



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

**Отчет по лабораторной работе №6**  
**«Обучение на основе глубоких Q-сетей»**

Студент группы ИУ5-25М  
Зозуля О.А.

Москва, 2023 г.

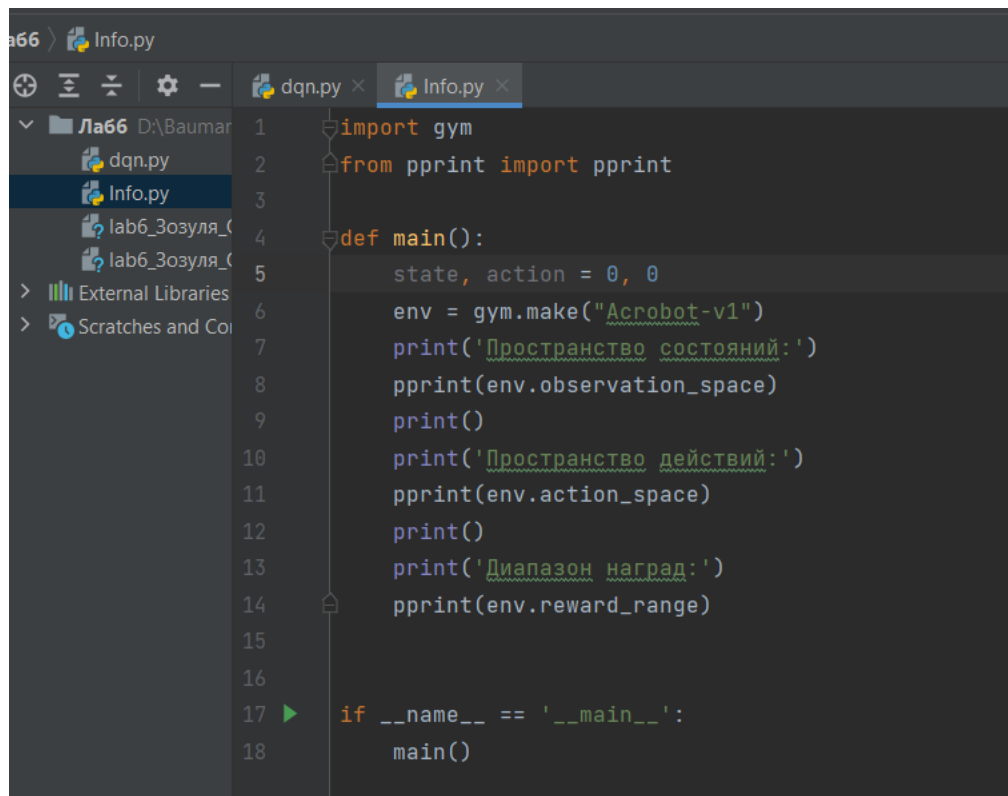
## Цель работы

Ознакомление с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.

## Задание

На основе рассмотренных на лекции примеров реализовать алгоритм DQN. В качестве среды можно использовать классические среды (в этом случае используется полносвязная архитектура нейронной сети) или игры Atari (в этом случае используется сверточная архитектура нейронной сети).

## Описание выполнения



```
1 import gym
2 from pprint import pprint
3
4 def main():
5     state, action = 0, 0
6     env = gym.make("Acrobot-v1")
7     print('Пространство состояний:')
8     pprint(env.observation_space)
9     print()
10    print('Пространство действий:')
11    pprint(env.action_space)
12    print()
13    print('Диапазон наград:')
14    pprint(env.reward_range)
15
16
17 if __name__ == '__main__':
18     main()
```

Рисунок 1 – Информация о среде

```

1  import gym
2  import math
3  import random
4  import matplotlib
5  import matplotlib.pyplot as plt
6  from collections import namedtuple, deque
7  from itertools import count
8
9  import torch
10 import torch.nn as nn
11 import torch.optim as optim
12 import torch.nn.functional as F
13
14
15 # Название среды
16 CONST_ENV_NAME = 'Acrobot-v1'
17 # Использование GPU
18 CONST_DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
19
20 # Элемент ReplayMemory в форме именованного кортежа
21 Transition = namedtuple('Transition',
22                        ('state', 'action', 'next_state', 'reward'))

```

Рисунок 2 – Импорт библиотек и определение начальных параметров

```

24 # Реализация техники Replay Memory
25 class ReplayMemory(object):
26
27     def __init__(self, capacity):
28         self.memory = deque([], maxlen=capacity)
29
30     def push(self, *args):
31         """
32         Сохранение данных в ReplayMemory
33         """
34         self.memory.append(Transition(*args))
35
36     def sample(self, batch_size):
37         """
38         Выборка случайных элементов размера batch_size
39         """
40         return random.sample(self.memory, batch_size)
41
42     def __len__(self):
43         return len(self.memory)
44

```

Рисунок 3 – Сохранения опыта на предыдущих шагах

```

46 class DQN_Model(nn.Module):
47
48     def __init__(self, n_observations, n_actions):
49         """
50         Инициализация топологии нейронной сети
51         """
52         super(DQN_Model, self).__init__()
53         self.layer1 = nn.Conv2d(n_actions, out_channels=32, kernel_size=8, stride=4)
54         self.layer2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2)
55         self.layer3 = nn.Linear(128, n_actions)
56
57     def forward(self, x):
58         """
59         Прямой проход
60         Вызывается для одного элемента, чтобы определить следующее действие
61         Или для batch'а во время процедуры оптимизации
62         """
63         x = F.relu(self.layer1(x))
64         x = F.relu(self.layer2(x))
65         x = F.flatten(self.layer2(x)),
66         return self.layer3(x)

```

Рисунок 4 – Модель полносвязной DQN

```

69 class DQN_Agent:
70
71     def __init__(self, env,
72                 BATCH_SIZE = 128,
73                 GAMMA = 0.99,
74                 EPS_START = 0.9,
75                 EPS_END = 0.05,
76                 EPS_DECAY = 1000,
77                 TAU = 0.005,
78                 LR = 1e-4
79                 ):
80         # Среда
81         self.env = env
82         # Размерности Q-модели
83         self.n_actions = env.action_space.n
84         state, _ = self.env.reset()
85         self.n_observations = len(state)
86         # Коэффициенты
87         self.BATCH_SIZE = BATCH_SIZE
88         self.GAMMA = GAMMA
89         self.EPS_START = EPS_START
90         self.EPS_END = EPS_END
91         self.EPS_DECAY = EPS_DECAY
92         self.TAU = TAU
93         self.LR = LR
94         # Модели
95         # Основная модель
96         self.policy_net = DQN_Model(self.n_observations, self.n_actions).to(CONST_DEVICE)
97         # Вспомогательная модель, используется для стабилизации алгоритма
98         # Обновление контролируется гиперпараметром TAU
99         # Используется подход Double DQN
100        self.target_net = DQN_Model(self.n_observations, self.n_actions).to(CONST_DEVICE)
101        self.target_net.load_state_dict(self.policy_net.state_dict())
102        # Оптимизатор
103        self.optimizer = optim.AdamW(self.policy_net.parameters(), lr=self.LR, amsgrad=True)
104        # Replay Memory
105        self.memory = ReplayMemory(10000)
106        # Количество шагов
107        self.steps_done = 0
108        # Длительность эпизодов
109        self.episode_durations = []
110

```

Рисунок 5 – Определение параметров для модели

```

112 def select_action(self, state):
113     """
114     Выбор действия
115     """
116     sample = random.random()
117     eps = self.EPS_END + (self.EPS_START - self.EPS_END) * \
118         math.exp(-1. * self.steps_done / self.EPS_DECAY)
119     self.steps_done += 1
120     if sample > eps:
121         with torch.no_grad():
122             # Если вероятность больше eps
123             # то выбирается действие, соответствующее максимальному Q-значению
124             # t.max(1) возвращает максимальное значение колонки для каждой строки
125             # [1] возвращает индекс максимального элемента
126             return self.policy_net(state).max(1)[1].view(1, 1)
127     else:
128         # Если вероятность меньше eps
129         # то выбирается случайное действие
130         return torch.tensor([self.env.action_space.sample()], device=CONST_DEVICE, dtype=torch.long)
131
132
133 def plot_durations(self, show_result=False):
134     plt.figure(1)
135     durations_t = torch.tensor(self.episode_durations, dtype=torch.float)
136     if show_result:
137         plt.title('Результат')
138     else:
139         plt.clf()
140         plt.title('Обучение...')
141     plt.xlabel('Эпизод')
142     plt.ylabel('Количество шагов в эпизоде')
143     plt.plot(durations_t.numpy())
144     plt.pause(0.001) # пауза
145

```

Рисунок 6 – Выбор действия и график шагов по эпизодам

```

147 def optimize_model(self):
148     """
149     Оптимизация модели
150     """
151     if len(self.memory) < self.BATCH_SIZE:
152         return
153     transitions = self.memory.sample(self.BATCH_SIZE)
154     # Транспонирование batch'a
155     # (https://stackoverflow.com/a/19343/3343043)
156     # Конвертация batch-массива из Transition
157     # в Transition batch-массивов.
158     batch = Transition(*zip(*transitions))
159
160     # Вычисление маски нефинальных состояний и конкатенация элементов batch'a
161     non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
162                                             batch.next_state)), device=CONST_DEVICE, dtype=torch.bool)
163     non_final_next_states = torch.cat([s for s in batch.next_state
164                                       if s is not None])
165     state_batch = torch.cat(batch.state)
166     action_batch = torch.cat(batch.action)
167     reward_batch = torch.cat(batch.reward)
168
169     # Вычисление Q(s_t, a)
170     state_action_values = self.policy_net(state_batch).gather(1, action_batch)
171
172     # Вычисление V(s_{t+1}) для всех следующих состояний
173     next_state_values = torch.zeros(self.BATCH_SIZE, device=CONST_DEVICE)
174     with torch.no_grad():
175         next_state_values[non_final_mask] = self.target_net(non_final_next_states).max(1)[0]
176     # Вычисление ожидаемых значений Q
177     expected_state_action_values = (next_state_values * self.GAMMA) + reward_batch
178
179     # Вычисление Huber loss
180     criterion = nn.SmoothL1Loss()
181     loss = criterion(state_action_values, expected_state_action_values.unsqueeze(1))
182
183     # Оптимизация модели
184     self.optimizer.zero_grad()
185     loss.backward()
186     # gradient clipping
187     torch.nn.utils.clip_grad_value_(self.policy_net.parameters(), 100)
188     self.optimizer.step()
189

```

Рисунок 7 – Оптимизация модели

```

191 def play_agent(self):
192     """
193     Проигрывание сессии для обученного агента
194     """
195     env2 = gym.make(CONST_ENV_NAME, render_mode='human')
196     state = env2.reset()[0]
197     state = torch.tensor(state, dtype=torch.float32, device=CONST_DEVICE).unsqueeze(0)
198     done = False
199     res = []
200     while not done:
201         action = self.select_action(state)
202         action = action.item()
203         observation, reward, terminated, truncated, _ = env2.step(action)
204         env2.render()
205
206         res.append((action, reward))
207
208         if terminated:
209             next_state = None
210         else:
211             next_state = torch.tensor(observation, dtype=torch.float32, device=CONST_DEVICE).unsqueeze(0)
212
213         state = next_state
214         if terminated or truncated:
215             done = True
216
217     print('Данные за эпизод: ', res)
218

```

Рисунок 8 – Проигрывание обученной модели

```

221 def learn(self):
222     """
223     Обучение агента
224     """
225     if torch.cuda.is_available():
226         num_episodes = 600
227     else:
228         num_episodes = 50
229
230     for i_episode in range(num_episodes):
231         # Инициализация среды
232         state, info = self.env.reset()
233         state = torch.tensor(state, dtype=torch.float32, device=CONST_DEVICE).unsqueeze(0)
234         for t in count():
235             action = self.select_action(state)
236             observation, reward, terminated, truncated, _ = self.env.step(action.item())
237             reward = torch.tensor([reward], device=CONST_DEVICE)
238
239             done = terminated or truncated
240             if terminated:
241                 next_state = None
242             else:
243                 next_state = torch.tensor(observation, dtype=torch.float32, device=CONST_DEVICE).unsqueeze(0)
244
245             # Сохранение данных в Replay Memory
246             self.memory.push(state, action, next_state, reward)
247
248             # Переход к следующему состоянию
249             state = next_state
250
251             # Выполнение одного шага оптимизации модели
252             self.optimize_model()
253
254             # Обновление весов target-сети
255             #  $\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$ 
256             target_net_state_dict = self.target_net.state_dict()
257             policy_net_state_dict = self.policy_net.state_dict()
258             for key in policy_net_state_dict:
259                 target_net_state_dict[key] = policy_net_state_dict[key]*self.TAU + target_net_state_dict[key]*(1-self.TAU)
260             self.target_net.load_state_dict(target_net_state_dict)
261
262             if done:
263                 self.episode_durations.append(t + 1)
264                 self.plot_durations()
265                 break
266
267
268 def main():
269     env = gym.make(CONST_ENV_NAME)
270     agent = DQN_Agent(env)
271     agent.learn()
272     agent.play_agent()
273
274 if __name__ == '__main__':
275     main()
276

```

Рисунок 9 – Обучение модели и запуск среды

## Вывод

Таким образом, удалось реализовать алгоритм DQN для среды обучения с подкреплением, таким образом ознакомившись с базовыми методами обучения с подкреплением на основе глубоких Q-сетей.