Шпаргалка по обучению с подкреплением / Концепции Cheatsheet (XeLaTeX)

Краткий справочник

April 2, 2025

Contents

1 RL: Основная Идея

Что такое Reinforcement Learning (RL)?

Обучение с подкреплением (RL) — это область машинного обучения, где **агент** учится принимать решения, взаимодействуя со **средой**. Цель агента — максимизировать суммарную **награду**, получаемую от среды за свои действия. Обучение происходит методом проб и ошибок.
Ключевые отличия:

- **От Supervised Learning:** В RL нет готовых пар "вход-правильный выход". Агент сам должен выяснить, какие действия ведут к лучшим результатам, ориентируясь только на сигнал награды (часто отложенный во времени).
- **От Unsupervised Learning:** В RL есть явный сигнал обратной связи — **награда**, который направляет обучение, в то время как в Unsupervised Learning основной целью является поиск структуры в данных без какой-либо явной обратной связи.

Аналогия: Подумай о дрессировке щенка. Щенок (агент) выполняет команды (действия) в комнате (среда). Если он выполняет команду правильно (например, садится), ты даешь ему лакомство (положительная награда). Если делает что-то не то (например, грызет тапок), ты можешь сказать "фу!" (отрицательная награда или ее отсутствие). Щенок постепенно учится выполнять действия, которые приносят больше "лакомств".

2 Ключевые Компоненты и Терминология

Основные понятия RL (Важно для собеседования!)

- **Агент (Agent):** Сущность, которая обучается и принимает решения (например, игрок в игре, робот, система рекомендаций).
- **Среда (Environment):** Внешний мир, с которым взаимодействует агент (например, игровое поле, комната для робота, веб-сайт для рекомендаций). Среда реагирует на действия агента, изменяет свое состояние и выдает награду.
- **Состояние (State, S):** Конкретная ситуация или конфигурация среды, которую наблюдает агент в данный момент времени (например, позиция фигур на доске, показания датчиков робота, история просмотров пользователя). Множество всех возможных состояний называется пространством состояний.
- **Действие (Action, A):** Выбор, который агент может сделать в данном состоянии (например, ход фигурой, движение мотора робота, показ определенного товара). Множество всех доступных действий (в данном состоянии или вообще) называется пространством действий.
- **Награда (Reward, R):** Числовой сигнал, получаемый агентом от среды после выполнения действия a в состоянии s. Показывает, насколько "хорошим" было это действие *сиюминутно*. Цель агента — максимизировать *суммарную* награду в долгосрочной перспективе, а не только немедленную.
- **Политика (Policy, π):** Стратегия агента, определяющая его поведение. Это отображение состояний в действия (детерминированная политика $\pi : S \to A$) или в распределение вероятностей над действиями (стохастическая политика $\pi(a|s) = P(A_t = a|S_t = s)$). **Именно оптимальную политику мы и стремимся найти в RL.**
- **Ценность Состояния (State-Value Function, $V^\pi(s)$):** Проще говоря, V-функция показывает, насколько "хорошо" находиться в состоянии s в долгосрочной перспективе, если следовать политике π. Формально: это ожидаемая суммарная дисконтированная награда, начиная из состояния s и далее следуя политике π. $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]$.
- **Ценность Действия (Action-Value Function, Q-function, $Q^\pi(s,a)$):** Проще говоря, Q-функция показывает, насколько "хорошо" выполнить действие a в состоянии s и затем действовать согласно политике π. Формально: это ожидаемая суммарная дисконтированная награда, начиная из состояния s, совершив действие a, и далее следуя политике π. $Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$. **Q-функция часто используется для выбора лучшего действия**, даже если сама политика не задана явно (выбираем действие a с максимальным $Q(s,a)$).
- **Эпизод (Episode):** Полная последовательность взаимодействий агента со средой от начального состояния до терминального (конечного) состояния. Актуально для *эпизодических задач* (игры, лабиринты). В *непрерывных задачах* (управление процессом) понятия эпизода может не быть, и для обеспечения сходимости суммарной награды часто необходимо использовать дисконтирование ($\gamma < 1$).

3 Цель Обучения и Дисконтирование

Максимизация Накопленной Награды

Основная цель RL — найти такую политику π, которая максимизирует ожидаемую **суммарную дисконтированную награду**. Мы не просто хотим получить большую награду сейчас, а максимизировать сумму наград на протяжении всего времени (эпизода или бесконечного горизонта).
Дисконтирование (Discount Factor, γ): Это параметр (число от 0 до 1, обычно

близкое к 1, например, 0.99), который определяет важность будущих наград по сравнению с немедленными. Суммарная дисконтированная награда (Return, G_t) в момент времени t рассчитывается как:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Зачем нужно дисконтирование ($\gamma < 1$)?

1. **Математическая необходимость:** Для задач с бесконечным горизонтом (без конечного состояния) сумма наград может расходиться. Дисконтирование гарантирует сходимость ряда, если награды ограничены.
2. **Интуитивная целесообразность:** Часто награды, полученные раньше, более важны и предсказуемы, чем награды, полученные далеко в будущем. Мы предпочитаем получить 100 долларов сегодня, а не через 10 лет.

Если $\gamma = 0$, агент становится "близоруким" и учитывает только немедленную награду. Если $\gamma = 1$, будущие награды учитываются так же, как и немедленные (используется для эпизодических задач, где сумма конечна).

4 Дилемма Exploration vs. Exploitation

Исследование или Использование?

Одна из фундаментальных проблем в RL — это баланс между исследованием и эксплуатацией.

- **Exploitation (Эксплуатация):** Использовать текущие знания о среде для выбора действий, которые, как известно агенту, приносят наибольшую награду. Пример: ходить в уже знакомый ресторан, который точно нравится.
- **Exploration (Исследование):** Пробовать новые, ранее не изведанные (или мало изведанные) действия, чтобы получить больше информации о среде и, возможно, найти лучший путь/стратегию, чем известный сейчас. Пример: попробовать новый ресторан, который может оказаться лучше (или хуже) знакомого.

Суть дилеммы:

- Слишком много эксплуатации $\Longrightarrow$ агент может "застрять" в локальном оптимуме, не найдя глобально наилучшей стратегии.
- Слишком много исследования $\Longrightarrow$ агент будет тратить много времени на неоптимальные действия, получая меньше награды в процессе обучения.

Необходим умный компромисс.
Простая стратегия: ϵ-greedy (эпсилон-жадная): С вероятностью $(1 - \epsilon)$ агент выбирает действие, которое считается лучшим согласно текущей оценке (например, с максимальным Q-значением) — это *exploitation*. С небольшой вероятностью ϵ (эпсилон, например, 0.1) агент выбирает случайное действие из всех доступных — это *exploration*. Часто ϵ уменьшают со временем: в начале обучения исследуем больше, затем — больше эксплуатируем накопленные знания.

5 Основные Подходы к RL (Обзорно)

Классификация Методов RL

Существует несколько способов классифицировать RL алгоритмы.
1. Model-Based vs. Model-Free:

- **Model-Based (На основе модели):** Агент сначала пытается построить *модель среды* (то есть выучить функции перехода $P(s'|s,a)$ и награды $R(s,a)$). Затем использует эту модель для *планирования* оптимальных действий (например, с помощью динамического программирования или поиска по дереву).
- **Model-Free (Без модели):** Агент учит *политику* или *функцию ценности* напрямую из опыта (взаимодействий со средой), не строя явной модели среды. Этот подход часто более применим к сложным задачам, где построение точной модели затруднительно или невозможно. **Большинство известных RL алгоритмов (Q-learning, DQN, Policy Gradients, A3C) — model-free.**

2. Value-Based vs. Policy-Based (и Actor-Critic): (Классификация для Model-Free методов)

- **Value-Based (На основе ценности):**
  - *Идея:* Обучить функцию ценности (обычно Q-функцию $Q(s,a)$).
  - *Политика:* Неявная (implicit). Агент выбирает действие, максимизирующее выученную Q-функцию в текущем состоянии: $\pi(s) = \arg\max_a Q(s,a)$.
  - *Пример:* ***Q-Learning***. Это классический **off-policy** (внеполитиковый) алгоритм. Off-policy означает, что алгоритм может обучаться оптимальной Q-функции (и, соответственно, оптимальной политике), используя данные, собранные при следовании другой, возможно, неоптимальной или исследовательской политике (например, ϵ-greedy). Он итеративно обновляет оценку Q-функции для пар (состояние, действие), используя полученный опыт (s, a, r, s'):

    $$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$$

    Смысл формулы: Новая оценка $Q(s,a)$ сдвигается в сторону "целевого значения" (target): $r+\gamma \max_{a'} Q(s',a')$. Это целевое значение состоит из немедленной награды r и максимальной ожидаемой будущей награды из следующего состояния s' (оцененной по текущей Q-функции). α (alpha) — это *скорость обучения* (learning rate).
  - *Другой пример:* Deep Q-Network (DQN) — использует нейронную сеть для аппроксимации Q-функции в задачах с большим пространством состояний.
- **Policy-Based (На основе политики):**
  - *Идея:* Напрямую параметризовать политику $\pi(a|s;\theta)$ (например, нейронной сетью с параметрами θ) и оптимизировать эти параметры для максимизации ожидаемой награды.
  - *Подход:* Обычно используются методы градиентного подъема (**Policy Gradients**), которые корректируют параметры θ в направлении, *максимизирующем ожидаемую суммарную дисконтированную награду (Return)*.
  - *Преимущество:* Хорошо работают в непрерывных пространствах действий, могут изучать стохастические политики.
  - *Пример:* REINFORCE.
- **Actor-Critic:**
  - *Идея:* Комбинировать Value-Based и Policy-Based подходы. Используются две модели (или две части одной модели):
    - **Actor (Актер):** Отвечает за выбор действий (учит *политику* $\pi(a|s;\theta)$).
    - **Critic (Критик):** Оценивает действия, выбранные Актером (учит *функцию ценности*, например, $V(s;w)$ или $Q(s,a;w)$).
  - *Принцип работы:* Критик помогает Актеру понять, насколько хороши были его действия. Критик предоставляет *более стабильную и менее шумную оценку качества действий* (по сравнению с использованием только сырой награды от среды), что ускоряет и стабилизирует обучение политики Актера.
  - *Примеры:* A2C (Advantage Actor-Critic), A3C (Asynchronous Advantage Actor-Critic), DDPG, SAC.

6 Примеры Применения RL

Где используется Обучение с Подкреплением?

RL добился впечатляющих результатов во многих областях:

- **Игры:** Обучение агентов игре на уровне человека или сверхчеловека (шахматы, го - AlphaGo/AlphaZero, видеоигры Atari, StarCraft).
- **Робототехника:** Обучение роботов ходьбе, манипуляциям с объектами, навигации.
- **Системы рекомендаций:** Персонализация контента или товаров, оптимизация долгосрочного вовлечения пользователя.
- **Оптимизация ресурсов:** Управление трафиком, распределение ресурсов в сетях, оптимизация рекламных кампаний и ставок (bidding).
- **Автономное вождение:** Принятие решений в сложных дорожных ситуациях (частично).
- **Химия и Биология:** Поиск новых молекул, оптимизация химических реакций.

7 Основные Вызовы в RL

Сложности и Ограничения

Несмотря на успехи, RL сталкивается с рядом серьезных проблем:

- **Требовательность к данным (Sample Inefficiency):** Большинству RL-алгоритмов (особенно model-free) требуется огромное количество взаимодействий со средой для обучения эффективной политике, что может быть дорого или невозможно в реальном мире.
- **Проблема присвоения награды (Credit Assignment Problem):** Сложно понять, какое именно действие в длинной последовательности привело к итоговой награде (особенно если награда редкая или отложенная).
- **Разработка функции награды (Reward Shaping):** Создание хорошей функции награды, которая корректно отражает желаемую цель и не приводит к нежелательному поведению агента, — часто сложная и нетривиальная задача.
- **Большие/Непрерывные пространства состояний и действий:** Стандартные методы (вроде табличного Q-learning) не работают. Требуется использование аппроксимации функций (нейронные сети) и более сложных алгоритмов.
- **Стабильность и Воспроизводимость:** Обучение RL-агентов может быть нестабильным, чувствительным к гиперпараметрам, и результаты бывает сложно воспроизвести.
- **Безопасность и Исследование:** Как позволить агенту безопасно исследовать среду, не совершая катастрофических ошибок (особенно в реальном мире, например, с роботами или автопилотами)?