# 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
```

```
Collecting datasets
  Downloading datasets-2.18.0-py3-none-any.whl (510 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 8.6 MB/s eta
0:00:00
ent already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 13.0 MB/s eta
0:00:00
ent already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in
/usr/local/lib/python3.10/dist-packages (from datasets) (4.66.2)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 11.6 MB/s eta
0:00:00
ent already satisfied: fsspec[http]<=2024.2.0,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.9.3)
Requirement already satisfied: huggingface-hub>=0.19.4 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in
```

```
/usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.19.4-
>datasets) (4.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2023.4)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1-
>pandas->datasets) (1.16.0)
Installing collected packages: dill, multiprocess, datasets
Successfully installed datasets-2.18.0 dill-0.3.8 multiprocess-0.70.16
```

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

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 collections import Counter
from typing import List
import string

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

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
True
```

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

```
{"type":"string"}
```

## 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')
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
```

```
access public models or datasets.
  warnings.warn(

{"model_id":"ddeded31528043d3a2dddaa58b1bb0f1","version_major":2,"vers
ion_minor":0}

{"model_id":"a52bd6879be04cbe82470c9ec7bad901","version_major":2,"vers
ion_minor":0}

{"model_id":"f5e5f956596b4b8dbc6464c9dbf6ec86","version_major":2,"vers
ion_minor":0}

{"model_id":"c200bfe4dfa34a01ad87ca7193b84887","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":"aed380cc9bde4be384ad5e1dddd61fc2","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)
```

## 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!

```
class my_RNN(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_classes: int = 4,
        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

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 = my_RNN(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type).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, eval_dataloader))
                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":"ec6533d2aeee4ca983107d05f988e9cc","version_major":2,"vers
ion_minor":0}

{"model_id":"013cb83932134993b6787453ed195637","version_major":2,"vers
ion_minor":0}

{"model_id":"9c79a10d9bd945a68dee70fd3ecdee43","version_major":2,"vers
ion_minor":0}

{"model_id":"625ed3283acc40ae9690c042de627dec","version_major":2,"vers
ion_minor":0}

{"model_id":"f7947e20fe48481b87c65e0b1fb9b620","version_major":2,"vers
ion_minor":0}
```


```
Starting training for mean
```

{"model_id":"e271d12d298f48409a0e1c4e536a7db0","version_major":2,"version_minor":0}

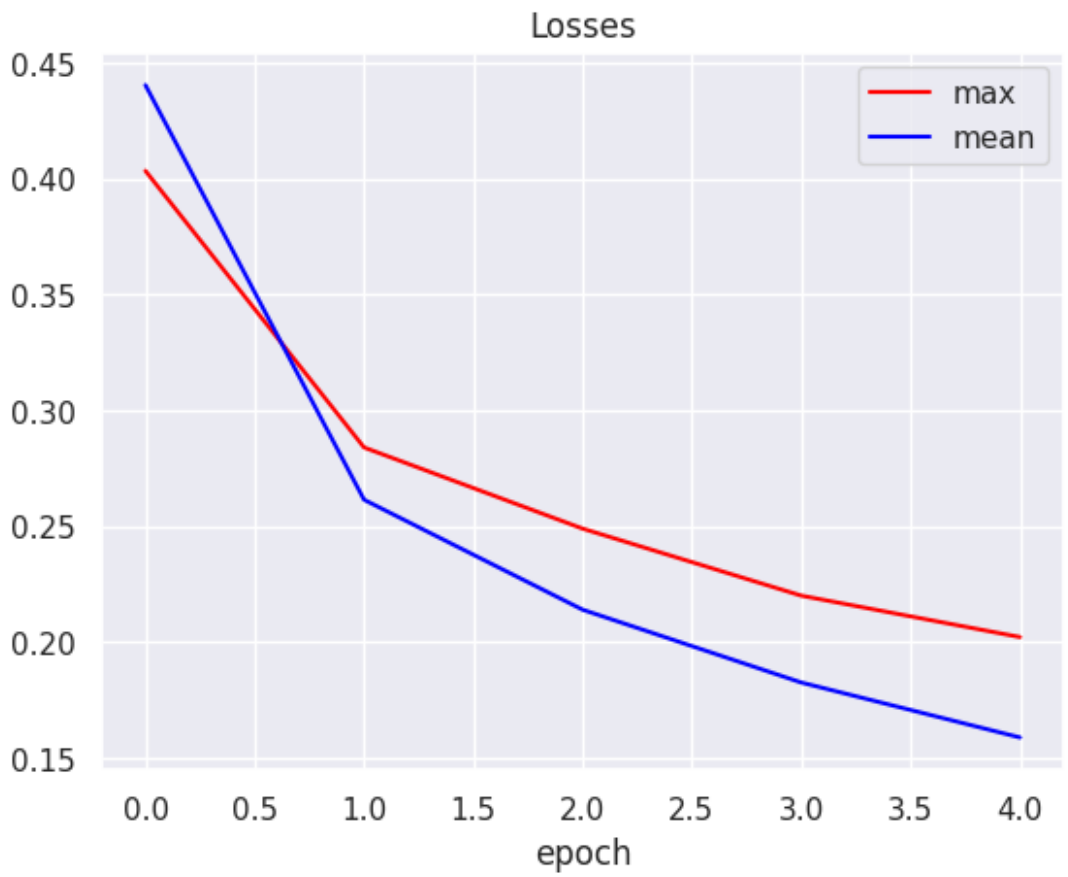
{"model_id":"748167541d6e41e89489cbfe004f8d2b","version_major":2,"version_minor":0}

{"model_id":"f9e85bd0366d494592f75a83aa54c214","version_major":2,"version_minor":0}

{"model_id":"3f74222019814ca4adb2e8d8056734b2","version_major":2,"version_minor":0}

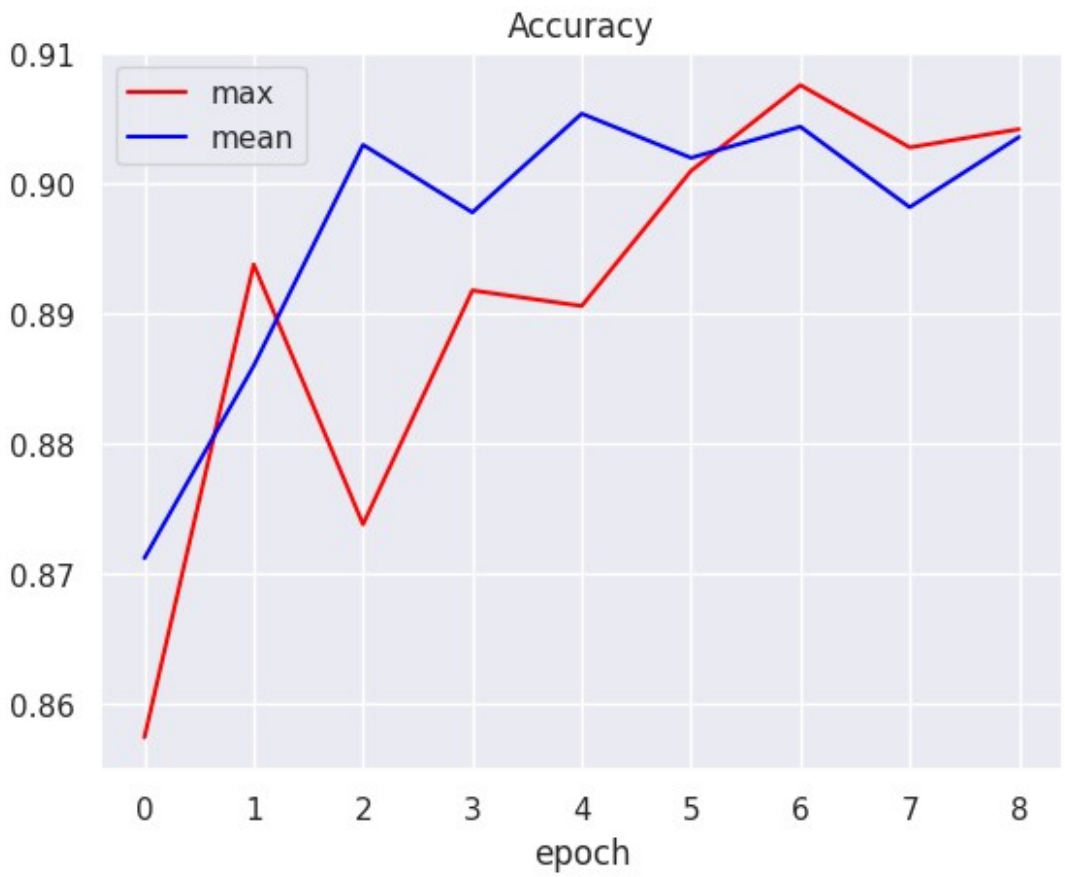
{"model_id":"606d7b09ca68477782212378d5e77d65","version_major":2,"version_minor":0}

```
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()
```




```
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: 90.76
Лучшая accuracy для подхода mean: 90.54
```




# Эксперимент 1

**Изменение параметра num_layers**

```
class ex1_my_RNN(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_lay: int,
num_classes: int = 4,
        aggregation_type: str = 'max'
        ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.RNN(hidden_dim, hidden_dim, num_layers=num_lay,
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

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


losses_type_2 = {}
acc_type_2 = {}
losses_type_3 = {}
acc_type_3 = {}
losses_type_4 = {}
acc_type_4 = {}

for num in [2,3,4]:
    for aggregation_type in ['max', 'mean']:
        print(f"Starting training for {aggregation_type}")
        losses = []
        acc = []

        model = ex1_my_RNN(
            hidden_dim=256, vocab_size=len(vocab), num_lay=num,
aggregation_type=aggregation_type).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, eval_dataloader))
                    model.train()

            losses.append(sum(epoch_losses) / len(epoch_losses))
        if num == 2:
            losses_type_2[aggregation_type] = losses
            acc_type_2[aggregation_type] = acc
        elif num == 3:
            losses_type_3[aggregation_type] = losses
            acc_type_3[aggregation_type] = acc
        else:
            losses_type_4[aggregation_type] = losses
            acc_type_4[aggregation_type] = acc


Starting training for max

{"model_id":"2652a4430f494af9bd71f8eae1c39dd4","version_major":2,"vers
ion_minor":0}

{"model_id":"5e1129ce40e8418b824cc354194dbb54","version_major":2,"vers
ion_minor":0}

{"model_id":"2d12b9afc488434599db8f6771c4ae16","version_major":2,"vers
ion_minor":0}

{"model_id":"8426656656a84cefa6a8b82206bbe888","version_major":2,"vers
ion_minor":0}

{"model_id":"311d11de97b94c5f8f63be18bc1d36a6","version_major":2,"vers
ion_minor":0}

Starting training for mean

{"model_id":"227402eb97b147abbfc6d9795b32e562","version_major":2,"vers
ion_minor":0}

{"model_id":"9da5d032e8cc4ecb97cd1acc2046781d","version_major":2,"vers
ion_minor":0}

{"model_id":"aec066e2e5ea4c38a87d1d79e8bc77b7","version_major":2,"vers
ion_minor":0}
```

{"model_id":"1dfaff4cf7fe4f21b54aa713f21ec121","version_major":2,"version_minor":0}

{"model_id":"eb7a1741ba574ba2a662218cb9a4b392","version_major":2,"version_minor":0}

```
Starting training for max
```

{"model_id":"3fa5abd2c94b400abc8f039293167d76","version_major":2,"version_minor":0}

{"model_id":"2c0218332e99427984fa2f102051cb4f","version_major":2,"version_minor":0}

{"model_id":"8ff9f623414c43868d1e859ede415f11","version_major":2,"version_minor":0}

{"model_id":"655b78db379f43529c98440246f2c658","version_major":2,"version_minor":0}

{"model_id":"98d0eabd5b8446baaf59385674ac803b","version_major":2,"version_minor":0}

```
Starting training for mean
```

{"model_id":"35db05189b6a467199172458e954ffce","version_major":2,"version_minor":0}

{"model_id":"8990c3b6c5664f279b84782a47a5667c","version_major":2,"version_minor":0}

{"model_id":"ea6a39b84d83434c9b857546b237d39d","version_major":2,"version_minor":0}

{"model_id":"805bc595f86e4b5d931c9521382f3652","version_major":2,"version_minor":0}

{"model_id":"66207eae6b9048eeb9dddd7571cd3716","version_major":2,"version_minor":0}

```
Starting training for max
```

{"model_id":"6adc7177c91b4bf9967742b1b0ef154b","version_major":2,"version_minor":0}

{"model_id":"72cedd5f4d184bb68539e1b27a9a69db","version_major":2,"version_minor":0}

{"model_id":"59a8fa41d3ab4e089df0c0767a8d412e","version_major":2,"version_minor":0}

{"model_id":"1aa8de5723c8432bb2ccfeaf20e00985","version_major":2,"version_minor":0}

{"model_id":"8845d731538e4e2082fb94c242dbf860","version_major":2,"version_minor":0}

```
Starting training for mean
```

{"model_id":"9ec71f6fd28c48d6a5fb0e88c762bdd3","version_major":2,"version_minor":0}

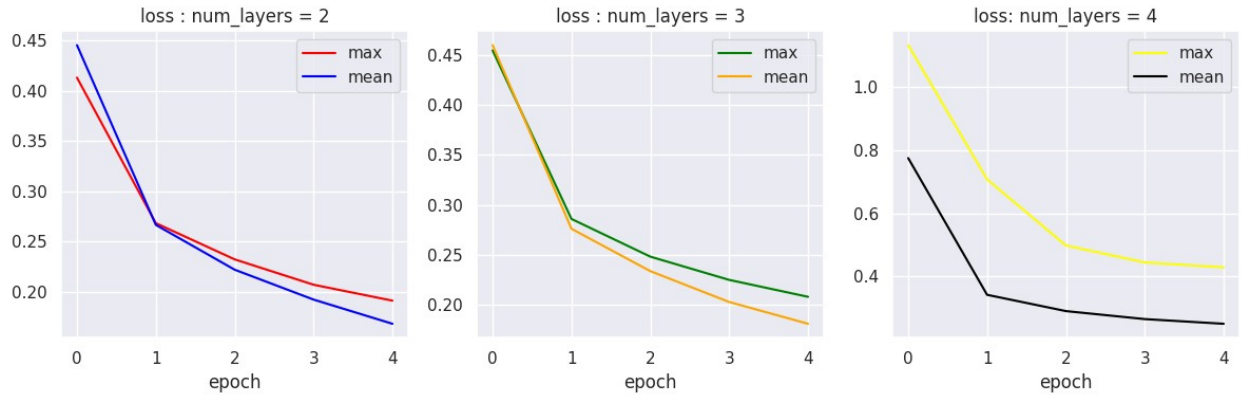
{"model_id":"030a6959cf2f43859f6c2a5be900e37c","version_major":2,"version_minor":0}

{"model_id":"67e6ee09c8d34f53838a455078ce1d67","version_major":2,"version_minor":0}

{"model_id":"6c3e0e0b46424ebea7e96ea19ee376a2","version_major":2,"version_minor":0}

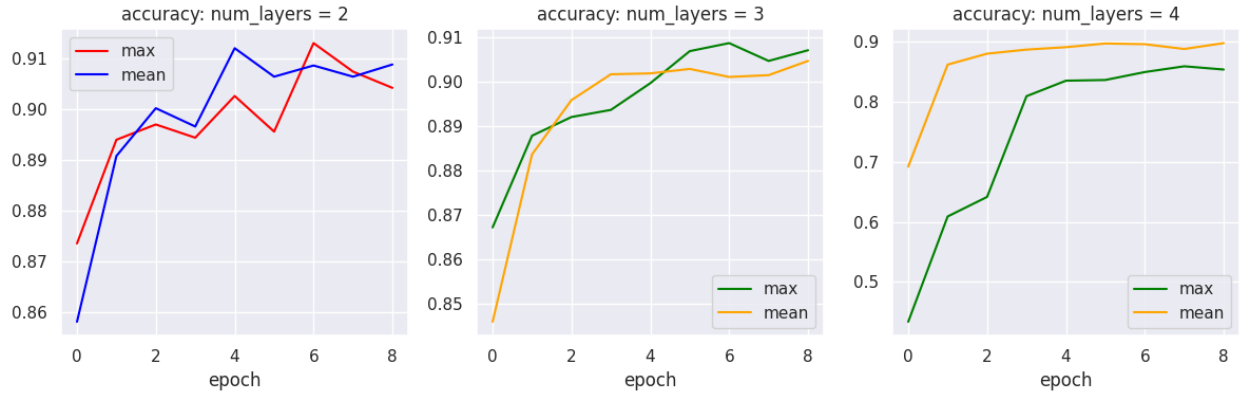
{"model_id":"23637aaef7784df7a9db8611a53e7b26","version_major":2,"version_minor":0}

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for (name, values), color in zip(losses_type_2.items(), ['red',
'blue']):
    axes[0].plot(np.arange(len(values)), values, color=color,
label=name)

axes[0].set_title('loss : num_layers = 2')
axes[0].set_xlabel("epoch")
axes[0].legend()

for (name, values), color in zip(losses_type_3.items(), ['green',
'orange']):
    axes[1].plot(np.arange(len(values)), values, color=color,
label=name)

axes[1].set_title('loss : num_layers = 3')
axes[1].set_xlabel("epoch")
axes[1].legend()

for (name, values), color in zip(losses_type_4.items(), ['yellow',
'black']):
    axes[2].plot(np.arange(len(values)), values, color=color,
label=name)

axes[2].set_title('loss: num_layers = 4')
axes[2].set_xlabel("epoch")
axes[2].legend()

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




```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for (name, values), color in zip(losses_type_2.items(), ['red',
'blue']):
    axes[0].plot(np.arange(len(acc_type_2[name][1:])),
acc_type_2[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с двумя слоями:
{(max(acc_type_2[name]) * 100):.2f}")

axes[0].set_title('accuracy: num_layers = 2')
axes[0].set_xlabel("epoch")
axes[0].legend()

for (name, values), color in zip(losses_type_3.items(), ['green',
'orange']):
    axes[1].plot(np.arange(len(acc_type_3[name][1:])),
acc_type_3[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с тремя слоями:
{(max(acc_type_3[name]) * 100):.2f}")

axes[1].set_title('accuracy: num_layers = 3')
axes[1].set_xlabel("epoch")
axes[1].legend()

for (name, values), color in zip(losses_type_4.items(), ['green',
'orange']):
    axes[2].plot(np.arange(len(acc_type_4[name][1:])),
acc_type_4[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с четырьмя слоями:
{(max(acc_type_4[name]) * 100):.2f}")

axes[2].set_title('accuracy: num_layers = 4')
axes[2].set_xlabel("epoch")
axes[2].legend()

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

```
Лучшая accuracy для подхода max с двумя слоями: 91.30
Лучшая accuracy для подхода mean с двумя слоями: 91.20
Лучшая accuracy для подхода max с тремя слоями: 90.86
Лучшая accuracy для подхода mean с тремя слоями: 90.46
Лучшая accuracy для подхода max с четырьмя слоями: 85.92
Лучшая accuracy для подхода mean с четырьмя слоями: 89.78
```




# Эксперимент 2

Попробуем GRU

```python
class ex2_my_GRU(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_lay: int,
num_classes: int = 4,
        aggregation_type: str = 'max'
        ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)

        self.rnn = nn.GRU(input_size = hidden_dim, hidden_size =
hidden_dim, num_layers = num_lay,
                          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

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 = ex2_my_GRU(
        hidden_dim=256, vocab_size=len(vocab),num_lay = 1,
aggregation_type=aggregation_type).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, eval_dataloader))
```

```
                model.train()

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

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

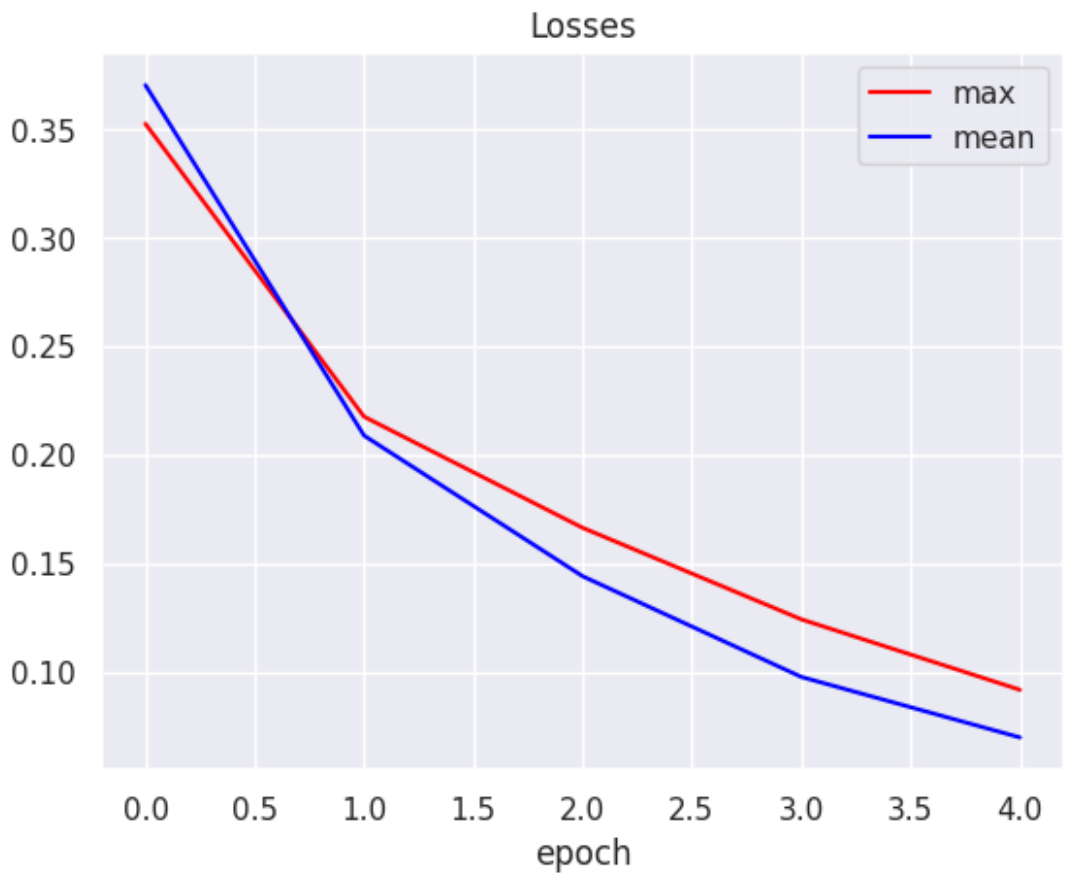
```
Starting training for max
```


```
{"model_id":"e4ae5b6f02334c3ca7ef416c45e941e7","version_major":2,"vers
ion_minor":0}

{"model_id":"d4ac64dc031d4d628f5313ffcdb6cf27","version_major":2,"vers
ion_minor":0}

{"model_id":"715179af0772431099eac07e05263dd5","version_major":2,"vers
ion_minor":0}

{"model_id":"3f49fbb0c4674504a1ee1c0f4cedd1fd","version_major":2,"vers
ion_minor":0}

{"model_id":"7830f609a2d443e3b8813aece3382800","version_major":2,"vers
ion_minor":0}
```


```
Starting training for mean
```

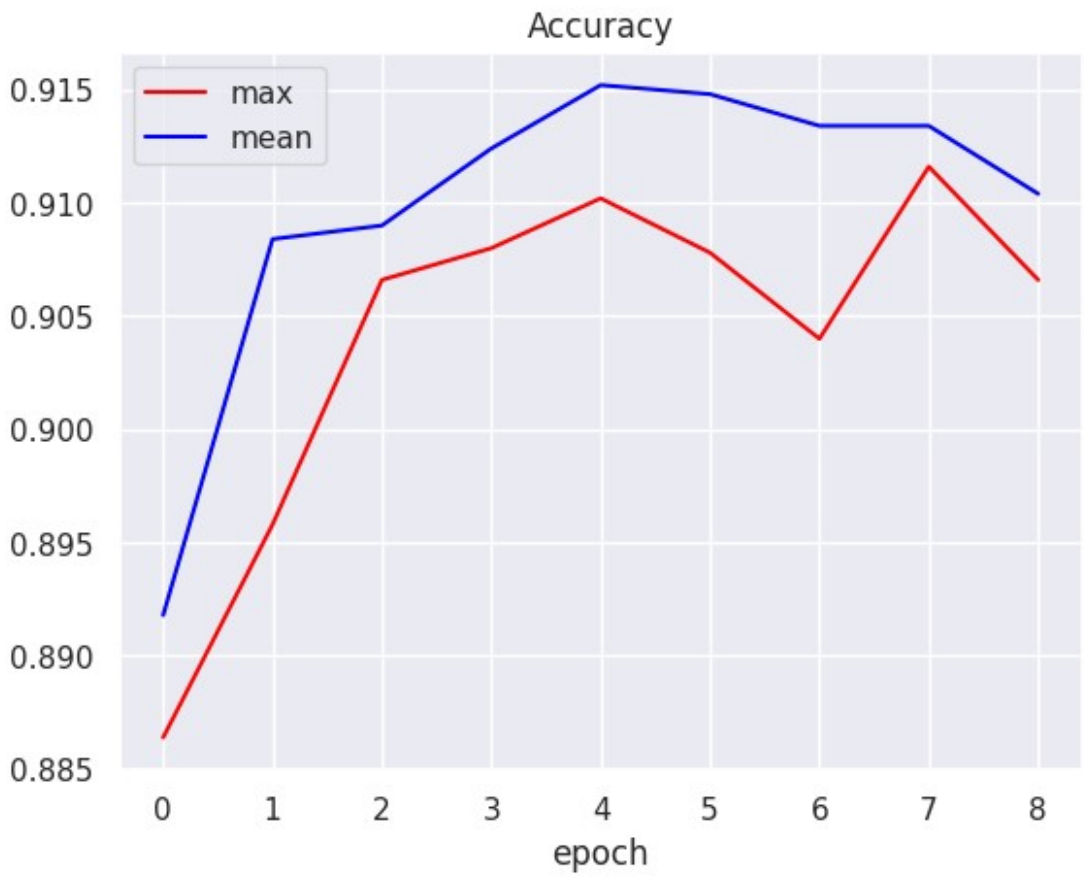

```
{"model_id":"c7356396c5174b079f44d4e4b1808f44","version_major":2,"vers
ion_minor":0}

{"model_id":"c189edd40c5043abb376b1270044b10a","version_major":2,"vers
ion_minor":0}

{"model_id":"4cae1fa294ea4ba9844d4558730a4c0f","version_major":2,"vers
ion_minor":0}

{"model_id":"1f9ef4481a524c729c42699338c3707b","version_major":2,"vers
ion_minor":0}

{"model_id":"cb2c1c5b93504e0bbfb3651800a7e449","version_major":2,"vers
ion_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()
```




```
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: 91.16
Лучшая accuracy для подхода mean: 91.52
```




# Эксперимент 3

Подбор гиперпараметров

```
class ex3_my_RNN(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, dropout: float,
num_lay: int, num_classes: int = 4,
        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=dropout)
```

```
        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

# возьмем сразу больше эпох, и анализируя графики, сделаем выводы,
какое количество эпох оптимально
num_epoch = 10
eval_steps = len(train_dataloader) // 2


losses_type_2 = {}
acc_type_2 = {}
losses_type_3 = {}
acc_type_3 = {}
losses_type_4 = {}
acc_type_4 = {}
# проверим другие значение дропаутов
for drop in [0.1,0.2,0.3]:
    for aggregation_type in ['max', 'mean']:
        print(f"Starting training for {aggregation_type}")
        losses = []
        acc = []

        model = ex3_my_RNN(
            hidden_dim=256, vocab_size=len(vocab),dropout=drop,
num_lay=1, aggregation_type=aggregation_type).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, eval_dataloader))
                    model.train()

            losses.append(sum(epoch_losses) / len(epoch_losses))
        if abs(drop - 0.1) < 0.05:
            losses_type_2[aggregation_type] = losses
            acc_type_2[aggregation_type] = acc
        elif abs(drop - 0.2) < 0.05:
            losses_type_3[aggregation_type] = losses
            acc_type_3[aggregation_type] = acc
        else:
            losses_type_4[aggregation_type] = losses
            acc_type_4[aggregation_type] = acc


Starting training for max

{"model_id":"99ec70103985483b9d0fa56b0bc1a447","version_major":2,"vers
ion_minor":0}

{"model_id":"d0e29a87931540b1aecde306c391eccf","version_major":2,"vers
ion_minor":0}

{"model_id":"fced569bdd8e46fca5824089fceeec3a","version_major":2,"vers
ion_minor":0}

{"model_id":"4ae26929c49a46a89382e2e183a0bfc5","version_major":2,"vers
ion_minor":0}

{"model_id":"9338fce2c7b54785bd306df07b2f8093","version_major":2,"vers
ion_minor":0}

{"model_id":"923116d7af6c4dbca5732c9f7b752075","version_major":2,"vers
ion_minor":0}

{"model_id":"86f0a21cdbf845258b9894da1b428df4","version_major":2,"vers
ion_minor":0}

{"model_id":"0185b832f0ec4d58be0f811c58cae42c","version_major":2,"vers
ion_minor":0}
```

{"model_id":"281bae6bebfb4d388a77f2c0507669c0","version_major":2,"version_minor":0}

{"model_id":"103d6e9ba843490a98d098ffb7b4565b","version_major":2,"version_minor":0}

```
Starting training for mean
```

{"model_id":"5dcc78110cda46c38fdfd5247bdf2484","version_major":2,"version_minor":0}

{"model_id":"52f8592e5b75410c94eb2efa74cdeb8d","version_major":2,"version_minor":0}

{"model_id":"8f2a92faf53a4056bd54251672a369e7","version_major":2,"version_minor":0}

{"model_id":"b55220126f2b448a9a0c52154ff13215","version_major":2,"version_minor":0}

{"model_id":"127a162b8c004e439e82ce0ebd20397a","version_major":2,"version_minor":0}

{"model_id":"39d40e468c4e48a3a171532e9fd13e64","version_major":2,"version_minor":0}

{"model_id":"5a92d2ebe19749bc8848e6e2bd2c7e38","version_major":2,"version_minor":0}

{"model_id":"ea966801b6c54a04a1577895f28d1646","version_major":2,"version_minor":0}

{"model_id":"181a4b1b780340ccb0e3e052d455fe8d","version_major":2,"version_minor":0}

{"model_id":"5fe067c6b1e347fd828054a4667a71ef","version_major":2,"version_minor":0}

```
Starting training for max
```

{"model_id":"2c3a3fe462fa4c5b81b06121b9fc9b8e","version_major":2,"version_minor":0}

{"model_id":"0234752f63a14fe193725442217a8018","version_major":2,"version_minor":0}

{"model_id":"68b8067d8f3b44b49e0a0cea0dfd030c","version_major":2,"version_minor":0}

{"model_id":"0f624c2bed2f4e52a349d5490ad2ce22","version_major":2,"version_minor":0}

{"model_id":"93074a7f91f04e8086d933963b1003cd","version_major":2,"version_minor":0}

{"model_id":"2a764489db82449e94facf5110d5206c","version_major":2,"version_minor":0}

{"model_id":"ffe866729d134ce8879dbf7e51bb0031","version_major":2,"version_minor":0}

{"model_id":"da243fdc5d744579a9ff4d154f5703a6","version_major":2,"version_minor":0}

{"model_id":"42f0b42a27b84cc293c4cbf0d4d814ec","version_major":2,"version_minor":0}

{"model_id":"1daf41e5fe0c4a0185b91aed3cebfe01","version_major":2,"version_minor":0}

```
Starting training for mean
```

{"model_id":"ee488e75ad9344beae0816800c80ac27","version_major":2,"version_minor":0}

{"model_id":"50c4da68c3a44b53ba0b5732c1dcf661","version_major":2,"version_minor":0}

{"model_id":"d9a327c9a66144fcace4af64d30c987f","version_major":2,"version_minor":0}

{"model_id":"798c3eaf9d024f49a52f6ba9217c306a","version_major":2,"version_minor":0}

{"model_id":"8876ffd903964d0e9e8face3ffb064ef","version_major":2,"version_minor":0}

{"model_id":"dc57aa14a3414648a71d333574591538","version_major":2,"version_minor":0}

{"model_id":"0afe165b6fd744b691e3b71552cae36b","version_major":2,"version_minor":0}

{"model_id":"82debbd58f1a4e349ca1b27923720e6a","version_major":2,"version_minor":0}

{"model_id":"859fe1de66834ae5b99e681e865fd54f","version_major":2,"version_minor":0}

{"model_id":"2082bdfc14e24fe3a3b6da4f01c31f4a","version_major":2,"version_minor":0}

```
Starting training for max
```

{"model_id":"e27d51f710fe4402843bacff32b97f2c","version_major":2,"version_minor":0}

{"model_id":"e2fcbf4caca04d8ebbc730f436ab102b","version_major":2,"version_minor":0}

{"model_id":"0890f04535e5484b87f74ba9a797ce14","version_major":2,"version_minor":0}

{"model_id":"d05b65a305d0460498ad46610c902c59","version_major":2,"version_minor":0}

{"model_id":"fad96288af454a09bb9ae2330a21d5bb","version_major":2,"version_minor":0}

{"model_id":"17e9193a81f445bfb7ac18ac95daaa4c","version_major":2,"version_minor":0}

{"model_id":"ea3ca8effe424e1db7d75b414a9ca4a0","version_major":2,"version_minor":0}

{"model_id":"9c6f990ea7e549ecbb6592770e34dda3","version_major":2,"version_minor":0}

{"model_id":"da7b365e3dbd4cba81f43ae2d59c34a7","version_major":2,"version_minor":0}

{"model_id":"cc5e0a8eb6a74376b3d8c17186a77126","version_major":2,"version_minor":0}

```
Starting training for mean
```

{"model_id":"eacc4621d6534fd4be4ffa1014966ce8","version_major":2,"version_minor":0}

{"model_id":"2b618a1b52524a9bacc8d5b5f6639927","version_major":2,"version_minor":0}

{"model_id":"c6494f6968494edd8bebe49214848994","version_major":2,"version_minor":0}

{"model_id":"9fb239887033475694 39f2732b0dfc3f","version_major":2,"version_minor":0}

{"model_id":"a4be750970184982b1875a553d530094","version_major":2,"version_minor":0}

{"model_id":"cde0255e12e5419a9cef51397601b845","version_major":2,"version_minor":0}

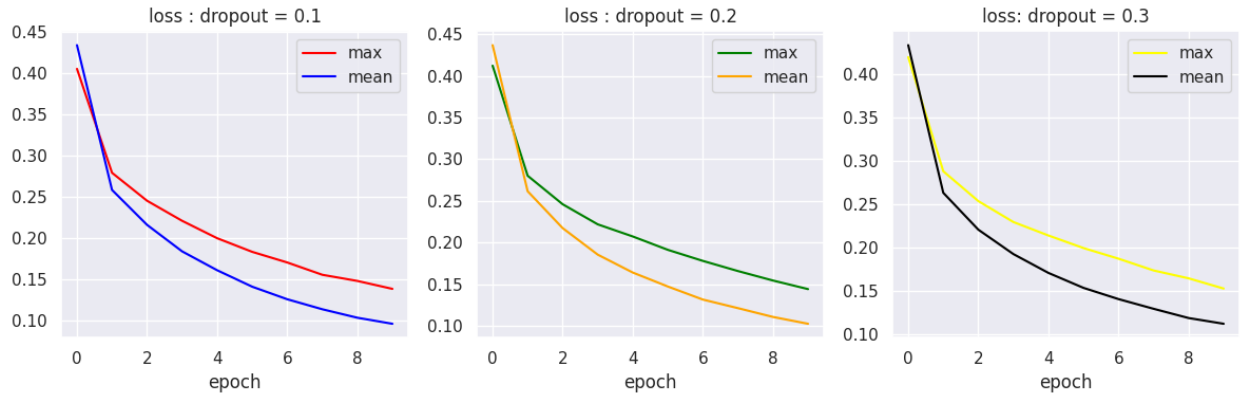
{"model_id":"27a8a201dfad40a097525bcaf77669e7","version_major":2,"version_minor":0}

{"model_id":"1503ff13efa14a4099d5021958631ebb","version_major":2,"version_minor":0}

{"model_id":"3a94d5b4678b4d83a28bc37a7121e3bf","version_major":2,"version_minor":0}

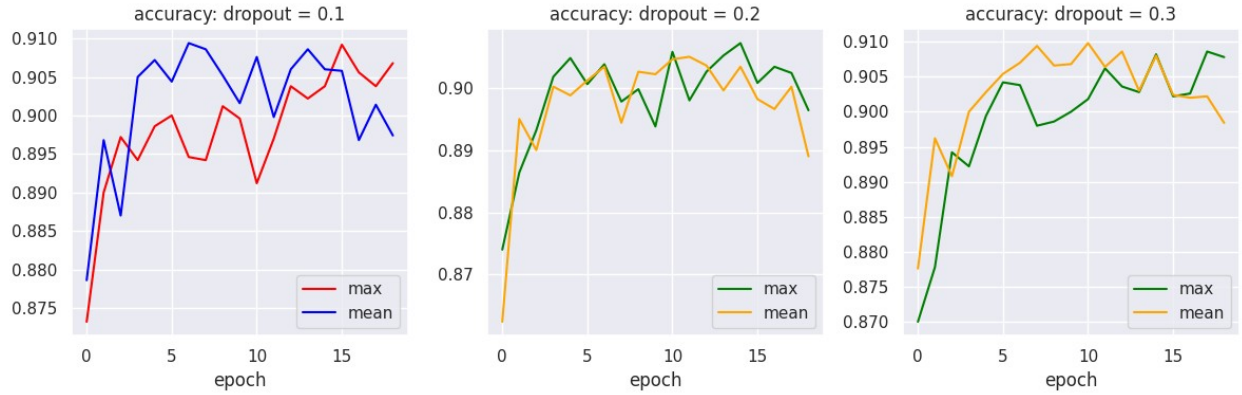
{"model_id":"a492bf31971a4a33a29a82982d61c014","version_major":2,"version_minor":0}

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for (name, values), color in zip(losses_type_2.items(), ['red',
'blue']):
    axes[0].plot(np.arange(len(values)), values, color=color,
label=name)

axes[0].set_title('loss : dropout = 0.1')
axes[0].set_xlabel("epoch")
axes[0].legend()

for (name, values), color in zip(losses_type_3.items(), ['green',
'orange']):
    axes[1].plot(np.arange(len(values)), values, color=color,
label=name)

axes[1].set_title('loss : dropout = 0.2')
axes[1].set_xlabel("epoch")
axes[1].legend()

for (name, values), color in zip(losses_type_4.items(), ['yellow',
'black']):
    axes[2].plot(np.arange(len(values)), values, color=color,
label=name)

axes[2].set_title('loss: dropout = 0.3')
axes[2].set_xlabel("epoch")
axes[2].legend()

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




```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for (name, values), color in zip(losses_type_2.items(), ['red',
'blue']):
    axes[0].plot(np.arange(len(acc_type_2[name][1:])),
acc_type_2[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с dropout = 0.1:
{(max(acc_type_2[name]) * 100):.2f}")

axes[0].set_title('accuracy: dropout = 0.1')
axes[0].set_xlabel("epoch")
axes[0].legend()

for (name, values), color in zip(losses_type_3.items(), ['green',
'orange']):
    axes[1].plot(np.arange(len(acc_type_3[name][1:])),
acc_type_3[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с dropout = 0.2:
{(max(acc_type_3[name]) * 100):.2f}")

axes[1].set_title('accuracy: dropout = 0.2')
axes[1].set_xlabel("epoch")
axes[1].legend()

for (name, values), color in zip(losses_type_4.items(), ['green',
'orange']):
    axes[2].plot(np.arange(len(acc_type_4[name][1:])),
acc_type_4[name][1:], color=color, label=name)
    print(f"Лучшая accuracy для подхода {name} с dropout = 0.3:
{(max(acc_type_4[name]) * 100):.2f}")

axes[2].set_title('accuracy: dropout = 0.3')
axes[2].set_xlabel("epoch")
axes[2].legend()

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

```
Лучшая accuracy для подхода max с dropout = 0.1: 90.92
Лучшая accuracy для подхода mean с dropout = 0.1: 90.94
Лучшая accuracy для подхода max с dropout = 0.2: 90.72
Лучшая accuracy для подхода mean с dropout = 0.2: 90.50
Лучшая accuracy для подхода max с dropout = 0.3: 90.86
Лучшая accuracy для подхода mean с dropout = 0.3: 90.98
```




# Финальная модель

Учтем все эксперименты для получения лучшего результата

```python
class Final_my_GRU(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_lay: int,
num_classes: int = 4,
        aggregation_type: str = 'max'
        ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)

        self.rnn = nn.GRU(input_size = hidden_dim, hidden_size =
hidden_dim, num_layers = num_lay,
                          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

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 = Final_my_GRU(
        hidden_dim=256, vocab_size=len(vocab),num_lay = 2,
aggregation_type=aggregation_type).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, eval_dataloader))
```

```
                model.train()

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

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

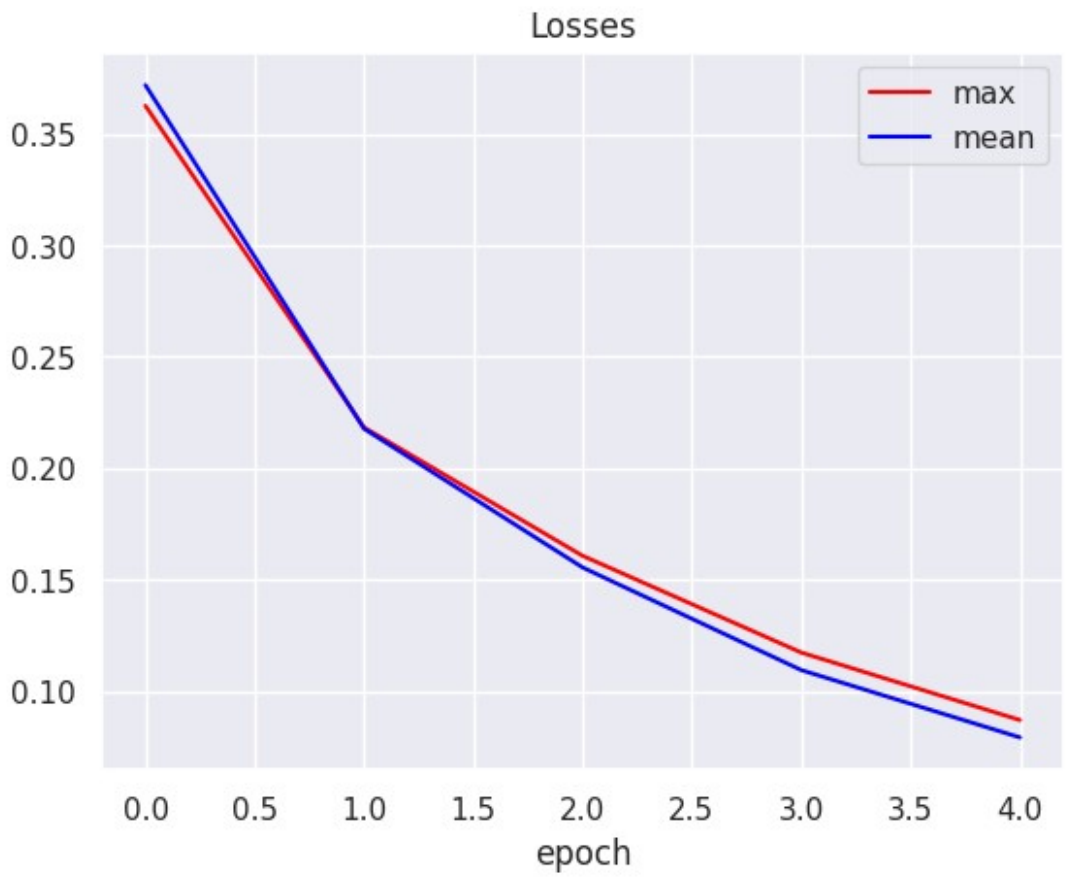
```
Starting training for max
```


```
{"model_id":"821643083d184ea4a6d614c0de3249bf","version_major":2,"vers
ion_minor":0}

{"model_id":"e794fff6af8f4522acd9bd22694e3ad0","version_major":2,"vers
ion_minor":0}

{"model_id":"8d39e6385a76479897ed0298318bb24a","version_major":2,"vers
ion_minor":0}

{"model_id":"96aabe3cef7448a3bfc01450a4926015","version_major":2,"vers
ion_minor":0}

{"model_id":"692ffacc4d9d447c99a9bfb8a9184457","version_major":2,"vers
ion_minor":0}
```


```
Starting training for mean
```


```
{"model_id":"b40d3b9625c845fab6d2bd4a64c48ee5","version_major":2,"vers
ion_minor":0}

{"model_id":"477b64f2fb8e418bb097e233a6f6d35b","version_major":2,"vers
ion_minor":0}

{"model_id":"d7a17d041e894844b0a6b445d0c91301","version_major":2,"vers
ion_minor":0}

{"model_id":"c40fb3d69b864240b2d228f4d08336fd","version_major":2,"vers
ion_minor":0}

{"model_id":"19fa938248ea40b28a62965aee1faf83","version_major":2,"vers
ion_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()
```




```
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: 91.24
Лучшая accuracy для подхода mean: 91.62
```


# Получено лучшее accuracy: 0.9162 > 0.915

# Вывод:

- Первый эксперемент заключался в том, чтобы проверить разное количество слоев RNN. Исходя из результатов эксперимента можно слелать вывод ,что оптимально брать два слоя. Если брать больше слоёв, 3, или 4, то результат только ухудшался, причем значительно. То есть для конкретного корпуса использование архитектуры с большим количеством слоев не оправдано. Можно так же отметить, что увеличесние количества слоев практически не влияло на скорость обучения.
- Во втором эксперименте обучалась GRU с теми же гиперпараметрами, что и обычная RNN. Результат был значительно выше, и даже с одним слоем accuracy перевалило за 0.915.
- В третьем эксперименте было увеличено количество эпох и рассмотрены различные dropout_rate. Увеличение droput_rate привело лишь к ухудшению качества модели на данном корпусе, хотя и не значительно. Количество эпох большее 5 (хотя об эпохах подробней в следующем пункте) для данных моделей, как показывают все графики, излишне.

- Во всех моделях на протяжении всех эпох обучения лоссы стабильно сходятся, но скорость сходимости падает довольно быстро у всех, кроме GRU.
- Но вот accuracy с методом аггрегации 'mean' судя по всем экспериментам показывает наилучшие результаты вообще на эпохе 3-4, дальше она сильно падает, что говорит о переобучении данного метода. Метод 'max' наоборот, постепенно растет и получает наилучшие значени чуть позднее 5-ой эпохи.
- Можно сделать однозначный вывод, что колебания качества обученных моделей существенно зависят от эпохи обучения
- Оптимальной моделью признана GRU с двумя слоями и методом аггрегации 'mean'. Она и показала наилучшее accuracy