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

Лабораторная работа №4
по дисциплине

«Методы машинного обучения»

на тему

«Алгоритм Policy Iteration.»

Выполнил:
студент группы ИУ5-22М
Лун Сыхань

Москва — 2024г.

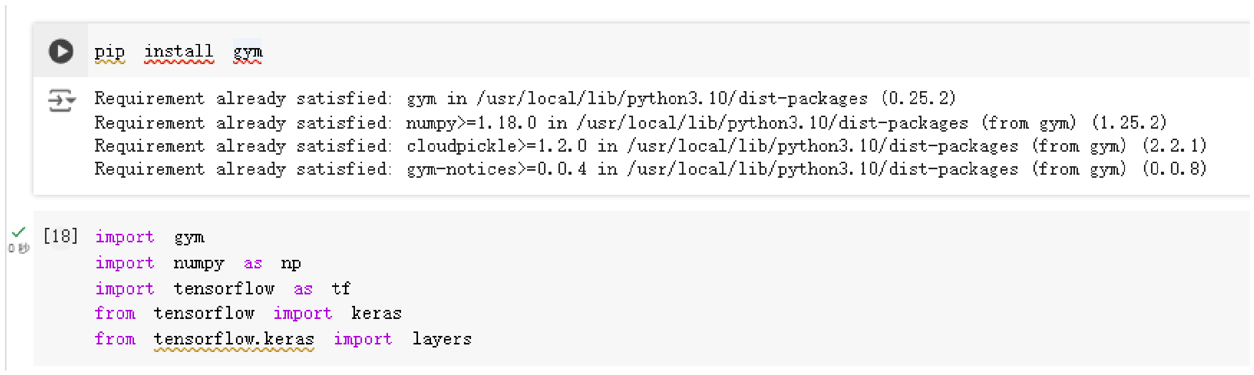
1. Цель лабораторной работы

Ознакомление с базовыми методами обучения с подкреплением.

2. Задание

На основе рассмотренного на лекции примера реализуйте алгоритм Policy Iteration для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

3. Ход выполнения работы



```
pip install gym
```

```
Requirement already satisfied: gym in /usr/local/lib/python3.10/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from gym) (1.25.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gym) (2.2.1)
Requirement already satisfied: gym-notices>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from gym) (0.0.8)
```

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

```
class PolicyNetwork(keras.Model):
    def __init__(self, num_actions):
        super(PolicyNetwork, self).__init__()
        self.dense1 = layers.Dense(24, activation='relu')
        self.dense2 = layers.Dense(24, activation='relu')
        self.logits = layers.Dense(num_actions)

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return tf.nn.softmax(self.logits(x))
```

```
def generate_episode(env, policy_network):
    states = []
    actions = []
    rewards = []
    state = env.reset()
    done = False
    while not done:
        state_tensor = tf.convert_to_tensor(state, dtype=tf.float32)
        state_tensor = tf.expand_dims(state_tensor, 0)
        action_probs = policy_network(state_tensor)
        action = np.random.choice(len(action_probs.numpy()[0]), p=np.squeeze(action_probs.numpy()))
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        states.append(state)
        actions.append(action)
        rewards.append(reward)
        state = next_state
    return np.array(states, dtype=np.float32), np.array(actions, dtype=np.int32), np.array(rewards, dtype=np.float32)
```

```
@tf.function
def policy_update(states, actions, rewards, policy_network, optimizer, gamma=1.0):
    num_actions = policy_network.layers[-1].units
    returns = tf.TensorArray(dtype=tf.float32, size=0, dynamic_size=True)
    G = tf.constant(0.0)
    for t in tf.range(len(rewards) - 1, -1, -1):
        G = rewards[t] + gamma * G
        returns = returns.write(t, G)
    returns = returns.stack()
    returns = (returns - tf.reduce_mean(returns)) / (tf.math.reduce_std(returns) + 1e-8)

    with tf.GradientTape() as tape:
        states_tensor = tf.convert_to_tensor(states, dtype=tf.float32)
        action_mask = tf.one_hot(actions, num_actions)
        action_probs = policy_network(states_tensor)
        selected_action_probs = tf.reduce_sum(action_probs * action_mask, axis=1)
        loss = -tf.reduce_mean(tf.math.log(selected_action_probs) * returns)

    grads = tape.gradient(loss, policy_network.trainable_variables)
    optimizer.apply_gradients(zip(grads, policy_network.trainable_variables))

def policy_improvement(env, policy_network, optimizer, gamma=1.0, num_episodes=1000):
    for _ in range(num_episodes):
        states, actions, rewards = generate_episode(env, policy_network)
        policy_update(states, actions, rewards, policy_network, optimizer, gamma)
```

```python
env = gym.make('CartPole-v1', new_step_api=True)
num_actions = env.action_space.n
policy_network = PolicyNetwork(num_actions)
optimizer = tf.optimizers.Adam(learning_rate=0.01)

# 运行策略迭代算法
policy_improvement(env, policy_network, optimizer, num_episodes=1000)

# 测试训练后的策略网络
num_test_episodes = 10
for episode in range(num_test_episodes):
    state = env.reset()
    done = False
    total_reward = 0
    while not done:
        state_tensor = tf.convert_to_tensor(state, dtype=tf.float32)
        state_tensor = tf.expand_dims(state_tensor, 0)
        action_probs = policy_network(state_tensor)
        action = np.argmax(action_probs.numpy())
        next_state, reward, terminated, truncated, _ = env.step(action)
        done = terminated or truncated
        state = next_state
        total_reward += reward
    print(f"Episode {episode + 1}: Total Reward = {total_reward}")
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function policy_update at 0x7d6d45ee5
WARNING:tensorflow:6 out of the last 6 calls to <function policy_update at 0x7d6d45ee5
Episode 1: Total Reward = 500.0
Episode 2: Total Reward = 500.0
Episode 3: Total Reward = 500.0
Episode 4: Total Reward = 500.0
Episode 5: Total Reward = 500.0
Episode 6: Total Reward = 500.0
Episode 7: Total Reward = 500.0
Episode 8: Total Reward = 500.0
Episode 9: Total Reward = 500.0
Episode 10: Total Reward = 500.0
```