Q1: Data Processing

- Tokenizer

  我藉由在 training code 裡面印出 tokenizer 的 type 如下：

  <class 'transformers.models.bert.tokenization_bert_fast.BertTokenizerFast'>

  並在 huggingface 中找到對應的程式碼，得知我使用的 pretrain model 的 tokenizer 是 based on WordPiece algorithm。

  WordPiece 和老師上課有介紹到的 BPE 都是屬於 subword tokenization 方法。WordPiece 與 BPE 類似，開始都是將詞切成 character 建立 volcabulay，差別主要在於如何重複合併 vocabulary 中的兩個 subword。不同於 BPE 是考慮 subword 在原句子中出現的頻率，WordPiece 則會考慮 subword 合併後，對於整體 corpus 的「likelihood」。

  「likelihood」是一種統計學概念，表示某事件發生的可能性。而應用在 WordPiece 中，則是指合併後的 subword 出現在 corpus 中的頻率，模型會計算並選擇使 likelihood 最大的 subword 來合併。

  所以比起 BPE，WordPiece 方法計算更複雜，但也更能考慮到語義，較好提升模型的 performance。

- Answer Span
  - How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?
    - 透過 BERT 的 offset_mapping，在 call tokenizer function 時傳入 return_offsets_mapping=True，使得 function 會 return 做完 tokenization 後，每個 token 在原本 text 中的位置 “offset_mapping”。

```
tokenized_examples = tokenizer(
    …
    return_offsets_mapping=True,
    …
)
offset_mapping = tokenized_examples.pop("offset_mapping")
```

      再遍歷 offset_mapping 中的 tokens，

      先透過 training data 中的 answer text 長度與 start 位置，標記目前 span 內的 answer 在原 context 中的開始與結束位置 start_char/end_char

```
start_char = answers["answer_start"][0]
end_char = start_char + len(answers["text"][0])
```

再透過 sequence_id 找到目前 span 在 token 中的開始與結束位置 token_start_index/token_end_index，表示此範圍中包含 answer

```
# Start token index of the current span in the text.
token_start_index = 0
while sequence_ids[token_start_index] != (1 if pad_on_right
else 0):
    token_start_index += 1

# End token index of the current span in the text.
token_end_index = len(input_ids) - 1
while sequence_ids[token_end_index] != (1 if pad_on_right
else 0):
    token_end_index -= 1
```

最後再找出 answer 對應在 token 中的開始與結束位置

```
while token_start_index < len(offsets) and
offsets[token_start_index][0] <= start_char:
    token_start_index += 1
tokenized_examples["start_positions"].append(token_start_index - 1)
while offsets[token_end_index][1] >= end_char:
    token_end_index -= 1
tokenized_examples["end_positions"].append(token_end_index + 1)
```

- After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?
  - 首先根據模型預測出的 answer start_logits 與 end_logits，分別取出前 n_best_size 個（我的設定是 50），由高到低排序，越高代表模型認為越有可能的答案開始與結束位置

```
start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 : -
1].tolist()
end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -
1].tolist()
```

再檢查確保每個 index 不超出 offset_mapping 範圍、對應的 token 為有效、對應的 offset_mapping[]為包含開始與結束位置的值故檢查長度需>=2

```
if (
    start_index >= len(offset_mapping)
    or end_index >= len(offset_mapping)
    or offset_mapping[start_index] is None
    or len(offset_mapping[start_index]) < 2
    or offset_mapping[end_index] is None
    or len(offset_mapping[end_index]) < 2
):
Continue
```

檢查開始的位置在結束位置之前，且不超過設定的最大 answer 長度

```
if end_index<start_index or end_index-start_index+1>max_answer_length:
   continue
```

最後將所有符合條件的 start_logits 與 end_logits 組合成一個預測，並將兩者相加的值作為預測的分數

```
prelim_predictions.append(
{
    "offsets": (offset_mapping[start_index][0],
offset_mapping[end_index][1]),
    "score": start_logits[start_index] + end_logits[end_index],
    "start_logit": start_logits[start_index],
    "end_logit": end_logits[end_index],
}
```

計算所有分數的 softmax，將分數轉為機率

```
scores = np.array([pred.pop("score") for pred in predictions])
exp_scores = np.exp(scores - np.max(scores))
probs = exp_scores / exp_scores.sum()
```

最後再找出所有機率中最佳的答案。在這段程式片段內 version_2_with_negative 是用來考慮無答案的情況，而我們的資料都是有答案的，所以不需要考慮此狀況，找出最高機率的 text 作為最終答案回傳即可。

```
if not version_2_with_negative:
    all_predictions[example["id"]] = predictions[0]["text"]
```

Q2: Modeling with BERTs and their variants

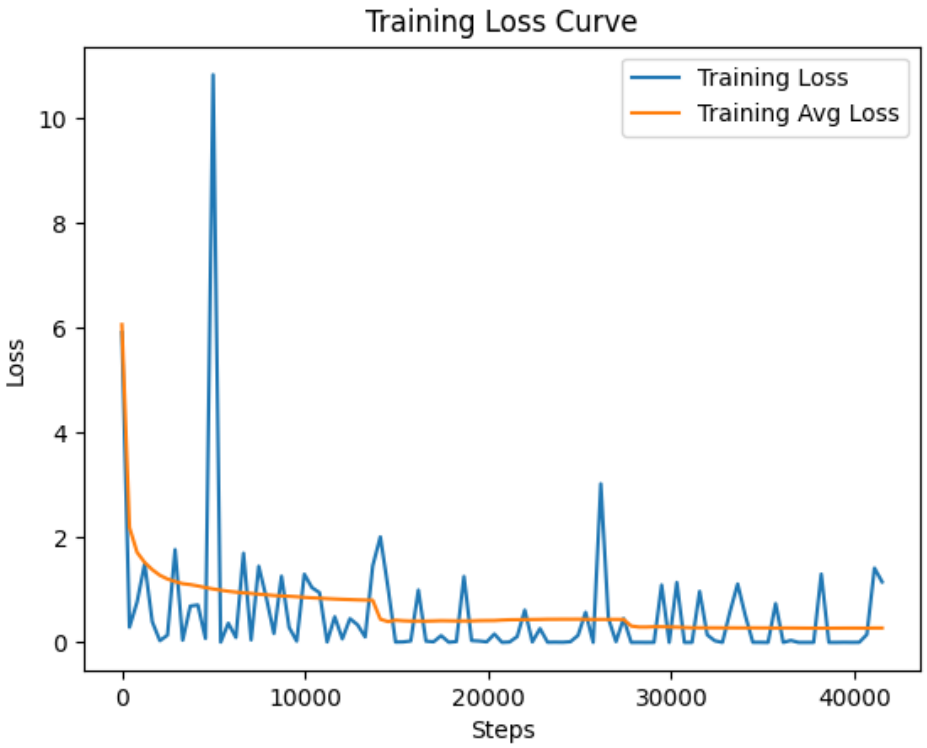
- Describe
  - Model：在 Multi choice 與 QA task 中，我使用的 model 分別為 hfl/chinese-lert-base 與 hfl/chinese-lert-large。LERT 是基於 BERT 的改進，對中文特別進行優化，加強對中文語境的理解。在這兩個任務上，比起使用 BERT，我使用 LERT 可以得到更好的結果，故最後選用 LERT 訓練。
  - Performance：
    - Exact Match (EM) on QA task：約 85.377
    - Accuracy on Multi choice task：約 0.9644
    - Public score on Kaggle：0.8035
  - Loss function：在這兩個 task 中，使用的 loss function 均是由模型自動配置的 Cross-Entropy Loss，更適合用在 multi choice 這種分類任務。
  - Optimization algorithm：AdamW(Adam with weight decay)，其中可以藉由調整 weight_decay 參數，藉由在 optimization 過程中對 loss function 的 penalty，避免模型 overfitting。但在我的 training 中沒有特別調整這個參數，讓它維持預設為 0，下次有機會再使用到 AdamW 可能會想試著調整看看。
  - learning rate：1.5e-5，batch size：2 (gradient_accumulation_steps=2，per_device_train_batch_size=1)，epochs=2(multichoice task)/3(QA task)。剛開始我先根據助教給的參數跑，再用老師上課提過的這篇 paper 中提到的規律，去調整不同的 learning rate 與 batch size 組合，最後選用這個組合是我能得到最好 performance。
- Try another type of pre-trained LMs and describe
  - Model：bert-base-chinese
  - Performance：
    - Exact Match (EM) on QA task：約 80.22
    - Accuracy on Multi choice task：約 0.958
  - The difference between pre-trained LMs (bert-base-chinese, chinese-lert-base)

	bert-base-chinese	chinese-lert-base
Model type	BERT	LERT
Architecture	Based on Transformer	Based on Transformer

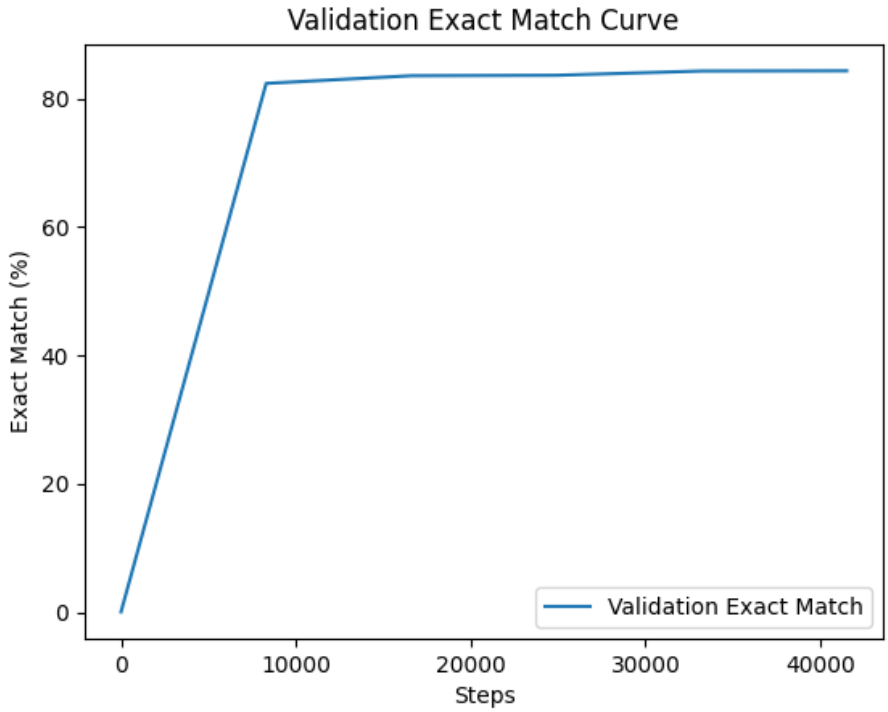
		另外融合 lexicon information 做訓練
Pretraining task	MLM、NSP	MLM、Lexicon-Aware MLM (加入 lexicon 資訊的 MLM)
Pretraining loss	MLM loss: Cross-Entropy loss NSP loss: Cross-Entropy loss	MLM loss: Cross-Entropy loss Lexicon-Aware loss: 加入對 word 邊界資訊的 loss
優點/缺點	適用於多種 NLP task/ 對於中文詞語理解可能較差	更好處理中文語言特性/ 模型較複雜，pretraining 成本高

Q3: Curves

- Learning curve of the loss value 這邊我畫了 average loss 與當下的 loss，共 100 個 data points。當下的 loss 可以看出 loss 在不同 data point 的變動，而 average loss 可以更好看出模型整體學習的進展。



- Learning curve of the Exact Match metric value



EM metric 我記錄了 5 個 data points，從第一次到第二次紀錄間大幅進步，之後便穩定大約都在 80%的位置。

Q4: Pre-trained vs Not Pre-trained

- The configuration of the model and how do you train this model (e.g., hyper-parameters).
  我使用原本 hfl/chinese-lert-large model 的 config，另外更改以下三個值：hidden_size=256、num_hidden_layers=4、num_attention_heads=4，以下為完整 config：

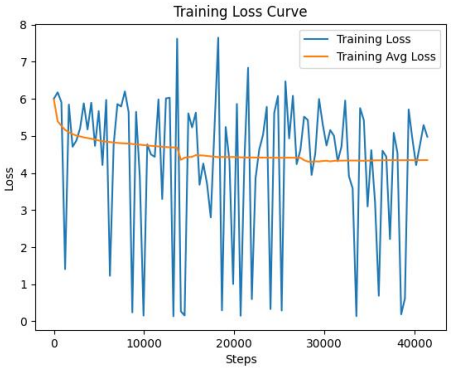

```
{
  "_name_or_path": "hfl/chinese-lert-large",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
```

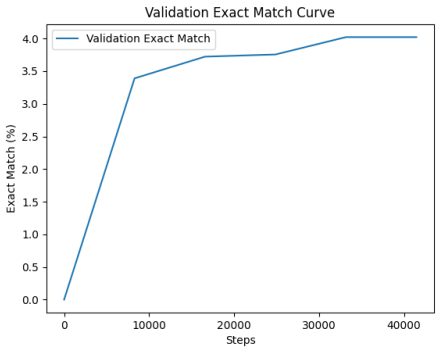
```
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 256,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 4,
  "num_hidden_layers": 4,
  "pad_token_id": 0,
  "pooler_fc_size": 1024,
  "pooler_num_attention_heads": 16,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.45.0.dev0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


tokenizer 和 hyper-parameters 也是使用與原本有 pretrained weights 的相同，可以更好看出兩者差異，以下是我使用的 hyper-parameters：learning rate：1.5e-5，batch size：2 (gradient_accumulation_steps=2，per_device_train_batch_size=1)，max_seq_length=512，n_best_size=50。

我使用原本的 training code 來做訓練，不傳入 model path 參數，使訓練過程中不會使用到 pretrained weight。

- Performance
  - Exact Match (EM)：約 4.02
  - loss curve and EM curve：





Performance 結果與有 pretrained weights 的 model 表現差異極大，可以看到 loss 波動極大，整體來說幾乎沒有甚麼下降，而 EM 值則是稍稍上升後就沒有再往上進步。

Q5: Bonus

- Model：allenai/longformer-base-4096。
- Performance：EM=2.5589
- Loss function：Cross-Entropy Loss
- optimization algorithm：AdamW with learning rate=1.5e-5 and batch size=2

一開始我用的 model 是 hfl/chinese-roberta-wwm-ext，但結果看起來很慘，於是我參考 ChatGPT 的意見，詢問她處理長 context 的 model，換成 longformer-base-4096。我使用原本 train QA task 的 code，在原本的作業中是只餵入 multichoice task 的 output relevant paragraph 去 train QA task，而在 end-to-end 中，差別在於我先將 training data 的四個 paragraphs 串成一整段，直接傳入 model 做訓練，不過結果看起來是失敗了…QQ。

Reference

- report 中使用 ChatGPT 4o 查詢名詞解釋，例如 likelihood、比較 LERT 與 BERT 等。
- 程式碼中使用 ChatGPT 4o 輔助，協助處理 data format、json 檔、幫忙 debug 等。
- Multichoice task training code based on huggingface example code.
- QA task training code based on huggingface example code.