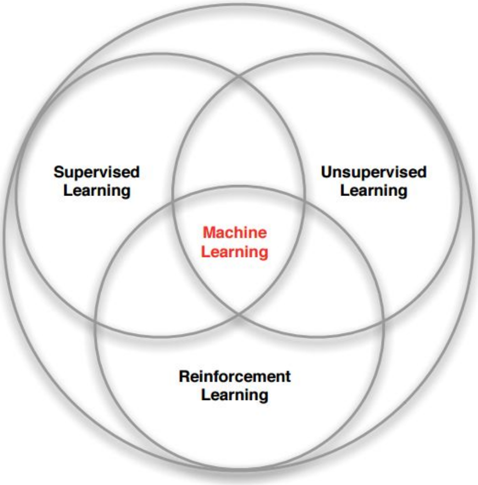
Introduction to Reinforcement Learning

Riashat Islam

Overview

Branches of Machine Learning

Reinforcement Learning



Motivations

- Goal-directed learning
- Learning from interaction with our surroundings
- What to do to achieve goals
- Foundational idea of learning and intelligence
- Computational approach to learning from interaction

Reinforcement Learning
Applications

- AlphaGo in the Game of Go
- Playing Atari Games
- Beating world champion in Backgammon
- Helicopter manoeuvres

Applications

Game of Go - Google DeepMind

https://www.youtube.com/watch?v=SUbqykXVx0A

https://www.youtube.com/watch?v=g-dKXOlsf98

Applications

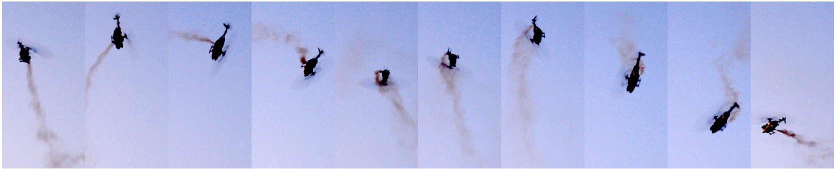
Playing Atari Games - Google DeepMind

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Applications

Helicopter Manouevres - Stanford

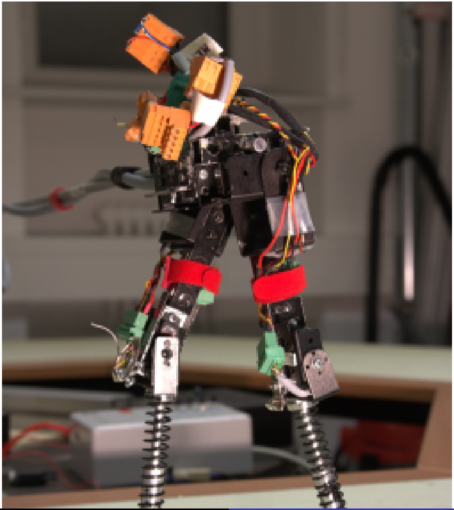
https://www.youtube.com/watch?v=VCdxqn0fcnE



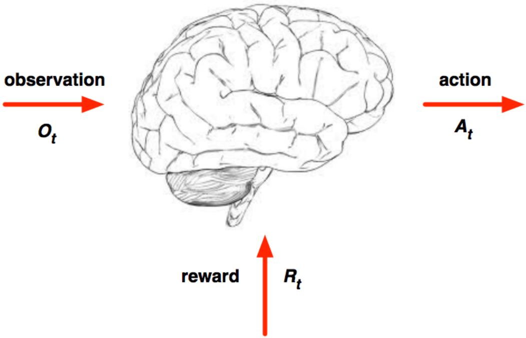
Applications

Robotics - Humanoid Robots

https://www.youtube.com/watch?v=370cT-OAzzM



Agent and Environment



Agent and Environment

- Learning agent interacting in the environment
- Learn what to do - to maximize reward
- Discover which actions will yield the most reward
- Actions may affect long term rewards
- Prefer actions already taken, or whether to try new actions

Key Ideas in RL

- Markov Decision Processes (MDPs)
- Rewards
- Policy of the agent
- Value Functions
- Environment Model and Dynamics

Sequential Decision Process and MDPs

- Decision-making to maximize utility/satisfaction
- Select actions to maximize goals
- Goal is to maximize expected cumulative reward

Goals and Rewards

- A reward signal R_t
- Feedback signal to indicate agent performance or behaviour
- Sequence of actions, states, and rewards
- Maximize cumulative reward of the agent

All goals can be described by the maximisation of the expected cumulative reward

- Do you agree with this statement?

Policy

- Describes the agent's behaviour
- A mapping from state to action
- Sequence of actions at each state to achieve goal
- Find the optimal policy - optimal behaviour of the agent

Value Functions

- Estimate how good it is for an agent to be in the current state
- Goodness - defined in terms of expected future rewards
- Value functions defined with respect to policies/behaviour

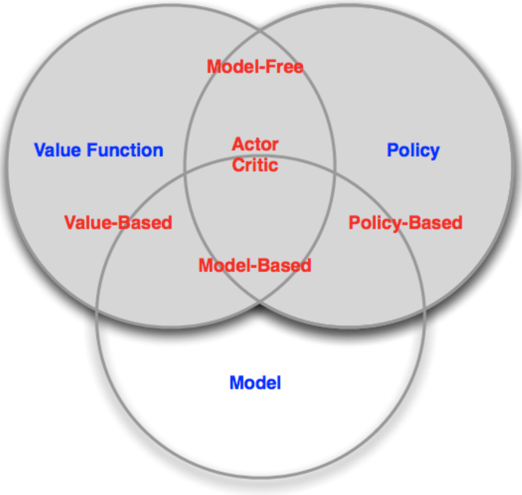
Model - Environment Dynamics

- The environment in which agent is located
- Defines state transitions
- The states the agent can go from current state
- The next immediate reward agent can achieve
- Model-Free : Unknown environment dynamics
- Model-Based: Known environment dynamics

Summary

- Agent interacts with the environment
- Agent to improve its policy
- Find optimal policy to maximize cumulative reward
- Policy also defined in terms of value functions
- Agent can be in an unknown or known environment

Overview


Model-Free
Value Function
Actor
Critic
Policy
Value-Based
Policy-Based
Model-Based
Model

Markov Decision Processes

Introduction to MDPs

- Model for sequential decision making under uncertainty
- Used to formulate RL problems
- Describes the environment of the RL agent
- Extension from Decision Theory - long term plan of actions

MDP Framework

- MDPs are discrete time state transition systems
- MDPs described by 5 components:
  - States: The state of the system needs to be observed by the decision maker when a decision has to be made.
  - Actions: Choose an action from the action set in the current state
  - Transition Probabilities $P(s_{t+1}|s_t, a_t$: Describes the dynamics of the world.
  - Reward $R(s)$ or $R(s, a)$: Real-valued reward that may depend on state, or both state and action.
  - γ is the discount factor $\gamma \in [0, 1]$

Markov Property

A state s_t is Markov if and only if:

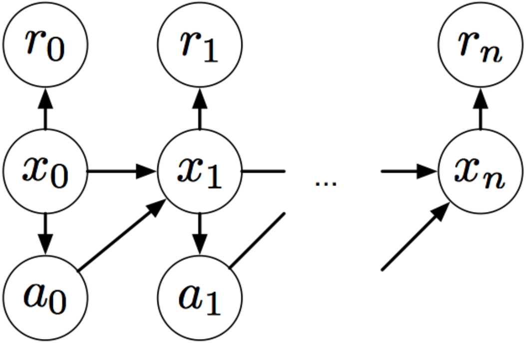
$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, s_2, s_3, ...s_t) \quad (1)$$

This means, the state captures all the relevant information from the history.

State transition probability $P(s_{t+1}|s_t)$ to define the transition probability from current state s_t to next or successor state s_{t+1}.

The conditional probability distribution of future states depends only on the present state, and not the sequence of events that preceded it

Graphical Model for MDPs



Return

The return G_t is the total discounted reward from time step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + .. = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2}$$

After $k+1$ time steps, the reward R is $\gamma^k R$

The discount factor γ is used to give more preference to immediate rewards over delayed rewards

Policy

Mapping from states to actions - at every state s_t, the agent can take action a_t that is defined by the policy $\pi(a|s)$

Policy defines the behaviour of the agent

Due to Markov property, the choice of action only depends on the current state s_t but not on any of the previous states

Policy is the distribution over actions given states

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{3}$$

Policy can be defined as the sequence of decision rules over the time steps

Value Functions

Estimate of how good it is for an agent to be in the current state

Goodness of a state defined in terms of future rewards or expected return

Value of a state under a policy π is $V^{\pi}(s)$. It is the expected return when starting in state s and following policy π.

$$v_{\pi}(s) =_{\pi} [G_t | S_t = s] \tag{4}$$

This is the same as:

$$v_{\pi}(s) =_{\pi} [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \tag{5}$$

Bellman Equation

Value functions satisfy recursive relationship (Dynamic Programming)

$$\begin{aligned} v_\pi(s) &= [G_t|S_t = s] \\ &= [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ..|S_t = s] \\ &= [R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ..)|S_t = s] \\ &= [R_{t+1} + \gamma G_t|S_t = s] \\ &= [R_{t+1} + \gamma v(s_{t+1})|S_t = s] \end{aligned} \tag{6}$$

Value functions can therefore be decomposed into immediate reward plus discounted value of successor state

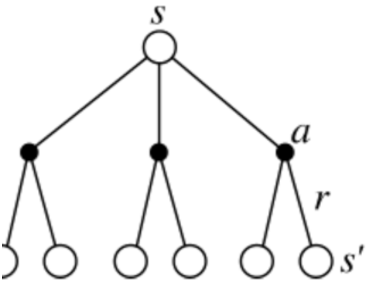
$$v_\pi(s) =_\pi [R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{7}$$

Bellman Equation for Value Functions

$$v_\pi(s) = [R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \tag{8}$$

Equivalently, we can write the value function as

$$v_\pi(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s') \tag{9}$$



Bellman Equation for Value Functions

We can therefore write Bellman equation for Value Functions as

$$v = R + \gamma P v \tag{10}$$

This Bellman equation could have been solved directly as

$$v = (I - \gamma P)^{-1} R \tag{11}$$

However, this direct solution is only possible for small MDPs

We look into iterative solutions for solving the Bellman equation (later!)

Value Functions and Policy

Action-Value Function Q^π

Similar to the value function V^π, we can also define the action-value function Q^π

It is the value of taking action a in state s under a policy π denoted as $Q^\pi(s, a)$.

It is the expected return starting from state s, taking the action a and following the policy π

$$Q^\pi(s, a) =_\pi [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \tag{12}$$

We can therefore write the Bellman equation for $Q^\pi(s, a)$

$$Q_\pi(s, a) =_\pi [R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{13}$$

Bellman Expectation Equations

We can write V^{π} in terms of Q^{π}

$$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) Q_{\pi}(s, a) \tag{14}$$

and

$$Q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss}^a V_{\pi}(s') \tag{15}$$

therefore, the value function is

$$V_{\pi}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_{\pi}(s')) \tag{16}$$

the action-value function is

$$Q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') Q_{\pi}(s', a') \tag{17}$$

Optimal Value Functions

The optimal value function is when we have the maximum value in all the states - maximum value function over all policies

$$V_*(s) = \max_\pi V_\pi(s) \tag{18}$$

Similarly, we also have the optima action-value function - the action-value is maximum over all policies

$$Q_*(s, a) = \max_\pi Q_\pi(s, a) \tag{19}$$

Optimal value functions means - we have the best performance of the agent in the environment

The agent behaves optimally in the environment - optimal policy

Optimal Policy (1)

An optimal policy of the agent which is better than or equal to all other policies

A policy π is better than another policy π' when its expected return is greater than or equal to that of π'.

A policy π is better than π' if and only if $V_\pi(s) \geq V_{\pi'}(s)$

All optimal policies π_* will therefore achieve the optimal $V_*(s)$ and $Q_*(s, a)$

Optimal Policy (2)

If we know the optimal action-value function, we can find the optimal policy. $Q_*(s, a)$ has maximum goodness over all states

If we know $Q_*(s, a)$, we know the optimal action to take at every state

Optimal policy π_* is defined as the optimal action to take at every state - following π_*, we can achieve goal state with maximum long term reward

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a \in A} Q_*(s, a) \\ 0, & \text{otherwise} \end{cases}$$

Bellman Optimality Equations

The optimal value functions are also recursively related by the Bellman optimality equations

$$v_*(s) = \max_a Q_*(s, a) \tag{20}$$

Bellman optimality for $Q_*(s, a)$ is

$$Q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s') \tag{21}$$

and $V_*(s)$ is given as

$$V_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s') \tag{22}$$

Finally, again writing $Q_*(s, a)$ as

$$Q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} Q_*(s', a') \tag{23}$$

Solving Bellman Optimality Equations

However, there exists no closed form solutions for the Bellman optimality equations

If we could solve for $V_*(s)$ and $Q_*(s, a)$ directly, RL problems would have been easy!

No closed form of $V_*(s)$ and $Q_*(s, a)$ for large MDPs

The Bellman optimality equations are non-linear

- We use iterative solution methods to find optimal value functions (next few lectures...)
  - Value Iteration and Policy Iteration
  - TD-Learning (Q-Learning and SARSA)

Policy Evaluation and Improvement

Policy Evaluation

- Use to evaluate a given policy π
- Solve the Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_\pi$
- Solve using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Iterative policy evaluation leads to convergence of v_π

Iterative Policy Evaluation

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')) \tag{24}$$

$$v^{k+1} = R^{\pi} + \gamma P^{\pi} v^k \tag{25}$$

- We therefore consider iterative solution methods
- Sequence of approximate value functions V_0, V_1, V_2
- Each successive approximation is therefore obtained by the Bellman equation for V^{π}
- V^{π} can be shown to converge as $k \to \infty$

Policy Improvement

- Given a policy π
  - Evaluate the policy π
  - $$v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \tag{26}$$
  - Policy improvement by acting greedily with respect to π
  - $$\pi^{'} = greedy(v_\pi) \tag{27}$$
- By iterative policy improvement, we want to find the optimal policy π^*
- This process is known as **policy iteration** which converges to π^*

Policy Improvement

We compute value functions of policies so that we can find better policies

For a given value function $V(s)$

- For state s, we want to know whether or not to change the policy
- We want to know how good it is to follow the current policy from state s which is $V^{\pi}(s)$

Consider policy improvement $\rightarrow$ select a in s and thereafter follow existing policy π. In other words, we compute $Q^{\pi}(s,a)$

- See whether $Q^{\pi}(s,a)$ is better or less than $V^{\pi}(s)$.

Policy Improvement

As before, first evaluate the policy by computing value functions

$$Q^{\pi}(s,a) = E_{\pi} r_{t+1} + \gamma V^{\pi}(s_{t+1}|s_t = s, a_t = a \tag{28}$$

- Improve the policy by acting greedily

$$\pi'(s) = \arg\max_{a \in A} Q^{\pi}(s,a) \tag{29}$$

- This improves the value from any state s over one step

$$Q^{\pi}(s,\pi'(s)) = \max_{a \in A} Q^{\pi}(s,a) \geq Q^{\pi}(s,\pi(s))$$

- This therefore improves the value function

$$Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s) \tag{30}$$

Policy Improvement

This improvement in policy by acting greedily is known as the policy improvement theorem

$$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \tag{31}$$

This means that the new policy π' must be as good as or better than the old policy π. That is, it must obtain greater or equal expected return from all states $s \in S$

$$V^{\pi'}(s) \geq V^{\pi}(s) \tag{32}$$

Policy Improvement

- If the policy improvements stop, i,e

$$Q^{\pi}(s, \pi'(s)) = \max_{a \in A} Q^{\pi}(s, a) = Q^{\pi}(s, \pi(s)) = V^{\pi}(s) \quad (33)$$

- This means that the Bellman optimality equation has been satisfied

$$V^{\pi}(s) = \max_{a \in A} Q^{\pi}(s, a) \quad (34)$$

- This means, we have reached the optimal value function

$$V^{\pi}(s) = V^{*}(s) \quad (35)$$

- Therefore, $\pi(s)$ is the optimal policy. $\pi = \pi^{*}$

Policy Iteration

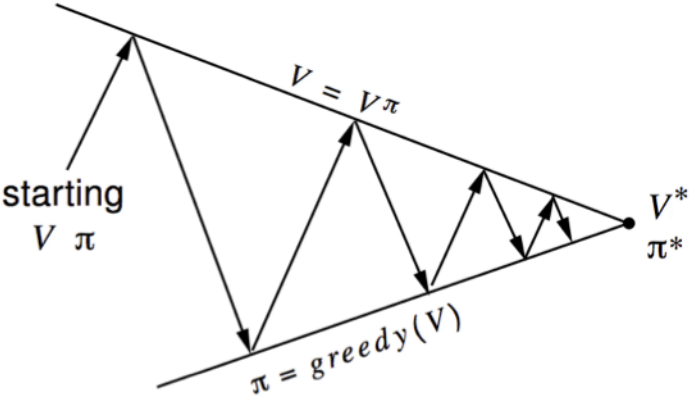
Policy Evaluation

- Iterative policy evaluation
- Estimate $V^{\pi} \rightarrow$ for given π compute V^{π}

Policy Improvement

- Greedy policy improvement
- Generate $\pi' \geq \pi$

Policy Iteration



Value Iteration

Instead of the two step process of policy iteration, we can instead use value iteration

- Goal is again to find optimal policy π
- Iteratively compute the Bellman optimality backup

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

a “sweep”

Value Iteration

$V_1 \rightarrow V_2 \rightarrow \ldots\ V^*$

- Using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in S$
  - Update $V_{k+1}(s)$ from $V_k(s')$
- This can be proved to converge to V^* (not included here)
- Unlike policy iteration, in value iteration there is no explicit greedy policy improvement step
- In other words, by improving the value function iteratively, policy itself is improved
- $$V_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s')) \tag{36}$$

Model Free Prediction

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 idea: $value = meanreturn$
- Caveat: can only apply MC to episodic MDPs (i,e. All episodes must terminate)

Monte-Carlo Policy Evaluation

- Goal : learn V^π from episodes of experience under policy π.
- Recall that the return is the total discounted return

$$v_t = r_{t+1} + \gamma r_{t+2} + ...\gamma^{T-1} r_T \tag{37}$$

- Recall that the value function is the expected return

$$V^\pi(s) = E_\pi[v_t | s_t = s] \tag{38}$$

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

Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

MC and TD

- Goal: learn V^{π} online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(s_t)$ towards actual return v_t

$$V(s_t) = V(s_t) + \alpha(v_t - V(s_t)) \tag{39}$$

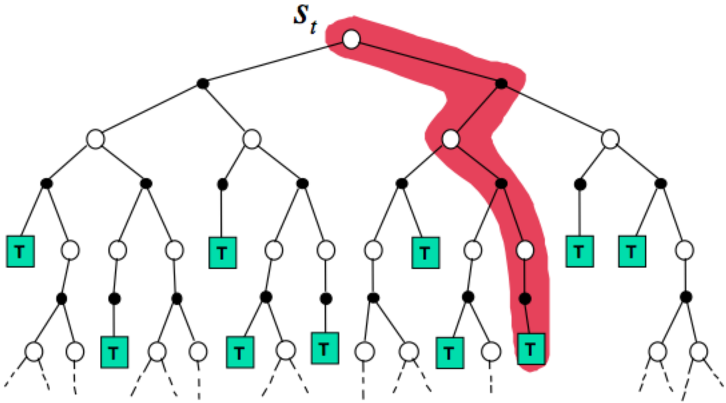
- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(s_t)$ towards estimated return $r_{t+1} + \gamma V(s_{t+1})$
  - $V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1} - V(s_t)))$
  - $r_{t+1} + \gamma V(s_{t+1})$ is called the TD target
  - $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is called the TD error

Advantages and Disadvantages of MC vs TD

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before retun 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

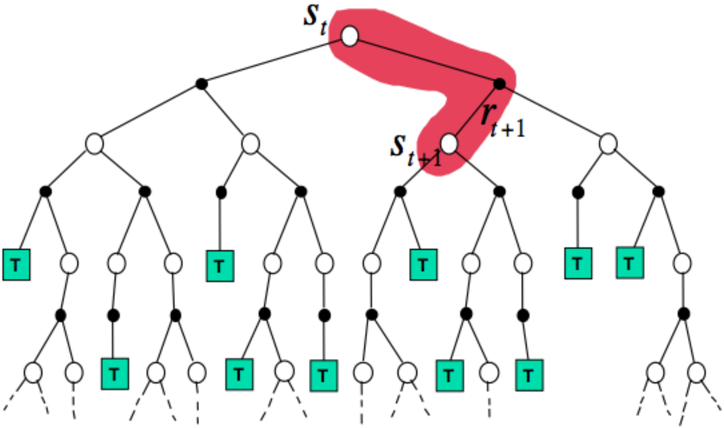
Monte-Carlo Backup

$$V(s_t) \leftarrow V(s_t) + \alpha\,(v_t - V(s_t))$$



Temporal-Difference Backup

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



Model Free Control

On and Off-Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

ϵ-Greedy Exploration

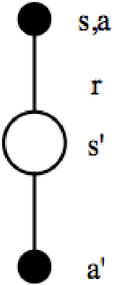
- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

$$\pi(s, a) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

MC vs TD Control

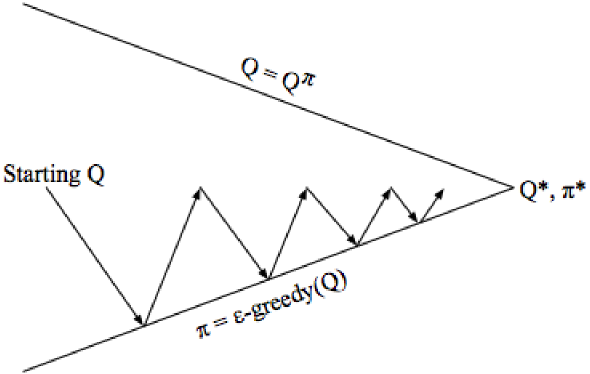
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(s, a)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Updating Action-Value Functions with Sarsa



$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q(s',a') - Q(s,a) \right)$$

On-Policy Control with Sarsa



Every time-step:

Policy evaluation Sarsa, $Q \approx Q^\pi$

Policy improvement ϵ-greedy policy improvement

Sarsa Algorithm for On-Policy Control

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

Off-Policy Learning

- Evaluate target policy $\pi(s,a)$ to compute $V^{\pi}(s)$ or $Q^{\pi}(s,a)$
- While following behaviour policy $\mu(s,a)$

$$\{s_1, a_1, r_2, ..., s_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Q-Learning

Off-Policy Control with Q-Learning

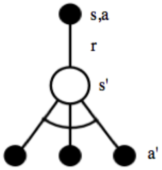
- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(s_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(s_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned}
& r_{t+1} + \gamma Q(s_{t+1}, a') \\
=& r_{t+1} + \gamma Q(s_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(s_{t+1}, a')) \\
=& r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a')
\end{aligned}$$

Q-Learning Control Algorithm



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

Theorem

Q-learning control converges to the optimal action-value function, $Q(s,a) \rightarrow Q^*(s,a)$

Q-Learning Algorithm for Off-Policy Control

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

Questions