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 import datasets

from torchtext.legacy import datasets
from torchtext.legacy.data import Field, LabelField, 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
```

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
LABEL = LabelField(dtype=torch.float)  # Поле метки
```

```
SEED = 0xDEAD

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

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

```
train, test = datasets.IMDB.splits(TEXT, LABEL)  # загрузим датасет
train, valid = train.split(random_state=random.seed(SEED))  # разобьем на части
```

```
downloading aclImdb_v1.tar.gz
aclImdb_v1.tar.gz: 100%|██████████| 84.1M/84.1M [00:03<00:00, 24.5MB/s]
```

```
TEXT.build_vocab(train)
LABEL.build_vocab(train)
```

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

train_iter, valid_iter, test_iter = BucketIterator.splits(
    (train, valid, test),
    batch_size = 64,
    sort_within_batch = True,
    device = device)
```

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.

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

```
        super().__init__()

        self.bidirectional = bidirectional
        self.dropout = dropout

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

        self.rnn = nn.LSTM(input_size=embedding_dim, hidden_size=hidden_dim,
                           num_layers=n_layers, dropout=dropout, bidirectional=bidirect

        # self.rnn = nn.GRU(input_size=embedding_dim, hidden_size=hidden_dim,
        #                     num_layers=n_layers, dropout=dropout, bidirectional=bidire
        if self.bidirectional:
            self.fc = nn.Linear(2 * hidden_dim, output_dim)  # YOUR CODE GOES HERE
        else:
            self.fc = nn.Linear(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.cpu(

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

        # for gru
        # packed_output, hidden= self.rnn(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
        #and apply dropout
        if self.bidirectional:
            hidden = torch.cat([hidden[-2,:,:], hidden[-1,:,:]], dim=1)  # YOUR CODE GO
        else:
            hidden = hidden[-1,:,:]

        #hidden = [batch size, hid dim * num directions] or [batch_size, hid dim * num
        hidden = nn.Dropout(p=self.dropout)(hidden)

        return self.fc(hidden)
```

Поиграйтесь с гиперпараметрами

```
vocab_size = len(TEXT.vocab)
emb_dim = 100
hidden_dim = 256
output_dim = 1
n_layers = 2
bidirectional = False
dropout = 0
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=3
```

```
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(reduction='sum', )

max_epochs = 20
```

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

```
%time
import numpy as np

min_loss = np.inf

cur_patience = 0

for epoch in range(1, max_epochs + 1):
    train_loss = 0.0
    model.train()
    pbar = tqdm(enumerate(train_iter), total=len(train_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        #YOUR CODE GOES HERE
        opt.zero_grad()
        output = model(batch.text[0].to(device), batch.text[1].to(device))
```

```
        loss = loss_func(output.squeeze(1), batch.label)
        loss.backward()
        train_loss += loss.item()
        opt.step()

    train_loss /= len(train_iter)
    val_loss = 0.0
    model.eval()
    pbar = tqdm(enumerate(valid_iter), total=len(valid_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        # YOUR CODE GOES HERE
        output = model(batch.text[0], batch.text[1])
        val_loss = loss_func(output.squeeze(1), batch.label).item()


    val_loss /= len(valid_iter)
    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,
model.load_state_dict(best_model)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.25 µs
Epoch: 1, Training Loss: 43.59233463593643, Validation Loss: 0.07081056853472176
Epoch: 2, Training Loss: 40.25798383420401, Validation Loss: 0.06330045603089414
Epoch: 3, Training Loss: 30.986384597137896, Validation Loss: 0.057714442075309104
Epoch: 4, Training Loss: 20.135748187990956, Validation Loss: 0.016463057469513456
Epoch: 5, Training Loss: 12.470059507084589, Validation Loss: 0.007022834935430753
Epoch: 6, Training Loss: 7.142795572968295, Validation Loss: 0.0020590225013635928
Epoch: 7, Training Loss: 7.073025200706329, Validation Loss: 0.010574527716232558
Epoch: 8, Training Loss: 6.931653290116874, Validation Loss: 0.0014015540985737817
Epoch: 9, Training Loss: 1.879023780053767, Validation Loss: 0.004321309974638082
Epoch: 10, Training Loss: 0.8456091671692629, Validation Loss: 0.0001896094783382900
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора на тестовом датасете.

Ответ:

```
pred_labels = []
true_labels = []
for batch in test_iter:
    pred_labels_batch = list((model(batch.text[0],batch.text[1]) > 0.5).float().cpu().r
    true_labels_batch = list(batch.label.cpu())

```

```
    pred_labels += pred_labels_batch
    true_labels += true_labels_batch
```

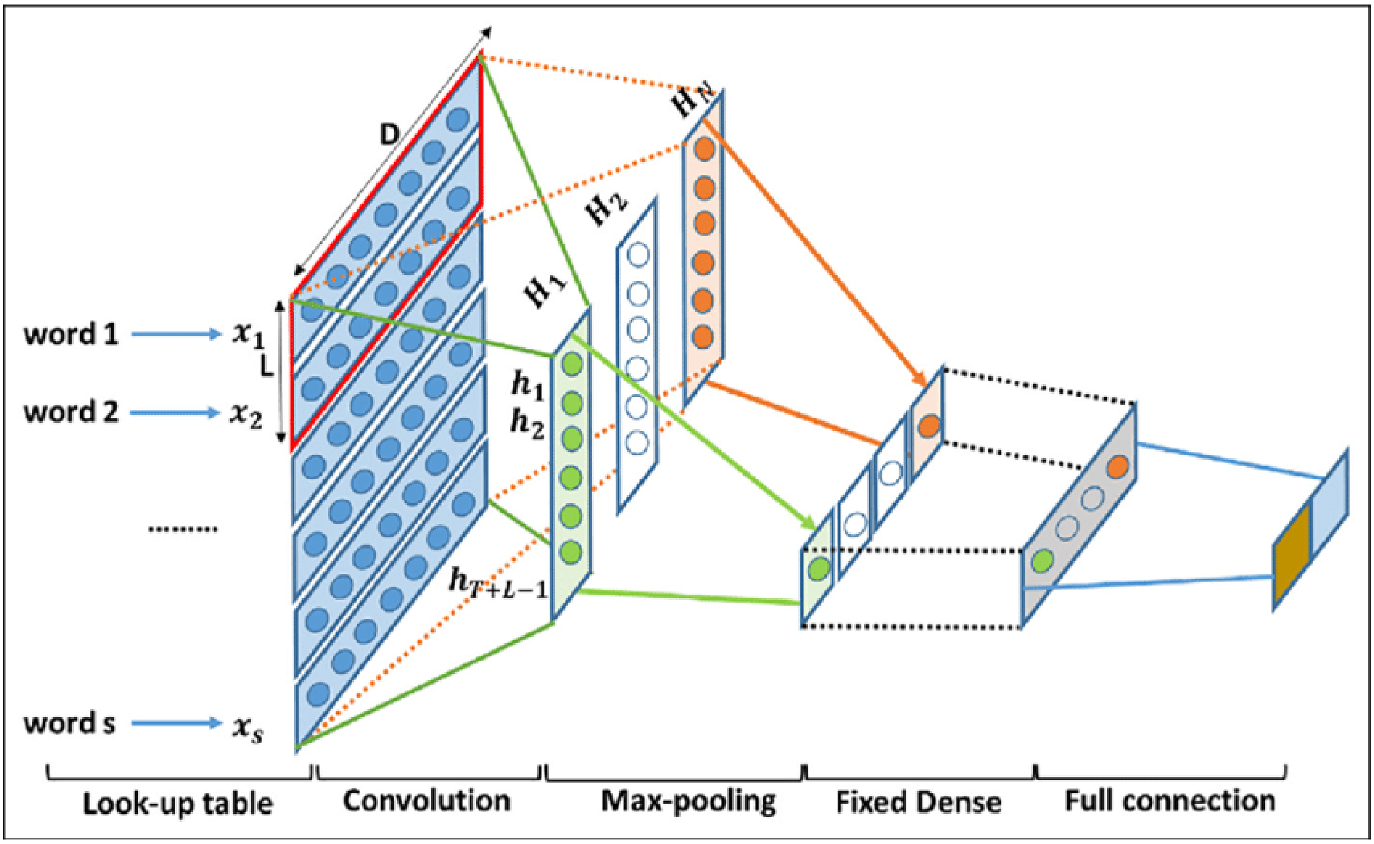
```
from sklearn.metrics import f1_score

f1_score(true_labels, pred_labels)
```

```
0.7941511251203922
```

lstm with bi - 0.8318078698271141
lstm without bi - 0.7941511251203922`
gru with bi 0.8255858138062064
gru without bi 0.5913393537155914

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 исполь
LABEL = LabelField(batch_first=True, dtype=torch.float)

train, tst = datasets.IMDB.splits(TEXT, LABEL)
```

```
trn, vld = train.split(random_state=random.seed(SEED))

TEXT.build_vocab(trn)
LABEL.build_vocab(trn)

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

```
train_iter, val_iter, test_iter = BucketIterator.splits(
        (trn, vld, tst),
        batch_sizes=(128, 256, 256),
        sort=False,
        sort_key= lambda x: len(x.src),
        sort_within_batch=False,
        device=device,
        repeat=False,
)
```

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.

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

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

        self.conv_0 = nn.Conv1d(emb_dim, out_channels, kernel_size=kernel_sizes[0], pad

        self.conv_1 = nn.Conv1d(emb_dim, out_channels, kernel_size=kernel_sizes[1], pad

        self.conv_2 = nn.Conv1d(emb_dim, out_channels, kernel_size=kernel_sizes[2], pad

        self.fc = nn.Linear(len(kernel_sizes) * out_channels, 1)

        self.dropout = nn.Dropout(dropout)


    def forward(self, text):

        embedded = self.embedding(text)

        embedded = embedded.permute(0, 2, 1)  # may be reshape here

```

```
        conved_0 = F.relu(self.conv_0(embedded))  # may be reshape here
        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)
```

```
kernel_sizes = [3, 4, 5]
vocab_size = len(TEXT.vocab)
out_channels=64
dropout = 0.5
dim = 300

model = CNN(vocab_size=vocab_size, emb_dim=dim, out_channels=out_channels,
            kernel_sizes=kernel_sizes, dropout=dropout)
```

```
model.to(device)
```

```
CNN(
  (embedding): Embedding(201630, 300)
  (conv_0): Conv1d(300, 64, kernel_size=(3,), stride=(1,))
  (conv_1): Conv1d(300, 64, kernel_size=(4,), stride=(1,))
  (conv_2): Conv1d(300, 64, kernel_size=(5,), stride=(1,))
  (fc): Linear(in_features=192, out_features=1, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

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

```
max_epochs = 30
```

Обучите!

```
min_loss = np.inf

cur_patience = 0

for epoch in range(1, max_epochs + 1):
    train_loss = 0.0
    model.train()
    pbar = tqdm(enumerate(train_iter), total=len(train_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        #YOUR CODE GOES HERE
        opt.zero_grad()
        output = model(batch.text)
```

```
        loss = loss_func(output.squeeze(1), batch.label)
        loss.backward()
        train_loss += loss.item()
        opt.step()

    train_loss /= len(train_iter)
    val_loss = 0.0
    model.eval()
    pbar = tqdm(enumerate(val_iter), total=len(val_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        # YOUR CODE GOES HERE
        output = model(batch.text)
        val_loss += loss_func(output.squeeze(1), batch.label).item()

    val_loss /= len(val_iter)
    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,
model.load_state_dict(best_model)
```

```
Epoch 1: 100%                                    137/137 [00:39<00:00, 3.55it/s]

Epoch 1: 97%                                     29/30 [00:07<00:00, 3.90it/s]
Epoch: 1, Training Loss: 0.5067113494350962, Validation Loss: 0.42657203574975333
Epoch 2: 100%                                    137/137 [00:38<00:00, 3.56it/s]

Epoch 2: 97%                                     29/30 [00:07<00:00, 3.88it/s]
Epoch: 2, Training Loss: 0.43463385213900657, Validation Loss: 0.3943417410055796
Epoch 3: 100%                                    137/137 [00:39<00:00, 3.82it/s]

Epoch 3: 97%                                     29/30 [00:07<00:00, 3.90it/s]
Epoch: 3, Training Loss: 0.3772826199113888, Validation Loss: 0.36494067311286926
```

Посчитайте f1-score вашего классификатора.

Ответ: 0.8473694279701549

```
pred_labels = []
true_labels = []
for batch in test_iter:
    pred_labels_batch = list((model(batch.text) > 0.5).float().cpu().numpy().reshape(-1
    true_labels_batch = list(batch.label.cpu().numpy())

    pred_labels += pred_labels_batch
    true_labels += true_labels_batch
f1_score(true_labels, pred_labels)
```

```
0.8473694279701549
Epoch: 7, Training Loss: 0.14652205117209985, Validation Loss: 0.37516842186450 96
```

Интерпретируемость

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

```
Epoch 9: 100%                                    137/137 [00:38<00:00, 3.4 it/s]
```

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

```
|████████████████████████████████| 1.4 MB 5.3 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(model, 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(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).item()
    pred_ind = round(pred)

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

    # 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

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

def add_attributions_to_visualizer(attributions, text, pred, pred_ind, label, delta, vi
    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(model, 'It was a fantastic performance !', label=1)
interpret_sentence(model, 'Best film ever', label=1)
```

```
interpret_sentence(model, 'Such a great show!', label=1)
interpret_sentence(model, 'It was a horrible movie', label=0)
interpret_sentence(model, 'I\'ve never watched something as bad', label=0)
interpret_sentence(model, 'It is a disgusting movie!', label=0)
interpret_sentence(model, 'an awfully good movie!', label=1)
interpret_sentence(model, 'movie is so good that I slept', label=0)
```

```
pred:  pos ( 0.95 ) , delta:  tensor([3.5385e-05], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.02 ) , delta:  tensor([6.6953e-08], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.39 ) , delta:  tensor([2.9517e-05], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.00 ) , delta:  tensor([4.1141e-05], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.17 ) , delta:  tensor([6.1732e-05], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.09 ) , delta:  tensor([2.6798e-05], device='cuda:0', dtype=torch.floa
pred:  neg ( 0.02 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  pos ( 0.61 ) , delta:  tensor([1.2505e-05], device='cuda:0', dtype=torch.floa
```

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 (0.95)	pos	1.64	It was a fantastic performance ! pad
pos	neg (0.02)	pos	1.58	Best film ever pad pad pad pad
pos	neg (0.39)	pos	0.82	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.27	It was a horrible movie pad pad
neg	neg (0.17)	pos	0.81	I've never watched something as bad pad
neg	neg (0.09)	pos	0.74	It is a disgusting movie! pad pad
pos	pos (0.64)	pos	1.69	Ужасно хороший фильм! pad pad pad pad
pos	neg (0.02)	pos	0.11	an awfully good movie! pad pad pad
pos	pos (0.95)	pos	1.64	It was a fantastic performance ! pad
pos	neg (0.02)	pos	1.58	Best film ever pad pad pad pad
pos	neg (0.39)	pos	0.82	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.27	It was a horrible movie pad pad
neg	neg (0.17)	pos	0.81	I've never watched something as bad pad
neg	neg (0.09)	pos	0.74	It is a disgusting movie! pad pad
pos	neg (0.02)	pos	0.11	an awfully good movie! pad pad pad
neg	neg (0.00)	pos	-0.61	Incredibly bad movie pad pad pad pad
neg	pos (0.67)	pos	2.35	movie is so good that I sleept
neg	pos (0.61)	pos	1.77	movie is so good that I slept
pos	pos (0.95)	pos	1.64	It was a fantastic performance ! pad
pos	neg (0.02)	pos	1.58	Best film ever pad pad pad pad
pos	neg (0.39)	pos	0.82	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.27	It was a horrible movie pad pad
neg	neg (0.17)	pos	0.81	I've never watched something as bad pad
neg	neg (0.09)	pos	0.74	It is a disgusting movie! pad pad
pos	neg (0.02)	pos	0.11	an awfully good movie! pad pad pad
neg	pos (0.61)	pos	1.77	movie is so good that I slept

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Predicted Attribution Attribution

Эмбеддинги слов

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

neg neg (0.00) pos -0.27 It was a horrible movie pad pad

```
TEXT.build_vocab(trn, vectors=GloVe())# YOUR CODE GOES HERE
# подсказка: один из импортов пока не использовался, быть может он нужен в строке выше
LABEL.build_vocab(trn)

word_embeddings = TEXT.vocab.vectors

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

```
.vector_cache/glove.840B.300d.zip: 2.18GB [07:00, 5.18MB/s]
100%|██████████| 2196016/2196017 [05:08<00:00, 7129.39it/s]
```

neg neg (0.00) pos -0.27 It was a horrible movie pad pad

```
train, tst = datasets.IMDB.splits(TEXT, LABEL)
trn, vld = train.split(random_state=random.seed(SEED))

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


train_iter, val_iter, test_iter = BucketIterator.splits(
        (trn, vld, tst),
        batch_sizes=(128, 256, 256),
        sort=False,
        sort_key= lambda x: len(x.src),
        sort_within_batch=False,
        device=device,
        repeat=False,
)
```

```
model = CNN(vocab_size=vocab_size, emb_dim=dim, out_channels=64,
            kernel_sizes=kernel_sizes, dropout=dropout)

word_embeddings = TEXT.vocab.vectors

prev_shape = model.embedding.weight.shape

model.embedding.weight.data.copy_(word_embeddings) # инициализируйте эмбэдинги

assert prev_shape == model.embedding.weight.shape
model.to(device)

opt = torch.optim.Adam(model.parameters())
```

Вы знаете, что делать.

```python
import numpy as np

min_loss = np.inf

cur_patience = 0

for epoch in range(1, max_epochs + 1):
    train_loss = 0.0
    model.train()
    pbar = tqdm(enumerate(train_iter), total=len(train_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        #YOUR CODE GOES HERE
        opt.zero_grad()
        output = model(batch.text)
        loss = loss_func(output.squeeze(1), batch.label)
        loss.backward()
        train_loss += loss.item()
        opt.step()

    train_loss /= len(train_iter)
    val_loss = 0.0
    model.eval()
    pbar = tqdm(enumerate(val_iter), total=len(valid_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
        # YOUR CODE GOES HERE
        output = model(batch.text)
        val_loss += loss_func(output.squeeze(1), batch.label).item()

    val_loss /= len(val_iter)
    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,
model.load_state_dict(best_model)
```

```
Epoch 1: 100%                                        137/137 [00:39<00:00, 3.30it/s]

Epoch 1: 25%                                         30/118 [00:07<00:23, 3.80it/s]
Epoch: 1, Training Loss: 0.30204641536204485, Validation Loss: 0.2985475242137909
Epoch 2: 100%                                        137/137 [00:39<00:00, 3.53it/s]

Epoch 2: 25%                                         30/118 [00:07<00:22, 3.86it/s]
Epoch: 2, Training Loss: 0.1839325635211311, Validation Loss: 0.28009043584267296
Epoch 3: 100%                                        137/137 [00:38<00:00, 3.58it/s]

Epoch 3: 25%                                         30/118 [00:07<00:22, 3.87it/s]
```

```
pred_labels = []
true_labels = []
for batch in test_iter:
    pred_labels_batch = list((model(batch.text) > 0.5).float().cpu().numpy().reshape(-1
    true_labels_batch = list(batch.label.cpu().numpy())

    pred_labels += pred_labels_batch
    true_labels += true_labels_batch
f1_score(true_labels, pred_labels)
```

```
0.8653351304987135
```

Посчитайте f1-score вашего классификатора.

Ответ: 0.8653351304987135

Проверим насколько все хорошо!

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

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

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

```
pred:  pos ( 0.92 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.16 ) , delta:  tensor([8.6935e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.37 ) , delta:  tensor([2.6935e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.00 ) , delta:  tensor([1.0840e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.50 ) , delta:  tensor([4.0545e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.00 ) , delta:  tensor([1.2201e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.01 ) , delta:  tensor([4.0200e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 0.56 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
```

```python
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 (0.92)	pos	1.85	It was a fantastic performance ! pad
pos	neg (0.16)	pos	1.25	Best film ever pad pad pad pad
pos	neg (0.37)	pos	1.54	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.46	It was a horrible movie pad pad
neg	neg (0.50)	pos	1.67	I've never watched something as bad pad
neg	neg (0.00)	pos	-0.91	It is a disgusting movie! pad pad
pos	neg (0.01)	pos	-0.19	an awfully good movie! pad pad pad
neg	pos (0.56)	pos	1.80	movie is so good that I slept

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.92)	pos	1.85	It was a fantastic performance ! pad
pos	neg (0.16)	pos	1.25	Best film ever pad pad pad pad
pos	neg (0.37)	pos	1.54	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.46	It was a horrible movie pad pad
neg	neg (0.50)	pos	1.67	I've never watched something as bad pad

0s completed at 2:39 PM