Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Для быстрого выполнения просмотрите [семинар (https://drive.google.com/file/d/1w_rTEWXQ_SA4YPXFjpkM0aU51bDgWLyI/view?usp=sharing)](https://drive.google.com/file/d/1w_rTEWXQ_SA4YPXFjpkM0aU51bDgWLyI/view?usp=sharing).

Models: Sentence Sentiment Classification

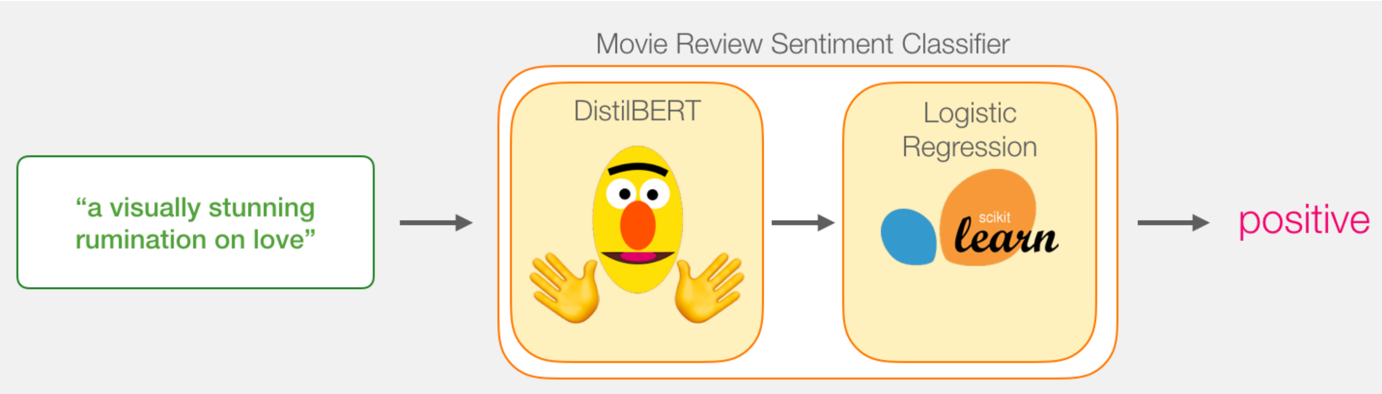
Our goal is to create a model that takes a sentence (just like the ones in our dataset) and produces either 1 (indicating the sentence carries a positive sentiment) or a 0 (indicating the sentence carries a negative sentiment). We can think of it as looking like this:

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 (https://nlp.stanford.edu/sentiment/index.html), 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.

Ввод [1]:

```
!pip install transformers
```

executed in 11ms, finished 15:58:52 2021-11-13

Transformers library doc (https://huggingface.co/transformers/)

HUGGING FACE

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

Star 36,299

Ввод [2]:

```
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')
```

executed in 1.94s, finished 15:58:54 2021-11-13

Importing the dataset

Ввод [3]:

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

executed in 1.01s, finished 15:58:55 2021-11-13

```
(6920, 2)
```

Out[3]:

	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.

Ввод [4]:

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

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

executed in 9.58s, finished 15:59:04 2021-11-13

```
Some weights of the model checkpoint at distilbert-base-uncased were not use
d when initializing DistilBertModel: ['vocab_projector.weight', 'vocab_trans
form.weight', 'vocab_layer_norm.bias', 'vocab_transform.bias', 'vocab_projec
tor.bias', 'vocab_layer_norm.weight']
- This IS expected if you are initializing DistilBertModel from the checkpoi
nt of a model trained on another task or with another architecture (e.g. ini
tializing a BertForSequenceClassification model from a BertForPreTraining mo
del).
- This IS NOT expected if you are initializing DistilBertModel from the chec
kpoint of a model that you expect to be exactly identical (initializing a Be
rtForSequenceClassification model from a BertForSequenceClassification mode
l).
```

Ввод [5]:

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

executed in 3.00s, finished 15:59:07 2021-11-13

Out[5]:

```
DistilBertModel(
  (embeddings): Embeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer): Transformer(
    (layer): ModuleList(
      (0): 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=T
```

Ввод [6]:

```
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)
```

executed in 45ms, finished 15:59:07 2021-11-13

```
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
        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
        sa_layer_norm
        ffn:
              dropout
              lin1
              lin2
        output_layer_norm
```

Preparing the dataset

Ввод [7]:

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

class ReviewsDataset(Dataset):
    def __init__(self, reviews, tokenizer, labels):
        self.labels = labels
        # tokenized reviews
        #self.tokenized = (tokenizer.tokenize(x) for x in 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(datase

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)}")
```

executed in 4.98s, finished 15:59:12 2021-11-13

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

Ввод [8]:

```
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)
```

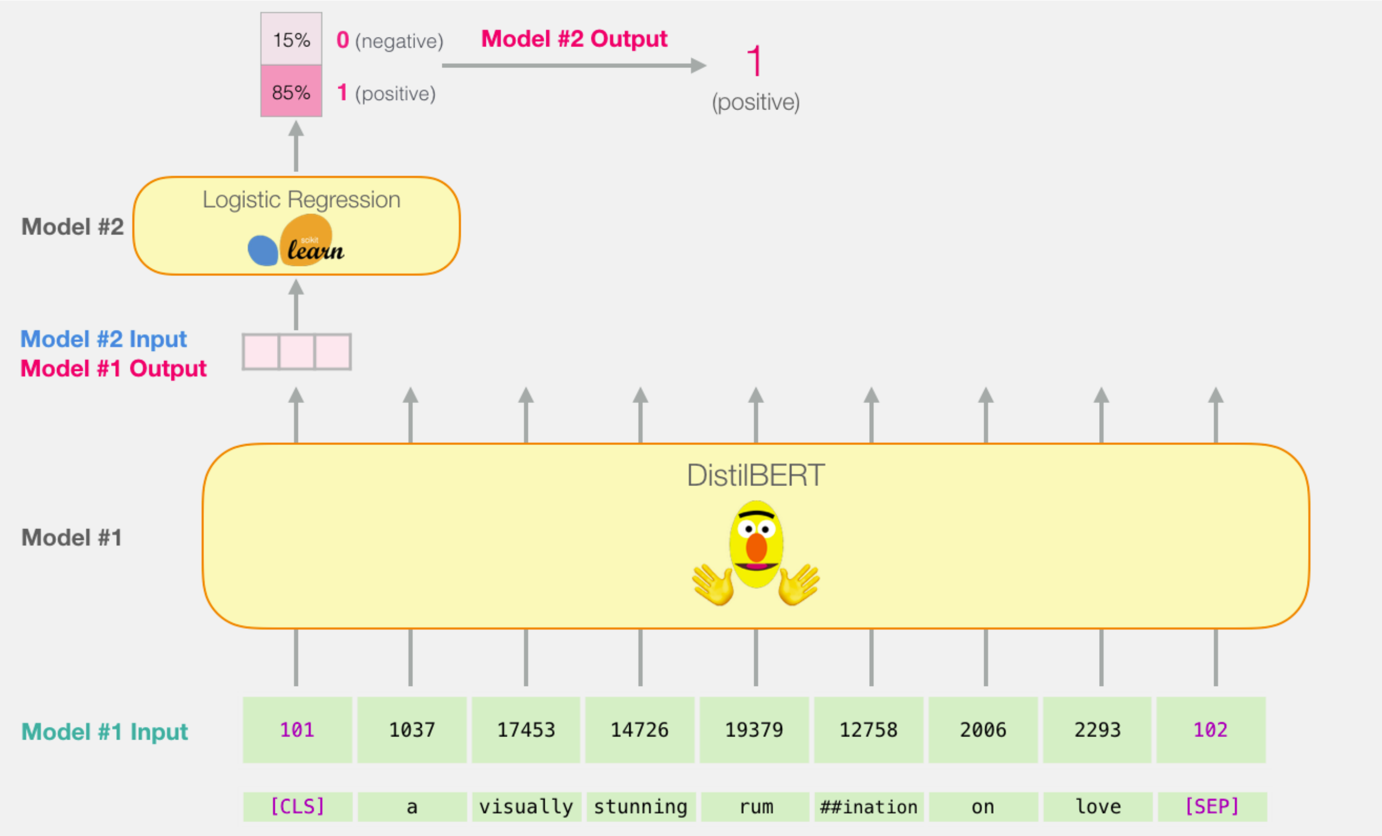
executed in 14ms, finished 15:59:12 2021-11-13

Ввод [9]:

```
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.LongTensor(inputs), "labels": torch.FloatTensor(labels), 'atten

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

executed in 14ms, finished 15:59:13 2021-11-13

Baseline



Ввод [10]:

```
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)
            input_data=batch['inputs'].to(device)
            mask=batch['attention_mask'].to(device)
            last_hidden_states=model(input_data,mask)
            labels.append(batch['labels'])
            features.append(last_hidden_states[0].cpu())

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

    return features, labels
```

executed in 14ms, finished 15:59:13 2021-11-13

Ввод [11]:

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

executed in 4.96s, finished 15:59:17 2021-11-13

```
0it [00:00, ?it/s]

0it [00:00, ?it/s]

0it [00:00, ?it/s]
```

Ввод [12]:

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

executed in 490ms, finished 15:59:18 2021-11-13

Out[12]:

```
0.8179190751445087
```

Fine-Tuning BERT

Define the model

Ввод [106]:

```
from torch import nn

class BertClassifier(nn.Module):
    def __init__(self, pretrained_model, dropout=0.2):
        super().__init__()

        self.bert = pretrained_model
        self.dropout = nn.Dropout(p=dropout)
        self.relu = nn.ReLU()

        self.clf_1=nn.Linear(768,64)
        self.clf_2=nn.Linear(64,1)

        self.softmax = nn.Sigmoid()

    def forward(self, inputs, attention_mask):

        outputs=self.bert(inputs,attention_mask=attention_mask)

        x=self.clf_1(outputs[0][:,0,:])
        x=self.dropout(self.relu(x))
        x=self.dropout(self.clf_2(x))
        proba=self.softmax(x)

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

executed in 9ms, finished 17:14:37 2021-11-13

Ввод [107]:

```
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()
```

executed in 2.32s, finished 17:14:39 2021-11-13

```
Some weights of the model checkpoint at distilbert-base-uncased were not use
d when initializing DistilBertModel: ['vocab_projector.weight', 'vocab_trans
form.weight', 'vocab_layer_norm.bias', 'vocab_transform.bias', 'vocab_projec
tor.bias', 'vocab_layer_norm.weight']
- This IS expected if you are initializing DistilBertModel from the checkpoi
nt of a model trained on another task or with another architecture (e.g. ini
tializing a BertForSequenceClassification model from a BertForPreTraining mo
del).
- This IS NOT expected if you are initializing DistilBertModel from the chec
kpoint of a model that you expect to be exactly identical (initializing a Be
rtForSequenceClassification model from a BertForSequenceClassification mode
l).
```

Ввод [108]:

```
def train(model, iterator, optimizer, criterion, clip, train_history=None, valid_history=No
    model.train()

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

        # don't forget about .to(device)
        optimizer.zero_grad()
        input_data=batch['inputs'].to(device)
        mask=batch['attention_mask'].to(device)
        labels=batch['labels'].to(device)
        output=model(input_data,mask).squeeze(1)
        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):

            input_data=batch['inputs'].to(device)
            mask=batch['attention_mask'].to(device)
            labels=batch['labels'].to(device)
            output=model(input_data,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
```

executed in 27ms, finished 17:14:39 2021-11-13

Ввод [109]:

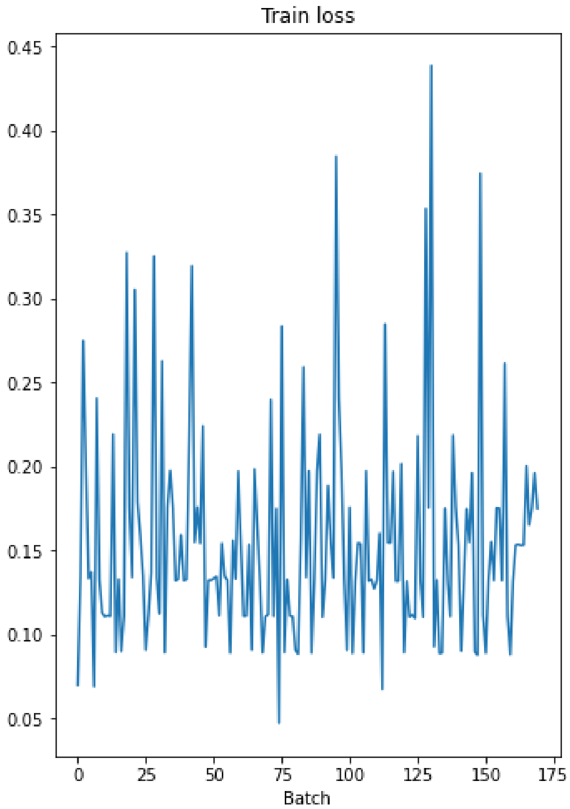
```
import time
import math
import matplotlib
matplotlib.rcParams.update({'figure.figsize': (16, 12), 'font.size': 14})
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import clear_output
```

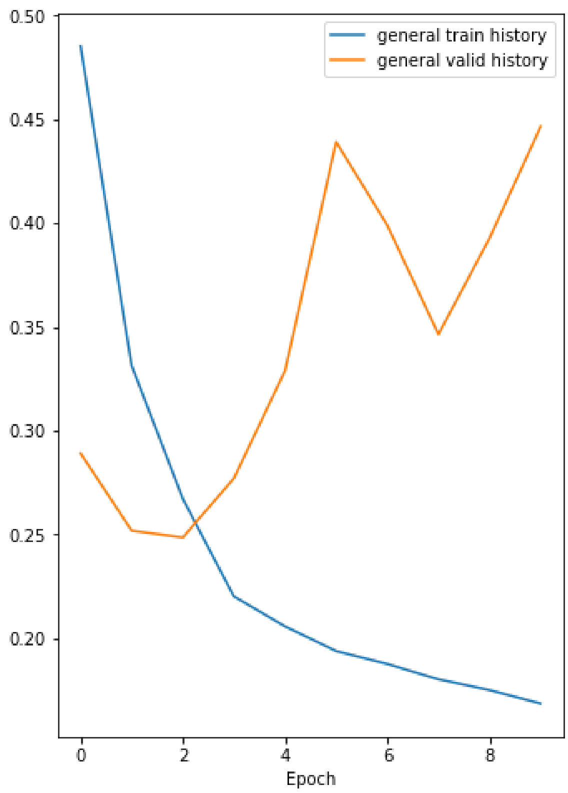
executed in 44ms, finished 17:14:39 2021-11-13

Ввод [110]:

```
train_history = []
valid_history = []

N_EPOCHS = 11
CLIP = 2

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, v
    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}')
```

executed in 3m 48s, finished 17:18:28 2021-11-13






```
Epoch: 11 | Time: 0m 20s
	Train Loss: 0.154 | Train PPL:   1.166
	 Val. Loss: 0.411 |  Val. PPL:   1.508
```

Ввод [112]:

```
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)):
        input_data=batch['inputs'].to(device)
        mask=batch['attention_mask'].to(device)
        labels=batch['labels']
        output=best_model(input_data,mask).cpu()
        pred_labels.append(torch.where(output.squeeze(1) <=0.5, 0., 1.))
        true_labels.append(labels.numpy())
```

executed in 1.06s, finished 17:20:01 2021-11-13

```
0it [00:00, ?it/s]
```

Ввод [113]:

```
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)
```

executed in 13ms, finished 17:20:01 2021-11-13

Out[113]:

```
0.8829479768786127
```

Ввод [114]:

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

executed in 11ms, finished 17:20:02 2021-11-13

Finetuned model from HUGGING FACE

[BertForSequenceClassification (https://huggingface.co/transformers/model_doc/bert.html?highlight=bertfor#transformers.BertForSequenceClassification)](https://huggingface.co/transformers/model_doc/bert.html?highlight=bertfor#transformers.BertForSequenceClassification)

Ввод [47]:

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification

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

executed in 27.0s, finished 16:21:22 2021-11-13

```
Downloading:   0%|          | 0.00/629 [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/255M [00:00<?, ?B/s]
```

Ввод [91]:

```
pred_labels = []
true_labels = []

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

        input_data=batch['inputs'].to(device)
        mask=batch['attention_mask'].to(device)
        labels=batch['labels']
        output=new_model(input_data,mask)['logits'].cpu()
        pred_labels.append(np.array(list(map(lambda out: 0 if out[0]>0 else 1,output))))
        true_labels.append(labels.numpy())

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

executed in 716ms, finished 16:48:45 2021-11-13

```
0it [00:00, ?it/s]
```

Out[91]:

```
0.9841040462427746
```

Ввод [92]:

```
model_structure(new_model)
```

executed in 20ms, finished 16:48:48 2021-11-13

```
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
            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
            sa_layer_norm
            ffn:
                dropout
                lin1
                lin2
            output_layer_norm
pre_classifier
classifier
dropout
```

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

Качество с помощью Fine-Tuning должно достигать 0.86.

dropout=0.25; optimizer = Adam(lr=2e-5); criterion = BCELoss(); dim hidden FC = 64 (clf_1 -(dim)> clf_2); N_EPOCHS = 11; CLIP = 2

Модель от HUGGING FACE дала результат (0.985) намного лучше, чем ручной Fine-Tuning (0.883), что говорит о том, что можно для данной можно написать более хороший класификатор (основное отличие между моделями в том, что у HG используется dim hidden FC = 768, то есть у их модели больше параметров, а также могут быть различия в препроцессинге).

Ввод [ ]:

```

```