Deep Learning School

Домашнее задание. Обучение языковой модели с помощью LSTM (10 баллов)

Э В этом задании Вам предстоит обучить языковую модель с помощью рекуррентной нейронной сети. В отличие от семинарского занятия, Вам необходимо будет работать с отдельными словами, а не буквами.

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

```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-3.4.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from datasets) (3.17.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (18.1.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (from datasets) (4.67.1)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocess<0.70.17 (from datasets)
  Downloading multiprocess-0.70.16-py311-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (from fsspec[http
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (from datasets) (3.11.13)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (from datasets) (0.28.
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->dataset
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.3
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (25.2.0
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (0.3
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (from aiohttp->datasets) (1.
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-h
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->datasets)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->dat
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.32.2->dat
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->datasets) (2025.1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas-
Downloading datasets-3.4.0-py3-none-any.whl (487 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 487.4/487.4 kB 8.6 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 8.7 MB/s eta 0:00:00
Downloading multiprocess-0.70.16-py311-none-any.whl (143 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 6.8 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 7.2 MB/s eta 0:00:00
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-3.4.0 dill-0.3.8 multiprocess-0.70.16 xxhash-3.5.0
```

Импорт необходимых библиотек

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

import numpy as np
import matplotlib.pyplot as plt

from tqdm.auto import tqdm
from datasets import load_dataset
from nltk.tokenize import sent_tokenize, word_tokenize
```

```
from sklearn.model_selection import train_test_split
import nltk

from collections import Counter
from typing import List

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

```
nltk.download('punkt')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

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

```
'cuda'
```

Подготовка данных

Воспользуемся датасетом imdb. В нем хранятся отзывы о фильмах с сайта imdb. Загрузим данные с помощью функции `load_dataset`

```
# Загрузим датасет
dataset = load_dataset('imdb')
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: 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 :
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(
README.md: 100%  7.81k/7.81k [00:00<00:00, 310kB/s]
train-00000-of-00001.parquet: 100%  21.0M/21.0M [00:01<00:00, 22.2MB/s]
test-00000-of-00001.parquet: 100%  20.5M/20.5M [00:00<00:00, 73.8MB/s]
unsupervised-00000-of-00001.parquet: 100%  42.0M/42.0M [00:00<00:00, 67.3MB/s]
Generating train split: 100%  25000/25000 [00:00<00:00, 63236.71 examples/s]
Generating test split: 100%  25000/25000 [00:00<00:00, 49523.74 examples/s]
Generating unsupervised split: 100%  50000/50000 [00:01<00:00, 39717.88 examples/s]
```

Препроцессинг данных и создание словаря (1 балл)

Далее вам необходмо самостоятельно произвести препроцессинг данных и получить словарь или же просто `set` строк. Что необходимо сделать:

1. Разделить отдельные тренировочные примеры на отдельные предложения с помощью функции `sent_tokenize` из бибилиотеки `nltk`. Каждое отдельное предложение будет одним тренировочным примером.
2. Оставить только те предложения, в которых меньше `word_threshold` слов.
3. Посчитать частоту вхождения каждого слова в оставшихся предложениях. Для деления предлоения на отдельные слова удобно использовать функцию `word_tokenize`.
4. Создать объект `vocab` класса `set`, положить в него служебные токены '<unk>', '<bos>', '<eos>', '<pad>' и vocab_size самых частовстречающихся слов.

```
sentences = []
word_threshold = 32

# Получить отдельные предложения и поместить их в sentences
```

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
```

```
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

```
sent_tokenize(dataset['train']['text'][0])
```

```
['I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967.',
 'I also heard that at first it was seized by U.S.',
 'customs if it ever tried to enter this country, therefore being a fan of films considered "controversial" I really had to see this for myself.<br /><br />The plot is centered around a young Swedish drama student named Lena who wants to learn everything she can about life.',
 'In particular she wants to focus her attentions to making some sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues in the United States.',
 'In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has sex with her drama teacher, classmates, and married men.<br /><br />What kills me about I AM CURIOUS-YELLOW is that 40 years ago, this was considered pornographic.',
 "Really, the sex and nudity scenes are few and far between, even then it's not shot like some cheaply made porno.",
 'While my countrymen mind find it shocking, in reality sex and nudity are a major staple in Swedish cinema.',
 'Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.<br /><br />I do commend the filmmakers for the fact that any sex shown in the film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in America.',
 'I AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and potatoes (no pun intended) of Swedish cinema.',
 "But really, this film doesn't have much of a plot."]
```

```
for sentence in tqdm(dataset['train']['text']):
    sentences.extend(
        [x.lower() for x in sent_tokenize(sentence, language='russian') if len(word_tokenize(x)) < word_threshold]
        )
```

100% 25000/25000 [00:36<00:00, 719.86it/s]

```
sentences
```

Показать скрытые выходные данные

```
print("Всего предложений:", len(sentences))
```

```
Всего предложений: 200848
```

Посчитаем для каждого слова его встречаемость.

```
words = Counter()

# Расчет встречаемости слов
for sentence in tqdm(sentences):
    for word in word_tokenize(sentence):
        words[word] += 1
```

100% 200848/200848 [00:21<00:00, 10645.73it/s]

Добавим в словарь `vocab_size` самых встречающихся слов.

```
vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
vocab_size = 40000

# Наполнение словаря
for word in words.most_common()[:vocab_size]:
  vocab.add(word[0])
```

```
assert '<unk>' in vocab
assert '<bos>' in vocab
assert '<eos>' in vocab
assert '<pad>' in vocab
assert len(vocab) == vocab_size + 4
```

```
print("Всего слов в словаре:", len(vocab))
```

```
Всего слов в словаре: 40004
```

Подготовка датасета (1 балл)

Далее, как и в семинарском занятии, подготовим датасеты и даталоадеры.

В классе `WordDataset` вам необходимо реализовать метод `__getitem__`, который будет возвращать сэмпл данных по входному idx, то есть список целых чисел (индексов слов).

Внутри этого метода необходимо добавить служебные токены начала и конца последовательности, а также токенизировать соответствующее предложение с помощью `word_tokenize` и сопоставить ему индексы из `word2ind`.

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


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]:
        tokenized_sentence = []
        # Допишите код здесь
        tokenized_sentence = [self.bos_id]
        tokenized_sentence += [word2ind.get(word, self.unk_id) for word in word_tokenize(self.data[idx])]
        tokenized_sentence += [self.eos_id]


        return tokenized_sentence

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


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

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

    sequences = torch.LongTensor(new_batch).to(device)

    new_batch = {
        'input_ids': sequences[:,:-1],
        'target_ids': sequences[:,1:]
    }

    return new_batch


train_sentences, eval_sentences = train_test_split(sentences, test_size=0.2)
eval_sentences, test_sentences = train_test_split(eval_sentences, test_size=0.5)

train_dataset = WordDataset(train_sentences)
eval_dataset = WordDataset(eval_sentences)
test_dataset = WordDataset(test_sentences)

batch_size = 128

train_dataloader = DataLoader(
    train_dataset, collate_fn=collate_fn_with_padding, batch_size=batch_size)

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

test_dataloader = DataLoader(
    test_dataset, collate_fn=collate_fn_with_padding, batch_size=batch_size)
```

Обучение и архитектура модели

Вам необходимо на практике проверить, что влияет на качество языковых моделей. В этом задании нужно провести серию экспериментов с различными вариантами языковых моделей и сравнить различия в конечной перплексии на тестовом множестве.

Возмоэные идеи для экспериментов:

- Различные RNN-блоки, например, LSTM или GRU. Также можно добавить сразу несколько RNN блоков друг над другом с помощью аргумента num_layers. Вам поможет официальная документация [здесь]
- Различные размеры скрытого состояния. Различное количество линейных слоев после RNN-блока. Различные функции активации.
- Добавление нормализаций в виде Dropout, BatchNorm или LayerNorm
- Различные аргументы для оптимизации, например, подбор оптимального learning rate или тип алгоритма оптимизации SGD, Adam, RMSProp и другие
- Любые другие идеи и подходы

После проведения экспериментов необходимо составить таблицу результатов, в которой описан каждый эксперимент и посчитана перплексия на тестовом множестве.

Учтите, что эксперименты, которые различаются, например, только размером скрытого состояния или количеством линейных слоев считаются, как один эксперимент.

Успехов!

Функция evaluate (1 балл)

Заполните функцию `evaluate`

```
def evaluate(model, criterion, dataloader) -> float:
    model.eval()
    perplexity = []
    with torch.no_grad():
        for batch in dataloader:
            logits = model(batch['input_ids']).flatten(start_dim=0, end_dim=1) # Посчитайте логиты предсказаний следующих сл
            loss = criterion(logits, batch['target_ids'].flatten())
            perplexity.append(torch.exp(loss).item())

    perplexity = sum(perplexity) / len(perplexity)

    return perplexity
```

Train loop (1 балл)

Напишите функцию для обучения модели.

```
def train_model(num_epoch, model, optimizer, criterion, scheduler=None):
    # Напишите код здесь
    losses = []
    perplexities = []

    for epoch in range(num_epoch):
        epoch_losses = []
        model.train()
        for batch in tqdm(train_dataloader, desc=f'Training epoch {epoch}:'):
            optimizer.zero_grad()
            logits = model(batch['input_ids']).flatten(start_dim=0, end_dim=1)
            loss = criterion(
                logits, batch['target_ids'].flatten())
            loss.backward()
            optimizer.step()

            epoch_losses.append(loss.item())

        loss = sum(epoch_losses) / len(epoch_losses)
        losses.append(loss)
        perplexity = evaluate(model, criterion, test_dataloader)
        perplexities.append(perplexity)
        if scheduler:
            scheduler.step()
        print(f'эпоха = {epoch}, loss = {loss}, perplexity = {perplexity}, lr = {optimizer.param_groups[0]["lr"]}')
    return losses, perplexities
```

```
def generate_sequence(model, starting_seq: str, max_seq_len: int = 128) -> str:
    device = 'cpu'
    model = model.to(device)
    input_ids = [word2ind['<bos>']] + [
        word2ind.get(word, word2ind['<unk>']) for word in word_tokenize(starting_seq)]
    input_ids = torch.LongTensor(input_ids).to(device)

    model.eval()
    with torch.no_grad():
        for i in range(max_seq_len):
            next_char_distribution = model(input_ids)[-1]
            next_char = next_char_distribution.squeeze().argmax()
            input_ids = torch.cat([input_ids, next_char.unsqueeze(0)])

            if next_char.item() == word2ind['<eos>']:
                break

    words = ' '.join([ind2word[idx.item()] for idx in input_ids])

    return words


def plot(losses, perplexities):
  plt.plot(np.arange(len(losses)), losses)
  plt.title('Losses')
  plt.xlabel("epoch")
  plt.show()

  plt.plot(np.arange(len(perplexities)), perplexities)
  plt.title('Perplexity')
  plt.xlabel("epoch")
  plt.show()


class LanguageModel(nn.Module):
    def __init__(self, hidden_dim, vocab_size, type_nn, num_layers):
        super().__init__()
        # Опишите свою нейронную сеть здесь
        self.embedding = nn.Embedding(vocab_size, hidden_dim)
        rnn_type = {'rnn': nn.RNN, 'gru': nn.GRU, 'lstm': nn.LSTM}[type_nn]
        self.rnn = rnn_type(hidden_dim, hidden_dim, batch_first=True, num_layers=num_layers)
        self.linear = nn.Linear(hidden_dim, hidden_dim)
        self.projection = nn.Linear(hidden_dim, vocab_size)

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

    def forward(self, input_batch: torch.Tensor) -> torch.Tensor:
        # А тут опишите forward pass модели
        embeddings = self.embedding(input_batch)  # [batch_size, seq_len, hidden_dim]
        output, _ = self.rnn(embeddings)  # [batch_size, seq_len, hidden_dim]
        output = self.dropout(self.linear(self.non_lin(output)))  # [batch_size, seq_len, hidden_dim]
        projection = self.projection(self.non_lin(output))  # [batch_size, seq_len, vocab_size]
        return projection
```

Первый эксперимент (2 балла)

Определите архитектуру модели и обучите её.

эксперимент 1.1

- берем самую простую сеть для начала (один слой RNN, hidden_dim = 256)

```
model_1 = LanguageModel(hidden_dim=256, vocab_size=len(vocab), type_nn='rnn', num_layers=1).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_1.parameters())


for a in train_dataloader:
  print(a)
  res = model_1(a['input_ids'])
  print(res)
  break
```

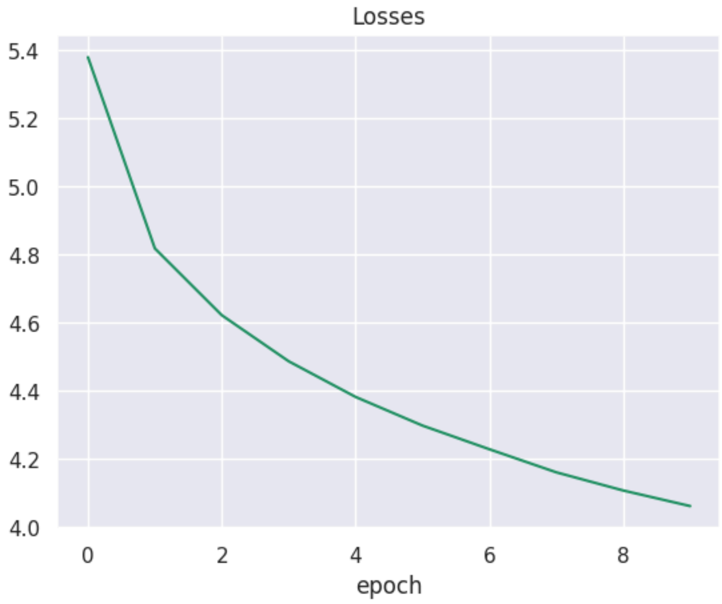
Показать скрытые выходные данные

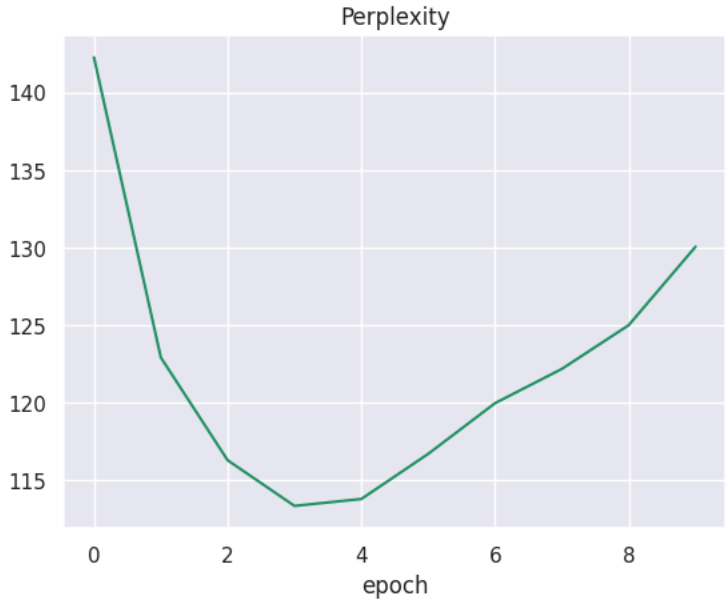
```
# Обучите модель здесь

losses_1, perplexities_1 = train_model(num_epoch=10, model=model_1, optimizer=optimizer, criterion=criterion)
```

```
Training epoch 0:: 100%    1256/1256 [02:12<00:00,  9.04it/s]
эпоха = 0, loss = 5.378464142228387, perplexity = 142.2553893168261, lr = 0.001
Training epoch 1:: 100%    1256/1256 [02:10<00:00,  9.81it/s]
эпоха = 1, loss = 4.817174610058973, perplexity = 122.93043245935137, lr = 0.001
Training epoch 2:: 100%    1256/1256 [02:09<00:00,  9.82it/s]
эпоха = 2, loss = 4.621186514568937, perplexity = 116.30543323686928, lr = 0.001
Training epoch 3:: 100%    1256/1256 [02:09<00:00,  9.16it/s]
эпоха = 3, loss = 4.485903041757596, perplexity = 113.3659650474597, lr = 0.001
Training epoch 4:: 100%    1256/1256 [02:09<00:00,  9.77it/s]
эпоха = 4, loss = 4.381243451385741, perplexity = 113.81143761896024, lr = 0.001
Training epoch 5:: 100%    1256/1256 [02:09<00:00,  9.88it/s]
эпоха = 5, loss = 4.297119844111667, perplexity = 116.71759533730282, lr = 0.001
Training epoch 6:: 100%    1256/1256 [02:09<00:00,  9.58it/s]
эпоха = 6, loss = 4.227645406107993, perplexity = 119.98320813999055, lr = 0.001
Training epoch 7:: 100%    1256/1256 [02:11<00:00,  9.71it/s]
эпоха = 7, loss = 4.1599253740659945, perplexity = 122.19182576950948, lr = 0.001
Training epoch 8:: 100%    1256/1256 [02:09<00:00,  9.69it/s]
эпоха = 8, loss = 4.106596268476195, perplexity = 125.00944830050135, lr = 0.001
Training epoch 9:: 100%    1256/1256 [02:09<00:00,  9.01it/s]
эпоха = 9, loss = 4.060875462118987, perplexity = 130.06591898924225, lr = 0.001
```

```
plot(losses_1, perplexities_1)
```






```
starting_seq_1 = 'I like movie'
max_seq_len = 128
generate_sequence(model_1, starting_seq_1, max_seq_len)
```

```
'<bos> <unk> like movie goers . i 'm not sure that i was n't expecting much . <eos>'
```

эксперимент 1.2

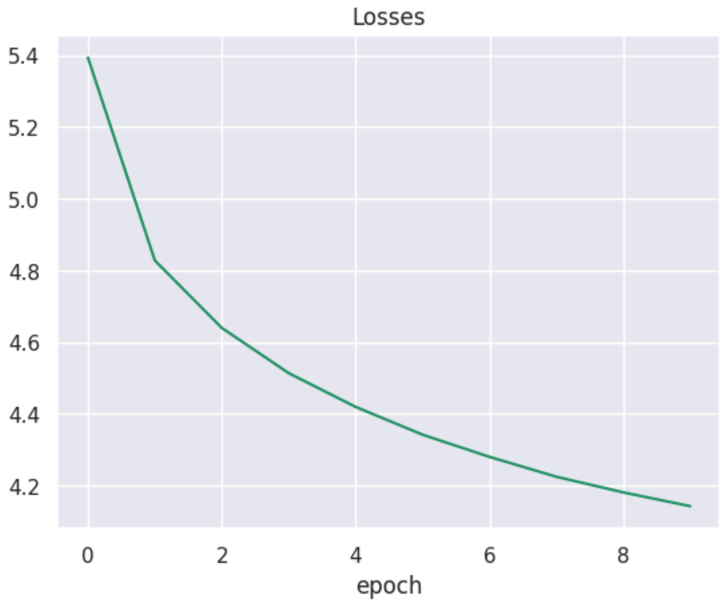
- добавляем переменный шаг оптмизатора

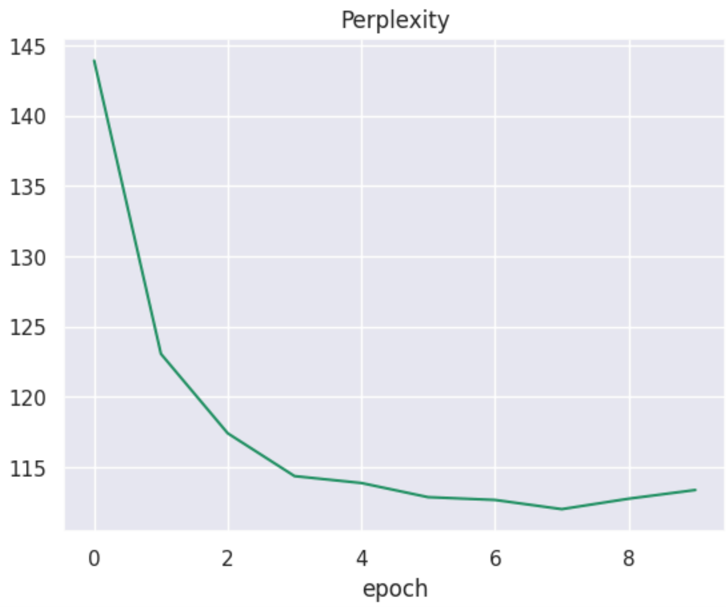
```
model_1_1 = model_1 = LanguageModel(hidden_dim=256, vocab_size=len(vocab), type_nn='rnn', num_layers=1).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_1.parameters())
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9) # ExponentialLR / StepLR
```

```
losses_1_1, perplexities_1_1 = train_model(num_epoch=10, model=model_1_1, optimizer=optimizer, criterion=criterion, schedule
```

```
Training epoch 0:: 100%    1256/1256 [02:13<00:00,  8.56it/s]
эпоха = 0, loss = 5.392974049422391, perplexity = 143.9029773299102, lr = 0.0009000000000000001
Training epoch 1:: 100%    1256/1256 [02:12<00:00,  9.70it/s]
эпоха = 1, loss = 4.827889321715968, perplexity = 123.04656880372649, lr = 0.0008100000000000001
Training epoch 2:: 100%    1256/1256 [02:12<00:00,  9.60it/s]
эпоха = 2, loss = 4.640049057781317, perplexity = 117.39660037095379, lr = 0.000729
Training epoch 3:: 100%    1256/1256 [02:11<00:00,  8.65it/s]
эпоха = 3, loss = 4.513982618690297, perplexity = 114.35054686722482, lr = 0.0006561000000000001
Training epoch 4:: 100%    1256/1256 [02:09<00:00,  9.84it/s]
эпоха = 4, loss = 4.4199350745814625, perplexity = 113.86058238813072, lr = 0.00059049
Training epoch 5:: 100%    1256/1256 [02:09<00:00,  9.24it/s]
эпоха = 5, loss = 4.342529796491004, perplexity = 112.85251816670606, lr = 0.000531441
Training epoch 6:: 100%    1256/1256 [02:09<00:00,  9.72it/s]
эпоха = 6, loss = 4.280743056801474, perplexity = 112.65680748034434, lr = 0.0004782969
Training epoch 7:: 100%    1256/1256 [02:09<00:00,  9.70it/s]
эпоха = 7, loss = 4.22524566957905, perplexity = 112.00076114144295, lr = 0.00043046721
Training epoch 8:: 100%    1256/1256 [02:09<00:00,  9.54it/s]
эпоха = 8, loss = 4.181714861256302, perplexity = 112.74284333636047, lr = 0.000387420489
Training epoch 9:: 100%    1256/1256 [02:09<00:00,  9.72it/s]
эпоха = 9, loss = 4.143341516233553, perplexity = 113.36287927323846, lr = 0.0003486784401
```

```
plot(losses_1_1, perplexities_1_1)
```






```
starting_seq_1 = 'I like movie'
max_seq_len = 128
```

```
generate_sequence(model_1_1, starting_seq_1, max_seq_len)
```

```
'<bos> <unk> like movie makers . but i do n't think that the movie was made in the film . but it was n't a good movie . <eos>'
```

Второй эксперимент (2 балла)

Попробуйте что-то поменять в модели или в пайплайне обучения, идеи для экспериментов можно подсмотреть выше.

эксперимент 2.1

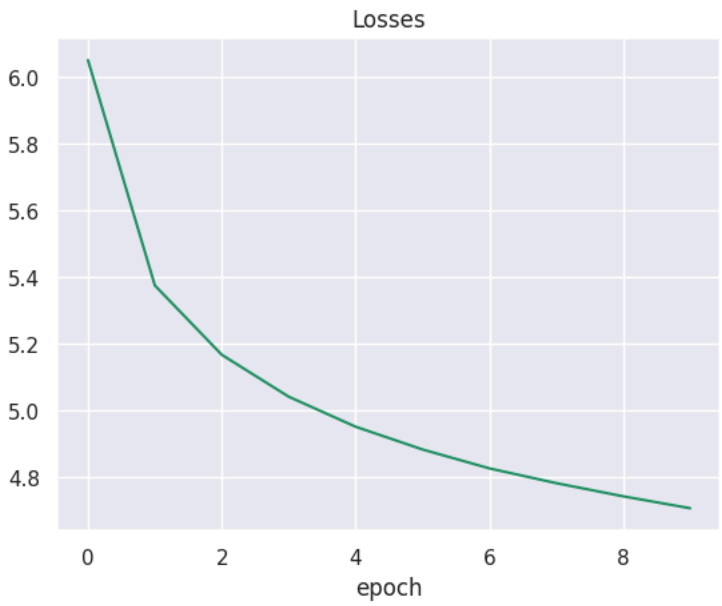
- возьмем слой GRU
- увеличим число слоев
- увеличим hidden_state

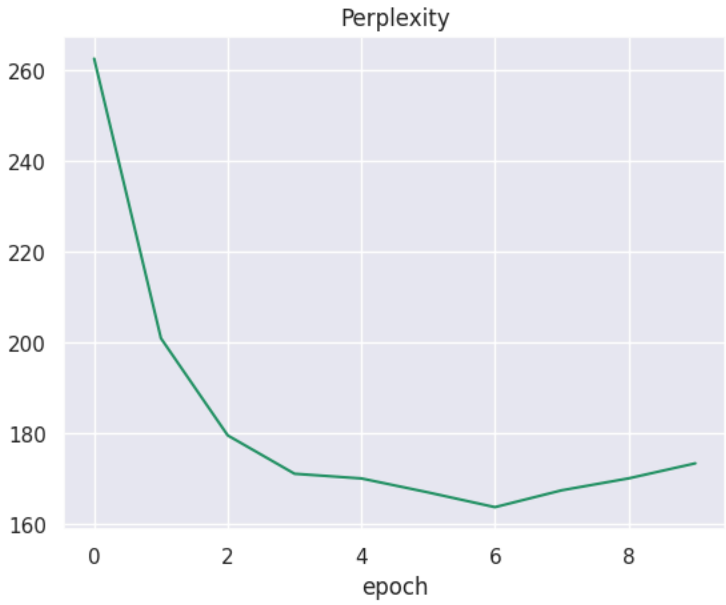
```
# Проведите второй эксперимент
model_2 = LanguageModel(hidden_dim=512, vocab_size=len(vocab), type_nn='gru', num_layers=3).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_2.parameters())
```

```
losses_2, perplexities_2 = train_model(num_epoch=10, model=model_2, optimizer=optimizer, criterion=criterion)
```

```
Training epoch 0:: 100%    1256/1256 [04:16<00:00,   4.94it/s]
эпоха = 0, loss = 6.0505735953901985, perplexity = 262.59094082777665, lr = 0.001
Training epoch 1:: 100%    1256/1256 [04:16<00:00,   4.91it/s]
эпоха = 1, loss = 5.375409513142459, perplexity = 201.00261158548344, lr = 0.001
Training epoch 2:: 100%    1256/1256 [04:16<00:00,   4.93it/s]
эпоха = 2, loss = 5.167739720101569, perplexity = 179.60454972382564, lr = 0.001
Training epoch 3:: 100%    1256/1256 [04:15<00:00,   4.92it/s]
эпоха = 3, loss = 5.042161784354289, perplexity = 171.1605898134268, lr = 0.001
Training epoch 4:: 100%    1256/1256 [04:15<00:00,   4.91it/s]
эпоха = 4, loss = 4.9521428225146735, perplexity = 170.12752712760002, lr = 0.001
Training epoch 5:: 100%    1256/1256 [04:15<00:00,   4.83it/s]
эпоха = 5, loss = 4.883955277834728, perplexity = 167.0521133325662, lr = 0.001
Training epoch 6:: 100%    1256/1256 [04:15<00:00,   4.93it/s]
эпоха = 6, loss = 4.82730225127214, perplexity = 163.80293429429364, lr = 0.001
Training epoch 7:: 100%    1256/1256 [04:15<00:00,   4.93it/s]
эпоха = 7, loss = 4.783010491519977, perplexity = 167.51698944674936, lr = 0.001
Training epoch 8:: 100%    1256/1256 [04:15<00:00,   4.95it/s]
эпоха = 8, loss = 4.743585213734086, perplexity = 170.13876862738542, lr = 0.001
Training epoch 9:: 100%    1256/1256 [04:15<00:00,   4.76it/s]
эпоха = 9, loss = 4.707871940864879, perplexity = 173.45724633089296, lr = 0.001
```

```
plot(losses_2, perplexities_2)
```






```
starting_seq_1 = 'I like movie'
max_seq_len = 128
generate_sequence(model_2, starting_seq_1, max_seq_len)
```

```
'<bos> <unk> like movie people are a huge . <unk> . and good films . <eos>'
```

экперимент 2.2

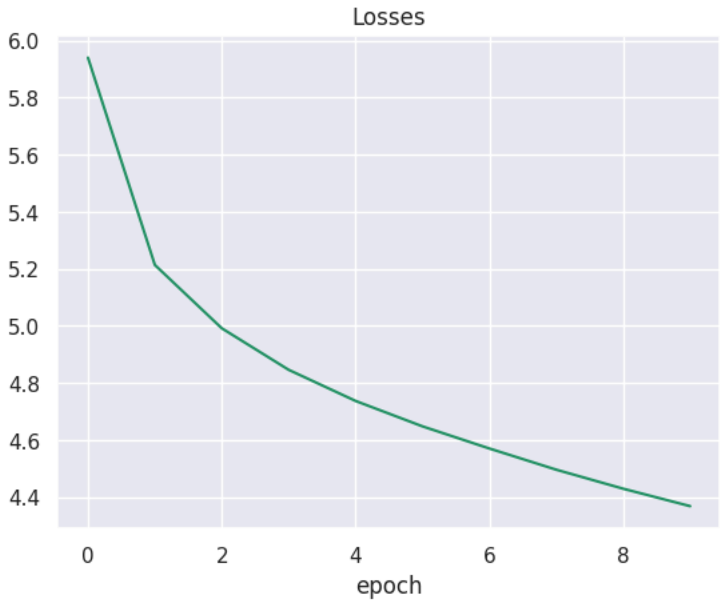
- возьмем переменный шаг оптимизатора по эпохам

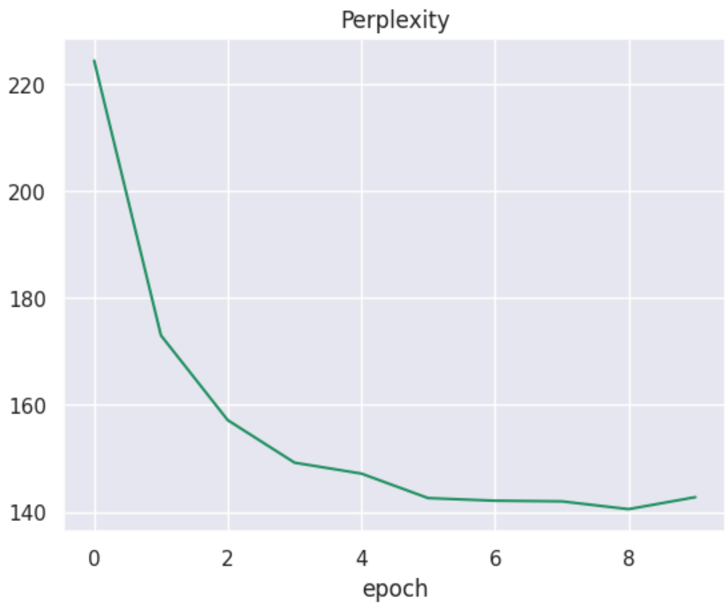
```
model_3 = LanguageModel(hidden_dim=512, vocab_size=len(vocab), type_nn='gru', num_layers=3).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_3.parameters())
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9) # ExponentialLR / StepLR
```

```
losses_3, perplexities_3 = train_model(num_epoch=10, model=model_3, optimizer=optimizer, criterion=criterion, scheduler=sche
```

```
Training epoch 0:: 100%   1256/1256 [04:16<00:00,  4.91it/s]
эпоха = 0, loss = 5.940239334562023, perplexity = 224.35274821481886, lr = 0.0009000000000000001
Training epoch 1:: 100%   1256/1256 [04:17<00:00,  4.93it/s]
эпоха = 1, loss = 5.213789637301378, perplexity = 173.01997900312873, lr = 0.0008100000000000001
Training epoch 2:: 100%   1256/1256 [04:17<00:00,  4.91it/s]
эпоха = 2, loss = 4.991764311198216, perplexity = 157.1794513775285, lr = 0.000729
Training epoch 3:: 100%   1256/1256 [04:17<00:00,  4.85it/s]
эпоха = 3, loss = 4.846132029773323, perplexity = 149.2376710442221, lr = 0.0006561000000000001
Training epoch 4:: 100%   1256/1256 [04:17<00:00,  4.76it/s]
эпоха = 4, loss = 4.737083963907448, perplexity = 147.19751404197353, lr = 0.00059049
Training epoch 5:: 100%   1256/1256 [04:17<00:00,  4.88it/s]
эпоха = 5, loss = 4.647990478451844, perplexity = 142.61664892305993, lr = 0.000531441
Training epoch 6:: 100%   1256/1256 [04:17<00:00,  4.90it/s]
эпоха = 6, loss = 4.570334340736365, perplexity = 142.128936816173, lr = 0.0004782969
Training epoch 7:: 100%   1256/1256 [04:16<00:00,  4.87it/s]
эпоха = 7, loss = 4.496066866786617, perplexity = 142.008181140681, lr = 0.00043046721
Training epoch 8:: 100%   1256/1256 [04:17<00:00,  4.87it/s]
эпоха = 8, loss = 4.42941877910286, perplexity = 140.55082867859275, lr = 0.000387420489
Training epoch 9:: 100%   1256/1256 [04:17<00:00,  4.78it/s]
эпоха = 9, loss = 4.368321799548569, perplexity = 142.7922179592643, lr = 0.0003486784401
```

```
plot(losses_3, perplexities_3)
```






```
starting_seq_1 = 'I like movie'
max_seq_len = 128
```

```
generate_sequence(model_3, starting_seq_1, max_seq_len)
```

```
'<bos> <unk> like movie . and the movie is a very good movie . <eos>'
```

экперимент 2.3

- поменяем оптимизатор на AdamW

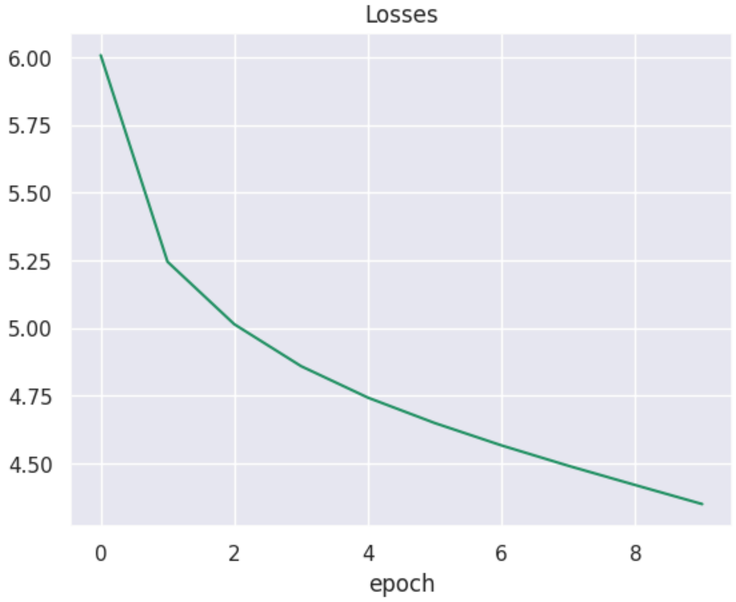
```
model_4 = LanguageModel(hidden_dim=512, vocab_size=len(vocab), type_nn='gru', num_layers=3).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.AdamW(model_4.parameters())
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9) # ExponentialLR / StepLR
```

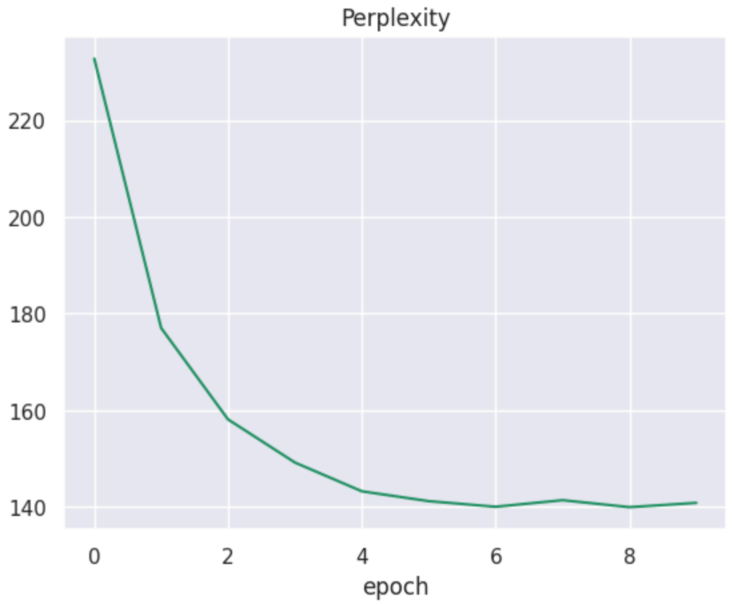
```
losses_4, perplexities_4 = train_model(num_epoch=10, model=model_4, optimizer=optimizer, criterion=criterion, scheduler=sche
```

```
Training epoch 0:: 100%    1256/1256 [04:17<00:00, 4.86it/s]
эпоха = 0, loss = 6.007832512734042, perplexity = 232.7152780909447, lr = 0.0009000000000000001
Training epoch 1:: 100%    1256/1256 [04:18<00:00, 4.83it/s]
эпоха = 1, loss = 5.2445176843624965, perplexity = 177.0240300658402, lr = 0.0008100000000000001
Training epoch 2:: 100%    1256/1256 [04:18<00:00, 4.89it/s]
эпоха = 2, loss = 5.0134022418101125, perplexity = 158.1190784235669, lr = 0.000729
Training epoch 3:: 100%    1256/1256 [04:19<00:00, 4.83it/s]
эпоха = 3, loss = 4.858410846275889, perplexity = 149.2049295218887, lr = 0.0006561000000000001
Training epoch 4:: 100%    1256/1256 [04:19<00:00, 4.89it/s]
эпоха = 4, loss = 4.7421163662224055, perplexity = 143.2618579257066, lr = 0.00059049
Training epoch 5:: 100%    1256/1256 [04:19<00:00, 4.75it/s]
эпоха = 5, loss = 4.648284921980208, perplexity = 141.22040339184414, lr = 0.000531441
Training epoch 6:: 100%    1256/1256 [04:19<00:00, 4.83it/s]
эпоха = 6, loss = 4.565524692748003, perplexity = 140.06135826353815, lr = 0.0004782969
Training epoch 7:: 100%    1256/1256 [04:20<00:00, 4.85it/s]
эпоха = 7, loss = 4.490207724130837, perplexity = 141.4216078254068, lr = 0.00043046721
Training epoch 8:: 100%    1256/1256 [04:20<00:00, 4.85it/s]
эпоха = 8, loss = 4.419146230266352, perplexity = 139.98011235522617, lr = 0.000387420489
Training epoch 9:: 100%    1256/1256 [04:20<00:00, 4.88it/s]
эпоха = 9, loss = 4.348910540721978, perplexity = 140.86248152423056, lr = 0.0003486784401
```

```
plot(losses_4, perplexities_4)
```

```
plot(losses_4, perplexities_4)
```






```
starting_seq_1 = 'I like movie'
max_seq_len = 128
generate_sequence(model_4, starting_seq_1, max_seq_len)
```

```
'<bos> <unk> like movie is a bit interesting , but it 's a very good movie . <eos>'
```

```
starting_seq_1 = 'this movie is very'
max_seq_len = 128
for index, model in enumerate([model_1, model_1_1, model_2, model_3, model_4]):
  res = generate_sequence(model, starting_seq_1, max_seq_len)
  print(f'{index} - {res}')
```

```
0 - <bos> this movie is very good . <eos>
1 - <bos> this movie is very good . <eos>
2 - <bos> this movie is very good . <eos>
3 - <bos> this movie is very good , but it 's a very good movie . <eos>
4 - <bos> this movie is very good , but it 's a very good movie . <eos>
```

```
starting_seq_1 = 'i really love this'
max_seq_len = 128
for index, model in enumerate([model_1, model_1_1, model_2, model_3, model_4]):
  res = generate_sequence(model, starting_seq_1, max_seq_len)
  print(f'{index} - {res}')
```

```
0 - <bos> i really love this movie , i was n't expecting much . <eos>
1 - <bos> i really love this movie , i was n't expecting much . <eos>
2 - <bos> i really love this movie . <eos>
3 - <bos> i really love this movie . <eos>
4 - <bos> i really love this movie . <eos>
```

```
starting_seq_1 = 'how are'
max_seq_len = 128
for index, model in enumerate([model_1, model_1_1, model_2, model_3, model_4]):
  res = generate_sequence(model, starting_seq_1, max_seq_len)
  print(f'{index} - {res}')
```

```
0 - <bos> how are you going to be a fan of the first time . <eos>
1 - <bos> how are you going to be a fan of the first time . <eos>
2 - <bos> how are you ? <eos>
3 - <bos> how are the characters to make this movie like this ? <eos>
4 - <bos> how are they ? <eos>
```

```
starting_seq_1 = 'genre of this movie is'
max_seq_len = 128
for index, model in enumerate([model_1, model_1_1, model_2, model_3, model_4]):
  res = generate_sequence(model, starting_seq_1, max_seq_len)
  print(f'{index} - {res}')
```

```
0 - <bos> genre of this movie is a very good movie . <eos>
1 - <bos> genre of this movie is a very good movie . <eos>
2 - <bos> genre of this movie is a good movie . <eos>
3 - <bos> genre of this movie is a great movie . <eos>
4 - <bos> genre of this movie is a very good movie . <eos>
```

Отчет (2 балла)

Опишите проведенные эксперименты. Сравните перплексии полученных моделей. Предложите идеи по улучшению качества моделей.

- **эксперимент 1.1**
  - перплексия min = 113
  - перплексия last = 130, обучение не стабильно
- **эксперимент 1.2**
  - перплексия min = 112
  - перплексия last = 113, обучение стабильно
- **эксперимент 2.1**
  - перплексия min = 164,
  - перплексия last = 173, обучение стабильно немного
- **эксперимент 2.2**
  - перплексия min = 140,
  - перплексия last = 142, обучение стабильно
- **эксперимент 2.3**
  - перплексия min = 140
  - перплексия last = 140, обучение стабильно
- оптимальная модель - модель из эксперимента 1.2
- улучшения возможные:
  - больше эпох взять
  - использование контекстных эмбедингов