Домашнее задание. Нейросетевая классификация текстов

В этом домашнем задании вам предстоит самостоятельно решить задачу классификации текстов на основе семинарского кода. Мы будем использовать датасет ag_news. Это датасет для классификации новостей на 4 темы: "World", "Sports", "Business", "Sci/Tech".

Установим модуль datasets, чтобы нам проще было работать с данными.

```
!pip install datasets
```

```
Requirement already satisfied: datasets in
/opt/conda/lib/python3.10/site-packages (2.1.0)
Requirement already satisfied: numpy>=1.17 in
/opt/conda/lib/python3.10/site-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=5.0.0 in
/opt/conda/lib/python3.10/site-packages (from datasets) (10.0.1)
Requirement already satisfied: dill in /opt/conda/lib/python3.10/site-
packages (from datasets) (0.3.6)
Requirement already satisfied: pandas in
/opt/conda/lib/python3.10/site-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/opt/conda/lib/python3.10/site-packages (from datasets) (2.28.2)
Requirement already satisfied: tqdm>=4.62.1 in
/opt/conda/lib/python3.10/site-packages (from datasets) (4.64.1)
Requirement already satisfied: xxhash in
/opt/conda/lib/python3.10/site-packages (from datasets) (3.2.0)
Requirement already satisfied: multiprocess in
/opt/conda/lib/python3.10/site-packages (from datasets) (0.70.14)
Requirement already satisfied: fsspec[http]>=2021.05.0 in
/opt/conda/lib/python3.10/site-packages (from datasets) (2023.5.0)
Requirement already satisfied: aiohttp in
/opt/conda/lib/python3.10/site-packages (from datasets) (3.8.4)
Requirement already satisfied: huggingface-hub<1.0.0,>=0.1.0 in
/opt/conda/lib/python3.10/site-packages (from datasets) (0.14.1)
Requirement already satisfied: packaging in
/opt/conda/lib/python3.10/site-packages (from datasets) (21.3)
Requirement already satisfied: responses<0.19 in
/opt/conda/lib/python3.10/site-packages (from datasets) (0.18.0)
Requirement already satisfied: attrs>=17.3.0 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(2.1.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
```

```
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(4.0.2)
Requirement already satisfied: yarl<2.0,>=1.0 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(1.9.1)
Requirement already satisfied: frozenlist>=1.1.1 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(1.3.3)
Requirement already satisfied: aiosignal>=1.1.2 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: filelock in
/opt/conda/lib/python3.10/site-packages (from huggingface-
hub<1.0.0,>=0.1.0->datasets) (3.12.0)
Requirement already satisfied: pyyaml>=5.1 in
/opt/conda/lib/python3.10/site-packages (from huggingface-
hub<1.0.0,>=0.1.0->datasets) (5.4.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/opt/conda/lib/python3.10/site-packages (from huggingface-
hub<1.0.0,>=0.1.0->datasets) (4.5.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/conda/lib/python3.10/site-packages (from packaging->datasets)
(3.0.9)
Requirement already satisfied: idna<4,>=2.5 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.19.0-
>datasets) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.19.0-
>datasets) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from requests>=2.19.0-
>datasets) (2023.5.7)
Requirement already satisfied: python-dateutil>=2.8.1 in
/opt/conda/lib/python3.10/site-packages (from pandas->datasets)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/lib/python3.10/site-packages (from pandas->datasets)
(2023.3)
Requirement already satisfied: six>=1.5 in
/opt/conda/lib/python3.10/site-packages (from python-dateutil>=2.8.1-
>pandas->datasets) (1.16.0)
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

Импорт необходимых библиотек

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import datasets

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from tqdm.auto import tqdm
from datasets import load_dataset
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords
import spacy

from collections import Counter
from typing import List
import string

import seaborn
seaborn.set(palette='summer')

nltk.download('punkt')
```

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

```
True
```

```
nltk.download('wordnet')
```

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

```
True
```

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

```
'cuda'
```

Подготовка данных

Для вашего удобства, мы привели код обработки датасета в ноутбуке. Ваша задача --- обучить модель, которая получит максимальное возможное качество на тестовой части.

```
# Загрузим датасет
dataset = datasets.load_dataset('ag_news')
```

{"model_id":"f090fe82a4c545afa47abee6c291608c","version_major":2,"version_minor":0}

Как и в семинаре, выполним следующие шаги:

- Составим словарь
- Создадим класс WordDataset
- Выделим обучающую и тестовую часть, создадим DataLoader-ы.

```
words = Counter()

for example in tqdm(dataset['train']['text']):
    # Приводим к нижнему регистру и убираем пунктуацию
    proccessed_text = example.lower().translate(
        str.maketrans('', '', string.punctuation))

    for word in word_tokenize(proccessed_text):
        words[word] += 1


vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
counter_threshold = 25

for char, cnt in words.items():
    if cnt > counter_threshold:
        vocab.add(char)

print(f'Размер словаря: {len(vocab)}')

word2ind = {char: i for i, char in enumerate(vocab)}
ind2word = {i: char for char, i in word2ind.items()}
```

{"model_id":"0a0a20f6cf3b40b6b779d0ab5aaf117f","version_major":2,"version_minor":0}

```
Размер словаря: 11842
```

```
class WordDataset:
    def __init__(self, sentences):
        self.data = sentences
        self.unk_id = word2ind['<unk>']
        self.bos_id = word2ind['<bos>']
        self.eos_id = word2ind['<eos>']
        self.pad_id = word2ind['<pad>']

    def __getitem__(self, idx: int) -> List[int]:
        processed_text = self.data[idx]['text'].lower().translate(
            str.maketrans('', '', string.punctuation))
```

```
        tokenized_sentence = [self.bos_id]
        tokenized_sentence += [
            word2ind.get(word, self.unk_id) for word in
word_tokenize(processed_text)
            ]
        tokenized_sentence += [self.eos_id]

        train_sample = {
            "text": tokenized_sentence,
            "label": self.data[idx]['label']
        }

        return train_sample

    def __len__(self) -> int:
        return len(self.data)


def collate_fn_with_padding(
    input_batch: List[List[int]], pad_id=word2ind['<pad>'],
max_len=256) -> torch.Tensor:
    seq_lens = [len(x['text']) for x in input_batch]
    max_seq_len = min(max(seq_lens), max_len)

    new_batch = []
    for sequence in input_batch:
        sequence['text'] = sequence['text'][:max_seq_len]
        for _ in range(max_seq_len - len(sequence['text'])):
            sequence['text'].append(pad_id)

        new_batch.append(sequence['text'])

    sequences = torch.LongTensor(new_batch).to(device)
    labels = torch.LongTensor([x['label'] for x in
input_batch]).to(device)

    new_batch = {
        'input_ids': sequences,
        'label': labels
    }

    return new_batch

train_dataset = WordDataset(dataset['train'])

np.random.seed(42)
idx = np.random.choice(np.arange(len(dataset['test'])), 5000)
eval_dataset = WordDataset(dataset['test'].select(idx))

batch_size = 32
train_dataloader = DataLoader(
```

```
    train_dataset, shuffle=True, collate_fn=collate_fn_with_padding,
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=collate_fn_with_padding,
batch_size=batch_size)
```

Постановка задачи

Ваша задача -- получить максимальное возможное accuracy на `eval_dataloader`. Ниже приведена функция, которую вам необходимо запустить для обученной модели, чтобы вычислить качество её работы.

```
def evaluate(model, eval_dataloader) -> float:
    """
    Calculate accuracy on validation dataloader.
    """

    predictions = []
    target = []
    with torch.no_grad():
        for batch in eval_dataloader:
            logits = model(batch['input_ids'])
            predictions.append(logits.argmax(dim=1))
            target.append(batch['label'])

    predictions = torch.cat(predictions)
    target = torch.cat(target)
    accuracy = (predictions == target).float().mean().item()

    return accuracy
```

Ход работы

Оценка за домашнее задание складывается из четырех частей:

Запуск базовой модели с семинара на новом датасете (1 балл)

На семинаре мы создали модель, которая дает на нашей задаче довольно высокое качество. Ваша цель --- обучить ее и вычислить `score`, который затем можно будет использовать в качестве бейзлайна.

В модели появится одно важное изменение: количество классов теперь равно не 2, а 4. Обратите на это внимание и найдите, что в коде создания модели нужно модифицировать, чтобы учесть это различие.

Проведение экспериментов по улучшению модели (2 балла за каждый эксперимент)

Чтобы улучшить качество базовой модели, можно попробовать различные идеи экспериментов. Каждый выполненный эксперимент будет оцениваться в 2 балла. Для получения полного балла за этот пункт вам необходимо выполнить по крайней мере 2 эксперимента. Не расстраивайтесь, если какой-то эксперимент не дал вам прироста к качеству: он все равно зачтется, если выполнен корректно.

Вот несколько идей экспериментов:

- **Модель RNN**. Попробуйте другие нейросетевые модели --- LSTM и GRU. Мы советуем обратить внимание на GRU, так как интерфейс этого класса ничем не отличается от обычной Vanilla RNN, которую мы использовали на семинаре.
- **Увеличение количества рекуррентных слоев модели**. Это можно сделать с помощью параметра `num_layers` в классе `nn.RNN`. В такой модели выходы первой RNN передаются в качестве входов второй RNN и так далее.
- **Изменение архитектуры после применения RNN**. В базовой модели используется агрегация со всех эмбеддингов. Возможно, вы захотите конкатенировать результат агрегации и эмбеддинг с последнего токена.
- **Подбор гиперпараметров и обучение до сходимости**. Возможно, для получения более высокого качества просто необходимо увеличить количество эпох обучения нейросети, а также попробовать различные гиперпараметры: размер словаря, `dropout_rate`, `hidden_dim`.

Обратите внимание, что главное правило проведения экспериментов --- необходимо совершать одно архитектурное изменение в одном эксперименте. Если вы совершите несколько изменений, то будет неясно, какое именно из изменений дало прирост к качеству.

Получение высокого качества (3 балла)

В конце вашей работы вы должны указать, какая из моделей дала лучший результат, и вывести качество, которое дает лучшая модель, с помощью функции `evaluate`. Ваша модель будет оцениваться по метрике `accuracy` следующим образом:

- $accuracy < 0.9$ --- 0 баллов;
- $0.9 \leq accuracy < 0.91$ --- 1 балл;
- $0.91 \leq accuracy < 0.915$ --- 2 балла;
- $0.915 \leq accuracy$ --- 3 балла.

Оформление отчета (2 балла)

В конце работы подробно опишите все проведенные эксперименты.

- Укажите, какие из экспериментов принесли улучшение, а какие --- нет.
- Проанализируйте графики сходимости моделей в проведенных экспериментах. Являются ли колебания качества обученных моделей существенными в зависимости от эпохи обучения, или же сходимость стабильная?

- Укажите, какая модель получилась оптимальной.

Желаем удачи!

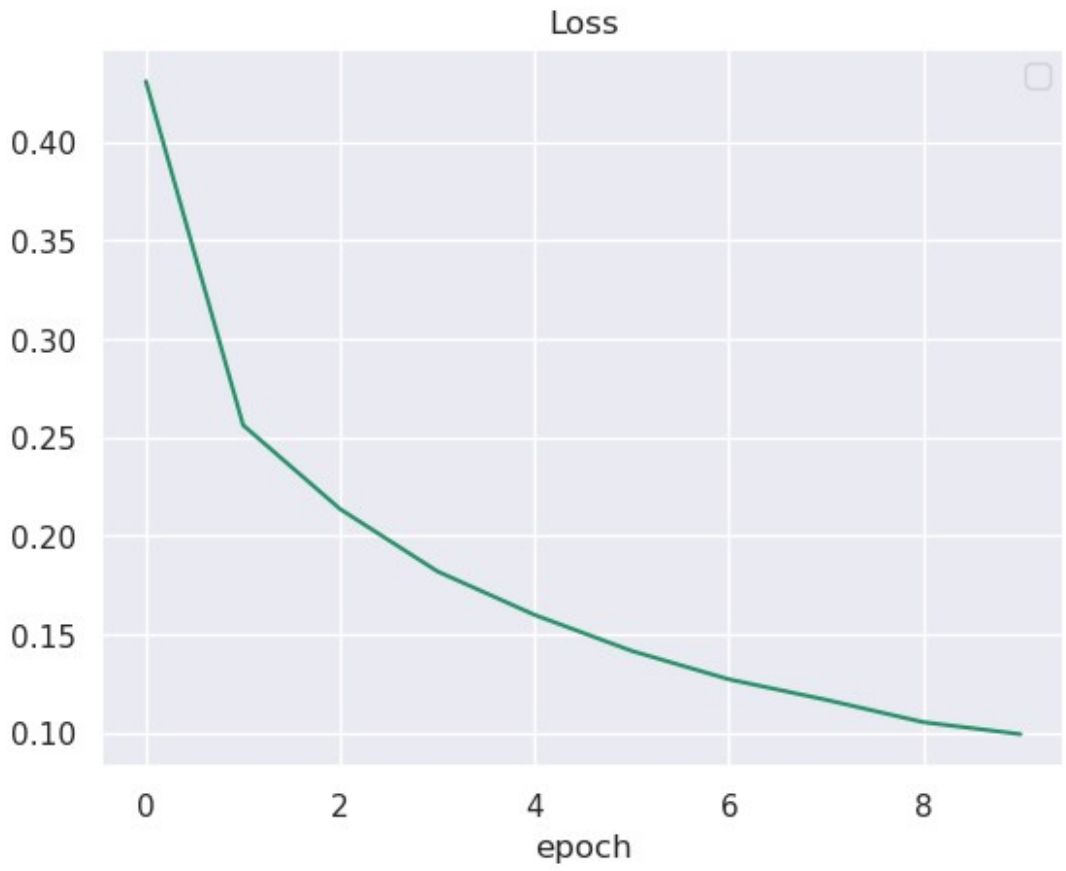
Запуск базовой модели с семинара на новом датасете

```
class CharLM(nn.Module):
    def __init__(self, hidden_dim: int, vocab_size: int, num_classes:
int = 4):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.RNN(hidden_dim, hidden_dim, batch_first=True)
        self.linear = nn.Linear(hidden_dim, hidden_dim)
        self.projection = nn.Linear(hidden_dim, num_classes)

        self.non_lin = nn.Tanh()
        self.dropout = nn.Dropout(p=0.1)

    def forward(self, input_batch) -> torch.Tensor:
        embeddings = self.embedding(input_batch)  # [batch_size,
seq_len, hidden_dim]
        output, _ = self.rnn(embeddings)  # [batch_size, seq_len,
hidden_dim]

        output = output.mean(dim=1) #[batch_size, hidden_dim]

        output = self.dropout(self.linear(output))  # [batch_size,
hidden_dim]
        prediction = self.projection(self.non_lin(output))  #
[batch_size, num_classes]

        return prediction

model = CharLM(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 10
eval_steps = len(train_dataloader) // 2

losses = []
acc = []

for epoch in range(num_epoch):
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
```

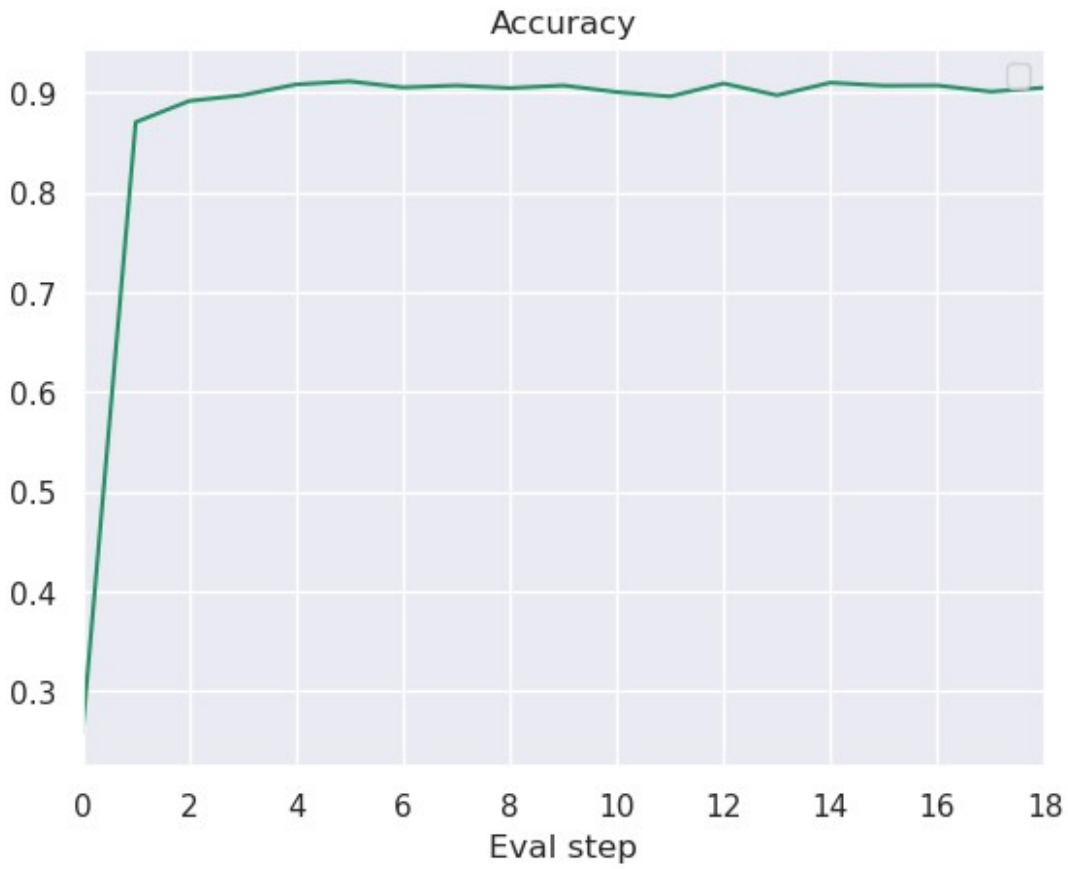
```
        loss.backward()
        optimizer.step()

        epoch_losses.append(loss.item())
        if i % eval_steps == 0:
            model.eval()
            acc.append(evaluate(model, eval_dataloader))
            model.train()

    losses.append(sum(epoch_losses) / len(epoch_losses))
```


```
{"model_id":"dd773537ec534a7782ed90cd07140392","version_major":2,"version_minor":0}

{"model_id":"6819968a624d48e1a4fbf43998184dfc","version_major":2,"version_minor":0}

{"model_id":"7b3fef27c15b4fa6b77d1154c6d417f1","version_major":2,"version_minor":0}

{"model_id":"c6e93ed3daa54f88a01a23533432d7b9","version_major":2,"version_minor":0}

{"model_id":"b20eb50bf9474d369aff819e0fe3a611","version_major":2,"version_minor":0}

{"model_id":"bc7e7347db8547be8692c1851d38f502","version_major":2,"version_minor":0}

{"model_id":"6c11b39d056441598bccac866bc945dd","version_major":2,"version_minor":0}

{"model_id":"f8d930ea4bd24e91816b3ec417a5d1e5","version_major":2,"version_minor":0}

{"model_id":"ce30cf4241bc4201840c2403c1ca8d2e","version_major":2,"version_minor":0}

{"model_id":"8a7dd7944a194c818fd6798c6bdf90d9","version_major":2,"version_minor":0}
```


```
plt.plot(losses)

plt.title('Loss')
plt.xlabel("epoch")
plt.legend()
plt.show()
```




```
plt.plot(acc)

plt.title('Accuracy')
plt.xlabel("Eval step")
plt.legend()
plt.xlim(0, 18)
plt.show()
```




```
acc[np.argmax(acc)], np.argmax(acc)
```

```
(0.9115999937057495, 5)
```

Модель запустили на новом датасете, видно что удалось достичь максимальной accuracy в 0.9116 на ~3 эпохе.

Проведение экспериментов по улучшению модели

Попробуем архитектуру с LSTM

```
class LSTMCharLM(nn.Module):
    def __init__(self, hidden_dim: int, vocab_size: int, num_classes:
int = 4):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.lstm = nn.LSTM(hidden_dim, hidden_dim, batch_first=True)
        self.linear = nn.Linear(hidden_dim, hidden_dim)
        self.projection = nn.Linear(hidden_dim, num_classes)

        self.non_lin = nn.Tanh()
        self.dropout = nn.Dropout(p=0.1)
```

```
    def forward(self, input_batch) -> torch.Tensor:
        embeddings = self.embedding(input_batch)  # [batch_size,
seq_len, hidden_dim]
        output, _ = self.lstm(embeddings)  # [batch_size, seq_len,
hidden_dim]

        output = output.mean(dim=1) #[batch_size, hidden_dim]

        output = self.dropout(self.linear(output))  # [batch_size,
hidden_dim]
        prediction = self.projection(self.non_lin(output))  #
[batch_size, num_classes]

        return prediction

model = LSTMCharLM(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 10
eval_steps = len(train_dataloader) // 2

losses = []
acc = []

for epoch in range(num_epoch):
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
        loss.backward()
        optimizer.step()

        epoch_losses.append(loss.item())
        if i % eval_steps == 0:
            model.eval()
            acc.append(evaluate(model, eval_dataloader))
            model.train()

    losses.append(sum(epoch_losses) / len(epoch_losses))
```


```
{"model_id":"4b84042015d041df8c645f5e2e85004e","version_major":2,"vers
ion_minor":0}

{"model_id":"353083f489d14b418865da8c700b9cdb","version_major":2,"vers
ion_minor":0}
```

{"model_id":"684bba967b6f4fc5aee4280b91f668bc","version_major":2,"version_minor":0}

{"model_id":"ddb3102974b74628983ed282b0001c08","version_major":2,"version_minor":0}

{"model_id":"df21a65506b546fd96d544d61d357c1e","version_major":2,"version_minor":0}

{"model_id":"9458d242df8849d2b000bea8aaf77cdb","version_major":2,"version_minor":0}

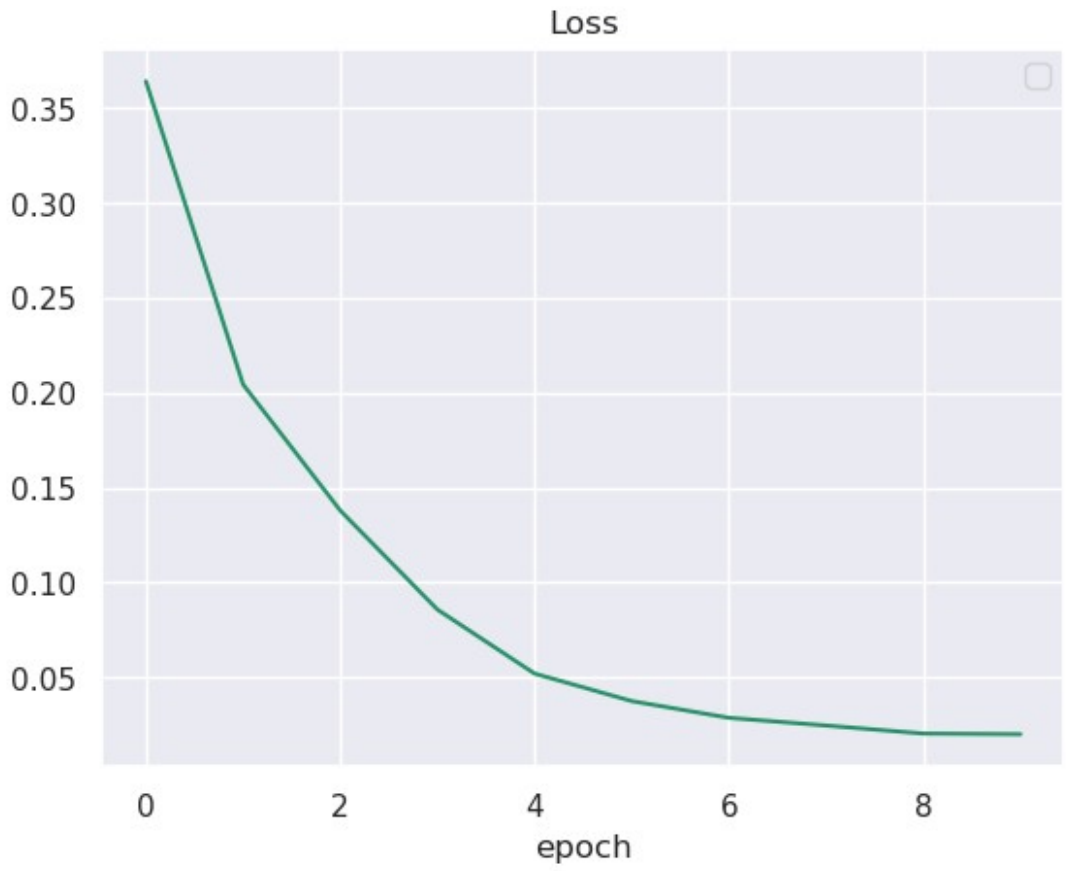
{"model_id":"00e57fee27cc42378f2e04a284db28c8","version_major":2,"version_minor":0}

{"model_id":"4418cf962a96490f940e55fa7d9ae11b","version_major":2,"version_minor":0}

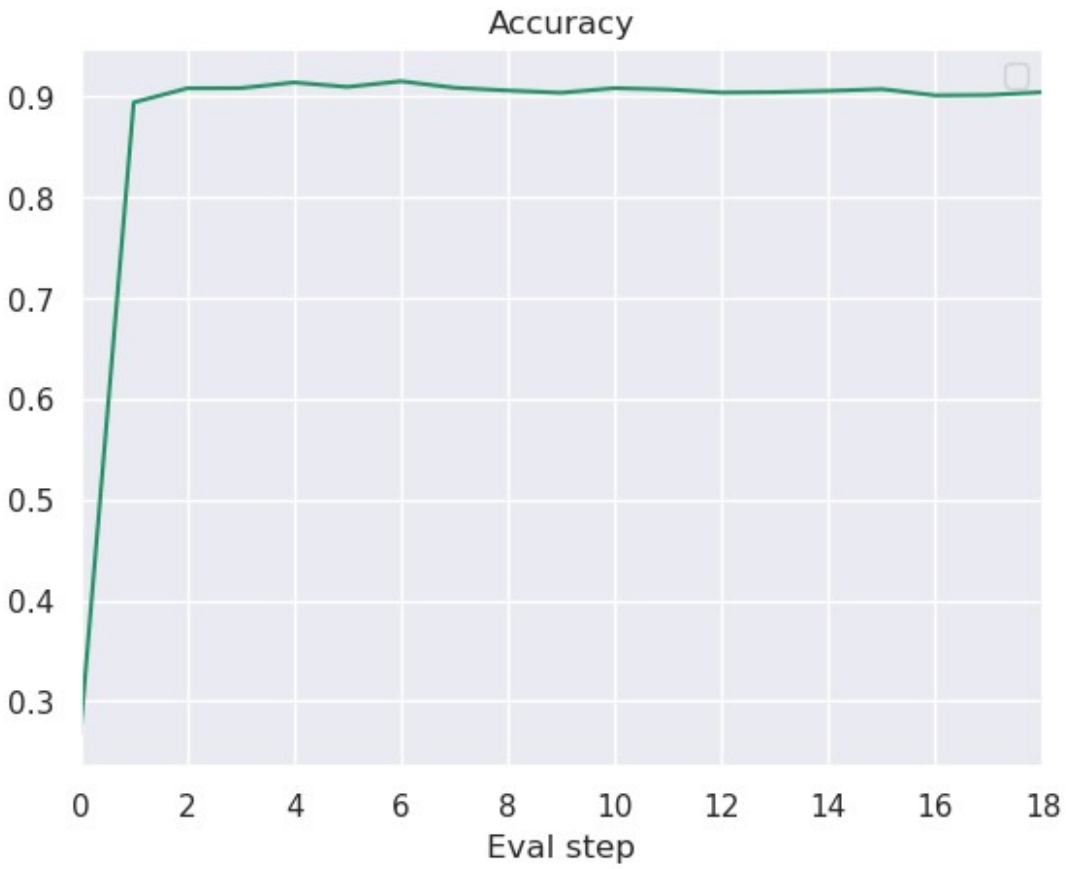
{"model_id":"3e088b042db643b3a2835f9a00149f46","version_major":2,"version_minor":0}

{"model_id":"b6e72b7916d144adba143b69d0df6e72","version_major":2,"version_minor":0}

```
plt.plot(losses)

plt.title('Loss')
plt.xlabel("epoch")
plt.legend()
plt.show()
```




```
plt.plot(acc)

plt.title('Accuracy')
plt.xlabel("Eval step")
plt.legend()
plt.xlim(0, 18)
plt.show()
```




```
acc[np.argmax(acc)], np.argmax(acc)
```

```
(0.9145999550819397, 6)
```

Видно, что нам удалось немного увеличить accuracy путем использования LSTM до 0.9146 на ~3 эпохе.

Попробуем увеличить количество слоев RNN

```
class CharLM2(nn.Module):
    def __init__(self, hidden_dim: int, vocab_size: int, num_classes:
int = 4):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.RNN(hidden_dim, hidden_dim, batch_first=True,
num_layers=2)
        self.linear = nn.Linear(hidden_dim, hidden_dim)
        self.projection = nn.Linear(hidden_dim, num_classes)

        self.non_lin = nn.Tanh()
        self.dropout = nn.Dropout(p=0.1)

    def forward(self, input_batch) -> torch.Tensor:
```

```
        embeddings = self.embedding(input_batch)  # [batch_size,
seq_len, hidden_dim]
        output, _ = self.rnn(embeddings)  # [batch_size, seq_len,
hidden_dim]

        output = output.mean(dim=1) #[batch_size, hidden_dim]

        output = self.dropout(self.linear(output))  # [batch_size,
hidden_dim]
        prediction = self.projection(self.non_lin(output))  #
[batch_size, num_classes]

        return prediction

model = CharLM2(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 10
eval_steps = len(train_dataloader) // 2

losses = []
acc = []

for epoch in range(num_epoch):
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
        loss.backward()
        optimizer.step()

        epoch_losses.append(loss.item())
        if i % eval_steps == 0:
            model.eval()
            acc.append(evaluate(model, eval_dataloader))
            model.train()

    losses.append(sum(epoch_losses) / len(epoch_losses))
```


```
{"model_id":"2c457d6ba4764aa5b81cf3a1b23dc923","version_major":2,"vers
ion_minor":0}

{"model_id":"8b7dfbab68574803866ef6b5fcb923a7","version_major":2,"vers
ion_minor":0}

{"model_id":"32e05d1e008b4c29bc1016cbf82027b6","version_major":2,"vers
ion_minor":0}
```

{"model_id":"3ffd0c03af694798b72e4e5106b585c6","version_major":2,"version_minor":0}

{"model_id":"fa6c8d7daff941909d99fadb058e7ca7","version_major":2,"version_minor":0}

{"model_id":"0f792157d43e40daa9362cd45ea6b6c2","version_major":2,"version_minor":0}

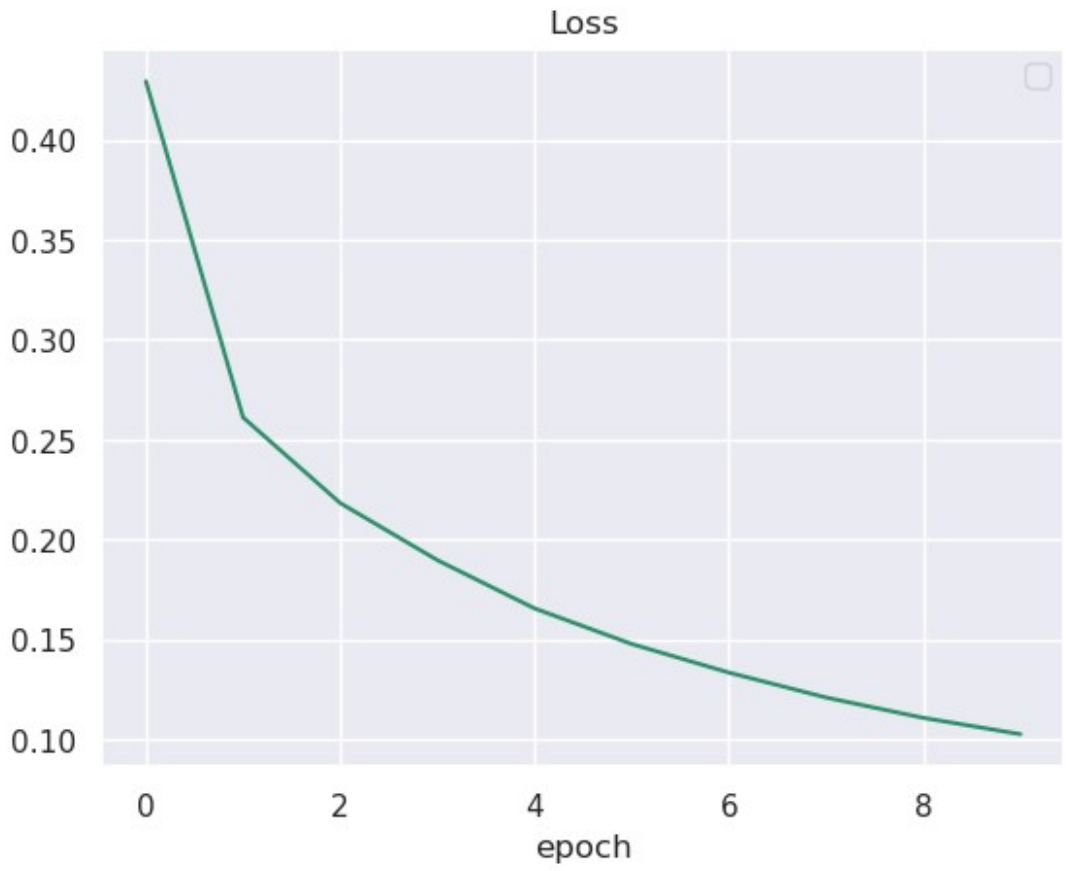
{"model_id":"17ec5424c98b4888ae44fbce36c4bd88","version_major":2,"version_minor":0}

{"model_id":"a6559906f622414c92850a6f36784d6e","version_major":2,"version_minor":0}

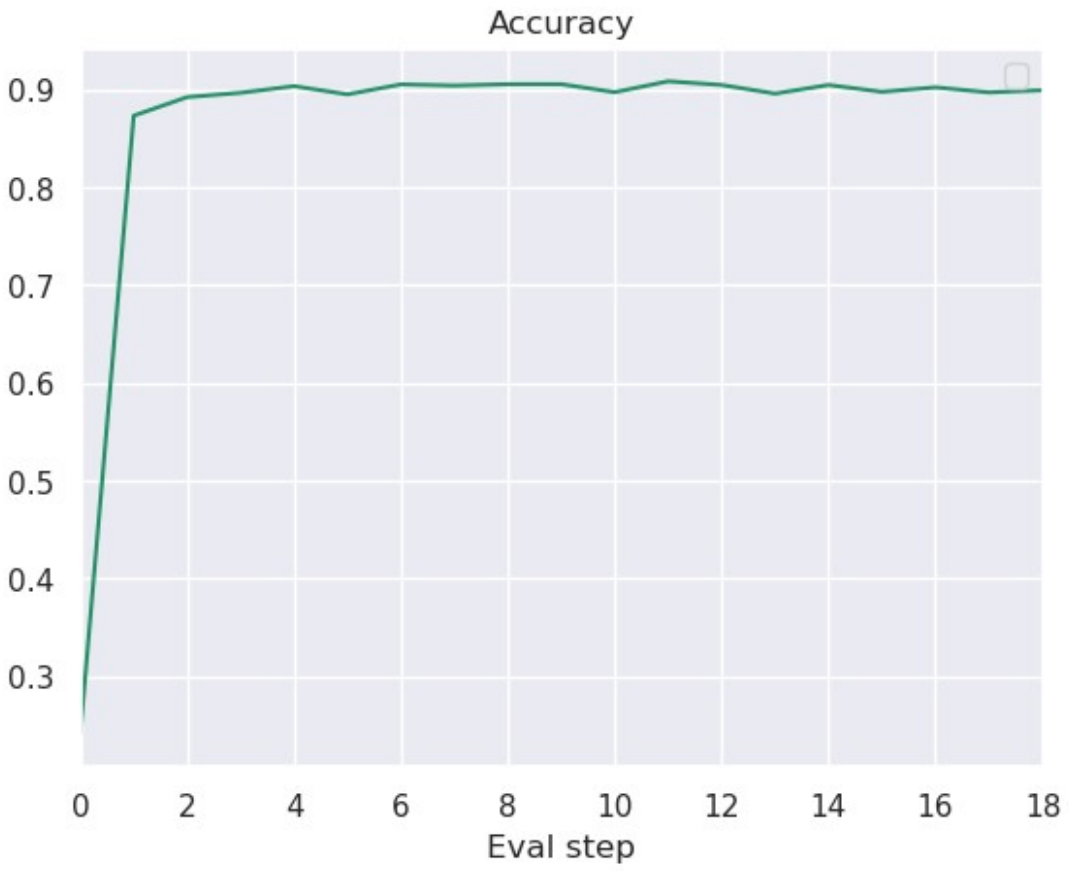
{"model_id":"d1c20158334d41de8f2af6844a262fa1","version_major":2,"version_minor":0}

{"model_id":"70468983602240d9ba59fb5a986b89d2","version_major":2,"version_minor":0}

```
plt.plot(losses)

plt.title('Loss')
plt.xlabel("epoch")
plt.legend()
plt.show()
```




```
plt.plot(acc)

plt.title('Accuracy')
plt.xlabel("Eval step")
plt.legend()
plt.xlim(0, 18)
plt.show()
```




```
acc[np.argmax(acc)], np.argmax(acc)
```

```
(0.9081999659538269, 11)
```

Видно, что в случае с RNN увеличение количества слоев приводит к уменьшению accuracy.

Попробуем поработать со словарем

Попробуем сначала просто убрать стоп-слова.

```
en = spacy.load('en_core_web_sm')
stopwords_spacy = en.Defaults.stop_words

nltk.download('stopwords')

stopwords_nltk = stopwords.words("english")
```

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

```
stopwords_united = set(stopwords_spacy).union(stopwords_nltk)

dataset = datasets.load_dataset('ag_news')
```

{"model_id":"f0f2697670024de5bcb0c63f7df1c23e","version_major":2,"version_minor":0}

{"model_id":"a0ac5a6a23de4029b2572b0c0981526d","version_major":2,"version_minor":0}

```
Downloading and preparing dataset ag_news/default (download: 29.88
MiB, generated: 30.23 MiB, post-processed: Unknown size, total: 60.10
MiB) to
/root/.cache/huggingface/datasets/ag_news/default/0.0.0/bc2bcb40336ace
1a0374767fc29bb0296cdaf8a6da7298436239c54d79180548...
```

{"model_id":"6f77502668d34fb888da480d3ead1775","version_major":2,"version_minor":0}

{"model_id":"3ed98bc65f314d43a32f493222e8a72d","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

```
Dataset ag_news downloaded and prepared to
/root/.cache/huggingface/datasets/ag_news/default/0.0.0/bc2bcb40336ace
1a0374767fc29bb0296cdaf8a6da7298436239c54d79180548. Subsequent calls
will reuse this data.
```

{"model_id":"075636ea78ed4fadbc8d06d06e4d6bd4","version_major":2,"version_minor":0}

```
words = Counter()

for example in tqdm(dataset['train']['text']):
    # Приводим к нижнему регистру и убираем пунктуацию
    proccessed_text = example.lower().translate(
        str.maketrans('', '', string.punctuation))

    words_tokenized = [word for word in word_tokenize(proccessed_text)
if word not in stopwords_united]

    for word in words_tokenized:
        words[word] += 1

vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
counter_threshold = 25

for char, cnt in words.items():
    if cnt > counter_threshold:
        vocab.add(char)

print(f'Размер словаря: {len(vocab)}')
```

```
word2ind = {char: i for i, char in enumerate(vocab)}
ind2word = {i: char for char, i in word2ind.items()}
```

{"model_id":"b5627a037c7f433589b3bfb777ef95ad","version_major":2,"version_minor":0}

```
Размер словаря: 11559
```

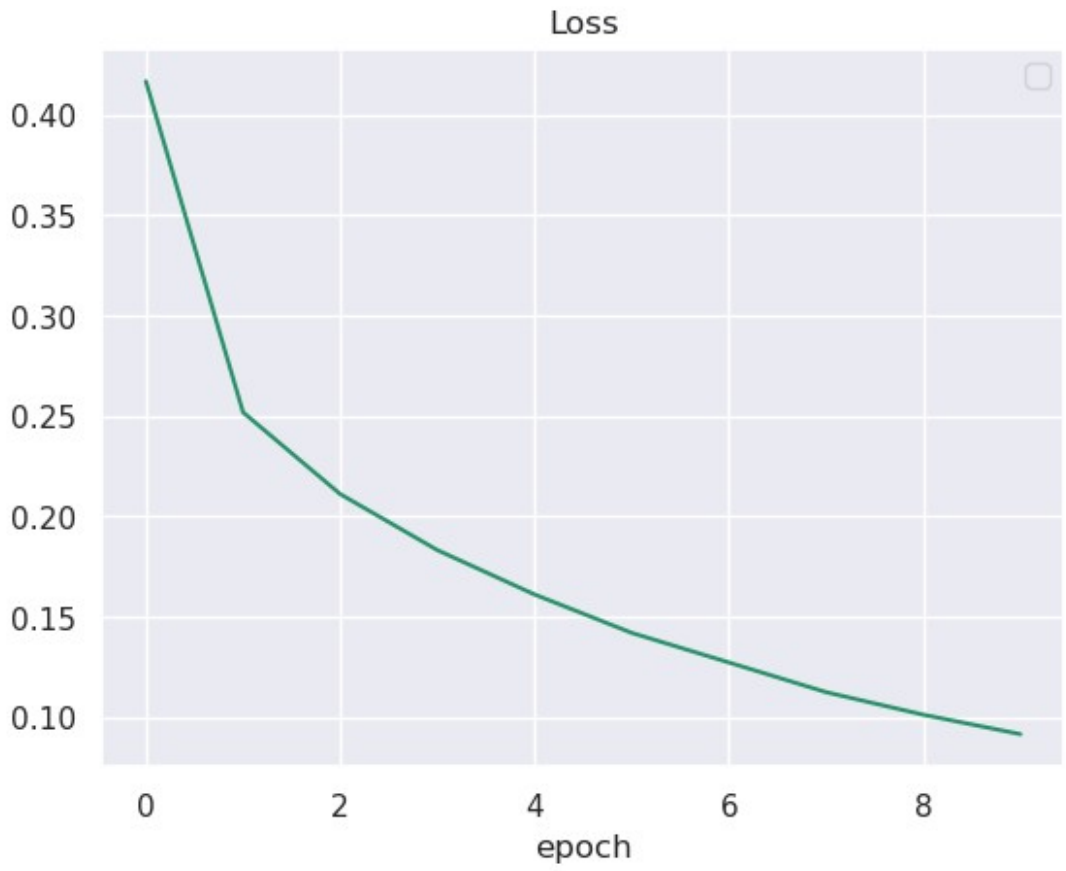
```
class WordDataset:
    def __init__(self, sentences):
        self.data = sentences
        self.unk_id = word2ind['<unk>']
        self.bos_id = word2ind['<bos>']
        self.eos_id = word2ind['<eos>']
        self.pad_id = word2ind['<pad>']

    def __getitem__(self, idx: int) -> List[int]:
        processed_text = self.data[idx]['text'].lower().translate(
            str.maketrans('', '', string.punctuation))
        tokenized_sentence = [self.bos_id]
        tokenized_sentence += [
            word2ind.get(word, self.unk_id) for word in
word_tokenize(processed_text) if word not in stopwords_united
        ]
        tokenized_sentence += [self.eos_id]

        train_sample = {
            "text": tokenized_sentence,
            "label": self.data[idx]['label']
        }

        return train_sample

    def __len__(self) -> int:
        return len(self.data)


def collate_fn_with_padding(
    input_batch: List[List[int]], pad_id=word2ind['<pad>'],
max_len=256) -> torch.Tensor:
    seq_lens = [len(x['text']) for x in input_batch]
    max_seq_len = min(max(seq_lens), max_len)

    new_batch = []
    for sequence in input_batch:
        sequence['text'] = sequence['text'][:max_seq_len]
        for _ in range(max_seq_len - len(sequence['text'])):
            sequence['text'].append(pad_id)

        new_batch.append(sequence['text'])
```

```
    sequences = torch.LongTensor(new_batch).to(device)
    labels = torch.LongTensor([x['label'] for x in
input_batch]).to(device)

    new_batch = {
        'input_ids': sequences,
        'label': labels
    }

    return new_batch

train_dataset = WordDataset(dataset['train'])

np.random.seed(42)
idx = np.random.choice(np.arange(len(dataset['test'])), 5000)
eval_dataset = WordDataset(dataset['test'].select(idx))

batch_size = 32
train_dataloader = DataLoader(
    train_dataset, shuffle=True, collate_fn=collate_fn_with_padding,
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=collate_fn_with_padding,
batch_size=batch_size)

model = CharLM(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 10
eval_steps = len(train_dataloader) // 2

losses = []
acc = []

for epoch in range(num_epoch):
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
        loss.backward()
        optimizer.step()

        epoch_losses.append(loss.item())
        if i % eval_steps == 0:
            model.eval()
            acc.append(evaluate(model, eval_dataloader))
```

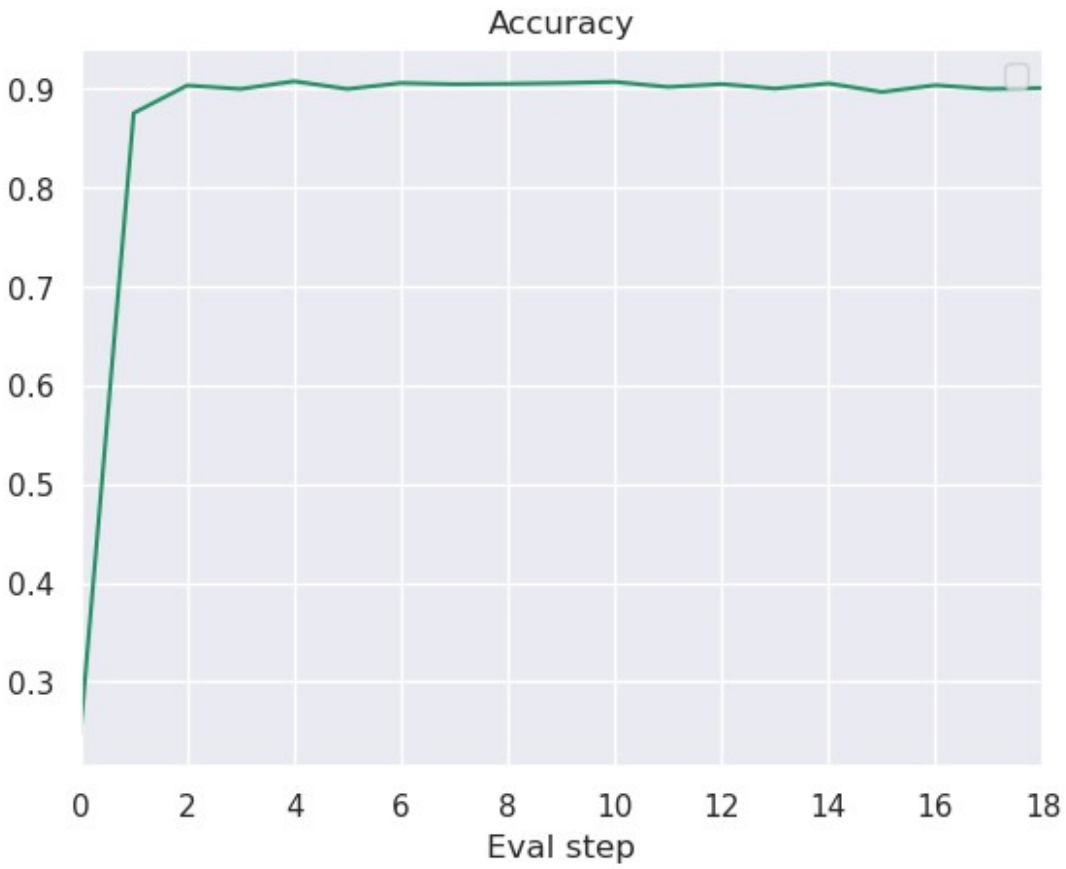
```
            model.train()

    losses.append(sum(epoch_losses) / len(epoch_losses))
```


```
{"model_id":"1c62febe0bbb44d88d47fdf52ced4905","version_major":2,"vers
ion_minor":0}

{"model_id":"cb160fbf708747cb8183417360ca6c9c","version_major":2,"vers
ion_minor":0}

{"model_id":"21ff95c34ee84d488c4cb1bd057a53cb","version_major":2,"vers
ion_minor":0}

{"model_id":"76c871e970844075b9b0ec734f81cf73","version_major":2,"vers
ion_minor":0}

{"model_id":"c96676a3ee0b491796ab91479987e7ff","version_major":2,"vers
ion_minor":0}

{"model_id":"c898dd6b692442f080ec53bc2dce406d","version_major":2,"vers
ion_minor":0}

{"model_id":"dbcdb35b623d4525b49158fc934d1eaf","version_major":2,"vers
ion_minor":0}

{"model_id":"ba2a9a780fe047e1bcba0c93fa63e47b","version_major":2,"vers
ion_minor":0}

{"model_id":"4406bbd33bc949d28db6de2e221c0b9c","version_major":2,"vers
ion_minor":0}

{"model_id":"8f38d12a10ea43b7be5b38cdfdd2f4d4","version_major":2,"vers
ion_minor":0}
```


```
plt.plot(losses)

plt.title('Loss')
plt.xlabel("epoch")
plt.legend()
plt.show()
```




```
plt.plot(acc)

plt.title('Accuracy')
plt.xlabel("Eval step")
plt.legend()
plt.xlim(0, 18)
plt.show()
```




```
acc[np.argmax(acc)], np.argmax(acc)
```

```
(0.9073999524116516, 4)
```

Видно, что это привело к падению accuracy модели.

Попробуем использовать лемматизацию и исключим стоп слова

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

!unzip /usr/share/nltk_data/corpora/wordnet.zip -d
/usr/share/nltk_data/corpora/

Archive:  /usr/share/nltk_data/corpora/wordnet.zip
   creating: /usr/share/nltk_data/corpora/wordnet/
  inflating: /usr/share/nltk_data/corpora/wordnet/lexnames
  inflating: /usr/share/nltk_data/corpora/wordnet/data.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/adv.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/index.verb
  inflating: /usr/share/nltk_data/corpora/wordnet/cntlist.rev
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adj
```

```
  inflating: /usr/share/nltk_data/corpora/wordnet/index.adj
  inflating: /usr/share/nltk_data/corpora/wordnet/LICENSE
  inflating: /usr/share/nltk_data/corpora/wordnet/citation.bib
  inflating: /usr/share/nltk_data/corpora/wordnet/noun.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/verb.exc
  inflating: /usr/share/nltk_data/corpora/wordnet/README
  inflating: /usr/share/nltk_data/corpora/wordnet/index.sense
  inflating: /usr/share/nltk_data/corpora/wordnet/data.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/data.adv
  inflating: /usr/share/nltk_data/corpora/wordnet/index.noun
  inflating: /usr/share/nltk_data/corpora/wordnet/adj.exc
```

```
en = spacy.load('en_core_web_sm')
stopwords_spacy = en.Defaults.stop_words

nltk.download('stopwords')

stopwords_nltk = stopwords.words("english")
```

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

```
stopwords_united = set(stopwords_spacy).union(stopwords_nltk)

dataset = datasets.load_dataset('ag_news')
```


```
{"model_id":"a65317d20ba441b0bbdf721b71b415a6","version_major":2,"vers
ion_minor":0}
```


```
words = Counter()

for example in tqdm(dataset['train']['text']):
    # Приводим к нижнему регистру и убираем пунктуацию
    proccessed_text = example.lower().translate(
        str.maketrans('', '', string.punctuation))

    words_tokenized = [
        word_lemmatized for word in word_tokenize(proccessed_text)
        if (word_lemmatized := lemmatizer.lemmatize(word)) not in
stopwords_united
    ]

    for word in words_tokenized:
        words[word] += 1

vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
counter_threshold = 25

for char, cnt in words.items():
    if cnt > counter_threshold:
        vocab.add(char)
```

```
print(f'Размер словаря: {len(vocab)}')

word2ind = {char: i for i, char in enumerate(vocab)}
ind2word = {i: char for char, i in word2ind.items()}
```

{"model_id":"6c502e46df7b47339a9aaf9901ce18a6","version_major":2,"version_minor":0}

```
Размер словаря: 10302
```

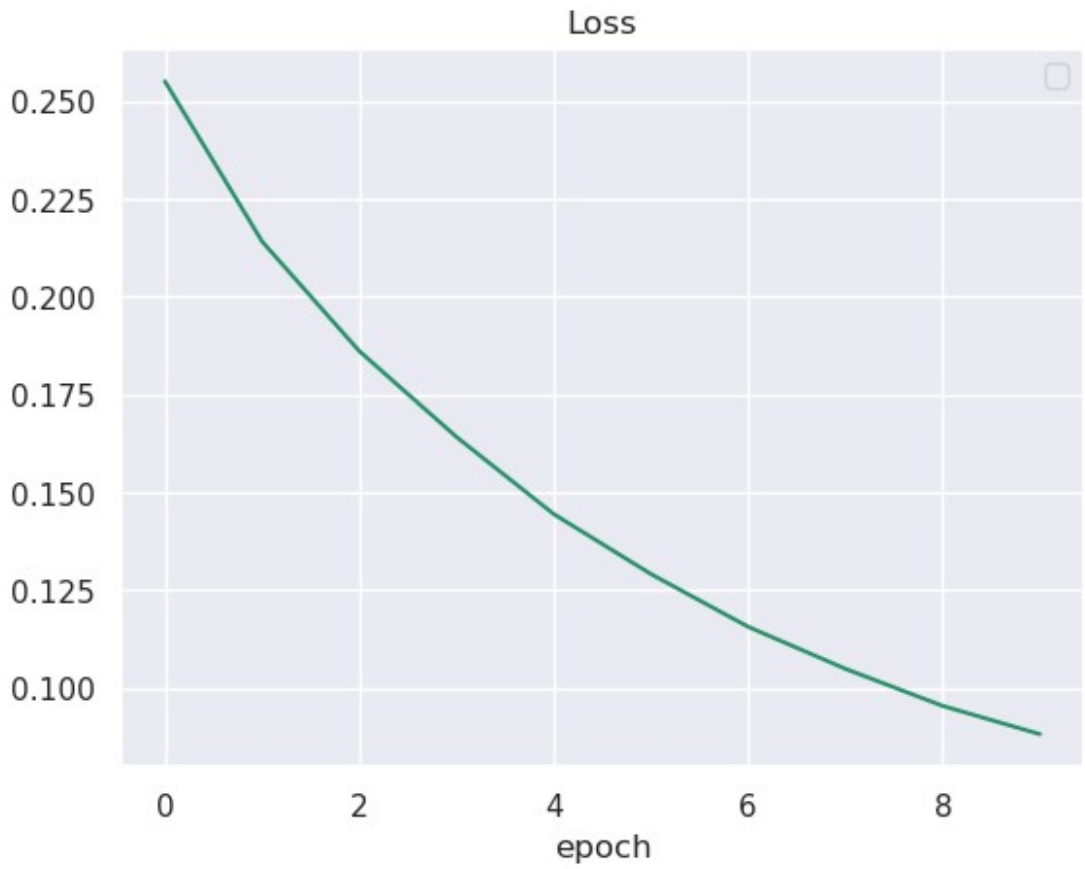
```
class WordDataset:
    def __init__(self, sentences):
        self.data = sentences
        self.unk_id = word2ind['<unk>']
        self.bos_id = word2ind['<bos>']
        self.eos_id = word2ind['<eos>']
        self.pad_id = word2ind['<pad>']

    def __getitem__(self, idx: int) -> List[int]:
        processed_text = self.data[idx]['text'].lower().translate(
            str.maketrans('', '', string.punctuation))
        tokenized_sentence = [self.bos_id]
        tokenized_sentence += [
            word2ind.get(word_lemmatized, self.unk_id) for word in
word_tokenize(processed_text)
            if (word_lemmatized := lemmatizer.lemmatize(word)) not in
stopwords_united
        ]
        tokenized_sentence += [self.eos_id]

        train_sample = {
            "text": tokenized_sentence,
            "label": self.data[idx]['label']
        }

        return train_sample

    def __len__(self) -> int:
        return len(self.data)


def collate_fn_with_padding(
    input_batch: List[List[int]], pad_id=word2ind['<pad>'],
max_len=256) -> torch.Tensor:
    seq_lens = [len(x['text']) for x in input_batch]
    max_seq_len = min(max(seq_lens), max_len)

    new_batch = []
    for sequence in input_batch:
        sequence['text'] = sequence['text'][:max_seq_len]
```

```
        for _ in range(max_seq_len - len(sequence['text'])):
            sequence['text'].append(pad_id)

        new_batch.append(sequence['text'])

    sequences = torch.LongTensor(new_batch).to(device)
    labels = torch.LongTensor([x['label'] for x in
input_batch]).to(device)

    new_batch = {
        'input_ids': sequences,
        'label': labels
    }

    return new_batch

train_dataset = WordDataset(dataset['train'])

np.random.seed(42)
idx = np.random.choice(np.arange(len(dataset['test'])), 5000)
eval_dataset = WordDataset(dataset['test'].select(idx))

batch_size = 32
train_dataloader = DataLoader(
    train_dataset, shuffle=True, collate_fn=collate_fn_with_padding,
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=collate_fn_with_padding,
batch_size=batch_size)

model = CharLM(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 10
eval_steps = len(train_dataloader) // 2

losses = []
acc = []

for epoch in range(num_epoch):
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
        loss.backward()
        optimizer.step()
```

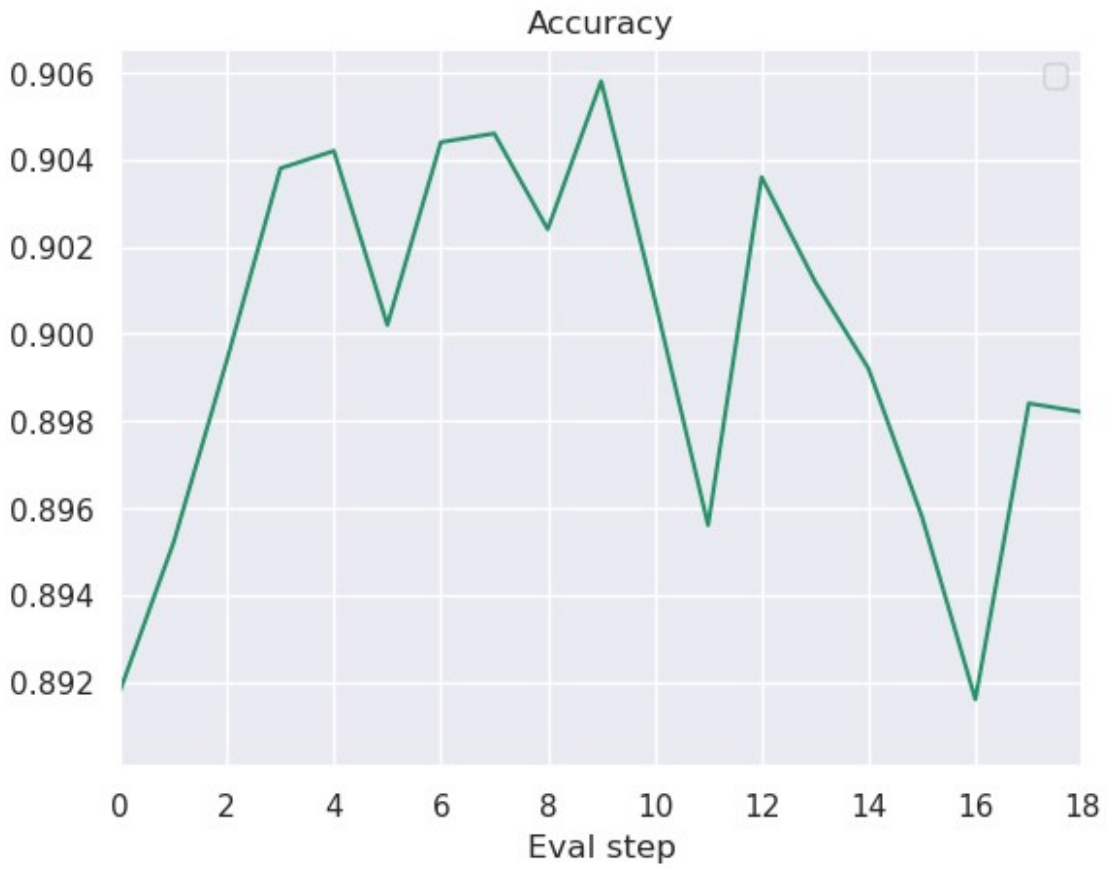
```
        epoch_losses.append(loss.item())
        if i % eval_steps == 0:
            model.eval()
            acc.append(evaluate(model, eval_dataloader))
            model.train()

    losses.append(sum(epoch_losses) / len(epoch_losses))
```


```
{"model_id":"f044d9128f5f4891b2839d40d5b3b6d0","version_major":2,"vers
ion_minor":0}

{"model_id":"85fdf3b93d544dc0a4cf796398d6591e","version_major":2,"vers
ion_minor":0}

{"model_id":"dfd3efcf304644a1b4f2419b30395b2d","version_major":2,"vers
ion_minor":0}

{"model_id":"8331c1b345724255a795b837803a217c","version_major":2,"vers
ion_minor":0}

{"model_id":"dd1844ac0e0c473da86e0d1617861132","version_major":2,"vers
ion_minor":0}

{"model_id":"fa45957cd99a48759f19f9e89729d523","version_major":2,"vers
ion_minor":0}

{"model_id":"99a973a7e30c4a96b08fe0c44eb50f12","version_major":2,"vers
ion_minor":0}

{"model_id":"bf6f6e577cd94841a60b8aa4ce9fa060","version_major":2,"vers
ion_minor":0}

{"model_id":"0b52a201cfe04c5d8f5a79828d1a567b","version_major":2,"vers
ion_minor":0}

{"model_id":"001a88ea751d424eb747967a9d9cfe23","version_major":2,"vers
ion_minor":0}
```


```
plt.plot(losses)

plt.title('Loss')
plt.xlabel("epoch")
plt.legend()
plt.show()
```




```
plt.plot(acc)

plt.title('Accuracy')
plt.xlabel("Eval step")
plt.legend()
plt.xlim(0, 18)
plt.show()
```




```
acc[np.argmax(acc)], np.argmax(acc)
```

```
(0.9057999849319458, 9)
```

Видно,что в данном случае метрика accuracy сильно флуктуирует.

Вывод

В работе были предприняты попытки улучшения метрики accuracy модели классификации текста относительно базовой модели (0.9116). Были опробованы 4 подхода:

- Использование LSTM слоя вместо RNN, что привело к увеличению accuracy до 0.9146 (или 0.915, если округлять до 3 знака). Полученный результат объясняется тем, что слой LSTM позволяет использовать более долгий контекст по сравнению со слоем RNN.
- Использование двух слоев RNN, что привело к уменьшению accuracy до 0.9082.
- Удаление стоп слов, что привело к уменьшению accuracy до 0.9074. Недостаточный эффект может быть связан с тем, что стоп слов оказалось всего около 370 штук, то есть размер словаря уменьшился незначительно по сравнению с базовой моделью (с 11842 до 11559).
- Использование лемматизации и удаление стоп слов, что привело к уменьшению accuracy до 0.9058. Размер словаря при этом уменьшился до 10302. Однако, в

данном случае метрика сильно колебалась по эпохам, и уже после первой была более 0.9. Возможно, это связано с большим значением learning rate.

Полученных результатов исследований показали, что оптимальной моделью будет модель со слоем LSTM. Однако, необходимо провести "чистку" словаря, так как в нем присутствуют слова, которые не отражают никакого смысла.