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
```

Импорт необходимых библиотек

```
import string
from collections import Counter
from typing import List

import datasets
import matplotlib.pyplot as plt
import nltk
import numpy as np
import seaborn
import torch
import torch.nn as nn
from nltk.tokenize import word_tokenize
from torch.utils.data import DataLoader
from tqdm.auto import tqdm

seaborn.set(palette='summer')

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\k142\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

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

```
'cpu'
```

Подготовка данных

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

Описание Набора

https://labelbox.com/datasets/ag-news/

AG News (AG's News Corpus) is a subdataset of AG's corpus of news articles constructed by assembling titles and description fields of articles from the 4 largest classes ("World", "Sports", "Business", "Sci/Tech") of AG's Corpus. The AG News contains 30,000 training and 1,900 test samples per class.

Считаем что обучающая выборка сбалансирована.

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

```
{"model_id":"76a94dbc21ff4df98c8a088ee671cd48","version_major":2,"vers
ion_minor":0}
```

```
C:\prj\DLS\dls_part_2_nlp\venv\Lib\site-packages\huggingface_hub\
file_download.py:143: UserWarning: `huggingface_hub` cache-system uses
symlinks by default to efficiently store duplicated files but your
machine does not support them in C:\Users\k142\.cache\huggingface\hub\
datasets--ag_news. Caching files will still work but in a degraded
version that might require more space on your disk. This warning can
be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING`
environment variable. For more details, see
https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer
Mode or to run Python as an administrator. In order to activate
developer mode, see this article:
https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-
device-for-development
  warnings.warn(message)
Xet Storage is enabled for this repo, but the 'hf_xet' package is not
installed. Falling back to regular HTTP download. For better
```

```
performance, install the package with: `pip install
huggingface_hub[hf_xet]` or `pip install hf_xet`

{"model_id":"1fa16c0909724dba88fcda1e060235da","version_major":2,"vers
ion_minor":0}

Xet Storage is enabled for this repo, but the 'hf_xet' package is not
installed. Falling back to regular HTTP download. For better
performance, install the package with: `pip install
huggingface_hub[hf_xet]` or `pip install hf_xet`

{"model_id":"cf72e15ca8774f358c30a2935c1566fd","version_major":2,"vers
ion_minor":0}

{"model_id":"b7d9be6a22c94f7ab8ea705a701e64d0","version_major":2,"vers
ion_minor":0}

{"model_id":"8ecc37f3017f47798599a9d85b26493f","version_major":2,"vers
ion_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
    prccessed_text = example.lower().translate(
        str.maketrans('', '', string.punctuation))

    for word in word_tokenize(prccessed_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":"27c044dce91f42a6ab3a8c0ac743b1df","version_major":2,"vers
ion_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)

dataset['train'][:2]
```

```
{'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.",
  'Carlyle Looks Toward Commercial Aerospace (Reuters) Reuters -
Private investment firm Carlyle Group,\\which has a reputation for
making well-timed and occasionally\\controversial plays in the defense
industry, has quietly placed\\its bets on another part of the
market.'],
 'label': [2, 2]}
```

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.

```
class CharLM(nn.Module):
    def __init__(
            self, hidden_dim: int, vocab_size: int, num_classes: int =
2,
            aggregation_type: str = 'max'
    ):
        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)

        self.aggregation_type = aggregation_type

    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]

        if self.aggregation_type == 'max':
            output = output.max(dim=1)[0]  #[batch_size, hidden_dim]
        elif self.aggregation_type == 'mean':
            output = output.mean(dim=1)  #[batch_size, hidden_dim]
        else:
            raise ValueError("Invalid aggregation_type")
```

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

        return prediction
```

Train loop

```
def evaluate(model) -> 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

model = CharLM(hidden_dim=256, vocab_size=len(vocab)).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 5
eval_steps = len(train_dataloader) // 2

losses_type = {}
acc_type = {}

for aggregation_type in ['max', 'mean']:
    print(f"Starting training for {aggregation_type}")
    losses = []
    acc = []

    model = CharLM(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, num_classes=4).to(device)
    criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
    optimizer = torch.optim.Adam(model.parameters())

    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))
                model.train()

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

    losses_type[aggregation_type] = losses
    acc_type[aggregation_type] = acc
```

```
Starting training for max
```

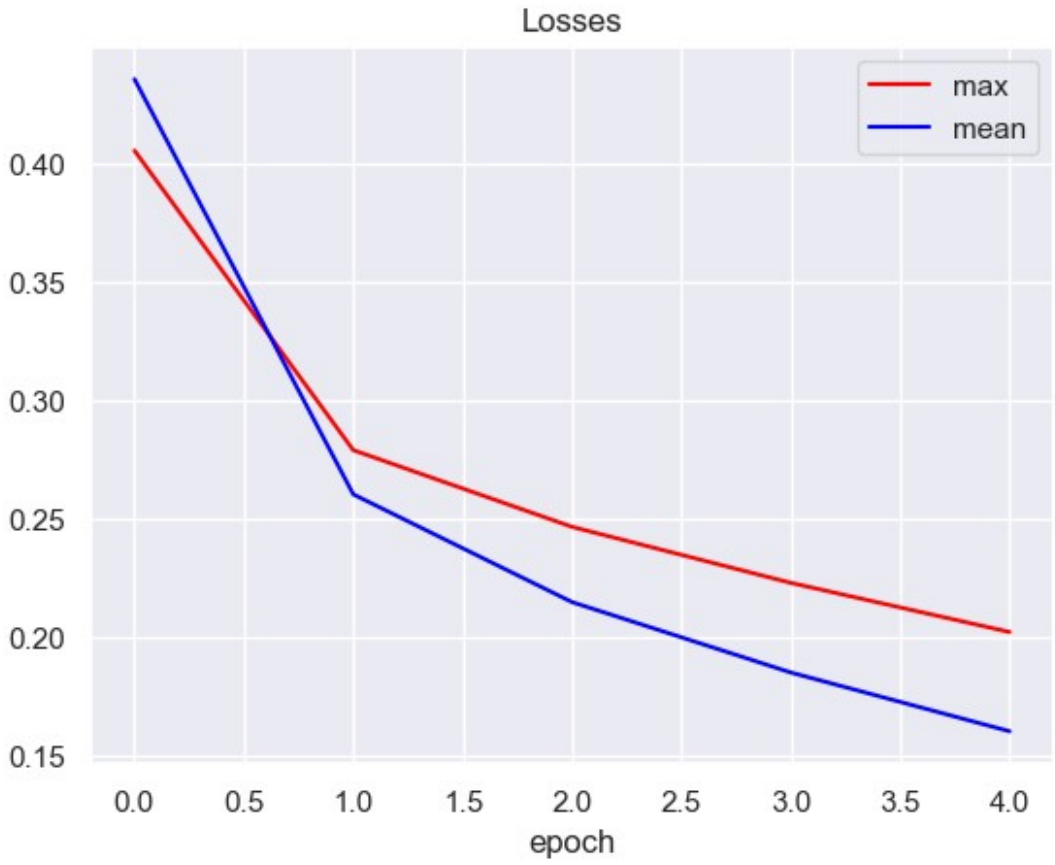

```
{"model_id":"4ba0432160b547a8873c88c52ae52dd0","version_major":2,"vers
ion_minor":0}

{"model_id":"64c561ee361e472ba02f53fed1e0b44c","version_major":2,"vers
ion_minor":0}

{"model_id":"e6258ec3827b4175973a841a1a63c115","version_major":2,"vers
ion_minor":0}

{"model_id":"015ff0ed71d64d679b9eaa85288c205a","version_major":2,"vers
ion_minor":0}

{"model_id":"2c861baba7b245c98487d23cdfb1a555","version_major":2,"vers
ion_minor":0}
```


```
Starting training for mean
```


```
{"model_id":"f5ed3ff050144ea5924d47665dec49be","version_major":2,"vers
ion_minor":0}

{"model_id":"014b52984d4d4b6fa5dd466a18e1caeb","version_major":2,"vers
ion_minor":0}

{"model_id":"de55cd2384184f99b4753b66ee932a88","version_major":2,"vers
ion_minor":0}

{"model_id":"624ad8e61652459a95452515a43a40f8","version_major":2,"vers
ion_minor":0}
```

{"model_id":"a33bcd199b3a4ba9a2bcfd1b6b215de5","version_major":2,"version_minor":0}

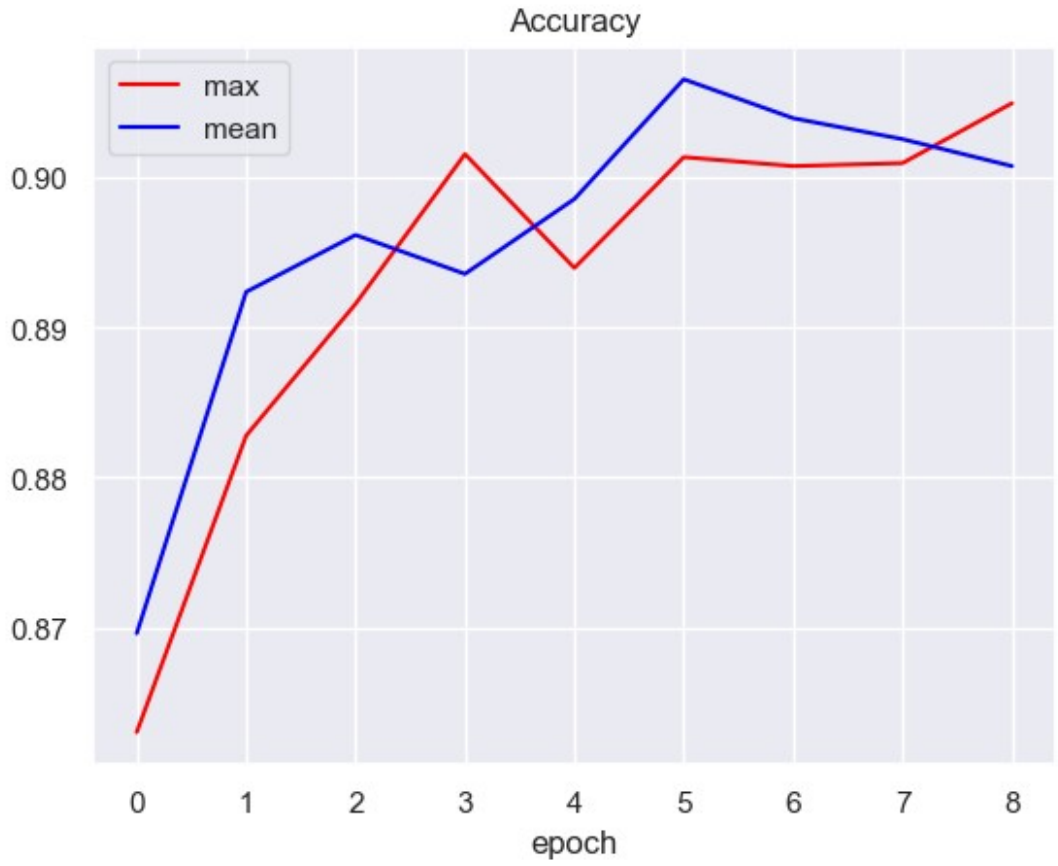
```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(losses_type[name])), losses_type[name],
color=color, label=name)

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



Ю

```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(acc_type[name][1:])), acc_type[name][1:],
color=color, label=name)
    print(f"Лучшая accuracy для подхода {name}: {(max(acc_type[name])
* 100):.2f}")

plt.title('Accuracy')
```

```
plt.xlabel("epoch")
plt.legend()
plt.show()

Лучшая accuracy для подхода max: 90.50
Лучшая accuracy для подхода mean: 90.66
```



Запустил обучение на новом датасете. обучение проводил на локальном компьютере без GPU. Общее время 30 мин. Около 3 мин на итерацию. Максимальная точность для mean 90.6

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.

Увеличение количества эпох

На последнем запуске видим что если модель при типе агрегации `mean` достигла максимума точности примерно на 5 эпохе и потом стала снижаться, то точность модели при типе агрегации `max` еще увеличивается и необходимо проверить что происходит при дальнейшем увеличении эпох.

```
num_epoch = 8
eval_steps = len(train_dataloader) // 2

losses_type = {}
acc_type = {}

for aggregation_type in ['max']:
    print(f"Starting training for {aggregation_type}")
    losses = []
    acc = []

    model = CharLM(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, num_classes=4).to(device)
    criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
    optimizer = torch.optim.Adam(model.parameters())

    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))
                model.train()

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

    losses_type[aggregation_type] = losses
    acc_type[aggregation_type] = acc
```

```
Starting training for max
```


```
{"model_id":"cab76552c5464f4f9a62fa028dda9895","version_major":2,"vers
ion_minor":0}

{"model_id":"4dc7edc7e9dd4145a8a7308f9f782941","version_major":2,"vers
ion_minor":0}

{"model_id":"5ae907435c8e4c6ea2a8655aa983f98f","version_major":2,"vers
ion_minor":0}

{"model_id":"f70541368e744906a9e62f208f32203c","version_major":2,"vers
ion_minor":0}

{"model_id":"36f1af0bd89b43629992f0962c12e4fc","version_major":2,"vers
ion_minor":0}

{"model_id":"7501a5c93a0a4e98a797d8c4eafc57c0","version_major":2,"vers
ion_minor":0}

{"model_id":"32a1c38480aa44ecaebc4b0544a59591","version_major":2,"vers
ion_minor":0}

{"model_id":"7b9aba97750447bea37b83704ac4d214","version_major":2,"vers
ion_minor":0}

{"model_id":"1e817b2692a44446bbb730d503cded6d","version_major":2,"vers
ion_minor":0}

{"model_id":"e89b49dc9d7948f481967b9855563033","version_major":2,"vers
ion_minor":0}
```


```
len(acc_type['max'])
```

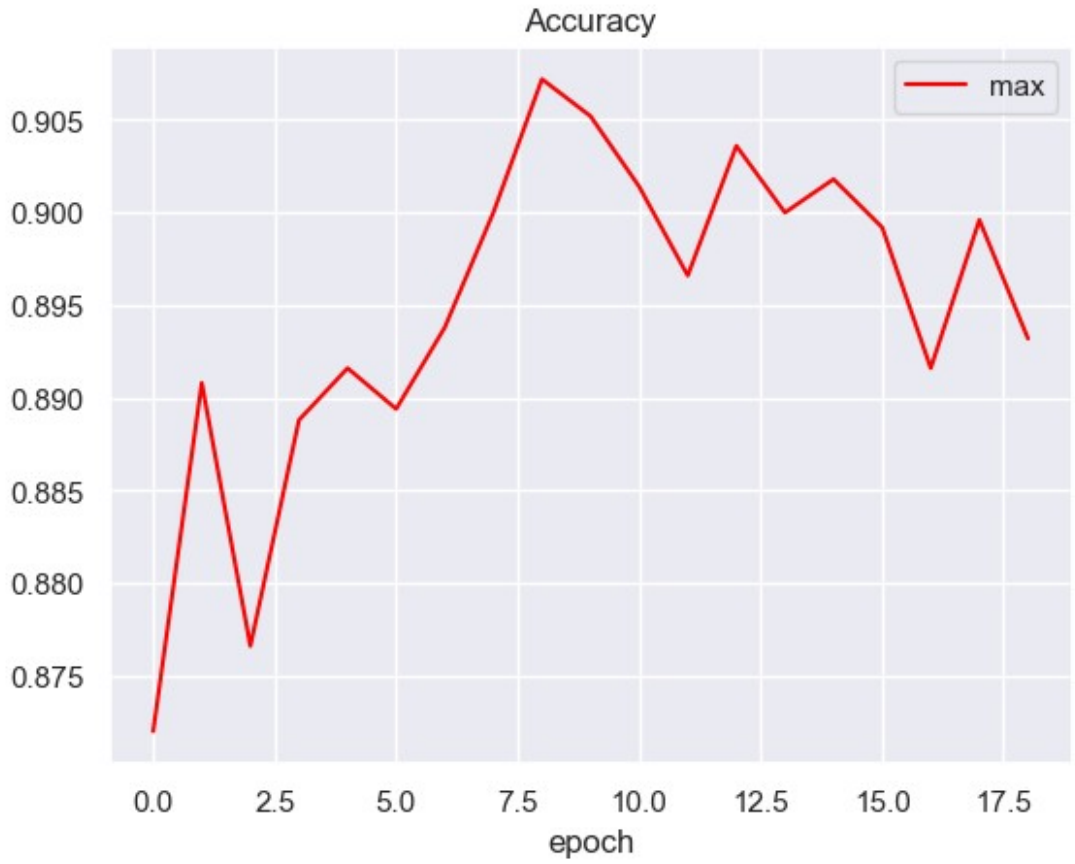
```
20
```

```
losses_type
```

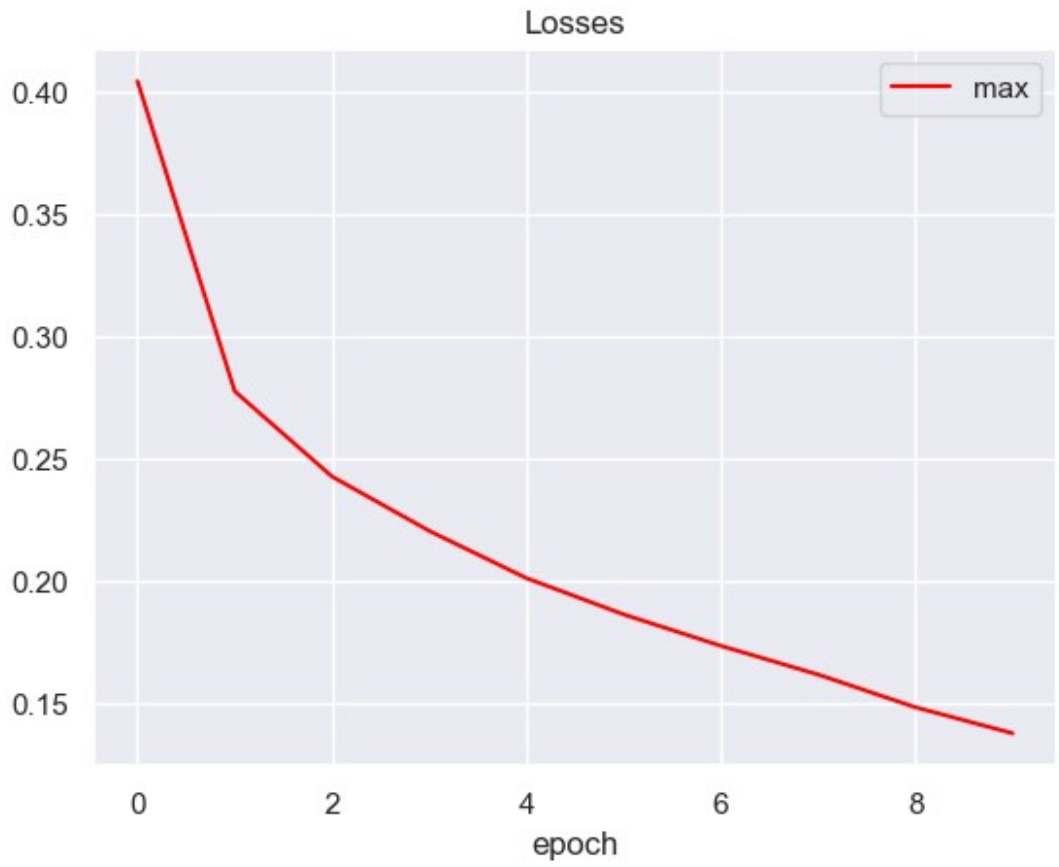
```
{'max': [0.40458921128114067,
  0.27772412172406913,
  0.2428571069329977,
  0.22061870145450035,
  0.2012836210814615,
  0.18643592583835125,
  0.17358287394841512,
  0.16182580639546115,
  0.1484499670823415,
  0.13780583850753805]}
```

```
for (name, values), color in zip(losses_type.items(), ['red']):
    plt.plot(np.arange(len(acc_type[name][1:])), acc_type[name][1:],
color=color, label=name)
    print(f"Лучшая accuracy для подхода {name}: {(max(acc_type[name])
* 100):.2f}")

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

```
Лучшая accuracy для подхода max: 90.72
```



```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(losses_type[name])), losses_type[name],
color=color, label=name)

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




```
evaluate(model)
```

```
0.9014000296592712
```

1 Эксперимент немного увеличил точность модели до 90.72 (ранее 90.60).
Дальнейшее обучение привело к переобучению и снижению качества до 90.14

Использовние GRU

```
class CharLM2(nn.Module):
    def __init__(
            self, hidden_dim: int, vocab_size: int, num_classes: int =
4, aggregation_type: str = 'max', num_layer=1):
```

```
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.gru = nn.GRU(hidden_dim, hidden_dim, batch_first=True,
num_layers=num_layer)
        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)

        self.aggregation_type = aggregation_type

    def forward(self, input_batch) -> torch.Tensor:
        embeddings = self.embedding(input_batch)
        output, hidden = self.gru(embeddings)  # hidden: [1,
batch_size, hidden_dim]

        if self.aggregation_type == 'max':
            output = output.max(dim=1)[0]  # [batch_size, hidden_dim]
        elif self.aggregation_type == 'mean':
            output = output.mean(dim=1)  # [batch_size, hidden_dim]
        elif self.aggregation_type == 'last':
            output = hidden[-1]  # [batch_size, hidden_dim] -
последнее состояние
        elif self.aggregation_type == 'first_last':
            # комбинация первого и последнего hidden states
            output = torch.cat([output[:, 0, :], output[:, -1, :]],
dim=1)
            output = self.linear(output)  # дополнительный linear
        else:
            raise ValueError("Invalid aggregation_type")

        output = self.dropout(self.linear(self.non_lin(output)))
        prediction = self.projection(self.non_lin(output))
        return prediction

aggregation_type = 'max'

model = CharLM2(hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, num_classes=4).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 5
eval_steps = len(train_dataloader) // 2

losses_type = {}
acc_type = {}

aggregation_type = 'max'
```

```
print(f"Starting training for {aggregation_type}")
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))
            model.train()

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

losses_type[aggregation_type] = losses
acc_type[aggregation_type] = acc
```

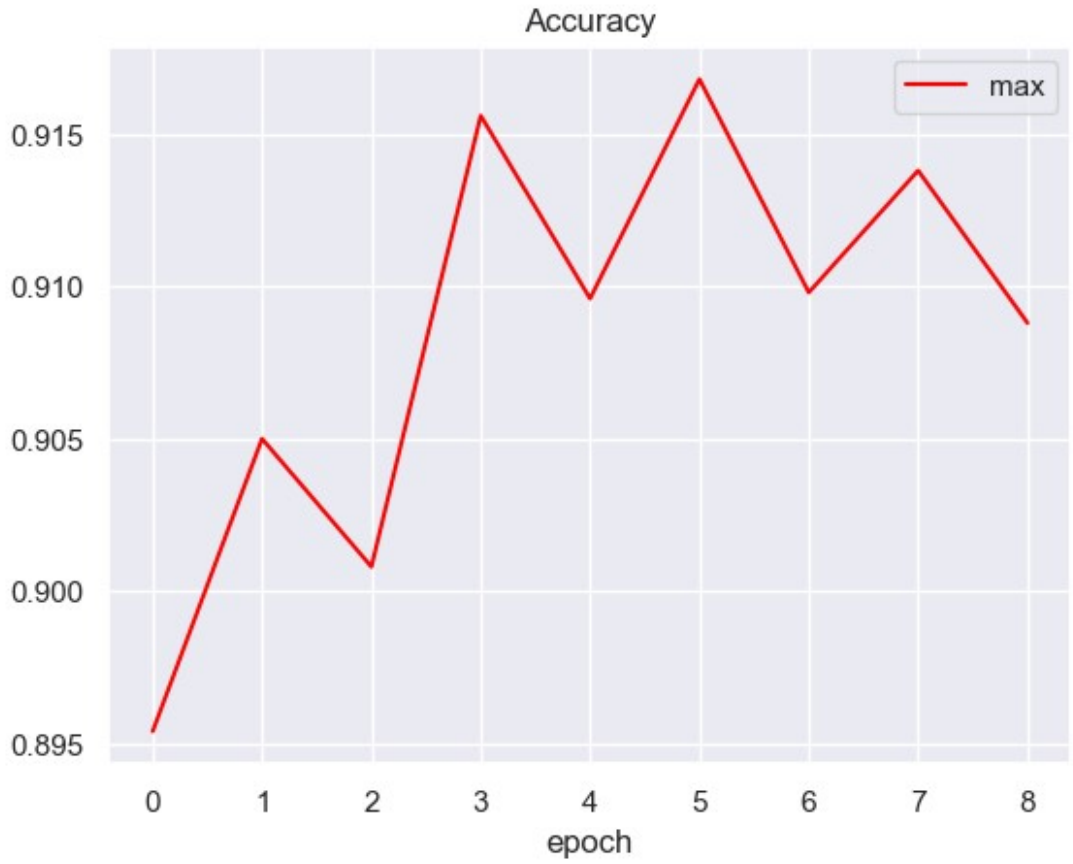
```
Starting training for max
```


```
{"model_id":"bdc3c3e4ed2a4b44b8ed3a3478741cd7","version_major":2,"vers
ion_minor":0}

{"model_id":"f08b08aca40a4d48a8e759a1b83e2a78","version_major":2,"vers
ion_minor":0}

{"model_id":"88f65125d9a9471b86d184e7277d79db","version_major":2,"vers
ion_minor":0}

{"model_id":"cacf315f465e4256ad07cfe69a5cd81c","version_major":2,"vers
ion_minor":0}

{"model_id":"f8579bbb67ab453ca8d22ff80651d0f1","version_major":2,"vers
ion_minor":0}
```

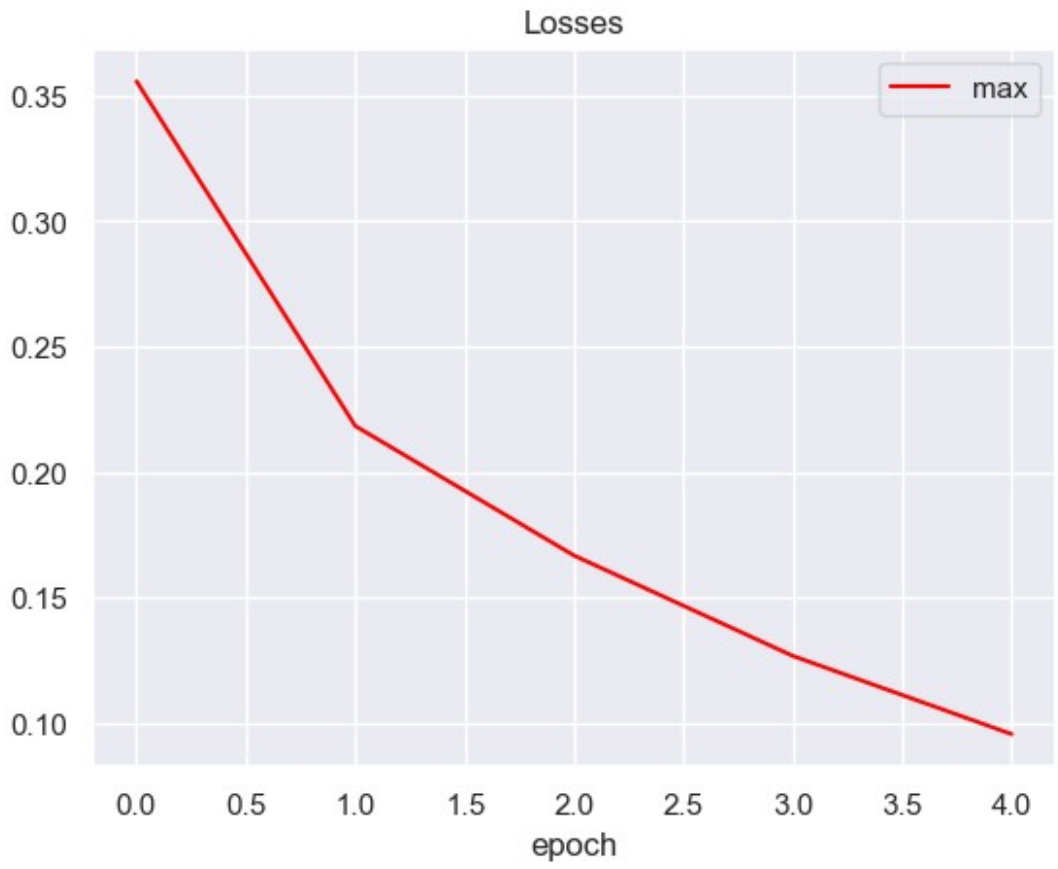

```
for (name, values), color in zip(losses_type.items(), ['red']):
    plt.plot(np.arange(len(acc_type[name][1:])), acc_type[name][1:],
color=color, label=name)
    print(f"Лучшая accuracy для подхода {name}: {(max(acc_type[name])
* 100):.2f}")

plt.title('Accuracy')
plt.xlabel("epoch")
```

```
plt.legend()
plt.show()
```

```
Лучшая accuracy для подхода max: 91.68
```




```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(losses_type[name])), losses_type[name],
color=color, label=name)

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



При использовании GRU и 4 эпох обучения получил 91.68. Агрегация - max

Использовние GRU + агрегация - last

```
aggregation_type = 'last'

model = CharLM2(hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, num_classes=4).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 5
eval_steps = len(train_dataloader) // 2

losses_type = {}
acc_type = {}

aggregation_type = 'last'

print(f"Starting training for {aggregation_type}")
losses = []
acc = []

for epoch in range(num_epoch):
```

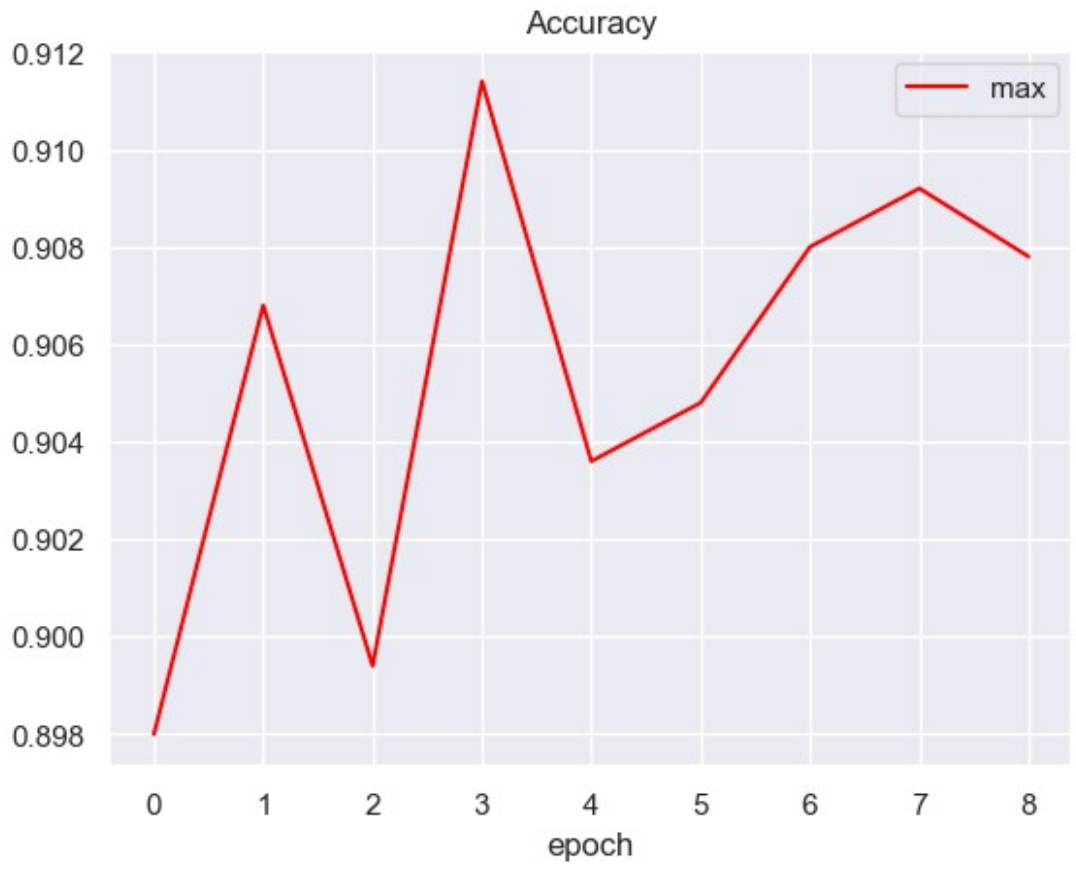
```
    epoch_losses = []
    model.train()
    for i, batch in enumerate(tqdm(train_dataloader, desc=f'Training
epoch {epoch}:')):
        optimizer.zero_grad()
        logits = model(batch['input_ids'])
        loss = criterion(logits, batch['label'])
        loss.backward()
        optimizer.step()

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

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

losses_type[aggregation_type] = losses
acc_type[aggregation_type] = acc
```

```
Starting training for max
```

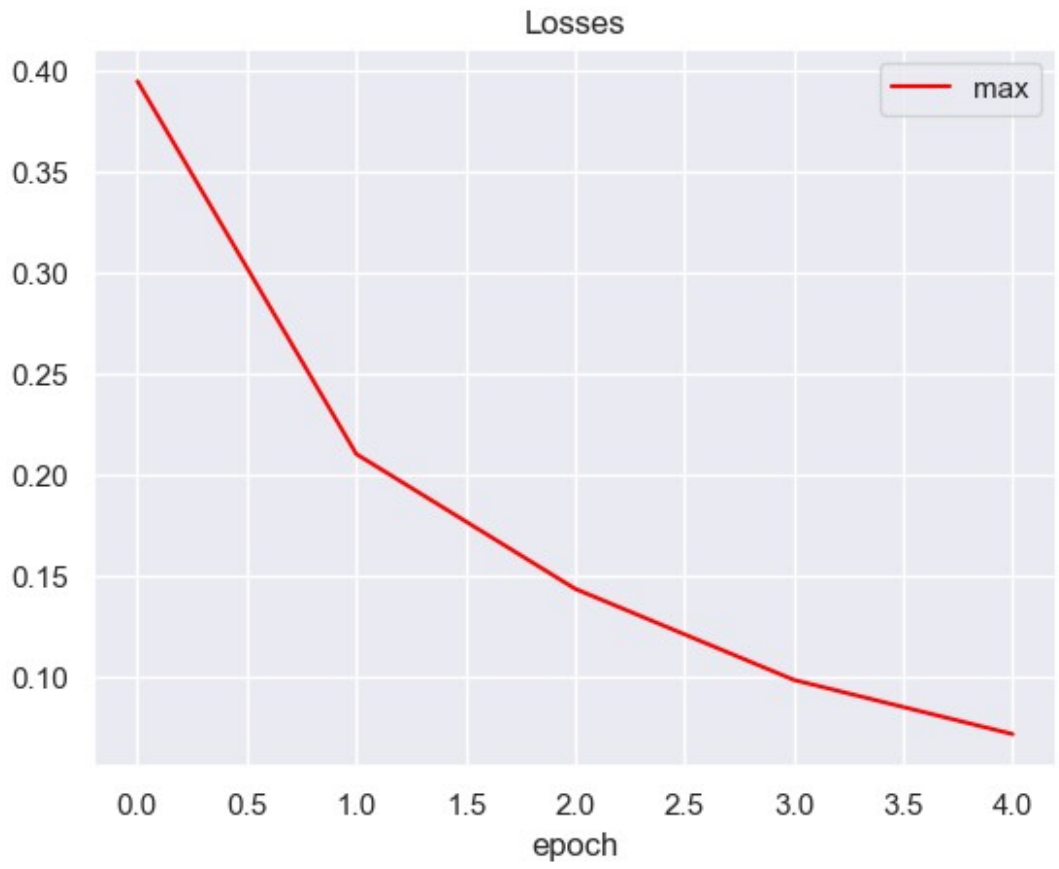

```
{"model_id":"807a4cde85e241ea8a079ba995a32b75","version_major":2,"vers
ion_minor":0}

{"model_id":"885386d299814c8fa2d03fac5db48f57","version_major":2,"vers
ion_minor":0}

{"model_id":"ad49b9f15941458c949630be61b325cd","version_major":2,"vers
ion_minor":0}

{"model_id":"abcffd5ca08341caaf0aa5f357d2debb","version_major":2,"vers
ion_minor":0}

{"model_id":"a6c8d026178848209c218abce97b7f29","version_major":2,"vers
ion_minor":0}
```


```
for (name, values), color in zip(losses_type.items(), ['red']):
    plt.plot(np.arange(len(acc_type[name][1:])), acc_type[name][1:],
color=color, label=name)
    print(f"Лучшая accuracy для подхода {name}: {(max(acc_type[name])
* 100):.2f}")

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

```
Лучшая accuracy для подхода max: 91.14
```




```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(losses_type[name])), losses_type[name],
color=color, label=name)

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



При использовании GRU и 4 эпох обучения получил 91.14. Агрегация - last

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.

```
aggregation_type = 'max'
```

Повторить результат оказалось крайне сложно, поэтому для лучшего результата я добавил еще 1 слой GRU всего 2 слоя. Также я включил сохранение снимков модели, чтобы потом воспроизвести лучший результат.

```
model = CharLM2(hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, num_classes=4,
                num_layer=2).to(device)
```

```
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model.parameters())

num_epoch = 5
eval_steps = len(train_dataloader) // 2

losses_type = {}
acc_type = {}

print(f"Starting training for {aggregation_type}")
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))
            torch.save(model.state_dict(), f'char_lm2_epoch_{epoch}-
{i}.pth')
            model.train()

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

losses_type[aggregation_type] = losses
acc_type[aggregation_type] = acc
```

```
Starting training for max
```


```
{"model_id":"8631ba2a26db4d5bb79c599520a0919f","version_major":2,"vers
ion_minor":0}

{"model_id":"e726542230e04fa88964cf7fb057c0e5","version_major":2,"vers
ion_minor":0}

{"model_id":"0b22dc5e23f6432eafc3664010827ebf","version_major":2,"vers
ion_minor":0}

{"model_id":"03a037d3659542b99b00488a9c34da53","version_major":2,"vers
ion_minor":0}
```

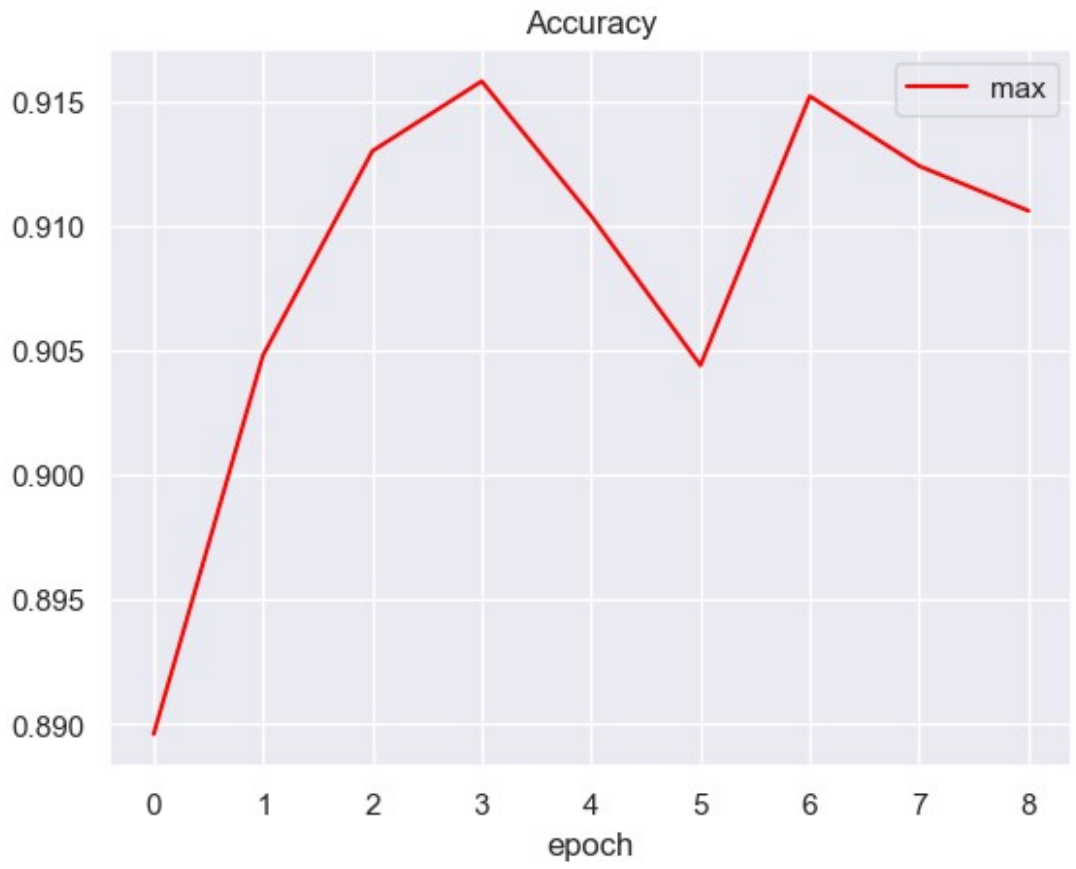
{"model_id":"a55bc7ca6a884472876eb303f44d4276","version_major":2,"version_minor":0}

acc_type

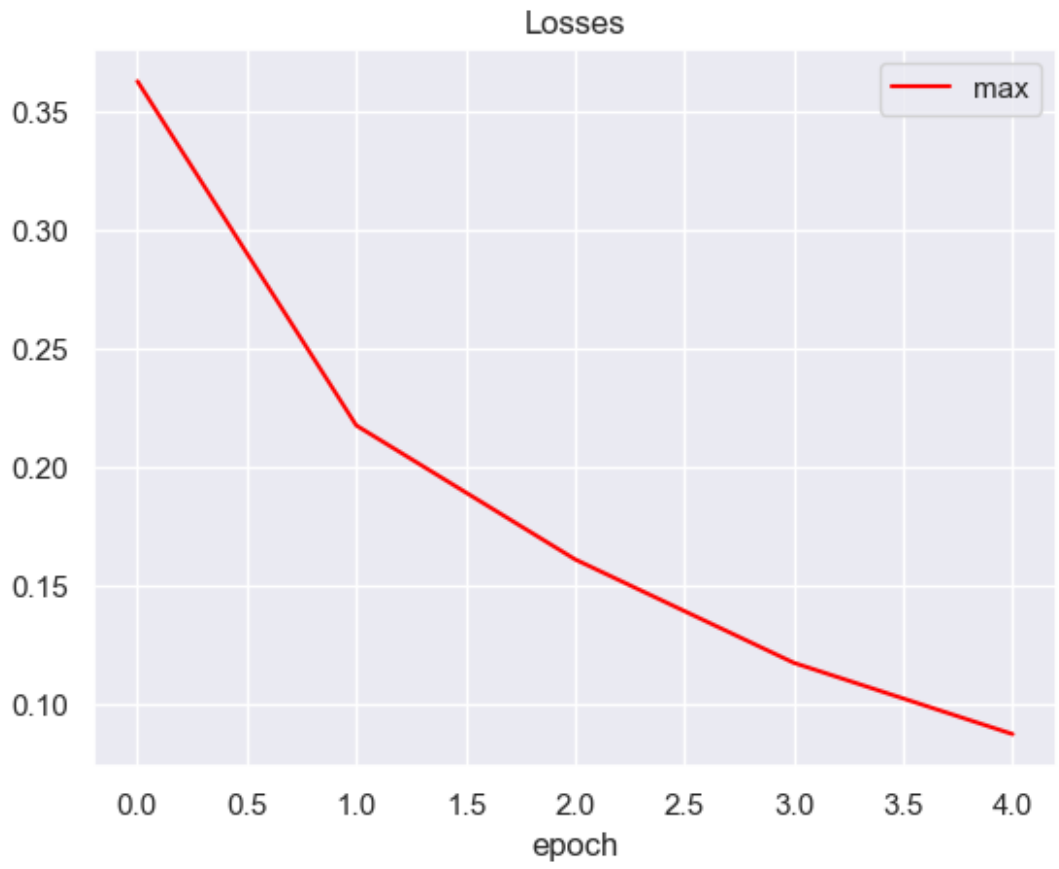
```
{'max': [0.25679999589920044,
  0.8895999789237976,
  0.9047999978065491,
  0.9129999876022339,
  0.9157999753952026,
  0.9103999733924866,
  0.9043999910354614,
  0.9151999950408936,
  0.9124000072479248,
  0.9106000065803528]}
```

```
for (name, values), color in zip(losses_type.items(), ['red']):
    plt.plot(np.arange(len(acc_type[name][1:])), acc_type[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name}: {(max(acc_type[name]) * 100):.2f}")

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

```
Лучшая accuracy для подхода max: 91.58
```




```
for (name, values), color in zip(losses_type.items(), ['red',
'blue']):
    plt.plot(np.arange(len(losses_type[name])), losses_type[name],
color=color, label=name)

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




```
torch.__version__
```

```
'2.8.0+cpu'
```

```
model.load_state_dict(torch.load('char_lm2_epoch_2-0.pth'))
```

```
<All keys matched successfully>
```

```
model.eval()
evaluate(model)
```

```
0.9157999753952026
```

по условиям задачи

- $0.915 \leq accuracy$ --- 3 балла.

мой результат - **0.9158** это 3 балла

```
from sklearn.metrics import classification_report


def evaluate_with_report(model, eval_dataloader, class_names=None) ->
tuple:
    """
```

```
    Calculate accuracy and return full classification report.
    """

    predictions = []
    targets = []

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

    predictions = torch.cat(predictions)
    targets = torch.cat(targets)

    accuracy = (predictions == targets).float().mean().item()

    # Generate detailed classification report
    classification_rep = classification_report(
        targets.cpu().numpy(),
        predictions.cpu().numpy(),
        target_names=class_names,
        zero_division=0
    )

    print("Accuracy:", accuracy)
    print("\nClassification Report:")
    print(classification_rep)

    return accuracy, classification_rep

class_names = ['World', 'Sports', 'Business', 'Sci/Tech']
model.eval()
_ = evaluate_with_report(model, eval_dataloader,
class_names=class_names)
```

```
Accuracy: 0.9157999753952026

Classification Report:
              precision    recall  f1-score   support

       World       0.92      0.92      0.92      1235
      Sports       0.96      0.96      0.96      1197
    Business       0.89      0.90      0.90      1284
    Sci/Tech       0.90      0.88      0.89      1284

    accuracy                           0.92      5000
   macro avg       0.92      0.92      0.92      5000
weighted avg       0.92      0.92      0.92      5000
```

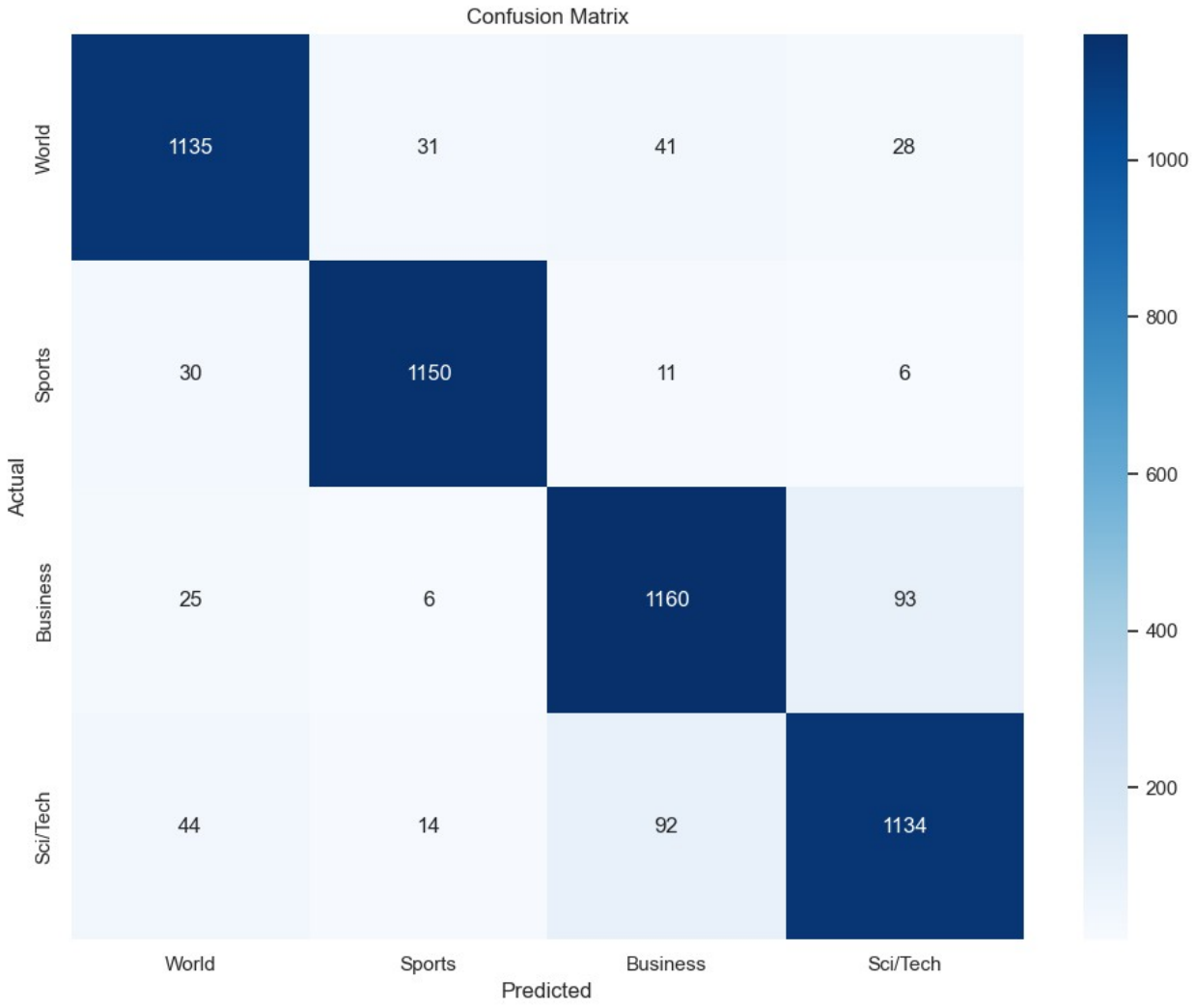
```python
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


def evaluate_detailed(model, eval_dataloader, class_names=None) ->
dict:
    """
    Accuracy and full classification report.
    """

    predictions = []
    targets = []
    probabilities = []

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

    predictions = torch.cat(predictions)
    targets = torch.cat(targets)
    probabilities = torch.cat(probabilities)

    # Convert to numpy for sklearn
    predictions_np = predictions.cpu().numpy()
    targets_np = targets.cpu().numpy()
    probabilities_np = probabilities.cpu().numpy()

    accuracy = (predictions == targets).float().mean().item()

    report = classification_report(
        targets_np,
        predictions_np,
        target_names=class_names,
        output_dict=True,
        zero_division=0
    )

    # Confusion matrix
    cm = confusion_matrix(targets_np, predictions_np)

    results = {
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': cm,
        'predictions': predictions_np,
        'targets': targets_np,
```

```
        'probabilities': probabilities_np
    }

    return results


def print_evaluation_results(results, class_names=None):
    """Print formatted evaluation results"""

    print("=" * 60)
    print("CLASSIFICATION EVALUATION REPORT")
    print("=" * 60)

    print(f"\nOverall Accuracy: {results['accuracy']:.4f}")

    print("\nDetailed Classification Report:")
    print("-" * 40)

    # Convert report dict to string format for pretty printing
    report_str = classification_report(
        results['targets'],
        results['predictions'],
        target_names=class_names,
        zero_division=0
    )
    print(report_str)

    # Print per-class metrics
    print("\nPer-class Metrics:")
    print("-" * 40)
    report_df = pd.DataFrame(results['report']).transpose()
    print(report_df.round(4))

    return results


def plot_confusion_matrix(results, class_names=None):
    """Plot confusion matrix"""

    plt.figure(figsize=(10, 8))
    cm = results['confusion_matrix']

    if class_names is None:
        class_names = [f'Class {i}' for i in range(len(cm))]

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)

    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
```

```
    plt.tight_layout()
    plt.show()

class_names = ['World', 'Sports', 'Business', 'Sci/Tech']
model.eval()
results = evaluate_detailed(model, eval_dataloader,
class_names=class_names)

_ = print_evaluation_results(results, class_names=class_names)
```

```
============================================================
CLASSIFICATION EVALUATION REPORT
============================================================

Overall Accuracy: 0.9158

Detailed Classification Report:
----------------------------------------
              precision    recall  f1-score   support

       World       0.92      0.92      0.92      1235
      Sports       0.96      0.96      0.96      1197
    Business       0.89      0.90      0.90      1284
    Sci/Tech       0.90      0.88      0.89      1284

    accuracy                           0.92      5000
   macro avg       0.92      0.92      0.92      5000
weighted avg       0.92      0.92      0.92      5000


Per-class Metrics:
----------------------------------------
              precision  recall  f1-score    support
World            0.9198  0.9190    0.9194  1235.0000
Sports           0.9575  0.9607    0.9591  1197.0000
Business         0.8896  0.9034    0.8964  1284.0000
Sci/Tech         0.8993  0.8832    0.8912  1284.0000
accuracy         0.9158  0.9158    0.9158     0.9158
macro avg        0.9165  0.9166    0.9165  5000.0000
weighted avg     0.9158  0.9158    0.9158  5000.0000
```

```
plot_confusion_matrix(results, class_names=class_names)
```



Отчет по эффективности классификационной модели

Модель показывает высокое и сбалансированное качество:

- **Accuracy (точность)**: 0.92 (92%) - отличный показатель
- **Macro F1-score**: 0.92 - модель хорошо работает по всем классам

Лучший класс - Sports

- **Precision**: 0.96 - из всех предсказанных "спортивных" новостей 96% действительно о спорте
- **Recall**: 0.96 - модель находит 96% всех спортивных новостей в датасете
- **F1-score**: 0.96 - идеально сбалансированные метрики

Хорошие классы - World, Sci/Tech, Business

- **World**: F1=0.92, хорошо сбалансирован
- **Sci/Tech**: F1=0.89, немного хуже на recall (88%) - иногда пропускает релевантные статьи

- **Business**: F1=0.90, похожая ситуация с Sci/Tech

Business и Sci/Tech могут путаться между собой - так как темы иногда пересекаются По конфужн матрице :

- Business (2) → Sci/Tech (3): 93 ошибок
- Sci/Tech (3) → Business (2): 92 ошибок
- Вероятные причины:

Пересекающаяся лексика: "tech companies," "IT business," "startups"

Общие темы: новости о tech-компаниях (Apple, Google) могут относиться к обеим категориям

Схожие контексты: финансирование tech-проектов, IPO tech-компаний

Вобщем и целом модель готова с данными характеристиками к установке в продакшн

Возможные дальнйшие улучшения

Добавить новые признаки в модель, которые различают бизнес и тех-контекст:

- похожесть / близость с помощью семантической модели
- перплексити с помощью языковой модели

```
# Бизнес-ключевые слова
business_words = ['profit', 'revenue', 'stock', 'market',
'investment',
                  'financial', 'earnings', 'business', 'company',
'sales']

# Tech-ключевые слова
tech_words = ['research', 'technology', 'scientific', 'study',
'development',
              'innovation', 'lab', 'experiment', 'discovery',
'digital']
```

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!

Отчет по экспериментам

1. Предварительно провел анализ корпуса данных на баланс классом
2. Я провел следующие эксперименты:

- **Увеличение количества эпох**

  На последнем запуске видим что если модель при типе агрегации `mean` достигла максимума точности примерно на 5 эпохе и потом стала снижаться, то точность модели при типе агрегации `max` еще увеличивается и необходимо проверить что происходит при дальнейшем увеличении эпох. после 3 эпохи качество колебания стабилизировались и изменялись в пределах 0.890 - 0.905 Эксперимент немного увеличил точность модели до 90.72 (ранее 90.60). Дальнейшее обучение привело к переобучению и снижению качества до 90.14

- **Использование архитектуры GRU**

  При использовании GRU и 4 эпох обучения получил 91.68. Агрегация - max после 2 эпохи качество колебания стабилизировались и изменялись в пределах 0.910 - 0.915

- **Использование архитектуры GRU и агрегации last**

  При использовании GRU + агрегация - last и 4 эпох обучения получил 91.14. Эффективность уменьшилась после 1 эпохи качество колебания стабилизировались и изменялись в пределах 0.900 - 0.912

- **Использование архитектуры GRU 2 слоя и shapshot**

  Для лучшего результата я добавил еще 1 слой GRU всего 2 слоя. Также я включил сохранение снимков модели, чтобы потом воспроизвести лучший результат. после 2 эпохи качество колебания стабилизировались и изменялись в пределах 0.905 - 0.915 мой результат - **0.9158** на 3 балла

1. Самой оптимальной оказалась архитектура GRU 2 слоя с качеством **0.9158**

   Для воспроизведения результата веса модели можно скачать здесь https://disk.yandex.ru/d/lz07pH8dwSMn4Q

2. По итогам отчета опделелил что в целом качество модели приемлемо на всех классах и установил направления по дальнейшему улучшению, в частности лучшего разделания классов 'Business' и 'Sci/Tech'