**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 и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Курс «Методы машинного обучения»

Отчет по лабораторной работе №7

«Алгоритмы Actor-Critic»

Выполнил:

студент группы ИУ5-21М Карпов Д.К.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю.Е.

Москва, 2023 г.

**Цель работы:** Ознакомление с базовыми методами обучения с подкреплением на основе алгоритмов Actor-Critic.

**Задание:** На основе рассмотренных на лекции примеров реализовать любой алгоритм семейства Actor-Critic для произвольной среды.

**Описание выполнения**

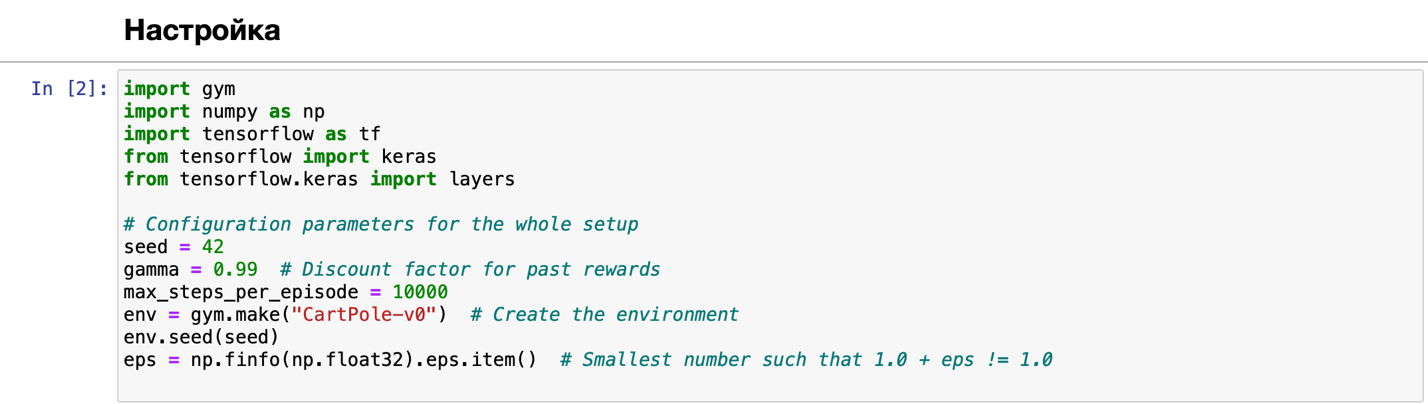


Рисунок 1 – Импорт библиотек и среды

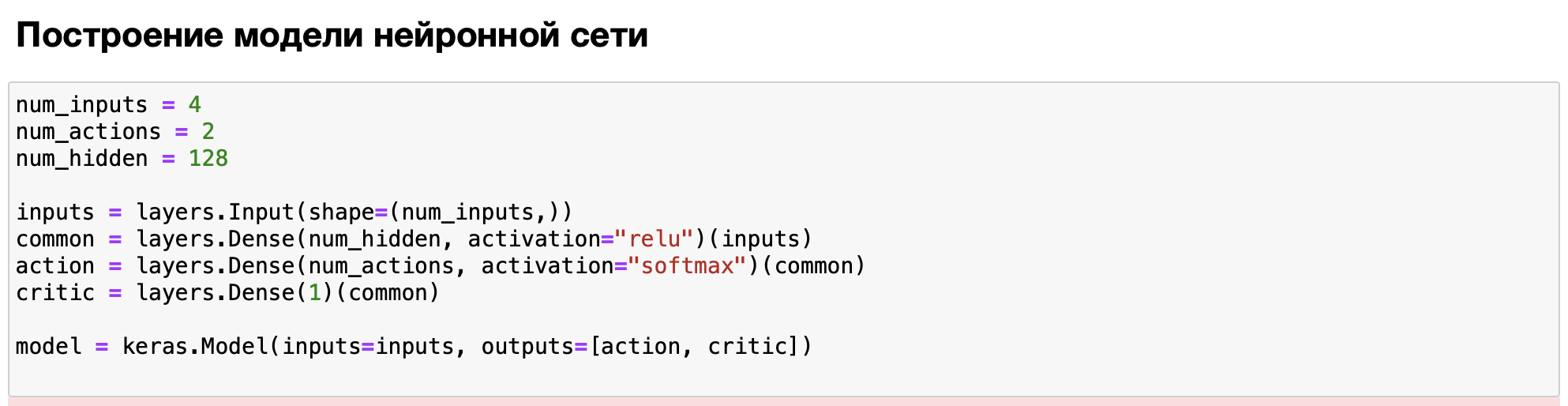


Рисунок 2 – Построение нейронной сети

**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: *# Run until solved* 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)  
  
 *# Predict action probabilities and estimated future rewards  
 # from environment state* action\_probs, critic\_value = model(state)  
 critic\_value\_history.append(critic\_value[0, 0])  
  
 *# Sample action from action probability distribution* action = np.random.choice(num\_actions, p=np.squeeze(action\_probs))  
 action\_probs\_history.append(tf.math.log(action\_probs[0, action]))  
  
 *# Apply the sampled action in our environment* state, reward, done, \_ = env.step(action)  
 rewards\_history.append(reward)  
 episode\_reward += reward  
  
 if done:  
 break  
  
 *# Update running reward to check condition for solving* running\_reward = 0.05 \* episode\_reward + (1 - 0.05) \* running\_reward  
  
 *# Calculate expected value from rewards  
 # - At each timestep what was the total reward received after that timestep  
 # - Rewards in the past are discounted by multiplying them with gamma  
 # - These are the labels for our critic* returns = []  
 discounted\_sum = 0  
 for r in rewards\_history[::-1]:  
 discounted\_sum = r + gamma \* discounted\_sum  
 returns.insert(0, discounted\_sum)  
  
 *# Normalize* returns = np.array(returns)  
 returns = (returns - np.mean(returns)) / (np.std(returns) + eps)  
 returns = returns.tolist()  
  
 *# Calculating loss values to update our network* history = zip(action\_probs\_history, critic\_value\_history, returns)  
 actor\_losses = []  
 critic\_losses = []  
 for log\_prob, value, ret in history:  
 *# At this point in history, the critic estimated that we would get a  
 # total reward = `value` in the future. We took an action with log probability  
 # of `log\_prob` and ended up recieving a total reward = `ret`.  
 # The actor must be updated so that it predicts an action that leads to  
 # high rewards (compared to critic's estimate) with high probability.* diff = ret - value  
 actor\_losses.append(-log\_prob \* diff) *# actor loss  
  
 # The critic must be updated so that it predicts a better estimate of  
 # the future rewards.* critic\_losses.append(  
 huber\_loss(tf.expand\_dims(value, 0), tf.expand\_dims(ret, 0))  
 )  
  
 *# Backpropagation* loss\_value = sum(actor\_losses) + sum(critic\_losses)  
 grads = tape.gradient(loss\_value, model.trainable\_variables)  
 optimizer.apply\_gradients(zip(grads, model.trainable\_variables))  
  
 *# Clear the loss and reward history* action\_probs\_history.clear()  
 critic\_value\_history.clear()  
 rewards\_history.clear()  
  
 *# Log details* episode\_count += 1  
 if episode\_count % 10 == 0:  
 template = "running reward: {:.2f} at episode {}"  
 print(template.format(running\_reward, episode\_count))  
  
 if running\_reward > 150: *# Condition to consider the task solved* print("Solved at episode {}!".format(episode\_count))  
 break

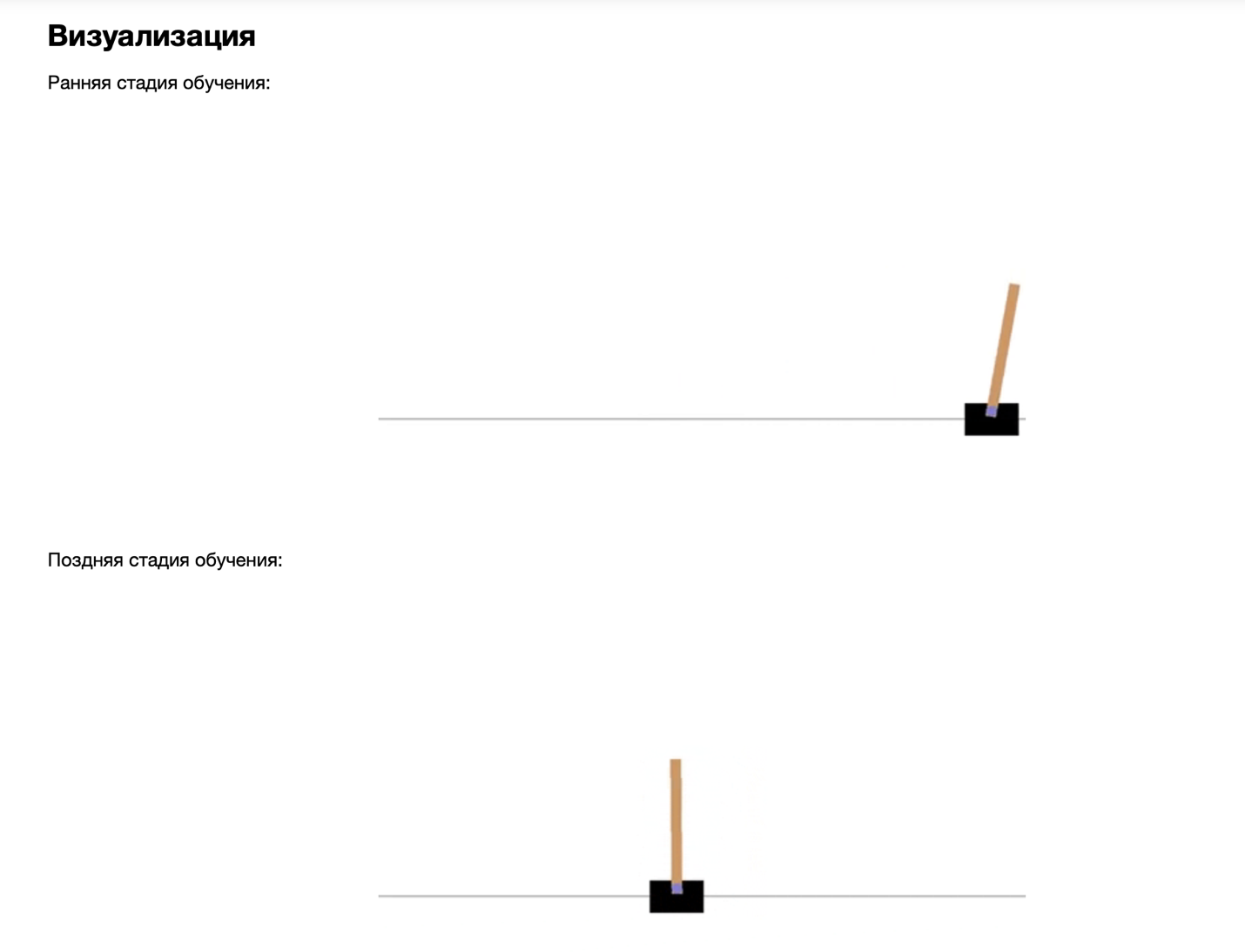


Рисунок 3 – Визуализация этапов обучения

**Вывод**

Таким образом, удалось реализовать алгоритм семейства Actor-Critic для среды обучения с подкреплением, таким образом ознакомившись с базовыми методами обучения с подкреплением на основе подобных алгоритмов.