МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по индивидуальному домашнему заданию
по дисциплине «Искусственные нейронные сети»
Тема: Генерация шуток по началу фразы

Студенты гр. 8383 _____ Шишкин И.В.
Степанов В.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель работы.

В рамках выполнения ИДЗ необходимо разработать и реализовать модель ИНС решающую определенную задачу на заданном датасете. Short jokes - датасет текста. Включает в себя 231657 коротких шуток на английском. Задача заключается в генерации шуток по началу фразы.

Требования

Требования к разрабатываемой модели:

- Модель должна быть разработана на языке Python с использованием Keras API
- Исходный код проекта должен быть в формате PEP8
- В исходном коде должны быть поясняющие комментарии
- Модель не должна быть избыточной (должен соблюдаться баланс между размером сети [кол-во слоев и нейронов] и качеством выдаваемого результата)
- Обучение модели должно быть стабильно (для предложенной архитектуры ИНС обучение должно приводить к примерно одним и тем же результатом, то есть не должно быть такого, что при обучении 10 сетей удовлетворительный результат дают только 5 из них)
- Плюсом будет анализ с использованием callback’а TensorBoard
- Плюсом будет разработка собственных callback’ов
- Плюсом будет создание модели из ансамбля ИНС

Требования к отчету:

- В отчете должно быть описание датасета, а также описание решаемой задачи
- В отчете должен быть проведен начальный анализ данных. Проведение статистического анализа, анализ на необходимость

нормировки данных, обоснование применения определенного вида нормировки, и.т.д.

- В отчете необходимо отразить весь процесс разработки: с чего началась разработка модели, на основании чего проводились те или иные изменения/корректировки, обоснование выбора тех или иных изменений/корректировок. По сути выполнение должно быть разбито на итерации, каждая итерация должна сопровождаться результатами модели на итерации, а также краткой выдержкой исходного кода, показывающая изменения на итерации
- В конце отчета должен быть приведен анализ результирующей модели, а также перечислены возникшие проблемы (и как они были решены) и проблемы, которые решить не удалось. Плюсом будет предложение по улучшению модели.
- В отчете должно быть указано, кто в бригаде за что отвечал (написание отчета не является зоной ответственности)
- В приложении должен быть исходный код
- Плюсом будет сравнение разработанной модели с методами решающими задачу и не относящимися к ИНС.

Выполнение работы.

Подготовка данных.

Датасет short jokes представляет собой файл csv, содержащий 231657 анекдотов. Длина анекдота составляет от 10 до 200 знаков. Каждая строка в файле содержит уникальный идентификатор и шутку.

Чтобы посмотреть, что именно из себя представляет датасет, в листинге 1 представлены первые 5 шуток в csv файле вместе с заголовками столбцов.

Листинг 1 - Первые 5 шуток в датасете

```
"ID","Joke"
```

```
1,"[me narrating a documentary about narrators] ""I can't hear what they're saying cuz I'm talking"""
2,"Telling my daughter garlic is good for you. Good immune system and keeps pests away.Ticks, mosquitos, vampires... men."
3,"I've been going through a really rough period at work this week It's my own fault for swapping my tampax for sand paper."
4,"If I could have dinner with anyone, dead or alive... ...I would choose alive. -B.J. Novak-"
5,"Two guys walk into a bar. The third guy ducks."
```

Как видно, есть два столбца: ID - идентификатор шутки, и сама шутка Joke. Данные из csv файла были загружены посредством pandas. В переменную jokes были взяты только шутки, так как ID нам не нужен (листинг 2).

Листинг 2 - Загрузка данных

```
short_jokes = pd.read_csv('./shortjokes.csv')[:20000]
jokes = []
for value in short_jokes['Joke']:
    jokes.append(value.lower())
```

Текст сразу при загрузке преобразовывается в нижний регистр. Для преобразования знаков препинания была написана функция `clean_punctuation(joke)` (листинг 3). В ней задано регулярное выражение, чтобы в списке шуток отобрать все токены, которые содержат в себе буквы (с учетом апострофа, например, для I’m) или следующие знаки препинания: точка, запятая, восклицательный знак, вопросительный, точка с запятой. Далее функция преобразовывает последовательности символов в один символ (“...?” –>”?”), убирает ненужные символы и в конце предложения ставит точку, если там нет этой точки, либо вопросительного знака, либо восклицательного. Например, шутка под номером 1, где присутствуют квадратные скобки: "[me narrating a

documentary about narrators] ""I can't hear what they're saying cuz I'm talking""" будет преобразована в следующий список:

['me', 'narrating', 'a', 'documentary', 'about', 'narrators', 'i', "can't", 'hear', 'what', "they're", 'saying', 'cuz', "i'm", 'talking', '.'].

Листинг 3 - Функция clean_punctuation

```
def clean_punctuation(joke):
    tokens = re.findall(r"[\w']+|[.,!?;]+", joke)
    cleaned = []
    for token in tokens:
        if '?' in token:
            cleaned.append('?')
        elif '!' in token:
            cleaned.append('!')
        elif '..' in token:
            cleaned.append('...')
        else:
            cleaned.append(token)
     if '.' not in cleaned[-1] and '?' not in cleaned[-1] and
'!' not in cleaned[-1]:
            cleaned.append('.')
    return " ".join(cleaned)
```

После функции clean_punctuaion мы получили более пригодный для нейросети текст. Но так как сеть не может воспринимать буквы, а только цифры, нужно привести текст в соответствующий вид. Для этого создается Tokenizer с фильтром, который содержит в себе возможные знаки, которые не относятся к знакам препинания. Мы собираемся отфильтровать наш список токенов и оставить только те токены, которых нет в этом списке.

Затем, с помощью функции fit_on_texts у класса Tokenizer создается словарь индексов и слов по популярности (к примеру, ключ с самым часто

встречающимся символом имеет значение 1). Это преобразование представлено в листинге 4.

Листинг 4 - Преобразование токенов

```
jokes = list(map(clean_punctuation, jokes))
text = ' '.join(jokes)
tokenizer =
Tokenizer(filters='"#$%&()*+,-/:;<=>@[\\]^_`{|}~\t\n')
tokenizer.fit_on_texts(jokes)
```

Вывод количества уникальных слов/символов:

Vocab Size 21600

Теперь мы можем преобразовать наш текст в индексы, которые нейросеть может воспринять (с помощью функции texts_to_sequences), но все еще есть одна проблема. В каждой шутке разное количество символов, а нейросети нужна определенная размерность входных данных. Сначала к каждой шутке, размер которой меньше 11 символов, мы добавляем нули в конец. После этого у нас не будет шуток размерностью меньше 11 символов. После преобразуем шутки в последовательности длиной 11 с шагом 3. Чтобы это сделать, была написана функция `generate_overlapping_encoded_sequences(jokes, maxlen, step)`, которая принимает на вход список шуток, необходимую длину каждой шутки, и шаг (листинг 5). Далее преобразовывает шутку так:

- Если шутка меньше заданного количества символов, то эта шутка отбрасывается.
- Если размер шутки равен заданному размеру, то она добавляется в массив.
- Если шутка больше заданного размера, то она разбивается на несколько шуток с определенным шагом. Например, разобьем шутку “me narrating a documentary about narrators "I can't hear what they're saying cuz I'm talking"” на шутки длинной 11 и шагом 3. В итоге

получится два предложения: “me narrating a documentary about narrators "I can't hear what” и “documentary about narrators "I can't hear what they're saying cuz”. Эти два предложения добавляются в массив.

Листинг 5 - Функция generate_overlapping_encoded_sequences

```
def generate_overlapping_encoded_sequences(jokes, maxlen, step):
    sentences = []
    next_words = []  # holds the targets
    for joke in jokes:
        for j in range(0, len(joke) - maxlen, step):
            sentences.append(joke[j: j + maxlen])
            next_words.append(joke[j + maxlen])
    return sentences, next_words
```

После преобразования из текста в индексы, первые 5 шуток будут выглядеть следующим образом:

0 = {list: 16} [17, 6565, 2, 2945, 50, 11339, 5, 86, 111, 13, 154, 383, 1328, 40, 392, 1]

1 = {list: 23} [537, 10, 563, 8232, 14, 103, 20, 7, 1, 103, 11340, 1848, 8, 657, 11341, 245, 1, 8233, 8234, 2022, 12, 214, 1]

2 = {list: 24} [126, 134, 106, 274, 2, 104, 3486, 746, 35, 175, 49, 483, 51, 10, 328, 2145, 20, 11342, 10, 11343, 20, 2023, 542, 1]

3 = {list: 23} [34, 5, 140, 24, 484, 23, 371, 243, 100, 957, 12, 12, 5, 80, 1770, 957, 1, 615, 1, 1175, 1, 11344, 1]

4 = {list: 12} [81, 228, 198, 64, 2, 112, 1, 3, 1017, 95, 1652, 1]

Как видно, самый часто встречающийся токен - это знак точки. Он стоит в конце каждой шутки.

Теперь у нас имеются последовательности длиной 11 символов, которые состоят из индексов. Но индексы эти целые, поэтому нужно

провести векторизацию (листинг 6). В качестве таргетов используются следующие слова для каждой последовательности (т.е., к примеру, возьмем первую шутку длиной 16 токенов. Первые 11 токенов - это входные данные, а 12-й токен - выходные).

Листинг 6 - Векторизация последовательностей

```
y = np.zeros((len(split_encoded_docs), vocab_size), dtype=np.bool)
for i, padded_doc in enumerate(split_encoded_docs):
    y[i, next_words[i]] = 1
```

Перед тем, как строить модель, опишем callback'и:

- Callback Tensorboard:

  tb_callback = TensorBoard(log_dir="./logs", histogram_freq=2, write_graph=True, embeddings_freq=1)

  Логирует обучение сети.

- Собственный callback под названием MyCallback в листинге 7.

  Генерирует текст в конце первой эпохи, последней, и в конце эпохи через определенный интервал (по умолчанию интервал = 1, т.е. в конце каждой эпохи).

Листинг 7 - MyCallback

```
class MyCallback(keras.callbacks.Callback):
    def __init__(self):
        super(MyCallback, self).__init__()

    def on_epoch_end(self, epoch, logs=None):
         if epoch == 0 or epoch == num_epochs - 1 or epoch % interval == 0:
            word_index = tokenizer.word_index
            index_to_word = dict(
                            (index, word) for word, index in word_index.items())
            max_words = 5
```

```
                maxlen = padded_docs.shape[1]

print("_____
_____")
                 start_index = random.randint(0, len(text.split('
')) - max_words - 1)
                          generated_text = " ".join(text.split('
')[start_index: start_index + max_words])
                                      integer_encoded_gen_text =
tokenizer.texts_to_sequences([generated_text])
                    readable_gen_text = " ".join(map(lambda key:
index_to_word[key], integer_encoded_gen_text[0]))
                print("Random Seed:")
                print(readable_gen_text)

                for _ in range(35):
                                       integer_encoded_gen_text =
tokenizer.texts_to_sequences([generated_text])
                                                padded_gen_text =
pad_sequences(integer_encoded_gen_text,             maxlen=maxlen,
padding='pre')
                           preds = model.predict(padded_gen_text,
verbose=0)[0]
                    next_index = sample(preds)
                    if next_index == 0:
                        break
                                        most_probable_next_word =
index_to_word[next_index]
                     print('Generated:', generated_text, 'Next: ',
most_probable_next_word)
                                      generated_text += " " +
most_probable_next_word
                                   readable_gen_text += " " +
most_probable_next_word
                                           generated_text = "
".join(generated_text.split(' ')[1:])
```

```
                        if most_probable_next_word in ('.', '?',
'!'):
                            break

                print('\nFull generated text:')
                print(readable_gen_text)

print("\n_____________________________________________________________
________")
```

Модель.

Создание шуток - сложная задача для нейронной сети, и она требует, чтобы модели понимали смысл шутки, чтобы генерировать новые. Именно поэтому для генерации текста мы выбрали рекуррентную нейронную сеть. Для того чтобы сеть могла понимать глубокий семантический смысл мы используем двунаправленные слои LSTM.

Схема модели представлена на рис. 1.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 11, 256)           5529600
_________________________________________________________________
bidirectional (Bidirectional (None, 11, 256)           394240
_________________________________________________________________
bidirectional_1 (Bidirection (None, 11, 256)           394240
_________________________________________________________________
bidirectional_2 (Bidirection (None, 256)               394240
_________________________________________________________________
dense (Dense)                (None, 2048)              526336
_________________________________________________________________
dense_1 (Dense)              (None, 21600)             44258400
=================================================================
Total params: 51,497,056
Trainable params: 51,497,056
Non-trainable params: 0
```

Рисунок 1 - Схема модели

В качестве первого слоя используется Embedding для преобразования входных данных в таблицу, где индексом для доступа является номер слова, а значение – это вектор вложения.

Следующие три слоя – двунаправленные слои LSTM, с размером выходного пространства 128. Для избежания переобучения в первых двух слоях были включены dropout и рекуррентный dropout с вероятностями удаления связей 10%. Так как флаг return_sequences установлен в True, эти слои будут выдавать на выход трехразмерные последовательности.

Последний слой состоит из нейронов, количество которых равно размеру словаря уникальных слов и символов. Используется функция активации softmax, поэтому модель выдаст “вероятности” того, насколько слово подходит в предложении. В итоге в предложение будет вставлено слово имеющее наибольшую вероятность.

В листинге 8 представлена реализация модели.

Листинг 8 - Модель

```
embedding_dim = 256
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=padded_docs.shape[1], mask_zero=True))
model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.1, return_sequences=True)))
model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.1, return_sequences=True)))
model.add(Bidirectional(LSTM(128)))
model.add(Dense(2048, kernel_regularizer=tf.keras.regularizers.l2(0.001), activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit(padded_docs, y, batch_size=512, epochs=num_epochs, callbacks=callbacks_list)
```

Модель обучается в течение 20 эпох. Каждая эпоха занимает примерно 12 минут времени. Результат обучения представлен в приложении А. Код программы представлен в приложении Б. На рис. 2 приведено изменение потерь (взято из TensorBoard).

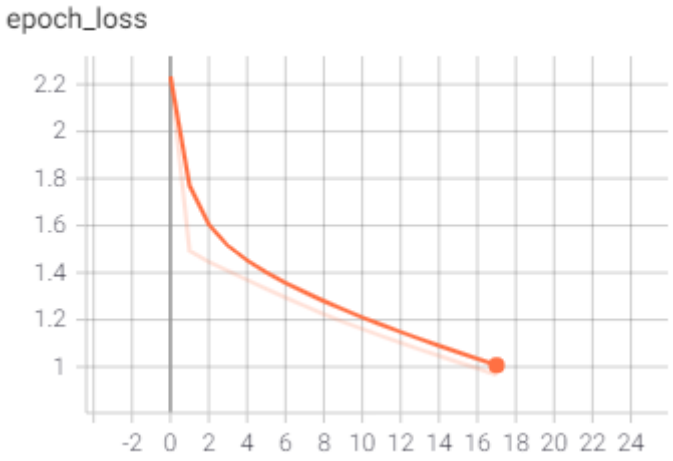

Рисунок 2 - График потерь

Примеры шуток, выдаваемые нейронной сетью:

1. short that he has trouble playing computer games . the only world robert does last says your knows don't and funny a ex gay in ?
2. holding a meatball sub . even if you're not nervous picture minutes like and me your see transplant knocks roses a duet i else a horrible to so did and park man my pig or but the lose play voice god old ... to work a machine want do and putin you she this gathering a ?
3. wife has started using it ! when i see someone pushing my open little golf cancer in preferred does it it let a really musician the she's all ?

Как видно, текст получается несвязным. Но при этом, слова генерируются примерно в том же контексте, и используется больше слов (на предыдущих эпохах сразу генерировался знак препинания).

ПРИЛОЖЕНИЕ А
ВЫВОД ПРОГРАММЫ ПОСЛЕ ОБУЧЕНИЯ НА 20 ЭПОХАХ

```
547/547 [==============================] - 780s 1s/step - loss: 3.7792

Epoch 00001: loss improved from inf to 2.23282, saving model to
./weights\weights-improvement-01-2.2328.hdf5

_________________________________________________________________
Random Seed:
mean our poos come out
Generated: mean our poos come out Next:  who
Generated: our poos come out who Next:  think
Generated: poos come out who think Next:  the

Full generated text:
mean our poos come out who think the


_________________________________________________________________
Epoch 2/20
547/547 [==============================] - 753s 1s/step - loss: 1.4944

Epoch 00002: loss improved from 2.23282 to 1.49234, saving model to
./weights\weights-improvement-02-1.4923.hdf5

_________________________________________________________________
Random Seed:
50 shades of grey girl
Generated: 50 shades of grey girl Next:  top
Generated: shades of grey girl top Next:  away
Generated: of grey girl top away Next:  in

Full generated text:
50 shades of grey girl top away in


_________________________________________________________________
Epoch 3/20
547/547 [==============================] - 773s 1s/step - loss: 1.4393

Epoch 00003: loss improved from 1.49234 to 1.44660, saving model to
./weights\weights-improvement-03-1.4466.hdf5

_________________________________________________________________
```

```
Random Seed:
a man two prosthetic legs
Generated: a man two prosthetic legs Next:  line
Generated: man two prosthetic legs line Next:  !

Full generated text:
a man two prosthetic legs line !


_________________________________________________________________
Epoch 4/20
547/547 [==============================] - 696s 1s/step - loss: 1.3960

Epoch 00004: loss improved from 1.44660 to 1.40844, saving model to
./weights\weights-improvement-04-1.4084.hdf5
_________________________________________________________________
Random Seed:
me a party animal then
Generated: me a party animal then Next:  bad
Generated: a party animal then bad Next:  .

Full generated text:
me a party animal then bad .


_________________________________________________________________
Epoch 5/20
547/547 [==============================] - 685s 1s/step - loss: 1.3599

Epoch 00005: loss improved from 1.40844 to 1.37007, saving model to
./weights\weights-improvement-05-1.3701.hdf5
_________________________________________________________________
Random Seed:
it's a lot harder to

Full generated text:
it's a lot harder to


_________________________________________________________________
Epoch 6/20
547/547 [==============================] - 685s 1s/step - loss: 1.3349
```

```
Epoch 00006: loss improved from 1.37007 to 1.33152, saving model to
./weights\weights-improvement-06-1.3315.hdf5

--------------------------------------------------------------------------------
Random Seed:
i bury them alive because

Full generated text:
i bury them alive because


--------------------------------------------------------------------------------
Epoch 7/20
547/547 [==============================] - 685s 1s/step - loss: 1.2797

Epoch 00007: loss improved from 1.33152 to 1.29417, saving model to
./weights\weights-improvement-07-1.2942.hdf5

--------------------------------------------------------------------------------
Random Seed:
of course . what
Generated: , of course . what Next:  mew
Generated: of course . what mew Next:  falsetto
Generated: course . what mew falsetto Next:  did
Generated: . what mew falsetto did Next:  him
Generated: what mew falsetto did him Next:  to

Full generated text:
of course . what mew falsetto did him to


--------------------------------------------------------------------------------
Epoch 8/20
547/547 [==============================] - 687s 1s/step - loss: 1.2522

Epoch 00008: loss improved from 1.29417 to 1.25897, saving model to
./weights\weights-improvement-08-1.2590.hdf5

--------------------------------------------------------------------------------
Random Seed:
nipples ? it's braille for
Generated: nipples ? it's braille for Next:  pads
Generated: ? it's braille for pads Next:  starts
Generated: it's braille for pads starts Next:  fire
Generated: braille for pads starts fire Next:  still
```

```
Full generated text:
nipples ? it's braille for pads starts fire still


_________________________________________________________________
Epoch 9/20
547/547 [==============================] - 697s 1s/step - loss: 1.2201

Epoch 00009: loss improved from 1.25897 to 1.22348, saving model to
./weights\weights-improvement-09-1.2235.hdf5
_________________________________________________________________
Random Seed:
then getting lost means
Generated: then , getting lost means Next:  happened
Generated: , getting lost means happened Next:  on
Generated: getting lost means happened on Next:  ugly
Generated: lost means happened on ugly Next:  right
Generated: means happened on ugly right Next:  more
Generated: happened on ugly right more Next:  arrrrrrrr
Generated: on ugly right more arrrrrrrr Next:  fun
Generated: ugly right more arrrrrrrr fun Next:  yes
Generated: right more arrrrrrrr fun yes Next:  .

Full generated text:
then getting lost means happened on ugly right more arrrrrrrr fun yes .


_________________________________________________________________
Epoch 10/20
547/547 [==============================] - 700s 1s/step - loss: 1.1924

Epoch 00010: loss improved from 1.22348 to 1.19102, saving model to
./weights\weights-improvement-10-1.1910.hdf5
_________________________________________________________________
Random Seed:
my cellf . me my

Full generated text:
my cellf . me my


_________________________________________________________________
Epoch 11/20
547/547 [==============================] - 702s 1s/step - loss: 1.1391
```

```
Epoch 00011: loss improved from 1.19102 to 1.16059, saving model to
./weights\weights-improvement-11-1.1606.hdf5

----------------------------------------------------------------------
Random Seed:
he's used to playing 18
Generated: he's used to playing 18 Next:  address
Generated: used to playing 18 address Next:  swirling
Generated: to playing 18 address swirling Next:  hate
Generated: playing 18 address swirling hate Next:  have
Generated: 18 address swirling hate have Next:  best
Generated: address swirling hate have best Next:  ship
Generated: swirling hate have best ship Next:  everyone
Generated: hate have best ship everyone Next:  you've
Generated: have best ship everyone you've Next:  lost
Generated: best ship everyone you've lost Next:  kraft
Generated: ship everyone you've lost kraft Next:  champions
Generated: everyone you've lost kraft champions Next:  ?

Full generated text:
he's used to playing 18 address swirling hate have best ship everyone
you've lost kraft champions ?


----------------------------------------------------------------------
Epoch 12/20
547/547 [==============================] - 725s 1s/step - loss: 1.1232

Epoch 00012: loss improved from 1.16059 to 1.13158, saving model to
./weights\weights-improvement-12-1.1316.hdf5

----------------------------------------------------------------------
Random Seed:
him perform an unidentifiable skill
Generated: him perform an unidentifiable skill Next:  .

Full generated text:
him perform an unidentifiable skill .


----------------------------------------------------------------------
Epoch 13/20
547/547 [==============================] - 726s 1s/step - loss: 1.0908
```

```
Epoch 00013: loss improved from 1.13158 to 1.10287, saving model to
./weights\weights-improvement-13-1.1029.hdf5

--------------------------------------------------------------------------------
Random Seed:
. son groggily what's updog

Full generated text:
. son groggily what's updog


--------------------------------------------------------------------------------
Epoch 14/20
547/547 [==============================] - 751s 1s/step - loss: 1.0661

Epoch 00014: loss improved from 1.10287 to 1.07335, saving model to
./weights\weights-improvement-14-1.0734.hdf5

--------------------------------------------------------------------------------
Random Seed:
but admiting i'm cold to
Generated: but admiting i'm cold to Next:  slap
Generated: admiting i'm cold to slap Next:  home
Generated: i'm cold to slap home Next:  yesterday
Generated: cold to slap home yesterday Next:  on
Generated: to slap home yesterday on Next:  another
Generated: slap home yesterday on another Next:  funeral
Generated: home yesterday on another funeral Next:  !

Full generated text:
but admiting i'm cold to slap home yesterday on another funeral !


--------------------------------------------------------------------------------
Epoch 15/20
547/547 [==============================] - 763s 1s/step - loss: 1.0331

Epoch 00015: loss improved from 1.07335 to 1.04715, saving model to
./weights\weights-improvement-15-1.0472.hdf5

--------------------------------------------------------------------------------
Random Seed:
a bit . what's the

Full generated text:
a bit . what's the
```

```
_________________________________________________________________
Epoch 16/20
547/547 [==============================] - 727s 1s/step - loss: 1.0067

Epoch 00016: loss improved from 1.04715 to 1.01913, saving model to
./weights\weights-improvement-16-1.0191.hdf5
_________________________________________________________________
Random Seed:
self aware inanimate objects .

Full generated text:
self aware inanimate objects .


_________________________________________________________________
Epoch 17/20
547/547 [==============================] - 703s 1s/step - loss: 0.9775

Epoch 00017: loss improved from 1.01913 to 0.99363, saving model to
./weights\weights-improvement-17-0.9936.hdf5
_________________________________________________________________
Random Seed:
seasoned vet . basic instinct
Generated: seasoned vet . basic instinct Next:  http
Generated: vet . basic instinct http Next:  world
Generated: . basic instinct http world Next:  !

Full generated text:
seasoned vet . basic instinct http world !


_________________________________________________________________
Epoch 18/20
547/547 [==============================] - 742s 1s/step - loss: 0.9502

Epoch 00018: loss improved from 0.99363 to 0.96516, saving model to
./weights\weights-improvement-18-0.9652.hdf5
_________________________________________________________________
Random Seed:
the moon even if

Full generated text:
```

```
the moon even if

_________________________________________________________________
Epoch 19/20
 21/547 [>.............................] - ETA: 11:02 - loss: 0.8997
Process finished with exit code -1
```

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ

```
import random
import re
import sys

import numpy as np
import pandas as pd
import tensorflow as tf
from keras.layers import Dense, LSTM, Bidirectional, Dropout, GRU
from keras.layers.embeddings import Embedding
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras.optimizers import RMSprop
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from tensorflow import keras


def clean_punctuation(joke):
    # убирает из строки все, что не соответствует регулярному выражению
    tokens = re.findall(r"[\w']+|[.,!?;]+", joke)
    cleaned = []

     # если в токене хранится несколько знаков препинания подряд, то,
например, если там есть вопрос, то в список
     # cleaned добавляется только вопросительный знак. Пример: ["...?"]
-> ["?"]
    for token in tokens:
        if '?' in token:
            cleaned.append('?')
        elif '!' in token:
            cleaned.append('!')
        elif '..' in token:
            cleaned.append('...')
        else:
            cleaned.append(token)
```

```
            # если предложение не заканчивается на '.', '?', '!', то в конец
ставится точка.
            if '.' not in cleaned[-1] and '?' not in cleaned[-1] and '!' not in
cleaned[-1]:
                cleaned.append('.')
            return " ".join(cleaned)


        def sample(preds, temperature=1.0):
            preds = np.asarray(preds.astype('float64'))
            preds = np.log(preds) / temperature
            exp_preds = np.exp(preds)
            preds = exp_preds / np.sum(exp_preds)
            probas = np.random.multinomial(1, preds, 1)
            return np.argmax(probas)


        def generate_overlapping_encoded_sequences(jokes, maxlen, step):
            sentences = []
            next_words = []  # holds the targets
            for joke in jokes:
                for j in range(0, len(joke) - maxlen, step):
                    sentences.append(joke[j: j + maxlen])
                    next_words.append(joke[j + maxlen])
            return sentences, next_words


        # загрузка данных
        short_jokes = pd.read_csv('./shortjokes.csv')[:20000]

        # т.к. в данных 2 столбца: ID и Joke, мы берем только Joke
        jokes = []
        for value in short_jokes['Joke']:
            jokes.append(value.lower())

        jokes = list(map(clean_punctuation, jokes))
        text = ' '.join(jokes)  # преобразование из списка в один текст

        tokenizer = Tokenizer(filters='"#$%&()*+,-/:;<=>@[\\]^_`{|}~\t\n')   #
filters - не учитывает выбранные символы
```

```
    # создает словарь индексов и слов по популярности (ключ с самым
популярным словом имеет значение 1)
    tokenizer.fit_on_texts(jokes)
    vocab_size = len(tokenizer.word_index) + 1  # количество уникальных
слов/символов
    print('Vocab Size', vocab_size)

    # разбиение шуток в последовательности длиной 11
    seq_length = 11
    step = 3
    integer_encoded_docs = tokenizer.texts_to_sequences(jokes)  # заменяет
слова и символы в тексте на значения из словаря
    integer_encoded_docs = pad_sequences(integer_encoded_docs,
                                         padding='post')  # добавление 0 к
спискам, размер которых меньше 11
    split_encoded_docs,                    next_words                    =
generate_overlapping_encoded_sequences(integer_encoded_docs,      seq_length,
step)
    # размерность padded_docs = (len(split_encoded_docs), 11)
    padded_docs = pad_sequences(split_encoded_docs, padding='post')
    next_words = np.asarray(next_words)  # нужно получить следующее слово
для каждого из этих
    print("Number of Sequences:", len(padded_docs))

    # Векторизация последовательностей
    y = np.zeros((len(padded_docs), vocab_size), dtype=np.bool)
    for i, padded_doc in enumerate(padded_docs):
        y[i, next_words[i]] = 1

    num_epochs = 20
    interval = 1


    class MyCallback(keras.callbacks.Callback):
        def __init__(self):
            super(MyCallback, self).__init__()

        def on_epoch_end(self, epoch, logs=None):
            if epoch == 0 or epoch == num_epochs - 1 or epoch % interval ==
0:
```

```
                    word_index = tokenizer.word_index  # словарь индексов и
слов, где ключи - слова
                index_to_word = dict(
                    (index, word) for word, index in word_index.items())  #
такой же словарь, где ключи - индексы
                max_words = 5
                maxlen = padded_docs.shape[1]

print("_____________________________________________________________________")
                    start_index = random.randint(0, len(text.split(' ')) -
max_words - 1)
                    generated_text = " ".join(text.split(' ')[start_index:
start_index + max_words])
                                                integer_encoded_gen_text    =
tokenizer.texts_to_sequences([generated_text])
                            readable_gen_text  =  "  ".join(map(lambda  key:
index_to_word[key], integer_encoded_gen_text[0]))
                print("Random Seed:")
                print(readable_gen_text)

                for _ in range(35):
                                                 integer_encoded_gen_text   =
tokenizer.texts_to_sequences([generated_text])
                                                          padded_gen_text    =
pad_sequences(integer_encoded_gen_text, maxlen=maxlen, padding='pre')
                    preds = model.predict(padded_gen_text, verbose=0)[0]
                    next_index = sample(preds)
                    if next_index == 0:
                        break
                    most_probable_next_word = index_to_word[next_index]
                            print('Generated:', generated_text, 'Next: ',
most_probable_next_word)
                    generated_text += " " + most_probable_next_word
                    readable_gen_text += " " + most_probable_next_word
                          generated_text = " ".join(generated_text.split('
')[1:])
                    if most_probable_next_word in ('.', '?', '!'):
                        break

                print('\nFull generated text:')
                print(readable_gen_text)
```

```
print("\n_____")


      filepath = "./weights/weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
      checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1,
save_best_only=True, mode='min')
      tb_callback = TensorBoard(log_dir="./logs", histogram_freq=2,
write_graph=True, embeddings_freq=1)
      callbacks_list = [checkpoint, MyCallback(), tb_callback]

      embedding_dim = 256
      model = Sequential()
      model.add(Embedding(vocab_size, embedding_dim,
input_length=padded_docs.shape[1], mask_zero=True))
      model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.1,
return_sequences=True)))
      model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.1,
return_sequences=True)))
      model.add(Bidirectional(LSTM(128)))
      model.add(Dense(2048,
kernel_regularizer=tf.keras.regularizers.l2(0.001), activation='relu'))
      model.add(Dense(vocab_size, activation='softmax'))
      # optimizer = RMSprop(lr=0.01)
      model.compile(loss='categorical_crossentropy', optimizer='adam')
      print(model.summary())
      model.fit(padded_docs, y, batch_size=512, epochs=num_epochs,
callbacks=callbacks_list)
```