СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2020. – № 1–2 (97).** – 7–25

АВТОМАТИЧЕСКОЕ УПРАВЛЕНИЕ И ИДЕНТИФИКАЦИЯ

УДК 519.24 DOI: 10.17212/2307-6879-2020-1-2-7-25

ПРИМЕНЕНИЕ АЛГОРИТМОВ DEEP Q-LEARNING И DOUBLE DEEP Q-LEARNING К ЗАДАЧЕ УПРАВЛЕНИЯ ПЕРЕВЕРНУТЫМ МАЯТНИКОМ*

А.А. ЕВСЕЕНКО[1], Д.О. РОМАННИКОВ[2]

[1] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, магистрант факультета автоматики и вычислительной техники. E-mail: vnattali@mail.ru*

[2] *630073, РФ, г. Новосибирск, пр. Карла Маркса, 20, Новосибирский государственный технический университет, кандидат технических наук, доцент кафедры автоматики. E-mail:dmitry.romannikov@gmail.com*

На сегодняшний день в мире бурно развивается такой раздел науки, как «искусственный интеллект». Системы, построенные на основе методов искусственного интеллекта, обладают свойством выполнять функции, которые традиционно считаются прерогативой человека. Искусственный интеллект обладает широким спектром областей исследований. Одной из таких областей является машинное обучение. В данной статье рассматриваются алгоритмы одного из подходов машинного обучения – обучение с подкреплением (англ. reinforcement learning, RL), по которому осуществляются много исследований и разработок в течение последних семи лет. Разработки и исследования по данному подходу в основном осуществляются для решения задач в играх Atari 2600 или им подобных. В данной статье обучение с подкреплением будет применяться к одному из динамических объектов – к перевернутому маятнику. В качестве модели указанного объекта рассматривается модель перевернутого маятника на тележке, взятая из библиотеки Gym, в которой находится много моделей, используемых для тестирования и анализа алгоритмов обучения с подкреплением. В статье приводится реализация и исследование двух алгоритмов из данного подхода – Deep Q-learning и Double Deep Q-learning. В качестве результата представлены графики обучения, тестирования и времени обучения для каждого алгоритма, на основе которых делается вывод, что желательно использовать алгоритм Double Deep Q-learning, потому что время обучения составляет приблизительно 2 минуты и осуществляется наилучшее управление моделью перевернутого маятника на тележке.

Ключевые слова: нейронные сети, искусственный интеллект, модель перевернутого маятника, Python, Gym, Pytorch, Deep Q-learning (DQN), Double Deep Q-learning (DDQN), обучение с подкреплением (reinforcement learning, RL)

* Статья получена 04 мая 2020 г.

ВВЕДЕНИЕ

Подход к обучению с подкреплением начал бурно развиваться после выпуска в 2013 году компанией DeepMind статьи о глубоком обучении с подкреплением для игр Atari 2600 [1]. При использовании обучения с подкреплением не нужно формировать тестовую и обучающую выборки. Обучение с подкреплением (англ. Reinforcement learning, RL) – один из подходов машинного обучения, в ходе которого агент обучается, взаимодействуя с некоторой средой. В качестве агента выступает нейронная сеть, обучаемая с помощью соответствующего алгоритма, а среды – модель объекта (объект) и ее поведение. В данной статье указанный подход машинного обучения используется для управления моделью перевернутого маятника на тележке. В качестве алгоритмов обучения с подкрепление реализуются и исследуются алгоритмы Deep Q-learning и Double Deep Q-learning.

1. ОПИСАНИЕ ЗАДАЧИ

Перевернутый маятник (рис. 1) состоит из передвижной тележки *1*, на которой расположен сам маятник *2*. Маятник прикреплен на шарнир *3* и может осуществлять вращение на 360°. Данный объект удерживается в перевернутом состоянии за счет изменения скорости тележки. Возможны два действия, оказываемые на тележку, при которых она движется горизонтально: вправо или влево. Основная задача заключается в разработке системы, которая удерживает шест перевернутого маятника в вертикальном положении. При использовании обучения с подкреплением для решения этой задачи необходимо для каждого состояния s объекта управления правильно выбирать действие, оказываемое на тележку с маятником. В статье используется модель перевернутого маятника на тележке из библиотеки Gym – CartPole-v0) [2].

У данной модели определены следующие параметры [3, 4].

1. *Состояние*, которое описывается заданными величинами:

- позиция тележки, которая принимает значения в диапазоне [–2.4…2.4];
- скорость тележки;
- угол отклонения шеста от вертикального положения, который принимает значения в диапазоне [–41.8°…41.8°];
- скорость изменения наклона угла шеста.

2. *Действие*, которое принимает значения:

- 0 – приложить к тележке горизонтальную силу, направленную влево;
- 1 – приложить к тележке горизонтальную силу, направленную вправо.

3. *Награда*, которая на каждом шаге равна единице, включая последний шаг.

4. *Начальное состояние* задается при помощи датчика равномерно распределенных значений в диапазоне [–0.5…0.5].

5. Случаи завершения эпизода:

- угол отклонения шеста от вертикального положения вышел из диапазона [–12°…12°];
- позиция тележки вышла из допустимого диапазона значений [–2.4…2.4];
- длина эпизода превышает значение 200.

Задача для данной модели считается решенной в обучении с подкреплением, если средняя награда за 100 последовательных эпизодов обучения не меньше значения 195.

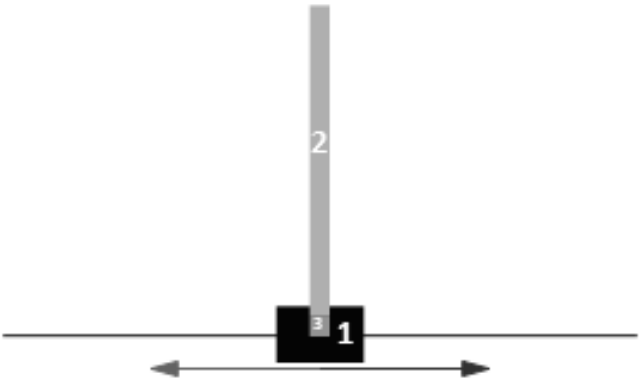

Рис. 1. Иллюстративное представление задачи перевернутого маятника

Модель перевернутого маятника из Gym реализована на основе статьи [5]. Она описывается следующими дифференциальными уравнениями без учета сил трения и сопротивления:

$$\ddot{\theta}_t = \frac{g \sin \theta_t + \cos \theta_t \left[\dfrac{-F_t - ml\dot{\theta}_t \sin \theta_t}{m_c + m} \right]}{l \left[\dfrac{4}{3} - \dfrac{m \cos^2 \theta_t}{m_c + m} \right]},$$

$$\ddot{x}_t = \frac{F_t + ml\left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right]}{m_c + m},$$

где θ_t – угол отклонения шеста от вертикального положения в момент времени t; $\dot{\theta}_t$ – скорость изменения наклона угла шеста в момент времени t; x_t – позиция тележки в момент времени t; $\dot{x}_t$ – скорость тележки в момент времени t; $g = 9.8$ м/с^2; $m_c = 1.0$ кг – масса тележки; $m = 0.1$ кг – масса шеста; $l = 0.5$ м – половина длины шеста; $F_t = \pm 10$ Н – сила, приложенная к центру масс тележки в момент времени t.

Управление перевернутым маятником в теории автоматического управления считается сложной задачей, так как сама система является неустойчивой, а переходный процесс передаточной функции – колебательным. Описав среду, в качестве которой выступает модель перевернутого маятника, необходимо описать агентов и алгоритмы обучения с подкреплением, которые нам позволят решить поставленную здесь задачу управления.

2. АЛГОРИТМЫ DEEP Q-LEARNING И DOUBLE DEEP Q-LEARNING

Алгоритм Deep Q-learning входит в группу алгоритмов Q-learning, оперирующих функцией качества $Q(s, a)$, обозначающей наибольшую возможную награду, которую может получить агент, выполняя действие a в состоянии s. Deep Q-learning использует нейронную сеть для аппроксимации функции $Q(s, a)$. Нейронная сеть получает состояние s, после чего она возвращает значения Q для каждого действия в этом состоянии.

Одна из особенностей использования нейронных сетей в обучении с подкреплением – это отсутствие гарантий сходимости. Для улучшения сходимости используется подход, известный как повторение опыта (*experience replay*). Данный подход предполагает наличие некоторого хранилища с определенным размером, который называется памятью воспроизведения (*replay memory*), где хранятся последние N (размер памяти воспроизведения) опытов агента (алгоритма), представленные в виде (s_t, a_t, r_t, s_{t+1}), где s_t – состояние объекта в момент времени t; a_t – действие в момент времени t; r_t – награда, полученная за выполнение действия a_t в момент времени t; s_{t+1} – состояние объекта

в момент времени $t+1$. При обучении используется случайная выборка определенного размера из памяти воспроизведения и применятся обновление Q-learning. После воспроизведения опыта агент выбирает и выполняет действие в соответствии с ε-жадной политикой. В итоге на основе описанного алгоритма в статье [1] и применительно к модели перевернутого маятника из библиотеки Gym получаем следующий пошаговый алгоритм:

```
1. Инициализовать память воспроизведения размерностью N ([state, action,
reward, next_state, done])
2. Инициализовать случайными весами функцию Q(s,a) (т. е. нейронную сеть)
3. Повторять для каждой игры
  4. Инициализовать s
  5. Повторять для каждого шага
    6. Выбрать a по s (ε-жадную)
    7. Выполнить a, найти r, s', done
    8. Занести в память воспроизведения [s, a, r, s', done]
    9. Выбрать случайным образом из памяти воспроизведения коллекцию [s,
a, r, s', done]
    10. y_j = { r_j,  если  done = true
              { r_j + γ max_a Q(s',a),  если  done = false
    11. Выполнить градиентный спуск на (y_j − Q(s, a))^2
    12. s = s'
```

Преимущества алгоритма, представленного в статье [1]:

1) каждый шаг опыта потенциально используется во многих обновлениях весов нейронной сети, это позволяет повысить эффективность данных;

2) обучение непосредственно из последовательных выборок неэффективно из-за сильной корреляции между выборками;

3) использование случайных выборок нарушает эти корреляции и, следовательно, уменьшает дисперсию в обновлениях.

Кроме алгоритма Deep Q-learning в нашей статье реализуется и исследуется еще один алгоритм из группы Q-learning – Double Deep Q-learning, который также был разработан компанией DeepMind в 2015 году и представлен в статье [6]. Идея Double Deep Q-learning заключается в том, что такие операции, как выбор действия и оценка действия, разделены на две нейронные сети, такие как основная (*online network*) и целевая (*target network*). При этом архитектура нейронных сетей остается одинаковой, целевая нейронная сеть явля-

ется копией основной, веса которой обновляются с определенной периодичностью. В данной алгоритме предлагается оценить ε-жадную политику в соответствии с основной нейронной сетью, но используя целевую сеть, чтобы оценить ее значения. В итоге на основе алгоритма статьи [6] и применительно к модели перевернутого маятника из библиотеки Gym получаем следующий алгоритм:

1. Инициализовать память воспроизведения размерностью N ([state, action, reward, next_state, done])
2. Инициализовать случайными весами функцию Q(s,a) (т. е. основную нейронную сеть)
3. Инициализовать случайными весами функцию Q`(s,a) (т. е. целевую нейронную сеть)
4. Инициализировать шаг обновления функции Q`(s,a)
  5. Повторять для каждой игры
  6. Инициализовать s
  7. Повторять для каждого шага
    8. Выбрать a по s (ε-жадную)
    9. Выполнить a, найти r, s', done
    10. Занести в память воспроизведения [s, a, r, s', done]
    11. Выбрать случайным образом из памяти воспроизведения коллекцию [s, a, r, s', done]
    12. $y_j = \begin{cases} r_j, & \text{если } done = true \\ r_j + \gamma Q`(s', \operatorname{argmax}_a Q(s', a)), & \text{если } done = false \end{cases}$
    13. Выполнить градиентный спуск на $(y_j - Q(s, a))^2$
    14. s = s'
    15. Обновить функцию Q`(s,a) в соответствии с установленным шагом

Алгоритм Double Deep Q-learning является некоторой модификацией алгоритма Deep Q-learning. Цель создания алгоритма Double Deep Q-learning заключается в том, чтобы получить большую часть преимуществ от Q-learning, сохраняя при этом остальную часть алгоритма Deep Q-learning без изменений для правильного сравнения, и с минимальными вычислительными затратами.

Описав принцип работы алгоритмов Deep Q-learning и Double Deep Q-learning, перейдем к их реализации и исследованию.

3. РЕАЛИЗАЦИЯ АЛГОРИТМОВ DEEP Q-LEARNING И DOUBLE DEEP Q-LEARNING

Для реализации алгоритмов Deeep Q-learning и Double Deep Q-learning потребовались следующие библиотеки Python [7]:

- math – библиотека для работы с числами и математическими операциями;
- random – библиотека для генерации случайных чисел;
- numpy – библиотека для работы с многомерными массивами данных;
- matplotlib.pyplot – библиотека для визуализации данных и построения графиков;
- time – библиотека для работы со временем.

Дополнительно также потребовались модули фреймворка PyTorch [8]:

- torch – модуль самого фреймворка PyTorch;
- torch.nn – модуль, который отвечает за архитектуру нейронных сетей;
- torch.optim – модуль, который отвечает за методы оптимизации нейронных сетей;
- torch.cuda – модуль, который позволяет использовать возможности видеокарты, поддерживающей архитектуру cuda.

Для работы с моделью перевернутого маятника (CartPole-v0) потребовались следующие инструменты из Gym [2]:

- env.reset() – завершает текущий эпизод и начинает новый; возвращает начальное состояние;
- env.render() – отображает текущее состояние среды;
- env.step() – совершает указанное действие; с его помощью получаем новое состояние, награду; завершает эпизод;
- env.action_space.n – показывает количество действий, определенных в модели;
- env.obsrevation_space.share[0] – показывает количество состояний у модели.

Разработанная нейронная сеть для двух алгоритмов представляет собой многослойную нейронную сеть, состоящую из трех слоев: 1-й слой состоит из четырех входных (по количеству состояний модели) и 128 выходных нейронов, 2-й слой состоит из 128 входных и выходных нейронов, а 3-й слой состоит из 128 входных и двух выходных (по количеству действий, определенных в модели) нейронов (рис. 2) [9–11].

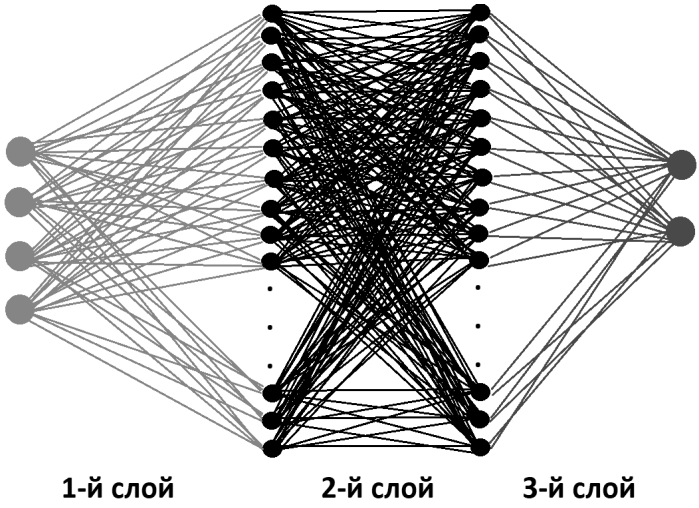

Рис. 2. Схема разработанной многослойной нейронной сети для двух алгоритмов: Deep Q-learning и Double Deep Q-learning

Обучение вышеприведенной сети выполняется методом стохастической оптимизации (*Adam*) [12].

Реализация алгоритмов Deep Q-learning и Double Deep Q-learning на языке программирования Python с использованием фреймворка машинного обучения PyTorch осуществлялась с использованием объектно-ориентированного подхода через классы: *class MNN* – класс, в котором описана структура разработанной нейронной сети (см. рис. 2); *class ReplayBuffer* – класс, в котором реализован метод воспроизведения опыта (*experience replay*); *class Tester* – класс, который отвечает за процесс тестирования обученной нейронной сети на модели перевернутого маятника на тележке; *class Configuration* – класс, в котором хранятся параметры алгоритмов обучения и нейронной сети; *class DQNAgent и DDQNAgent* – классы, в которых реализованы алгоритмы работы Deep Q-learning и Double Deep Q-learning, описанные в разделе 2 настоящей статьи; *class Trainer* – класс, в котором реализован процесс обучения нейронной сети в соответствии с алгоритмом обучения (Deep Q-learning или Double Deep Q-learning). Взаимосвязь классов и подробное их представление показано на рис. 3 и 4.

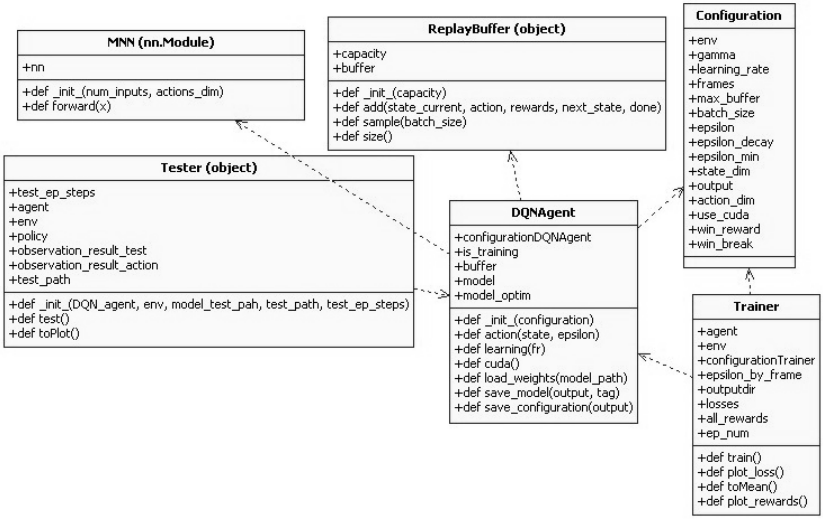

Рис. 3. Диаграмма классов реализации алгоритма Deep Q-learning и машинного обучения Reinforcement Learning

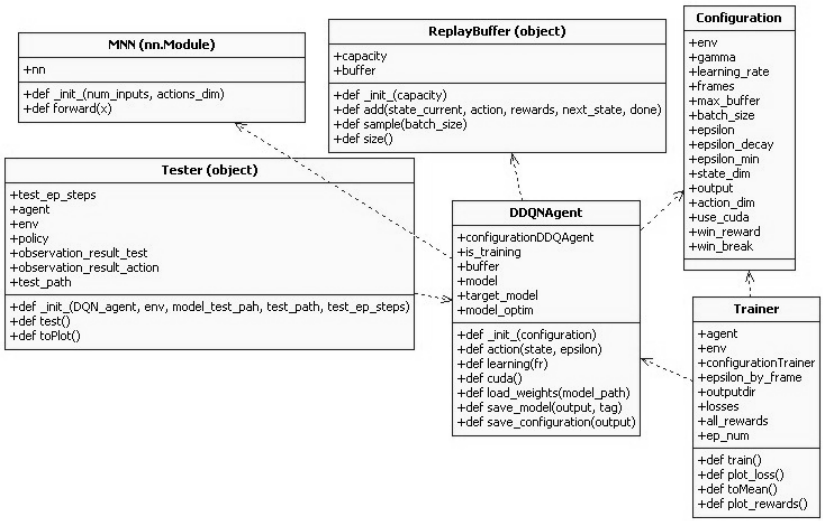

Рис. 4. Диаграмма классов реализации алгоритма Double Deep Q-learning и машинного обучения Reinforcement Learning

Ниже приведем реализацию работы алгоритма Deep Q-learning, представленную в разделе 2, на языке программирования Python с использованием фреймворка PyTorch:

```
#обучение по DQN
def learning(self, fr):
  #получаем данные из буфера воспроизведения
  state_current, action, reward, state_next, done =
self.buffer.sample(self.configigurationDQAgent.BATCH_SIZE)
  #вытаскиваем из тензора
  state_current = torch.tensor(state_current, dtype=torch.float)
  state_next = torch.tensor(state_next, dtype=torch.float)
  action = torch.tensor(action, dtype=torch.long)
  reward = torch.tensor(reward, dtype=torch.float)
  done = torch.tensor(done, dtype=torch.float)
  #если испольуется CUDA, то используем CUDA
  if self.configigurationDQAgent.USE_CUDA:
    state_current = state_current.cuda()
    state_next = state_next.cuda()
    action = action.cuda()
    reward = reward.cuda()
    done = done.cuda()
  #определяем значения Q(s,a) для текущего state_current
  q_values = self.model(state_current).cuda()
  #определяем значения Q(s,a) для следующего state_next
  next_q_values = self.model(state_next).cuda()
  #определяем максимальное значение state_next
  next_q_value = next_q_values.max(1)[0]
  #определяем значение Q(s,a) для текущего state_current по действию
action
  q_value = q_values.gather(1, action.unsqueeze(1)).squeeze(1)
  #высчитываем y
  expected_q_value = reward + self.configigurationDQAgent.GAMMA *
next_q_value * (1 - done)
  #Вычисляем ошибку или функцию потерь
  loss = (q_value - expected_q_value.detach()).pow(2).mean()
  #оптимизируем модель
  self.model_optim.zero_grad()
  loss.backward()
  self.model_optim.step()
#возвращаем потерю или ошибку
return loss.item()
```

Далее покажем реализацию работы алгоритма Double Deep Q-learning, представленную в разделе 2, на языке программирования Python с использованием фреймворка PyTorch:

```
#обучение по DDQN
def learning(self, fr):
  #получаем данные из буфера воспроизведения
  state_current, action, reward, state_next, done =
self.buffer.sample(self.configigurationDQAgent.BATCH_SIZE)
  #вытаскиваем из тензора
  state_current = torch.tensor(state_current, dtype=torch.float)
  state_next = torch.tensor(state_next, dtype=torch.float)
  action = torch.tensor(action, dtype=torch.long)
  reward = torch.tensor(reward, dtype=torch.float)
  done = torch.tensor(done, dtype=torch.float)
  #если испольуется CUDA, то используем CUDA
  if self.configigurationDQAgent.USE_CUDA:
    state_current = state_current.cuda()
    state_next = state_next.cuda()
    action = action.cuda()
    reward = reward.cuda()
    done = done.cuda()
  #определяем значения Q(s,a) для текущего state_current
  q_values = self.model(state_current).cuda()
  #определяем значения Q(s,a) для следующего state_next
  next_q_values = self.model(state_next).cuda()
  #определяем значения Q`(s,a) для следующего state_next
  next_q_state_values = self.target_model(state_next).cuda()
  #определяем значение Q(s,a) для текущего state_current по действию
action
  q_value = q_values.gather(1, action.unsqueeze(1)).squeeze(1)
  #находим значение Q`(s,argmax_aQ(s,a)) для следующего state_next
  next_q_value = next_q_state_values.gather(1,
next_q_values.max(1)[1].unsqueeze(1)).squeeze(1)
  #высчитываем y
  expected_q_value = reward + self.configigurationDQAgent.GAMMA*
next_q_value * (1 - done)
  #Вычисляем ошибку или функцию потерь
  loss = (q_value - expected_q_value.detach()).pow(2).mean()
  #оптимизируем модель
  self.model_optim.zero_grad()
```

```
  loss.backward()
  self.model_optim.step()
  #обновляем Q`(s,a) в соответствии с установленным шагом
  if fr % self.configigurationDQAgent.UPDATE_TARGET_INTERVAL == 0:
    self.target_model.load_state_dict(self.model.state_dict())
#возвращаем потерю или ошибку
return loss.item()
```

4. ИССЛЕДОВАНИЕ АЛГОРИТМОВ DEEP Q-LEARNING И DOUBLE DEEP Q-LEARNING

Исследование алгоритмов Deep Q-learning и Double Deep Q-learning, реализованных в разделе 3 данной статьи, проходило в 2 этапа: обучение и тестирование.

1. Процесс обучения представлял собой обучение нейронной сети по одному из двух реализованных алгоритмов с одинаковыми исходными параметрами. Параметры при использовании алгоритма Deep Q-learning: *gamma* = 0.99; ε = 1, ε*_min* = 0.01, ε*_decay* = 500 – для расчета ε-жадной политики; *frame* = 160 000 – количество кадров(шагов) обучения; *learning_rate* = 10^{-3} – скорость обучения; *max_buffer* = 1000 – объем памяти воспроизведения; *batch_size* = 128 – размер случайной партии из памяти воспроизведения; *win_reward* = 198. Параметры при использовании алгоритма Double Deep Q-learning такие же, как у алгоритма Deep Q-learning, только добавляется параметр *tau* = 100 – обновление целевой сети (*target network*). Для достижения решения поставленной задачи управления перевернутым маятником на тележке обучение завершается, если за последние 100 эпизодов обучения среднее значение наград превышает или равно 198 (*win_reward*) и награда за последний эпизод обучения больше или равна 198. Данное значение завершения обучения было выбрано равным 198 на основе исследований, представленных в статье [1].

2. Процесс тестирования представлял собой применение обученной нейронной сети для управления перевернутым маятником. Тестирование проводилось до 200 временных шагов, которые указаны в параметрах завершения эпизода модели перевернутого маятника на тележке CartPole-v0 (см. раздел 2).

Исследование проводилось с использованием возможностей видеокарты MSI GeForce GTX 1050 Ti [13], которая поддерживает архитектуру cuda [14]. Исследования алгоритмов Deep Q-learning и Double Deep Q-learning осуществлялись по времени обучения, по количеству эпизодов обучения и по тому, как управляется модель перевернутого маятника обученной нейронной сетью (рис. 5–9) [15].

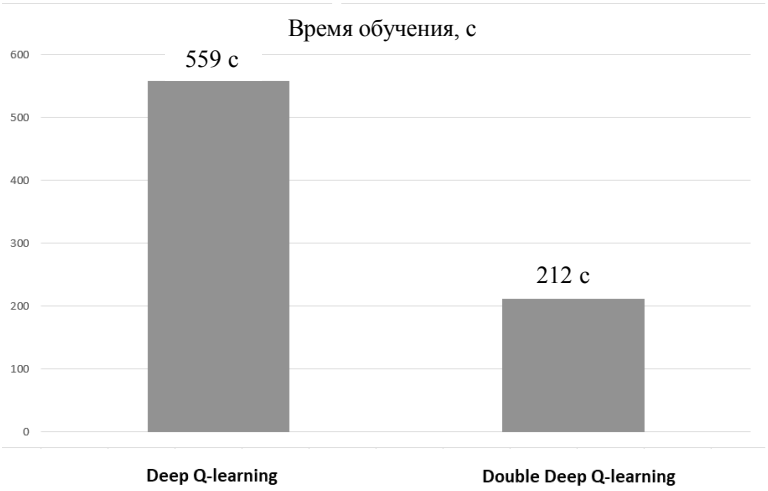

Рис. 5. Время обучения нейронной сети при использовании алгоритмов Deep Q-learning и Double Deep Q-learning

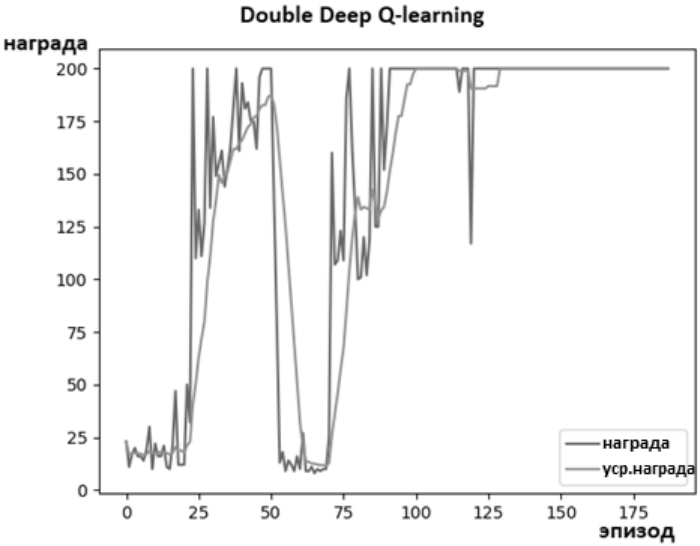

Рис. 6. Процесс обучения при использовании алгоритма Double Deep Q-learning

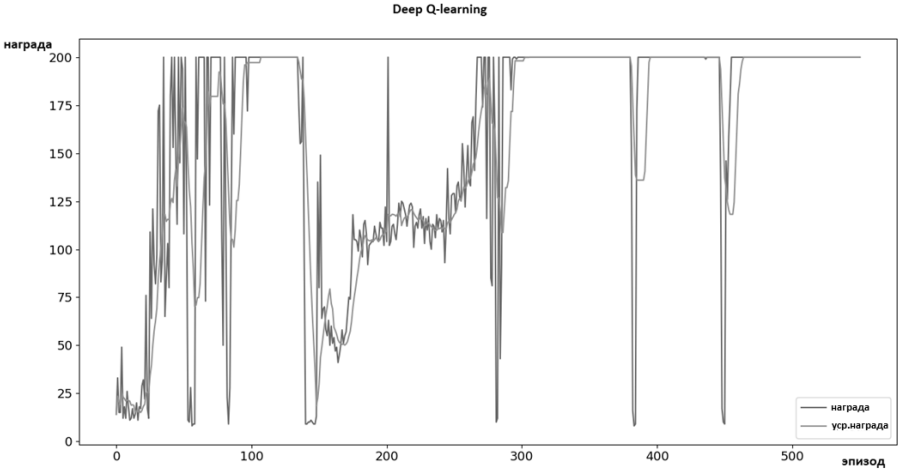

Рис. 7. Процесс обучения при использовании алгоритма Deep Q-learning

Как видно из рис. 5, алгоритм Double Deep Q-learning намного быстрее обучает нейронную сеть, чем алгоритм Deep Q-learning. Кроме того, количество эпизодов при обучении с помощью алгоритма Double Deep Q-learning составило всего 187. Начиная с 88-го эпизода у нейронной сети появилась возможность управлять моделью перевернутого маятника (так как награда составила 200 – максимальное время устойчивости данной модели) (см. рис. 6). Однако чтобы также начать управлять моделью при использовании алгоритма Deep Q-learning, потребовался 551 эпизод обучения, только с 451-го эпизода нейронная сеть была в состоянии управлять моделью перевернутого маятника (рис. 7).

Протестируем обученные нейронные сети на управление моделью перевернутого маятника на тележке (рис. 8 и 9). При обученной нейронной сети маятник должен удерживаться в вертикальном положении на всём временном промежутке (т. е. в течение 200 временных шагов). Такие параметры, как скорость тележки и скорость изменения наклона шеста, должны сходиться к отметке 0, и если есть отклонения в позиции тележки, то они не должны слишком сильно отклоняться от оси абсцисс [16]. С этим справилась нейронная сеть, обученная по алгоритму Double Deep Q-learning.

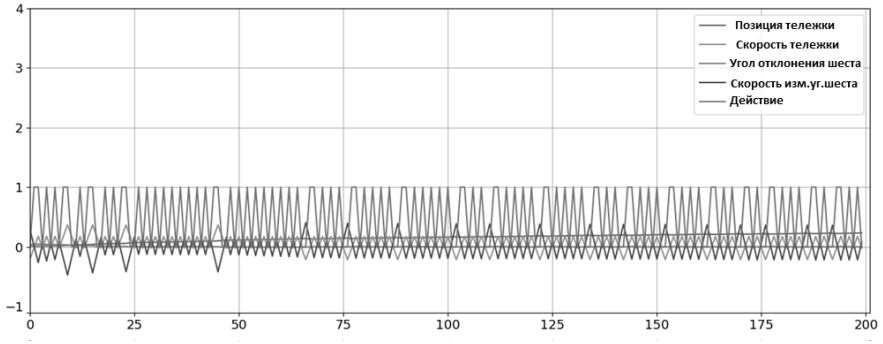

Рис. 8. Процесс тестирования после использования алгоритма Deep Q-learning

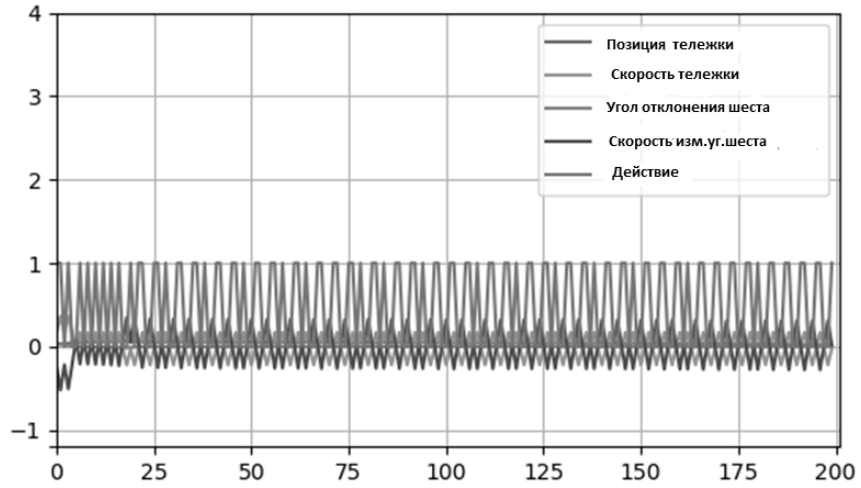

Рис. 9. Процесс тестирования после использования алгоритма Double Deep Q-learning

На основе рис. 8 и 9 можно сделать вывод, что для получения наилучшего результата по управлению моделью перевернутого маятника с минимальными затратами по времени и количеству эпизодов обучения лучше использовать алгоритм Double Deep Q-learning.

ЗАКЛЮЧЕНИЕ

В настоящей работе были рассмотрены 2 алгоритма обучения с подкреплением – Deep Q-learning и Double Deep Q-learning. Указанные алгоритмы были применены для решения задачи управления динамической системы перевернутого маятника, взятой из библиотеки моделей обучения с подкреплением Gym. В результате исследования был сделан вывод, что для наилучшего управления моделью перевернутого маятника при минимальных затратах на время обучения и количество эпизодов обучения лучше использовать алгоритм Double Deep Q-learning.

СПИСОК ЛИТЕРАТУРЫ

1. Playing Atari with deep reinforcement learning / V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. – URL: https://arxiv.org/abs/1312.5602 (accessed: 03.07.2020).

2. Gym: официальный сайт проекта Gym. – URL: http://gym.openai.com/html (дата обращения: 03.07.2020).

3. Перевернутый маятник. – URL: http://www.100byte.ru/python/cartPole/cartPole.html (дата обращения: 03.07.2020).

4. CartPole-v0 // OpenAI Wiki. – URL: https://github.com/openai/gym/wiki/CartPole-v0 (accessed: 03.07.2020).

5. *Barto A.G., Sutton R.S., Anderson C.W.* Neuronlike adaptive elements that can solve difficult learning control problems // IEEE Transactions on Systems, Man, and Cybernetics. – 1983. – Vol. SMC-13, N 5. – P. 834–846. – URL: http://www.derongliu.org/adp/adp-cdrom/Barto1983.pdf (accessed: 03.07.2020).

6. *Hasselt H. van, Guez A., Silver D*. Deep reinforcement learning with Double Q-learning. – URL: https://arxiv.org/abs/1509.06461 (accessed: 03.07.2020).

7. Python: website. – URL: https://www.python.org/ (accessed: 03.07.2020).

8. PyTorch: website. – URL: https://pytorch.org/ (accessed: 03.07.2020).

9. *Нильсон Н.* Искусственный интеллект. – М.: Мир, 1973. – 273 с.

10. *Хайкин С.* Нейронные сети: полный курс: пер. с англ. – 2-е изд. – М.: Вильямс, 2006. – 1104 с.

11. *Kingma D.P., Ba J.L*. Adam: a method for stochastic optimization // International Conference on Learning Representations (ICLR 2015). – Ithaca, NY, 2015. – URL: https://arxiv.org/pdf/1412.6980.pdf (accessed: 03.07.2020).

12. *Sutton R.S., Barto A.G.* Introduction to reinforcement learning. – Cambridge: MIT Press, 1998.

13. Видеокарта MSI GeForce GTX 1050 Ti // MSI: web-сайт. – URL: https://ru.msi.com/Graphics-card/support/GeForce-GTX-1050-Ti-GAMING-4G (дата обращения: 03.07.2020).

14. CUDA Toolkit Documentation v11.0.171 // Developer Zone NVIDIA. – URL: https://docs.nvidia.com/cuda/ (accessed: 03.07.2020).

15. *Evseenko A.A.* Analysis of the applicability of artificial intelligence methods to solving problems of stabilization of dynamic systems / research adviser D.O. Romannikov, language adviser R.A. Chesnokova // Progress through innovations: proceedings 2019 VIIIth International academic and research conference of graduate and postgraduate students, March 28, 2019, Novosibirsk, Russia. – Novosibirsk, 2019. – P. 55–57. – ISBN 978-5-7782-3848-0.

16. *Романников Д.О.* Исследование работы нейронных сетей на примере задачи управления перевернутым маятником // Сборник научных трудов НГТУ. – 2018. – № 1 (91). – С. 95–103.

Евсеенко Алла Александровна, магистрант факультета автоматики и вычислительной техники Новосибирского государственного технического университета. E-mail: vnattali@mail.ru

Романников Дмитрий Олегович, кандидат технических наук, доцент кафедры автоматики Новосибирского государственного технического университета. Основное направление научных исследований – нейронные сети, сети Петри. Имеет более 60 публикаций. E-mail: dmitry.romannikov@gmail.com

DOI: 10.17212/2307-6879-2020-1-2-7-25

Application of Deep Q-learning and Double Deep Q-learning algorithms to the task of control an inverted pendulum*

A.A. Evseenko[1], D.O. Romannikov[2]

[1] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, master's student of the faculty of automation and computing engineering. E-mail: vnattali@mail.ru*

[2] *Novosibirsk State Technical University, 20 Karl Marx Prospekt, Novosibirsk, 630073, Russian Federation, candidate of technical sciences, docent of the automation department. E-mail:dmitry.romannikov@gmail.com*

Today, such a branch of science as «artificial intelligence» is booming in the world. Systems built on the basis of artificial intelligence methods have the ability to perform functions that are traditionally considered the prerogative of man. Artificial intelligence has a wide range of research areas. One such area is machine learning. This article discusses the algorithms of one

* *Received 04 May 2020.*

of the approaches of machine learning – reinforcement learning (RL), according to which a lot of research and development has been carried out over the past seven years. Development and research on this approach is mainly carried out to solve problems in Atari 2600 games or in other similar ones. In this article, reinforcement training will be applied to one of the dynamic objects – an inverted pendulum. As a model of this object, we consider a model of an inverted pendulum on a cart taken from the gym library, which contains many models that are used to test and analyze reinforcement learning algorithms. The article describes the implementation and study of two algorithms from this approach, Deep Q-learning and Double Deep Q-learning. As a result, training, testing and training time graphs for each algorithm are presented, on the basis of which it is concluded that it is desirable to use the Double Deep Q-learning algorithm, because the training time is approximately 2 minutes and provides the best control for the model of an inverted pendulum on a cart.

Keywords: neural networks, artificial intelligence, inverted pendulum model, python, gym, pytorch, Deep Q-learning (DQN), Double Deep Q-learning (DDQN), reinforcement learning (RL)

REFERENCES

1. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. *Playing Atari with deep reinforcement learning*. Available at: https://arxiv.org/abs/1312.5602 (accessed 03.07.2020).

2. *Gym: official site of the project Gym*. Available at: http://gym.openai.com/html (accessed 03.07.2020).

3. *Perevernutyi mayatnik* [Inverted pendulum]. Available at: http://www.100byte.ru/python/cartPole/cartPole.html (accessed 03.07.2020).

4. CartPole-v0. *OpenAI Wiki*. Available at: https://github.com/openai/gym/wiki/CartPole-v0 (accessed 03.07.2020).

5. Barto A.G., Sutton R.S., Anderson C.W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1983, vol. SMC-13, no. 5, pp. 834–846. Available at: http://www.derongliu.org/adp/adp-cdrom/Barto1983.pdf (accessed 03.07.2020).

6. Hasselt H. van, Guez A., Silver D. *Deep reinforcement learning with Double Q-learning*. Available at: https://arxiv.org/abs/1509.06461 (accessed 03.07.2020).

7. *Python: website*. Available at: https://www.python.org/ (accessed 03.07.2020).

8. *PyTorch: website*. Available at: https://pytorch.org/ (accessed 03.07.2020).

9. Nilsson N. *Problem-solving methods in artificial intelligence*. New York, McGraw-Hill, 1971 (Russ. ed.: Nil'son N. *Iskusstvennyi intellect*. Moscow, Mir Publ., 1973. 273 p.).

10. Haykin S. *Neural networks: a comprehensive foundation*. 2nd ed. Upper Saddle River, N.J., Prentice Hall, 1999 (Russ. ed.: Khaikin S. *Neironnye seti: polnyi kurs*. 2nd ed. Moscow, Williams Publ., 2006. 1104 p.).

11. Kingma D.P., Ba J.L. Adam: a method for stochastic optimization. *International Conference on Learning Representations (ICLR 2015)*, Ithaca, NY, 2015. Available at: https://arxiv.org/pdf/1412.6980.pdf (accessed 03.07.2020).

12. Sutton R.S., Barto A.G. *Introduction to reinforcement learning*. Cambridge, MIT Press, 1998.

13. Video card MSI GeForce GTX 1050 Ti. *Official site MSI*. (In Russian). Available at: https://ru.msi.com/Graphics-card/support/GeForce-GTX-1050-Ti-GAMING-4G (accessed 03.07.2020).

14. CUDA Toolkit Documentation v11.0.171. *Developer Zone NVIDIA*. Available at: https://docs.nvidia.com/cuda/ (accessed 03.07.2020).

15. Evseenko A.A. Analysis of the applicability of artificial intelligence methods to solving problems of stabilization of dynamic systems. *Progress through innovations: proceedings 2019 VIIIth International academic and research conference of graduate and postgraduate students*, March 28, 2019, Novosibirsk, Russia, pp. 55–57. ISBN 978-5-7782-3848-0.

16. Romannikov D.O. Issledovanie raboty neironnykh setei na primere zadachi upravleniya perevernutym mayatnikom [Investigation of the work of neural networks on the example of the problem of the control of the back panel]. *Sbornik nauchnykh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta = Transaction of scientific papers of the Novosibirsk state technical university*, 2018, no. 1 (91), pp. 95–103.

Для цитирования:

Евсеенко А.А., Романников Д.О. Применение алгоритмов Deep Q-learning и Double Deep Q-learning к задаче управления перевернутым маятником // Сборник научных трудов НГТУ. – 2020 – № 1–2 (97). – С. 7–25. – DOI: 10.17212/2307-6879-2020-1-2-7-25.

For citation:

Evseenko A.A., Romannikov D.O. Primenenie algoritmov Deep Q-learning i Double Deep Q-learning k zadache upravleniya perevernutym mayatnikom [Application of Deep Q-learning and Double Deep Q-learning algorithms to the task of control an inverted pendulum]. *Sbornik nauchnykh trudov Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta = Transaction of scientific papers of the Novosibirsk state technical university*, 2020, no. 1–2 (97), pp. 7–25. DOI: 10.17212/2307-6879-2020-1-2-7-25.