



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 вознаграждение 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
        # вознаграждение, 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 будущих 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
        {}"

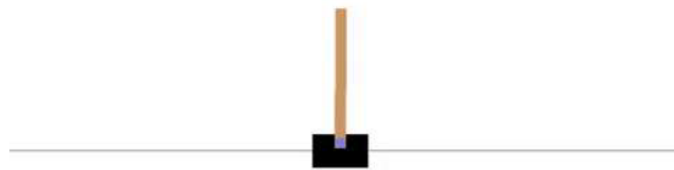
        # print(template.format(running_reward,
episode_count))
        print("something's happening")

        if running_reward > 150:      # Условие считать задачу
        решённой
            print("Решено в эпизоде
        {}!".format(episode_count))
            break

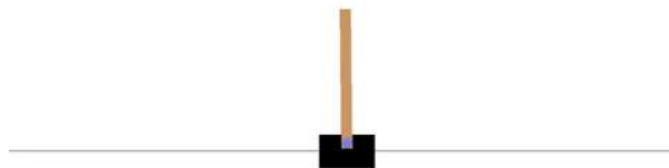
```

something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
something's happening  
Решено в эпизоде 143!

Визуализация:



Ранняя стадия обучения:



Поздняя стадия обучения: