ADL-2022 Homework 2 Report

> Student Name: 陳耕宇
>
> Student ID: B07902063

Q1: Data processing

1. Tokenizer:

   I use the default pretrained tokenizer for both context-selection and question-answering tasks, which are `bert-base-chinese` and `hfl/chinese-roberta-wwm-ext-large` on huggingface respectively.

   - bert-base-chinese
   - hfl/chinese-roberta-wwm-ext-large

   Both use a character-based tokenization for Chinese words and the WordPiece algorithm for all others. The former is to simply split each character as a token, since each Chinese word represents its meaning on its own. This is not the case for English or other language alphabets. For non-Chinese words, the WordPirce algorithm first split each word into character level as symbols, and start to merge the pair of symbols which increases the probability on the data the most after merging. The merging process continues, and stops when a max size of vocabulary is reached or the increased probability value is lower than some certain threshold.

   For `bert-base-chinese` in context-selection task, each sample is a tuple of 4 choices, and each of which is the question for this sample concatenated with one of 4 choice contexts. Each sample is then tokenized by the pretrained tokenizer with truncation max length set to 512. The padding will be done in each batch.

   For `hfl/chinese-roberta-wwm-ext-large`, in downstream question-answering task, each sample is a question combined with a context. The truncation max length is set to 384, but overflowing tokens are used. This means if a context is too long, we will split it into several samples instead of just truncate. A stride number 128 is used to indicate the overlapping length for each split sample, which can prevent the case that the answer being split into two samples. Note that only the context is split, and the question in each sample is entirely intact.

2. a. The `PreTrainedTokenizer` class in transformers package includes an argument called `return_offsets_mapping`, which can be used to return the offset mapping for each token. It is a list of tuples which reveals the start-position and end-position of a token in the original context. With this, we can first index the start-position in original context, and search through all tokens in the sample to find which token has the corresponding offset.

   b. The offset mapping introduced above helps us finding the start-position and end-position in orignal context given an token index. We can just use it to return the position for the token which gives the best score.

Q2: Modeling with BERTs and their variants

1. The following is all about the first version of my works. For context-selection task, I use the following model (the `config.json` used and output by transformers)

   ```json
   {
     "_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": "4.19.0.dev0",
     "type_vocab_size": 2,
     "use_cache": true,
     "vocab_size": 21128
   }
   ```

   The valuation accuracy is 0.9588, the loss function is cross entropy loss, and the optimizer is AdamW with a linear scheduler.

   For question-answering task, the model is used

   ```json
   {
     "_name_or_path": "bert-base-chinese",
     "architectures": [
       "BertForQuestionAnswering"
     ],
     "attention_probs_dropout_prob": 0.1,
     "classifier_dropout": null,
   ```

```json
  "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": "4.19.0.dev0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

The EM score on the validation set is 0.7996. The loss function is the cross entropy loss average for start-position and end-position. The optimizer is AdamW with a linear scheduler.

The combination of both models on public test data on Kaggle has a score of 0.76311.

2. Since the validation score for context-selection task is already very high, I chose to change only the model for question-answering task. The pretrained model I use is showed in the following.

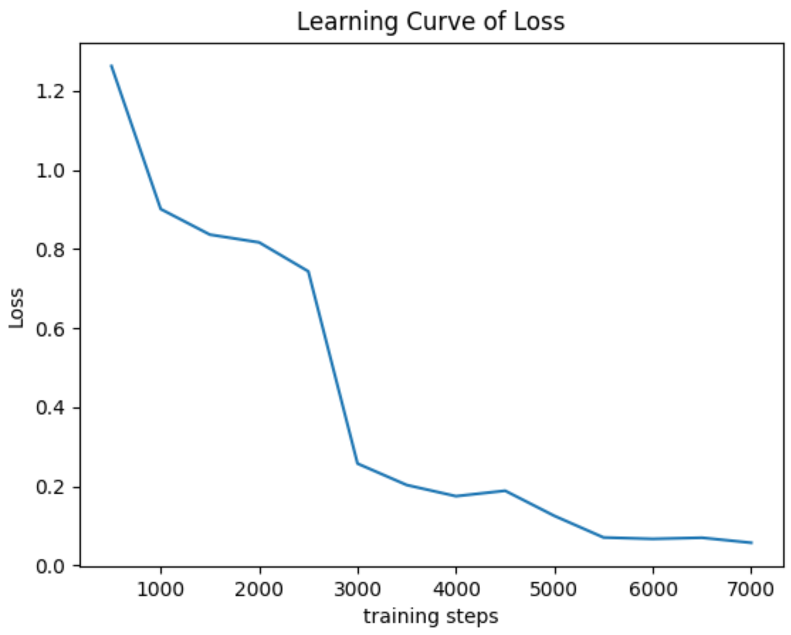
```json
{
 "_name_or_path": "hfl/chinese-roberta-wwm-ext-large",
 "architectures": [
 "BertForQuestionAnswering"
 ],
 "attention_probs_dropout_prob": 0.1,
 "bos_token_id": 0,
 "classifier_dropout": null,
 "directionality": "bidi",
 "eos_token_id": 2,
 "hidden_act": "gelu",
 "hidden_dropout_prob": 0.1,
 "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": "4.19.0.dev0",
 "type_vocab_size": 2,
 "use_cache": true,
 "vocab_size": 21128
}
```

The EM score on validation set is 0,8398, and the combination if both models on public test data on Kaggle has a score of 0.80831, which are higher than the original model.

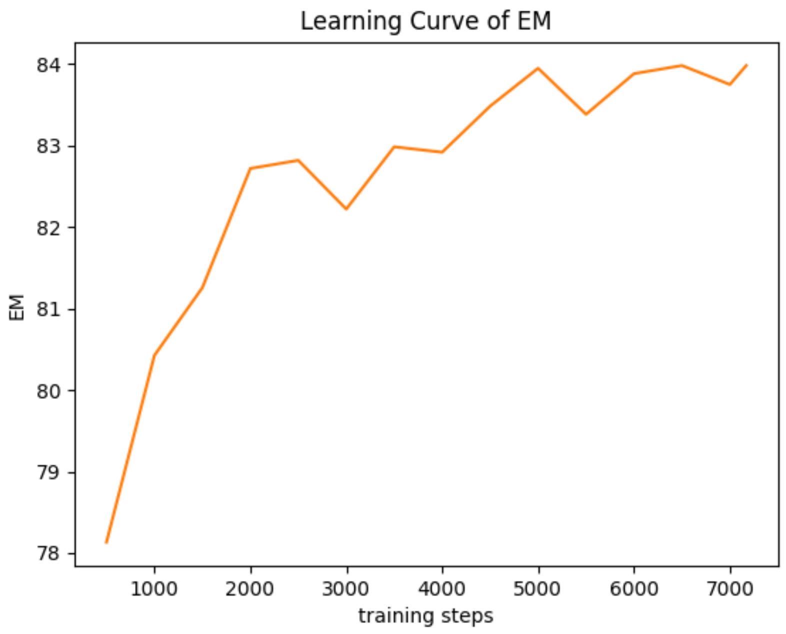
On of the main differences between the model `bert-base-chinese` and `hfl/chinese-roberta-wwm-ext-large` is the whole-word-masking (WWM) scheme. The WWM scheme refers to masking entire words rather than randomly choosing tokens to mask in pertaining phase. This helps the model capture the meaning in the context more accurately. Some other differences includes that the latter use a RoBERTa model, extended training data set and a large parameter setting. The base parameter model (`bert-base-chinese`) includes 12-hidden-layer, 768-hidden-size, 12-attention-heads, and totally 110M parameters setting, while the large parameter model (`hfl/chinese-roberta-wwm-ext-large`) enjoys a 24-hidden-layer, 1024-hidden-size, 16-attention-heads, and totally 330M parameters setting.

Q3: Curves

The learning curve of loss for my QA model (pretrained by `hfl/chinese-roberta-wwm-ext-large`)



The learning curve of EM score on validation set for my QA model (pretrained by `hfl/chinese-roberta-wwm-ext-large`)



Q4: Pretrained vs Not Pretrained

I compare the model fine-tuned from `hfl/chinese-roberta-wwm-ext-large` discussed above with a non-pretrained model for the question-answering task. The non-pretrained one is made by the same training python scripts except loading only configuration of some pretrained model name rather than the model weights. This is done by the following snippet:

```
# Load model weights
model = AutoModelForQuestionAnswering.from_pre_trained(model_name_or_path)
# Load model configuration without loading weights
model = AutoModelForQuestionAnswering.from_comfig(model_config)
```

The both experiments share the same model architecture (`config.json`) as follows:

```
{
 "_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": "4.19.0.dev0",
 "type_vocab_size": 2,
 "use_cache": true,
 "vocab_size": 21128
}
```

The result based on pretrained model goes (`all_results.json` generated by transformers):

```
{
    "epoch": 3.0,
    "eval_exact_match": 2.4925224327018944,
    "eval_f1": 2.4925224327018944,
    "eval_samples": 5510,
    "train_loss": 5.590312631093641,
    "train_runtime": 10912.1242,
    "train_samples": 38237,
    "train_samples_per_second": 10.512,
    "train_steps_per_second": 0.657,
}
```

And the result based on non-pretrained model is shown:

```
{
    "epoch": 3.0,
    "eval_exact_match": 83.98138916583582,
    "eval_f1": 83.98138916583582,
    "eval_samples": 5510,
    "train_loss": 0.4040121578604939,
    "train_runtime": 10915.1279,
    "train_samples": 38237,
    "train_samples_per_second": 10.509,
    "train_steps_per_second": 0.657
}
```

Obviously the model based on BERT behaves much better than those without pretrained weights in performance, though they have almost same training time.