УДК 004 **Информационные технологии**

Чачанидзе Елизавета Романовна, *студент Московского Государственного Технического Университета им. Н.Э. Баумана, Россия, г. Москва*

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ PROXIMAL POLICY OPTIMIZATION И SOFT-ACTOR-CRITIC

Аннотация: Данная статья призвана провести сравнительный анализ двух алгоритмов, которые используется для машинного обучения с подкреплением. Данные алгоритмы используются в библиотеке Unity ML-agents, из-за чего возникает необходимость их сравнения. В статье описываются основным математические особенности алгоритмов и на основании различий в работе каждого из алгоритмов составляется сравнительная таблица. После анализа таблицы сделаны выводы о том, в каких условиях лучше использовать каждый алгоритм.

Ключевые слова: машинное обучение; алгоритм; среда, агент.

Annotation: This article is intended to conduct a comparative analysis of two algorithms that are used for machine learning with reinforcement. These algorithms are used in the Unity ML-agents library, which makes it necessary to compare them. The article describes the main mathematical features of the algorithms and based on the differences in the operation of each of the algorithms, a comparative table is compiled. After analyzing the table, conclusions are made about the best conditions for using each algorithm.

Keywords: machine learning; algorithm; environment, agent.

1 Алгоритмы для оптимизации политики агентов

На рисунке 3 ниже представлена схема связи между существующими семействами алгоритмов машинного обучения с подкреплением [1]. Как видно

на схеме, семейство policy optimization является наиболее обширным. Оно же и является наиболее часто используемым и наиболее эффективным в большинстве случаев.

Далее в этой будут приведены математические описания основ алгоритмов оптимизации политики агентов. В теории машинного обучения выделяют понятие policy gradient. Это понятие относится именно к тем алгоритмам, которые предсказывают оптимальные действия агентов напрямую. Алгоритмы такого типа используются для обучения агентов в 3-d средах. Библиотека ML-agents для Unity предоставляет наиболее простой доступ к обучению агентов с помощью обучения с подкреплением. Данная библиотека имеет реализацию двух алгоритмов из данного семейства – PPO и SAC.

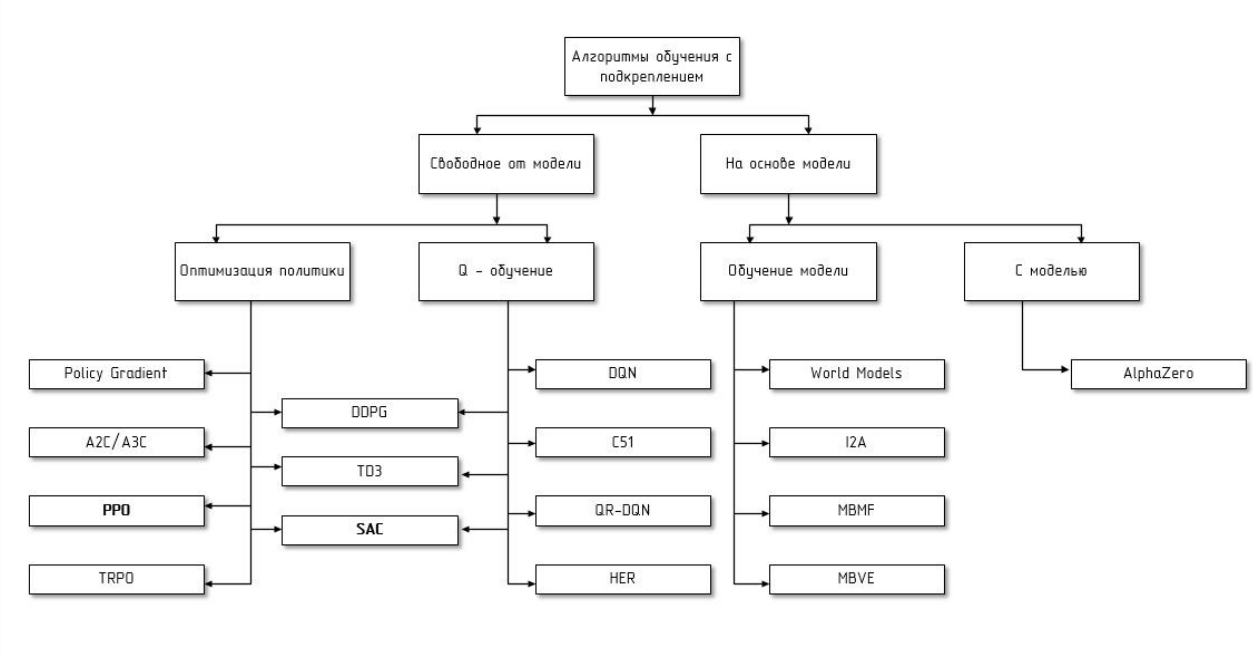

Рисунок 1. Семейство алгоритмов обучения с подкреплением.

Для того, чтобы использовать алгоритмы с policy gradient необходимо выражение, которое можно вычислить. Для этого необходимо предпринять два шага. Первый шаг — это аналитическое определение градиента эффективности политики, которое должно иметь ожидаемое значение. Вторым шагом является формирование ожидаемого значения эффективности политики, которое можно вычислить с помощью данных из симулируемых сред.

2 Proximal Policy Optimization

Proximal Policy Optimization или сокращенно PPO это алгоритм, основной мотивацией которого является вопрос: как получить наибольшее возможное улучшение результатов за один шаг на основе тех данных, что уже имеются и при этом не ухудшить текущие показатели эффективности политики. PPO старается делать свои новые политики более похожими на старые с помощью более простых методов чем другие алгоритмы из его семейства. При этом PPO имеет такую же эффективность. Существуют две основных вариации PPO: PPO-Penalty и PPO-clip. PPO-Clip используется гораздо чаще, поэтому далее фокус именно на этой версии алгоритма. Данные версии отличаются способом подтверждения того, что новая политика не слишком сильно отличается от старой.

Основные уравнения PPO представлены далее:

PPO обновляет свои политики следующим образом:

$$\theta_{k+1} = argmax_{\theta} E_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k\theta)] \quad (1)$$

где: θ_{k+1} это параметры k+1 политики;

$E_{s,a\sim\pi_{\theta_k}}$ это ожидаемое значение при состоянии s и действии агента a на основании политики π_{θ_k};

$L(s,a,\theta_k\theta)$ это целевая функция.

Обычно требуется несколько шагов симуляции чтобы максимизировать целевую функцию. Здесь целевая функция L это:

$$L(s,a,\theta_k\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), clip\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right), \quad (2)$$

где: $\pi_\theta(a|s)$ и $\pi_{\theta_k}(a|s)$ это политики действий с соответствующими параметрами θ и θ_k;

$A^{\pi_{\theta_k}}(s,a)$ – это преимущество, которое агент получает при следовании политике π_{θ_k};

ϵ – это гиперпараметр, который и определяет, насколько сильно отличается новая политика от предыдущей.

Существует упрощенная версия уравнения (2), которая более четко описывает влияние гиперпараметра эпсилон на сохранение политики действий. Данная версия используется при программировании данного алгоритма в Tensorflow [2].

$$L(s,a,\theta_k\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right), \quad (3)$$

где: g это функция:

$$g(\epsilon, A) = \begin{cases}(1+\epsilon)A & A \geq 0\\ (1-\epsilon)A & A < 0\end{cases} \quad (4)$$

Далее рассмотрим более подробно что происходит в единичном случае с парой состояние-действие (s,a) в различных случаях.

Преимущество А больше нуля: положим что выбранная пара состояние-действие (s,a) приводит к положительному значению А, в таком случае вклад этой пары можно свести к:

$$L(s,a,\theta_k\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right)A^{\pi_{\theta_k}}(s,a) \quad (5)$$

Так как преимущество А положительно, целевое значение увеличиться если действие станет более вероятным, то есть если $\pi_\theta(a|s)$ увеличиться. Однако, наличие min в этом выражение ставит ограничение на то, насколько сильно может увеличиться целевое значение. Как-только $\pi_\theta(a|s)$ становится больше $(1+\epsilon)\pi_{\theta_k}(a|s)$, условие о минимальности вступает в силу и выражение не может принять значение выше $(1+\epsilon)A^{\pi_{\theta_k}}(s,a)$. Таким образом алгоритм обеспечивает, что новая политика не будет выгодной, если она слишком отличается от старой.

Преимущество А меньше нуля: положим что выбранная пара состояние-действие (s,a) приводит к отрицательному значению А, в таком случае вклад этой пары можно свести к:

$$L(s,a,\theta_k\theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right)A^{\pi_{\theta_k}}(s,a) \quad (6)$$

Так как преимущество A отрицательно, целевое значение увеличиться если действие станет менее вероятным, то есть если $\pi_\theta(a|s)$ уменьшится. Однако, наличие max в этом выражение ставит ограничение на то, насколько сильно может увеличиться целевое значение. Как-только $\pi_\theta(a|s)$ становится меньше $(1-\epsilon)\pi_{\theta_k}(a|s)$, условие о максимальности вступает в силу и выражение ограничивается значением $(1-\epsilon)A^{\pi_{\theta_k}}(s,a)$. И в данном случае алгоритм также обеспечивает, что новая политика не будет выгодной, если она слишком отличается от старой.

PPO тренирует стохастическую политику придерживаясь существующий политики. Это означает, что алгоритм исследует возможные новые действия на основе уже существующей политики. То, насколько случайными будут результаты следующего шага вычислений зависит от начальных условий и процедуры обучения. Обычно, в процессе обучения политика становится менее случайной, так как правила алгоритма подразумевают, что действия, которые приносили хорошие награды в прошлом, должны продолжать использоваться далее. Такой подход может создать так называемый «локальный оптимум». Существует гораздо более оптимальное решение, но так как действия, которые в итоге могли бы к ними привести были отметены в самом начале вычислений, итоговая политика является оптимальной только для определенного набора данных.

3 Soft-Actor-Critic

Мягкий Актор-Критик (англ. Soft Actor Critic) далее SAC это алгоритм, который оптимизирует стохастическую политику действий, не опираясь на существующую политику. Он похож на многие другие алгоритмы, хотя и не является прямым потомком ни одного из них. Данный алгоритм выступает своего рода связующим звеном между ними.

Основной особенностью SAC является регуляризация энтропии. Политика действий создается таким образом, что оптимизироваться компромисс между ожидаемой наградой и энтропией, которая является показателем случайности в политике действий. Данный компромисс тесно связан с отношением между

эффективностью и исследованием внутри политики действий. Чем выше энтропия, тем больше возможных новых вариантов действий рассматривается в процессе обучения. Контроль над энтропией может также помочь избежать ситуации с «локальным оптимумом»

Чтобы полностью описать принцип работы SAC, необходимо описать как именно происходит обучение с подкреплением в условиях, когда над энтропией имеется контроль.

Энтропия — это переменная, которая определяет, насколько случайной может быть другая случайная переменная. Так, чем выше энтропия, тем менее предвзята случайная величина.

Пусть x это случайная переменная с плотностью вероятности P. Энтропия H переменной x вычисляется с помощью P следующим образом:

$$H(P) = E_{x\sim P}[-logP(x)] \quad (7)$$

где: $E_{x\sim P}$ это ожидаемое значение случайной величины с плотностью вероятности P.

В теории обучения с подкреплением с присутствием энтропии, агент получает дополнительную награду на каждом шаге в зависимости от значения энтропии на этом шаге. Таким образом, основное уравнение обучения с подкреплением по нахождению оптимальной политики π^* приводится к следующему виду:

$$\pi^* = argmax_\pi E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(R(s_t, a_t, s_{t+1}) + \alpha H\big(\pi(\cdot\,|s_t)\big)\right)\right] \quad (8)$$

где: $E_{\tau\sim\pi}$ ожидаемое значение при траектории действий τ полученных на основании политики π;

γ^t – функция дисконтирования, необходимая для сходимости бесконечной суммы наград R;

$R(s_t, a_t, s_{t+1})$ – награда, получаемая агентом;

$\pi(\cdot\,|s_t)$ – стохастическая политика;

$\alpha > 0$ это компромиссный коэффициент.

Функция $V^{\pi}(s)$, отображающая получаемые значения в случае, если всегда придерживаться политики π и начинать в состоянии s приобретает следующий вид:

$$V^{\pi}(s) = E_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\left(R(s_t,a_t,s_{t+1}) + \alpha H\big(\pi(\cdot\,|s_t)\big)\right)|s_0 = s\right] \quad (9)$$

где начальное состояние равно s.

Функция $Q^{\pi}(s,a)$, отображающая результаты, если в таких же условиях принять действие a (не обязательно диктуемое существующей политикой) и далее придерживаться политики π, в ситуации с энтропией выглядит следующим образом:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=0}^{\infty}\gamma^{t}H(\pi(\cdot\,|s_t))|s_0 = s, a_0 = a] \quad (10)$$

Между $V^{\pi}(s)$ и $Q^{\pi}(s,a)$ существует связь:

$$V^{\pi}(s) = E_{\alpha\sim\pi}[Q^{\pi}(s,a)] + \alpha H\big(\pi(\cdot\,|s)\big) \quad (11)$$

где: $E_{\alpha\sim\pi}$ ожидаемое значение при действии a выбранном на основании политики π.

Для функций Q и V существуют уравнения Беллмана, которые диктуют следующие: значение в стартовой точке равно ожидаемой награде из данного состояния плюс ожидаемая награда от следующего шага. Они существуют, чтобы научить агентов думать наперед.

Уравнение Беллмана для Q^{π} в случае с энтропией:

$$Q^{\pi}(s,a) = E_{s'\sim P,a'\sim\pi}\left[R(s,a,s') + \gamma\left(Q^{\pi}(s',a') + \alpha H\big(\pi(\cdot\,|s')\big)\right)\right]$$
$$= E_{s'\sim P,}[R(s,a,s') + \gamma V^{\pi}(s')] \quad (12)$$

где: $E_{s'\sim P,a'\sim\pi}$ ожидаемое значение при состоянии s' выбранном случайно из -

SAC одновременно обучается политике π_{θ} и двум Q-функциям: $Q_{\varphi_1}Q_{\varphi_2}$, где φ – это параметры функции Q. Существуют две версии алгоритма SAC, которые являются общепринятым стандартом на данный момент. Первая версия алгоритма использует фиксированное значение коэффициента регуляризации

энтропии α. Вторая версия изменяет α в процессе обучения модели. Вторая версия считается более предпочтительной, но для простоты и скорости вычислений часто используют первую [3].

Если переписать полученное выше уравнение Беллмана (12) используя определение энтропии (7), то можно получить следующее выражение:

$$Q^{\pi}(s,a) = E_{s'\sim P, a'\sim\pi}\left[R(s,a,s') + \gamma\left(Q^{\pi}(s',a') + \alpha H\big(\pi(\cdot\,|s')\big)\right)\right]$$
$$= E_{s'\sim P, a'\sim\pi}\left[R(s,a,s') + \gamma\big(Q^{\pi}(s',a') - \alpha log\pi(a'|s')\big)\right] \quad (14)$$

Далее считается, что следующие действие a' заново вычисляется на основании существующей политики, для ясности обозначается как $\tilde{a}'$

$$Q^{\pi}(s,a) \approx r + \gamma(Q^{\pi}(s',\tilde{a}') - \alpha log\pi(\tilde{a}'|s') \quad (15)$$

где: $\tilde{a}'\sim\pi(\cdot\,|s')$.

Политика действий должна на каждой итерации состояния среды пытаться максимизировать ожидаемую будущую награду и ожидаемую будущую энтропию. Таким образом, политика действий должна стараться максимизировать $V^{\pi}(s)$:

$$V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a)] + \alpha H(\pi(\cdot\,|s)$$
$$= E_{a\sim\pi}[Q^{\pi}(s,a) - \alpha log\,\pi(a|s)] \quad (16)$$

Для того, чтобы оптимизировать параметры данной функции – ожидаемое действие вычисляется на основании нормального распределения гаусса от случайной величины. Это делается для того, чтобы данное уравнение не имело зависимостей от параметров существующей политики и действие $\widetilde{a_\theta}(s,\xi)$ зависело исключительно от случайных событий.

$$E_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a) - \alpha log\,\pi_\theta(a|s)]$$
$$= E_{\xi\sim N}\left[Q^{\pi_\theta}\big(s,\widetilde{a_\theta}(s,\xi)\big) - \alpha log\,\pi_\theta(\widetilde{a_\theta}(s,\xi)|s)\right] \quad (17)$$

где: ξ – нормально распределенная случайная величина.

Для последнего шага необходимо заменить Q^{π_θ} на одну из функций, которую предполагает алгоритм-критик Q_{φ_1} и Q_{φ_2}. Выбирается минимальная из них. Это еще одна из уникальных особенностей SAC, помимо контроля над

энтропией. В остальном оптимизация политики действий происходит схожим образом с родственным алгоритмами.

$$max_\theta = E_{s\sim D,\xi\sim N}[min_{j=1,2} Q_{\varphi_j}(s, \widetilde{a_\theta}(s,\xi)) - \alpha log\, \pi_\theta(\widetilde{a_\theta}(s,\xi)|s)]$$

где: $E_{s\sim D,\xi\sim N}$ ожидаемое значение при состоянии s взятом случайно из буфера прошлых эпизодов D и ξ являющейся нормально распределенной случайной величиной.

SAC обучает стохастическую политику с контролем энтропии и обучается новому придерживаясь существующей политики. Коэффициент α контролирует компромисс между эффективностью (скоростью обучения) и тем, на сколько сильно модель экспериментирует с возможными действиями во время обучения. Для большинства сред коэффициент должен подбираться индивидуально, чтобы найти наилучшее значения для быстрого и эффективного обучения.

В некоторых реализациях SAC для ускорения работы самые первые действия агента вычисляются без энтропии, и затем только агент начинает экспериментировать с возможными действиями.

4 Сравнительный анализ

В библиотеке Unity ML-agents реализуются оба вышеописанных алгоритма. Однако, они используются для разных целей и имеют свои достоинства и недостатки. Далее в таблице 2 приведены основные различия между двумя алгоритмами.

Таблица 2. Сравнительный анализ PPO и SAC

Параметр сравнения	**PPO**	**SAC**
Выбор действия	PPO выбирает наилучшее действие на основе текущих наблюдений агента в текущий момент времени.	SAC выбирает действие с учётом всех наблюдений и аппроксимаций за все время наблюдения.
Тип действий агентов	Дискретные и непрерывные	Непрерывные. Для работы с дискретными действиями требуется дополнительная настройка.

Параллельные вычисления	Да	Нет
Скорость работы	Быстро	Медленно в начале, быстро в конце
Исследование новых действий	Если алгоритм находит эффективную стратегию, то он продолжит придерживаться ее, тем самым создается риск попасть в так называемый «локальный оптимум»	Алгоритм имеет контроль над энтропией, случайностью действий и, как следствие, исследует больше возможных действий.

PPO работает быстрее и эффективнее на простых средах, в которых мала вероятность нахождения не стандартных способов решения проблемы.

SAC более пригоден для сред имеющих множество элементов и требующих определенных уровень экспериментирования от агента. Несмотря на то, что SAC будет медленнее, качество полученной модели будет лучше.

Библиографический список:

1. Kinds of RL-Algorithms. Электронный ресурс. URL: [https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below]. Дата обращения: 25.04.2020.

2. Proximal policy optimization algorithms. arXiv 2017. J Schulman, F Wolski, P Dhariwal, A Radford, O Klimov.

3. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. T Haarnoja, A Zhou, P Abbeel, S Levine.