NTU ADL hw1 Report

Q1: Data processing

Tokenizer

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

The pre-trained model I used is `hfl/chinese-roberta-wwm-ext-large`. Roberta's tokenizer is derived from GPT-2 tokenzier, it's a BPE(Byte-Pair-Encoding).
BPE first splits text into words and calculate the frequency, then splits words into chars.
After that, add the most-frequent chars sequence into vocabs.
Continue doing above operation until vocal size reaches `vocab_size` hypermeter.

Answer Span

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

While doing tokenization, we can enable two flags: `return_overflowing_token` and `return_offsets_mapping`.
Then tokenizer will return `overflow_to_sample_mapping` and `offset_mapping` these two data for us to track text and input_ids.
Then we can simplying use while loops to find the positions of answer span.

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

Select the start/end positions below `n_best_size` parameter. Then map both start/end positions combinations back into text.
Select the prediction with most score: probability score of start position + probability score of end position.

Q2: Modeling with BERTs and their variants

Describe

1. **Your model**

I use the pre-trained model **hfl/chinese-roberta-wwm-ext-large** (https://huggingface.co/hfl/chinese-roberta-wwm-ext-large) for both multiple-choice and question-answering tasks.
This is RoBERTa model and used chinese corups to pretrain.
RoBERTa shares the same model architecture with BERT, but a different dynamic masking techniques and tokenizer approach (BPE).

2. **The performance of your model.**

After fine-tuning, multiple-choice model reaches `accuracy` 96.2.
Fine-tuned question-answering model reaches `val_exact_match` 84.3.

3. **The loss function you used.**

Both tasks utilizes CrossEntroyLoss for model optimization.
Mutliple-choice calculates the loss between ground-truth label index and predicted label index.
Question-answerings calculates the loss between ground-truth and predicts both start/end positions, then average them.

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

For aboth training, I uses AdamW optimizer.
MC: Learning rate: 2e-5, Batch size: 4
QA: Learning rate: 8e-5, Effective batch size: 32 (distributed training on 4 gpus)

Try another type of pre-trained LMs and describe

1. **Your model**

At first, I used **bert-base-chinese** (https://huggingface.co/bert-base-chinese).
I also tried hfl/chinese-bert-wwm-ext, hfl/chinese-roberta-wwm-ext.

2. **The performance of your model.**

> bert-base-chinese reachs about 0.78 exact match score.
> hfl/chinese-bert-wwm-ext reachs about 0.79 exact match score.
> hfl/chinese-roberta-wwm-ext reachs about 0.81 exact match score.
> hfl/chinese-roberta-wwm-ext-large reaches about 0.83 exact match score.
> Finally, *-large model with larger batch size gives the best performance.

3. **The difference between pre-trained LMs (architecture, pretraining loss, etc.)**

> BERT and RoBERTa has similar model architectures, but the loss is vary from 1e-5 to 3e-5.
> Pre-trained dataset wwm-ext can have a significant increase on accuracy.

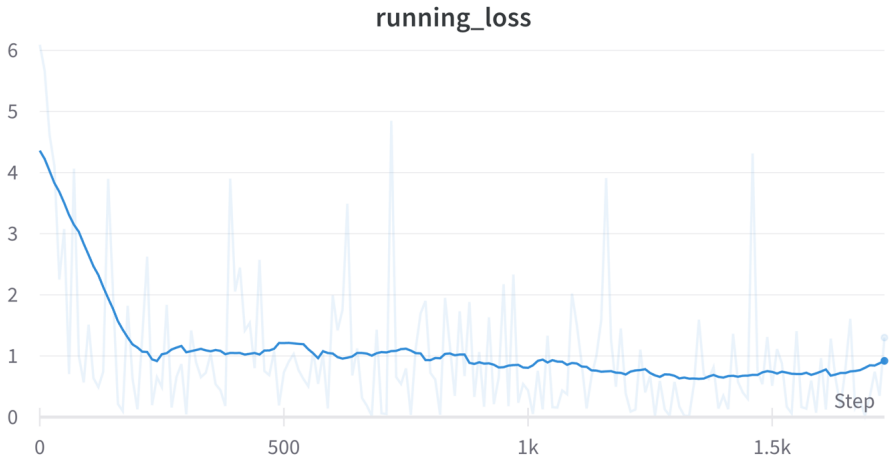
Q3: Curves

Plot the validation set learning curve of your span selection (extractive QA) model. Please make sure there are at least 5 data points in each curve.

1. **Learning curve of the loss value**

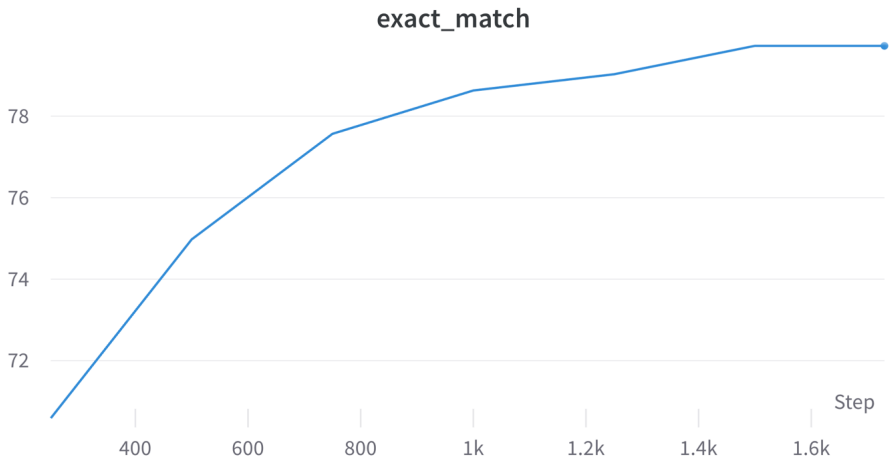
validation loss curve

train running loss curve (with 20 steps running average)



2. **Learning curve of the Exact Match metric value**

validation exact_match curve



Q4: Pre-trained vs Not Pre-trained

Train a transformer-based model (you can choose either paragraph selection or span selection) from scratch (i.e. without pretrained weights).

Describe

1. **The configuration of the model and how do you train this model (e.g., hyper-parameters).**

I adopt the BERT config. But with smaller `hidden_size` , `num_attention_heads` and `num_hidden_layers` as below.
I trained the model from scratch 20 epochs, using learning rate 2e-5 (constant lr_scheduler).
The tokenizer I use is BERTTokenizer.

```
{
    "attention_probs_dropout_prob": 0.1,
    "classifier_dropout": null,
    "hidden_act": "gelu",
    "hidden_dropout_prob": 0.1,
    "hidden_size": 384,
    "initializer_range": 0.02,
    "intermediate_size": 3072,
    "layer_norm_eps": 1e-12,
    "max_position_embeddings": 512,
    "model_type": "bert",
    "num_attention_heads": 4,
    "num_hidden_layers": 4,
    "pad_token_id": 0,
    "position_embedding_type": "absolute",
    "transformers_version": "4.33.3",
    "type_vocab_size": 2,
    "use_cache": true,
    "vocab_size": 30522
}
```


2. **The performance of this model v.s. BERT.**

The best performance I got with from-scratch model is exact match score near 6.5%, which is far from using pre-trained model.
I firstly tried training the model for 20 epochs, but the result is quite poor, it's only a ~4% exact match accuracy.
I then trained the model for 200 epochs, however, the model start over-fitted at about 40 epochs.
So I think the model is not capable with the ability to understand the context and just try to remember the train set inputs.
The pre-trained model really helped a lot for downstream tasks.