ADL HW1 Report

NTU CSIE, R12922051
資工碩一 陳韋傑

Q1: Data processing (2%)

```
>>> input = tokenizer("But what about second breakfast?")
>>> print(input)
{'input_ids': [101, 1252, 1184, 1164, 1248, 6462, 136, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1]}
```

- Tokenizer: I used the corresponding tokenizer for each model. The tokenizer takes a sequence of words and maps it to a sequence of numbers or ids (input_ids) for the model to take as input.
  Also, there are some other attributes that the model needs in order to learn. (For example, token_type_ids denotes if the tokens are in the same sentence, attention_mask denotes if the token is masked)

- Answer Span: I used the same method provided by sample code.
  - After tokenization, if the answer is not given or out of the span, use CLS token index instead. Otherwise, use 'offset_mapping' provided by tokenizer to find the correct answer span.
  - After model prediction, first collect all the start/end logits and rule out the impossible ones, then sort the possible ones by their confidence score (which is start logits + end logits). Finally, use 'offset_mapping' to find the original text, as final prediction.

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

The first result I submitted used BERT (bert-base-chinese).

- Loss Function: CrossEntropyLoss
  (These two task use Classification loss/Total span extraction loss respectively)
- Optimizer: AdamW
- Learning Rate: 5e-5
- Scheduler: Linear
- Epochs: 3
- Valid Batch Size: 8
- Max Sequence Length: 512
- Public score: 0.7676 / 0.7543

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

The best result I submitted used LERT (chinese-lert-base / chinese-lert-large).

- Hyperparameters are the same as BERT and all the other experiments.
- Score: 0.8210 / 0.8229 (Rank 6th / 2nd on Kaggle)

Model Comparison

Compared to BERT, LERT is trained on three types of linguistic tasks (POS, NER, DEP) along with the original MLM task, also LERT used a linguistically informed pre-training (LIP) strategy, which learns fundamental linguistic knowledge faster.

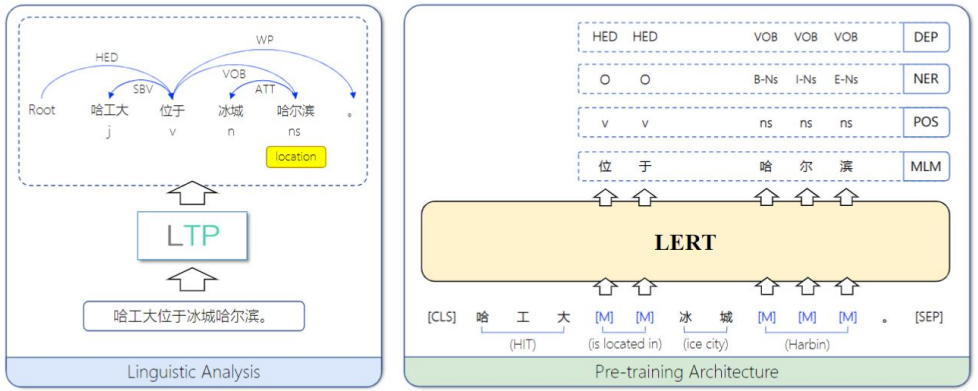


Figure 1: Overview of LERT. We only use one text segment in the input for simplicity, i.e., we still use two text segments for implementation, separated by [SEP] token.

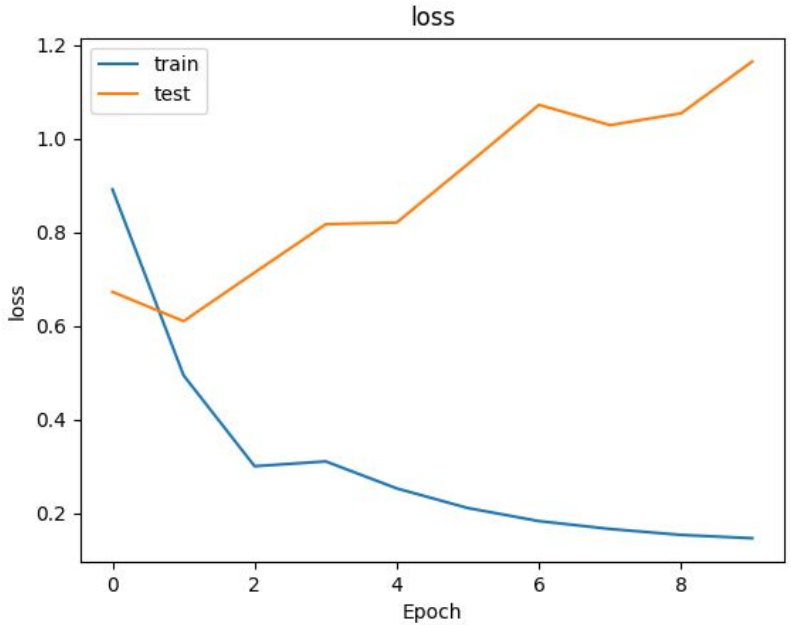
Q2: Modeling with BERTs and their variants (4%)

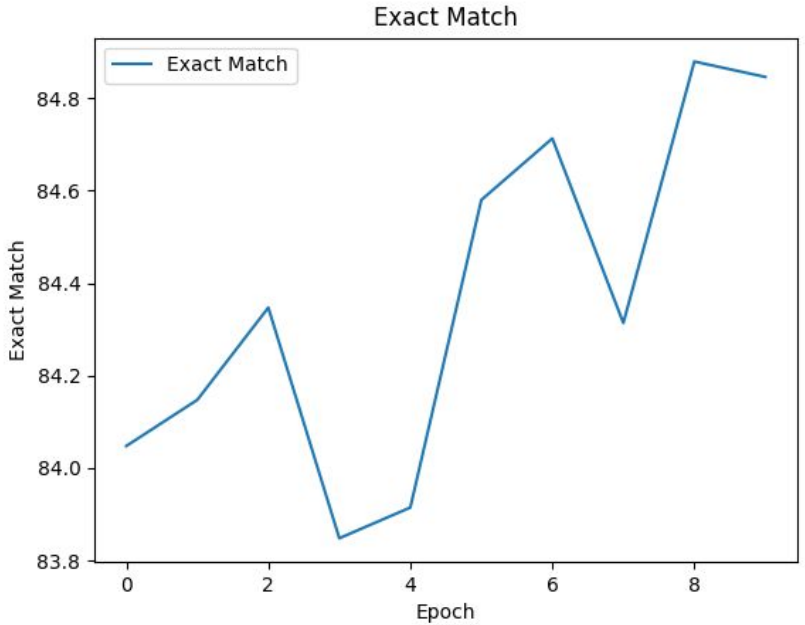
Model/Task performance (Validation)	Multiple Choices (Accuracy)	Extractive QA (Exact Match)	Final Score on Kaggle (Public / Private)
bert-base-chinese	0.9545	0.8275	0.7676 / 0.7543
hfl/chinese-bert-wwm-ext	0.9518	0.8358	0.7613 / 0.7480
hfl/chinese-roberta-wwm-ext*	0.9591	0.8517	0.7911 / 0.8004
hfl/chinese-macbert-base*	**0.9641**	0.8521	0.7984 / 0.7967
hfl/chinese-pert-base*	0.9541	**0.8740**	0.8156 / 0.7931
hfl/chinese-lert-base*	0.9611	0.8657	**0.8210 / 0.8229**

* The experiment is run on base model for multiple choices, large model for extractive QA, due to GPU VRAM limits

Q3: Curves (1%)

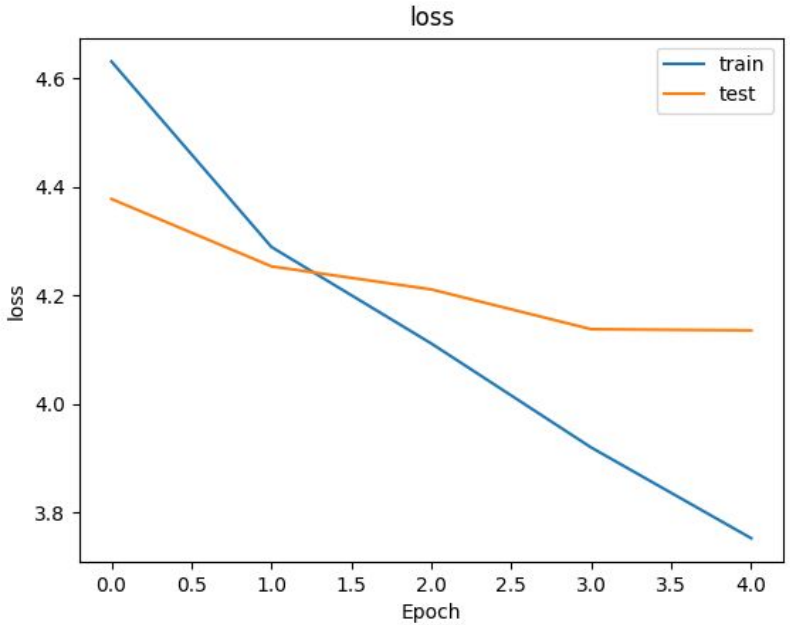
The curves are plotted when trainning pretrained LERT (total epoch = 10).

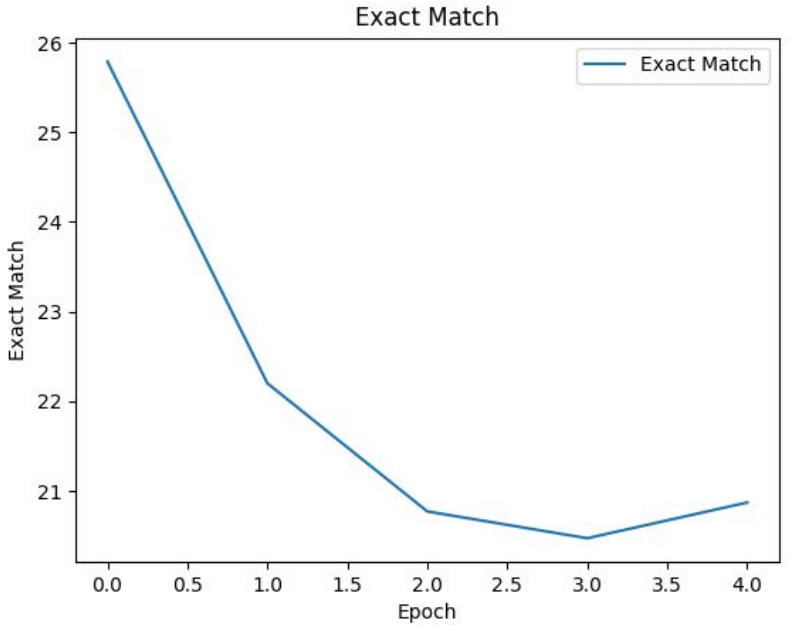




Q3: Curves (1%)

The curves are plotted when trainning BERT from scratch (total epoch = 5).



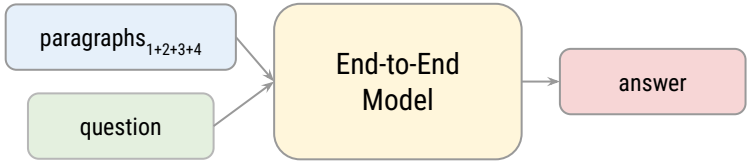


Q4: Pre-trained vs Not Pre-trained (2%)

Model/Task performance (Validation)	Multiple Choices (Accuracy)	Extractive QA (Exact Match)
bert-base-chinese (pretrained)	0.9545	0.8275
bert-base-chinese (scratch, epoch = 5)	0.4221	0.2087

I used the same hyperparameter setting, except that I increase total epochs to 5, in order to let the model learns longer. However, we can see that using BERT to train from scratch is too hard using the given amount of data only and the model cannot perform well in these task.

Q5: Bonus (2%)



Although I don’t have time to implement this question, I have a general idea on how to train a end-to-end model for these two task:

- Concatenate all 4 possible paragraphs as context
- Pass the concatenated context and the question to model
- Directly train the model on extractive QA task

We should choose the model that is able to take longer inputs than BERT, for example, GPT-3 can take around 2048 tokens as input.