Homework2

R08525116 吳承哲

Q1: Data processing

1. Tokenizer:

First, download the vocabulary from hugging face, and it can encode the chinese words to word vectors. The multiple choice task uses “bert-base-chinese”, and the question answering task uses "hfl/chinese-macbert-large". In the second task, I tokenize the data with truncation and padding, but keep the overflows using a stride. Therefore, one data may generate several input features if it has a long context. The inputs of BERT are used [CLS] and [SEP] to mark. The beginning of sentence is [CLS], and the end is [SEP].

I set the max length is 384, stride is 128 and padding side is right.

2. Answer Span:

a. Because the data could generate several features if it has a long context, I need a map from a feature to its corresponding the data. To compute the start positions and end positions, the offsets mappings will generate a map from token to character position in the original context. Then, I need generate the label data. I use for loop to traverse all data. In the beginning, label impossible answers with the index of the [CLS] token, and grab the sequence corresponding to the data to know what are context and question. One data can give several spans, it’s the index of the data containing the span of text. We also set [CLS] as answer if no answers are given. If the answers exit, I save the start position and end position, and then detect whether the answer is out of the span. If yes, I move the start index and end index to the two ends of the answer.

b. Build a map to data’s corresponding features, and then loop all data. There are one or more indices of the features associated to the current data. Loop all features, and get the predictions of model for each feature. This is what will allow me to map some positions in our logits to span of texts in the original context. Then traverse all possibilities for those start and end logits. Don’t consider out-of-span answers, either because the indices out of bounds or correspond to part of the input_ids that are not in context and answers with a length that is either < 0 or > max answer length. In the pretty rare edge case I have not a single non-null prediction, I create a fake prediction to avoid failure. Finally, pick the final best answer.

Q2: Modeling with BERTs and their variants:

1. Describe

Multiple Choice

a. Model: BERT (“bert-base-chinese”)


```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "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",
  "type_vocab_size": 2,
  "vocab_size": 21128
}
```


b. Data: public data

Accuracy: 0.942

c. Loss: default loss

d. Optimizer: AdamW

Learning: 0.00005

Weight decay: 0.01

Batch size: 2

Gradient accumulation steps: 32

Question Answering

a. Model: BERT (“bert-base-chinese”)

b. Data: public data

EM = 0.789

F1 = 0.867

c. Loss: default Loss

d. Optimizer: AdamW
Learning rate: 0.00002
Weight decay: 0.01
Batch size: 2
Gradient accumulation steps: 32

2. Try another type of pretrained model and describe:

Question Answering

a. Model: MacBERT("hfl/chinese-macbert-large")

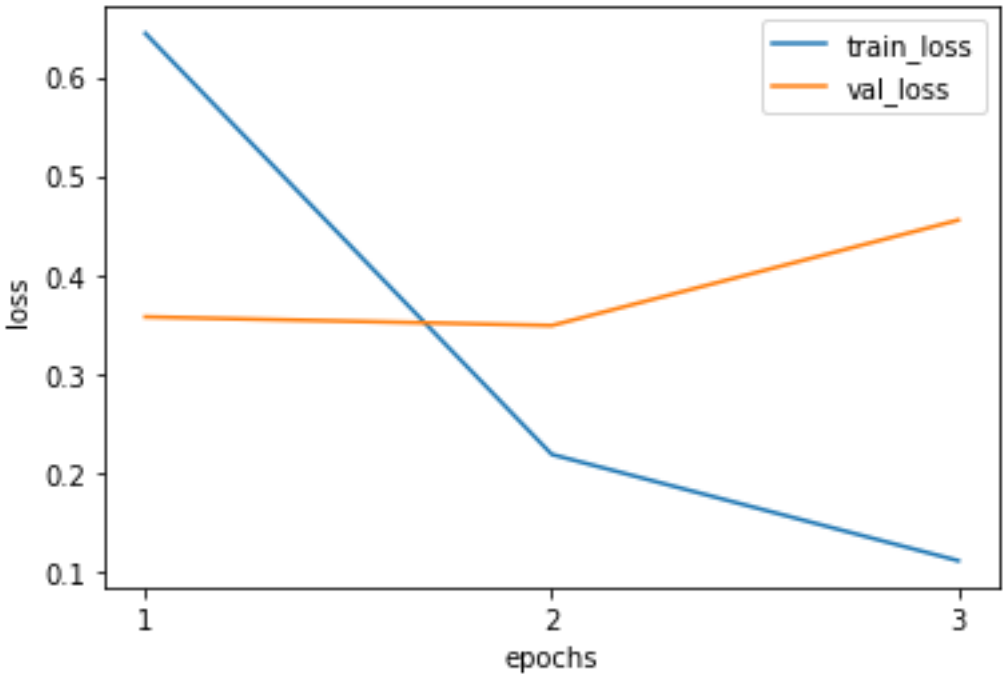

```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "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",
  "type_vocab_size": 2,
  "vocab_size": 21128
}
```


b. Data: public data
EM = 0.855
F1 = 0.916

c. Loss: default Loss

d. Optimizer: AdamW
Learning rate: 0.00002
Weight decay: 0.01
Batch size: 2
Gradient accumulation steps: 32

Q3: Curve



I couldn't use eval.py to get the EM and F1 from transformers.trainer, so I just plotted the loss curve instead.

Q4: Pretrained vs Not Pretrained

Configuration:

Attention heads: 4
Hidden layer: 4
Hidden size: 200
Batch size: 16
Learning rate: 0.0001
Epochs: 3
Weight decay: 0.01
Gradient accumulation steps: 32

I used the same script as Q2, but I just modified the pretrained model to not pretrained model.

Evaluate on validation data.

BERT	Pretrained	Not pretrained
EM	0.789	0.035
F1	0.867	0.059

I tried many configurations for not pretrained model, but the model seemed to not learn anything. Therefore, the performance is really bad.