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

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