ADL HW2 Report R11944004

Student ID: R11944004

Department: GINM 11

Name: 李勝維

Q1: Data processing

1. Tokenizer

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

Though the English bert tokenizer utilizes subword tokenization, the Chinese bert tokenizer only splits the sequence into each individual words. ex: 李勝維 is tokenized into {李, 勝, 維}.

Aside from splitting each words, the tokenizer also adds special tokens:

1. [CLS] token: This token represents the start of each sequence. Its purpose is to represent the whole sequence. (i.e., the contextual embedding of [CLS] is the sequence embedding)
2. [SEP] token: This token seperates the two sentences in each input. In this instance, [SEP] seperates the paragraph and the question.
3. [PAD] token: This token pads the sequence of each batch to the same length.
4. [UNK] token: This token represents the words that are not in the dictoinary of the tokenizer.

2. Answer Span

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

Thanks to the huggingface tokenizers library, its RUST implementation tokenizers provide some usefull utility functions, including the char_to_token() function. I use the char_to_token() function to find the token index of the tokenized output. The input of the function is the original index of the word, and it outputs the corresponding word’s token index in the tokenized sequence.

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?

Before I answer this question, I have to describe how I obtained the probability of answer span start/end position. I first splits the paragraph into disjoint windows with length = 20, than I predict the probability of start/end span for each window.

After obtaining the start/end span probability of each window, I choose the window with the largest probability of “start probability + end probability ,” then I get the span inside the choosen window. Finally, I generate paired chinese parenthesis such as 「, 」, 《, … if an unpaired parenthesis exists in the result.

Q2: Modeling with BERTs and their variants

1. Describe

I choose to divide the problem into two parts: multiple choice and question answering. The MC model first chooses the most relevant paragraph out of four options, than the QA model generates the answer according to the paragraph and question.

So, I will describe the two models in each section.

a. Configuration of the transformer model

Multiple choice model

```
{
  "_name_or_path": "hfl/chinese-bert-wwm-ext",
  "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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

Question answering model

```
{
  "_name_or_path": "hfl/chinese-bert-wwm-ext",
  "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": 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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

b. Performance of your model.

Dataset	EM
Validation set	0.690
Public test set	0.705

c. The loss function you used.

I use cross-entropy loss to optimize the both model.

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

Multiple choice model

Optimizer: AdamW

Learning rate: 0.00002 (2e-5)

Batch size: 32 (8 * 4 accumulation steps)

Epoch: 9

LR scheduling: linear warmup for the first 300 steps

Others: bf16 mixed precision training

Question answering model

Optimizer: AdamW

Learning rate: 0.00002 (2e-5)

Batch size: 32 (8 * 4 accumulation steps)

Epoch: 10

LR scheduling: linear warmup for the first 300 steps

Others: fp16 mixed precision training

2. Try another type of pretrained model and describe

a. Configuration of the transformer model

I use macbert-large to replace bert-base as my pretrained model.

Multiple choice model

```
{
  "_name_or_path": "hfl/chinese-macbert-large",
  "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": 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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

Question answering model

```
{
  "_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": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```

b. Performance of your model.

The performance difference of multiple choice model:

Setting	Validation Accuracy
bert-base	0.96
macbert-large	0.97

The performace difference of the whole pipeline:

Setting	Public test set EM
bert-base	0.725
macbert-large	0.778

c. The difference between pretrained model

The main difference between macbert and bert is the pretraining objective. The authors of macbert wants to solve the discrepency between pretraining and finetuning that the [MASK] token doesn't appear during finetuning.

The MAC stands for "mask as correction". In particularly, instead of masking words with [MASK] token, it utilizes existing word2vec embeddings to find the most similar word, and "mask" the word with the most similar word. Besides MAC, the authors also introduces N-gram masking and whole word masking to the training procedure.

Here are some examples of different masking strategies:

Method	Result
Original sentence	we use a language model to predict the probability of the next word.
Tokenized sentence	we use a language model to pre## ##di ##ct the pro## ##ba ##bility of the next word.

Method	Result
Bert mask	we use a language [M] to [M] ##di ##ct the pro [M] ##bility of the next word .
Whole word masking	we use a language [M] to [M] [M] [M] the [M] [M] [M] of the next word .
N-gram masking	we use a [M] [M] to [M] [M] [M] the [M] [M] [M] [M] [M] next word .
Mask as correction	we use a text system to ca ##lc ##ulate the po ##si ##bility of the next word .

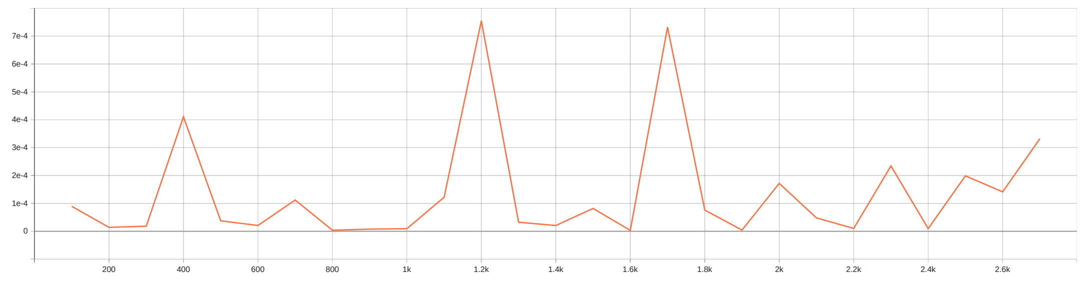
We can see that MAC replaces language → text, model → system, predict → calculate, etc.

Macbert also learns to reconstruct the original sequence by a classifier head, which uses cross-entropy loss to optimize the model.

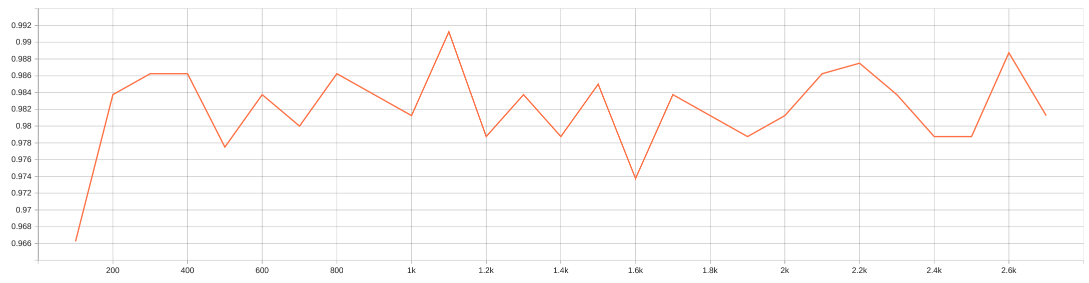
The other components (tokenizer, model architecture, etc.) are identical to bert.

Q3: Curves

1. Plot the learning curve of your QA model
    a. Learning curve of loss (0.5%)



    b. Learning curve of EM (0.5%)



The horizontal axis represents training steps (i.e., each mini-batch)

The curves are logged per 100 steps

Q4: Pretrained vs Not Pretrained

I trained the model on the MC dataset.

1. The configuration of the model and how do you train this model

Configuration (identical to bert-base)

```
{
  "architectures": [
    "BertForMultipleChoice"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "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,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}
```

How do I train the model

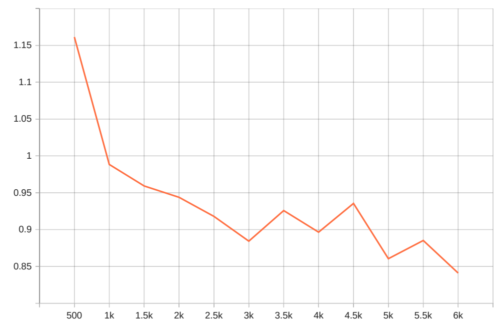
I directly train the model to minimize the cross-entropy loss of the softmax normalized outputs for four paragraphs, and the one-hot ground truth

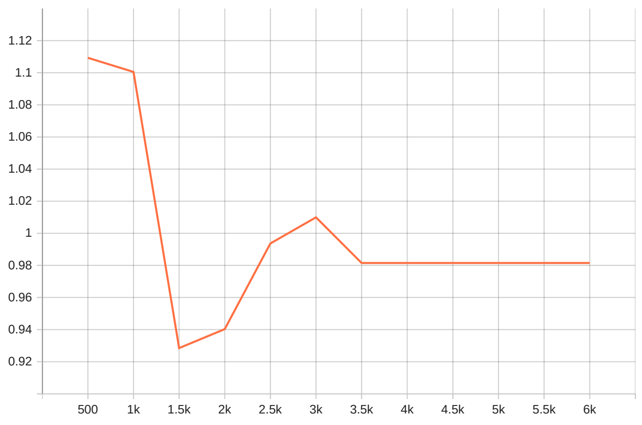
$$loss = CrossEntropy(softmax(\{h_1^{[CLS]}, h_2^{[CLS]}, h_3^{[CLS]}, h_4^{[CLS]}\}), gt)$$

where gt is the grond truth, $h_i^{[CLS]}$ means the contextual embedding of [CLS] token of the i-th paragraph

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

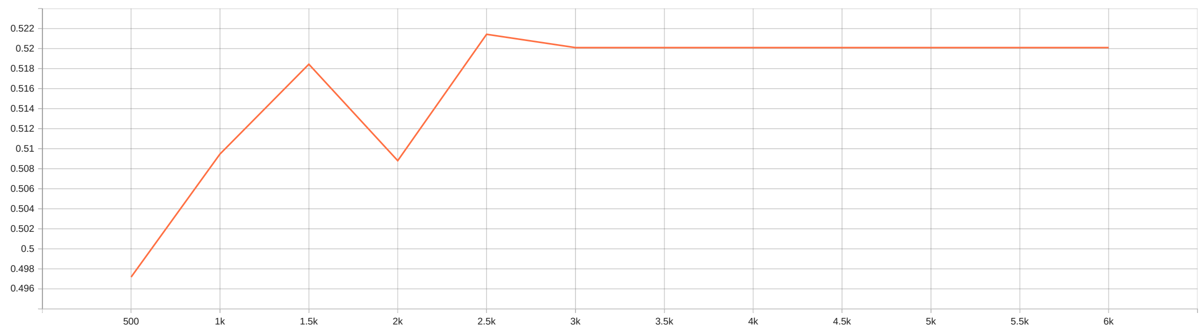
Here are some metrics during training:





The left figure is training loss, and the right figure is validation loss

We can see that the training loss keeps decrease, but the validation loss does not. It seems that the model overfits fairly fast when it is only trained on a small dataset.



The is the evaluation accuracy during training, we can conclude that overfitting occurs.

Performance comparision (on MC dataset)

Model	Validation Accuracy
Pretrained	0.96
Trained from scratch	0.52

We can see that the performance benefits significantly from pretraining.

Q5: Train a BERT-based model on HW1 dataset and describe

a. Your model

I use distilbert-base-uncased as my model for both quesitons in HW1. The advantage of distilbert over bert is that it performs similarly, but has a way smaller model size (6 layers vs 12 layers) compared to bert.

The configuration of my model is

```
{
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForSequenceClassification" / "DistilBertForTokenClassification"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "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,
  "torch_dtype": "float32",
  "transformers_version": "4.22.2",
  "vocab_size": 30522
}
```

b. Performance of your model

The RNN results are from my own implementation of HW1.

The code for this question is under `/r11944004/Q5 bonus/`

Intent classification

Method	Validation Accuracy
RNN	0.891
distilbert	0.945

Slot tagging

Method	Validation F1-score	Validatoin Sequence Accuracy
RNN	0.78	0.798

Method	Validation F1-score	Validatoin Sequence Accuracy
distilbert	0.83	0.825

We can conclude that even with an underpowered bert-base method, distilbert, it still outperforms the RNN-based method by a large margin

c. The loss function you used

I use the cross entropy loss to optimize both models.

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

Intent classification

Optimizer: AdamW

Learning rate: 0.00002 (2e-5)

Batch size: 16

Epoch: 5

Weight decay: 0.01

Slot tagging

Optimizer: AdamW

Learning rate: 0.00002 (2e-5)

Batch size: 16

Epoch: 5

Weight decay: 0.01