ADL HW1 Report　R12944062 李泓賢

Q1：

(A)

```
"model": {
  "type": "WordPiece",
  "unk_token": "[UNK]",
  "continuing_subword_prefix": "##",
  "max_input_chars_per_word": 100,
  "vocab": {
```

```
"刻": 1174,
"剑": 1175,
"剎": 1176,
"剂": 1177,
"剃": 1178,
"則": 1179,
"剉": 1180,
```

BERT Tokenizer 採用的是”WordPiece”，如左圖，在處理中文文字時，會將中文字一個個做切割，而非一個詞一個詞地切割，如右圖。

```
{
  "cls_token": "[CLS]",
  "mask_token": "[MASK]",
  "pad_token": "[PAD]",
  "sep_token": "[SEP]",
  "unk_token": "[UNK]"
}
```

另外，一些 BERT 的 special token 也可以在 special_tokens_map.json 裡找到。由於我有使用 pad_to_max_length，因此[PAD]就可以起到這樣的作用；[MASK]在 pretrain 時可以用來遮住模型預期要 predict 的 original tokens；[CLS]與[SEP]可以拿來當作句子的開頭以及結尾，就如老師上課說的一樣；[UNK]則是指 BERT vocabulary 沒看過的單字。

(B1)

```python
# Let's label those examples!
tokenized_examples["start_positions"] = []
tokenized_examples["end_positions"] = []

for i, offsets in enumerate(offset_mapping):
    # We will label impossible answers with the index of the CLS token.
    input_ids = tokenized_examples["input_ids"][i]
    cls_index = input_ids.index(tokenizer.cls_token_id)

    # Grab the sequence corresponding to that example (to know what is the context and what is the question).
    sequence_ids = tokenized_examples.sequence_ids(i)

    # One example can give several spans, this is the index of the example containing this span of text.
    sample_index = sample_mapping[i]
    answers = examples[answer_column_name][sample_index]
    # If no answers are given, set the cls_index as answer.
    if answers == None:
        tokenized_examples["start_positions"].append(cls_index)
        tokenized_examples["end_positions"].append(cls_index)
    else:
        # Start/end character index of the answer in the text.
        start_char = answers["start"]
        end_char = start_char + len(answers["text"])

        # 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

        # Detect if the answer is out of the span (in which case this feature is labeled with the CLS index).
        if not (offsets[token_start_index][0] <= start_char and offsets[token_end_index][1] >= end_char):
            tokenized_examples["start_positions"].append(cls_index)
            tokenized_examples["end_positions"].append(cls_index)
        else:
            # Otherwise move the token_start_index and token_end_index to the two ends of the answer.
            # Note: we could go after the last offset if the answer is the last word (edge case).
            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)
```

如上圖，一開始我們會擁有 start_char 與 end_char 兩個 index，也就是每個 token 對應原本 char 的起始以及結尾處，接著利用這兩個 index 作為判斷標準，從頭往後以及從後往回找到 token_start_index 以及 token_end_index，即可得出 start_positions 以及 end_positions。

(B2)

```
# Compute the softmax of all scores (we do it with numpy to stay independent from torch/tf in this file, using
# the LogSumExp trick).
scores = np.array([pred.pop("score") for pred in predictions])
exp_scores = np.exp(scores - np.max(scores))
probs = exp_scores / exp_scores.sum()

# Include the probabilities in our predictions.
for prob, pred in zip(probs, predictions):
    pred["probability"] = prob

# Pick the best prediction. If the null answer is not possible, this is easy.
if not version_2_with_negative:
    all_predictions[example["id"]] = predictions[0]["text"]
else:
    # Otherwise we first need to find the best non-empty prediction.
    i = 0
    while predictions[i]["text"] == "":
        i += 1
    best_non_null_pred = predictions[i]

    # Then we compare to the null prediction using the threshold.
    score_diff = null_score - best_non_null_pred["start_logit"] - best_non_null_pred["end_logit"]
    scores_diff_json[example["id"]] = float(score_diff)  # To be JSON-serializable.
    if score_diff > null_score_diff_threshold:
        all_predictions[example["id"]] = ""
    else:
        all_predictions[example["id"]] = best_non_null_pred["text"]
```

會將每個 logit 算是一個 pair，即(start_logit, end_logit)，作為一個可能的答案。接著如上圖，前三行有點類似將各個 logit 的 scores 做正規化，進而求出各自的機率 probs，接著就可以找到最大機率值，將其 span 作為答案。最後再將這個 span 的 start_position 與 end_position 反推找到其在原本 context 的位置。

Q2:

(A)

Model for MC and QA: hfl/chinese-lert-base

https://huggingface.co/hfl/chinese-lert-base

Performance:

During QA Validation:

```
10/21/2023 11:23:01 - INFO - __main__ - ***** Running training *****
10/21/2023 11:23:01 - INFO - __main__ -   Num examples = 27675
10/21/2023 11:23:01 - INFO - __main__ -   Num Epochs = 5
10/21/2023 11:23:01 - INFO - __main__ -   Instantaneous batch size per device = 4
10/21/2023 11:23:01 - INFO - __main__ -   Total train batch size (w. parallel, distributed & accumulation) = 8
10/21/2023 11:23:01 - INFO - __main__ -   Gradient Accumulation steps = 2
10/21/2023 11:23:01 - INFO - __main__ -   Total optimization steps = 17300
100%|████████████████████████████████████| 17300/17300 [2:25:50<00:00,  2.45it/s]1
0/21/2023 13:48:52 - INFO - __main__ - ***** Running Evaluation *****
10/21/2023 13:48:52 - INFO - __main__ -   Num examples = 3941
10/21/2023 13:48:52 - INFO - __main__ -   Batch size = 1
100%|████████████████████████████████████| 3009/3009 [00:10<00:00, 280.10it/s]
10/21/2023 13:50:39 - INFO - __main__ - Evaluation metrics: 0.8185443668993021██ | 2992/3009 [00:10<00:00, 287.32it/s]
Configuration saved in E:\Neil\ADL\ADLhw1\qa_lert_base_5_epoch_per_device_train_batch_size_4\config.json
Model weights saved in E:\Neil\ADL\ADLhw1\qa_lert_base_5_epoch_per_device_train_batch_size_4\pytorch_model.bin
tokenizer config file saved in E:\Neil\ADL\ADLhw1\qa_lert_base_5_epoch_per_device_train_batch_size_4\tokenizer_config.js
on
Special tokens file saved in E:\Neil\ADL\ADLhw1\qa_lert_base_5_epoch_per_device_train_batch_size_4\special_tokens_map.js
on
100%|████████████████████████████████████| 17300/17300 [2:27:38<00:00,  1.95it/s]

(hw1) E:\Neil\ADL\ADLhw1>
```

EM for validation data: 0.8185

On Kaggle:

QA_lert_base_5_epoch_per_device_train_batch_size_4.csv Complete · 1d ago	0.77326	0.78571

EM for testing data on Kaggle: 0.773(private), 0.785(public)

Loss Function:

torch.nn.CrossEntropyLoss()

Optimization Algorithm:

AdamW()

Epoch, Learning Rate, and Batch Size:

Epoch: 1 for mc and 5 for qa

Learning Rate:3e-5

Total Batch Size:4*2=8 (args.per_device_train_batch_size = 4

--gradient_accumulation_steps = 2)

(B)

Model for MC and QA: bert-base-chinese

https://huggingface.co/bert-base-chinese

Performance:

During QA Validation:

```
10/22/2023 16:22:31 - INFO - __main__ - ***** Running training *****
10/22/2023 16:22:31 - INFO - __main__ -   Num examples = 27643
10/22/2023 16:22:31 - INFO - __main__ -   Num Epochs = 5
10/22/2023 16:22:31 - INFO - __main__ -   Instantaneous batch size per device = 4
10/22/2023 16:22:31 - INFO - __main__ -   Total train batch size (w. parallel, distributed & accumulation) = 8
10/22/2023 16:22:31 - INFO - __main__ -   Gradient Accumulation steps = 2
10/22/2023 16:22:31 - INFO - __main__ -   Total optimization steps = 17280
100%|████████████████████████████████████████| 17280/17280 [2:23:45<00:00,  2.43it/s]1
0/22/2023 18:46:17 - INFO - __main__ - ***** Running Evaluation *****
10/22/2023 18:46:17 - INFO - __main__ -   Num examples = 3934
10/22/2023 18:46:17 - INFO - __main__ -   Batch size = 1
100%|██████████████████████████████████████████| 3009/3009 [00:28<00:00, 105.31it/s]
10/22/2023 18:48:38 - INFO - __main__ - Evaluation metrics: 0.7813226985709538███| 3005/3009 [00:28<00:00, 109.82it/s]
Configuration saved in E:\Neil\ADL\ADLhw1\QA_BERT\config.json
Model weights saved in E:\Neil\ADL\ADLhw1\QA_BERT\pytorch_model.bin
tokenizer config file saved in E:\Neil\ADL\ADLhw1\QA_BERT\tokenizer_config.json
Special tokens file saved in E:\Neil\ADL\ADLhw1\QA_BERT\special_tokens_map.json
100%|████████████████████████████████████████| 17280/17280 [2:26:07<00:00,  1.97it/s]

(hw1) E:\Neil\ADL\ADLhw1>
```

EM for validation data: 0.7813

On Kaggle:

QA_Bertbase.csv Complete (after deadline) · now	0.73712	0.7613

EM for testing data on Kaggle: 0.737(private), 0.761(public)

Loss Function:

torch.nn.CrossEntropyLoss()

Optimization Algorithm:

AdamW()

Epoch, Learning Rate, and Batch Size:

Epoch: 1 for mc and 5 for qa

Learning Rate:3e-5

Total Batch Size:4*2=8(args.per_device_train_batch_size = 4

--gradient_accumulation_steps = 2)

(same as the model described previously)

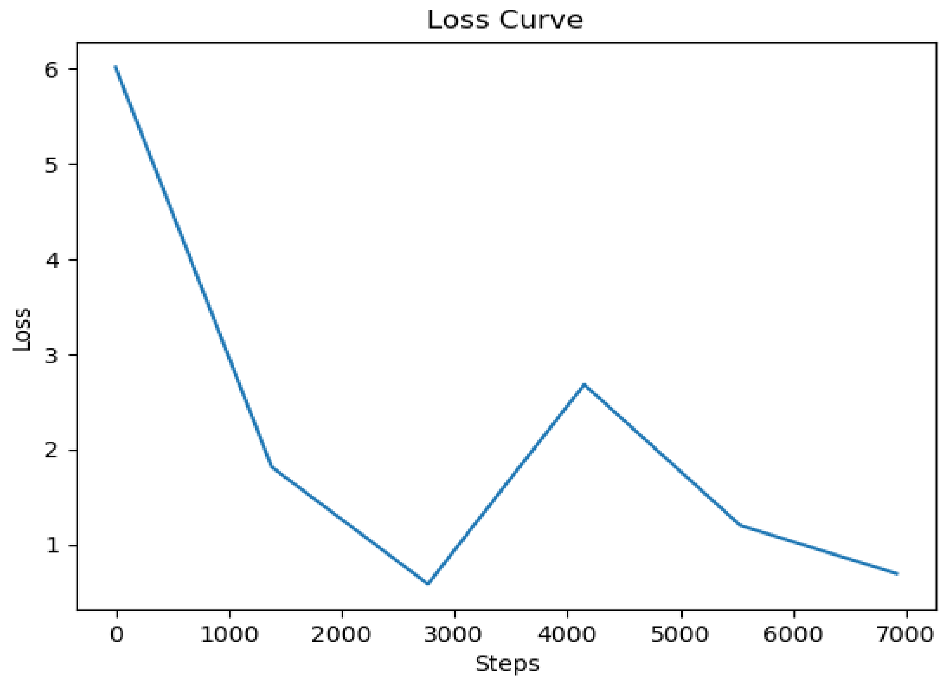
可以看出，chinese-lert-base 與 bert-base-chinese 在相同的 config 下，效果更好。

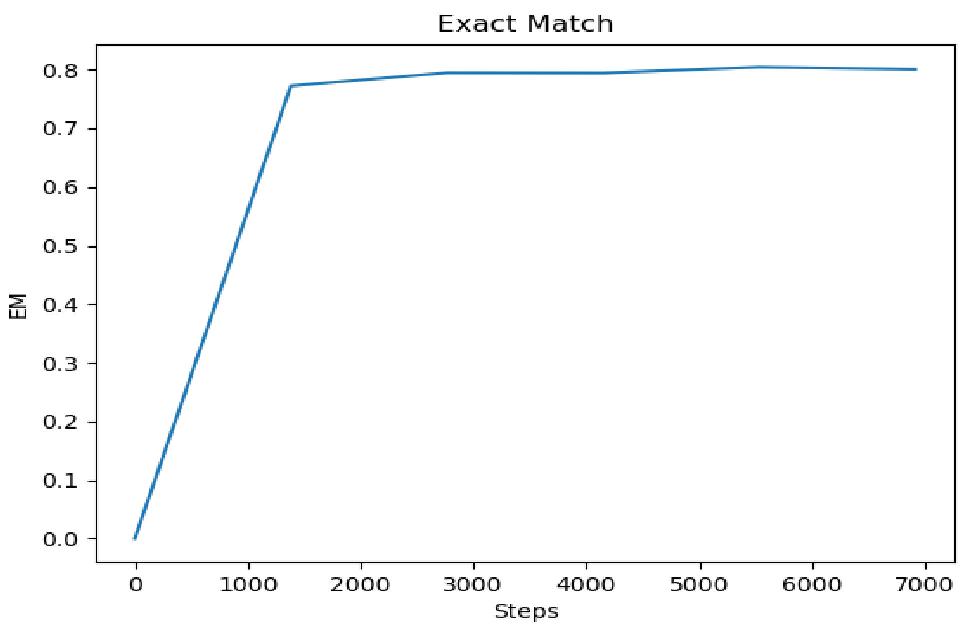
Q3:

For chinese-lert-base:

我的圖是第一個 epoch 的 Train Learning Curve。

X 軸為 steps，意即 train 過程中 model 一次看 4 個 batch 稱為一個 step，因此大約有 27675/4=7000 個 steps。我將總 steps 數除以 5 作為一個 step interval，如此一來就會有 5 個 data points。





For bert-base-chinese:

如前頁所述，相同的方法將 examples 分為 5 個 data points。