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.

In [46]:
```
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

from sklearn.metrics import f1_score
```

In [ ]:
```
SEED = 1234

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

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

In [87]:
```
TEXT = Field(sequential=True, lower=True, include_lengths=True)  # Поле текста
LABEL = LabelField(dtype=torch.float)  # Поле метки
```

Датасет, на котором мы будем проводить эксперименты - это комментарии к фильмам из сайта IMDB.

In [88]:
```
train, test = datasets.IMDB.splits(TEXT, LABEL)  # загрузим датасет
train, valid = train.split(random_state=random.seed(SEED))  # разобьем на части
```

In [89]:
```
TEXT.build_vocab(train)
LABEL.build_vocab(train)
```

In [90]:
```
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)
```

Посмотрим, что выдает итератор

In [51]:
```
(text, text_lengths), label = next(iter(train_iter))
```

In [52]:
```
text, text.shape
```

Out[52]:
```
(tensor([[  5821,   4376,      2,  ...,      9,      9,      2],
         [191423,    270,    180,  ...,    275,    375,     77],
         [     8,      2,      5,  ...,     10,     10,    438],
         ...,
         [  1205,    207,    248,  ...,  48564,     29,    197],
         [    17,     19,   1018,  ...,  48815,  59606,  19026],
         [   487,    672,  33469,  ...,      1,      1,      1]],
        device='cuda:0'), torch.Size([114, 64]))
```

In [53]:
```
text_lengths, text_lengths.shape
```

Out[53]:
```
(tensor([114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114,
         114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 114,
         114, 114, 114, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113,
         113, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113, 113,
         113, 113, 113, 113, 113, 113, 113, 113], device='cuda:0'),
 torch.Size([64]))
```

In [54]:
```
label, label.shape
```

Out[54]:
```
(tensor([1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 1.,
         0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 0., 1., 0., 1.,
         0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0.,
         0., 1., 1., 1., 1., 0., 1., 1., 1., 0.], device='cuda:0'),
 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.

In [55]:
```
class RNNBaseline(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_layers,
                 bidirectional, dropout, pad_idx):

        super().__init__()
        self.num_directions = bidirectional + 1
        self.embedding = nn.Embedding(vocab_size, embedding_dim,
                                      padding_idx = pad_idx).to(device)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=dropout,
```

```
                              bidirectional=bidirectional)  # YOUR CODE GOES HERE
        self.fc = nn.Linear(hidden_dim * self.num_directions, output_dim)  # YOUR CODE GOES HERE
        self.dropout = nn.Dropout(dropout)


    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)
        #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) if self.num_directions == 2 else hidden[-1]  # YOUR CODE GOES
        hidden = self.dropout(hidden)


        #hidden = [batch size, hid dim * num directions] or [batch_size, hid dim * num directions]

        return self.fc(hidden).flatten()
```

Поиграйтесь с гиперпараметрами

In [74]:

```
vocab_size = len(TEXT.vocab)
emb_dim = 300
hidden_dim = 256
output_dim = 1
n_layers = 4
bidirectional = True
```

```
dropout = 0.2
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=3
# my custom params
max_grad_norm = 0.1
```

In [75]:
```
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
)
```

In [76]:
```
model = model.to(device)
```

In [77]:
```
opt = torch.optim.Adam(model.parameters())
loss_func = nn.BCEWithLogitsLoss()

max_epochs = 20
```

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

In [78]:
```
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()
        (text, text_lengths), label = batch
        predict = model(text, text_lengths)
        loss = loss_func(predict, label)
        loss.backward()
```

```
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        opt.step()
        train_loss += loss.item()

    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
        (text, text_lengths), label = batch
        predict = model(text, text_lengths)
        loss = loss_func(predict, label)
        val_loss += loss.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, val_loss))
model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.6577718360145597, Validation Loss: 0.6504536700450768

Epoch: 2, Training Loss: 0.5587749980444455, Validation Loss: 0.5076152240320787

Epoch: 3, Training Loss: 0.3531457634411589, Validation Loss: 0.38660544460102664

Epoch: 4, Training Loss: 0.1946592913111196, Validation Loss: 0.4133057494537305

Epoch: 5, Training Loss: 0.08980402345582163, Validation Loss: 0.4782852578466221
```

Out[78]:
```
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора на тестовом датасете.

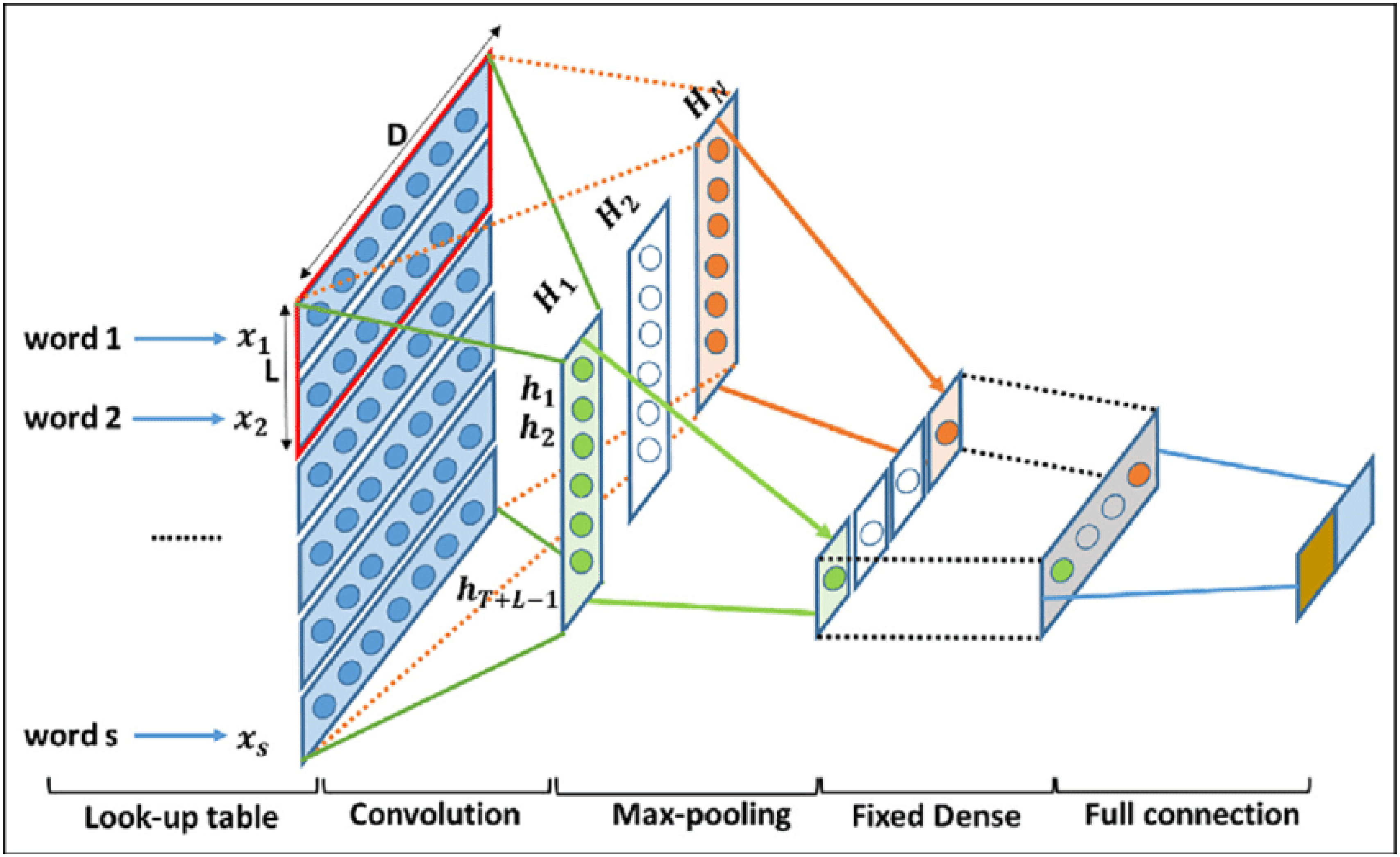
Ответ:

In [79]:
```
model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)

predict_labels = []
labels = []
for it, batch in pbar:
    (text, text_lengths), label = batch
    predict_label = model(text, text_lengths) > 0
    predict_labels.extend(predict_label.tolist())
    labels.extend(label.tolist())
f1 = f1_score(labels, predict_labels)
print(f"f1_score = {f1:.4}")
```

```
f1_score = 0.844
```

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!

In [100…

```
TEXT = Field(sequential=True, lower=True, batch_first=True)  # batch_first тк мы используем conv
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"
```

In [101…

```
train_iter, valid_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,
)
```

Посмотрим на итератор

In [82]:

```
text, label = next(iter(valid_iter))
```

In [ ]:

```
text.shape
```

Out[ ]:

```
torch.Size([256, 1057])
```

In [ ]:

```
label.shape
```

Out[ ]:

```
torch.Size([256])
```

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.

In [103…

```
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])  # YOUR CODE GOES HERE
        self.conv_1 = nn.Conv1d(emb_dim, out_channels, kernel_size=kernel_sizes[1])  # YOUR CODE GOES HERE
        self.conv_2 = nn.Conv1d(emb_dim, out_channels, kernel_size=kernel_sizes[2])  # YOUR CODE GOES HERE
        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).flatten()
```

In [134…

```
kernel_sizes = [3, 4, 5]
vocab_size = len(TEXT.vocab)
out_channels = 64
dropout = 0.2
dim = 300
patience = 3
# my custom params
max_grad_norm = 1

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

In [135…

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

Out[135…

```
CNN(
  (embedding): Embedding(202268, 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=0.2, inplace=False)
)
```

In [136…

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

In [137…

```
max_epochs = 30
```

Обучите!

In [138…

```
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()
        text, label = batch
        predict = model(text)
        loss = loss_func(predict, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        opt.step()
        train_loss += loss.item()

    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
        text, label = batch
        predict = model(text)
        loss = loss_func(predict, label)
        val_loss += loss.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, val_loss))
model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.571613120119067, Validation Loss: 0.44881325562795005

Epoch: 2, Training Loss: 0.40739470785551696, Validation Loss: 0.3937748700380325

Epoch: 3, Training Loss: 0.3151943307288372, Validation Loss: 0.3639407426118851

Epoch: 4, Training Loss: 0.23513055391555285, Validation Loss: 0.34165724913279216

Epoch: 5, Training Loss: 0.1731998088912372, Validation Loss: 0.33317116151253384

Epoch: 6, Training Loss: 0.12281479493436152, Validation Loss: 0.3312919095158577

Epoch: 7, Training Loss: 0.08288537694589936, Validation Loss: 0.36669970701138177

Epoch: 8, Training Loss: 0.05695953304423903, Validation Loss: 0.35555475354194643
```

Out[138…

```
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора.

Ответ:

In [139…

```
model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)

predict_labels = []
labels = []
for it, batch in pbar:
    text, label = batch
```

```
    predict_label = model(text) > 0
    predict_labels.extend(predict_label.tolist())
    labels.extend(label.tolist())
f1 = f1_score(labels, predict_labels)
print(f"f1_score = {f1:.4}")
```

```
f1_score = 0.8697
```

Интерпретируемость

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

In [ ]:
```
!pip install -q captum
```

```
|████████████████████████████████| 4.4MB 5.9MB/s
```

In [140…
```
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)
```

In [141…
```
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).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_records_ig)

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))
```

In [142…

```
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)
```

```
pred:  pos ( 1.00 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.01 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  pos ( 1.00 ) , delta:  tensor([4.7495e-05], device='cuda:0', dtype=torch.float64)
```

```
pred:  neg ( 0.02 ) , delta:  tensor([9.3134e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.32 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.41 ) , delta:  tensor([6.5641e-05], device='cuda:0', dtype=torch.float64)
```

Попробуйте добавить свои примеры!

In [143…

```
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 (1.00)**	**pos**	**1.45**	It was a fantastic performance ! pad
pos	**neg (0.01)**	**pos**	**-0.47**	Best film ever pad pad pad pad
pos	**pos (1.00)**	**pos**	**1.48**	Such a great show! pad pad pad
neg	**neg (0.02)**	**pos**	**-0.06**	It was a horrible movie pad pad
neg	**neg (0.32)**	**pos**	**0.70**	I've never watched something as bad pad
neg	**neg (0.41)**	**pos**	**0.46**	It is a disgusting movie! pad pad

Out[143…

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	**pos (1.00)**	**pos**	**1.45**	It was a fantastic performance ! pad
pos	**neg (0.01)**	**pos**	**-0.47**	Best film ever pad pad pad pad
pos	**pos (1.00)**	**pos**	**1.48**	Such a great show! pad pad pad
neg	**neg (0.02)**	**pos**	**-0.06**	It was a horrible movie pad pad
neg	**neg (0.32)**	**pos**	**0.70**	I've never watched something as bad pad
neg	**neg (0.41)**	**pos**	**0.46**	It is a disgusting movie! pad pad

Эмбэдинги слов

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

In [151…

```
kernel_sizes = [3, 4, 5]
dropout = 0.2
dim = 300
# my custom params
max_grad_norm = 2

TEXT.build_vocab(trn, vectors=GloVe(dim=dim))  # 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
vocab_size = len(TEXT.vocab)
```

In [152…

```
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, valid_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,
)
```

In [153…

```
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 = nn.Parameter(word_embeddings)  # инициализируйте эмбэдинги

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

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

Вы знаете, что делать.

In [154…

```
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()
        text, label = batch
        predict = model(text)
        loss = loss_func(predict, label)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
        opt.step()
        train_loss += loss.item()

    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
        text, label = batch
        predict = model(text)
        loss = loss_func(predict, label)
        val_loss += loss.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, val_loss))
model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.44054551672761455, Validation Loss: 0.32145943144957223

Epoch: 2, Training Loss: 0.24896514122068447, Validation Loss: 0.2785116364558538

Epoch: 3, Training Loss: 0.12991056284003885, Validation Loss: 0.27669415970643363

Epoch: 4, Training Loss: 0.049044410922449, Validation Loss: 0.2939944138129552

Epoch: 5, Training Loss: 0.015551697167085253, Validation Loss: 0.3237018610040347
```

Out[154…

```
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора.

Ответ:

In [155…

```
model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)

predict_labels = []
labels = []
for it, batch in pbar:
    text, label = batch
    predict_label = model(text) > 0
    predict_labels.extend(predict_label.tolist())
    labels.extend(label.tolist())
f1 = f1_score(labels, predict_labels)
print(f"f1_score = {f1:.4}")
```

```
f1_score = 0.8817
```

Проверим насколько все хорошо!

In [156…

```
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)
```

```
pred:  pos ( 0.98 ) , delta:  tensor([4.2085e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([5.2274e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.02 ) , delta:  tensor([6.0991e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([1.8015e-05], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.25 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([9.9167e-05], device='cuda:0', dtype=torch.float64)
```

In [157…

```
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	1.87	It was a fantastic performance ! pad
pos	neg (0.00)	pos	0.98	Best film ever pad pad pad pad
pos	neg (0.02)	pos	1.54	Such a great show! pad pad pad
neg	neg (0.00)	pos	0.51	It was a horrible movie pad pad
neg	neg (0.25)	pos	1.90	I've never watched something as bad pad
neg	neg (0.00)	pos	0.49	It is a disgusting movie! pad pad

Out[157…

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.98)	pos	1.87	It was a fantastic performance ! pad
pos	neg (0.00)	pos	0.98	Best film ever pad pad pad pad
pos	neg (0.02)	pos	1.54	Such a great show! pad pad pad
neg	neg (0.00)	pos	0.51	It was a horrible movie pad pad

neg	**neg (0.25)**	**pos**	**1.90**	I've never watched something as bad pad
neg	**neg (0.00)**	**pos**	**0.49**	It is a disgusting movie! pad pad