1. ЛЕКЦИЯ. Задача обучения с подкреплением

Обучение с подкреплением (reinforcement learning - RL) — это обучение тому, что надо делать, как следует отображать ситуации в действия, чтобы максимизировать некоторый сигнал поощрения (вознаграждения), принимающий числовые значения. Обучаемому не говорят, какое действие следует предпринять, как это имеет место в большинстве подходов к обучению машин. Вместо этого он, пробуя выполнять различные действия, должен найти, какие из них принесут ему наибольшее вознаграждение. В наиболее интересных и важных случаях действия могут влиять не только на вознаграждение, получаемое немедленно, но также и на возникающую ситуацию, а через нее – на все последующие поощрения. Эти две характеристики – поиск методом проб и ошибок, а также отсроченные поощрения — представляют собой две наиболее важные отличительные черты обучения с подкреплением (RL).

Термин подкрепление пришел из поведенческой психологии и обозначает награду или наказание за некоторый результат, зависящий не только от самих принятых решений, но и внешних, не обязательно подконтрольных факторов.

Обучение с подкреплением (RL) отличается от обучения с учителем, которое рассматривается в большинстве современных исследований по обучению машин, статистическому распознаванию образов, искусственным нейронным сетям в частности. Обучение с учителем – это обучение по примерам, предъявляемым некоторой информированной внешней инстанцией. Это важный вид обучения, однако без привлечения дополнительных средств он не пригоден для обучения через взаимодействие. В задачах, решаемых на основе взаимодействия, зачастую непрактично пытаться получать примеры требуемого поведения, которые были бы одновременно корректными и представительными для всех ситуаций, в которых должен действовать агент. На «ничейных» территориях, в ситуациях, когда обучение более всего и нужно, агент должен быть в состоянии учиться только на основе своего собственного опыта.

Модель взаимодействия агента со средой.

Связь с оптимальным управлением.

Математическая модель приходит к похожей формализации следующим образом: для начала, нужно ввести какую-то модель мира (в рамках лапласовского детерминизма, как учения о причинности, закономерности, генетической связи, взаимодействии и обусловленности всех явлений и процессов, происходящих в мире). Тогда можно предположить, что скорости и ускорения всех атомов вселенной задают ее текущее состояние, а изменение

состояния происходит согласно некоторым дифференциальным уравнением:

$$\dot{s} = f(s, t)$$

Коли агент может как-то взаимодействовать со средой или влиять на нее, мы можем промоделировать взаимодействие следующим образом: скажем, что в каждый момент времени t агент в зависимости от текущего состояния среды s выбирает некоторое действие («управление») $a(s, t)$:

$$\dot{s} = f(s, a(s, t), t)$$

Для моделирования награды положим, что в каждый момент времени агент получает наказание или штраф (cost – расходы, тогда любой штраф можно переделать в награду домножением на минус, и наоборот) в объёме $L(s, a(s, t), t)$. Итоговой наградой агента полагаем суммарную награду, то есть интеграл по времени:

$$\begin{cases} -\int L(s, a(s, t), t)dt \to max \\ \dot{s} = f(s, a(s, t), t) \end{cases}$$

За исключением того, что для полной постановки требуется задать ещё начальные и конечные условия, поставленная задача является общей формой задачи оптимального управления. При этом обратим внимание на сделанные при постановке задачи предположения:

• время непрерывно;

• мир детерминирован;

• мир нестационарен (функция f напрямую зависит от времени t);

• модель мира предполагается известной, то есть функция f задана явно в самой постановке задачи.

Принципиально последнее: теория рассматривает способы для заданной системы дифференциальных уравнений поиска оптимальных $a(s, t)$ управлений аналитически. Проводя аналогию с задачей максимизации некоторой, например, дифференцируемой функции $f(x) \to max$, теория оптимального управления ищет необходимые условия решения: например, что в экстремуме функции обязательно градиент равен нулю $\nabla_x f(x) = 0$. Никакого «обучения методом проб и ошибок» здесь не предполагается.

В обучении с подкреплением вместо поиска решения аналитически мы будем строить итеративные методы оптимизации, искать аналоги, например, градиентного спуска. В такой процедуре неизбежно появится цепочка приближений решений: мы начнём с какой-то функции, выбирающей действия, возможно, не очень удачной и не способной набрать большую награду, но с ходом работы алгоритма награда будет оптимизироваться и способ выбора

агентом действий будет улучшаться.

Также RL исходит из других предположений: время дискретно, а среда — стохастична, но стационарна. В частности, в рамках этой теории можно построить алгоритмы для ситуации, когда модель мира агенту неизвестна, и единственный способ поиска решений — обучение на собственном опыте взаимодействия со средой.

Марковская цепь

В обучении с подкреплением мы зададим модель мира следующим образом: будем считать, что существуют некоторые «законы физики», возможно, стохастические, которые определяют следующее состояние среды по предыдущему. При этом предполагается, что в текущем состоянии мира содержится вся необходимая информация для выполнения перехода, или, иначе говоря, выполняется свойство Марковости: процесс зависит только от текущего состояния и не зависит от всей предыдущей истории.

Определение 1: Марковской цепью называется пара $(\mathcal{S}, \mathcal{P})$, где:

• $\mathcal{S}$ — множество состояний.

• $\mathcal{P}$ — вероятности переходов $\{p(s_{t+1}|s_t)|t \in \{0, 1, \dots\}, s_t, s_{t+1} \in \mathcal{S}\}$.

Если дополнительно задать стартовое состояние s_0, можно рассмотреть процесс, заданный марковской цепью. В момент времени $t = 0$ мир находится в состоянии s_0; формируется случайная величина $s_1 \sim p(s_1|s_0)$, и мир переходит в состояние s_1; формируется $s_2 \sim p(s_2|s_1)$, и так далее до бесконечности.

Далее делаем важное предположение, что законы мира не изменяются с течением времени. В аналогии с окружающей нас действительностью это можно интерпретировать примерно, как «физические константы не изменяются со временем».

Определение 2: Марковская цепь называется однородной или стационарной, если вероятности переходов не зависят от времени:

$$\forall t: p(s_{t+1}|s_t) = p(s_1|s_0)$$

По определению, переходы $\mathcal{P}$ стационарных марковских цепей задаются единственным условным распределением $p(s'|s)$. Апостроф $'$ канонично используется для обозначения «следующих» моментов времени, будем активно пользоваться этим обозначением.

Среда

Как и в оптимальном управлении, для моделирования влияния агента на среду в вероятности переходов достаточно добавить зависимость от выбираемых агентом действий. Итак, наша модель среды — это «управляемая» марковская цепь.

Определение 3: Средой называется тройка $(\mathcal{S}, \mathcal{A}, \mathcal{P})$, где:

• $\mathcal{S}$ — пространство состояний, некоторое множество.

• $\mathcal{A}$ — пространство действий, некоторое множество.

• $\mathcal{P}$ — функция переходов или динамика среды: вероятности $p(s'|s,a)$

В таком определении среды заложена марковость (независимость переходов от истории) и стационарность (независимость от времени). Время при этом дискретно, в частности, нет понятия «времени принятия решения агентом»: среда, находясь в состоянии s, ожидает от агента действие $a \in A$, после чего совершает шаг, сэмплируя (формируя выборки) следующее состояние $s' \sim p(s'|s,a)$.

Если, среда также предполагается полностью наблюдаемой: агенту при выборе a_t доступно всё текущее состояние s_t в качестве входа. Иными словами, в рамках данного предположения понятия состояния и наблюдения для нас будут эквивалентны. Будем строить теорию в рамках этого упрощения; если среда не является полностью наблюдаемой, задача существенно усложняется, и необходимо переходить к формализму частично наблюдаемых Марковских процессов принятия решений MDP (Markov Decision Process).

Пример: Пространство состояний – пространство конфигураций Кубика-Рубика. Пространство действий состоит из 12 элементов (нужно выбрать одну из 6 граней, каждую из которых можно повернуть двумя способами). Следующая конфигурация однозначно определяется текущей конфигурацией и действием, соответственно среда Кубика-Рубика детерминирована: задаётся вырожденным распределением, или, что тоже самое, обычной детерминированной функцией $s' = f(s,a)$.

Действия

Нас будут интересовать два вида пространства действий $\mathcal{A}$:

а) конечное, или дискретное пространство действий: $|\mathcal{A}| < +\infty$. Также будем предполагать, что число действий $|A|$ достаточно мало.

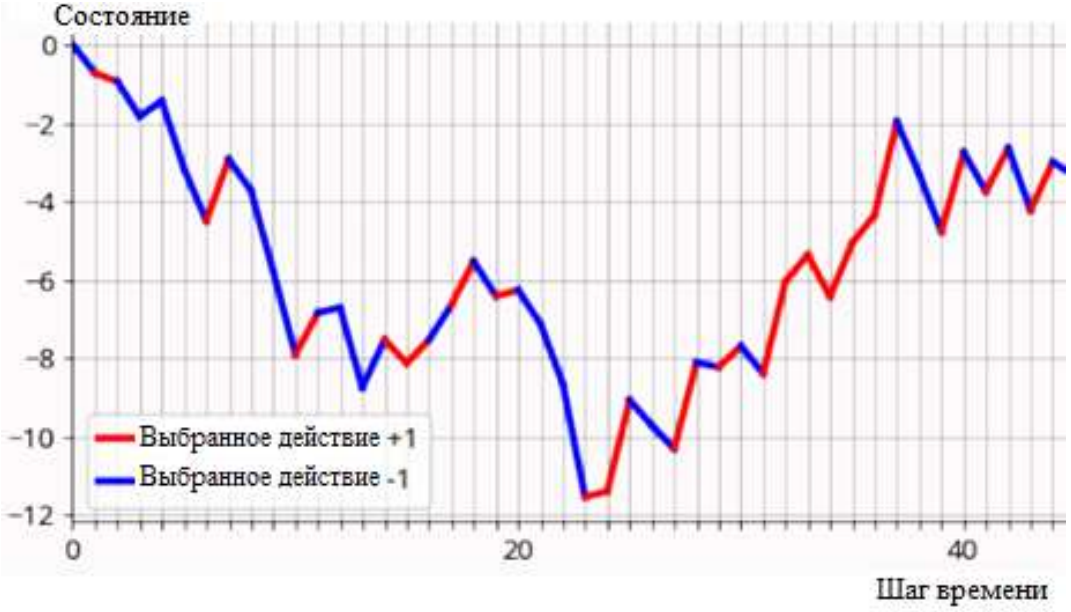
б) непрерывное пространство действий: $A \subseteq [-1,1]^m$. Выбор именно отрезков $[-1,1]$ является не ограничивающем общности распространённым соглашением. Задачи с таким пространством действий также называют задачами непрерывного управления.

В общем случае процесс выбора агентом действия в текущем состоянии может быть стохастичен. Таким образом, объектом поиска будет являться распределение $\pi(a|s), a \in \mathcal{A}, s \in \mathcal{S}$. Заметим, что факт, что если будет достаточно искать стратегию в классе стационарных (не зависящих от времени), вообще говоря, потребует обоснования.

Траектории

Определение 4: Набор $T := (s_0, a_0, s_1, a_1, s_2, a_2, s_3, a_3 \dots)$ называется траекторией.

Пример: Пусть в среде состояния описываются одним вещественным числом, $\mathcal{S} \equiv \mathbb{R}$, у агента есть два действия $\mathcal{A} = \{+1, -1\}$, а следующее состояние определяется как $s' = \mathrm{s} + \mathrm{a} + \varepsilon$, где $\varepsilon \sim N(0,1)$. Начальное состояние полагается равным нулю $s_0 = 0$. Сгенерируем пример траектории для случайной стратегии (вероятность выбора каждого действия равна 0.5):



Поскольку траектории – это случайные величины, которые заданы по постановке задачи конкретным процессом порождения (действия генерируются из некоторой стратегии, состояния — из функции переходов), можно расписать распределение на множестве траекторий:

Определение 5: Для данной среды, политики π и начального состояния $s_0 \in S$ распределение, из которого приходят траектории $\mathcal{T}$, называется распределение траекторий:

$$p(\mathcal{T}) = p(a_0, s_1, a_1 \dots) = \prod_{t \geq 0} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Математические ожидания по траекториям, которые будем обозначать $\mathbb{E}_{\mathcal{T}}$. Под этим подразумевается бесконечная цепочка вложенных математических ожиданий:

$$\mathbb{E}_{\mathcal{T}}(\cdot) := \mathbb{E}_{\pi(a_0|s_0)} \mathbb{E}_{p(s_1|s_0,a_0)} \mathbb{E}_{\pi(a_1|s_1)} \dots (\cdot) \qquad (1.1)$$

Поскольку часто придётся раскладывать эту цепочку, договоримся о следующем сокращении:

$$\mathbb{E}_{\mathcal{T}}(\cdot) := \mathbb{E}_{a_0} \mathbb{E}_{s_1} \mathbb{E}_{a_1} \dots (\cdot)$$

Однако в такой записи стоит помнить, что действия приходят из некоторой

зафиксированной политики π, которая неявно присутствует в выражении. Для напоминания об этом будет, где уместно, использоваться запись $\mathbb{E}_{\mathcal{T}\sim\pi}$

Марковский процесс принятия решений (Markov Decision Process – MDP)

Для того, чтобы сформулировать задачу, необходимо в среде задать агенту цель – некоторый функционал для оптимизации. По сути, марковский процесс принятия решений – это среда плюс награда. Будем пользоваться следующим определением:

Определение 6: Марковский процесс принятия решений (Markov Decision Process, MDP) — это четвёрка $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, где:

• $\mathcal{S}, \mathcal{A}, \mathcal{P}$ – среда.

• $r: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – функция награды.

Сам процесс выглядит следующим образом. Для момента времени $t = 0$ начальное состояние мира полагается s_0; будем считать, оно дано дополнительно и фиксировано. Агент наблюдает всё состояние целиком и выбирает действие $a_0 \in A$. Среда отвечает генерацией награды $r(s_0, a_0)$ и формирует следующее состояние $s1 \sim p(s'|s_0, a_0)$. Агент выбирает действие $a_1 \in \mathcal{A}$, получает награду $r(s_1,\ a_1)$, состояние s_2, и так далее до бесконечности.

Формальное введение MDP (марковского процесса принятия решений) в разных источниках чуть отличается, и из-за различных договорённостей одни и те же утверждения могут выглядеть совсем непохожим образом в силу разных исходных обозначений. Важно, что суть и все основные теоретические результаты остаются неизменными.

Теорема 1 – Эквивалентные определения MDP: Эквивалентно рассматривать MDP, где

• функция награды зависит только от текущего состояния;

• функция награды является стохастической;

• функция награды зависит от тройки (s, a, s');

• переходы и генерация награды задаётся распределением $p(r, s'|s, a)$;

• начальное состояние стохастично и генерируется из некоторого распределения $s_0 \sim p(s_0)$.

Всю стохастику можно «засовывать» в $p(s'|s, a)$. Например, стохастичность начального состояния можно «убрать», создав отдельное начальное состояние, из которого на первом шаге агент вне зависимости от выбранного действия перейдёт в первое по стохастичному правилу.

Покажем, что от самого общего случая (генерации награды и состояний из распределения $p(r, s'|s, a)$ можно перейти к детерминированной функции награды только от текущего состояния. Добавим в описание состояний информацию о последнем действии и последней полученной агентом награде, то есть для каждого возможного (имеющего ненулевую вероятность) перехода (s, a, r, s') размножим s' по числу возможных исходов $p(r|s, a)$ и по числу действий. Тогда можно считать, что на очередном шаге вместо $r(s, a)$ агенту выдаётся $r(s')$, и вся стохастика процесса формально заложена только в функции переходов.

Считать функцию награды, детерминированной удобно, поскольку позволяет не городить по ним мат. ожидания (иначе нужно добавлять образцы наград в определение траекторий). В любом формализме всегда принято считать, что агент сначала получает награду и только затем наблюдает очередное состояние.

Определение 7: MDP называется конечным или табличным, если пространства состояний и действий конечны: $|\mathcal{S}| < \infty, |\mathcal{A}| < \infty$.

Эпизодичность

Во многих случаях процесс взаимодействия агента со средой может при определённых условиях «заканчиваться», причём факт завершения доступен агенту.

Определение 8: Состояние s называется терминальным в MDP, если $\forall a \in \mathcal{A}$:

$$P(s' = s|s, a) = 1 \quad r(s, a) = 0$$

то есть с вероятностью 1 агент не сможет покинуть состояние.

Считается, что на каждом шаге взаимодействия агент дополнительно получает для очередного состояния s значение предиката (соответствующий) $done(s) \in \{0,1\}$, является ли данное состояние терминальным. По сути, после попадания в терминальное состояние дальнейшее взаимодействие бессмысленно (дальнейшие события тривиальны), и, считается, что возможно произвести перезагрузку (ресет) среды в s_0, то есть начать процесс взаимодействия заново.

Введение терминальных состояний именно таким способом позволит всюду в теории писать суммы по времени до бесконечности, не рассматривая отдельно случай завершения за конечное время.

Для агента все терминальные состояния в силу постановки задачи неразличимы, и их описания среда обычно не возвращает (вместо этого на практике она обычно проводит ресет и возвращает s_0 следующего эпизода).

Определение 9: Один цикл процесса от стартового состояния до терминального называется эпизодом.

Продолжительности эпизодов (количество шагов взаимодействия) при этом, конечно, могут различаться от эпизода к эпизоду.

Определение 10: Среда называется эпизодичной, если для любой стратегии процесс взаимодействия гарантированно завершается не более чем за некоторое конечное T^{max} число шагов.

Теорема 2 – Граф эпизодичных сред есть дерево: В эпизодичных средах вероятность оказаться в одном и том же состоянии дважды в течение одного эпизода равна нулю.

Доказательство. Если для некоторого состояния s при некоторой комбинации действий через T шагов агент с вероятностью $p > 0$ вернётся в s, при повторении такой же комбинации действий в силу марковости с вероятностью $p^n > 0$ эпизод будет длиться не менее nT шагов для любого натурального n. Иначе говоря, эпизоды могут быть неограниченно долгими.

Дисконтирование.

Задача заключается в том, чтобы найти стратегию π, максимизирующую среднюю суммарную награду. Формально, явно задан функционал для оптимизации:

$$\mathbb{E}_{\mathcal{T}} \sim \pi \sum_{t \geq 0} r_t \ \rightarrow \ max \tag{1.2}$$

где $r_t := r(s_t, a_t)$ — награда на шаге t.

Необходимо исключить из рассмотрения MDP, где данный функционал может улететь в бесконечность (награда может быть бесконечной, начинаются всякие парадоксы, рассмотрения которых надо избежать) или не существовать вообще. Во-первых, введём ограничение на модуль награды за шаг, подразумевая, что среда не может поощрять или наказывать агента бесконечно сильно:

$$\forall s, a: |r(s, a)| \leq r^{max} \tag{1.3}$$

Чтобы избежать парадоксов, этого условия нам не хватит. Могут возникнуть ситуации, где суммарной награды просто не существует (например, если агент в бесконечном процессе всегда получает +1 на чётных шагах и -1 на

нечётных)

Введём дисконтирование, коэффициент которого традиционно обозначают γ:

Определение 11: Дисконтированной кумулятивной наградой или общий доход для траектории $\mathcal{T}$ с коэффициентом $\gamma \in (0,1]$ называется

$$R(\mathcal{T}) := \sum_{t\geq 0} \gamma^t r_t \quad (1.4)$$

У дисконтирования есть важная интерпретация: полагается, что на каждом шаге с вероятностью $1-\gamma$ взаимодействие обрывается, и итоговым результатом агента является та награда, которую он успел собрать до прерывания. Это даёт приоритет получению награды в ближайшее время перед получением той же награды через некоторое время. Математически смысл дисконтирования, во-первых, в том, чтобы в совокупности с требованием (1.3) гарантировать ограниченность оптимизируемого функционала, а во-вторых, выполнение условий некоторых теоретических результатов, которые явно требуют $\gamma < 1$. В силу последнего, гамму часто рассматривают как часть MDP

Определение 12: Счет (score – скор) стратегии π в данном MDP называется

$$J(\pi) := \mathbb{E}_{\mathcal{T}\sim\pi} R(\mathcal{T}) \quad (1.5)$$

Итак, задачей обучения с подкреплением является оптимизация для заданного MDP средней дисконтированной кумулятивной награды:

$$J(\pi) \rightarrow max$$

Таким образом, везде подразумевается выполнение требования ограниченности награды (1.3), а также или дисконтирования $\gamma < 1$, или эпизодичности среды

Утверждение 1: При сделанных предположениях скор (счет) ограничен.

Доказательство. Если $\gamma < 1$, то по свойству геометрической прогрессии для любых траекторий $\mathcal{T}$:

$$R(\mathcal{T}) = \left|\sum_{t\geq 0} \gamma^t r_t\right| \leq \frac{1}{1-\gamma} r^{max}$$

Если же $\gamma = 1$, но эпизоды гарантированно заканчиваются не более чем за T^{max} шагов, то суммарная награда не превосходит по модулю $T^{max} r^{max}$. Следовательно, скор (счет) как математическое ожидание от ограниченной величины также удовлетворяет этим ограничениям.

Алгоритмы обучения с подкреплением

Условия задачи RL (обучение с подкреплением)

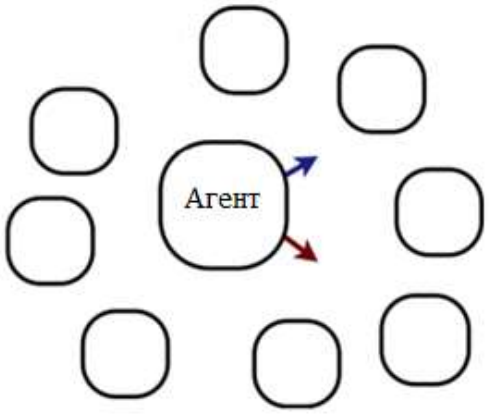
Основной постановкой в обучении с подкреплением является задача

нахождения оптимальной стратегии на основе собственного опыта взаимодействия. Это означает, что алгоритму обучения изначально доступно только:

• вид пространства состояний – количество состояний в нём в случае конечного числа, или размерность пространства $\mathbb{R}^d$ в случае признакового описания состояний.

• вид пространства действий — непрерывное или дискретное. Некоторые алгоритмы будут принципиально способны работать только с одним из этих двух видов.

• взаимодействие со средой, то есть возможность для предоставленной алгоритмом стратегии π генерировать траектории $\mathcal{T} \sim \pi$; иными словами, принципиально доступны только сэмплы (единицы выборки) из распределение траекторий.



Итак, в отличие от обучения с учителем, где датасет «дан алгоритму на вход», здесь агент должен сам собрать данные. Находясь в некотором состоянии, обучающийся агент обязан выбрать ровно одно действие, получить ровно один сэмпл s' и продолжить взаимодействие (накопление опыта — сбор сэмплов) из s'. Собираемые в ходе взаимодействия данные и представляют собой всю доступную агенту информацию для улучшения стратегии.

Определение 13: Пятёрки $T := (s, a, r, s', done)$, где $r := r(s, a)$, $s' \sim p(s'|s, a)$, $done := done(s')$,называются переходами (где done – сделанный).

Таким образом, в RL-алгоритме должна быть прописана стратегия взаимодействия со средой во время обучения, которая может отличаться от «итоговой» стратегии, предназначенной для использования в среде по итогам обучения.

Сравнение с обучением с учителем

Необходимость собирать данные внутри самого алгоритма – одно из ключевых отличий задачи RL от обучения с учителем, где выборка подаётся алгоритму на вход. Здесь же «сигнал» для обучения предоставляется дизайном

функции награды; на практике, чтобы говорить о том, что встретилась задача RL, необходимо задать MDP (марковский процесс принятия решений можно предоставить как среду в виде симулятора или интерфейс для взаимодействия настоящего робота в реальном мире, так и функцию награды). Зачастую если какую-то «репрезентативную» выборку собрать можно, всегда проще и лучше обратиться к обучению с учителем как менее общей постановке задачи.

Утверждение 2: Задача обучения с учителем (с заданной функцией потерь) является частным случаем задачи RL.

Доказательство. Пусть дана обучающая выборка в виде пар (x, y), x – входной объект, y — целевая переменная. Скажем, что начальное состояние есть описание объекта x, случайно сэмплированного из обучающей выборки (начальное состояние будет случайно). Агент выбирает метку класса $\hat{y} \sim \pi(\hat{y}|x)$. После этого он получает награду за шаг в размере $-\ Loss(\hat{y}, y)$ и игра завершается. Оптимизация такой функции награды в среднем будет эквивалентно минимизации функции потерь в обучении с учителем.

Поскольку RL всё-таки предполагает, что в общем случае эпизоды длиннее одного шага, агент будет своими действиями влиять на дальнейшие состояния, которые он встречает. «Управление» марковской цепью куда более существенная часть оптимизации RL алгоритмов, нежели максимизация наград за шаг, именно из этого свойства возникает большинство специфических именно для RL особенностей. И зачастую именно в таких задачах появляется такая непреодолимая для обучения с учителем проблема, как то, что правильный ответ никакому человеку, в общем-то, неизвестен. Не знаем мы обычно оптимальный наилучший ход в шахматах или как правильно поворачивать конечности роботу, чтобы начать ходить.

Да, доступной помощью для алгоритма могут быть данные от эксперта, то есть записи взаимодействия со средой некоторой стратегии (или разных стратегий), не обязательно, вообще говоря, оптимальной. Алгоритм RL, возможно, сможет эти данные как-то использовать, или хотя бы как-либо на них предобучиться. В простейшем случае предобучение выглядит так: если в алгоритме присутствует параметрически заданная стратегия π_θ, можно клонировать поведение, т.е. восстанавливать по парам s, a из всех собранных экспертом траекторий функцию $\mathcal{S} \to \mathcal{A}$ (это обычная задача обучения с учителем), учиться воспроизводить действия эксперта. Задача обучения по примерам её решения около-оптимальным экспертом называется имитационным обучением; однако, если эксперт не оптимален, обученная стратегия вряд ли будет действовать хоть сколько-то лучше. Здесь можно провести прямую

аналогию с задачей обучения с учителем, где верхняя граница качества алгоритма определяется качеством разметки; если разметка зашумлена и содержит ошибки, обучение вряд ли удастся.

В общем же случае мы считаем, что на вход в алгоритм никаких данных эксперта не поступает. Поэтому говорят, что в обучении с подкреплением нам не даны «правильные действия», и, когда мы будем каким-либо образом сводить задачу к задачам регрессии и классификации, нам будет важно обращать внимание на то, как мы «собираем себе разметку» из опыта взаимодействия со средой и какое качество мы можем от этой разметки ожидать. В идеале, с каждым шагом алгоритм сможет собирать себе всё более и более «хорошую» разметку (получать примеры траекторий с всё большей суммарной наградой), за счёт неё выбирать всё более и более оптимальные действия, и как бы «вытягивать сам себя из болота».

Концепция model-free (без модели) алгоритмов.

Ещё одним возможным существенным изменением сеттинга (среда, в которой происходит действие) задачи является наличие у агента прямого доступа к функции переходов $p(s'|s,a)$ и функции награды.

Определение 14: Будем говорить, что у агента есть симулятор или «доступ к функции переходов», если он знает функцию награды и может в любой момент процесса обучения сэмплировать произвольное число сэмплов из $p(s'|s,a)$ для любого набора пар s, a.

Симулятор – по сути копия среды, которую агент во время обучения может откатывать к произвольному состоянию. Симулятор позволяет агенту строить свою стратегию при помощи планирования – рассмотрения различных вероятных версий предстоящего будущего и использования их для текущего выбора действия. При использовании планирования в идеальном симуляторе никакого процесса непосредственного обучения в алгоритме может не быть, поскольку если на руках есть симулятор, то данные собирать не нужно: можно из симулятора сколько захотим данных получить.

Примером задач, в которых у алгоритма есть симулятор, являются пятнашки или кубик-рубик. Любые задачи, в которых среда реализована виртуально, можно рассматривать как задачи, где у агента есть симулятор. Примером задач, в которых у алгоритма принципиально нет симулятора, являются любые задачи реальной робототехники. Важно, что даже если окружение реального робота симулируется виртуально, такая симуляция неточна — отличается от реального мира. В таких ситуациях можно говорить, что имеется неидеальный симулятор.

По умолчанию всегда считается, что доступа к динамике среды нет, и единственное, что предоставляется алгоритму – среда, с которой возможно взаимодействовать. Можно ли свести такую задачу к планированию? В принципе, алгоритм может пытаться обучать себе подобный симулятор – строить генеративную модель, по s, a выдающую $s', r(s,a), done(s')$ – и сводить таким образом задачу к планированию. Приближение тогда, естественно, будет неидеальным, да и обучение подобного симулятора сопряжено с рядом других нюансов. Например, в сложных средах в описании состояний может хранится колоссальное количество информации, и построение моделей, предсказывающих будущее, может оказаться вычислительно неподъёмной и неоправданно дорогой задачей.

Одна из фундаментальных парадигм обучения с подкреплением, вероятно, столь же важная, как парадигма сквозного (end-to-end) обучения для глубокого обучения — идея model-free обучения. Давайте не будем учить динамику среды и перебирать потенциальные варианты будущего для поиска хороших действий, а выучим напрямую связь между текущим состоянием и оптимальными действиями.

Определение 15: Алгоритм RL классифицируется как model-free, если он не использует и не пытается выучить модель динамики среды $p(s'|s,a)$.

On-policy vs Off-policy (включенная политика против выключенной)

В model-free алгоритмах сбор данных становится важной составной частью: определяя политику взаимодействия со средой, мы влияем на то, для каких состояний s, а мы получим сэмпл s' из функции переходов. Собираемые данные — траектории — алгоритм может запоминать, например, в памяти. Но не каждый алгоритм RL сможет пользоваться такими сохранёнными данными, и поэтому возникает ещё одна важная классификация RL алгоритмов.

Определение 16: Алгоритм RL называется off-policy, если он может использовать для обучения опыт взаимодействия произвольной стратегии.

Определение 17: Алгоритм RL называется on-policy, если для очередной итерации алгоритма ему требуется опыт взаимодействия некоторой конкретной, предоставляемой самим алгоритмом, стратегии.

Некоторое пояснение названия этих терминов: обычно в нашем алгоритме явно или неявно будет присутствовать некоторая «текущая», «наилучшая» найденная стратегия π: та самая целевая политика (target policy), которую алгоритм выдаст в качестве ответа, если его работу прервать. Если алгоритм умеет улучшать её (проводить очередную итерацию обучения), используя сэмплы любой другой произвольной стратегии μ, то мы проводим «off-policy»

обучение: обучаем политику «не по ней же самой». On-policy алгоритмам будет необходимо «отправлять в среду конкретную стратегию», поскольку они будут способны улучшать стратегию π лишь по сэмплам из неё же самой, будут «привязаны к ней»; это существенное ограничение. Другими словами, если обучение с подкреплением — обучение на основе опыта, то on-policy алгоритмы обучаются на своих ошибках, когда off-policy алгоритмы могут учиться как на своих, так и на чужих ошибках.

Off-policy алгоритм должен уметь проводить очередной шаг обучения на произвольных траекториях, сгенерированных произвольными (возможно, разными, возможно, неоптимальными) стратегиями. Понятие принципиально важно тем, что алгоритм может потенциально переиспользовать траектории, полученные старой версией стратегии со сколь угодно давних итераций. Если алгоритм может переиспользовать опыт, но с ограничениями (например, только с недавних итераций, или только из наилучших траекторий), то мы всё равно будем относить его к on-policy, поскольку для каждой новой итерации алгоритма нужно будет снова собирать сколько-то данных. Это не означает, что для on-policy алгоритма совсем бесполезны данные от (возможно, неоптимального) эксперта; почти всегда можно придумать какую-нибудь эвристику, как воспользоваться ими для хотя бы инициализации (при помощи того же клонирования поведения). Важно, что off-policy алгоритм сможет на данных произвольного эксперта провести «полное» обучение, то есть условно сойтись к оптимуму при достаточном объёме и разнообразии экспертной информации, не потребовав вообще никакого дополнительного взаимодействия со средой.

Классификация RL-алгоритмов

При рассмотрении алгоритмов RL мы начнём с рассмотрения именно model-free алгоритмов. Их часто делят на следующие подходы:

• мета-эвристики никак не используют внутреннюю структуру взаимодействия среды и агента, и рассматривают задачу максимизации счета стратегии $J(\pi)$ (1.5) как задачу «black box (черного ящика) оптимизации»: можно примерно оценивать, чему равно значение функционала для разных стратегий, а структура задачи — формализм MDP — не используется;

• value-based (основанные на цене) алгоритмы получают оптимальную стратегию неявно через теорию оценочных функций.

• policy gradient (градиент политики) алгоритмы максимизируют счет стратегии $J(\pi)$, используя оценки градиента функционала по параметрам стратегии; можем помочь процессу оптимизации, правильно воспользовавшись оценочными функциями.

Критерии оценки RL-алгоритмов

При оценивании алгоритмов принципиально соотношение трёх критериев:

• производительность (performance): Монте-Карло оценка значения $J(\pi)$;

• реальное время работы (wall-clock time), потребовавшееся алгоритму для достижения такого результата (в полной аналогии с классическими методами оптимизации);

• эффективность выборки (sample efficiency): количество сэмплов (или шагов) взаимодействия со средой, потребовавшихся алгоритму. Этот фактор может быть ключевым, если взаимодействие со средой дорого (например, обучается реальный робот);

Конечно, на практике мы хотим получить как можно больший производительность при наименьших затратах ресурсов. Время работы алгоритма реального время работы (wall-clock time) и эффективность по сэмплам связаны между собой, но их следует различать в силу того, что в разных средах сбор данных – проведение одного шага взаимодействия в среде – может быть, как относительно дешёвым, так и очень дорогим. Например, если мы говорим о реальных роботах, то один шаг взаимодействия со средой – сверхдорогая операция, по сравнению с которыми любые вычисления на компьютере можно считать очень дешёвыми. Тогда нам выгодно использовать алгоритм, который максимально эффективен по сэмплам. Если же среда задана симулятором, а у нас ещё и есть куча серверов, чтобы параллельно запустить много таких симуляторов, то сбор данных для нас становится дешёвой процедурой. Тогда выгодно не гнаться за эффективностью выборки и выбирать вычислительно дешёвые алгоритмы.

Очень условно классы RL алгоритмов располагаются по эффективности выборки в следующем порядке: самыми неэффективными по требуемым объёмам данных алгоритмами являются мета-эвристики. Зато в них практически не будет никаких вычислений: очень условно, на каждое обновление весов модели будет приходиться сбор нескольких сотен переходов в среде. Их будет иметь смысл применять, если есть возможность параллельно собирать много данных.

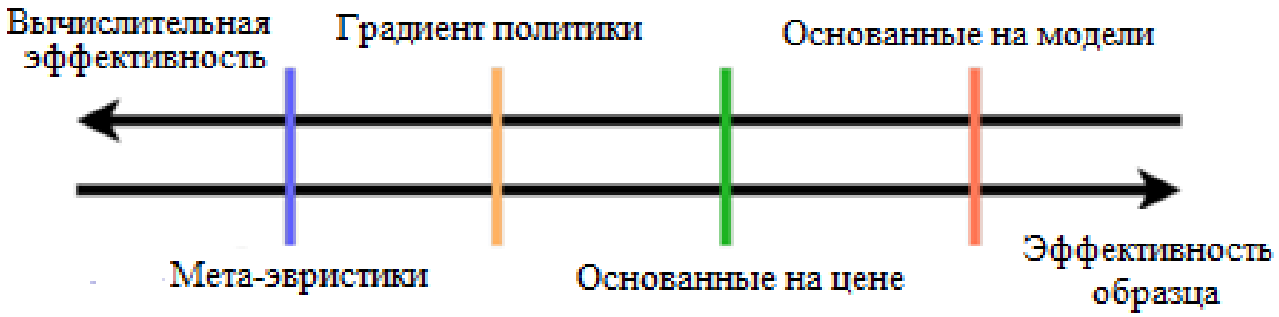
Далее, в градиентном политики подходе мы будем работать в on-policy режиме: текущая, целевая, политика будет отправляться в среду, собирать сколько-то данных (например, мини-батч (батч — это составная часть) переходов, условно проводить порядка 32 – 64 шагов в среде) и затем делать одно обновление модели. Очень приближённо и с массой условностей говорят, что эффективность по сэмплам будет на порядок выше, чем у эволюционных

алгоритмов, за счёт проведения на порядок большего количества вычислений: на одно обновление весов приходится сбор 30-60 переходов.

В value-based (основанные на цене) подходе за счёт off-policy режима работа можно будет контролировать соотношение числа собираемых переходов к количеству обновлений весов, но по умолчанию обычно полагают, что алгоритм собирает 1 переход и проводит 1 итерацию обучения.

Наконец, model-based (основанный на модели) вычислительно страшно тяжеловесные алгоритмы, но за счёт этого можно попытаться добиться максимальной эффективности по сэмплам.

Отразить четыре класса алгоритмов можно на условном рисунке:



Дизайн функции награды

Алгоритмы RL предполагают, что награда, как и среда, заданы, «поданы на вход». Понятно, что если для практического применения обучения с учителем часто является необходимость размечивать данные – «предоставлять обучающий сигнал» – то в RL необходимо аккуратно описать задачу при помощи функции награды

Определение 18: «Reward hypothesis – Гипотеза вознаграждения»: любую интеллектуальную задачу можно задать (определить) при помощи функции награды.

В обучении с подкреплением принято полагать эту гипотезу истинной. Но так ли это? RL будет оптимизировать ту награду, которую ему предоставят, и дизайн функции награды в ряде практических задач оказывается проблемой. Общее практическое правило звучит так: хорошая функция награды поощряет агента за достигнутые результаты, а не за то, каким способом агент этих результатов добивается. Это логично: если вдруг дизайнеру награды кажется, что он знает, как решать задачу, то вероятно, RL не особо и нужен.

Есть, однако, один способ, как можно функцию награды модифицировать, не изменяя решаемую задачу, то есть эквивалентным образом. К сигналу «из среды» можно что-то добавлять при условии, что на следующем шаге это что-то обязательно будет вычтено. Другими словами, можно применять трюк, который

называется телескопирующая сумма (telescoping sums): для любой последовательности a_t, т. ч. $\lim_{t\to\infty} a_t = 0$

$$\sum_{t\geq 0}^{\infty}(a_{t+1} - a_t) = -a_0 \quad (1.6)$$

Определение 19: Пусть дана некоторая функция $\Phi(s): S \to R$, которую назовём потенциалом, и которая удовлетворяет двум требованиям: она ограничена и равна нулю в терминальных состояниях. Будем говорить, что мы проводим формирование вознаграждения (reward shaping) при помощи потенциала $\Phi(s)$, если мы заменяем функцию награды по следующей формуле:

$$r^{\text{new}}(\text{s}, \text{a}, \text{s}') := \text{r}(\text{s}, \text{a}) + \gamma\Phi(\text{s}') - \Phi(\text{s})$$

Формирование вознаграждения позволяет поменять награду, «не ломая» задачу. Это может быть способом борьбы с разреженной функцией (имеют нулевые значения) награды, если удаётся придумать какой-нибудь хороший потенциал. Конечно, для этого нужно что-то знать о самой задаче, и на практике формирование вознаграждения – это инструмент внесения каких-то априорных знаний, дизайна функции награды.