ADL Hw2

R11944014 戴靖婷

tags: ADL Python Report

Q0: Sample Code

- My codes are adjusted from the sample codes of huggingface.
  - Context Selection (Multiple Choice):
    - *train_mc_no_trainer.py* and *test_mc.py* from *run_swag_no_trainer.py*
    - https://github.com/huggingface/transformers/tree/main/examples/pytorch/multiple-choice (https://github.com/huggingface/transformers/tree/main/examples/pytorch/multiple-choice)
    - Data Preprocessing:
      - Preprocess the data format as same as SWAG in *preprocess_function*.
      - Map the relevant context indices to the relevant context texts.
  - Question Answering:
    - *train_qa.py* and *test_qa.py* from *run_qa.py*
    - https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering (https://github.com/huggingface/transformers/tree/main/examples/pytorch/question-answering)
    - Data Preprocessing:
      - Before loading datset, preprocess the data format as same as SQuAD.
      - Map the relevant context indices to the relevant context texts.
      - Change some key names of the dictionaries.
      - Turn *text* and *answer_start* to list type.

Q1: Data Processing

1. Tokenizer:

- Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.
  - tokenizer_class: BertTokenizer
  - BertTokenizer is a subword tokenization algorithm, which is able to avoid unseen words or relevant words with similarity.
  - Moreover, it is a WordPiece method that splits a word into subwords by maximizing the likelihood of the training data in the vocabulary.

    - WordPiece computes a score for each pair:
        - score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)
        - The algorithm first merges the pairs whose individual parts are less frequent in the vocabulary by dividing the frequency of the pair by the product of the frequencies.

2. Answer Span:

- How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?
    - Since BERT uses subword tokenization, it splits a word into subwords, and thus changes the *start_position* of the original context.
    - Set *return_offsets_mapping = True*, which return the corresponding start/end of the character to each token.
    - Create a map called *offset_mapping* in order to map from token to character position.
    - Iterate each *offset_mapping* for finidng the answer span of the text. Then, check whether the answer is in the span or not.
- After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?
    - The model predicts the probability of answer span start/end position, then remove the out-of-scope answers.
        - Such as the indices out of bounds, length < 0 or > max_answer_length.
    - Next, calculate the probability of the corresponding start/end position by exponential and keep the one with the best probability as the answer span.

Q2: Modeling with BERTs and their variants

Describe BERT model.

- your model: **bert-base-chinese**

- Context Selection:


```
{
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "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",
  "torch_dtype": "float32",
  "transformers_version": "4.20.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

    - Question Answering:


```
{
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "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",
  "torch_dtype": "float32",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


- performance of your model:
    - Overall performance on kaggle:
        - public score: 0.73869
        - private score: 0.75158
    - Context Selection:
        - eval_accuracy: 0.9531405782652044
    - Question Answering:
        - eval_exact_match: 77.99933532735128
        - eval_f1: 77.99933532735128
- The loss function:
    - Context Selection:
        - Loss=MaskedLanguageModelingLoss+NextSequencePredictionLoss
        - Ignore the masked token (index = 100).
    - Question Answering:
        - The number of labels is more than one, which is a classification loss computed by *Cross-Entropy*.
- The optimization algorithm, learning rate and batch size.
    - Context Selection:

        - optimizer=AdamW(lr=3e−5,batch_size=2)
            - batch_size: 2 (per_gpu_train_batch_size 1 * gradient_accumulation_steps 2)
    - Question Answering:
        - optimizer=AdamW(lr=3e−5,batch_size=2)
            - batch_size: 2 (per_gpu_train_batch_size 1 * gradient_accumulation_steps 2)

Try another type of pretrained model and describe

- your model: **chinese-roberta-wwm-ext-large**
    - Context Selection:


```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext-large",
  "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": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

    - Question Answering:

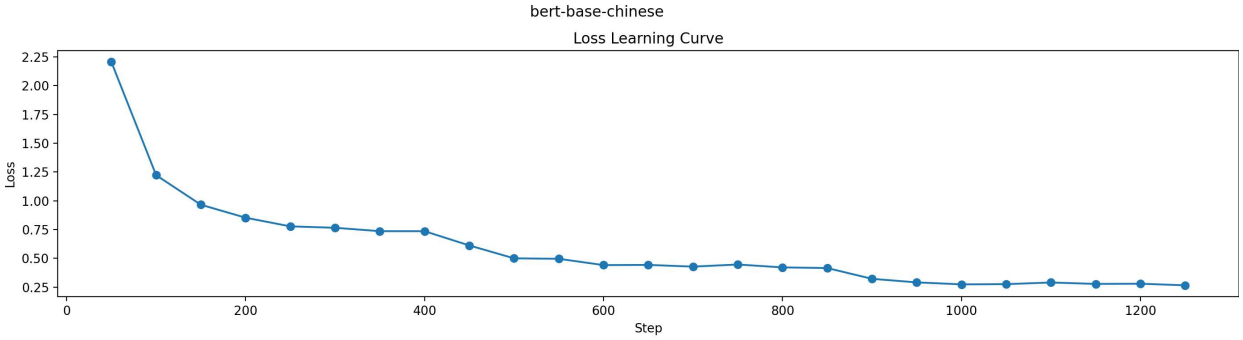

```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext-large",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "directionality": "bidi",
  "eos_token_id": 2,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

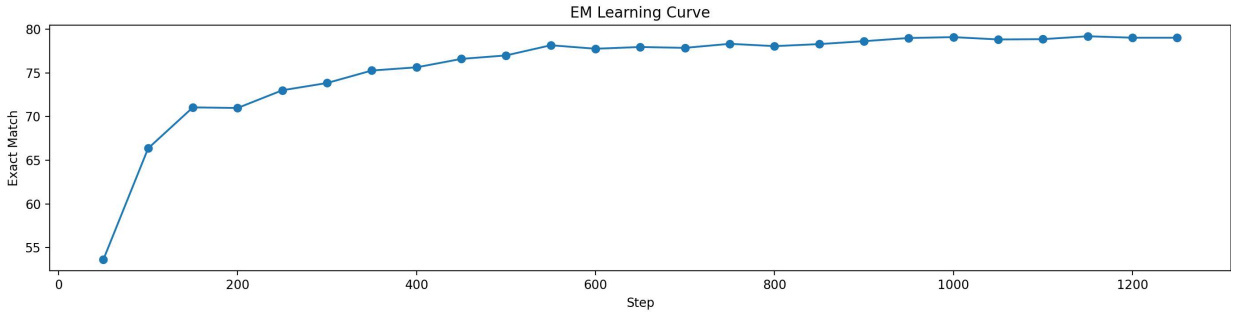

- performance of your model:
    - Overall performance on kaggle:
        - public score: 0.81193
        - private score: 0.80578
    - Context Selection:
        - eval_accuracy: 0.9611166500498505
    - Question Answering:
        - eval_exact_match: 83.88168826852775
        - eval_f1: 83.88168826852775
- The loss function:
    - Context Selection:
        - Loss=MaskedLanguageModelingLoss+NextSequencePredictionLoss
        - Ignore the masked token (index = 100).
    - Question Answering:
        - The number of labels is more than one, which is a classification loss computed by *Cross-Entropy*.
- The optimization algorithm, learning rate and batch size.
    - Context Selection:
        - optimizer=AdamW(lr=3e−5,batch_size=2)
            - batch_size: 2 (per_gpu_train_batch_size 1 * gradient_accumulation_steps 2)
    - Question Answering:

        - optimizer=Adam(lr=3e−5,batch_size=64)
            - batch_size: 64 (per_gpu_train_batch_size 8 * gradient_accumulation_steps 8)
- Differences:
    - BERT uses static masking, which means that the same part of the sentences is maked in each epoch. On the contrary, RoBERTa uses dynamic masking. In different epochs, the different part of the sentences are masked.
    - Another difference is that RoBERTa removes NSP task and keeps only MLM task.
    - Also, I used the large one of RoBERTa which is pre-trained on a larger vocabulary to make it more robust. Besides, larger batch size also improves the performance.

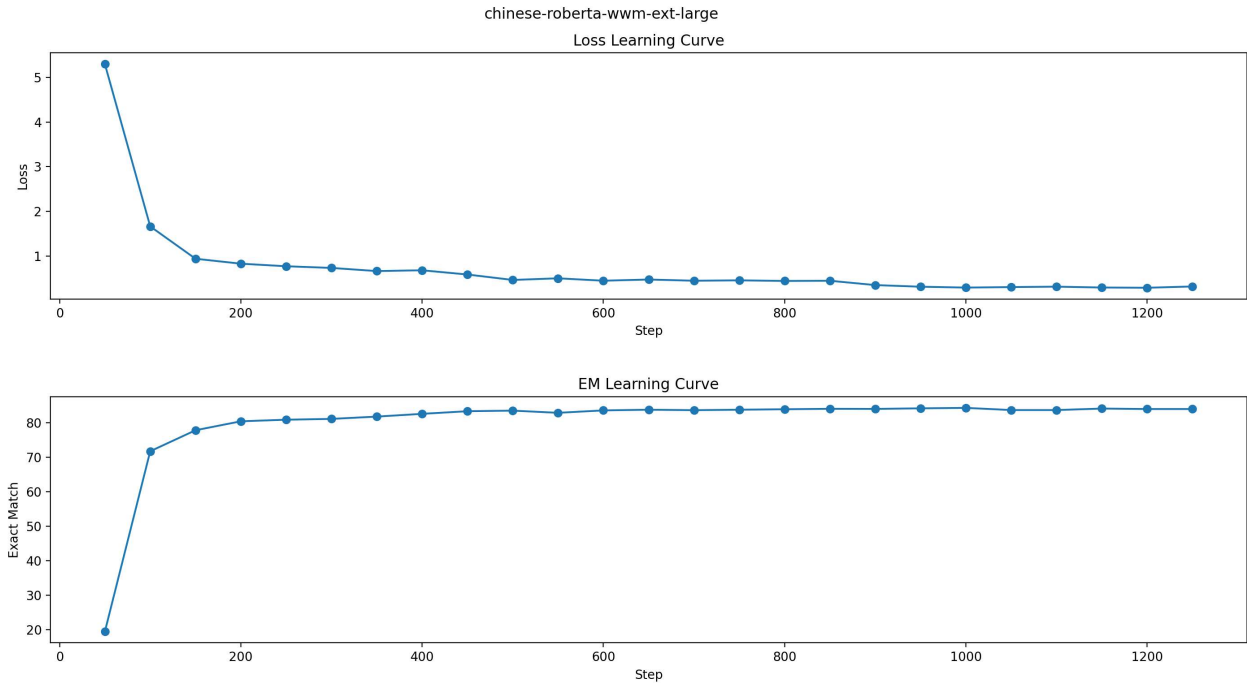
Q3: Curves

- Plot the learning curve (loss and EM) of your QA model
    - bert-base-chinese





- chinese-roberta-wwm-ext-large



- Discussion:
  - I plot each 50 steps for clear observations.
  - We may observe that the two learning curves (loss and EM) of *chinese-roberta-wwm-ext-large* converge faster and better than the ones of *bert-base-chinese*.
  - The result also reflects on the evaluation and test accuracy that *chinese-roberta-wwm-ext-large* performs better.

Q4: Pretrained vs Not Pretrained

- Train a transformer model from scratch (without pretrained weights) on the dataset (you can choose either MC or QA)
- Describe the following information of **QA**.

- the configuration of the model


```
{
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 384,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "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": 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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


- how you train this model
  - The adjusted configuration is from the configuration of *bert-base-chinese*. I simply reduced the hidden_size, num_attention_heads, and num_hidden_layers.
  - Use *–config_name* instead of *–model_name_or_path*.
- the performance of this model v.s. BERT
  - BERT:
    - eval_exact_match: 77.99933532735128
    - eval_f1: 77.99933532735128
  - Model from scratch:
    - exact_match: 4.685942173479561
    - f1: 4.685942173479561