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

Это домашнее задание проходит в формате peer-review. Это означает, что его будут проверять ваши однокурсники. Поэтому пишите разборчивый код, добавляйте комментарии и пишите выводы после проделанной работы.

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.

In [1]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.18.0-py3-none-any.whl (510 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 9.0 MB/s eta
0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-p
ackages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dis
t-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in /usr/local/lib/python3.1
0/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in /usr/local/lib/python3.10/
dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 15.5 MB/s eta
0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pac
kages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.1
0/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/di
st-packages (from datasets) (4.66.2)
Collecting xxhash (from datasets)
  Downloading xxhash-3.4.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl (194 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 12.7 MB/s eta
0:00:00
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 19.0 MB/s eta
0:00:00
Requirement already satisfied: fsspec[http]<=2024.2.0,>=2023.1.0 in /usr/lo
cal/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-pa
ckages (from datasets) (3.9.3)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/py
thon3.10/dist-packages (from datasets) (0.20.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from datasets) (24.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dis
t-packages (from datasets) (6.0.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
```

```
0/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/d
ist-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.
10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python
3.10/dist-packages (from aiohttp->datasets) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/
dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/py
thon3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/li
b/python3.10/dist-packages (from huggingface-hub>=0.19.4->datasets) (4.10.
0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/p
ython3.10/dist-packages (from requests>=2.19.0->datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
st-packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python
3.10/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.10/dist-packages (from requests>=2.19.0->datasets) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/di
st-packages (from pandas->datasets) (2023.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
ackages (from python-dateutil>=2.8.1->pandas->datasets) (1.16.0)
Installing collected packages: xxhash, dill, multiprocess, datasets
Successfully installed datasets-2.18.0 dill-0.3.8 multiprocess-0.70.16 xxha
sh-3.4.1
```

Импорт необходимых библиотек

In [2]:

```
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 string
import pandas as pd

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

In [3]:

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

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

Out[3]: True

In [4]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[4]: 'cuda'

Подготовка данных

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

In [5]:
```
# Загрузим датасет
dataset = load_dataset('imdb')
```

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

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.

In [5]:
```
```

Дополнительно здесь убираем ненужные служебные символы

In [6]:
```
print(string.punctuation)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

In [7]:
```
punctuation_upd = "".join(list(set(string.punctuation) - set(".`'-/")))
print(punctuation_upd)
```

```
_{:@"#*]!;,>()+|?<~[&\}$=^%
```

In [8]:
```
sentences = []
word_threshold = 32
# word_threshold = 100

for sentence in tqdm(dataset['train']['text']):
    sentences.extend(
        [x.lower().translate(str.maketrans(punctuation_upd, ' '*len(punctua
        )

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

In [9]:
```
dataset['train']['text'][0]
```

Out[9]:
```
'I rented I AM CURIOUS-YELLOW from my video store because of all the contro
versy that surrounded it when it was first released in 1967. I also heard t
hat at first it was seized by U.S. customs if it ever tried to enter this c
ountry, 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 abou
t life. In particular she wants to focus her attentions to making some sort
of documentary on what the average Swede thought about certain political is
sues such as the Vietnam War and race issues in the United States. In betwe
en asking politicians and ordinary denizens of Stockholm about their opinio
ns 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 ag
o, 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 com
mend the filmmakers for the fact that any sex shown in the film is shown fo
r artistic purposes rather than just to shock people and make money to be s
hown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good fil
m for anyone wanting to study the meat and potatoes (no pun intended) of Sw
edish cinema. But really, this film doesn\'t have much of a plot.'
```

In [10]:
```
sentences[0:3]
```

Out[10]:
```
['i rented i am curious-yellow from my video store because of all the contr
oversy that surrounded it when it was first released in 1967',
 'the plot is centered around a young swedish drama student named lena who
wants to learn everything she can about life',
 'in between asking politicians and ordinary denizens of stockholm about th
eir opinions on politics  she has sex with her drama teacher  classmates  a
nd married men']
```

In [11]:
```
print(len(dataset['train']['text']))
print("Всего предложений:", len(sentences))
```

```
25000
Всего предложений: 272181
```

Посчитаем для каждого слова его встречаемость.

In [12]:
```
words = Counter()

count = 0
for sentence in tqdm(sentences):
    for word in sentence.split(' '):
        if word != '':
            words[word] += 1

# Расчет встречаемости слов
```

Добавим в словарь `vocab_size` самых встречающихся слов.

In [13]:
```
len(words)
```

Out[13]: 93854

In [14]:
```
words_df_idxs = np.arange(len(words))
```

In [15]:
```
words_df = pd.DataFrame(data=list(words.items()),columns=['key','val'],inde
```

In [16]:
```
words_df_sorted = words_df.sort_values(by=['val'], ascending=False)
```

In [17]:
```
words_df_sorted_reindexed = words_df_sorted.reset_index(drop=True)
```

In [18]:
```
words_df_sorted_reindexed[0:40000]
```

Out[18]:

	key	val
0	the	230338
1	a	112403
2	and	109866
3	of	96851
4	to	91518
...	...	...
39995	captor	2
39996	crumpled	2
39997	goble	2

	key	val
39998	musn't	2
39999	felled	2

40000 rows × 2 columns

In [19]:
```
vocab = set(['<unk>', '<bos>', '<eos>', '<pad>'])
vocab_size = 40000
# vocab_size = len(words_df_sorted_reindexed)

for i in words_df_sorted_reindexed[0:vocab_size]['key'].to_list():
    vocab.add(i)

# Наполнение словаря
```

In [20]:
```
assert '<unk>' in vocab
assert '<bos>' in vocab
assert '<eos>' in vocab
assert '<pad>' in vocab
assert len(vocab) == vocab_size + 4
```

In [21]:
```
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` .

In [22]:
```
word2ind = {char: i for i, char in enumerate(vocab)}
ind2word = {i: char for char, i in word2ind.items()}
```

In [23]:

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

        return tokenized_sentence

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

In [24]:

```
def collate_fn_with_padding(
    input_batch: List[List[int]], pad_id=word2ind['<pad>']) -> torch.Tenso
    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
```

In [25]:

```
np.random.seed(42)

train_sentences, eval_sentences = train_test_split(sentences, test_size=0.
eval_sentences, test_sentences = train_test_split(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_si

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
```

In [26]:
```
index = 10
print(train_sentences[index])
print(train_dataset[index])
print(list(map(ind2word.get, train_dataset[index])))
```

```
i've just watch 2 films of pang brothers  the eye and one take only
[20517, 1318, 3170, 19561, 23837, 5433, 29086, 3784, 8838, 31635, 9615, 133
47, 28603, 7567, 15638, 21879, 5894]
['<bos>', "i've", 'just', 'watch', '2', 'films', 'of', 'pang', 'brothers',
'<unk>', 'the', 'eye', 'and', 'one', 'take', 'only', '<eos>']
```

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`

In [27]:
```
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_di
            loss = criterion(logits, batch['target_ids'].flatten())
            perplexity.append(torch.exp(loss).item())

    perplexity = sum(perplexity) / len(perplexity)

    return perplexity
```

In [28]:
```
def generate_sequence(model, starting_seq: str, max_seq_len: int = 32) -> 
    device = 'cpu'
    model = model.to(device)
    input_ids = [word2ind['<bos>']] + [word2ind.get(word, word2ind['<unk>'
    input_ids = torch.LongTensor(input_ids).to(device)

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

            if next_word.item() == word2ind['<eos>']:
                break
    return input_ids

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

    # return words
```

Train loop (1 балл)

Напишите функцию для обучения модели.

In [29]:
```
def train_model(model, criterion, optimizer, train_dataloader, val_dataloa
    losses = []
    perplexities = []

    for epoch in range(epochs):
        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_di
            loss = criterion(logits, batch['target_ids'].flatten())
            loss.backward()
            optimizer.step()
            epoch_losses.append(loss.item())

        losses.append(sum(epoch_losses) / len(epoch_losses))
        perplexities.append(evaluate(model, criterion, val_dataloader))

    return losses, perplexities
```

Первый эксперимент (2 балла)

Определите архитектуру модели и обучите её.

In [30]:
```
class LanguageModel(nn.Module):
    def __init__(self, hidden_dim: int, vocab_size: int, embedding_len: int
                 N_layers: int = 1, bd: bool = False, ln: bool = False, GRU
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_len)
        if GRU_or_LSTM == 'GRU':
            self.rnn = nn.GRU(embedding_len, hidden_dim, num_layers=N_laye
        elif GRU_or_LSTM == 'LSTM':
            self.rnn = nn.LSTM(embedding_len, hidden_dim, num_layers=N_lay
        self.layer_norm = nn.LayerNorm(hidden_dim)
        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)
        self.ln = ln
        # Опишите свою нейронную сеть здесь

    def forward(self, input_batch: torch.Tensor) -> torch.Tensor:
        embeddings = self.embedding(input_batch)  # [batch_size, seq_len, 
        output, _ = self.rnn(embeddings)  # [batch_size, seq_len, hidden_d
        if self.ln:
            output = self.layer_norm(output)
        output = self.dropout(self.linear(self.non_lin(output)))  # [batch_
        if self.ln:
            output = self.layer_norm(output)
        projection = self.projection(self.non_lin(output))  # [batch_size, 

        return projection
        # А тут опишите forward pass модели
```

За базовую модель возьмем 1-слойную GRU с размером скрытого слоя hidden_dim = 256, активацией tanh и dropout rate = 0.1 для единственного линейного слоя после rnn

In [40]:
```
model = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embedding_len=
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model.parameters())
```

In [41]:
```
model
```

Out[41]:
```
LanguageModel(
  (embedding): Embedding(40004, 128)
  (rnn): GRU(128, 256, batch_first=True)
  (layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (batch_norm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
  (linear): Linear(in_features=256, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
```

```
  (dropout): Dropout(p=0.1, inplace=False)
)
```

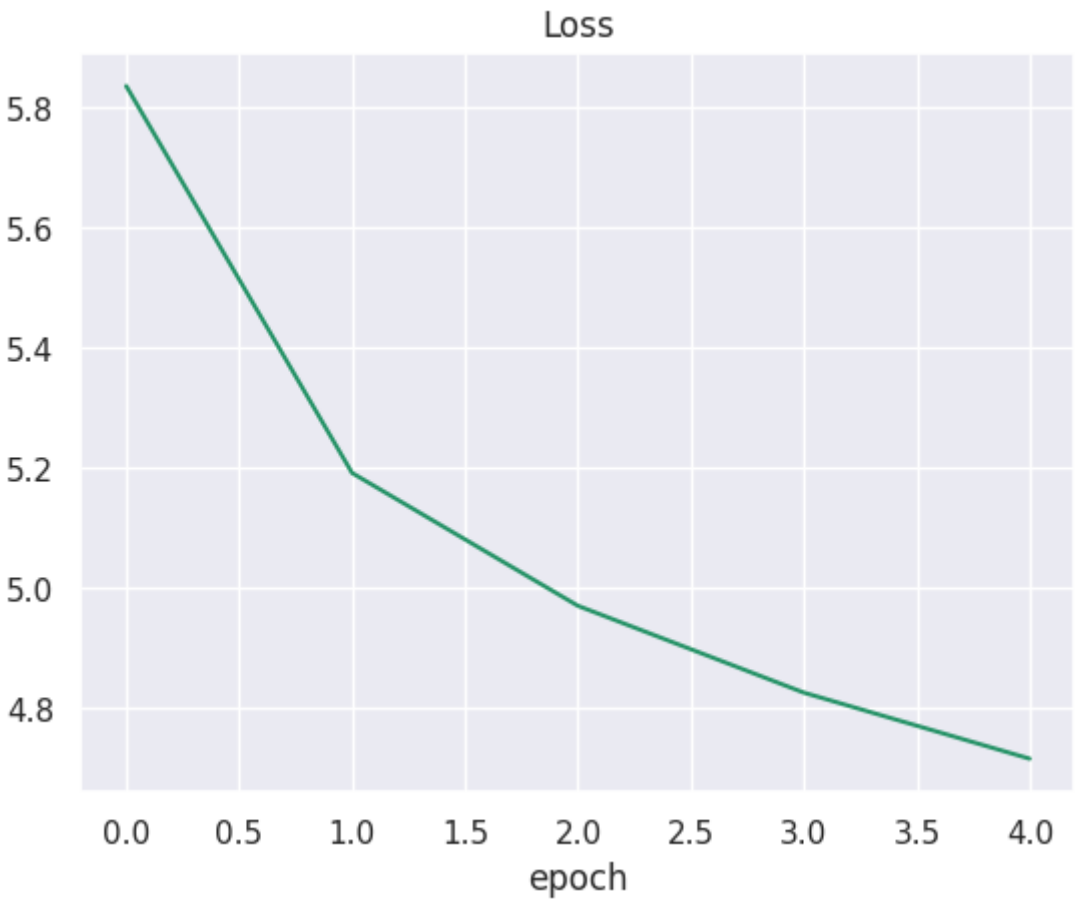
In [42]:

```
%%time

history = train_model(model, criterion=criterion, optimizer=optimizer,
                      train_dataloader=train_dataloader, val_dataloader=eva
                      epochs=5)

# Обучите модель здесь
```

```
CPU times: user 20min 14s, sys: 4.64 s, total: 20min 18s
Wall time: 20min 34s
```

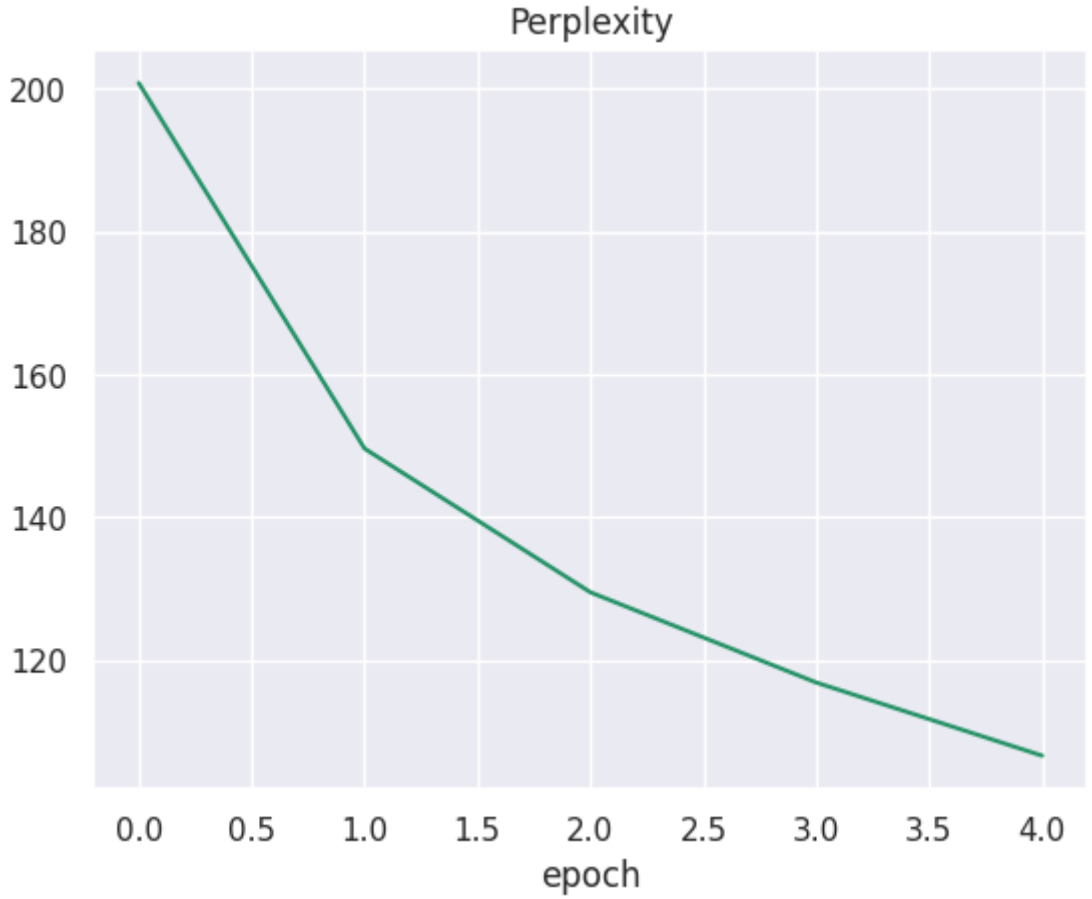
In [47]:

```
plt.plot(np.arange(len(history[0])), history[0])
plt.title('Loss')
plt.xlabel("epoch")
plt.show()
```



In [48]:

```
plt.plot(np.arange(len(history[1])), history[1])
plt.title('Perplexity')
plt.xlabel("epoch")
plt.show()
```



In [49]:
```
perplexity_test = evaluate(model, criterion, test_dataloader)
```

In [50]:
```
print(f'perplexity_test = {np.around(perplexity_test,0)}')
```

```
perplexity_test = 107.0
```

In [51]:
```
out_ids = generate_sequence(model, starting_seq="i want to say")
```

In [52]:
```
print(out_ids)
print([ind2word[out_ids[i].item()] for i in range(0,len(out_ids))])
```

```
tensor([23792,  2849, 13158, 26237, 25880, 36020,  2849,  6429, 39450,  431
0,
          855, 34100,  2849,  6429, 27141,  9392, 26237, 21159, 36267, 1634
0])
['<bos>', 'i', 'want', 'to', 'say', 'that', 'i', 'was', 'a', 'kid', '<unk>'
, 'but', 'i', 'was', 'so', 'excited', 'to', 'see', 'it', '<eos>']
```

In [52]:
```

```

Второй эксперимент (2 балла)

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

Увеличим количество слоев RNN (GRU) с 1 до 3

In [53]:
```
model_Nl = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embedding_
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_Nl.parameters())
```

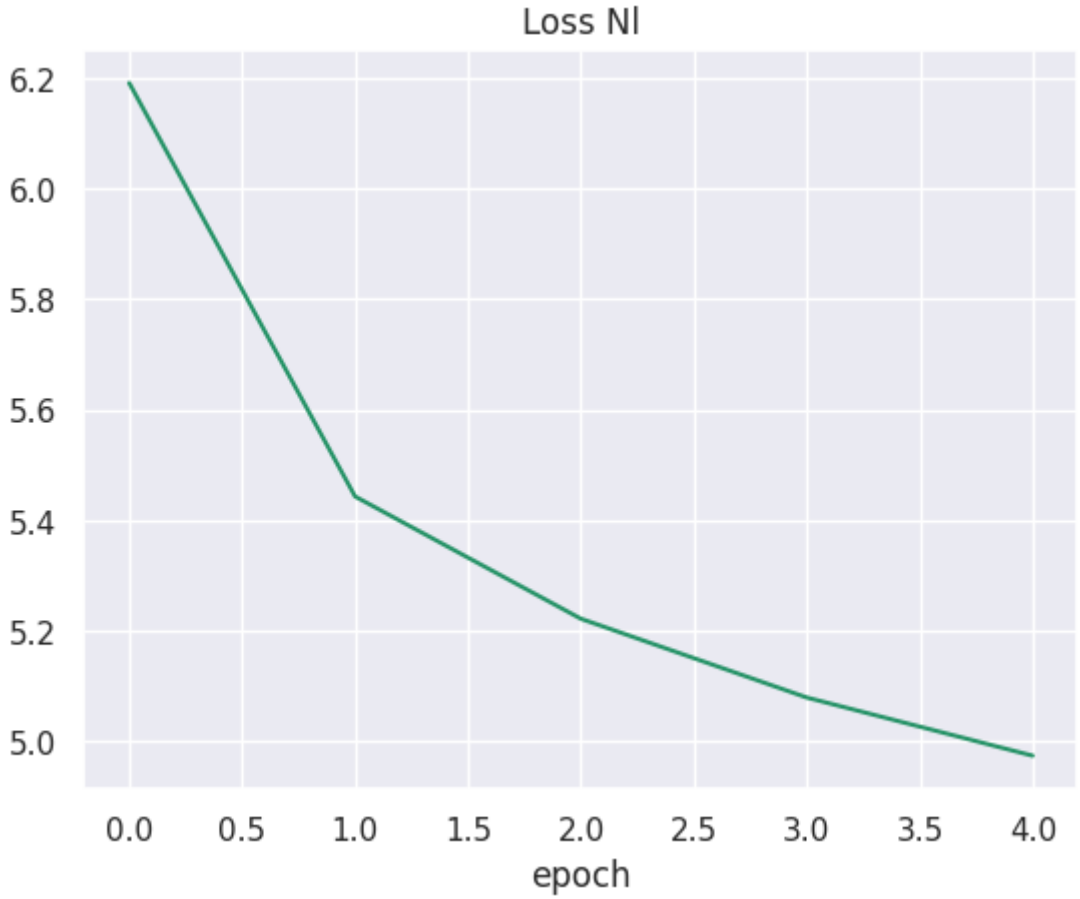
In [54]:
```
model_Nl
```

Out[54]:
```
LanguageModel(
  (embedding): Embedding(40004, 128)
  (rnn): GRU(128, 256, num_layers=3, batch_first=True)
  (layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (batch_norm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
  (linear): Linear(in_features=256, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
  (dropout): Dropout(p=0.1, inplace=False)
)
```

In [55]:
```
%%time

history_Nl = train_model(model_Nl, criterion=criterion, optimizer=optimize
                         train_dataloader=train_dataloader, val_dataloader=eva
                         epochs=5)

# Обучите модель здесь
```

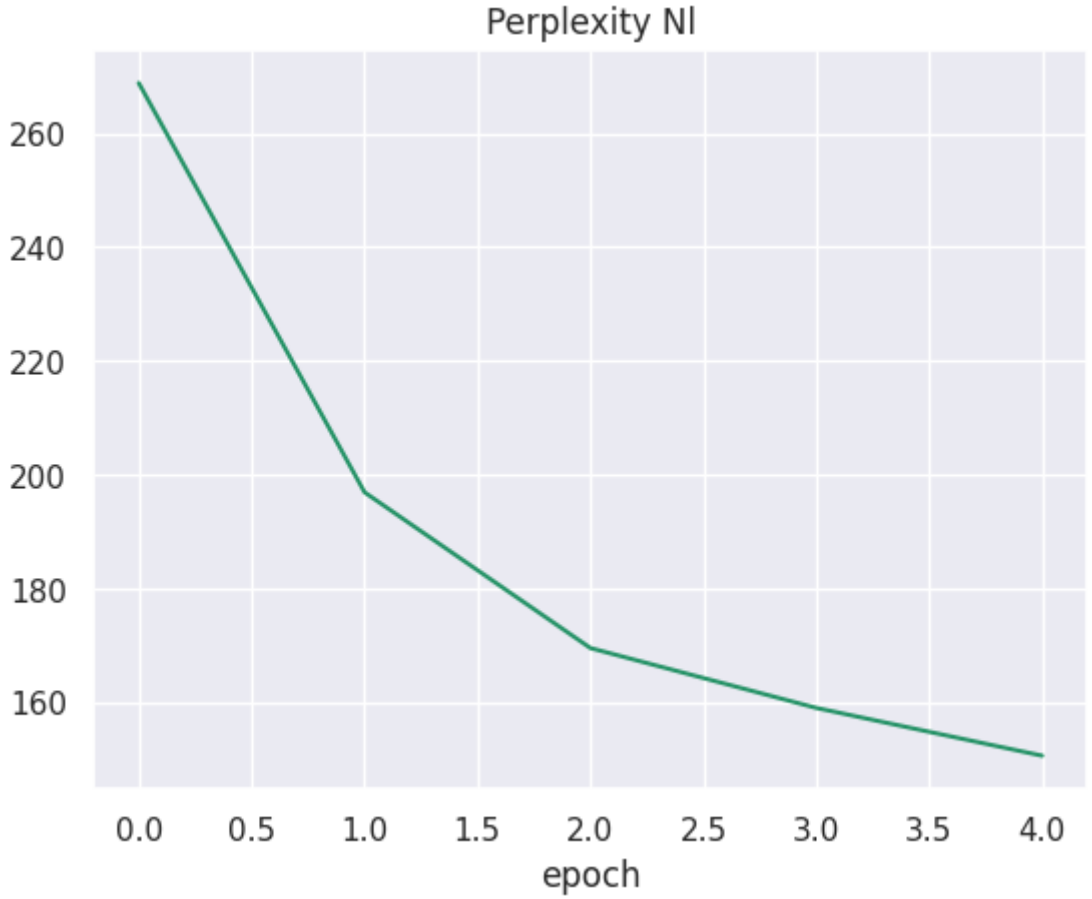
```
CPU times: user 21min 42s, sys: 5.1 s, total: 21min 47s
Wall time: 22min 7s
```

In [56]:
```
plt.plot(np.arange(len(history_Nl[0])), history_Nl[0])
plt.title('Loss Nl')
plt.xlabel("epoch")
plt.show()
```



In [57]:

```
plt.plot(np.arange(len(history_Nl[1])), history_Nl[1])
plt.title('Perplexity Nl')
plt.xlabel("epoch")
plt.show()
```



In [60]:
```
perplexity_test_Nl = evaluate(model_Nl, criterion, test_dataloader)
```

In [61]:
```
print(f'perplexity_test_Nl = {np.around(perplexity_test_Nl,0)}')
```

```
perplexity_test_Nl = 151.0
```

In [62]:
```
out_ids_Nl = generate_sequence(model_Nl, starting_seq="i want to say")
```

In [63]:
```
print(out_ids_Nl)
print([ind2word[out_ids_Nl[i].item()] for i in range(0,len(out_ids_Nl))])
```

```
tensor([23792,  2849, 13158, 26237, 25880, 36020,  2849,  6429, 39450, 1604
3,
        12020, 31021, 37360, 16340])
['<bos>', 'i', 'want', 'to', 'say', 'that', 'i', 'was', 'a', 'fan', 'of', '
the', 'movie', '<eos>']
```

In [ ]:
```
```

Попробуем Biderectional GRU

In [33]:
```
model_bd = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embedding_
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_bd.parameters())
```

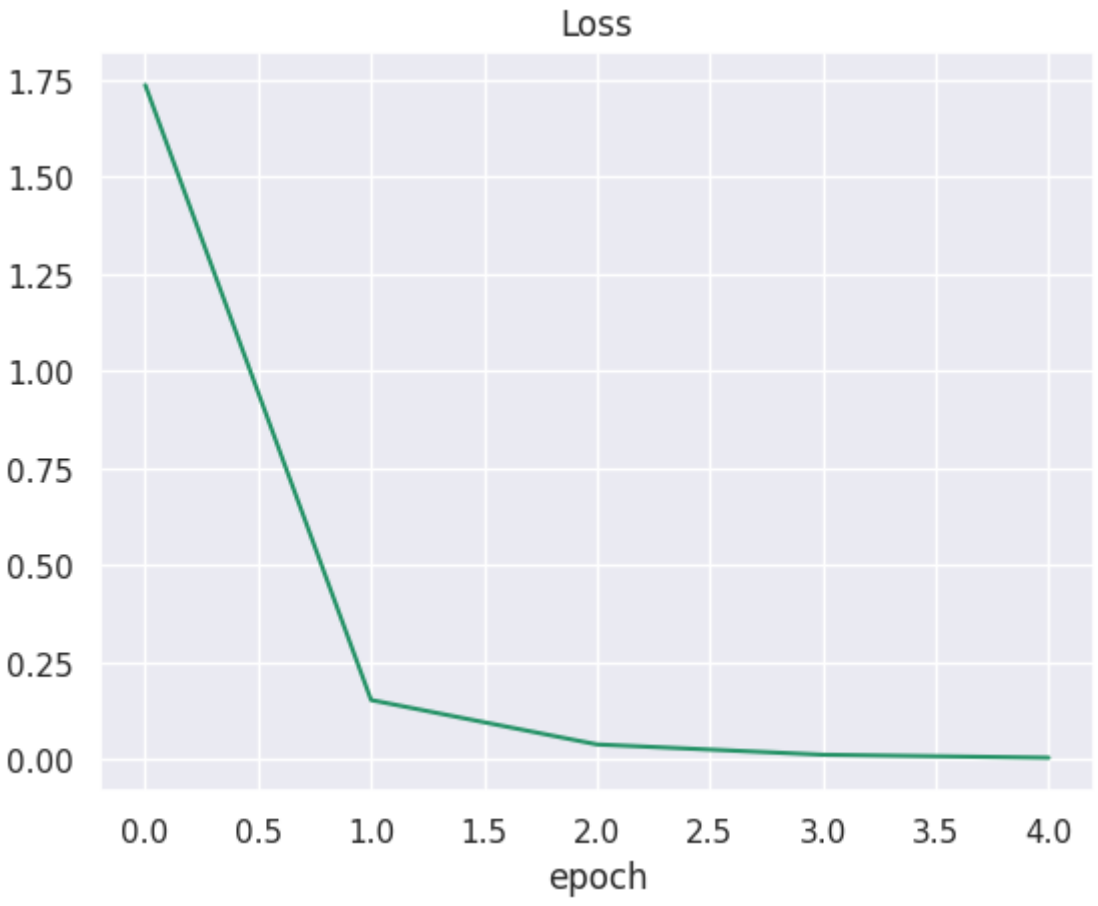
In [34]:
```
model_bd
```

Out[34]:
```
LanguageModel_bd(
  (embedding): Embedding(40004, 128)
  (rnn): GRU(128, 256, batch_first=True, bidirectional=True)
  (linear): Linear(in_features=512, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
  (dropout): Dropout(p=0.1, inplace=False)
)
```

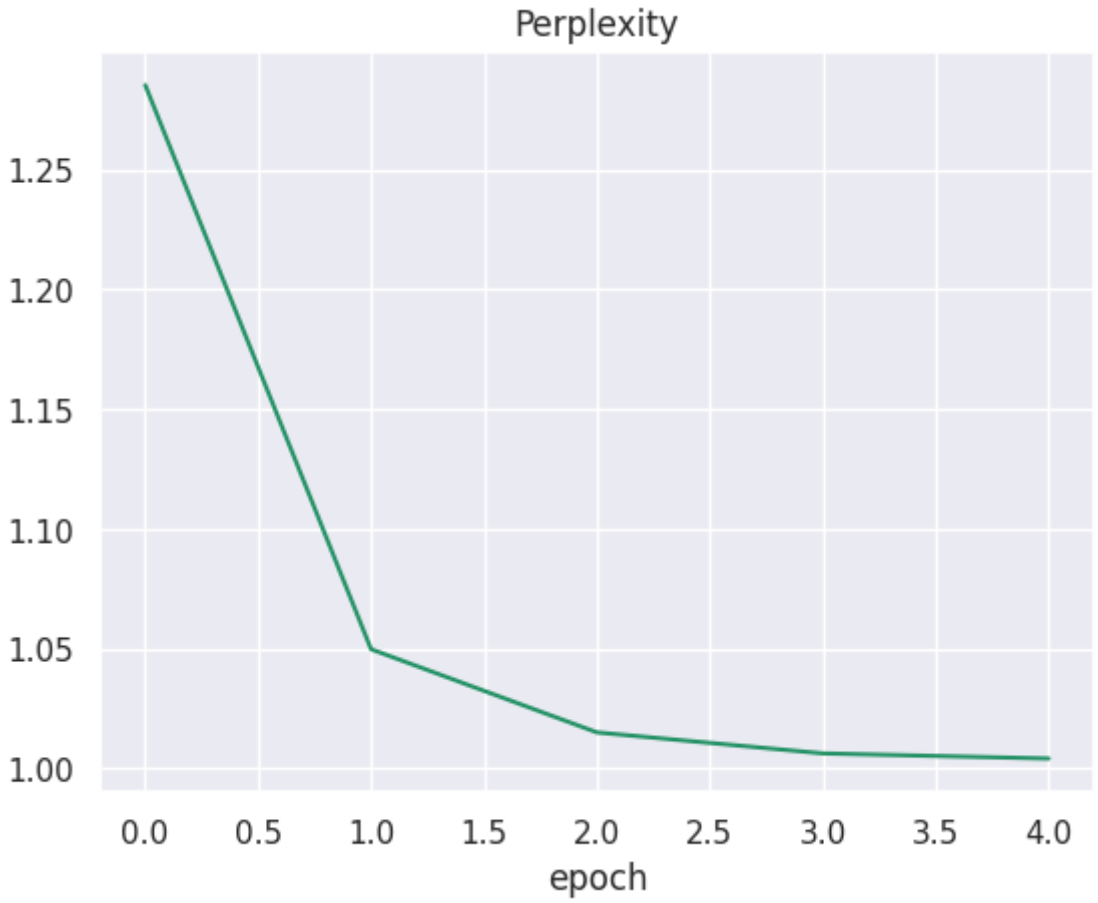
In [35]:
```
%%time

history_bd = train_model(model_bd, criterion=criterion, optimizer=optimize
                         train_dataloader=train_dataloader, val_dataloader=eva
                         epochs=5)

# Обучите модель здесь
```

```
CPU times: user 20min 32s, sys: 5.89 s, total: 20min 38s
Wall time: 21min 3s
```

In [36]:
```
plt.plot(np.arange(len(history_bd[0])), history_bd[0])
plt.title('Loss')
plt.xlabel("epoch")
plt.show()
```



In [37]:
```
plt.plot(np.arange(len(history_bd[1])), history_bd[1])
plt.title('Perplexity')
plt.xlabel("epoch")
plt.show()
```



In [38]:
```
perplexity_test_bd = evaluate(model_bd, criterion, test_dataloader)
```

In [39]:
```
print(f'perplexity_test_bd = {np.around(perplexity_test_bd,0)}')
```

```
perplexity_test_bd = 1.0
```

In [46]:
```
out_ids_bd = generate_sequence(model_bd, starting_seq="i want to say")
```

In [47]:
```
print(out_ids_bd)
print([ind2word[out_ids_bd[i].item()] for i in range(0,len(out_ids_bd))])
```

```
tensor([10633, 32896, 17678, 28436, 39648, 13923])
['<bos>', 'i', 'want', 'to', 'say', '<eos>']
```

In [ ]:

In [31]:

Добавим Layer norm

In [31]:
```
model_ln = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embedding_
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_ln.parameters())
```

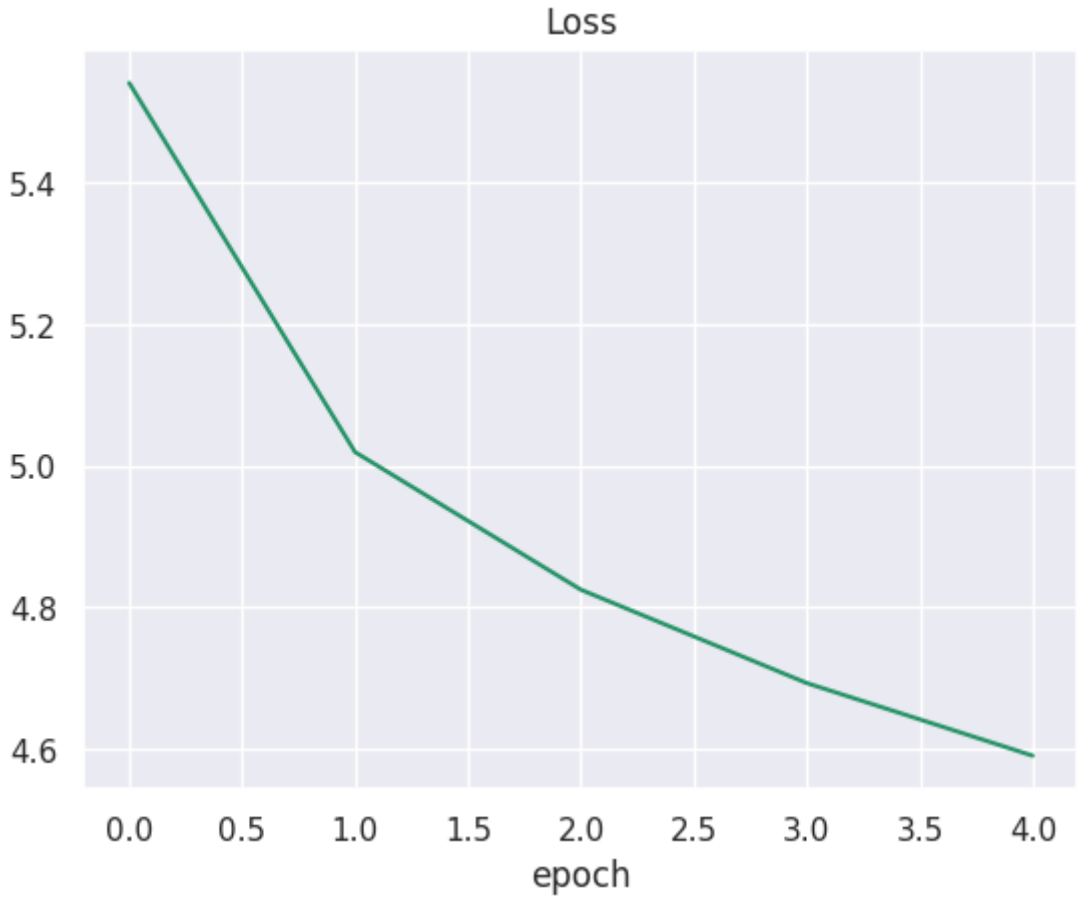
In [32]:
```
model_ln
```

Out[32]:
```
LanguageModel(
  (embedding): Embedding(40004, 128)
  (rnn): GRU(128, 256, batch_first=True)
  (layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (batch_norm): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, trac
k_running_stats=True)
  (linear): Linear(in_features=256, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
  (dropout): Dropout(p=0.1, inplace=False)
)
```

In [33]:
```
%%time

history_ln = train_model(model_ln, criterion=criterion, optimizer=optimize
                         train_dataloader=train_dataloader, val_dataloader=eva
                         epochs=5)

# Обучите модель здесь
```

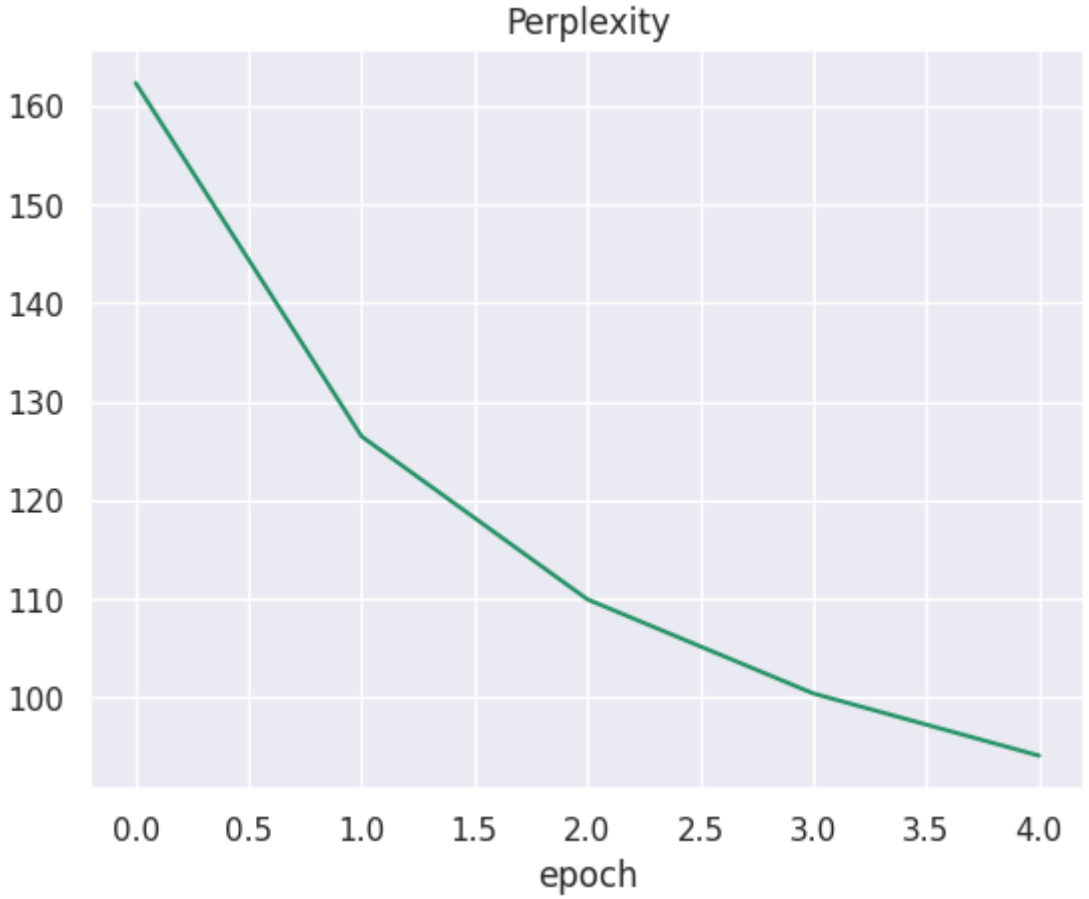
```
CPU times: user 20min 13s, sys: 4.8 s, total: 20min 18s
Wall time: 20min 35s
```

In [34]:
```
plt.plot(np.arange(len(history_ln[0])), history_ln[0])
plt.title('Loss')
plt.xlabel("epoch")
plt.show()
```



In [35]:

```
plt.plot(np.arange(len(history_ln[1])), history_ln[1])
plt.title('Perplexity')
plt.xlabel("epoch")
plt.show()
```



In [36]:
```
perplexity_test_ln = evaluate(model_ln, criterion, test_dataloader)
```

In [37]:
```
print(f'perplexity_test_ln = {np.around(perplexity_test_ln,0)}')
```

```
perplexity_test_ln = 94.0
```

In [38]:
```
out_ids_ln = generate_sequence(model_ln, starting_seq="i want to say")
```

In [39]:
```
print(out_ids_ln)
print([ind2word[out_ids_ln[i].item()] for i in range(0,len(out_ids_ln))])
```

```
tensor([23792,  2849, 13158, 26237, 25880, 36020, 23588, 12752, 39450, 3065
1,
        37360,   855, 34100, 36267, 12752, 39450, 18269, 37360, 16340])
['<bos>', 'i', 'want', 'to', 'say', 'that', 'this', 'is', 'a', 'great', 'mo
vie', '<unk>', 'but', 'it', 'is', 'a', 'good', 'movie', '<eos>']
```

In [ ]:

In [ ]:

Попробуем в базовой модели заменить GRU на LSTM

In [65]:
```
model_LSTM = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embeddin
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_LSTM.parameters())
```

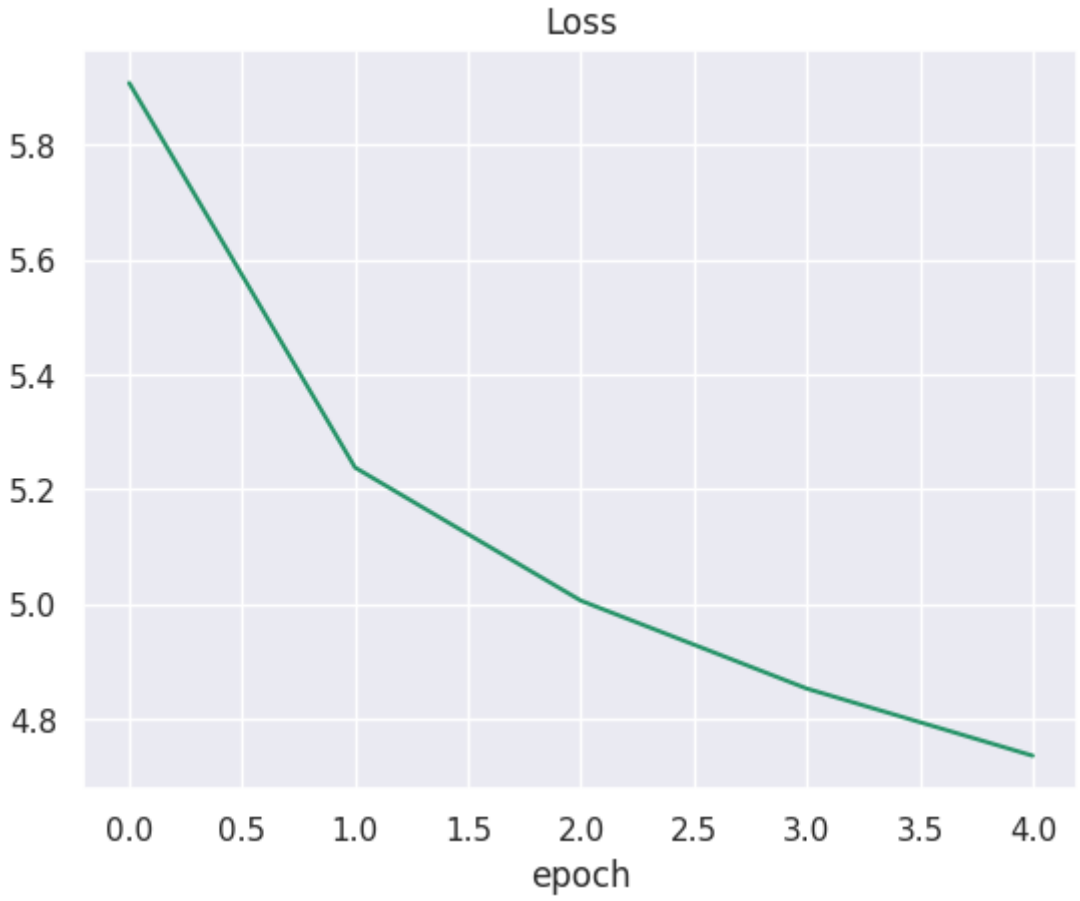
In [66]:
```
model_LSTM
```

Out[66]:
```
LanguageModel(
  (embedding): Embedding(40004, 128)
  (rnn): LSTM(128, 256, batch_first=True)
  (layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (linear): Linear(in_features=256, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
  (dropout): Dropout(p=0.1, inplace=False)
)
```

In [67]:
```
%%time

history_LSTM = train_model(model_LSTM, criterion=criterion, optimizer=optir
                      train_dataloader=train_dataloader, val_dataloader=eva
                      epochs=5)

# Обучите модель здесь
```

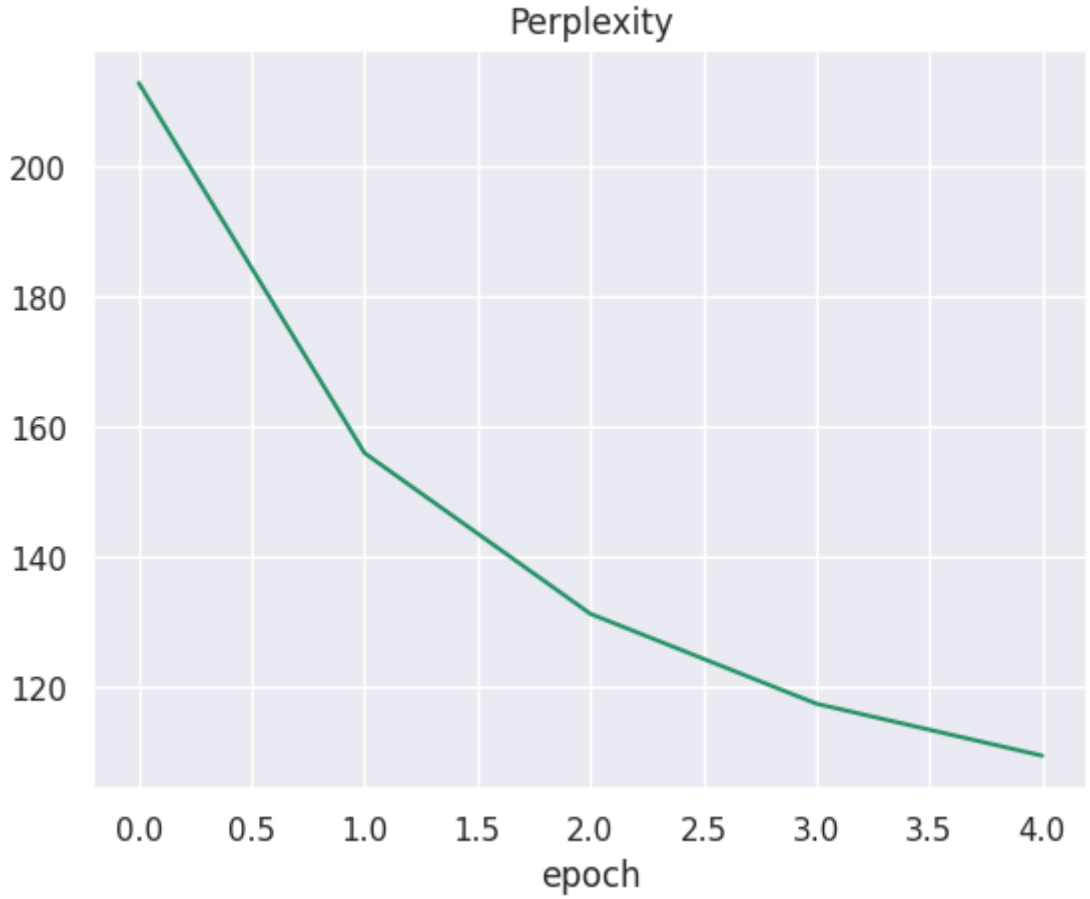
```
CPU times: user 20min 20s, sys: 4.73 s, total: 20min 24s
Wall time: 20min 44s
```

In [68]:
```
plt.plot(np.arange(len(history_LSTM[0])), history_LSTM[0])
plt.title('Loss')
plt.xlabel("epoch")
plt.show()
```



In [69]:

```
plt.plot(np.arange(len(history_LSTM[1])), history_LSTM[1])
plt.title('Perplexity')
plt.xlabel("epoch")
plt.show()
```



In [70]:
```
perplexity_test_LSTM = evaluate(model_LSTM, criterion, test_dataloader)
```

In [71]:
```
print(f'perplexity_test_LSTM = {np.around(perplexity_test_LSTM,0)}')
```

```
perplexity_test_LSTM = 110.0
```

In [72]:
```
out_ids_LSTM = generate_sequence(model_LSTM, starting_seq="i want to say")
```

In [73]:
```
print(out_ids_LSTM)
print([ind2word[out_ids_LSTM[i].item()] for i in range(0,len(out_ids_LSTM)
```

```
tensor([23792,  2849, 13158, 26237, 25880, 36020,  2849,  6429, 20089, 3945
0,
        16043, 12020, 31021, 37360, 16340])
['<bos>', 'i', 'want', 'to', 'say', 'that', 'i', 'was', 'not', 'a', 'fan',
'of', 'the', 'movie', '<eos>']
```

In [73]:
```

```

In [ ]:
```

```

Для конфигурации, показывающей лучшие местрики (с layer norm) увеличим количество эпох обучения до 10

In [31]:
```
model_ln = LanguageModel(hidden_dim=256, vocab_size=len(vocab), embedding_
criterion = nn.CrossEntropyLoss(ignore_index=word2ind['<pad>'])
optimizer = torch.optim.Adam(model_ln.parameters())
```

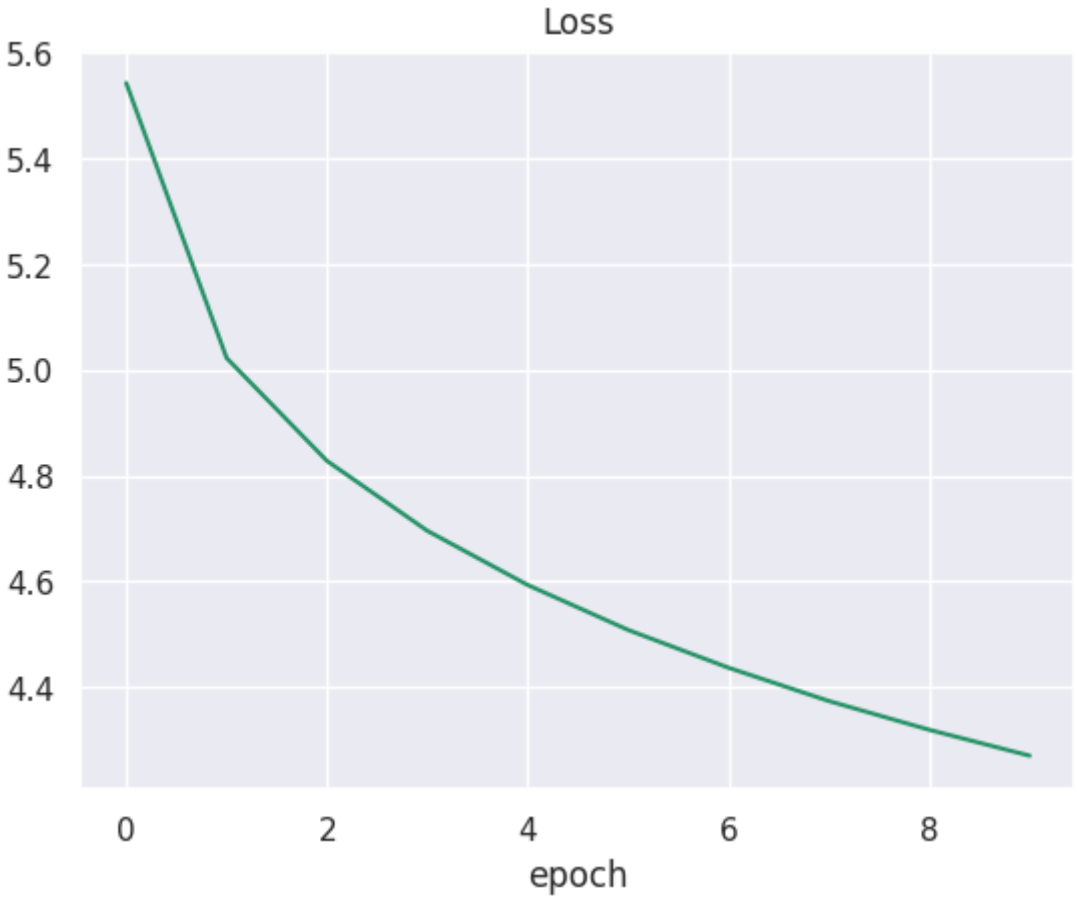
In [32]:
```
model_ln
```

Out[32]:
```
LanguageModel(
  (embedding): Embedding(40004, 128)
  (rnn): GRU(128, 256, batch_first=True)
  (layer_norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (linear): Linear(in_features=256, out_features=256, bias=True)
  (projection): Linear(in_features=256, out_features=40004, bias=True)
  (non_lin): Tanh()
  (dropout): Dropout(p=0.1, inplace=False)
)
```

In [33]:
```
%%time

history_ln_10 = train_model(model_ln, criterion=criterion, optimizer=optim
                      train_dataloader=train_dataloader, val_dataloader=eva
                      epochs=10)

# Обучите модель здесь
```

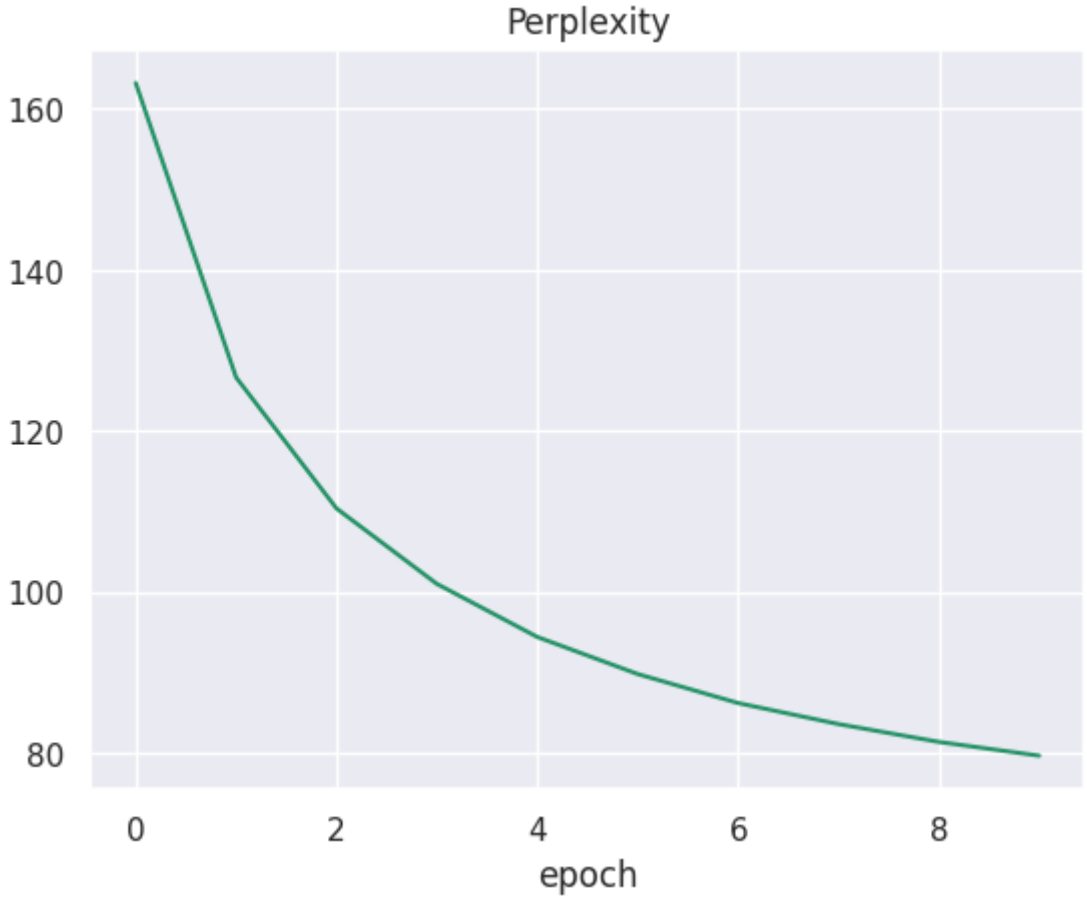
```
CPU times: user 38min 49s, sys: 8.5 s, total: 38min 58s
Wall time: 39min 41s
```

In [34]:
```
plt.plot(np.arange(len(history_ln_10[0])), history_ln_10[0])
plt.title('Loss')
plt.xlabel("epoch")
plt.show()
```



In [35]:

```
plt.plot(np.arange(len(history_ln_10[1])), history_ln_10[1])
plt.title('Perplexity')
plt.xlabel("epoch")
plt.show()
```



In [36]:
```
perplexity_test_ln_10 = evaluate(model_ln, criterion, test_dataloader)
```

In [37]:
```
print(f'perplexity_test_ln_10 = {np.around(perplexity_test_ln_10,0)}')
```

```
perplexity_test_ln_10 = 80.0
```

In [38]:
```
out_ids_ln_10 = generate_sequence(model_ln, starting_seq="i want to say")
```

In [39]:
```
print(out_ids_ln_10)
print([ind2word[out_ids_ln_10[i].item()] for i in range(0,len(out_ids_ln_1(
```

```
tensor([20517, 25222, 34148, 36162, 36654, 32278, 25222, 18763, 11463, 1940
8,
        36162, 19561, 33431, 17741,  5894])
['<bos>', 'i', 'want', 'to', 'say', 'that', 'i', 'was', 'very', 'surprised'
, 'to', 'watch', 'this', 'movie', '<eos>']
```

In [ ]:
```
```

Отчет (2 балла)

Опишите проведенные эксперименты. Сравните перплексии полученных моделей.

Предложите идеи по улучшению качества моделей.

---------- Отчет

Для начала, я слегка изменил базовую предобработку текста:

- Убрал символы разметки br /br / изобилующие в текстах
- Удалил точки, а остальную пунктуацию заменил на пробелы (кроме символов `'-/ которые часто должны содержаться в слове, чтобы верно отразить его смысл)

За базовую модель была взята следующая архитектура:

- Эмбединг длинной 128
- GRU с одним слоем размерности 256
- Линейный слой размерности 256 с вероятностью дропаута 0.1

Эта архитектура показала на тесте perplexity = 107

Далее я попробовал увеличить количество слоев в GRU с 1 до 3. Это дало уменьшение perplexity на тесте до 151. Также я попробовал bidirectional GRU и это дало ужасный результат. Модель всегда предсказывает конец предложения в силу особенностей архитектуры. Проехали.

Хорошо показало себя добавление Layer Norm: perplexity на тесте уменьшилась до 94.

Замена GRU на LSTM не дала улучшений: базовая модель, где вместо GRU взята LSTM дала perplexity = 110.

Таким образом, самой удачной моделью оказалась GRU с Layer Norm. Для этой модели я провел 10 эпох обучения вместо 5 как раньше (это дало улучшение так как на валидации perplexity с ростом количества эпох только улучшалось). на тесте для такой модели с 10 эпохами perplexity = 80.

Можно было бы еще поработать над предобработкой текста, а также попробовать более сложные архитектуры / большое количество эпох, но так как обучение нынешней архитектуры на 10 эпохах занимает 40 минут дальнейшие эксперименты слишком затратны по времени.

In [ ]: