**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Информатика, искусственный интеллект и системы управления |
| КАФЕДРА | Системы обработки информации и управления |

# Лабораторная работа №7

**«Алгоритмы Actor-Critic»**

Студент группы ИУ5-23М

Кучин Е.А.

Москва, 2023 г.

Код:

import gym

import numpy as np

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# from tensorflow.python.keras.layers import Dense

# Параметры конфигурации для всей установки

seed = 42

gamma = .99 # Discount factor для прошлых вознаграждений

max\_steps\_per\_episode = 1000

env = gym.make("CartPole-v0")   # Создание среды

env.seed(seed)

eps = np.finfo(np.float32).eps.item()   # Наименьшее число, такое, что 1.0 + eps != 1.0

# ! Построение модели нейронной сети

num\_inputs = 4

num\_actions = 2

num\_hidden = 128

inputs = layers.Input(shape=(num\_inputs,))

common = layers.Dense(num\_hidden, activation = "relu")(inputs)

action = layers.Dense(num\_actions, activation = "softmax")(common)

critic = layers.Dense(1)(common)

model = keras.Model(inputs = inputs, outputs = [action, critic])

# ! Обучение

optimizer = keras.optimizers.Adam(learning\_rate = 0.01)

huber\_loss = keras.losses.Huber()

action\_probs\_history = []

critic\_value\_history = []

rewards\_history = []

running\_reward = 0

episode\_count = 0

while True:     # Выполнять до решения

    state = env.reset()

    episode\_reward = 0

    with tf.GradientTape() as tape:

        for timestep in range(1, max\_steps\_per\_episode):

            # env.render(); Adding this line would show the attempts of the agent in a pop-up window

            state = tf.convert\_to\_tensor(state)

            state = tf.expand\_dims(state, 0)

            # Предсказываем вероятности действий и предполагаемое будущее вознаграждене из состояния среды

            action\_probs, critic\_value = model(state)

            critic\_value\_history.append(critic\_value[0, 0])

            # Выборка действий из распределения вероятностей действий

            action = np.random.choice(num\_actions, p = np.squeeze(action\_probs))

            action\_probs\_history.append(tf.math.log(action\_probs[0, action]))

            # Применение выбранного действия в нашем окружении

            state, reward, done, \_ = env.step(action)

            rewards\_history.append(reward)

            episode\_reward += reward

            if done:

                break

        # Обновление вознаграждения за выполнение для проверки условия решения

        running\_reward = .05 \* episode\_reward + (1 - .05) \* running\_reward

        # Вычислите ожидаемое значение из вознаграждений

        # - На каждом временном шаге каково общее возаграждение, полученное после этого временного шага

        # - Вознаграждение в прошлом дисконтируются путём умножения их на гамму

        # - Это метки для нашего критика

        returns = []

        discounted\_sum = 0

        for r in rewards\_history[::1]:

            discounted\_sum = r + gamma \* discounted\_sum

            returns.insert(0, discounted\_sum)

        # Нормализация

        returns = np.array(returns)

        returns = (returns - np.mean(returns)) / (np.std(returns) + eps)

        returns = returns.tolist()

        # Вычисление значений потерь для обновления нашей сети

        history = zip(action\_probs\_history, critic\_value\_history, returns)

        actor\_losses = []

        critic\_losses = []

        for log\_prob, value, ret in history:

            # В этот момент истории критик оценил, что в будущем мы получим общую награду = `value`. Мы предприняли

            # действие с вероятностью log\_prob и в итоге получили общую награду = `ret`. Актёр должен быть обновлён

            # таким образом, чтобы он подсказывал действия, которое приведёт к высокой награде (по сравнению с оцен

            # кой критика) с высокой вероятностью.

            diff = ret - value

            actor\_losses.append(-log\_prob \* diff)   # actor loss

            # Критик должен быть обновлён так, чтобы он предсказывал лучшую оценку будующих вознаграждений

            critic\_losses.append(

                huber\_loss(tf.expand\_dims(value, 0), tf.expand\_dims(ret, 0))

            )

        # Обратное распространение

        loss\_value = sum(actor\_losses) + sum(critic\_losses)

        grads = tape.gradient(loss\_value, model.trainable\_variables)

        optimizer.apply\_gradients(zip(grads, model.trainable\_variables))

        # Очистка истории потерь и вознаграждений

        action\_probs\_history.clear()

        critic\_value\_history.clear()

        rewards\_history.clear()

    # Log details

    episode\_count += 1

    if episode\_count % 10 == 0:

        # template = "вознаграждение: {:.2f} за эпизод {}"

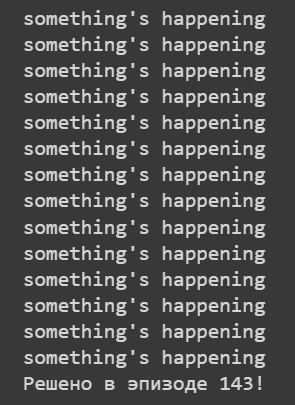
        # print(template.fomat(running\_reward, episode\_count))

        print("something's happening")

    if running\_reward > 150:    # Условие считать задачу решённой

        print("Решено в эпизоде {}!".format(episode\_count))

        break



Визуализация:

