Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Embeddings

Привет! В этом домашнем задании мы с помощью эмбеддингов решим задачу семантической классификации твитов.

Для этого мы воспользуемся предобученными эмбеддингами word2vec.

Для начала скачаем датасет для семантической классификации твитов:

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

```
Downloading...
From: https://drive.google.com/uc?id=1eE1FiUkXkcbw0McId4i7qY-L8hH-_Qph
To: /content/archive.zip
84.9MB [00:00, 97.9MB/s]
Archive:  archive.zip
  inflating: training.1600000.processed.noemoticon.csv
```

Импортируем нужные библиотеки:

```
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", header=None, names=["emotion", "id", "date", "flag
```

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 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

Выведем несколько примеров твитов, чтобы понимать, с чем мы имеем дело:

```
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 kim n u guys like have a sleepover or whatever,
School email won't open  and I have geography stuff on there to revise! *Stupid School* :'(
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 site  Enjoy knitting it!
```

Как вилим, тексты твитов очень "грязные". Нужно предобработать датасет, прежде чем строить для него модель классификации.

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

Токенизируем текст, избавимся от знаков пунктуации и выкинем все слова, состоящие менее чем из 4 букв:

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

```
@ claire_nelson i ' m on the north devon coast the next few weeks will be down in devon again in may sometime i hope though !
['@', 'claire_nelson', 'i', "'", 'm', 'on', 'the', 'north', 'devon', 'coast', 'the', 'next', 'few', 'weeks', 'will', 'be', 'dow
```

```
filtered_line = [w for w in line if all(c not in string.punctuation for c in w) and len(w) > 3]
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.

Данная модель выдает эмбеддинги для **слов**. Строить по эмбеддингам слов эмбеддинги предложений мы будем ниже.

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

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

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

```
(300,)
```

Нормализуем эмбеддинги, прежде чем обучать на них сеть. (наверное, вы помните, что нейронные сети гораздо лучше обучаются на нормализованных данных)

```
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 in word2vec and len(w) > 3]
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, 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, word2vec: gensim.models.Word2Vec):
        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
        line = self.tokenizer.tokenize(text.lower())[2:]
        filtered_line = [w for w in line if all(c not in string.punctuation for c in w) and len(w) > 3 and w in self.word2vec]
        return filtered_line
        # Получи все токены из текста и профильтруй их

    def get_embeddings_(self, tokens):
        # Получи эмбеддинги слов и усредни их
        embeddings = [(self.word2vec.get_vector(w) - self.mean) / self.std for w in tokens]
        # Получи эмбеддинги слов и усредни их

        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)
print(dev.data["emotion"].unique())
```

```
[0 4]
```

Отлично, мы готовы с помощью эмбеддингов слов превращать твиты в векторы и обучать нейронную сеть.

Превращать твиты в векторы, используя эмбеддинги слов, можно несколькими способами. А именно такими:

Average embedding (2 балла)

Это самый простой вариант, как получить вектор предложения, используя векторные представления слов в предложении. А именно: вектор предложения есть средний вектор всех слов в предлоежнии (которые остались после токенизации и удаления коротких слов, конечно).

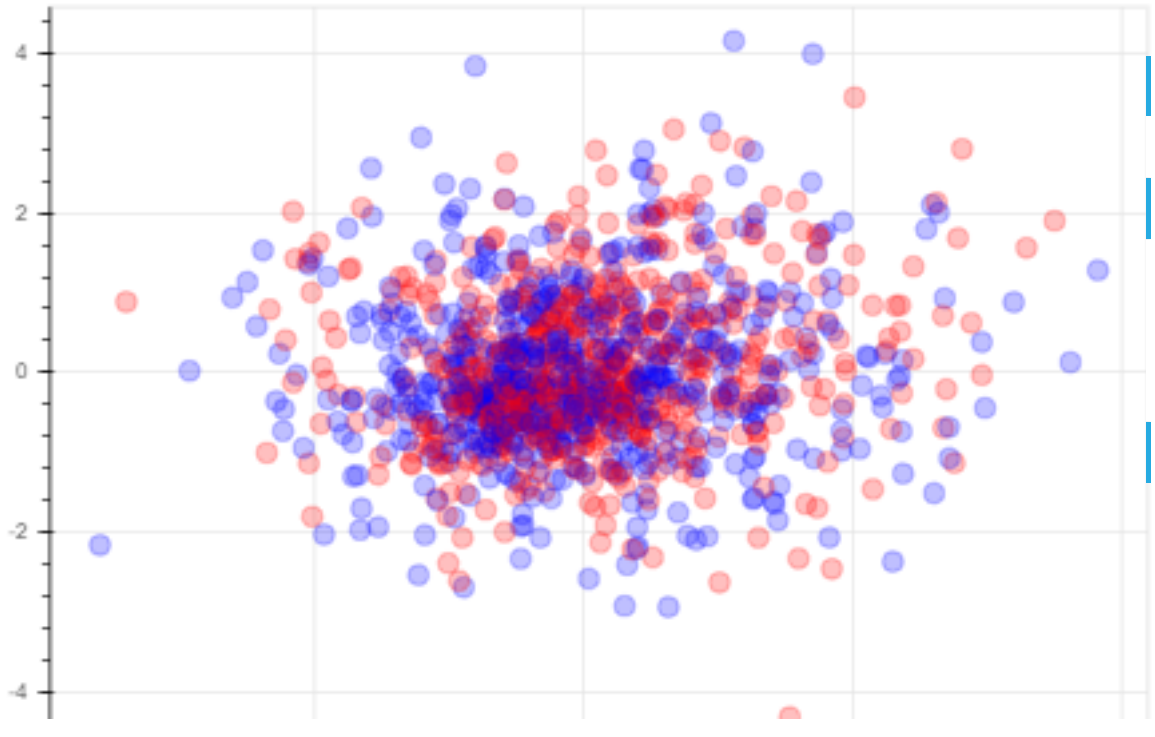
```
indexes = np.arange(len(dev))
np.random.shuffle(indexes)
example_indexes = indexes[::1000]

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"]))
```

```
1280
```

Давайте сделаем визуализацию полученных векторов твитов тренировочного (dev) датасета. Так мы увидим, насколько хорошо твиты с разными target значениями отделяются друг от друга, т.е. насколько хорошо усреднение эмбеддингов слов предложения передает информацию о предложении.

Для визуализации векторов надо получить их проекцию на плоскость. Сделаем это с помощью `PCA`. Если хотите, можете вместо PCA использовать TSNE: так у вас получится более точная проекция на плоскость (а значит, более информативная, т.е. отражающая реальное положение векторов твитов в пространстве). Но TSNE будет работать намного дольше.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


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


%matplotlib inline
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_source)

    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"]]
    )
```



Скорее всего, на визуализации нет четкого разделения твитов между классами. Это значит, что по полученным нами векторам твитов не так-то просто определить, к какому классу твит пренадлежит. Значит, 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(targets)}


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, shuffle=True, drop_last=True, collate_fn=average_emb
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=False, drop_last=False, collate_fn=average_e
```

Определим функции для тренировки и теста модели:

```
from tqdm.notebook import tqdm
import torch.nn as nn
import sklearn


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)

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

```
    mean_acc = 0
    model.eval()
    with torch.no_grad():
        for batch in pbar:
            features = batch["features"].to(device)
            targets = batch["targets"].to(device)

            # Получи предсказания модели
            output = model(features)
            # Посчитай лосс
            loss = criterion(output, targets)
            probs = nn.Softmax(dim = 1)(output)
            test_pred_labels = torch.tensor([x.argmax() for x in probs])
            # Посчитай точность модели
            acc = sklearn.metrics.accuracy_score(targets.cpu().numpy(), test_pred_labels.numpy())

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

            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: {mean_acc / len(test_loader):.4}")

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

Создадим модель, оптимизатор и целевую функцию. Вы можете сами выбрать количество слоев в нейронной сети, ваш любимый оптимизатор и целевую функцию.

```
from torch.optim import Adam


# Не забудь поиграться с параметрами ;)
vector_size = dev.word2vec.vector_size
num_classes = 2
lr = 1e-2
num_epochs = 5

# Твоя модель
model = nn.Sequential(
    nn.Linear(300, 50),
    nn.ReLU(),
    nn.Linear(50,100),
    nn.ReLU(),
    nn.Linear(100,2),
    # Нету слоя softmax так как используется CrossEntropyLoss()
)

model = model.cuda()
# Твой лосс
criterion = torch.nn.CrossEntropyLoss()
# Твой оптимайзер
optimizer = Adam(model.parameters(), lr=lr)
```

Наконец, обучим модель и протестируем её.

После каждой эпохи будем проверять качество модели на валидационной части датасета. Если метрика стала лучше, будем сохранять модель. **Подумайте, какая метрика (точность или лосс) будет лучше работать в этой задаче?**

```
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"]
```

```
Epoch 1. Train Loss: 0.5329: 100%  1000/1000 [01:51<00:00, 8.94it/s]

Test Loss: 0.5392, Test Acc: 0.7232: 100%  250/250 [10:12<00:00, 2.45s/it]

{'Test Loss': 0.5391562975645066, 'Test Acc': 0.72320703125}
Epoch 2. Train Loss: 0.5277: 100%  1000/1000 [01:50<00:00, 9.07it/s]

Test Loss: 0.5352, Test Acc: 0.7258: 100%  250/250 [07:49<00:00, 1.88s/it]

{'Test Loss': 0.5351700341701507, 'Test Acc': 0.72577734375}
Epoch 3. Train Loss: 0.5108: 100%  1000/1000 [01:48<00:00, 9.24it/s]

Test Loss: 0.532, Test Acc: 0.7279: 100%  250/250 [05:28<00:00, 1.31s/it]

{'Test Loss': 0.5319678614139557, 'Test Acc': 0.72789453125}
Epoch 4. Train Loss: 0.5123: 100%  1000/1000 [01:50<00:00, 9.02it/s]

Test Loss: 0.533. Test Acc: 0.7264: 100%  250/250 [03:03<00:00. 1.36it/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: 0.5533, Test Acc: 0.7168: 100%  313/313 [00:42<00:00, 7.33it/s]

{'Test Loss': 0.5321890258560547, 'Test Acc': 0.7269899410942492}
```

Embeddings for unknown words (8 баллов)

Пока что использовалась не вся информация из текста. Часть информации фильтровалось – если слова не было в словаре эмбеддингов, то мы просто превращали слово в нулевой вектор. Хочется использовать информацию по-максимуму. Поэтому рассмотрим другие способы обработки слов, которых нет в словаре. А именно:

- Для каждого незнакомого слова будем запоминать его контекст(слова слева и справа от этого слова). Эмбеддингом нашего незнакомого слова будет сумма эмбеддингов всех слов из его контекста. (4 балла)
- Для каждого слова текста получим его эмбеддинг из Tfidf с помощью `TfidfVectorizer` из sklearn. Итоговым эмбеддингом для каждого слова будет сумма двух эмбеддингов: предобученного и Tfidf-ного. Для слов, которых нет в словаре предобученных эмбеддингов, результирующий эмбеддинг будет просто полученный из Tfidf. (4 балла)

Реализуйте оба варианта **ниже**. Напишите, какой способ сработал лучше и ваши мысли, почему так получилось.

```
# 1 вариант
class TwitterDataset_V1(Dataset):
    def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str, word2vec: gensim.models.Word2Vec):
        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
        line = self.tokenizer.tokenize(text.lower())[2:]

        filtered_line = [w for w in line if all(c not in string.punctuation for c in w) and len(w) > 3]
        return filtered_line
        # Получи все токены из текста и профильтруй их

    def get_embeddings_(self, tokens):
        # Получи эмбеддинги слов и усредни их
        embeddings = []
        length = len(tokens)
        for i, w in enumerate(tokens):
            if w in word2vec:
                embeddings.append(self.word2vec.get_vector(w))
            else:
                context_indexs = np.arange(max(i - 2, 0), min(length, i + 3))
                context_indexs = context_indexs[context_indexs != i]
                try:
                    unknown_emb = sum([self.word2vec.get_vector(w) for w in tokens[context_indexs]]) / len(context_indexs)
                    print(unknown_emb)
                except:
                    unknown_emb = np.zeros(300)
                embeddings.append(unknown_emb)
            embeddings = [(emb - self.mean) / self.std for emb in embeddings]
            # Получи эмбеддинги слов и усредни их

            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_V1(dev_data, "text", "emotion", word2vec)
print(dev.data["emotion"].unique())

     [0 4]


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"]))

     1280


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(targets)}


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, shuffle=True, drop_last=True, collate_fn=average_emb
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=False, drop_last=False, collate_fn=average_e


from torch.optim import Adam
```

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

# Твоя модель
model = nn.Sequential(
    nn.Linear(300, 50),
    nn.ReLU(),
    nn.Linear(50,100),
    nn.ReLU(),
    nn.Linear(100,2),
    # Нету слоя softmax так как используется CrossEntropyLoss()
)

model = model.cuda()
# Твой лосс
criterion = torch.nn.CrossEntropyLoss()
# Твой оптимайзер
optimizer = Adam(model.parameters(), lr=lr)


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"]
```

```
Epoch 1. Train Loss: 0.5384: 100%  1000/1000 [02:00<00:00, 8.27it/s]

Test Loss: 0.5339, Test Acc: 0.7158: 100%  250/250 [00:35<00:00, 7.12it/s]

{'Test Loss': 0.5403606986999512, 'Test Acc': 0.72100390625}
Epoch 2. Train Loss: 0.5177: 100%  1000/1000 [01:54<00:00, 8.72it/s]

Test Loss: 0.5375, Test Acc: 0.7275: 100%  250/250 [00:34<00:00, 7.15it/s]

{'Test Loss': 0.5385386496782303, 'Test Acc': 0.7223359375}
```

```
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: 0.5897, Test Acc: 0.7012: 100%  313/313 [00:41<00:00, 7.50it/s]

{'Test Loss': 0.5401272769934072, 'Test Acc': 0.7198139227236422}
```

```
# 2 ВАРИАНТ

dev_data
```

	emotion	id	date	flag	user	text
0	0	1564500154	Mon Apr 20 03:47:08 PDT 2009	NO_QUERY	zourzouvillys	@Claire_Nelson i'm on the north devon coast th...
1	4	1957039896	Thu May 28 23:21:01 PDT 2009	NO_QUERY	twinkleval	@jhicks i will think of you on Sunday! Who ...
2	4	1557601862	Sun Apr 19 05:03:53 PDT 2009	NO_QUERY	andrewbulloch	Out in the garden with the kids debating wheth...
3	4	1823599026	Sat May 16 22:25:01 PDT 2009	NO_QUERY	vikchopra	@FrVerona thank u my love...u've shown me the ...
4	4	2186814798	Mon Jun 15 19:22:35 PDT 2009	NO_QUERY	jobrofan16	is with @jonasbrosfan1 going to buy LVATT tog...
...	...	...	...	...	...	...

```
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, word2vec: gensim.models.Word2Vec, weights: Dict[s
        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  * self.weights.get(token, 1) for token in tokens]

        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 get_tf_idf_(self):
        # Надо обучить tfidf на очищенном тексте. Но он принимает только список текстов, а не список списка токенов. Надо превратить
        tokenized_texts = list(map(lambda x: " ".join(self.get_tokens_(x)), self.data["text"]))
        tf_idf = TfidfVectorizer()
        tf_idf.fit(tokenized_texts)
        # Обучи tf-idf
        return dict(zip(tf_idf.get_feature_names(), tf_idf.idf_))


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


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"]))
```

```
1280
```

```
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, shuffle=True, drop_last=True, collate_fn=average_eml
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=False, drop_last=False, collate_fn=average_e


import torch.nn as nn
from torch.optim import Adam
from scipy.special import softmax


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

```
model = nn.Sequential(
    nn.Linear(300, 50),
    nn.ReLU(),
    nn.Linear(50,100),
    nn.ReLU(),
    nn.Linear(100,2),
)# Твоя модель
model = model.cuda()
criterion = torch.nn.CrossEntropyLoss()# Твой лосс
optimizer = Adam(model.parameters(), lr=lr)


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: 0.5558: 100%  1000/1000 [02:06<00:00, 7.91it/s]

Test Loss: 0.5473, Test Acc: 0.7266: 100%  250/250 [00:38<00:00, 6.47it/s]

{'Test Loss': 0.5642596063613892, 'Test Acc': 0.7026171875}
{'Test Loss': 0.5385386496782303, 'Test Acc': 0.7223359375}
Epoch 2. Train Loss: 0.5336: 100%  1000/1000 [02:05<00:00, 7.97it/s]

Test Loss: 0.5541, Test Acc: 0.7197: 100%  250/250 [00:37<00:00, 6.61it/s]

{'Test Loss': 0.5626786932945251, 'Test Acc': 0.70487890625}
{'Test Loss': 0.5385386496782303, 'Test Acc': 0.7223359375}
```

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

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: 0.5925, Test Acc: 0.6992: 100%  313/313 [00:46<00:00, 6.71it/s]

{'Test Loss': 0.5638687814386508, 'Test Acc': 0.7022794778354633}
```

```
#Выполнил Ильясов Тимур Камилевич
```

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

В целом при тесте получилось одно и тоже качество определеия. Точность в том и в другом способе почти одинакова. Это втом числе связано с тем, что была применена одинаковая схема нейронной сети. Улучшить качество модели к сожалению не получилось, так как PCA своей трансформацией так и не помог разделить фичи качественно, поэтому некоторые объекты, принадлежащие разным классам сливались в однои определять в таком случае итоговый ответ достаточно сложно даже с очень хорошим классификатором или Векторной машиной

Было опробовано достаточно много моделей, в основном полносвязные глубокие нейронные сети без дропаута и батчнорма. Различные функции активации, по типу ReLU или SoftMax. Также пробовалось соединять несколько предобученных блоков между собой и дотренировывать, но ничего хорошего не вышло :(Возможно стоило обучать на большем кол-ве эпох, но так как время было ограничено, то я попросту не успел дотренировать до нормальной точности данное решение, возможно стоило попробовать другой оптимизатор или функцию ошибки (чисто по приколу выбрал CrossEntropyLoss :)) Также lr я не так часто менял.