Московский государственный университет имени М. В. Ломоносова
механико-математический факультет
кафедра Математической теории интелектуальных систем

Курсовая работа
студента 1 курса магистратуры

Манкаева Нарана Николаевича

Реализация алгоритма обучения с подкреплением Advantage Actor Critic с оценкой General Advantage Estimation на примере восьминогого робота.

Научный руководитель:
к.ф.-м.н.
В.С. Половников

Москва, 2021

Оглавление

1. Введение

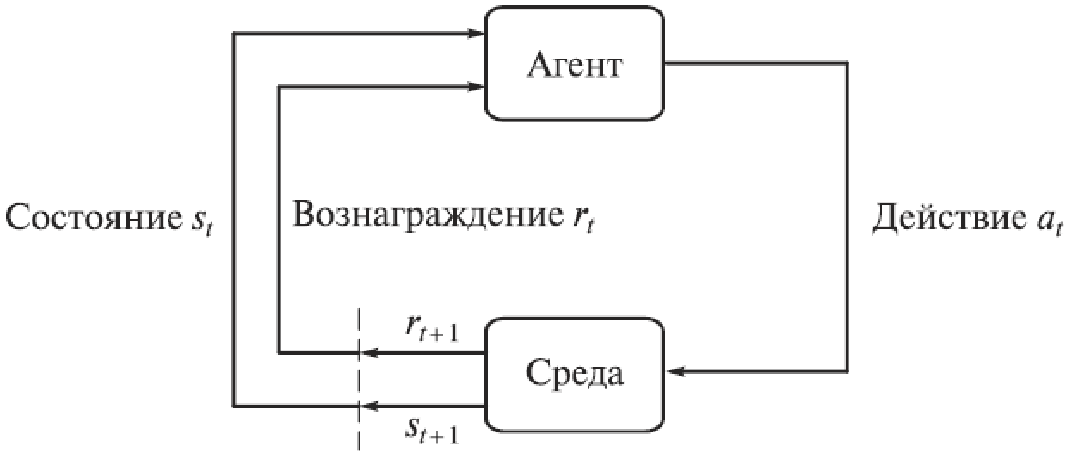
На сегодняшний день обучение с подкреплением (англ. Reinforcment Learning, RL) является одним из основополагающих способов машинного обучения, активно применяющийся на практике. При помощи обучения с подкреплением, или RL, были достигнуты успехи при создании искусственного интеллекта для сложных стратегических игр таких как го, сёги и шахматы [1]; RL используется в сфере финансов при создании трейдинговых ботов [2]; также RL применяется для обучения компьютеров управлением роботов как смоделированных, так и реальных, физических [3, 4].

Основной концепцией обучения с подкреплением является взаимодействие испытуемой системы (*агент*), находящейся в некотором *состоянии* $s_t \in S$, где S – множество возможных состояний, со *средой* в течении дискретных временных шагов $t = 0, 1, 2, 3, \ldots$. Взаимодействие происходит путем совершения *действия* $a_t \in A(s_t)$, где $A(s_t)$ – множество действий, возможных в состоянии s_t. Это действие переводит агента из одного состояния s_t в другое s_{t+1}, при этом за совершенные действия среда выдает агенту *награду* $r_{t+1} \in \mathbb{R}$ [5]. Рисунок 1 иллистрирует взаимодействие агент – среда .

На каждом временном шаге агент осуществляет отображение из множества состояний на множество вероятностей выбора каждого из возможных действий. Это отображение называется *стратегией* агента и обозначается π_t, где $\pi_t(s, a)$ – вероятность того, что $a_t =$

a, если $s_t = s$. Главной целью агента является максимизация суммы всех вознаграждений, которую он получит в долгосрочной перспективе.

Рис. 1. Цикл взаимодействия агента со средой



Изменение стратегии агента в зависимости от имеющегося опыта происходит согласно алгоритмам обучения с подкреплением. Одним из основных алгоритмов, используемых в RL, является алгоритм *Advantage actor critic* (или A2C) [6]. Его преимущество содержится в сочетании двух типов алгоритмов обучения с подкреплением (на основе *стратегий* и на основе *ценностей*). Ключевыми элементами алгоритмов на основе ценностей являются *функция ценности состояния* $V^{\pi}(s)$ и *функция ценности действия* $Q^{\pi}(s, a)$ [5]. Функция ценности состосяния $V^{\pi}(s)$ называется ожидаемая выгода, полученная агентом согласно стратегии π при начальном состоянии s. Функция ценности действия $Q^{\pi}(s,a)$ – это ожидаемая выгода, полученная согласно стратегии π при начальном состоянии s и осуществленном дейсвтии a. Алгоритмы на основе ценностей учатся выбирать действия, опираясь на прогнозируемое значение функций ценностей входного состояния или действия [5, 7]. Агенты, использующеие алгоритымы на основе стратегий, непосредственно изучают стратегию (распределение вероятностей действий) [5, 8].

В данной работе рассматривается восьминогий робот, построенный в симмуляционной среде Mujoco. Целью курсовой работы является обучение робота передвижению при помощи алгоритма Advantage actor critic с оценкой *функции преимущества* (англ. *Advantage*) через *General Advantage Estimation* [9].

2. Алгоритм Advantage actor critic

2.1. Алгоритмы обучения с подкреплением

Алгоритмы обучения с подкреплением делятся на две основные ветки: алгоритмы, имеющие доступ к модели среды и не имеющие. Под моделью среды подразумевается функция, которая предсказывает переходы между состояниями и их вознограждения.

Основным преимуществом алгоритмов, использующих модель, является то, что они позволяют агенту с помощью модели среды планировать, заранее узнавать, что произойдет с рядом возможных вариантов действий, и выбрать один из них. С помощью этого агенты могут преобразовать результаты заблаговременного планирования в усвоенную стратегию.

Но главный недостаток таких алгоритмов заключается в том, что агент может использовать модель очень прямолинейно, в результате чего он будет показывать хорошие результаты именно на этой изученной модели, но вести себя неоптимально (или даже ужасно) в реальной среде. В связи с этим в данный момент распространены алгоритмы, не задействующие модель в своем обучении. Так же это связано с их менее трудной реализацией и легкой настройкой.

Алгоритмы свободные от модели в свою очередь подразделяются на алгоритмы, основывающиеся на оптимизации стратегии (алгоритм REINFORCE [10]), и на алгоритмы, нацеленные на использовании функций ценностей (Q-learning [7]). Рассматриваемый в данной работе алгоритм Advantage Actor Critic содежит в себе элементы двух этих типов алгоритмов, что дает ему преимущество при его реализации на реальных задачах.

В большей степени алгоритм A2C основывается на алгоритме REINFORCE, так как алгоритм A2C в некотором роде является его модернизированной версией. Поэтому для понимания алгоритма A2C нам необходимо и описать алгоритм REINFORCE. Рассмотрим данные алгоритмы подробнее.

2.2. Алгоритм REINFORCE

Алгоритм REINFORCE, как уже было сказано, является алгоритмом оптимизации стратегии $\pi_\theta(s|a)$, где θ некоторый параметр. Как и во всех алгоритмах обучения с подкреплением его главной целью является максимизировать суммарное вознаграждение $R = \sum_{t=1}^{T} r_t$, где T – шаг, на котором произошел переход в терминальное состояние. Для достижения этой цели в алгоритме REINFORCE используется метод оптимизации, называемый методом градиентного спуска [11]. Метод градиентного спуска в алгоритме REINFORCE применяется для функции

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[R_\tau\right] = \int p_\theta(\tau) R_\tau d\tau,$$

где τ обозначение некоторого сценария — последовательности состояний и произведенных в них действий: $\tau = (s_1, a_1, s_2, a_2, ...s_T, a_T)$; $R_\tau = \sum_\tau r(s_t, a_t)$ — сумма всех вознаграждений, полученных в ходе сценария; $p_\theta(\tau)$ — вероятность реализации сценария.

Вероятность реализации сценария зависит от поведения среды, которое задается вероятностями перехода между состояниями $p(s_{t+1}|s_t, a_t)$, распределением начальных состояний $p(s_1)$ и поведением агента, которое определяется его стохастической стратегией $\pi_\theta(a_t|s_t)$. Вероятностное распределение над сценариями, таким образом, задается как

$$p_\theta(\tau) = p_\theta(s_1, a_1, ...s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Так как алгоритм REINFORCE является алгоритмом свободным от модели, то вероятности перехода между состояниями агенту не известны. Это затрудняет расчет $\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) R_\tau d\tau$, необходимый для применения метода градиентного спуска. Однако можно заметить, что $p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$. Тогда, делая приведенную замену, формула для градиента математического ожидания суммарного выигрыша примет следующий вид

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R_\tau d\tau = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R_\tau \right]$$

Так как $p_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$, то $\log p_\theta(\tau)$ разложится в сумму:

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \left(\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)\right)$$

В свою очередь,

$$\nabla_\theta \log p_\theta(\tau) = \underbrace{\nabla_\theta \log p(s_1)}_{=0} + \sum_{t=1}^{T} \left(\nabla_\theta \log \pi_\theta(a_t|s_t) + \underbrace{\nabla_\theta \log p(s_{t+1}|s_t, a_t)}_{=0}\right) =$$

$$= \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Тогда, подставляя это выражение в формулу градиента $\nabla_\theta J(\theta)$ мы получим

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R_\tau \right]$$

Заметим, что в получившееся выражение для $\nabla_\theta J(\theta)$ уже напрямую не входят значения $p(s_{t+1}|s_t, a_t)$ и $p(s_1)$, которые были нам неизвестны.

Таким образом, если у нас есть в наличии сценарий τ и соответсвующее ему значение вознаграждения R_τ, мы можем вычислить величину $\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right) R_\tau$. Поэтому, если у нас есть выборка из N уже известных сценариев $\tau^i = (s_1^i, a_1^i, ...s_{T^i}^i, a_{T^i}^i)$, полученная из распределения $\tau \sim p_\theta(\tau)$, то мы можем посчитать приблизительное значение $\nabla_\theta J(\theta)$ по методу Монте-Карло [12] – вычислив выборочное среднее случайной величины:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right) R_{\tau^i} =$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T^i}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_{t=1}^{T^i} r(s_t^i, a_t^i)\right)$$

Несмещенная выборка сценариев τ из вероятностного распределения $p_\theta(\tau)$, в свою очередь, находится нами из взаимодействия агента со средой при фикисрованном параметре θ.

Таким образом, окончательный вид алгоритма REINFORCE будет таким:

1) Прогнать N сценариев τ_i со стратегией $\pi_\theta(a|s)$;

2) Вычислить

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T^i}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_{t=1}^{T^i} r(s_t^i, a_t^i)\right);$$

3) Обновить параметр $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$;

4) Если не сошлись к экстремуму, повторить цикл.

Несмотря на свое преимущество в виде простоты реализации алгоритм REINFORCE имеет ряд недостатков таких как: низкая скорость работы из-за необходимости многократно выполнять взаимодейсвие со средой для получения выборки сценариев, при этом для обновленного параметра θ количество взаимодействий не уменьшается; большая дисперсия случайной величины $\nabla_\theta \log p_\theta(\tau) R_\tau$, так как для различных τ значения R_τ могут сильно различаться.

Но существуют некоторые модернизации алгоритма для нивелирования эффектов этих недостатков. Для того чтобы уменьшить дисперсию случайной величины $\nabla_\theta \log p_\theta(\tau) R_\tau$ необходимо воспользоваться так называемыми *опорными значениями* b (англ. *baseline*) [13]. Заметим, что если b константа относительно τ, то

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau)(R_\tau - b) \right] = E_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) R_\tau \right].$$

Но при этом дисперсия случайной величины будет зависеть от b. Поэтому регулируя опроное значение b, можно добиться уменьшения дисперсии случайной величины.

Также, можно заметить, что нет необходимости рассматривать всю траекторию сценария τ для суммы вознаграждений $r(s_t, a_t)$, так как в момент времени t от действия a_t зависит только $r(s_{t'}, a_{t'})$ для $t' \leq t$. Поэтому выражение для $\nabla_\theta J(\theta)$ примет следующий вид:

$$\nabla_\theta J(\theta) \approx E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \underbrace{\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right)}_{= Q_{\tau,t}} \right],$$

где $Q_{\tau,t}$ будем называть *будущим выигрышем* на шаге t в сценарии τ.

2.3. Алгоритм Advantage Actor Critic

Алгоритм A2C основывается на этих двух типах улучшения алгоритма REINFORCE, при этом задействуя в своем обучении функции ценности состояния $V^\pi(s)$ и функции ценности действия $Q^\pi(s,\ a)$. Рассмотрим выведенную формулу для $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i) Q_{\tau_i,t}\ .$$

Здесь $Q_{\tau_i,t}$ – оценка будущего выигрыша из состояния s_t^i при условии действия a_t^i, которая базируется только на одном сценарии τ_i. Это плохое приближение ожидаемого будущего выигрыша – истинным ожидаемым будущим выигрышем является значение функции ценности действия $Q^\pi(s,\ a)$, которое выражается формулой:

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t, a_t]$$

В целях уменьшения дисперсии случайной величины в алгоритме A2C используется опорное значение b равное значению функции ценности состояния $V^\pi(s)$, которое выражается следующей формулой:

$$V^\pi(s_t) = E_{a_t\sim\pi_\theta(a_t|s_t)}[Q^\pi(s_t, a_t)] = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t]$$

Таким образом, вместо ожидаемого будущего выигрыша $Q_{\tau_i,t}$ при оценке $\nabla_\theta J(\theta)$ будем использовать так называемую *функцию преимущества* $A^\pi(s_t, a_t)$ (англ. *advantage*):

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Эта функция показывает преимущество действия a_t в состоянии s_t над остальными действиями в этом состоянии. То есть, то насколько выгоднее выбрать именно действие a_t в отличие от остальных действий.

В итоге мы имеем:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)A^\pi(s_t^i, a_t^i)$$

Для того чтобы оценить значения $A^\pi(s_t^i, a_t^i)$ нет необходимости оценивать оба значения функции ценности состояния $V^\pi(s_t)$ и функции ценности действия $Q^\pi(s_t, a_t)$. Если воспользоваться уравнением Беллмана можно выразить функцию преимущества только через функцию ценности состояния [14].

Уравнение Беллмана:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + E_{s_{t+1}\sim p(s_{t+1}|s_t, a_t)}[V^\pi(s_{t+1})] \approx r(s_t, a_t) + V^\pi(s_{t+1})$$

Тогда выражение для функции преимущества примет следующий вид:

$$A^\pi(s_t^i, a_t^i) = Q^\pi(s_t, a_t) - V^\pi(s_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)$$

Теперь, для того чтобы оценить значения $A^\pi(s_t^i, a_t^i)$, нам нужно уметь оценивать только $V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'})|s_t]$. Мы можем делать это, опять же, с помощью метода Монте-Карло, но это будет работать не существенно быстрее, чем обычный алгоритм REINFORCE. Вместо этого заметим, что при фиксированных s_t и a_t выполняется:

$$V^\pi(s_t) = r(s_t, a_t) + V^\pi(s_{t+1})$$

Таким образом, если мы имеем некоторую изначальную оценку $V^{\pi}(s_t)$ для всех s, мы можем можем обновлять эту оценку аналогично алгоритму Q-learning [7]:

$$V^{\pi}(s_t) \leftarrow (1-\beta)V^{\pi}(s_t) + \beta(r(s_t, a_t) + V^{\pi}(s_{t+1}))$$

Здесь β – *коэффициент обучения* (англ. *learning rate*) для функции ценности состояния $V^{\pi}(s_t)$.

Такой пересчет мы можем производить каждый раз, когда агент получает вознаграждение за действие. Так мы получим оценку ценности текущего состояния, не зависящую от выбранного сценария развития событий τ, а значит, и оценка функции преимущества не будет зависеть от выбора конкретного сценария. Это сильно снижает дисперсию случайной величины $\nabla_{\theta} \log \pi_{\theta}(a_t^i|s_t^i) A^{\pi}(s_t^i, a_t^i)$, что делает оценку $\nabla_{\theta} J(\theta)$ достаточно точной даже в том случае, когда мы используем всего один сценарий для ее подсчета:

$$\nabla_{\theta} J(\theta) \approx \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) A^{\pi}(s_t, a_t)$$

Окончательный вид алгоритма Advantage actor critic будет следующим:

1) производим действие $a \sim \pi_{\theta}(a|s)$, переходим в состояние s' и получаем вознаграждение r;

2) $V^{\pi}(s) \leftarrow (1-\beta)V^{\pi}(s) + \beta(r + V^{\pi}(s'))$;

3) $A^{\pi}(s, a) \leftarrow r + V^{\pi}(s') - V^{\pi}(s)$;

4) $\nabla_\theta J(\theta) \leftarrow \nabla_\theta \log \pi_\theta(a|s) A^\pi(s, a)$;

5) $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$;

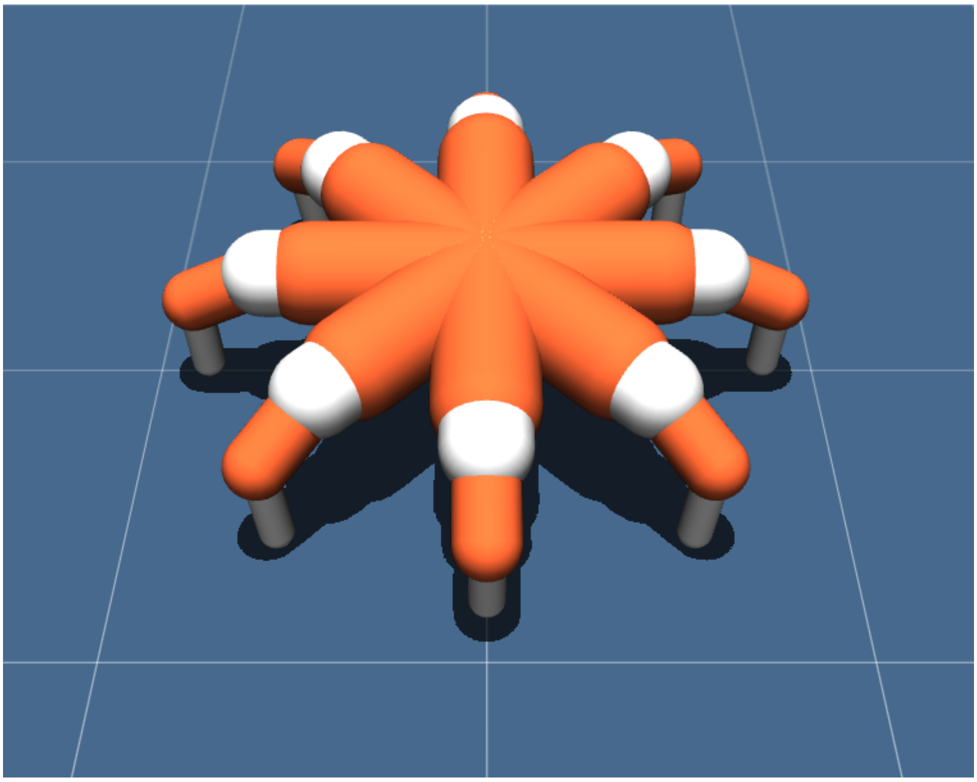
6) Если не сошлись к экстремуму, повторить цикл.

В алгоритме Advantage actor critic под actor'ом подразумевается компонента, которая оптимизирует стратегию $\pi_\theta(a|s)$. Critic, в свою очередь, подсчитывает значения функции ценности состояния $V^\pi(s)$. То есть actor определяет дальнейшее действие, а critic оценивает, насколько то или иное действие выгодно, основываясь на функции преимущества $A^\pi(s, a)$.

3. Задача обучения восьминогого робота

В данной работе рассматривается восьминогий робот, построенный в симмуляционной среде Mujoco. На рисунке 2 изображена модель робота в начальном положении в этой симмуляционной среде.

Рис. 2. Модель восьминогого робота в симмуляционной среде Mujoco



На входе агенту предоставляется множество чисел из Mujoco, необходимые для описания состояний агента: относительные позиции, углы вращения, скорости, ускорения частей тела робота, и т.д.

(примерно 800 признаков). В свою очередь, на выходе нейросети будем ожидать 24 числа – углы поворота шарниров, на которых закреплены конечности.

Целью агента будет максимизировать суммарную награду за *эпизод*. Под эпизодом подразумевается длительное взаимодействие агента со средой, начавшееся с определенного начального состояния и завершающееся терминальным. В нашей задаче эпизод завершается, если робот упал или если прошло 3000 шагов симмуляции. При каждом шаге симмуляции агент получает награду по следующей формуле:

$$r_t = \Delta x * 1000 + 0.5$$

Т.е. целью агента будет увеличивать свою координату x и не падать до конца эпизода.

Таким образом условие задачи обучения робота будет следующим: найти функцию $\pi : \mathbb{R}^{800} \to \mathbb{R}^{24}$, для которой награда за эпизод будет наибольшей. Другими словами найти оптимальную стратегию

$$\pi^* = \arg\max_{\pi} J(\pi_\theta) \ ,$$

где $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{n} r_t\right]$.

Решать данную задачу мы будем с помощью выше описанного алгоритма обучения с подкреплением Advantage actor critic. Алгоритм A2C задействует в своем обучение две нейросети: нейросеть актора для оптимизации стратегии $\pi_\theta(a|s)$ и нейросеть критика, оценивающая значения функции ценности состояния $V^{\pi}(s)$.

Нейросеть критика, как уже было сказано, необходима нам для получения оценки значений функции ценности состояния $V^{\pi}(s)$. Оценка значения функции ценности состояния $V^{\pi}(s)$, в свою очередь, нам необходима для оценки функции преимущества $A^{\pi}(s,a)$ (3-й пункт в алгоритме A2C, описанном в прошлом разделе). Но в данной работе в отличии от обычного алгоритма A2C оценивать функцию преимущества $A^{\pi}(s,a)$ мы будем с помощью метода General Advantage Estimation (или просто GAE) [9], который значительно увеличивает точность этой оценки.

Для применения метода оценки функции преимущества GAE в алгоритме A2C, нам необходимо заменить старую оценку функции преимущества $A^{\pi}(s_t,a_t) = r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$ на следующую:

$$A_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+1}^V \ ,$$

где $\delta_t^V = r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$; γ и λ некоторые гиперпараметры.

После получения значений функции преимущества $A_t^{\pi}(s,a)$ мы наконец можем оптимизировать стратегию $\pi_\theta(a|s)$. Оптимизация стартегии $\pi_\theta(a|s)$ реализуется с помощью метода градиентного спуска, применяемый к функции $J(\pi_\theta)$, так называемой функции ожидаемого суммарного вознаграждения. Для применения градиентного спуска необходимо вычислить $\nabla J(\pi_\theta)$, который в нашем случае равен:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)\right],$$

где $A^\pi(s_t, a_t)$ оценивается с помощью выше описанного метода GAE.

Можно заметить, что функция потерь для нейросети актора тогда будет следующей:

$$loss = -\log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)$$

Минус появился за счёт того, что мы хотим максимизировать $J(\pi_\theta)$.

Для реализции данных шагов был написан на языке Python соответствующий комплект программ для нейросети критика и нейросети актора (Приложение А). Для обучения этих нейросетей, в свою очерель, был написан генератор данных (Приложение Б), который выдает кортежи вида:

$$(s_t, a_t, V^\pi(s_t), \sum_{t'=t}^{T} r(s_{t'}, a_{t'})).$$

(Для полного ознакомления с кодом программы перейдите по следующей ссылке: `https://github.com/narmanka/A2C-GAE-8legrobot`)

После запуска написанного кода и длительного обучения нейросетей была получена модель восьминогового робота, которая демонстрирует активное передвижение по оси x, что говорит об успешном обучении робота. Динамика робота представлена на рисунке 3 и 4.

Также проследить за динамикой восьминогого робота можно по видеозаписям, расположенных на следующих ссылках:

Камера 1 –

https://drive.google.com/file/d/1N8eXENDS3XEbKQEv23w_JGFXBgnXZTCW/view?usp=sharing

Камера 2 –

https://drive.google.com/file/d/1dqAldfSOO-nDlCQPm9hODoxPcrluWKqT/view?usp=sharing

Замедленное движение (x0.125) –

https://drive.google.com/file/d/1coEiQTFwbj9as6F0u7No-G-FeRbsw0ib/view?usp=sharing

Рис. 3. Динамика восьминогого робота

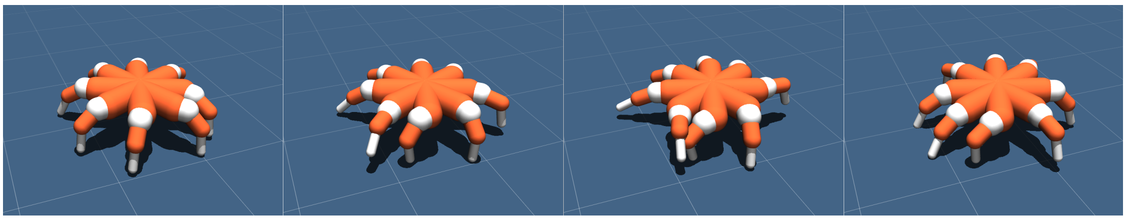
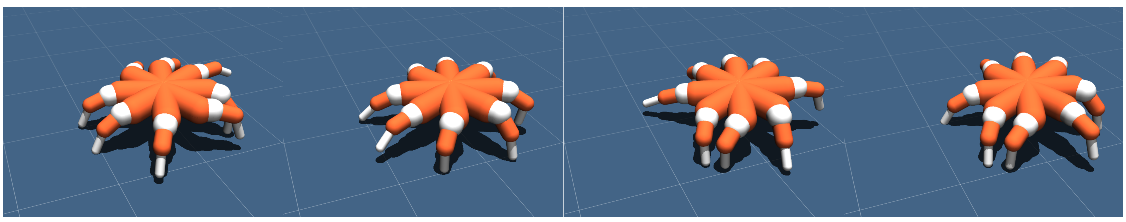


Рис. 4. Динамика восьминогого робота (продолжение)



Из предложенных видеозаписей и рисунков можно заметить, что робот научился достаточно быстро передвигаться путем синхронного отталкивания от земли симметрично-парных ног, таким образом не теряя равновесие и приобретая большую скорость. При этом передняя пара ног и задняя пара ног хоть и участвуют в "прыжках"робота, но фактически являются ногами для удержания баланса. Большую часть скорости приносят две пары боковых ног.

Данное делегирование задач на определенные пары ног являются обоснованной и логичной, так как целью робота является не только быстрое передвижение, но и удержание баланса в течение этого передвижения.

4. Заключение

В ходе работы была построена модель восьминогого робота в симмуляционной среде Mujoco. Также была поставлена задача обучения робота передвижению, для решения которой был применен алгоритм Advantage actor critic с оценкой функции преимущества $A^{\pi}(s,a)$ методом GAE. Для применения алгоритма A2C и решения данной задачи был разработан комплект программ на языке Python.

При выполнении задачи алгоритм Advantage actor critic показал хоршие результаты, успешно обучив восьминогого робота передвижению. Оценка функции преимущества $A^{\pi}(s,a)$ методом GAE так же показала отличные результаты на примере нашей задачи.

Список литературы

1. Silver D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play //Science. – 2018. – T. 362. – №. 6419. – C. 1140-1144.
2. Huang C. Y. Financial trading as a game: A deep reinforcement learning approach //arXiv preprint arXiv:1807.02787. – 2018.
3. Sallab A. E. L. et al. Deep reinforcement learning framework for autonomous driving //Electronic Imaging. – 2017. – T. 2017. – №. 19. – C. 70-76.
4. Kuindersma S. et al. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot //Autonomous robots. – 2016. – T. 40. – №. 3. – C. 429-455.
5. Sutton R. S., Barto A. G. Reinforcement learning: An introduction. – MIT press, 2018.
6. Mnih V. et al. Asynchronous methods for deep reinforcement learning //International conference on machine learning. – PMLR, 2016. – C. 1928-1937.
7. Watkins C. J. C. H., Dayan P. Q-learning //Machine learning. – 1992. – T. 8. – №. 3-4. – C. 279-292.
8. Silver D. et al. Deterministic policy gradient algorithms //International conference on machine learning. – PMLR, 2014. – C. 387-395.

9. Schulman J. et al. High-dimensional continuous control using generalized advantage estimation //arXiv preprint arXiv:1506.02438. – 2015.
10. Sutton R. S. et al. Policy gradient methods for reinforcement learning with function approximation //Advances in neural information processing systems. – 2000. – C. 1057-1063.
11. Ruder S. An overview of gradient descent optimization algorithms //arXiv preprint arXiv:1609.04747. – 2016.
12. Metropolis N., Ulam S. The monte carlo method //Journal of the American statistical association. – 1949. – T. 44. – №. 247. – C. 335-341.
13. Greensmith E., Bartlett P. L., Baxter J. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning //Journal of Machine Learning Research. – 2004. – T. 5. – №. 9.
14. Baird L. Residual algorithms: Reinforcement learning with function approximation //Machine Learning Proceedings 1995. – Morgan Kaufmann, 1995. – C. 30-37.

Приложение А

Код актора

```
class ActorNetworkContinuous:
    def __init__(self):
        self.state_ph = tf.placeholder(tf.float32,
        shape=[None, observation_space])
        l1 = tf.layers.dense(self.state_ph, units=100,
        activation=tf.nn.tanh)
        l2 = tf.layers.dense(l1, units=50,
        activation=tf.nn.tanh)
        l3 = tf.layers.dense(l2, units=25, activation=tf.nn.tanh)
        mu = tf.layers.dense(l3, units=action_space)
        log_std = tf.get_variable(name='log_std',
        initializer=-0.5*np.ones(action_space,dtype=np.float32))
        std = tf.exp(log_std)
        self.action_op = (mu +
                          tf.random.normal(shape=tf.shape(mu)) * std)
        # Training
        self.weight_ph = tf.placeholder(shape=[None],
        dtype=tf.float32)
        self.action_ph = tf.placeholder(shape=[None, action_space],
        dtype=tf.float32)
```

```
        action_logprob = gaussian_loglikelihood(self.action_ph, mu,
        log_std)
        self.loss = -tf.reduce_mean(action_logprob * self.weight_ph)
        optimizer = tf.train.AdamOptimizer(learning_rate=
        actor_learning_rate)
        self.update_op = optimizer.minimize(self.loss)
```

Код критика

```
class CriticNetwork:
    def __init__(self):
        self.state_ph = tf.placeholder(tf.float32,
        shape=[None, observation_space])
        l1 = tf.layers.dense(self.state_ph, units=100,
        activation=tf.nn.tanh)
        l2 = tf.layers.dense(l1, units=50, activation=tf.nn.tanh)
        l3 = tf.layers.dense(l2, units=25, activation=tf.nn.tanh)
        output = tf.layers.dense(l3, units=1)
        self.value_op = tf.squeeze(output, axis=-1)

        # Training
        self.value_ph = tf.placeholder(shape=[None], dtype=tf.float32)
        self.loss = tf.losses.mean_squared_error(self.value_ph,
        self.value_op)
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=
critic_learning_rate)
self.update_op = optimizer.minimize(self.loss)
```

Приложение Б

Код генератора данных

```
def generate_batch(envs, batch_size, replay_buffer_size):
    envs_number = envs.num_envs
    observations = [[0 for i in range(observation_space)]
    for i in range(envs_number)]
    replay_buffer = np.empty((0,4), np.float32)
    rollouts = [np.empty((0, 3)) for i in range(envs_number)]
    while True:
        history = {'reward': [], 'max_action': [],
                   'mean_advantage': [], 'mean_value': []}
        replay_buffer = replay_buffer[batch_size:]

        # Main sampling cycle
        while len(replay_buffer) < replay_buffer_size:
            actions = sess.run(actor.action_op,
            feed_dict={actor.state_ph: observations})
            observations_old = observations
            observations, rewards, dones, _ = envs.step(actions *
            angle_normalization)
            observations /= angle_normalization
            history['max_action'].append(np.abs(actions).max())
```

```
time_point = np.array(list(zip(observations_old, actions,
rewards)))
for i in range(envs_number):
    rollouts[i] = np.append(rollouts[i], [time_point[i]],
    axis=0)
if dones.all():
    print('WARNING: envs are in sync!!
    This makes sampling inefficient!')
done_indexes = np.arange(envs_number)[dones]
for i in done_indexes:
    rewards_trajectory = rollouts[i][:, 2].copy()
    history['reward'].append(rewards_trajectory.sum())
    advantage, values = estimate_advantage(states=
    np.array(rollouts[i][:, 0].tolist()),
    rewards=rewards_trajectory)
    history['mean_value'].append(values.mean())
    history['mean_advantage'].append(advantage.mean())
    rollouts[i][:, 2] = advantage
    discounted_reward_to_go = discount_cumsum(
    rewards_trajectory, coef=discount_factor)
    rollout = np.hstack((rollouts[i],
    replay_buffer = np.append(replay_buffer, rollout,
```

```
                axis=0)
                rollouts[i] = np.empty((0, 3))
        np.random.shuffle(replay_buffer)
        replay_buffer = replay_buffer[:replay_buffer_size]
        yield replay_buffer[:batch_size], history
a = generate_batch(envs, 8, 64)
for i in range(10):
    next(a)
next(a)[0]
```