Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Лабораторная работа № 7

по дисциплине «Методы машинного обучения»

Алгоритмы Actor-Critic

ИСПОЛНИТЕЛЬ:

студент ИУ5-25М
Мацнев А.А.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

"__"___________2023 г.

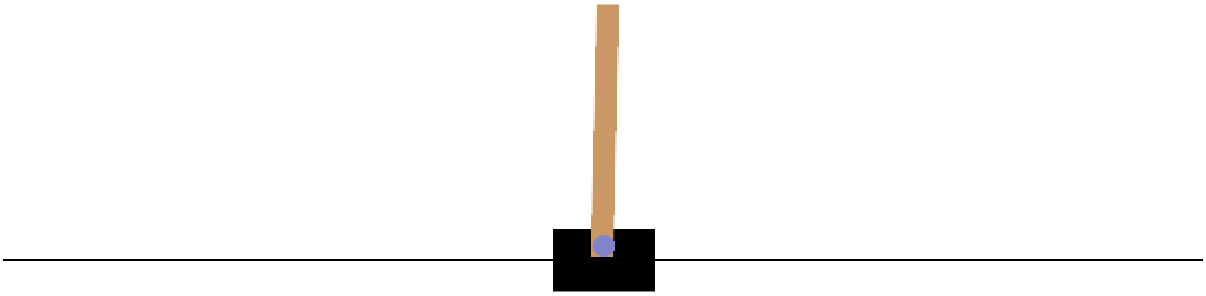
Москва, 2023

Задание

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.

Выполнение задания

Реализуйте любой алгоритм семейства Actor-Critic для произвольной среды.



Текст программы:

```
import gym
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Configuration parameters for the whole setup
seed = 42
gamma = 0.99  # Discount factor for past rewards
max_steps_per_episode = 10000
env = gym.make("CartPole-v1")  # Create the environment
#env.seed(seed)
eps = np.finfo(np.float32).eps.item()  # Smallest number such that 1.0 + eps != 1.0

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])

      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()
          print(state)
          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[0])
                  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
                  observation, reward, terminated, truncated, _ = 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 > 195:  # Condition to consider the task solved
    print("Solved at episode {}!".format(episode_count))
    break
```