ADL HW2

資工AI所碩一 r11922a05 林聖硯

Q1: Data processing

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

The baseline model that I use for mutiple choice task is **hfl/chinese-bert-wwm-ext** (source). Similar to BERT-chinese model, the tokenization process of this model could be divided into two steps, which are basic tokenizer and wordpiece tokenizer respectively. (source - bert tokenization github)

First, the whole sentence will be fed into the **basic tokenizer**, during which the input will be turned into unicode, invalid unicode character will be removed, white space will be added in front of and after the character, punctuation will be splitted and finally split the sentence again with white spaces. The result of this basic tokenizer are tokens of the original sentence. Then, these tokens will be processed through the **wordpiece tokenizer**. In this process, we'll perform use a greedy longest-match-first algorithm to perform tokenization and turn the tokens into subwords (i.e. token that starts with ##)

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

The object `offset_mapping` (source) of bert tokenizer could help us determine the corresponding start and end character in the original text given the original our token. `offset_mapping` is a tuple which stores the start position of the corresponding character and end position of the corresponding character of a token in a tokenized sentence. Thus, if we have the start and end position (char-level) of the answer, we could use `offset_mapping` to determine the start and end position of the answer token.

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?

source - SQUAD qa colab, utils_qa.py (@hugging face github repo)

Step1: we need to deal with impossible answer. There are two situations that we consider it as an impossible answer.

1. the indices are out of bounds
2. the indices correspond to part of the input ids that are not in the context

In these two cases, we'll assign start and end indices corresponding to the index of the CLS token and still compute their output logits. The sum of them is called **the minimum score**

Step2: Loop through all the features associated to the current example. (one question may be connected to different sub-sentences of a context and then taken as model input, the sub-sentences are called features here, thus one question may get different answer based on different features / sub-setences). Then, we will use the score obtained by **adding** the start and end logits of each feature. We won't try to order all the possible answers and limit ourselves to with a hyper-parameter we call `n_best_size`. We'll pick the best indices in the start and end logits and gather all the answers this predicts. After checking if each one is valid, we will sort them by their score and keep the best one.

Q2: Modeling with BERTs and their variants

1. Describe
    a. your model (configuration of the transformer model)

    The baseline model that I use is **hfl/chinese-bert-wwm-ext**

```
BertConfig {
  "_name_or_path": "hfl/chinese-bert-wwm-ext",
  "architectures": [
    "BertForMaskedLM"
  ],
  "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,
  "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",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


b. performance of your model.

Multiple choice acc (valid set)	0.956
Question answering EM (valid set)	0.8054
Kaggle public score (test set)	0.77396

c. the loss function you used.

nn.CrossEntropyLoss()

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

optimizer = AdamW, learning rate = 3e-5, batch size = 2

2. Try another type of pretrained model and describe (2%)
   a. your model (configuration of the transformer model)

The second model that I used is **hfl/chinese-roberta-wwm-ext**.

```
BertConfig {
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForMaskedLM"
  ],
  "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",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


b. performance of your model.

Multiple choice acc (valid set)	0.962
Question answering EM (valid set)	0.8189
Kaggle public score (test set)	0.78209

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

The following points are main differences between BERT and RoBERTa

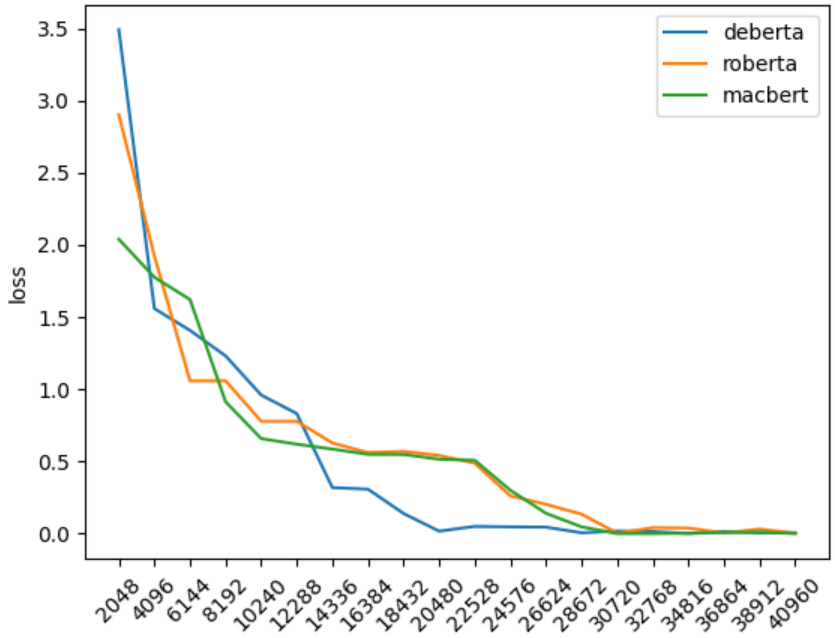
- Roberta (ttl data=160G, BookCorpus (16G), CC-NEWS (76G), OpenWebText (38G) and Stories (31G)) used more training data than BERT (BookCorpus, 16G)

- Roberta used dynamic masking pattern instead of static masking pattern during training phases
- Roberta replaced next sentence prediction (NSP) objective **with full sentences without NSP** (i.e. inputs are packed with sentences that are sampling from one or more documents. When training data is reached the end of document, sentences from other documents will be sampled. We also add an extra separator token between the documents. The number of token is at most 512.)
- Roberta was trained on longer sequences (125K steps with 2k sequences) than Bert-base (1 million steps with a batch size of 256 sequences)

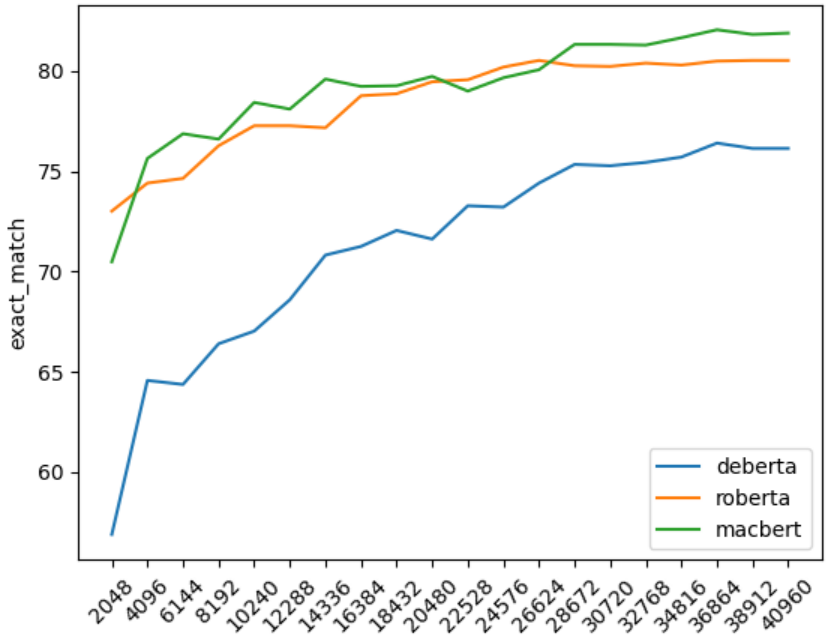
Q3: Curves

1. Plot the learning curve of your QA model

   a. Learning curve of loss



b. Learning curve of EM



Q4: Pretrained vs Not Pretrained

1. Train a transformer model from scratch (without pretrained weights) on the dataset (you can choose either MC or QA)

In this task, I chose the **multiplce choice(MC)** task and I used **bert-base-chinese** model to answer this question.

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

```
BertConfig {
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMaskedLM"
  ],
  "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",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


- To load a unpretrained model, just load `config` from class method `AutoConfig.from_pretrained` method and then use it as input of the class method `AutoModelForMultipleChoice.from_config` rather than directly using `AutoModelForMultipleChoice.from_pretrained`.
- The training setting of this model is the same as the setting that I use for **hfl/chinese-bert-wwm-ext model** as the following shows

```
--max_length=512 \
--per_device_train_batch_size=2 \
--gradient_accumulation_steps=8 \
--weight_decay=0.01 \
--learning_rate=1e-5 \
--lr_scheduler_type=cosine \
--num_train_epochs=1 \
--checkpointing_steps=128 \
--seed=42 \
```

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

Performance	bert without pretrained weight	bert with pertrained weight
MC accuracy	0.57203	0.94571

Q5: Bonus: HW1 with BERTs

1. Train a BERT-based model on HW1 dataset and describe
   a. your model

The model that I use in Q5 is distilbert-base-uncased.


```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.22.2",
  "vocab_size": 30522
}
```


b. performance of your model.

1. Intent classification (1%)
2. Slot tagging (1%)

Task	Valid set acc	Kaggle public score	Kaggle private score
Intent classification	0.95931	0.95022	0.95511
Slot tagging	0.84458	0.83226	0.81233

c. the loss function you used.

nn.CrossEntropyLoss()

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

AdamW, learning rate = 3e-5, batch size = 32