ADL hw1
R12922121 游諠澐

Q1: Data processing

Tokenizer:

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

I use the model “hfl/chinese-macbert-large” provided by HuggingFace with the HuggingFace API AutoTokenizer to implement my tokenization algorithm. This algorithm will directly create a instance of the relevant architecture tokenizer class, in my case, “BertTokenizer”.
In English, BertTokenizer splits the text into sub-words, the so-called WordPiece tokenization method. However, Chinese words cannot be further divided into subwords, so BertTokenizer splits every individual Chinese word.
If the size of tokenizer is larger than the model’s input embeddings, then token embeddings are resized.

Answer Span:

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

   I set the parameter “return_offsets_mapping” to 1 in Tokenizer, specifically the Bert Tokenizer, I used. The Tokenizer includes “offset_mapping”, which contains all start and end position of each sub-word.
   Here's the breakdown of the process:
   1. Iterate through the "offset_mapping," which means examining all examples in the dataset and going through the token positions.
   2. Obtain the start and end positions in character data.
   3. Locate the start and end positions in the tokenized data.

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?**

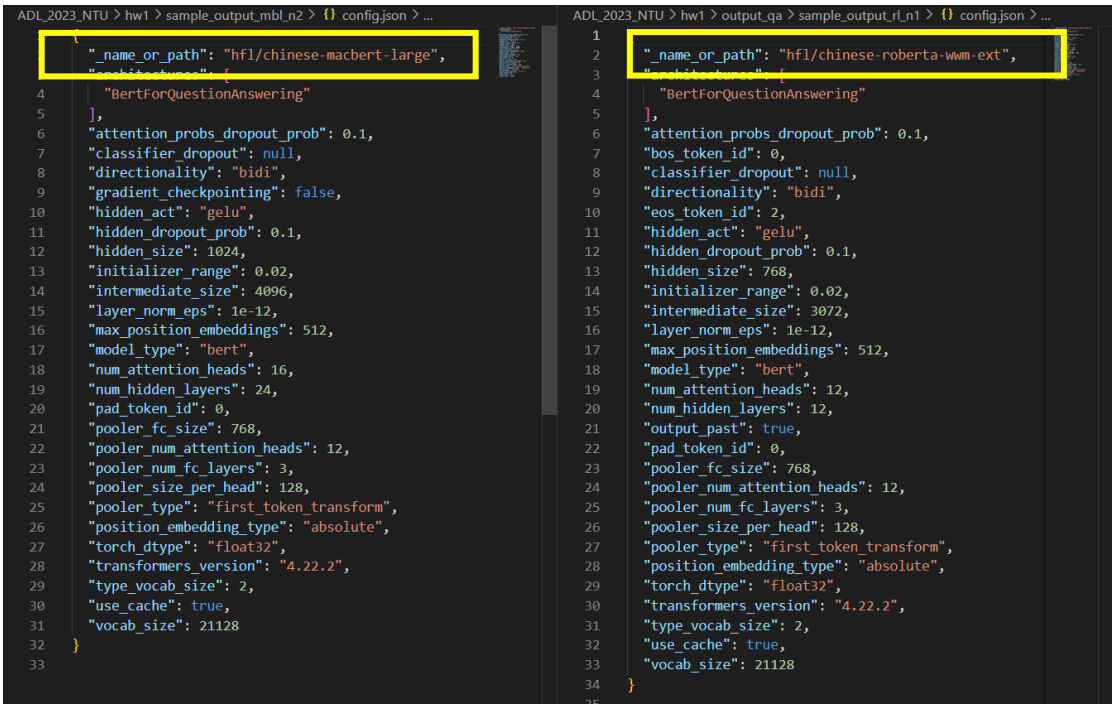
   We will choose the highest probability predictions as answer.
   Do post processing function, which contains below steps:
   (1) For every example,
   (2) Argsort the start indexes and end indexes

(3) Iterate through all the indexes. If the index is rational, save the offsets (from the “offset mapping”), score, start and end logit.
(4) Use the offsets to gather the answer in the original context.

Q2: Modeling with BERTs and their variants

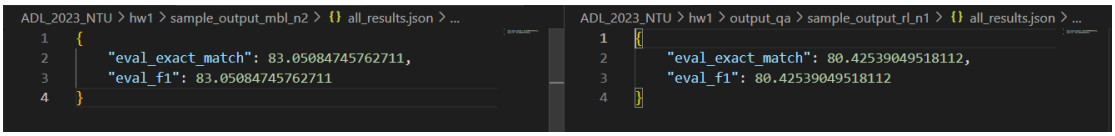
- **Your model:**
  I employ the hfl/chinese-roberta-wwm-ext pretrained model for multiple-choice tasks, whereas I choose the hfl/chinese-macbert-large pretrained model when working on span selection.
- **The performance of your model:**
  a. Multiple choice:
  eval_accuracy: 0.9561316051844466
  b. Question answering:
  eval_exact_match: 83.05084745762711
  eval_f1: 83.05084745762711
- **The loss function you used:**
  Cross entropy loss
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  a. Multiple answer:
    - Optimizer: AdamW
    - Learning rate: 3e-5
    - Batch size: 1
    - Gradient accumulation: 2
  b. Question answering
    - Optimizer: AdamW
    - Learning rate: 3e-5
    - Batch size: 1
    - Gradient accumulation: 2

- **Try another type of pre-trained LMs and describe**
- **Your model:**
  I compared hfl/chinese-macbert-large and hfl/chinese-roberta-wwm-ext, with all other hyperparameters fixed. The left side is hfl/chinese-macbert-large, and the other side is hfl/chinese-roberta-wwm-ext.



- The performance of your model.

  (left: hfl/chinese-macbert-large / right: hfl/chinese-roberta-wwm-ext)



We can observed that the model hfl/chinese-macbert-large is better, because it has a larger size in all hidden size, intermediate size, number of hidden layers. In addition, in the paper “Revisiting Pre-Trained Models for Chinese Natural Language Processing”, they summarized that macbert has a better performance than roberta-wwm-ext.

- **The difference between pre-trained LMs**

  Masking:

  The chinese-roberta-wwm-ext model utilizes Whole Word Masking(WWM) while the chinese-macbert-large model (MLM as correction BERT) both whole word masking as well as N-gram masking strategies for selecting candidate tokens for masking.

  Model size:

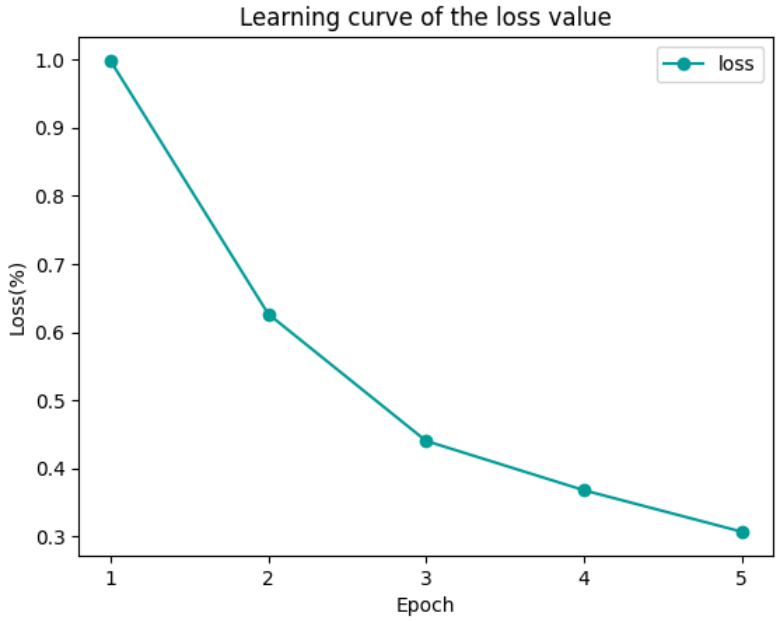
  The chinese-macbert-large model has a hidden size of 1024, while chinese-roberta-wwm-ext only has 768. Additionally, the intermediate size of chinese-macbert-large(4096) is larger than chinesse-roberta-wwm-ext(3072).

Q3: Curves

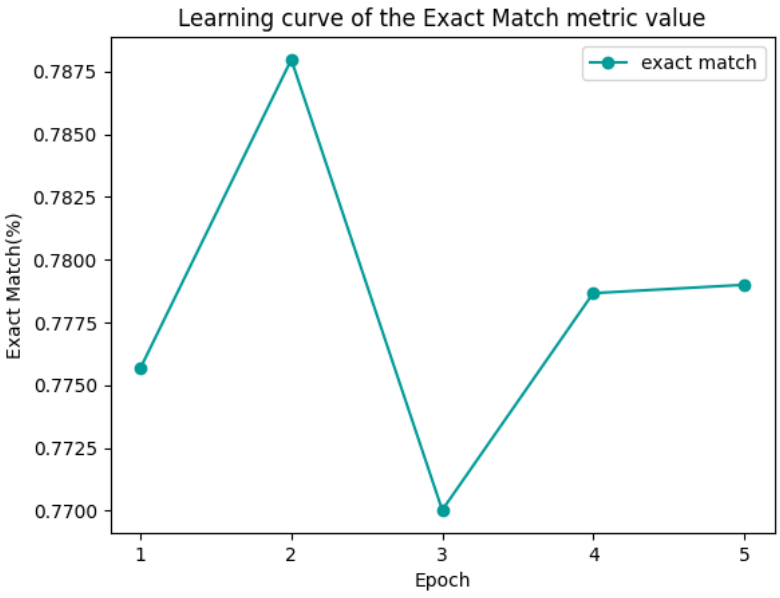
Plot the learning curve of your span selection (extractive QA) model. You can plot both the training and validation set or only one set. Please make sure there are at least 5 data points in each curve.

Learning curve of the loss value (0.5%)

(plot on training set)



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



Q4: Pre-trained vs Not Pre-trained

Train a transformer-based model (you can choose either paragraph selection or span selection) from scratch (i.e. without pretrained weights).
The configuration of the model and how do you train this model (e.g., hyper-parameters).

I switched the model for **multiple choice** portion to a non-pretrained model, still using “hfl/chinese-roberta-wwm-ext”, with keeping the hyperparameters unchanged. The parameters are stated below:

- max_seq_length 512
- per_device_train_batch_size 1
- gradient_accumulation_steps 2
- learning_rate 3e-5
- num_train_epochs 1

- **The performance of this model v.s. BERT.**
  In multiple choice:
  eval_accuracy: 0.021601861083416416
  Compare to the model with pretrained weight, there is a decreases of 97.7492% in accuracy from the original model to this model.

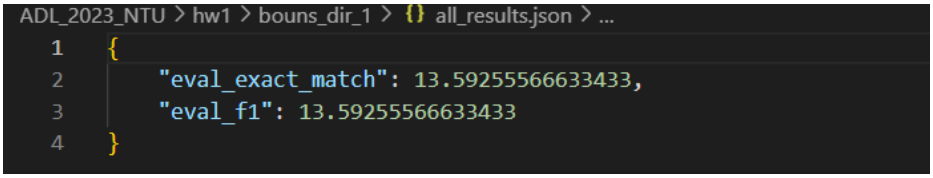
Report - Q5: Bonus (2%)
Instead of the paragraph selection + span selection pipeline approach, train a end-to-end transformer-based model and describe

- **Your model.**
- Objective: input 4 paragraphs and output the answer
  Approach: I take advantage of the span selection task, which the models takes in one paragraphs and the answer text/position.
  I concatenate four paragraphs in advanced and gives it into the model directly. However, different from the original tasks, we input a much more larger paragraphs, this could cause a severe decrease in accuracy due to the masking couldn't cover the whole paragraphs. Thus, I change the model to hfl/chinese-xlnet-base, which can take in a bigger input.

  The below pictures is how I deal with the new paragraph and new answer start position.

```
437         ''' where i don don so jiao '''
438         def convert_context(example):
439             # preprocess data
440             whole_para = ""
441             new_start = 0
442             isInclude = True
443             for x in example["paragraphs"]:
444                 whole_para += context_file[x]
445                 if x != example["relevant"] and isInclude:
446                     new_start += len(context_file[x])
447                 else:
448                     new_start += example["answer"]["start"]
449                     isInclude = False
450 
451             example["context"] = whole_para
452             example["answer"]["answer_start"] = [new_start]
453             example["answer"]["text"] = [example["answer"]["text"]]
454             example["answer"].pop("start")
455             return example
456 
457         raw_datasets = raw_datasets.map(convert_context)
458         raw_datasets = raw_datasets.rename_column("answer", "answers")
459         ''' where i end don don so jiao '''
```

- **The performance of your model.**
  The result is lower compared to the original two-step method. One possible reason could be the limitations of my hardware, thus the low training complexity.



- **The loss function you used.**
  Cross entropy
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  I use Adam, with learning rate 4e-5 and total effective batch size 2.
  (per_device_train_batch_size:1, gradient_accumulation_steps 2)
  Max_seq_length 1024