ADL HW1 Report

資工碩一 林席葦 **R14922092**

Q1

Tokenizer:
In bert-base-chinese model, it first splits each Chinese character as one token, so do the punctuation marks. For example, “作業一好難” -> [“作”, “業”, “一”, “好”, “難”]。A [CLS] token is added at the start; [SEP] separates question and context.

The next step is WordPiece subword segmentation. It tries to match the longest possible substrings from each token. When a substring can’t be found in the vocab then it’s split into smaller words until matches are found. It’s marked as [UNK] if none match is found.

Answer Span:
“Offset_mapping” is used to map and record the range of tokens in the original string. The start position is the first token whose offset start ≤ the start character, and the end position is the last token whose offset end ≥ the end character. In case the answer spans several tokens, then pick the start/end token index that covers the whole span.

After prediction, the start/end combinations with the top N highest probabilities will be picked. These combinations are filtered by the rules: “start index ≤ end index” and “answer length ≤ max answer length.” It sorts the combinations and picks the highest-scoring valid span.

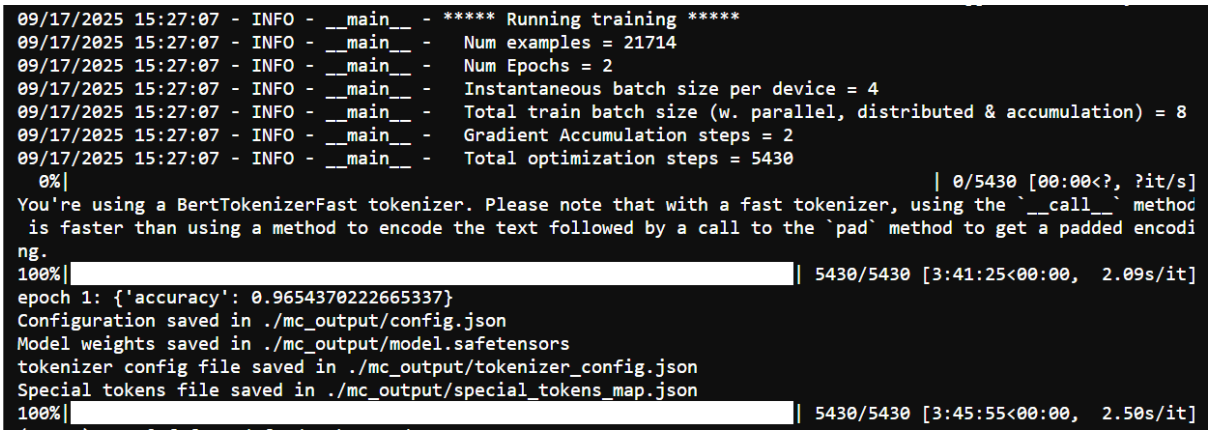
Q2

Initially I tried the model “bert-base-chinese,” and set the following functions and hyperparameters:

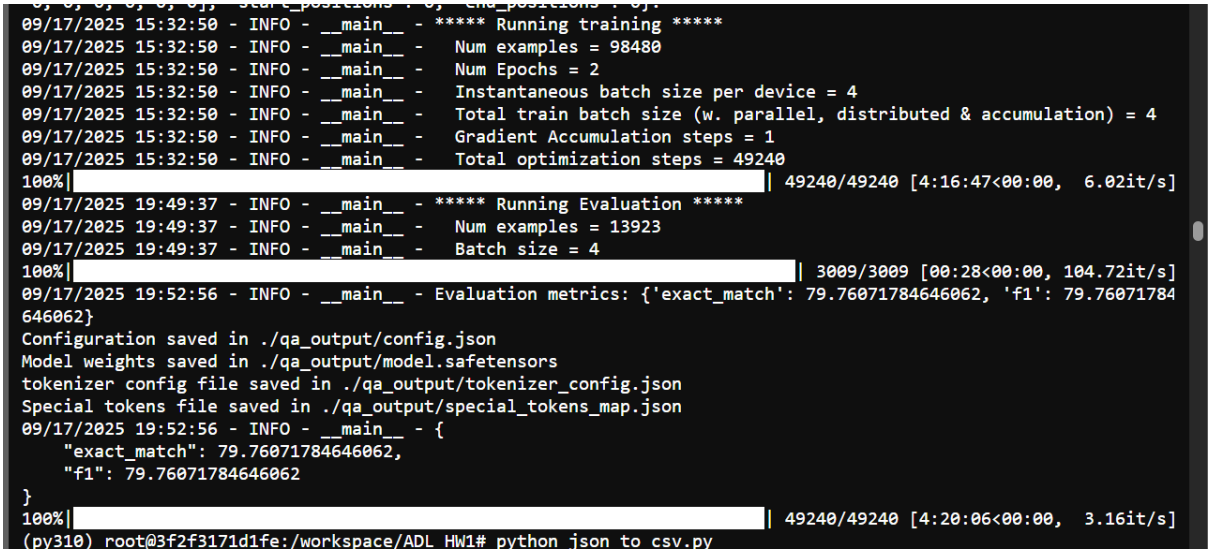
- Loss Function: Cross-Entropy Loss
- Optimizer: AdamW
- Learning Rate: 5e-5
- Epochs: 2
- Per Device Train/Eval Batch Size: 4
- Total Batch Size: 4
- Max Sequence Length: 256
- Public score: 0.6819

However, the performance wasn’t very ideal, so I tried other models and changed some parameters. Another model I’d like to mention here is “hfl/chinese-lert-base,” which yielded a public score of 0.7825 on Kaggle. The hyperparameters were essentially the same as the previous ones. Below are two screenshots of the training process for the chinese-lert-base model.

Multiple Choice:



Extractive QA:



After adjusting some hyperparameters under the chinese-lert-base model, the performance enhanced to a public score of 0.7825 on Kaggle. The max sequence length was changed to 512. The epoch was changed to 3. The gradient accumulation step was changed to 4.

To compare Bert and Lert models, the training data of Bert is mainly Chinese Wikipedia, while Lert is more diverse, including news, Wiki, novels, and even online texts. Besides, Bert only relies on masked language modeling (MLM), while Lert includes MLM plus 3 linguistically-related tasks during pretraining (LIP). Hence, Lert generally performs better in QA and classification tasks.

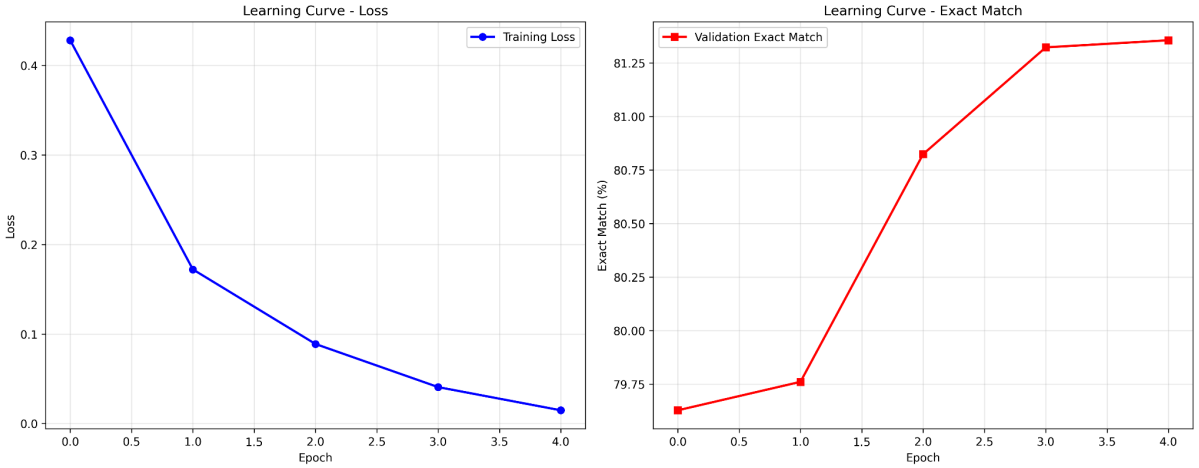
In addition, I also tried other models, and the results are shown in the following table.

Model/Task performance	F1 Score on Extractive QA Task (Exact Match)	Score on Kaggle (Public)
bert-base-chinese	63.1439	0.6819
hfl/chinese-lert-base	79.7607	0.7825

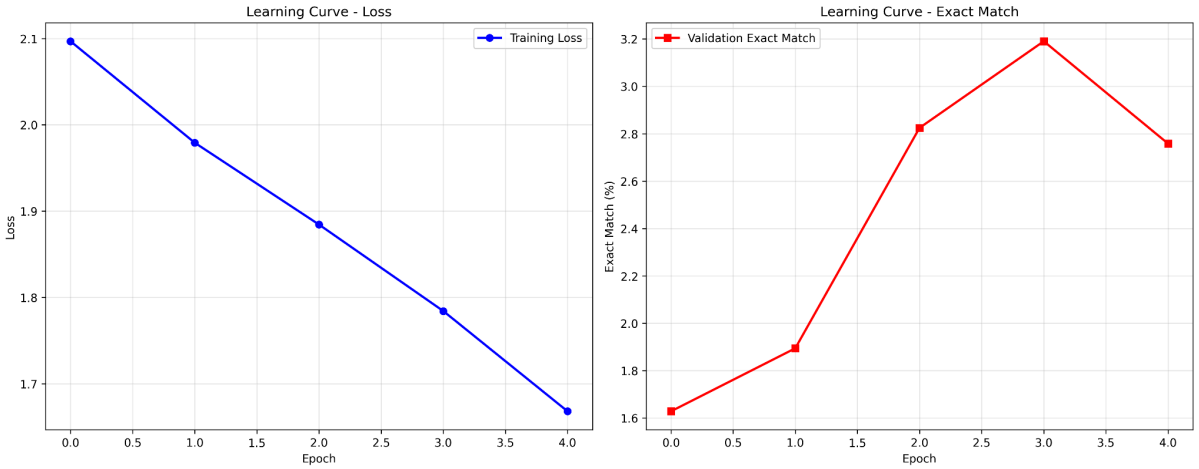
hfl/chinese-roberta-wwm-ext-large	81.7547	0.8012
hfl/chinese-bert-wwm-ext	78.7969	0.7556

Q3

The curves are plotted when training on span selection using a pre-trained chinese-lert-base model (total epoch = 5).



The curves are plotted when training on span selection from scratch (total epoch = 5).



The exact match value of both pre-trained model and training from scratch might increase if there's more total epoch.

Q4

I chose to train a model from scratch for the span selection task, with the model configuration comparison listed below:

Pre-trained BERT:

- Layers: 12
- Hidden Size: 768
- Attention Heads: 12
- Intermediate Size: 3072
- Parameters: ~110M
- Learning Rate: 2e-5

Model Trained from Scratch:

- Layers: 6
- Hidden Size: 512
- Attention Heads: 8
- Intermediate Size: 2048
- Parameters: ~25M
- Learning Rate: 1e-4

Other hyperparameter settings are the same in the two models, both with 5 epochs and chinese-lert-base model. However, as shown in the table, the result of training from scratch significantly underperforms when using a limited amount of training data.

Performance Comparison (chinese-lert-base, epoch = 5):

Model / Task performance	Span Selection (Exact Match)
Pre-trained BERT	0.8136
Train from Scratch	0.0276

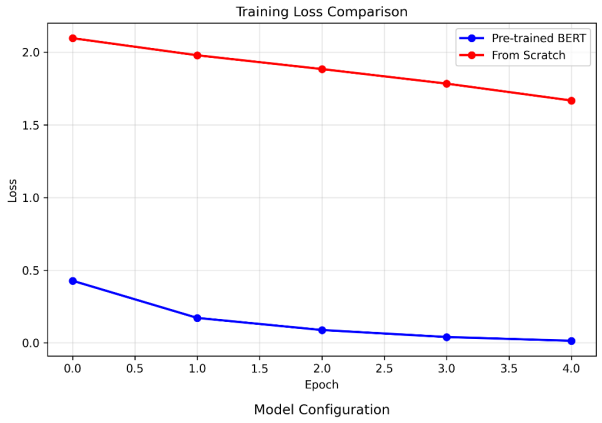
Training Metrics from Pre-trained BERT:

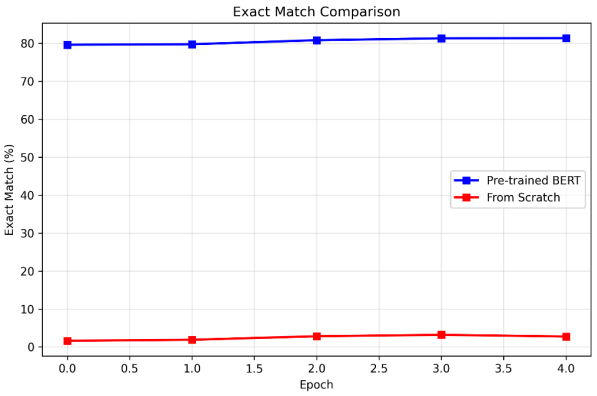
epoch	train_loss	eval_exact_match	eval_f1
0	0.427918	79.627783	79.627783
1	0.172070	79.760718	79.760718
2	0.088760	80.824194	80.824194
3	0.040642	81.322699	81.322699
4	0.014750	81.355932	81.355932

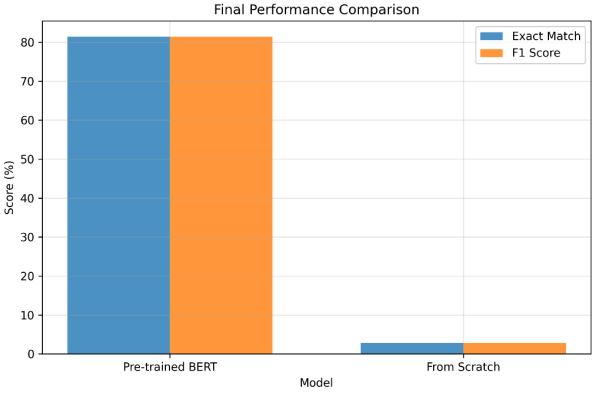
Training Metrics from Scratch:

epoch	train_loss	eval_exact_match	eval_f1
0	2.096856	1.628448	1.672759
1	1.979223	1.894317	1.938629
2	1.884396	2.824859	2.847015
3	1.784209	3.190429	3.234740
4	1.668035	2.758391	2.802703

The figures below compare the two models on their training loss, exact match, and final performance.







Q5

End-to-End Model Architecture (No separate paragraph selection step):
- Base Model: DeBERTa-v3-base
- Context Window: 1024 tokens (vs 512 in 2-step approach)
- Total_loss: QA Loss + 0.1 * Paragraph Loss (QA Loss: Cross-entropy for start/end positions; Paragraph Loss: Encourages attention on relevant paragraphs)
- Modification: Added paragraph scorer for multi-paragraph relevance (Linear layer to weight paragraph relevance)
- Description: The model concatenates all candidate paragraphs into a single long context. It requires more memory.

Optimization Details:
- Algorithm: AdamW
- Learning Rate: 3e-5 with linear warmup
- Batch Size: 1 per device (16 gradient accumulation)
- Mixed Precision: FP16 for memory efficiency

Model Performance (Training Time: 4 hr 15 min):

epoch	train_loss	learning_rate
0	1.0706	2.02e-05
1	0.4183	1.01e-05
2	0.2542	2.18e-08

Screenshot of training the end-to-end model: