# 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
```

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
from torch import nn
import torch.optim as optim
wnl = nltk.WordNetLemmatizer()
nltk.download('stopwords')
from nltk.corpus import stopwords
stopWords = set(stopwords.words('english'))
from nltk.tokenize import word_tokenize, sent_tokenize
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
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"


import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/archive/training.1600000.processed.noemoticon.csv
```

```
data = pd.read_csv("/kaggle/input/archive/training.1600000.processed.noemoticon.csv", enco
```

Посмотрим на данные:

```
data.head()
```

| | emotion | id | date | flag | user | |
|---|---|---|---|---|---|---|
| **0** | 0 | 1467810369 | Mon Apr 06 22:19:45 PDT 2009 | NO_QUERY | _TheSpecialOne_ | @ |
| **1** | 0 | 1467810672 | Mon Apr 06 22:19:49 PDT 2009 | NO_QUERY | scotthamilton | is |
| **2** | 0 | 1467810917 | Mon Apr 06 22:19:53 PDT 2009 | NO_QUERY | mattycus | @Ke |
| **3** | 0 | 1467811184 | Mon Apr 06 22:19:57 PDT 2009 | NO_QUERY | ElleCTF | |
| **4** | 0 | 1467811193 | Mon Apr 06 22:19:57 PDT 2009 | NO_QUERY | Karoli | @ |

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
School email won't open  and I have geography stuff on there to revise! *Stupid Schoc
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  E
```

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

Чтобы сравнивать различные методы обработки текста/модели/прочее, разделим

```
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(len(line))
```

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

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

```
north devon coast next weeks will down devon again sometime hope though
12
```

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("glove-wiki-gigaword-50") #api.load("glove-wiki-gigaword-50") #api.loa
```

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

```
# print(emb_line)
```

```
(50,)
```

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 word
# print(sum(norm_emb_line))
print(sum(norm_emb_line).shape)
print([all(norm_emb_line[i] == emb_line[i]) for i in range(len(emb_line))])
print(len(norm_emb_line))
```

```
(50,)
[False, False, False, False, False, False, False, False, False, False, False, False]
12
```

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, word2v
        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):
        line = tokenizer.tokenize(text.lower())
        if line[0] == '@': line[1] = '@'
        line = ' '.join(w for w in line if all(c not in string.punctuation for c in w))
```

```
            tokens = ' '.join(wnl.lemmatize(word) for word in word_tokenize(text.lower())  if
                              )
            return tokens

        def get_embeddings_(self, tokens):

            embeddings = np.zeros((1, self.word2vec.vector_size))

            k = 0
            for w in tokens.split():
              if w in word2vec:
                embeddings = embeddings + word2vec.get_vector(w)
                k += 1

            if k != 0:
              embeddings = (embeddings/k- self.mean)/ self.std

            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)


    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(np.asarray(examples["features"]).shape)
```

```
1280
(1280, 50)
```

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"]))
print(np.asarray(examples["features"]).shape)
```

```
1280
(1280, 50)
```

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.

```
print(np.asarray(examples['features']).shape)
```

```
(1280, 50)
```

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

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

```
print(np.asarray(examples['features']).shape)
```

```
(1280, 50)
```

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

```
(1280, 2)
```

```
print(np.sum(np.asarray(examples['features'])))
print(sum(examples["transformed_features"]))
print(np.sum(examples["targets"]))
```

```
888.5250155866177
[6.93001212e-13 8.89288643e-13]
636
```

```
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
```

```
BokehJS 2.4.2 successfully loaded.
```

```
print(sum(examples["transformed_features"]))
```

```
[6.93001212e-13 8.89288643e-13]
```

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

```
Figure(id = '1004', …)
```

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 = 300
num_workers = 2

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=T
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=F
```

Определим функции для тренировки и теста модели:

```
def binary_acc(y_pred, y_test):
    y_pred_tag = torch.round(torch.sigmoid(y_pred))

    correct_results_sum = (y_pred_tag == y_test).sum().float()
    acc = correct_results_sum/y_test.shape[0]
    acc = torch.round(acc * 100)

    return acc


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:

        targets = batch['targets'].to(device)
        features = batch['features'].to(device)

        # Получи предсказания модели
        predictions = model(features)
        predictions = predictions.view(predictions.size(0))

        loss = criterion(predictions, targets.float())
        acc = binary_acc(predictions, targets.float())

        # Посчитай лосс
        # Обнови параметры модели
        loss.backward()
        optimizer.step()
```

```
            optimizer.zero_grad()

            pbar.set_description(f"Epoch {e + 1}. Train Loss: {loss:.4}, Test Acc: {acc:.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:
                targets = batch['targets'].to(device)
                features = batch['features'].to(device)


                predictions = model(features)
                predictions = predictions.view(predictions.size(0))

                loss = criterion(predictions, targets.float())
                acc = binary_acc(predictions, targets.float())

                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_a

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

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

```
class BinaryClassification(nn.Module):
    def __init__(self):
        super(BinaryClassification, self).__init__()
        # Number of input features 200
        hid_size = 50
        self.layer_1 = nn.Linear(hid_size, hid_size*2)
        self.layer_2 = nn.Linear(hid_size*2, hid_size*2)
        self.layer_out = nn.Linear(hid_size*2, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(hid_size*2)
        self.batchnorm2 = nn.BatchNorm1d(hid_size*2)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
```

```
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

```
model = BinaryClassification()
model.to(device)
print(model)
criterion = nn.BCEWithLogitsLoss()  #nn.BCEWithLogitsLoss(), nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
```

```
BinaryClassification(
  (layer_1): Linear(in_features=50, out_features=100, bias=True)
  (layer_2): Linear(in_features=100, out_features=100, bias=True)
  (layer_out): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
  (batchnorm1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
  (batchnorm2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
)
```

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
num_epochs = 6
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:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.5596354678098714, 'Test Acc': 70.58430913348946}
Epoch 2. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.5569766768945743, 'Test Acc': 70.64754098360656}
Epoch 3. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.5547985402775593, 'Test Acc': 70.79976580796253}
Epoch 4. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
```

## 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.

## Модель с предобученными эмбедингами + Tfidf

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.

```
import spacy
nlp = spacy.load("en_core_web_sm")

from sklearn.feature_extraction.text import TfidfVectorizer

import matplotlib.pyplot as plt


tokenizer = nltk.WordPunctTokenizer()

vectorizer = TfidfVectorizer(analyzer="word", tokenizer=nltk.word_tokenize,
```

```
        preprocessor=None, stop_words='english', max_features=None)
    tfidf = vectorizer.fit_transform(dev_data['text'])
    dictionary = vectorizer.get_feature_names_out()


    dict_tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))


    maximum = max(dict_tfidf.values())


    dict_tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_/maximum))


    class TwitterDataset_tfidf(Dataset):
        def __init__(self, data: pd.DataFrame, feature_column: str, target_column: str,
                     word2vec: gensim.models.Word2Vec, dict_tfidf : dict):
            self.tokenizer = nltk.WordPunctTokenizer()

            self.data = data

            self.feature_column = feature_column
            self.target_column = target_column

            self.word2vec = word2vec
            self.dict_tfidf = dict_tfidf

            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):
            line = tokenizer.tokenize(text.lower())
            if line[0] == '@': line[1] = '@'
            line = ' '.join(w for w in line if all(c not in string.punctuation for c in w))

            tokens = ' '.join(wnl.lemmatize(word) for word in word_tokenize(text.lower())  if
                              )
            return tokens

        def get_embeddings_(self, tokens):

            embeddings = np.zeros((1, self.word2vec.vector_size))
            embeddings_tfidf = np.zeros((1, self.word2vec.vector_size))
```

```
        k = 0
        j = 0
        for w in tokens.split():
          if w in word2vec:
            embeddings += word2vec.get_vector(w)
            k += 1
          if w in dict_tfidf:
            embeddings_tfidf += np.ones(word2vec.vector_size)*dict_tfidf[w]
            j += 1

        if k != 0:
          embeddings = (embeddings/k - self.mean)/ self.std
        if j != 0:
          embeddings += embeddings_tfidf/j
        if j != 0 and k != 0:
          embeddings = embeddings/2

        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_tfidf(dev_data, "text", "emotion", word2vec, dict_tfidf)


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(np.asarray(examples["features"]).shape)
```

```
1280
(1280, 50)
```

```
model_tfidf = BinaryClassification()
model_tfidf.to(device)
print(model_tfidf)
```

```
BinaryClassification(
  (layer_1): Linear(in_features=50, out_features=100, bias=True)
  (layer_2): Linear(in_features=100, out_features=100, bias=True)
  (layer_out): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.1, inplace=False)
```

```
  (batchnorm1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
  (batchnorm2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
)
```

```
batch_size = 300
num_workers = 2

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=T
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=F


best_metric = np.inf
num_epochs = 6
for e in range(num_epochs):
    training(model_tfidf, optimizer, criterion, train_loader, e, device)
    log = testing(model_tfidf, 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:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.76846694646172, 'Test Acc': 46.85831381733021}
Epoch 2. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.765742547269169, 'Test Acc': 46.65807962529274}
Epoch 3. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.7670423303331647, 'Test Acc': 46.7576112412178}
Epoch 4. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.7700118568379092, 'Test Acc': 46.62646370023419}
Epoch 5. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.767337334867942, 'Test Acc': 46.80679156908665}
Epoch 6. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
{'Test Loss': 0.7707017044552037, 'Test Acc': 46.43793911007026}
```

## Модель с дополненным по контексту словарем

Создадим дополнительно словарь новых слов и добавим его в словарь word2vec. В качестве вектора новых слов берем сумму ближайших слов слева и справа, если они

есть в словаре word2vec. В случае, если для слова не нашелся ненулевой вектор для

```
numb_of_words = dict()
new_words = dict()
vect = []
for i in range(len(dev_data['text'])):
  line = tokenizer.tokenize(dev_data["text"][i].lower())
  if line[0] == '@': line[1] = '@'
  line = [w for w in line if all(c not in string.punctuation for c in w)]

  for j in range(len(line)):
    if line[j] not in word2vec:
      vect = np.zeros(word2vec.vector_size)
      for k in range(j, len(line)):
        if line[k] in word2vec:
          vect = word2vec.get_vector(line[k])
          break
      if line[j] not in numb_of_words:
          numb_of_words[line[j]] = 0
      if sum(vect) != 0:
        numb_of_words[line[j]] += 1
      new_words[line[j]] = vect
      for k in range(0, j):
        if line[j-k] in word2vec:
          vect_right = word2vec.get_vector(line[j-k])
          break
      if sum(vect) != 0:
        numb_of_words[line[j]] += 1
      new_words[line[j]] = new_words[line[j]] + vect
      if sum(new_words[line[j]]) == 0:
        new_words.pop(line[j])
        numb_of_words.pop(line[j])


for w in new_words:
  if numb_of_words[w] != 0:
    new_words[w] = new_words[w]/numb_of_words[w]


word2vec.add_vectors(list(new_words.keys()), list(new_words.values()))


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


model_add = BinaryClassification()
model_add.to(device)
print(model_add)
```

```
     BinaryClassification(
       (layer_1): Linear(in_features=50, out_features=100, bias=True)
       (layer_2): Linear(in_features=100, out_features=100, bias=True)
       (layer_out): Linear(in_features=100, out_features=1, bias=True)
       (relu): ReLU()
       (dropout): Dropout(p=0.1, inplace=False)
       (batchnorm1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
```

```
  (batchnorm2): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_
)
```

```
batch_size = 300
num_workers = 2

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=T
valid_loader = DataLoader(valid, batch_size=batch_size, num_workers=num_workers, shuffle=F


best_metric = np.inf
num_epochs = 6
for e in range(num_epochs):
    training(model_add, optimizer, criterion, train_loader, e, device)
    log = testing(model_add, 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:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7116204735406388, 'Test Acc': 52.01053864168618}
 Epoch 2. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7118050962058386, 'Test Acc': 52.05269320843092}
 Epoch 3. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7112475088524874, 'Test Acc': 52.131147540983605}
 Epoch 4. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7115534299709758, 'Test Acc': 52.02459016393443}
 Epoch 5. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7114944520823012, 'Test Acc': 52.1288056206089}
 Epoch 6. Train Loss: 0:   0%|          | 0/3413 [00:00<?, ?it/s]
 Test Loss: 0, Test Acc: 0:   0%|          | 0/854 [00:00<?, ?it/s]
 {'Test Loss': 0.7103249367841233, 'Test Acc': 52.039812646370024}
```