Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Для быстрого выполнения просмотрите [семинар].

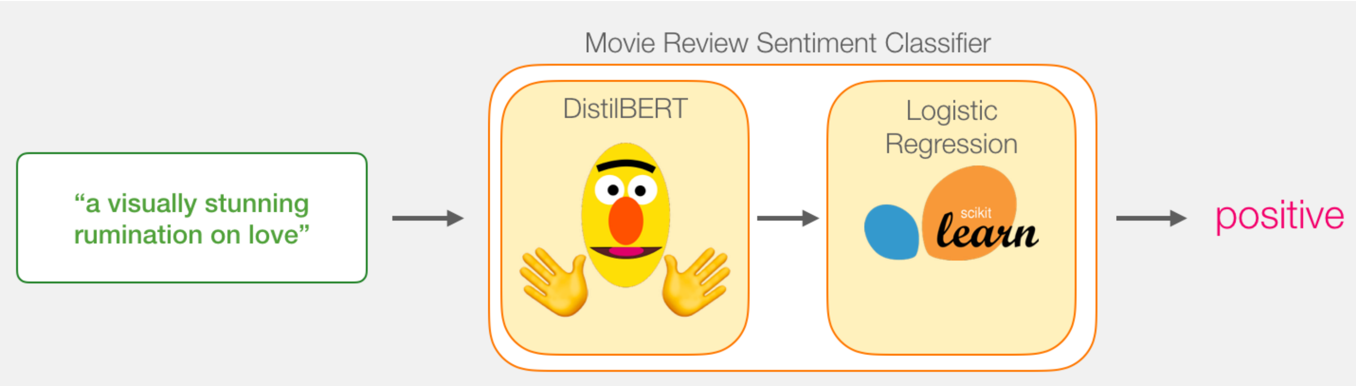
Models: Sentence Sentiment Classification

Our goal is to create a model that takes a sentence (just like the ones in our dataset) and produces either 1 (indicating the sentence carries a positive sentiment) or a 0 (indicating the sentence carries a negative sentiment). We can think of it as looking like this:

Under the hood, the model is actually made up of two model.

- DistilBERT processes the sentence and passes along some information it extracted from it on to the next model. DistilBERT is a smaller version of BERT developed and open sourced by the team at HuggingFace. It’s a lighter and faster version of BERT that roughly matches its performance.
- The next model, a basic Logistic Regression model from scikit learn will take in the result of DistilBERT’s processing, and classify the sentence as either positive or negative (1 or 0, respectively).

The data we pass between the two models is a vector of size 768. We can think of this of vector as an embedding for the sentence that we can use for classification.



Dataset

The dataset we will use in this example is [SST2](), which contains sentences from movie reviews, each labeled as either positive (has the value 1) or negative (has the value 0):

sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

Installing the transformers library

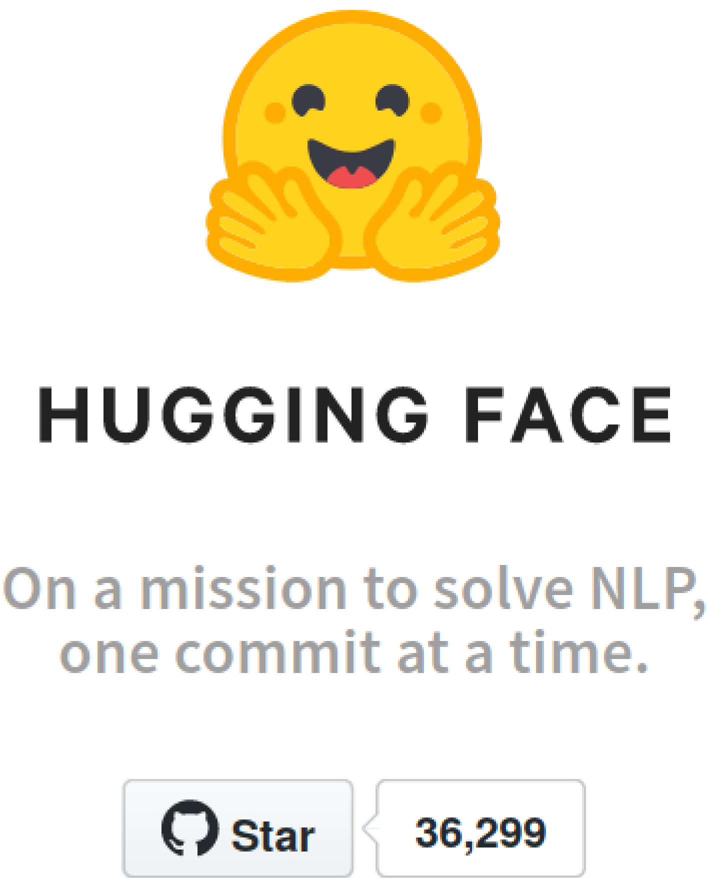
Let's start by installing the huggingface transformers library so we can load our deep learning NLP model

```
!pip install transformers
```

```
Collecting transformers
  Downloading https://files.pythonhosted.org/packages/d8/b2/57495b5309f09fa501866e22
     |████████████████████████████████| 2.1MB 7.4MB/s
Collecting tokenizers<0.11,>=0.10.1
  Downloading https://files.pythonhosted.org/packages/ae/04/5b870f26a858552025a62f16
     |████████████████████████████████| 3.3MB 54.5MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/lo
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (f
Collecting sacremoses
  Downloading https://files.pythonhosted.org/packages/75/ee/67241dc87f266093c533a2d4
     |████████████████████████████████| 901kB 69.3MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from s
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: tokenizers, sacremoses, transformers
Successfully installed sacremoses-0.0.45 tokenizers-0.10.2 transformers-4.5.1
```

[Transformers library doc](#)




```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset

```
df = pd.read_csv(
    'https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/tra
    delimiter='\t',
    header=None
)
print(df.shape)
df.head()
```

```
(6920, 2)
```

	0	1
0	a stirring , funny and finally transporting re...	1
1	apparently reassembled from the cutting room f...	0
2	they presume their audience wo n't sit still f...	0
3	this is a visually stunning rumination on love...	1
4	jonathan parker 's bartleby should have been t...	1

Using BERT for text classification.

Let's now load a pre-trained BERT model.

```
# For DistilBERT, Load pretrained model/tokenizer:

model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTok
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

Downloading: 100% 232k/232k [00:00<00:00, 855kB/s]

Downloading: 100% 28.0/28.0 [00:00<00:00, 46.6B/s]

Downloading: 100% 466k/466k [00:00<00:00, 1.31MB/s]

Downloading: 100% 442/442 [00:00<00:00, 947B/s]

Downloading: 100% 268M/268M [00:05<00:00, 51.8MB/s]

```
# look at the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
model.eval()
```

```
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)

        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
      )
      (3): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
      )
      (4): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
      )
      (5): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
```

```
            )
          )
        )
      )
```

```
from termcolor import colored

colors = ['red', 'green', 'blue', 'yellow']

def model_structure(layer, margin=0, item_color=0):
    for name, next_layer in layer.named_children():

        next = (0 if not list(next_layer.named_children()) else 1)
        print(colored(' ' * margin + name, colors[item_color]) + ':' * next)
        model_structure(next_layer, margin + len(name) + 2, (item_color + 1) % 4)

model_structure(model)
```

```
                                   out_lin
                          sa_layer_norm
                          ffn:
                               dropout
                               lin1
                               lin2
                          output_layer_norm
                       2:
                          attention:
                                     dropout
                                     q_lin
                                     k_lin
                                     v_lin
                                     out_lin
                          sa_layer_norm
                          ffn:
                               dropout
                               lin1
                               lin2
                          output_layer_norm
                       3:
                          attention:
                                     dropout
                                     q_lin
                                     k_lin
                                     v_lin
                                     out_lin
                          sa_layer_norm
                          ffn:
                               dropout
                               lin1
                               lin2
                          output_layer_norm
                       4:
                          attention:
                                     dropout
                                     q_lin
                                     k_lin
                                     v_lin
                                     out_lin
                          sa_layer_norm
                          ffn:
```

```
            ffn:
                dropout
                lin1
                lin2
            output_layer_norm
        5:
            attention:
                    dropout
                    q_lin
                    k_lin
                    v_lin
                    out_lin
            sa_layer_norm
            ffn:
                dropout
                lin1
                lin2
            output_layer_norm
```

Preparing the dataset

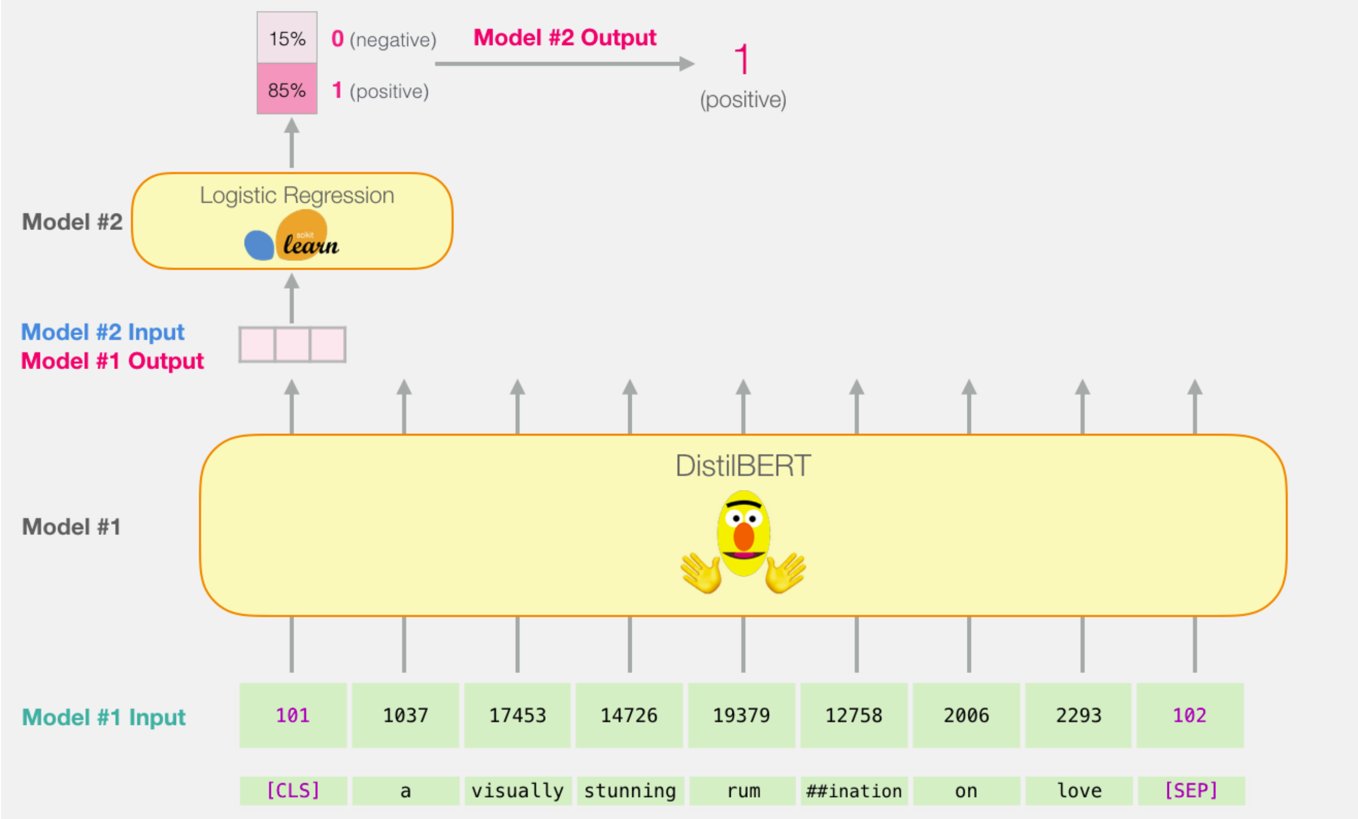
```
from torch.utils.data import Dataset, random_split

class ReviewsDataset(Dataset):
    def __init__(self, reviews: pd.Series, tokenizer, labels: pd.Series):
        self.labels = labels
        # tokenized reviews
        self.tokenized = reviews.apply(lambda x: tokenizer.encode(x, add_special_tokens=Tr

    def __getitem__(self, idx):
        return {"tokenized": self.tokenized[idx], "label": self.labels[idx]}

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

dataset = ReviewsDataset(df[0], tokenizer, df[1])

# DON'T CHANGE, PLEASE
train_size, val_size = int(.8 * len(dataset)), int(.1 * len(dataset))
torch.manual_seed(2)
train_data, valid_data, test_data = random_split(dataset, [train_size, val_size, len(datas

print(f"Number of training examples: {len(train_data)}")
print(f"Number of validation examples: {len(valid_data)}")
print(f"Number of testing examples: {len(test_data)}")
```

```
Number of training examples: 5536
Number of validation examples: 692
Number of testing examples: 692
```

```
from torch.utils.data import Sampler

class ReviewsSampler(Sampler):
    def __init__(self, subset, batch_size=32):
        self.batch_size = batch_size
```

```
        self.subset = subset

        self.indices = subset.indices
        # tokenized for our data
        self.tokenized = np.array(subset.dataset.tokenized)[self.indices]

    def __iter__(self):

        batch_idx = []
        # index in sorted data
        for index in np.argsort(list(map(len, self.tokenized))):
            batch_idx.append(index)
            if len(batch_idx) == self.batch_size:
                yield batch_idx
                batch_idx = []

        if len(batch_idx) > 0:
            yield batch_idx

    def __len__(self):
        return np.ceil(len(self.tokenized) / self.batch_size).astype(int)


from torch.utils.data import DataLoader

def get_padded(values):
    max_len = 0
    for value in values:
        if len(value) > max_len:
            max_len = len(value)

    padded = np.array([value + [0]*(max_len-len(value)) for value in values])

    return padded

def collate_fn(batch):

    inputs = []
    labels = []
    for elem in batch:
        inputs.append(elem['tokenized'])
        labels.append(elem['label'])

    inputs = get_padded(inputs) # padded inputs
    attention_mask = np.where(inputs != 0, 1, 0)

    return {"inputs": torch.tensor(inputs), "labels": torch.FloatTensor(labels), 'attentic

train_loader = DataLoader(train_data, batch_sampler=ReviewsSampler(train_data), collate_fr
valid_loader = DataLoader(valid_data, batch_sampler=ReviewsSampler(valid_data), collate_fr
test_loader = DataLoader(test_data, batch_sampler=ReviewsSampler(test_data), collate_fn=cc
```

Baseline




```
from tqdm.notebook import tqdm

def get_xy(loader):
    features = []
    labels = []

    with torch.no_grad():
        for batch in tqdm(loader):

            # don't forget about .to(device)
            # '''your code'''
            inputs = batch["inputs"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            batch_labels = batch["labels"]

            last_hidden_states = model(inputs, attention_mask=attention_mask)

            features.append(last_hidden_states[0].cpu())
            labels.append(batch_labels.cpu())

    features = torch.cat([elem[:, 0, :] for elem in features], dim=0).numpy()
    labels = torch.cat(labels, dim=0).numpy()

    return features, labels


train_features, train_labels = get_xy(train_loader)
valid_features, valid_labels = get_xy(valid_loader)
test_features, test_labels = get_xy(test_loader)
```

100% 173/173 [00:02<00:00, 63.63it/s]

100% 22/22 [00:00<00:00, 60.47it/s]

100% 22/22 [00:00<00:00, 60.43it/s]

```
lr_clf = LogisticRegression()
lr_clf.fit(train_features, train_labels)
lr_clf.score(test_features, test_labels)
```

```
0.8208092485549133
```

Fine-Tuning BERT

Define the model

```
from torch import nn

class BertClassifier(nn.Module):
    def __init__(self, pretrained_model, dropout=0.1):
        super().__init__()

        self.bert = pretrained_model
        self.dropout = nn.Dropout(p=dropout)
        self.relu = nn.ReLU()
        # '''your code'''
        self.fc1 = nn.Linear(in_features=pretrained_model.config.dim, out_features=1000)
        self.fc2 = nn.Linear(in_features=1000, out_features=1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, inputs, attention_mask):
        # '''your code'''
        last_hidden_states = model(inputs, attention_mask=attention_mask)
        x = self.relu(self.dropout(self.fc1(last_hidden_states[0][:, 0, :])))
        proba = self.sigmoid(self.fc2(x))

        # proba = [batch_size, ] - probability to be positive
        return proba
```

```
import torch.optim as optim

# DON'T CHANGE
model = model_class.from_pretrained(pretrained_weights).to(device)
bert_clf = BertClassifier(model).to(device)
# you can change
optimizer = optim.Adam(bert_clf.parameters(), lr=2e-5)
criterion = nn.BCELoss()
```

```
def train(model, iterator, optimizer, criterion, clip, train_history=None, valid_history=N
    model.train()
```

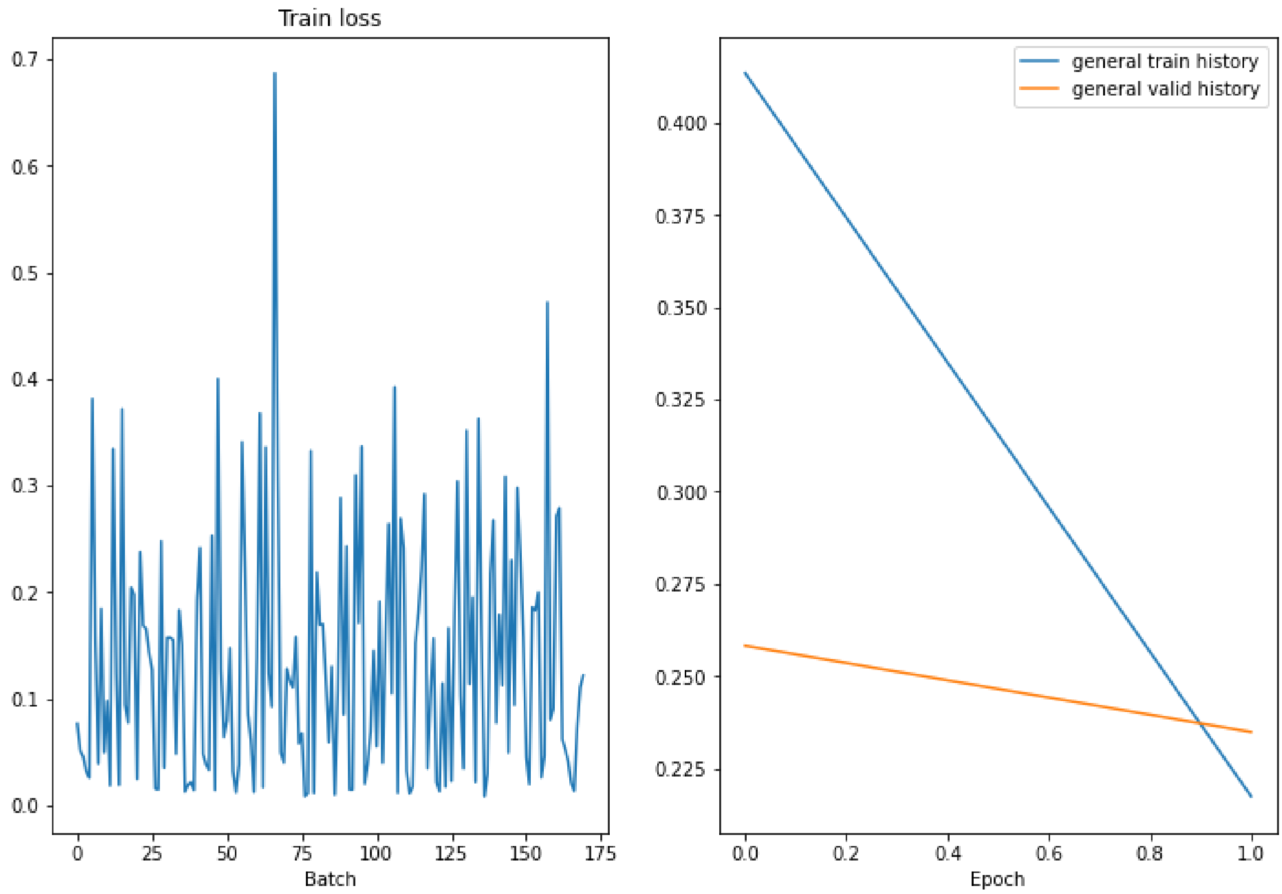
```
    model.train()

    epoch_loss = 0
    history = []
    for i, batch in enumerate(iterator):

        # don't forget about .to(device)
        # '''your code'''
        inputs = batch["inputs"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)

        optimizer.zero_grad()

        # '''your code'''
        output = model(inputs, attention_mask=attention_mask).squeeze(-1)
        # print("output.shape", output.shape, "\t", "labels.shape", labels.shape)

        loss = criterion(output, labels)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        epoch_loss += loss.item()

        history.append(loss.cpu().data.numpy())
        if (i+1)%10==0:
            fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))

            clear_output(True)
            ax[0].plot(history, label='train loss')
            ax[0].set_xlabel('Batch')
            ax[0].set_title('Train loss')
            if train_history is not None:
                ax[1].plot(train_history, label='general train history')
                ax[1].set_xlabel('Epoch')
            if valid_history is not None:
                ax[1].plot(valid_history, label='general valid history')
            plt.legend()

            plt.show()


    return epoch_loss / (i + 1)

def evaluate(model, iterator, criterion):

    model.eval()

    epoch_loss = 0

    history = []

    with torch.no_grad():

        for i, batch in enumerate(iterator):
```

```
        for i, batch in enumerate(iterator):

            # '''your code'''
            inputs = batch["inputs"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)

            output = model(inputs, attention_mask=attention_mask).squeeze(-1)

            loss = criterion(output, labels)

            epoch_loss += loss.item()

    return epoch_loss / (i + 1)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

```
import time
import math
import matplotlib
matplotlib.rcParams.update({'figure.figsize': (16, 12), 'font.size': 14})
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import clear_output
```

```
train_history = []
valid_history = []

N_EPOCHS = 3
CLIP = 1

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(bert_clf, train_loader, optimizer, criterion, CLIP, train_history,
    valid_loss = evaluate(bert_clf, valid_loader, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(bert_clf.state_dict(), 'best-val-model.pt')

    train_history.append(train_loss)
    valid_history.append(valid_loss)
```

```
    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}')
```




```
Epoch: 03 | Time: 0m 16s
	Train Loss: 0.134 | Train PPL:   1.143
	 Val. Loss: 0.281 |  Val. PPL:   1.325
```

```
best_model = BertClassifier(model).to(device)
best_model.load_state_dict(torch.load('best-val-model.pt'))

pred_labels = []
true_labels = []

best_model.eval()
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_loader), total=len(test_loader)):
        # '''your code'''
        inputs = batch["inputs"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].cpu()

        true_labels.append(labels.numpy())

        # '''your code'''
        output = best_model(inputs, attention_mask=attention_mask).squeeze(-1)
        pred_label = (output > 0.5).cpu().numpy().astype(int)
        pred_labels.append(pred_label)
```

100% 22/22 [00:00<00:00, 53.68it/s]

```
from sklearn.metrics import accuracy_score

true_labels = np.concatenate(true_labels, axis=0)
pred_labels = np.concatenate(pred_labels, axis=0)
accuracy_score(true_labels, pred_labels)
```

```
0.880057803468208
```

```
assert accuracy_score(true_labels, pred_labels) >= 0.86
```

Finetuned model from **HUGGING FACE**

[BertForSequenceClassification]

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# we have the same tokenizer
# new_tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased-finetuned-sst-2-e
new_model = AutoModelForSequenceClassification.from_pretrained("distilbert-base-uncased-fi
```

Downloading: 100% 629/629 [00:00<00:00, 25.3kB/s]

Downloading: 100% 268M/268M [00:04<00:00, 59.0MB/s]

```
pred_labels = []
true_labels = []

new_model.eval()
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_loader), total=len(test_loader)):
        # '''your code'''
        inputs = batch["inputs"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].cpu()

        true_labels.append(labels.numpy())

        # '''your code'''
        logits = new_model(inputs, attention_mask=attention_mask).logits
        pred_label = logits.argmax(1)
        pred_labels.append(pred_label.cpu())

true_labels = np.concatenate(true_labels, axis=0)
pred_labels = np.concatenate(pred_labels, axis=0)
accuracy_score(true_labels, pred_labels)
```

100% 22/22 [00:00<00:00, 60.27it/s]

```
0.9841040462427746
```

```
model_structure(new_model)
```

```
distilbert:
            embeddings:
                        word_embeddings
                        position_embeddings
                        LayerNorm
                        dropout
            transformer:
                         layer:
                               0:
                                  attention:
                                            dropout
                                            q_lin
                                            k_lin
                                            v_lin
                                            out_lin
                                  sa_layer_norm
                                  ffn:
                                       dropout
                                       lin1
                                       lin2
                                  output_layer_norm
                               1:
                                  attention:
                                            dropout
                                            q_lin
                                            k_lin
                                            v_lin
                                            out_lin
                                  sa_layer_norm
                                  ffn:
                                       dropout
                                       lin1
                                       lin2
                                  output_layer_norm
                               2:
                                  attention:
                                            dropout
                                            q_lin
                                            k_lin
                                            v_lin
                                            out_lin
                                  sa_layer_norm
                                  ffn:
                                       dropout
                                       lin1
                                       lin2
                                  output_layer_norm
                               3:
                                  attention:
                                            dropout
                                            q_lin
                                            k_lin
                                            v_lin
```

```
                    out_lin
            sa_layer_norm
            ffn:
                dropout
                lin1
                lin2
```

Напишите вывод о своих результатах. В выводы включите ваши гиперпараметры.

Качество с помощью Fine-Tuning должно достигать 0.86.

- **Bert** очень удобно и эффективно делает эмбединги текстов, которые можно классифицировать даже простой логистической регрессией. Получается **accuracy = 0.82**.
- Если же сделать **fine-tuning** и немного усложнить выходные классифицирующие слои, то результат еще лучше. Здесь был добавлен (кроме выходного) еще один внутренний линейный слой на **1000 нейронов** с **relu** и **dropout=0.1**. Обучалось **3 эпохи** с **lr=2e-5**. В итоге получилось **accuracy = 0.88**.
- Специализированный же класс и вовсе дает **accuracy = 0.984**

Очень удобная библиотека!

0 сек.   выполнено в 10:34