Лабораторная работа №7.

Рекуррентные нейронные сети (RNN)

(Генерация имен рекуррентными нейронными сетями)

1. Предварительные действия

```
import numpy as np
import matplotlib.pyplot as plt
```

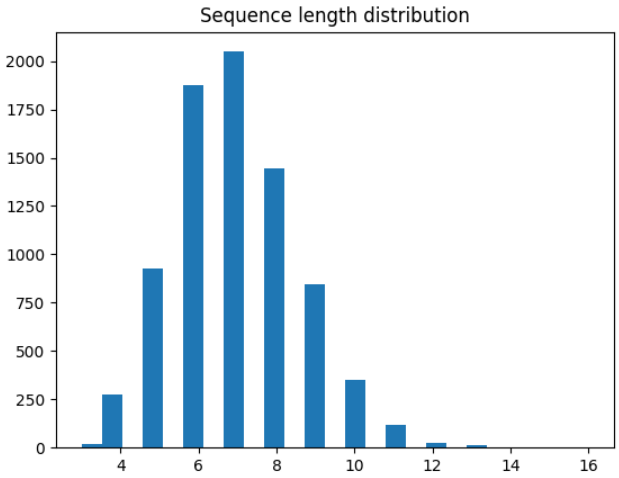
Чтение данных.

Файл «names» следует скачать по ссылке:
https://github.com/sshkhr/Practical_RL/blob/master/week7_%5Brecap%5D_rnn/names

```
import os
start_token = " "

with open("names") as f:
    lines = f.read()[:-1].split('\n')
    lines = [start_token + name for name in lines]

print('n samples = ', len(lines))
for x in lines[::1000]:
    print(x)

MAX_LENGTH = max(map(len, lines))
print("max length =", MAX_LENGTH)

plt.title('Sequence length distribution')
plt.hist(list(map(len, lines)), bins=25)
plt.show()
```

Вывод:

```
n samples =  7944
 Abagael
 Claresta
 Glory
 Liliane
 Prissie
 Geeta
 Giovanne
 Piggy
max length = 16
```



2. Обработка текста

Сначала нам нужно собрать «словарь» всех уникальных токенов, то есть уникальных персонажей. Затем мы можем закодировать входные данные как последовательность идентификаторов символов.

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# получить все уникальные символы из строк (включая заглавные буквы и символы)
tokens = <ВАШ КОД>

tokens = list(tokens)

n_tokens = len(tokens)
print('n_tokens = ', n_tokens)

assert 50 < n_tokens < 60
```

Вывод:

n_tokens = 55

Далее превратите все от символов в идентификаторы. Так как манипуляции со строками в Tensorflow довольно сложны, поэтому мы их обойдем. Мы будем подавать на вход нашей рекуррентной нейронной сети идентификаторы символов из нашего словаря.

Чтобы создать такой словарь, давайте присвоим каждому символу его индекс в списке токенов.

```
# словарь символов -> его идентификатор (индекс в списке токенов)
token_to_id = <ВАШ КОД>
```

```
assert len(tokens) == len(token_to_id), "словари должны иметь идинаковый размер"
for i in range(n_tokens):
    assert token_to_id[tokens[i]
                       ] == i, "идентификатор токена должен быть его позицией в списке токенов"

print("Seems alright!")
```

Вывод:

Seems alright!

```
def to_matrix(names, max_len=None, pad=token_to_id[' '], dtype='int32'):
    """Преобразует список имен в приемлемую для rnn матрицу"""
    max_len = max_len or max(map(len, names))
    names_ix = np.zeros([len(names), max_len], dtype) + pad

    for i in range(len(names)):
        name_ix = list(map(token_to_id.get, names[i]))
        names_ix[i, :len(name_ix)] = name_ix

    return names_ix
```

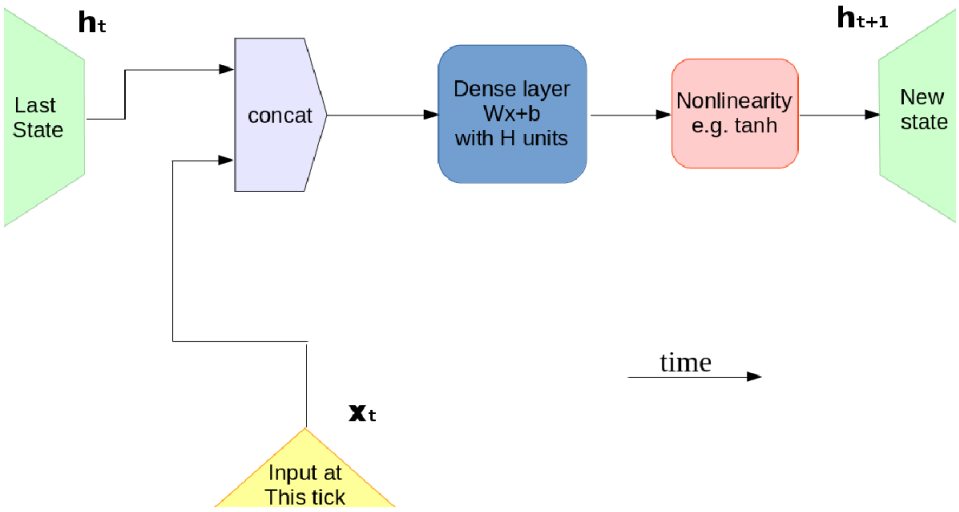
```
# Пример: привести 4 случайных имени в матричный вид, дополнить нулями
print('\n'.join(lines[::2000]))
print(to_matrix(lines[::2000]))
```

Вывод:

```
Abagael
 Glory
 Prissie
 Giovanne
[[50 22 13  9  3  9 12 37 50]
 [50  5 37 53  0 14 50 50 50]
 [50 54  0  8 32 32  8 12 50]
 [50  5  8 53 30  9 28 28 12]]
```

3. Рекурентная нейронная сеть

Мы можем переписать рекуррентную нейронную сеть как последовательное применение основных слоев к входным данным и предыдущему состоянию RNN.



Поскольку мы обучаем языковую модель, также должны быть:

- Слой эмбединга (embedding), который преобразует идентификатор символа x_t в вектор.
- Выходной слой, который предсказывает вероятности следующего токена.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import keras
import keras.layers as L
```

```
emb_size, rnn_size = 16, 64

# слой эмбединга, который преобразует id символа в вектор
embed_x = L.Embedding(n_tokens, emb_size)

# основной слой, который отображает ввод и предыдущее состояние в новое
скрытое состояние, [x_t,h_t] -> h_t+1
get_h_next = L.Dense(rnn_size, activation='tanh')

# основной слой, который сопоставляет текущее скрытое состояние с
вероятностями символов [h_t+1]->P(x_t+1|h_t+1)
get_probas = L.Dense(n_tokens, activation='softmax')
```

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
def rnn_one_step(x_t, h_t):
    """
    Шаг рекуррентной нейронной сети, который создает следующее состояние и
выходные данные
     учитывая предыдущий ввод и предыдущее состояние.
     Мы будем вызывать этот метод несколько раз, чтобы создать всю
последовательность.
     :param x_t: вектор токена, int32[batch_size,]
     :param h_t: матрица предыдущего состояния, float32[batch_size, rnn_size]

     Следуйте инструкциям для завершения функции.
    """
    # 1. преобразовать идентификатор персонажа во встраивание (используйте
слой embed_x)
    x_t_emb = embed_x(tf.reshape(x_t, [-1, 1]))[:, 0]

    # 2. объединить x _embedding_ и предыдущее состояние h (по последней оси)
    <ВАШ КОД>

    # 3. вычислить следующее состояние с учетом встраивания h и x
    <ВАШ КОД>

    # 4. получить вероятности для языковой модели P(x_next | h_next)
    <ВАШ КОД>

    return next_h, next_probas
```

```
input_sequence = tf.placeholder('int32', (None, MAX_LENGTH))
batch_size = tf.shape(input_sequence)[0]

# начальное скрытое состояние
h0 = tf.zeros([batch_size, rnn_size])
```

```
# TEST: один шаг RNN
h1, p_y1 = rnn_one_step(input_sequence[:, 0], h0)

dummy_data = np.arange(MAX_LENGTH * 2).reshape([2, -1])
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
test_h1, test_p_y1 = sess.run([h1, p_y1],  {input_sequence: dummy_data})

assert test_h1.shape == (len(dummy_data), rnn_size)
```

```
assert test_p_y1.shape == (
    len(dummy_data), n_tokens) and np.allclose(test_p_y1.sum(-1), 1)
```

4. Цикл RNN

Как только rnn_one_step будет готов, давайте применим его в цикле к символам имени, чтобы получить прогноз. Давайте предположим, что на данный момент все имена имеют длину не более 16, поэтому мы можем просто перебирать их в цикле for. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
h_prev = h0
predicted_probas = []

for t in range(MAX_LENGTH):
    x_t = input_sequence[:, t]

    # Задание: вычислить следующий токен probas следующее скрытое состояние
    probas_next, h_next = <ВАШ КОД>

    # КОНЕЦ ВАШЕГО КОДА

    predicted_probas.append(probas_next)
    h_prev = h_next

predicted_probas = tf.stack(predicted_probas, axis=1)
```

```
assert predicted_probas.shape.as_list() == [None, MAX_LENGTH, n_tokens]
assert h_prev.shape.as_list() == h0.shape.as_list()
```

5. RNN: потери и градиенты

Давайте соберем матрицу прогнозов и соответствующих им правильных ответов. Затем нашу сеть можно обучить, сводя к минимуму кроссэнтропию между предсказанными вероятностями и этими ответами.

```
predictions_matrix = predicted_probas[:, :-1]
answers_matrix = tf.one_hot(input_sequence[:, 1:], n_tokens)

print('predictions_matrix:', predictions_matrix.shape)
print('answers_matrix:', predictions_matrix.shape)
```

Вывод:

predictions_matrix: (?, 15, 55)
answers_matrix: (?, 15, 55)

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

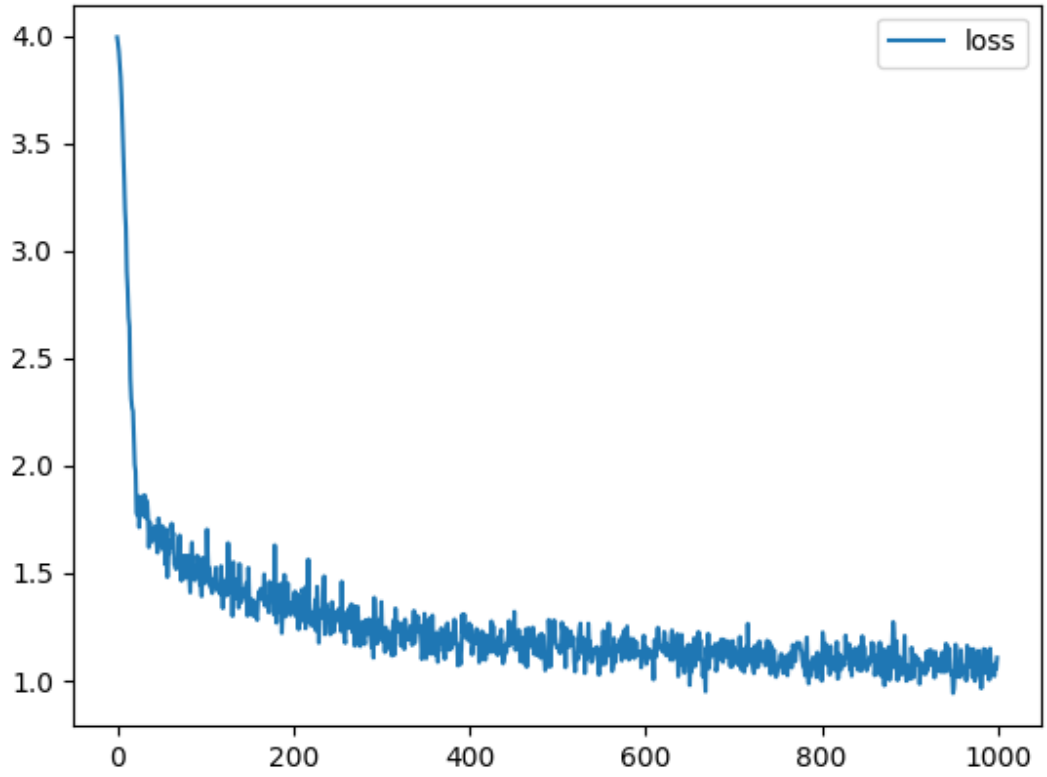
```
# определяют потерю как категориальную кроссэнтропию. Имейте в виду, что
прогнозы — это вероятности, а НЕ логиты!
loss = <YOUR_CODE>

optimize = tf.train.AdamOptimizer().minimize(loss)
```

Цикл обучения

```
from random import sample
sess.run(tf.global_variables_initializer())

history = []
```

```
for i in range(1000):
    batch = to_matrix(sample(lines, 32), max_len=MAX_LENGTH)

    loss_i, _ = sess.run([loss, optimize], {input_sequence: batch})

    history.append(loss_i)
    if (i+1) % 100 == 0:
        plt.plot(history, label='loss')
        plt.legend()
        plt.show()

assert np.mean(history[:10]) > np.mean(history[-10:]), "RNN didn't converge."
```

Вывод:



6. RNN: выборка

После того, как мы немного обучили нашу сеть, давайте приступим к фактической генерации материала. Все, что нам нужно, это функция rnn_one_step, которую вы написали выше.

```
x_t = tf.placeholder('int32', (None,))
h_t = tf.Variable(np.zeros([1, rnn_size], 'float32'))

next_h, next_probs = rnn_one_step(x_t, h_t)
```

```
def generate_sample(seed_phrase=' ', max_length=MAX_LENGTH):
    '''
    Функция генерирует текст по заданной фразе длиной не менее SEQ_LENGTH.
     :param seed_phrase: символы префикса. RNN просят продолжить фразу
     :param max_length: максимальная длина вывода, включая seed_phrase
     :param temperature: коэффициент для выборки. более высокая температура
производит более хаотичные выходы,
                         меньшая температура сходится к единственному
наиболее вероятному выходу
    '''
    x_sequence = [token_to_id[token] for token in seed_phrase]
    sess.run(tf.variables_initializer([h_t]))

    # подать сид-фразу, если таковая имеется
    for ix in x_sequence[:-1]:
        sess.run(tf.assign(h_t, next_h), {x_t: [ix]})

    # начать генерацию
    for _ in range(max_length-len(seed_phrase)):
        x_probs, _ = sess.run([next_probs, tf.assign(h_t, next_h)], {
                               x_t: [x_sequence[-1]]})
        x_sequence.append(np.random.choice(n_tokens, p=x_probs[0]))

    return ''.join([tokens[ix] for ix in x_sequence])
```

```
for _ in range(10):
    print(generate_sample())

for _ in range(10):
    print(generate_sample(' Trump'))
```

Вывод:

Kamr
Linde
Lamren
Cers
Alera
Auny
B ren
fahill
Ba dila
Zine
Trumpel
Trump
Trumpon
Trumpia
Trumpe
Trumpa
Trumpate
Trumpelrin
Trumpa
Trumpellit

Задания к лабораторной работе №1

1. Написать свой код к разделам 1 - 6 согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>

```
<ВАШ КОД: >
```

2. Запустить полученную программу, получить ожидаемые выходные данные.

Задания к лабораторной работе №2

1. Замените библиотеку Tensorflow на Pytorch и воспроизведите все эксперименты из разделов 1-6.

Требование к отчету

1. В качестве отчета принимаются два Python файла с кодом (для Tensorflow и Pytorch).
2. Код должен быть в достаточной мере прокомментирован.
3. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip