


# 1 Архитектура RNN. Классификация текста.

Привет! В это семинаре мы познакомимся с задачей классификации текста, на примере поиска тематики новости, а также с двумя из основных архитектур рекуррентных нейросетей – RNN и GRU.

Нам потребуется одна библиотека от HuggingFace  под названием `datasets`. Она содержит большое число датасетов, которые используются в NLP.

Ввод [4]:

```
#!pip install datasets
```

executed in 4ms, finished 11:22:10 2021-10-09

Ввод [1]:

```
import random
import numpy as np

import nltk
import gensim.downloader as api

import torch
import torch.nn as nn
import datasets
```

executed in 5.00s, finished 11:23:27 2021-10-09

Ввод [2]:

```
# За детерминизм!
SEED = 0xDEAD
random.seed(SEED)
np.random.seed(SEED)
torch.random.manual_seed(SEED)
torch.cuda.random.manual_seed_all(SEED)
```

executed in 17ms, finished 11:34:11 2021-10-09

Загрузим датасет новостей: `AgNews`. В нем разделены тексты на 4 темы: `World`, `Sports`, `Business`, `Sci/Tech`. Посмотрим на структуру датасета и на примеры текстов:

Ввод [3]:

```
dataset = datasets.load_dataset("ag_news")  
dataset["train"]
```

executed in 1.52s, finished 11:34:37 2021-10-09

Using custom data configuration default  
Reusing dataset ag\_news (C:\Users\BIT\.cache\huggingface\datasets\ag\_news\default\0.0.0\bc2bcb40336ace1a0374767fc29bb0296cdaf8a6da7298436239c54d79180548)

100%

2/2 [00:00<00:00, 51.29it/s]

Out[3]:

```
Dataset(  
  features: ['text', 'label'],  
  num_rows: 120000  
)
```

Ввод [9]:

```
dataset["train"][0]
```

executed in 17ms, finished 11:39:20 2021-10-09

Out[9]:

```
{'text': "Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers, Wall Street's dwindling\\band of ultra-cynics, are seeing green again.",  
 'label': 2}
```

В dataset находятся train и test части датасета.

Ввод [5]:

```
dataset
```

executed in 6ms, finished 11:34:44 2021-10-09

Out[5]:

```
DatasetDict(  
  train: Dataset(  
    features: ['text', 'label'],  
    num_rows: 120000  
  )  
  test: Dataset(  
    features: ['text', 'label'],  
    num_rows: 7600  
  )  
)
```

Чтобы превращать текст из набора слов в набор векторов мы будем использовать предобученные эмбединги. Посмотрим на их список и выберем один из них.

Ввод [6]:

```
print("\n".join(api.info()['models'].keys()))
```

executed in 304ms, finished 11:35:47 2021-10-09

```
fasttext-wiki-news-subwords-300
conceptnet-numberbatch-17-06-300
word2vec-ruscorpora-300
word2vec-google-news-300
glove-wiki-gigaword-50
glove-wiki-gigaword-100
glove-wiki-gigaword-200
glove-wiki-gigaword-300
glove-twitter-25
glove-twitter-50
glove-twitter-100
glove-twitter-200
__testing_word2vec-matrix-synopsis
```

Ввод [7]:

```
word2vec = api.load("glove-twitter-50")
```

executed in 2m 36s, finished 11:38:24 2021-10-09

```
[=====] 100.0% 199.5/199.5MB do
wnloaded
```

Токенизируем наш текст с помощью NLTK.

Ввод [10]:

```
MAX_LENGTH=128
```

```
tokenizer = nltk.WordPunctTokenizer()
```

```
dataset = dataset.map( # обращаемся к dataset (train u test)
    lambda item: {
        "tokenized": tokenizer.tokenize(item["text"])[ :MAX_LENGTH] # создаём tokenized для
    }
)
```

executed in 13.2s, finished 11:39:37 2021-10-09

100%

120000/120000 [00:12<00:00,

9857.32ex/s]

100%

7600/7600 [00:00<00:00, 9817.31ex/s]

Ввод [11]:

```
dataset
```

executed in 15ms, finished 11:40:01 2021-10-09

Out[11]:

```
DatasetDict({
  train: Dataset({
    features: ['text', 'label', 'tokenized'],
    num_rows: 120000
  })
  test: Dataset({
    features: ['text', 'label', 'tokenized'],
    num_rows: 7600
  })
})
```

Ввод [16]:

```
dataset['train']['tokenized'][0]
```

executed in 7.86s, finished 11:42:58 2021-10-09

Out[16]:

```
['Wall',  
'St',  
'.',  
'Bears',  
'Claw',  
'Back',  
'Into',  
'the',  
'Black',  
'(',  
'Reuters',  
)',  
'Reuters',  
'-',  
'Short',  
'-',  
'sellers',  
',',  
'Wall',  
'Street',  
''',  
's',  
'dwindling',  
'\\',  
'band',  
'of',  
'ultra',  
'-',  
'cynics',  
',',  
'are',  
'seeing',  
'green',  
'again',  
'.']
```

Создадим мапинг из токенов в индексы

Ввод [17]:

```
word2idx = {word: idx for idx, word in enumerate(word2vec.index2word)} # ключ - слово, знач  
# word2vec.index2word - по индексам возвращает слово (лист слов)  
# word2idx - по слову возвращает индекс
```

executed in 331ms, finished 11:45:07 2021-10-09

Ввод [20]:

word2idx

executed in 46ms, finished 11:45:53 2021-10-09

Out[20]:

```
{'<user>': 0,
 '.': 1,
 ':': 2,
 'rt': 3,
 ',': 4,
 '<repeat>': 5,
 '<hashtag>': 6,
 '<number>': 7,
 '<url>': 8,
 '!': 9,
 'i': 10,
 'a': 11,
 '"': 12,
 'the': 13,
 '?': 14,
 'you': 15,
 'to': 16,
 '(' : 17.
```

Переведем токены в индексы

Ввод [26]:

```
def encode(word):
    if word in word2idx.keys():
        return word2idx[word]
    return word2idx["unk"]
```

executed in 9ms, finished 11:55:50 2021-10-09

Ввод [27]:

```
dataset = dataset.map(
    lambda item: { # item элемент словаря из словаря {'train':..., 'test':...}
        "features": [encode(word) for word in item["tokenized"]] # заменяем набор текста из
    }
)
```

executed in 46.5s, finished 11:56:37 2021-10-09

100%

120000/120000 [00:27&lt;00:00,

4267.99ex/s]

100%

7600/7600 [00:01&lt;00:00, 4390.21ex/s]

Ввод [28]:

dataset["train"][0]

executed in 21ms, finished 11:56:40 2021-10-09

Out[28]:

```
{'text': "Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-
sellers, Wall Street's dwindling\\band of ultra-cynics, are seeing green aga
in.",
'label': 2,
'tokenized': ['Wall',
'St',
'.',
'Bears',
'Claw',
'Back',
'Into',
'the',
'Black',
'(',
'Reuters',
')',
'Reuters',
'-',
'Short',
'-',
'sellers',
',',
'Wall',
'Street',
'',
's',
'dwindling',
'\\',
'band',
'of',
'ultra',
'-',
'cynics',
',',
'are',
'seeing',
'green',
'again',
'.'],
'features': [62980,
62980,
1,
62980,
62980,
62980,
62980,
13,
62980,
17,
62980,
20,
62980,
28,
62980,
```

```

28,
49286,
4,
62980,
62980,
48,
137,
214902,
370,
1645,
39,
8606,
28,
380053,
4,
70,
1321,
1745,
389,
1]]}

```

Ввод [29]:

```
dataset.remove_columns_(["text", "tokenized"]) # так как у нас есть фичи (индексы векторов)
```

executed in 22ms, finished 11:58:12 2021-10-09

C:\Users\BIT\AppData\Local\Temp\ipykernel\_7068\3782232894.py:1: FutureWarning: remove\_columns\_ is deprecated and will be removed in the next major version of datasets. Use DatasetDict.remove\_columns instead.  
dataset.remove\_columns\_(["text", "tokenized"])

Переведем в тензоры

Ввод [31]:

```
dataset.set_format(type='torch')
```

executed in 6ms, finished 12:01:45 2021-10-09

Ввод [32]:

```
dataset["train"][0]
```

executed in 40ms, finished 12:01:46 2021-10-09

Out[32]:

```

{'label': tensor(2),
 'features': tensor([ 62980,  62980,      1,  62980,  62980,  62980,  62980,
13,  62980,
                17,  62980,    20,  62980,    28,  62980,    28, 49286,
4,
                62980,  62980,    48,   137, 214902,   370,  1645,    39,    8
606,
                28, 380053,     4,    70,   1321,   1745,   389,    1])}]

```

Хотим склеить объекты разной длины в батчи. Для этого давайте напишем `collate_fn`.



Ввод [33]:

```
def collate_fn(batch): # torch.Long для дискретных матриц
    max_len = max(len(row["features"]) for row in batch) # для одинакового представления дан
    input_embeds = torch.empty((len(batch), max_len), dtype=torch.long) # матрица длина бат
    labels = torch.empty(len(batch), dtype=torch.long) # вектор лейблов
    for idx, row in enumerate(batch): # заполняем input_embeds данными; row словарь(featu
        to_pad = max_len - len(row["features"]) # кол-во элементов для допадинга
        input_embeds[idx] = torch.cat((row["features"], torch.zeros(to_pad))) # в соответ.
        labels[idx] = row["label"] # в соответствующий по индексу элемент вектора лейблов в
    return {"features": input_embeds, "labels": labels}
```

executed in 18ms, finished 12:01:48 2021-10-09

Ввод [53]:

```
from torch.utils.data import DataLoader

loaders = { # словарь('train':train_loader, 'test':test_loader)
    k: DataLoader(
        ds, shuffle=(k=="train"), batch_size=32, collate_fn=collate_fn
    ) for k, ds in dataset.items()
} # k - train, test; ds - словарь(features, labels)
```

executed in 6ms, finished 12:59:02 2021-10-09

## 1.1 CNN

Первая модель, которую мы рассмотрим: CNN. Одномерная конволюция достаточно хорошо справляется с задачей классификации. В конце надо собрать вектор текста с помощью `AdaptiveMaxPool1d` или `AdaptiveAvgPool1d`. Для классификации можно собрать любую Feed Forward Network.

Ввод [58]:

`!nvidia-smi`

executed in 118ms, finished 13:07:28 2021-10-09

Sat Oct 9 13:07:28 2021

```

+-----+
--+
| NVIDIA-SMI 472.12      Driver Version: 472.12      CUDA Version: 11.4
|
|-----+-----+-----+
--+
| GPU   Name               TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. EC
C |
| Fan   Temp   Perf   Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute
M. |
|                               |                    |           MIG
M. |
|=====+=====+=====+
==|
|    0  NVIDIA GeForce ... WDDM | 00000000:01:00.0 On |           N/
A |
| N/A    44C    P8      2W /  N/A | 2657MiB / 8192MiB |      0%    Defaul
t |
|                               |                    |           N/
A |
+-----+-----+-----+
--+

```

```

+-----+
--+
| Processes:
|
| GPU   GI    CI          PID    Type    Process name                        GPU Memor
y |
|       ID    ID                                   Usage
|
|=====+=====+=====+
==|
|    0   N/A   N/A       7068      C     ...onda\envs\deep\python.exe        N/A
|
|    0   N/A   N/A      10408     C+G   Insufficient Permissions             N/A
|
+-----+-----+-----+
--+

```

Ввод [38]:

```

class CNNModel(nn.Module): # делаем свёртки на векторах
    def __init__(self, embed_size, hidden_size, num_classes=4): # может быть слова из 3 сло
        super().__init__()
        self.embeddings = nn.Embedding(len(word2idx), embedding_dim=embed_size) # 1193514 x
        self.cnn = nn.Sequential(
            nn.Conv1d(embed_size, hidden_size, kernel_size=3, padding=1, stride=2), # embed
            nn.BatchNorm1d(hidden_size),
            nn.ReLU(),
            nn.Conv1d(hidden_size, hidden_size, kernel_size=3, padding=1, stride=2),
            nn.BatchNorm1d(hidden_size),
            nn.ReLU(),
            nn.Conv1d(hidden_size, hidden_size, kernel_size=3, padding=1, stride=2),
            nn.BatchNorm1d(hidden_size),
            nn.ReLU(),
            nn.AdaptiveMaxPool1d(1), # берёт максимум по всем каналам, изменяя только после
            nn.Flatten(),
        )
        self.cl = nn.Sequential(
            nn.Linear(hidden_size, num_classes) # (batch_size x hidden_size x 1) -> (batch_
        )

    def forward(self, x): # x - batch_size x seq_len (строки-предложения, которые являются
        x = self.embeddings(x) # подаются индексы получаются эмбединги (batch_size, seq_l
        x = x.permute(0, 2, 1) # меняет размерности в соответствующем порядке цифр (batch_si
        x = self.cnn(x) #Conv-BN-RELU+Conv-BN-RELU+Conv-BN-RELU+AMaxPool
        prediction = self.cl(x) # batch_size x num_classes
        return prediction

```

< executed in 20ms, finished 12:56:34 2021-10-09 >

Ввод [39]:

```

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

model = CNNModel(word2vec.vector_size, 50).to(device) # 50 - hidden_size - на сколько канал
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

num_epochs = 1

```

< executed in 3.59s, finished 12:56:41 2021-10-09 >

Подготовим функцию для обучения модели:

Ввод [43]:

```

from tqdm.notebook import tqdm, trange

def training(model, criterion, optimizer, num_epochs, loaders, max_grad_norm=2):
    for e in trange(num_epochs, leave=False):
        model.train()
        num_iter = 0
        pbar = tqdm(loaders["train"], leave=False)
        for batch in pbar:
            optimizer.zero_grad()
            input_embeds = batch["features"].to(device)
            labels = batch["labels"].to(device)
            prediction = model(input_embeds)
            loss = criterion(prediction, labels)
            loss.backward()
            if max_grad_norm is not None:
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
            optimizer.step()
            num_iter += 1
        valid_loss = 0
        valid_acc = 0
        num_iter = 0
        model.eval()
        with torch.no_grad():
            correct = 0
            num_objs = 0
            for batch in loaders["test"]:
                input_embeds = batch["features"].to(device)
                labels = batch["labels"].to(device)
                prediction = model(input_embeds)
                valid_loss += criterion(prediction, labels)
                correct += (labels == prediction.argmax(-1)).float().sum() # argmax(-1) - (
                num_objs += len(labels)
                num_iter += 1

        print(f"Valid Loss: {valid_loss / num_iter}, accuracy: {correct/num_objs}")

```

<
>

executed in 12ms, finished 12:57:39 2021-10-09

Ввод [54]:

```
training(model, criterion, optimizer, num_epochs, loaders)
```

executed in 1m 43.4s, finished 13:00:48 2021-10-09

Valid Loss: 0.4036913812160492, accuracy: 0.8590788841247559

## 1.2 RNN

Вторая модель: RNN. Это рекуррентная сеть, она использует скрытое состояние из прошлой итерации для создания нового. Это описывается с помощью формул:

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

Напишем этот модуль на Torch !

Ввод [121]:

```

NN(nn.Module):
    __init__(self, embed_size, hidden_size):
        super().__init__()

        self.embed_size = embed_size
        self.hidden_size = hidden_size

        self.w_h = nn.Parameter(torch.rand(hidden_size, hidden_size)) # W.T
        self.b_h = nn.Parameter(torch.rand((1, hidden_size)))
        self.w_x = nn.Parameter(torch.rand(embed_size, hidden_size)) # V.T # (batch_size x embed_size)
        self.b_x = nn.Parameter(torch.rand(1, hidden_size))

    forward(self, x, hidden=None):
        """
        x - torch.FloatTensor with the shape (bs, seq_length, emb_size)
        hidden - torch.FloatTensor with the shape (bs, hidden_size)
        return: torch.FloatTensor with the shape (bs, hidden_size)
        """
        if hidden is None:
            hidden = torch.zeros((x.size(0), self.hidden_size)).to(x.device) # (batch_size x hidden_size)
            seq_length = x.size(1)
            for cur_idx in range(seq_length):
                hidden = torch.tanh( # hidden задан рекуррентно
                    x[:, cur_idx] @ self.w_x + self.b_x + hidden @ self.w_h + self.b_h # x[:, cur_idx] @ self.w_h + self.b_h
                ) # x[:, cur_idx] @ self.w_x -- (batch_size x embed_size * embed_size x hidden_size -> hidden_size)
            return hidden

```

executed in 17ms, finished 14:21:37 2021-10-09

Ввод [122]:

```

class RNNModel(nn.Module):
    def __init__(self, embed_size, hidden_size, num_classes=4):
        super().__init__()
        self.embeddings = nn.Embedding(len(word2idx), embed_size)
        self.rnn = RNN(embed_size, hidden_size)
        self.cls = nn.Linear(hidden_size, num_classes) # (batch_size x hidden_size) -> (batch_size x num_classes)

    def forward(self, x):
        x = self.embeddings(x) # (batch_size, seq_len, embed_dim)
        hidden = self.rnn(x)
        output = self.cls(hidden)
        return output

```

executed in 7ms, finished 14:21:37 2021-10-09

Ввод [123]:

```

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

model = RNNModel(word2vec.vector_size, 50).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

num_epochs = 1
max_grad_norm = 1.0

```

executed in 679ms, finished 14:21:38 2021-10-09

Ввод [124]:

```
training(model, criterion, optimizer, num_epochs, loaders, max_grad_norm)
```

executed in 4m 54s, finished 14:26:33 2021-10-09

Valid Loss: 1.4723517894744873, accuracy: 0.2510526180267334

## 1.3 GRU

Третья модель: GRU. Она усложненная версия RNN. Главная идея GRU: гейты. Так реализуется "память" модели – она маскирует часть старого скрытого состояния, создавая на этом месте новое. Модель GRU описывается следующим образом:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn}))$$

$$h_t = (1 - z_t) * n_t + z_t * h_{(t-1)}$$

Ввод [125]:

```

class GRU(nn.Module):
    def __init__(self, embed_size, hidden_size):
        super().__init__()

        self.embed_size = embed_size
        self.hidden_size = hidden_size
# параметры для r
        self.w_rh = nn.Parameter(torch.rand(hidden_size, hidden_size))
        self.b_rh = nn.Parameter(torch.rand((1, hidden_size)))
        self.w_rx = nn.Parameter(torch.rand(embed_size, hidden_size))
        self.b_rx = nn.Parameter(torch.rand(1, hidden_size))
# параметры для z
        self.w_zh = nn.Parameter(torch.rand(hidden_size, hidden_size))
        self.b_zh = nn.Parameter(torch.rand((1, hidden_size)))
        self.w_zx = nn.Parameter(torch.rand(embed_size, hidden_size))
        self.b_zx = nn.Parameter(torch.rand(1, hidden_size))
# параметры для n
        self.w_nh = nn.Parameter(torch.rand(hidden_size, hidden_size))
        self.b_nh = nn.Parameter(torch.rand((1, hidden_size)))
        self.w_nx = nn.Parameter(torch.rand(embed_size, hidden_size))
        self.b_nx = nn.Parameter(torch.rand(1, hidden_size))

    def forward(self, x, hidden = None):
        """
        x - torch.FloatTensor with the shape (bs, seq_length, emb_size)
        hidden - torch.FloatTensor with the shape (bs, hidden_size)
        return: torch.FloatTensor with the shape (bs, hidden_size)
        """
        if hidden is None:
            hidden = torch.zeros((x.size(0), self.hidden_size)).to(x.device) # (batch_size,

        for cur_idx in range(x.size(1)):
            r = torch.sigmoid(
                x[:, cur_idx] @ self.w_rx + self.b_rx + hidden @ self.w_rh + self.b_rh
            )
            z = torch.sigmoid(
                x[:, cur_idx] @ self.w_zx + self.b_zx + hidden @ self.w_zh + self.b_zh
            )
            n = torch.tanh(
                x[:, cur_idx] @ self.w_nx + self.b_nx + r * (hidden @ self.w_nh + self.b_nh)
            )
            hidden = (1 - z) * n + z * hidden

        return hidden

```

executed in 20ms, finished 14:31:24 2021-10-09

Ввод [126]:

```
class GRUModel(nn.Module):
    def __init__(self, embed_size, hidden_size, num_classes=4):
        super().__init__()
        self.embed = nn.Embedding(len(word2idx), embed_size)
        self.gru = GRU(embed_size, hidden_size)
        self.cls = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = self.embed(x)
        hidden = self.gru(x)
        output = self.cls(hidden)
        return output
```

executed in 16ms, finished 14:31:25 2021-10-09

Ввод [127]:

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

model = GRUModel(word2vec.vector_size, 50).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

num_epochs = 1
max_grad_norm = 1.0
```

executed in 663ms, finished 14:31:26 2021-10-09

Ввод [128]:

```
training(model, criterion, optimizer, num_epochs, loaders, max_grad_norm)
```

executed in 13m 38s, finished 14:45:03 2021-10-09

Valid Loss: 0.8999207019805908, accuracy: 0.6455262899398804

## 1.4 GRU + Embeddings

Мы не просто так загрузили эмбединги в начале. Давай использовать их вместо случайной инициализации! Для этого надо немного переделать способ подачи данных в модель и добавить в модель модуль Embedding . По-экспериментирuem на модели GRU .

Ввод [135]:

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

model = GRUModel(word2vec.vector_size, 50).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

num_epochs = 1
max_grad_norm = 1.0
```

executed in 675ms, finished 15:14:49 2021-10-09



Ввод [133]:

```
rad():
dx in word2idx.items():
    in word2vec:
el.embed.weight[idx] = torch.from_numpy(word2vec.get_vector(word)) # на соответствующую строку
```

executed in 1m 56.0s, finished 15:00:44 2021-10-09

C:\Users\BIT\AppData\Local\Temp\ipykernel\_7068\1175556816.py:4: UserWarning: The given NumPy array is not writeable, and PyTorch does not support non-writeable tensors. This means you can write to the underlying (supposedly non-writeable) NumPy array using the tensor. You may want to copy the array to protect its data or make it writeable before converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered internally at ..\torch\csrc\utils\tensor\_numpy.cpp:141.)

```
model.embed.weight[idx] = torch.from_numpy(word2vec.get_vector(word))
```

Ввод [134]:

```
training(model, criterion, optimizer, num_epochs, loaders, max_grad_norm)
```

executed in 13m 42s, finished 15:14:26 2021-10-09

Valid Loss: 0.4243246912956238, accuracy: 0.8534210324287415

Попробуем заморозить эмбединги на первых итерациях обучения. Это поможет не сильно портить наши эмбединги на первых итерациях.

Ввод [136]:

```
def freeze_embeddings(model, req_grad=False):
    embeddings = model.embed
    for c_p in embeddings.parameters():
        c_p.requires_grad = req_grad
```

executed in 8ms, finished 15:15:00 2021-10-09

Ввод [137]:

```

def training_freeze(model, criterion, optimizer, num_epochs, loaders, max_grad_norm=2, num_
freeze_embeddings(model)
for e in trange(num_epochs, leave=False):
    model.train()
    num_iter = 0
    pbar = tqdm(loaders["train"], leave=False)
    for batch in pbar:
        if num_iter > num_freeze_iter and e < 1:
            freeze_embeddings(model, True)
            optimizer.zero_grad()
            input_embeds = batch["features"].to(device)
            labels = batch["labels"].to(device)
            prediction = model(input_embeds)
            loss = criterion(prediction, labels)
            loss.backward()
            if max_grad_norm is not None:
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_grad_norm)
            optimizer.step()
            num_iter += 1
    valid_loss = 0
    valid_acc = 0
    num_iter = 0
    model.eval()
    with torch.no_grad():
        correct = 0
        num_objs = 0
        for batch in loaders["test"]:
            input_embeds = batch["features"].to(device)
            labels = batch["labels"].to(device)
            prediction = model(input_embeds)
            valid_loss += criterion(prediction, labels)
            correct += (labels == prediction.argmax(-1)).float().sum()
            num_objs += len(labels)
            num_iter += 1

    print(f"Valid Loss: {valid_loss / num_iter}, accuracy: {correct/num_objs}")

```

executed in 23ms, finished 15:15:01 2021-10-09

Ввод [138]:

```

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

model = GRUModel(word2vec.vector_size, 50).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

num_epochs = 1
max_grad_norm = 1.0

```

executed in 635ms, finished 15:15:03 2021-10-09

Ввод [139]:

```
with torch.no_grad():  
    for word, idx in word2idx.items():  
        if word in word2vec:  
            model.embed.weight[idx] = torch.from_numpy(word2vec.get_vector(word))
```

executed in 1m 52.2s, finished 15:16:55 2021-10-09

Ввод [140]:

```
training_freeze(model, criterion, optimizer, num_epochs, loaders, max_grad_norm)
```

executed in 12m 52s, finished 15:29:47 2021-10-09

Valid Loss: 0.46335598826408386, accuracy: 0.8368420600891113

Обычно с заморозкой качество получается лучше, но не в этот раз

Эмбеддинги можно дообучать для того, чтобы разные по смыслу слова, но встречающиеся вместе часто, имели схожие эмбеддинги