УДК 004.8

Сравнение современных подходов обучения интеллектуальных агентов в среде ViZDoom

Черепанов Ефим Андреевич
Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И. Ульянова (Ленина)
Студент

Аннотация
Статья посвящена сравнению современных подходов обучения интеллектуальных агентов в среде ViZDoom. Рассмотрены алгоритмы DQN и A3C. Описан принцип работы данных алгоритмов, осуществлена их реализация на языке Python для данной среды. Проведён вычислительный эксперимент и выполнен сравнительный анализ данных решений.
Ключевые слова: ViZDoom, DQN, A3C, обучение с подкреплением, интеллектуальные агенты.

Comparison of modern approaches for training intelligent agents in the ViZDoom environment

Cherepanov Efim Andreevich
Saint-Petersburg Electrotechnical University "LETI"
Student

Abstract
This article is devoted to comparison of modern approaches for training intelligent agents in the ViZDoom environment. DQN and A3C algorithms are considered. Working principles of these algorithms are described and implemented in Python for this environment. A computational experiment and comparative analysis of these solutions was carried out.
Keywords: ViZDoom, DQN, A3C, reinforcement learning, intelligent agents.

В современном мире всё более широкое распространение получает искусственный интеллект, который может применяться для решения огромного спектра задач, в том числе задач классификации [1]. Одним из перспективных направлений в области искусственного интеллекта является машинное обучение, и в частности обучение с подкреплением [2]. Обучение с подкреплением – один из способов машинного обучения, в ходе которого испытуемая система (агент) обучается, взаимодействуя с некоторой средой. Откликом среды на принятые решения являются сигналы подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда и её модель. Развитие данного направления является

актуальным, так как данное обучение очень похоже на изучение окружающей среды человеком.

Постановка задачи

Обучение с подкреплением является подразделом машинного обучения, изучающего, как агент должен действовать в окружении, чтобы максимизировать некоторый долговременный выигрыш.

Данное окружение формулируется как марковский процесс принятия решений (МППР) с конечный множеством состояний. Вероятности выигрышей и перехода состояний в МППР обычно являются случайными величинами, но стационарными в рамках задачи.

Марковский процесс принятия решений представляет собой кортеж $(S, A, P.(\cdot,\cdot), R.(\cdot,\cdot), \gamma)$ из пяти составляющих:

- S - конечное множество состояний
- A - конечное множество действий (A_s - конечное множество действий доступных для состояния s)
- $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ - вероятность, что действие a в состоянии s во время t перейдёт в состояние s' ко времени $t + 1$
- $R_a(s, s')$ - вознаграждение, получаемое после перехода в состояние s' из состояния s с вероятностью перехода $P_a(s, s')$
- $\gamma \in [0, 1]$ - коэффициент дисконтирования, который представляет собой разницу влияния будущих и настоящих вознаграждений

В произвольный момент времени t агент характеризуется состоянием s_t и множеством возможных действий A_{s_t}. Выбирая действие a, он перейдёт в состояние s_{t+1} и получит выигрыш r_t. Основываясь на таком взаимодействии с окружающей средой, агент, обучающийся с подкреплением, должен выработать стратегию $\pi : S \to A$, которая максимизирует величину $R = r_0 + r_1 + \cdots + r_n$ в случае МППР, имеющего терминальное состояние, или величину $R = \sum_t \gamma^t r_t$ для МППР без терминальных состояний.

Описание алгоритмов

В данной статье рассмотрены следующие алгоритмы:

- DQN (Deep Q-Network)
- A3C (Asynchronous Advantage Actor-Critic)

Алгоритм DQN основывается на понятии cumulative return over time – это общее количество наград, которое мы можем получить от текущего момента времени до конца игры. Оптимальное поведение данной величины можно описать функцией $Q^*(s, a)$, которая возвращает одно число. Данная функция позволяет оценить, какой будет cumulative reward до конца игры, если мы находимся в состоянии s и предпринимаем действие a. Вычисляем значение функции для всех a и выбираем максимальный вариант. В рамках

алгоритма DQN для вычисления данной функции используется рекуррентное уравнение Bellman Equation:

$$Q^*(s,a) = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \quad (1)$$

Данное выражение означает, что если известно с какой вероятностью агент в состоянии s переходит в следующее состояние s' выполнив действие a, то для текущего состояния функция Q^* должна быть эквивалентна максимуму для возможных действий. Данное уравнение используется как итеративный шаг улучшения функции.

В алгоритме DQN с целью улучшения сходимости применяется метод experience replay. Основная идея этого метода в том, что в памяти сохраняются N предыдущих переходов, а при обучении используется случайная выборка из данных переходов.

Ещё одним улучшением является использование ε-greedy policy. Данная стратегия заключается в том, что вместо выбора наиболее выгодного действия агент с вероятностью ε выбирает случайное действие.

Таким образом, данная нейронная сеть представляет собой CNN (convolutional neural network), где входными данными являются пиксели текущей картинки игры, а выходными – значения функции $Q(s,a)$ для каждого возможного действия a.

Впервые данный алгоритм был представлен в статье Human-level control through deep reinforcement learning [3]. Псевдокод алгоритма представлен на рисунке 1.

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M do
  Initialize sequence s_1 = {x_1} and preprocessed sequenced φ_1 = φ(s_1)
  for t = 1, T do
    With probability ε select a random action a_t
    otherwise select a_t = max_a Q*(φ(s_t), a; θ)
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set s_{t+1} = s_t, a_t, x_{t+1} and preprocess φ_{t+1} = φ(s_{t+1})
    Store transition (φ_t, a_t, r_t, φ_{t+1}) in D
    Sample random minibatch of transitions (φ_j, a_j, r_j, φ_{j+1}) from D
    Set { r_j                                  for terminal φ_{j+1}
        { r_j + γ max_{a'} Q(φ_{j+1}, a'; θ)   for non − terminal φ_{j+1}
    Perform a gradient descent step on (y_j − Q(φ_j, a_j; θ))^2
    according to equation 1
  end for
end for
```

Рисунок 1 – Псевдокод алгоритма DQN

A3C является policy gradient алгоритмом, который полагается на оптимизацию параметров стратегии в соответствии с ожидаемой суммарной наградой, используя для этого метод градиентного спуска. Алгоритмы типа

actor-critic оперируют как стратегией (actor), так и функцией полезности (critic). Для расчёта градиента используется функция полезности, вычисляемая во время перехода в новое состояние по методу temporal difference при помощи уравнения Беллмана (1).

Впервые данный алгоритм был описан в статье Asynchronous Methods for Deep Reinforcement Learning [4]. В представленном алгоритме используется несколько копий нейронной сети и несколько экземпляров среды, где каждый агент взаимодействует со своей копией среды. Во время обучения результаты, полученные интеллектуальными агентами, объединяются в глобальную модель.

Псевдокод алгоритма представлен на рисунке 2.

```
// Assume global shared parameter vectors θ and θ_v
// and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
  Reset gradients: dθ ← 0 and dθ_v ← 0
  Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
  t_start = t
  Get state s_t
  repeat
    Perform a_t according to policy π(a_t|s_t; θ')
    Receive reward r_t and new state s_{t+1}
    t ← t + 1
    T ← T + 1
  until terminal s_t or t − t_start == t_max
  R = { 0                for terminal s_t
      { V(s_t, θ'_v)     for non − terminal s_t  // Bootstrap from last state
  for i ∈ {t − 1, ..., t_start} do
    R ← r_i + γR
    Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
    Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2/∂θ'_v
  end for
  Perform asynchronous update of θ using dθ and of θ_v using dθ_v
until T > T_max
```

Рисунок 2 – Псевдокод алгоритма A3C

Вычислительный эксперимент

Существует множество различных средств моделирования, с помощью которых могут сравниваться алгоритмы машинного обучения, но все они требуют детального математического описания не только агентов, но и всех элементов среды, в которой они существуют, например [5].

Наиболее популярным инструментом для сравнения алгоритмов является платформа OpenAI Gym. Преимущество данной платформы в том, что она содержит набор различных сред и позволяет решать самые разнообразные задачи, например [6].

Но, несмотря на преимущества платформы OpenAI Gym, платформа ViZDoom является более гибко настраиваемой и удобной для разработки и эффективного сравнения алгоритмов для среды Doom, поэтому для эксперимента использовалась именно эта платформа.

ViZDoom – это исследовательская платформа, основанная на игре Doom, для изучения обучения с подкреплением из необработанной графической информации [7]. Платформа позволяет разрабатывать интеллектуальных агентов, которые играют в Doom, используя графическую информацию с экрана.

Сценарий эксперимента следующий. В закрытой комнате появляется агент и неподалёку перед ним – монстр. У агента имеется оружие (пистолет). Каждый эпизод монстр появляется в случайном месте. Он неподвижен и погибает с одного выстрела. Задача агента попасть выстрелом в монстра. Игроку доступно 3 действия: шаг влево, шаг вправо и выстрел. Награды: 100 очков за попадание в монстра, -1 за каждый тик времени, -6 за промах при выстреле. На рисунке 3 изображено визуальное представление данного сценария.

Рисунок 3 – Сценарий эксперимента

Несмотря на то, что разработкой игры или агентов сегодня можно заниматься на практически любом языке программирования, например [8], самым популярным языком в области разработки интеллектуальных агентов является Python. Поэтому алгоритмы реализованы на языке Python с

использованием фреймворка Tensorflow [9]. В рамках эксперимента использовалась вычислительная мощность процессора Intel Core i3-4005U.

Результат эксперимента приведён в таблицах 1 и 2, где

- mean – среднее арифметическое значение награды
- std – стандартное отклонение значения награды
- min – минимальное значение награды
- max – максимальное значение награды

Таблица 1 – Эксперимент с DQN

Время	**Количество эпизодов**	**Количество шагов**	**mean**	**std**	**min**	**max**
00:03:59	255	5000	-197.32	200.68	-390.00	95.00
00:08:02	501	10000	-231.70	182.68	-390.00	95.00
00:12:08	696	15000	-261.38	176.10	-400.00	95.00
00:16:14	868	20000	-328.41	58.74	-395.00	95.00
00:20:21	1091	25000	-3.29	116.96	-380.00	95.00
00:24:36	2388	30000	69.53	23.78	-380.00	95.00
00:28:52	3986	35000	70.08	18.82	-75.00	95.00
00:33:08	5602	40000	73.51	17.73	-75.00	95.00
00:37:25	7279	45000	74.01	18.20	-100.00	95.00
00:41:42	8831	50000	75.73	15.94	-86.00	95.00

Количество агентов в эксперименте для алгоритма A3C – 3.

Таблица 2 – Эксперимент с A3C

Время	**Количество эпизодов**	**Количество шагов**	**mean**	**std**	**min**	**max**
00:03:32	1752	25000	-190.73	236.21	-410.00	95.00
00:06:49	3118	50000	-190.82	242.26	-410.00	95.00
00:10:08	4532	75000	-193.63	241.35	-410.00	95.00
00:13:31	5911	100000	-215.51	237.36	-410.00	95.00
00:16:58	7782	125000	16.86	120.14	-410.00	95.00
00:20:35	10123	150000	-186.36	183.62	-410.00	95.00
00:25:23	15832	175000	71.11	28.93	-385.00	95.00
00:31:08	24432	200000	76.71	15.97	-315.00	95.00
00:36:53	33392	225000	76.11	23.65	-345.00	95.00
00:42:38	42463	250000	77.02	19.01	-345.00	95.00

После обучения произведено тестирование обученных агентов на 30 игровых эпизодах. Результаты представлены в таблице 3.

Таблица 3 – Результаты теста обученных агентов

Алгоритм	mean	min	max
DQN	75.23	21.00	95.00
A3C	83.33	45.00	95.00

Заключение

По результатам эксперимента можно сделать вывод, что алгоритм DQN показывает более стабильные результаты на протяжении всего времени обучения и требует меньшее количество шагов обучения. В свою очередь алгоритм A3C показывает лучшие результаты на более длинной дистанции за счёт возможности асинхронного обучения с помощью нескольких агентов параллельно, но требует больших вычислительных мощностей.

Из данного эксперимента следует, что алгоритм A3C является более эффективным при возможности асинхронного обучения с помощью нескольких агентов одновременно. Если такой возможности нет, то алгоритм DQN является гораздо более эффективным.

Библиографический список

1. Беляев С.А., Гордеева Т.В. Применение методов классификации для анализа визитных карточек в мобильном телефоне // Электронный журнал: Программные продукты, системы и алгоритмы. 2018. №1. С.20-25. DOI: 10.15827/2311-6749.26.298.
2. Reinforcement learning // Wikipedia. URL: https://en.wikipedia.org/wiki/Reinforcement_learning (дата обращения: 10.03.2018).
3. Mnih V. Human-level control through deep reinforcement learning // Nature, 2015. 529-533 с.
4. Mnih V. Asynchronous Methods for Deep Reinforcement Learning // ICML, 2016.
5. Беляев С.А., Матросов В.В. Опыт создания среды имитационного моделирования // III Всероссийская научно-практическая конференция «Теоретические и прикладные проблемы развития и совершенствования автоматизированных систем управления военного назначения» (22.11.2017 г., СПб). Сборник тезисов / СПб.: Военно-космическая академия имени А.Ф. Можайского. 2017. С.232.
6. Беляев С.А., Михнович А.Г. Современные подходы к решению задачи стабилизации перевернутого маятника // Электронный журнал: Программные продукты, системы и алгоритмы. 2017. №2. DOI: 10.15827/2311-6749.23.237.
7. ViZDoom // ViZDoom. URL: https://www.vizdoom.cs.put.edu.pl (дата обращения: 10.03.2018).
8. Беляев С.А. Разработка игр на языке JavaScript. Учебное пособие. СПб.: Изд-во Лань, 2016. 128 с..

9. TensorFlow library // TensorFlow. URL: https://www.tensorflow.org/ (дата обращения: 10.03.2018).
10. Millington I. Artificial Intelligence for Games. Elsevier, 2006. 563-641 с.