Q1: Data processing

1. Tokenizer

   BERT tokenizer is called wordpiece, which is similar to byte pair encoding. Both of them divide each word to subwords. After the division of all words, a language model is made and we can select these subwords according to some predefined rule. Besides, this is the main difference between wordpiece and BPE is that the former chooses these subwords depending on their probability computed by the model, while the former selects them based on frequency. To get the result, we will do the selection several times until we reach the threshold.

2. Answer span

   a. **How did you convert the answer span on characters to position on tokens after BERT tokenization?**

   Tokenizer will return the start and end position on characters. Then, we have to find the start and end position on tokens with an iterative approach. During the iteration, once we found that the position on character of the first/last characters of the token which indicated by *token_start*/*token_end* matches the positions we got from tokenizer, we succeed.

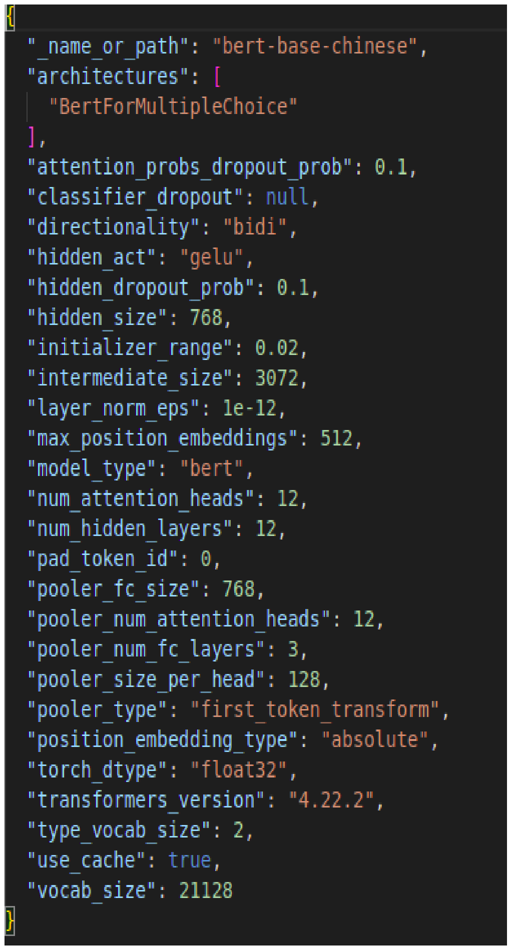
   b. **What rules did you apply to determine the final start/end position?**

   After the prediction, we got *start_logits* and *end_logits* as 2 lists. The 2 lists represent the probability distribution of start/end tokens for each token. To find the most possible pair of start and end position, we can traverse all possible and legal ($start <= end$) combination and record the product of their probability as the score. At last, since we only have to find one answer, we can simply pick the pair with the highest score as our final start/end position.

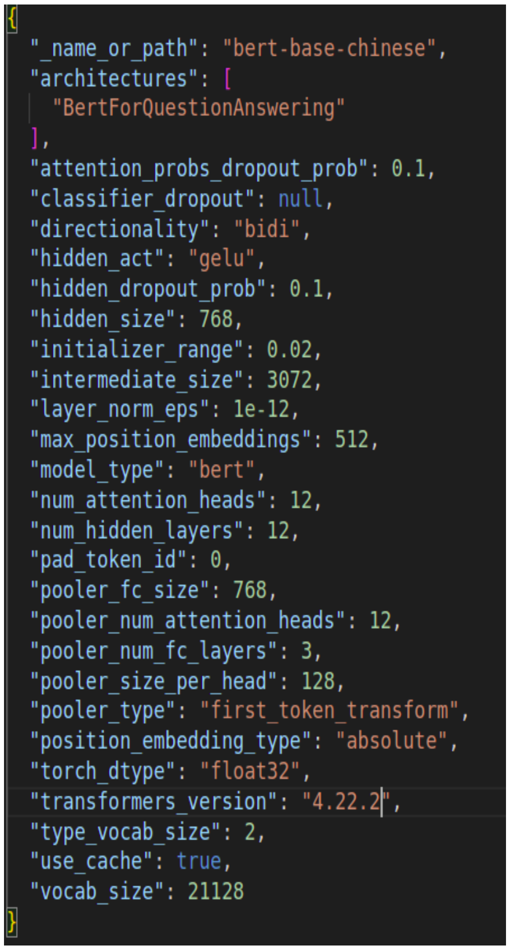
Q2: Modeling with BERTs and their variants

1. BERT

   a. model:



```
{
  "_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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```



```
{
  "_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
}
```

   b. performance: 0.74874/80.492 (for kaggle and EM)

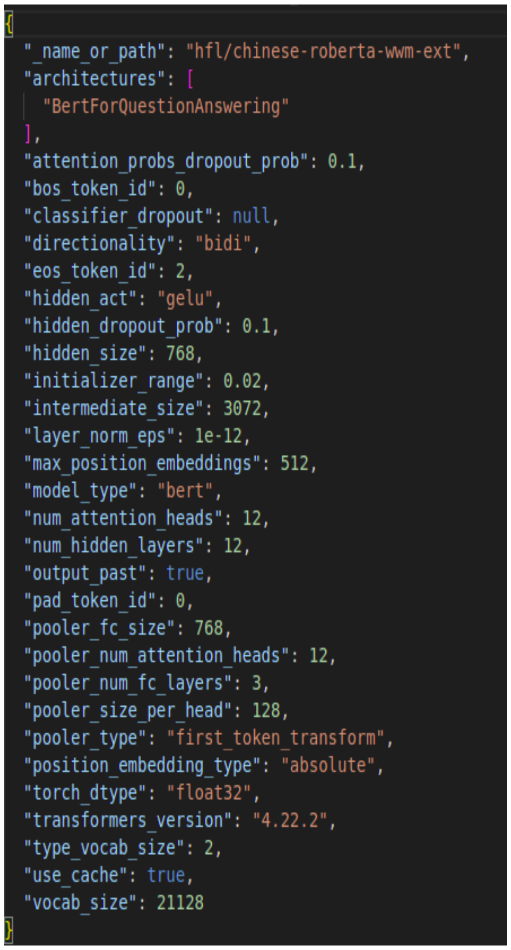
   c. loss function: cross entropy + label smoothing
      label smoothing: make the probability not one-hot

   d. Relative Parameter:

      i. optimization algorithm: Adamw
      ii. learning rate: 0.00003
      iii. batch size:
         per_device_train_batch_size = 2
         gradient_accumulation_steps = 8

2. roberta

   a. model:



```
{
  "_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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

   b. performance: 0.79927/82.021(for kaggle and EM)

   c. difference:

      i. Masking
      RoBERTa uses dynamic masking, which means or different Epochs different part of the sentences are masked, while BERT uses static masking i.e. the same part of the sentence is masked in each Epoch.

      ii. Remove NSP Task
      Results showed that NSP task isn't very useful for pre-training the BERT model. Hence, RoBERTa only uesd MLM tasks.

      iii. More data Points

      iiii. Large Batch size

Q3: Curves

These 2 curves below shows the data of my roberta model

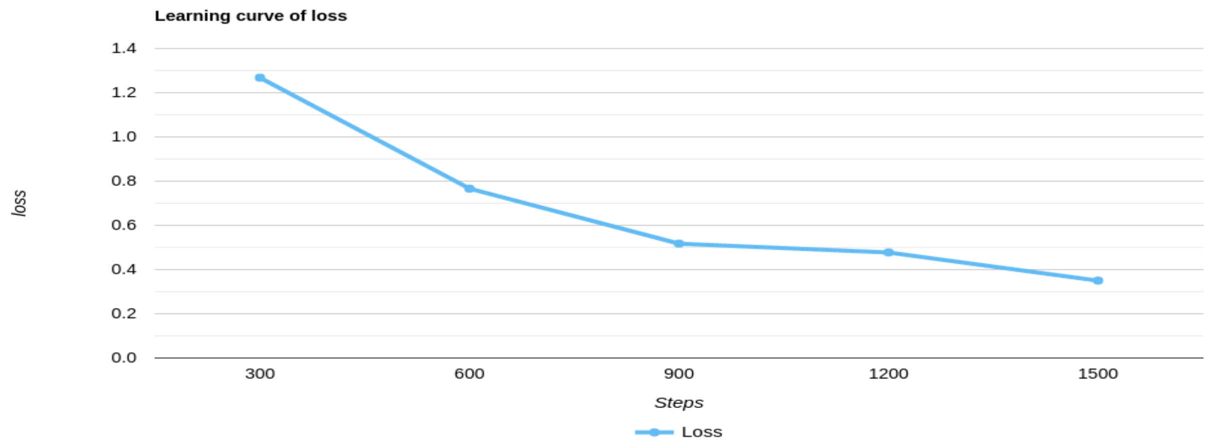


Figure 1: Learning curve of loss

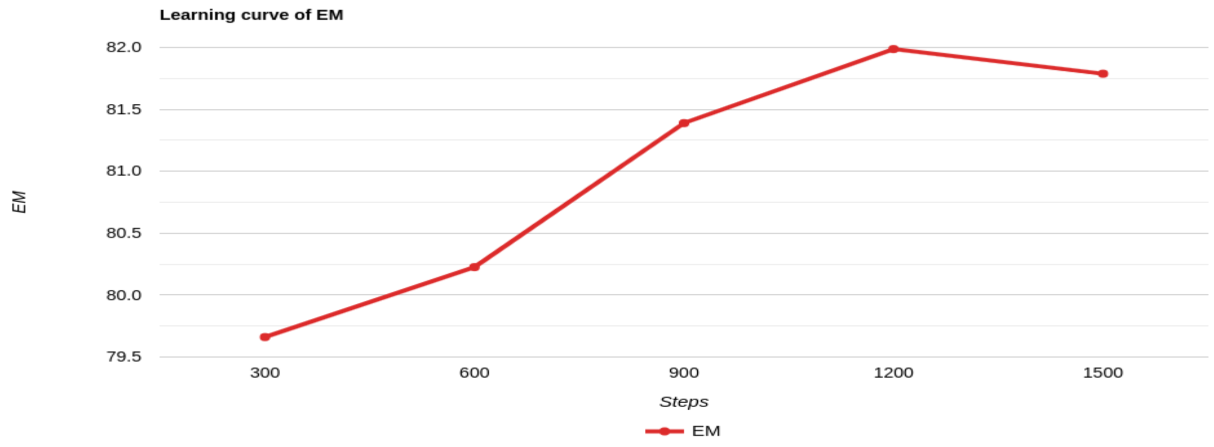
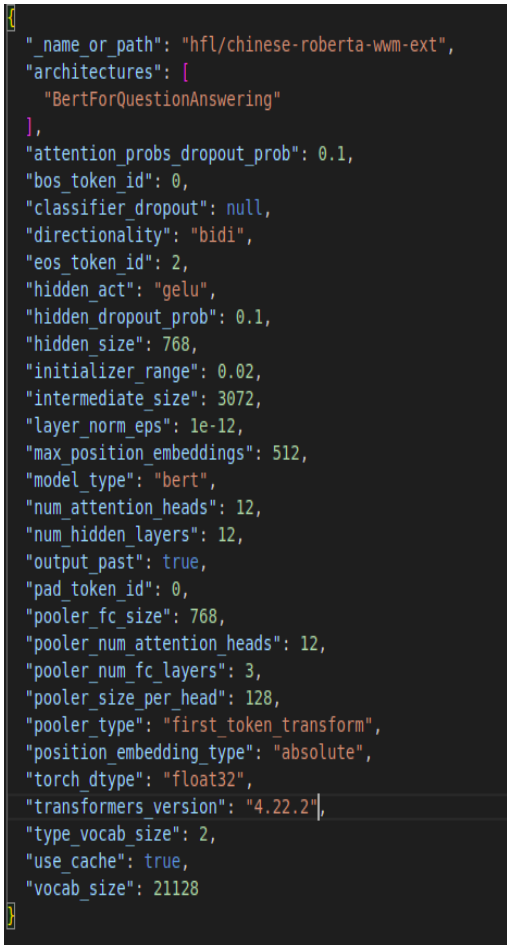


Figure 2: Learning curve of EM

Q4: Pretrained vs Not Pretrained

I choose question-answering to do this experiment.

a. configuration: I used the same configuration as Q2, but use *from_config* instead of *from_pretrained*



```
{
  "_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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

b. performance: Since the result of *eval_exact_match* has signigicant difference, I didn't submit it to kaggle to save my quota. EM of non-pretrained and bert model are 4.819 and 80.492 respectively.