Reinforcement Learning

1 Generally

Action space A is the space containing all possible actions.

State space S is the space containing all possible states.

Transition function $P : S, A, S \to [0, 1]$ and $P^a_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a)$.

Reward function $P : S, A, (S) \to \mathbb{R}$ and $R^a_{ss'} = E[r_{t+1} | a_t = a, s_t = s, s_{t+1} = s']$.

Discount rate $0 \leq \gamma \leq 1$ weights the reward across time.

A stochastic policy π is a mapping from each state s and action a to a probability $\pi(a|s)$.

Return at time t is the reward accumulated starting from the next time step $R_t = r_{t+1} + \ldots + r_T$ in episodic tasks. T is the final time step in the terminal (absorbing) state.

Discounted return at time t is the $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$.

State value function is the value of state s under policy π: $V^\pi(s) = E_\pi[R_t | s_t = s]$.

State-action value function is the value of starting from s, taking the action a (not following π, and thereafter following π: $Q^\pi(s, a) = E_\pi[R_t | s_t = s, a_t = a]$.

A stochastic process has the Markov property if the conditional probability distribution of future states of the process depends only upon the present state $P(s_{t+1} = s', r_{t+1} = r | s_t, a_t)$.

RL prediction problem is to evaluate a policy by computing a value function of the policy.

RL control problem is to find an optimal policy and value function that maximises the expected return.

In the reward matrix R the rows are states and columns are actions.

In the transition matrix P^a the rows are previous states and columns are next states.

In policy matrix π the rows are states and columns are actions.

For continuous tasks $T = \infty$. Cannot have both $T = \infty$ and $\gamma = 1$ or define absorbing states as transitioning to themselves with a reward of zero.

Bellman Expectation Equations for Policy Evaluation

$$\begin{aligned} V_\pi(s) &= E_\pi[R_t | s_t = s] \\ &= E_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s] \\ &= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_\pi(s')] \\ Q_\pi(s, a) &= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \sum_{a'} Q_\pi(s', a')] \end{aligned}$$

Bellman Optimality Equations for Greedy Policy Improvement

$$\begin{aligned} V^*(s) &= \max_\pi V_\pi(s) \\ &= \max_a Q_{\pi_*}(s, a) \\ &= \max_a E_{\pi_*}[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')] \\ Q^*(s, a) &= \max_\pi Q_\pi(s, a) \\ &= E[R_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a] \\ &= E[R_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a] \\ &= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')] \\ &= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')] \end{aligned}$$

Optimal policy π_* is such that $\pi \geq \pi'$ iff $V_\pi(s) \geq V_{\pi'}(s)$.

There exists a unique solution to the Bellman equations and policy and value iterations converge to it.

Tabular RL stores state-values for each state as an array for finite state and finite action problems.

Model-based algorithm requires transition and reward functions.

For tabular model-based problem, the Bellman Expectation Equations are $|S|$ linear equations with $|S|$ unknowns.

Backup is updating the value of a state using values of future states. Full backup is when every state is backed up since we do not sample future states.

Learning rate for the tabular case is $0 \leq \alpha \leq 1$ and any for the linear function approximation.

Bootstrapping is using one or more estimated values in the update step for the same kind of estimated value.

An MDP with a defined policy defines a Markov Chain. A Markov Chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

Generalised Policy Iteration is letting policy evaluation and policy improvement processes interact. This means having identifiable policies and value functions, with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy. A GPI algorithm is not guaranteed to converge to an optimal policy.

Dynamic Programming requires a model, bootstraps. Converges to optimal policy.

Monte Carlo does not require a model, does not bootstrap. More robust to violations of the Markov assumption because does not bootstrap, better matches the training data, implicitly learns the transitions.

TD does not require a model, bootstraps. Converges faster than MC because it moves towards a better estimate that takes transitions into account, or it bootstraps.

TD(λ) maximises long term goals.

Why would you use? Difference between. When are guaranteed to converge?

We can define an optimal policy if we are given the state-value function and have the model.

2 Dynamic Programming

Iterative Policy Evaluation uses Bellman Expectation Equation starting from arbitrary V_0 and converges to V^π. At every iteration every state is backed up.

Policy Improvement Theorem for deterministic policies π, π', if $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$ for all s then $V^{\pi'}(s) \geq V^\pi(s)$.

Policy Improvement

$$\begin{aligned}\pi'(s) &= \operatorname*{argmax}_a Q^\pi(s, a) \\ &= \operatorname*{argmax}_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]\end{aligned}$$

Policy Iteration: arbitrarily initialise V and π_0, perform Policy Evaluation, perform Policy Improvement, iterate. Converges to an optimal policy.

Value Iteration is like Policy Iteration but with only a single backup of each state in the Policy Evaluation step. Converges to an optimal policy which is computed at the end.

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

Asynchronous DP performs evaluation and improvement computations without going through all the states or in any specific ordering and with any state or state-action values currently available. To converge, it needs to visit all states in expectation. It might help improve convergence by avoiding states that do not appear in optimal trajectories. Examples: value iteration with only one state updated per iteration, real-time DP.

3 Monte Carlo

Monte Carlo – model-free, simulator does not explicitly define a transition and reward function, no bootstrapping, prediction and control. More robust towards violations of Markov assumption because no bootstrap.

First-visit MC

Every-visit MC

Exploring starts

Off-policy method evaluates the estimation policy π while following the behaviour policy π'. Compute the weighted average of returns from behaviour policy, weighting factors are the probability of the moves being in estimation policy, i.e. weight each return by relative probability of being generated by π and π'.

On-policy

On-policy MC, off-policy MC

Estimation and behaviour policies in off-policy RL algorithms

Incremental off-policy MC control

Exploration in MC: on-policy that explores off-policy exploring starts

4 Temporal Difference

TD – model-free, bootstrapping,

TD(0)

Target

On-policy TD control is SARSA Off-policy TD control is Q-learning

On-line performance of SARSA vs Q-learning: cliff walking example

Compare lambda parameter with the discount factor.

Difficulty implementing Q(lambda)

TD converges faster than Monte Carlo because it moves towards a better estimate that takes transitions into account, or it bootstraps.

5 Eligibility Traces

Complex backup

n-step return lambda-return

Main problem with them is keeping information of all future states and back them up.

6 Function Approximation

7 Semi-MDPs and Options

8 Inverse Reinforcement Learning

9 Partially Observable MDPs

10 Multi-Agent RL