落當作訓練材料，也可能存在超出最大長度範圍的問題，因此需要產生 offset_mapping 來將這些被修改過的內容 mapping 到最原始的文本內容。

5. start_positions：利用 offset_mapping，得出的處理過後的文本中答案起始位置
6. end_positions：利用 offset_mapping，得出的處理過後的文本中答案的結尾位置

2. Answer Span
    a. 使用 offset_mapping 找出對應的位置即可
    b. probability of start/end position 會是一個 matrix，選出機率最大的數值即為最有可能的答案位置

Q2：Modeling with BERTs and their variants

Submit version

MC model

- hfl/chinese-roberta-wwm-ext

MC performance

- Accuracy = 0.9621136590229312
- Loss = 0.14429429173469543

MC args

- per_device_train_batch_size 2
- gradient_accumulation_steps 4
- num_train_epochs 1
- learning_rate 3e-5

MC loss_fn & optimizer

- Loss_fn = NllLoss()
- optimizer = AdamW()

MC config

```
Model config BertConfig {
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0, 😀
  "classifier_dropout": null,
  "directionality": "bidi",
  "eos_token_id": 2, 😀
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true, 😀
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

Other version

MC model

- bert-base-chinese

MC performance

- Accuracy = 0.9538052509139249
- Loss = 0.1543034166097641

MC args

- per_device_train_batch_size 1
- gradient_accumulation_steps 2
- num_train_epochs 1
- learning_rate 3e-5

MC loss_fn & optimizer

- Loss_fn = NllLoss()
- optimizer = AdamW()

MC config

```
Model config BertConfig {
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

QA model

- hfl/chinese-roberta-wwm-ext

QA performance

- Accuracy = 80.32568959787305
- Loss = 0.46005253875198726

QA args

- per_device_train_batch_size 4
- gradient_accumulation_steps 2
- num_train_epochs 2
- learning_rate 3e-5

QA loss_fn & optimizer

- Loss_fn = CrossEntropy()
- optimizer = AdamW()

QA config

Same as MC config

QA model

- bert-base-chinese

QA performance

- Accuracy = 75.63974742439349
- Loss = 0.9447763674347937

QA args

- per_device_train_batch_size 1
- gradient_accumulation_steps 2
- num_train_epochs 1
- learning_rate 3e-5

QA loss_fn & optimizer

- Loss_fn = CrossEntropy()
- optimizer = AdamW()

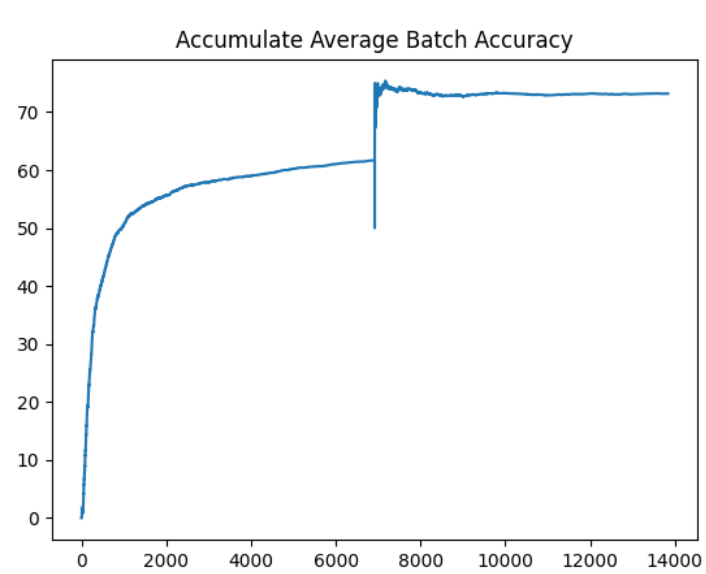
QA config

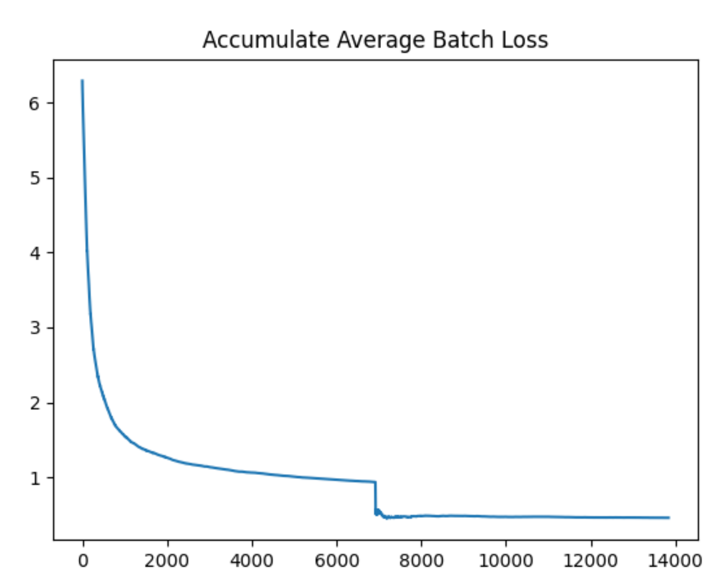
Same as MC config

結論：

config 中尾部有 😀 標註的地方，是 hfl/chinese-roberta-wwm-ext 和 bert-base-chinese 不一樣的地方。兩者在 Multiple Choice 任務中性能差距不大，但在 Question Answering 任務中 Accuracy 可以提升將近 5%。

Q3：Curves





總共跑了兩個 epoch， 因為數量不到 5 點，所以改畫每個batch的累積平均（每個batch都會算一次截至目前為止的平均值並畫一個點）。由於用於累加的變數會在每次進入 training 以前歸零，因此 epoch 0 結束以後模型已經有一定的準確度，epoch 1 累計的起始值比較高，並不會像 epoch 0 一開始會受到很多 accuracy = 0 / loss 很高 的影響。圖中產生垂直跳點就是 epoch 0 和 epoch 1 的交界

Q4：Pretrained vs Not Pretrained

Same model config

```
Model config BertConfig {
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

Same args

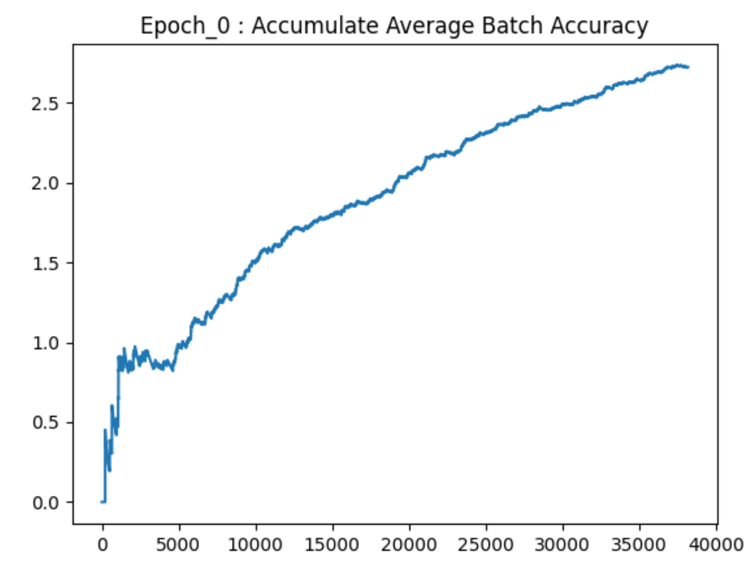
```
accelerate launch run_qa_no_trainer.py \
  --train_file ./Dataset/train.json \
  --validation_file ./Dataset/valid.json \
  --test_file ./QA_sheet.json \
  --context_file ./Dataset/context.json \
  --model_name_or_path bert-base-chinese \
  --max_seq_length 384 \
  --doc_stride 128 \
  --per_device_train_batch_size 1 \
  --gradient_accumulation_steps 2 \
  --num_train_epochs 1 \
  --learning_rate 3e-5 \
  --checkpointing_steps "epoch" \
  --output_dir ./tmp/$DATASET_PATH/ \
  --prediction_csv_dir ./prediction.csv \
```

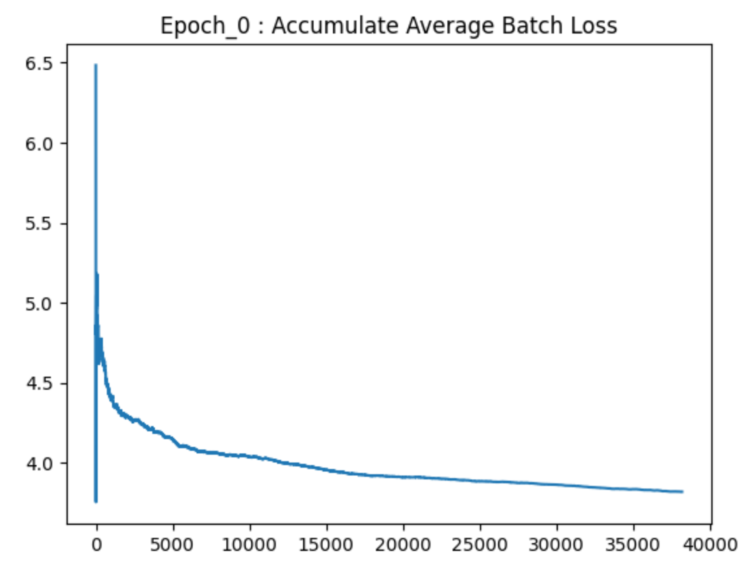
Without pretrain

```
total epochs = 1
total Dataloader load times(step) = 38143
total Optimizer update times (step / gradient_accumulation_step)  = 19072


epoch = 0

=> train:
    epoch_loss = 3.8207039135621215
    epoch_acc = 2.723959835356422
    epoch_best_acc = 2.723959835356422, epoch_best_loss = 3.8207039135621215

=> eval (per epoch) :
    epoch_acc = 4.453306746427384
    epoch_best_acc = 4.453306746427384
```

💡 因為只有 train 1 epoch 所以 Accuracy 和 Loss 都是用累計平均的

結論：

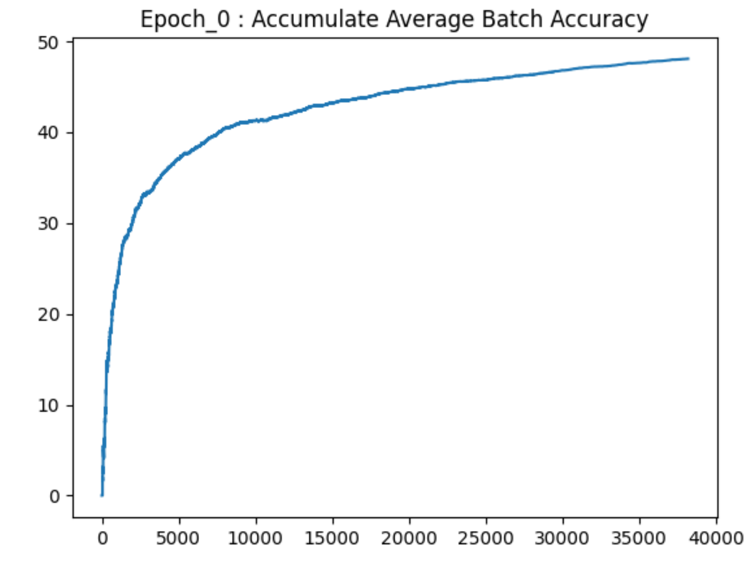


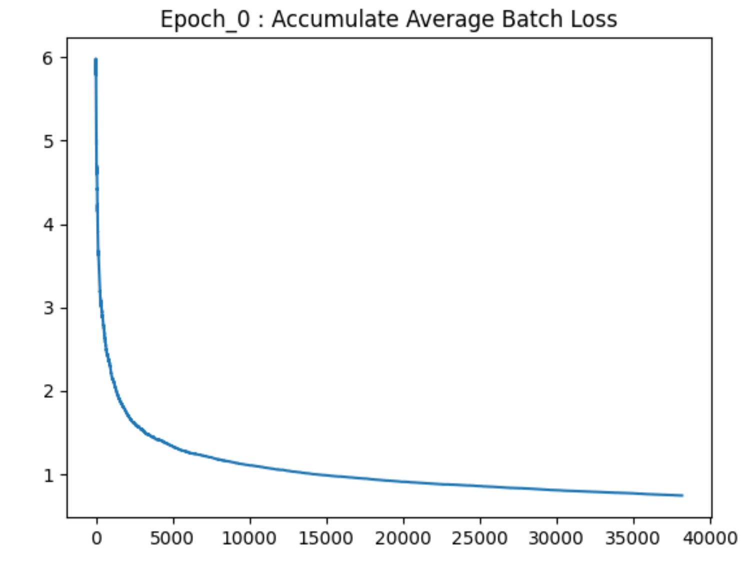


With pretrain

```
total epochs = 1
total Dataloader load times(step) = 38143
total Optimizer update times (step / gradient_accumulation_step)  = 19072


epoch = 0

=> train:
    epoch_loss = 0.7443623440900035
    epoch_acc = 48.092703772645045
    epoch_best_acc = 48.092703772645045, epoch_best_loss = 0.7443623440900035

=> eval (per epoch) :
    epoch_acc = 75.54004652708541
    epoch_best_acc = 75.54004652708541
```

💡 因為只有 train 1 epoch 所以 Accuracy 和 Loss 都是用累計平均的





結論：

沒有 pretrian 基本上 train 不起來，全部設定皆與 "with pretrain" 的一樣，但是 loss 和 accuracy 天差地遠，不知道要 train 多少才可能和有 "with pretrain" 一樣（好像也不太可能因為看圖感覺 loss 已經卡住了）