ADL 2022 Spring – Homework 2 Report

資工所碩一　吳承光　R10922186

Q1: Data Processing (2%)

1. Tokenizer (1%)

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

Take "bert-base-chinese" as an example, the tokenizer performs a subword tokenization algorithm WordPiece to tokenize English words, and character-based tokenization to tokenize Chinese characters.

a. WordPiece: it first initializes a vocabulary to include every character in the training corpus, and then learns to merge them into pairs that maximize the likelihood of training corpus. It does so by finding the symbol pair (s_1, s_2) of which $\frac{P(s_1,s_2)}{P(s_1)P(s_2)}$ is the greatest among all symbol pairs, where P(x) denotes the probability of x. (Reference: [3])

b. Character-based tokenization: this algorithm just treats each Chinese character as a token.

Additionally, there are some special tokens added to the tokenized sentence to serve special purpose. [PAD] is for padding, [UNK] is for unknown token, [MASK] is for the masked-language-modeling (MLM) pre-training objective, [CLS] is added to the beginning of the sentence for classification purpose, and [SEP] is for partitioning sentences if required.

The following example shows the real tokenization result performed by the bert-base-chinese tokenizer:

Original sentence: [我有幾張 tables]

Tokenized sentence: [[CLS]，我，有，幾，張, table, ##s, [SEP]]

* Note: ## denotes the continuation of the previous token

2. Answer Span (1%)

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

After BERT tokenization, the original positions of the tokens can be retrieved by offset mapping. Afterwards, the token whose original positions correspond to / contain the start position is the start token, and the token whose original positions correspond to / contain the end position is the end token.

(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?
A naïve way to do this is to attribute a score to each (start position, end position) pair by taking the product of corresponding probabilities of the two positions, and the valid pair (i.e. start position < end position) with the maximum score (probability product) will serve as the final prediction.
The huggingface script (reference: [4]) slightly modifies this process by taking the highest 20 start position logits (y^s) and highest 20 end position logits (y^e), and find the pair with the highest logits summation $y^s + y^e$. One additional limitation is that the answer span should not be greater than a specified max length (L):

$$y^{s*}, y^{e*} = \text{argmax}\left(y^s_{t_s} + y^e_{t_e}\right), 0 < t_e - t_s \leq L$$

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

1. Model Description (2%)

(1) Model Configuration

* Model name: bert-base-chinese (for both models)

* Configuration:


```
BertConfig {
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "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",
  "transformers_version": "4.17.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


(See the next page)

(2) Performance

a. Context Selection

Learning Rate	1e-5	2e-5	3e-5	4e-5	5e-5
Accuracy	0.9644	**0.9654**	0.9608	0.9551	0.9518

(Note: accuracies are computed on the validation set from valid.json)

b. Question Answering (of known ground truth context)

Learning Rate	1e-5	2e-5	3e-5	4e-5	5e-5
Exact Match	0.7926	**0.7930**	0.7873	0.7790	0.7717

(Note: exact matches are computed on the validation set from valid.json)

c. End-to-End Evaluation (QA of unknown context)

Data	Validation Set	Kaggle Public	Kaggle Private
Exact Match (EM)	0.7753	0.7685	0.7552

(Note1: validation set is constructed from valid.json)

(Note2: model configuration of the shown performance is listed in subsection (4))

(3) Loss Function

a. Context selection

The loss (L) of an individual sample (4 question-context sentence pairs) is calculated by the below equation, where $\hat{y}_i$ denotes the ground truth label of whether the i-th context (1-based indexing) is relevant to the question (1 if True and 0 otherwise), and y_i denotes the model's output logits of the i-th question-context sentence pair.

$$L = \sum_{i=1}^{4} \hat{y}_i log(softmax(y_i))$$

b. Question answering

The loss (L) of an individual sample (a question-context pair) is expressed by the following equation:

$$L = -\frac{1}{2}\sum_{t=1}^{T} (\hat{y}_t^s log\big(softmax(y_t^s)\big) + \hat{y}_t^e log\big(softmax(y_t^e)\big))$$

, where $\hat{y}_t^s$ denotes whether the t-th token (1-based indexing) is the start position (1 if True and 0 otherwise), $\hat{y}_t^e$ denotes whether the t-th token is

the end position, y_t^s denotes the start position logit of the *t*-th token, y_t^e denotes the end position logit of the *t*-th token, and *T* denotes the context length.

(4) Optimization Algorithm, Learning Rate, and Batch Size

* Final configuration:

a. Optimization algorithm: Adam (Kingma & Ba, 2014)

b. Learning rate: 2e-5

c. Batch size: 16

(Note: no schedulers are used)

2. Description of Another Type of Pretrained Model (2%)

(1) Model Configuration

* Model name: hfl/chinese-roberta-wwm-ext (for both models)

* Configuration:


```
BertConfig {
  "_name_or_path": "hfl/chinese-roberta-wwm-ext",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "directionality": "bidi",
  "eos_token_id": 2,
  "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,
  "output_past": true,
  "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",
  "transformers_version": "4.17.0",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 21128
}
```


(2) Performance

a. Context Selection

Learning Rate	1e-5	2e-5	3e-5	4e-5	5e-5
Accuracy	0.9634	**0.9668**	0.9644	0.9621	0.9621

(Note: accuracies are computed on the validation set from valid.json)

b. Question Answering (of known ground truth context)

Learning Rate	1e-5	2e-5	3e-5	4e-5	5e-5
Exact Match	0.8179	**0.8232**	0.8159	0.8082	0.8059

(Note: exact matches are computed on the validation set from valid.json)

c. End-to-End Evaluation (QA of unknown context)

Data	Validation Set	Kaggle Public	Kaggle Private
Exact Match (EM)	0.8006	0.7920	0.7886

(Note1: validation set is constructed from valid.json)
(Note2: model configuration: optimizer=Adam, learning rate=2e-5, batch size=16)

(3) Difference between Pretrained Models (e.g. architecture, pretraining loss, etc.)
Some differences between bert-base-chinese and chinese-roberta-wwm-ext are listed below:

a. **Pre-training tasks** (BERT vs. RoBERTa [4]): the BERT model leverages two pre-training tasks, masked-language modeling (MLM) and next sentence prediction (NSP), whereas RoBERTa only uses MLM to pre-train the model.
b. **Word masking** (no-wwm vs. wwm): during pre-training, BERT randomly masks WordPiece tokens without constraints, whereas chinese-roberta-wwm-ext uses whole word masking (WWM) to only mask Chinese "phrases" segmented by segmentation tool to maintain the integrity of the language.
c. **Training data and steps** (no-ext vs. ext [6]): chinese-roberta-wwm-ext adds more training data (5.4 billion tokens in total) to the original dataset used by BERT (0.4 billion tokens in total) and also trains for more steps.

(see the next page)

Q3: Curves (1%)

1. **Plot the Learning Curve of the QA Model**

   * Model of below plots:

   a. QA Model name: hfl/chinese-roberta-wwm-ext

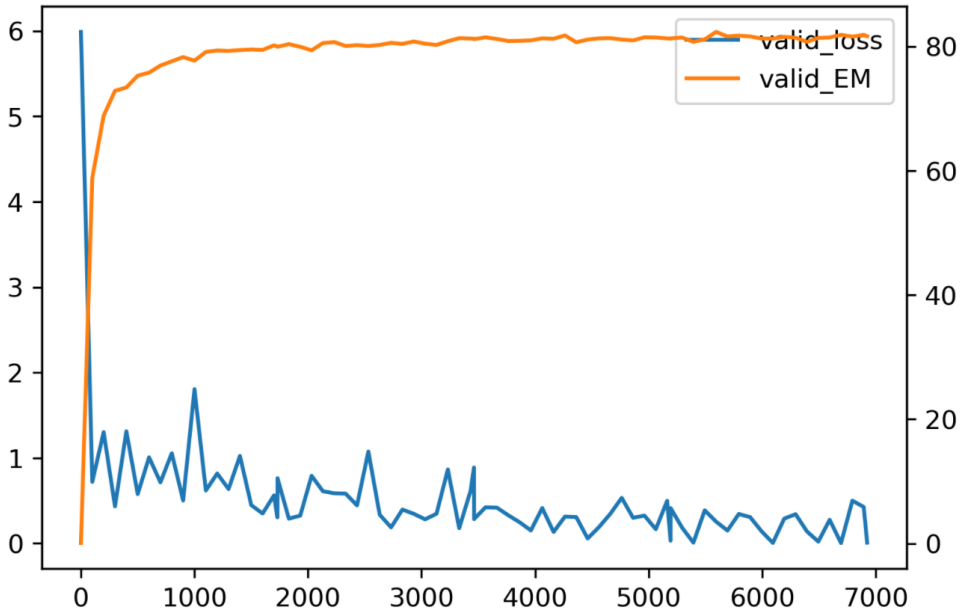
   b. Optimization algorithm: Adam (Kingma & Ba, 2014)

   c. Learning rate: 2e-5

   d. Batch size: 16

   (1) Learning Curve of Loss (0.5%)

   (2) Learning Curve of Exact Match (EM) (0.5%)



(The plots are combined together: blue line = validation loss; orange line = validation exact match (EM); x-axis = training steps; y-axis, left = loss; y-axis, right = EM)

Q4: Pretrained vs Not Pretrained (2%)

1. **Describe Configuration of the Model and Training Process**

   * Here **QA** models are chosen for comparison between pretrained and not pretrained model performance

   (1) Model Configuration

   Here the configurations of three different model sizes are chosen for performance comparison. The differences are highlighted by **red boxes**.

Mini	Small	Base

BertConfig {
"_name_or_path": "bert-base-chinese",
"architectures": [
"BertForMaskedLM"
],
"attention_probs_dropout_prob": 0.1,
"classifier_dropout": null,
"directionality": "bidi",
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 256,
"initializer_range": 0.02,
"intermediate_size": 1024,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 4,
"num_hidden_layers": 4,
"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",
"transformers_version": "4.15.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
}
	BertConfig {
	"_name_or_path": "bert-base-chinese",
	"architectures": [
	"BertForMaskedLM"
	],
	"attention_probs_dropout_prob": 0.1,
	"classifier_dropout": null,
	"directionality": "bidi",
	"hidden_act": "gelu",
	"hidden_dropout_prob": 0.1,
	"hidden_size": 512,
	"initializer_range": 0.02,
	"intermediate_size": 2048,
	"layer_norm_eps": 1e-12,
	"max_position_embeddings": 512,
	"model_type": "bert",
	"num_attention_heads": 8,
	"num_hidden_layers": 4,
	"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",
	"transformers_version": "4.15.0",
	"type_vocab_size": 2,
	"use_cache": true,
	"vocab_size": 21128
	}
		BertConfig {
		"_name_or_path": "bert-base-chinese",
		"architectures": [
		"BertForMaskedLM"
		],
		"attention_probs_dropout_prob": 0.1,
		"classifier_dropout": null,
		"directionality": "bidi",
		"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",
		"transformers_version": "4.15.0",
		"type_vocab_size": 2,
		"use_cache": true,
		"vocab_size": 21128
		}

(2) Training specs

a. Optimization algorithm: Adam (Kingma & Ba, 2014)

b. Learning rate: 2e-5

c. Batch size: 16

d. Number of epochs: 4

2. Performance (Not Pretrained vs. BERT)

(1) Evaluation on Validation Set (valid.json)

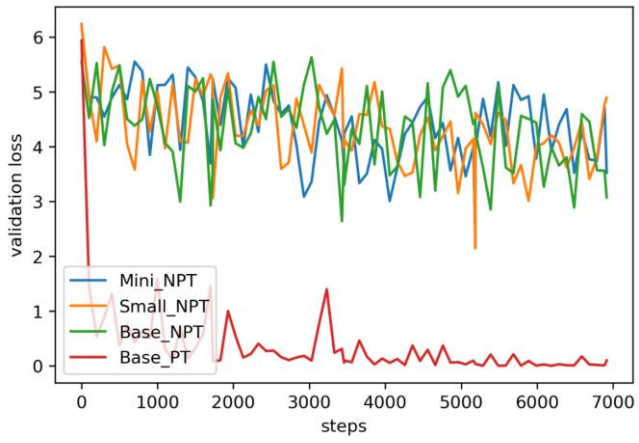
Model	Mini (NPT)	Small (NPT)	Base (NPT)	Base (PT)
Exact Match	0.0449	0.0558	0.0625	0.7930

* Abbreviations: NPT = not pre-trained; PT = pre-trained

* Base (PT) denotes pre-trained bert-base-chinese QA model

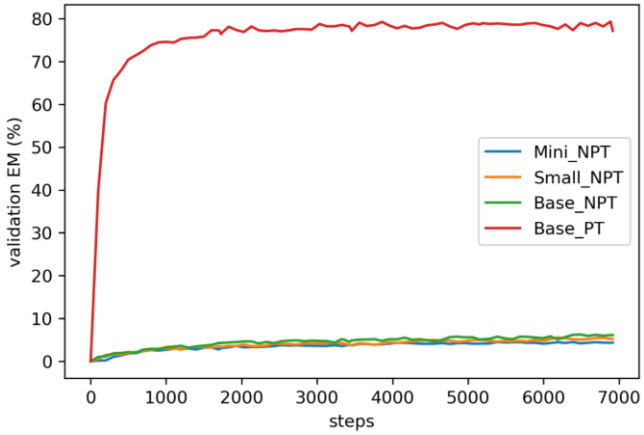
(2) Training Curves

a. Validation Loss

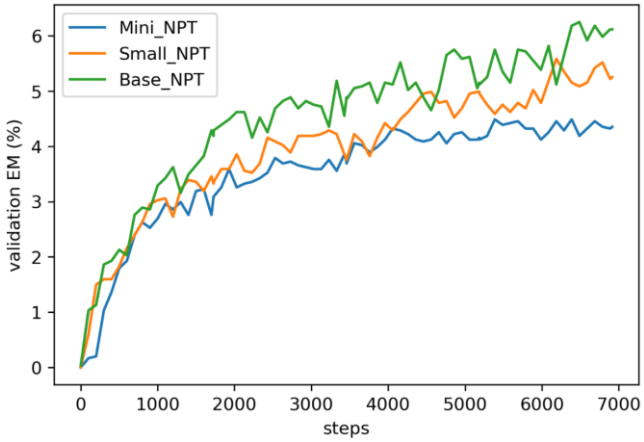


b. Validation Exact Match (EM)

(a) All Curves



(b) Curves of Not Pre-trained Models



From the learning curves of not pre-trained models, one can tell that they haven't converged yet. Therefore, training for more epochs may help boost the performance. However, from the overall comparison of pre-trained versus not pre-trained models, it is obviously that the pre-trained model outperforms not pre-trained ones by a huge margin for the QA task.

Q5: Bonus: HW1 with BERTs (2%)

* Mission: train a BERT-based model on HW1 dataset

1. Intent Classification

(1) Model Description

First, the sequence goes through BERT encoder for context-enriched feature extraction, and the final hidden state of the [CLS] token is fed into a fully-connected (FC) layer to compute the logits of the 150 intent classes:

$$h_{[CLS]}, h_1, \dots, h_T, h_{[SEP]} = BERT(x_1, x_2, \dots, x_T)$$
$$y_1, y_2, \dots, y_{150} = FC(h_{[CLS]})$$

, where $x_1, x_2, \dots, x_T$ denote the input sequence, $h_{[CLS]}, h_1, \dots, h_T, h_{[SEP]}$ denote the hidden states of the final layer, $y_1, y_2, \dots, y_{150}$ denote the logits of the intent classes, and *T* the input sequence length.

(2) Loss Function

Cross entropy is used to compute the loss between prediction and label. In the below equation, *L* denotes the loss of an instance, $\hat{y}_i \in \{0, 1\}$ denotes the ground truth of whether the input belongs to intent class *i*, and y_i denotes the output logit of intent class *i* from the FC layer.

$$L = -\sum_{i=1}^{150} \hat{y}_i \log(softmax(y_i))$$

(3) Optimization Algorithm, Learning Rate, and Batch Size

a. Performance Comparison on Validation Set (Optimizer: Adam)

Name	lr=2e-5	lr=3e-5	lr=5e-5
bs=16	0.9697	0.9677	0.9623
bs=32	0.9683	**0.9703**	0.9663

* Abbreviations: lr=learning rate, bs=batch size

* Note: the choice of learning rates and batch size are according to the recommendations of the original BERT paper (Devlin et al., 2019)

b. Final Hyperparameters

(a) Optimizer: Adam

(b) Learning rate: 3e-5

(c) Batch size: 32

(4) Final Performance

a. Classification accuracy: **0.9703** (on the validation set)

2. Slot Tagging (Not Implemented)

(1) Model Description

(2) Loss Function

(3) Optimization Algorithm, Learning Rate, and Batch Size

(4) Final Performance

(See the next page for reference)

Reference

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171-4186).

[2] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[3] https://huggingface.co/docs/transformers/tokenizer_summary

[4] https://huggingface.co/course/chapter7/7

[5] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[6] https://github.com/ymcui/Chinese-BERT-wwm#%E6%A8%A1%E5%9E%8B%E5%AF%B9%E6%AF%94