

NTU-ADL-HW2-Report (R11922A16 資工碩 — 柳宇澤)

tags: NTU ADL

- NTU-ADL-HW2-Report (R11922A16 資工碩一 柳宇澤).
- Q1: Data processing (2%).
 - 1. Tokenizer (1%).
 - a. Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.
 - 2. Answer Span (1%):
 - a. How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?

- b. After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?
- Q2: Modeling with BERTs and their variants (4%).
 - 1. Describe (2%).
 - a. your model (configuration of the transformer model).
 - b. performance of your model.
 - c. the loss function you used.
 - d. The optimization algorithm (e.g. Adam), learning rate and batch size.
 - 2. Try another type of pretrained model and describe (2%).
 - a. your model
 - b. performance of your model

- c. the difference between pretrained model (architecture, pretraining loss, etc.).
- Q3: Curves (1%).
 - 1. Plot the learning curve of your QA model
 - a. Learning curve of loss (0.5%).
 - b. Learning curve of EM (0.5%).
- Q4: Pretrained vs Not Pretrained (2%).
 - 1. Describe
 - a. The configuration of the model and how do you train this model
 - b. the performance of this model v.s. BERT

Q1: Data processing (2%)

1. Tokenizer (1%)

a. Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.


I used Wordpiece tokenization as my tokenization algorithm.

1. Split

Like BPE, WordPiece starts from a small vocabulary including the special tokens used by the model and the initial alphabet. Since it identifies subwords by adding a prefix (like ## for BERT), each word is initially split by adding that prefix to all the characters inside the word. Thus, the initial alphabet contains all the characters present at the beginning of a word and the characters present inside a word preceded by the WordPiece prefix.

2. Merge

Then, again like BPE, WordPiece learns merge rules. The main difference is the way the pair to be merged is selected. Instead of selecting the most frequent pair, WordPiece computes a score for each pair, using the following formula:

$$\text{score} = (\text{freq_of_pair}) / (\text{freq_of_first_element} \times \text{freq_of_}$$


By dividing the frequency of the pair by the product of the frequencies of each of its parts, the algorithm prioritizes the merging of pairs where the individual parts are less frequent in the vocabulary.

Then continuing Merge until we reach the desired vocabulary size.

3. Tokenization

Starting from the word to tokenize, WordPiece finds the longest subword that is in the vocabulary, then splits on it. When the tokenization gets to a stage where it's not

possible to find a subword in the vocabulary, the whole word is tokenized as unknown, [UNK] .

2. Answer Span (1%):

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

Let processed (Question, Context) be examples . We will loop through all the features associated to each example. Then, we access `features["offset_mapping"]` , which allows us to map some the positions in our logits to span of texts in the original text.

b. After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?

Sum the probabilities of `start` and `end` position as `score` , then choose the `start-end` pair with highest `score` .

Q2: Modeling with BERTs and their variants (4%)

1. Describe (2%)

a. your model (configuration of the transformer model)

- Multiple choice

```
{
  "_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  
}
```

- QA

```
{
  "_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 of your model.

	MC	QA
Hit/EM	0.9485	0.7896

c. the loss function you used.

BCE loss

d. The optimization algorithm (e.g. Adam), learning rate and batch size.

- Optimization alg: AdamW

- learning rate: 0.00003
- batch size: 8

2. Try another type of pretrained model and describe (2%)

a. your model

- Multiple choice

```

{
  "_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
}

```

- QA

```
{
  "_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
}
```

b. performance of your model

Column 1	MC	QA
Hit / EM	0.9501	0.8431

c. the difference between pretrained model (architecture, pretraining loss, etc.)

For example, BERT -> xlnet, or BERT -> BERT-wwm-ext.
You can find these models in huggingface's Model Hub.

I used bert-base-chinese -> roberta-wwm-ext-large .

First of all, roberta does dynamic masking rather than bert's fixed masking. What's more, roberta also does byte-level encoding while bert does word-level encoding.

Roberta also remove NSP task in bert.

wwm means doing Whole Word Masking rather than WordPiece tokens.

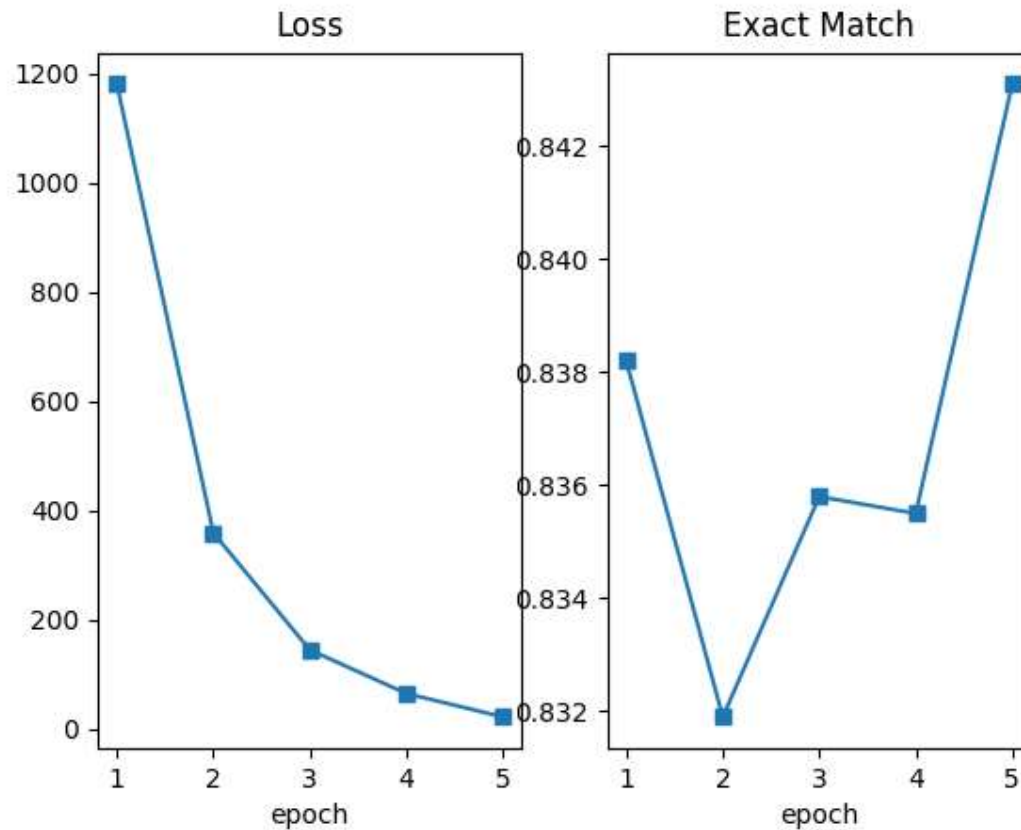
large means the deeper learnable layers or larger dimensions, including intermediate_size (3072→4096) , num_hidden_layers (12→24) , num_attention_heads

(12→16) and hidden_size (768→1024) , etc.

Q3: Curves (1%)

1. Plot the learning curve of your QA model

- a. Learning curve of loss (0.5%)
- b. Learning curve of EM (0.5%)



Q4: Pretrained vs Not Pretrained (2%)

1. Describe

a. The configuration of the model and how do you train this model

- Config

```
{  
  "architectures": [  
    "BertForQuestionAnswering"  
  ],  
  "attention_probs_dropout_prob": 0.1,  
  "classifier_dropout": null,  
  "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,  
  "position_embedding_type": "absolute",  
  "torch_dtype": "float32",  
  "transformers_version": "4.22.2",  
  "type_vocab_size": 2,  
  "use_cache": true,  
}
```

```
"vocab_size": 30522  
}
```

- Training

I choose to train QA model. I choose Bert as the model of this task and train it with same training data. For the tokenizer, I used pretrained tokenizer from `hfl/chinese-roberta-wwm-ext-large`.

Training parameters:

- Epoch: 5
- Learning rate: $3e-5$
- Batch size: 32
- Doc stride: 128
- Max_seq_length: 512

b. the performance of this model v.s. BERT

	Scratch	Bert
EM	0.0465	0.7896