Q1: Data processing

1. Tokenizer

   a. I used RoBERTa as my tokenizer. The tokenization algorithm of RoBERTa is **BPE** (Byte Pair Encoding). What BPE has done is to count the frequency of each word shown in the corpus and split words into several tokens. Then, BPE counts the freqency of consecutive byte pairs and merge the most frequent pair (merge 2 tokens into 1 token) iteratively. And we can determine the number of iteration or the number of maximum tokens to control the number of tokens.

2. Answer Span

   a. BertTokenizer in HuggingFace.co provides a mapping **offset_mapping** with the start position and end position of characters for each token. Therefore, I could simply use the mapping to find tokens.

   For example, these pairs mean the start and end position of characters for each token

   ```
   [(0, 0), (0, 3), (4, 8), (9, 13), (14, 18), (19, 22), (23, 28), (29, 33), (34, 37), (37, 38)
   ```

   b. It's because the start position and end position with the highest probability may be impossible (end position > start position), I used the summnation of the probability for each pair of start and end position and sorted them. Therefore, we could find a highest resonable pair of start and end position.

Q2: Modeling with BERTs and their variants

1. Describe

- a. Model:

  1. Multiple choice

     input_ids$_i$, attention_mask$_i$, token_type_id$_i$ = Tokenizer(input_seq)

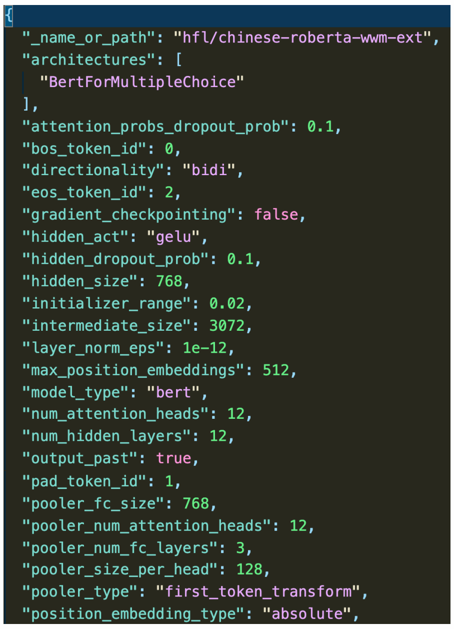
     , where input_seq is [CLS] Q_i [SEP] $P_{i,j}$ [SEP] , Q_i is the i-th question and $P_{i,j}$ is the j-th option for Q_i

     out_1$_i$ = RoBERTa(input_ids$_i$, attention_mask$_i$, token_type_id$_i$)

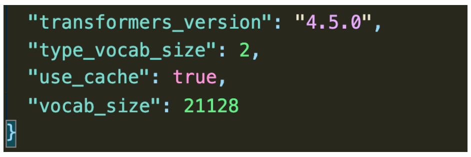
     out_2$_i$ = Dropout(out_1$_i$)

     logits$_i$ = Linear(out_2$_i$)

     choice$_i$ = argmax(logits$_i$)



```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "directionality": "bidi",
  "eos_token_id": 2,
  "gradient_checkpointing": false,
  "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": 1,
  "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",
```



```
  "transformers_version": "4.5.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

2. Question Answering

   input_ids$_i$, attention_mask$_i$, token_type_id$_i$ = Tokenizer(input_seq)

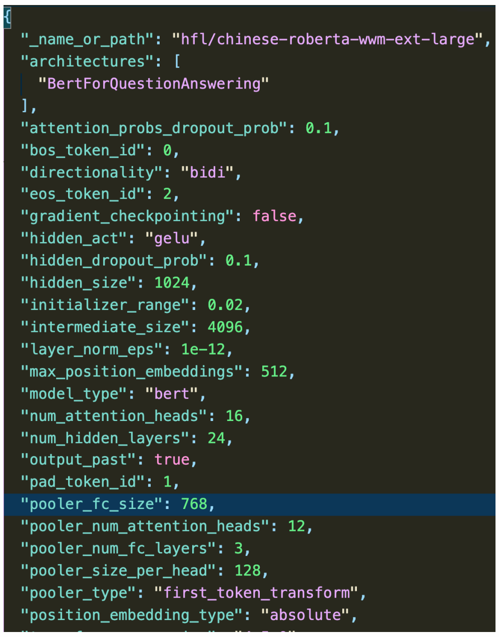
   , where input_seq is [CLS] Q_i [SEP] P_i [SEP] , Q_i is the i-th question and P_i is the i-th paragraph

   out_1$_i$ = RoBERTa-Large(input_ids$_i$, attention_mask$_i$, start_position$_i$, end_position$_i$)

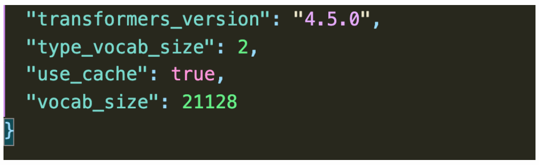
   start_logits$_i$, end_logtis$_i$ = Linear(out_1$_i$)

   logits_sum$_i$ = start_logits$_{i,j}$ + end_logits$_{i,k}$ where j, k $\in [1, 512]$ and remove nonsnese combination

   start_position$_i$, end_position$_i$ = max(logits_sum$_i$)



```
{
  "_name_or_path": "hfl/chinese-roberta-wwm-ext-large",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "directionality": "bidi",
  "eos_token_id": 2,
  "gradient_checkpointing": false,
  "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": 1,
  "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",
```

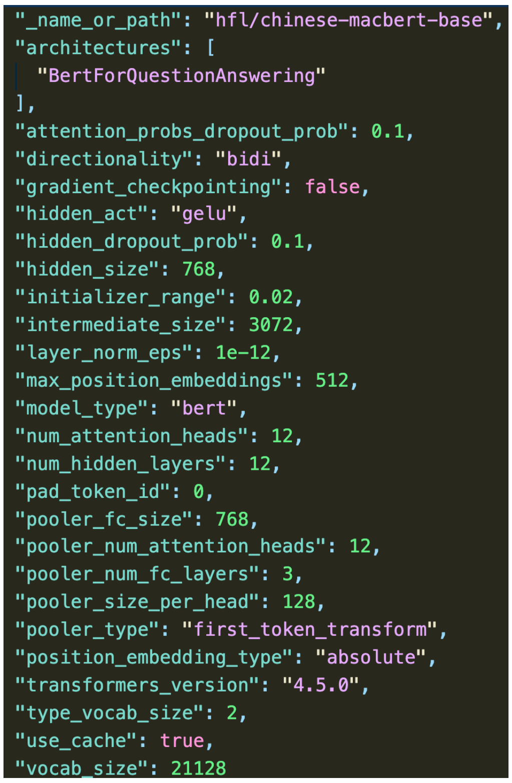


```
  "transformers_version": "4.5.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

- b. Performance: 0.952 (context selection)/ 0.826 (EM), 0.881(F1)
- c. Loss Function: based on source code from HuggingFace.co, the loss function is **CrossEntropyLoss**
- d.
  - optimization algorithm: AdamW
  - learning rate: 5×10^{-5} (with learning rate scheduler)
  - batch size: 16
  - Train Epoch: 3

2. Try another type of pretrained model and describe

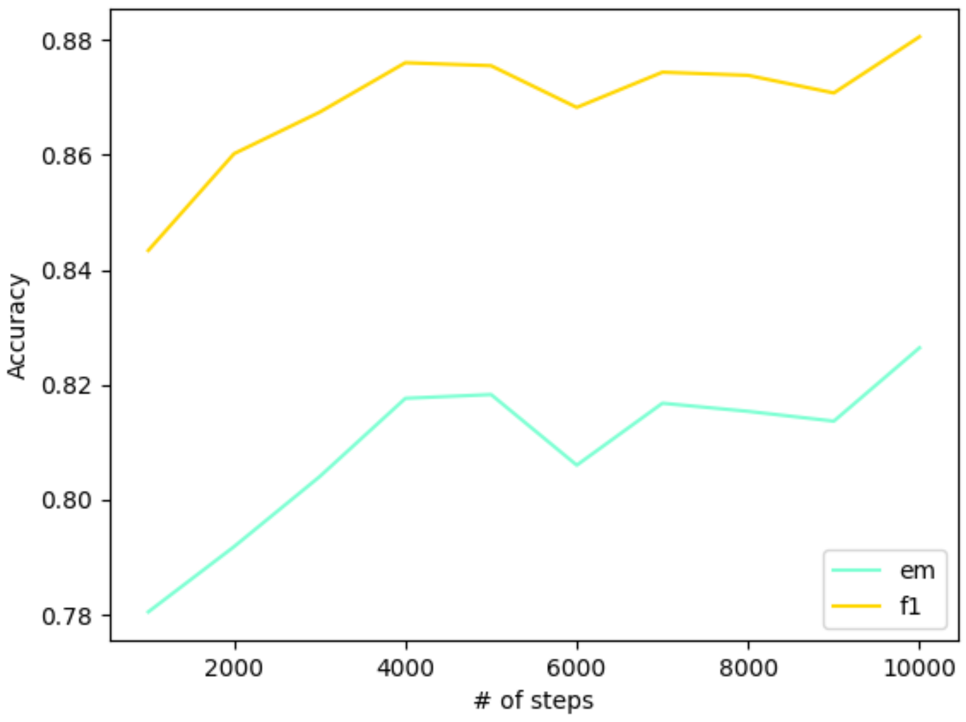
- a. Model:
  1. Context Selection Model is the same as the privous one
  2. Question Answering:



```
"_name_or_path": "hfl/chinese-macbert-base",
"architectures": [
  "BertForQuestionAnswering"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"gradient_checkpointing": false,
"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",
"transformers_version": "4.5.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```

- b. Performance: 0.806(EM); 0.863(F1)
- c. Loss Function: based on source code from HuggingFace.co, the loss function is **CrossEntropyLoss**
  - The difference between MacBERT and original BERT is that **similar words** are used for the masking instead of [MASK] tokens. Also, whole word masking(WWM), N-gram masking, sentence order prediction(SOP) are used as well. The main architecture is basically the same as BERT.
- d.
  - optimization algorithm: AdamW
  - learning rate: 5×10^{-5} (with learning rate scheduler)
  - batch size: 16
  - Train Epoch: 3

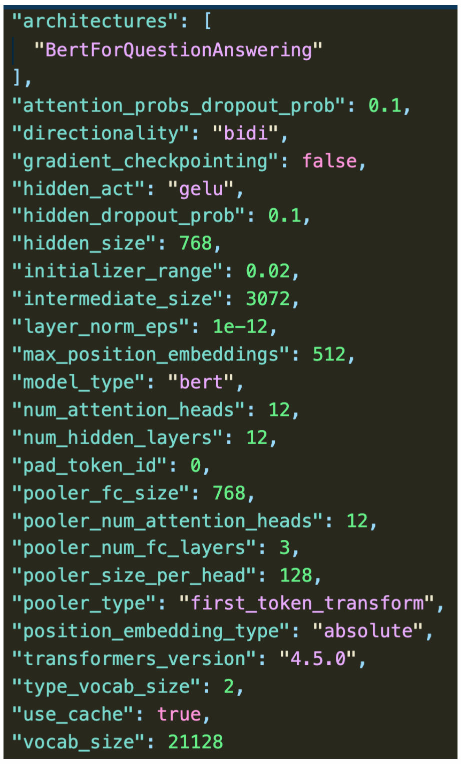
Q3: Curves



Q4: Pretrained vs Not Pretrained

The Context Selection parts are the same. This comparsion only foucses on QA task.

- Not Pretrained Model:
  - Configuration:



```
"architectures": [
  "BertForQuestionAnswering"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"gradient_checkpointing": false,
"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",
"transformers_version": "4.5.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```

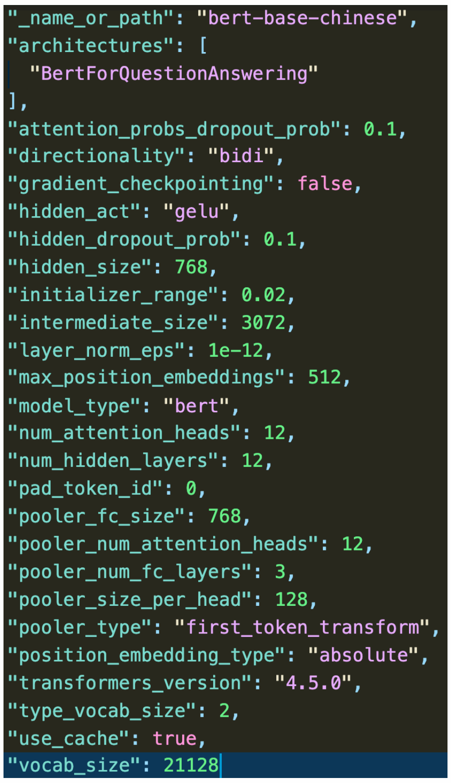
- For not pretrained model, I used the configuration of **bert-base-chinese** without pretraining. The 2 lines below are the only thing that I changed from the pretrained model.

  ```
  config = AutoConfig.from_pretrained('bert-base-chinese')

  model = AutoModelForQuestionAnswering.from_config(config)
  ```

  - Learning rate: 5×10^{-5}
  - Train Epoch: 5
  - Performance: 0.077(EM) ; 0.132(F1)

- Pretrained Model:
  - Configuration:



```
"_name_or_path": "bert-base-chinese",
"architectures": [
  "BertForQuestionAnswering"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"gradient_checkpointing": false,
"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",
"transformers_version": "4.5.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```

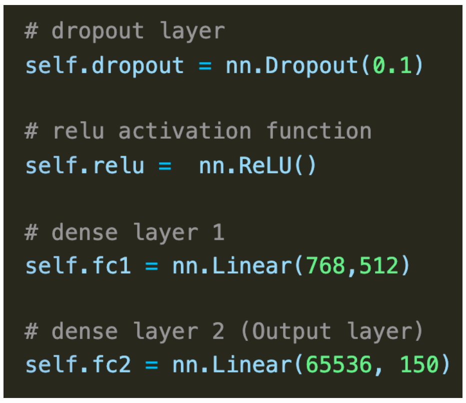
- Learning rate: 5×10^{-5}
- Train Epoch: 5
- Performance: 0.756(EM); 0.813(F1)

Q5: Bonus: HW1 with BERTs

Intent Prediction

- I used **bert-base-uncased** model as my pretrained model and the configuration is basically the same as the previous question. (768 hidden_size, 12 attention_heads, etc). However, there are additional two downstream linear layers to classify intents and the procedure is as below.

  input $\rightarrow$ Tokenizer $\rightarrow$ BERT $\rightarrow$ Linear $\rightarrow$ ReLU, Dropout $\rightarrow$ Linear $\rightarrow$ Softmax $\rightarrow$ output



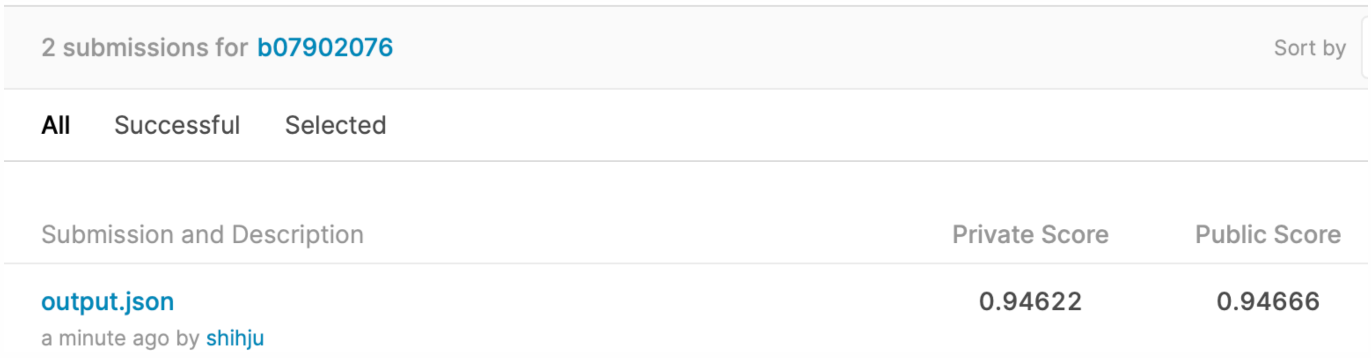
```
# dropout layer
self.dropout = nn.Dropout(0.1)

# relu activation function
self.relu =  nn.ReLU()

# dense layer 1
self.fc1 = nn.Linear(768,512)

# dense layer 2 (Output layer)
self.fc2 = nn.Linear(65536, 150)
```

- Performance: 0.94622 (Private Score); 0.94666 (Public Score)



2 submissions for b07902076

Sort by

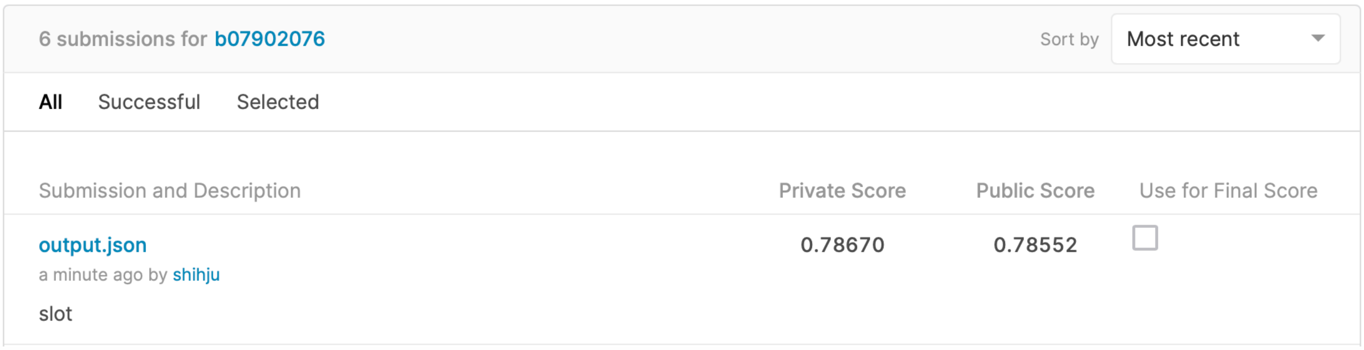
All Successful Selected

Submission and Description	Private Score	Public Score
output.json a minute ago by shihju	0.94622	0.94666

- I used **CrossEntropyLoss** as loss function.
- Optimization Algorithm: **AdamW**
- Learning Rate: 1×10^{-5}
- Batch Size: 16
- Train Epoch: 1

Slot Tagging

- I used **bert-base-uncased** model as my pretrained model and configuration is basically the same as previous question. However, in this task, I used **BertForTokenClassification** pretrained model instead of Bert Model along with down stream linear layers. The performance is better than my RNN models.
- Performance: 0.78670 (Private Score); 0.78552 (Public Score)



6 submissions for b07902076

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
output.json a minute ago by shihju slot	0.78670	0.78552	☐

- I used **CrossEntropyLoss** as loss function.
- Optimization Algorithm: **AdamW**
- Learning Rate: 5×10^{-5}
- Batch Size: 16
- Train Epoch: 5