Photo credit: Pexels

Multi Class Text Classification With Deep Learning Using BERT

Natural Language Processing, NLP, Hugging Face

Most of the researchers submit their research papers to academic conference because its a faster way of making the results available. Finding and selecting a suitable conference has always been challenging especially for young researchers.

However, based on the previous conferences proceeding data, the researchers can increase their chances of paper acceptance and publication. We will try to solve this text classification problem with deep learning using BERT.

Almost all the code were taken from this tutorial, the only difference is the data.

The Data

The dataset contains 2,507 research paper titles, and have been manually classified into 5 categories (i.e. conferences) that can be downloaded from here.

Explore and Preprocess

```
import torch
from tqdm.notebook import tqdm

from transformers import BertTokenizer
from torch.utils.data import TensorDataset

from transformers import BertForSequenceClassification

df = pd.read_csv('data/title_conference.csv')
df.head()
```

conf_explore.py hosted with ❤ by GitHub view raw

conf_explore.py

	Title	Conference
0	Innovation in Database Management: Computer Sc...	VLDB
1	High performance prime field multiplication fo...	ISCAS
2	enchanted scissors: a scissor interface for su...	SIGGRAPH

Table 1

```
df['Conference'].value_counts()
```

```
ISCAS        864
INFOCOM      515
VLDB         423
WWW          379
SIGGRAPH     326
Name: Conference, dtype: int64
```

Figure 1

You may have noticed that our classes are imbalanced, and we will address this later on.

Encoding the Labels

```
possible_labels = df.Conference.unique()

label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index
label_dict
```

label_encoding.py hosted with ❤ by GitHub view raw

label_encoding.py

```
{'VLDB': 0, 'ISCAS': 1, 'SIGGRAPH': 2, 'INFOCOM': 3, 'WWW': 4}
```

```
df['label'] = df.Conference.replace(label_dict)
```

Train and Validation Split

Because the labels are imbalanced, we split the data set in a stratified fashion, using this as the class labels.

Our labels distribution will look like this after the split.

```python
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(df.index.values,
                                                  df.label.values,
                                                  test_size=0.15,
                                                  random_state=42,
                                                  stratify=df.label.values)

df['data_type'] = ['not_set']*df.shape[0]

df.loc[X_train, 'data_type'] = 'train'
df.loc[X_val, 'data_type'] = 'val'

df.groupby(['Conference', 'label', 'data_type']).count()
```

train_test_split.py hosted with ❤ by GitHub view raw

train_test_split.py

			Title
Conference	**label**	**data_type**	
INFOCOM	3	train	438
		val	77
ISCAS	1	train	734
		val	130
SIGGRAPH	2	train	277
		val	49
		train	359

		train	322
WWW	4		
		val	57

Figure 2

BertTokenizer and Encoding the Data

Tokenization is a process to take raw texts and split into tokens, which are numeric data to represent words.

- Constructs a BERT tokenizer. Based on WordPiece.
- Instantiate a pre-trained BERT model configuration to encode our data.
- To convert all the titles from text into encoded form, we use a function called `batch_encode_plus` , and we will proceed train and validation data separately.
- The 1st parameter inside the above function is the title text.
- `add_special_tokens=True` means the sequences will be encoded with the special tokens relative to their model.
- When batching sequences together, we set `return_attention_mask=True` , so it will return the attention mask according to the specific tokenizer defined by the `max_length` attribute.
- We also want to pad all the titles to certain maximum length.
- We actually do not need to set `max_length=256` , but just to play it safe.
- `return_tensors='pt'` to return PyTorch.
- And then we need to split the data into `input_ids` , `attention_masks` and `labels` .
- Finally, after we get encoded data set, we can create training data and validation data.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
```

```
    df[df.data_type=='train'].Title.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    return_tensors='pt'
)

encoded_data_val = tokenizer.batch_encode_plus(
    df[df.data_type=='val'].Title.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    return_tensors='pt'
)


input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(df[df.data_type=='train'].label.values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(df[df.data_type=='val'].label.values)

dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)
```

tokenizer_encoding.py hosted with ❤ by **GitHub** view raw

tokenizer_encoding.py

BERT Pre-trained Model

We are treating each title as its unique sequence, so one sequence will be classified to one of the five labels (i.e. conferences).

- `bert-base-uncased` is a smaller pre-trained model.
- Using `num_labels` to indicate the number of output labels.

- We also don't need `output_hidden_states`.

```
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

BERT_pretrained_model.py hosted with ❤ by GitHub view raw

BERT_pretrained_model.py

Data Loaders

- `DataLoader` combines a dataset and a sampler, and provides an iterable over the given dataset.
- We use `RandomSampler` for training and `SequentialSampler` for validation.
- Given the limited memory in my environment, I set `batch_size=3`.

```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

batch_size = 3

dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=batch_size)

dataloader_validation = DataLoader(dataset_val,
                                   sampler=SequentialSampler(dataset_val),
                                   batch_size=batch_size)
```

data_loaders.py hosted with ❤ by GitHub view raw

data_loaders.py

Optimizer & Scheduler

- To construct an optimizer, we have to give it an iterable containing the parameters to optimize. Then, we can specify optimizer-specific options such as the learning rate, epsilon, etc.

- Create a schedule with a learning rate that decreases linearly from the initial learning rate set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial learning rate set in the optimizer.

```
from transformers import AdamW, get_linear_schedule_with_warmup

optimizer = AdamW(model.parameters(),
                  lr=1e-5,
                  eps=1e-8)

epochs = 5

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=len(dataloader_train)*epc
```

optimizer_scheduler.py hosted with ❤ by GitHub view raw

optimizer_scheduler.py

Performance Metrics

We will use f1 score and accuracy per class as performance metrics.

```
from sklearn.metrics import f1_score

def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')

def accuracy_per_class(preds, labels):
    label_dict_inverse = {v: k for k, v in label_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'Class: {label_dict_inverse[label]}')
```

Get started Open in app

performance_metrics.py

Training Loop

```
import random

seed_val = 17
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in dataloader_val:

        batch = tuple(b.to(device) for b in batch)

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        with torch.no_grad():
            outputs = model(**inputs)

        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
        true_vals.append(label_ids)

```

```
    true_vals = np.concatenate(true_vals, axis=0)

    return loss_val_avg, predictions, true_vals

for epoch in tqdm(range(1, epochs+1)):

    model.train()

    loss_train_total = 0

    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False,
    for batch in progress_bar:

        model.zero_grad()

        batch = tuple(b.to(device) for b in batch)

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        outputs = model(**inputs)

        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        scheduler.step()

        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch


    torch.save(model.state_dict(), f'data_volume/finetuned_BERT_epoch_{epoch}.model')

    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')
```

```python
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score (Weighted): {val_f1}')
```


training_loop.py

```
Epoch 1
Training loss: 0.9007002512753849
Validation loss: 0.6143069127574563
F1 Score (Weighted): 0.7791319217695921


Epoch 2
Training loss: 0.5381144283001613
Validation loss: 0.6438471145765294
F1 Score (Weighted): 0.8207824902152685


Epoch 3
Training loss: 0.3589318487629241 7
Validation loss: 0.723008230609435
F1 Score (Weighted): 0.8463474188661483


Epoch 4
Training loss: 0.269252320019934 9
Validation loss: 0.7796335518272365
F1 Score (Weighted): 0.8341132163207956


Epoch 5
Training loss: 0.1815635446356576 6
Validation loss: 0.810808273508132 1
F1 Score (Weighted): 0.8441012614273822
```

Figure 3

```python
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)

model.to(device)

model.load_state_dict(torch.load('data_volume/finetuned_BERT_epoch_1.model', map_locatio

_, predictions, true_vals = evaluate(dataloader_validation)
accuracy_per_class(predictions, true_vals)
```

loading_evaluating.py hosted with ❤ by GitHub view raw

loading_evaluating.py

```
Class: VLDB
Accuracy: 45/64

Class: ISCAS
Accuracy: 124/130

Class: SIGGRAPH
Accuracy: 29/49

Class: INFOCOM
Accuracy: 65/77

Class: WWW
Accuracy: 33/57
```

Figure 4

Jupyter notebook can be found on Github. Enjoy the rest of the weekend!

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. Take a look.

Get this newsletter

NLP Nlp Tutorial Text Classification Document Classification Machine Learning