Intro. to NLP - HW3 Report

> 學號：0812253
>
> 姓名：陳彥廷
>
> Kaggle user name：0812253

報告需求

- [x] ~~Describe all the methods you have implemented. (60%)~~
- [x] ~~Did you preprocess your data from the dataset? Why? And how?~~
  ~~(Did you encounter the problem that the input length is longer than the maximum sequence length of the model you use? How did you solve this problem?) (30%)~~
- [x] ~~What difficulties did you encounter in this assignment? How did you solve it? (10%)~~

目錄

概述

這次作業主要嘗試使用各種不同的 pretrained model 進行 fine-tuning 來達成目標。經過多次嘗試最後選擇使用自己在網路上找到的 `chinese-roberta-wwm-ext-large` 這個 pretrained model 實作。

這次作業首次嘗試使用 tensorflow 實作，在 M1 晶片上的體驗極差，即便使用 miniforge 開設 venv 還是有很多問題，因此最後由於時間因素將整個作業搬運至 Google Colab 上面運行。（早知道一開始就使用熟悉的 pytorch ==）

資料前處理

首先因為給定的資料是 json format，因此會利用下方的函數 `format_dataset` 將其轉為比較好處理的 dataframe。因為一篇文章可能會有多個問題，因此需要把文章多複製幾次來對應每一個問題。目標是將原先的 json format 處理成以下形式的 dataframe：

answerKey	article	question	options
最終的答案。會有 0, 1, 2, 3 四種選擇對應的就是 options 的 array index。	回答這個問題所需要的文章	問題	題目的選項，是一個長度為 4 的陣列，如果選項不夠的話會用空字串補滿。

實作程式碼如下：

```
def format_dataset(json_path):
    format_df = pd.DataFrame(columns=['answerKey', 'article', 'question', 'options'])

    with open(json_path, 'r') as f:
        data = json.load(f)

    not_test_dataset = False
    if 'answer' in data[0][1][0].keys():
        not_test_dataset = True

    for paragraph in data:
        for topic in paragraph[1]:
            options = []
```

```
            for choice in topic['choice']:
                options.append(choice)
            if len(options) < 4: # <- 1.
                for i in range(4 - len(options)):
                    options.append(' ')

            if not_test_dataset: # <- 2.
                format_df = format_df.append({
                    'answerKey': get_answer_key(topic['choice'], topic['answer']),
                    'article': paragraph[0][0],
                    'question': topic['question'],
                    'options': options,
                }, ignore_index=True)
            else:
                format_df = format_df.append({
                    'article': paragraph[0][0],
                    'question': topic['question'],
                    'options': options,
                }, ignore_index=True)
    format_df['answerKey'] = pd.to_numeric(format_df['answerKey']) # <- 3.
    return format_df
```

1. 這個部分主要是把不足 4 個選項的問題補上空字串符合 tokenize input dimension。
2. 這裡因為 test dataset 也需要做一樣的預處理，不過他不會有 answer，因此做了點特別處理。
3. 這裡因為處理完成的 answer key 的 datatype 會是 object 因此需要特別轉為 int type。

處理成一筆一筆的資料之後，接下來就是使用 BertTokenizer 進行 tokenize。至於這邊為何使用 BertTokenizer 而不是 RobertTokenizer ，因為我使用的是根據 transformer 來調用這個 pretrained mdoel，因此根據官方的說明必須使用 BERTTokenizer，如下所示：

使用Huggingface-Transformers

依托于🤗transformers库，可轻松调用以上模型。

```
tokenizer = BertTokenizer.from_pretrained("MODEL_NAME")
model = BertModel.from_pretrained("MODEL_NAME")
```

注意：本目录中的所有模型均使用BertTokenizer以及BertModel加载，请勿使用RobertaTokenizer/RobertaModel！

模型官方連結：

GitHub - ymcui/Chinese-BERT-wwm: Pre-Training with Whole Word Masking for Chinese BERT（中文BERT-wwm系列模型）

在自然语言处理领域中，预训练语言模型（Pre-trained Language Models）已成为非常重要的基础技术。为了进一步促进中文信息处理的研究发展，我们发布了基于全词掩码（Whole Word Masking）技术的中文预训练模型BERT-wwm，以及与此技术密切相关的模型：BERT-wwm-ext，RoBERTa-wwm-ext，RoBERTa-wwm-ext-large, RBT3, RBTL3等。 本项目基于谷歌官方BERT： https://github.com/google-research/bert

https://github.com/ymcui/Chinese-BERT-wwm

主要會先把 input string format 成下列格式之後匯入 tokenizer：

```
<s> article </s> question </s> choices[0] </s>
<s> article </s> question </s> choices[1] </s>
...
```

實作程式碼如下：

```
def preprocessor(dataset, tokenizer):
    all_input_ids = []
    all_attention_mask = []
```

```
    for i in range(len(dataset)):
        article = dataset['article'].iloc[i]
        question = dataset['question'].iloc[i]
        options = ast.literal_eval(str(dataset['options'].iloc[i])) # <- 1.
        choice_features = []

        for j in range(len(options)):
            option = options[j]
            # 2.
            input_string = '<s>' + ' ' + article + ' ' + '</s>' + ' ' + question + ' ' + '</s>' + ' ' + option + ' ' + '</s>'
            input_string = re.sub(r'\s+', ' ', input_string)

            input_ids = tokenizer(input_string,
                                  max_length=MAX_LEN,
                                  add_special_tokens=False)['input_ids']

            attention_mask = [1] * len(input_ids)

            padding_id = tokenizer.pad_token_id
            padding_length = 450 - len(input_ids)

            input_ids = input_ids + [padding_id]*padding_length
            attention_mask = attention_mask + [0]*padding_length

            assert len(input_ids) == MAX_LEN
            assert len(attention_mask) == MAX_LEN

            choice_features.append({'input_ids': input_ids,
                                    'attention_mask': attention_mask})

        all_input_ids.append(np.asarray([cf['input_ids'] for cf in choice_features], dtype='int32'))
        all_attention_mask.append(np.asarray([cf['attention_mask'] for cf in choice_features], dtype='int32'))

    return all_input_ids, all_attention_mask
```

1. 這邊要特別注意 `dataset['options'].iloc[i]` 出來的格式不會是 string 因此 ast 套件會報錯，因此要手動協助轉成 string。這邊的 `ast.literal_eval` 函數會將自己判斷將字串形式的 python 資料結構轉成原生的 python 資料結構。這裡的 choice 是陣列，所以會轉成 list。
2. 這個部分就是將 input string 轉換成固定形式準備要匯入 tokenizer 。

這個部分我們也連帶實作了 attention mask 的部分。

attention mask 用於指示哪些部分需要在模型中被關注。在進行預訓練的 transformer 模型中，attention mask 可以被用來指示哪些部分是有效的序列部分，哪些部分是 padding 部分。

在 attention mask 中，每個位置都有一個對應的值，1 表示該位置有效，0 表示該位置是 padding。在訓練或預測時，模型會忽略 attention mask 中的 0，只專注於 1。這樣可以確保模型只在有效輸入上進行預測，而不是在無效 padding 上。

模型實作

首先，對於 preprocessing 處理完成回傳的結果，我都會將其利用 np array 存儲以便待會使用。

```
dev_input_ids, dev_attention_mask = preprocessor(dataset=dev_data, tokenizer=tokenizer)
dev_input_ids = np.asarray(dev_input_ids, dtype='int32')
dev_attention_mask = np.asarray(dev_attention_mask, dtype='int32')
np.save(COLAB_FILE_PREFIX + 'model/race_dev_input_ids', dev_input_ids)
np.save(COLAB_FILE_PREFIX + 'model/race_dev_attention_mask', dev_attention_mask)
```

這邊 COLAB_FILE_PREFIX 是由於我在 Google Colab 上面訓練，因此需要存在 mounted 的 drive 裏面。

以下是模型訓練的 code，對於每一個部分會詳細說明：

```
# training
train_input_ids = np.load(COLAB_FILE_PREFIX + 'model/race_train_input_ids.npy')
train_attention_mask = np.load(COLAB_FILE_PREFIX + 'model/race_train_attention_mask.npy')
```

```
train_dict = {'input_ids': train_input_ids, 'attention_mask': train_attention_mask}
train_labels = train_data['answerKey']
viola = tf.data.Dataset.from_tensor_slices((train_dict, train_labels.values))

resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
tf.config.experimental_connect_to_cluster(resolver)
# This is the TPU initialization code that has to be at the beginning.
tf.tpu.experimental.initialize_tpu_system(resolver)
print("All devices: ", tf.config.list_logical_devices('TPU'))

strategy = tf.distribute.TPUStrategy(resolver)
viola = viola.shuffle(32).batch(8).cache().prefetch(tf.data.experimental.AUTOTUNE)

with strategy.scope():
    model = TFBertForMultipleChoice.from_pretrained('hfl/chinese-roberta-wwm-ext-large')
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
    model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

model.fit(viola, epochs=4)
model.save_pretrained(COLAB_FILE_PREFIX + 'model/roberta_model')
```

首先就是將剛剛存下來的 np object load 出來。

然後 configure 一下 TPU 的設置，這裡 Colab 好像有支援免費的 TPU 因此沒有像是在 local 端 training 會噴 warnining。

`strategy.scope()` 這行程式碼是在 TensorFlow 2.0 中支持分散式訓練的套件，其中 scope 用來定義執行分散式訓練的範圍。

- 這邊的 optimizer 使用的是 Adam
- Loss funtcion 則是普通的 cross entropy

訓練完成之後將模型儲存起來。

```
test_dict = {'input_ids': test_input_ids,
             'attention_mask': test_attention_mask}
pred = model.predict(test_dict)
ans = []
for log in pred[0]:
    ans.append(np.argmax(log) + 1)
df = pd.DataFrame({'index': [i for i in range(len(ans))], 'answer': ans})
df.to_csv(COLAB_FILE_PREFIX+'submission.csv', index=False)
```

最後由於 test dataset 也已經利用 preprocessing 處理完成了，可以直接包裝好利用 predict 函數會出結果。

這裡簡單做了一些處理使其符合要求的範例輸出，並且上傳 `submission.csv` 到 kaggle 中。
玉

有沒有遇到困難

在調整 tensorflow 的時候遇到非常劇烈的版本兼容問題。例外，也花了蠻多時間才找到適合中文的 pretrained model…