《人工智能软件开发与实践》

（ 2023 学年 秋季 学期）

作

业

报

告

学　　号：___

姓　　名：____

班　　级：__

任课教师：___

作业报告

实验名称： **使用 BERT 基于 TEXTCNN 的文本分类**

成绩：

实验类别： 验证/综合型实验　　**实验要求：** 1 人 1 组　**时间：** 2023 年 9 月 14 日

一、 实验目的

基于 pytorch，使用 Bert（动态磁向量），实现 TextCNN 的结构框架，并完成一个文本多分类的任务。

二、 实验要求

需要了解的知识点：

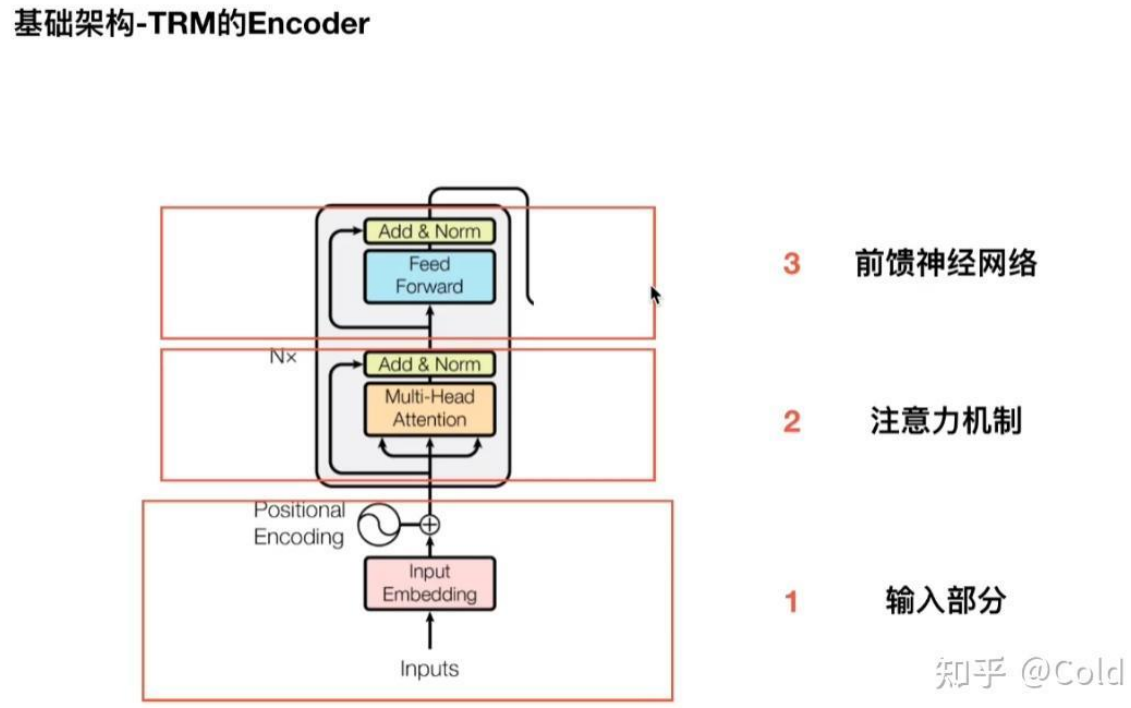
1、 文本进行特征表示：基于词向量的方法

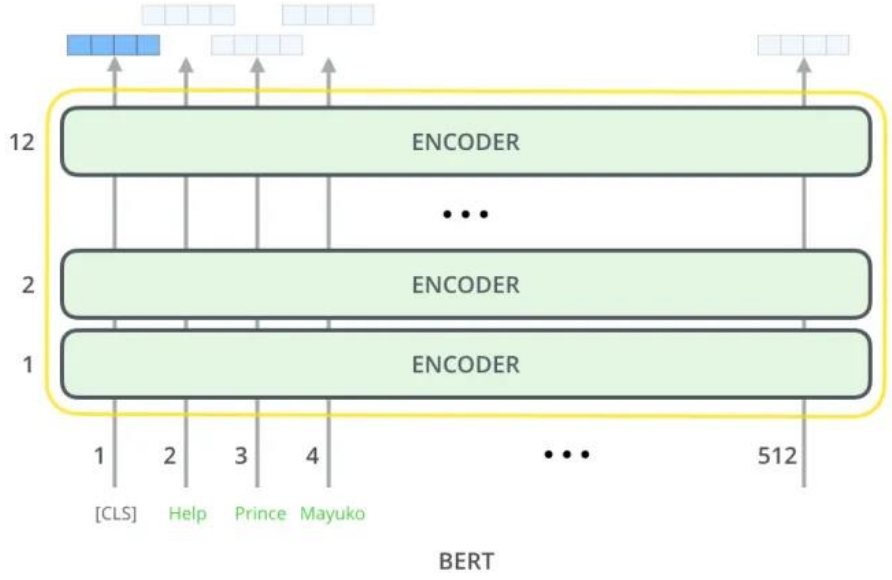
（1） 使用 Bert 预训练的动态磁向量进行初始化，输出不同上下文 apple 对应的词向量。

2、 使用 Bert 代替实验 8 中的 glove 部分，尝试微调，记录当前实验环境下，一个 batch 的训练时间。（时间太长，不用完全等模型收敛，记录后强制结束即可）。

3、 Fix Bert 参数，使用实验 8 的数据和代码，尝试 finetune，记录封闭测试和开发测试结果。

Bert 原理图





三、 实验内容

1. 实验：bert 基本使用

1、 装载英文 bert tokenizer 和 bert-base-cased 模型
2、 用 Bert 为句子”Apple company does not sell the apple.”编码
3、 输出句子转化后的 ID
4、 分别输出句子编码后单词‘CLS’，‘Apple’，‘apple’和‘SEP’,四个词对应的编码
5、 分别计算‘Apple’和‘apple’，‘CLS’和‘Apple’，‘CLS’和‘SEP’之间的距离
6、 输入句子“I have a [MASK] named Charlie.”，重新加载 BertForMaskedLM 模型，通过 bert 预测[mask] 位置最可能的单词
7、 输入句子“I have a cat.”，重新加载 BertForNextSentencePrediction 模型，通过 bert 预测下一句。

四、 使用的算法名称（若无，可以不填）

五、 程序源码

实验 1：将能产生第六部分结果的程序语句写在下面

```
import torch
from transformers import BertTokenizer, BertModel, BertForMaskedLM,
BertForNextSentencePrediction
import logging
import string

# 装载英文Bert tokenizer和bert-base-cased模型
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
model = BertModel.from_pretrained('bert-base-cased')

# 用Bert为句子"Apple company does not sell the apple."编码
sentence = "Apple company does not sell the apple."
```

```
tokens = tokenizer.tokenize(sentence)
print("Tokens:", tokens)
# 输出句子转化后的 ID
sentence_ids = tokenizer.convert_tokens_to_ids(tokens)
print("Sentence IDs:", sentence_ids)
print("Decoded tokens:", tokenizer.convert_ids_to_tokens(sentence_ids))

# 分别输出句子编码后单词'CLS'、'Apple'、'apple'和'SEP'四个词对应的编码
print("Word IDs:")
for word in ['[CLS]', 'Apple', 'apple', '[SEP]']:
    print(word, tokenizer.encode(word, add_special_tokens=False)[0])

# 分别计算'Apple'和'apples'、'[CLS]'和'Apple'、'[CLS]'和'[SEP]'之间的距离
words_list = [('Apple', 'apples'), ('[CLS]', 'Apple'), ('[CLS]', '[SEP]')]
embedding = model.get_input_embeddings().weight
for words in words_list:
    word0vec = embedding[tokenizer.encode(words[0], add_special_tokens=False)[0]]
    word1vec = embedding[tokenizer.encode(words[1], add_special_tokens=False)[0]]
    distant = torch.dist(word0vec, word1vec).item()
    print(f"Distance between {words[0]} and {words[1]}:{distant}")

# 输入句子"I have a [MASK] named Charlie."，重新加载 BertForMaskedLM 模型，通过 bert
预测[mask]位置最可能的单词
logging.getLogger("transformers").setLevel(logging.ERROR)
masked_sentence = "I have a [MASK] named Charlie."
masked_token_ids = tokenizer.encode(masked_sentence, add_special_tokens=True,
padding='max_length', truncation=True, max_length=10, return_tensors='pt')
# 创建 attention_mask
attention_mask = torch.ones_like(masked_token_ids)
bert_masked_model = BertForMaskedLM.from_pretrained('bert-base-cased')
outputs = bert_masked_model(input_ids=masked_token_ids,
attention_mask=attention_mask)
predictions = torch.argmax(outputs.logits, dim=-1)
predicted_word = tokenizer.decode(predictions[0][masked_token_ids.squeeze(0) ==
tokenizer.mask_token_id].item())
predicted_word = predicted_word.replace(" ", "")
print("Predicted word:", predicted_word)

# 输入句子"I have a cat."，重新加载 BertForNextSentencePrediction 模型，通过 bert 预测
下一句。
this_sentence = "I have a cat."
masked_sentence = this_sentence
max_length = 10
predicted_sentence = ''
for i in range(max_length):
    masked_sentence = masked_sentence + " [MASK]"
```

```
    masked_token_ids = tokenizer.encode(masked_sentence, add_special_tokens=True,
padding='max_length', truncation=False,
                                        max_length=2 * max_length, return_tensors='pt')
    attention_mask = torch.ones_like(masked_token_ids)
    outputs = bert_masked_model(input_ids=masked_token_ids,
attention_mask=attention_mask)
    predictions = torch.argmax(outputs.logits, dim=-1)
    predicted_word = tokenizer.decode(predictions[0][masked_token_ids.squeeze(0)
== tokenizer.mask_token_id].item())
    predicted_word = predicted_word.replace(" ", "")
    masked_sentence = masked_sentence.replace("[MASK]", predicted_word)
    predicted_sentence = predicted_sentence + predicted_word + " "
    if predicted_word in string.punctuation:
        break
print("Predicted sentence:", predicted_sentence)
```

六、 程序运行结果（将程序运行结果的截图拷贝至此处，或者填写实验结果）

实验 1 输出：

1、句子切分 token 列表
2、句子转化 ID 列表
3、从 ID 列表还原切分列表
4、‘CLS’，‘Apple’，‘apple’和‘SEP’四个词的编码
5、‘Apple’和‘apple’，‘CLS’和‘Apple’，‘CLS’和‘SEP’之间的距离
6、输入句子“I have a [MASK] named Charlie.”，通过 bert 预测[mask]位置最可能的单词。
7、输入句子“I have a cat.”，通过 bert 预测下一句。

Tokens: ['Apple', 'company', 'does', 'not', 'sell', 'the', 'apple', '.']

Sentence IDs: [7302, 1419, 1674, 1136, 4582, 1103, 12075, 119]

Decoded tokens: ['Apple', 'company', 'does', 'not', 'sell', 'the', 'apple', '.']

Word IDs:

[CLS] 101

Apple 7302

apple 12075

[SEP] 102

Distance between Apple and apples:1.4442212581634521

Distance between [CLS] and Apple:1.9221112728118896

Distance between [CLS] and [SEP]:1.3039658069610596

Predicted word: son

Predicted sentence: '

七、心得体会和遇到的困难

第七项任务句子预测有较大难度。