УДК 004.02
DOI: 10.15827/0236-235X.122.284-290

Дата подачи статьи: 09.02.18
2018. Т. 31. № 2. С. 284–290

СОВРЕМЕННЫЕ ПОДХОДЫ К ОБУЧЕНИЮ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ В СРЕДЕ Atari

Д.А. Коробов ¹, студент, mroyalruby@yandex.ru
С.А. Беляев ¹, к.т.н., доцент, bserge@bk.ru

¹ Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина), ул. Профессора Попова, 5, г. Санкт-Петербург, 197376, Россия

Статья посвящена разработке и исследованию алгоритмов машинного обучения в задаче обучения интеллектуальных агентов в среде Atari, представляющей эмуляцию игровой консоли Atari2600 при помощи платформы OpenAI Gym. Основные цели исследования – описание и сравнение современных алгоритмов обучения с подкреплением в различных ситуациях, выявление их достоинств и недостатков, а также формирование предложений по повышению эффективности этих алгоритмов.

Авторы описывают математическую модель задачи обучения с подкреплением в виде марковского процесса принятия решений и формируют критерий оптимальности системы. Отдельно отмечены особенности, характерные для среды Atari.

Описаны принцип работы существующих решений, а также используемые в них инструменты. Рассмотрены базовый подход Q-обучения и его известная модификация с использованием нейронной сети. Представлена идея алгоритма на основе градиента по стратегиям. Дано описание подхода алгоритмов типа «актор-критик», а также рассмотрена его асинхронная многопоточная версия. Все рассмотренные алгоритмы описаны в виде псевдокода.

Предложен способ повышения эффективности обучения с помощью выделения признаков. Разработан алгоритм генерации вектора признаков на основе распознавания образов. Описаны этапы формирования вектора признаков и его использования в процессе обучения.

Алгоритмы реализованы, проведен эксперимент с их использованием. Выполнен сравнительный анализ результатов и получены выводы об эффективности алгоритмов. Предложены идеи по дальнейшему увеличению скорости и качества обучения интеллектуальных агентов.

Ключевые слова: Atari, интеллектуальные агенты, обучение с подкреплением, Q-обучение, градиент по стратегиям, асинхронный актор-критик, распознавание образов, выделение признаков.

Интеллектуальными агентами принято называть самообучающиеся программы, способные играть в классические и видеоигры. Согласно [1], «агентом является все, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов». Интеллектуальные агенты способны моделировать поведение человека в задачах сбора и обработки информации, при этом они могут активно перемещаться внутри информационного пространства в направлении цели.

Одной из платформ, позволяющих проводить разработку и исследование алгоритмов машинного обучения, является OpenAI Gym, включающая в себя возможность эмуляции игровой консоли Atari2600. В качестве входных данных среда Atari предоставляет последовательность изображений экрана размера 210×160 пикселей, а также общий игровой счет и индикатор победы или поражения. На основе описанных входных данных агент должен выбрать одно из возможных действий с помощью обучения с подкреплением – машинное обучение, направленное на решение задачи принятия решения о выборе действия в заданной ситуации таким образом, чтобы максимизировать выгоду в долгосрочной перспективе [2]. В частности, интерес представляют интеллектуальные агенты, не зависящие от правил игры, то есть способные проходить несколько различных игр Atari без изменения

алгоритма. В дальнейшем разработанные алгоритмы могут быть применены в робототехнике при движении роботов-гуманоидов, в проектировании человеко-машинных интерфейсов, а также в системах алгоритмической торговли.

Формальная постановка задачи

Стандартным способом формализации задачи обучения с подкреплением является представление ее в виде *марковского процесса принятия решений* (МППР) [3]. МППР представляет способ редукции задачи принятия решения к следующим ключевым компонентам: множеству состояний S , множеству действий A , функции перехода $P : S \times A \rightarrow \rightarrow Dist(S)$, функции награды $P : S \times A \rightarrow \mathbb{R}$ и фактору дисконтирования γ , где $Dist(S)$ – пространство всех распределений вероятности на множестве X . В среде Atari предполагается, что S и A – конечные множества. На каждом шаге t агент исследует состояние $s_t \in S$ и выбирает действие $a_t \in A$. Далее агент получает награду $r_t := R(s_t, a_t)$ и новое состояние s_{t+1} из распределения $P(\cdot | s_t, a_t)$. В МППР переходы удовлетворяют свойству марковости: для любых s_{t+1} и для любой истории состояний и действий $s_0, a_0, \dots, s_t, a_t$ выполняется следующее равенство:

$$\begin{aligned} \Pr(s_{t+1} | s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t, a_t) = \\ = \Pr(s_{t+1} | s_t, a_t) := P(s_{t+1} | s_t, a_t). \end{aligned} \quad (1)$$

Результат действия зависит только от текущего состояния и не зависит от предыдущих состояний. Поведение агента описывается стратегией $\pi : S \rightarrow \rightarrow \text{Dist}(A)$, которая отображает множество состояний в распределение вероятностей над множеством действий. Под стратегией понимаются весовые коэффициенты модели, корректируемые агентом в процессе обучения.

Важным является то, что в среде Atari отдаленные награды менее значимы, чем ближайшие. Фактор дисконтирования γ может иметь значение 0 или 1: 0 означает, что только настоящее имеет значение, а 1 – что будущие награды имеют полный вес в принятии решения:

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) = r_t + \gamma R_{t+1}.$$

Цель агента – найти оптимальный план – стратегию, максимизирующую ожидаемую полезность из любого состояния s_t , определяемую как математическое ожидание суммы дисконтированных будущих наград: $\mathbb{E} \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t \right]$, где последователь-

ность будущих состояний $\{s_{t+1}, s_{t+2}, \dots\}$ индуцируется функцией перехода P и стратегией π .

Существующие решения

Классические алгоритмы обучения с подкреплением на каждом этапе оценивают модель состояния будущего полностью, присваивают ценность каждому действию и затем выбирают действие с наивысшим значением ценности. Впоследствии появились модели, использующие новые методы принятия решений. Нейронные сети, составленные из нескольких слоев линейных моделей, позволяют строить абстракции в виде иерархии признаков, а сверточные нейронные сети являются наиболее мощным инструментом для извлечения информации из изображений [4]. В данной статье рассмотрены следующие современные алгоритмы, использующие нейронные сети:

- Deep Q-learning (глубокое Q-обучение);
- REINFORCE (алгоритм на основе градиентов по стратегиям);
- АЗС (алгоритм асинхронного актора-критика).

Алгоритмы Q-обучения оперируют функцией качества $Q(s, a)$, обозначающей наибольший возможный счет, которого можно достичь в конце игры после выбора действия a в состоянии s . Функцию $Q(s, a)$ можно определить рекурсивно:

$$\begin{aligned} Q^*(s, a) &= R(s, a) = \\ &= \gamma \sum_{s' \in S} T(s, a, s') * \max_{a'} Q^*(s', a'), \end{aligned} \quad (2)$$

где $T(s, a, s')$ – вероятность перехода в состояние s' из состояния s в случае выбора действия a . Однако, так как функции $R(s, a)$ и $T(s, a, s')$ априори неизвестны, требуется оценивать $Q^*(s, a)$ и обновлять

оценку по мере исследования игры в соответствии с различными действиями a в различных состояниях s , учитывая при этом награду r и следующее состояние s' . Правило итеративного обновления оценки Q выглядит следующим образом:

$$\begin{aligned} Q(s, a) &= Q(s, a) + \\ &+ \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right], \end{aligned} \quad (3)$$

где γ – фактор дисконтирования; α – скорость обучения, определяющая степень значимости новых наблюдений при обновлении оценки. При $\alpha = 1$, например, предыдущие оценки не учитываются.

В начале обучения оценка $\max_{a'} Q(s', a')$ может быть далека от истинного значения, но по мере обновления $Q(s, a)$ гарантируется сходимость к истинной функции [5]. Основная идея алгоритма Q-обучения выглядит следующим образом:

Инициализировать $Q(s, a)$ произвольным образом
Получить начальное состояние s
для каждого эпизода:

Выбрать и выполнить действие a
Получить награду r и новое состояние s'
 $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
 $s = s'$

В случае, когда среда имеет дискретное пространство состояний, а агент дискретное количество возможных действий, модель динамики среды представляет собой одношаговую матрицу перехода: $T(s(t+1) \mid s(t), s(a))$. Данная стохастическая матрица содержит все вероятности перехода в желаемое состояние из текущего при выборе каждого действия. В среде Atari пространство состояний после масштабирования без потери информации является множеством изображений размера 64×64 пикселя, а агент имеет 18 возможных действий, значит, размер матрицы перехода вычисляется следующим образом: $|S \times S \times A| \approx |(66.7 \times 10^9) \times (66.7 \times 10^9) \times 18|$, что при 32-битном размере элемента матрицы требует 3.4×10^{14} Гб памяти. Из-за невозможности хранить такой объем памяти требуется аппроксимировать таблицу переходов.

Алгоритм Deep Q-learning (DQN), описанный впервые в [6], использует нейронную сеть для аппроксимации функции Q . Теперь вместо того, чтобы обновлять функцию Q напрямую, нейронной сети передается состояние s , после чего сеть возвращает значения Q для каждого возможного действия.

Одной из особенностей использования нейронных сетей в обучении с подкреплением является отсутствие гарантий сходимости. Для улучшения сходимости применяется метод сохранения предыдущего опыта в памяти (experience replay memory). Вместо обучения сети на каждом переходе в новое состояние и отбрасывания этого перехода в памяти сохраняются N предыдущих переходов, а при обучении используется случайная выборка из данных

переходов [7]. При выборе действия теперь также используется ϵ -жадная стратегия, которая заключается в том, что вместо выбора наиболее выгодного действия агент с вероятностью ϵ выбирает случайное действие. В итоге с учетом рассмотренных замечаний алгоритм DQN выглядит следующим образом:

```

Инициализировать память переходов  $D$ 
Инициализировать модель вычисления  $Q(s, a)$ 
                                         случайными весами
Получить начальное состояние  $s$ 
для каждого эпизода:
    Выбрать и выполнить действие  $a$ : {
        с вероятностью  $\epsilon$  выбрать случайное
                                         действие
        иначе выбрать  $a = \arg \max_{a'} Q(s, a')$ 
    }
    Получить награду  $r$  и новое состояние  $s'$ 
    Сохранить переход  $\langle s, a, r, s' \rangle$  в памяти  $D$ 
    Получить выборку переходов  $\langle ss, aa, rr, ss' \rangle$ 
                                         из памяти
    Вычислить  $tt$  для каждого элемента выборки {
        если  $ss'$  – терминальное состояние,
                                         то  $tt = rr$ 
        иначе  $tt = rr + \gamma \max_{a'} Q(ss', aa')$ 
    }
    Обучить сеть, используя  $(tt - Q(ss, aa))^2$ 
                                         в качестве функции потерь
     $s = s'$ 

```

Алгоритмы на основе градиентов по стратегиям (policy gradient) представляют собой тип алгоритмов обучения с подкреплением, которые полагаются на оптимизацию параметров стратегии в соответствии с ожидаемой суммарной наградой, используя для этого градиентный спуск [8]. Классическим алгоритмом данного типа является REINFORCE, особенность которого в том, что в процессе обновления стратегии (корректировки весов модели) он учитывает всю траекторию агента. Под траекторией понимается последовательность всех переходов – четверок объектов $S \times A \rightarrow S \times R$ (состояние, действие, новое состояние, награда) на протяжении периода одной игры. Задачей алгоритма является максимизация ожидаемой суммарной награды:

$$\rho(s) = \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \mid s \right]. \quad (4)$$

Пусть $R(\tau)$ – математическое ожидание суммарной награды для траектории τ . Вероятность каждой траектории задается стратегией, определяющей действия, которые предпринимает агент. Следовательно, формулу (4) можно переписать следующим образом:

$$\rho(s) = \sum_{\tau} P_{\theta}(\tau) * R(\tau), \quad (5)$$

где θ определяет параметры обучаемой модели. С целью максимизации функции $\rho(s)$ вычисляется

градиент ρ по θ , который после преобразований, описанных в [3], представляется в виде

$$\frac{\partial \rho(s)}{\partial \theta} = \mathbb{E}_{\tau} \left[R(\tau) * \frac{\partial \log(P_{\theta}(\tau))}{\partial \theta} \right]. \quad (6)$$

Градиент $\nabla_{\theta}(P_{\theta}(\tau)) = \frac{\partial \log(P_{\theta}(\tau))}{\partial \theta}$ – это вектор,

направленный в сторону возрастания функции $\rho(s)$. Аппроксимируя математическое ожидание $\mathbb{E}_{\tau}$ средним арифметическим по T переходам в рамках одного эпизода (игры), получаем правило обновления вектора коэффициентов θ , максимизирующее функцию суммарной награды:

$$\theta_{t+1} = \theta_t + \alpha * R(\tau) * \nabla_{\theta}(P_{\theta}(\tau)). \quad (7)$$

Псевдокод алгоритма будет следующим:

```

Вход: модель, задающая стратегию
                                          $\pi(a \mid s, \theta), \forall a \in A, s \in S$ 
Инициализировать веса модели  $\theta$ 
повторять бесконечно:
    Сгенерировать эпизод  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ ,
                                         следуя стратегии  $\pi$ 
    для каждого шага  $t=0, \dots, T-1$ :
         $R_t \leftarrow$  вычислить
         $\theta \leftarrow \theta + \alpha * R_t * \nabla_{\theta}(P_{\theta}(t))$ 

```

Алгоритмы типа «актор-критик» оперируют как стратегией (актор), так и функцией полезности (критик), формируя стратегию подобно алгоритму REINFORCE. Отличие заключается в том, что для расчета градиента используется функция полезности, а не траектория. Во время перехода в новое состояние система получает сигнал о значении награды и вычисляет функцию полезности по методу временной разности (temporal difference) при помощи уравнения Беллмана [9]. Функция полезности затем используется для корректировки весов модели.

В статье [10], описывающей асинхронные методы для глубокого обучения с подкреплением, был представлен алгоритм асинхронного актора-критика (Asynchronous Advantage Actor-Critic – A3C), использующий несколько копий нейронной сети и несколько экземпляров среды, где каждый агент взаимодействует со своей собственной копией среды. Результаты, полученные несколькими агентами, в дальнейшем объединяются в глобальную модель. Рассматриваемая версия алгоритма использует функцию преимущества A , которая оценивается, используя дисконтированную сумму будущих наград R : $A = R - V(s)$, где $V(s)$ – функция, обозначающая наибольший счет, которого можно достичь из состояния s , и не зависящая от действий агента. Псевдокод алгоритма A3C:

```

Задать глобальные общие векторы параметров  $\theta$ 
                                         и  $\theta_v$  и глобальный общий счётчик
 $T = 0$ 
Задать векторы параметров  $\theta'$  и  $\theta_v'$  для каждого
                                         потока

```

Инициализировать счётчик шагов потока $t \leftarrow 1$
повторять:

Обновить градиенты: $d\theta \leftarrow 0$ и $d\theta_v \leftarrow 0$

Синхронизировать: $\theta' = \theta$ и $\theta'_v = \theta_v$

$t_{start} = t$

Получить состояние s_t

повторять:

Выполнить действие a_t согласно

стратегии $\pi(a_t | s_t, \theta')$

Получить награду r_t и новое состояние s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

до тех пор пока s_t терминал или $t - t_{start} = t_{max}$

$R = \begin{cases} 0, & \text{для терминала } s_t \\ V(s_t, \theta'_v), & \text{для нетерминала } s_t \end{cases}$

для $i \in \{t-1, \dots, t_{start}\}$:

$R \leftarrow r_i + \gamma * R$

$d\theta \leftarrow d\theta + \nabla_{\theta} \log \pi(a_i | s_i, \theta') (R - V(s_i, \theta'))$

$d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i, \theta'_v))^2 / \partial \theta'_v$

Выполнить асинхронное обновление θ и θ_v

пока $T \leq T_{max}$

В качестве модели в рассмотренных алгоритмах предлагается использовать сверточную нейронную сеть. Значительными ограничениями применимости данных алгоритмов являются огромная размерность входных данных и ресурсоемкость задачи обучения при использовании пикселей в качестве входных данных, за счет чего обучение даже на высокопроизводительном компьютере может занимать десятки часов.

Редукция задачи посредством распознавания образов

Увеличить скорость и качество обучения может формирование информативных признаков [11]. Использование вектора признаков вместо набора пикселей позволит интеллектуальному агенту ориентироваться лишь на значимые характеристики входных данных и, следовательно, быстрее обу-

читься. Авторами предлагается вариант алгоритма генерации вектора признаков, основанный на обнаружении набора классов игровых существностей (персонажей и окружения) и нахождении положения экземпляров данных классов на каждом изображении экрана. В его основу легли следующие наблюдения: большинство игр Atari имеют статический фон и небольшое число игровых существностей, перемещающихся по экрану. Все игровые существности могут быть разделены на небольшое число непересекающихся классов. При этом задача разбивается на два этапа: на первом этапе для новой игры строится вектор признаков, а на втором проводится обучение с его использованием.

1. Предобработка. Включает в себя обнаружение фона и выделение сгущений – групп связанных и совместно движущихся пикселей. Далее сгущения разделяются на множество классов C . Под классом объектов понимается множество всех возможных форм (фигур), задающих игровую существность, где формы представляют различные кадры в анимации объекта. Классы могут содержать обрезанные вариации формы, например, при частичном выходе объекта за пределы экрана. Пример результатов различных этапов предобработки изображен на рисунке.

2. Обучение. Как только агент начинает действовать, на каждом шаге распознаются объекты классов $c \in C$, с их помощью происходит генерация вектора признаков.

Метод распознавания объектов классов использует классы, полученные на предыдущем этапе. Формально ставится задача нахождения списка $\Psi(c)$, содержащего координаты и скорости всех экземпляров класса c на экране X_t для каждого класса $c \in C$. Сначала строится список всех возможных фигур всех классов, который затем упорядочивается по размеру от самой большой фигуры к самой маленькой. Далее изображение экрана сканируется на предмет наличия каждой фигуры o из списка. Если совпадение найдено, пиксели помечаются как

распознанные и в дальнейшем не учитываются. Центр масс распознанных пикселей добавляется в список $\Psi(o.class)$.

За распознаванием объектов классов следует этап генерации вектора признаков для обучения модели. В результате вектор признаков содержит следующие данные:

- количество объектов каждого класса (в зависимости от наличия или отсутствия определенных объектов действия агента могут различаться);
- абсолютное положение каждого объекта (координаты объектов – основная информация, необходимая для построения эффективной стратегии);
- векторы скорости объектов (направление движения играет значительную роль при принятии решения о выборе следующего действия; скорость определяется разностью координат объектов ψ_i и ψ_{i-1}).

На каждом экране класс c может иметь от 0 до $n_c(t)$ экземпляров, где $n_c(t)$ не является постоянным и зависит от этапа игры. Однако все алгоритмы обучения с подкреплением требуют на входе вектор фиксированной длины. Следовательно, необходимо найти способ эффективной генерации вектора фиксированной длины из распознанных объектов.

Пусть $N_c = \max_i n_c(t)$ – максимально возможное число объектов класса c , которые встретились в последовательности изображений экрана $[X_1, X_2, \dots, X_n]$. Тогда для нахождения числа N_c для каждого класса $c \in C$ необходимо на этапе предобработки сохранять и максимальное количество встретившихся объектов каждого класса.

При использовании данного подхода вектор признаков будет иметь размер, зависящий от суммы максимального количества объектов каждого класса. Помимо количества объектов n класса c , в векторе признаков содержатся $2 \cdot n$ координат (x и y) и $2 \cdot n$ скоростей (v_x и v_y). Значит, итоговый размер вектора будет следующим:

$$N = \sum_c (1 + 2 \cdot N_c + 2 \cdot N_c) = size(C) + 4 \cdot \sum_c N_c \quad (8)$$

Псевдокод общего алгоритма генерации вектора признаков на основе распознавания образов:

Пусть заданы C – множество классов и N_c – максимальное возможное количество экземпляров класса c для данной игры

Входные данные: X_i – изображение экрана

Выходные данные: V_i – вектор признаков

1. $\Psi_i \leftarrow \text{распознать_объекты_класса}(X_i)$

2. вычислить скорости (Ψ_i, Ψ_{i-1})

3. для $c \in C$:

$n \leftarrow \text{количество непустых объектов класса } \Psi_i(c)$

Добавить n в вектор V_i

для $\psi \in \Psi_i(c)$:

если ψ – не пустой:

Добавить $[\psi.x, \psi.x, \psi.v_x, \psi.v_y]$ в вектор V_i

иначе:

Добавить $[null, null, null, null]$
в вектор V_i

OpenAI Gym.

Сравнительная характеристика алгоритмов

Эксперимент проводился при помощи открытой платформы машинного обучения OpenAI Gym. Реализованные на языке Python с использованием фреймворка Tensorflow алгоритмы исследовались в следующих играх Atari: Pong, Breakout, SpaceInvaders, MsPacman. В качестве оборудования использовалась виртуальная машина Google Cloud, имеющая 4 Гб оперативной памяти и один виртуальный CPU, где под виртуальным CPU понимается вычислительная мощность, эквивалентная мощности одного ядра процессора Intel Xeon E5 v3 (Haswell) – 2.3 GHz.

Для проведения оценки результатов введем следующие понятия. Пусть эпизод – одна тренировочная игра от начала партии до перезагрузки (победы или поражения). Принятие решения о выборе действия a в состоянии s с переходом в состояние s' назовем итерацией. Шагом обучения будем называть процесс обучения сети, то есть пересчет весов модели, который может выполняться либо каждую итерацию (как в REINFORCE), либо через каждые k итераций.

Первый эксперимент проводился в среде Pong. Модели обучались на вычислительном сервере в течение 32 часов. Оценка результатов проводилась с помощью 100 тестовых игр, затем вычислялись средний и максимальный игровой счет в выборке. Также оценивались следующие параметры: количество эпизодов и количество шагов обучения (табл. 1).

Таблица 1

Эксперимент 1 – pong

Table 1

Experiment 1 – pong

Алгоритм	Время обучения (часов)	Количество эпизодов	Количество шагов обучения	Средний счет	Максимальный счет
REINFORCE	2	1476	348000	-14.8	-11
	8	4792	1329000	-10.2	-7
	32	11035	4782000	-11.8	-4
DQN	2	167	48000	-18.1	-17
	8	598	186000	-13.3	-7
	32	1490	686000	-11.2	-4
A3C	2	813	4300	-16.9	-11
	8	2075	12000	-13.2	-10
	32	7359	45000	-7.8	-2

Алгоритм A3C требует наименьшего числа шагов обучения на эпизод игры (6 шагов/эпизод) по сравнению с REINFORCE (756 шагов/эпизод) и DQN (460 шагов/эпизод), что позволяет сократить

количество обновлений модели. Тем не менее, для алгоритма DQN требуется наименьшее число эпизодов по сравнению с другими алгоритмами при сравнимом конечном результате (максимальном счете). Учитывая, что производительность разработанного метода формирования вектора признаков пропорциональна количеству эпизодов, было принято решение применить алгоритм распознавания образов к алгоритму DQN.

Задачей второго эксперимента было исследование возможностей использования DQN в различных играх. Для каждой игры представлены три результата: до начала обучения (стратегия случайного выбора действия), а также счет после 32 часов обучения на пикселях и в результате применения метода формирования вектора признаков (табл. 2). В первом случае в классическом варианте DQN используется сверточная нейронная сеть следующего вида: слой с 32 фильтрами размера 8×8 с шагом 4, слой с 64 фильтрами размера 4×4 с шагом 2, слой с 64 фильтрами размера 3×3 с шагом 1, полносвязный слой из 512 нейронов. Каждый из слоев также содержит ReLU-блок. В варианте, включающем процедуру распознавания образов, полученный вектор признаков проходит обучение лишь на полносвязном слое, содержащем 512 нейронов и ReLU-блок, минуя при этом сверточные слои.

Таблица 2

Эксперимент 2 – исследование DQN

Table 2

Experiment 2 – research on DQN

Игра	Стратегия случайного выбора действия		Обучение с использованием сверточной нейронной сети		Обучение с использованием вектора признаков	
	A	B	A	B	A	B
Pong	-20.2	-19	-11.2	-4	-9.1	2
Ms-Pacman	216	420	562	1210	798	1730
Space-Invaders	207.5	290	296	560	345	735
Break-out	1.3	3	3.2	7	6.3	12

Примечание: в таблице A – средний счет, B – максимальный счет.

По результатам эксперимента можно сделать вывод, что алгоритм DQN показывает хорошие результаты в игре MsPacman. Вероятно, это связано с тем, что в данной игре агенту довольно просто принять решение, ориентируясь на отдельное состояние экрана. Неплохие результаты обучения в игре Pong связаны с достаточно простыми правилами игры и небольшим числом сущностей (3 объекта). В таких динамичных играх, как SpaceInvaders и Breakout, есть сложности в оценке динамики по одному состоянию без использования траектории и расчет Q-функции оказывается не слишком эффек-

тивным. Алгоритм, использующий метод генерации признаков на основе распознавания образов, во всех рассмотренных случаях увеличивает эффективность обучения, позволяя при этом использовать более простую модель нейронной сети.

Следует учесть, что во всех рассмотренных вариантах модель самостоятельно генерировала обучающие выборки на основе предыдущего опыта либо случайного выбора действия. Дальнейшее улучшение эффективности алгоритмов возможно путем совмещения обучения с подкреплением и обучения с учителем, например, предоставляя нейронной сети выборки, сформированные человеком.

Заключение

Задача обучения интеллектуальных агентов является актуальной и имеет достаточно широкую область применения. В данной статье были рассмотрены современные алгоритмы обучения с подкреплением в среде Atari. Предложенный вариант алгоритма генерации вектора признаков на основе распознавания образов позволяет увеличить скорость и качество обучения, производя при этом обработку видеопотока в режиме реального времени.

При помощи рассмотренных алгоритмов обучения с подкреплением можно решить другие важные задачи, стоящие перед искусственным интеллектом, в том числе создание роботов, способных обучаться подобно человеку, передвигаться на ногах, самостоятельно принимать решения и научиться понимать естественный язык.

Литература

1. Рассел С., Норвиг П. Искусственный интеллект. Современный подход. М.: Вильямс, 2007. 1410 с.
2. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. Playing Atari with deep reinforcement learning. NIPS Deep Learning Workshop, 2013. URL: <https://arxiv.org/pdf/1312.5602.pdf> (дата обращения: 08.02.2018).
3. Саттон Р.С., Барто Э.Г. Обучение с подкреплением. М.: Бином, 2017. 400 с.
4. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. М.: ДМК-Пресс, 2017. 652 с.
5. Melo F.S. Convergence of Q-learning: A simple proof. Tech. Rep. 2011. URL: <http://users.isr.ist.utl.pt/~mtjspaan/reading-Group/ProofQlearning.pdf> (дата обращения: 08.02.2018).
6. Mnih V. Human-level control through deep reinforcement learning. Nature, 2015, vol. 518, no. 7540, pp. 529–533.
7. Schaul T., Quan J., Antonoglou I., Silver D. Prioritized experience replay. Proc. ICLR-16. 2016. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/prioritized-replay.pdf (дата обращения: 08.02.2018).
8. Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M. Deterministic policy gradient algorithms. Proc. ICML-14. 2014. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/deterministic-policy-gradients.pdf (дата обращения: 08.02.2018).
9. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. СПб: Питер, 2017. 480 с.
10. Mnih V., Badia A.P., Mirza M., Graves A., Harley T., Lillicrap T.P., Silver D., Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. Proc. ICML-16, 2016. URL: <https://arxiv.org/pdf/1602.01783.pdf> (дата обращения: 08.02.2018).

11. Liu D.R., Li H.L., Wang D. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *Intern. J. Automation and Computing*. 2015, vol. 12, pp. 229–242.

Software & Systems
DOI: 10.15827/0236-235X.122.284-290

Received 09.02.18
2018, vol. 31, no. 2, pp. 284–290

MODERN APPROACHES TO TRAINING INTELLIGENT AGENTS IN THE Atari ENVIRONMENT

D.A. Korobov¹, Student, royalruby@yandex.ru

S.A. Belyaev¹, Ph.D. (Engineering), Associate Professor, bserge@bk.ru

¹ St. Petersburg Electrotechnical University "LETI", Prof. Popov St. 5, St. Petersburg, 197376, Russian Federation

Abstract. The article describes the research and development of machine learning algorithms for the problem of training intelligent agents in the Atari environment. The environment is the game console Atari2600 emulation on OpenAI Gym platform. The main aims of the research are to describe and compare modern learning algorithms with reinforcement in a number of cases, to expose their advantages and drawbacks and to give suggestions in order to increase their effectiveness.

The authors describe a mathematical model of the reinforcement learning task in terms of Markov Decision Process and form a system optimality criterion. In addition, they note the Atari environment properties.

There is a description of an operation principle of existing solutions, as well as the tools used in them. The paper considers a basic Q-learning approach and its well-known modification that uses a neural network. It also presents the idea of the algorithm based on policy gradient. The paper gives a description of the approach of “actor-critic” algorithms and considers its asynchronous multithreaded version. All the described algorithms are presented in pseudocode.

The authors propose a method of increasing the effectiveness of learning using feature extraction. The developed algorithm for feature vector generation is based on image recognition. The paper describes the steps of feature vector selection and its usage during the learning process.

The described algorithms are implemented, and the experiment is conducted. A comparative analysis of the results gave conclusions on the efficiency of the algorithms. Finally, the authors propose some ideas for further increasing the speed and efficiency of training of intelligent agents.

Keywords: Atari, intelligent agents, reinforcement learning, Q-learning, policy gradient, asynchronous actor-critic, pattern recognition, feature extraction.

References

1. Russell S., Norvig P. *Artificial Intelligence. A Modern Approach*. 2nd ed., Pearson Publ., 2006, 1112 p. (Russ. ed.: Moscow, Vilyams Publ., 2007, 1410 p.).
2. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M. Playing Atari with deep reinforcement learning. *NIPS Deep Learning Workshop*. USA, 2013. Available at: <https://arxiv.org/pdf/1312.5602.pdf> (accessed February 8, 2018).
3. Sutton R.S., Barto A.G. *Reinforcement Learning: An Introduction*. A Bradford Book Publ., 1998, 322 p. (Russ. ed.: Moscow, Binom Publ., 2017, 400 p.).
4. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016, 775 p. (Russ. ed.: Moscow, DMK-Press, 2017, 652 p.).
5. Melo F.S. *Convergence of Q-learning: A simple proof*. Institute of Systems and Robotics. Tech. Rep. Lisbon. 2011. Available at: <http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf> (accessed February 8, 2018).
6. Mnih V. Human-level control through deep reinforcement learning. *Nature*. 2015, vol. 518, no. 7540, pp. 529–533.
7. Schaul T., Quan J., Antonoglou I., Silver D. Prioritized experience replay. *Proc. ICLR-16*. 2016. Available at: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/prioritized-replay.pdf (accessed February 8, 2018).
8. Silver D., Lever G., Heess N., Degris T., Wierstra D., Riedmiller M. Deterministic policy gradient algorithms. *Proc. ICML-14*. 2014. Available at: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/deterministic-policy-gradients.pdf (accessed February 8, 2018).
9. Nikolenko S., Kadurin A., Arkhangelskaya E. *Glubokoe obuchenie* [Deep Learning]. Piter Publ., 2017, 480 p.
10. Mnih V., Badia A.P., Mirza M., Graves A., Harley T., Lillicrap T.P., Silver D., Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. *Proc. ICML-16*. 2016. Available at: <https://arxiv.org/pdf/1602.01783.pdf> (accessed February 8, 2018).
11. Liu D.R., Li H.L., Wang D. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *Int. J. of Automation and Computing*. 2015, vol. 12, pp. 229–242.