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

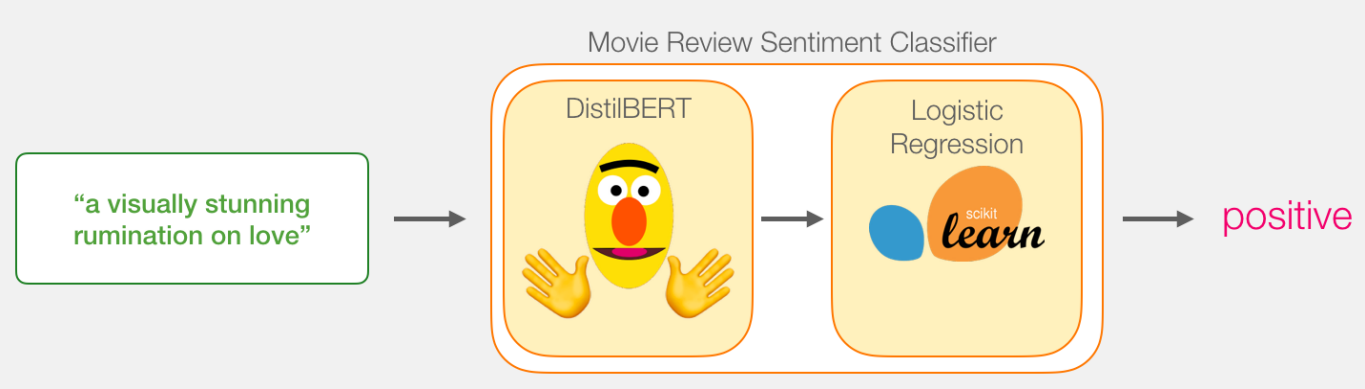
Under the hood, the model is actually made up of two model.

- 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

ep learning

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
1 !pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.12.5-py3-none-any.whl (3.1 MB)
     |████████████████████████████████| 3.1 MB 5.2 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
     |████████████████████████████████| 895 kB 27.0 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (fr
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-pac
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.mar
```

```
     |████████████████████████████████| 3.3 MB 36.5 MB/s
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packa
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.1.2-py3-none-any.whl (59 kB)
     |████████████████████████████████| 59 kB 6.5 MB/s
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_
     |████████████████████████████████| 596 kB 39.9 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7
Requirement already satisfied: pyparsing<3,>=2.0.2 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sa
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: pyyaml, tokenizers, sacremoses, huggingface-hub, trans
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.1.2 pyyaml-6.0 sacremoses-0.0.46 tokenizers-
```

[Transformers library doc]

HUGGING FACE

On a mission to solve NLP,
one commit at a time.


```
1 import numpy as np
2 import pandas as pd
3
4 from sklearn.model_selection import train_test_split
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.model_selection import GridSearchCV
7 from sklearn.model_selection import cross_val_score
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
10 from torch import nn
11 import torch.nn.functional as F
12
13 import transformers as ppb
14
15 import warnings
16 warnings.filterwarnings('ignore')
```

Importing the dataset

```
1 df = pd.read_csv(
2     'https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/
3     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.DistilBert
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

```
Downloading: 100%  226k/226k [00:00<00:00, 754kB/s]
Downloading: 100%  28.0/28.0 [00:00<00:00, 505B/s]
Downloading: 100%  455k/455k [00:00<00:00, 1.02MB/s]
Downloading: 100%  483/483 [00:00<00:00, 9.00kB/s]
Downloading: 100%  256M/256M [00:09<00:00, 26.9MB/s]
```



```
t used when ir
ckpoint of a m
checkpoint of
```

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

```
      (lin2): Linear(in_features=3072, out_features=768, bias=True)
    )
    (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  )
  (2): 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)
```

```
      (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)
  )
  (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)
    )
```



```
    (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)
```

```
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)
```

```
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:
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
   output_layer_norm
4:
   attention:
              dropout
              q_lin
              k_lin
              v_lin
              out_lin
   sa_layer_norm
   ffn:
        dropout
        lin1
        lin2
   output_layer_norm
5:
   attention:
              dropout
```

```
                q_lin
                k_lin
                v_lin
                out_lin
```

Preparing the dataset

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

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

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

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

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

# 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(da

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
```



```
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 = []
```

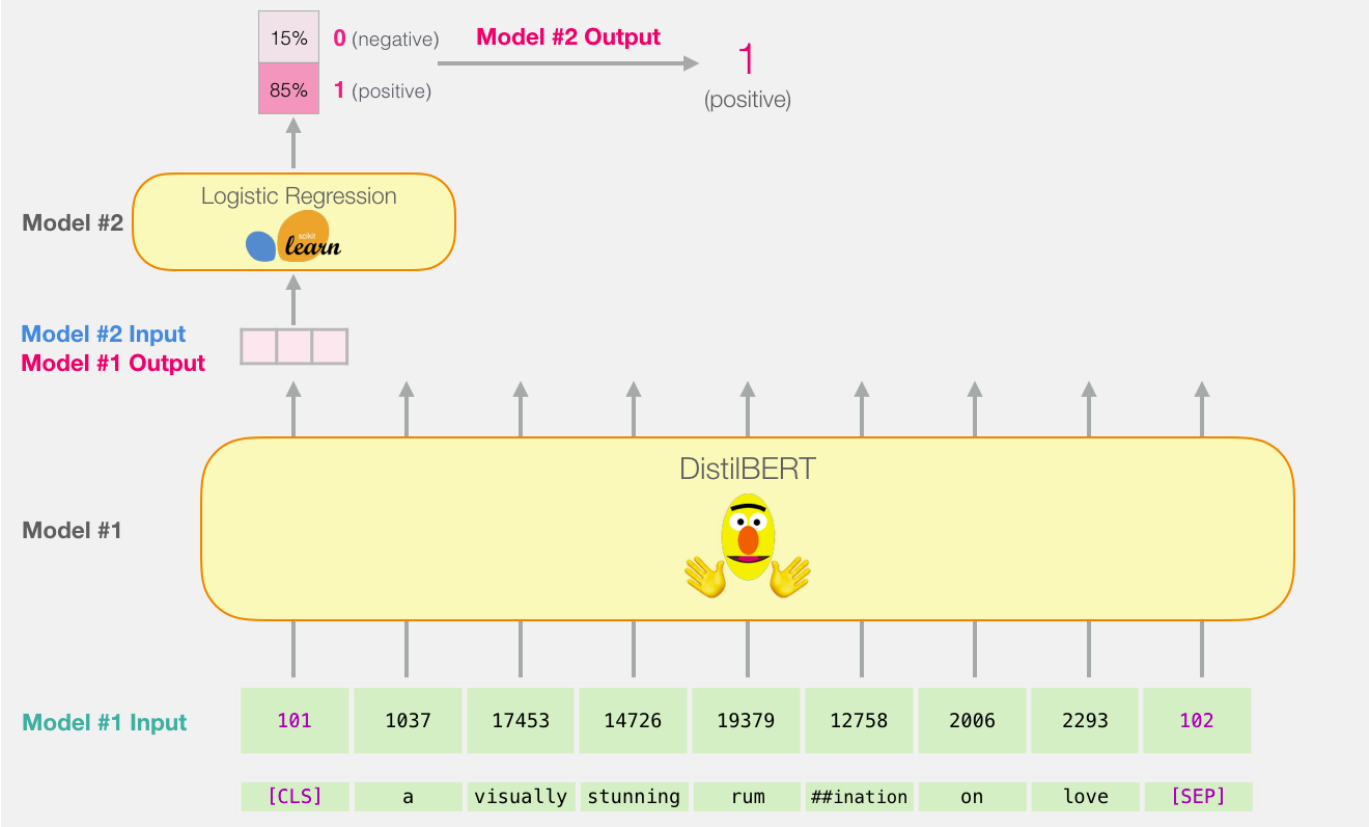
```
        # index in sorted data
        for index in np.argsort(list(map(len, self.tokenized))):
            batch_idx.append(index)
            if len(batch_idx) == self.batch_size:
                yield batch_idx
                batch_idx = []

        if len(batch_idx) > 0:
            yield batch_idx

    def __len__(self):
        return len(self.dataset)
```

```
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),
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

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

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)
            batch_inputs = batch['inputs'].to(device)
            batch_attention_mask = batch['attention_mask'].to(device)
            # no need to put on GPU for now
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
            # forward
            last_hidden_states = model(input_ids=batch_inputs,
                                       attention_mask=batch_attention_mask)

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

    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)
```

```
173/? [00:08<00:00, 11.52it/s]

22/? [00:01<00:00, 16.76it/s]

22/? [00:01<00:00, 15.85it/s]
```

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

```
0.815028901734104
```

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()

        self.out = nn.Linear(in_features=768, out_features=1)

    def forward(self, inputs, attention_mask):
        last_hidden_states = model(input_ids=inputs, attention_mask=attention_mask)[0]
        # [batch size, max seq len, bert hidden dim]

        features = last_hidden_states[:, 0, :] # [batch_size, bert hidden dim]

        proba = proba.view(-1) # [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.BCEWithLogitsLoss()
```

```
Some weights of the model checkpoint at distilbert-base-uncased were not used when ir
```

```
- This IS expected if you are initializing DistilBertModel from the checkpoint of a m
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of
```

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

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

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


        optimizer.zero_grad()

        # forward + backward + optimize
        output = model(inputs, attention_mask)
        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()

        # # for debugging
        # break


    return epoch_loss / (i + 1)

def evaluate(model, iterator, criterion):

    model.eval()
```

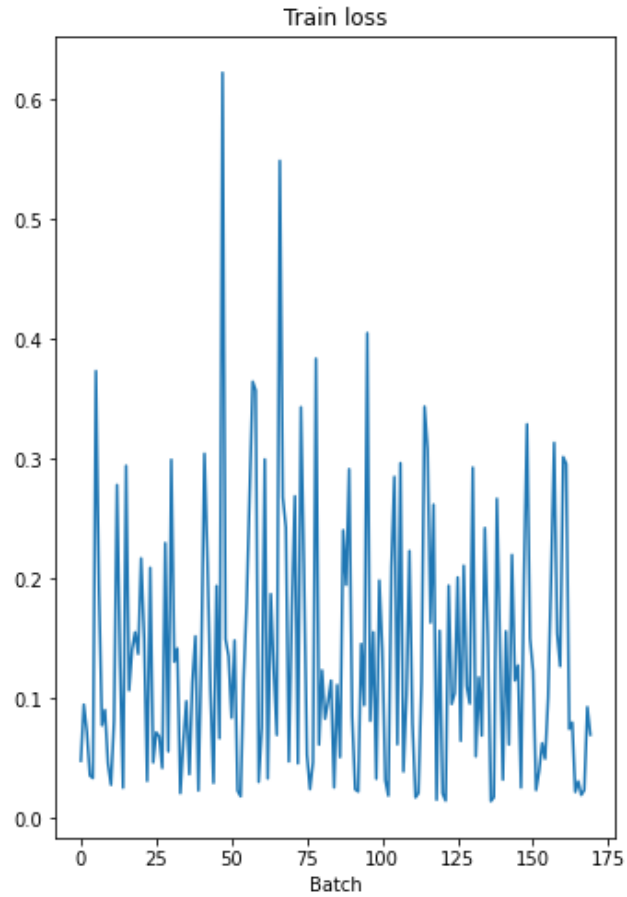
```
    epoch_loss = 0

    history = []

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            inputs = batch['inputs'].to(device)
            labels = batch['labels'].to(device)
            attention_mask = batch['attention_mask'].to(device)

            output = model(inputs, attention_mask)
            loss = criterion(output, labels)

            epoch_loss += loss.item()

            # # for debugging
            # break

    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
```

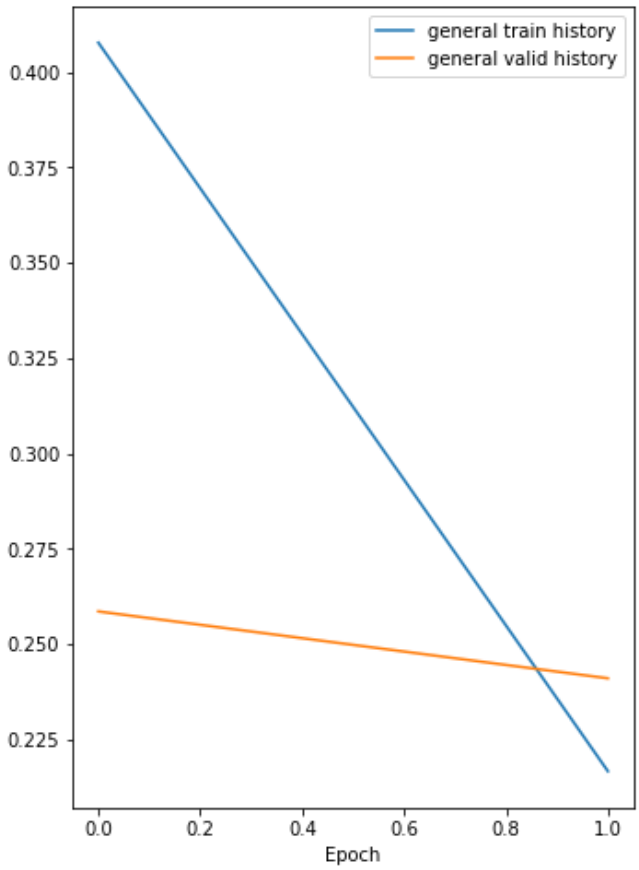
```
import time
import math
import matplotlib
matplotlib.rcParams.update({'figure.figsize': (16, 12), 'font.size': 14})
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import clear_output
```



```

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_histor
    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 38s
        Train Loss: 0.136 | Train PPL:   1.145
```



```
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)):

        inputs = batch['inputs'].to(device)
        attention_mask = batch['attention_mask'].to(device)
```

```
        labels = batch['labels'] # used with sklearn later => stays on cpu

        output = best_model(inputs, attention_mask)
        binary_output = (torch.sigmoid(output) > 0.5).to(torch.int)

        true_labels.append(labels.numpy())
        pred_labels.append(binary_output.cpu().numpy())
```

22/? [00:01<00:00, 13.91it/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)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-6f540048575b> in <module>()
      1 from sklearn.metrics import accuracy_score
      2
----> 3 true_labels = np.concatenate(true_labels, axis=0)
      4 pred_labels = np.concatenate(pred_labels, axis=0)
      5 accuracy_score(true_labels, pred_labels)

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: zero-dimensional arrays cannot be concatenated
```

SEARCH STACK OVERFLOW

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

Finetuned model from **HUGGING FACE**

BertForSequenceClassification

Automatic saving failed. This file was updated remotely or in another tab. Show diff

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

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

Downloading: 100% 629/629 [00:00<00:00, 16.1kB/s]

Downloading: 100% 255M/255M [00:09<00:00, 31.2MB/s]

```
pred_labels = []
true_labels = []

new_model.eval()
with torch.no_grad():
```

```
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_loader)):

        inputs = batch['inputs'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'] # used with sklearn later => stays on cpu

        output = best_model(inputs, attention_mask)
        binary_output = (torch.sigmoid(output) > 0.5).to(torch.int)

        true_labels.append(labels.numpy())
        pred_labels.append(binary_output.cpu().numpy())

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

```
22/? [00:01<00:00, 13.60it/s]
```

```
0.869942196531792
```

```
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:
                                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. Все происходит крайне быстро, в отличие от gpt. Качество предобученной модели и baseline не сильно отличается.

```
1
```