Дата последней сборки документа:
15 апреля 2022 г. в 06:45

Общие и специальные вопросы оптимизации

Подвойский А.О.

Содержание

1. Полезные ссылки

https://mujoco.org/
Элегантный фреймворк для RL (NB!): https://github.com/thu-ml/tianshou
Фреймворк JORDLY: https://github.com/kakaoenterprise/JORLDY
Курс по прикладному RL: https://github.com/yandexdataschool/Practical_RL
Курс Сергея Левина: https://www.youtube.com/playlist?list=PL_iWQOsE6TfURIIhCrlt-wj9ByIVpbfGc

2. Задача обучения с подкреплением

Под желаемым результатом мы будем понимать максимизацию некоторой скалярной величины, называемой *наградой* (reward). Интеллектуальную сущность (систему/робота/алгоритм), принимающую решения, будем называть *агентом* (agent).

Агент взаимодействует с миром или *средой* (environment), которая задается зависящем от времени *состоянием* (state). Агенту в каждый момент времени в общем случае доступно только некоторое *наблюдение* (observation) текущего состояния мира. Сам агент задает процедуру выбора *действия* (action) по доступным наблюдениям; эту процедуру далее будем называть *стратегией* или *политикой* (policy). Процесс взаимодействия агента и среды задается *динамикой среды* (world dynamics), определяющей правила смены состояний среды во времени и генерации награды.

Буквы s, a, r зарезервируем для состояний, действий и наград соответственно. Буквой t будем обозначать время в процессе взаимодействия.

2.1. Модель взаимодействия агента со средой

2.1.1. Марковская цепь

Свойство Марковости: процесс зависит только от текущего состояния и не зависит от всей предыдущей истории.

Марковской цепью (Markov chain) называется пара $(\mathcal{S}, \mathcal{P})$, где $\mathcal{S}$ – множество состояний, а $\mathcal{P}$ – вероятности переходов $\{p(s_{t+1}|s_t)|t \in \{0, 1, \dots, \}, s_t, s_{t+1} \in \mathcal{S}\}$.

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.

2.1.2. Среда / Окружение

Средой (environment) называется тройка $\mathcal{S}, \mathcal{A}, \mathcal{P}$, где $\mathcal{S}$ – множество (пространство) состояний, $\mathcal{A}$ – множество (пространство) действий, $\mathcal{P}$ – функция переходов или динамика среды: вероятности $p(s'|s, a)$.

Замечание

Если среда не является полностью наблюдаемой, задача существенно усложняется, и необходимо переходить к формализму частично наблюдаемых MDP (partially observable MDP, PoMDP)

2.1.3. Действия

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

- конечное, или *дискретное пространство действий* (discrete action space): $|\mathcal{A}| < +\infty$. Мы также будем предполагать, что число действий $|\mathcal{A}|$ достаточно мало,
- *непрерывное пространство действий* (continuous domain): $\mathcal{A} \subseteq [-1, 1]^m$. Выбор именно отрезков $[-1, 1]$ не является ограничивающем общности распространенным соглашением. Задачи с таким пространством действий также называют задачами непрерывного управления.

2.1.4. Траектории

Набор $\mathcal{T} := (s_0, a_0, s_1, a_1, s_2, a_2, \ldots)$ называется *траекторией*.

Для данной среды, политики π и начального состояния $s_0 \in \mathcal{S}$ распределение, из которого приходят траектории $\mathcal{T}$, называется *trajectory distibution*

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

Мы часто будем рассматривать *мат. ожидание по траекториям*, которые будем обозначать $\mathbb{E}_{\mathcal{T}}$. Под этим подразумевается бесконечная цепочка вложенных мат. ожиданий

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

Поскольку часто придется раскладывать эту цепочку, договоримся о следующем сокращении

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

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 $\mathbb{E}_{\mathcal{T} \sim \pi}$.

2.1.5. Марковский процесс принятия решений (MDP)

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

Мы будем пользоваться следующим определением: *Марковский процесс принятия решений* (Markov Decision Process, MDP) – это четверка $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, где

- $\mathcal{S}, \mathcal{A}, \mathcal{P}$ – среда,
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – *функция вознаграждения* (reward function).

Сам процесс выглядит следующим образом. Для момента времени $t = 0$ начальное состояние мира полагается s_0. Агент наблюдает все состояние целиком и выбирает действие $a_0 \in \mathcal{A}$. Среда отвечает генерацией награды $r(s_0, a_0)$ и сэмплирует следующее состояние $s_1 \sim p(s'|s_0, a_0)$. Агент выбирает $a_1 \in \mathcal{A}$, получает вознаграждение $r(s_1, a_1)$, состояние s_2, и так далее до бесконечности.

Замечание

Считать функцию вознаграждения *детерминированной* удобно, поскольку позволяет не городить по ним мат. ожидания (иначе нужно добавлять сэмплы наград в определение траектории)

В любом формализме всегда принято считать, что агент сначала получает награду и только затем наблюдает очередное состояние.

Марковский процесс принятия решений (MDP) называется *конечным* (finite MDP) или табличным, если пространства состояний и действий конечны: $|\mathcal{S}| < \infty$, $|\mathcal{A} < \infty|$.

2.1.6. Эпизодичность

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

Состояние s называется *терминальным* (terminal) в MDP, если $\forall a \in \mathcal{A}$:

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

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 $done(s) \in \{0, 1\}$, является ли данное состояние терминальным. По сути, после попадания в терминальное состояние дальнейшее взаимодействие бессмысленно (дальнейшие события тривиальны), и, считается, что возможно произвести *reset* среды в s_0, то есть начать процесс взаимодействия заново.

Один цикл процесса от стартового состояния до терминального называется *эпизодом* (episode). Продолжительности эпизодов (количество шагов взаимодействия) при этом, конечно, могут различаться от эпизода к эпизоду.

Среда называется *эпизодичной* (episodic), если для любой стратегии процесс взаимодействия гарантированно завершается не более чем за некоторое конечное $T^{\max}$ число шагов.

Замечание

В эпизодичных средах вероятность оказаться в одном и том же состоянии дважды равна нулю

На практике, в средах обычно существуют терминальные состояния, но нет гарантии завершения эпизодов за ограниченное число шагов. Это лечат при помощи таймера – жесткого ограничения, требующего по истечении $T^{\max}$ шагов проводить в среде ресет.

2.1.7. Дисконтирование

Наша задача заключается в том, чтобы найти стратегию π, максимизирующую среднюю суммарную награду. Формально, нам явно задан функционал для оптимизации

$$\mathbb{E}_{\mathcal{T}\sim\pi} \sum_{t\geqslant 0} r_t \to \max_{\pi},$$

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

Мы хотим исключить из рассмотрения MDP, где данный функционал может улететь в бесконечность или не существовать вообще. Во-первых, введем ограничение на модуль награды за шаг, подразумевая, что среда не может поощрять или наказывать агента бесконечно сильно

$$\forall s, a : |r(s, a)| \leqslant r^{\max} \tag{1}$$

Чтобы избежать парадоксов, этого условия нам не хватит. Введем *дисконтирование* (discounting), коэффициент которого традиционно обозначают γ.

Дисконтированной кумулятивной наградой (discounted cumulative reward) или `total return` для траектории $\mathcal{T}$ с коэффициентом $\gamma \in (0, 1]$ называется

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

У дисконтирования есть важная интерпретация [1, стр. 13]: мы полагаем, что на каждом шаге с вероятностью $1 - \gamma$ взаимодействие обрывается, и итоговым результатом агента является та награда, которую он успел собрать до прерывания. Это дает приоритет получению награды

в *ближайшее* время перед получением той же награды через некоторое время. Математически смысл дисконтирования в том, чтобы в совокупности с требованием (1) гарантировать ограниченность оптимизируемого функционала.

Скором (score или performance) стратегии π в данном MDP называется [1, стр. 13]

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

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

$$J(\pi) \to \max_{\pi}$$

То есть, другими словами, мы ищем наиболее эффективную стратегию π, которая максимизирует среднее накопленное вознаграждение.

2.2. Алгоритмы обучения с подкреплением

Классификация наиболее популярных алгоритмов приведена на

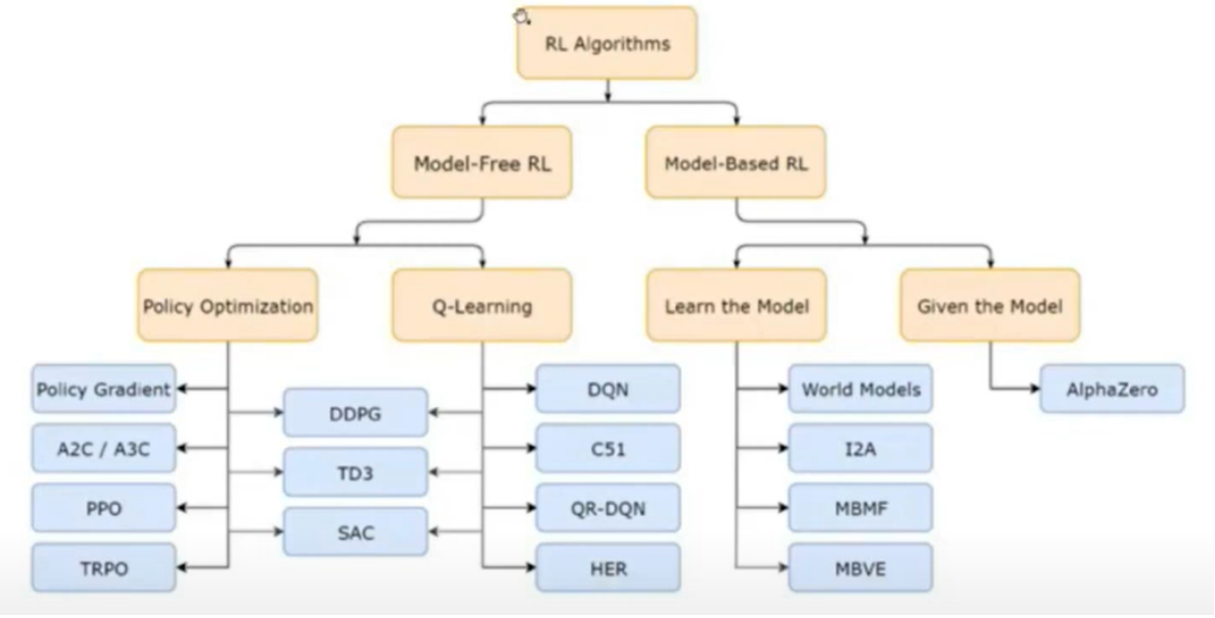

Рис. 1. Классификация алгоритмов обучения с подкреплением

2.2.1. Условия задачи RL

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 нем 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 предоставленой алгоритмом стратегии π генерировать траектории $\mathcal{T} \sim \pi$; иными словами, принципиально доступны только сэмплы из trajectory distribution.

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.

Пятерки $\mathbb{T} := (s, a, r, s', done)$, где $r := r(s, a)$, $s' \sim p(s'|s, a)$, $done := done(s')$.

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

Необходимость собирать данные внутри самого алгоритма – одно из ключевых отличий задачи RL от обучения с учитилем, где выборка подается алгоритму на вход.

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. Алгоритмы 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* (behavior cloning), 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 ее решения около-оптимальным экспертом называется *имитационным обучением* (imitation learning). 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.

2.2.2. Концепция model-free алгоритмов

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

2.2.3. On-policy vs Off-policy

Алгоритм RL называется *off-policy*, если он может использовать для обучения опыт взамодействия произвольной стратегии.

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

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 итерации.

Если алгоритм может переиспользовать опыт, но с ограничениями (например, только с недавних итераций, или только из наилучших траекторий), то мы все равно будем относить его к on-

policy, поскольку для каждой новой итерации алгоритма нужно будет снова собирать сколько-то данных.

Важно, что off-policy алгоритм сможет на данных произвольного эксперта провести «полное» обучение, то есть условно сойтись к оптимуму при достаточном объеме и разнообразии экспертной информации, не потребовав вообще никакого дополнительного взаимодействия со средой.

2.2.4. Классификация RL-алгоритмов

Model-free алгоритмы часто делят на следующие подходы:

- мета-эвристики,
- value-based,
- policy gradient.

2.2.5. Сложности задачи RL

Проблема застревания в локальных оптимумах приходит напрямую из методов оптимизации. Другие проблемы куда более характерны именно для RL. Допустим, агент совершает какое-то действие, которое запускает в среде некоторый процесс. Процесс протекает сам по себе без какого-либо дальнейшего вмешательства агента и завершается через много шагов, приводя к награде. Это *прблема отложенного сигнала* (delayed reward) – среда дает фидбэк агенту спустя какое-то (вообще говоря, неограниченно длительное) время.

Поскольку функция награды может быть произвольная, довольно типично, когда сигнал от среды – неконстантная награда за шаг – приходит очень редко. Это проблема *разреженной награды* (sparse reward).

Еще одна очень важная проблема – *дилемма исследования-использования* (exploration-exploitation trade-off).

2.2.6. Дизайн функции награды

И есть еще одна, вероятно, главная проблема. Откуда берется награда? Алгоритмы RL предполагают, что награда, как и среда, заданы, «поданы на вход», и эту проблему наши алгоритмы обучения, в отличие от предыдущих, решать идеологически не должны.

Но понятно, что если для практического применения обучения с учителем боттлнеком часто является необходимость размечать данные – «предоставлять обучающий сигнал» – то в RL необходимо аккуратно описать задачу при помощи функции награды.

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 решить задачу, то вероятно, RL не особо и нужен. К сожалению, такая «хорошая» функция награды обычно разреженная.

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

$$r^{new}(s, a, s') := r(s, a) + \gamma\Phi(s') - \Phi(s) \tag{2}$$

На практике reward shaping – это инструмент внесения каких-то априорных знаний. Проведение любого reward shaping по формуле (2) не меняет задачи.

Список иллюстраций

Список литературы

1. *Иванов* Конспект по обучению с подкреплением, 2022
2. *Пантлеев А. В., Летова Т.А,* Методы оптимизации в примерах и задачах. – СПб.: Издательство «Лань», 2015. – 512 с.
3. *Вороноцова Е.А.* Выпуклая оптимизация. – М.: МФТИ, 2021. – 364 с.
4. *Бурков А.* Машинное обучение без лишних слов. – СПб.: Питер, 2020. – 192 с.
5. *Бизли Д.* Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с.