Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Задание 3

Классификация текстов

В этом задании вам предстоит попробовать несколько методов, используемых в задаче классификации, а также понять насколько хорошо модель понимает смысл слов и какие слова в примере влияют на результат.

```
import pandas as pd
import numpy as np
import torch

from torchtext.legacy import datasets

from torchtext.legacy.data import Field, LabelField
from torchtext.legacy.data import BucketIterator

from torchtext.vocab import Vectors, GloVe

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import random
from tqdm.autonotebook import tqdm


import warnings
warnings.filterwarnings("ignore")
```

В этом задании мы будем использовать библиотеку torchtext. Она довольна проста в использовании и поможет нам сконцентрироваться на задаче, а не на написании Dataloader-а.

```
TEXT = Field(sequential=True, lower=True, include_lengths=True)  # Поле текста # sequential=True т.к. в
LABEL = LabelField(dtype=torch.float)  # Поле метки
```

```
SEED = 1234
```

```
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
```

Датасет на котором мы будем проводить эксперементы это комментарии к фильмам из сайта IMDB.

```
train_Dataset, test_Dataset = datasets.IMDB.splits(TEXT, LABEL)  # загрузим датасет
train_Dataset, valid_Dataset = train_Dataset.split(random_state=random.seed(SEED))  # разобьем на части
```

```
downloading aclImdb_v1.tar.gz
aclImdb_v1.tar.gz: 100%|██████████| 84.1M/84.1M [00:05<00:00, 15.2MB/s]
```

```
type(train_Dataset)
```

```
torchtext.legacy.data.dataset.Dataset
```

```
len(train_Dataset), len(valid_Dataset), len(test_Dataset)
```

```
(17500, 7500, 25000)
```

```
for example in train_Dataset:
    print(type(example))
    print(dir(example))
    break
```

```
<class 'torchtext.legacy.data.example.Example'>
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
```

```
TEXT.build_vocab(train_Dataset)
LABEL.build_vocab(train_Dataset)
```

```
LABEL.vocab.freqs
```

```
Counter({'neg': 8810, 'pos': 8690})
```

```
TEXT.vocab.freqs
```

```
          'where': 4319,
          'almost': 2136,
          'recite': 14,
          'line': 881,
          'dialogue': 728,
          'before': 2365,
          'hearing': 136,
          'liked': 1009,
          'it-': 13,
          'anymore,': 31,
          'refreshing': 109,
          'change.': 55,
          'interesting': 1682,
          'kelly': 206,
          'preston,': 3,
          'leaf': 11,
          'phoenix': 24,
          'lea': 9,
          'thomson': 5,
          'early': 1037,
          'roles,': 103,
          'tom': 450,
          'skerrit': 1,
          'kate': 167,
          'capshaw': 18,
```

```
             'add': 496,
             'substance': 87,
             'light': 435,
             'fluffy': 14,
             'plot.': 325,
             'absolutely': 1040,
             'loved': 931,
             'robot': 92,
             'named': 533,
             'jinx,': 2,
             'cute,': 51,
             'unfortunately': 444,
             'emotion': 170,
             'characters.': 442,
             'inspirational': 27,
             'own': 2103,
             'way,': 596,
             'note': 277,
             'filmed': 425,
             'nasa': 13,
             'spacecamp': 7,
             'alabama': 4,
             '(i': 510,

             'think).': 8,
             "haven't": 545,
             'seen': 3726,
             '"incredible': 1,
             'journey"': 3,
             'since': 1875,
             'child,': 92,
             "can't": 2485,
             'compare': 206,
             ...})
```

```
len(TEXT.vocab.stoi)
```

```
202779
```

```
list(TEXT.vocab.stoi.items())[-10:]
```

```
[('“him”', 201373),
 ('“it’s', 201374),
 ('“jean', 201375),
 ('“little', 201376),
 ('“mad', 201377),
 ('“mr.', 201378),
 ('“playboy”', 201379),
 ('“sanatorium”', 201380),
 ('“x”,', 201381),
 ('£100', 201382)]
```

```
TEXT.vocab.stoi["£100"]
```

```
201382
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

torchtext_train_dataloader, torchtext_valid_dataloader, torchtext_test_dataloader = BucketIterator.spl
    (train_Dataset, valid_Dataset, test_Dataset),
    batch_size=64,
    device = device,
    sort_key=lambda x: len(x.text),
    sort=False,
    shuffle=True,
```

```
    sort_within_batch=True
    )
```

```
torchtext_train_dataloader.create_batches()
torchtext_valid_dataloader.create_batches()
torchtext_test_dataloader.create_batches()
```

```
dir(torchtext_train_dataloader)
```

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 '_iterations_this_epoch',
 '_random_state_this_epoch',
 '_restored_from_state',
 'batch_size',
 'batch_size_fn',
 'batches',
 'create_batches',
 'data',
 'dataset',
 'device',
 'epoch',
 'init_epoch',
 'iterations',
 'load_state_dict',
 'random_shuffler',
 'repeat',
 'shuffle',
 'sort',
 'sort_key',
 'sort_within_batch',
 'splits',
 'state_dict',
 'train']
```

```
[len(text.text) for text in torchtext_train_dataloader.data()] # распределение длин предложений
```

```
 156,
 49,
```

```
71,
121,
127,
115,
298,
110,
477,
145,
114,
244,
138,
232,
239,
124,
173,
396,
109,
110,
75,
142,
631,
150,
133,
58,
127,
150,
151,
182,
139,
296,
156,
335,
150,
188,
193,
272,

132,
156,
60,
352,
137,
401,
67,
226,
129,
223,
587,
545,
558,
456,
202,
226,
267,
305,
122,
451,
102.
```

```
len(torchtext_train_dataloader.data()) # 17500 примеров в torchtext_train_dataloader
```

```
17500
```

```
len(torchtext_valid_dataloader.data()) # 7500 примеров для валидации
```

```
7500
```

```
len(torchtext_test_dataloader.data()) # 25000 примеров для теста
```

```
25000
```

```
for batch_no, batch in enumerate(torchtext_train_dataloader):
    text, batch_len = batch.text # text.size() -> seq_len, batch_size
    print(text, batch_len, sep="\n")
    print(batch.label)
    break
```

```
tensor([[     9,     49,     10,  ...,   7828,      3,  94314],
        [    85,      9,     20,  ...,     10,    764, 141347],
        [    98,     82,     14,  ...,      7,   2525,     13],
        ...,
        [    24,    108, 198439,  ...,      1,      1,      1],
        [    15,    103,     13,  ...,      1,      1,      1],
        [ 14562, 179732,    728,  ...,      1,      1,      1]],
       device='cuda:0')
tensor([297, 297, 297, 297, 297, 297, 296, 296, 296, 296, 296, 296, 295, 295,
        295, 295, 295, 295, 294, 294, 294, 294, 294, 294, 293, 293, 293, 293,
        293, 293, 293, 293, 293, 292, 292, 292, 292, 292, 292, 291, 291, 291,
        291, 290, 290, 290, 290, 290, 290, 290, 289, 289, 289, 289, 289, 289,
        289, 289, 288, 288, 288, 288, 288, 288], device='cuda:0')
tensor([1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
        1., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0.,
        1., 1., 0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1., 1., 1., 1.,
        0., 0., 0., 0., 1., 1., 0., 0., 0., 1.], device='cuda:0')
```

```
text.size(), batch_len.size(), batch.label.size()
```

```
(torch.Size([297, 64]), torch.Size([64]), torch.Size([64]))
```

RNN

Для начала попробуем использовать рекурентные нейронные сети. На семинаре вы познакомились с GRU, вы можете также попробовать LSTM. Можно использовать для классификации как hidden_state, так и output последнего токена.

```
# Для инициализации self.rnn(см. ниже) очевидно нужно создать модель рекуррентной нейронной сети:
# Очевидно, это может быть:
# 1) RNN
# 2) GRU
# 3) LSTM

# Конечно же можно воспользоваться определёнными в семинаре рекуррентными блоками, но воспользуемся
# реализованными уже в PyTorch
```

```
from torch.nn import RNN, GRU, LSTM
```

```
class RNNBaseline(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers,
                 bidirectional, dropout, pad_idx):

        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx = pad_idx)

        # YOUR CODE GOES HERE
        self.lstm = LSTM(input_size=embedding_dim, hidden_size=hidden_dim, num_layers=n_layers, bidirec
```

```
        # YOUR CODE GOES HERE
        self.fc = nn.Linear(2 * hidden_dim, output_dim)


    def forward(self, text, text_lengths):

        #text = [sent len, batch size]

        embedded = self.embedding(text)

        #embedded = [sent len, batch size, emb_dim]

        #pack sequence
        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, text_lengths).to(device)

        # cell arg for LSTM, remove for GRU
        packed_output, (hidden, cell) = self.lstm(packed_embedded)

        #unpack sequence
        output, output_lengths = nn.utils.rnn.pad_packed_sequence(packed_output)

        #output = [sent len, batch size, hid dim * num directions]
        #output over padding tokens are zero tensors

        #hidden = [num layers * num directions, batch size, hid dim]
        #cell = [num layers * num directions, batch size, hid dim]

        #concat the final forward (hidden[-2,:,:]) and backward (hidden[-1,:,:]) hidden layers
        #and apply dropout

        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)  # YOUR CODE GOES HERE

        #hidden = [batch size, hid dim * num directions] or [batch_size, hid dim * num directions]
        #print(hidden.size()) # (batch_size, 2 * hidden_size)
        return self.fc(hidden) # (batch_size, output_dim)
```

Поиграйтесь с гиперпараметрами

```
vocab_size = len(TEXT.vocab) # размер словаря(кол-во слов в словаре)
emb_dim = 100 # размерность embeddings
hidden_dim = 256 # размерность скрытого состояния
output_dim = 2 # кол-во выходных слоёв после линейного слоя
n_layers = 2 # кол - во рекуррентных ячеек
bidirectional = True
dropout = 0.2
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=7


model = RNNBaseline(
    vocab_size=vocab_size,
    embedding_dim=emb_dim,
    hidden_dim=hidden_dim,
    output_dim=output_dim,
    n_layers=n_layers,
    bidirectional=bidirectional,
    dropout=dropout,
    pad_idx=PAD_IDX
)


model = model.to(device)
```

```
opt = torch.optim.Adam(model.parameters())
loss_func = nn.BCEWithLogitsLoss()
```

Обучите сетку! Используйте любые вам удобные инструменты, Catalyst, PyTorch Lightning или свои велосипеды.

```
from sklearn.metrics import f1_score


import numpy as np

min_loss = np.inf

cur_patience = 0
max_epochs = 20

for epoch in range(1, max_epochs + 1):

    train_loss = 0.0
    train_f1_score = 0.0

    model.train()

    pbar = tqdm(enumerate(torchtext_train_dataloader), total=len(torchtext_train_dataloader), leave=Fa
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:

        #YOUR CODE GOES HERE
        text, text_lengths = batch.text
        y_true = batch.label

        opt.zero_grad()

        y_pred = model(text, text_lengths.cpu())
        #print(predictions.size())
        loss = loss_func(y_pred[:, 1], y_true)
        loss.backward()
        opt.step()

        train_loss += loss.detach().cpu().item()
        train_f1_score += f1_score(y_true.cpu().numpy(), torch.argmax(y_pred, dim=1).cpu().numpy())
    train_loss /= len(torchtext_train_dataloader)
    train_f1_score /= len(torchtext_train_dataloader)

    val_loss = 0.0
    val_f1_score = 0.0

    model.eval()

    pbar = tqdm(enumerate(torchtext_valid_dataloader), total=len(torchtext_valid_dataloader), leave=Fa
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:
        # YOUR CODE GOES HERE
        with torch.no_grad():
            text, text_lengths = batch.text
            y_true = batch.label
```

```
            y_pred = model(text, text_lengths.cpu())

            loss = loss_func(y_pred[:, 1], y_true)

            val_loss += loss.cpu().item()
            val_f1_score += f1_score(y_true.cpu().numpy(), torch.argmax(y_pred, dim=1).cpu().numpy())
    val_loss /= len(torchtext_valid_dataloader)
    val_f1_score /= len(torchtext_valid_dataloader)

    if val_loss < min_loss:
        min_loss = val_loss
        best_model = model.state_dict()
    else:
        cur_patience += 1
        if cur_patience == patience:
            cur_patience = 0
            break

    print('Epoch: {}, Training Loss: {}, Validation Loss: {}'.format(epoch, train_loss, val_loss))
    print("---------  Training f1_score: {}, Validation f1_score: {}".format(train_f1_score, val_f1_sco

model.load_state_dict(best_model)
```

```
Epoch 1: 100%                                   274/274 [01:23<00:00, 3.75it/s]

Epoch 1: 100%                                   118/118 [00:10<00:00, 9.69it/s]

Epoch: 1, Training Loss: 0.6295029827713097, Validation Loss: 0.5650419489306918
---------  Training f1_score: 0.6102908851470303, Validation f1_score: 0.7394398467345911

Epoch 2: 100%                                   274/274 [01:22<00:00, 1.84it/s]

Epoch 2: 99%                                    117/118 [00:10<00:00, 11.80it/s]

Epoch: 2, Training Loss: 0.5622195035871798, Validation Loss: 0.6186277644108917
---------  Training f1_score: 0.6906687395612868, Validation f1_score: 0.6791930785746728

Epoch 3: 100%                                   274/274 [01:22<00:00, 3.42it/s]

Epoch 3: 99%                                    117/118 [00:10<00:00, 10.97it/s]
```

```
test_loss = 0.0
test_f1_score = 0.0

model.eval()

pbar = tqdm(enumerate(torchtext_test_dataloader), total=len(torchtext_test_dataloader), leave=False)

for it, batch in pbar:
    with torch.no_grad():
        text, text_lengths = batch.text
        y_true = batch.label

        y_pred = model(text, text_lengths.cpu())

        loss = loss_func(y_pred[:, 1], y_true)

        test_loss += loss.cpu().item()
        test_f1_score += f1_score(y_true.cpu().numpy(), torch.argmax(y_pred, dim=1).cpu().numpy())
test_loss /= len(torchtext_test_dataloader)
test_f1_score /= len(torchtext_test_dataloader)

print("Testing Loss: {}".format(test_loss))
print("Testing f1_score: {}".format(test_f1_score))
```

```
100%                                   390/391 [00:32<00:00, 13.41it/s]

Testing Loss: 0.9582511080652857
Testing f1_score: 0.8278642686492178

Epoch 9: 100%                                   274/274 [01:22<00:00, 5.45it/s]
```

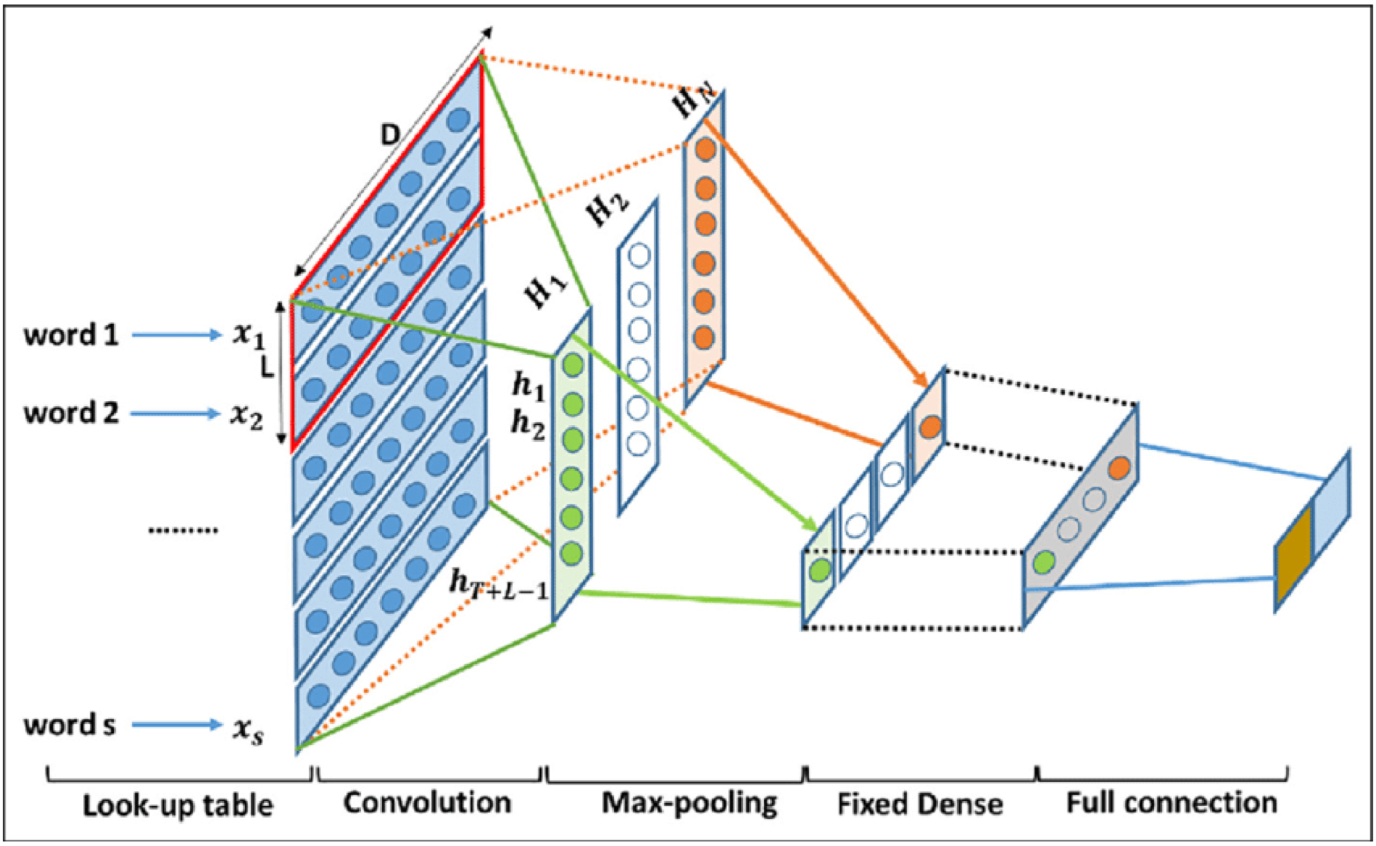
Посчитайте f1-score вашего классификатора на тестовом датасете.

Ответ:

```
# Testing f1_score: 0.8278642686492178
```

```
Epoch 10: 100%                                   118/118 [00:10<00:00, 8.5 it/s]
```

CNN



Для классификации текстов также часто используют сверточные нейронные сети. Идея в том, что как правило сентимент содержат словосочетания из двух-трех слов, например "очень хороший фильм" или "невероятная скука". Проходясь сверткой по этим словам мы получим какой-то большой скор и выхватим его с помощью MaxPool. Далее идет обычная полносвязная сетка. Важный момент: свертки применяются не последовательно, а параллельно. Давайте попробуем!

```
TEXT = Field(sequential=True, lower=True, batch_first=True)  # batch_first тк мы используем conv
LABEL = LabelField(batch_first=True, dtype=torch.float)

train_Dataset1, test_Dataset1 = datasets.IMDB.splits(TEXT, LABEL)
train_Dataset1, valid_Dataset1 = train_Dataset1.split(random_state=random.seed(SEED))

TEXT.build_vocab(train_Dataset1)
LABEL.build_vocab(train_Dataset1)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
downloading aclImdb_v1.tar.gz
aclImdb_v1.tar.gz: 100%|██████████| 84.1M/84.1M [00:02<00:00, 36.6MB/s]
```

```
len(train_Dataset1), len(valid_Dataset1), len(test_Dataset1)
```

```
(17500, 7500, 25000)
```

```
torchtext_train_dataloader1, torchtext_valid_dataloader1, torchtext_test_dataloader1 = BucketIterator.s
        (train_Dataset1, valid_Dataset1, test_Dataset1),
        batch_sizes=(128, 256, 256),
        sort=False,
        sort_key= lambda x: len(x.text),
        sort_within_batch=True,
        device=device,
```

```
        repeat=False,
`
len(torchtext_train_dataloader1), len(torchtext_valid_dataloader1), len(torchtext_test_dataloader1)
```

```
(137, 30, 98)
```

```
torchtext_train_dataloader1.create_batches()
torchtext_valid_dataloader1.create_batches()
torchtext_test_dataloader1.create_batches()
```

Вы можете использовать Conv2d с `in_channels=1, kernel_size=(kernel_sizes[0], emb_dim))` или Conv1d с `in_channels=emb_dim, kernel_size=kernel_size[0]`. Но хорошенько подумайте над shape в обоих случаях.

```
emb_dim = 100
kernel_sizes = [3, 4, 5]
out_channels=64
```

```
# достанем один батч, чтобы мы могл отслеживать размеры тензора, проходящего
# через модель
```

```
for batch_no, batch in enumerate(torchtext_train_dataloader1):
    text = batch.text # text.size() -> seq_len, batch_size
    label = batch.label
    print(text)
    print(label)
    break
```

```
tensor([[   49,    88,  4630,  ..., 19563,   116,  4842],
        [    2,  1282,    17,  ..., 10491,     1,     1],
        [   10,     7,     3,  ...,     1,     1,     1],
        ...,
        [10545, 10545,    10,  ...,     1,     1,     1],
        [    9,  4649,   309,  ...,     1,     1,     1],
        [    2, 10868,     7,  ...,     1,     1,     1]], device='cuda:0')
tensor([1., 0., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0.,
        1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
        0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
        0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1.,
        0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1.,
        1., 1., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 0., 1.,
        1., 0., 1., 0., 1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
        1., 1.], device='cuda:0')
```

```
text = text.cpu()
label = label.cpu()
```

```
text.size(), label.size()
```

```
(torch.Size([168, 2278]), torch.Size([128]))
```

```
# Небольшое изменение в процессе корректировки
# Т.к. последующий шаг по интерпретации модели не отрабатаывает, сетую на то, что это
# связано с тем, что модель возвращает 2 класса, а не 1, как подразумевалось авторами,
# поэтому я видоизменяю выход модели, и слегка передылваю train_val_loop, по сравнению
# с реализацией аналогичных пунктов, приведенных выше
```

```
# Будем итеративно строить нашу модель
model_ = nn.Sequential()
model_.add_module("emd", nn.Embedding(len(TEXT.vocab), emb_dim))
```

```
b1 = model_(text)
b1.size() # batch_size, seq_length, embedding_dim
```

```
torch.Size([168, 2278, 100])
```

```
# поменяем порядок , чтобы мы правильно применяли свёрточные слои
class Permute(nn.Module):

    def forward(self, x):
        return  x.permute((0, 2, 1))


model_.add_module("permute", Permute())
```

```
b2 = model_(text)
b2.size() # batch_size, embedding_dim, seq_length
```

```
torch.Size([168, 100, 2278])
```

```
# Теперь, когда резмерности батча приведены в правильный порядок добавим свёртки

model_.add_module("conv1", nn.Conv1d(in_channels=emb_dim, out_channels=out_channels, kernel_size=kernel
```

```
b3 = model_(text)
b3.size()
```

```
torch.Size([168, 64, 2275])
```

```
# также необходимо добавить пулинги

model_.add_module("pool1", nn.MaxPool1d(kernel_size=b3.size(2)))
```

```
b4 = model_(text)
b4.size()
```

```
torch.Size([168, 64, 1])
```

```
model_ # пример структуры получившейся модели, теперь реализуем полноценную модельку на основе экспери
```

```
Sequential(
  (emd): Embedding(201383, 100)
  (permute): Permute()
  (conv1): Conv1d(100, 64, kernel_size=(4,), stride=(1,))
  (pool1): MaxPool1d(kernel_size=2275, stride=2275, padding=0, dilation=1, ceil_mode=False)
)
```

```
kernel_sizes = [3, 4, 5]
vocab_size = len(TEXT.vocab)
emb_dim = 100
hidden_dim = 256
out_channels = 64
out_channel = 1
dropout = 0.5
```

```
class CNN(nn.Module):
    def __init__(
        self,
        vocab_size,
        emb_dim,
        out_channels,
        kernel_sizes,
        dropout=0.5,
        out_channel=1
    ):
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, emb_dim)

        # YOUR CODE GOES HERE
        self.conv_0 = nn.Conv1d(in_channels=emb_dim, out_channels=out_channels, kernel_size=kernel_size
        self.conv_1 = nn.Conv1d(in_channels=emb_dim, out_channels=out_channels, kernel_size=kernel_size
        self.conv_2 = nn.Conv1d(in_channels=emb_dim, out_channels=out_channels, kernel_size=kernel_size

        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(len(kernel_sizes) * out_channels, out_channel)


    def forward(self, text):

        embedded = self.embedding(text) # batch_size, seq_length, emb_dim

        embedded = embedded.permute((0, 2, 1)) # batch_size, emb_dim, seq_length

        conved_0 = F.relu(self.conv_0(embedded)) # batch_size, out_channels, *
        conved_1 = F.relu(self.conv_1(embedded))  # may be reshape here
        conved_2 = F.relu(self.conv_2(embedded))  # may be reshape here

        pooled_0 = F.max_pool1d(conved_0, conved_0.shape[2]).squeeze(2)
        pooled_1 = F.max_pool1d(conved_1, conved_1.shape[2]).squeeze(2)
        pooled_2 = F.max_pool1d(conved_2, conved_2.shape[2]).squeeze(2)

        cat = self.dropout(torch.cat((pooled_0, pooled_1, pooled_2), dim=1))

        return self.fc(cat)


cnn_model = CNN(vocab_size=vocab_size, emb_dim=emb_dim, out_channels=out_channels,
            kernel_sizes=kernel_sizes, dropout=dropout)


cnn_model.to(device)
```

```
CNN(
  (embedding): Embedding(201383, 100)
  (conv_0): Conv1d(100, 64, kernel_size=(3,), stride=(1,))
  (conv_1): Conv1d(100, 64, kernel_size=(3,), stride=(1,))
  (conv_2): Conv1d(100, 64, kernel_size=(3,), stride=(1,))
  (dropout): Dropout(p=0.5, inplace=False)
  (fc): Linear(in_features=192, out_features=1, bias=True)
)
```

```
opt = torch.optim.Adam(cnn_model.parameters())
loss_func = nn.BCEWithLogitsLoss()
```

```
max_epochs = 20
patience = 7
```

Обучите!

```
import numpy as np

min_loss = np.inf

cur_patience = 0

for epoch in range(1, max_epochs + 1):

    train_f1_score = 0.0
    train_loss = 0.0

    cnn_model.train()

    pbar = tqdm(enumerate(torchtext_train_dataloader1), total=len(torchtext_train_dataloader1), leave=F
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:
        #YOUR CODE GOES HERE
        text = batch.text
        y_true = batch.label

        opt.zero_grad()

        y_pred = cnn_model(text) # (batch_size, 1)

        loss = loss_func(y_pred, y_true)
        loss.backward()
        opt.step()

        train_loss += loss.detach().cpu().item()
        train_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().nu
    train_loss /= len(torchtext_train_dataloader1)
    train_f1_score /= len(torchtext_train_dataloader1)

    val_f1_score = 0.0
    val_loss = 0.0

    cnn_model.eval()

    pbar = tqdm(enumerate(torchtext_valid_dataloader1), total=len(torchtext_valid_dataloader1), leave=F
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:
        # YOUR CODE GOES HERE
        with torch.no_grad():
            text = batch.text
            y_true = batch.label

            y_pred = cnn_model(text).squeeze()

            loss = loss_func(y_pred, y_true)

            val_loss += loss.cpu().item()
            val_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().
    val_loss /= len(torchtext_valid_dataloader1)
```

```
        val_f1_score /= len(torchtext_valid_dataloader1)

        if val_loss < min_loss:
            min_loss = val_loss
            best_model = cnn_model.state_dict()
        else:
            cur_patience += 1
            if cur_patience == patience:
                cur_patience = 0
                break

        print('Epoch: {}, Training Loss: {}, Validation Loss: {}'.format(epoch, train_loss, val_loss))
        print("---------  Training f1_score: {}, Validation f1_score: {}".format(train_f1_score, val_f1_sco

    cnn_model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.48682590884013766, Validation Loss: 0.43820490340391793
---------  Training f1_score: 0.7641495323186296, Validation f1_score: 0.8080211969677539
Epoch: 2, Training Loss: 0.44896458687573454, Validation Loss: 0.42924417853355407
---------  Training f1_score: 0.785061448098559, Validation f1_score: 0.7953441559887299
Epoch: 3, Training Loss: 0.4222770122280956, Validation Loss: 0.3911970357100169
---------  Training f1_score: 0.805632912463562, Validation f1_score: 0.8304493831160948
Epoch: 4, Training Loss: 0.374747602935255, Validation Loss: 0.37426924804846445
---------  Training f1_score: 0.8311339703175923, Validation f1_score: 0.8380426241273989
Epoch: 5, Training Loss: 0.34397409236344106, Validation Loss: 0.35686516265074414
---------  Training f1_score: 0.8470058393453125, Validation f1_score: 0.8488624928684261
Epoch: 6, Training Loss: 0.3062836676836014, Validation Loss: 0.34661132593949634
---------  Training f1_score: 0.8704236839917929, Validation f1_score: 0.8531555652605135
Epoch: 7, Training Loss: 0.26181138170896656, Validation Loss: 0.340271465977033
---------  Training f1_score: 0.8917240052456298, Validation f1_score: 0.8543952248320663
Epoch: 8, Training Loss: 0.2194620413284232, Validation Loss: 0.3449074973662694
---------  Training f1_score: 0.9136340345579217, Validation f1_score: 0.8601033675974821
Epoch: 9, Training Loss: 0.17989584042208037, Validation Loss: 0.35917299886544546
---------  Training f1_score: 0.9291953153323878, Validation f1_score: 0.8554526884931596
Epoch: 10, Training Loss: 0.14049360824980006, Validation Loss: 0.36960606773694354
---------  Training f1_score: 0.947368961869564, Validation f1_score: 0.8541833167646988
Epoch: 11, Training Loss: 0.10692255193517156, Validation Loss: 0.39880796273549396
---------  Training f1_score: 0.9603527807932396, Validation f1_score: 0.8494349442935625
Epoch: 12, Training Loss: 0.08981524391548477, Validation Loss: 0.43147728343804675
---------  Training f1_score: 0.9686784920951304, Validation f1_score: 0.8432076895377386
Epoch: 13, Training Loss: 0.06441939048414683, Validation Loss: 0.45489110549290973
---------  Training f1_score: 0.9784241308437239, Validation f1_score: 0.8395926109837831
<All keys matched successfully>
```

```
test_loss = 0.0
test_f1_score = 0.0

cnn_model.eval()

pbar = tqdm(enumerate(torchtext_test_dataloader1), total=len(torchtext_test_dataloader1), leave=False)

for it, batch in pbar:
    with torch.no_grad():
        text = batch.text
        y_true = batch.label

        y_pred = cnn_model(text).squeeze()

        loss = loss_func(y_pred, y_true)

        test_loss += loss.cpu().item()
        test_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().nur
test_loss /= len(torchtext_test_dataloader1)
test_f1_score /= len(torchtext_test_dataloader1)
```

```
print("Testing Loss: {}".format(test_loss))
print("Testing f1_score: {}".format(test_f1_score))
```

```
Testing Loss: 0.5149682751115487
Testing f1_score: 0.8255127660813958
```

Посчитайте f1-score вашего классификатора.

Ответ:

```
# Testing f1_score: 0.8255127660813958
```

Интерпретируемость

Посмотрим, куда смотрит наша модель. Достаточно запустить код ниже.

```
!pip install -q captum
```

```
|████████████████████████████████| 1.4 MB 4.2 MB/s
```

```
from captum.attr import LayerIntegratedGradients, TokenReferenceBase, visualization

PAD_IND = TEXT.vocab.stoi['pad']

token_reference = TokenReferenceBase(reference_token_idx=PAD_IND)
lig = LayerIntegratedGradients(cnn_model, cnn_model.embedding)


def forward_with_softmax(inp):
    logits = model(inp)
    return torch.softmax(logits, 0)[0][1]

def forward_with_sigmoid(input):
    return torch.sigmoid(cnn_model(input))


# accumalate couple samples in this array for visualization purposes
vis_data_records_ig = []

def interpret_sentence(model, sentence, min_len = 7, label = 0):
    model.eval()
    text = [tok for tok in TEXT.tokenize(sentence)]
    if len(text) < min_len:
        text += ['pad'] * (min_len - len(text))
    indexed = [TEXT.vocab.stoi[t] for t in text]

    model.zero_grad()

    input_indices = torch.tensor(indexed, device=device)
    input_indices = input_indices.unsqueeze(0)

    # input_indices dim: [sequence_length]
    seq_length = min_len

    # predict
    pred = forward_with_sigmoid(input_indices).squeeze()
```

```
        pred_ind = round(pred.item())

        # generate reference indices for each sample
        reference_indices = token_reference.generate_reference(seq_length, device=device).unsqueeze(0)

        # compute attributions and approximation delta using layer integrated gradients
        attributions_ig, delta = lig.attribute(input_indices, reference_indices, \
                                               n_steps=5000, return_convergence_delta=True)

        print('pred: ', LABEL.vocab.itos[pred_ind], '(', '%.2f'%pred, ')', ', delta: ', abs(delta))

        add_attributions_to_visualizer(attributions_ig, text, pred, pred_ind, label, delta, vis_data_record

    def add_attributions_to_visualizer(attributions, text, pred, pred_ind, label, delta, vis_data_records):
        attributions = attributions.sum(dim=2).squeeze(0)
        attributions = attributions / torch.norm(attributions)
        attributions = attributions.cpu().detach().numpy()

        # storing couple samples in an array for visualization purposes
        vis_data_records.append(visualization.VisualizationDataRecord(
                                attributions,
                                pred,
                                LABEL.vocab.itos[pred_ind],
                                LABEL.vocab.itos[label],
                                LABEL.vocab.itos[1],
                                attributions.sum(),
                                text,
                                delta))

    interpret_sentence(cnn_model, 'It was a fantastic performance !', label=1)
    interpret_sentence(cnn_model, 'Best film ever', label=1)
    interpret_sentence(cnn_model, 'Such a great show!', label=1)
    interpret_sentence(cnn_model, 'It was a horrible movie', label=0)
    interpret_sentence(cnn_model, 'I\'ve never watched something as bad', label=0)
    interpret_sentence(cnn_model, 'It is a disgusting movie!', label=0)
```

```
pred:  pos ( 0.96 ) , delta:  tensor([8.9544e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([3.2459e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.11 ) , delta:  tensor([1.1704e-06], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([1.1465e-06], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.08 ) , delta:  tensor([4.0419e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.38 ) , delta:  tensor([6.6384e-05], device='cuda:0', dtype=torch.float64)
```

Попробуйте добавить свои примеры!

```
print('Visualize attributions based on Integrated Gradients')
visualization.visualize_text(vis_data_records_ig)
```

```
Visualize attributions based on Integrated Gradients
```

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.96)	pos	1.39	It was a fantastic performance ! pad
pos	neg (0.00)	pos	1.57	Best film ever pad pad pad pad
pos	neg (0.11)	pos	1.26	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.18	It was a horrible movie pad pad
neg	neg (0.08)	pos	0.92	I've never watched something as bad pad

Эмбеддинги слов

Вы ведь не забыли, как мы можем применить знания о word2vec и GloVe. Давайте попробуем!

```
TEXT2 = Field(sequential=True, use_vocab=True, lower=True, batch_first=True)
LABEL2 = LabelField(use_vocab=True, batch_first=True, dtype=torch.float)
```

neg neg (0.00) pos -0.18 It was a horrible movie pad pad

```
train_Dataset2, test_Dataset2 = datasets.IMDB.splits(TEXT2, LABEL2)  # загрузим датасет
train_Dataset2, valid_Dataset2 = train_Dataset2.split(random_state=random.seed(SEED))
```

```
TEXT2.build_vocab(train_Dataset2, vectors="glove.42B.300d")
LABEL2.build_vocab(train_Dataset2)
```

```
.vector_cache/glove.42B.300d.zip: 1.88GB [06:40, 4.68MB/s]
100%|██████████| 1917493/1917494 [04:18<00:00, 7420.99it/s]
```

```
# посмотрим, что получилось
TEXT2.vocab.stoi
```

```
 'there.': 944,
 "'the": 945,
 'forced': 946,
 'subject': 947,
 'particular': 948,
 'team': 949,
 'unfortunately,': 950,
 'mystery': 951,
 'scenes,': 952,
 'reviews': 953,
 'weak': 954,
 'average': 955,
 'lee': 956,
 'then,': 957,
 'fantastic': 958,
 'male': 959,
 'crap': 960,
 'forward': 961,
 'there,': 962,
 'interested': 963,
 'political': 964,
 'writers': 965,
 'crime': 966,
 'decides': 967,
 'sister': 968,
 'minute': 969,
 'wait': 970,
 'waiting': 971,
 'york': 972,
 'you.': 973,
 '(a': 974,
```

```
          'plain': 975,
          'premise': 976,
          'whatever': 977,
          'attempts': 978,
          'follow': 979,
          'nature': 980,
          'slightly': 981,
          'sounds': 982,
          'up,': 983,
          'casting': 984,
          'dialog': 985,
          'directors': 986,
          'telling': 987,
          'hold': 988,
          'storyline': 989,
          'admit': 990,
          'fast': 991,
          'pay': 992,
          'sequences': 993,
          'worked': 994,

          'dr.': 995,
          'editing': 996,
          'fails': 997,
          'man,': 998,
          'season': 999,
          ...})
```

```
TEXT2.vocab.vectors.size()
```

```
torch.Size([201383, 300])
```

```
TEXT2.vocab.freqs
```

```
          'review.': 61,
          'am': 1842,
          'huge': 640,
          'denver': 21,
          'fan.': 95,
          'large': 330,
          'music': 1528,
          'vinyl.': 2,
          'saw': 2140,
          'originally': 178,
          'tv': 1437,
          'vinyl': 5,
          'album': 32,
          'cd.': 12,
          'cd': 49,
          'later': 982,
          'release.': 76,
          'release': 329,
          'several': 951,
          'songs': 492,
          'though.': 232,
          'released': 525,
          'songs.': 50,
          'surprise': 295,
          'sale': 23,
          '$75.00.': 1,
          'wow': 40,
          'worth': 1515,
          'much.': 284,
          'amount': 326,
          'selling': 73,
          'treasure.': 26,
          'vhs': 143,
```

```
             'dvd.': 167,
             'love': 3753,
             'version.': 119,
             'available': 208,
             'please': 477,
             'let': 1079,
             'know.': 129,
             'thanks': 279,
             '1930,europe': 1,
             'received': 154,
             'shock': 192,
             "bunuel's": 7,
             "'l'age": 1,
             "dor'": 1,
             'released,': 55,
             'causing': 67,
             'riot': 33,
             'paris': 160,
             'screened': 28,
             'there,resulting': 1,
             'banned': 56,
             'something': 3061,

             'forty': 45,
             'years.': 347,
             ...})
```

```
LABEL2.vocab.freqs
```

```
Counter({'neg': 8810, 'pos': 8690})
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
torchtext_train_dataloader2, torchtext_valid_dataloader2, torchtext_test_dataloader2 = BucketIterator.s
        (train_Dataset2, valid_Dataset2, test_Dataset2),
        batch_sizes=(128, 256, 256),
        sort=False,
        sort_key= lambda x: len(x.text),
        sort_within_batch=True,
        device=device,
        repeat=False,
)
```

```
torchtext_train_dataloader2.create_batches()
torchtext_valid_dataloader2.create_batches()
torchtext_test_dataloader2.create_batches()
```

```
word_embeddings = TEXT2.vocab.vectors

kernel_sizes = [3, 4, 5]
vocab_size = len(TEXT2.vocab)
dropout = 0.5
dim = 300
```

```
cnn_model = CNN(vocab_size=vocab_size, emb_dim=dim, out_channels=64,
            kernel_sizes=kernel_sizes, dropout=dropout, out_channel=1)
```

```
cnn_model.embedding.weight = nn.Parameter(word_embeddings)
```

```
cnn_model.to(device)

opt = torch.optim.Adam(cnn_model.parameters())
loss_func = nn.BCEWithLogitsLoss()
```

Вы знаете, что делать.

```
import numpy as np

min_loss = np.inf

cur_patience = 0
max_epochs = 30
patience = 10

for epoch in range(1, max_epochs + 1):

    train_f1_score = 0.0
    train_loss = 0.0

    cnn_model.train()

    pbar = tqdm(enumerate(torchtext_train_dataloader2), total=len(torchtext_train_dataloader2), leave=F
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:
        #YOUR CODE GOES HERE
        text = batch.text
        y_true = batch.label

        opt.zero_grad()

        y_pred = cnn_model(text).squeeze() # (batch_size, 1)

        loss = loss_func(y_pred, y_true)
        loss.backward()
        opt.step()

        train_loss += loss.detach().cpu().item()
        train_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().nu
    train_loss /= len(torchtext_train_dataloader2)
    train_f1_score /= len(torchtext_train_dataloader2)

    val_f1_score = 0.0
    val_loss = 0.0

    cnn_model.eval()

    pbar = tqdm(enumerate(torchtext_valid_dataloader2), total=len(torchtext_valid_dataloader2), leave=F
    pbar.set_description(f"Epoch {epoch}")

    for it, batch in pbar:
        # YOUR CODE GOES HERE
        with torch.no_grad():
            text = batch.text
            y_true = batch.label

            y_pred = cnn_model(text).squeeze()

            loss = loss_func(y_pred, y_true)
```

```
            val_loss += loss.cpu().item()
            val_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().
    val_loss /= len(torchtext_valid_dataloader2)
    val_f1_score /= len(torchtext_valid_dataloader2)

    if val_loss < min_loss:
        min_loss = val_loss
        best_model = cnn_model.state_dict()
    else:
        cur_patience += 1
        if cur_patience == patience:
            cur_patience = 0
            break

    print('Epoch: {}, Training Loss: {}, Validation Loss: {}'.format(epoch, train_loss, val_loss))
    print("---------  Training f1_score: {}, Validation f1_score: {}".format(train_f1_score, val_f1_sco

cnn_model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.5170622289615826, Validation Loss: 0.37479890485604606
---------  Training f1_score: 0.7227050853277115, Validation f1_score: 0.8484763487973538
Epoch: 2, Training Loss: 0.3238445248482001, Validation Loss: 0.3088692535956701
---------  Training f1_score: 0.8661820597874371, Validation f1_score: 0.876596899054871
Epoch: 3, Training Loss: 0.21121851871483519, Validation Loss: 0.29524263391892114
---------  Training f1_score: 0.9198239274432632, Validation f1_score: 0.8802437915572958
Epoch: 4, Training Loss: 0.10931290079751153, Validation Loss: 0.31057442476352054
---------  Training f1_score: 0.9653082993357345, Validation f1_score: 0.8824474816964846
Epoch: 5, Training Loss: 0.04905268771533113, Validation Loss: 0.33222740491231284
---------  Training f1_score: 0.9892060025186922, Validation f1_score: 0.8743631561518297
Epoch: 6, Training Loss: 0.020829322877047706, Validation Loss: 0.35986773669719696
---------  Training f1_score: 0.9968369780339038, Validation f1_score: 0.8717647277184574
Epoch: 7, Training Loss: 0.010444752244877011, Validation Loss: 0.3876693914333979
---------  Training f1_score: 0.9989760174572805, Validation f1_score: 0.8726698792858613
Epoch: 8, Training Loss: 0.005724110318531357, Validation Loss: 0.40828407953182855
---------  Training f1_score: 0.9995736723331387, Validation f1_score: 0.8732805739435051
Epoch: 9, Training Loss: 0.003957570732713942, Validation Loss: 0.42761072764794034
---------  Training f1_score: 0.999674163872666, Validation f1_score: 0.8715067169708058
Epoch: 10, Training Loss: 0.002548571434262868, Validation Loss: 0.4448671688636144
---------  Training f1_score: 0.9999425254324962, Validation f1_score: 0.8726647988769772
Epoch: 11, Training Loss: 0.0023080487365929585, Validation Loss: 0.46146749953428906
---------  Training f1_score: 0.9998256164035344, Validation f1_score: 0.8679611088681931
Epoch: 12, Training Loss: 0.0018969821560121801, Validation Loss: 0.4754878282546997
---------  Training f1_score: 0.9998961298803763, Validation f1_score: 0.8699650309897676
<All keys matched successfully>
```

```
test_loss = 0.0
test_f1_score = 0.0

cnn_model.eval()

pbar = tqdm(enumerate(torchtext_test_dataloader2), total=len(torchtext_test_dataloader2), leave=False)

for it, batch in pbar:
    with torch.no_grad():
        text = batch.text
        y_true = batch.label

        y_pred = cnn_model(text).squeeze()

        loss = loss_func(y_pred, y_true)

        test_loss += loss.cpu().item()
        test_f1_score += f1_score(y_true.cpu().numpy(), (torch.sigmoid(y_pred).cpu() > 0.5).float().nur
test_loss /= len(torchtext_test_dataloader2)
```

```
test_f1_score /= len(torchtext_test_dataloader2)

print("Testing Loss: {}".format(test_loss))
print("Testing f1_score: {}".format(test_f1_score))
```

```
Testing Loss: 0.47554718201257745
Testing f1_score: 0.8591374029459167
```

Посчитайте f1-score вашего классификатора.

Ответ:

```
# Testing f1_score: 0.8591374029459167

# как можно заметить инициализация предобученными эмбеддингами улучшает качество предсказания модели
```

Проверим насколько все хорошо!

```
PAD_IND = TEXT2.vocab.stoi['pad']

token_reference = TokenReferenceBase(reference_token_idx=PAD_IND)
lig = LayerIntegratedGradients(cnn_model, cnn_model.embedding)
vis_data_records_ig = []

interpret_sentence(cnn_model, 'It was a fantastic performance !', label=1)
interpret_sentence(cnn_model, 'Best film ever', label=1)
interpret_sentence(cnn_model, 'Such a great show!', label=1)
interpret_sentence(cnn_model, 'It was a horrible movie', label=0)
interpret_sentence(cnn_model, 'I\'ve never watched something as bad', label=0)
interpret_sentence(cnn_model, 'It is a disgusting movie!', label=0)
```

```
pred:  pos ( 0.98 ) , delta:  tensor([0.0003], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([3.9156e-05], device='cuda:0', dtype=torch.float64)
pred:  pos ( 0.78 ) , delta:  tensor([4.6364e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([6.9329e-06], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.22 ) , delta:  tensor([7.0864e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([7.4362e-05], device='cuda:0', dtype=torch.float64)
```

```
print('Visualize attributions based on Integrated Gradients')
visualization.visualize_text(vis_data_records_ig)
```

```
Visualize attributions based on Integrated Gradients
```

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	**pos (0.98)**	**pos**	**1.66**	It was a fantastic performance ! pad
pos	**neg (0.00)**	**pos**	**1.32**	Best film ever pad pad pad pad
neg	**neg (0.00)**	**pos**	**-0.19**	It was a horrible movie pad pad
neg	**neg (0.22)**	**pos**	**1.44**	I've never watched something as bad pad
neg	**neg (0.00)**	**pos**	**-0.29**	It is a disgusting movie! pad pad

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	**pos (0.98)**	**pos**	**1.66**	It was a fantastic performance ! pad
pos	**neg (0.00)**	**pos**	**1.32**	Best film ever pad pad pad pad
pos	**pos (0.78)**	**pos**	**1.44**	Such a great show! pad pad pad
neg	**neg (0.00)**	**pos**	**-0.19**	It was a horrible movie pad pad
neg	**neg (0.22)**	**pos**	**1.44**	I've never watched something as bad pad
neg	**neg (0.00)**	**pos**	**-0.29**	It is a disgusting movie! pad pad

✓ 0 сек. выполнено в 19:52