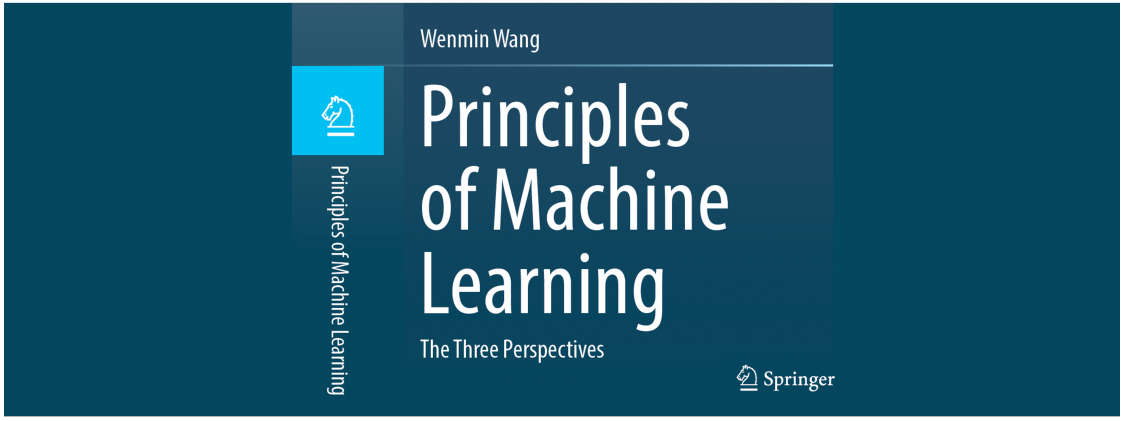

Wenmin Wang
Principles of Machine Learning
The Three Perspectives
Principles of Machine Learning
Springer

Part III Paradigms

10 Reinforcement Learning Paradigm

Contents

About Reinforcement Learning

- Reinforcement learning has solid theoretical foundations and broad application fields.
- Theoretical foundations: behavioral psychology, Markov decision process, and Monte Carlo learning.
- Application fields: artificial intelligence, operations research, control theory, game theory, and information theory.
- Concerns with: how agents take actions in an environment in order to maximize the notion of cumulative reward.
- Development stages: ancient reinforcement learning, neoteric reinforcement learning, and modern reinforcement learning.

Development Stages

- **Ancient Reinforcement Learning**: 1890s∼1950s

  The stage of reinforcement learning based on study of animal behaviors:

  - Respondent conditioning, by Ivan Pavlov;
  - Operant conditioning, by Edward Thorndike, and Burrhus Skinner.
    The concept of “reinforcement” learning was first proposed.

- **Neoteric Reinforcement Learning**: 1950s∼1970s

  The stage of reinforcement learning based on automatic control theory.

  - Neural-Analog Reinforcement Systems, by Marvin Minsky;
  - Reinforcement Learning Control Systems, by M. D.Waltz and King-sun Fu;
  - Reinforcement Learning Control and Pattern Recognition Systems, by Jerry Mendel and Robert McLaren.

Development Stages

- **Modern Reinforcement Learning**: 1980s∼

  The stage of modern computational reinforcement learning.

  Markov decision process is used as theoretical model to study how an intelligent agent interacts with the environment and takes action based on current state and reward.

  A milestone is the PhD thesis *Temporal Credit Assignment in Reinforcement Learning* by Richard Sutton in 1984, under the supervision of prof. Andrew Barto.

  Sutton and Barto are considered the founders of modern computational reinforcement learning.

Richard Sutton and Andrew Barto

What is Reinforcement Learning

Definition: Reinforcement learning

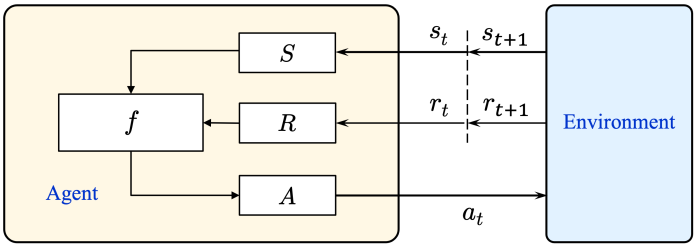
Reinforcement learning (RL) is a sequential decision method in machine learning, which is modeled as Markov decision process. The intelligent agent in RL interacts with the environment, taking corresponding action based on current state and feedback from the environment.

A Markov decision process (MDP) is formalized as a 4-tuple MDP $= \langle S, A, P, R \rangle$:

- S is the state set,
- A is the action set,
- P is the state transition probability,
- R is the reward function.

What is Reinforcement Learning

$$f : S \times R \to A$$



Two types of RL based on the model of MDP:

- Model-based: all elements in the model are known, can employ planning method.
- Model-free: existing uncertain elements in the model, uses learning approach.

Policy

Policy: Given a MDP $= \langle S, A, P, R \rangle$, its control policy π at time point t is the action a_t to be taken under the state s_t, represented as:

$$\pi\left(a_t | s_t\right) = \mathbb{P}\left[A\left(t\right) = a_t | S\left(t\right) = s_t\right].$$

Apply π to MDP, let $t \leftarrow 0$, and get current s_t, then RL forms following procedure:

(1) $a_t \leftarrow \pi\left(a_t | s_t\right)$: according to policy to take an action $a \in A$;

(2) $r_{t+1} \leftarrow R\left(s_t, a_t\right)$: using reward function to get current reward $r \in R$;

(3) $s_{t+1} \leftarrow P\left(s_{t+1} | s_t, a_t\right)$: with transition probability to get current state $s \in S$;

(4) $t \leftarrow t + 1$, and goto 1.

So the state-action sequence is generated: $\{s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots\ldots\}$.

Policy is the core issue for MDP optimal control to determine the actions.

Return

Return: It refers to the weighted cumulative rewards in RL.

For the return G_t from time-step t:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Where, γ is the discount factor, and $0 \leq \gamma \leq 1$.

(1) If $\gamma = 0$, then $G_t = r_{t+1}$, i.e., it only focuses on immediate reward r_{t+1}.

(2) If $0 < \gamma < 1$, then G_t is convergent, and equal to the immediate reward plus a part of future discounted rewards.

(3) If $\gamma = 1$, then G_t becomes: $G_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1}$. In this case, G_t does not converge.

Value Function

State-Value Function: The value function of a state s under policy π, is denoted as:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}^{\pi}\left[G_t \mid S(t)=s\right] \\ &= \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid S(t)=s\right]. \end{aligned}$$

State-Action-Value Function: Similarly, the state-action-value function of state s and action a under policy π, is denoted as:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}^{\pi}\left[G_t \mid S(t)=s, A(t)=a\right] \\ &= \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid S(t)=s, A(t)=a\right]. \end{aligned}$$

Bellman Equations

Bellman Equation for State-Value Function:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid S(t)=s\right] \\
&= \mathbb{E}^{\pi}\left[r_{t+1}+\gamma\sum_{k=1}^{\infty}\gamma^{k} r_{t+k+1} \mid S(t)=s\right] \\
&= \mathbb{E}^{\pi}\left[r_{t+1}+\gamma V^{\pi}(s_{t+1}) \mid S(t)=s\right] \\
&= \sum_{s'} P\left(s' \mid s, \pi(s)\right)\left(R(s,a)+\gamma V^{\pi}\left(s'\right)\right).
\end{aligned}
$$

$$
V^{*}(s) = \max_{a\in A}\sum_{s'\in S} P\left(s' \mid s, a\right)\left(R(s,a)+\gamma V^{*}\left(s'\right)\right).
$$

Bellman Equations

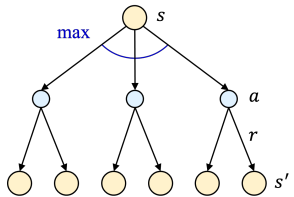
Bellman Equation for State-Action-Value Function:

$$
\begin{aligned}
Q^{*}(s,a) &= \mathbb{E}^{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k} \mid S(t)=s, A(t)=a\right] \\
&= \mathbb{E}^{\pi}\left[R\left(s,a,s'\right)+\gamma \max_{a'} Q^{*}(s',a') \mid S(t)=s, A(t)=a\right] \\
&= \sum_{s' \in S} P\left(s'|s,a\right)\left(R(s,a)+\gamma \max_{a'} Q^{*}\left(s',a'\right)\right).
\end{aligned}
$$

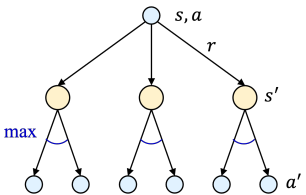
Comparison on Bellman Equations

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P\left(s'|s,a\right)\left(R\left(s,a\right) + \gamma V^*\left(s'\right)\right).$$

$$Q^*(s,a) = \sum_{s' \in S} P\left(s'|s,a\right)\left(R\left(s,a\right) + \gamma \max_{a'} Q^*\left(s',a'\right)\right).$$



(a)



(b)

Bellman equations for (a) state-value function and (b) state-action-value function.

Model-Based and Model-Free

Model-Based Methods: For a MDP $= \langle S, A, P, R \rangle$ where all the elements are known beforehand, so it can based on the model directly for optimal control.

Those methods are planning-based, under the general name of *dynamic programming* (DP), that is also called *decision-theoretic planning*, as an effective solution to multistage processes.

Model-Free Methods: In most sequential decision-making tasks, it is difficult to rely on the model of MDP $= \langle S, A, P, R \rangle$, since P and R are unknown.

But, the agent can indirectly perceive P and R through interaction with the environment, generates state and reward samples, and then uses them for optimal control.

Those methods are learning-based, under the general name of *reinforcement learning* (RL). It can be called *decision-theoretic learning*.

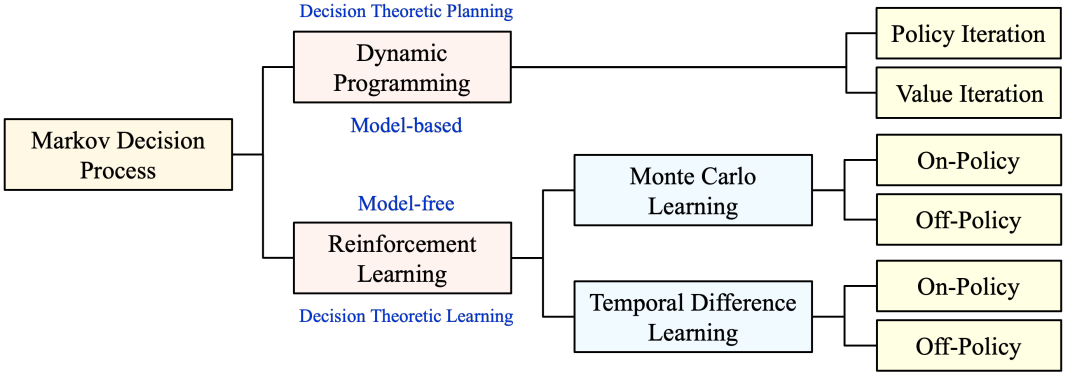
On-Policy and Off-Policy

On-policy learning: The policy for evaluation and improvement is the same as the policy used for decision-making.

Off-policy learning: The policy for evaluation and improvement, on the other hand, is different from the policy used for decision-making.

On-policy	Off-policy
Using the deterministic outcomes or samples from the target policy to train the algorithm.	Training on the distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy.
Learning policy π from the episodes that generated using the π.	Learning policy π from the episodes generated using another policy ϕ.
Learning while doing the job.	Learning while watching other one doing the job.

Hierarchical Relationships



Dynamic programming and Reinforcement learning.

About the Dynamic Programming

Dynamic programming (DP) is proposed by Richard Bellman in early 1950s, used to solving the optimization problem of multistep decision processes.

It transforms the multi-step process into a series of single-step problems, using the relationships between steps to solve them one by one.

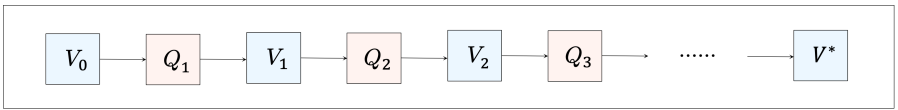
In decision-theoretic planning, dynamic programming is as a model-based method, used to optimize control of MDP and calculate its optimal control policy.

The two typical methods of dynamic programming are value iteration and policy iteration, proposed by Richard Bellman and Ronald Howard respectively, based on the Bellman equations.

Dynamic Programming

Value Iteration: It focuses entirely on directly evaluating its value function, calculates the necessary updates in real time, combining the process of policy evaluation and policy improvement.

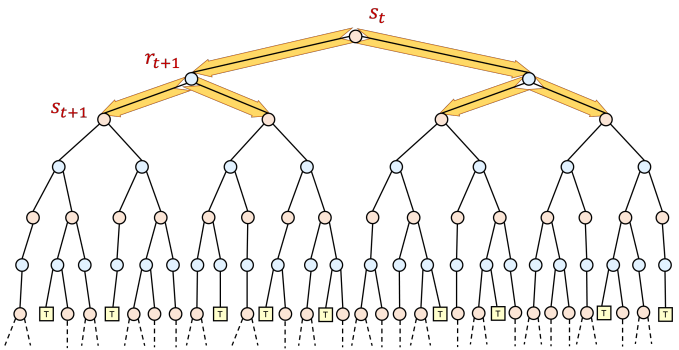
$$
\begin{aligned}
Q_{t+1}(s,a) &= \sum_{s' \in S} P\left(s' \mid s,a\right)\left(R(s,a) + \gamma V_t\left(s'\right)\right). \\
V_{t+1}(s) &= \max_a Q_{t+1}(s,a).
\end{aligned}
$$



Dynamic Programming

Value Iteration: The algorithm is to update $V(s_t)$ toward estimated return $V(s_{t+1})$:

$$V(s_t) \leftarrow \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})].$$



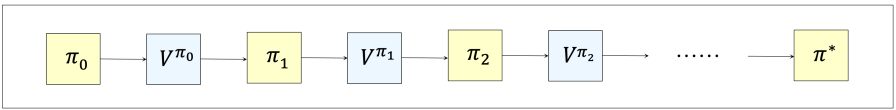
Dynamic Programming

Policy Iteration: Be divided into two processes:

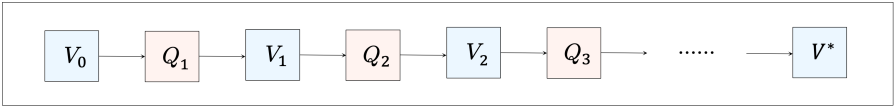
(1) Policy evaluation: assesses the current policy by calculating its value function;

(2) Policy improvement: obtains the improved policy through the maximization of value function.

$$
\begin{aligned}
V_{k+1}^{\pi}(s) &= \sum_{s' \in S} P\left(s' \mid s, \pi(s)\right)\left(R\left(s, \pi(s)\right) + \gamma V_k^{\pi}\left(s'\right)\right). \\
\pi'(s) &= \arg\max_{a \in A} \sum_{s' \in S} P\left(s' \mid s, a\right)\left(R(s, a) + \gamma V^{\pi}\left(s'\right)\right).
\end{aligned}
$$



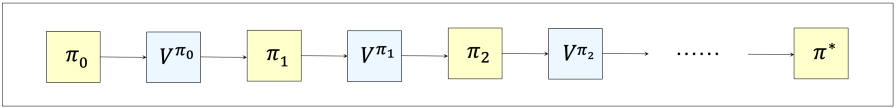
Dynamic Programming



Value iteration: generates a sequence of alternating value-action and value functions.

Pros: each iteration is very computationally efficient;

Cons: convergence is only asymptotic, expensive.



Policy iteration: generates a sequence of alternating policies and value functions.

Pros: converge in a finite number of iterations;

Cons: each iteration requires a full policy evaluation, and might be expensive.

About the Monte Carlo Learning

Monte Carlo (MC) methods, we have already learned, are a set of methods of estimating the probability of a random event or the expected value of a random variable through repeated random sampling.

Monte Carlo learning (MC learning) is one of the model-free methods, learned form a sample sequence of state, action, and reward at each episode, called experience.

It equivalents to sampling from the probability distribution and reward function.

This sample sequence can either be the real experience of the agent interacting with the environment, or the simulated experience obtained through some method.

An episode at time $\{0, 1, \ldots, T\}$ is a sequence consisting of states, actions, and rewards:

$$\left\{s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1}\right\}.$$

Monte Carlo Learning

For the first visit, MC learning takes $V^{\pi}(s)$ as the first visit:

$$V^{\pi}(s) = \mathbb{E}^{\pi}\left[G_t \mid S(t) = s\right].$$

Where the G_t needs to wait until the final visit, and is used as the target of $V(s_t)$:

$$V(s) = \mathbb{E}\left[G_t \mid S(t) = s\right].$$

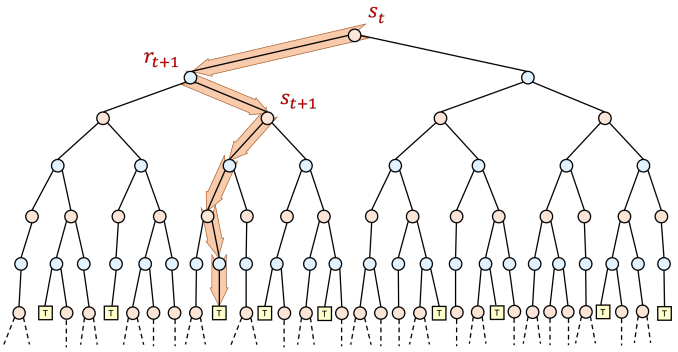
The state-value is estimated by mean return:

$$\begin{aligned} V(s) &= \mathbb{E}\left[G_t \mid S(t) = s\right] = \frac{1}{N(s_t)} \sum_{i=1}^{t} G(i) \\ &= V(s_{t-1}) + \frac{1}{N(s_t)} \left(G(t) - V(s_{t-1})\right). \end{aligned}$$

Monte Carlo Learning

Let constant step-size parameter $\alpha \approx \frac{1}{N(s_t)}$, then the every-visit MC learning is:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(G_t - V(s_t)\right).$$



About the Temporal Difference Learning

Temporal difference (TD) learning is another class of the model-free methods, the combination of MC learning and DP:

- Like MC learning, it learns directly from experiences, without the model of MDP;
- Like DP, it updates based in part on learned estimates, without waiting for final outcome.

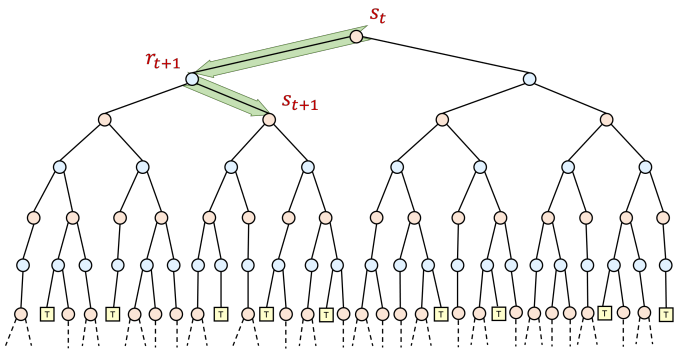
But, TD learning is different from MC learning:

- MC learning needs to wait for the final result of an episode before they can predict the state-value function;
- TD learning can predict the value functions after knowing the next result within an episode, without waiting for the final result of the episode, which is called bootstrapping.

Temporal Difference Learning

TD learning only needs to wait for next step, using r_{t+1} and $V(s_{t+1})$ to calculate $V(s_t)$:

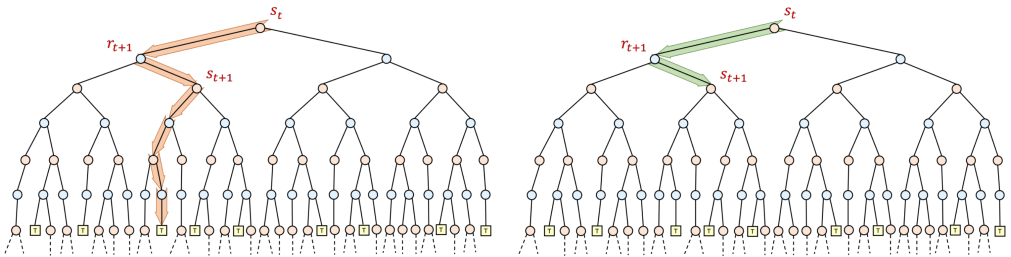
$$V(s_t) \leftarrow V(s_t) + \alpha\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right).$$



Temporal Difference Learning

Comparing MC with TD learning, the deference are: G_t vs. $r_{t+1} + \gamma V(s_{t+1})$, since:

- MC learning: $V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$
- TD learning: $V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$.



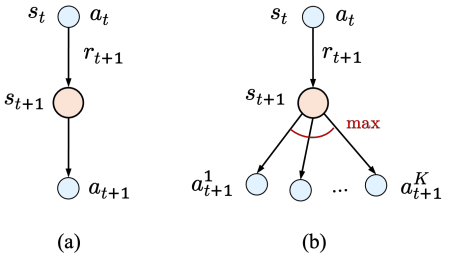
The $\delta_t \doteq r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called *TD error*:

Temporal Difference Learning

SARSA Algorithm vs. Q-Learning Algorithm

SARSA (state–action–reward–state–action): an on-policy TD algorithm, named according to its process. See figure (a).

Q-Learning: an off-policy TD algorithm, to learn the policy of what action to choose under what circumstances. See figure (b).



$$\text{SARSA:} \quad Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right).$$

$$Q\text{-learning:} \quad Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a+1} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right).$$

About the Eligibility Traces

Eligibility traces are one of the basic mechanisms of reinforcement learning.

An eligibility trace is a brief record of an event, marking the parameters associated with the event and suitable for learning changes as eligible.

When a TD error occurs, the eligibility trace is only necessary to look for the cause of the error from the events with eligible marks.

Eligibility traces unify and TD learning and MC learning.

here are two ways to view eligibility traces: **forward view** and **backward view**.

Eligibility Trace Mechanisms

Forward view: It refers to the expectation of all future rewards for each visited state by reinforcement learning algorithm.

Specifically, it decides how to update through all future rewards and states, like a person standing on the state flow and looking forward from current state.

The n-step return of TD learning is defined as:

$$G_{t:t+n} \doteq r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \hat{V}\left(s_{t+n}, \boldsymbol{w}_{t+n-1}\right).$$

For $TD\left(\lambda\right)$, it is not only n-step return, but also λ-return, defined as:

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t.$$

Eligibility Trace Mechanisms

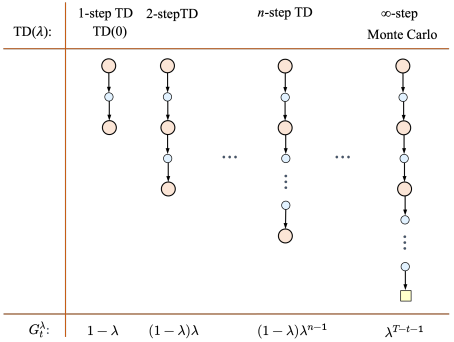
Since $0 \leq \lambda \leq 1$, so a series of methods are generated with the change of λ.

The eligibility traces in $TD(\lambda)$ are:

(1) When $\lambda = 0$, the figure is simplified to only the first term, which is 1-step TD learning, $TD(0)$.

(2) When $\lambda = 1$, the figure is simplified to only the last term, which is the Monte Carlo (MC) learning.

(3) The n^{th} step, called n-step TD.



The eligibility traces serve as a bridge between TD and MC learning.

Eligibility Trace Mechanisms

Backward view: Each update of the backward view depends on current TD error and the eligibility traces of past events.

In $TD(\lambda)$, eligibility trace vector $\boldsymbol{z}_t$ is initialized to 0, then increasing over time:

$$\boldsymbol{z}_t \doteq \gamma\lambda\boldsymbol{z}_{t-1} + \nabla\hat{V}(s_t, \boldsymbol{w}_t). \quad 0 \leq t \leq T.$$

Eligibility trace tracks $\boldsymbol{z}_t$ to determine which components made positive or negative contributions to recent state estimation.

The TD error of state-value prediction is:

$$\delta_t \doteq r_{t+1} + \gamma\hat{V}(s_{t+1}, \boldsymbol{w}_t) - \hat{V}(s_t, \boldsymbol{w}_t).$$

Where the weight vector $\boldsymbol{w}_t$ is updated at each step, and $\boldsymbol{w}_{t+1} \doteq \boldsymbol{w}_t + \alpha\delta_t\boldsymbol{z}_t$.

Eligibility Trace Mechanisms

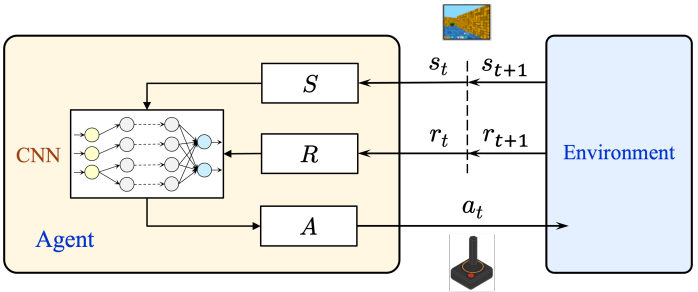
Forward View	Backward View
Cannot be directly implemented because it uses rewards and states that will occur after each step but have not actually occurred yet.	Providing an incremental mechanism that can approximate the forward view and can be accurately implemented offline.
Which is *acausal relations*	Which is *causal relations*
Be most useful for understanding what is computed by methods using eligibility traces.	Be more appropriate for developing intuition about the algorithms themselves.
Unifying TD learning and MC learning, and has important theoretical significance.	
Which is called *theoretical view*	Which is called *mechanistic view*.

About Deep Reinforcement Learning

- Deep reinforcement learning (DRL) is the product of combining reinforcement learning with deep learning.
- A series of algorithms represented by deep Q-network, have achieved the results comparable to or even better than the human level in video games and machine games.
- We next introduce the deep Q-network, the first DRL model proposed by DeepMind in 2013.

Deep Q-Network

The deep Q-network (DQN) combines CNN with Q-learning that is used to control the AI video game system on Atari 2600 console.



The CNN in DQN consists of 3 convolutional layers and 2 fully connected layers. The input is a $84 \times 84 \times 4$ image, and the output corresponds to the actions of joystick.

Deep Q-Network

The approximate value function of DQN is denoted as $Q(s, a; \theta_t)$, where the parameter θ_t is the weight of the Q-network in the t iteration.

Firstly, it uses experience replay for training. the agent first stores the experience at each moment t, $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ into the dataset $D = \{e_1, e_2, \ldots, e_N\}$, then uses the method of uniform random sampling to extract data from this dataset, that is $e \sim D$, and uses the extracted data to train the network.

Secondly, a target network is set up to handle the TD error in the TD algorithm separately. DQN uses stochastic gradient descent to update network parameters:

$$\theta_{t+1} = \theta_t + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t) - Q(s_t, a_t; \theta_t) \right) \nabla_{\theta_t} Q(s_t, a_t; \theta_t).$$

Where, $r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t)$ is the TD target.

Comparison of Reinforcement Learning

Comparison of representative reinforcement learning methods and algorithms.

Model	Method	Algorithm	Policy	State/Action Space
Model-based	Dynamic Programming	Value Iteration	Optimal	Discrete/Discrete
		Policy Iteration	Optimal	Discrete/Discrete
Model-free	Monte Carlo Learning	Exploring Starts	On-policy	Discrete/Discrete
		Importance Sampling	Off-policy	Discrete/Discrete
	TD Learning	SARSA	On-policy	Discrete/Discrete
		Q-Learning	Off-policy	Discrete/Discrete
	TD + Eligibility Traces	SARSA(λ)	On-policy	Discrete/Discrete
		Q-Learning(λ)	Off-policy	Discrete/Discrete
	Deep RL	DQN	Off-policy	Cont./Discrete
		A3C	On-policy	Cont./Cont.

Thank You