# ADL HW1 Report

B09611048 蘇蓁葳

## Data Processing

1. Tokenizer: Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.
   We use Bert-tokenizer for the model and use `AutoTokenizer` to load it into our program.
   The Tokenizer do the things below.
   (1) Split the context:
   The tokenizer will split the context into “readable words”.
   Example: [“ADL 作業好難嗚嗚”] -> [“ADL”, “作”, “業”, “好”, “難”, “嗚”, “嗚”]
   (2) Label the context:
   Since the model can only read “number”, the tokenizer will then convert the words into ids (or vectors) that the training model can read.
   Example: [“ADL”, “作”, “業”, “好”, “難”, “嗚”, “嗚”] -> [1234, 5678, 9876, 5432, 9876, 2345, 2345]
   (3) The AutoTokenizer will generate tokenize_id, attention_mask, offset mapping as well.

2. Answer Span:
   (1) How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?
   AutoTokenizer will generate offset_mapping array to tell us which word correspond to which position, for the example above: [“ADL”, “作”, “業”, “好”, “難”, “嗚”, “嗚”]. In the original string, “ADL” are consider as 3 digits, but after tokenizing, it only represent 1 word, the program call adjust the start position for the tokenized array using the offset_mapping array. (This situation my be more common in English sentence.)

   (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?
   We calculate the score of a result using the sum of start and end logits, then we delete the combinations that does is not reasonable (i.e. end_position < start_position). Then we calculate the probability of the combination by `(exp(score))/(sum of all exp(score)), then choose the combination with the highest probability for the result!

## Modeling with BERTs and their variants

### My model:

1. Multiple question:

   **Base model: hfl/chinise-macbert-base**

   Base model Config:


```
{
  "_name_or_path": "hfl/chinese-macbert-base",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "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": "4.34.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


   Model Performance: 0.9668

| Training parameters | Value/method |
|---|---|
| Learning rate | 2e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 1 |
| optimization algorithm | AdamW |

2. Question answering:

**Base model: hfl/chinise-macbert-large**

**Base model config:**


```
{
  "_name_or_path": "hfl/chinese-macbert-large",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "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,
  "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": "4.34.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


Model Performance: 0.8239

| Training parameters | Value/method |
|---|---|
| Learning rate | 3e-5 |
| Batch size | gradient_accumulation_steps (2) * per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 3 |
| optimization algorithm | AdamW |

**Other model:**

1. Multiple question:

   **Base model: ckiplab/bert-tiny-chinese**

   **Base model config:**


```
{
  "_name_or_path": "ckiplab/bert-tiny-chinese",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 312,
  "initializer_range": 0.02,
  "intermediate_size": 1248,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 4,
  "pad_token_id": 0,
  "pooler_fc_size": 312,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "tokenizer_class": "BertTokenizerFast",
  "torch_dtype": "float32",
  "transformers_version": "4.34.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


   Model Performance: 0.53772

| Training parameters | Value/method |
|---|---|
| Learning rate | 2e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 1 |
| optimization algorithm | AdamW |

2. Question answering:

**Base model: ckiplab/bert-tiny-chinese**

**Base model config:**


```
{
  "_name_or_path": "ckiplab/bert-tiny-chinese",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 312,
  "initializer_range": 0.02,
  "intermediate_size": 1248,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 4,
  "pad_token_id": 0,
  "pooler_fc_size": 312,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "tokenizer_class": "BertTokenizerFast",
  "torch_dtype": "float32",
  "transformers_version": "4.34.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


Model Performance: 0.06945 (first epoch), 0.437078(fifth epoch)

| Training parameters | Value/method |
|---|---|
| Learning rate | 3e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 5 |
| optimization algorithm | AdamW |

**Differnece between my model and the other models:**

Difference config:


```
{
  "_name_or_path": "ckiplab/bert-tiny-chinese",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 312,
  "initializer_range": 0.02,
  "intermediate_size": 1248,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 4,
  "pad_token_id": 0,
  "pooler_fc_size": 312,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "tokenizer_class": "BertTokenizerFast",
  "torch_dtype": "float32",
  "transformers_version": "4.34.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

```
{
  "_name_or_path": "hfl/chinese-macbert-large",
  "architectures": [
    "BertForQuestionAnswering"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "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,
  "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": "4.34.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


Main difference:

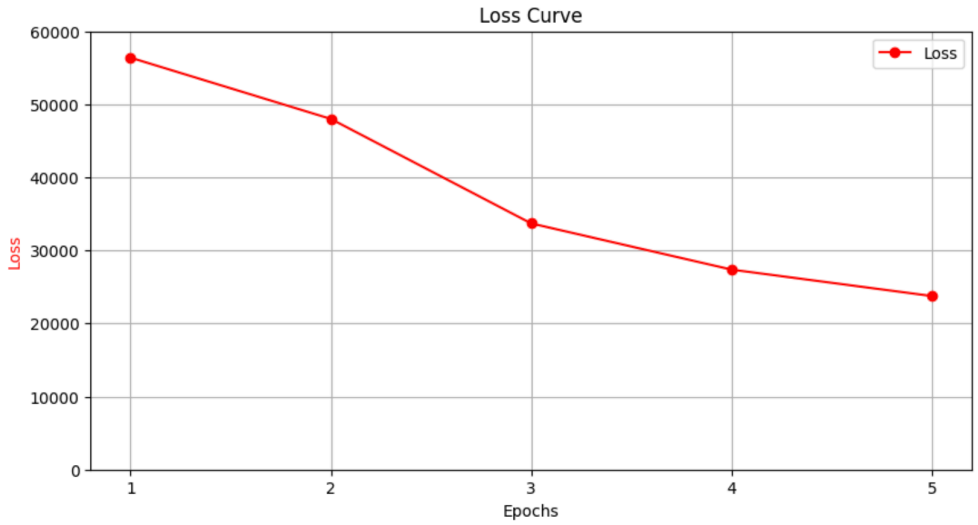
**ckiplab/bert-tiny-chinese** has much smaller hidden layers, (only 4, comparing to 24 for **hlf/chinese-macbert-base**), it has smaller attention head and pooler_fc_size as well, which will have decrease the performance of the model.

And about the architecture, macbert add MLM as mask correction, it will weakens the gap between pre-training and adjustment levels by using similar single-word masks. So it will have better performance comparing to basic bert.

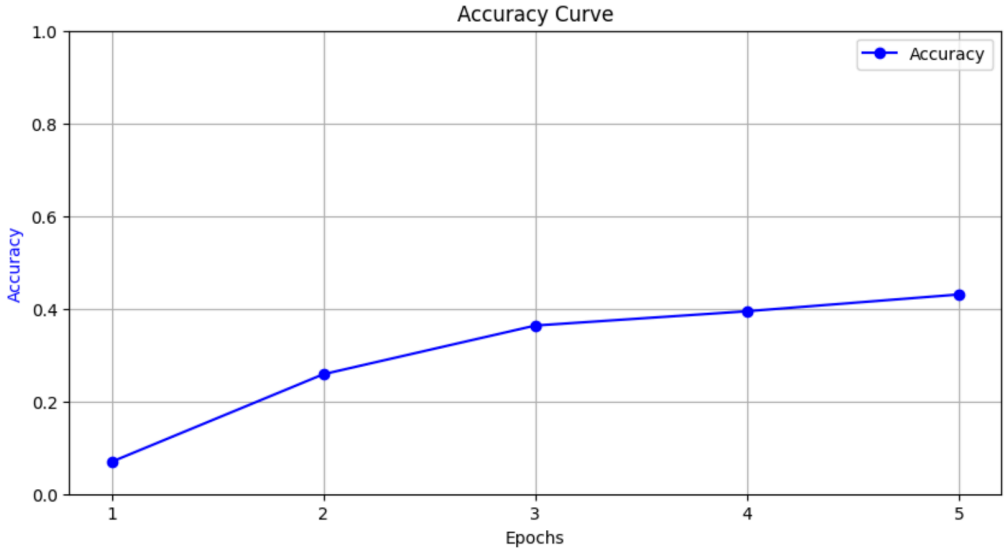
## Curves

I train 5 epochs with the base model hfl/Chinese-macbert-tiny

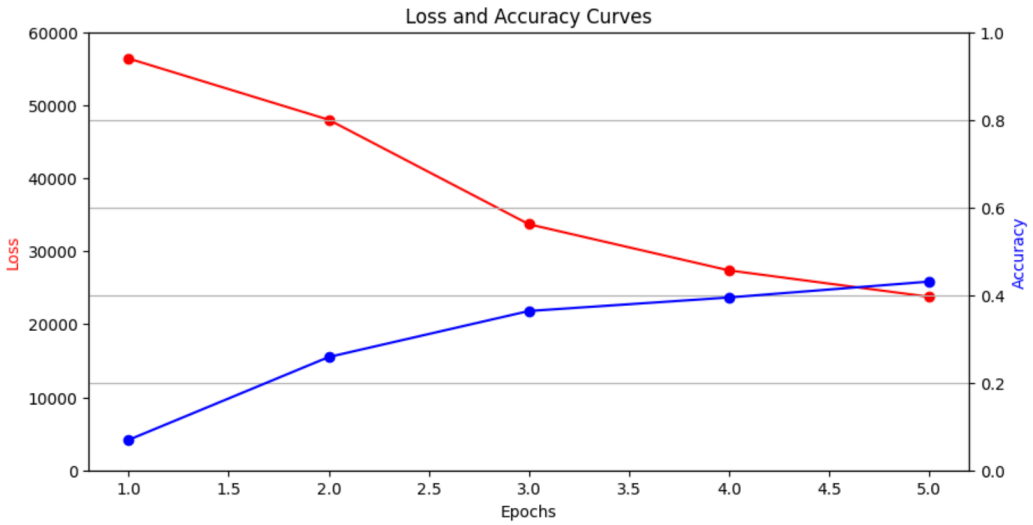
Loss function: Cross Entropy. The loss become lower per epoch.




Accuracy: Exact match, the exact match has increase through per epoch.




Drawing them together:



## Pre-trained vs Not Pre-trained

When training a Pre-trained model, we load the tokenizer, the config and the model to the program:

```
config = AutoConfig.from_pretrained(args.model_name_or_path, trust_remote_code=args.trust_remote_code)
tokenizer = AutoTokenizer.from_pretrained(
            args.model_name_or_path,
            use_fast=not args.use_slow_tokenizer,
            trust_remote_code=args.trust_remote_code
          )
model = AutoModelForMultipleChoice.from_pretrained(
            args.model_name_or_path,
            from_tf=bool(".ckpt" in args.model_name_or_path),
            config=config,
            trust_remote_code=args.trust_remote_code,
        )
```

And when we want to train a not Pre-trained model from an existing structure, we load the tokenizer and the config only. We get the config and the tokenizer from our args.model_name_or_path, and build the not-pre-trained model structure from config.

```
config = AutoConfig.from_pretrained(args.model_name_or_path, trust_remote_code=args.trust_remote_code)
tokenizer = AutoTokenizer.from_pretrained(
            args.model_name_or_path,
            use_fast=not args.use_slow_tokenizer,
            trust_remote_code=args.trust_remote_code
        )
model = AutoModelForMultipleChoice.from_config(config, trust_remote_code=args.trust_remote_code)
```

The config of my not pre-trained model:


```
{
  "_name_or_path": "hfl/chinese-macbert-base",
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "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": "4.34.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


Training parameters:

| | Value/method |
|---|---|
| Learning rate | 3e-5 |
| Batch size | gradient_accumulation_steps (2) * per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 5 |

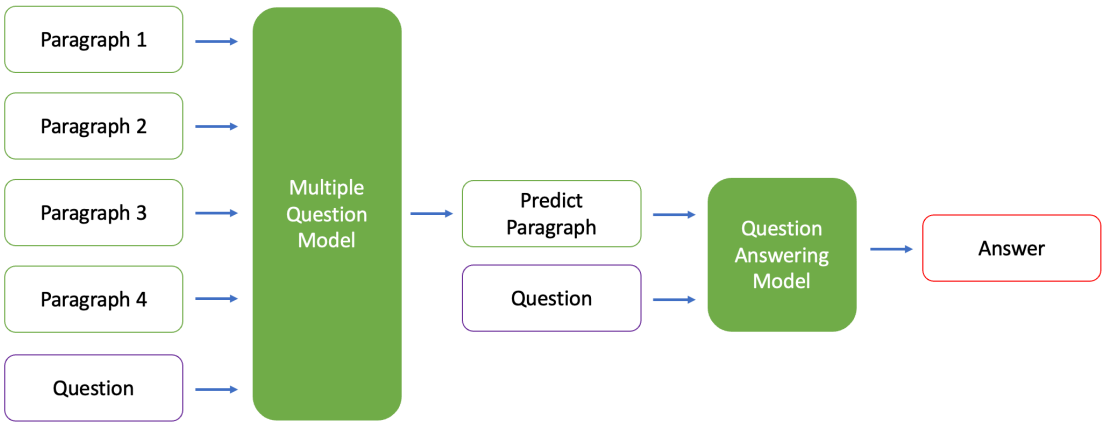
The performance of the non pretrained model (Training on the multiple question problem):

| Model | |
|---|---|
| Pre-trained (hfl/chinese-macbert-base) | 0.9667663675639747 |
| Not-pretrained | 0.5357261548687271 |

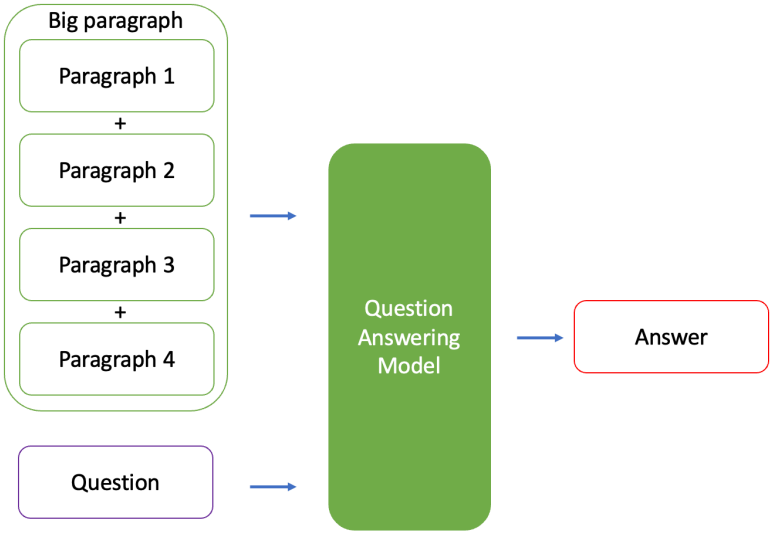
## End to end model

I first stitch the four paragraphs together, and adjust the start number of the answer text for training. Then we use the QA model to train and predict the answer. (Then we can view as a big paragraph and a corresponding start number, it is the same as the second part (question answering part) of the original model.

Original model:




End to end model:




Code:

```
for j in range(len(examples["paragraphs"])):
    big_paragraph = ''
    answer_insex = 0
    for index in range(len(examples["paragraphs"][j])):
      # print([examples["paragraphs"][j][index]])
      big_paragraph += context_list [examples["paragraphs"][j][index]]
      if examples["paragraphs"][j][index] == examples["relevant"][j]:
        answer_index = index
        # print(answer_index)
    # second_sentence = [context_list [index] for index in examples["relevant"]]
    second_sentence.append(big_paragraph)
    # print(big_paragraph)

    for i in range(answer_index):
      examples["answer"][j]["start"] += len(context_list [examples["paragraphs"][j][i]])
```

I tried various base model for this approach, with different parameters, and I choose 3 of them to do futher explanation.

1. bert-base-chinese

   Model Performance: 0.0

| Training parameters | Value/method |
|---|---|
| Learning rate | 2e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (2) = 4 |
| Loss Function | Cross Entropy [1] |
| training epochs | 1 |
| optimization algorithm | AdamW |

   Bert-base-chinese can only handle mostly 512 sequency length, so it is normal the accuracy is such poor. I'd observed the answer and most of them stops at offset 512.

2. hfl/chinese-xlnet-base with 5e-5 learning rate

   Model Performance: 0.0

| Training parameters | Value/method |
|---|---|
| Learning rate | 5e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (1) = 2 |
| Loss Function | Cross Entropy [1] |
| training epochs | 1 |
| optimization algorithm | AdamW |

   I then switch to xlnet, a model with larger context window, But the result of the model is mostly "empty", and its prediction accuracy is very low as well.

3. hfl/chinese-xlnet-base with 2e-5 learning rate and 3 epochs

   Model Performance: 0.003656

| Training parameters | Value/method |
|---|---|
| Learning rate | 5e-5 |
| Batch size | gradient_accumulation_steps (2) *<br>per_device_train_batch_size (1) = 2 |
| Loss Function | Cross Entropy [1] |
| training epochs | 1 |
| optimization algorithm | AdamW |

   I increase the epoch and the learning rate for xlnet, after 3 epochs, the accuracy has shortly increase but it is still poor. I think training more epochs might help increase the accuracy.

**Reference:**


1. The loss function of the model:
   https://github.com/huggingface/transformers/blob/main/src/transformers/models/bert/modeling_bert.py#L1808
   We can find the applied lost function of the models in the huggingface transformer bert source code
2. Bert tokenizer:
   https://huggingface.co/learn/nlp-course/zh-CN/chapter2/4?fw=tf