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==0.10.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting torchtext==0.10.0
  Downloading torchtext-0.10.0-cp37-cp37m-manylinux1_x86_64.whl (7.6 MB)
     |████████████████████████████████| 7.6 MB 12.3 MB/s
Collecting torch==1.9.0
  Downloading torch-1.9.0-cp37-cp37m-manylinux1_x86_64.whl (831.4 MB)
     |████████████████████████████████| 831.4 MB 2.8 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from t
```

```
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: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: torch, torchtext
  Attempting uninstall: torch
    Found existing installation: torch 1.12.1+cu113
    Uninstalling torch-1.12.1+cu113:
      Successfully uninstalled torch-1.12.1+cu113
  Attempting uninstall: torchtext
    Found existing installation: torchtext 0.13.1
    Uninstalling torchtext-0.13.1:
      Successfully uninstalled torchtext-0.13.1
ERROR: pip's dependency resolver does not currently take into account all the package
torchvision 0.13.1+cu113 requires torch==1.12.1, but you have torch 1.9.0 which is in
torchaudio 0.12.1+cu113 requires torch==1.12.1, but you have torch 1.9.0 which is inc
Successfully installed torch-1.9.0 torchtext-0.10.0
```

```
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
```

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

```
SEED = 1234

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

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

```
train, 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 [00:08<00:00, 10.4MB/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.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(hidden_dim * 2, 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)
```

```
        # 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

        hidden = self.dropout(torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1))  # YOUR

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

        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 = True
dropout = 0.2
PAD_IDX = TEXT.vocab.stoi[TEXT.pad_token]
patience=3


rnn_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
)


rnn_model.to(device)


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

max_epochs = 20
```

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

```
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:
      text_train, text_lenghts_train = batch.text
      labels_train = batch.label
      opt.zero_grad()
      predictions = rnn_model(text_train, text_lenghts_train.cpu()).squeeze(1)
      loss = loss_func(predictions, labels_train)

      loss.backward()
      opt.step()
      train_loss+=loss.item()
    train_loss /= len(pbar)
    val_loss = 0.0
    rnn_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:
        text_val, text_lenghts_val = batch.text
        labels_val = batch.label
        predictions = rnn_model(text_val, text_lenghts_val.cpu()).squeeze(1)
        loss = loss_func(predictions, labels_val)
        val_loss+=loss.item()
    val_loss /= len(pbar)
    if val_loss < min_loss:
        min_loss = val_loss
        best_model = rnn_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, va
rnn_model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.6617807550151853, Validation Loss: 0.6382843321662838
```

```
Epoch: 2, Training Loss: 0.5652698951698568, Validation Loss: 0.6818380871061551
```

```
Epoch: 3, Training Loss: 0.4290856053903155, Validation Loss: 0.5142671789153147
```

```
Epoch: 4, Training Loss: 0.29302065079882195, Validation Loss: 0.48861709716966595
```

```
Epoch: 5, Training Loss: 0.1988173043559285, Validation Loss: 0.4906157370088464
```

```
from sklearn.metrics import f1_score
```

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

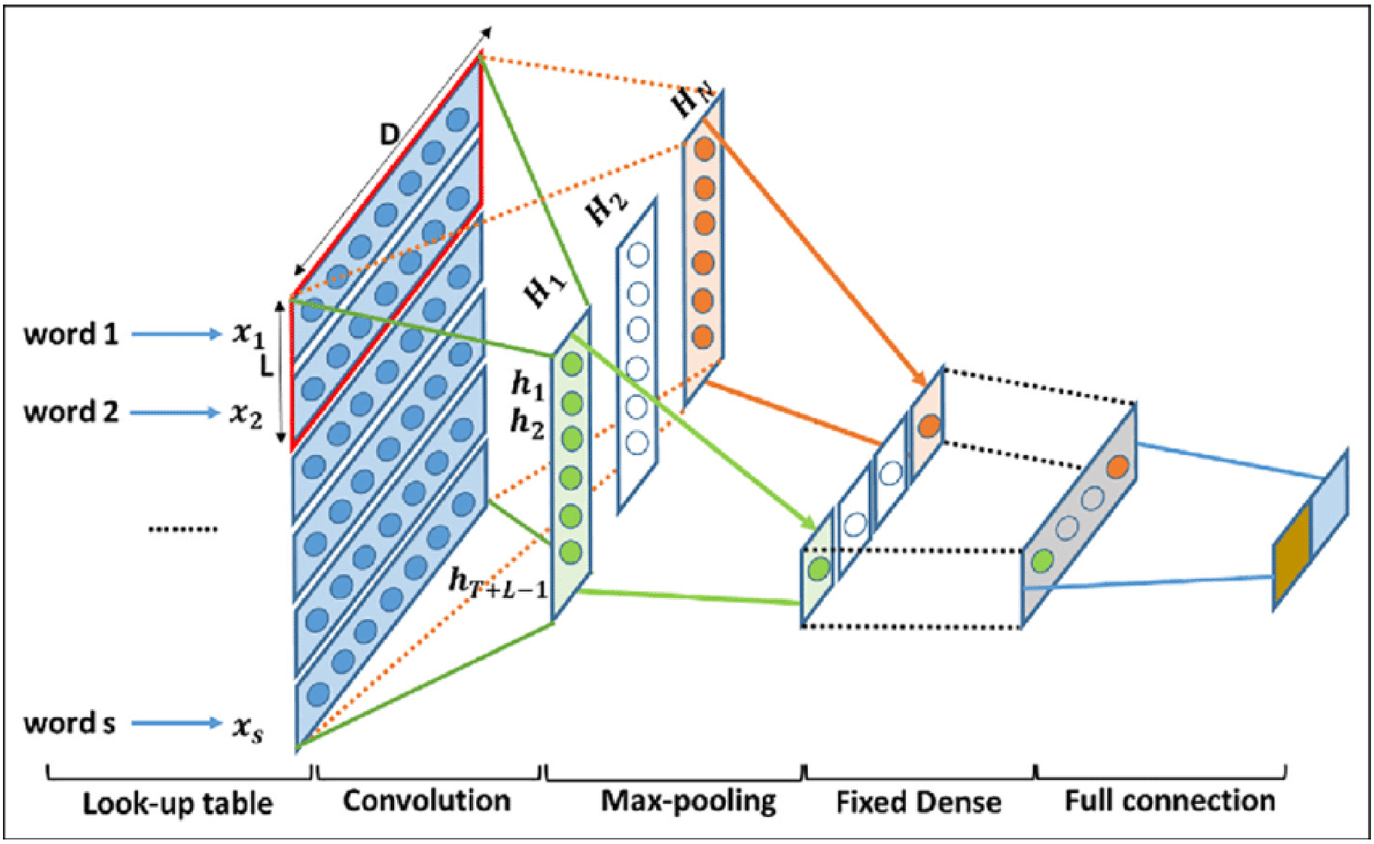
```
score = 0.0
loss_test = 0.0
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Epoch {epoch}")
with torch.no_grad():
  for it, batch in pbar:
    text_test, text_lenghts_test = batch.text
    labels_test = batch.label
    predictions = rnn_model(text_test, text_lenghts_test.cpu()).squeeze(1)
    probabilities = torch.round(torch.sigmoid(predictions))

    score += f1_score(probabilities.cpu().numpy(), labels_test.cpu().numpy())
print(score/len(test_iter))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1580: Undef
  _warn_prf(average, "true nor predicted", "F-score is", len(true_sum))
0.6905453883205751
```

Посчитайте f1-score вашего классификатора на тестовом датасете.

Ответ: 0.6905453883205751

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)
SEED = 1234
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.Conv2d(in_channels=1, out_channels=out_channels, kernel_size=(ker

        self.conv_1 = nn.Conv2d(in_channels=1, out_channels=out_channels, kernel_size=(ker

        self.conv_2 = nn.Conv2d(in_channels=1, out_channels=out_channels, kernel_size=(ker

        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.unsqueeze(1) # may be reshape here

        conved_0 = F.relu(self.conv_0(embedded)).squeeze(3)  # may be reshape here
        conved_1 = F.relu(self.conv_1(embedded)).squeeze(3)   # may be reshape here
        conved_2 = F.relu(self.conv_2(embedded)).squeeze(3)  # 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
```

```
cnn_model = CNN(vocab_size=vocab_size, emb_dim=dim, out_channels=out_channels,
            kernel_sizes=kernel_sizes, dropout=dropout)
cnn_model.to(device)
```

```
CNN(
  (embedding): Embedding(201849, 300)
  (conv_0): Conv2d(1, 64, kernel_size=(3, 300), stride=(1, 1))
  (conv_1): Conv2d(1, 64, kernel_size=(3, 300), stride=(1, 1))
  (conv_2): Conv2d(1, 64, kernel_size=(3, 300), stride=(1, 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):
    train_loss = 0.0
    cnn_model.train()
    pbar = tqdm(enumerate(train_iter), total=len(train_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
      text_train = batch.text
      labels_train = batch.label
      opt.zero_grad()
      predictions = cnn_model(text_train).squeeze(1)
      loss = loss_func(predictions, labels_train)
      loss.backward()
      opt.step()
      train_loss+=loss.item()
    train_loss /= len(pbar)
    val_loss = 0.0
    cnn_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:
        text_val = batch.text
        labels_val = batch.label
        predictions_val = cnn_model(text_val).squeeze(1)
        loss = loss_func(predictions_val, labels_val)
        val_loss+=loss.item()
```

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

        print('Epoch: {}, Training Loss: {}, Validation Loss: {}'.format(epoch, train_loss, va
    cnn_model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.6231619794003285, Validation Loss: 0.4713860899209976
```

```
score = 0.0
loss_test = 0.0
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Epoch {epoch}")
with torch.no_grad():
  for it, batch in pbar:
    text_test = batch.text
    labels_test = batch.label
    predictions = cnn_model(text_test).squeeze(1)
    probabilities = torch.round(torch.sigmoid(predictions))

    score += f1_score(probabilities.cpu().numpy(), labels_test.cpu().numpy())
print(score/len(test_iter))
```

```
0.4551508716462945
```

Посчитайте f1-score вашего классификатора.

Ответ:0.4551508716462945

Интерпретируемость

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

```
Epoch: 8, Training Loss: 0.19782186283247313, Validation Loss: 0.3100925008455912
```

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

```
|████████████████████████████████| 1.4 MB 15.6 MB/s
Epoch: 9, Training Loss: 0.15433798028822363, Validation Loss: 0.3120909551779429
```

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

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

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


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

def forward_with_sigmoid(input):
    return torch.sigmoid(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(cnn_model, 'It was a fantastic performance !', label=1)
interpret_sentence(cnn_model, 'Best film ever', label=1)
interpret_sentence(cnn_model, 'Such a great show!', label=1)
interpret_sentence(cnn_model, 'It was a horrible movie', label=0)
```

```
interpret_sentence(cnn_model, 'I\'ve never watched something as bad', label=0)
interpret_sentence(cnn_model, 'It is a disgusting movie!', label=0)
```

```
pred:  pos ( 0.99 ) , delta:  tensor([0.0001], device='cuda:0', dtype=torch.float64)
pred:  pos ( 0.99 ) , delta:  tensor([8.2322e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 1.00 ) , delta:  tensor([3.2406e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.13 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
pred:  neg ( 0.12 ) , delta:  tensor([3.4001e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 0.96 ) , delta:  tensor([0.0002], device='cuda:0', dtype=torch.float64)
```

Попробуйте добавить свои примеры!

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

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

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.99)	pos	0.46	It was a fantastic performance ! pad
pos	pos (0.99)	pos	0.94	Best film ever pad pad pad pad
pos	pos (1.00)	pos	0.98	Such a great show! pad pad pad
neg	neg (0.13)	pos	-1.31	It was a horrible movie pad pad
neg	neg (0.12)	pos	-1.72	I've never watched something as bad pad
neg	pos (0.96)	pos	-0.37	It is a disgusting movie! pad pad

Legend: ☐ Negative ☐ Neutral ☐ Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.99)	pos	0.46	It was a fantastic performance ! pad
pos	pos (0.99)	pos	0.94	Best film ever pad pad pad pad
pos	pos (1.00)	pos	0.98	Such a great show! pad pad pad
neg	neg (0.13)	pos	-1.31	It was a horrible movie pad pad

Эмбеддинги слов

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

```
gl = GloVe()
TEXT.build_vocab(trn, vectors= gl)# 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,
)


emb_cnn_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 = emb_cnn_model.embedding.weight.shape

emb_cnn_model.embedding.weight = nn.Parameter(word_embeddings)

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

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

Вы знаете, что делать.

```
import numpy as np

min_loss = np.inf

cur_patience = 0

for epoch in range(1, max_epochs + 1):
    train_loss = 0.0
    emb_cnn_model.train()
    pbar = tqdm(enumerate(train_iter), total=len(train_iter), leave=False)
    pbar.set_description(f"Epoch {epoch}")
    for it, batch in pbar:
```

```
    for it, batch in pbar:
      text_train = batch.text
      labels_train = batch.label
      opt.zero_grad()
      predictions = emb_cnn_model(text_train).squeeze(1)
      loss = loss_func(predictions, labels_train)
      loss.backward()
      opt.step()
      train_loss+=loss.item()
    train_loss /= len(pbar)
    val_loss = 0.0
    emb_cnn_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:
        text_val = batch.text
        labels_val = batch.label
        predictions_val = emb_cnn_model(text_val).squeeze(1)
        loss = loss_func(predictions_val, labels_val)
        val_loss+=loss.item()
    val_loss /= len(pbar)
    if val_loss < min_loss:
        min_loss = val_loss
        best_model = emb_cnn_model.state_dict()
    else:
        cur_patience += 1
        if cur_patience == patience:
            cur_patience = 0
            break

    print('Epoch: {}, Training Loss: {}, Validation Loss: {}'.format(epoch, train_loss, va
emb_cnn_model.load_state_dict(best_model)
```

```
Epoch: 1, Training Loss: 0.5017942601311816, Validation Loss: 0.36061601241429647
Epoch: 2, Training Loss: 0.32416706896611375, Validation Loss: 0.3133739014466604
Epoch: 3, Training Loss: 0.21269160998563696, Validation Loss: 0.2948601697882017
Epoch: 4, Training Loss: 0.10823970519169404, Validation Loss: 0.3085709715882937
Epoch: 5, Training Loss: 0.044260995000274514, Validation Loss: 0.342825702826182
<All keys matched successfully>
```

```
score = 0.0
loss_test = 0.0
pbar = tqdm(enumerate(test_iter), total=len(test_iter), leave=False)
pbar.set_description(f"Epoch {epoch}")
with torch.no_grad():
  for it, batch in pbar:
    text_test = batch.text
    labels_test = batch.label
    predictions = emb_cnn_model(text_test).squeeze(1)
    probabilities = torch.round(torch.sigmoid(predictions))

    score += f1_score(probabilities.cpu().numpy(), labels_test.cpu().numpy())
print(score/len(test_iter))
```

```
0.46335027654530275
```

Посчитайте f1-score вашего классификатора.

Ответ: 0.46335027654530275

Проверим насколько все хорошо!

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

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

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

```
pred:  pos ( 0.87 ) , delta:  tensor([3.2894e-06], device='cuda:0', dtype=torch.float
pred:  neg ( 0.21 ) , delta:  tensor([2.0628e-05], device='cuda:0', dtype=torch.float
pred:  pos ( 0.83 ) , delta:  tensor([2.5098e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.00 ) , delta:  tensor([4.0084e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.31 ) , delta:  tensor([2.2480e-05], device='cuda:0', dtype=torch.float
pred:  neg ( 0.00 ) , delta:  tensor([2.0704e-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 (0.87)	pos	1.33	It was a fantastic performance ! pad
pos	pos (0.83)	pos	1.28	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.81	It was a horrible movie pad pad
neg	neg (0.31)	pos	0.16	I've never watched something as bac
neg	neg (0.00)	pos	-1.14	It is a disgusting movie! pad pad

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
pos	pos (0.87)	pos	1.33	It was a fantastic performance ! pad
pos	neg (0.21)	pos	1.51	Best film ever pad pad pad pad
pos	pos (0.83)	pos	1.28	Such a great show! pad pad pad
neg	neg (0.00)	pos	-0.81	It was a horrible movie pad pad
neg	neg (0.31)	pos	0.16	I've never watched something as bac
neg	neg (0.00)	pos	-1.14	It is a disgusting movie! pad pad

✓ 0 сек. выполнено в 23:46