



SNS COLLEGE OF TECHNOLOGY



Coimbatore-35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19CSE303 – ARTIFICIAL INTELLIGENCE

III YEAR IV SEM

UNIT V – LEARNING

TOPIC : Reinforcement Learning



Outline

- Introduction
- Element of reinforcement learning
- Reinforcement Learning Problem
- Problem solving methods for RL



Introduction

Machine learning: Definition

- Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data, such as from sensor data or databases.

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data .



Machine learning Type:

With respect to the feedback type to learner:

- ❑ **Supervised learning** : Task Driven (Classification)
- ❑ **Unsupervised learning** : Data Driven (Clustering)
- ❑ **Reinforcement learning** —
 - Close to human learning.
 - Algorithm learns a policy of how to act in a given environment.
 - Every action has some impact in the environment, and the environment provides rewards that guides the learning algorithm.



Supervised Learning Vs Reinforcement Learning

Supervised Learning

- Step: 1

Teacher: Does picture 1 show a car or a flower?

Learner: A flower.

Teacher: No, it's a car.

Step: 2

Teacher: Does picture 2 show a car or a flower?

Learner: A car.

Teacher: Yes, it's a car.

Step: 3



Supervised Learning Vs Reinforcement Learning (Cont...)

Reinforcement Learning

Step: 1

World: You are in state 9. Choose action A or C.

Learner: Action A.

World: Your reward is 100.

Step: 2

World: You are in state 32. Choose action B or E.

Learner: Action B.

World: Your reward is 50.

Step: 3



Introduction (Cont..)



- **Meaning of Reinforcement:** Occurrence of an event, in the proper relation to a response, that tends to increase the probability that the response will occur again in the same situation.

Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment.

- Reinforcement Learning is learning how to act in order to maximize a numerical reward.



Introduction (Cont..)

- **Reinforcement learning** is not a type of neural network, nor is it an alternative to neural networks. Rather, it is an orthogonal approach for Learning Machine.
- Reinforcement learning emphasizes learning feedback that evaluates the learner's performance **without providing standards of correctness** in the form of behavioral targets.

Example: Bicycle learning



Element of reinforcement learning



- ❑ **Agent:** Intelligent programs
- ❑ **Environment:** External condition
- ❑ **Policy:**
 - ❑ Defines the agent's behavior at a given time
 - ❑ A mapping from states to actions
 - ❑ Lookup tables or simple function

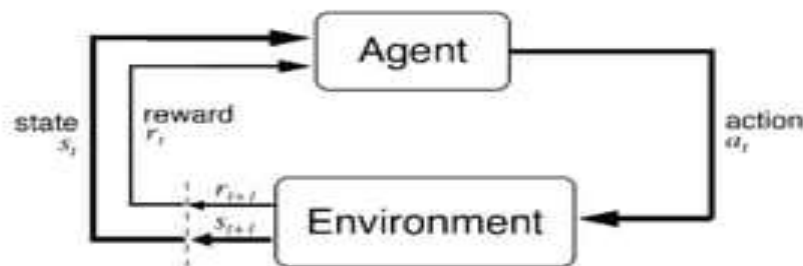


Element of reinforcement learning

- **Reward function :**
 - Defines the goal in an RL problem
 - Policy is altered to achieve this goal
- **Value function:**
 - Reward function indicates what is good in an immediate sense while a value function specifies what is good in the long run.
 - Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- **Model of the environment :**
 - Predict mimic behavior of environment,
 - Used for planning & if Know current state and action then predict the resultant next state and next reward.



Agent- Environment Interface



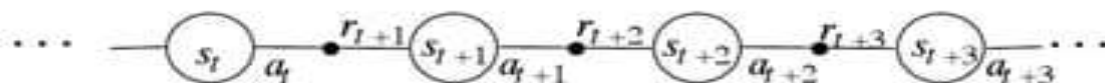
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \dots$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \mathfrak{R}$

and resulting next state : s_{t+1}





Steps for Reinforcement Learning

1. The agent observes an input state
2. An action is determined by a decision making function (policy)
3. The action is performed
4. The agent receives a scalar reward or reinforcement from the environment
5. Information about the reward given for that state / action pair is recorded



Silent Features of Reinforcement Learning :

- **Set of problems** rather than a set of techniques
- Without specifying how the task is to be achieved.

- **“RL as a tool”** point of view:
 - RL is training by rewards and punishments.
 - Train tool for the computer learning.

- **The learning agent’s point of view:**
 - RL is learning from trial and error with the world.
 - Eg. how much reward I much get if I get this.



Reinforcement Learning (Cont..)

- Reinforcement Learning uses **Evaluative Feedback**
- **Purely Evaluative Feedback**
 - Indicates how good the action taken is.
 - Not tell if it is the best or the worst action possible.
 - Basis of methods for function optimization.
- **Purely Instructive Feedback**
 - Indicates the correct action to take, independently of the action actually taken.
 - Eg: Supervised Learning
- **Associative and Non Associative:**



Associative & Non Associative Tasks



- **Associative :**
 - Situation Dependent
 - Mapping from situation to the actions that are best in that situation
- **Non Associative:**
 - Situation independent
 - No need for associating different action with different situations.
 - Learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is non stationary.



Reinforcement Learning (Cont..)

Exploration and exploitation

Greedy action: Action chosen with greatest estimated value.

Greedy action: a case of **Exploitation**.

Non greedy action: a case of **Exploration**, as it enables us to improve estimate the non-greedy action's value.

- **N-armed bandit Problem:**

We have to choose from n different options or actions.

We will choose the **one with maximum reward**.



Bandits Problem



One-Bandit
“arms”

Pull arms sequentially so as to maximize the total expected reward

It is non associative and evaluative



Greedy action - action with the highest estimated reward.

- **ϵ -greedy**
 - Most of the time the greedy action is chosen
 - Every once in a while with a small probability ϵ , an action is selected at random. The action is selected uniformly, independent of the action-value estimates.
- **ϵ -soft** - The best action is selected with probability $(1 - \epsilon)$ and the rest of the time a random action is chosen uniformly.



ϵ -Greedy Action Selection Method :

Let the a^* is the greedy action at time t and $Q_t(a)$ is the value of action a at time.

Greedy Action Selection:

$$a_t = a_t^* = \arg \max_a Q_t(a)$$

■ ϵ -greedy

$$a_t = \begin{cases} a_t & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$



Action Selection Policies (Cont...)

- **Softmax** –
 - Drawback of ϵ -greedy & ϵ -soft: Select **random actions uniformly**.
 - Softmax remedies this by:
 - Assigning a **weight with each actions**, according to their action-value estimate.
 - A random action is selected with regards to the weight associated with each action
 - The worst actions are unlikely to be chosen.
 - This is a good approach to take where the worst actions are very unfavorable.



Softmax Action Selection(Cont...)

- Problem with ϵ -greedy: Neglects action values
- Softmax idea: grade action probs. by estimated values.
- Gibbs, or Boltzmann action selection, or exponential weights:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}}$$

τ is the “computational temperature”

At $\tau \rightarrow 0$ the Softmax action selection method become the same as greedy action selection.



Incremental Implementation

- Sample average:

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a}$$

- Incremental computation:

$$Q_{k+1} = Q_k + \frac{1}{k+1} [r_{k+1} - Q_k]$$

- **Common update rule form:**

$$\text{NewEstimate} = \text{OldEstimate} + \text{StepSize} [\text{Target} - \text{OldEstimate}]$$

- The expression [Target - Old Estimate] is an error in the estimate.
- It is reduce by taking a step toward the target.
- In proceeding the (t+1)st reward for action a the step-size parameter will be $1/(t+1)$.



Some terms in Reinforcement Learning



- **The Agent Learns a Policy:**

- Policy at step t , π_t , a mapping from states to action probabilities will be:

$$\pi_t(s, a) = \text{probability that } a_t = a \text{ when } s_t = s$$

- Agents changes their policy with Experience.
- Objective: get as much reward as possible over a long run.

- **Goals and Rewards**

- A goal should specify what we want to achieve, not how we want to achieve it.



Some terms in RL (Cont...)

■ Returns

- Rewards in long term
- **Episodes**: Subsequence of interaction between agent-environment e.g., plays of a game, trips through a maze.

■ Discount return

- The geometrically discounted model of return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- Used to: where γ , $0 \leq \gamma \leq 1$, is the **discount rate**
 - To determine the present value of the future rewards
 - Give more weight to earlier rewards

The Markov Property

- Environment's response at (t+1) depends only on State (s) & Action (a) at t,

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

Markov Decision Processes

□ Finite MDP

- Transition Probabilities:

$$P_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \text{ for all } s, s' \in S, a \in A(s).$$

- Expected Reward

$$R_{ss'}^a = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \text{ for all } s, s' \in S, a \in A(s).$$

Transition Graph: Summarize dynamics of finite MDP.



Value function

- States-action pairs function that estimate how good it is for the agent to be in a given state
- Type of value function
 - **State-Value function**
 - The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State-value function for policy π :

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- **Action-Value function**

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

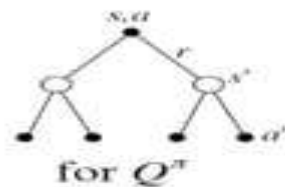
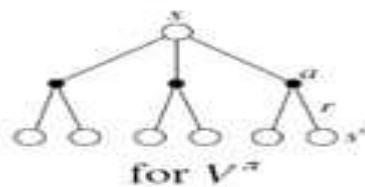
Action-value function for policy π :

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$



- Basis of the update or backup operations
- backup diagrams to provide graphical summaries of the algorithms. (State value & Action value function)

Backup diagrams:



Bellman Equation for a Policy π

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^\pi(s')]$$



Model-free and Model-based Learning



- **Model-based learning**

- Learn from the model instead of interacting with the world
- Can visit arbitrary parts of the model
- Also called indirect methods
- E.g. Dynamic Programming

- **Model-free learning**

- Sample Reward and transition function by interacting with the world
- Also called direct methods



Problem Solving Methods for RL

- 1) Dynamic programming
- 2) Monte Carlo methods
- 3) Temporal-difference learning.



1. Dynamic programming

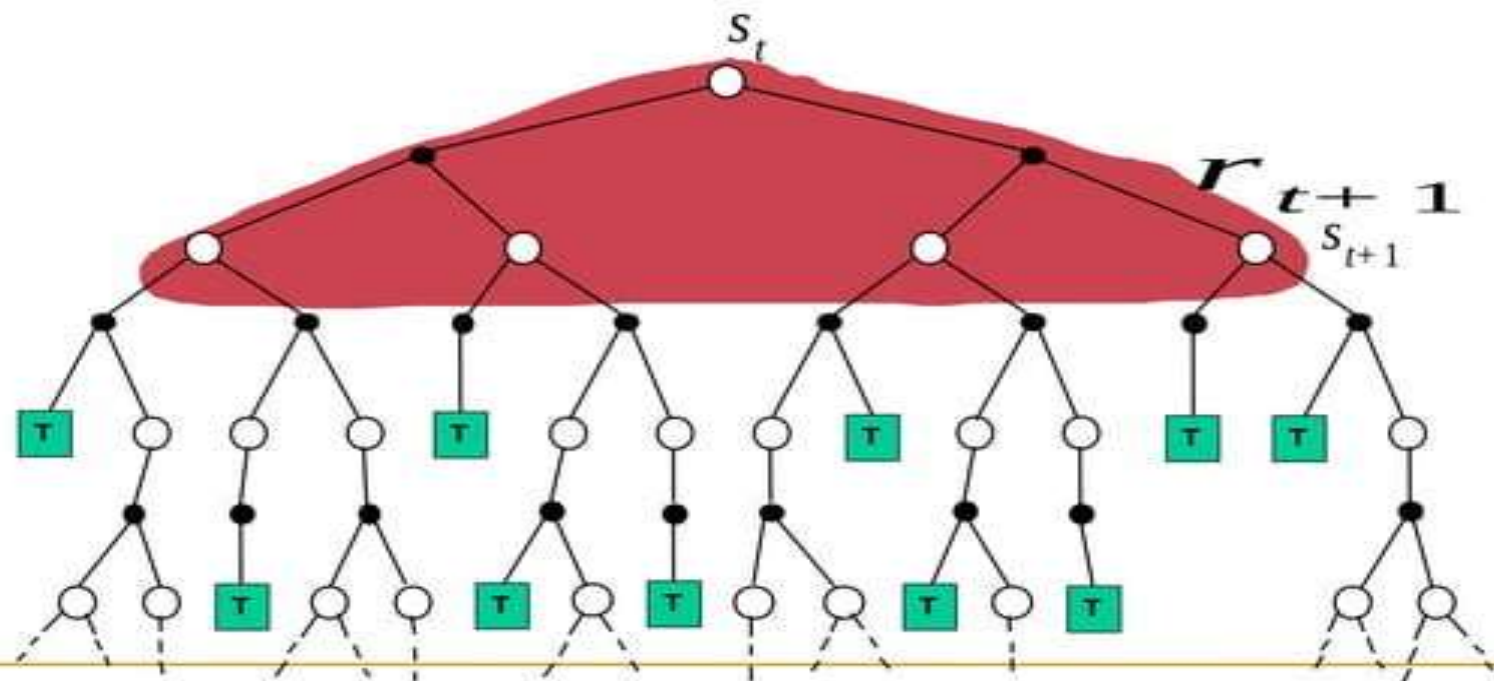
- Classical solution method
- Require a complete and accurate model of the environment.
- Popular method for Dynamic programming
 - **Policy Evaluation** : Iterative computation of the value function for a given policy (prediction Problem)
 - **Policy Improvement**: Computation of improved policy for a given value function.

$$V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_t) \}$$



Dynamic Programming

$$V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_{t+1}) \}$$





Policy Evaluation

Policy Evaluation: for a given policy π , compute the state-value function V^π

Recall: **State - value function for policy π :**

$$V^\pi(s) = E_\pi \{ R_t \mid s_t = s \} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$


Bellman equation for V^π :

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^\pi(s')]$$



Iterative Methods

$$V_0 \rightarrow V_1 \rightarrow \dots \rightarrow V_k \rightarrow V_{k+1} \rightarrow \dots \rightarrow V^\pi$$

a "sweep" 

A sweep consists of applying a **backup operation** to each state.

A **full policy-evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$



Iterative Policy Evaluation

```
Input  $\pi$ , the policy to be evaluated
Initialize  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 
Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx V^\pi$ 
```

$$\text{NewEstimate} = \text{OldEstimate} + \text{StepSize}[\text{Target} - \text{OldEstimate}]$$



Policy Improvement

Suppose we have computed V^π for a deterministic policy π .

For a given state s ,
would it be better to do an action $a \neq \pi(s)$?

The value of doing a in state s is:

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} \mathcal{P}_{ss'}^a \left[\mathcal{R}_{ss'}^a + \gamma V^\pi(s') \right] \end{aligned}$$

It is better to switch to action a for state s if and only if

$$Q^\pi(s, a) > V^\pi(s)$$

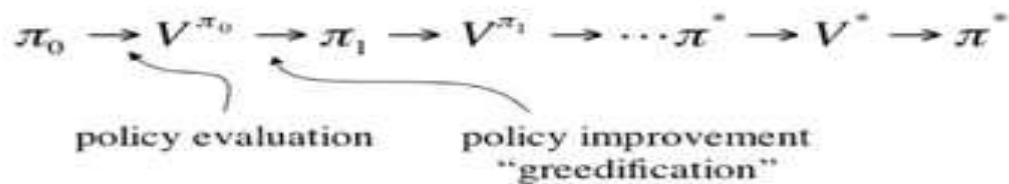
Computation of improved policy for a given value function



- Policy Evaluation and Policy improvement together leads
 - **Policy Iteration**
 - **Value Iteration:**



Policy Iteration : Alternates policy evaluation and policy improvement steps



1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} [\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

policy-stable \leftarrow true

For each $s \in \mathcal{S}$:

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$

If $b \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop; else go to 2



Value Iteration

- Combines evaluation and improvement in one update

Recall the **full policy-evaluation backup**:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$

Here is the **full value-iteration backup**:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$$



Value Iteration:

Combines evaluation and improvement in one update

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$$\Delta \leftarrow 0$$

For each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π , such that

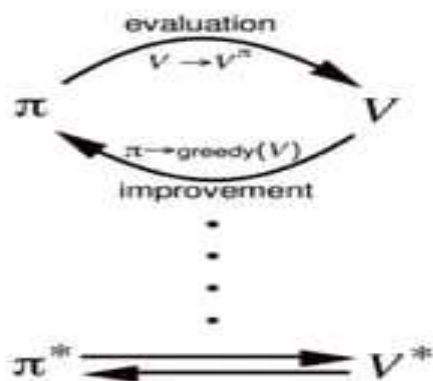
$$\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$$



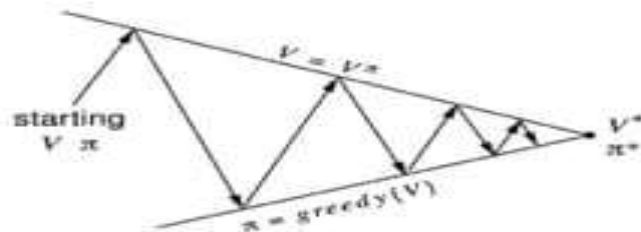
Generalized Policy Iteration (GPI)

Consist of two iteration process,

- **Policy Evaluation** : Making the value function consistent with the current policy
- **Policy Improvement**: Making the policy greedy with respect to the current value function



A geometric metaphor for convergence of GPI:





2. Monte Carlo Methods

■ **Features of Monte Carlo Methods**

- ❑ No need of Complete knowledge of environment
- ❑ Based on averaging sample returns observed after visit to that state.
- ❑ Experience is divided into **Episodes**
- ❑ Only after completing an episode, value estimates and policies are changed.
- ❑ Don't require a model
- ❑ Not suited for step-by-step incremental computation

- Compute same value function
- Same step as in DP
 - Policy evaluation
 - Computation of V_{Π} and Q_{Π} for a fixed arbitrary policy (Π).
 - Policy Improvement
 - Generalized Policy Iteration



To find value of a State

- Estimate by experience, average the returns observed after visit to that state.
- More the return, more is the average converge to expected value

Monte Carlo Policy Evaluation

- ❑ *Goal:* learn $V^\pi(s)$
- ❑ *Given:* some number of episodes under π which contain s
- ❑ *Idea:* Average returns observed after visits to s



- ❑ *Every-Visit MC:* average returns for *every* time s is visited in an episode
- ❑ *First-visit MC:* average returns only for *first* time s is visited in an episode



First-visit Monte Carlo Policy evaluation



Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

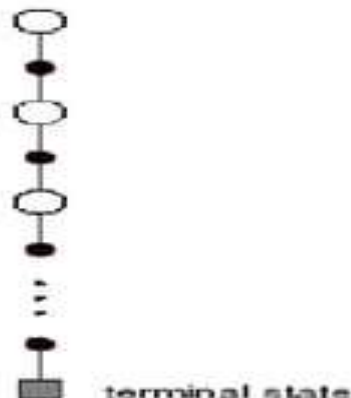
Append R to $Returns(s)$

$V(s) \leftarrow average(Returns(s))$



Backup diagram for Monte Carlo

- Entire episode included
- Only one choice at each state (unlike DP)
- MC does not bootstrap
- Time required to estimate one state does not depend on the total number of states



Monte Carlo Estimation of Action Values (Q)

- Monte Carlo is most useful when a model is not available
 - We want to learn Q^*
- $Q^\pi(s,a)$ - average return starting from state s and action a following π
- Also converges asymptotically *if* every state-action pair is visited
- Exploring starts*: Every state-action pair has a non-zero probability of being the starting pair



Monte Carlo Exploring Starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

Repeat forever:

(a) Generate an episode using exploring starts and π

(b) For each pair s, a appearing in the episode:

$R \leftarrow$ return following the first occurrence of s, a

Append R to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

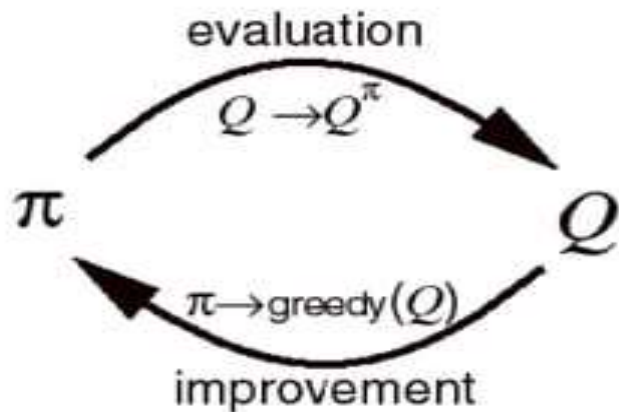
(c) For each s in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

In **MCES**, all returns for each state-action pair are accumulated and averaged, irrespective of what Policy was in force when they were observed



Monte Carlo Control



- ❑ **MC policy iteration:** Policy evaluation using MC methods followed by policy improvement
- ❑ **Policy improvement step:** greedify with respect to value (or action-value) function



Monte Carlo Method (Cont...)

- All actions should be selected infinitely for optimization
- MC exploring starts can not converge to any suboptimal policy.

- Agents have 2 approaches
 - **On-policy**
 - **Off-policy**



On-policy Monte Carlo Control

- ❑ *On-policy*: learn about policy currently executing
- ❑ How do we get rid of exploring starts?
 - Need *soft* policies: $\pi(s,a) > 0$ for all s and a
 - e.g. ϵ -soft policy:

$$\frac{\epsilon}{|A(s)|} \qquad 1 - \epsilon + \frac{\epsilon}{|A(s)|}$$

non-max greedy

- ❑ Similar to GPI: move policy *towards* greedy policy (i.e. ϵ -soft)
- ❑ Converges to best ϵ -soft policy



On-policy MC algorithm

Initialize, for all states and actions

$Q(s,a)$ = arbitrary

$Returns(s,a)$ = empty list

π = an arbitrary soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s,a appearing in the episode:

R = return following the first occurrence of s,a

Append R to $Returns(s,a)$

$Q(s,a)$ = average($Returns(s,a)$)

(c) For each s in the episode:

a^* = $\arg \max_a Q(s,a)$

For all actions in s :

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \epsilon / |A(s)| & \text{if } a = a^* \\ \epsilon / |A(s)| & \text{if } a \neq a^* \end{cases}$$



Off Policy MC algorithm

Off-policy: Follow one policy and learn about another

- Behavior policy* generates behavior in environment
- Estimation policy* is policy being learned about

Behaviour policy can be soft (so exploring starts not needed)
and estimation policy can be deterministic (so fully greedy)

How do we evaluate one policy following another?

- Weight returns from behavior policy by probability they would occur using the estimation policy
- Easier to evaluate estimation policy when behavior policy is similar!



Monte Carlo and Dynamic Programming



- MC has several advantage over DP:
 - Can learn from interaction with environment
 - No need of full models
 - No need to learn about ALL states
 - No bootstrapping



3. Temporal Difference (TD) methods

- Learn from experience, like MC
 - Can learn directly from interaction with environment
 - No need for full models
- Estimate values based on estimated values of next states, like DP
- Bootstrapping (like DP)
- **Issue to watch for:** maintaining sufficient exploration



TD Prediction

Policy Evaluation (the prediction problem):

for a given policy π , compute the state-value function

Simple every - visit Monte Carlo method : V^{π}

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

↑
target

The simplest TD method, TD(0) :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

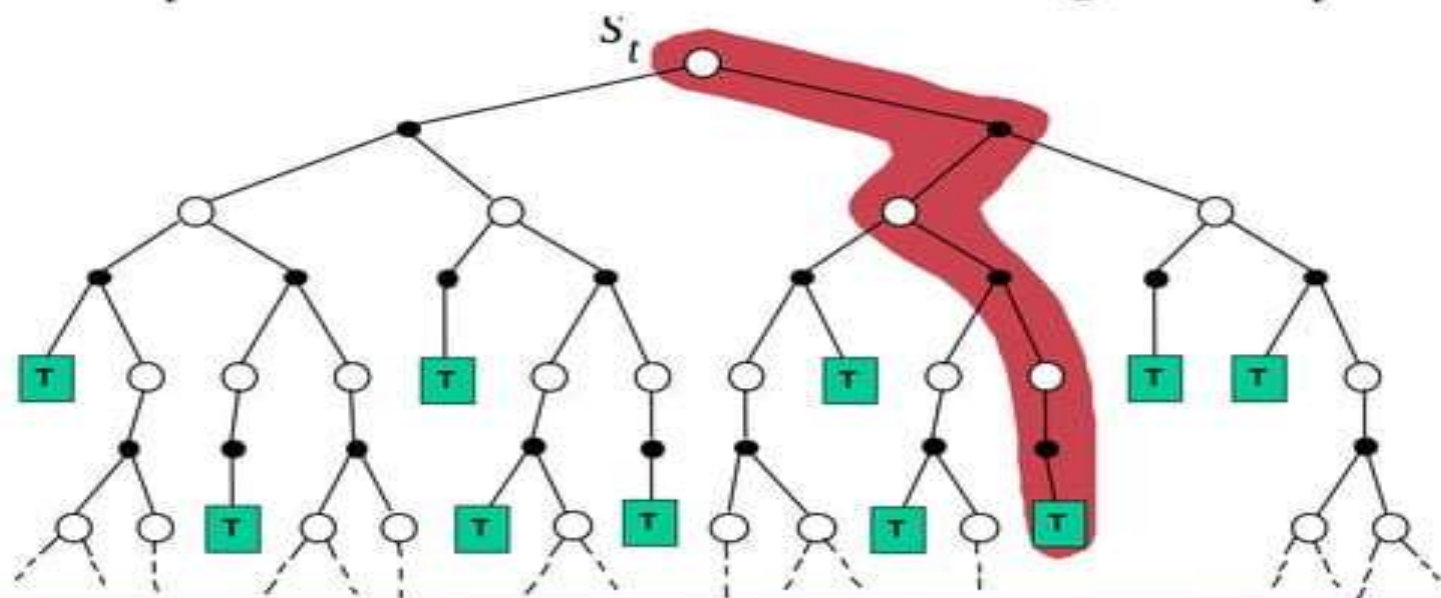
↑
target



Simple Monte Carlo

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

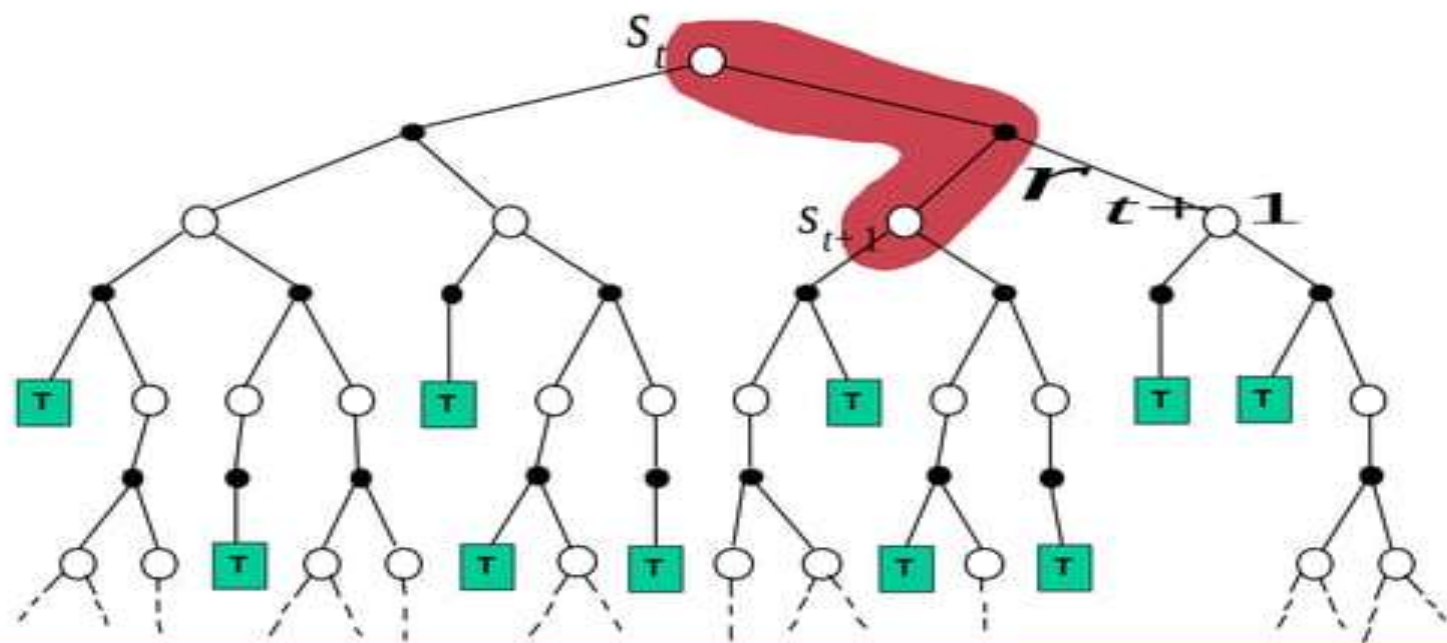
where R_t is the actual return following state s_t .





Simplest TD Method

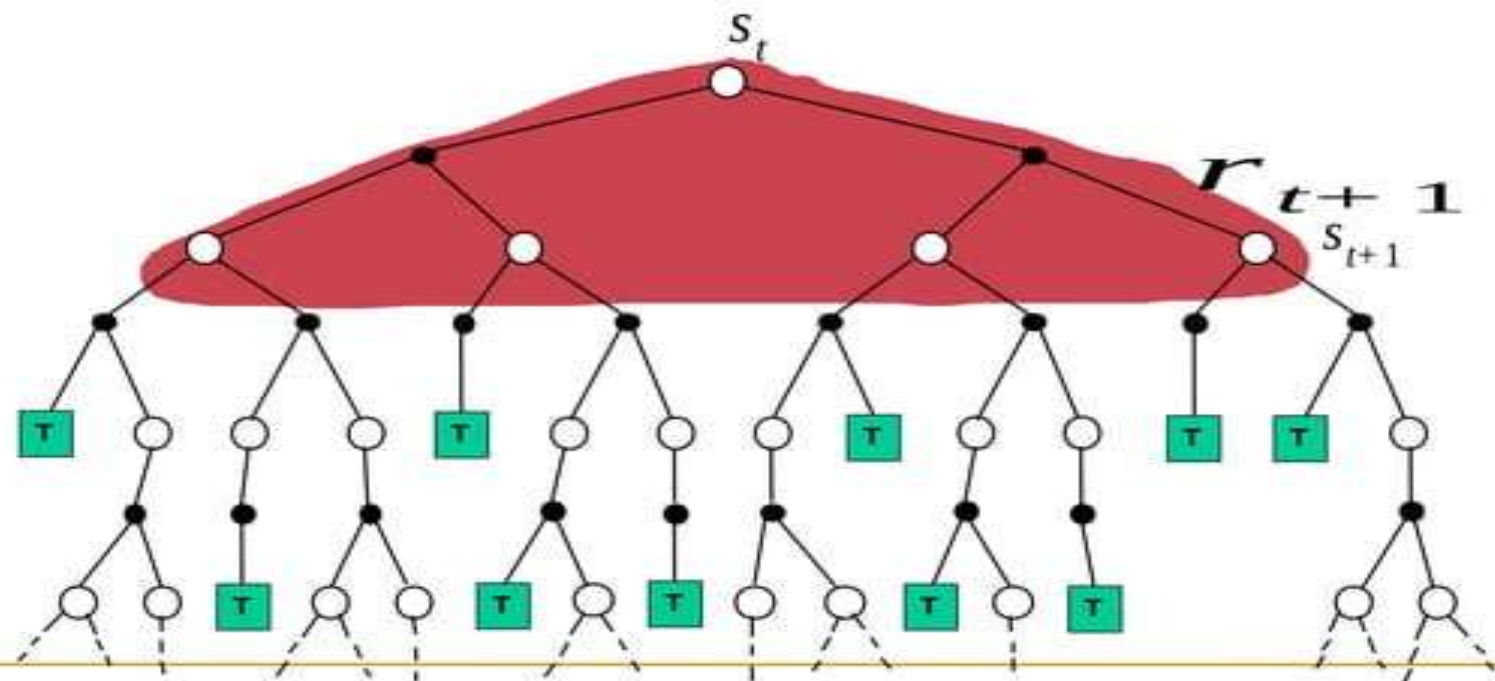
$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$





Dynamic Programming

$$V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_{t+1}) \}$$





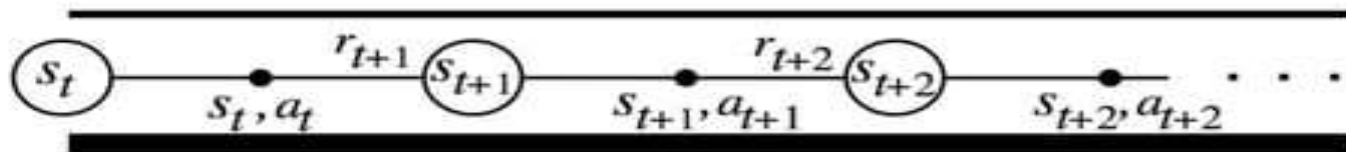
TD methods bootstrap and sample

- **Bootstrapping:** update involves an *estimate*
 - MC does not bootstrap
 - DP bootstraps
 - TD bootstraps
- **Sampling:** update does not involve an *expected value*
 - MC samples
 - DP does not sample
 - TD samples



Learning An Action-Value Function

Estimate Q^p for the current behavior policy p .



After every transition from a nonterminal state s_t , do this

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

If s_{t+1} is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$.



Q-Learning: Off-Policy TD Control

One - step Q- learning :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Repeat (for each step of episode):

 Choose a from s using policy derived from Q (e.g., ϵ -greedy)

 Take action a , observe r, s'

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

$s \leftarrow s'$;

 until s is terminal



Sarsa: On-Policy TD Control

S A R S A: State Action Reward State Action

Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a)$ arbitrarily

Repeat (for each episode):

 Initialize s

 Choose a from s using policy derived from Q (e.g., ϵ -greedy)

 Repeat (for each step of episode):

 Take action a , observe r, s'

 Choose a' from s' using policy derived from Q (e.g., ϵ -greedy)

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

$s \leftarrow s'; a \leftarrow a';$

 until s is terminal



Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
 - You can learn **before** knowing the final outcome
 - Less memory
 - Less peak computation
 - You can learn **without** the final outcome
 - From incomplete sequences
- Both MC and TD converge



References: RL

- Univ. of Alberta
 - <http://www.cs.ualberta.ca/~sutton/book/ebook/node1.html>
 - Sutton and Barto, "Reinforcement Learning an introduction."
- Univ. of South Wales
 - <http://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html>



Thank You