REINFORCEMENT LEARNING
(part 2)

Nguyen Do Van, PhD

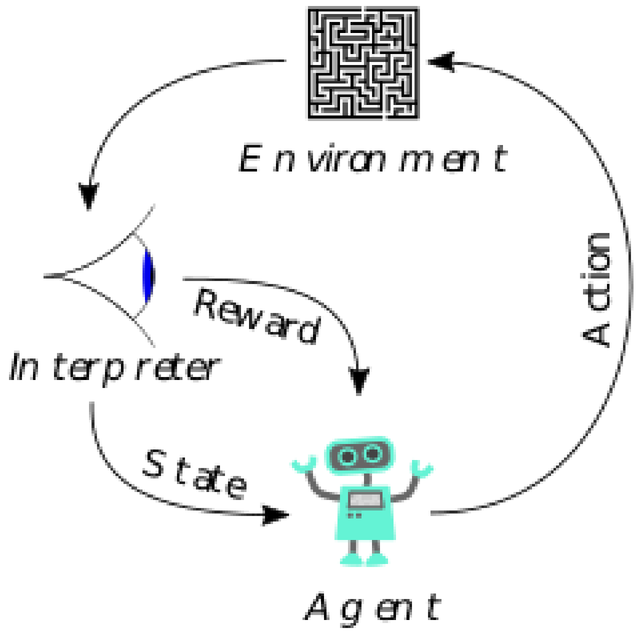
VIASM
Vietnam Institute for
Advanced Study in Mathematics

Reinforcement Learning

- Online Learning
- Value Function Approximation
- Policy Gradients

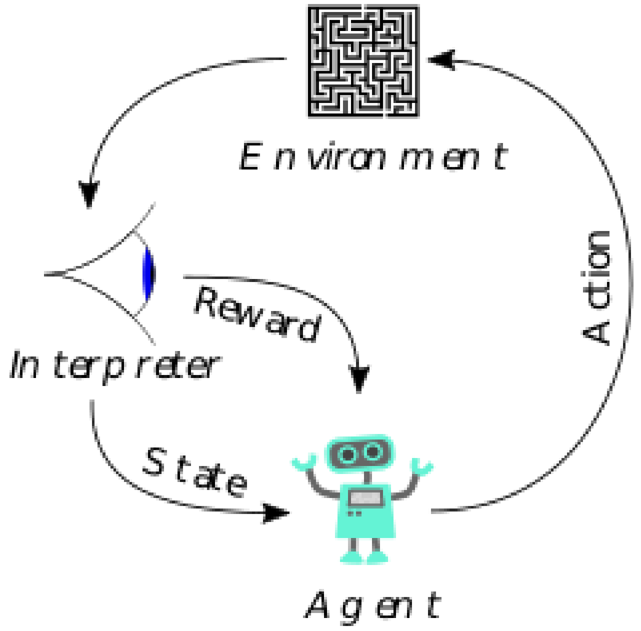
Reinforcement learning: Recall

- Making good decision to do new task: fundamental challenge in AI, ML
- Learn to make good sequence of decisions
- Intelligent agents learning and acting
  - Learning by trial-and-error, in real time
  - Improve with experience
  - Inspired by psychology:
    - Agents + environment
    - Agents select action to maximize *cumulative* rewards



Reinforcement learning: Recall

- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at environment step



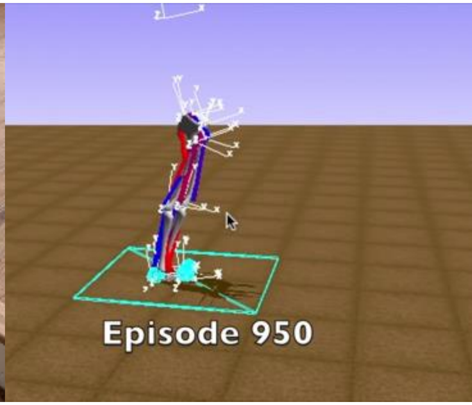
Vietnam Institute for Advanced Study in Mathematics

Reinforcement learning: Recall

- Policy - maps current state to action
- Value function - prediction of value for each state and action
- Model - agent’s representation of the environment.

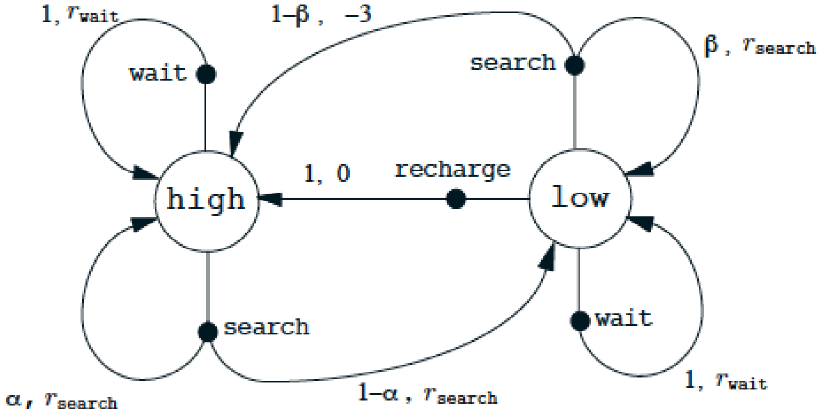
Markov Decision Process (Model of the environment)

- Terminologies:



In a Markov Decision Process:

s, s'	states
a	action
r	reward
$\mathcal{S}$	set of all nonterminal states
$\mathcal{S}^+$	set of all states, including the terminal state
$\mathcal{A}(s)$	set of all actions possible in state s
$\mathcal{R}$	set of all possible rewards
t	discrete time step
$T, T(t)$	final time step of an episode, or of the episode including time t
A_t	action at time t
S_t	state at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
$p(s', r \mid s, a)$	probability of transition to state s' with reward r, from state s and action a
$p(s' \mid s, a)$	probability of transition to state s', from state s taking action a



Vietnam Institute for Advanced Study in Mathematics

Bellman's equation

- State value function (for a fixed policy with discount)

$$V^\pi(s) = \sum_{a\in A} \pi(s,a)\, [\, \underbrace{R(s,a)}_{\text{Immediate}} + \gamma \underbrace{\sum_{s'\in S} T(s,a,s') V^\pi(s')}_{\text{Future expected sum of rewards}} \,]$$

- State-action value function (Q-function)

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') [\sum_{a'\in A} \pi(s',a') Q^\pi(s',a')]$$

- When S is a finite set of states, this is a system of linear equations (one per state)
- Belman's equation in matrix form: $V^\pi = R^\pi + \gamma\, T^\pi\, V^\pi$

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'\in S} p(s'|a,s) V^\pi(s')$$

Optimal Value, Q and policy

- Optimal V: the highest possible value for each s under any possible policy
- Satisfies the bellman Equation $V^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V^*(s') \right]$
- Optimal Q-function $Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V^*(s')$
- Optimal policy: $\pi^*(s,a) = \arg\max_a Q^*(s,a)$

Dynamic Programming (DP)

- Assuming full knowledge of Markov Decision Process
- It is used for planning in an MDP
- For prediction
  - Input: MDP (S,A,P,R,γ) and policy π
  - Output: value function v_π
- For controlling
  - Input: MDP (S,A,P,R,γ) and policy π
  - Output: Optimal value function v_* and optimal policy π_*

ONLINE LEARNING

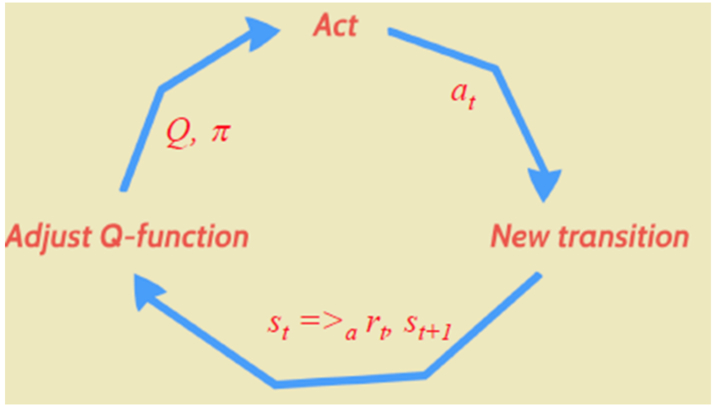
Model-free Reinforcement Learning

Partially observable environment, Monte Carlo, TD, Q-Learning

Vietnam Institute for
Advanced Study in Mathematics

Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from complete episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat:
  - Can only apply MC to episodic MDPs
  - All episodes must terminate



Vietnam Institute for Advanced Study in Mathematics

Monte-Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Value function is expected return

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

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

State Visit Monte-Carlo Policy Evaluation

- To evaluate state s
- At time-step t that state s is visited in an episode
  - Visiting state s: first or every time-step
- Increase counter N(s) = N(s) + 1
- Increate total return S(s) = S(s) + G_t
- Value is estimated by mean return V(s) = S(s)/N(s)
- By law of large number $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Incremental Monte-Carlo Updates

- Learning from experience
- Update V(s) incrementally after full game $S_1, A_1, R_2, ..., S_T$
- For each state S_t, with actual return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

- With learning rate

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

Temporal-Difference Learning

- Model-free: no knowledge of MDP
- Do not wait for episodes, learn from incomplete episode by bootstrapping
- Update value $V(s_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

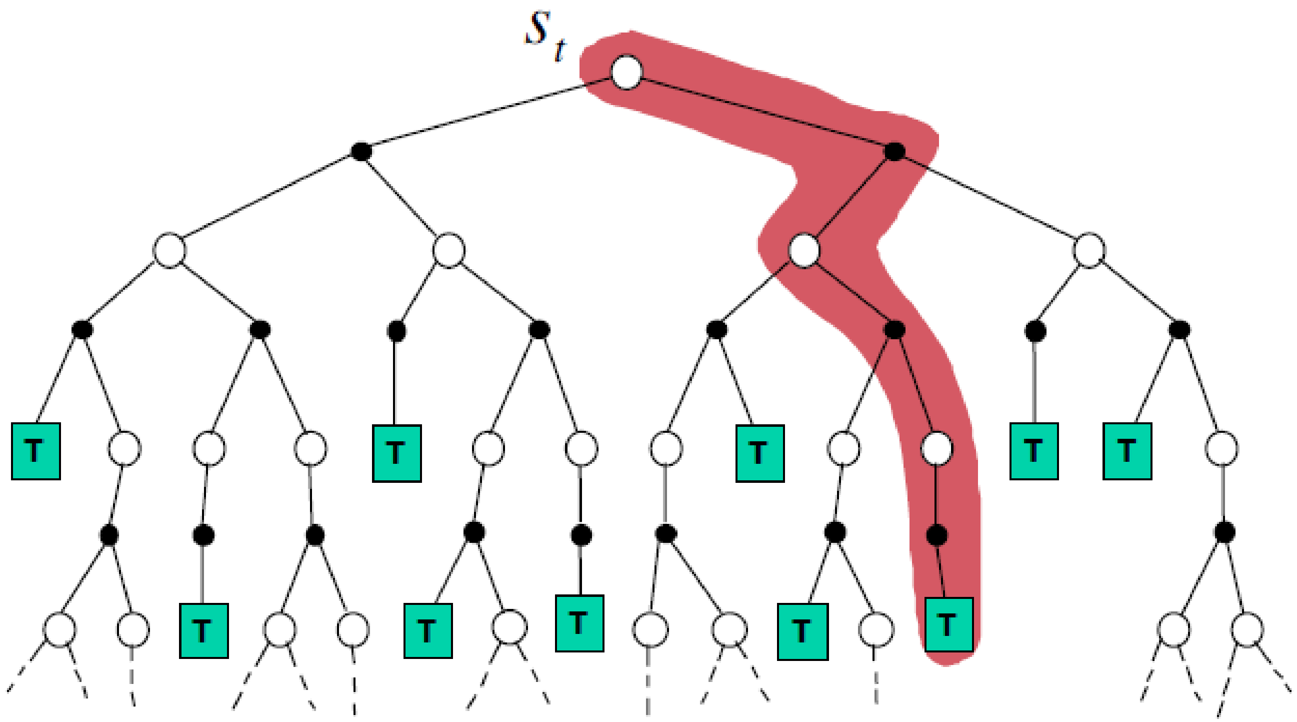
$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{(\overbrace{R_{t+1} + \gamma V(S_{t+1})}^{\text{TD Target}} - V(S_t))}_{\text{TD error}}$$

Monte-Carlo and Temporal Difference

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

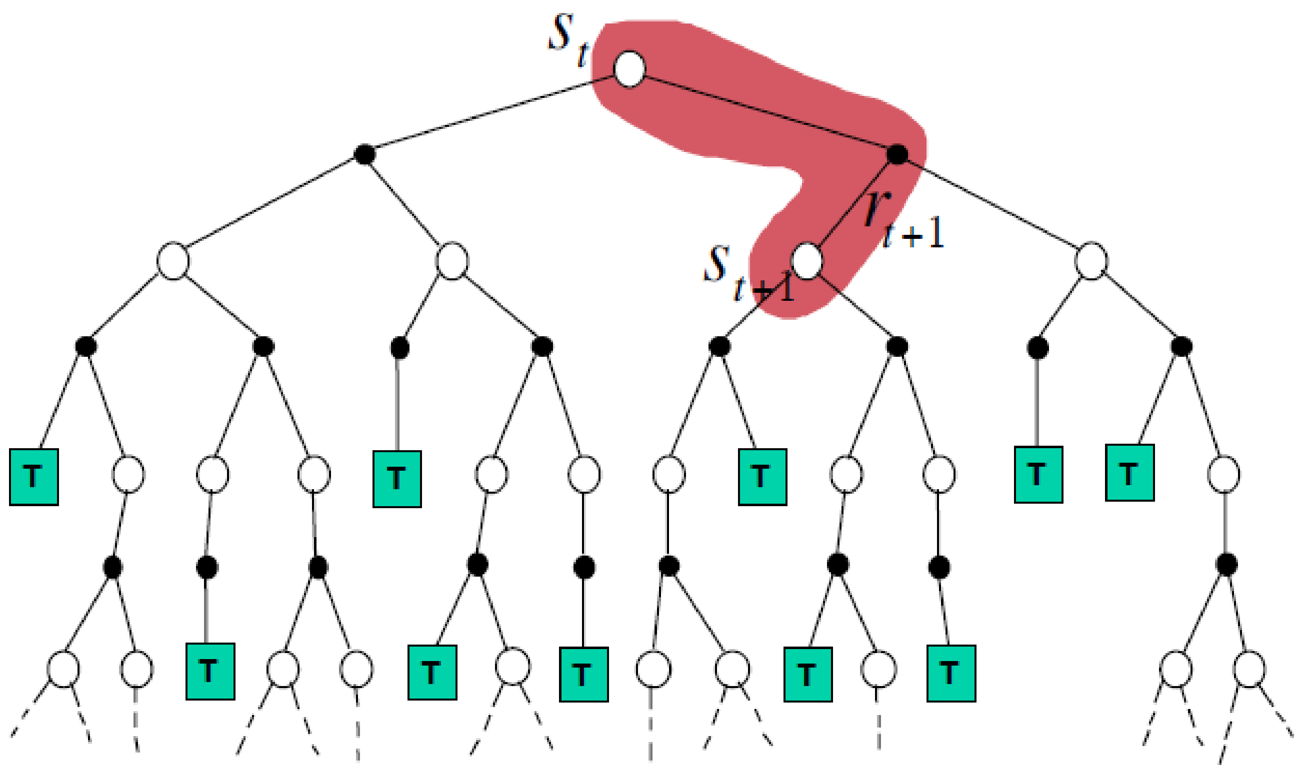
Monte-Carlo Backup

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



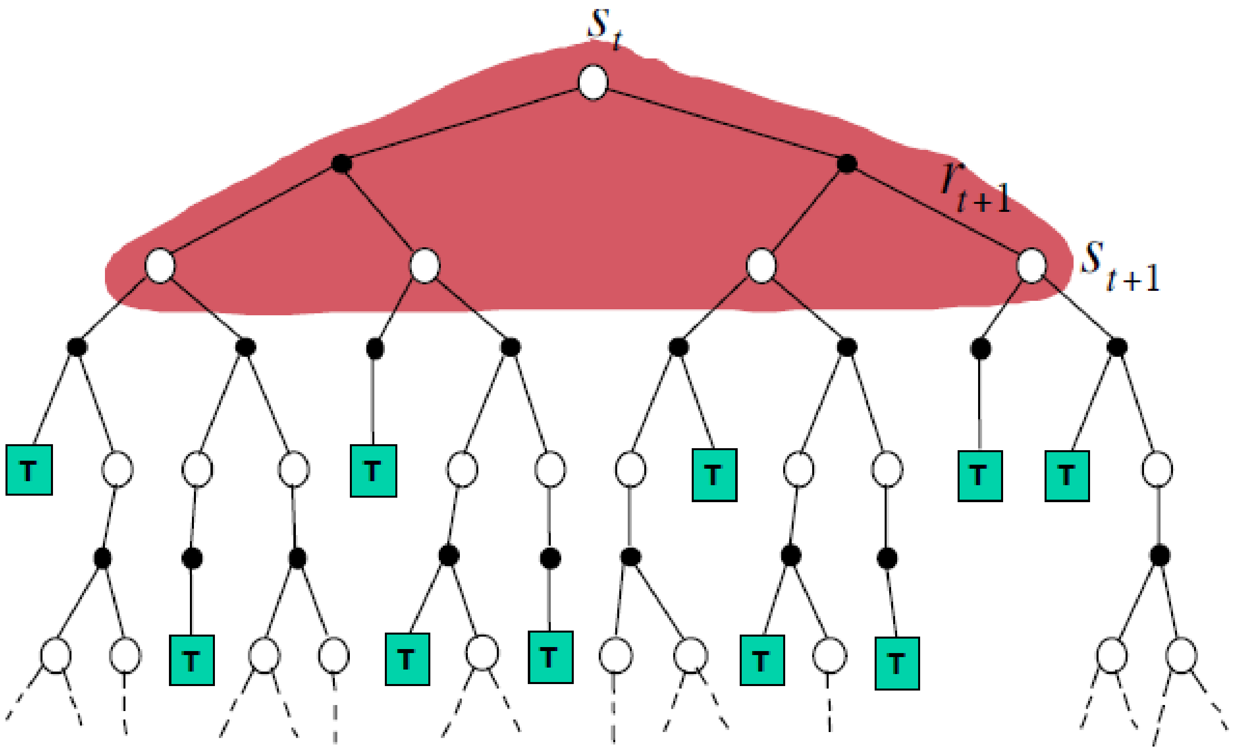
Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



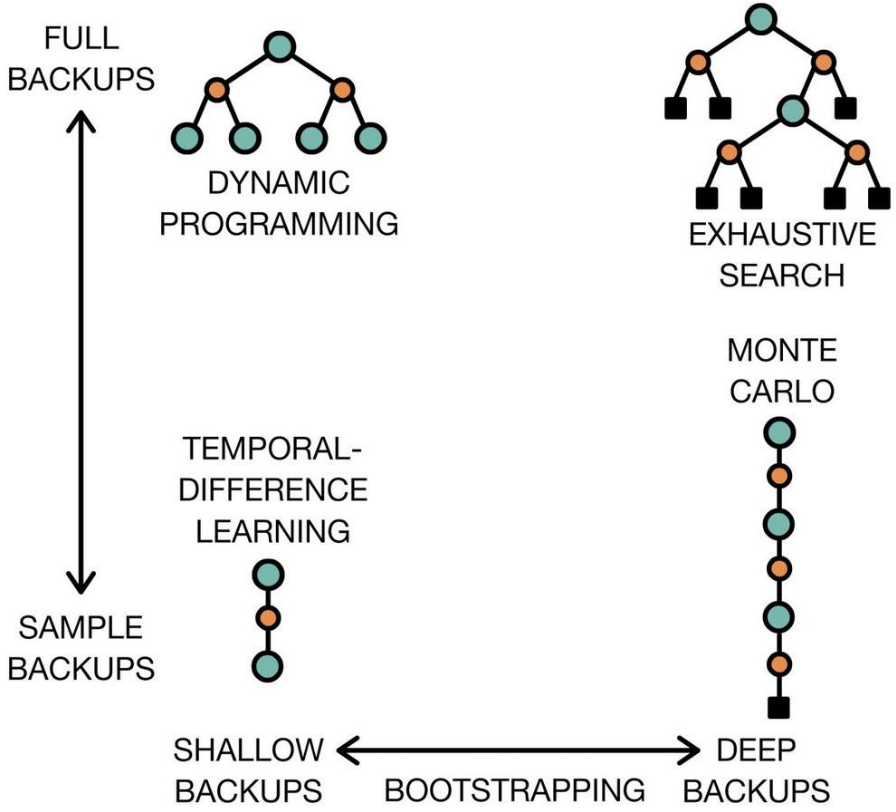
Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right]$$



Bootstrapping and Sampling

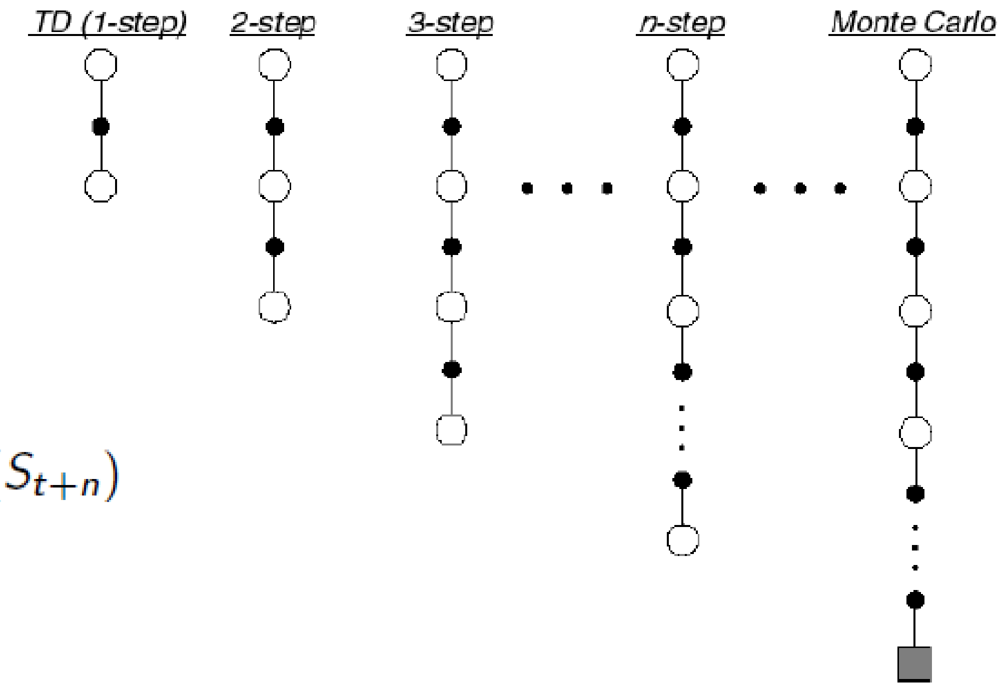
- Bootstrapping: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- Sampling: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples



N-step prediction

- n-step return

$$
\begin{array}{lll}
n = 1 & (TD) & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 & & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots & & \vdots \\
n = \infty & (MC) & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T
\end{array}
$$



- Define n-step return

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

- n-step temporal-difference learning

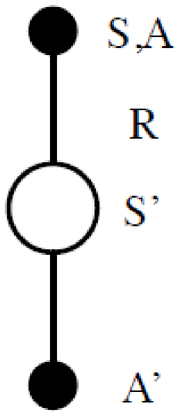
$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right)$$

Vietnam Institute for Advanced Study in Mathematics

On-policy Learning

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

- Advantage of TD:
  - Lower variance
  - Online
  - Incomplete sequence
- Sarsa:
  - Apply TD to Q(S,A)
  - Use policy improvement eg ϵ-greedy
  - Update every time-step



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

Sarsa Algorithm

Initialize any Q(s,a) and Q (terminate-state, null) =0

Repeat (for each episode)

  Initialize S

  Choose A from S using Q (eg ϵ-greedy)

  Repeat (for steps of episode)

    Take A, observe R, S'

    Chose A' from S' using Q (eg ϵ-greedy)

    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$

    $S \leftarrow S';\ A \leftarrow A';$

Until S is terminal

Off-Policy Learning

- Evaluate target policy π(a|s) to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While following policy μ(a|s)

$$\{S_1, A_1, R_2, \dots, S_t\} \sim \mu$$

- Advantages:
  - Learning from observing human or other agents
  - Reuse experience generated from old policies $\pi_1, \pi_2, \pi_3, \dots, \pi_{t-1}$
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

Q-Learning

- Off-policy learning action-value Q(s,a)
- No importance sampling is required
- Off policy: Next action is chosen by $A_{t+1} \sim \mu(\cdot|S_t)$
- Q-Learning: choose alternative successor $A' \sim \pi(\cdot|S_t)$
- Update $Q(S_t,A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

- Improve policy by greedy

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_{a'} Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

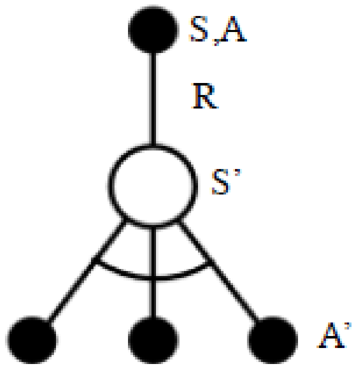
Q-Learning

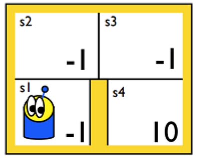
Update equation

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$$

Algorithm

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'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal





$\alpha = .7$

	⬆	⬇	⬅	➡
s_1	0	0	0	0
s_2	0	0	0	0
s_3	0	0	0	0
s_4	0	0	0	0

Q-Table

Q-Learning Table version

Visualization and Codes

- https://cs.stanford.edu/people/karpathy/reinforcejs/index.html

VALUE FUNCTION APPROXIMATION

State representation in complex environments

Linear Function Approximation

Gradient Descent and Update rules

Function approximation

- So far we have represented value function by a lookup table
  - Every state s has an entry V(s), or
  - Every state-action pair (s,a) has an entry Q(s,a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

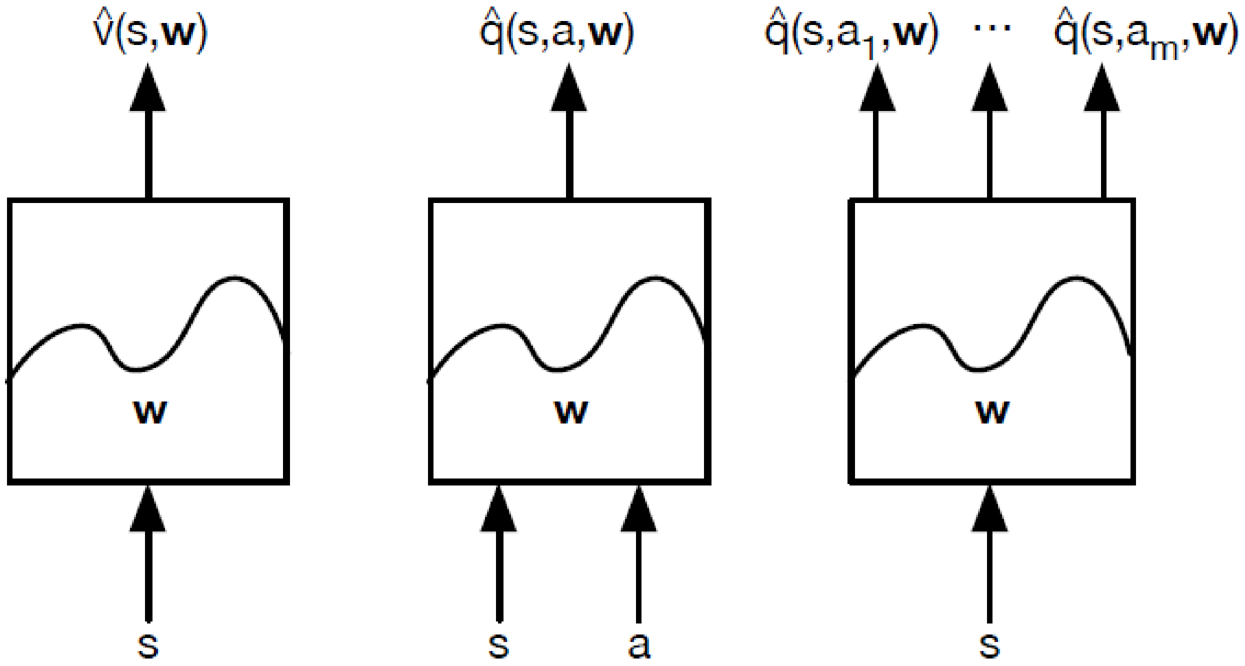
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states

Type of Value Function Approximation

- Differentiable function approximation
  - ❑ Linear combination of feature
    - Robots: distance from checking point, target, dead mark, wall
    - Business Intelligence Systems: Trends in stock market
  - ❑ Neural Network
    - Deep Q Learning
- Training strategies



Value Function by Stochastic Gradient Descent

- Goal: find parameter w minimizing mean-squared error between approximate value function and true state value on π

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S,\mathbf{w}))^2\right]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) \\ &= \alpha\mathbb{E}_\pi\left[(v_\pi(S) - \hat{v}(S,\mathbf{w}))\nabla_\mathbf{w}\hat{v}(S,\mathbf{w})\right]\end{aligned}$$

- Stochastic gradient descent sample

$$\Delta\mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S,\mathbf{w}))\nabla_\mathbf{w}\hat{v}(S,\mathbf{w})$$

Linear Value Function Approximation

- Represent state by a feature vector

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- Feature examples:
  - Distance to obstacle by lidar
  - Angle to target
  - Energy level of robot
- Represent a value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$$

Linear Value Function Approximation

- Objective function is quadratic in parameter w

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on global optimum
- Update rule:

Update = step-size x predict error x feature value

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \mathbf{x}(S)$$

Incremental Prediction Algorithms

- Value function $v_\pi(s)$ is assumed to be given by supervisors
- In reinforcement learning, there is only rewards instead
- In online learning (practice), a target for $v_\pi(s)$ is used

- For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

- For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

- For TD(λ), the target is the λ-return G_t^λ

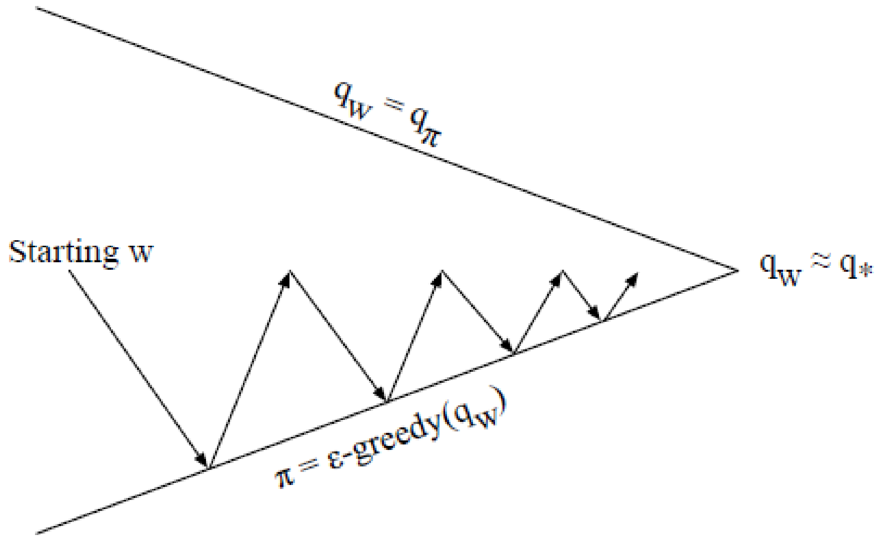
$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

Control with Value Function

- Policy evaluation Approximate policy evaluation

$$\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$$

- Policy improvement e-greedy policy improvement



Vietnam Institute for Advanced Study in Mathematics

Action-Value Function Approximation

- Approximate the action-value function (Q-value)

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value function and true value function with π

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$
$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state and action by a feature vector $\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$
- Represent action-value function by linear combination

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update $\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$

$$\Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

- Using target update in practice
  - MC $\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$
  - TD $\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$

POLICY GRADIENT

Policy-Based Reinforcement Learning

- Last part: value (and action-value) functions are approximate by parameterized function:

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- Generate policy from value function, e.g using e-gready
- In this part: directly parameterize policy

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

- Effective in high-dimensional or continuous action spaces

Policy Objective Functions

- Goal: given policy $\pi_\theta(s,a)$ with parameters θ, find the best θ
- Define objective function to measure quality of policy
  - In episodic environments, objective function is the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

  - In continuing environments, objective function is average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  - Or average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \mathcal{R}_s^a$$

where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Optimization

- Policy based RL is an optimization problem
- Find θ that maximize objective function J(θ)
- Policy gradient algorithms search for a local maximum in J(θ) by ascending gradient of the policy $\Delta\theta = \alpha \nabla_\theta J(\theta)$

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR},$ *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \; Q^{\pi_\theta}(s, a) \right]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameter by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

```
function REINFORCE
    Initialise θ arbitrarily
    for each episode {s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T} ~ π_θ do
        for t = 1 to T − 1 do
            θ ← θ + α∇_θ log π_θ(s_t, a_t) v_t
        end for
    end for
    return θ
end function
```

Reducing Variance using a Critic

- Monte-Carlo policy gradient has high variance
- A Critic is used to estimate action-value function

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic: Updates action-value function parameter w
  - Actor: Updates policy parameter θ
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a)\ Q_w(s, a)$$

Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value function approximation $Q_w(s, a)$
- Critic: Update w by linear TD(0)
- Actor: Update θ by policy gradient

```
function QAC
    Initialise s, θ
    Sample a ~ π_θ
    for each step do
        Sample reward r = R_s^a; sample transition s' ~ P_{s,·}^a
        Sample action a' ~ π_θ(s', a')
        δ = r + γQ_w(s', a') − Q_w(s, a)
        θ = θ + α∇_θ log π_θ(s, a)Q_w(s, a)
        w ← w + βδφ(s, a)
        a ← a', s ← s'
    end for
end function
```

Recap on Reinforcement Learning 02

- Online Learning
  - Model-free Reinforcement Learning
  - Partially observable environment,
  - Monte Carlo
  - Temporal Difference
  - Q-Learning

- Value Function Approximation
  - State representation in complex environments
  - Linear Function Approximation
  - Gradient Descent and Update rules

- Policy Gradient
  - Objective Function
  - Gradient Ascent
  - REINFORCE, Actor-Critic

Questions?

THANK YOU!

Vietnam Institute for
Advanced Study in Mathematics