109403034 賴昀廷 BERT 的 Colab 連結

前情提要:

我沒有用 Colab 跑，是事後測完才丟上去雲端，因此截圖畫面有些會跟 colab 長的不太一樣

最高準確率:

這是用 ERNIE 的模型跑出來的結果，準確率為 95.58，ERNIE 的 Colab 連結

```
[32]: correct = 0
      for idx, pred in enumerate(res['pred']):
        if pred == res['label'][idx]:
          correct += 1
      print('test accuracy = %.4f'%(correct/len(test_df)))

      test accuracy = 0.9558
```

```
from datetime import datetime
parameters = {
    "num_class": 2,
    "time": str(datetime.now()).replace(" ", "_"),
    # Hyperparameters
    "model_name": 'ERNIE',
    "config": 'nghuyong/ernie-2.0-large-en',
    "dropout": 0.5,
    "learning_rate": 1e-5,
    "epochs": 3,
    "max_len": 512,
    "batch_size": 16,

}
```

使用的模型:

這次的作業總共使用了 BERT、ERNIE 以及 RoBERTa 三個模型(都有附上連結了)，其中效果最好的是 ERNIE，它後期都能穩定跑出 94、95 的準確率，反觀原本的 BERT，最後大部分都在 92~95 之間徘徊，而 RoBERTa 則是只測過一次 94 後就沒有再測試了。最一開始其實有試過 BigBird，但是它的訓練時間好

長，而且準確率慘不忍睹，只有 50 幾，所以試了兩三次後就果斷放棄跳槽了。

RoBERTa:

[15]:
```
from datetime import datetime
parameters = {
    "num_class": 2,
    "time": str(datetime.now()).replace(" ", "_"),
    # Hyperparameters
    "model_name": 'RoBERTa',
    "config": 'roberta-base',
    "dropout": 0.5, #0.1
    "learning_rate": 1e-5, #1e-3
    "epochs": 5, #3
    "max_len": 512,  #512
    "batch_size": 32,

}
```

[33]:
```
correct = 0
for idx, pred in enumerate(res['pred']):
  if pred == res['label'][idx]:
    correct += 1
print('test accuracy = %.4f'%(correct/len(test_df)))
```

```
test accuracy = 0.9406
```

一些參數跟 acc 的結果:

	acc	dropout	max_len	batch_size	learning_rate	epochs
RoBERTa	90.64	0.5	256	16	`3e-5	5
	92.8	0.4	256	16	`1e-5	5
	93.84	0.5	512	16	`1e-5	5
	94.06	0.5	512	32	`1e-5	5
ERNIE	92.78	0.3	512	16	`1e-5	3
	93.52	0.4	512	16	`1e-5	3
	94.32	0.5	512	16	`2e-5	3
	95.58	0.5	512	16	`1e-5	3

這個表格是我寫作業過程所調的超參數紀錄，從中挑出幾個放在這。一開始最心寒的就是完全不知道該從何調起，慢慢摸索後才知道 lr 在越後期要越調越小讓它慢慢收斂，另外我也發現每個模型能用的 batch size 也不太一樣，像是

RoBERTa 可以用到 32，但是 ERNIE 最高到 18 記憶體就會爆掉，但我沒有仔細去研究是不是跟模型的結構或是運算方式有關。

心得:

這次能夠用預訓練模型真的快樂很多，但是在一開始因為少加了 optimizer 進去，訓練出來的準確率一直在 50 打轉，卡了超久才被同學解救出來。另外一個有趣的點是可以額外挑自己有興趣的模型套進來試，雖然在一開始進到 Hugging Face 的頁面有點慌張，不知道要從何下手，研究怎麼套到程式碼中就花了很多時間，但最後成功的時候真的蠻有成就感的。在大致學會用法後應該會對專題幫助很大! 最累的部分應該就是一直調參數，有幾次甚至把 40000 筆的資料丟進去訓練，跑了 5 小時才跑完，但準確率不見起色，真的有夠懊惱的。

TODO 們

ToDo1

```python
# get predict result
def get_pred(logits):
  ##########
    probs = torch.softmax(logits, dim=1)
    preds = torch.argmax(probs, dim=1)
    return preds

  ##########
```

ToDo2

```python
# calculate confusion metrics
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
def cal_metrics(pred, ans,average):
  ##########
  # todo #
  ##########
    pred = pred.detach().cpu().numpy() # 將 tensor 轉為 numpy array
    ans = ans.detach().cpu().numpy()

    acc = accuracy_score(ans, pred)
    f1 = f1_score(ans, pred, average='macro')
    recall = recall_score(ans, pred, average='macro')
    precision = precision_score(ans, pred, average='macro')

    return acc, f1, recall, precision
```

ToDo3

[8]:

```python
import pandas as pd

all_data = [] # a list to save all data

##########
# todo #
##########

from sklearn.model_selection import train_test_split

# 載入資料集
dataset = load_dataset('imdb')

# 取出 train 和 test 資料
train_data = dataset['train']
test_data = dataset['test']

# 將資料轉換成 Pandas DataFrame
train_df = pd.DataFrame(train_data)
test_df = pd.DataFrame(test_data)

# 合併 train 和 test 資料
all_df = pd.concat([train_df, test_df], ignore_index=True)

# 重新進行分割
train_df, test_df = train_test_split(all_df, test_size=0.2, random_state=42)
```

ToDo4

```
# transform text to its number
def tokenize(self,input_text):
  ##########
  encoded_dict = self.tokenizer.encode_plus(
                        input_text,
                        add_special_tokens = True,
                        max_length = self.max_len,
                        padding='max_length',
                        truncation=True,
                        return_attention_mask = True,
                        return_token_type_ids = True,
                        return_tensors = 'pt',
                   )

  return encoded_dict['input_ids'].squeeze(), encoded_dict['attention_mask'].squeeze(), encoded_dict['token_type_ids'].squeeze()

  ##########
```

ToDo5

```
class BertClassifier(BertPreTrainedModel):
  def __init__(self, config, args):
    super(BertClassifier, self).__init__(config)
    self.bert = BertModel(config)

    ##########
    self.dropout = nn.Dropout(args['dropout'])
    self.classifier = nn.Linear(config.hidden_size, args['num_class'])
    ##########

    self.init_weights()

  # forward function, data in model will do this
  def forward(self, input_ids=None, attention_mask=None, token_type_ids=None, position_ids=None,
              head_mask=None, inputs_embeds=None, labels=None, output_attentions=None,
              output_hidden_states=None, return_dict=None):

    ##########
    outputs = self.bert(input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids,
                        position_ids=position_ids,
                        head_mask=head_mask,
                        inputs_embeds=inputs_embeds,
                        output_attentions=output_attentions,
                        output_hidden_states=output_hidden_states,
                        return_dict=return_dict)

    pooled_output = outputs[1]
    pooled_output = self.dropout(pooled_output)
    logits = self.classifier(pooled_output)

    return logits
```

ToDo6

```
for epoch in range(parameters["epochs"]):

    st_time = time.time()
    train_loss, train_acc, train_f1, train_rec, train_prec = 0.0, 0.0, 0.0, 0.0, 0.0
    step_count = 0

    ##########
    model.train()
    for data in train_loader:
      ids, masks, token_type_ids, labels = [t.to(device) for t in data]

      logits = model(input_ids = ids,
              token_type_ids = token_type_ids,
              attention_mask = masks)
      acc, f1, rec, prec = cal_metrics(get_pred(logits), labels, 'macro')
      loss = loss_fct(logits, labels)

      # backward pass
      optimizer.zero_grad()
      loss.backward()

      # update model parameters
      optimizer.step()

      train_loss += loss.item()
      train_acc += acc
      train_f1 += f1
      train_rec += rec
      train_prec += prec
      step_count+=1
    ##########
```

ToDo7

```
def predict_one(query, model):

  ##########
    model.eval()
    tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
    max_len = 128

    # Tokenize and encode the query
    encoded_query = tokenizer.encode_plus(
        query,
        max_length=max_len,
        add_special_tokens=True,
        return_token_type_ids=False,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt'
    )

    # Get the input ids and attention mask from the encoded query
    input_ids = encoded_query['input_ids'].to(device)
    attention_mask = encoded_query['attention_mask'].to(device)

    # Predict probabilities for each class
    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs[0]
        probs = F.softmax(logits, dim=0).squeeze().tolist()

    # Get the predicted class with the highest probability
    pred = torch.argmax(logits, dim=0).item()
  ##########
    return probs, pred
```