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.

```
!pip install torchtext==0.8.1
```

```
Collecting torchtext==0.8.1
  Downloading https://files.pythonhosted.org/packages/13/80/046f0691b296e755ae884df3
     |████████████████████████████████| 7.0MB 3.4MB/s
Collecting torch==1.7.1
  Downloading https://files.pythonhosted.org/packages/90/5d/095ddddc91c8a769a68c791c
     |████████████████████████████████| 776.8MB 23kB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from t
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
ERROR: torchvision 0.9.0+cu101 has requirement torch==1.8.0, but you'll have torch 1
Installing collected packages: torch, torchtext
  Found existing installation: torch 1.8.0+cu101
    Uninstalling torch-1.8.0+cu101:
      Successfully uninstalled torch-1.8.0+cu101
  Found existing installation: torchtext 0.9.0
    Uninstalling torchtext-0.9.0:
      Successfully uninstalled torchtext-0.9.0
Successfully installed torch-1.7.1 torchtext-0.8.1
```

```
!pip install torch==1.7.1+cu101 torchvision==0.8.2+cu101 torchaudio==0.7.2 -f
```

```
Usage:
  pip3 install [options] <requirement specifier> [package-index-options] ...
  pip3 install [options] -r <requirements file> [package-index-options] ...
  pip3 install [options] [-e] <vcs project url> ...
  pip3 install [options] [-e] <local project path> ...
  pip3 install [options] <archive url/path> ...

-f option requires 1 argument
```

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

from torchtext import datasets

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

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.

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

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/field.py:150: UserWarning: Fiel
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
/usr/local/lib/python3.7/dist-packages/torchtext/data/field.py:150: UserWarning: Labe
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
```

```
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, 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 [00:11<00:00, 7.59MB/s]
/usr/local/lib/python3.7/dist-packages/torchtext/data/example.py:78: UserWarning: Exa
  warnings.warn('Example class will be retired soon and moved to torchtext.legacy. Pl
```

```
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)
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/iterator.py:48: UserWarning: Bu
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
```

```
for it, batch in enumerate(train_iter):
    print(batch.text)
    print(batch.label)
    print(len(batch.text[0]))
    break
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/batch.py:23: UserWarning: Batch
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
(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'), tensor([114, 114, 114, 114, 114, 114, 114, 114, 114, 114, 11
        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'))
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')
114
```

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.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_

        self.fc = nn.Linear(n_layers*hidden_dim, 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)
        # print("packed_embedded=", packed_embedded.data.shape)


        # 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 la
        #and apply dropout

        hiddens = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), 1) #None  # YOUR CODE GOES H
        hiddens = self.dropout(hiddens)

        return self.fc(hiddens)
```

Поиграйтесь с гиперпараметрами

```
vocab_size = len(TEXT.vocab)
emb_dim = 100
hidden_dim = 256
output_dim = 1
n_layers = 2
bidirectional = True
dropout = 0.8
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=5


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()
loss_func1 = nn.MSELoss(reduction="sum")

max_epochs = 20


x = torch.tensor([[1, 2, 3, 4]])
torch.unsqueeze(x, 2)
```

```
     tensor([[[1],
              [2],
              [3],
              [4]]])
```

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

```
def list_pred(list_pred):
    a = []
    for i in range(len(list_pred)):
        if i > 0.5:
            a.append(1)
        else:
            a.append(0)

    return a


import numpy as np
from sklearn.metrics import accuracy_score

min_loss = np.inf

cur_patience = 0

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

    all_preds_train = []
    all_label_train = []
    all_preds_val = []
    all_label_val = []
    correct_train = 0
    correct_val = 0
    num_obj_train = 0
    num_obj_val = 0

    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:
        opt.zero_grad()
        emb = batch.text[0].to(device)
        len_emb = batch.text[1].cpu()
        label = torch.unsqueeze(batch.label, 1).to(device)
        preds = model(emb, len_emb)
        loss = loss_func(preds, label)
        loss.backward()
        train_loss += loss
        opt.step()

        predsr = np.where(preds.cpu() > 0, 1, 0)
        labelr = np.where(label.cpu() > 0, 1, 0)
        c = predsr == labelr
        correct_train += np.count_nonzero(c)
        num_obj_train += len(batch.label)

        trn = np.sum(predsr, axis=1)
        lbl = np.sum(labelr, axis=1)
        trn = trn.tolist()
        lbl = lbl.tolist()
        all_preds_train.append(trn)
```

```
        all_preds_train.append(trn)
        all_label_train.append(lbl)

        #YOUR CODE GOES HERE

    train_loss /= len(train_iter)
    val_loss = 0.0
    model.eval()
    with torch.no_grad():
      pbar = tqdm(enumerate(valid_iter), total=len(valid_iter), leave=False)
      pbar.set_description(f"Epoch {epoch}")
      for it, batch in pbar:
          emb = batch.text[0].to(device)
          len_emb = batch.text[1].cpu()
          label = torch.unsqueeze(batch.label, 1).to(device)
          preds = model(emb, len_emb)

          predsr = np.where(preds.cpu() > 0, 1, 0)
          labelr = np.where(label.cpu() > 0, 1, 0)
          c = predsr == labelr
          correct_val += np.count_nonzero(c)
          num_obj_val += len(batch.label)

          val_loss += loss_func(preds.data, label)

          val = np.sum(predsr, axis=1)
          lbl_val = np.sum(labelr, axis=1)
          val = val.tolist()
          lbl_val = lbl_val.tolist()
          all_preds_val.append(val)
          all_label_val.append(lbl_val)
          # YOUR CODE GOES HERE
      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: {}, Training accuracy: {}, Valid
model.load_state_dict(best_model)
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/batch.py:23: UserWarning: Batch
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
Epoch: 1, Training Loss: 0.6482466459274292, Validation Loss: 0.5980693101882935, Tra
Epoch: 2, Training Loss: 0.5436175465583801, Validation Loss: 0.5214125514030457, Tra
Epoch: 3, Training Loss: 0.5754403471946716, Validation Loss: 0.5855757594108582, Tra
Epoch: 4, Training Loss: 0.5563846230506897, Validation Loss: 0.6016130447387695, Tra
Epoch: 5, Training Loss: 0.3923629820346832, Validation Loss: 0.41213083267211914, Tr
Epoch: 6, Training Loss: 0.22027228772640228, Validation Loss: 0.4108792543411255, Tr
Epoch: 7, Training Loss: 0.11888790875673294, Validation Loss: 0.43305104970932007, T
Epoch: 8, Training Loss: 0.06524424254894257, Validation Loss: 0.5048354268074036, Tr
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора на тестовом датасете.

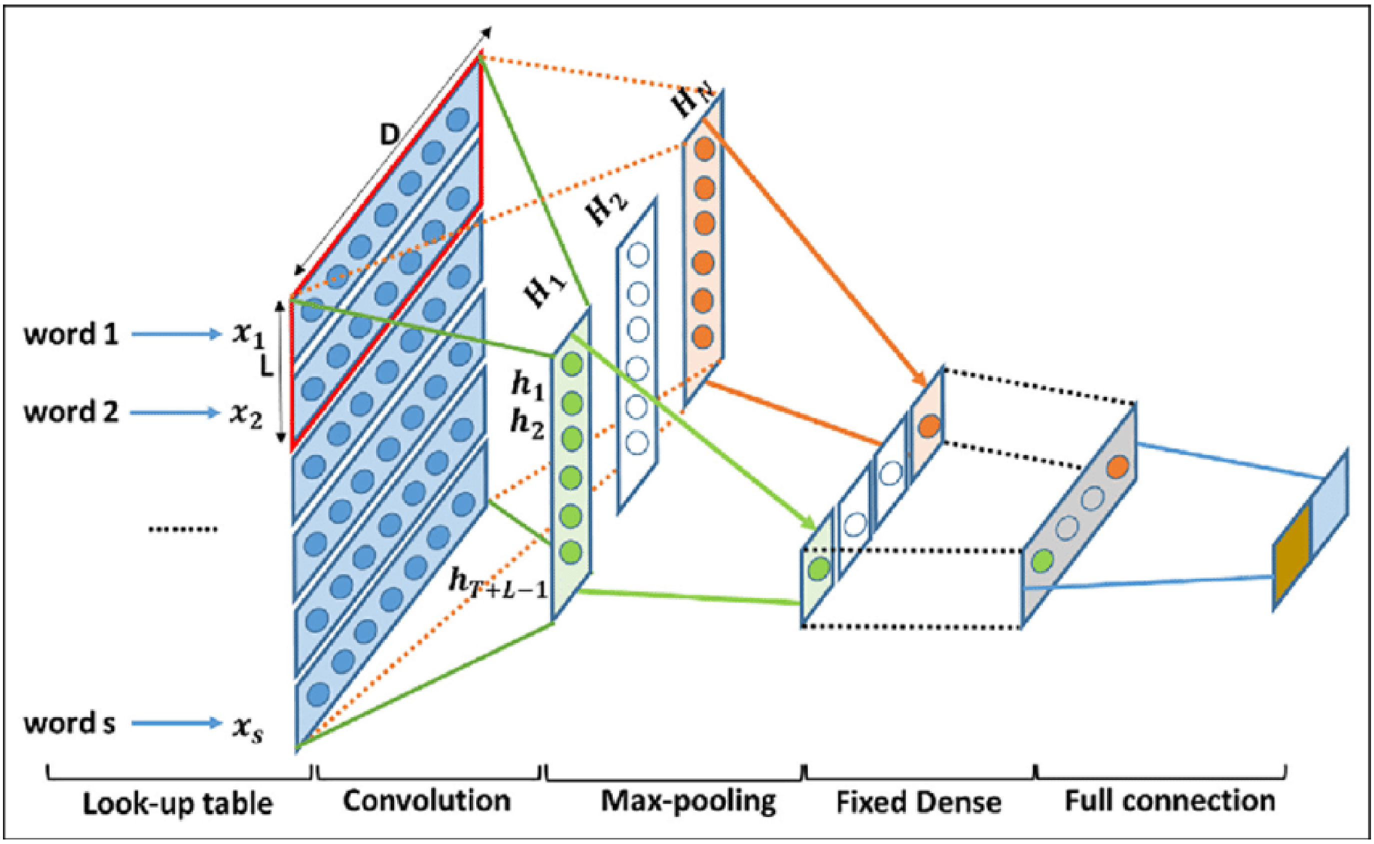
Ответ:

```
flat_preds_list = [item for sublist in all_preds_val for item in sublist]
flat_label_list = [item for sublist in all_label_val for item in sublist]


from sklearn.metrics import f1_score
f1_score(flat_label_list, flat_preds_list, average='weighted')

    0.8418184572607531
```

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 используе
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"
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/field.py:150: UserWarning: Fiel
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
/usr/local/lib/python3.7/dist-packages/torchtext/data/field.py:150: UserWarning: Labe
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
/usr/local/lib/python3.7/dist-packages/torchtext/data/example.py:78: UserWarning: Exa
  warnings.warn('Example class will be retired soon and moved to torchtext.legacy. Pl
```

```
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,
)
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/iterator.py:48: UserWarning: Bu
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
```

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], paddin

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

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

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

        self.dropout = nn.Dropout(dropout)
```

```
    def forward(self, text):

        embedded = self.embedding(text) # [batck_size, seq_len, embed_dim]
        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
patience=2

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(202268, 300)
  (conv_0): Conv1d(300, 64, kernel_size=(3,), stride=(2,), padding=(1,))
  (conv_1): Conv1d(300, 64, kernel_size=(4,), stride=(2,), padding=(1,))
  (conv_2): Conv1d(300, 64, kernel_size=(5,), stride=(2,), padding=(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!

```
import numpy as np

min_loss = np.inf

cur_patience = 0
```

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

    CNN_all_preds_train = []
    CNN_all_label_train = []
    CNN_all_preds_val = []
    CNN_all_label_val = []
    CNN_correct_train = 0
    CNN_correct_val = 0
    CNN_num_obj_train = 0
    CNN_num_obj_val = 0
    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:
        opt.zero_grad()
        emb = batch.text.to(device)
        # print("emb Shape: ", emb.shape)
        # print("emb: ", emb)
        label = torch.unsqueeze(batch.label, 1).to(device)
        # print("label Shape: ", label.shape)
        # print("label: ", label)
        preds = model(emb)
        loss = loss_func(preds, label)
        loss.backward()
        train_loss += loss
        opt.step()

        predsr = np.where(preds.cpu() > 0, 1, 0)
        labelr = np.where(label.cpu() > 0, 1, 0)
        c = predsr == labelr
        CNN_correct_train += np.count_nonzero(c)
        CNN_num_obj_train += len(batch.label)

        trn = np.sum(predsr, axis=1)
        lbl = np.sum(labelr, axis=1)
        trn = trn.tolist()
        lbl = lbl.tolist()
        CNN_all_preds_train.append(trn)
        CNN_all_label_train.append(lbl)
        #YOUR CODE GOES HERE

    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:
        emb = batch.text.to(device)
        #len_emb = batch.text[1].cpu()
        label = torch.unsqueeze(batch.label, 1).to(device)
        preds = model(emb)
```

```
        predsr = np.where(preds.cpu() > 0, 1, 0)
        labelr = np.where(label.cpu() > 0, 1, 0)
        c = predsr == labelr
        CNN_correct_val += np.count_nonzero(c)
        CNN_num_obj_val += len(batch.label)

        val_loss += loss_func(preds.data, label)

        val = np.sum(predsr, axis=1)
        lbl_val = np.sum(labelr, axis=1)
        val = val.tolist()
        lbl_val = lbl_val.tolist()
        CNN_all_preds_val.append(val)
        CNN_all_label_val.append(lbl_val)
        # YOUR CODE GOES HERE

    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: {}, Training accuracy: {}, Valid
model.load_state_dict(best_model)
```

```
/usr/local/lib/python3.7/dist-packages/torchtext/data/batch.py:23: UserWarning: Batcl
  warnings.warn('{} class will be retired soon and moved to torchtext.legacy. Please
Epoch: 1, Training Loss: 0.6645534634590149, Validation Loss: 0.5257830023765564, Tra
Epoch: 2, Training Loss: 0.5165377259254456, Validation Loss: 0.47080883383750916, Tr
Epoch: 3, Training Loss: 0.4343946874141693, Validation Loss: 0.4132627248764038, Tra
Epoch: 4, Training Loss: 0.3629172146320343, Validation Loss: 0.384289413690567, Trai
Epoch: 5, Training Loss: 0.2830201983451843, Validation Loss: 0.36714044213294983, Tr
Epoch: 6, Training Loss: 0.21790607273578644, Validation Loss: 0.3537408113479614, Tr
Epoch: 7, Training Loss: 0.14971472322940826, Validation Loss: 0.35701942443847656, T
<All keys matched successfully>
```

Посчитайте f1-score вашего классификатора.

Ответ:

```
CNN_flat_preds_list = [item for sublist in CNN_all_preds_val for item in sublist]
CNN_flat_label_list = [item for sublist in CNN_all_label_val for item in sublist]
```

```
f1_score(CNN_flat_label_list, CNN_flat_preds_list, average='weighted')
```

```
0.8467762355182719
```

Интерпретируемость

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

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

```
     |████████████████████████████████| 4.4MB 3.8MB/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).unsq

    # 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(de
```

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

def add_attributions_to_visualizer(attributions, text, pred, pred_ind, label, delta, vis_d
    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)
```

```
pred:  pos ( 1.00 ) , delta:  tensor([0.0003], device='cuda:0', dtype=torch.float64)
pred:  pos ( 0.97 ) , delta:  tensor([8.6919e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 1.00 ) , delta:  tensor([6.2853e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.01 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.02 ) , delta:  tensor([9.2424e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 0.94 ) , delta:  tensor([8.4931e-06], device='cuda:0', dtype=torch.float
```

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 (1.00)	pos	0.60	It was a fantastic performance ! pad
pos	pos (0.97)	pos	0.14	Best film ever pad pad pad pad
pos	pos (1.00)	pos	0.67	Such a great show! pad pad pad
neg	neg (0.01)	pos	-1.00	It was a horrible movie pad pad

Эмбэдинги слов

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

```
TEXT.build_vocab(trn, vectors=GloVe(name='6B', dim=300))# 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


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

assert prev_shape == model.embedding.weight.shape
```

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

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

Вы знаете, что делать.

```
import numpy as np

patience=3

min_loss = np.inf

cur_patience = 0

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

    EMBCNN_all_preds_train = []
    EMBCNN_all_label_train = []
    EMBCNN_all_preds_val = []
    EMBCNN_all_label_val = []
    EMBCNN_correct_train = 0
    EMBCNN_correct_val = 0
    EMBCNN_num_obj_train = 0
    EMBCNN_num_obj_val = 0
    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:

        opt.zero_grad()
        emb = batch.text.to(device)
        label = torch.unsqueeze(batch.label, 1).to(device)
        preds = model(emb)
        loss = loss_func(preds, label)
        loss.backward()
        train_loss += loss
        opt.step()

        predsr = np.where(preds.cpu() > 0, 1, 0)
        labelr = np.where(label.cpu() > 0, 1, 0)
        c = predsr == labelr
        EMBCNN_correct_train += np.count_nonzero(c)
        EMBCNN_num_obj_train += len(batch.label)

        trn = np.sum(predsr, axis=1)
        lbl = np.sum(labelr, axis=1)
        trn = trn.tolist()
        lbl = lbl.tolist()
        EMBCNN_all_preds_train.append(trn)
        EMBCNN_all_label_train.append(lbl)
        #YOUR CODE GOES HERE
```

```
    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:

        emb = batch.text.to(device)
        label = torch.unsqueeze(batch.label, 1).to(device)
        preds = model(emb)

        predsr = np.where(preds.cpu() > 0, 1, 0)
        labelr = np.where(label.cpu() > 0, 1, 0)
        c = predsr == labelr
        EMBCNN_correct_val += np.count_nonzero(c)
        EMBCNN_num_obj_val += len(batch.label)

        val_loss += loss_func(preds.data, label)

        val = np.sum(predsr, axis=1)
        lbl_val = np.sum(labelr, axis=1)
        val = val.tolist()
        lbl_val = lbl_val.tolist()
        EMBCNN_all_preds_val.append(val)
        EMBCNN_all_label_val.append(lbl_val)
        # YOUR CODE GOES HERE
    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: {}, Training accuracy: {}, Valid
model.load_state_dict(best_model)
```

Посчитайте f1-score вашего классификатора.

Ответ:

```
EMBCNN_flat_preds_list = [item for sublist in EMBCNN_all_preds_val for item in sublist]
EMBCNN_flat_label_list = [item for sublist in EMBCNN_all_label_val for item in sublist]
```

```
f1_score(EMBCNN_flat_label_list, EMBCNN_flat_preds_list, average='weighted')
```

```
0.8706160394010135
```

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)
```

```
pred:  pos ( 0.91 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([6.2463e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.03 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([6.8991e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.09 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([7.4932e-05], device='cuda:0', dtype=torch.float
```

```
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 (0.91)	pos	1.89	It was a fantastic performance ! pad
pos	neg (0.00)	pos	1.57	Best film ever pad pad pad pad
pos	neg (0.03)	pos	1.69	Such a great show! pad pad pad
neg	neg (0.00)	pos	0.01	It was a horrible movie pad pad
neg	neg (0.09)	pos	1.29	I've never watched something as bad pad
neg	neg (0.00)	pos	0.09	It is a disgusting movie! pad pad

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.91)	pos	1.89	It was a fantastic performance ! pad
pos	neg (0.00)	pos	1.57	Best film ever pad pad pad pad
pos	neg (0.03)	pos	1.69	Such a great show! pad pad pad
neg	neg (0.00)	pos	0.01	It was a horrible movie pad pad