David Silver - Reinforcement learning (2015)

Tom Rochette <tom.rochette@coreteks.org>

July 24, 2025 — daae079c

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- Is it possible to determine the contribution of individual contributors to a project using something like the eligibility traces?
- Can a reinforcement learning agent learn without experimenting with the world?
  - It seems possible if it has access to a model (simulator) of the environment it would act in
- Reinforcement learning is a good tool to attempt to solve a problem with a large search space, however when presented with an existing state and different types of problems (e.g., given the state space of software programs, find program X), it is not the proper framework/tool to use… So what should be used here?

1 Overview

2 Lecture 1 - Introduction to Reinforcement Learning

- Environment: A program with a state, which upon receiving input from an agent, runs its program and emits a response (observation) and may update its state
- The environment's program and state is generally smaller than what it can generate, which becomes what the agent can observe. In turn, the job of the agent is to attempt to reconstruct both program and state in order to map from the observable environment the current state of the program, such that the agent may be able to predict exactly how his actions will impact the environment in return. In the perfect case where the agent knows its state and the impact of its actions on the environment, it can thus control the next state such that it is the state desired by the agent
- A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \ldots, S_t]$$

- Markov property: The future is independent of the past given the present

$$H_{1:t} \to S_t \to H_{t+1:\infty}$$

- The environment state S_t^e is Markov
- The history H_t is Markov

2.1 Components of a reinforcement learning agent

- Policy: agent's behavior function
  - A map from state to action
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a \mid s) = \mathbb{P}[A = a \mid S = s]$
- Value function: how good is each state and/or action
  - Used to evaluate the goodness/badness of states
  - $v_\pi(s) = \mathbb{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \mid S_t = s]$
- Model: agent's representation of the environment
  - Transitions model: $\mathcal{P}$ predicts the next state
    - $\mathcal{P}^a_{ss'} = \mathbb{P}[S' = s' \mid S = s, A = a]$
  - Rewards model: $\mathcal{R}$ predicts the next (immediate) reward
    - $\mathcal{R}^a_s = \mathbb{E}[R \mid S = s, A = a]$

2.2 RL agents categories

- Value based
  - Value function
  - No policy (implicit)
- Policy based
  - Policy
  - No value function
- Actor critic
  - Policy
  - Value function
- Model free
  - Policy and/or value function
  - No model
- Model based
  - Policy and/or value function
  - Model

3 Lecture 2 - Markov Decision Processes

3.1 Markov Process

- A Markov Process (or Markov Chain) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
  - $\mathcal{S}$ is a (finite) set of states
  - $\mathcal{P}$ is a transition probability matrix
    - $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

3.2 Markov Reward Process

- A Markov Reward Process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a (finite) set of states
  - $\mathcal{P}$ is a transition probability matrix
    - $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
  - $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
  - γ is a discount factor, $\gamma \in [0, 1]$
- The return G_t is the total discounted reward from time-step t

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

3.3 Value Function

- The value function $v(s)$ gives the long term value of state s
- The state value function $v(s)$ of an MRP is the expected return starting from state s

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

- The Bellman equation

$$\begin{aligned} v &= \mathcal{R} + \gamma \mathcal{P} v \\ (I - \gamma \mathcal{P}) &= \mathcal{R} \\ v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

3.4 Markov Decision Process

- A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ is a transition probability matrix
    - $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
  - γ is a discount factor, $\gamma \in [0, 1]$
- A policy π is a distribution over actions given states

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$

- Policies are stationary (time-independent)
- The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

- The action-value function $q_\pi(s, a)$ is the expected return starting from state s

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

3.5 Optimal Value Function

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

- The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

3.6 Optimal Policy

- Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

- For any Markov Decision Process
  - There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
  - All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$
  - All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

- An optimal policy can be found by maximizing over $q_*(s, a)$,

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

4 Lecture 4 - Model-Free Prediction

4.1 Monte-Carlo Reinforcement Learning

- Learns directly from episodes of experience
- Uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs (episodes must terminate)
- MC policy evaluation uses empirical mean return instead of expected return
- Two dimensions
  - Bootstraping: DP/TD
  - Sampling: MC/TD
- n-Step Return

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- TD(λ), averaging n-Step Returns using a decay value λ

$$G_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

- Geometric decaying, why?
  - Memoryless (does not necessitate storage)
- Eligibility traces
  - Frequency heuristic: assign credit to most frequent states
  - Recency heuristic: assign credit to most recent states
  - Combines both heuristics

5 Lecture 5 - Model-Free Control

- On-policy learning
  - "Learn on the job"
- Off-policy learning
  - "Look over someone's shoulder"
- ϵ-Greedy Exploration
  - With probability $1 - \epsilon$, choose the greedy action
  - With probability ϵ, choose an action at random
- For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'} \geq v_\pi(s)$
- Greedy in the Limit with Infinite Exploration (GLIE)
  - All state-action pairs are explored infinitely many times,

$$\lim_{k \to \infty} N_k(s, a) = \infty$$

  - The policy converges on a greedy policy,

$$\lim_{k \to \infty} \pi_k(a \mid s) = \mathbf{1}(a = \arg\max_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example, ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$
- SARSA

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

- Monte-Carlo learning is a really bad idea off-policy, it does not work because over many steps, your target policy and your behavior policy never match enough to be useful

6 Lecture 6 - Value Function Approximation

- DQN uses experience replay and fixed Q-targets
  - Take action a_t according to ϵ-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
  - Sample random mini-batch of transition (s, a, r, s') from $\mathcal{D}$
  - Compute Q-learning targets with respect to fixed parameters w^-
  - Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

7 Lecture 7 - Policy Gradient

- Advantage of policy-based RL
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance
- Whenever state aliasing occurs a stochastic policy can do better than a deterministic policy
- For any differentiable policy $\pi_\theta(s, a)$, for any of the policy objective functions $J = J_1$, J_{avR}, or $\frac{1}{1-\gamma} J_{aaV}$, the policy gradient is

$$\underbrace{\nabla_\theta J(\theta)}_{policy\ gradient} = \underbrace{\mathbb{E}_{\pi_\theta}[\underbrace{\nabla_\theta \log \pi_\theta(s, a)}_{score\ function} \underbrace{Q^{\pi_\theta}(s, a)}_{action-value\ function}]}_{expectation\ under\ policy\ \pi_\theta}$$

- Monte-Carlo Policy Gradient methods are slow because they require many iterations to get to a final solution
- Actor-Critic algorithm:
  - Critic: Updates action-value function parameters w
  - Actor: Updates policy parameter θ, in the direction suggested by the critic

$$\underbrace{\nabla_\theta J(\theta)}_{policy\ gradient} = \underbrace{\mathbb{E}_{\pi_\theta}[\underbrace{\nabla_\theta \log \pi_\theta(s, a)}_{score\ function} \underbrace{Q_w(s, a)}_{action-value\ approximator}]}_{expectation\ under\ policy\ \pi_\theta}$$

- Reduce variance using a baseline
  - A baseline function B(s) is substracted from the policy gradient
  - The goal is to normalize the reward
  - A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
  - The advantage function $A^{\pi_\theta}(s, a)$

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \quad \underbrace{\nabla_\theta J(\theta)}_{policy\ gradient} = \underbrace{\mathbb{E}_{\pi_\theta}[\underbrace{\nabla_\theta \log \pi_\theta(s, a)}_{score\ function} \underbrace{A^{\pi_\theta}(s, a)}_{advantage\ function}]}_{expectation\ under\ policy\ \pi_\theta}$$

- The TD error δ_{π_θ} is an unbiased estimate of the advantage function

$$\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta} \mid s, a] = A^{\pi_\theta}(s, a)$$

8 Lecture 8 - Integrating Learning and Planning

- Model-Based RL
  - Advantages
    - Can efficiently learn model by supervised learning methods
    - Can reason about model uncertainty
  - Disadvantages
    - Two sources of approximation error (the model and the value function)
- Model-based RL is only as good as the estimated model
- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function (and/or policy) from real and simulated experience
- Monte-Carlo Tree Search explores and expands the most promising parts of the tree while ignoring the parts of the search tree which are useless or providing bad results
- Advantages of Monte-Carlo Tree Search
  - Highly selective best-first search
  - Evaluates states dynamically
  - Uses sampling to break the curse of dimensionality
  - Works for "black-box" models (only requires samples)
  - Computationally efficient, anytime, parallelizable

9 Lecture 9 - Exploration and Exploitation

- Three approaches to exploration
  - Random exploration
  - Optimist in the face of uncertainty
  - Information state space
- Two exploration spaces
  - State-action
  - Parameter
- Multi-armed bandit
  - A tuple $\langle \mathcal{A}, \mathcal{R} \rangle$
  - $\mathcal{A}$ is a known set of actions
  - $\mathcal{R}^a(r) = \mathbb{P}[R = r \mid A = a]$ is an unknown probability distribution over rewards
  - At each step t the agent selects an action $A_t \in \mathcal{A}$
  - The environment generates a reward $R_t \sim \mathcal{R}^{A_t}$
  - The goal is to maximize cumulative reward $\sum_{\tau=1}^{t} R_\tau$
- The action value is the mean reward for action a

$$q(a) = \mathbb{E}[R|A = a]$$

- The optimal value v_*

$$v_* = q(a^*) = \max_{a \in \mathcal{A}} q(a)$$

- The regret is the opportunity loss for one step

$$I_t = \mathbb{E}[v_* - q(A_t)]$$

- The total regret is the total opportunity loss

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} v_* - q(A_\tau)\right]$$

- The asymptotic total regret is at least logarithmic in the number of steps

$$\lim_{t\to\infty} L_t \geq \log t \sum_{a|\Delta_a>0} \frac{\Delta_a}{\mathrm{KL}(\mathcal{R}^a||\mathcal{R}^{a^*})}$$

- Hoeffding's inequality: Let $X_1, \ldots, X_t$ be i.i.d. random variables in [0, 1], and let $\overline{X}_t = \frac{1}{\tau}\sum_{\tau=1}^{t} X_\tau$ be the sample mean. Then

$$\mathbb{P}[\mathbb{E}[X] > \overline{X}_t + u] \leq e^{-2tu^2}$$

- The UCB1 algorithm

$$A_t = \arg\max_{a\in\mathcal{A}} \left[Q_t(a) + \sqrt{\frac{2\log t}{N_t(a)}}\right]$$

- Probability matching selects action a according to probability that a is the optimal action
- Probability matching is optimistic in the face of uncertainty
- Thompson sampling is a sample-based probability matching

$$\pi(a) = \mathbb{E}\left[\mathbf{1}(Q(a) = \max_{a'} Q(a')) \mid R_1, \ldots, R_{t-1}\right]$$

- Contextual bandit
  - A tuple $\langle \mathcal{A}, \mathcal{S}, \mathcal{R}\rangle$
  - $\mathcal{A}$ is a known set of actions
  - $\mathcal{R}_s^a(r) = \mathbb{P}[R = r \mid S = s, A = a]$ is an unknown probability distribution over rewards
  - At each step t
    - The environment generates a state $S_t \sim \mathcal{S}$
    - The agent selects an action $A_t \in \mathcal{A}$
    - The environment generates a reward $R_t \sim \mathcal{R}_{S_t}^{A_t}$
  - The goal is to maximize cumulative reward $\sum_{\tau=1}^{t} R_\tau$

10 See also

11 References

- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html