ADL23-HW1-Report

- **R12945064 王鴻霖**

Report - Q1: Data processing (2%)

Tokenizer (1%):

Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.

1. MC:

将每个问题复制四次，以便与四个段落配对。每个问题配对一个段落，然后将所有这些配对传递给分词器。使用分词器将文本转换为 BatchEncoding，同时应用`padding`和`truncation`，并设置 max_length 为 512，以确保所有文本对具有相同的长度并适合模型处理。

2. QA:

通过`relevant`变量选择了与每个问题相关的正确上下文。创建文本配对，將每个问题与相应的上下文配对，然后将这些配对传递给分词器。分词器将文本转换为 BatchEncoding 格式，padding , truncation , 并且將 max_length 设置为 512，与 MC 的处理大致相同。

額外设置:

- **`return_offsets_mapping` 和 `return_overflowing_tokens`设置为`True`**，以处理超过 512 个字符的长文本，并保留有关文本如何被截断和分段的信息。
- **`doc_stride` 设置为 256**，在处理超过 512 个字符的长文本时，每个新的分段将从前一个分段的 256 个字符处开始，相當於一個視窗每次向後移動 256 個字符，这样就会有重叠区域，以确保不会丢失可能包含答案的文本。
- **`offset_mapping`** 将字符位置映射到 token 位置以确定答案的确切范围。如果文本被截断或分段，
- **`overflow_to_sample_mapping`** 用于跟踪每个分段与原始文本中的哪个部分相对应，包括一些較長的文本，因爲有時文本長度超過 max_length，框選的範圍內沒有答案，需要通過 overflow_to_sample_mapping 跟踪每个分段与原始文本中的哪个部分相对应。如果没有答案或者 token 的答案范围超出实际答案就會將 start end position 设置为`cls_index`。兩者都發生的話就會将答案的 token 移到正确答案的两端。

Answer Span (1%):

1. **How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?**

以中文句子 "鼓是一种打击乐器" 为例来解释。当对这个句子进行分词时，可能会得到以下的（tokens）：1. 鼓 2. 是 3. 一 4. 种 5. 打 6. 击 7. 乐 8. 器

在这个例子中，每个汉字都被当作一个令牌，所以令牌的位置和字符的位置是一一对应的。例如，token "鼓" 对应的字符位置是 0，token "是" 对应的字符位置是 1，以此类推。如果我们想要知道 token "打击" 对应的字符位置。在原始字符序列中，"打击" 的起始位置是 4，结束位置是 6。但是在 token 序列中，"打" 和 "击" 是分开的，分别是 5 和 6。所以需要使用 offset_mapping。它會为每个 token 提供了其在原始字符序列中的起始和结束位置。我們就可以準確地知道每個 token 在原始字符中的位置，從而能夠準確地把字符上的答案范围起止位置轉換成 token 上的位置。

2. **After your model predicts the probability of answer span start/end position, what**

rules did you apply to determine the final start/end position?

首先把所有可能的開始和結束位置的配對按照他們的分數由高到低排列。然後，check 每一個配對，看看它們是否存在一些問題。例如，結束位置在開始位置之前，或是答案長度過長或過短。如果有就把他們 pass，之後會選擇分數最高且合理的開始和結束位置組合作為最終的答案范围。這樣就可以找到一個既可能是正確答案又合理的答案范围。

Report - Q2: Modeling with BERTs and their variants (4%)

1. Describe (2%)

i. Your model.

MC:


```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "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",
  "torch_dtype": "float32",
  "transformers_version": "4.34.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

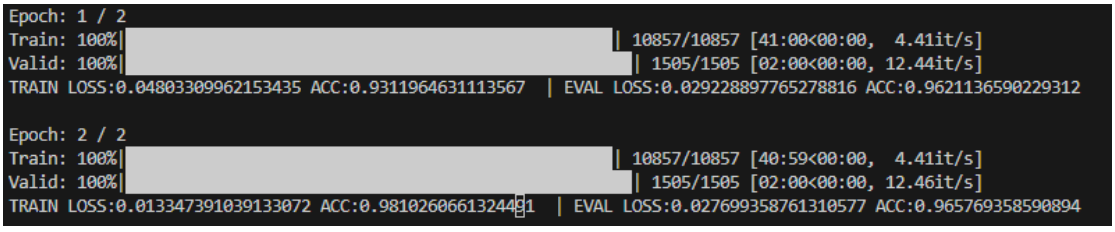

QA:


```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "directionality": "bidi",
```

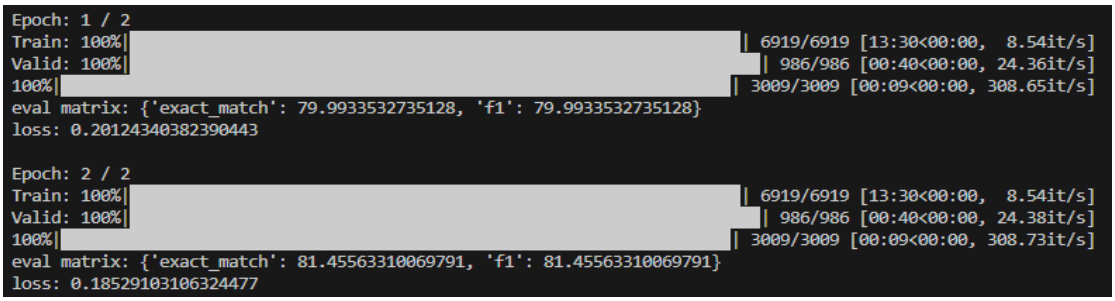
```
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "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",
  "torch_dtype": "float32",
  "transformers_version": "4.34.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


ii.The performance of your model.

MC:



QA:



Kaggle：

iii.The loss function you used.

- **Cross Entropy Loss**

iv.The optimization algorithm (e.g. Adam), learning rate and batch size.

MC&QA:

- Optimization 使用 AdamW
- learning rate = 3e-5
- batch size = 2
- accumulation step = 4
- Scheduler ： get_cosine_schedule_with_warmup
- total_step = len(train_dataset) * num_epoch // (batch_size * accumlation steps)
- warmup_step = total_step * 0.10

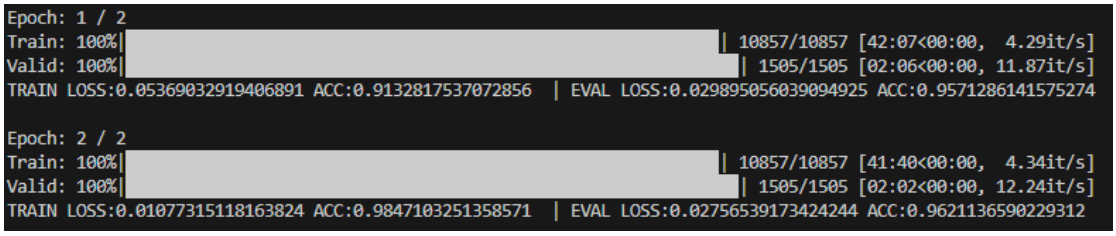
2. Try another type of pre-trained LMs and describe (2%)

i. Your model.

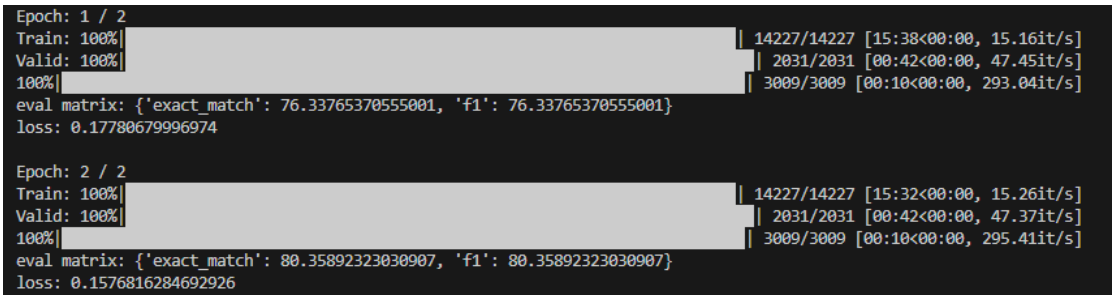
Bert-base-chinese

ii. The performance of your model.

MC



QA



Kaggle

iii. The difference between pre-trained LMs (architecture, pretraining loss, etc.)

预训练策略:

- "bert-base-chinese" 使用标准的 BERT 预训练策略，包括 Masked Language Model (MLM) 和 Next Sentence Prediction (NSP) 两种预训练任务.
- "chinese-roberta-wwm-ext" 使用了 Whole Word Masking (WWM) 策略，而不是原始 BERT 的 Subword Masking 策略. WWM 策略在掩码时会覆盖整个词，而不是词的一部分.

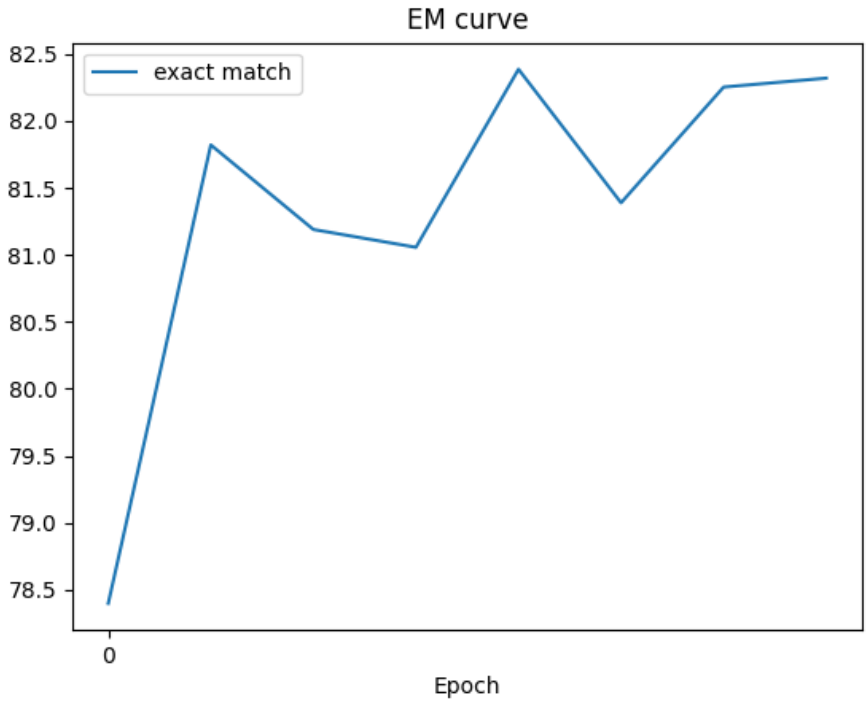
分词方式:

- "bert-base-chinese" 通常使用基于字符的分词方式，将每个中文字符视为一个单独的 token.
- "chinese-roberta-wwm-ext" 使用了更细粒度的分词方式，能够将一些常见的中文词汇保留为单独的 token.

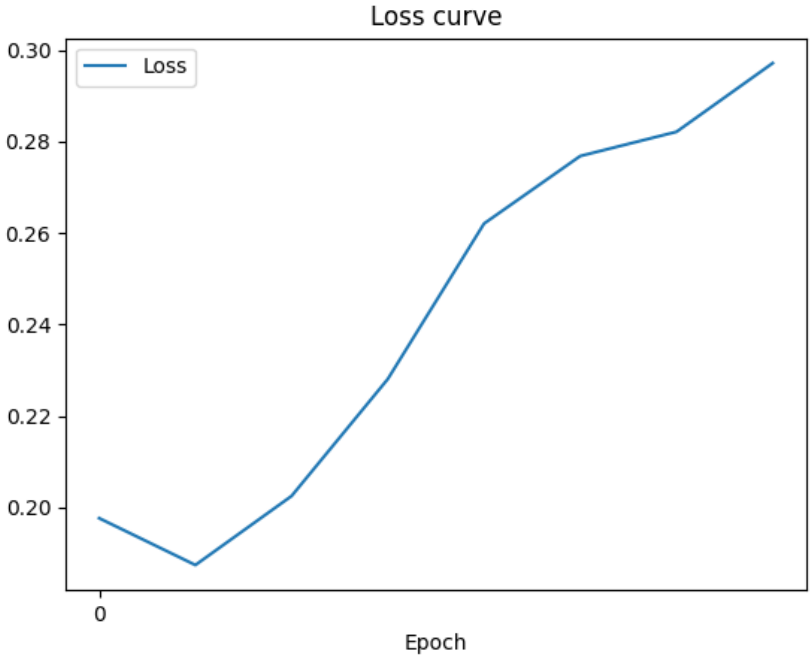
Report - Q3: Curves (1%)

Plot the learning curve of your span selection (extractive QA) model. You can plot both the training and validation set or only one set. Please make sure there are at least 5 data points in each curve.

i. Learning curve of the loss value (0.5%)



ii. Learning curve of the Exact Match metric value (0.5%)



Report - Q4: Pre-trained vs Not Pre-trained (2%)

Train a transformer-based model (you can choose either paragraph selection or span selection) from scratch (i.e. without pretrained weights).

Describe

A:The configuration of the model and how do you train this model (e.g., hyper-parameters).

大部分配置與上述的 model 相同，不同的點在於讀取 model 時：

- **QA**

```
model = AutoModelForQuestionAnswering.from_pretrained(args.model_name_or_path,)
model.resize_token_embeddings(len(tokenizer))
model.to(args.device)
```

- **From scratch**

```
config = AutoConfig.from_pretrained(args.model_name_or_path)
model = AutoModelForQuestionAnswering.from_config(config)
model.resize_token_embeddings(len(tokenizer))
model.to(args.device)
```

The performance of this model v.s. BERT.

- **This model：**

```
Epoch: 1 / 2
Train: 100%| | 14247/14247 [15:41<00:00, 15.12it/s]
Valid: 100%| | 2034/2034 [00:42<00:00, 47.35it/s]
100%| | 3009/3009 [00:10<00:00, 298.03it/s]
eval matrix: {'exact_match': 3.688933200398804, 'f1': 3.688933200398804}
loss: 1.1632747767860527

Epoch: 2 / 2
Train: 100%| | 14247/14247 [15:56<00:00, 14.90it/s]
Valid: 100%| | 2034/2034 [00:43<00:00, 47.14it/s]
100%| | 3009/3009 [00:10<00:00, 295.10it/s]
eval matrix: {'exact_match': 4.719175805915587, 'f1': 4.719175805915587}
loss: 1.1242902477313543
```

- **BERT：**

```
Epoch: 1 / 2
Train: 100%| | 6919/6919 [13:30<00:00,  8.54it/s]
Valid: 100%| | 986/986 [00:40<00:00, 24.36it/s]
100%| | 3009/3009 [00:09<00:00, 308.65it/s]
eval matrix: {'exact_match': 79.9933532735128, 'f1': 79.9933532735128}
loss: 0.20124340382390443

Epoch: 2 / 2
Train: 100%| | 6919/6919 [13:30<00:00,  8.54it/s]
Valid: 100%| | 986/986 [00:40<00:00, 24.38it/s]
100%| | 3009/3009 [00:09<00:00, 308.73it/s]
eval matrix: {'exact_match': 81.45563310069791, 'f1': 81.45563310069791}
loss: 0.18529103106324477
```

可以發現如果不使用 pretrain model 的話，準確率會非常的低。