Введение в обработку естественного языка

Урок 9. Языковое моделирование

Практическое задание

Домашнее задание к уроку 9

Задание
Разобраться с моделькой генерации текста, собрать самим или взять датасет с вебинара и обучить генератор текстов

Выполнил ***Соковнин ИЛ***

В [1]:
```
import tensorflow as tf

import numpy as np
import os
import time
```

В [2]:
```
import pandas as pd
import numpy as np
import re
```

В [3]:
```
!mkdir data
```

```
mkdir: cannot create directory ‘data’: File exists
```

В [4]:
```
from google.colab import files

upload = files.upload()
!mv 'evgenyi_onegin.txt' 'data/evgenyi_onegin.txt'
```

Выбрать файлы Файл не выбран

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving evgenyi_onegin.txt to evgenyi_onegin.txt
```

В [5]:
```
!ls data/*.txt
```

```
data/evgenyi_onegin.txt
```

В [6]:
```
path_to_file = '/content/data/evgenyi_onegin.txt'
```

В [7]:
```
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')

# length of text is the number of characters in it
print('Length of text: {} characters'.format(len(text)))
```

```
Length of text: 286984 characters
```

В [8]:
```
print(text[:500])
```

```
Александр Сергеевич Пушкин

                                Евгений Онегин
                                Роман в стихах

                        Не мысля гордый свет забавить,
                        Вниманье дружбы возлюбя,
                        Хотел бы я тебе представить
                        Залог достойнее тебя,
                        Достойнее души прекрасной,
                        Святой исполненной мечты,
                        Поэзии живой и ясной,
                        Высо
```

В [9]:
```
# text = text + text
```

В [10]:
```
text = text.split('\n\n')
```

In [11]:
```
text[0:5]
```

Out[11]:
```
['Александр Сергеевич Пушкин',
 '                                Евгений Онегин\n                                  Роман в стихах',
 '                        Не мысля гордый свет забавить,\n                          Вниманье дружбы возлюбя,\n
Хотел бы я тебе представить\n                          Залог достойнее тебя,\n                          Достойнее души прек
расной,\n                        Святой исполненной мечты,\n                        Поэзии живой и ясной,\n
Высоких дум и простоты;\n                          Но так и быть - рукой пристрастной\n                          Прими собр
анье пестрых глав,\n                          Полусмешных, полупечальных,\n                          Простонародных, идеаль
ных,\n                        Небрежный плод моих забав,\n                          Бессонниц, легких вдохновений,\n
Незрелых и увядших лет,\n                          Ума холодных наблюдений\n                          И сердца горестных за
мет.',
 '                                ГЛАВА ПЕРВАЯ',
 '                                  И жить торопится и чувствовать спешит.\n
Кн. Вяземский.']
```

In [12]:
```
len(text)
```

Out[12]:
```
782
```

In [13]:
```
text_only = []

for line in text:
    if len(line) < 350:
        continue
    else:
        text_only.append(line)

len(text_only)
```

Out[13]:
```
376
```

In [14]:
```
text_only[1]
```

Out[14]:
```
'                        "Мой дядя самых честных правил,\n                          Когда не в шутку занемог,\n
Он уважать себя заставил\n                          И лучше выдумать не мог.\n                          Его пример другим н
аука;\n                        Но, боже мой, какая скука\n                          С больным сидеть и день и ночь,\n
Не отходя ни шагу прочь!\n                          Какое низкое коварство\n                          Полуживого забавлят
ь,\n                        Ему подушки поправлять,\n                          Печально подносить лекарство,\n
Вздыхать и думать про себя:\n                          Когда же черт возьмет тебя!"'
```

In [15]:
```
data = pd.DataFrame(text_only)
data = data.rename(columns={0: "text"})
data.head(3)
```

Out[15]:

	text
0	Не мысля гордый свет з...
1	"Мой дядя самых честны...
2	Так думал молодой пове...

In [16]:
```
def exclude_punctuation(txt):
    txt = "".join(c for c in txt if c not in exclude)
    txt = re.sub("\n", " \n", txt)
    return txt
```

In [17]:
```
# data_1 = data
```

In [18]:
```
# !pip install pymorphy2
!pip install stop-words
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/s
imple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: stop-words in /usr/local/lib/python3.7/dist-packages (2018.7.23)
```

In [19]:
```
from string import punctuation
from stop_words import get_stop_words

exclude = set(punctuation)
sw = set(get_stop_words("ru"))
```

In [20]:
```
data['text_splited'] = data['text'].apply(exclude_punctuation)
```

Type *Markdown* and LaTeX: α^2

B [21]:
```
data.head(3)
```

Out[21]:

	text	text_splited
0	Не мысля гордый свет з...	Не мысля гордый свет з...
1	"Мой дядя самых честны...	Мой дядя самых честных...
2	Так думал молодой пове...	Так думал молодой пове...

B [22]:
```
def preprocess_text(txt):
    txt = str(txt)
    txt = txt.lower()
    txt = re.sub("\n", "aaa", txt)
    new_txt =[]
    for word in txt.split():
        if word == "aaa":
            word = " \n"
        new_txt.append(word)

    return new_txt
```

B [23]:
```
data['text_splited'] = data['text_splited'].apply(preprocess_text)
```

B [24]:
```
data
```

Out[24]:

	text	text_splited
0	Не мысля гордый свет з...	[не, мысля, гордый, свет, забавить, \n, внима...
1	"Мой дядя самых честны...	[мой, дядя, самых, честных, правил, \n, когда...
2	Так думал молодой пове...	[так, думал, молодой, повеса, \n, летя, в, пы...
3	Служив отлично благоро...	[служив, отлично, благородно, \n, долгами, жи...
4	Когда же юности мятежн...	[когда, же, юности, мятежной, \n, пришла, евг...
...	...	...
371	А счастье было так воз...	[а, счастье, было, так, возможно, \n, так, бл...
372	Она ушла. Стоит Евгени...	[она, ушла, стоит, евгений, \n, как, будто, г...
373	Кто б ни был ты, о мой...	[кто, б, ни, был, ты, о, мой, читатель, \n, д...
374	Прости ж и ты, мой спу...	[прости, ж, и, ты, мой, спутник, странный, \n...
375	Но те, которым в дружн...	[но, те, которым, в, дружной, встрече, \n, я,...

376 rows × 2 columns

B [25]:
```
# Создадим словарь наших текстов
vocab = []
for ts in data.text_splited:
    for t in ts:
        vocab.append(t)

# dictionary = [ t for ts in df.text_splited for t in ts ]

vocab[:5]
```

Out[25]:
```
['не', 'мысля', 'гордый', 'свет', 'забавить']
```

B [26]:
```
len(vocab)
```

Out[26]:
```
27414
```

B [27]:
```
from collections import Counter

# Подсчитать частоту слов в списке и отсортировать по частоте
counts = Counter(vocab)
# counts.items()
print(dict(list(counts.items())[:5]))
```

```
{'не': 384, 'мысля': 1, 'гордый': 4, 'свет': 17, 'забавить': 1}
```

B [28]:
```
# Сортировка по частоте
sorted_counts = sorted(counts.items(), key=lambda item: (-item[1]))
sorted_counts[:5]
```

Out[28]:
```
[(' \n', 4917), ('и', 1005), ('в', 601), ('не', 384), ('он', 294)]
```

B [29]:
```
# Частотный словарь
freq_dictionary = list(tp[0] for tp in sorted_counts)
freq_dictionary[:10]
```

Out[29]:
```
[' \n', 'и', 'в', 'не', 'он', 'на', 'с', 'я', 'но', 'как']
```

B [30]:
```
def get_w2i_i2w(column_data):

    dump = list(column_data.values)
    dump_txt_split = []
    for sublist in dump:
        for item in sublist:
            dump_txt_split.append(item)


    vocab = sorted(set(dump_txt_split))

    # Creating a mapping from unique characters to indices
    word2idx = {u:i for i, u in enumerate(vocab)}
    idx2word = np.array(vocab)

    print(len(vocab), len(word2idx), len(idx2word))
    return word2idx, idx2word
```

B [31]:
```
# word2idx = {u:i for i, u in enumerate(vocab)}
# idx2word = np.array(vocab)

word2idx, idx2word = get_w2i_i2w(data['text_splited'])
data['int_text_splited'] = data['text_splited'].apply(lambda x: [word2idx[c] for c in x])
```

```
8427 8427 8427
```

B [32]:
```
# word2idx
```

B [33]:
```
idx2word
```

Out[33]:
```
array([' \n', '1', '10', ..., 'ясные', 'ясным', 'ящик'], dtype='<U20')
```

B [34]:
```
data.head(3)
```

Out[34]:

	text	text_splited	int_text_splited
0	Не мысля гордый свет з...	[не, мысля, гордый, свет, забавить, \n, внима...	[3817, 3634, 1358, 6327, 2071, 0, 878, 1844, 9...
1	"Мой дядя самых честны...	[мой, дядя, самых, честных, правил, \n, когда...	[3487, 1912, 6292, 8129, 5394, 0, 2788, 3817, ...
2	Так думал молодой пове...	[так, думал, молодой, повеса, \n, летя, в, пы...	[7249, 1867, 3505, 4930, 0, 3102, 565, 5872, 3...

B [35]:
```
def get_all_int(column_data):

    values = column_data.values
    text_as_int = []
    print(values[0])
    for sublist in values:
        for item in sublist:
            text_as_int.append(item)
    text_as_int = np.array(text_as_int)

    return text_as_int

text_as_int = get_all_int(data['int_text_splited'])
```

```
[3817, 3634, 1358, 6327, 2071, 0, 878, 1844, 923, 0, 7991, 542, 8375, 7313, 5450, 0, 2181, 1782, 7314, 0, 1782, 1893, 5
478, 0, 6397, 2589, 3389, 0, 5373, 2028, 2446, 8420, 0, 1187, 1863, 2446, 5769, 0, 4072, 7249, 2446, 560, 6220, 5630,
0, 5591, 6781, 4774, 1252, 0, 5169, 5165, 0, 5764, 2463, 0, 3836, 4891, 3486, 2069, 0, 295, 3039, 611, 0, 3928, 2446, 7
604, 3084, 0, 7706, 7980, 3645, 0, 2446, 6471, 1364, 2194]
```

B [36]:
```
# text_as_int
```

B [37]:
```
import numpy as np

unique, counts = np.unique(text_as_int, return_counts=True)

result = np.column_stack((unique, counts))
print (result)
```

```
[[   0 4917]
 [   1    1]
 [   2    1]
 ...
 [8424    1]
 [8425    1]
 [8426    2]]
```

train and target

B [38]:
```
# The maximum length sentence you want for a single input in characters
seq_length = 50
examples_per_epoch = len(text_as_int)//(seq_length+1)

# Create training examples / targets
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

for i in char_dataset.take(10):
    print(idx2word[i.numpy()])
```

```
не
мысля
гордый
свет
забавить


вниманье
дружбы
возлюбя
```

B [39]:
```
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)
sequences
```

Out[39]:
```
<BatchDataset element_spec=TensorSpec(shape=(51,), dtype=tf.int64, name=None)>
```

B [40]:
```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)
```

Print the first example input and target values:

B [41]:
```
for input_example, target_example in  dataset.take(1):
    print('Input data: ', repr(' '.join(idx2word[input_example.numpy()])))
    print('Target data:', repr(' '.join(idx2word[target_example.numpy()])))
```

```
Input data:  'не мысля гордый свет забавить  \n вниманье дружбы возлюбя  \n хотел бы я тебе представить  \n залог досто
йнее тебя  \n достойнее души прекрасной  \n святой исполненной мечты  \n поэзии живой и ясной  \n высоких дум и простот
ы  \n но так и быть рукой пристрастной  \n прими собранье пестрых глав  \n'
Target data: 'мысля гордый свет забавить  \n вниманье дружбы возлюбя  \n хотел бы я тебе представить  \n залог достойне
е тебя  \n достойнее души прекрасной  \n святой исполненной мечты  \n поэзии живой и ясной  \n высоких дум и простоты
\n но так и быть рукой пристрастной  \n прими собранье пестрых глав  \n полусмешных'
```

B [42]:
```
# Batch size
BATCH_SIZE = 64

# Buffer size to shuffle the dataset
BUFFER_SIZE = 10000

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)

dataset
```

Out[42]:
```
<BatchDataset element_spec=(TensorSpec(shape=(64, 50), dtype=tf.int64, name=None), TensorSpec(shape=(64, 50), dtype=tf.
int64, name=None))>
```

B [43]:
```
# Length of the vocabulary in chars
vocab_size = len(idx2word)

# The embedding dimension
embedding_dim = 128

# Number of RNN units
rnn_units = 1024
```

!!!

B [44]:
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim),

        tf.keras.layers.GRU(rnn_units,
                            return_sequences=True,
                            stateful=False,
                            recurrent_initializer='glorot_uniform'),

        tf.keras.layers.GRU(rnn_units,
                            return_sequences=True,
                            stateful=False,
                            recurrent_initializer='glorot_uniform'),

         tf.keras.layers.GRU(rnn_units,
                            return_sequences=True,
                            stateful=False,
                            recurrent_initializer='glorot_uniform'),

        tf.keras.layers.Dense(vocab_size)
    ])
    return model
```

B [45]:
```
# def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
#     model = tf.keras.Sequential([
#         tf.keras.layers.Embedding(vocab_size, embedding_dim,
#                                   batch_input_shape=[batch_size, None]),

#         tf.keras.layers.LSTM(rnn_units,
#                              return_sequences=True,
#                              stateful=True,
#                              recurrent_initializer='glorot_uniform'),

#         tf.keras.layers.LSTM(rnn_units,
#                              return_sequences=True,
#                              stateful=True,
#                              recurrent_initializer='glorot_uniform'),

#          tf.keras.layers.LSTM(rnn_units,
#                              return_sequences=True,
#                              stateful=True,
#                              recurrent_initializer='glorot_uniform'),

#         tf.keras.layers.LSTM(rnn_units,
#                              return_sequences=True,
#                              stateful=True,
#                              recurrent_initializer='glorot_uniform'),

#         tf.keras.layers.Dense(vocab_size)
#     ])
#     return model
```

B [46]:
```
model = build_model(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    rnn_units=rnn_units,
    batch_size=BATCH_SIZE)
```

B [47]:
```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")
```

```
(64, 50, 8427) # (batch_size, sequence_length, vocab_size)
```

B [48]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         1078656

 gru (GRU)                   (None, None, 1024)        3545088

 gru_1 (GRU)                 (None, None, 1024)        6297600

 gru_2 (GRU)                 (None, None, 1024)        6297600

 dense (Dense)               (None, None, 8427)        8637675

=================================================================
Total params: 25,856,619
Trainable params: 25,856,619
Non-trainable params: 0
_________________________________________________________________
```

B [49]:
```
example_batch_predictions[0]
```

Out[49]:
```
<tf.Tensor: shape=(50, 8427), dtype=float32, numpy=
array([[-2.11247519e-04, -4.92446648e-04,  1.53959656e-04, ...,
        -7.24026395e-05, -2.15041538e-04,  1.75769746e-04],
       [-4.27933701e-04, -8.09994410e-04, -5.78693798e-05, ...,
         1.11310386e-04, -4.08001710e-04,  2.35781772e-04],
       [-4.45326092e-04, -8.05551419e-04, -3.49194192e-06, ...,
         1.89978731e-04, -7.25351856e-04, -3.49327136e-04],
       ...,
       [-2.21765353e-04, -9.09685856e-04, -3.43999738e-04, ...,
         1.53568003e-03,  2.62600050e-04,  5.71347598e-04],
       [-4.51837492e-04, -5.76044898e-04, -1.95283865e-04, ...,
         1.34727766e-03, -2.02961077e-04,  1.66701051e-04],
       [-3.03942390e-04, -2.87522125e-04,  1.35594819e-04, ...,
         9.28442751e-04, -4.94893815e-04, -3.98775272e-04]], dtype=float32)>
```

B [50]:
```
sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()
sampled_indices
```

Out[50]:
```
array([7738, 2063,  693, 4260, 2636, 6058, 2111,  280, 1174,  713, 8237,
        463, 2102, 6057, 4179, 2839, 1449, 6100, 7990, 5563, 6892,  717,
       4712, 1182, 1925,  741, 5725, 3832, 2895, 6874, 6402, 5736, 5040,
       7625, 4757, 1546, 7383, 7192, 1578, 6192, 5808,  768,  742, 4139,
       3417, 5283, 2797, 6167,  674, 8394])
```

B [51]:
```
# print(idx2word[input_example_batch[0]])
print("Input: \n", repr(" ".join(idx2word[input_example_batch[0]])))
print()
# print("Next Word Predictions: \n", repr("".join(idx2word[8426])))
print("Next Word Predictions: \n", repr(" ".join(idx2word[sampled_indices])))
```

```
Input:
 'благословляя колеи  \n и рвы отеческой земли зато зимы порой холодной  \n езда приятна и легка  \n как стих без мысли
в песне модной  \n дорога зимняя гладка  \n автомедоны наши бойки  \n неутомимы наши тройки  \n и версты теша праздный
взор  \n в глазах мелькают как забор 43  \n'

Next Word Predictions:
 'умы журнальной веселились огнем казаться растворила завеса бесконечный выражает весть шалунью брака забыт рассыпался
обман кончину гроб резвилась хоры признаться сплетен ветвей первых высказывал единой вечное пронзенный небреженьем крас
ногорья сошел сгустились просит поедем угрюм перстами дать тогосего счастию де роняет пружина вздохи вечности няни мину
ту посох кой рождающийся верная ягоды'
```

Train the model

B [52]:
```
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

example_batch_loss = loss(target_example_batch, example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size, sequence_length, vocab_size)")
print("scalar_loss:      ", example_batch_loss.numpy().mean())
```

```
Prediction shape:  (64, 50, 8427)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       9.039164
```

B [53]:
```
model.compile(optimizer='adam', loss=loss)
```

Configure checkpoints

B [54]:
```
# Directory where the checkpoints will be saved
checkpoint_dir = './training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    # save_freq=88*5,
    period=20,
    save_weights_only=True)
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batche
s seen.
```

Execute the training

B [55]:
```
EPOCHS = 200
```

B [56]:
```
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
8/8 [==============================] - 1s 151ms/step - loss: 4.7484
Epoch 67/200
8/8 [==============================] - 1s 151ms/step - loss: 4.7002
Epoch 68/200
8/8 [==============================] - 1s 150ms/step - loss: 4.6333
Epoch 69/200
8/8 [==============================] - 1s 151ms/step - loss: 4.5914
Epoch 70/200
8/8 [==============================] - 1s 152ms/step - loss: 4.5452
Epoch 71/200
8/8 [==============================] - 1s 151ms/step - loss: 4.4753
Epoch 72/200
8/8 [==============================] - 1s 151ms/step - loss: 4.4698
Epoch 73/200
8/8 [==============================] - 1s 152ms/step - loss: 4.4341
Epoch 74/200
8/8 [==============================] - 1s 151ms/step - loss: 4.3743
Epoch 75/200
8/8 [==============================] - 1s 151ms/step - loss: 4.3233
```

Generate text

B [57]:
```
tf.train.latest_checkpoint(checkpoint_dir)
```

Out[57]:
```
'./training_checkpoints/ckpt_200'
```

B [58]:
```
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
model.build(tf.TensorShape([1, None]))
```

B [59]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, None, 128)         1078656

 gru_3 (GRU)                  (None, None, 1024)        3545088

 gru_4 (GRU)                  (None, None, 1024)        6297600

 gru_5 (GRU)                  (None, None, 1024)        6297600

 dense_1 (Dense)              (None, None, 8427)        8637675

=================================================================
Total params: 25,856,619
Trainable params: 25,856,619
Non-trainable params: 0
_________________________________________________________________
```

B [60]:
```
def generate_text(model, start_string, tmprt):
    # Evaluation step (generating text using the learned model)

    start_string = exclude_punctuation(start_string)
    #print(start_string)
    start_string_asis = preprocess_text(start_string)

    # Number of characters to generate
    num_generate = 30

    # Converting our start string to numbers (vectorizing)
    input_eval = [word2idx[s] for s in start_string_asis]
    input_eval = tf.expand_dims(input_eval, 0)

    # Empty string to store our results
    text_generated = []

    # Low temperature results in more predictable text.
    # Higher temperature results in more surprising text.
    # Experiment to find the best setting.
    # temperature = 0.1
    temperature = tmprt

    # Here batch size == 1
    model.reset_states()
    for i in range(num_generate):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)
        # using a categorical distribution to predict the character returned by the model
        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()

        # Pass the predicted character as the next input to the model
        # along with the previous hidden state
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2word[predicted_id])

    return (start_string + ' '.join(text_generated))
```

B [61]:
```
print(generate_text(model, start_string=u"зима ", tmprt=1.3))
```

```
зима крыльца мне душой взором слушал влюбилась неожиданный молвы
 девичьих вздор
 поскачет русские свеч вздор то крестом несмелой
 свободной пробуждена дев чистой уж темно полон ночь стократ свобода
```

B [62]:
```
start_string= 'Холодных чистых как зима '
```

B [63]:
```
print(generate_text(model, start_string=start_string, tmprt=1))
```

```
Холодных чистых как зима то журнальной ее застыла
 вражда шевельнулась
 на потопленные цельным пред хочу в неделю русские свеч крестом полей другим радужным benedetta та мог ее сменил искрен
ней тайны тайны напомнил
```

B [68]:
```
print(generate_text(model, start_string=start_string, tmprt=0.5))
```

```
Холодных чистых как зима
 не имея
 и средь полей другим радужным benedetta


 и тайну




 и злость русские свеч вздор


 но
```

B [65]:
```
print(generate_text(model, start_string=start_string, tmprt=5))
```

```
Холодных чистых как зима поэтам прелестной сидят ревнивые попав встает поколенья воскресить уединенье москве умен забуд
ешь героиной тюрьмы мною ты много мыслил полукруг враки речка жертвой язвительному одушевленные беготня столпы зимой ош
ибкой бутылкой страстный
```

B [65]: