به نام خدا

عنوان:

گزارش پروژه اول تکلیف چهارم شبکه عصبی

استاد:

دکتر منصوری

دانشجو:

محمدعلی مجتهدسلیمانی-۴۰۳۳۹۰۴۵۰۴

تاریخ:

۱۴۰۳/۱۰/۲۱

Table of Content

اضافه کردن کتابخانه‌ها

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score, accuracy_score
from tqdm.notebook import tqdm
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix
import seaborn as sns
```

از کتابخانه **numpy** برای انجام عملیات هایی عددی با ماتریس ها و آرایه ها استفاده کرده ایم.

از **pandas** برای انجام عملیات های با **data frame** ها یا داده های جدولی مخصوصا برای تحلیل داده ها استفاده کرده ایم.

با استفاده از **matplotlib** نمودار هایی را که میخواهیم نمایش میدهیم.

کتابخانه **torch** و زیر مجموعه آن برای کار با شبکه های عصبی و مدل های آن مورد استفاده قرار میگیرد همچنین موارد **utils** را با استفاده از همین کتابخانه اضافه کرده ایم به عنوان مثال **dataloader** برای ساختن **dataset** ها و وارد کردن داده های خودمان در هر کدام از **batch**.

از **optim** برای استفاده از الگوریتم های بهینه سازی مانند **Adam** وارد کردیم.

از **BertTokenizer** برای توکن کردن متن برای مدل **BERT** استفاده کردیم.

از BertForSequenceClassification برای انجام طبقه بندی استفاده کرده ایم.

از AdamW که یک نوعی از بهینه ساز Adam است که معمولا برای مدل های Transformer استفاده میشود.

برا کار با فایل از os استفاده کرده ایم. از کتابخانه sklean نیز استفاده کرده ایم. برای محاسبه accuracy و f1-score به آن نیاز داریم. از tqdm نیز استفاده کردیم که عملکرد loop هارا بهبود میبخشد.

تنظیمات اولیه

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_name = 'bert-base-uncased'
num_epochs = 10
batch_size = 16
learning_rate = 2e-5
save_dir = './saved_models'
data_path = './IMDB.csv'
```

در این قسمت تعداد epoch و تعداد دسته ها را مشخص کردیم. نرخ یادگیری را نیز مشخص کردیم و مقدار مناسبی گذاشتیم. همچنین یک فایل به نام saved_models ساخته ایم چون اشتراک google colab بنده محدود بود و ممکن بود وسط روند آموزش قطع شود بعد از هر epoch مقادیر را در آن ذخیره میکند و در اجرای مجدد از آن استفاده میکنیم. IMDB همان دیتاست ما هست.

```
if not os.path.exists(save_dir):
    os.makedirs(save_dir)
```

مطمئن میشویم folderیی به نامی که در سری قبل بهش اشاره کردیم وجود دارد اگر نداشت آن را میسازیم.

تنظیم dataset

```
class IMDBDataset(Dataset):
    def __init__(self, reviews, labels, tokenizer, max_len):
        self.reviews = reviews
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.reviews)

    def __getitem__(self, item):
        review = str(self.reviews[item])
        label = self.labels[item]

        encoding = self.tokenizer.encode_plus(
            review,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
```

```
def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = IMDBDataset(
        reviews=df.review.to_numpy(),
        labels=df.sentiment.to_numpy(),
        tokenizer=tokenizer,
        max_len=max_len
    )
    return DataLoader(ds, batch_size=batch_size, num_workers=2)
```

کلاس IMDBDataset از Dataset که بالاتر وارد کرده بودیم ارث بری کرده است. همچنین تابع constructor آن را ساخته ایم تا یک مقدار دهی اولیه به متغیر ها بکند. Review آرایه ما برای کار با متن کاربران است. هر کدام از این متن ها طبیعتا label دارند. همچنین از tokenizer استفاده کردیم که یک object از transformers است استفاده کردیم تا کلمات را توکن بکنیم. از تابع __len__ برای تعداد نمونه ها استفاده کردیم تا مقدار review ها را بدست بیاوریم.

__getitem__ این تابع اصلی ما هست که میخواهیم یک نمونه را بر اساس اندیس آن پردازش بکنیم.

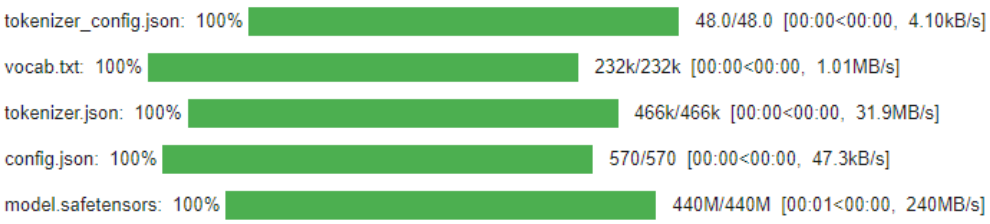
تابع create_data_loader یک object از Dataloader که در بالاتر وارد کرده بودیم میسازد که داده ها را از دیتاست درون دسته ها وارد میکند.

```
df = pd.read_csv(data_path)


if df['sentiment'].dtype == 'object':
    label_map = {'positive': 1, 'negative': 0}
    df['sentiment'] = df['sentiment'].map(label_map)
```

از مسیر مشخص شده فایل excel را میخوانیم و برای برچسب های ستون sentiment یک مقدار عددی در نظر میگیریم.

تفکیک داده آموزشی و تستی، توکن سازی، مشخص کردن optimizer

```
train_df = df.sample(frac=0.8, random_state=42)  # 80% for training
test_df = df.drop(train_df.index)


tokenizer = BertTokenizer.from_pretrained(model_name)
MAX_LEN = 512

train_data_loader = create_data_loader(train_df, tokenizer, MAX_LEN, batch_size)
test_data_loader = create_data_loader(test_df, tokenizer, MAX_LEN, batch_size)


model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)
model = model.to(device)


optimizer = AdamW(model.parameters(), lr=learning_rate)
loss_fn = nn.CrossEntropyLoss().to(device)
```



این قطعه کد داده ها را برای آموزش و تست آماده میکند. یک مدل BERT را مقدار دهی اولیه کردیم.

train_df = df.sample(): این قطعه کد dataframe اصلی را به مجموعه آموزشی تبدیل میکند. که به میزان ۸۰ درصد از داده ها است. برای تست هم همین شکل است.

سپس میخواهیم برای هر دو مجموعه آموزشی و تست توکن سازی کنیم. یعنی اینکه متن را به یک نمایشی از اعداد تبدیل میکنیم که مدل BERT بتواند آنها را درک بکند.

همچنین یک DataLoader برای مجموعه آموزشی و مجموعه تست ایجاد کردیم.

:model = BertForSequenceClassification.from_pretrained()

در این خط ما مدل از قبل آموزش دیده BERT را وارد میکنیم.

در ادامه optimizer و تابع loss را مشخص میکنیم.

روند آموزش و ارزیابی

```
# --- Training Loop ---
train_losses = []
test_losses = []
train_f1_scores = []
test_f1_scores = []
```

```
    # --- Evaluation ---
    model.eval()
    test_loss = 0
    test_correct = 0
    test_total = 0
    test_preds = []
    test_targets = []
```

```
test_f1 = f1_score(test_targets, test_preds)
test_f1_scores.append(test_f1)

print(f'Train Loss: {avg_train_loss:.4f}, Train F1: {train_f1:.4f}')
print(f'Test Loss: {avg_test_loss:.4f}, Test F1: {test_f1:.4f}')

torch.save(model.state_dict(), os.path.join(save_dir, f'model_epoch_{epoch+1}.pth'))
```

در این قطعه کد اصلی که روند یادگیری مدل ما هست مدل را آماده یادگیری میکنیم و پارامتر ها را میدهیم و در نهایت روند یادگیری را بررسی میکنیم به همراه گزارش accuracy و f1-score و loss.

torch.save(): با استفاده از این کد بعد از اتمام هر epoch مقادیر در فایلی که به آن اشاره کردیم ذخیره میکنیم تا بعدا بتوانیم استفاده بکنیم.

optimizer.zero_grad(): گرادیان پارامترهای مدل را صفر می کند. این بسیار مهم است زیرا گرادیان ها با هر epoch جمع می شوند و ما باید آنها را قبل از محاسبه گرادیان برای دسته فعلی پاک کنیم.

optimizer.step(): پارامترهای مدل را بر اساس گرادیان های محاسبه شده با استفاده از بهینه ساز (AdamW) به روز می کند.

بعد مدل را با model.eval روی حالت ارزیابی قرار میدهیم. ما در حین ارزیابی نیازی به گرادیان نداریم زیرا پارامترهای مدل را به روز نمی کنیم.

در ۱۰ دور یادگیری را بررسی کرده‌ایم:

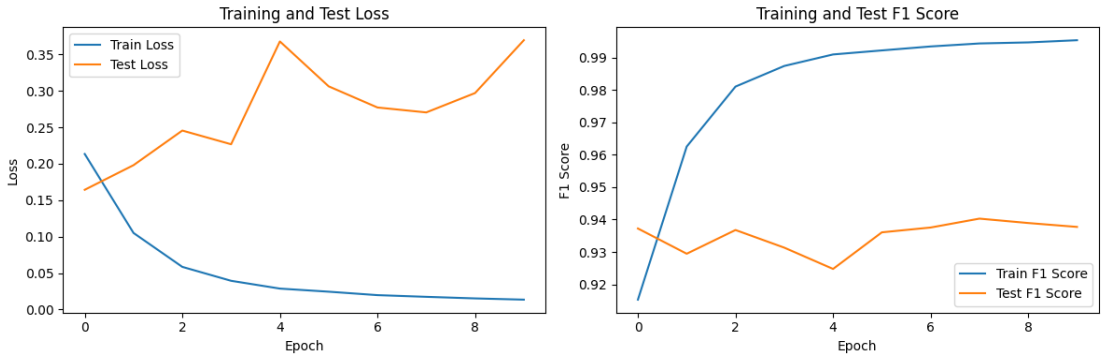
عکس ها در صفحه بعد قرار دارند.

```
Epoch 1/10
----------
Training: 100% 2500/2500 [12:52<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.08it/s]
Train Loss: 0.2135, Train F1: 0.9153
Test Loss: 0.1643, Test F1: 0.9372
Epoch 2/10
----------
Training: 100% 2500/2500 [12:50<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.09it/s]
Train Loss: 0.1051, Train F1: 0.9625
Test Loss: 0.1981, Test F1: 0.9294

Epoch 5/10
----------
Training: 100% 2500/2500 [12:50<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.06it/s]
Train Loss: 0.0287, Train F1: 0.9910
Test Loss: 0.3680, Test F1: 0.9248
Epoch 6/10
----------
Training: 100% 2500/2500 [12:50<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.07it/s]
Train Loss: 0.0244, Train F1: 0.9922
Test Loss: 0.3063, Test F1: 0.9361

Epoch 9/10
----------
Training: 100% 2500/2500 [12:50<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.12it/s]
Train Loss: 0.0152, Train F1: 0.9947
Test Loss: 0.2971, Test F1: 0.9389
Epoch 10/10
----------
Training: 100% 2500/2500 [12:50<00:00,  3.25it/s]
Testing: 100% 625/625 [01:02<00:00, 10.11it/s]
Train Loss: 0.0135, Train F1: 0.9954
Test Loss: 0.3697, Test F1: 0.9377
```

ادامه دور ها در فایل نوت بوک موجود هستند.

```
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_f1_scores, label='Train F1 Score')
plt.plot(test_f1_scores, label='Test F1 Score')
plt.xlabel('Epoch')
plt.ylabel('F1 Score')
plt.title('Training and Test F1 Score')
plt.legend()

plt.tight_layout()
plt.show()
```

نمودار f1-score و accuracy



گزارش نهایی f1-score و accuracy:

عکس در تصویر بعد قرار دارد.

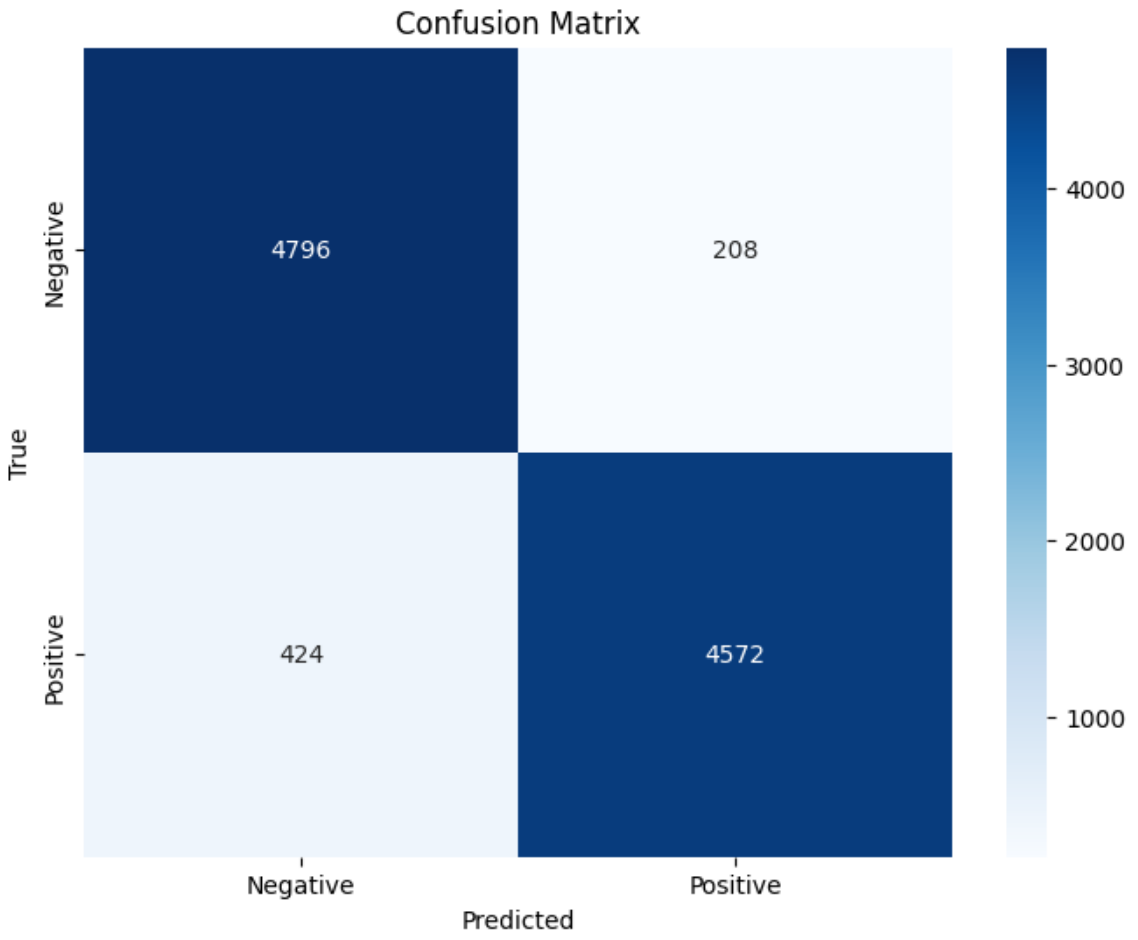
```
model.eval()
all_preds = []
all_targets = []

with torch.no_grad():
    for batch in test_data_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        _, predicted = torch.max(outputs.logits, 1)

        all_preds.extend(predicted.cpu().numpy())
        all_targets.extend(labels.cpu().numpy())


final_accuracy = accuracy_score(all_targets, all_preds)
print(f'Final Test Accuracy: {final_accuracy:.4f}')

final_f1 = f1_score(all_targets, all_preds)
print(f'Final Test F1 Score: {final_f1:.4f}')
```

```
Final Test Accuracy: 0.9368
Final Test F1 Score: 0.9354
```

برای محاسبه این مقدار باید مدل را در حالت ارزیابی قرار میدادیم.

نمایش ماتریس confusion matrix

عکس در تصویر بعد وجود دارد.



مثال عملی

در ادامه برای بررسی مدل دو جمله را به مدل داده ایم و میخواهیم بررسی کنیم که ببینیم آیا مدلی که آموزش داده ایم میتواند جملات را خوب درک کند و طبقه بکند یا خیر:

```
model_path = os.path.join(save_dir, 'model_epoch_10.pth')
model.load_state_dict(torch.load(model_path, map_location=device))
model.eval()
```

ابتدا مدل را به حالت ارزیابی میبریم دوباره و از پارامتر های موجود که در دور (epoch) دهم بدست آوردیم استفاده میکنیم تا ارزیابی را انجام بدهیم.

```
pred_sentences = ['worst movie of my life, will never watch movies from this series',
                  'Wow, blew my mind, what a movie by Marvel, animation and story is amazing']
```

۲ جمله مد نظر که یکی بار معنایی منفی دارد و دیگری مثبت.

```
encoded_inputs = tokenizer(pred_sentences, max_length=128, padding=True, truncation=True, return_tensors='p
input_ids = encoded_inputs['input_ids'].to(device)
attention_mask = encoded_inputs['attention_mask'].to(device)
```

```
with torch.no_grad():
    outputs = model(input_ids, attention_mask=attention_mask)
```

```
probabilities = torch.nn.functional.softmax(outputs.logits, dim=1)
predicted_labels = torch.argmax(probabilities, dim=1)
predicted_labels = predicted_labels.cpu().numpy()
```

```
label_map = {0: 'Negative', 1: 'Positive'}
predicted_class_names = [label_map[label] for label in predicted_labels]
```

جملات را توکن سازی کردیم و به عدد تبدیل کردیم. گرادیان را از بین بردیم زیرا دیگر نیازی به گرادیان نداریم و مدل در حالت ارزیابی است. برای label ها ۲ مقدار عددی در نظر گرفتیم.

```
for i, sentence in enumerate(pred_sentences):
    print(f"{sentence} : {predicted_class_names[i]}")
```

```
worst movie of my life, will never watch movies from this series : Negative
Wow, blew my mind, what a movie by Marvel, animation and story is amazing : Positive
```

نتیجه پیشبینی مدل که میبینیم مدل به درستی توانسته است بار معنایی جمله را درک بکند و به درستی تقسیم بندی بکند.