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.

Ввод [1]:

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

from sklearn.metrics import f1_score

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

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

executed in 1.73s, finished 17:37:29 2021-10-10

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.

Ввод [2]:

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

executed in 13ms, finished 17:37:29 2021-10-10

Ввод [3]:

```
SEED = 1234

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

executed in 13ms, finished 17:37:29 2021-10-10

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

Ввод [4]:

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

executed in 14.2s, finished 17:37:44 2021-10-10

Ввод [5]:

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

executed in 1.39s, finished 17:37:45 2021-10-10

Ввод [6]:

```
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)
```

executed in 61ms, finished 17:37:45 2021-10-10

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.

Ввод [7]:

```
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_la

        self.fc = nn.Linear(in_features=hidden_dim*2,out_features=output_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, text, text_lengths):

        #text = [sent len, batch size] (sents = columns)

        embedded = self.embedding(text)

        #embedded = [sent len, batch size, emb dim] (z = emb_word)

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

        # 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] # 1 слой - forward, 2
        #cell = [num layers * num directions, batch size, hid dim]

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

        hidden = torch.cat((hidden[-2,:,:],hidden[-1,:,:]),1)
        hidden=self.dropout(hidden)
        #hidden = [batch size, hid dim * num directions] or [batch_size, hid dim * num dire

        return self.fc(hidden)
```

executed in 14ms, finished 17:37:45 2021-10-10

Поиграйтесь с гиперпараметрами

Ввод [8]:

```
vocab_size = len(TEXT.vocab)
emb_dim = 100
hidden_dim = 256
output_dim = 1
n_layers = 2
bidirectional = True
dropout = 0.2
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=3
```

executed in 13ms, finished 17:37:45 2021-10-10

Ввод [9]:

```
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
)
```

executed in 221ms, finished 17:37:45 2021-10-10

Ввод [10]:

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

executed in 2.28s, finished 17:37:48 2021-10-10

Ввод [11]:

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

max_epochs = 20
```

executed in 13ms, finished 17:37:48 2021-10-10

Ввод [12]:

```
next(iter(train_iter)).text[0].shape
```

executed in 60ms, finished 17:37:48 2021-10-10

Out[12]:

```
torch.Size([114, 64])
```

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

Ввод [13]:

```
import numpy as np

min_loss = np.inf

cur_patience = 0

all_preds = []
all_labels = []

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:
        opt.zero_grad()
        input_embeds=batch.text[0].to(device)
        text_length=batch.text[1].cpu()
        labels=torch.unsqueeze(batch.label,1).to(device)
        preds=model(input_embeds,text_length)
        loss=loss_func(preds,labels)
        loss.requires_grad_()
        loss.backward()
        train_loss+=loss
        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}")
    with torch.no_grad():
        for it, batch in pbar:
            with torch.no_grad():
                input_embeds=batch.text[0].to(device)
                text_length=batch.text[1].cpu()
                all_labels+=batch.label.tolist()
                labels=torch.unsqueeze(batch.label,1).to(device)
                preds=model(input_embeds,text_length)
                all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
                val_loss+=loss_func(preds,labels)

    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
model.load_state_dict(best_model)
```

executed in 5m 24s, finished 17:43:11 2021-10-10

```
Epoch: 1, Training Loss: 0.6429314613342285, Validation Loss: 0.625158369541
1682
```

```
Epoch: 2, Training Loss: 0.5516434907913208, Validation Loss: 0.5389180779
457092

Epoch: 3, Training Loss: 0.39293333888053894, Validation Loss: 0.43005791306
495667

Epoch: 4, Training Loss: 0.2657189965248108, Validation Loss: 0.421782463788
9862

Epoch: 5, Training Loss: 0.16688859462738037, Validation Loss: 0.43720921874
046326

Epoch: 6, Training Loss: 0.10236067324876785, Validation Loss: 0.52485865354
53796
```

Out[13]:

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

Ввод [14]:

```
torch.save(best_model,'bestRNN')
```

executed in 295ms, finished 17:43:12 2021-10-10

Ввод [15]:

```
gc.collect()
torch.cuda.empty_cache()
torch.cuda.reset_max_memory_allocated()
torch.cuda.synchronize()
test_loss = 0.0
all_preds = []
all_labels = []

model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Test")
with torch.no_grad():
    for it, batch in pbar:
        with torch.no_grad():
            input_embeds=batch.text[0].to(device)
            text_length=batch.text[1].cpu()
            all_labels+=batch.label.tolist()
            labels=torch.unsqueeze(batch.label,1).to(device)
            preds=model(input_embeds,text_length)
            all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
            test_loss+=loss_func(preds,labels)

test_loss /= len(test_iter)
print('Test Loss: {}'.format(test_loss))
```

executed in 43.5s, finished 17:43:55 2021-10-10

```
Test Loss: 0.6932282447814941
```

Ввод [16]:

```
f1_score(all_labels,all_preds,average='weighted')
```

executed in 92ms, finished 17:43:55 2021-10-10

Out[16]:

```
0.8191608287451669
```

Посчитайте f1-score вашего классификатора на тестовом датасете.

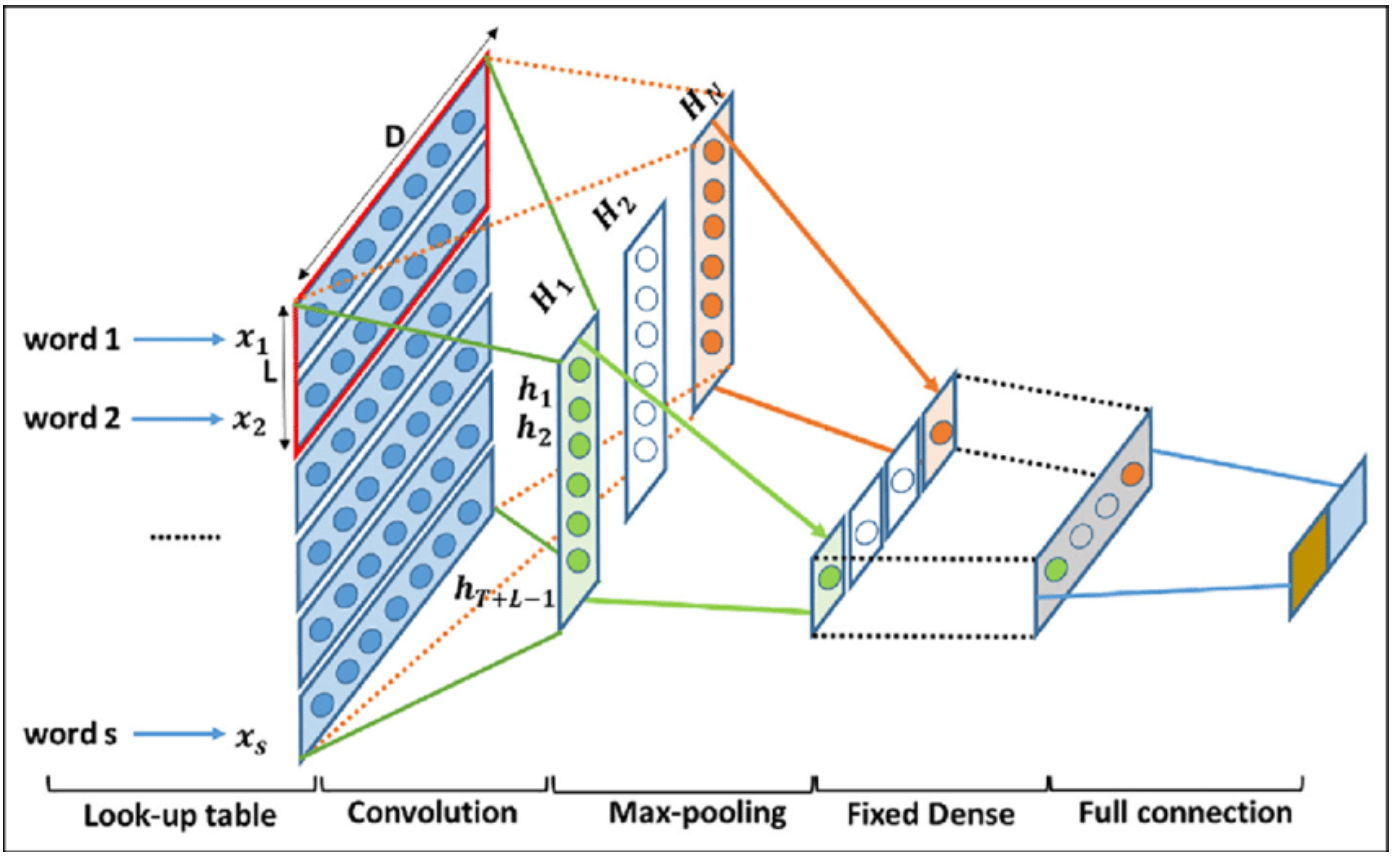
Ответ: 0.819399011037634

Ввод [17]:

```
gc.collect()
torch.cuda.empty_cache()
torch.cuda.reset_max_memory_allocated()
torch.cuda.synchronize()
```

executed in 380ms, finished 17:43:56 2021-10-10

CNN



Для классификации текстов также часто используют сверточные нейронные сети. Идея в том, что как правило предложения 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!

Ввод [18]:

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

executed in 13.0s, finished 17:44:09 2021-10-10

Ввод [19]:

```
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,
)
```

executed in 124ms, finished 17:44:09 2021-10-10

Ввод [20]:

```
next(iter(train_iter)).text.shape
```

executed in 64ms, finished 17:44:09 2021-10-10

Out[20]:

```
torch.Size([128, 891])
```

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.

Ввод [21]:

```
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(in_channels=emb_dim,out_channels=out_channels,kernel_size=k

        self.conv_1 = nn.Conv1d(in_channels=emb_dim,out_channels=out_channels,kernel_size=k

        self.conv_2 = nn.Conv1d(in_channels=emb_dim,out_channels=out_channels,kernel_size=k

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

        self.dropout = nn.Dropout(dropout)


    def forward(self, text):

        #text = [sent len, batch size] (sents = columns)
        embedded = self.embedding(text)
        #print('1',embedded.shape)
        #embedded = [sent len, batch size, emb dim] (z = emb_word)
        #embedded = embedded.unsqueeze(1)
        #print('2',embedded.shape)
        embedded = embedded.permute(0,2,1)
        #print('3',embedded.shape)

        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)
```

executed in 16ms, finished 17:44:09 2021-10-10

Ввод [22]:

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

executed in 480ms, finished 17:44:09 2021-10-10

Ввод [23]:

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

executed in 88ms, finished 17:44:09 2021-10-10

Out[23]:

```
CNN(
  (embedding): Embedding(202065, 300)
  (conv_0): Conv1d(300, 64, kernel_size=(3,), stride=(2,))
  (conv_1): Conv1d(300, 64, kernel_size=(4,), stride=(2,))
  (conv_2): Conv1d(300, 64, kernel_size=(5,), stride=(2,))
  (fc): Linear(in_features=192, out_features=1, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)
```

Ввод [24]:

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

executed in 12ms, finished 17:44:09 2021-10-10

Ввод [25]:

```
max_epochs = 30
```

executed in 16ms, finished 17:44:09 2021-10-10

Обучите!

Ввод [26]:

```
import numpy as np

min_loss = np.inf

cur_patience = 0

all_preds = []
all_labels = []

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:
        opt.zero_grad()
        input_embeds=batch.text
        labels=torch.unsqueeze(batch.label,1).to(device)
        preds=model(input_embeds)
        loss=loss_func(preds,labels)
        loss.requires_grad_()
        loss.backward()
        train_loss+=loss
        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}")
    with torch.no_grad():
        for it, batch in pbar:
            with torch.no_grad():
                input_embeds=batch.text
                all_labels+=batch.label.tolist()
                labels=torch.unsqueeze(batch.label,1).to(device)
                preds=model(input_embeds)
                all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
                val_loss+=loss_func(preds,labels)

    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, val
model.load_state_dict(best_model)
```

executed in 3m 14s, finished 17:47:24 2021-10-10

```
Epoch: 1, Training Loss: 0.6644595861434937, Validation Loss: 0.526606559753
418

Epoch: 2, Training Loss: 0.5289759635925293, Validation Loss: 0.4846778512
```

```
0010376

Epoch: 3, Training Loss: 0.44773128628730774, Validation Loss: 0.42374905943
870544

Epoch: 4, Training Loss: 0.36872854828834534, Validation Loss: 0.38789847493
17169

Epoch: 5, Training Loss: 0.28996771574020386, Validation Loss: 0.36692911386
48987

Epoch: 6, Training Loss: 0.2166883945465088, Validation Loss: 0.375882804393
7683

Epoch: 7, Training Loss: 0.1570805311203003, Validation Loss: 0.430489987134
93347
```

Out[26]:

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

Ввод [27]:

```
torch.save(best_model,'bestCNN')
```

executed in 833ms, finished 17:47:25 2021-10-10

Ввод [28]:

```
gc.collect()
torch.cuda.empty_cache()
torch.cuda.reset_max_memory_allocated()
torch.cuda.synchronize()
test_loss = 0.0
all_preds = []
all_labels = []

model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Test")
with torch.no_grad():
    for it, batch in pbar:
        input_embeds=batch.text
        all_labels+=batch.label.tolist()
        labels=torch.unsqueeze(batch.label,1).to(device)
        preds=model(input_embeds)
        all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
        test_loss+=loss_func(preds,labels)

test_loss /= len(test_iter)
print('Test Loss: {}'.format(test_loss))
```

executed in 10.9s, finished 17:47:36 2021-10-10

```
Test Loss: 0.42569705843925476
```

Ввод [29]:

```
f1_score(all_labels,all_preds,average='weighted')
```

executed in 110ms, finished 17:47:36 2021-10-10

Out[29]:

```
0.8309058843945097
```

Посчитайте f1-score вашего классификатора.

Ответ: 0.8309058843945097

Интерпретируемость

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

Ввод [30]:

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

executed in 14ms, finished 17:47:36 2021-10-10

Ввод [31]:

```
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)
```

executed in 332ms, finished 17:47:36 2021-10-10

Ввод [32]:

```
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).unsqu

    # 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(del

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

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

executed in 14ms, finished 17:47:36 2021-10-10

Ввод [33]:

```
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)
```

executed in 47.7s, finished 17:48:24 2021-10-10

```
pred:  pos ( 0.81 ) , delta:  tensor([2.7922e-05], device='cuda:0', dtype=to
rch.float64)
pred:  neg ( 0.27 ) , delta:  tensor([3.8100e-05], device='cuda:0', dtype=to
rch.float64)
pred:  pos ( 0.99 ) , delta:  tensor([9.8730e-05], device='cuda:0', dtype=to
rch.float64)
pred:  neg ( 0.02 ) , delta:  tensor([4.4304e-05], device='cuda:0', dtype=to
rch.float64)
pred:  neg ( 0.35 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.
float64)
pred:  pos ( 0.54 ) , delta:  tensor([3.5911e-05], device='cuda:0', dtype=to
rch.float64)
```

Попробуйте добавить свои примеры!

Ввод [34]:

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

executed in 14ms, finished 17:48:24 2021-10-10

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

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	**pos (0.81)**	**pos**	**1.98**	It was a fantastic performance ! pad
pos	**neg (0.27)**	**pos**	**1.18**	Best film ever pad pad pad pad
pos	**pos (0.99)**	**pos**	**1.41**	Such a great show! pad pad pad
neg	**neg (0.02)**	**pos**	**-0.35**	It was a horrible movie pad pad
neg	**neg (0.35)**	**pos**	**0.32**	I've never watched something as bad pad
neg	**pos (0.54)**	**pos**	**0.35**	It is a disgusting movie! pad pad

Эмбеддинги слов

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

Ввод [35]:

```
gc.collect()
torch.cuda.empty_cache()
torch.cuda.reset_max_memory_allocated()
torch.cuda.synchronize()
```

executed in 519ms, finished 17:48:24 2021-10-10

Ввод [36]:

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

executed in 5.23s, finished 17:48:30 2021-10-10

Ввод [37]:

```
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,
)
```

executed in 14.2s, finished 17:48:44 2021-10-10

Ввод [38]:

```
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)

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

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

executed in 726ms, finished 17:48:44 2021-10-10

Вы знаете, что делать.

Ввод [39]:

```
import numpy as np

min_loss = np.inf

cur_patience = 0

all_preds = []
all_labels = []

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:
        opt.zero_grad()
        input_embeds=batch.text
        labels=torch.unsqueeze(batch.label,1).to(device)
        preds=model(input_embeds)
        loss=loss_func(preds,labels)
        loss.requires_grad_()
        loss.backward()
        train_loss+=loss
        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}")
    with torch.no_grad():
        for it, batch in pbar:
            with torch.no_grad():
                input_embeds=batch.text
                all_labels+=batch.label.tolist()
                labels=torch.unsqueeze(batch.label,1).to(device)
                preds=model(input_embeds)
                all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
                val_loss+=loss_func(preds,labels)

    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, val
model.load_state_dict(best_model)
```

executed in 1m 29.5s, finished 17:50:14 2021-10-10

```
Epoch: 1, Training Loss: 0.519368588924408, Validation Loss: 0.3886339664459
2285

Epoch: 2, Training Loss: 0.3089892268180847, Validation Loss: 0.3559171557
```

```
4264526

Epoch: 3, Training Loss: 0.1747584044933319, Validation Loss: 0.382631689310
07385

Epoch: 4, Training Loss: 0.07251763343811035, Validation Loss: 0.43725851178
16925
```

Out[39]:

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

Ввод [40]:

```
torch.save(best_model,'bestCNN_EMB')
```

executed in 855ms, finished 17:50:15 2021-10-10

Ввод [41]:

```
gc.collect()
torch.cuda.empty_cache()
torch.cuda.reset_max_memory_allocated()
torch.cuda.synchronize()
test_loss = 0.0
all_preds = []
all_labels = []

model.eval()
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Test")
with torch.no_grad():
    for it, batch in pbar:
        input_embeds=batch.text
        all_labels+=batch.label.tolist()
        labels=torch.unsqueeze(batch.label,1).to(device)
        preds=model(input_embeds)
        all_preds+=list(map(lambda pred:np.where(pred > 0, 1, 0),preds.cpu()))
        test_loss+=loss_func(preds,labels)

test_loss /= len(test_iter)
print('Test Loss: {}'.format(test_loss))
```

executed in 11.2s, finished 17:50:26 2021-10-10

```
Test Loss: 0.566675066947937
```

Ввод [42]:

```
f1_score(all_labels,all_preds,average='weighted')
```

executed in 108ms, finished 17:50:26 2021-10-10

Out[42]:

```
0.7918142211356327
```

Посчитайте f1-score вашего классификатора.

Ответ: 0.7918142211356327

Проверим насколько все хорошо!

Ввод [43]:

```
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)
```

executed in 48.8s, finished 17:51:15 2021-10-10

```
pred:  pos ( 0.96 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.
float64)
pred:  neg ( 0.05 ) , delta:  tensor([4.6202e-05], device='cuda:0', dtype=to
rch.float64)
pred:  pos ( 0.97 ) , delta:  tensor([5.3753e-05], device='cuda:0', dtype=to
rch.float64)
pred:  neg ( 0.00 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.
float64)
pred:  neg ( 0.19 ) , delta:  tensor([8.2251e-05], device='cuda:0', dtype=to
rch.float64)
pred:  neg ( 0.01 ) , delta:  tensor([1.7849e-05], device='cuda:0', dtype=to
rch.float64)
```

Ввод [44]:

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

executed in 15ms, finished 17:51:15 2021-10-10

```
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**	**1.46**	It was a fantastic performance ! pad
pos	**neg (0.05)**	**pos**	**-0.41**	Best film ever pad pad pad pad
pos	**pos (0.97)**	**pos**	**1.46**	Such a great show! pad pad pad
neg	**neg (0.00)**	**pos**	**-0.70**	It was a horrible movie pad pad
neg	**neg (0.19)**	**pos**	**0.30**	I've never watched something as bad pad
neg	**neg (0.01)**	**pos**	**-0.34**	It is a disgusting movie! pad pad

Ввод [ ]:

```

```