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.

```
# import sys
# !{sys.executable} -m pip install seaborn
```

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
[nltk_data]     /Users/olyamukhomorova/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.

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

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":"f779caf4aec3421c82167aa8d7257083","version_major":2,"version_minor":0}

```
Размер словаря: 11842
```

```
len(words), len(vocab)
```

```
(102166, 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.

Здесь я копирую класс `CharLM` из семинара без изменений:

```
class CharLM(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

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

```
/usr/local/Cellar/jupyterlab/4.2.3/libexec/lib/python3.12/site-
packages/transformers/utils/generic.py:441: UserWarning:
torch.utils._pytree._register_pytree_node is deprecated. Please use
torch.utils._pytree.register_pytree_node instead.
  _torch_pytree._register_pytree_node(
```

Для визуализации кривых обучения и качества я определяю новую функцию `plot_learning_curves` и маппинг в цвета для соответствующих экспериментов `colors`.

В этом маппинге для датасетов `train` & `eval` я задаю соответствующие пары темного и светлого цветов (например, 'cornflowerblue' и 'lightskyblue') чтобы можно было легко визуально сравнивать как train vs. eval, так и eval-ы для разных экспериментов.

```
colors = {
    'train': ['cornflowerblue', 'orangered', 'olivedrab',
'mediumpurple', 'royalblue', 'orange', 'tomato', 'crimson', 'hotpink',
'brown'],
    'eval': ['lightskyblue', 'darkorange', 'yellowgreen', 'plum',
'lightblue', 'gold', 'lightsalmon', 'palevioletred', 'lightpink',
'rosybrown'],
}

def plot_learning_curves(losses_type, acc_type, plot_sets=['train',
'eval'], param_sets=[], param_name='', acc_start_from=1):

    plt.figure(figsize=(12,5))

    plt.subplot(121)
    for set_name in plot_sets:
        colormap = dict(zip(losses_type.keys(), colors[set_name]))
        for param in param_sets:
            label = 'Set: ' + set_name + ', ' + param_name + ': '+
str(param)
            plt.plot(np.arange(1, num_epoch+1, 1), losses_type[param]
[set_name], color=colormap[param], label=label)
    plt.title('Avg loss value over epochs')
    plt.xlabel('Epoch')
    plt.xticks(np.arange(1, num_epoch+1, 1), fontsize=10)
    plt.legend(prop={'size': 8})


    plt.subplot(122)
    for set_name in plot_sets:
        colormap = dict(zip(acc_type.keys(), colors[set_name]))
        for param in param_sets:
            label = 'Set: ' + set_name + ', ' + param_name + ': '+
str(param)
            plt.plot(np.arange(0.5+0.5, num_epoch+0.5, 0.5),
acc_type[param][set_name][acc_start_from:], color=colormap[param],
label=label)
            print(f"Лучшая accuracy для {param_name}={str(param)} на
{set_name}: {(max(acc_type[param][set_name]) * 100):.2f}")
            print(f"Последняя accuracy для {param_name}={str(param)}
на {set_name}: {(acc_type[param][set_name][-1] * 100):.2f}")
            print()
    plt.title('Accuracy over epochs')
    plt.xlabel('Epoch')
    plt.xticks(np.arange(0.5+0.5, num_epoch+0.5, 0.5), fontsize=10)
    plt.legend(prop={'size': 8})
```

```
    plt.show()
```

Запускаю модель из семинара:

```
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 = {
        'eval': [],
        'train': [],
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = CharLM(
        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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
```

```
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for max
```


```
{"model_id":"56a2da199dea4178890d99eaffa77900","version_major":2,"vers
ion_minor":0}

{"model_id":"8dfa5a7f662c441b82790b1e131d3e65","version_major":2,"vers
ion_minor":0}

{"model_id":"7721aa5ec2864cd18c47d32c6c89edc0","version_major":2,"vers
ion_minor":0}

{"model_id":"88e2ac5edbe8409aa539cc6cfd68a33a","version_major":2,"vers
ion_minor":0}

{"model_id":"91cb8dd7f11e4b018a8ba3ac465220ec","version_major":2,"vers
ion_minor":0}
```

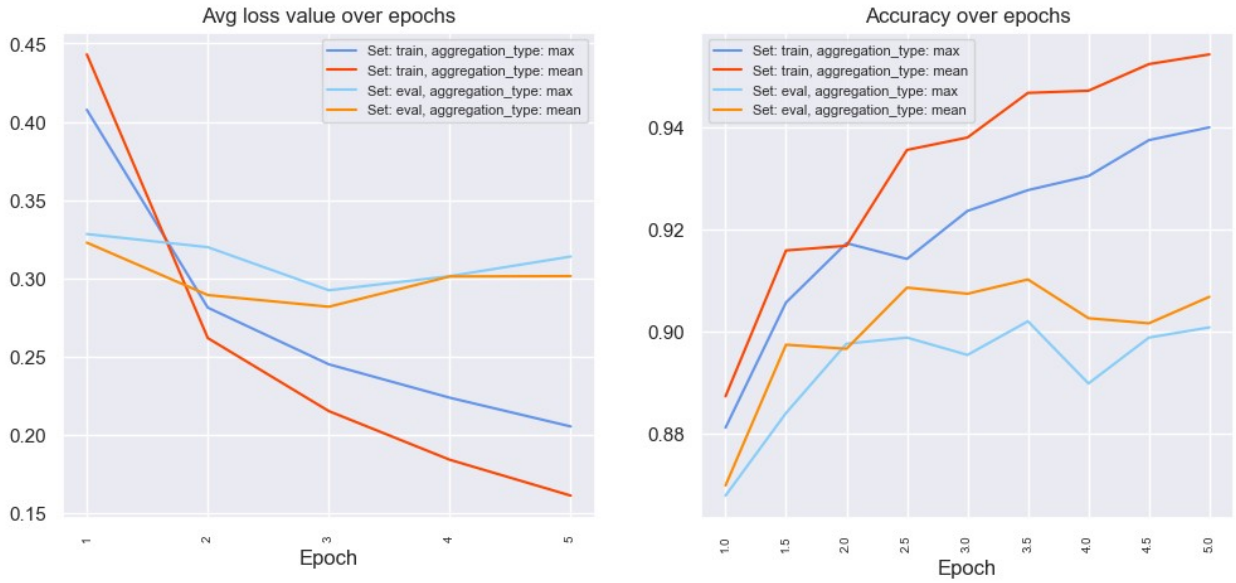

```
Starting training for mean
```


```
{"model_id":"091dd47892cf43929dc6cd6e7e1b0e46","version_major":2,"vers
ion_minor":0}

{"model_id":"08c17ab79a1449b2af640fb09673d4e6","version_major":2,"vers
ion_minor":0}

{"model_id":"15cc10af3e564962be552ae9a73f2807","version_major":2,"vers
ion_minor":0}

{"model_id":"08687164baf9473190d105d4af3d587b","version_major":2,"vers
ion_minor":0}

{"model_id":"7538eaff580e46399e89b5aee6b05bd2","version_major":2,"vers
ion_minor":0}
```


Визуализирую полученное качество:

```
plot_learning_curves(losses_type, acc_type, param_sets=['max',
'mean'], param_name='aggregation_type')
```

```
Лучшая accuracy для aggregation_type=max на train: 94.00
Последняя accuracy для aggregation_type=max на train: 94.00

Лучшая accuracy для aggregation_type=mean на train: 95.43
Последняя accuracy для aggregation_type=mean на train: 95.43

Лучшая accuracy для aggregation_type=max на eval: 90.20
Последняя accuracy для aggregation_type=max на eval: 90.08

Лучшая accuracy для aggregation_type=mean на eval: 91.02
Последняя accuracy для aggregation_type=mean на eval: 90.68
```



"оранжевая" пара показала себя лучше "синей", т.е. за baseline возьмем модель `aggregation_type=mean`, `epochs=5`. **Baseline accuracy=90.68**

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.

Эксперимент N°1

Замечание: все последующие эксперименты с гиперпараметрами я буду проводить с RNN (а не GRU, LSTM, которые наверняка дадут более мощный буст и так), так как мне интересно, получится ли выбить какое-то качество на этой задаче именно с RNN.

(Соответсвует пункту 3 в предложенных экспериментах выше)

Изменение архитектуры после применения RNN. В базовой модели используется агрегация со всех эмбеддингов. Возможно, вы захотите конкатенировать результат агрегации и эмбеддинг с последнего токена.

```
class UpgradedLM(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_classes: int = 4,
hidden_rnn_layers: int = 1,
        aggregation_type: str = 'max', dropout_prob: float = 0.1,
concat_aggregation: bool = False,
        ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.RNN(hidden_dim, hidden_dim, batch_first=True,
num_layers=hidden_rnn_layers)
        # размер слоев увеличится вдвое, если ъотим конкатериновать
опцией concat_aggregation, и не изменится если не будем
конкатенировать
        self.linear =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation),
hidden_dim+hidden_dim*int(concat_aggregation))
        self.projection =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation), num_classes)

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

```
        self.aggregation_type = aggregation_type
        self.concat_aggregation = concat_aggregation

    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' and self.concat_aggregation
is False:
            output = output.max(dim=1)[0] #[batch_size, hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is False:
            output = output.mean(dim=1) #[batch_size, hidden_dim]
        elif self.aggregation_type == 'max' and
self.concat_aggregation is True:
            output = torch.cat((output.max(dim=1)[0], output[:, -
1, :]), dim=1) #[batch_size, 2*hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is True:
            output = torch.cat((output.mean(dim=1), output[:, -1, :]),
dim=1) #[batch_size, 2*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
```

Запустим эксперимент с конкатенацией. Здесь я также попробую оба занчения для `aggregation_type`:

```
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={aggregation_type}
and concat_aggregation=True")
    losses = {
        'eval': [],
        'train': [],
```

```
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = UpgradedLM(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type=aggregation_type, concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for aggregation_type=max and concat_aggregation=True
```

{"model_id":"96b84d33eea041938a57061726b18856","version_major":2,"version_minor":0}

{"model_id":"99c977dca18047de847835c17410dbc5","version_major":2,"version_minor":0}

{"model_id":"f86e325a09e849429de9784392d9b95f","version_major":2,"version_minor":0}

{"model_id":"ea8d4af4670a4563b5dcbd7067929a83","version_major":2,"version_minor":0}

{"model_id":"a15f306be6cd460e800f71ee5dc69c26","version_major":2,"version_minor":0}

```
Starting training for aggregation_type=mean and
concat_aggregation=True
```

{"model_id":"2e9149feb5514e8aab0b71241db297a3","version_major":2,"version_minor":0}

{"model_id":"77e74c00574a46b588808572151bfadf","version_major":2,"version_minor":0}

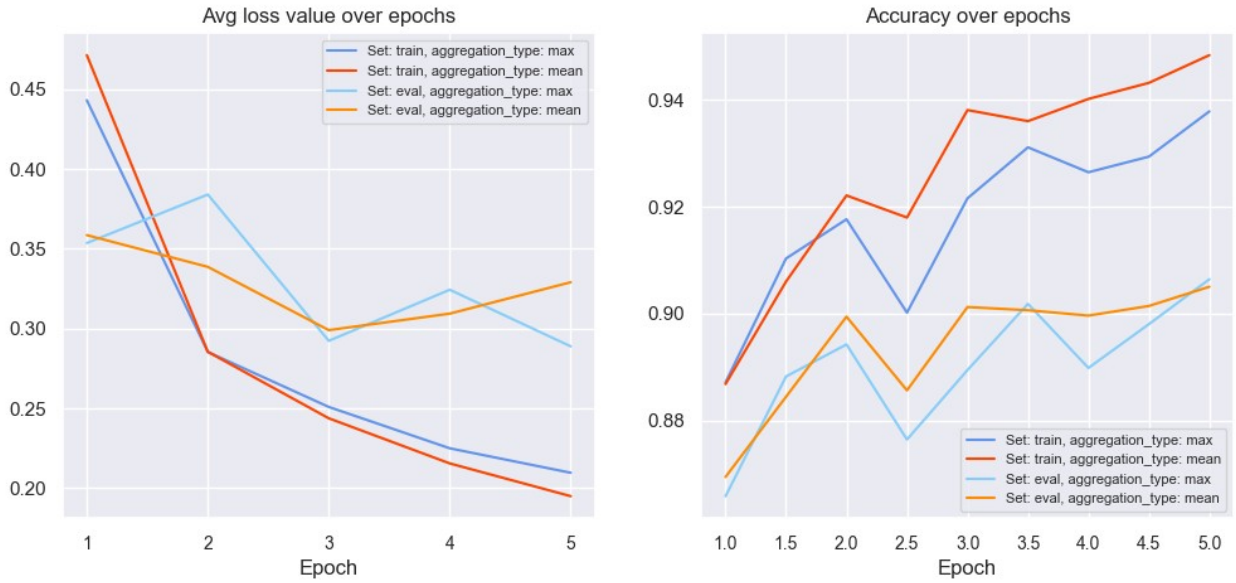
{"model_id":"25c2fab2434a48bdb7c25ac3f2fcd811","version_major":2,"version_minor":0}

{"model_id":"02a702c9f9c547b68102cf4563c2b877","version_major":2,"version_minor":0}

{"model_id":"1de141c4004546d089e24e5e73686106","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=['max',
'mean'], param_name='aggregation_type')
```

```
Лучшая accuracy для aggregation_type=max на train: 93.78
Последняя accuracy для aggregation_type=max на train: 93.78

Лучшая accuracy для aggregation_type=mean на train: 94.83
Последняя accuracy для aggregation_type=mean на train: 94.83

Лучшая accuracy для aggregation_type=max на eval: 90.64
Последняя accuracy для aggregation_type=max на eval: 90.64

Лучшая accuracy для aggregation_type=mean на eval: 90.50
Последняя accuracy для aggregation_type=mean на eval: 90.50
```



Опять же, аггрегация средним перформит получше (чуть меньше лосс, чуть "повыше" кривая качества). Тем не менее, в сранении с бейзлайном прироста особо нет: 90.68 -> 90.50 (mean) или 90.64 (max).

Эксперимент N°2

(Соответсвует пункту 2 в предложенных экспериментах выше)

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.

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


losses_type = {}
acc_type = {}

for num_layers in [1, 2, 3, 4]:
    print(f"Starting training for num_layers={num_layers}")
    losses = {
        'eval': [],
        'train': [],
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = UpgradedLM(
        hidden_dim=256, vocab_size=len(vocab),
```

```
aggregation_type='mean', hidden_rnn_layers=num_layers,
concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for num_layers=1
```


```
{"model_id":"5549cd1ffb494ad581cf2327679c061d","version_major":2,"vers
ion_minor":0}

{"model_id":"bcfe810bbea4408f876f2e1954fa50b5","version_major":2,"vers
ion_minor":0}

{"model_id":"4e4734939635406d95ef4da3c7e5d1e9","version_major":2,"vers
ion_minor":0}

{"model_id":"c4cb5f1039ab4374b8b6b4747f4fa316","version_major":2,"vers
ion_minor":0}
```

{"model_id":"7f1e848c3e1243f4925784c1d6592839","version_major":2,"version_minor":0}

```
Starting training for num_layers=2
```

{"model_id":"3a99268dc82f456a91ac44d965b6f27e","version_major":2,"version_minor":0}

{"model_id":"044dfe40bdc54e969209dfaa9b25a486","version_major":2,"version_minor":0}

{"model_id":"a7ca0b93d05245adb8873a3fd9d4ead0","version_major":2,"version_minor":0}

{"model_id":"379b9fbae06c40739b63bc0061ace8d9","version_major":2,"version_minor":0}

{"model_id":"a6259c98f0ac40beb7d928e5ed6f698c","version_major":2,"version_minor":0}

```
Starting training for num_layers=3
```

{"model_id":"042c0786fd8643a5b47a16dec9fb39bb","version_major":2,"version_minor":0}

{"model_id":"e4618865f307417394190ad7fde3b9f8","version_major":2,"version_minor":0}

{"model_id":"9d73fd69e1f64cdca1f51828f3c88caa","version_major":2,"version_minor":0}

{"model_id":"3be161b028214be288a43468c48bad80","version_major":2,"version_minor":0}

{"model_id":"8fdc6980c6bf4d38ae047e14d1747aab","version_major":2,"version_minor":0}

```
Starting training for num_layers=4
```

{"model_id":"5655d76ce9a94ca6a394907c98adbffa","version_major":2,"version_minor":0}

{"model_id":"5207958f3c054d0ba2c3fa6d3b1bbda7","version_major":2,"version_minor":0}

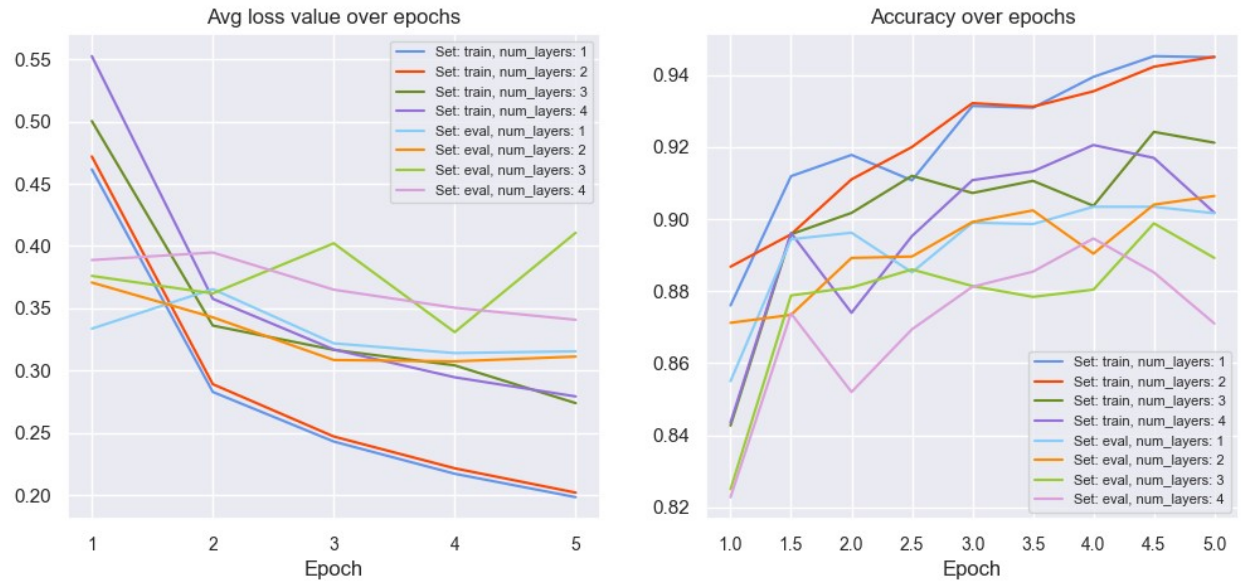
{"model_id":"6937c7dc0d1b4c95b9238450b3ece09e","version_major":2,"version_minor":0}

{"model_id":"97395dae0caf4cd59d9926f76f851e63","version_major":2,"version_minor":0}

{"model_id":"58a14af9b1c942888d2e4ef779f061a0","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=[1,2,3,4],
param_name='num_layers')
```

```
Лучшая accuracy для num_layers=1 на train: 94.52
Последняя accuracy для num_layers=1 на train: 94.49

Лучшая accuracy для num_layers=2 на train: 94.51
Последняя accuracy для num_layers=2 на train: 94.51

Лучшая accuracy для num_layers=3 на train: 92.42
Последняя accuracy для num_layers=3 на train: 92.12

Лучшая accuracy для num_layers=4 на train: 92.05
Последняя accuracy для num_layers=4 на train: 90.17

Лучшая accuracy для num_layers=1 на eval: 90.34
Последняя accuracy для num_layers=1 на eval: 90.16

Лучшая accuracy для num_layers=2 на eval: 90.64
Последняя accuracy для num_layers=2 на eval: 90.64

Лучшая accuracy для num_layers=3 на eval: 89.88
Последняя accuracy для num_layers=3 на eval: 88.92

Лучшая accuracy для num_layers=4 на eval: 89.46
Последняя accuracy для num_layers=4 на eval: 87.10
```



В этом эксперименте лучше всего себя проявила модель с num_layers=2, однако качество все еще не перевалило за 0.91. Также я пробовала значения параметра num_layers 5 и 10, получалось только хуже.

Эксперимент №3

(Соответсвует пункту 4 в предложенных экспериментах выше)

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.

В этом эксперименте меняю `dropout_rate`.

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


losses_type = {}
acc_type = {}

for dropout_prob in [0.001, 0.01, 0.1, 0.25, 0.5]:
    print(f"Starting training for dropout_prob={dropout_prob}")
    losses = {
        'eval': [],
        'train': [],
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = UpgradedLM(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type='mean', hidden_rnn_layers=2,
dropout_prob=dropout_prob, concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
```

```
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for dropout_prob=0.001
```


```
{"model_id":"2d5383c53ee0461fbbf488ea492c2bc7","version_major":2,"vers
ion_minor":0}

{"model_id":"cea7892dd51042b9a63b599f14f4539c","version_major":2,"vers
ion_minor":0}

{"model_id":"e8bd62caed6f4cd08cb554869d606eec","version_major":2,"vers
ion_minor":0}

{"model_id":"5147728a55ed4da49f1b2ef9c6158525","version_major":2,"vers
ion_minor":0}

{"model_id":"518cc64981424ecc92d17a2a73eec22f","version_major":2,"vers
ion_minor":0}
```


```
Starting training for dropout_prob=0.01
```


```
{"model_id":"ff3df36469a74f3bbf5b3e8a3f92ef1c","version_major":2,"vers
ion_minor":0}

{"model_id":"36ae2b2b812d4b74b215231c3d0b3231","version_major":2,"vers
ion_minor":0}

{"model_id":"b69cbbec37b640c580304b8f9693b593","version_major":2,"vers
ion_minor":0}

{"model_id":"8bd1c7df6a6f4643a8ec2637be7b4f26","version_major":2,"vers
ion_minor":0}

{"model_id":"c7adcac8aed94436b6b504833db3151e","version_major":2,"vers
ion_minor":0}
```


```
Starting training for dropout_prob=0.1
```

{"model_id":"6b905cd351004a58b934abb3ca36afe3","version_major":2,"version_minor":0}

{"model_id":"0a94b0701a654166a3d726215e72a3a7","version_major":2,"version_minor":0}

{"model_id":"d716db222a904178ae4b150e123bab52","version_major":2,"version_minor":0}

{"model_id":"710a5d54db3f4db3b3c8f1d042684de8","version_major":2,"version_minor":0}

{"model_id":"faad5ab22b3c4c66a991f9ef7a14e69b","version_major":2,"version_minor":0}

```
Starting training for dropout_prob=0.25
```

{"model_id":"4c5139ba2f8040c8a3fa14ecd7d3cd51","version_major":2,"version_minor":0}

{"model_id":"e6fa05de4fec4972ba9b5c6ae1fb751a","version_major":2,"version_minor":0}

{"model_id":"d8c55392eef84e8497956d53d0aec6f7","version_major":2,"version_minor":0}

{"model_id":"2567b60827e94857957b8923a3b21175","version_major":2,"version_minor":0}

{"model_id":"877c74b56db54d2996c8251a697dfc97","version_major":2,"version_minor":0}

```
Starting training for dropout_prob=0.5
```

{"model_id":"d21bb81367f94b108f71811beae9fd66","version_major":2,"version_minor":0}

{"model_id":"692ce91d30d04d89b74f1496ae166bb2","version_major":2,"version_minor":0}

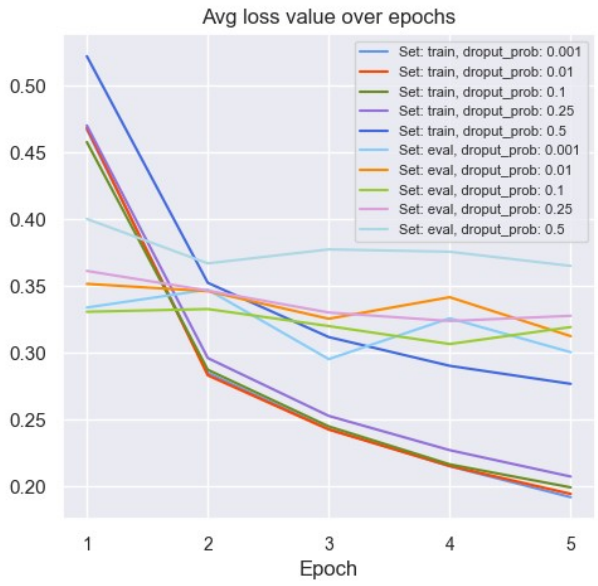
{"model_id":"66be14b4379a40109da8a94119dfa498","version_major":2,"version_minor":0}

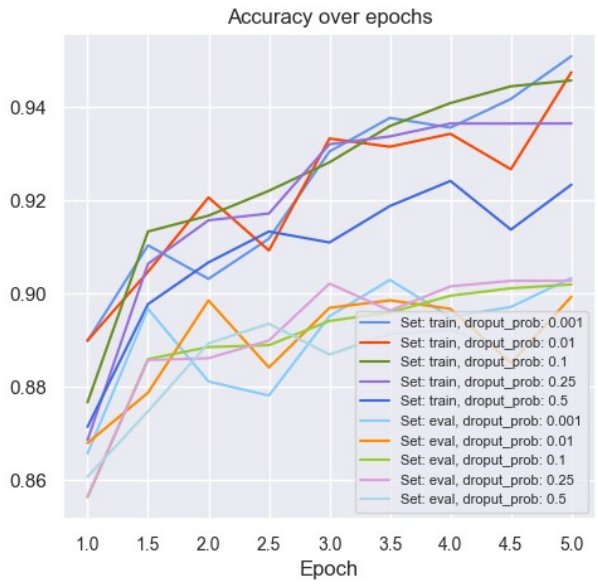
{"model_id":"86e698f3c88c4b11bf101c3e0ee7923f","version_major":2,"version_minor":0}

{"model_id":"38920d7e2b444a1da9595c84d4077594","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=[0.001, 0.01, 0.1, 0.25, 0.5], param_name='droput_prob')
```

```
Лучшая accuracy для droput_prob=0.001 на train: 95.10
Последняя accuracy для droput_prob=0.001 на train: 95.10

Лучшая accuracy для droput_prob=0.01 на train: 94.75
Последняя accuracy для droput_prob=0.01 на train: 94.75

Лучшая accuracy для droput_prob=0.1 на train: 94.58
Последняя accuracy для droput_prob=0.1 на train: 94.58

Лучшая accuracy для droput_prob=0.25 на train: 93.65
Последняя accuracy для droput_prob=0.25 на train: 93.65

Лучшая accuracy для droput_prob=0.5 на train: 92.42
Последняя accuracy для droput_prob=0.5 на train: 92.34

Лучшая accuracy для droput_prob=0.001 на eval: 90.34
Последняя accuracy для droput_prob=0.001 на eval: 90.34

Лучшая accuracy для droput_prob=0.01 на eval: 89.94
Последняя accuracy для droput_prob=0.01 на eval: 89.94

Лучшая accuracy для droput_prob=0.1 на eval: 90.20
Последняя accuracy для droput_prob=0.1 на eval: 90.20

Лучшая accuracy для droput_prob=0.25 на eval: 90.28
Последняя accuracy для droput_prob=0.25 на eval: 90.28

Лучшая accuracy для droput_prob=0.5 на eval: 89.36
Последняя accuracy для droput_prob=0.5 на eval: 89.08
```





Выбранное изначально значение 0.1 и так достаточно хорошее. Можно было бы выбрать 0.25, но разница в качестве не существенная. Оставляю 0.1.

Теперь меняем размер `hidden_dim`:

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


losses_type = {}
acc_type = {}

for hidden_dim in [32, 128, 256, 512]:
    print(f"Starting training for hidden_dim={hidden_dim}")
    losses = {
        'eval': [],
        'train': [],
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = UpgradedLM(
        hidden_dim=hidden_dim, vocab_size=len(vocab),
aggregation_type='mean', hidden_rnn_layers=2, dropout_prob=0.1,
concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
```

```
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for hidden_dim=32
```


```
{"model_id":"b3769a2126604873ba8f7c433930c5b8","version_major":2,"vers
ion_minor":0}

{"model_id":"759fe66d050d4bb78400b667c6f6475d","version_major":2,"vers
ion_minor":0}

{"model_id":"8d425da9d6e64bdbb06b5d61b2e852d3","version_major":2,"vers
ion_minor":0}

{"model_id":"f2481663538e4714ac1c40911cff18b2","version_major":2,"vers
ion_minor":0}

{"model_id":"d94143f0794544429053b0339121db67","version_major":2,"vers
ion_minor":0}
```


```
Starting training for hidden_dim=128
```


```
{"model_id":"681907afd78d451f9d8c7224ec3fe0d4","version_major":2,"vers
ion_minor":0}

{"model_id":"e2647b20b04b4649bcb1a8e7eac2fed3","version_major":2,"vers
ion_minor":0}

{"model_id":"073c130f0fab4fa3a9419832fe920556","version_major":2,"vers
ion_minor":0}

{"model_id":"3043ee5160f241ebbbe0cc565b42c45b","version_major":2,"vers
ion_minor":0}

{"model_id":"e18bb65072b840b5ae5aa1869c372960","version_major":2,"vers
ion_minor":0}
```


```
Starting training for hidden_dim=256
```


```
{"model_id":"325ed22459df4759b2ac63c84747a6dd","version_major":2,"vers
ion_minor":0}

{"model_id":"cd1464e9aabe4aa6a0dbdcef83ed77fb","version_major":2,"vers
ion_minor":0}
```

{"model_id":"94becbb51ef041d08eae9f869b790f7f","version_major":2,"version_minor":0}

{"model_id":"a5eaf87cf88e4ee18de7c54462976a1b","version_major":2,"version_minor":0}

{"model_id":"7b5a8b28ce4e46878ea540acd4c1f6ac","version_major":2,"version_minor":0}

```
Starting training for hidden_dim=512
```

{"model_id":"5d6c81d5640d4f0488571c792756d5e8","version_major":2,"version_minor":0}

{"model_id":"2e35ab20ac0b49a9bb62b7c3bc06c8f5","version_major":2,"version_minor":0}

{"model_id":"53a489b0114349e48c4c01948f1ffba0","version_major":2,"version_minor":0}

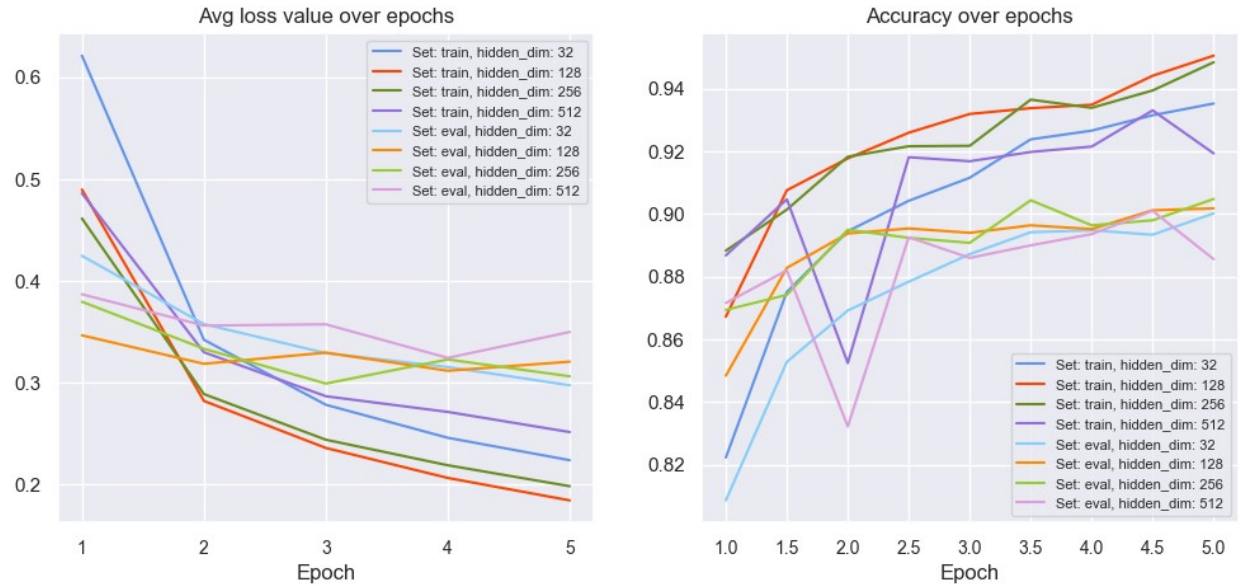
{"model_id":"b8414219f8db4b3da99568f22ff54a66","version_major":2,"version_minor":0}

{"model_id":"4deff43affed4f1f851bb1e9be8277c2","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=[32, 128, 256, 512], param_name='hidden_dim')
```

```
Лучшая accuracy для hidden_dim=32 на train: 93.53
Последняя accuracy для hidden_dim=32 на train: 93.53

Лучшая accuracy для hidden_dim=128 на train: 95.06
Последняя accuracy для hidden_dim=128 на train: 95.06

Лучшая accuracy для hidden_dim=256 на train: 94.84
Последняя accuracy для hidden_dim=256 на train: 94.84

Лучшая accuracy для hidden_dim=512 на train: 93.31
Последняя accuracy для hidden_dim=512 на train: 91.94

Лучшая accuracy для hidden_dim=32 на eval: 90.02
Последняя accuracy для hidden_dim=32 на eval: 90.02

Лучшая accuracy для hidden_dim=128 на eval: 90.18
Последняя accuracy для hidden_dim=128 на eval: 90.18

Лучшая accuracy для hidden_dim=256 на eval: 90.48
Последняя accuracy для hidden_dim=256 на eval: 90.48

Лучшая accuracy для hidden_dim=512 на eval: 90.10
```

```
Последняя accuracy для hidden_dim=512 на eval: 88.56
```



Ситуация аналогичная: значение 256 для RNN уже достаточно хорошее. Для облегчения модели можно выбрать 128 при незначительных потерях в качестве, но т.к. в задании есть определенные целевые отсечки по качеству, я оставлю 256.

Поэкспериментируем с размером словаря `counter_threshold` (отличие этого эксперимента в том что нам нужно переопределять соответствующие словари с индексами и датасет):

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


losses_type = {}
acc_type = {}

for counter_threshold in [5, 10, 25, 50, 100, 1000]:
    print(f"Starting training for
counter_threshold={counter_threshold}")

    vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
    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()}
```

```
    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=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

    eval_dataloader = DataLoader(
        eval_dataset, shuffle=False, collate_fn=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

    losses = {
        'eval': [],
        'train': [],
    }
    acc = {
        'eval': [],
        'train': [],
    }

    model = UpgradedLM(
        hidden_dim=256, vocab_size=len(vocab),
aggregation_type='mean', hidden_rnn_layers=2, dropout_prob=0.1,
concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
                acc['train'].append(evaluate(model, train_dataloader))
```

```
                model.train()

        losses['train'].append(sum(epoch_losses) / len(epoch_losses))

        # compute loss for eval dataset at the end of each epoch
        epoch_losses = []
        with torch.no_grad():
            for batch in eval_dataloader:
                logits = model(batch['input_ids'])
                loss = criterion(logits, batch['label'])
                epoch_losses.append(loss.item())

        losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for counter_threshold=5
Размер словаря: 27694
```


```
{"model_id":"bf8918c92ec14c0d92c579c3d064cf38","version_major":2,"vers
ion_minor":0}

{"model_id":"308bbbfa93ff42f88c816b33d25ecf77","version_major":2,"vers
ion_minor":0}

{"model_id":"047da80ea69f4550825e9aa150db59c3","version_major":2,"vers
ion_minor":0}

{"model_id":"f3d444ebaf284ef09e03b561a0b49553","version_major":2,"vers
ion_minor":0}

{"model_id":"65170768133c43c68767ea9d48c95a3c","version_major":2,"vers
ion_minor":0}
```


```
Starting training for counter_threshold=10
Размер словаря: 19562
```


```
{"model_id":"a5b5a27305a64a1dabdf77f538475f90","version_major":2,"vers
ion_minor":0}

{"model_id":"9b45d20500e146bbb610025dd0a94e0a","version_major":2,"vers
ion_minor":0}

{"model_id":"bfb3843c4758432ea7c3610e1722922c","version_major":2,"vers
ion_minor":0}

{"model_id":"8e8f0709b5f14b7ab217b755c50b5192","version_major":2,"vers
ion_minor":0}

{"model_id":"4ab309a85f2c4381aa6535d11a9fdb1e","version_major":2,"vers
ion_minor":0}
```

```
Starting training for counter_threshold=25
Размер словаря: 11842
```


```
{"model_id":"13ee76f2e0ac46e08f20040b5d68cb2d","version_major":2,"vers
ion_minor":0}

{"model_id":"661e1264e4f44e66944b76e0e51bf1fb","version_major":2,"vers
ion_minor":0}

{"model_id":"a1cf701590974d678abfdcae7d840fd6","version_major":2,"vers
ion_minor":0}

{"model_id":"eee66fd7fc604084b455427a695cea85","version_major":2,"vers
ion_minor":0}

{"model_id":"fc324ffadff841668bc573af76925c89","version_major":2,"vers
ion_minor":0}
```


```
Starting training for counter_threshold=50
Размер словаря: 7787
```


```
{"model_id":"073eecbead0f4107ae8f75f32c8f2239","version_major":2,"vers
ion_minor":0}

{"model_id":"c3fb57c8a2f24ababfb2a66a57332758","version_major":2,"vers
ion_minor":0}

{"model_id":"a1ce38436b85481a90c4dd0265b08b3b","version_major":2,"vers
ion_minor":0}

{"model_id":"b362ad27395f4ca18ec8f03510721058","version_major":2,"vers
ion_minor":0}

{"model_id":"8fd0894c02b149efa7d2f5a0a92f9f3e","version_major":2,"vers
ion_minor":0}
```


```
Starting training for counter_threshold=100
Размер словаря: 4954
```

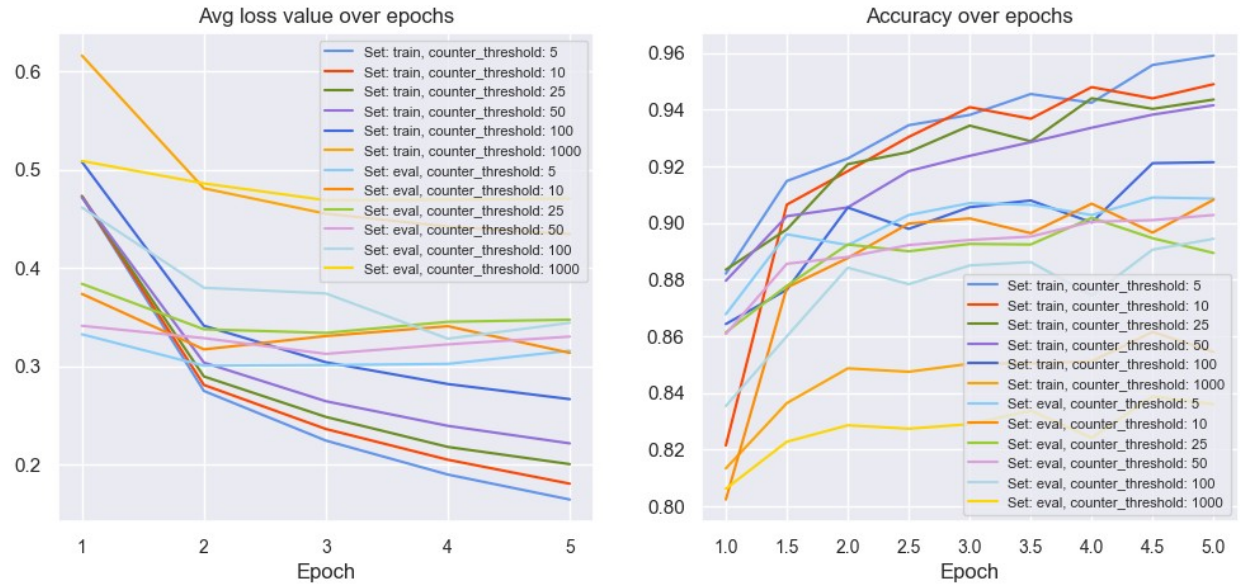

```
{"model_id":"b5726a2ca254499bbfa92e7b26f25b61","version_major":2,"vers
ion_minor":0}

{"model_id":"76a8f90f7f684456a6ab4e786fe71712","version_major":2,"vers
ion_minor":0}

{"model_id":"3f084795d1704d378f222a85de719ecd","version_major":2,"vers
ion_minor":0}

{"model_id":"8b197da468a647a6a93902068ff85e35","version_major":2,"vers
ion_minor":0}

{"model_id":"c219dbf5167c44e2bde663ad03c6e47b","version_major":2,"vers
ion_minor":0}
```

```
Starting training for counter_threshold=1000
Размер словаря: 605

{"model_id":"32acc902b7a14623823d6e1e98b865c3","version_major":2,"vers
ion_minor":0}

{"model_id":"5b2d966b222c4a20b0c2520da3b78492","version_major":2,"vers
ion_minor":0}

{"model_id":"99f8165bd58e43d38c78f889870873f3","version_major":2,"vers
ion_minor":0}

{"model_id":"fdb384fdc4cd475c815b3c6f7a1d1e48","version_major":2,"vers
ion_minor":0}

{"model_id":"cde2d50c74a1476aa3feac6a9ea742ab","version_major":2,"vers
ion_minor":0}
```

```
plot_learning_curves(losses_type, acc_type, param_sets=[5, 10, 25, 50,
100, 1000], param_name='counter_threshold')
```

```
Лучшая accuracy для counter_threshold=5 на train: 95.91
Последняя accuracy для counter_threshold=5 на train: 95.91

Лучшая accuracy для counter_threshold=10 на train: 94.90
Последняя accuracy для counter_threshold=10 на train: 94.90

Лучшая accuracy для counter_threshold=25 на train: 94.40
Последняя accuracy для counter_threshold=25 на train: 94.35

Лучшая accuracy для counter_threshold=50 на train: 94.16
Последняя accuracy для counter_threshold=50 на train: 94.16

Лучшая accuracy для counter_threshold=100 на train: 92.14
Последняя accuracy для counter_threshold=100 на train: 92.14

Лучшая accuracy для counter_threshold=1000 на train: 86.15
Последняя accuracy для counter_threshold=1000 на train: 85.45

Лучшая accuracy для counter_threshold=5 на eval: 90.90
Последняя accuracy для counter_threshold=5 на eval: 90.86

Лучшая accuracy для counter_threshold=10 на eval: 90.82
Последняя accuracy для counter_threshold=10 на eval: 90.82

Лучшая accuracy для counter_threshold=25 на eval: 90.18
Последняя accuracy для counter_threshold=25 на eval: 88.94

Лучшая accuracy для counter_threshold=50 на eval: 90.28
Последняя accuracy для counter_threshold=50 на eval: 90.28

Лучшая accuracy для counter_threshold=100 на eval: 89.44
```

```
Последняя accuracy для counter_threshold=100 на eval: 89.44

Лучшая accuracy для counter_threshold=1000 на eval: 83.88
Последняя accuracy для counter_threshold=1000 на eval: 83.60
```



Тенденция такова, что чем больше словарь (меньше отсечка), тем лучше каечство. Я в качестве оптимальной конфигурации выбираю `counter_threshold=10` (чем меньше значение, тем дольше обучение, это минус большого словаря).

Финально, в этом эксперименте посмотрим на какой эпохе стоит остановить обучение:

```
num_epoch = 20
eval_steps = len(train_dataloader) // 2


losses_type = {}
acc_type = {}

print(f"Starting training for num_epoch={num_epoch}")

counter_threshold = 10
vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
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()}
```

```
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=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)


losses = {
    'eval': [],
    'train': [],
}
acc = {
    'eval': [],
    'train': [],
}

model = UpgradedLM(
    hidden_dim=256, vocab_size=len(vocab), aggregation_type='mean',
hidden_rnn_layers=2, dropout_prob=0.1,
concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
            acc['train'].append(evaluate(model, train_dataloader))
```

```
            model.train()

    losses['train'].append(sum(epoch_losses) / len(epoch_losses))

    # compute loss for eval dataset at the end of each epoch
    epoch_losses = []
    with torch.no_grad():
        for batch in eval_dataloader:
            logits = model(batch['input_ids'])
            loss = criterion(logits, batch['label'])
            epoch_losses.append(loss.item())

    losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for num_epoch=20
Размер словаря: 19562
```


```
{"model_id":"da9e7a6f55e54d55b763ffff44dbdb37","version_major":2,"vers
ion_minor":0}

{"model_id":"e7b0226a7c3b4b6d93d493d2defe0fd5","version_major":2,"vers
ion_minor":0}

{"model_id":"cdf0c7be629d4bd1b8ed0e09d807f3da","version_major":2,"vers
ion_minor":0}

{"model_id":"1e4f9afacfd049139946a9323594ca5b","version_major":2,"vers
ion_minor":0}

{"model_id":"79157ca9d16344eab0ed659ea1f0a8d8","version_major":2,"vers
ion_minor":0}

{"model_id":"95f2fec453a241428cdbba3f28bee777","version_major":2,"vers
ion_minor":0}

{"model_id":"e82433e609e8480f96f5bc5c792a48e3","version_major":2,"vers
ion_minor":0}

{"model_id":"4110eb7ffb1448e1bea4d23223876fa6","version_major":2,"vers
ion_minor":0}

{"model_id":"ff921c94cc6749cbb3786549c72ecade","version_major":2,"vers
ion_minor":0}

{"model_id":"cea937bdfc7c4b75a8f3b84a1e8bb766","version_major":2,"vers
ion_minor":0}

{"model_id":"110277d5641d4d2baf9525173590be7c","version_major":2,"vers
ion_minor":0}
```

{"model_id":"37f3fef25c774ca5811e916d58d60fbc","version_major":2,"version_minor":0}

{"model_id":"763ad5d0f93f40e18c0e6fd68fd7fef8","version_major":2,"version_minor":0}

{"model_id":"c6bc2e49dd0b475d9eae730646ccd05b","version_major":2,"version_minor":0}

{"model_id":"a9f5fb1921084b00ad62fe86328be201","version_major":2,"version_minor":0}

{"model_id":"c27c61f60c514e699ce2c85fd504e342","version_major":2,"version_minor":0}

{"model_id":"ccac48948b164a149ea22cb51ade0104","version_major":2,"version_minor":0}

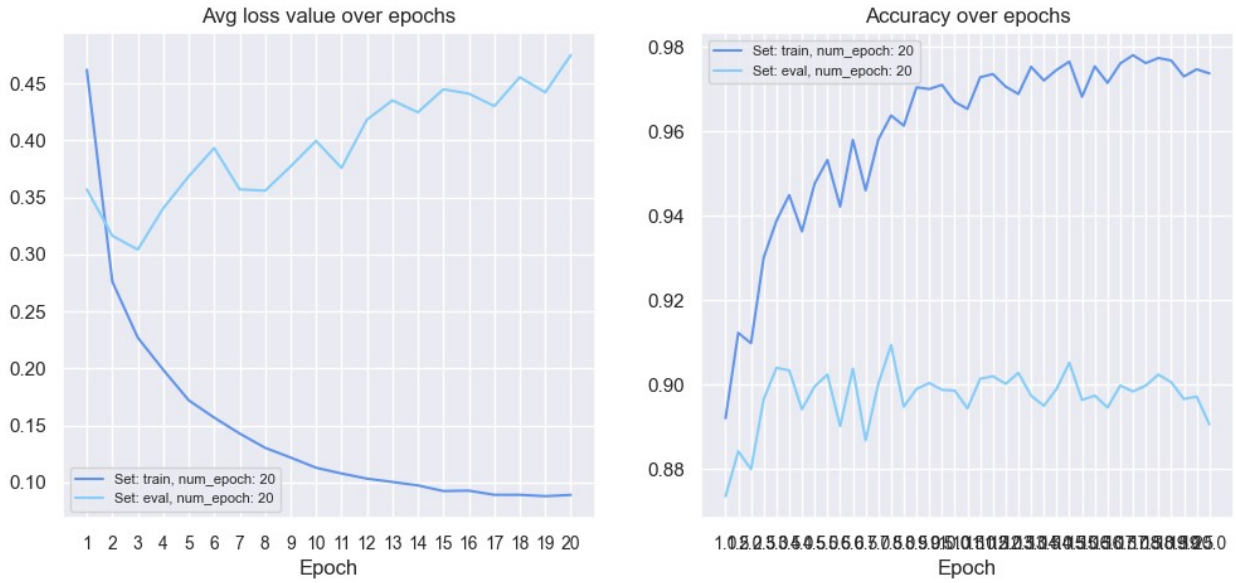
{"model_id":"199cfd6bce5342e5a65b62c29149688d","version_major":2,"version_minor":0}

{"model_id":"ec4e6bd30f3c48eaa0cfb3328db8ba43","version_major":2,"version_minor":0}

{"model_id":"3cb0fe29cf3849ac9cb65e5d5918a522","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=[20], param_name='num_epoch')
```

```
Лучшая accuracy для num_epoch=20 на train: 97.80
Последняя accuracy для num_epoch=20 на train: 97.37

Лучшая accuracy для num_epoch=20 на eval: 90.94
Последняя accuracy для num_epoch=20 на eval: 89.06
```



Больше 7-8 эпох делать для такой конфигурации RNN смысла нет (качество не растет, но растет лосс и потенциально переобучение + последующие эксперименты дольше по времени). Я выбираю 7 эпох. Замеряем финальное качество:

```
num_epoch = 7
eval_steps = len(train_dataloader) // 2


losses_type = {}
acc_type = {}

print(f"Starting training for num_epoch={num_epoch}")

counter_threshold = 10
vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
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()}

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=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)


losses = {
    'eval': [],
    'train': [],
}
acc = {
    'eval': [],
    'train': [],
}

model = UpgradedLM(
    hidden_dim=256, vocab_size=len(vocab), aggregation_type='mean',
hidden_rnn_layers=2, dropout_prob=0.1,
concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
            acc['train'].append(evaluate(model, train_dataloader))
            model.train()

    losses['train'].append(sum(epoch_losses) / len(epoch_losses))

    # compute loss for eval dataset at the end of each epoch
    epoch_losses = []
    with torch.no_grad():
        for batch in eval_dataloader:
```

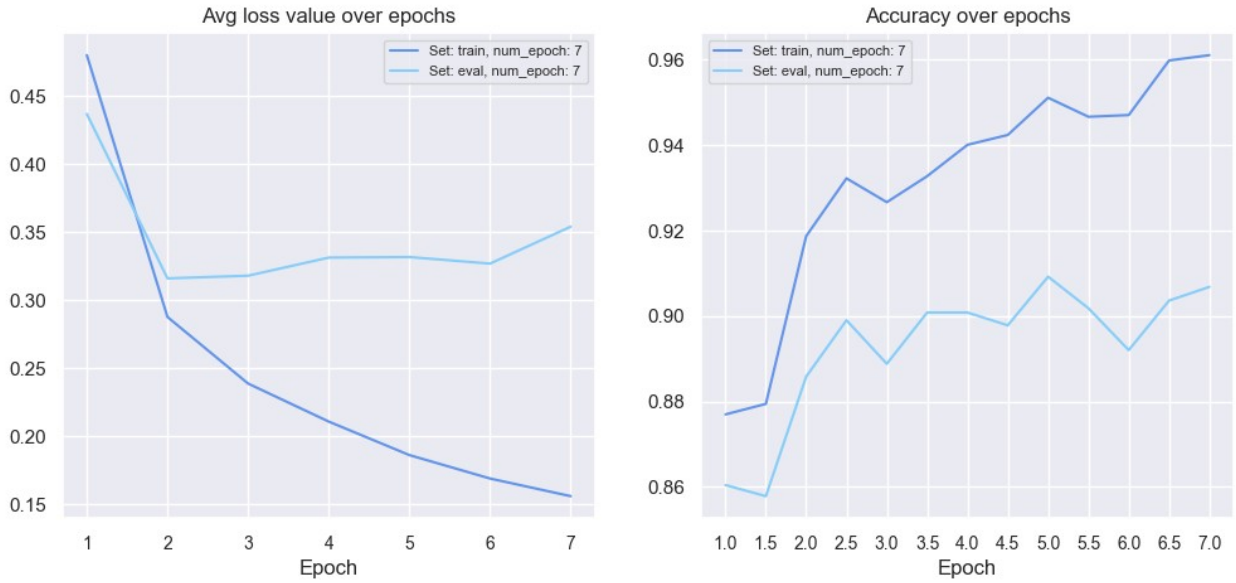
```
            logits = model(batch['input_ids'])
            loss = criterion(logits, batch['label'])
            epoch_losses.append(loss.item())

    losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

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

```
Starting training for num_epoch=7
Размер словаря: 19562
```


```
{"model_id":"a0cf1311062543b39cd9aa7bdd3e2134","version_major":2,"version_minor":0}

{"model_id":"f2f5aad325894e86b4382988fb0f0bd2","version_major":2,"version_minor":0}

{"model_id":"82c4959a109043b593e4b8ac428d8e71","version_major":2,"version_minor":0}

{"model_id":"b02dad8936f1499aa4beb2a7ea6838fc","version_major":2,"version_minor":0}

{"model_id":"60e61d5c537c467e8437b650216270df","version_major":2,"version_minor":0}

{"model_id":"5d7f8b74b885475995c69f8177e17535","version_major":2,"version_minor":0}

{"model_id":"5bb34996e348485f88ac3eb1192d74c5","version_major":2,"version_minor":0}
```


```
plot_learning_curves(losses_type, acc_type, param_sets=[7],
param_name='num_epoch')
```

```
Лучшая accuracy для num_epoch=7 на train: 96.11
Последняя accuracy для num_epoch=7 на train: 96.11

Лучшая accuracy для num_epoch=7 на eval: 90.92
Последняя accuracy для num_epoch=7 на eval: 90.68
```



Итого финальная цифра - **90.68**, такая же как бейзлайн □□□.

Замечу что в процессе эксперимента с размером словаря я делала шафл данных (семпл eval датасета разный на бейзлайне и тут), так что строго говоря не совсем сравнимые цифры, но все равно можно сделать общий вывод (локально в экспериментах я то выбирала наилучшие конфигурации): подбор гиперпараметров на RNN особо прироста не дал и в категорию accuracy>=0.91 модель не попадает.

Эксперимент N°4

(Соответсвует пункту 1 в предложенных экспериментах выше)

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.

GRU (отличается только подход, гиперпараметры остались те же):

```
class UpgradedLMGRU(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_classes: int = 4,
hidden_rnn_layers: int = 1,
        aggregation_type: str = 'max', dropout_prob: float = 0.1,
concat_aggregation: bool = False,
        ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.GRU(hidden_dim, hidden_dim, batch_first=True,
num_layers=hidden_rnn_layers)
        self.linear =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation),
hidden_dim+hidden_dim*int(concat_aggregation))
```

```
        self.projection =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation), num_classes)

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

        self.aggregation_type = aggregation_type
        self.concat_aggregation = concat_aggregation

    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' and self.concat_aggregation
is False:
            output = output.max(dim=1)[0] #[batch_size, hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is False:
            output = output.mean(dim=1) #[batch_size, hidden_dim]
        elif self.aggregation_type == 'max' and
self.concat_aggregation is True:
            output = torch.cat((output.max(dim=1)[0], output[:, -
1, :]), dim=1) #[batch_size, 2*hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is True:
            output = torch.cat((output.mean(dim=1), output[:, -1, :]),
dim=1) #[batch_size, 2*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

vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
for char, cnt in words.items():
    if cnt > 10:
        vocab.add(char)

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

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=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

eval_dataloader = DataLoader(
    eval_dataset, shuffle=False, collate_fn=lambda x:
collate_fn_with_padding(x, pad_id=word2ind['<pad>']),
batch_size=batch_size)

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


losses_type = {}
acc_type = {}

print(f"Starting training for GRU")

losses = {
    'eval': [],
    'train': [],
}
acc = {
    'eval': [],
    'train': [],
}

model = UpgradedLMGRU(
    hidden_dim=256, vocab_size=len(vocab), aggregation_type='mean',
hidden_rnn_layers=2, dropout_prob=0.1,
    concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
            acc['train'].append(evaluate(model, train_dataloader))
            model.train()

    losses['train'].append(sum(epoch_losses) / len(epoch_losses))

    # compute loss for eval dataset at the end of each epoch
    epoch_losses = []
    with torch.no_grad():
        for batch in eval_dataloader:
            logits = model(batch['input_ids'])
            loss = criterion(logits, batch['label'])
            epoch_losses.append(loss.item())

    losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

losses_type['GRU'] = losses
acc_type['GRU'] = acc
```

```
Starting training for GRU
```


```
{"model_id":"3f1ff61a70c94d62b5294ad66b74e6d3","version_major":2,"vers
ion_minor":0}

{"model_id":"12a5689517ec4f1aa582c260f7e7c98d","version_major":2,"vers
ion_minor":0}

{"model_id":"ca057c0e82cb4702bdf856306c9c9aa8","version_major":2,"vers
ion_minor":0}

{"model_id":"d803f22b5ddf490cbec20dd10775e9f4","version_major":2,"vers
ion_minor":0}

{"model_id":"dadfc6b20da645168de295a289d0ac79","version_major":2,"vers
ion_minor":0}

{"model_id":"51dbe9f839204557b68ff628c84d9873","version_major":2,"vers
ion_minor":0}

{"model_id":"1849ab9fb3894cd793526fdfa650feee","version_major":2,"vers
ion_minor":0}

{"model_id":"6247dea8186645f1af2b002f35a11bda","version_major":2,"vers
ion_minor":0}

{"model_id":"907eec8a5824406788817dfaaf75155a","version_major":2,"vers
ion_minor":0}
```

{"model_id":"202de5cd9caa4e3e916cce77fca5129c","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=['GRU'], param_name='GRU')
```

```
Лучшая accuracy для GRU=GRU на train: 99.11
Последняя accuracy для GRU=GRU на train: 99.10

Лучшая accuracy для GRU=GRU на eval: 91.60
Последняя accuracy для GRU=GRU на eval: 91.00
```

И, как я и ожидала, GRU сеть дала наиболее заметный прирост как по качеству, так и уменьшение по лоссу. Модели для обучения требуется не очень много эпох (3-4), получили качество **0.91** (последнее) но в максимуме 91.60 (правда на 4.5 эпохе).

Попробуем заменить на `concat_aggregation=False`:

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


losses_type = {}
acc_type = {}

print(f"Starting training for GRU")

losses = {
    'eval': [],
    'train': [],
}
```

```
acc = {
    'eval': [],
    'train': [],
}

model = UpgradedLMGRU(
    hidden_dim=256, vocab_size=len(vocab), aggregation_type='mean',
hidden_rnn_layers=2, dropout_prob=0.1,
    concat_aggregation=False).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['eval'].append(evaluate(model, eval_dataloader))
            acc['train'].append(evaluate(model, train_dataloader))
            model.train()

    losses['train'].append(sum(epoch_losses) / len(epoch_losses))

    # compute loss for eval dataset at the end of each epoch
    epoch_losses = []
    with torch.no_grad():
        for batch in eval_dataloader:
            logits = model(batch['input_ids'])
            loss = criterion(logits, batch['label'])
            epoch_losses.append(loss.item())

    losses['eval'].append(sum(epoch_losses) / len(epoch_losses))

losses_type['GRU'] = losses
acc_type['GRU'] = acc
```

```
Starting training for GRU
```

{"model_id":"802cfae789fe4e72b2bdd571268e4662","version_major":2,"version_minor":0}

{"model_id":"8c6247dae7414866a543239795c0114f","version_major":2,"version_minor":0}

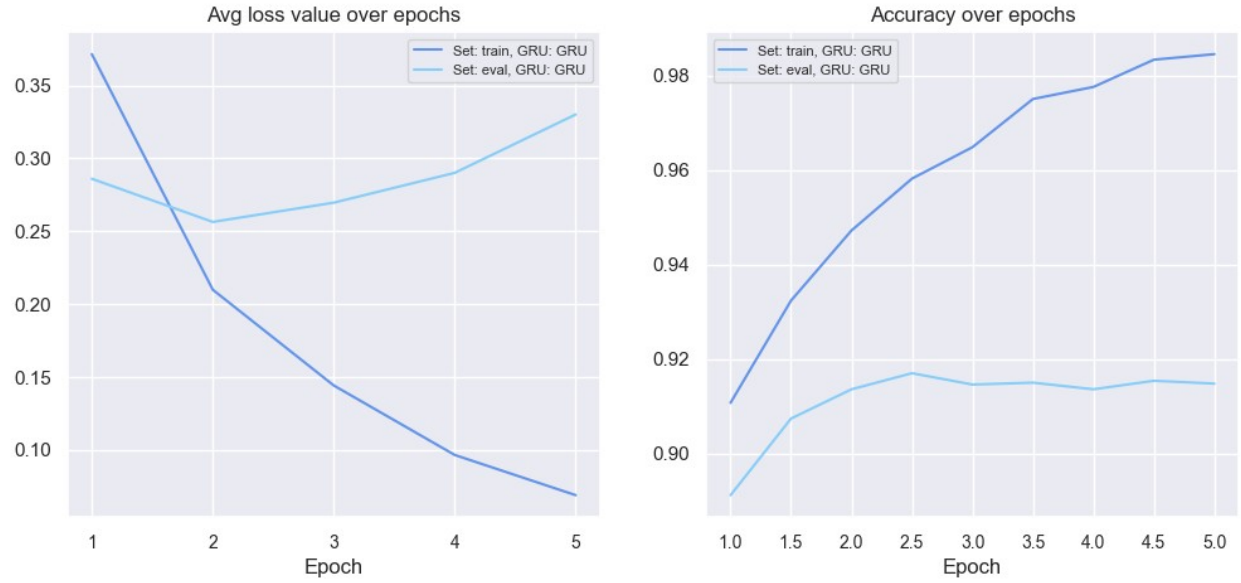
{"model_id":"c36b326e156e4a1faf883e60ab4333d8","version_major":2,"version_minor":0}

{"model_id":"14e0181647b249e3ae94a23b3c59a9ed","version_major":2,"version_minor":0}

{"model_id":"e3b0b31d479e4655bd356cd6015c391b","version_major":2,"version_minor":0}

```
plot_learning_curves(losses_type, acc_type, param_sets=['GRU'],
param_name='GRU')
```

```
Лучшая accuracy для GRU=GRU на train: 98.45
Последняя accuracy для GRU=GRU на train: 98.45

Лучшая accuracy для GRU=GRU на eval: 91.70
Последняя accuracy для GRU=GRU на eval: 91.48
```



Попробуем LSTM:

```
class UpgradedLMLSTM(nn.Module):
    def __init__(
        self, hidden_dim: int, vocab_size: int, num_classes: int = 4,
hidden_rnn_layers: int = 1,
        aggregation_type: str = 'max', dropout_prob: float = 0.1,
concat_aggregation: bool = False
        ):
        super().__init__()
```

```
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        self.rnn = nn.LSTM(hidden_dim, hidden_dim, batch_first=True,
num_layers=hidden_rnn_layers)
        self.linear =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation),
hidden_dim+hidden_dim*int(concat_aggregation))
        self.projection =
nn.Linear(hidden_dim+hidden_dim*int(concat_aggregation), num_classes)

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

        self.aggregation_type = aggregation_type
        self.concat_aggregation = concat_aggregation

    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' and self.concat_aggregation
is False:
            output = output.max(dim=1)[0] #[batch_size, hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is False:
            output = output.mean(dim=1) #[batch_size, hidden_dim]
        elif self.aggregation_type == 'max' and
self.concat_aggregation is True:
            output = torch.cat((output.max(dim=1)[0], output[:, -
1, :]), dim=1) #[batch_size, 2*hidden_dim]
        elif self.aggregation_type == 'mean' and
self.concat_aggregation is True:
            output = torch.cat((output.mean(dim=1), output[:, -1, :]),
dim=1) #[batch_size, 2*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 = 10
eval_steps = len(train_dataloader) // 2


losses_type = {}
```

```
acc_type = {}

print(f"Starting training for LSTM")

losses = {
    'eval': [],
    'train': [],
}
acc = {
    'eval': [],
    'train': [],
}

model = UpgradedLMLSTM(
    hidden_dim=256, vocab_size=len(vocab), aggregation_type='mean',
hidden_rnn_layers=2, dropout_prob=0.1,
    concat_aggregation=True).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['eval'].append(evaluate(model, eval_dataloader))
            acc['train'].append(evaluate(model, train_dataloader))
            model.train()

    losses['train'].append(sum(epoch_losses) / len(epoch_losses))

    # compute loss for eval dataset at the end of each epoch
    epoch_losses = []
    with torch.no_grad():
        for batch in eval_dataloader:
            logits = model(batch['input_ids'])
            loss = criterion(logits, batch['label'])
            epoch_losses.append(loss.item())

    losses['eval'].append(sum(epoch_losses) / len(epoch_losses))
```

```
losses_type['LSTM'] = losses
acc_type['LSTM'] = acc
```

```
Starting training for LSTM
```


```
{"model_id":"b4ddf33b5da14fe789707b248385b156","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"7ac97396a91b41f0b2e2e7577d382c55","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"db17e3eb0b8b414fa832d02fa38f8048","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"c7d5a9f44b3a468cbd3c9bdeacacc18c","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"30121d1c51a24b76b8ee5c6b845a45d3","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"7575bb69e0d744a2a8a47b6408d5e3cb","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"26d922750a244ad28165edc364a09ea0","version_major":2,"vers
ion_minor":0}
```

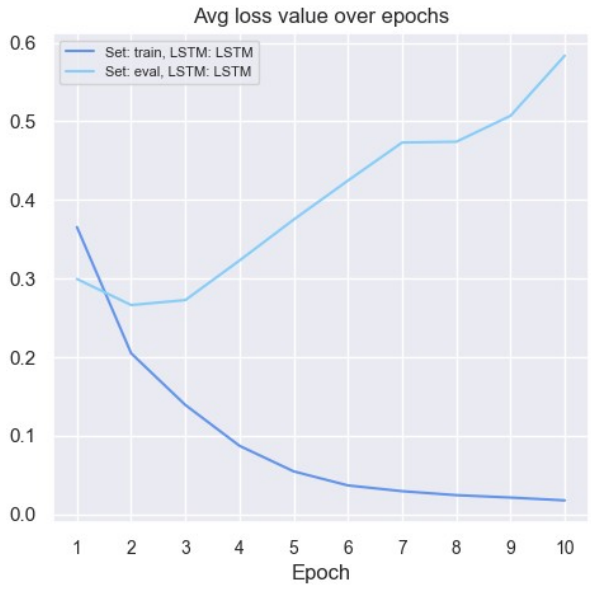
```
{"model_id":"e7eac183601c4eefb356270c98b685ae","version_major":2,"vers
ion_minor":0}
```

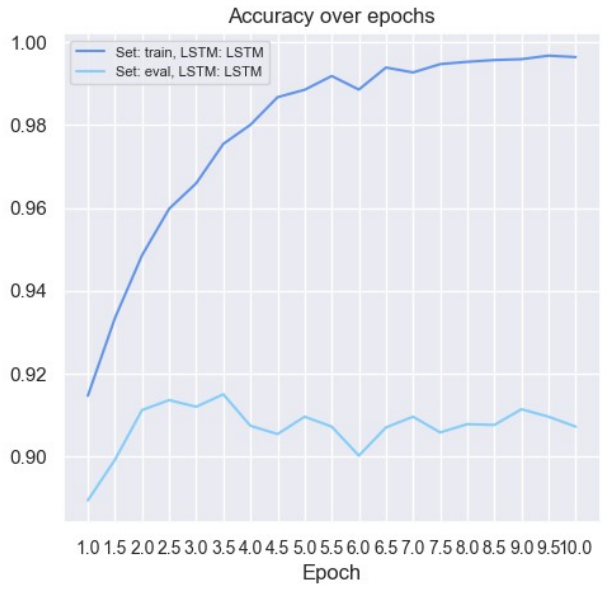
```
{"model_id":"95855e7de9ca404ab093a800b9536a28","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"492f00f7965341e6bffc23cf85cb5b24","version_major":2,"vers
ion_minor":0}
```


```
plot_learning_curves(losses_type, acc_type, param_sets=['LSTM'],
param_name='LSTM')
```

```
Лучшая accuracy для LSTM=LSTM на train: 99.67
Последняя accuracy для LSTM=LSTM на train: 99.64

Лучшая accuracy для LSTM=LSTM на eval: 91.50
Последняя accuracy для LSTM=LSTM на eval: 90.72
```





Примерно похожие результаты, но чуть не дотягивает до GRU на зафиксированном семпле eval.

Итого лучшее качество: 91.48 (последняя эпоха) и 91.70 (максимальное значение на эпохе 2.5).

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!

Вывод:

- мне было интересно посмотреть какие из экспериментов принесут наибольший прирост по качеству для так называемого vanilla RNN прежде чем применять GRU/LSTM
- я попробовала потвикать почти все предложенные в задании гиперпараметры
  - конкатенация последнего эмбеддинга к аггрегации почти ничего не дало
  - dropout rate и так был хорошо выставлен
  - размер скрытых слоев (256) тоже подобран оптимально, можно было бы сократить до 128 без значимых потерь
  - аггрегация mean показала себя лучше max
  - дополнительный слой rnn (num_layers) добавил качества по сравнению с семинарской бейзлайновой моделью
  - чем больше размер словаря, тем качество лучше, но дольше обучается модель; я чуть уменьшила отсечку (10) по сравнению с бейзлайном (25)
- в целом, все эти твики значимого прироста, позволяющего побить отсечку в 0.91 не особо помогли
- наибольший прирост дал переход на GRU и LSTM, GRU показал себя чуть лучше (на моем конкретном eval)
- получилось подобрать модель с accuracy = **0.9148** (финально) и 0.9170 (в максимуме, на 2.5 эпохе, не совсем честно).
- хотела бы попробовать bidirectional варианты, но т.к. тренировалась на cpu, времени не хватило.