УНИВЕРСИТЕТ ИТМО

Н.А. Жукова, И.А. Куликов, А.Н. Субботин

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ЛАБОРАТОРНЫЕ РАБОТЫ

Санкт-Петербург
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Н.А. Жукова, И.А. Куликов, А.Н. Субботин

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ЛАБОРАТОРНЫЕ РАБОТЫ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.04.04 Программная инженерия
в качестве Учебно-методическое пособие для реализации основных профессиональных образовательных программ высшего образования магистратуры

Санкт-Петербург
2022

Жукова Н.А., ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ЛАБОРАТОРНЫЕ РАБОТЫ– СПб: Университет ИТМО, 2022. – 95 с.

Рецензент(ы):
Пшеничный Кирилл Анатольевич, кандидат геолого-минералогических наук, доцент (квалификационная категория "ординарный доцент") факультета безопасности информационных технологий, Университета ИТМО.

Учебно-методическое пособие разработано в соответствии с программой дисциплины «Обучение с подкреплением» и предназначено для студентов, обучающихся по направлению подготовки 09.04.04 Программная инженерия. Учебно-методическое пособие содержит описание лабораторных работ и рекомендации по их выполнению.

Университет ИТМО – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

Содержание

ВВЕДЕНИЕ

Предлагаемое учебно-методическое пособие содержит лабораторные работы по курсу «Обучение с подкреплением» и задания для закрепления материала в конце каждой лабораторной работы. Для выполнения работ необходимы базовые знания теории искусственных нейронных сетей, машинного обучения, интерпретатора Python и командной строки. Лабораторные работы являются взаимосвязанными, рекомендуется их выполнять последовательно, начиная с первой лабораторной работы. Требуется установка Python и ряда библиотек по инструкциям, которые даны в описаниях лабораторных работ. Операционная система может быть установлена любая: Windows, Linux, но версии не старше 2012 года.

Учебное издание предназначено для студентов магистратуры и всех интересующихся искусственными нейронными сетями, машинным обучением и, в частности, разделом «Обучение с подкреплением» с применением языка Python.

Лабораторная работа №1. Основы работы с библиотеками Gym, Tensorflow и PyTorch

Библиотека Gym

1. Установка

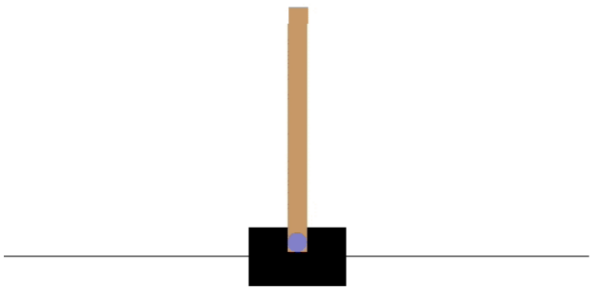
```
pip install gym
```

2. Инициализация и визуализация среды.

Запускаем экземпляр среды ***CartPole-v0*** для 1000 меток времени (шагов), отображаем среду для каждого шага.

```
import gym
env = gym.make('CartPole-v0')

env.reset()

for _ in range(1000):

   env.render()

   env.step(env.action_space.sample()) # take a random action

env.close()
```

После запуска появляется окно, которое визуализирует классическую проблемы с перевернутым маятником.

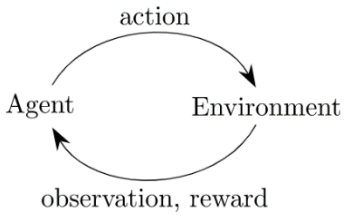


3. Наблюдения (Observations)

Функция step среды возвращает состояние среды. На самом деле step возвращает четыре значения:

- **observation** (object): объект, относящийся к среде. Например, пиксельные данные с камеры, углы и скорости суставов робота или состояние доски в настольной игре.
- **reward** (float): сумма вознаграждения, полученного за предыдущее действие.
- **done** (Boolean): флаг того, пришло ли время снова сбросить настройки среды. Большинство (но не все) задач разделены на четко определенные эпизоды, а выполненное значение True указывает на завершение эпизода. (Например, возможно, шест наклонился слишком далеко, или вы потеряли свою последнюю жизнь.)
- **info** (dict): диагностическая информация, полезная для отладки. Иногда она может быть полезна для обучения (например, она может содержать необработанные вероятности последнего изменения состояния среды).

Ниже приведён пример реализации классического «цикла агент-среда». На каждом временном шаге агент выбирает действие, а среда возвращает наблюдение и вознаграждение.



Процесс стартует путем вызова функции reset(), которая возвращает первичное наблюдение. Таким образом, правильнее переписать предыдущий код для учета флага done:

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
  observation = env.reset()
  for t in range(100):
    env.render()
    print(observation)
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
    if done:
      print("Episode finished after {} timesteps".format(t+1))
      break
env.close()
```

В Выводе будет указано, на каком шаге закончился каждый эпизод:

...

[-0.07825026 0.17098394 0.17772266 0.04518264]

[-0.07483058 -0.02618224 0.17862631 0.38824934]

[-0.07535423 0.16601425 0.1863913   0.15677905]

[-0.07203394 -0.03121928 0.18952687 0.50198734]

[-0.07265832 -0.22843622 0.19956662 0.84790343]

Episode finished after 16 timesteps

3. Пространства (Spaces)

Каждая среда взаимодействует с двумя пространствами: **action_space** и **observation_space**. Эти пространства являются атрибутами среды, и они описывают формат валидных действий и наблюдений:

```
import gym
env = gym.make('CartPole-v0')
print(env.action_space)
#> Discrete(2)
print(env.observation_space)
#> Box(4,)
```

Вывод:

Discrete(2)

Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)

Допускается фиксированный диапазон неотрицательных чисел в дискретном пространстве, поэтому допустимыми действиями являются либо 0, либо 1. Пространство Box представляет собой n-мерный блок, поэтому действительные наблюдения будут массивом из 4 чисел. Мы также можем проверить границы Box:

```
print(env.observation_space.high)
#> array([ 2.4       ,        inf,  0.20943951,        inf])
print(env.observation_space.low)
#> array([-2.4       ,       -inf, -0.20943951,       -inf])
```

Вывод:

[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]

[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]

Этот самоанализ может быть полезен для написания универсального кода, который работает во многих различных средах. Box и Discrete являются наиболее распространенными пространствами. Вы можете взять пример из пространства или проверить, что что-то ему принадлежит:

```
from gym import spaces
space = spaces.Discrete(8) # Set with 8 elements {0, 1, 2, ..., 7}
x = space.sample()
assert space.contains(x)
assert space.n == 8
```

Для получения списка зарегистрированных сред можно выполнить:

```
from gym import envs
print(envs.registry.all())
```

Вывод:

dict_values([EnvSpec(CartPole-v0), EnvSpec(CartPole-v1), EnvSpec(MountainCar-v0), EnvSpec(MountainCarContinuous-v0), EnvSpec(Pendulum-v1), EnvSpec(Acrobot-v1), EnvSpec(LunarLander-v2), EnvSpec(LunarLanderContinuous-v2), EnvSpec(BipedalWalker-v3), EnvSpec(BipedalWalkerHardcore-v3), EnvSpec(CarRacing-v0), EnvSpec(Blackjack-v1), …)

Задание №1

Создать среду Taxi-v3, отобразить кадры среды, значение временного шага, код состояния, код действия, значение вознаграждения для 10 случайных действий.

Библиотека Tensorflow

1. Установка

```
pip install tensorflow
```

2. Импорт библиотеки

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
```

3. Загрузка и подготовка набора данных MNIST

http://yann.lecun.com/exdb/mnist/

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
# Add a channels dimension
x_train = x_train[..., tf.newaxis].astype("float32")
x_test = x_test[..., tf.newaxis].astype("float32")
```

Вывод:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

11493376/11490434 [==============================] - 1s 0us/step

11501568/11490434 [==============================] - 1s 0us/step
```

4. Создание тренировочного и тестового наборов данных

Использование **tf.data** для пакетной обработки и перемешивания набора данных:

```
train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(10000).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

5. Создание модели

Создание модели **tf.keras** , используя API подкласса модели Keras:

```
class MyModel(Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10)
  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
# Create an instance of the model
model = MyModel()
```

Выбор оптимизатора и функции потерь для обучения:

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
```

Выбор показателей для измерения потерь и точности модели. Эти метрики накапливают значения за эпохи, а затем выводят общий результат.

```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')
test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

6. Обучение и тестирование модели

Использование **tf.GradientTape** для обучения модели:

```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  train_loss(loss)
  train_accuracy(labels, predictions)
```

Тестирование модели:

```
@tf.function
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)
  test_loss(t_loss)
  test_accuracy(labels, predictions)

EPOCHS = 5
for epoch in range(EPOCHS):
  # Reset the metrics at the start of the next epoch
  train_loss.reset_states()
  train_accuracy.reset_states()
  test_loss.reset_states()
```

```
 test_accuracy.reset_states()
 for images, labels in train_ds:
  train_step(images, labels)
 for test_images, test_labels in test_ds:
  test_step(test_images, test_labels)
 print(
  f'Epoch {epoch + 1}, '
  f'Loss: {train_loss.result()}, '
  f'Accuracy: {train_accuracy.result() * 100}, '
  f'Test Loss: {test_loss.result()}, '
  f'Test Accuracy: {test_accuracy.result() * 100}'
 )
```

Вывод:

Epoch 1, Loss: 0.14055445790290833, Accuracy: 95.85333251953125, Test Loss: 0.06531758606433868, Test Accuracy: 97.93000030517578

Epoch 2, Loss: 0.04441937804222107, Accuracy: 98.68000030517578, Test Loss: 0.05592753738164902, Test Accuracy: 98.13999938964844

Epoch 3, Loss: 0.023894023150205612, Accuracy: 99.23833465576172, Test Loss: 0.057369183748960495, Test Accuracy: 98.27999877929688

Epoch 4, Loss: 0.013493617996573448, Accuracy: 99.58000183105469, Test Loss: 0.0697859525680542, Test Accuracy: 98.1199951171875

Epoch 5, Loss: 0.009707454591989517, Accuracy: 99.66999816894531, Test Loss: 0.06326959282159805, Test Accuracy: 98.2699966430664

Задание №2

Используя библиотеку tensorflow, создайте, обучите и оцените свою модель, используя датасет, отличный от MNIST

Библиотека PyTorch

Задание №3

1. Самостоятельно изучить функции библиотеки PyTorch для Python: https://pytorch.org/get-started/locally/, https://pytorch.org/docs/stable/index.html

2. Используя библиотеку pytorch, повторить создание, обучение и оценку модели из Задания №2

Требование к отчету

1. В качестве отчета принимаются три Python файла (для каждого из заданий).
2. Код должен быть в достаточной мере прокомментирован.
3. Локальные датасеты также должны входить в отчет.
4. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip

Лабораторная работа №2. Кросс-энтропийный метод (CEM)

1. Инициализация среды.

```
import sys, os
import gym
import numpy as np
env = gym.make("Taxi-v3")
env.reset()
env.render()
n_states = env.observation_space.n
n_actions = env.action_space.n
print("n_states=%i, n_actions=%i" % (n_states, n_actions))
```

После запуска появляется окно с визуализацией модели и значениями ее параметров.

2. Создание стохастической политики

Политика должна быть вероятностным распределением.

policy[s,a] = P(выполнить действие a | в состоянии s)

Поскольку мы по-прежнему используем целочисленные представления состояний и действий, для представления политики можно использовать двумерный массив. Инициализируйте политику равномерно, то есть вероятности всех действий должны быть равными:

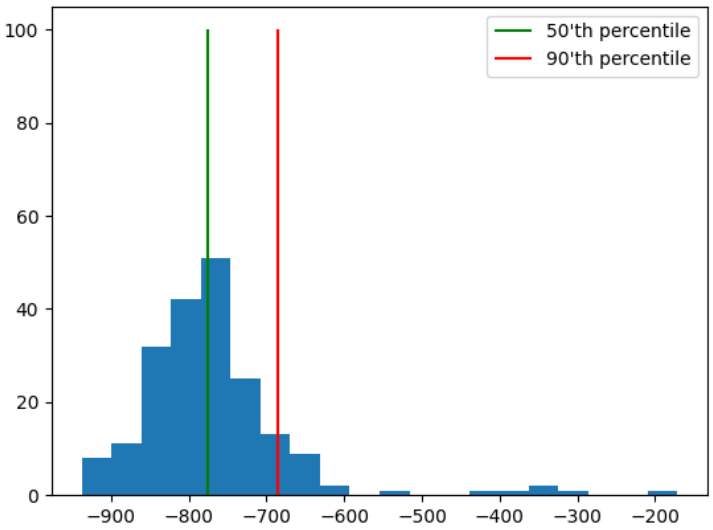
```
def initialize_policy(n_states, n_actions):
    < Ваш код: создать массив для хранения вероятности действий >
    return policy
policy = initialize_policy(n_states, n_actions)

assert type(policy) in (np.ndarray, np.matrix)
assert np.allclose(policy, 1./n_actions)
assert np.allclose(np.sum(policy, axis=1), 1)
```

3. Игра с моделью

```
def generate_session(env, policy, t_max=10**4):
    """
      Играть до конца или t_max тиков.
     :param policy: массив вида [n_states,n_actions] с вероятностями действий
     :returns: список состояний, список действий и сумма наград
    """
    states, actions = [], []
    total_reward = 0.

    s = env.reset()
    for t in range(t_max):
        # Hint: вы можете использовать np.random.choice для выборки
        # https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html
        a = <ВАШ КОД: пример действия из policy>
        new_s, r, done, info = env.step(a)
        # Запись информацию, которая получена из среды.
        states.append(s)
        actions.append(a)
        total_reward += r
        s = new_s
        if done:
            break
    return states, actions, total_reward

s, a, r = generate_session(env, policy)
assert type(s) == type(a) == list
assert len(s) == len(a)
assert type(r) in [float, np.float64]

# Визуализация начального распределения вознаграждения
import matplotlib.pyplot as plt
sample_rewards = [generate_session(env, policy, t_max=1000)[-1] for _ in range(200)]
```

```
plt.hist(sample_rewards, bins=20)
plt.vlines([np.percentile(sample_rewards, 50)], [0], [100], label="50'th percentile", color='green')
plt.vlines([np.percentile(sample_rewards, 90)], [0], [100], label="90'th percentile", color='red')
plt.legend()
plt.show()
```

Вывод:



3. Шаги кросс-энтропийного метода

```
def select_elites(states_batch, actions_batch, rewards_batch, percentile):
    """
    Выберите состояния и действия из игры, которые имеют награды >= процентиль
     :paramstates_batch: список списков состояний,states_batch[session_i][t]
     :param action_batch: список списков действий, action_batch[session_i][t]
     :param rewards_batch: список наград, rewards_batch[session_i]
    :returns: elite_states,elite_actions, одномерные списки состояний и соответствующих действий лучших сессий
    """
    reward_threshold = <ВАШ КОД: вычисление минимального вознаграждения для лучших сессий. Hint: используйте np.percentile()>
    elite_states = <ВАШ КОД>
    elite_actions = <ВАШ КОД>
    return elite_states, elite_actions

states_batch = [
    [1, 2, 3],     # игра1
```

```
    [4, 2, 0, 2],  # игра2
    [3, 1],        # игра3
]
actions_batch = [
    [0, 2, 4],     # игра1
    [3, 2, 0, 1],  # игра2
    [3, 3],        # игра3
]
rewards_batch = [
    3,  # игра1
    4,  # игра2
    5,  # игра3
]
test_result_0 = select_elites(states_batch, actions_batch, rewards_batch, percentile=0)
test_result_30 = select_elites(states_batch, actions_batch, rewards_batch, percentile=30)
test_result_90 = select_elites(states_batch, actions_batch, rewards_batch, percentile=90)
test_result_100 = select_elites(states_batch, actions_batch, rewards_batch, percentile=100)
assert np.all(test_result_0[0] == [1, 2, 3, 4, 2, 0, 2, 3, 1])  \
    and np.all(test_result_0[1] == [0, 2, 4, 3, 2, 0, 1, 3, 3]), \
    "Для процентиля 0 вы должны вернуть все состояния и действия в хронологическом порядке."
assert np.all(test_result_30[0] == [4, 2, 0, 2, 3, 1]) and \
    np.all(test_result_30[1] == [3, 2, 0, 1, 3, 3]), \
    "Для процентиля 30 вы должны выбрать состояния/действия только из двух первых"
assert np.all(test_result_90[0] == [3, 1]) and \
    np.all(test_result_90[1] == [3, 3]), \
    "Для процентиля 90 вы должны выбирать состояния/действия только из одной игры."
assert np.all(test_result_100[0] == [3, 1]) and\
    np.all(test_result_100[1] == [3, 3]), \
    "Убедитесь, что вы используете >=, а не >. Также дважды проверьте, как вы вычисляете процентиль."

def get_new_policy(elite_states, elite_actions):
    """
    Учитывая список лучших состояний/действий от select_elites,
     возвращает новую политику, где вероятность каждого действия пропорциональна
       policy[s_i,a_i] ~ #[появления s_i и a_i в элитарных состояниях/действиях]
    Не забудьте нормализовать политику, чтобы получить действительные вероятности и обработать случай 0/0.
     Для состояний, в которых вы никогда не находились, используйте равномерное распределение (1/n_actions для всех состояний).
     :param Elite_states: одномерный список состояний лучших сессий.
```

```
    :param Elite_actions: одномерный список действий лучших сессий.
    """

    from collections import defaultdict

        new_policy = np.zeros([n_states, n_actions])

    <ВАШ КОД: Установите вероятности для действий, которые привели к лучшим состояниям и действиям >
    # Не забыть выставить 1/n_действий для всех действий в неизвестных состояниях.
    return new_policy
```

4. Тренировочный цикл

Сгенерируйте сеансы, выберите N лучших и подстройтесь под них.

```
def show_progress(rewards_batch, log, percentile, reward_range=[-990, +10]):
    """
    Удобная функция, отображающая прогресс обучения
    """
    mean_reward = np.mean(rewards_batch)
    threshold = np.percentile(rewards_batch, percentile)
    log.append([mean_reward, threshold])
    plt.figure(figsize=[8, 4])
    plt.subplot(1, 2, 1)
    plt.plot(list(zip(*log))[0], label='Mean rewards')
    plt.plot(list(zip(*log))[1], label='Reward thresholds')
    plt.legend()
    plt.grid()
    plt.subplot(1, 2, 2)
    plt.hist(rewards_batch, range=reward_range)
    plt.vlines([np.percentile(rewards_batch, percentile)],
               [0], [100], label="percentile", color='red')
    plt.legend()
    plt.grid()
    clear_output(True)
    print("mean reward = %.3f, threshold=%.3f" % (mean_reward, threshold))
    plt.show()

# сбросить политику на всякий случай
policy = initialize_policy(n_states, n_actions)

#Эксперимент
n_sessions = 250     # число сессий
```

```
percentile = 50      # процент сессий с наивысшей наградой
learning_rate = 0.5  # насколько быстро обновляется политика, по шкале от 0 до 1
log = []
for i in range(100):
    sessions = [ <ВАШ КОД: генерирование списка n_sessions новых сессий> ]
    states_batch, actions_batch, rewards_batch = zip(*sessions)
    elite_states, elite_actions = <ВАШ КОД: выбор лучших состояний и действий >
    new_policy = <ВАШ КОД: вычисление нового policy>
    policy = learning_rate * new_policy + (1 - learning_rate) * policy
    # display results on chart
    show_progress(rewards_batch, log, percentile)
```

Задания к лабораторной работе

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

<ВАШ КОД: >

2. Запустить полученную программу, посмотреть на графиках процесс обучения.

3. Проанализировать, как сходится задача такси, и объяснить почему она быстро сходится от менее чем -1000 до почти оптимального значения, а затем снова снижается до -50/-100. Ответ приложить к отчету.

Требование к отчету

1. В качестве отчета принимается Python файл с кодом и текстовый файл с ответом на Задание № 3.
2. Код должен быть в достаточной мере прокомментирован.
3. Локальные датасеты также должны входить в отчет.
4. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip

Лабораторная работа №3. Марковский процесс принятия решений (MDP)

1. Подготовительные действия

В работе будет использован внешний модель mdp.py, который следует разместить в папке проекта с данной лабораторной работой. Ссылка для скачивания файла: https://github.com/yandexdataschool/Practical_RL/blob/master/week02_value_based/mdp.py

```
import sys, os
transition_probs = {
   's0': {
      'a0': {'s0': 0.5, 's2': 0.5},
      'a1': {'s2': 1}
   },
   's1': {
      'a0': {'s0': 0.7, 's1': 0.1, 's2': 0.2},
      'a1': {'s1': 0.95, 's2': 0.05}
   },
   's2': {
      'a0': {'s0': 0.4, 's2': 0.6},
      'a1': {'s0': 0.3, 's1': 0.3, 's2': 0.4}
   }
}
rewards = {
   's1': {'a0': {'s0': +5}},
   's2': {'a1': {'s0': -1}}
}
from mdp import MDP
mdp = MDP(transition_probs, rewards, initial_state='s0')
```

Теперь можно использовать MDP аналогично Gym среде.

```
print('initial state =', mdp.reset())
next_state, reward, done, info = mdp.step('a1')
print('next_state = %s, reward = %s, done = %s' % (next_state, reward, done))
```

Вывод:

```
initial state = s0

next_state = s2, reward = 0.0, done = False
```

У модуля также есть другие методы, которые понадобятся для итераций по значению.

```
print("mdp.get_all_states =", mdp.get_all_states())
print("mdp.get_possible_actions('s1') = ", mdp.get_possible_actions('s1'))
print("mdp.get_next_states('s1', 'a0') = ", mdp.get_next_states('s1', 'a0'))
print("mdp.get_reward('s1', 'a0', 's0') = ", mdp.get_reward('s1', 'a0', 's0'))
print("mdp.get_transition_prob('s1', 'a0', 's0') = ", mdp.get_transition_prob('s1', 'a0', 's0'))
```

Вывод:

mdp.get_all_states = ('s0', 's1', 's2')

mdp.get_possible_actions('s1') = ('a0', 'a1')

mdp.get_next_states('s1', 'a0') = {'s0': 0.7, 's1': 0.1, 's2': 0.2}

mdp.get_reward('s1', 'a0', 's0') = 5

mdp.get_transition_prob('s1', 'a0', 's0') = 0.7

Визуализация графа (не обязательно)

```
from mdp import has_graphviz
from IPython.display import display
print("Graphviz available:", has_graphviz)

if has_graphviz:
    from mdp import plot_graph, plot_graph_with_state_values,
plot_graph_optimal_strategy_and_state_values
    display(plot_graph(mdp))
```

2. Итерации по значениям

Итерации по значениям (Value Iterations или VI). Вот псевдокод для VI:

1. Initialize $V^{(0)}(s) = 0$, for all s
2. For $i = 0, 1, 2, \ldots$
3. $V_{(i+1)}(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')]$, for all s

Во-первых, давайте напишем функцию для вычисления функции значения состояния-действия Q^{π} , определенной следующим образом:

$$Q_i(s,a) = \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] \quad (1)$$

```
def get_action_value(mdp, state_values, state, action, gamma):
    """ Вычисляет Q(s,a) из формулы (1) описания лабораторной работы """
    <ВАШ КОД>
    return <ВАШ КОД>

import numpy as np
test_Vs = {s: i for i, s in enumerate(sorted(mdp.get_all_states()))}
assert np.isclose(get_action_value(mdp, test_Vs, 's2', 'a1', 0.9), 0.69)
assert np.isclose(get_action_value(mdp, test_Vs, 's1', 'a0', 0.9), 3.95)
```

Используя Q(s,a), теперь мы можем определить «следующую» V(s) для итерации значения.

$$V_{(i+1)}(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] = \max_a Q_i(s,a) \quad (2)$$

```
def get_new_state_value(mdp, state_values, state, gamma):
    """ Вычисление следующего V(s) по формуле (2). В процессе не меняйте state_values. """
    if mdp.is_terminal(state):
        return 0
    < ВАШ КОД >
    return < ВАШ КОД >
```

Наконец, объединим все написанные функции в рабочий алгоритм итерации значений.

```
# параметры
gamma = 0.9          # дисконт MDP
num_iter = 100       # максимальное число итераций, за исключением инициализации
# Останавливаем VI если новые значения ближе к старым менее чем:
min_difference = 0.001
# инициализация V(s)
state_values = {s: 0 for s in mdp.get_all_states()}
if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))
for i in range(num_iter):
    # Вычисление новых начений состояния, используя ранее написанные функции.
    # Они имеют формат словаря {state : float V_new(state)}
    new_state_values = <ВАШ КОД>
    assert isinstance(new_state_values, dict)
    # Вычисление отклонений
    diff = max(abs(new_state_values[s] - state_values[s])
               for s in mdp.get_all_states())
    print("iter %4i   |   diff: %6.5f   |   " % (i, diff), end="")
    print('   '.join("V(%s) = %.3f" % (s, v) for s, v in state_values.items()))
    state_values = new_state_values
    if diff < min_difference:
        print("Terminated")
        break

if has_graphviz:
    display(plot_graph_with_state_values(mdp, state_values))
print("Final state values:", state_values)
```

```
assert abs(state_values['s0'] - 3.781) < 0.01
assert abs(state_values['s1'] - 7.294) < 0.01
assert abs(state_values['s2'] - 4.202) < 0.01
```

Теперь давайте используем эти V∗(s) для поиска оптимальных действий в каждом состоянии.

$$\pi^*(s) = argmax_a \sum_{s'} P(s'|s,a) \cdot [r(s,a,s') + \gamma V_i(s')] = argmax_a Q_i(s,a) \quad (3)$$

Единственное отличие от V(s) в том, что здесь мы берем не max, а argmax: найти такое действие с максимальным Q(s,a).

```
def get_optimal_action(mdp, state_values, state, gamma=0.9):
   """ Находит оптимальное действие используя формулу (3). """
   if mdp.is_terminal(state):
      return None
   <ВАШ КОД>
   return <ВАШ КОД>
```

```
assert get_optimal_action(mdp, state_values, 's0', gamma) == 'a1'
assert get_optimal_action(mdp, state_values, 's1', gamma) == 'a0'
assert get_optimal_action(mdp, state_values, 's2', gamma) == 'a1'
assert get_optimal_action(mdp, {'s0': -1e10, 's1': 0, 's2': -2e10}, 's0', 0.9) == 'a0', \
   "Убедитесь, что вы правильно обрабатываете отрицательные значения Q произвольной
величины."
assert get_optimal_action(mdp, {'s0': -2e10, 's1': 0, 's2': -1e10}, 's0', 0.9) == 'a1', \
   "Убедитесь, что вы правильно обрабатываете отрицательные значения Q произвольной
величины."
```

```
# Измерение среднего вознаграждения агента
s = mdp.reset()
rewards = []
for _ in range(10000):
   s, r, done, _ = mdp.step(get_optimal_action(mdp, state_values, s, gamma))
   rewards.append(r)
print("average reward: ", np.mean(rewards))
assert(0.40 < np.mean(rewards) < 0.55)
```

Вывод:

```
iter   0   |   diff: 3.50000   |   V(s0) = 0.000   V(s1) = 0.000   V(s2) = 0.000

iter   1   |   diff: 0.64500   |   V(s0) = 0.000   V(s1) = 3.500   V(s2) = 0.000

iter   2   |   diff: 0.58050   |   V(s0) = 0.000   V(s1) = 3.815   V(s2) = 0.645
```

…

iter 57 | diff: 0.00110 | V(s0) = 3.779 V(s1) = 7.292 V(s2) = 4.200

iter 58 | diff: 0.00099 | V(s0) = 3.780 V(s1) = 7.293 V(s2) = 4.201

Terminated

Final state values: {'s0': 3.7810348735476405, 's1': 7.294006423867229, 's2': 4.202140275227048}

average reward: 0.4553

3. Среда Frozen lake

```
#Frozen Lake Env
from mdp import FrozenLakeEnv
mdp = FrozenLakeEnv(slip_chance=0)

mdp.render()
```

Вывод:

*FFF

FHFH

FFFH

HFFG

```
def value_iteration(mdp, state_values=None, gamma=0.9, num_iter=1000, min_difference=1e-5):
  """ выполняет шаги итерации значения num_iter, начиная с state_values. То же, что и раньше, но в функции """
  state_values = state_values or {s: 0 for s in mdp.get_all_states()}
  for i in range(num_iter):
    # Вычислите новые значения состояния, используя функции, которые вы определили выше. Это должен быть dict {state: new_V(state)}
    new_state_values = <ВАШ КОД>
    assert isinstance(new_state_values, dict)
    # Вычисление отклонений
    diff = max(abs(new_state_values[s] - state_values[s])
           for s in mdp.get_all_states())
    print("iter %4i   |   diff: %6.5f   |   V(start): %.3f " %
        (i, diff, new_state_values[mdp._initial_state]))
    state_values = new_state_values
    if diff < min_difference:
```

```
            break
    return state_values

state_values = value_iteration(mdp)

s = mdp.reset()
mdp.render()
for t in range(100):
    a = get_optimal_action(mdp, state_values, s, gamma)
    print(a, end='\n\n')
    s, r, done, _ = mdp.step(a)
    mdp.render()
    if done:
        break
```

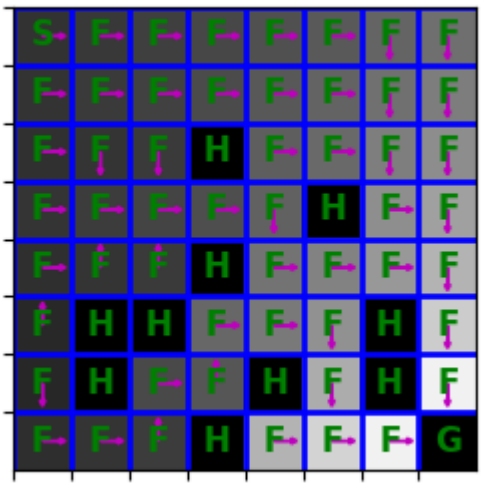
Визуализация. Обычно интересно посмотреть, что алгоритм на самом деле умеет. Для этого мы нанесем на график функции значений состояния и оптимальные действия на каждом шаге VI.

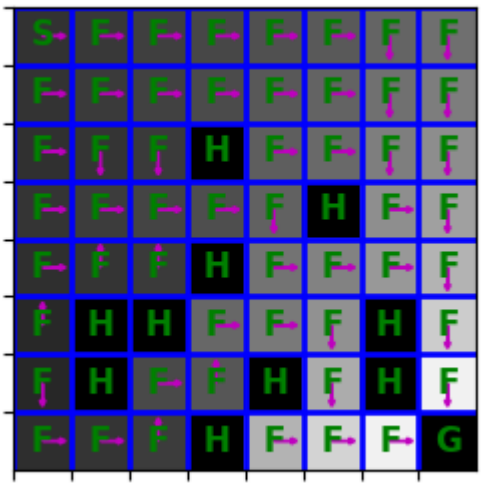
```
import matplotlib.pyplot as plt
def draw_policy(mdp, state_values):
    plt.figure(figsize=(3, 3))
    h, w = mdp.desc.shape
    states = sorted(mdp.get_all_states())
    V = np.array([state_values[s] for s in states])
    Pi = {s: get_optimal_action(mdp, state_values, s, gamma) for s in states}
    plt.imshow(V.reshape(w, h), cmap='gray', interpolation='none', clim=(0, 1))
    ax = plt.gca()
    ax.set_xticks(np.arange(h)-.5)
    ax.set_yticks(np.arange(w)-.5)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    Y, X = np.mgrid[0:4, 0:4]
    a2uv = {'left': (-1, 0), 'down': (0, -1), 'right': (1, 0), 'up': (0, 1)}
    for y in range(h):
        for x in range(w):
            plt.text(x, y, str(mdp.desc[y, x].item()),
                     color='g', size=12,  verticalalignment='center',
                     horizontalalignment='center', fontweight='bold')
            a = Pi[y, x]
            if a is None:
                continue
```

```
        u, v = a2uv[a]
        plt.arrow(x, y, u*.3, -v*.3, color='m',
                  head_width=0.1, head_length=0.1)
    plt.grid(color='b', lw=2, ls='-')
    plt.show()

state_values = {s: 0 for s in mdp.get_all_states()}
for i in range(10):
    print("after iteration %i" % i)
    state_values = value_iteration(mdp, state_values, num_iter=1)
    draw_policy(mdp, state_values)
# please ignore iter 0 at each step

from IPython.display import clear_output
from time import sleep
mdp = FrozenLakeEnv(map_name='8x8', slip_chance=0.1)
state_values = {s: 0 for s in mdp.get_all_states()}
for i in range(30):
    clear_output(True)
    print("after iteration %i" % i)
    state_values = value_iteration(mdp, state_values, num_iter=1)
    draw_policy(mdp, state_values)
    sleep(0.5)
# please ignore iter 0 at each step
```

Вывод:

```
after iteration 0

iter    0   |   diff: 1.00000   |   V(start): 0.000

after iteration 1

iter    0   |   diff: 0.90000   |   V(start): 0.000

after iteration 2

iter    0   |   diff: 0.81000   |   V(start): 0.000

…

after iteration 28

iter    0   |   diff: 0.00000   |   V(start): 0.198

after iteration 29

iter    0   |   diff: 0.00000   |   V(start): 0.198
```





Массовые испытания.

```
mdp = FrozenLakeEnv(slip_chance=0)
state_values = value_iteration(mdp)
total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))
print("average reward: ", np.mean(total_rewards))
assert(1.0 <= np.mean(total_rewards) <= 1.0)
print("Well done!")

# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.1)
state_values = value_iteration(mdp)
total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
```

```
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))
print("average reward: ", np.mean(total_rewards))
assert(0.8 <= np.mean(total_rewards) <= 0.95)
print("Well done!")

# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.25)
state_values = value_iteration(mdp)
total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))
print("average reward: ", np.mean(total_rewards))
assert(0.6 <= np.mean(total_rewards) <= 0.7)
print("Well done!")

# Значение среднего вознаграждения агента
mdp = FrozenLakeEnv(slip_chance=0.2, map_name='8x8')
state_values = value_iteration(mdp)
total_rewards = []
for game_i in range(1000):
    s = mdp.reset()
    rewards = []
    for t in range(100):
        s, r, done, _ = mdp.step(
            get_optimal_action(mdp, state_values, s, gamma))
        rewards.append(r)
        if done:
            break
    total_rewards.append(np.sum(rewards))
```

```
print("average reward: ", np.mean(total_rewards))
assert(0.6 <= np.mean(total_rewards) <= 0.8)
print("Well done!")
```

Вывод (пример по одному из тестов):

```
iter    0   |   diff: 0.75000   |   V(start): 0.000

iter    1   |   diff: 0.50625   |   V(start): 0.000

…

iter   19   |   diff: 0.00003   |   V(start): 0.325

iter   20   |   diff: 0.00002   |   V(start): 0.325

iter   21   |   diff: 0.00001   |   V(start): 0.325

average reward:  0.631

Well done!
```

Задания к лабораторной работе

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

<ВАШ КОД: >

2. Запустить полученную программу, получить ожидаемые выходные данные.

Требование к отчету

1. В качестве отчета принимается Python файл с кодом.
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

Лабораторная работа №4. Q-обучение

1. Подготовительные действия

```
import sys, os
# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'
import numpy as np
```

```
import matplotlib.pyplot as plt
from collections import defaultdict
import random
import math
import numpy as np
```

2. Класс QLearningAgent

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

```
class QLearningAgent:
    def __init__(self, alpha, epsilon, discount, get_legal_actions):
        """
        Q-Learning Agent
        based on https://inst.eecs.berkeley.edu/~cs188/sp19/projects.html
        Переменные экземпляра, к которым у вас есть доступ
          - self.epsilon (исследование)
          - self.alpha (скорость обучения)
          - self.discount (дисконт, она же гамма)
        Функции, которые вы должны использовать
          - self.get_legal_actions(state) {состояние, хешируемое -> список действий, каждое из которых хешируемое}
            который возвращает разрешенные действия для состояния
          - self.get_qvalue (состояние, действие)
            который возвращает Q (состояние, действие)
          - self.set_qvalue (состояние, действие, значение)
            который устанавливает Q (состояние, действие): = значение
        !!!Важно!!!
        Примечание: пожалуйста, избегайте прямого использования self._qValues.
            Для этого есть специальный self.get_qvalue/set_qvalue.
        """
        self.get_legal_actions = get_legal_actions
        self._qvalues = defaultdict(lambda: defaultdict(lambda: 0))
        self.alpha = alpha
        self.epsilon = epsilon
        self.discount = discount
    def get_qvalue(self, state, action):
        """ Возвращает Q(state,action) """
        return self._qvalues[state][action]
    def set_qvalue(self, state, action, value):
        """ Устанавливает Qvalue для [state,action] в определенное значение """
```

```
        self._qvalues[state][action] = value
    #---------------------ВЫ НАЧИНАЕТЕ ДОБАВЛЯТЬ СВОЙ КОД С ЭТОГО МЕСТА---------------------#
    def get_value(self, state):
        """
        Вычислите оценку вашего агента V (s), используя текущие значения q.
         V(s) = max_over_action Q(состояние,действие) по возможным действиям.
         Примечание: обратите внимание, что значения q могут быть отрицательными.
        """
        possible_actions = self.get_legal_actions(state)
        # If there are no legal actions, return 0.0
        if len(possible_actions) == 0:
            return 0.0
        <ВАШ КОД>
        return value
    def update(self, state, action, reward, next_state):
        """
        Необходимо обновить значение Q-Value:
          Q(s,a) := (1 - alpha) * Q(s,a) + alpha * (r + gamma * V(s'))
        """
        # agent parameters
        gamma = self.discount
        learning_rate = self.alpha
        <ВАШ КОД>
        self.set_qvalue(state, action, <ВАШ КОД: Q-value> )
    def get_best_action(self, state):
        """
        Вычислите наилучшее действие для состояния (используя текущие значения q).
        """
        possible_actions = self.get_legal_actions(state)
        # If there are no legal actions, return None
        if len(possible_actions) == 0:
            return None
        <ВАШ КОД>
        return best_action
    def get_action(self, state):
        """
        Вычислите действие, которое нужно предпринять в текущем состоянии, включая
исследование.
         С вероятностью self.epsilon мы должны предпринять случайное действие.
             иначе - лучшее действие политики (self.get_best_action).
         Примечание. Чтобы выбрать случайным образом из списка, используйте random.choice(list).
```

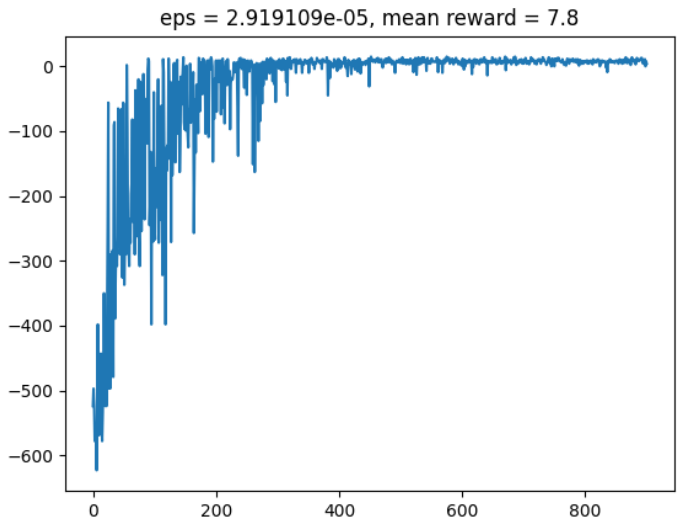
```
        Чтобы выбрать True или False с заданной вероятностью, сгенерируйте универсальное число в [0, 1]
        и сравните с вашей вероятностью
    """
    # Pick Action
    possible_actions = self.get_legal_actions(state)
    action = None
    # If there are no legal actions, return None
    if len(possible_actions) == 0:
       return None
    # agent parameters:
    epsilon = self.epsilon
    <ВАШ КОД>
    return chosen_action
```

Попробуем на среде такси. Здесь мы используем агент *qlearning* на такси *env* от *openai gym*. Вам нужно будет вставить сюда несколько функций агента.

```
import gym
env = gym.make("Taxi-v3")
n_actions = env.action_space.n

agent = QLearningAgent(
   alpha=0.5, epsilon=0.25, discount=0.99,
   get_legal_actions=lambda s: range(n_actions))

def play_and_train(env, agent, t_max=10**4):
   """
   Эта функция должна
    - запустить полную игру, действия заданы политикой e-greeding агента
    - обучать агента, используя agent.update(...) всякий раз, когда это возможно
    - вернуть общую награду
   """
   total_reward = 0.0
   s = env.reset()
   for t in range(t_max):
      # get agent to pick action given state s.
      a = <ВАШ КОД>
      next_s, r, done, _ = env.step(a)
      # train (update) agent for state s
      <ВАШ КОД>
      s = next_s
      total_reward += r
```

```
        if done:
            break
    return total_reward

from IPython.display import clear_output
rewards = []
for i in range(1000):
    rewards.append(play_and_train(env, agent))
    agent.epsilon *= 0.99
    if i % 100 == 0:
        clear_output(True)
        plt.title('eps = {:e}, mean reward = {:.1f}'.format(agent.epsilon, np.mean(rewards[-10:])))
        plt.plot(rewards)
        plt.show()
```

Вывод:



3. Бинаризованные пространства состояний

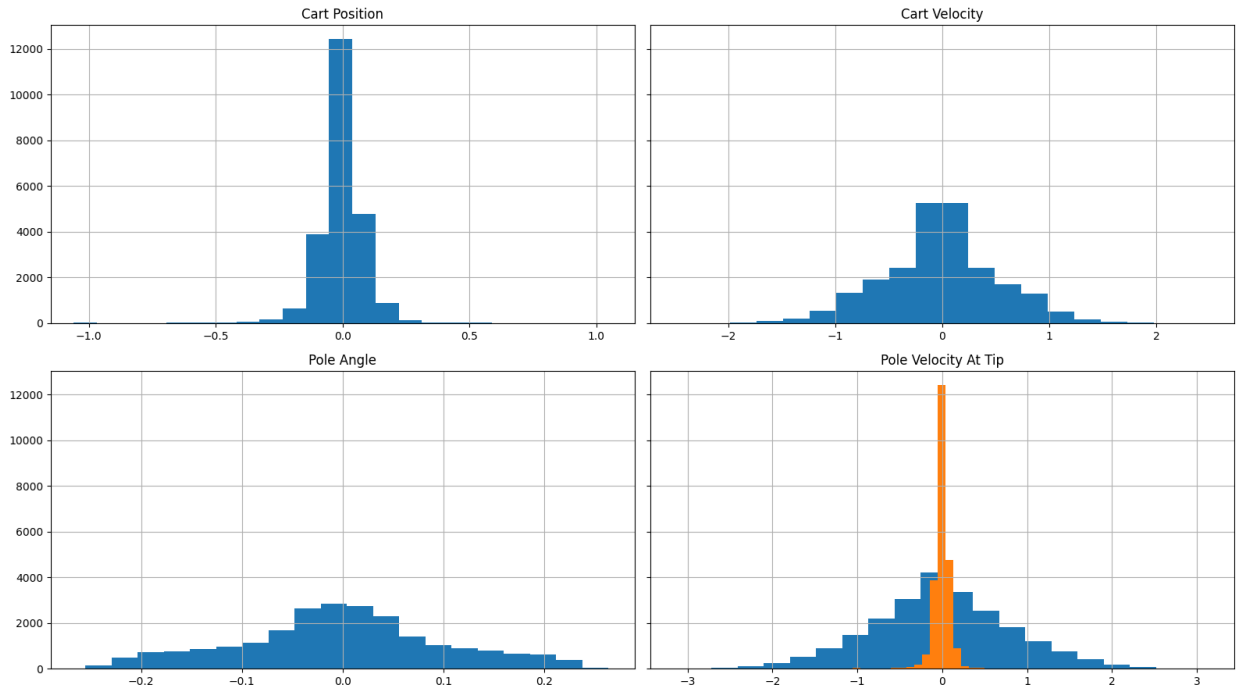
Используйте агент для эффективного обучения на CartPole-v0. Эта среда имеет непрерывный набор возможных состояний, поэтому вам придется каким-то образом сгруппировать их в бины. Самый простой способ — использовать round(x, n_digits) (или np.round) для округления действительного числа до заданного количества цифр. Сложность заключается в том, чтобы правильно подобрать n_digits для каждого состояния для эффективного обучения. Обратите внимание, что вам нужно преобразовывать состояние не в целые числа, а в кортежи любых значений.

```
def make_env():
    return gym.make('CartPole-v0').env  # .env unwraps the TimeLimit wrapper
env = make_env()
n_actions = env.action_space.n
print("first state: %s" % (env.reset()))
plt.imshow(env.render('rgb_array'))
```

Нам нужно оценить распределения наблюдений. Для этого мы сыграем несколько игр и запишем все состояния.

```
seen_observations = []
for _ in range(1000):
    seen_observations.append(env.reset())
    done = False
    while not done:
        s, r, done, _ = env.step(env.action_space.sample())
        seen_observations.append(s)
seen_observations = np.array(seen_observations)
for obs_i in range(env.observation_space.shape[0]):
    plt.hist(seen_observations[:, obs_i], bins=20)
    plt.show()
```

Вывод:



4. Бинаризованная среда

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

```
from gym.core import ObservationWrapper
class Binarizer(ObservationWrapper):
    def observation(self, state):
        # Hint: Используйте round(x, n_digits).
        # Вы можете выбрать разные n_digits для каждого измерения..
        state = <ВАШ КОД: округлите состояние до нескольких значимых цифр>
        return tuple(state)

env2 = Binarizer(gym.make("CartPole-v0")).env

seen_observations = []
for _ in range(1000):
    seen_observations.append(env.reset())
    done = False
    while not done:
        s, r, done, _ = env.step(env.action_space.sample())
        seen_observations.append(s)
        if done:
            break
seen_observations = np.array(seen_observations)
for obs_i in range(env.observation_space.shape[0]):
    plt.hist(seen_observations[:, obs_i], bins=20)
    plt.show()
```

Задания к лабораторной работе

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

<ВАШ КОД: >

2. Запустить полученную программу, получить ожидаемые выходные данные.

Требование к отчету

1. В качестве отчета принимается Python файл с кодом.
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

Лабораторная работа №5. Построение нейронных сетей для Q-обучения с помощью PyTourch и Tensorflow

1. Использование Tensorflow

В этом разделе вы научите нейронную сеть Tensorflow выполнять Q-обучение.

```
import sys, os
# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'

import gym
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

env = gym.make("CartPole-v0").env
env.reset()
n_actions = env.action_space.n
state_dim = env.observation_space.shape
plt.imshow(env.render("rgb_array"))

import tensorflow as tf
import keras
import keras.layers as L
tf.compat.v1.reset_default_graph()
sess = tf.compat.v1.InteractiveSession()
keras.backend.set_session(sess)
assert not tf.test.is_gpu_available(), \
    "Пожалуйста, выполните это задание без графического процессора. Если вы используете графический процессор, код " \
    "будет работать намного медленнее из-за большого количества операций копирования в память графического процессора и из нее." \
    "Чтобы отключить графический процессор в Colab, выберите «Runtime» → «Change runtime type» → «None»."
```

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

```
network = keras.models.Sequential()
network.add(L.InputLayer(state_dim))
<ВАШ КОД: создайте сеть для приблизительного q-обучения>

def get_action(state, epsilon=0):
    """
    примеры действий с эпсилон-жадной политикой
     Резюме: с p = epsilon выберите случайное действие, иначе выберите действие с наибольшим Q
(s, a)
    """
    q_values = network.predict(state[None])[0]
    < ВАШ КОД >
    return < ВАШ КОД: эпсилон - жадно выбранное действие >
```

Проверка:

```
assert network.output_shape == (None, n_actions), "пожалуйста, убедитесь, что ваша модель
отображает состояние s -> [Q(s,a0),..., Q(s, a_last)]"
assert network.layers[-1].activation == keras.activations.linear, "пожалуйста, убедитесь, что вы
предсказываете q-значения без нелинейности"
# проверка эпсилон-жадных исследований
s = env.reset()
assert np.shape(get_action(s)) == (), "верните только одно действие (integer)"
for eps in [0., 0.1, 0.5, 1.0]:
    state_frequencies = np.bincount([get_action(s, epsilon=eps) for i in range(10000)],
minlength=n_actions)
    best_action = state_frequencies.argmax()
    assert abs(state_frequencies[best_action] - 10000 * (1 - eps + eps / n_actions)) < 200
    for other_action in range(n_actions):
        if other_action != best_action:
            assert abs(state_frequencies[other_action] - 10000 * (eps / n_actions)) < 200
    print('e=%.1f tests passed'%eps)
```

Вывод:

```
e=0.0 tests passed

e=0.1 tests passed

e=0.5 tests passed
```

Q-обучение через градиентный спуск

Теперь мы будем обучать Q-функцию нашего агента, минимизируя потери TD:

$$L = \frac{1}{N}\sum_i (Q_\theta(s,a) - [r(s,a) + \gamma \cdot max_{a'} Q_-(s',a')])^2$$

Где

- s,a,r,s′ — текущее состояние, действие, вознаграждение и следующее состояние соответственно.
- γ — коэффициент дисконтирования, определенный двумя ячейками выше.

Сложная часть связана с $Q^-(s',a')$. С инженерной точки зрения это то же самое, что и Q_θ — результат политики вашей нейронной сети. Однако при градиентном спуске мы не будем распространять через него градиенты, чтобы сделать обучение более стабильным.Для этого мы будем использовать функцию tf.stop_gradient, которая говорит: «Считайте эту вещь постоянной при выполнении backprop».

```
# Создание плейсхолдеров для <s, a, r, s'> кортежа и специального индикатора окончания игры (is_done = True)
states_ph = keras.backend.placeholder(dtype='float32', shape=(None,) + state_dim)
actions_ph = keras.backend.placeholder(dtype='int32', shape=[None])
rewards_ph = keras.backend.placeholder(dtype='float32', shape=[None])
next_states_ph = keras.backend.placeholder(dtype='float32', shape=(None,) + state_dim)
is_done_ph = keras.backend.placeholder(dtype='bool', shape=[None])

#задание q-значений для всех действий в текущем состоянии
predicted_qvalues = network(states_ph)
#выборка q-значений для выбранных действий
predicted_qvalues_for_actions = tf.reduce_sum(predicted_qvalues * tf.one_hot(actions_ph, n_actions), axis=1)
```

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

```
gamma = 0.99
# вычислить q-значения для всех действий в следующих состояниях
predicted_next_qvalues = <ВАШ КОД: применить ИНС для получения q-значений для next_states_ph>
# вычислить V * (next_states), используя предсказанные следующие q-значения
next_state_values = <ВАШ КОД>
# вычислить «целевые q-значения» для потерь — это то, что находится внутри квадратных скобок в приведенной выше формуле
target_qvalues_for_actions = <ВАШ КОД>
# в последнем состоянии будем использовать упрощенную формулу: Q(s,a) = r(s,a), так как s' не существует
target_qvalues_for_actions = tf.where(is_done_ph, rewards_ph, target_qvalues_for_actions)
```

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

```
#потери среднеквадратичной ошибки для минимизации
loss = (predicted_qvalues_for_actions - tf.stop_gradient(target_qvalues_for_actions)) ** 2
loss = tf.reduce_mean(loss)
# обучающая функция, похожая на agent.update(state, action, reward, next_state) из табличного агента
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)

assert tf.gradients(loss, [predicted_qvalues_for_actions])[0] is not None, "убедитесь, что вы обновляете q-значения для выбранных действий, а не только для всех действий"
assert tf.gradients(loss, [predicted_next_qvalues])[0] is None, "убедитесь, что вы не распространяете градиент w.r.t. Q_ (c', a')"
assert predicted_next_qvalues.shape.ndims == 2, "убедитесь, что вы предсказали значения q для всех действий в следующем состоянии"
assert next_state_values.shape.ndims == 1, "убедитесь, что вы вычислили V (s') как максимум только по оси действий, а не по всем осям"
assert target_qvalues_for_actions.shape.ndims == 1, "что-то не так с целевыми значениями q, они должны быть вектором"
```

Игры с моделью.

```
sess.run(tf.global_variables_initializer())

def generate_session(env, t_max=1000, epsilon=0, train=False):
    """ играть с env с приблизительным агентом q-обучения и одновременно тренировать его """
    total_reward = 0
    s = env.reset()
    for t in range(t_max):
        a = get_action(s, epsilon=epsilon)
        next_s, r, done, _ = env.step(a)
        if train:
            sess.run(train_step, {
                states_ph: [s], actions_ph: [a], rewards_ph: [r],
                next_states_ph: [next_s], is_done_ph: [done]
            })
        total_reward += r
        s = next_s
        if done:
            break
    return total_reward
```

```
epsilon = 0.5

for i in range(1000):
    session_rewards = [generate_session(env, epsilon=epsilon, train=True) for _ in range(100)]
    print("epoch #{}\tmean reward = {:.3f}\tepsilon = {:.3f}".format(i, np.mean(session_rewards), epsilon))
    epsilon *= 0.99
    assert epsilon >= 1e-4, " Убедитесь, что эпсилон всегда отличен от нуля во время обучения "
    if np.mean(session_rewards) > 300:
        print("You Win!")
        break
```

Вывод:

```
epoch #0    mean reward = 13.440    epsilon = 0.500
epoch #1    mean reward = 13.980    epsilon = 0.495
epoch #2    mean reward = 13.320    epsilon = 0.490
epoch #3    mean reward = 15.910    epsilon = 0.485
…
```

Не ждите, что вознаграждение агента будет плавно расти. Учтите следующее:

- __ среднее вознаграждение__ — это среднее вознаграждение за игру. Для правильной реализации он может оставаться низким в течение каких-то 10 эпох, затем начать расти, сильно колеблясь, и сходится на ~50-100 шагов в зависимости от архитектуры сети. Если оно не достигнет целевого значения к концу цикла for, попробуйте увеличить количество скрытых нейронов или посмотрите на эпсилон.
- __ epsilon__ — готовность агента исследовать. Если вы видите, что значение этого агента уже < 0,01 эпсилон до того, как оно станет как минимум 200, просто сбросьте его обратно на 0,1–0,5.

Задания к лабораторной работе №1

1. Написать свой код к разделу «*Использование Tensorflow*» 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 раздела «*Использование Tensorflow*».

Требование к отчету

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

Лабораторная работа №6. Простой policy gradient алгоритм (REINFORCE)

1. Использование Tensorflow

Как и раньше для Q-обучения мы разработаем сеть TensorFlow для изучения CartPole-v0 с помощью градиента политик (REINFORCE).

```
import sys, os
# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'

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

env = gym.make("CartPole-v0")
# gym compatibility: unwrap TimeLimit
if hasattr(env, '_max_episode_steps'):
    env = env.env
env.reset()
n_actions = env.action_space.n
state_dim = env.observation_space.shape
plt.imshow(env.render("rgb_array"))
```

Для алгоритма REINFORCE нам понадобится модель, которая предсказывает вероятности действий при заданных состояниях. Для стабильной работы не включайте слой softmax в вашу сетевую архитектуру. Мы будем использовать softmax или log-softmax, где это уместно.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
sess = tf.InteractiveSession()

# создать входные переменные. Нам нужны только <s, a, r> для REINFORCE
ph_states = tf.placeholder('float32', (None,) + state_dim, name="states")
ph_actions = tf.placeholder('int32', name="action_ids")
ph_cumulative_rewards = tf.placeholder('float32', name="cumulative_returns")
```

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

```
import keras
import keras.layers as L

<ВАШ КОД: определить сетевой граф, используя TF>
logits = <ВАШ КОД: символьный вывод ИНС _before_ softmax>
policy = tf.nn.softmax(logits)
log_policy = tf.nn.log_softmax(logits)

# инициализация параметров модели
sess.run(tf.global_variables_initializer())

def predict_probs(states):
    """
    Прогнозировать вероятности действий при заданных состояниях.
     :paramstates: numpy массив форм [batch, state_shape]
     :returns: numpy массив формы [пакет, n_actions]
    """
    return policy.eval({ph_states: [states]})[0]
```

Игра с моделью.

Теперь мы можем использовать нашего недавно созданного агента для игры. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
def generate_session(env, t_max=1000):
    """
    Сыграйте полную сессию с агентом REINFORCE.
     Возвращает последовательности состояний, действий и наград.
    """
    # массивы для записи сессии
    states, actions, rewards = [], [], []
    s = env.reset()
```

```
    for t in range(t_max):
        # массив вероятностей действий pi(a|s)
        action_probs = predict_probs(s)
        # Пример действия с заданной вероятностью
        a = <ВАШ КОД>
        new_s, r, done, info = env.step(a)
        # записать историю сессий, для последующего обучения
        states.append(s)
        actions.append(a)
        rewards.append(r)
        s = new_s
        if done:
            break
    return states, actions, rewards

# Проверка
states, actions, rewards = generate_session(env)
```

Расчет кумулятивных вознаграждений.

$$
\begin{aligned}
G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\
&= \sum_{i=t}^{T} \gamma^{i-t} r_i \\
&= r_t + \gamma * G_{t+1}
\end{aligned}
$$

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

```
def get_cumulative_rewards(rewards,  # вознаграждение на каждом шаге
                           gamma=0.99  # дисконт для вознаграждение
                           ):
    """
    Возьмите список немедленных вознаграждений r(s,a) за всю сессию
     и вычислить кумулятивную доходность (также известную как G(s,a) в Sutton '16).
     G_t = r_t + гамма*r_{t+1} + гамма^2*r_{t+2} + ...
     Простой способ вычислить кумулятивное вознаграждение состоит в том, чтобы выполнить итерацию, начиная с последнего к первому временному шагу и рекуррентно вычислить G_t = r_t + gamma*G_{t+1}
     Вы должны вернуть массив/список кумулятивных вознаграждений с таким же количеством элементов, как и в начальных вознаграждениях.
```

```
    """
    < ВАШ КОД >
    return < ВАШ КОД: массив кумулятивных вознаграждений>


assert len(get_cumulative_rewards(range(100))) == 100
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, 0, 0, 1, 0], gamma=0.9),
    [1.40049, 1.5561, 1.729, 0.81, 0.9, 1.0, 0.0])
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, -2, 3, -4, 0], gamma=0.5),
    [0.0625, 0.125, 0.25, -1.5, 1.0, -4.0, 0.0])
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, 2, 3, 4, 0], gamma=0),
    [0, 0, 1, 2, 3, 4, 0])
print("looks good!")
```

Функция потерь и обновления.

Теперь нам нужно определить цель и обновить градиент политики.

Наша целевая функция:

$$J \approx \frac{1}{N} \sum_{s_i, a_i} G(s_i, a_i)$$

REINFORCE определяет способ вычисления градиента ожидаемого вознаграждения по отношению к параметрам политики. Формула выглядит следующим образом:

$$\nabla_\theta \hat{J}(\theta) \approx \frac{1}{N} \sum_{s_i, a_i} \nabla_\theta \log \pi_\theta(a_i \mid s_i) \cdot G_t(s_i, a_i)$$

Мы можем использовать возможности Tensorflow для автоматического дифференцирования, определяя нашу целевую функцию следующим образом:

$$\hat{J}(\theta) \approx \frac{1}{N} \sum_{s_i, a_i} \log \pi_\theta(a_i \mid s_i) \cdot G_t(s_i, a_i)$$

Когда вы вычисляете градиент этой функции по отношению к весам сети θ , он станет именно градиентом политики.

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

```
# Этот код выбирает логарифмические вероятности (log pi(a_i|s_i)) для тех действий, которые были фактически сыграны.
indices = tf.stack([tf.range(tf.shape(log_policy)[0]), ph_actions], axis=-1)
log_policy_for_actions = tf.gather_nd(log_policy, indices)

# Цель политики, как в последней формуле. Пожалуйста, используйте reduce_mean, а не reduce_sum.
# Вы можете использовать log_policy_for_actions для получения логарифмических вероятностей предпринятых действий.
# Также помните, что ранее мы определили ph_cumulative_rewards.
J = <ВАШ КОД>
```

Напоминаем, что для дискретного распределения вероятностей (подобного тому, которое дает наша политика) энтропия определяется как:

$$\text{entropy}(p) = -\sum_{i=1}^{n} p_i \cdot \log p_i$$

```
# Регуляризация энтропии. Если вы его не добавите, политика быстро испортится до
# будучи детерминированным, вредит исследованию.
entropy = <ВАШ КОД: вычислить энтропию. Не забудьте знак!>

#Максимизация X аналогична минимизации -X, отсюда и знак.
loss = -(J + 0.1 * entropy)

def train_on_session(states, actions, rewards, t_max=1000):
    """в полном объеме обучает агента градиенту политики"""
    cumulative_rewards = get_cumulative_rewards(rewards)
    update.run({
        ph_states: states,
        ph_actions: actions,
        ph_cumulative_rewards: cumulative_rewards,
    })
    return sum(rewards)

# Инициализация оптимизированных параметров
sess.run(tf.global_variables_initializer())
```

Актуальное обучение

```
for i in range(100):
    rewards = [train_on_session(*generate_session(env)) for _ in range(100)]  # создание новой сессии
    print("mean reward: %.3f" % (np.mean(rewards)))
    if np.mean(rewards) > 300:
        print("You Win!")  # но обучение может быть продолжено
        break
```

Вывод:

looks good!

mean reward: 26.530

mean reward: 25.540

mean reward: 33.940

mean reward: 65.190

mean reward: 109.290

mean reward: 175.360

mean reward: 232.870

mean reward: 248.980

mean reward: 159.860

mean reward: 230.800

mean reward: 459.320

You Win!

Задания к лабораторной работе №1

1. Написать свой код к разделу «*Использование Tensorflow*» 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 раздела «*Использование Tensorflow*».

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

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

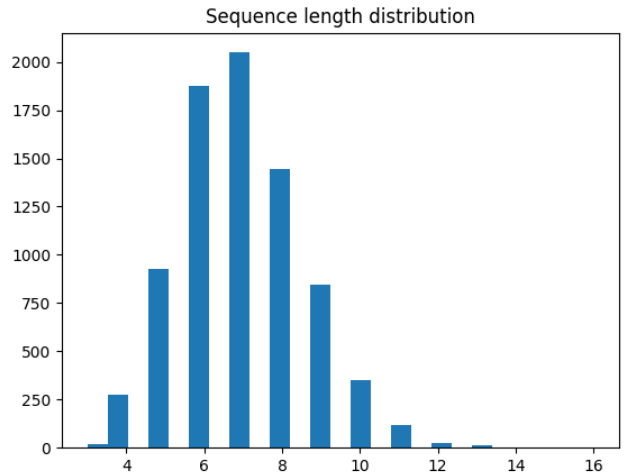
Prissie

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:

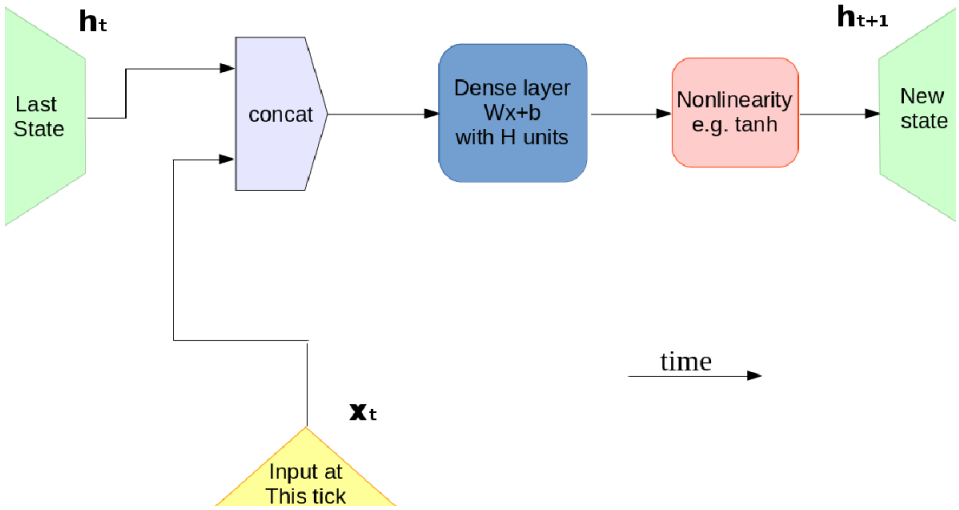
```
Abagael

 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)
```

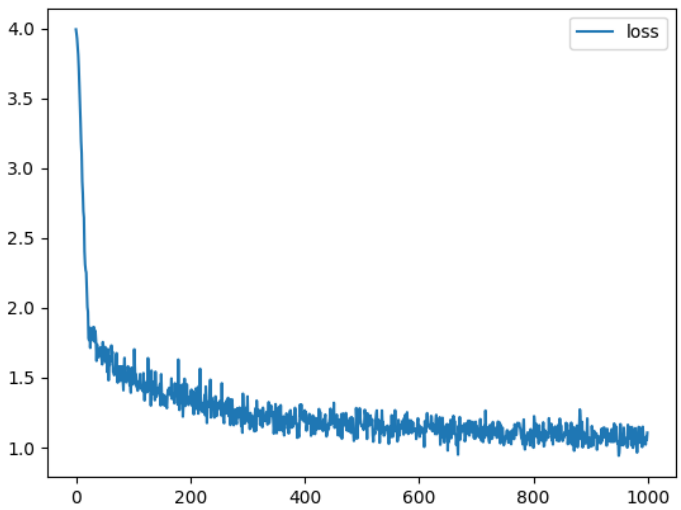
Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# 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

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

Лабораторная работа №8. Обучение с подкреплением для seq2seq

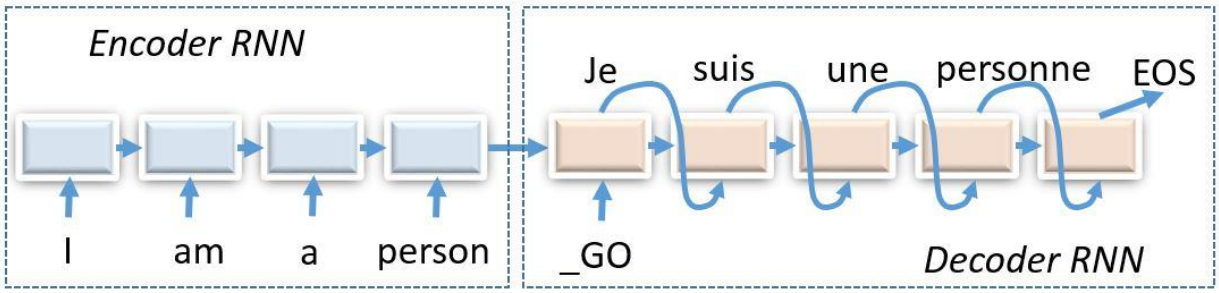
На этот раз мы решим задачу транскрипции слов на иврите на английском языке, также известную как g2p (grapheme2phoneme) слово (последовательность букв на исходном языке) -> перевод (последовательность букв на языке перевода). В отличие от того, что делают большинство методов глубокого обучения, мы не только будем

обучать модель для максимизации вероятности правильного перевода, но и использовать обучение с подкреплением, чтобы фактически научить модель переводить с минимальным количеством ошибок.

О задаче.

Одним из примечательных свойств иврита является то, что это согласный язык. То есть в письменном языке нет гласных. Можно представить гласные с диакритическими знаками над согласными, но никто не ожидает, что люди будут делать это в повседневной жизни.

Таким образом, некоторые символы иврита будут соответствовать нескольким английским буквам, а другие — ни одной, поэтому мы должны использовать архитектуру кодировщик-декодер, чтобы понять это.



Архитектуры кодер-декодер предназначены для преобразования чего угодно во что угодно, в том числе:

- Системы машинного перевода и разговорного диалога
- Подписи к изображениям и image2latex (сверточный кодировщик, рекуррентный декодер)
- Генерация изображений по подписям (рекуррентный кодировщик, сверточный декодер)
- Grapheme2phoneme - конвертировать слова в транскрипты

Мы выбрали упрощенный машинный перевод с иврита на английский для слов и коротких фраз (на уровне символов), так как его можно относительно быстро обучить даже без кластера графического процессора.

1. Предварительные действия

В работе используется файлы:

- «main_dataset.txt», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/main_dataset.txt
- «basic_model_tf.py», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/basic_model_tf.py
- «», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/basic_model_torch.py

```
import sys, os, time
# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")) == 0:
   os.environ['DISPLAY'] = ':1'

EASY_MODE = True       #Если True, переводит только фразы короче 20 символов (намного проще).
                       #Полезно для начального кодирования.
                       #Если false, работает со всеми фразами
MODE = "he-to-en"                       #способ перевода. Или "he-to-en" или "en-to-he"
MAX_OUTPUT_LENGTH = 50 if not EASY_MODE else 20  #максимальная длина _generated_ вывода, не влияет на обучение
REPORT_FREQ      = 100                  #насколько часто оценивать меру проверки
```

2. Препроцессинг

Мы будем хранить набор данных в виде словаря {слово1:[перевод1,перевод2,...], слово2:[...],...}. В основном это связано с тем, что многие слова имеют несколько правильных переводов. Эта функциональность уже реализована.

```
import numpy as np
from collections import defaultdict
word_to_translation = defaultdict(list)  # наш словарь
bos = '_'
eos = ';'
with open("main_dataset.txt", encoding='utf8') as fin:
   for line in fin:
      en, he = line[:-1].lower().replace(bos, ' ').replace(eos,
                                          ' ').split('\t')
      word, trans = (he, en) if MODE == 'he-to-en' else (en, he)
      if len(word) < 3:
         continue
      if EASY_MODE:
         if max(len(word), len(trans)) > 20:
```

```
            continue
        word_to_translation[word].append(trans)
print("size = ", len(word_to_translation))
```

Вывод:

```
size = 130114
```

```
# получить все уникальные строки на исходном языке
all_words = np.array(list(word_to_translation.keys()))
# получить все уникальные строки на языке перевода
all_translations = np.array(
    [ts for all_ts in word_to_translation.values() for ts in all_ts])
```

Разделить набор данных.

Мы удерживаем 10% всех слов, которые будут использоваться для проверки.

```
from sklearn.model_selection import train_test_split
train_words, test_words = train_test_split(
    all_words, test_size=0.1, random_state=42)
```

Построение словарей

Теперь нам нужно создать словари, которые сопоставляют строки с идентификаторами токенов и наоборот. Нам понадобятся эти записи, когда мы будем вводить обучающие данные в модель или преобразовывать выходные матрицы в английские слова.

```
from voc import Vocab
inp_voc = Vocab.from_lines(''.join(all_words), bos=bos, eos=eos, sep='')
out_voc = Vocab.from_lines(''.join(all_translations), bos=bos, eos=eos, sep='')
```

```
# перевод строк в идентификаторы и обратно.
batch_lines = all_words[:5]
batch_ids = inp_voc.to_matrix(batch_lines)
batch_lines_restored = inp_voc.to_lines(batch_ids)
print("lines")
print(batch_lines)
print("\nwords to ids (0 = bos, 1 = eos):")
print(batch_ids)
print("\nback to words")
print(batch_lines_restored)
```

Вывод:

lines

['משתמש:צלף/!' 'קריאה סימן' 'תבנית:!!' '$9.99' "תבנית:'"]

words to ids (0 = bos, 1 = eos):

[[0 127 138 139 127 138 27 135 125 132 16   3   1]

 [0 130 122 127 128   2 136 137 122 113 117   1   1]

 [0 139 114 129 122 139 27   3   3   1   1   1   1]

 [0   6 26 15 26 26   1   1   1   1   1   1   1]

 [0 139 114 129 122 139 27   8   1   1   1   1   1]]

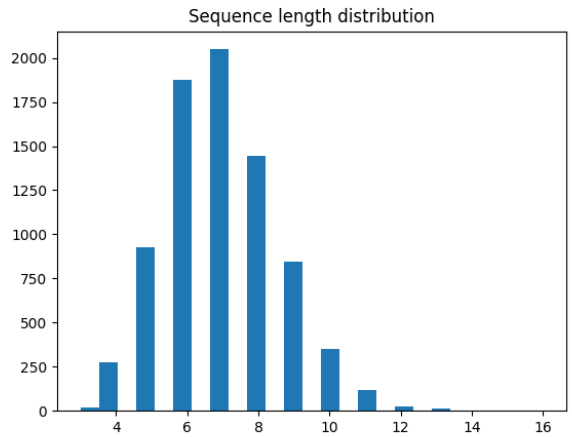
back to words

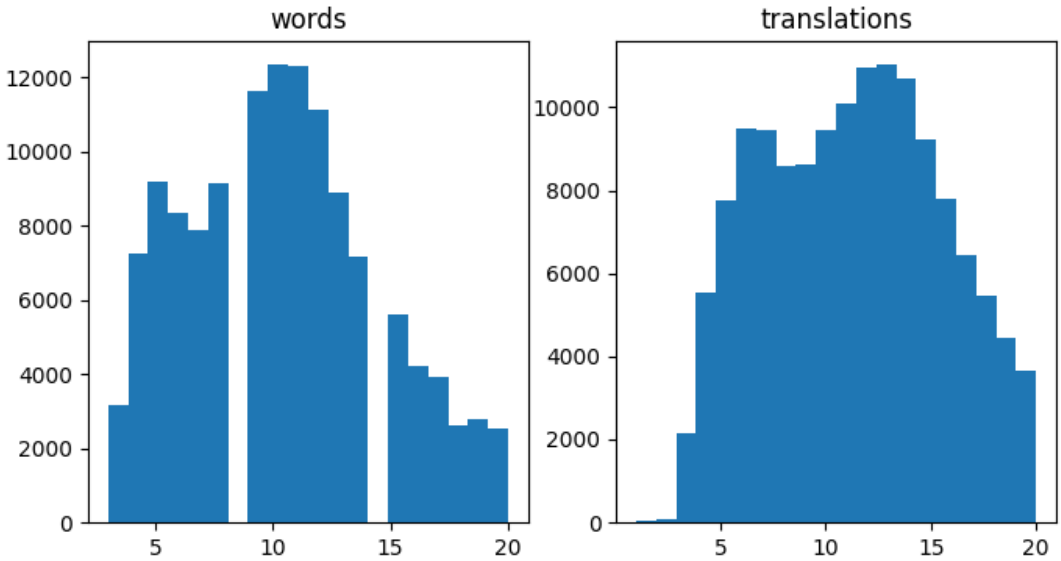
['משתמש:צלף/!', 'קריאה סימן', 'תבנית:!!', '$9.99', "תבנית:'"]

Нарисуйте распределение длины слова/перевода, чтобы оценить объем задачи.

```
import matplotlib.pyplot as plt
plt.figure(figsize=[8, 4])
plt.subplot(1, 2, 1)
plt.title("words")
plt.hist(list(map(len, all_words)), bins=20)
plt.subplot(1, 2, 2)
plt.title('translations')
plt.hist(list(map(len, all_translations)), bins=20)
plt.show()
```

Вывод:





3.

Развертывание кодировщика-декодера

Наша архитектура состоит из двух основных блоков:

- Кодер считывает слова посимвольно и выводит кодовый вектор (обычно это функция последнего состояния RNN).
- Декодер берет этот кодовый вектор и производит перевод символ за символом.

Затем он передается в модель, которая следует этому простому интерфейсу:

- model.symbolic_translate(inp, **flags) -> out, logp — принимает символическую матрицу слов на иврите int32, создает выходные токены, выбранные из модели, и выводит логарифмические вероятности для всех возможных токенов на каждом тике.
- model.symbolic_score(inp, out, **flags) -> logp - принимает символьные матрицы int32 слов на иврите и их английские переводы. Вычисляет логарифмические вероятности всех возможных английских символов с учетом английских префиксов и еврейского слова.
- model.weights - веса всех слоев модели [список переменных]

С помощью этих двух методов вы можете реализовать все виды прогнозирования и обучения.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
tf.compat.v1.reset_default_graph()
s = tf.compat.v1.InteractiveSession()
from basic_model_tf import BasicTranslationModel
```

```
model = BasicTranslationModel('model', inp_voc, out_voc,
                              emb_size=64, hid_size=128)

s.run(tf.compat.v1.global_variables_initializer())

# Поэкспериментируйте с symbolic_translate и symbolic_score
inp = tf.placeholder_with_default(np.random.randint(
   0, 10, [3, 5], dtype='int32'), [None, None])
out = tf.placeholder_with_default(np.random.randint(
   0, 10, [3, 5], dtype='int32'), [None, None])
# перевести ввод (с необученной моделью)
sampled_out, logp = model.symbolic_translate(inp, greedy=False)
print("\nSymbolic_translate output:\n", sampled_out, logp)
print("\nSample translations:\n", s.run(sampled_out))
```

Вывод:

```
Symbolic_translate output:

 Tensor("transpose_1:0", shape=(?, ?), dtype=int32) Tensor("LogSoftmax:0", shape=(?, ?, 283),
dtype=float32)

Sample translations:

 [[  0 109  66  82  78 119 169  92  16  32 216]

 [  0 118  94 194 124 281  99 110  10   7  66]

 [  0  92 247 154 197 138 133  96  28  56  34]]
```

```
# оценка logp(out | inp) с необученным вводом
logp = model.symbolic_score(inp, out)
print("\nSymbolic_score output:\n", logp)
print("\nLog-probabilities (clipped):\n", s.run(logp)[:, :2, :5])
```

Вывод:

```
Symbolic_score output:

 Tensor("LogSoftmax_1:0", shape=(?, ?, 283), dtype=float32)

Log-probabilities (clipped):

 [[[  0.       -69.07755   -69.07755   -69.07755   -69.07755  ]

  [ -5.6537566  -5.651446   -5.6586833  -5.6411147  -5.649842 ]]

 [[  0.       -69.07755   -69.07755   -69.07755   -69.07755  ]
```

```
 [ -5.660492   -5.654201   -5.6510887  -5.631667   -5.6455073]]

[[  0.       -69.07755   -69.07755   -69.07755   -69.07755  ]

 [ -5.661183   -5.647964   -5.653283   -5.626666   -5.647372 ]]]
```

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

```
# Секция для самостоятельных заданий
input_sequence = tf.placeholder('int32', [None, None])
greedy_translations, logp = <ВАШ КОД: создавайте символические переводы с помощью greedy = True>
def translate(lines):
    """
    Вам дан список входных строк.
     Заставьте свою нейронную сеть перевести их.
     :return: список строк вывода
    """
    # Преобразуйте линии в матрицу индексов
    lines_ix = <ВАШ КОД>
    # Вычислить переводы в виде индексов
    trans_ix = s.run(greedy_translations, { <ВАШ КОД: feed_dict> })
    # Преобразование переводов обратно в строки
    return out_voc.to_lines(trans_ix)

print("Sample inputs:", all_words[:3])
print("Dummy translations:", translate(all_words[:3]))
assert isinstance(greedy_translations,
                  tf.Tensor) and greedy_translations.dtype.is_integer, "trans должен быть тензором целых чисел (идентификаторов токенов)"
assert translate(all_words[:3]) == translate(
    all_words[:3]), "убедитесь, что перевод является детерминированным (используйте greedy=True и отключите все шумовые слои)"
assert type(translate(all_words[:3])) is list and (type(translate(all_words[:1])[0]) is str or type(
    translate(all_words[:1])[0]) is unicode), "translate(lines) должен возвращать последовательность строк!"
print("Tests passed!")
```

Вывод:

```
Sample inputs: ['טיפולתיים' 'קלרמון-פראן' '1948:קטגוריה']
Dummy translations: ['ďààɳñ""ῶààあɳɱɳɳɔɔûûь熊ɽấとと''''', 'ď守守ﻝﭘﭘ!ęɳąŭɽė-ﻋﭖèèạﺻﻦɳŋŋ??úɪ', 'ò8ēēźぼぼぼeeeeeeeeeeeeeeeeeee']
Tests passed!
```

Скоринговая функция

LogLikelihood — плохая оценка производительности модели.

- Если мы предсказываем нулевую вероятность один раз, это не должно разрушить всю модель.
- Достаточно выучить только один перевод, если правильных несколько.
- Важно то, сколько ошибок сделает модель при переводе!

Поэтому будем использовать минимальное расстояние Левенштейна. Оно измеряет, сколько символов нам нужно добавить/удалить/заменить из перевода модели, чтобы сделать его идеальным. В качестве альтернативы можно использовать BLEU/RougeL на уровне персонажа или другие подобные показатели.

Загвоздка здесь в том, что расстояние Левенштейна не дифференцируемо: оно даже не является непрерывным. Мы не можем обучить нашу нейронную сеть максимизировать ее с помощью градиентного спуска.

```
import editdistance
def get_distance(word, trans):
    """
    Функция, которая принимает слово и прогнозирует перевод
     и оценивает расстояние редактирования (Левенштейна) до ближайшего правильного перевода
    """
    references = word_to_translation[word]
    assert len(references) != 0, "wrong/unknown word"
    return min(editdistance.eval(trans, ref) for ref in references)
def score(words, bsize=100):
    """функция, вычисляющая расстояние Левенштейна для случайных выборок bsize"""
    assert isinstance(words, np.ndarray)
    batch_words = np.random.choice(words, size=bsize, replace=False)
    batch_trans = translate(batch_words)
    distances = list(map(get_distance, batch_words, batch_trans))
    return np.array(distances, dtype='float32')

# должно быть около 5-50 и быстро уменьшаться после тренировки
[score(test_words, 10).mean() for _ in range(5)]
```

Предварительное обучение под наблюдением

Здесь мы определяем функцию, которая обучает нашу модель за счет максимизации логарифмической вероятности, также известной как минимизация кроссэнтропии. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# импорт служебных функций
from basic_model_tf import initialize_uninitialized, infer_length, infer_mask,
select_values_over_last_axis
class supervised_training:
    # переменная для ввода и правильных ответов
    input_sequence = tf.placeholder('int32', [None, None])
    reference_answers = tf.placeholder('int32', [None, None])
    # Вычислить логарифмические вероятности всех возможных токенов на каждом шаге.
Использовать интерфейс модели.
    logprobs_seq = <ВАШ КОД>
    # вычисление средней кроссэнтропии
    crossentropy = - select_values_over_last_axis(logprobs_seq, reference_answers)
    mask = infer_mask(reference_answers, out_voc.eos_ix)
    loss = tf.reduce_sum(crossentropy * mask)/tf.reduce_sum(mask)
    # Оптимизатор веса сборки. Используйте model.weights, чтобы получить все обучаемые
параметры.
    train_step = <ВАШ КОД>
# инициализировать параметры оптимизатора, сохраняя при этом модель нетронутой
initialize_uninitialized(s)
```

Актуальное обучение на минипакетах.

```
import random
def sample_batch(words, word_to_translation, batch_size):
    """
    выборка случайной партии слов и случайный правильный перевод для каждого слова
     пример использования:
     batch_x,batch_y = sample_batch(train_words, word_to_translations,10)
    """
    # выбор слов
    batch_words = np.random.choice(words, size=batch_size)
    # выбор переводов
    batch_trans_candidates = list(map(word_to_translation.get, batch_words))
    batch_trans = list(map(random.choice, batch_trans_candidates))
    return inp_voc.to_matrix(batch_words), out_voc.to_matrix(batch_trans)
```

```
bx, by = sample_batch(train_words, word_to_translation, batch_size=3)
print("Source:")
print(bx)
print("Target:")
print(by)
```

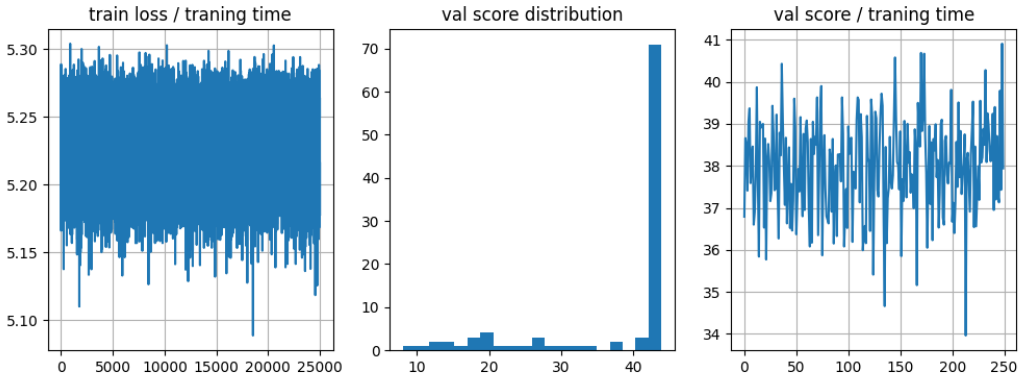
Вывод:

```
Source:

[[  0 137 113 114 122 134   8   1   1   1   1   1   1   1   1]

 [  0 116 127 122 121 137 122   2 138 127 122 113 136 117   1]

 [  0 121 137 121 136 118 114 137   1   1   1   1   1   1   1]]

Target:

[[ 0 50 33 55 41 35 58  1  1  1  1  1  1  1  1  1  1]

 [ 0 36 45 41 52 50 57  2 51 40 37 45 57 33 43 33  1]

 [ 0 52 33 50 52 33 43 47 54 37 50  1  1  1  1  1  1]]
```

```
from tqdm import tqdm, trange  # либо tqdm_notebook,tnrange
loss_history = []
editdist_history = []
for i in trange(25000):
    bx, by = sample_batch(train_words, word_to_translation, 32)
    feed_dict = {
        supervised_training.input_sequence: bx,
        supervised_training.reference_answers: by
    }
    loss, _ = s.run([supervised_training.loss,
                     supervised_training.train_step], feed_dict)
    loss_history.append(loss)
    if (i+1) % REPORT_FREQ == 0:
        current_scores = score(test_words)
        editdist_history.append(current_scores.mean())
        plt.figure(figsize=(12, 4))
        plt.subplot(131)
        plt.title('train loss / traning time')
        plt.plot(loss_history)
        plt.grid()
        plt.subplot(132)
        plt.title('val score distribution')
```

```
    plt.hist(current_scores, bins=20)
    plt.subplot(133)
    plt.title('val score / traning time')
    plt.plot(editdist_history)
    plt.grid()
    plt.show()
    print("llh=%.3f, mean score=%.3f" %
        (np.mean(loss_history[-10:]), np.mean(editdist_history[-10:])))
# Примечание: это нормально, если потери колеблются вверх и вниз, если в среднем есть улучшение в долгосрочной перспективе (например, партии по 5 тыс.)
```

Вывод:



100%|XXXXXXXX| 25000/25000 [11:23<00:00, 36.56it/s]

llh=5.223, mean score=38.278

```
for word in train_words[:10]:
   print("%s -> %s" % (word, translate([word])[0]))
```

Вывод:

```
מגילת המקדש -> þþạạےęņππŋŋŋŋpņpķ?êêêảảrảr
תבנית:user ady ->  ,ᒣèèללxņďτīpķ''' 老老ɳéééééé$$$$œ°
קטגוריה:שליטי איראן -> jźḩרצצמˡfɲɯ7íssננלôọṣ星ộṣしˋɱ̣ứåï'' 老老老ɳéééééé$
קרב אום-כתף -> þþþųےęņąąpᒣėŋŋŋŋعęņ?שמצצנג
קנאבידיול -> @ṣĸĸĸĸứû'' çéééŭöŭëמŭמ
1 סקיילרק -> ďħīâ"ńららʼʼチチב&&&ʻららķ榎榎
קטגוריה:המטבח הדני -> þþےąąpᒣėŋŋŋŋŋ仙仙仙*••ϕ…?………ƀ…ƀ……t……ōōōaaů.
ד, <- מורוזיני סקו'צנרפèèʋëḻṭŋ仙仙ïḥ仙ïïa::ævɔɔ$老老老°榎榎榎榎ɥĕěووو
sysrq -> j$$œ$œ$œ御守守פפפ
רביע אבו-ח'ליל -> jű̃ɔɔåɔộṣåộṣ→ï→ïvvū`````ąąąąpė-_
```

```
test_scores = []
for start_i in trange(0, len(test_words), 32):
    batch_words = test_words[start_i:start_i+32]
    batch_trans = translate(batch_words)
    distances = list(map(get_distance, batch_words, batch_trans))
    test_scores.extend(distances)

print("Supervised test score:", np.mean(test_scores))
```

Вывод:

```
Supervised test score: 41.44697202582232
```

4. Подготовка к обучению с подкреплением

Сначала нам нужно определить функцию потерь как пользовательскую операцию tf. Самый простой способ сделать это — использовать оболочку tensorflow.py_func.

```
def my_func(x):
  # x будет массивом numpy
  return np.sinh(x)
inp = tf.placeholder(tf.float32)
y = tf.py_func(my_func, [inp], tf.float32)
```

Ваша задача состоит в том, чтобы реализовать функцию _compute_levenshtein, которая принимает матрицы слов и переводов вместе с входными масками, затем преобразует их в фактические слова и фонемы и вычисляет min-levenshtein с помощью функции get_distance. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
def _compute_levenshtein(words_ix, trans_ix):
    """
    Пользовательская операция тензорного потока, которая вычисляет потери Левенштейна для
прогнозируемого транса.
     Параметры:
     -words_ix - матрица буквенных индексов, shape=[batch_size,word_length]
     -words_mask - матрица нулей/единиц,
       1 означает "слово еще не закончено"
       0 означает, что "слово уже закончилось, и это заполнение"
     - trans_mask - матрица выходных буквенных индексов, shape=[batch_size,translation_length]
     - trans_mask - матрица нулей/единиц, похожая на words_mask, но для trans_ix
     Пожалуйста, реализуйте функцию и убедитесь, что она проходит тесты из следующей ячейки.
    """
    # конвертировать слова в строки
    words = <ВАШ КОД: восстановить слова (список строк) из words_ix. Использовать словарь>
    assert type(words) is list and type(
        words[0]) is str and len(words) == len(words_ix)
    # конвертировать переводы в список
    translations = <ВАШ КОД: восстановить trans (список списков фонем) из trans_ix>
    assert type(translations) is list and type(
        translations[0]) is str and len(translations) == len(trans_ix)
    # вычислить расстояния Левенштейна. может быть произвольным кодом Python.
    distances = <ВАШ КОД: примените get_distance к каждой паре [words, translations]>
    assert type(distances) in (list, tuple, np.ndarray) and len(
        distances) == len(words_ix)
    distances = np.array(list(distances), dtype='float32')
    return distances
def compute_levenshtein(words_ix, trans_ix):
    out = tf.py_func(_compute_levenshtein, [words_ix, trans_ix, ], tf.float32)
    out.set_shape([None])
    return tf.stop_gradient(out)
```

Простой набор тестов, чтобы убедиться, что ваша реализация верна. Подсказка: если вы столкнетесь с какими-либо ошибками, не стесняйтесь использовать печать изнутри _compute_levenshtein.

```
# тесты
# выборка случайной партии (слова, правильный транс, неправильный транс)
batch_words = np.random.choice(train_words, size=100)
batch_trans = list(map(random.choice, map(
    word_to_translation.get, batch_words)))
```

```
batch_trans_wrong = np.random.choice(all_translations, size=100)
batch_words_ix = tf.constant(inp_voc.to_matrix(batch_words))
batch_trans_ix = tf.constant(out_voc.to_matrix(batch_trans))
batch_trans_wrong_ix = tf.constant(out_voc.to_matrix(batch_trans_wrong))
# calculate_levenshtein равен нулю для идеальных переводов
correct_answers_score = compute_levenshtein(
   batch_words_ix, batch_trans_ix).eval()
assert np.all(correct_answers_score ==
         0), "безупречный перевод получил ненулевой балл Левенштейна!"
print("Everything seems alright!")
```

Вывод:

```
Everything seems alright!
```

```
# compute_levenshtein соответствует фактической функции подсчета очков
wrong_answers_score = compute_levenshtein(
   batch_words_ix, batch_trans_wrong_ix).eval()
true_wrong_answers_score = np.array(
   list(map(get_distance, batch_words, batch_trans_wrong)))
assert np.all(wrong_answers_score ==
         true_wrong_answers_score), "для некоторого слова символическое расстояние Левенштейна
отличается от фактического расстояния Левенштейна"
print("Everything seems alright!")
```

Вывод:

```
Everything seems alright!
```

5. Градиент самокритичной политики

В этом разделе вы реализуете алгоритм, называемый обучением самокритичной последовательности. Алгоритм представляет собой ванильный градиент политики со специальной базовой линией.

$$\nabla J = E_{x\sim p(s)} E_{y\sim \pi(y|x)} \nabla log \pi(y|x) \cdot (R(x,y) - b(x))$$

Здесь вознаграждение R(x,y) является отрицательным расстоянием Левенштейна (поскольку мы его минимизируем). Базовый показатель b(x) показывает, насколько хорошо модель справляется со словом x. На практике это означает, что мы

вычисляем базовый уровень как показатель жадного перевода,

$$b(x) = R(x, y_{greedy}(x)).$$

Отметим, что мы уже получили необходимые выходные данные: model.greedy_translations, model.greedy_mask, и нам нужно только вычислить левенштейн с помощью функции calculate_levenshtein.

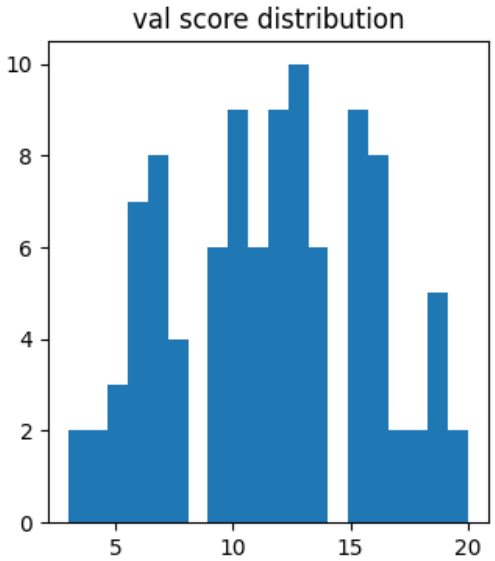
```
class trainer:
   input_sequence = tf.placeholder('int32', [None, None])
   # использовать модель для __sample__ символических переводов с учетом input_sequence
   sample_translations, sample_logp = <ВАШ КОД>
   # использовать модель для __greedy__ символических переводов, заданных input_sequence
   greedy_translations, greedy_logp = <ВАШ КОД>
   rewards = - compute_levenshtein(input_sequence, sample_translations)
   # вычислить __negative__ levenshtein для жадного режима
   baseline = <ВАШ КОД>
   # вычислить преимущество, используя вознаграждение и базовый уровень
   advantage = <ВАШ КОД: вычислить аознаграждение>
   assert advantage.shape.ndims == 1, "преимущество должно быть в форме [batch_size]"
   # вычислить log_pi(a_t|s_t), shape = [batch, seq_length]
   logprobs_phoneme = <ВАШ КОД>
   # ^-- подсказка: посмотрите, как реализована кроссэнтропия в контролируемой потере обучения выше
   # обратите внимание на знак - его нельзя умножать на -1 :)
   # Градиент политики вычислений
   # или, скорее, суррогатная функция, чей градиент является градиентом политики
   J = logprobs_phoneme*advantage[:, None]
   mask = infer_mask(sample_translations, out_voc.eos_ix)
   loss = - tf.reduce_sum(J*mask) / tf.reduce_sum(mask)
   # регуляризация с отрицательной энтропией. Не забудьте знак!
   # примечание: для энтропии вам нужны вероятности для всех токенов (sample_logp), а не только для phoneme_logprobs
   entropy = <ВЫШ КОД: вычислить энтропийную матрицу формы[пакет, длина последовательности], H = -sum(p*log_p), не забудьте знак!>
   # подсказка: вы можете получить выборочные вероятности из sample_logp, используя математику :)
   assert entropy.shape.ndims == 2, "пожалуйста, убедитесь, что поэлементная энтропия имеет форму [batch,time]"
   loss -= 0.01*tf.reduce_sum(entropy*mask) / tf.reduce_sum(mask)
   #вычислить обновления веса, обрезать по норме
```

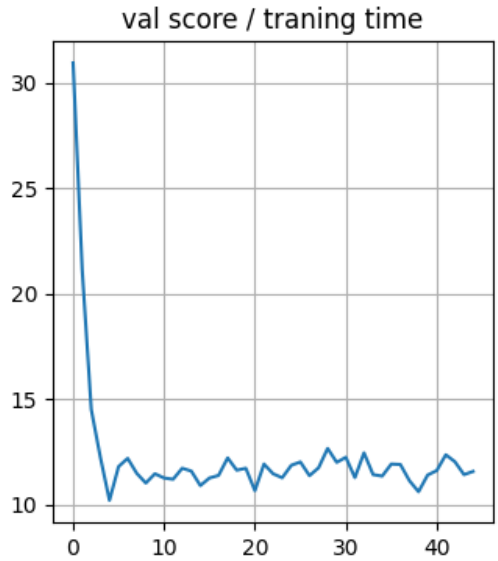
```
    grads = tf.gradients(loss, model.weights)
    grads = tf.clip_by_global_norm(grads, 50)[0]
    train_step = tf.train.AdamOptimizer(
        learning_rate=1e-5).apply_gradients(zip(grads, model.weights,))
initialize_uninitialized()
```

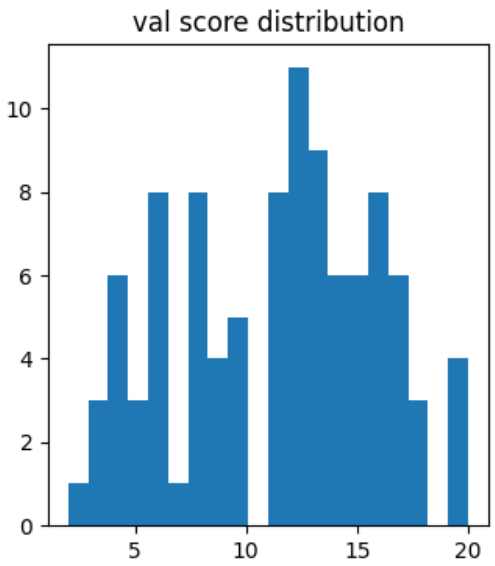
Обучение градиенту политики

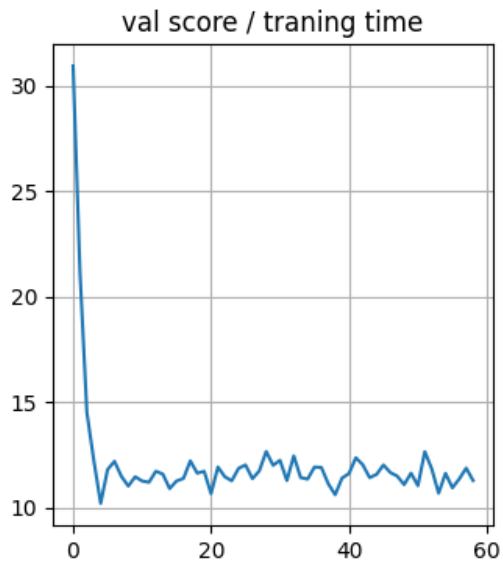
```
loss_history = []
editdist_history = []
for i in trange(10000):
    bx = sample_batch(train_words, word_to_translation, 32)[0]
    pseudo_loss, _ = s.run([trainer.loss, trainer.train_step], {
                        trainer.input_sequence: bx})
    loss_history.append(
        pseudo_loss
    )
    if (i+1) % REPORT_FREQ == 0:
        current_scores = score(test_words)
        editdist_history.append(current_scores.mean())
        plt.figure(figsize=(8, 4))
        plt.subplot(121)
        plt.title('val score distribution')
        plt.hist(current_scores, bins=20)
        plt.subplot(122)
        plt.title('val score / traning time')
        plt.plot(editdist_history)
        plt.grid()
        plt.show()
        print("J=%.3f, mean score=%.3f" %
            (np.mean(loss_history[-10:]), np.mean(editdist_history[-10:])))
```

Вывод:









100%|XXXXXXX| 10000/10000 [17:46<00:00, 9.38it/s]

J=-0.103, mean score=10.657

```
for word in train_words[:10]:
    print("%s -> %s" % (word, translate([word])[0]))
test_scores = []
for start_i in trange(0, len(test_words), 32):
    batch_words = test_words[start_i:start_i+32]
```

```
    batch_trans = translate(batch_words)
    distances = list(map(get_distance, batch_words, batch_trans))
    test_scores.extend(distances)
print("Supervised test score:", np.mean(test_scores))
```

Вывод:

```
מגילת המקדש -> aie aie
תבנית:user ady -> aie aie
קטגוריה:שליטי איראן -> aie aiea
קרב אום-כתף -> aie aie
קנאבידיול -> aie aie
1 סקיילרק -> aie aie
קטגוריה:המטבח הדני -> aie aie
פרנצ'סקו מורוזיני -> aie a aie
sysrq -> aie
רביע אבו-ח'ליל -> aie a aie
```

Задания к лабораторной работе №1

1. Написать свой код к разделам 1 - 5 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 1-5.

Требование к отчету

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

Лабораторная работа №9. Алгоритм Trust Region Policy Optimization (TRPO)

В этой работе мы напишем код оптимизации политики одного доверенного региона. Как обычно, он содержит несколько разных частей, которые мы собираемся воспроизвести.

1. Предварительные действия

```
import gym
from time import sleep
from IPython.display import clear_output
import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
import gym
env = gym.make("Acrobot-v1")
env.reset()
observation_shape = env.observation_space.shape
n_actions = env.action_space.n
print("Observation Space", env.observation_space)
print("Action Space", env.action_space)
```

Вывод:

```
Observation Space Box([ -1.  -1.  -1.  -1. -12.566371 -28.274334], [ 1.  1.  1.  1.
12.566371 28.274334], (6,), float32)

Action Space Discrete(3)
```

```
import matplotlib.pyplot as plt
plt.imshow(env.render('rgb_array'))
```

2. Определение сети

При всей своей сложности TRPO по своей сути является еще одним методом градиента политики. По сути, это означает, что мы на самом деле тренируем стохастическую политику. Будет использована нейронная сеть.

```
# входной тензор
observations_ph = tf.placeholder(
   shape=(None, observation_shape[0]), dtype=tf.float32)
# Совершенные действия
actions_ph = tf.placeholder(shape=(None,), dtype=tf.int32)
# "G = r + gamma*r' + gamma^2*r'' + ..."
cummulative_returns_ph = tf.placeholder(shape=(None,), dtype=tf.float32)
```

```
# Вероятность действий из предыдущей итерации
old_probs_ph = tf.placeholder(shape=(None, n_actions), dtype=tf.float32)

all_inputs = [observations_ph, actions_ph,
              cummulative_returns_ph, old_probs_ph]

def denselayer(name, x, out_dim, nonlinearity=None):
    with tf.variable_scope(name):
        if nonlinearity is None:
            nonlinearity = tf.identity
        x_shape = x.get_shape().as_list()
        w = tf.get_variable('w', shape=[x_shape[1], out_dim])
        b = tf.get_variable(
            'b', shape=[out_dim], initializer=tf.constant_initializer(0))
        o = nonlinearity(tf.matmul(x, w) + b)
        return o
```

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

```
sess = tf.InteractiveSession()
nn = observations_ph
<ВАШ КОД: ИНС>
policy_out = <ВАШ КОД: слой, который прогнозирует дейтсвия log-probabilities>
probs_out = tf.exp(policy_out)
weights = tf.trainable_variables()
sess.run(tf.global_variables_initializer())
```

3. Действия и развертывания

В этом разделе мы определим функции $a \sim \pi_\theta(a|s)$, которые выполняют действия и развертывания $< s_0, a_0, s_1, a_1, s_2, a_2, \ldots s_n, a_n >$.

```
# функция
def act(obs, sample=True):
    """
    Выборка действия из распространения политики (образец = True) или выполнение наиболее вероятного действия (образец = False)
    :param: obs - единичный вектор наблюдения
    :param sample: если True, выборки из \pi, в противном случае выполняется наиболее вероятное действие
    :returns: действие (одно целое число) и вероятности для всех действий"""
    probs = sess.run(probs_out, feed_dict={
```

```
            observations_ph: obs.reshape((1, -1))})[0]
    if sample:
        action = int(np.random.choice(n_actions))
    else:
        action = int(np.argmax(probs))
    return action, probs

# демо
print("sampled:", [act(env.reset()) for _ in range(5)])
print("greedy:", [act(env.reset(), sample=False) for _ in range(5)])
```

Вывод:

```
sampled: [(1, array([0.00321023,       nan,       nan], dtype=float32)), (0, array([       nan, 0.03026781,
nan], dtype=float32)), (2, array([0.0192967 , 0.01878625,       nan], dtype=float32)), (1, array([0.0144313,
nan,       nan], dtype=float32)), (2, array([nan, nan, nan], dtype=float32))]

greedy: [(1, array([0.00756173,       nan,       nan], dtype=float32)), (2, array([0.00590326, 0.03594141,
nan], dtype=float32)), (1, array([0.0043306,       nan,       nan], dtype=float32)), (1, array([0.00883704,
nan,       nan], dtype=float32)), (0, array([nan, nan, nan], dtype=float32))]
```

Вычислите кумулятивное вознаграждение так же, как вы это делали в ванильном REINFORCE.

```
def get_cummulative_returns(r, gamma=1):
    """
    Вычисляет кумулятивное вознаграждение со скидкой при немедленном вознаграждении
    G_i = r_i + gamma*r_{i+1} + gamma^2*r_{i+2} + ...
    Известный также как R(s,a).
    """
    r = np.array(r)
    assert r.ndim >= 1
    return scipy.signal.lfilter([1], [1, -gamma], r[::-1], axis=0)[::-1]

# простая демонстрация для вознаграждения [0,0,1,0,0,1]
get_cummulative_returns([0, 0, 1, 0, 0, 1], gamma=0.9)
```

Развертывание

```
def rollout(env, act, max_pathlength=2500, n_timesteps=50000):
    """
    Создание выпусков для обучения.
     :param:env - окружение, в котором мы будем производить действия по генерации выкатов.
     :param:act - функция, которая может возвращать политику и действие с учетом наблюдения.
     :param: max_pathlength - максимальный размер одного пути, который мы генерируем.
```

```
    :param: n_timesteps - общая сумма размеров всех путей, которые мы генерируем.
    """
    paths = []
    total_timesteps = 0
    while total_timesteps < n_timesteps:
        obervations, actions, rewards, action_probs = [], [], [], []
        obervation = env.reset()
        for _ in range(max_pathlength):
            action, policy = act(obervation)
            obervations.append(obervation)
            actions.append(action)
            action_probs.append(policy)
            obervation, reward, done, _ = env.step(action)
            rewards.append(reward)
            total_timesteps += 1
            if done or total_timesteps == n_timesteps:
                path = {"observations": np.array(obervations),
                        "policy": np.array(action_probs),
                        "actions": np.array(actions),
                        "rewards": np.array(rewards),
                        "cumulative_returns": get_cummulative_returns(rewards),
                        }
                paths.append(path)
                break
    return paths

paths = rollout(env, act, max_pathlength=5, n_timesteps=100)
print(paths[-1])
assert (paths[0]['policy'].shape == (5, n_actions))
assert (paths[0]['cumulative_returns'].shape == (5,))
assert (paths[0]['rewards'].shape == (5,))
assert (paths[0]['observations'].shape == (5,)+observation_shape)
assert (paths[0]['actions'].shape == (5,))
print('It\'s ok')
```

Вывод:

```
{'observations': array([[ 0.99940807,  0.03440231,  0.9998454 , -0.01758484, -0.03131559,
         0.08672629],
       [ 0.9999422 ,  0.01075217,  0.9992699 ,  0.03820498, -0.19845793,
         0.45750472],
```

[0.9996275 , -0.0272931 , 0.99253607, 0.12195132, -0.17179134,

0.3627333],

[0.9991583 , -0.04101944, 0.98993856, 0.14149801, 0.03745227,

-0.17025656],

[0.9992117 , -0.03969756, 0.9921739 , 0.12486374, -0.02492967,

0.00533957]], dtype=float32), 'policy': array([[0.56596464, nan, nan],

[0.4888617 , nan, nan],

[0.4916529 , nan, nan],

[0.580484 , nan, nan],

[0.55329365, nan, nan]], dtype=float32), 'actions': array([2, 1, 0, 2, 0]), 'rewards': array([-1., -1., -1., -1., -1.]), 'cumulative_returns': array([-5., -4., -3., -2., -1.])}

It's ok

4. Функция потерь

Теперь давайте определим функции потерь и ограничения для фактического обучения TRPO.

Суррогатное вознаграждение должно быть:

$$J_{surr} = \frac{1}{N} \sum_{i=0}^{N} \frac{\pi_\theta(s_i, a_i)}{\pi_{\theta_{old}}(s_i, a_i)} A_{\theta_{old}(s_i, a_i)}$$

Для простоты давайте пока будем использовать кумулятивную доходность вместо преимущества:

$$J'_{surr} = \frac{1}{N} \sum_{i=0}^{N} \frac{\pi_\theta(s_i, a_i)}{\pi_{\theta_{old}}(s_i, a_i)} G_{\theta_{old}(s_i, a_i)}$$

Или, альтернативно, минимизируйте суррогатную потерю:

$$L_{surr} = -J'_{surr}$$

```
# выбрать вероятности выбранных действий
batch_size = tf.shape(observations_ph)[0]
probs_all = tf.reshape(probs_out, [-1])
```

```
probs_for_actions = tf.gather(probs_all, tf.range(
   0, batch_size) * n_actions + actions_ph)
old_probs_all = tf.reshape(old_probs_ph, [-1])
old_probs_for_actions = tf.gather(
   old_probs_all, tf.range(0, batch_size) * n_actions + actions_ph)
```

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

```
# Вычислить суррогатные потери: отрицательный политический градиент с выборкой по важности
L_surr = <ВАШ КОД: вычисляйте суррогатные потери, также известные как _отрицательный_ градиент политик с выборкой по важности>

# вычислить и вернуть суррогатный градиент политики
def var_shape(x):
   return [k.value for k in x.get_shape()]
def numel(x):
   return np.prod(var_shape(x))
def flatgrad(loss, var_list):
   grads = tf.gradients(loss, var_list)
   return tf.concat([tf.reshape(grad, [numel(v)])
                     for (v, grad) in zip(var_list, grads)], 0)
flat_grad_surr = flatgrad(L_surr, weights)
```

Мы можем подниматься по этим градиентам до тех пор, пока $pi_\theta(a|s)$ удовлетворяет ограничению:

$$E_{s,\pi_{\Theta_t}}\Big[KL(\pi(\Theta_t, s) \;||\; \pi(\Theta_{t+1}, s))\Big] < \alpha$$

где

$$KL(p||q) = E_p log(\frac{p}{q})$$

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

```
# Вычислить расхождение Кульбака-Лейблера (см. формулу выше)
# Примечание: вам нужно суммировать KL и энтропию по всем действиям, а не только по тем, которые совершил агент
old_log_probs = tf.log(old_probs_ph+1e-10)
kl = <ВАШ КОД: рассчитайте расхождение Кульбака-Лейблера>
```

```
# Вычислите энтропию политики
entropy = <ВАШ КОД: вычислите энтропию политики. Не забудьте знак!>
losses = [L_surr, kl, entropy]
```

Линейный поиск

TRPO по своей сути включает восходящий градиент суррогатной политики, ограниченный дивергенцией KL. Чтобы обеспечить соблюдение этого ограничения, мы будем использовать линейный поиск.

```
def linesearch(f, x, fullstep, max_kl):
    """
    Linesearch находит лучшие параметры нейронных сетей в направлении полного шага,
ограниченного расхождением KL.
     :param:f - функция, которая возвращает потери, kl и произвольную третью компоненту.
     :param: x - старые параметры нейросети.
     :param: fullstep - направление, в котором ищем.
     :param: max_kl - ограничение расхождения KL.
     :возвращается:
    """
    max_backtracks = 10
    loss, _, _ = f(x)
    for stepfrac in .5**np.arange(max_backtracks):
        xnew = x + stepfrac * fullstep
        new_loss, kl, _ = f(xnew)
        actual_improve = new_loss - loss
        if kl <= max_kl and actual_improve < 0:
            x = xnew
            loss = new_loss
    return x
```

6. Обучение

В этом разделе мы строим остальные части нашего вычислительного графа.

```
def slice_vector(vector, shapes):
    """
    Разбивает символьный вектор на несколько символьных тензоров заданных форм.
     Вспомогательная функция, используемая для разглаживания градиентов, касательных и т. д.
     :param vector: 1-мерный символьный вектор
     :param shape: список или кортеж фигур (список, кортеж или символ)
     :returns: список символических тензоров заданных форм
    """
```

```
    assert len(vector.get_shape()) == 1, "vector must be 1-dimensional"
    start = 0
    tensors = []
    for shape in shapes:
        size = np.prod(shape)
        tensor = tf.reshape(vector[start:(start + size)], shape)
        tensors.append(tensor)
        start += size
    return tensors

# средний уровень в conjugate_gradient
conjugate_grad_intermediate_vector = tf.placeholder(
    dtype=tf.float32, shape=(None,))
# разрезать flat_tangent на куски для каждого веса
weight_shapes = [sess.run(var).shape for var in weights]
tangents = slice_vector(conjugate_grad_intermediate_vector, weight_shapes)
# Расхождение KL, где первый аргумент фиксирован
kl_firstfixed = tf.reduce_sum((tf.stop_gradient(probs_out) * (tf.stop_gradient(
    tf.log(probs_out)) - tf.log(probs_out)))) / tf.cast(batch_size, tf.float32)
# вычислить информационную матрицу Фишера (используется для сопряженных градиентов и для оценки KL)
gradients = tf.gradients(kl_firstfixed, weights)
gradient_vector_product = [tf.reduce_sum(
    g[0] * t) for (g, t) in zip(gradients, tangents)]
fisher_vec_prod = flatgrad(gradient_vector_product, weights)
```

7. Вспомогательные функции

Сопряженные градиенты

Поскольку TRPO включает оптимизацию с ограничениями, нам нужно будет решить Ax=b, используя сопряженные градиенты. В общем, CG — это алгоритм, который решает Ax=b, где A положительно определено. A — матрица Гессе, поэтому A положительно определена.

```
def conjugate_gradient(f_Ax, b, cg_iters=10, residual_tol=1e-10):
    """
    Этот метод решает систему уравнений Ax=b, используя итерационный метод, называемый сопряженными градиентами.
    :f_Ax: функция, которая возвращает Ax
    :b: цели для топора
    :cg_iters: сколько итераций должен делать этот метод
    :residual_tol: эпсилон для стабильности
```

```
    """
    p = b.copy()
    r = b.copy()
    x = np.zeros_like(b)
    rdotr = r.dot(r)
    for i in range(cg_iters):
        z = f_Ax(p)
        v = rdotr / (p.dot(z) + 1e-8)
        x += v * p
        r -= v * z
        newrdotr = r.dot(r)
        mu = newrdotr / (rdotr + 1e-8)
        p = r + mu * p
        rdotr = newrdotr
        if rdotr < residual_tol:
            break
    return x

# Этот код проверяет сопряженные градиенты
A = np.random.rand(8, 8)
A = np.matmul(np.transpose(A), A)
def f_Ax(x):
    return np.matmul(A, x.reshape(-1, 1)).reshape(-1)
b = np.random.rand(8)
w = np.matmul(np.matmul(inv(np.matmul(np.transpose(A), A)),
                  np.transpose(A)), b.reshape((-1, 1))).reshape(-1)
print(w)
print(conjugate_gradient(f_Ax, b))
```

Вывод:

```
[-149.86919555 -108.59416764  131.789362    359.94990928 -156.74041779
  -49.10863303 -209.36261454  124.61801108]
[-149.86938114 -108.59413385  131.78915904  359.94983677 -156.74051792
  -49.10868357 -209.36247309  124.61786152]
```

```
# Скомпилируйте функцию, которая экспортирует веса сети в виде вектора
flat_weights = tf.concat([tf.reshape(var, [-1]) for var in weights], axis=0)

# ... и еще одна функция, которая импортирует вектор обратно в веса сети
flat_weights_placeholder = tf.placeholder(tf.float32, shape=(None,))
```

```
assigns = slice_vector(flat_weights_placeholder, weight_shapes)

load_flat_weights = [w.assign(ph) for w, ph in zip(weights, assigns)]
```

8. Главный цикл TRPO

```
import time
from itertools import count
from collections import OrderedDict
# это гиперпараметр TRPO. Он контролирует, насколько большим может быть расхождение KL
между старой и новой политикой на каждом этапе.
max_kl = 0.01
cg_damping = 0.1  # Эти параметры упорядочивают дополнение к
numeptotal = 0  # тому количеству эпизодов, которые мы сыграли.
start_time = time.time()
for i in count(1):
   print("\n********** Iteration %i ************" % i)
   # Generating paths.
   print("Rollout")
   paths = rollout(env, act)
   print("Made rollout")
   # Обновление политики.
   observations = np.concatenate([path["observations"] for path in paths])
   actions = np.concatenate([path["actions"] for path in paths])
   returns = np.concatenate([path["cumulative_returns"] for path in paths])
   old_probs = np.concatenate([path["policy"] for path in paths])
   inputs_batch = [observations, actions, returns, old_probs]
   feed_dict = {observations_ph: observations,
            actions_ph: actions,
            old_probs_ph: old_probs,
            cummulative_returns_ph: returns,
            }
   old_weights = sess.run(flat_weights)
   def fisher_vector_product(p):
      """gets intermediate grads (p) and computes fisher*vector """
      feed_dict[conjugate_grad_intermediate_vector] = p
      return sess.run(fisher_vec_prod, feed_dict) + cg_damping * p
   flat_grad = sess.run(flat_grad_surr, feed_dict)
   stepdir = conjugate_gradient(fisher_vector_product, -flat_grad)
   shs = .5 * stepdir.dot(fisher_vector_product(stepdir))
   lm = np.sqrt(shs / max_kl)
   fullstep = stepdir / lm
```

```
    # Вычислить новые веса с помощью линейного поиска в направлении, которое мы нашли с помощью компьютерной графики.
    def losses_f(flat_weights):
        feed_dict[flat_weights_placeholder] = flat_weights
        sess.run(load_flat_weights, feed_dict)
        return sess.run(losses, feed_dict)
    new_weights = linesearch(losses_f, old_weights, fullstep, max_kl)
    feed_dict[flat_weights_placeholder] = new_weights
    sess.run(load_flat_weights, feed_dict)
    # Сообщить о текущем прогрессе
    L_surr, kl, entropy = sess.run(losses, feed_dict)
    episode_rewards = np.array([path["rewards"].sum() for path in paths])
    stats = OrderedDict()
    numeptotal += len(episode_rewards)
    stats["Total number of episodes"] = numeptotal
    stats["Average sum of rewards per episode"] = episode_rewards.mean()
    stats["Std of rewards per episode"] = episode_rewards.std()
    stats["Entropy"] = entropy
    stats["Time elapsed"] = "%.2f mins" % ((time.time() - start_time)/60.)
    stats["KL between old and new distribution"] = kl
    stats["Surrogate loss"] = L_surr
    for k, v in stats.items():
        print(k + ": " + " " * (40 - len(k)) + str(v))
    i += 1
```

Вывод:

```
********** Iteration 1 ************

Rollout

Made rollout

Total number of episodes:                100

Average sum of rewards per episode:      -500.0

Std of rewards per episode:              0.0

Entropy:                                 [nan]

Time elapsed:                            0.64 mins

KL between old and new distribution:     nan

Surrogate loss:                          [nan]
```

…

********** Iteration 4 ************

Rollout

Made rollout

Total number of episodes: 403

Average sum of rewards per episode: -495.009900990099

Std of rewards per episode: 24.995641223785736

Entropy: [nan]

Time elapsed: 2.55 mins

KL between old and new distribution: nan

Surrogate loss: [nan]

…

Задания к лабораторной работе №1

1. Написать свой код к разделам 1 - 8 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 1-8.

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

Лабораторная работа №10. Поиск по дереву Монте-Карло (MCTS)

Поиск по дереву Монте-Карло (MCTS) — это эвристический алгоритм поиска, который показывает отличные результаты в таких сложных областях, как го и шахматы. Алгоритм строит дерево поиска, итеративно обходит его и оценивает его узлы с помощью моделирования методом Монте-Карло.

1. Предварительные действия

```
import gym
import numpy as np
import matplotlib.pyplot as plt
# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
import os
if type(os.environ.get("DISPLAY")) is not str or len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'
```

Но прежде, чем мы это сделаем, нам сначала нужно создать оболочку для сред Gym, позволяющую сохранять и загружать игровые состояния для облегчения возврата. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
import gym
from gym.core import Wrapper
from pickle import dumps, loads
from collections import namedtuple
# контейнер для функции get_result ниже. Работает так же, как кортеж, но красивее
ActionResult = namedtuple(
    "action_result", ("snapshot", "observation", "reward", "is_done", "info"))
class WithSnapshots(Wrapper):
    """
    Создает оболочку, поддерживающую сохранение и загрузку состояний среды.
    Требуется для алгоритмов планирования.
    Этот класс будет иметь доступ к основной среде как self.env, например:
    - self.env.reset() # сбросить исходную среду
    - self.env.ale.cloneState() # сделать снимок для atari. загрузить с помощью .restoreState()
    - ...
    Вы также можете использовать reset() и step() напрямую для удобства.
    - s = self.reset() # то же, что и self.env.reset()
    - s, r, done, _ = self.step(action) # то же, что и self.env.step(action)
    Обратите внимание, что, хотя вы можете использовать self.render(), это создаст окно, которое
нельзя замариновать.
    Таким образом, вам нужно будет вызвать self.close(), прежде чем травление снова заработает.
```

```
    """
    def get_snapshot(self, render=False):
        """
        :returns: состояние среды, которое можно загрузить с помощью load_snapshot
         Снимки гарантируют одинаковое поведение env при каждой загрузке.
         Предупреждение! Снимки могут быть произвольными вещами (строки, целые числа, json,
кортежи)
         Не рассчитывайте на то, что они будут строками рассола при реализации MCTS.
         Примечание разработчика: убедитесь, что возвращаемый вами объект не будет затронут
         все, что происходит с окружающей средой после ее сохранения.
         Вы не должны, например, возвращать self.env.
         В случае сомнений используйте pickle.dumps или deepcopy.
        """
        if render:
            self.render()  # close popup windows since we can't pickle them
            self.close()

        if self.unwrapped.viewer is not None:
            self.unwrapped.viewer.close()
            self.unwrapped.viewer = None
        return dumps(self.env)
    def load_snapshot(self, snapshot, render=False):
        """
        Загружает снимок как текущее состояние env.
         Не следует менять снапшот на месте (в случае сомнений — глубокая копия).
        """
        assert not hasattr(self, "_monitor") or hasattr(
            self.env, "_monitor"), "не могу вернуться во время записи"
        if render:
            self.render()  # закрыть всплывающие окна, так как мы не можем загрузиться в них
            self.close()
        self.env = loads(snapshot)
    def get_result(self, snapshot, action):
        """
        Удобная функция, которая
         - загружает снимок,
         - совершает действие через self.step,
         - и снова делает снимок :)
         :returns: следующий снимок, next_observation, награда, is_done, информация
         По сути, он возвращает следующий снимок и все, что вернул бы env.step.
        """
```

```
    < ВАШ КОД: зашрузка, коммит, взять снимок >
    return ActionResult(
        < ВАШ КОД: next_snapshot >,  # заполните переменные
        < ВАШ КОД: next_observation >,
        < ВАШ КОД: reward >,
        < ВАШ КОД: is_done >,
        < ВАШ КОД: info >,
    )
```

2. Пробы со снимками

Давайте проверим нашу обертку. Сначала сбрасываем окружение и сохраняем его, далее случайным образом играем какие-то действия и восстанавливаем наше окружение из снимка. Оно должно быть таким же, как наше предыдущее начальное состояние.

```
# создание окружения
env = WithSnapshots(gym.make("CartPole-v0"))
env.reset()
n_actions = env.action_space.n

print("initial_state:")
plt.imshow(env.render('rgb_array'))
env.close()
# создание первого снимка
snap0 = env.get_snapshot()

# проигрывание без снимков (быстрее)
while True:
   is_done = env.step(env.action_space.sample())[2]
   if is_done:
      print("Whoops! We died!")
      break
print("final state:")
plt.imshow(env.render('rgb_array'))
env.close()
# перезагрузка первичного состояния
env.load_snapshot(snap0)
print("\n\nAfter loading snapshot")
plt.imshow(env.render('rgb_array'))
env.close()
# получить результат (snapshot, observation, reward, is_done, info)
res = env.get_result(snap0, env.action_space.sample())
```

```
snap1, observation, reward = res[:3]
# второй шаг
res2 = env.get_result(snap1, env.action_space.sample())
```

3. MCTS: поиск по дереву Монте-Карло

Мы начнем с реализации класса Node — простого класса, который действует как узел MCTS и поддерживает некоторые шаги алгоритма MCTS.

```
assert isinstance(env, WithSnapshots)
```

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

```
class Node:
    """Узел дерева для MCTS.
     Каждый узел соответствует результату выполнения определенного действия (self.action)
     в определенном состоянии (самородитель) и, по сути, является одной рукой в многоруком бандите, который
     мы моделируем в этом состоянии."""
    # метаданные:
    parent = None  # родительский узел
    qvalue_sum = 0.  # сумма Q-значений из всех визитов (numerator)
    times_visited = 0  # счетчик визитов (denominator)
    def __init__(self, parent, action):
        """
        Создает пустой узел без дочерних элементов.
         Делает это, совершая действие и записывая результат.
         :param parent: родительский узел
         :param action: действие для фиксации с родительского узла
        """
        self.parent = parent
        self.action = action
        self.children = set()  # набор дочерних узлов
        # получение результатов действий и их сохранение
        res = env.get_result(parent.snapshot, action)
        self.snapshot, self.observation, self.immediate_reward, self.is_done, _ = res
    def is_leaf(self):
        return len(self.children) == 0
    def is_root(self):
        return self.parent is None
    def get_qvalue_estimate(self):
        return self.qvalue_sum / self.times_visited if self.times_visited != 0 else 0
```

```
  def ucb_score(self, scale=10, max_value=1e100):
    """
    Вычисляет верхнюю границу ucb1, используя текущее значение и количество посещений для узла и его родителя.
     :param scale: Умножает на это верхнюю границу. Из неравенства Хёффдинга
                предполагает, что диапазон вознаграждения равен [0, шкала].
     :param max_value: значение, представляющее бесконечность (для непосещенных узлов).

    """
    if self.times_visited == 0:
      return max_value
    # вычислить аддитивный компонент ucb-1 (добавлять к среднему значению)
    # подсказка: вы можете использовать self.parent.times_visited для N раз, когда узел рассматривался,
    # и self.times_visited для n посещений
    U = < ВАШ КОД: >
    return self.get_qvalue_estimate() + scale * U
  # MCTS шаги
  def select_best_leaf(self):
    """
    Выбирает лист с наивысшим приоритетом для расширения.
     Делает это, рекурсивно выбирая узлы с лучшим результатом UCB-1, пока не достигнет листа.
    """
    if self.is_leaf():
      return self
    children = self.children
    # Выберите дочерний узел с наивысшей оценкой UCB. Возможно, вы захотите реализовать некоторые эвристики
    # чтобы по-умному разорвать связи, хотя CartPole должен прекрасно работать и без них.
    best_child = < ВАШ КОД: >

    return best_child.select_best_leaf()
  def expand(self):
    """
    Расширяет текущий узел, создавая все возможные дочерние узлы.
     Затем возвращается один из этих детей.
    """
    assert not self.is_done, "не может расширяться из терминального состояния"
    for action in range(n_actions):
      self.children.add(Node(self, action))
    # Если вы реализовали какие-либо эвристики в select_best_leaf(), они будут использоваться
```

```
здесь.
    # В противном случае это эквивалентно выбору некоторого неопределенного только что созданного дочернего узла.
    return self.select_best_leaf()
  def rollout(self, t_max=10 ** 4):
    """
    Играйте в игру от этого состояния до конца (готово) или за t_max шагов.
     На каждом этапе выбирайте действие случайным образом (подсказка: env.action_space.sample()).
     Вычислите сумму наград от текущего состояния до конца эпизода.
     Примечание 1: используйте env.action_space.sample() для выбора случайного действия.
     Примечание 2: если узел является терминальным (self.is_done имеет значение True), просто верните self.immediate_reward.
    """
    # установка среды в требуемое состояние
    env.load_snapshot(self.snapshot)
    obs = self.observation
    is_done = self.is_done
    < ВАШ КОД: обеспечить развертывание и рассчет вознаграждения >
    return rollout_reward
  def propagate(self, child_qvalue):
    """
    Использует дочернее Q-значение (сумму вознаграждений) для рекурсивного обновления родителей.
    """
    # вычисленные узла Q-значения
    my_qvalue = self.immediate_reward + child_qvalue
    # обновление qvalue_sum и times_visited
    self.qvalue_sum += my_qvalue
    self.times_visited += 1
    # распространяться вверх
    if not self.is_root():
      self.parent.propagate(my_qvalue)
  def safe_delete(self):
    """безопасное удаление для предотвращения утечки памяти в некоторых версиях Python"""
    del self.parent
    for child in self.children:
      child.safe_delete()
      del child
```

```
class Root(Node):
    def __init__(self, snapshot, observation):
        """
        создает специальный узел, который действует как корень дерева
         :snapshot: снимок (из env.get_snapshot), с которого можно начать планирование
         :observation: последнее наблюдение за окружающей средой
        """
        self.parent = self.action = None
        self.children = set()  # выбор дочернего узла
        # root: загрузка снимка и наблюдение
        self.snapshot = snapshot
        self.observation = observation
        self.immediate_reward = 0
        self.is_done = False
    @staticmethod
    def from_node(node):
        """инициализирует узел как root"""
        root = Root(node.snapshot, node.observation)
        # copy data
        copied_fields = ["qvalue_sum", "times_visited", "children", "is_done"]
        for field in copied_fields:
            setattr(root, field, getattr(node, field))
        return root
```

4. Основной цикл MCTS

Со всем, что мы реализовали, MCTS сводится к простому фрагменту кода. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
def plan_mcts(root, n_iters=10):
    """
    строит дерево с поиском по дереву Монте-Карло для n_iters итераций
     :param root: узел дерева для планирования
     :param n_iters: сколько циклов select-expand-simulate-propagete сделать
    """
    for _ in range(n_iters):
        node = < ВАШ КОД: выберите лучший лист >
        if node.is_done:
            # Все развертывания с терминального узла пусты и, следовательно, имеют 0
вознаграждений.
            node.propagate(0)
```

```
    else:
        # Разверните лучший лист. Выполните выкат из него. Распространите результаты вверх.
        # Обратите внимание, что здесь у вас есть некоторая свобода действий при выборе
источника распространения.
        # Любой разумный выбор должен работать.
        < ВАШ КОД >
```

5. Планирование и запуск

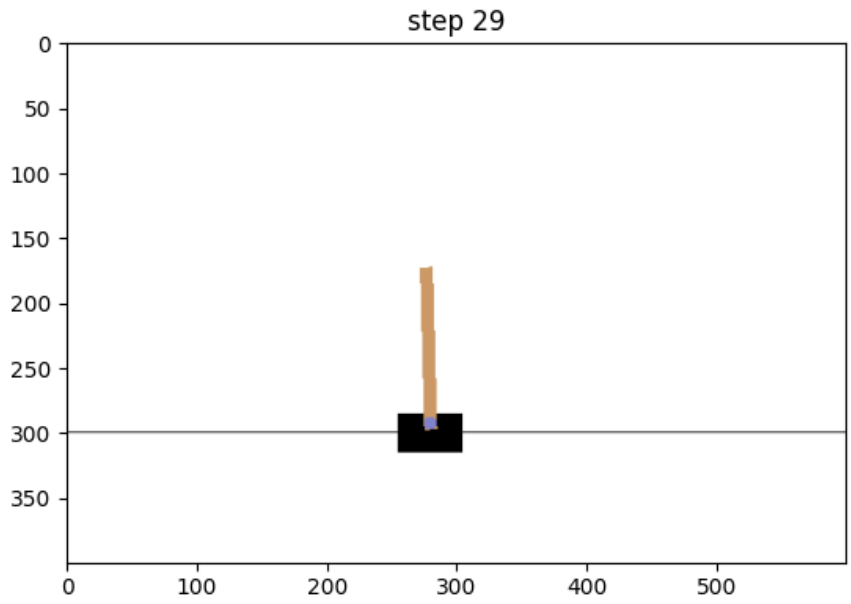
Давайте используем нашу реализацию MCTS, чтобы найти оптимальную политику. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
env = WithSnapshots(gym.make("CartPole-v0"))
root_observation = env.reset()
root_snapshot = env.get_snapshot()
root = Root(root_snapshot, root_observation)
```

```
env = WithSnapshots(gym.make("CartPole-v0"))
root_observation = env.reset()
root_snapshot = env.get_snapshot()
root = Root(root_snapshot, root_observation)
# планируем от корня:
plan_mcts(root, n_iters=1000)
from IPython.display import clear_output
from itertools import count
from gym.wrappers import Monitor
total_reward = 0  # sum of rewards
test_env = loads(root_snapshot)  # env used to show progress
for i in count():
  # получение лучшего дочернего узла
  best_child = <ВАШ КОД: выбор дочернего узла с максимальным средним вознаграждением>
  # выбор действия
  s, r, done, _ = test_env.step(best_child.action)
  # показ снимка
  clear_output(True)
  plt.title("step %i" % i)
  plt.imshow(test_env.render('rgb_array'))
  plt.show()
  total_reward += r
  if done:
    print("Finished with reward = ", total_reward)
```

```
        break
    # отбросить нереализованную часть дерева
    for child in root.children:
        if child != best_child:
            child.safe_delete()
    # объявить лучшего потомка новым корнем
    root = Root.from_node(best_child)
    assert not root.is_leaf(), \
        "У нас закончилось дерево! Нужно больше планирования! Попробуйте вырастить дерево прямо внутри цикла"
    # Возможно, вы захотите запустить больше планов здесь
    # <ВАШ КОД>
```

Вывод:



Задания к лабораторной работе

1. Написать свой код к разделам 1 - 8 согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>

<ВАШ КОД: >

2. Запустить полученную программу, получить ожидаемые выходные данные.

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

ЗАКЛЮЧЕНИЕ

Предлагаемое учебно-методическое пособие позволяет обучающимся приобрести практические навыки при изучении дисциплины «Обучение с подкреплением». Выполнение лабораторных работ предполагает использование языка Python и работу с библиотеками Gym, Tensorflow и PyTorch.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) Саттон Р. С. Обучение с подкреплением [Электронный ресурс] / Р. С. Саттон, Э. Г. Барто ; пер. с англ. — 2-е изд. (эл.). — Электрон. текстовые дан. (1 файл pdf : 402 с.). — М. : БИНОМ. Лаборатория знаний, 2014.

2) Foundations of Deep Reinforcement Learning: Theory and Practice in Python by Laura Graesser, Wah Loon Keng Released December 2019. Publisher(s): Addison-Wesley Professional ISBN: 9780135172490

3) Интернет-ресурс: https://karenyyy.github.io/year-archive/

4) Интернет-ресурс: https://github.com/sshkhr/Practical_RL/

5) Интернет-ресурс: https://colab.research.google.com/github/yandexdataschool/Practical_RL/

Жукова Наталия Александровна
Куликов Игорь Александрович
Субботин Алексей Николаевич

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ: ЛАБОРАТОРНЫЕ РАБОТЫ

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж

Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А