Reinforcement Learning Revisited

mn

Sunday 14th May, 2017

Contents

1 RL Problem

1.1 Problem Formulation

Agent and *Environment*: At each time step t the agent is in state S_t; it executes *Action* A_t, comes into a new state S_{t+1} and receives scalar *Reward* R_{t+1}.

History is the sequence of states/observations, actions, and rewards:

$$H_t = S_1, A_1, R_2, S_2, A_2, \dots, A_{t-2}, R_{t-1}, S_{t-1}$$

The *Goal* of the agent is to maximize the *return* (total amount of rewards it receives (starting at time $t+1$)). The *Return* G_t is some specific function of the reward sequence. E.g.

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

where $0 \leq \gamma \leq 1$ is the *discount rate*. The discount rate is introduced for mathematical reasons (needs to be < 1 for infinite sequences) and accounts for the intuition that immediate results matter more than far away uncertain results.

State $S_t \in \mathcal{S}$ is a function of history: $S_t = f(H_t)$. One can distinguish the environment state S_t^e, the agent state S_t^a. The information state (= Markov state) contains all useful information from the history.

Markov Property: the future is independent of the past given the present. A state S_t is Markov $\iff$ $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \dots, S_t]$.

Sometimes, the agent-environment interactions can be broken down into *sub-sequences*, called *episodes*. Each episode ends in a special state called the *terminal state*. Sometimes one needs to distinguish the set of all *non-terminal states* $\mathcal{S}$ from the set of all states plus the terminal state, $\mathcal{S}^+$.

If the agent-environment interaction does not break naturally into identifiable episodes, we call these *continuous tasks*, which can be limitless in time $T = \infty$. To unify the notation for episodic and continuous tasks, an absorbing state is introduced as the terminal state of an episode that transitions only to itself and generates zero rewards.

1.2 MDP

A RL task that satisfies the Markov property is called a *Markov Decision Process* (MDP). If the state and action spaces are finite, then it is called a *finite MDP*.

A particular finite MDP is defined by its state and action sets and the *one-step dynamics* of the environment: given any state s and action a, the probability of each possible pair of next state s' and reward r is

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\}$$

Given these dynamics, one can compute anything else.

- the *expected reward* is

$$r(s,a) = E[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

$$r(s) = E[R_{t+1}|S_t = s] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} p(a|s) p(s', r|s, a)$$

- the *state-transition probabilities*:

$$p(s'|s,a) = Pr\{S_{t+1} = s' | S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

$$p(s'|s) = Pr\{S_{t+1} = s' | S_t = s\} = \sum_{r \in \mathcal{R}} \sum_{a} p(a|s) p(s', r|s, a)$$

- the *expected rewards for state-action-next-step triples*:

$$r(s, a, s') = E[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r p(s', r|s, a)}{p(s'|s, a)}$$

- probability of reward in a state:

$$p(r|s) = Pr\{R_{t+1} = r|S_t = s\} = \sum_{s',a} p(a|s)p(s',r|s,a)$$

- note that $p(a|s) = \pi(a|s)$ is the stochastic policy of the agent

1.3 Value Functions

Value Function is a prediction of future awards: *how good* is it to be in a state or to perform an action in a state.

Stochastic *Policy* is a mapping from each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$ to the probability of taking action a when in state s:

$$\pi(a|s) = Pr[A_t = a|S_t = s]$$

Deterministic policy is a mapping from state to action $\pi : s \to a$. MDP policies depend only on the current state (not the history) and are therefore stationary (time-independent).

The *value* of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

The *state-value function for policy* π for MDP is

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\middle| S_t = s\right]$$

The *action-value function for policy* π, starting from state s, taking the action a, and thereafter following π is

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\middle| S_t = s, A_t = a\right]$$

1.4 Bellman equations

Bellman equation for v_π expresses the recursive relationship between the value of a state and the values of its successor states. The value function v_π is the unique solution to its Bellman equation.

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = E_\pi[R_{t+1}] + \gamma E[G_{t+1}|S_t = s] && (1)\\ &= \sum_{s',r,a} p(a|s)p(s',r|s,a)r + \gamma \sum_{s'} E_\pi[G_{t+1}|S_{t+1} = s'] && (2)\\ &= \sum_{s',r,a} p(a|s)p(s',r|s,a)r + \gamma \sum_{s'} \left(\sum_{r,a} p(a|s)p(s',r|s,a)v_\pi(s')\right) && (3)\\ &= \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \qquad \forall s \in \mathcal{S} && (4) \end{aligned}$$

It can be represented in vector form as a set of linear equations:

$$\boldsymbol{v}_\pi = \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \boldsymbol{v}_\pi$$

where $\boldsymbol{v}_\pi \in \mathbb{R}^{N_s}$ is a vector of state values for all states s, $\mathbf{r}_\pi \in \mathbb{R}^{N_s}$ is the vector of expected rewards over all states ($\mathbf{r}_\pi(s) = E[R_{t+1}|S_t = s, A_t = a]$), $\mathbf{P}_\pi \in \mathbb{R}^{N_s \times N_s}$ is the state transition probability matrix ($\mathbf{P}_{s,s'} = Pr[S_{t+1} = s'|S_t = s, A_t = a]$).

It can be solved directly

$$\boldsymbol{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{r}_\pi$$

with computational complexity of $O(n^3)$ for n states, which is possible only for small numbers of states. Though, there are iterative methods for large MRPs, such as dynamic programming, Monte-Carlo evaluation, and temporal-difference learning.

Similarly, the Bellman equation for the action-value function is

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[R_{t+1} + \gamma q_\pi(s',a')|S_t = s, A_t = a] = \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))$$

1.5 Optimal Value Functions

Optimal policy

$$\pi_* : \ \pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & \text{else} \end{cases}$$

Optimal state-value function is the maximum value function over all policies:

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

Optimal action-value function is the maximum action-value function over all policies:

$$\begin{aligned} q_*(s,a) &= \max_\pi q_\pi(s,a) && (5) \\ &= E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] && (6) \end{aligned}$$

Bellman optimality equations:

$$\begin{aligned} v_*(s) = \max_\pi v_\pi(s) = \max_a q_*(s,a) &= \max_a E[R_{t+1} + \gamma v_*(s')|S_t = s, A_t = a] && (7) \\ &= \max_a \sum_{s',r} p(s',r|s,a) r + \gamma \max_a \sum_{s',r} p(s',r|s,a) v_*(s') && (8) \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] && (9) \end{aligned}$$

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')|S_t = s, A_t = a] = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \qquad (10)$$

1.6 Planning / Learning Solutions

Policy evaluation: find $v_\pi(s)$ given π. Can be solved directly or via iterative approaches.

Policy improvement: greedy deterministic. $\pi'(s) = \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \implies q_\pi(s,\pi'(s)) \geq q_\pi(s,\pi(s)) \forall s \implies v_{\pi'}(s) \geq v_\pi(s)$. If $\pi'(s) == \pi(s) \iff \pi'(s) == \pi_*(s)$.

Policy iteration: alternate between policy evaluation and improvement.

Value iteration: update policy at every time step.

Generalized policy iteration: policy iteration at finer granularity.

It is possible to solve the Bellman equations theoretically, though this is rarely possible in practice. The direct solution relies on at least three critical assumptions: (1) accurate knowledge of the dynamics of the environment; (2) enough computational resources; (3) Markov property. For example, the state space is usually so huge that it's not feasible to solve the direct problem in a reasonable amount of time. Therefore, one typically has to rely on approximate solutions.

2 Dynamic Programming

Dynamic stands for "sequential or temporal component" of the problem, while *Programming* is to be understood in the mathematical sense which refers more to an approach or policy how to solve a problem (cf. linear programming or quadratic programming). *Dynamic programming* is a method for solving complex problems by breaking them down into sub-problems. It is a very general solution method to problems which have two properties: (1) optimal solution can be decomposed into sub-problems, and (2) sub-problems recur many times and can be cached and reused.

In terms of MDP, the Bellman equation gives recursive decomposition (1), and the value function stores and reuses solutions (2).

Dynamic programming assumes full knowledge of the MDP and is used for *planning*.

- For prediction — input: MDP + policy π; output: value function v_π
- For control — input: MDP; output: optimal value function v_* and optimal policy π_*

2.1 Policy Iteration

Iterate until converging to a policy, then take this policy as the new one, and estimate the value function for this policy; take the greedy policy for the new value function, etc. That is, we *iterate by policies*.

It will converge to the optimal value function and policy.

2.2 Value Iteration

In this case, we update the used policy with every computation of the value function. This value function may not correspond to any policy.

2.3 Synchronous Programming Algorithms

All of these are planning problems:

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

2.4 Asynchronous Dynamic Programming

Asynchronous DP backs up states individually, in any order. Still guaranteed to converge to the optimal value function.

- In-place DP: simply overriding the state value with the newly computed one.
- Prioritized Sweeping: use magnitude of Bellman error to guide state selection.
- Real-time DP

2.5 Backup

DP uses *full-width* backups.

3 Model-Free Prediction

3.1 Monte Carlo Learning

- learns directly from experience
- model free: unknown MDP
- learns from complete episodes
- based on the mean return

3.1.1 Incremental Mean

$$\mu_k = \frac{1}{k}\sum_{j}^{k} x_j = \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) = \frac{1}{k}(x_k + (k-1)\mu_{k-1}) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \tag{11}$$

3.1.2 Incremental Monte-Carlo Updates

$$N(S_t) \leftarrow N(S_t) + 1 \qquad v(S_t) += \frac{G_t - v(S_t)}{N(S_t)}$$

In non-stationary problems, track a running mean, ie. forget old episodes:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$$

3.2 Temporal-Difference Learning

- learns directly from experience
- model-free
- learns from incomplete episodes, by *bootstrapping*
- updates a guess towards a guess

3.3 Summary on Model-Free Prediction Methods

- MC has high variance, zero bias
  - good convergence properties (even with function approximation)
  - not very sensitive to initial value
  - simple to understand and to use
  - does not exploit Markov property
    - usually more efficient in non-Markov environments
  - does not bootstrap
  - samples
- TD has low variance, some bias
  - usually more efficient than MC
  - TD(0) converges to $v_\pi(s)$ (not always with function approximation)
  - more sensitive to initial value
  - TD exploits Markov property
    - usually more efficient in Markov environments
  - bootstraps
  - samples

Bootstrapping: update involves an estimate

- MC does not
- DP and TD do

Sampling: update samples an expectation

- DP does not
- MC and TD do

4 Model-Free Control

Relationship between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman expectation eq for $v_\pi(s)$	Iterative policy evaluation	TD learning
Bellman expectation eq for $q_\pi(s,a)$	Q-policy iteration	SARSA
Bellman optimality eq for $q_*(s,a)$	Q-value iteration	Q-Learning (SARSAMAX)

5 Terminology

State, *Agent* and *Environment*: with every action the agent observes the environment and is affected by the environment by transitioning to the next state and receiving a reward.

The *Goal* of the agent is to maximize the *return* G_t.

Discount rate: γ

Markov Property: the future is independent of the past given the present.

Episode, sub-sequence, terminal state, continuous task

A RL task that satisfies the *Markov property* is called a *Markov Decision Process* (MDP). If the state and action spaces are finite, then it is called a *finite MDP*:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$$

Value Function is a prediction of future awards.

Policy is a mapping from state to action (deterministic or stochastic).

Bellman equation expresses the recursive relationship between the state value v and the values of its successor states; and similar for state-action function q.

Optimal policy, state-value function, action-value function

Model-based vs. Model-free: synonymous with planning and learning methods.

Planning vs. *Learning*: methods that require a model (e.g. DP or heuristic search) are called planning methods, while model-free methods (e.g. MC or TD) are called learning methods.

Prediction vs. *Control*: finding the value function of a policy π is called *prediction.* Finding the optimal policy and value function is called *control.*

Model:

Backup:

Bootstrapping: update involves an estimate: (DP/TD do bootstraps)

Sampling: update samples an expectation (MC/TD do sampling)

5.1 Algorithms

- γ - discount rate
- α - forgetting old episodes; e.g. tracking running mean in MC
- σ
- δ

5.1.1 DP

- Synchronous: for prediction & control
- Asynchronous

5.1.2 MC

5.1.3 TD

5.1.4 HS

Heuristic search

6 Reviews

6.1 Sutton & Barto, 2016: RL: An Introduction

6.1.1 Chapter 1: The Reinforcement Learning Problem

6.1.2 Part 1: Tabular Solution Methods

6.1.3 Chapter 2: Multi-arm Bandits

Addresses a special case of a RL problem where there is only a single state.

6.1.4 Chapter 3: Finite Markov Decision Processes

Provides the general problem formulation of the RL problem in terms of MDP, and its main ideas, including Bellman equations and value functions. Discusses the Agent-Environment interface; goals, rewards, and the return.

6.1.5 Chapter 4: Dynamic Programming

This and the next two chapters describe fundamental classes of methods for solving MDP problems. Dynamic programming methods are well developed mathematically, but require a complete and accurate model of the environment.

6.1.6 Chapter 5: Monte Carlo Methods

Monte Carlo methods don't require a model and are conceptually simple, but are not well suited for step-by-step incremental computation.

6.1.7 Chapter 6: Temporal-Difference Learning

Temporal-difference methods don't require a model and are fully incremental, but are more complex to analyze.

6.1.8 Chapter 7: Multi-step Bootstrapping

Describes how the strengths of Monte Carlo and the temporal-difference methods can be combined via the use of eligibility traces.

6.1.9 Chapter 8: Planning and Learning with Tabular Methods

Describes how the temporal-difference learning methods can be combined with model learning and planning methods (such as dynamic programming) for a complete and unified solution to the tabular RL problem.

6.1.10 Part 2: Approximate Solution Methods

6.1.11 Part 3: Looking Deeper

6.2 David Silver, 2015: RL UCL Course

6.2.1 Lecture 1: Intro to RL

- Policy is agent's behavior, i.e. a map from state to action
  - Deterministic policy:
    $$a = \pi(s)$$
  - Stochastic policy:
    $$\pi(a|s) = P[A_t = a|S_t = s]$$
- Model: A model predicts that the environment will do next:
  - $\mathcal{P}$ predicts the next state, eg.
    $$\mathcal{P}^a_{ss'} = P[S_{t+1} = s'|S_t = s, A_t = a]$$
  - $\mathcal{R}$ predicts the next (immediate) reward, eg.
    $$\mathcal{R}^a_s = E[R_{t+1}|S_t = s, A_t = a]$$
- Categorizing RL agents:
  - value based (no policy (implicit), value function)
  - 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)
- Exploration and Exploitation

6.2.2 Lecture 2: Markov Decision Processes (MDP)

- MDP formally describes an environment for RL
- Markov Property: the future is independent of the past given the present

- State Transition Matrix defines transition probabilities from all states s to all successor states s':

$$\mathcal{P} = \left[something \right]$$

- Markov Process (or Markov Chain) is a memoryless random process, ie. a sequence of random states $S_1, S_2, \ldots$ with the Markov property. It is a tuple $(\mathcal{S}, \mathcal{P})$
- Markov Reward Process is a Markov chain with values $(\mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma)$
  - The Return G_t is the total discounted reward from time-step t:

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

  - State Value Function of an MRP is the expected return starting from state s:

$$v(s) = E[G_t | S_t = s]$$

  - MRP Bellman Equation is a linear equation:

$$\boldsymbol{v} = \mathbf{R} + \gamma \mathbf{P} \boldsymbol{v}$$

    * can be solved directly, with computational complexity of $O(n^3)$ for n states, which is possible only for small states.
    * Many iterative methods for large MRPs:
      - dynamic programming
      - Monte-Carlo evaluation
      - Temporal-Difference learning
- Markov Decision Process (MDP) is a Markov Reward Process with decisions. It is an environment in which all states are Makrov $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
  - Policies: ..
  - Value functions
    * The state-value function $v_\pi(s) = E_\pi[G_t | S_t = s]$ of an MDP is the expected return starting from state s and then following policy π.
    * The action-value function $q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$ is the expected return starting from state s, taking action a, and then following policy π.
  - Bellman Expectation Equation can be expressed via

$$v_\pi = \mathbf{R}^\pi + \gamma \mathbf{P}^\pi v_\pi$$

  - Optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$ is the maximum value function over all policies.
  - Optimal action-value function $q_*(s,a) = \max_\pi q_\pi(s,a)$ is the maximum action-value function over all policies.

6.2.3 Lecture 3: Planning by Dynamic Programming

6.2.4 Lecture 4: Model-Free Prediction

6.2.5 Lecture 5: Model-Free Control

6.2.6 Lecture 6: Value Function Approximation

6.2.7 Lecture 7: Policy Gradient Methods

6.2.8 Lecture 8: Integrating Learning and Planning

6.2.9 Lecture 9: Exploration and Exploitation

6.2.10 Lecture 10: Case Study: RL in Classic Games

7 Notations

$\mathcal{S}, \mathcal{A}, \mathcal{R}$ are sets of all possible states, actions and rewards.

8 References

- Sutton & Barto: `https://webdocs.cs.ualberta.ca/~sutton/`
- Silver: youtube, slides
- OpenAI gym
- WildML rl
- Udacity GeorgiaTech rl course