Embeddings

Привет! Сегодня ты поработаешь с эмбеддингами: сделаешь классификатор эмоции твитов. Для начала, загрузи их:

```
!gdown https://drive.google.com/uc?id=1eE1FiUkXkcbw0McId4i7qY-L8hH-_Qph&export=down
!unzip archive.zip
```

```
Downloading...
From: https://drive.google.com/uc?id=1eE1FiUkXkcbw0McId4i7qY-L8hH-_Qph
To: /content/archive.zip
84.9MB [00:00, 128MB/s]
Archive:  archive.zip
replace training.1600000.processed.noemoticon.csv? [y]es, [n]o, [A]ll, [N]one,
```

Заимпортируй библиотеки и сделай работу скриптов вопспроизводимой.

```
import math
import random
import string

import numpy as np
import pandas as pd
import seaborn as sns

import torch
import nltk
import gensim
import gensim.downloader as api


random.seed(42)
np.random.seed(42)
torch.random.manual_seed(42)
torch.cuda.random.manual_seed(42)
torch.cuda.random.manual_seed_all(42)

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


data = pd.read_csv("training.1600000.processed.noemoticon.csv", encoding="latin", h
```

Посмотрим на данные

```
data.head()
```

	emotion	id	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49	NO_QUERY	scotthamilton	is upset that he can't update his Facebook

```
examples = data["text"].sample(10)
print("\n".join(examples))
```

```
@chrishasboobs AHHH I HOPE YOUR OK!!!
@misstoriblack cool , i have no tweet apps  for my razr 2
@TiannaChaos i know  just family drama. its lame.hey next time u hang out with
School email won't open  and I have geography stuff on there to revise! *Stupi
upper airways problem
Going to miss Pastor's sermon on Faith...
on lunch....dj should come eat with me
@piginthepoke oh why are you feeling like that?
gahh noo!peyton needs to live!this is horrible
@mrstessyman thank you glad you like it! There is a product review bit on the
```

Текст очень грязные. Надо добавить очистку текста в его предобработку.

Чтобы сравнивать различные методы обработки текста/модели/прочее, разделим датасет на dev(для обучения модели) и test(для получения качества модели).

```
indexes = np.arange(data.shape[0])
np.random.shuffle(indexes)
dev_size = math.ceil(data.shape[0] * 0.8)

dev_indexes = indexes[:dev_size]
test_indexes = indexes[dev_size:]

dev_data = data.iloc[dev_indexes]
test_data = data.iloc[test_indexes]

dev_data.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)
```

Обработка текста

Стокенизируем текст, избавим от знаков пунктуации и мелких слов.

```
tokenizer = nltk.WordPunctTokenizer()
line = tokenizer.tokenize(dev_data["text"][0].lower())
print(" ".join(line))
```

```
@ claire_nelson i ' m on the north devon coast the next few weeks will be dowr
```

```
filtered_line = [w for w in line if all(c not in string.punctuation for c in w) and
print(" ".join(filtered_line))
```

```
north devon coast next weeks will down devon again sometime hope though
```

Загрузим предобученную модель эмбеддингов. Если хотите, можно попробовать другую. Полный список можно найти здесь: https://github.com/RaRe-Technologies/gensim-data .

```
word2vec = api.load("word2vec-google-news-300")
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:252: UserW
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
```

```
emb_line = [word2vec.get_vector(w) for w in filtered_line if w in word2vec]
print(sum(emb_line).shape)
```

```
(300,)
```

Эмбеддинги не нормализированны, поэтому это тоже надо делать (нейросети и не только любят нормальные данные).

```
mean = np.mean(word2vec.vectors, 0)
std = np.std(word2vec.vectors, 0)
norm_emb_line = [(word2vec.get_vector(w) - mean) / std for w in filtered_line if w
print(sum(norm_emb_line).shape)
print([all(norm_emb_line[i] == emb_line[i]) for i in range(len(emb_line))])
```

```
(300,)
[False, False, False, False, False, False, False, False, False, False, False,
```

Сделаем датасет, который будет по запросу возвращать подготовленные данные.

```
from torch.utils.data import Dataset, random_split
```

```
class TwitterDataset(Dataset):
    def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str,
        self.tokenizer = nltk.WordPunctTokenizer()

        self.data = data

        self.feature_column = feature_column
        self.target_column = target_column

        self.word2vec = word2vec

        self.label2num = lambda label: 0 if label == 0 else 1
        self.mean = np.mean(word2vec.vectors, axis=0)
        self.std = np.std(word2vec.vectors, axis=0)
```

```
    def __getitem__(self, item):
        text = self.data[self.feature_column][item]
        label = self.label2num(self.data[self.target_column][item])

        tokens = self.get_tokens_(text)
        embeddings = self.get_embeddings_(tokens)

        return {"feature": embeddings, "target": label}

    def get_tokens_(self, text):
        # Получи все токены из текста и профильтруй их
        tokens = self.tokenizer.tokenize(text)
        filtered_tokens =[w for w in tokens if all(c not in string.punctuation for
        return filtered_tokens

    def get_embeddings_(self, tokens):
        #embeddings = np.mean([self.word2vec.get_vector(w) for w in tokens if w in
        #print(tokens)
        embeddings = [(self.word2vec.get_vector(w) - self.mean) / self.std for w in
        #print(embeddings)

        if len(embeddings) == 0:
            embeddings = np.zeros((1, self.word2vec.vector_size))
        else:
            embeddings = np.array(embeddings)
            if len(embeddings.shape) == 1:
                embeddings = embeddings.reshape(-1, 1)

        return embeddings

    def __len__(self):
        return self.data.shape[0]


dev = TwitterDataset(dev_data, "text", "emotion", word2vec)
#token_list = dev.get_tokens_("mama! +mila ramu, u io pppe")
#dev.get_embeddings_(token_list)
```

Average embedding

Попробуем получить векторное представление предложения из эмбеддингов слов. Самый простой вариант: усреднить вектора по всем словам. Полученный вектор можно отправить любому классификатору как вектор признаков.

Посмотрим, насколько хорошо усреднее работает для определение эмоций твитов. Сделаем их визуализацию.

```
indexes = np.arange(len(dev))
np.random.shuffle(indexes)
example_indexes = indexes[::1000]
```

```
examples = {"features": [np.sum(dev[i]["feature"], axis=0) for i in example_indexes
            "targets": [dev[i]["target"] for i in example_indexes]}
```

```
print(len(examples["features"]))
print(example_indexes)
```

```
1280
[ 170683  197302  769273 ...  803776 1073670  827384]
```

```
print(examples.keys())
```

```
dict_keys(['features', 'targets'])
```

Для визуализации векторов надо получить их проекцию на плоскость. Сделаем это с помощью `PCA`. Можно получить более аккуратными алгоритмами, но данный алгоритм покажет сложность задачи и поможет оценить требования к классификатору.

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2)
#examples["transformed_features"] = # Обучи PCA на эмбеддингах слов
examples["transformed_features"] = pca.fit_transform(examples['features'])    # Обучи
```

```
print(examples["transformed_features"].shape)
```

```
(1280, 2)
```

```
import bokeh.models as bm, bokeh.plotting as pl
from bokeh.io import output_notebook
output_notebook()

def draw_vectors(x, y, radius=10, alpha=0.25, color='blue',
                 width=600, height=400, show=True, **kwargs):
    """ draws an interactive plot for data points with auxilirary info on hover """
    data_source = bm.ColumnDataSource({ 'x' : x, 'y' : y, 'color': color, **kwargs

    fig = pl.figure(active_scroll='wheel_zoom', width=width, height=height)
    fig.scatter('x', 'y', size=radius, color='color', alpha=alpha, source=data_sour

    fig.add_tools(bm.HoverTool(tooltips=[(key, "@" + key) for key in kwargs.keys()]
    if show: pl.show(fig)
    return fig


draw_vectors(
    examples["transformed_features"][:, 0],
    examples["transformed_features"][:, 1],
    color=[["red", "blue"][t] for t in examples["targets"]]
    )
```

Для обученния и тестирования нейросетевой модели сделаем отдельные функции.

```
from tqdm.notebook import tqdm


def training(model, optimizer, criterion, train_loader, epoch, device="cpu"):
    pbar = tqdm(train_loader, desc=f"Epoch {e + 1}. Train Loss: {0}")
    model.train()
    for batch in pbar:
        features = batch["features"].to(device)
        targets = batch["targets"].to(device)

        #optimizer.zero_grad()
        preds = model(features)      # Получи предсказания модели
        optimizer.zero_grad()
        loss = criterion(preds,targets) # Посчитай лосс
        loss.backward() # Обнови параметры модели
        optimizer.step()

        pbar.set_description(f"Epoch {e + 1}. Train Loss: {loss:.4}")


def testing(model, criterion, test_loader, device="cpu"):
    pbar = tqdm(test_loader, desc=f"Test Loss: {0}, Test Acc: {0}")
    mean_loss = 0
    mean_acc = 0
    model.eval()
    with torch.no_grad():
        for batch in pbar:
            features = batch["features"].to(device)
            targets = batch["targets"].to(device)

            preds = model(features) # Получи предсказания модели
            loss = criterion(preds,targets) # Посчитай лосс
            #acc_tmp = (targets.argmax(-1)==preds.argmax(-1).float().numpy()) # Пос
            #acc = float(100 * acc_tmp.sum()/len(acc_tmp))
            #print((targets == preds).sum())
            #print(preds)
            #print(preds.shape)
            #print(targets)
            acc_temp = (preds[:,1]>preds[:,0]).int()
            #print(acc_temp)
            #print((acc_temp == targets).sum().item())
            #print(len(targets))
            acc = (acc_temp == targets).sum().item()/len(targets)
            #print(acc)

            mean_loss += loss.item()
            #mean_acc += acc.item()
            mean_acc += acc

            pbar.set_description(f"Test Loss: {loss:.4}, Test Acc: {acc:.4}")

    pbar.set_description(f"Test Loss: {mean_loss / len(test_loader):.4}, Test Acc:
```

```
    return {"Test Loss": mean_loss / len(test_loader), "Test Acc": mean_acc / len(t
```

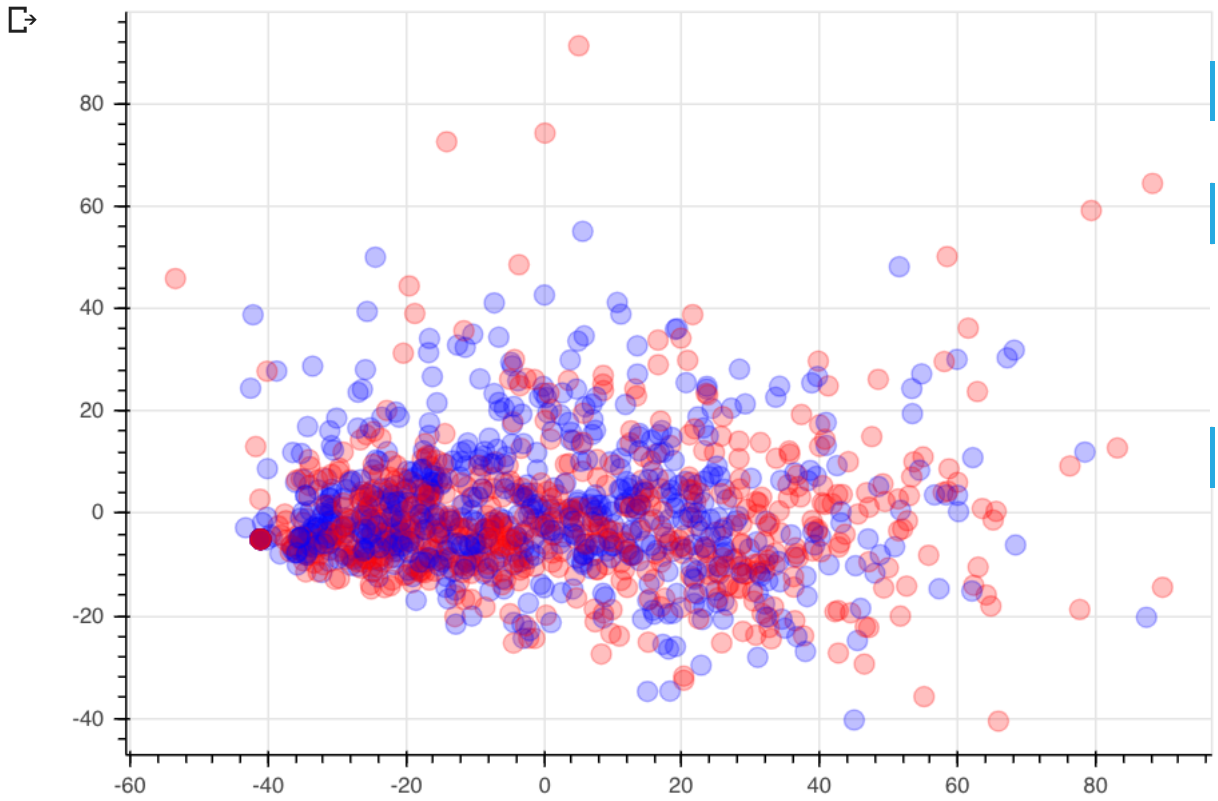
Создадим модель, оптимизатор и целевую функцию. Можете создавать любую модель, с любым количеством слоев, функций активации и прочее.

```
import torch.nn as nn
from torch.optim import Adam

class TwoLayerNet(nn.Module):
  def __init__(self, D_in, H, D_out):
    super(TwoLayerNet, self).__init__()
    self.linear1 = nn.Linear(D_in,H)
    self.linear2 = nn.Linear(H,D_out)
    self.softmax = nn.Sigmoid()
  def forward(self,x):
    h_relu = self.linear1(x).clamp(min=0)
    y_pred = self.linear2(h_relu)
    preds = self.softmax(y_pred)
    return preds

# Не забудь поиграться с параметрами ;)
vector_size = dev.word2vec.vector_size
num_classes = 2
lr = 1e-2
num_epochs = 1

device = "cuda" if torch.cuda.is_available() else "cpu"
model = TwoLayerNet(vector_size,20,num_classes).to(device) # Твоя модель
model = model.cuda()
criterion = nn.CrossEntropyLoss() # Твой лосс
#criterion = nn.BCEWithLogitsLoss()
optimizer = Adam(model.parameters(),lr=lr) # Твой оптимайзер
```

Обучим модель и протестируем её. После каждой эпохи, проверим качество модели по валидационной части датасета. Если метрика стала лучше, будем сохранять модель. Подумай, какая метрика (точность или лосс) будет лучше работать в этой задаче?

Здесь и далее реализованно с помощью лосс. Если думаешь, что лучше сравнивать модель через качество, то поменяй код выбора модели.

```
best_metric = np.inf
for e in range(num_epochs):
    training(model, optimizer, criterion, train_loader, e, device)
    log = testing(model, criterion, valid_loader, device)
    print(log)
    if log["Test Loss"] < best_metric:
        torch.save(model.state_dict(), "model.pt")
        best_metric = log["Test Loss"]
```



Figure(id = '1223', …)

Скорее всего, ты увидел не сильно различающиеся классы. Возможно, обычный линейный классификатор не очень хорошо справится с задачей. Надо будет делать глубокую(хотя бы два слоя) нейронную сеть.

Подготовим загрузчики данных. Усреднее векторов будем делать в "батчевалке" (`collate_fn`). Она используется для того, чтобы собирать из данных `torch.Tensor` батчи, которые можно отправлять в модель.

```
from torch.utils.data import DataLoader


batch_size = 1024
num_workers = 4

def average_emb(batch):
    features = [np.mean(b["feature"], axis=0) for b in batch]
    targets = [b["target"] for b in batch]

    return {"features": torch.FloatTensor(features), "targets": torch.LongTensor(ta


train_size = math.ceil(len(dev) * 0.8)

train, valid = random_split(dev, [train_size, len(dev) - train_size])

train_loader = DataLoader(train, batch_size=batch_size, num_workers=num_workers, sh
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, sh
```

```
Epoch 1. Train Loss: 0.5562: 100%  1000/1000 [04:13<00:00, 3.95it/s]

Test Loss: 0.5572, Test Acc: 0.7371: 100%  250/250 [01:39<00:00, 2.50it/s]
```

```
test_loader = DataLoader(
    TwitterDataset(test_data, "text", "emotion", word2vec),
    batch_size=batch_size,
    num_workers=num_workers,
    shuffle=False,
    drop_last=False,
    collate_fn=average_emb)

model.load_state_dict(torch.load("model.pt", map_location=device))

print(testing(model, criterion, test_loader, device=device))
```

```
Test Loss: 0.5583, Test Acc: 0.7357: 100%  313/313 [01:41<00:00, 3.07it/s]

{'Test Loss': 0.5583250532135035, 'Test Acc': 0.735691643370607}
```

TF-iDF

Вместо обычного усреднения эмбеддингов их можно дополнительно перевзвесить. Для этого воспользуемся алгоритмом `TD-iDF`. Он уже реализован в библиотеке `scikit-learn`, остается только его добавить в наш пайплайн.

```
from collections import defaultdict
from typing import Dict

from sklearn.feature_extraction.text import TfidfVectorizer


class TwitterDatasetTfIdf(TwitterDataset):
    def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str,
        super().__init__(data, feature_column, target_column, word2vec)

        if weights is None:
            self.weights = self.get_tf_idf_()
        else:
            self.weights = weights

    def get_embeddings_(self, tokens):
        embeddings = [(self.word2vec.get_vector(token) - self.mean) / self.std  * s

        if len(embeddings) == 0:
            embeddings = np.zeros((1, self.word2vec.vector_size))
        else:
            embeddings = np.array(embeddings)
            if len(embeddings.shape) == 1:
                embeddings = embeddings.reshape(-1, 1)
```

```
                embeddings = embeddings.reshape(-1, 1)

        return embeddings

    def get_tf_idf_(self):
        # Надо обучить tfidf на очищенном тексте. Но он принимает только список текстов, а не
        # Надо превратить второе в первое
        data_list = self.data[self.feature_column].values.tolist()
        filt_list = []
        for sent in data_list:
          sent_list = ""
          token_list = self.get_tokens_(sent)
          for tk in token_list:
            if (tk in self.word2vec.vocab)&(tk in self.word2vec):
              sent_list += tk + ' '
          filt_list.append(sent_list)

        tf_idf = TfidfVectorizer()
        # Обучи tf-idf
        print(filt_list[:5])

        tf_idf.fit_transform(filt_list)
        return dict(zip(tf_idf.get_feature_names(), tf_idf.idf_))



dev = TwitterDatasetTfIdf(dev_data, "text", "emotion", word2vec)
```

```
['north devon coast next weeks will down Devon again sometime hope though ', '
```

```
print(dev[0]["feature"])
```

```
[[ -3.116123   -9.387659    2.8407967 ...   3.0150862  23.524979
  -22.10702  ]
 [-21.44588     1.5792899  -7.1001697 ...   0.5254118 -11.286701
    7.0245066]
 [  1.9050117   7.5626154 -21.14618   ...   1.6055622   6.964936
    2.6587818]
 ...
 [  7.6573157  10.500135   -0.6423716 ...  -8.070188    6.1698217
 -14.854714 ]
 [  0.651636    5.8196692   9.398581  ...  -2.5179298   1.8694897
  -3.4729521]
 [  3.8191984   0.8723012  -1.0635513 ...  -1.0331593   4.9795895
  -2.766693 ]]
```

```
dev_data['text'].head(10)
```

```
0    @Claire_Nelson i'm on the north devon coast th...
1    @jhicks  i will think of you on Sunday!   Who ...
2    Out in the garden with the kids debating wheth...
3    @FrVerona thank u my love...u've shown me the ...
4    ig with @jonasbrosfan1  going to buy LYAMM tog...
7    @Teddy_Salad_Tears ok, cleansing of the soul &...
```

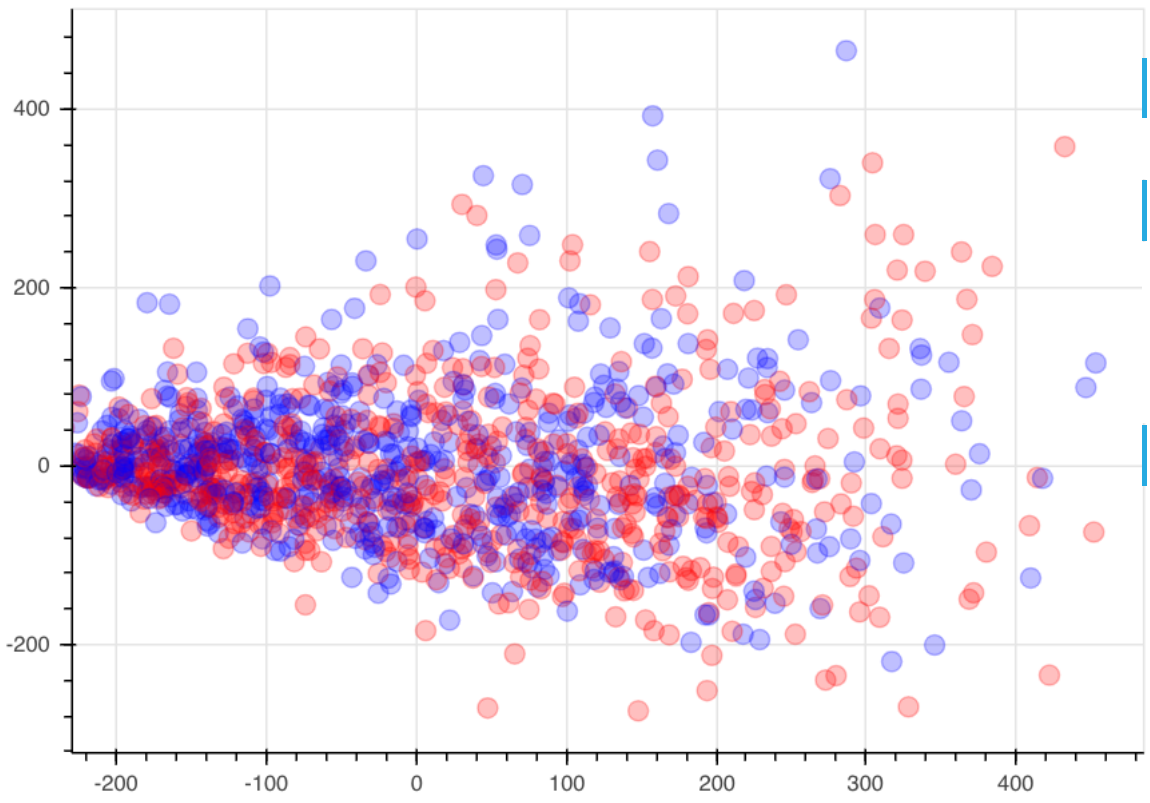
Посмотрим на сложность получившейся задачи используя визуализацию через `PCA`.

```
indexes = np.arange(len(dev))
np.random.shuffle(indexes)
example_indexes = indexes[::1000]

#feat = [dev_2[i]["target"] for i in example_indexes if dev_2[i] in word2vec]
examples = {"features": [np.sum(dev[i]["feature"], axis=0) for i in example_indexes
            "targets": [dev[i]["target"] for i in example_indexes]}
print(len(examples["features"]))
```

```
1280
```

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)
examples["transformed_features"] = pca.fit_transform(examples['features']) # Ты знае
```

```
draw_vectors(
    examples["transformed_features"][:, 0],
    examples["transformed_features"][:, 1],
    color=[["red", "blue"][t] for t in examples["targets"]]
    )
```



Figure(id = '1480', …)

Создать нейросетку, обучим её на этих данных.

```
train_size = math.ceil(len(dev) * 0.8)

train, valid = random_split(dev, [train_size, len(dev) - train_size])

train_loader = DataLoader(train, batch_size=batch_size, num_workers=num_workers, sh
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, sh


# Не забудь поиграться с параметрами ;)

class TwoLayerNet(nn.Module):
  def __init__(self, D_in, H, D_out):
    super(TwoLayerNet, self).__init__()
    self.linear1 = nn.Linear(D_in,H)
    self.linear2 = nn.Linear(H,D_out)
    self.softmax = nn.Sigmoid()
  def forward(self,x):
    h_relu = self.linear1(x).clamp(min=0)
    y_pred = self.linear2(h_relu)
    preds = self.softmax(y_pred)
    return preds

# Не забудь поиграться с параметрами ;)
vector_size = dev.word2vec.vector_size
num_classes = 2
lr = 1e-2
num_epochs = 1

device = "cuda" if torch.cuda.is_available() else "cpu"
model = TwoLayerNet(vector_size,40,num_classes).to(device) # Твоя модель
model = model.cuda()
criterion = nn.CrossEntropyLoss() # Твой лосс
optimizer = Adam(model.parameters(),lr=lr) # Твой оптимайзер


best_metric = np.inf
for e in range(num_epochs):
    training(model, optimizer, criterion, train_loader, e, device)
    print(testing(model, criterion, valid_loader, device))
    print(log)
    if log["Test Loss"] < best_metric:
        torch.save(model.state_dict(), "model.pt")
        best_metric = log["Test Loss"]
```

```
Epoch 1. Train Loss: 0.5763: 100%                1000/1000 [02:54<00:00, 5.74it/s]

Test Loss: 0.5708, Test Acc: 0.7222: 100%        250/250 [00:52<00:00, 4.76it/s]

{'Test Loss': 0.5707776043415069, 'Test Acc': 0.7221953125}
{'Test Loss': 0.557197573184967, 'Test Acc': 0.73705859375}
```

```
test = TwitterDatasetTfIdf(test_data, "text", "emotion", word2vec, weights=dev.weig

test_loader = DataLoader(
    test,
    batch_size=batch_size,
    num_workers=num_workers,
    shuffle=False,
    drop_last=False,
    collate_fn=average_emb)

model.load_state_dict(torch.load("model.pt", map_location=device))

print(testing(model, criterion, test_loader, device=device))
```

```
Test Loss: 0.5804, Test Acc: 0.7109: 100%          313/313 [00:57<00:00, 5.48it/s]

{'Test Loss': 0.569619735208944, 'Test Acc': 0.7240696136182109}
```

Есть ли разница в качестве между способами? Получилось ли улучшить качество модели?

Качество модели упало на 1%.

Сделай небольшое исследование:

- Попробуй сделать несколько нейросеток в качестве классификатора
- Попробуй другие предобученные эмбеддинги
- Попробуй очистить текст от ников ("@username"), url-ов и других символов

Для реализации последнего тебе могут помочь регулярные выражения (`import re`). Напише ниже отчет, что ты попробовал и что получилось.

Делал обучение с 1 и 2,4 эпохи - удалось добиться качества до 78%. Пробовал делать другие архитектуры с болиши количеством слоев - идет переполнение колаба и сложно нормально исследовать. 3,5 слоев линейных давалю чут более лучшие рузьтаты, но не более 80%.
