Reinforcement Learning:

A Tutorial.

L. O. Olagoke[1]

[1]School of Computer And Communication Systems
Ecole Polytechnique Federale de Lausanne

Statistical Seq Processing
June 2017

Table of Contents

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Table of Contents

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Reinforcement Learning: What is it about?

It is about (the agent) learning from interactions (with environment) to achieve a goal by taking actions aimed at maximizing numerical reward. It is simply learning the act of decision making.

The learner and decision maker is called the agent. The thing it interacts with, comprising of everything outside the agent is called the environment. The learner is not told which action to take as in most forms of machine learning but instead must discover which actions maximize a reward by trying them out. The environment respond to those actions and present a new situation.

Difference from other Machine Learning Paradigms

How does Reinforcement Learning differs from other Machine learning paradigms?

- There is no supervision only a reward signal
- Trial-and-error search and delayed reward
- Feedback is delayed, not instantaneous
- Time matters (sequential non i.i.d data points)
- Agent action affect the response it gets
- RL is defined by characterizing a learning problem and not by characterizing a learning method. The idea here is to characterize the important aspect of the real problem facing the agent interacting with the environment to achieve a goal through its actions. The goal of the agant must relate to the state of the environment.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Reward

A Reward is a scalar feedback signal. It indicates how well the agent is doing at step t. The agent goal is to maximise cumulative reward. Reinforcement Learning is based on reward hypothesis.

Definition - Reward Hypothesis

All goals can be described by the maximization of expected cumulative reward. ??

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Agent and Environment

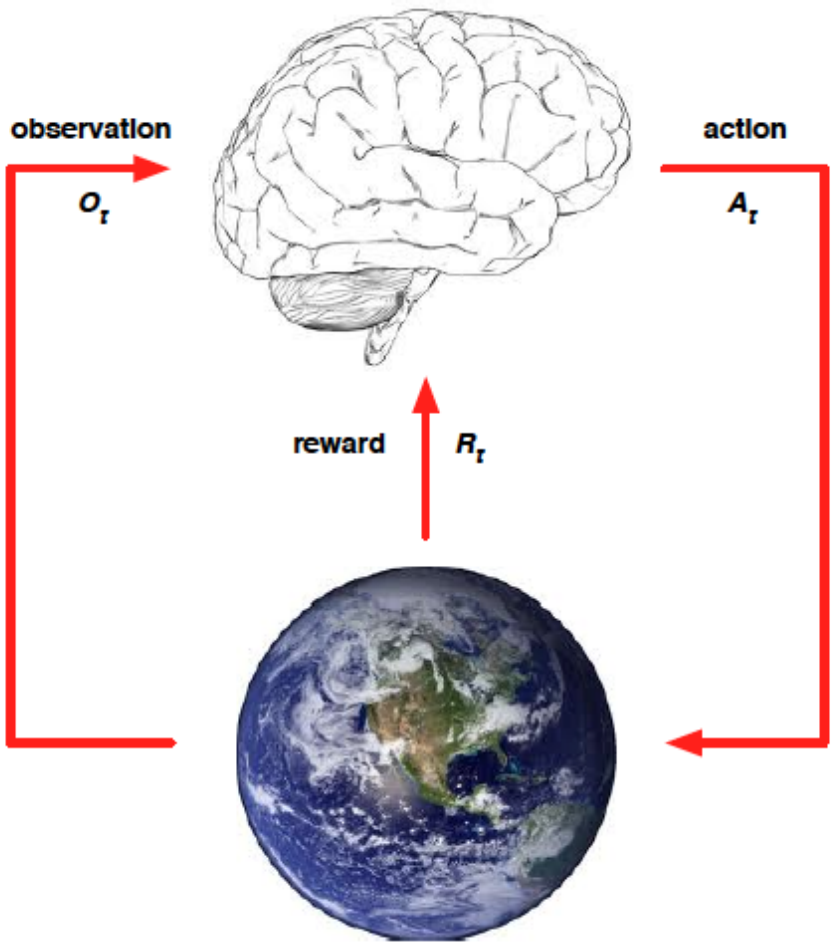


Figure: Agent and Environment

- 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.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Environment State

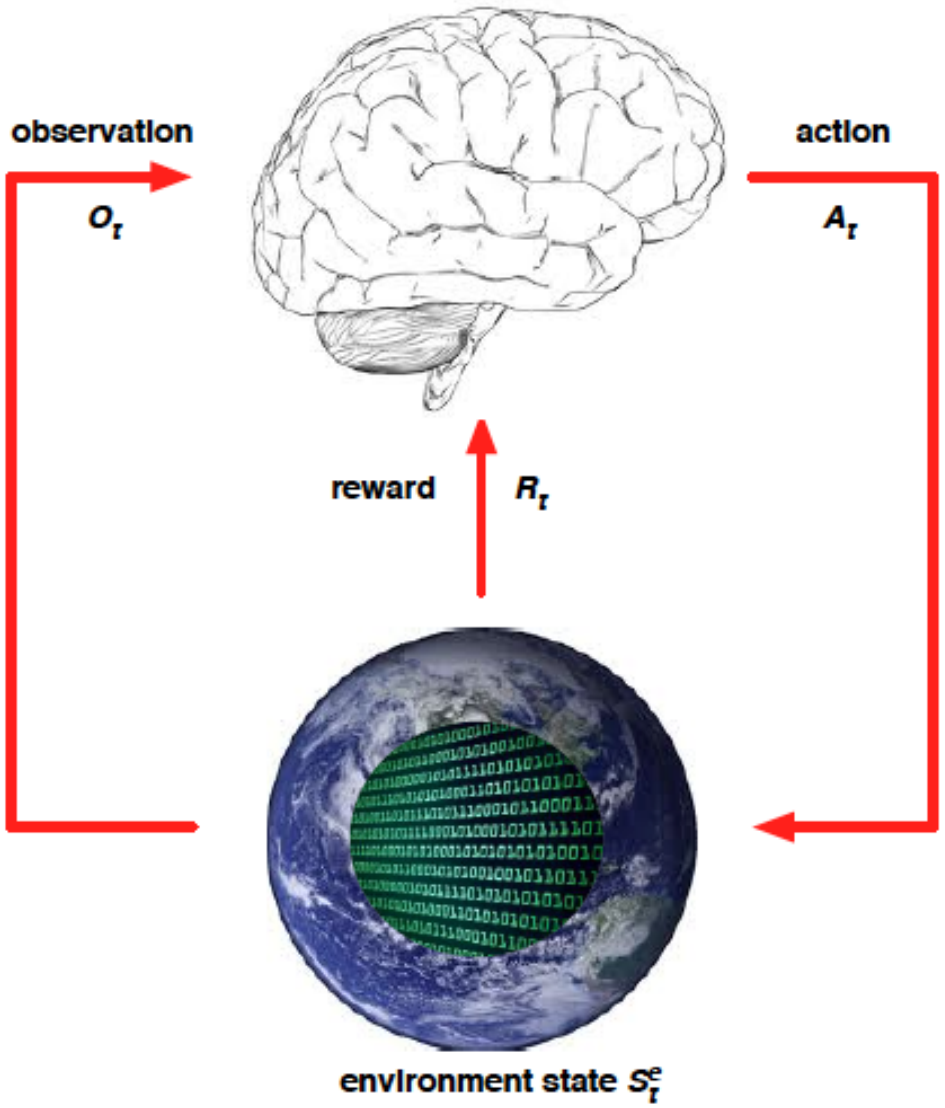


Figure: Agent and Environment

- The environment sate S_t^e is the environment's private representation. It is whatever process/algorithm/data by which the environment chooses its next observation or reward.
- The environment state is not usually visible to the agent.
- Even if the S_t^e is visible to the agent, it may contain irrelevant information.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Agent State

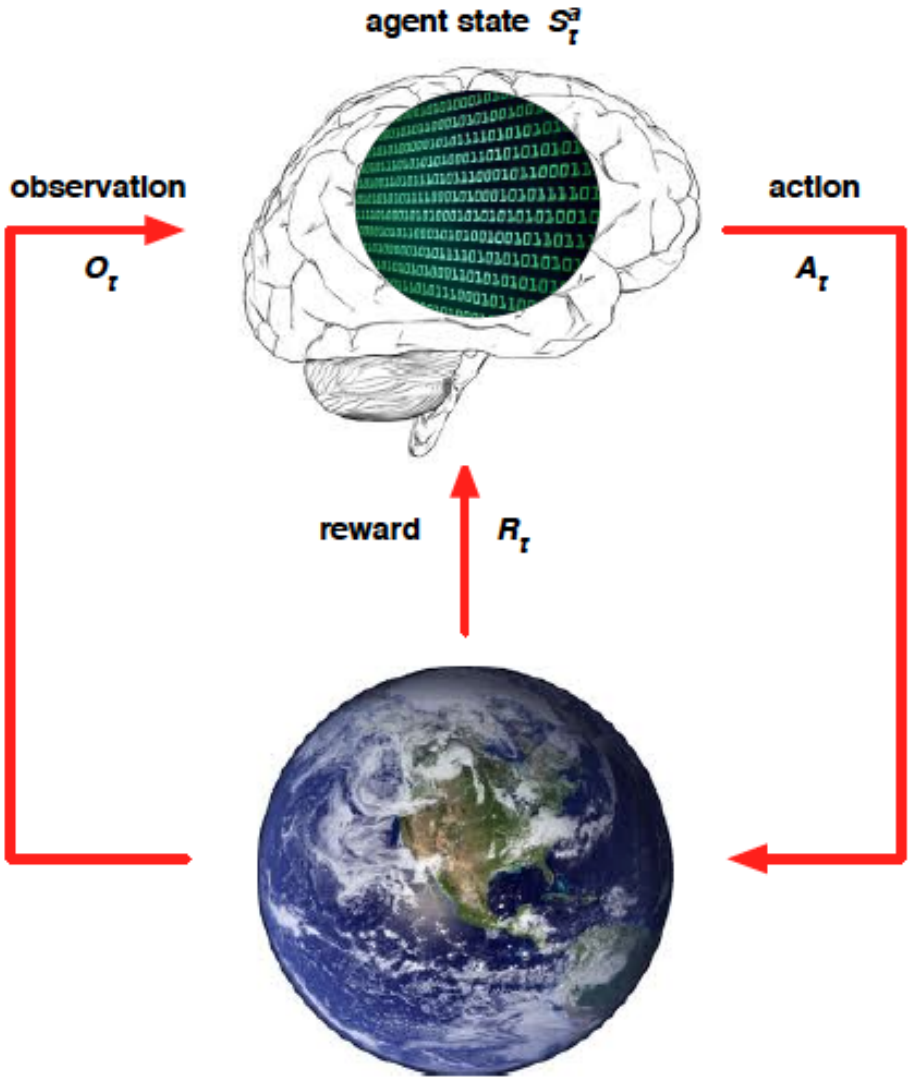


Figure: Agent and Environment

- The agent sate S_t^a is the agent's private representation.
- It is the information used by learning algorithm
- It can be any function of history

$$S_t^a = f(H_t)$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Partially And Fully Observable Environments

- Full observability: agent directly observes environment state.
- Agent State = Environment State = Markov State
- This defines Markov Decision Process

- Partial observability: agent indirectly observes environment.
- Agent state $\neq$ Environment state
- This defines a Partially Observabe Markov Decision Process (POMDP)

In POMDP, the agent must construct its own state representation S_t^a. Using for E.g.

- Complete History : $S_t^a = H_t$
- Beliefs of the environment : $S_t^a = (\mathbb{P}[S_t^e = s^1], ... \mathbb{P}[S_t^e = s^n])$
- Recurrent neural network. $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

RL Agents Components

- Policy.

Definition

Agent behaviour function. Roughly, we say it is a mapping from perceived states to actions to be taken when in those states. It is sufficient enough to determine agent's behaviour.

- deterministic policy: action taken in state s under deterministic policy π.

$$a = \pi(s)$$

- Stochastic policy: probability of taking action a in state s under stochastic policy π.

$$\pi(a|s) = P(A_t = a|S_t = s)$$

RL Agents Components

- Policy.
- Value Function

Definition

Value function of a state is the total reward an agent is expected to accumulate over the future starting from that state. It specifies how good a action or function is. E.g

$$V_\pi = E_\pi \left(R + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... | \ S_t = s \right)$$

RL Agents Components

- Policy.
- Value Function
- Model

Definition

Model: is the agent representation of the environment. It is whatever an agent can use to predict how the environment will respond to its actions.

- $\mathcal{P}$ predicts the next state [1]:

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

- $\mathcal{R}$ predicts the next immediate reward.

$$\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

[1] We assume that the Markov property is satisfied

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Categorizing RL Agents (1)

- Value Based
  - Value Function
  - No Policy
- Policy based
  - Policy
  - No Value Function
- Actor critic
  - Policy
  - Value Function

Categorizing RL Agents (2)

- Model Free
  - Policy and/or Value function
  - No model
- Model Based
  - Policy and/or Value Function
  - Model
- Actor critic
  - Policy
  - Value Function

Introduction

In this slide

Introduction

In this slide
the text will be partially visible

Introduction

In this slide
the text will be partially visible
And finally everything will be there

Table of Contents

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Markov Decision Process - Definition

A Markov process is a memroryless random process or (or sequence of state in our case) that satisfies the Markov property.

Definition

A Markov process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$ where

- $\mathcal{S}$ is a (finite) set of states. Keeping finite states enables us to work in terms of sums and probabilities rather than integrals and probability densities, but the argument could easily be extended to include continuous states and rewards
- $\mathcal{P}$ is a state transition probability matrix:

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Markov Reward process

A Markov Reward process is a Markov chain with values.

Definition

A Markov Reward process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states.
- $\mathcal{P}$ is a state transition probability matrix: $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- γ is discount factor, $\gamma \in [0,1]$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Return

Definition

The Return $\mathcal{G}_t$ is the total discounted reward from time-step t

- $\mathcal{G}_t = R_t + \gamma R_{t+2} + ... = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1}$

- γ is the discount rate $\gamma \in [0,1]$
- value of receiving reward R after k+1 step is $\gamma^k R$
- This value immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation.
  - γ close to 1 leads to "far-sighted" evaluation.

Why Discount?

- Mathematically convenient to discount reward
- Avoid infinite returns in cyclic Markov process
- This value immediate reward above delayed reward.
- uncertainty about the future may not be fully represented.
- Animal/Human behavior show reference for immediate reward
- it is possible to use undiscounted Markov reward $\gamma = 1$. e.g if all sequences terminate.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Value Function

The value function v(s) gives long term value of state s.

Definition

The state value Function v(s) of an MRP is the expected return starting from state s

- $v(s) = \mathbb{E}[\mathcal{G}_t | S_t = s]$

Value Function can be decomposed into 2 parts:

- Immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

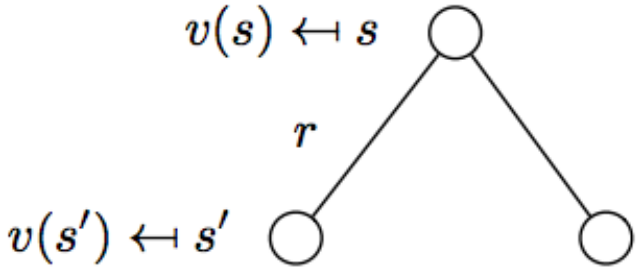
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Bellman equation for Markov Reward Process (MRP)

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + ...) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (\mathcal{G}_{t+1}) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]
\end{aligned}
$$

Bellman equation for MRP 2

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$



$$v(s) = R_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} v(s')]$$

The Bellman equation is a linear equation. It can be solved directly for small MRPs:

$v = \mathcal{R} + \gamma \mathcal{P} v$

$(I - \gamma \mathcal{P}) v = \mathcal{R}$

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

There are many iterative process for large MRPs:

- Dynamic programming
- Monte Carlo Evaluation
- Temporal Difference learning

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Markov Reward Process

A Markov Decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

Definition

A Markov Decision process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states.
- $\mathcal{A}$ *is a finite set of actions*
- $\mathcal{P}$ is a state transition probability matrix: $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = A\right]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}^{a}_{s} = \mathbb{E}[R_{t+1} | S_t = s, A_t]$
- γ is discount factor, $\gamma \in [0,1]$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Policies

Definition

A Policy is a distribution over action given states.

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

- A policy fully defines the behavior of an agent
- MDP policies depend on the current state (not the history)
- Policies are stationary (time-independent) :

$$A_t \sim \pi(.|S_t), \forall > 0$$

Value Function

state-value function

A state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following a policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[\mathcal{G}_t | S_t = s \right]$$

The Bellman equation for $v_\pi(s)$ is:

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right]$$

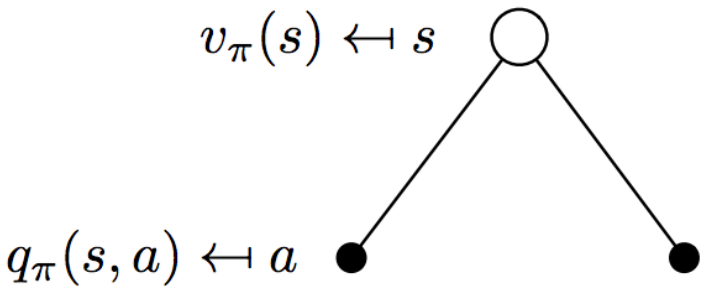
Action-Value Function

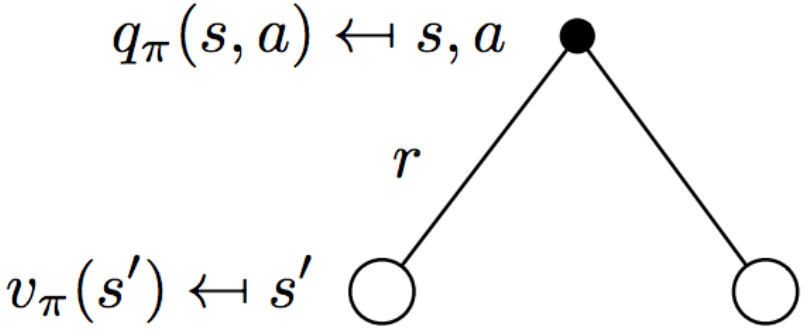
The action-value function $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π.

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\mathcal{G}_t | S_t = s, A_t = a \right]$$

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

Bellman Equation for V^π and Q^π



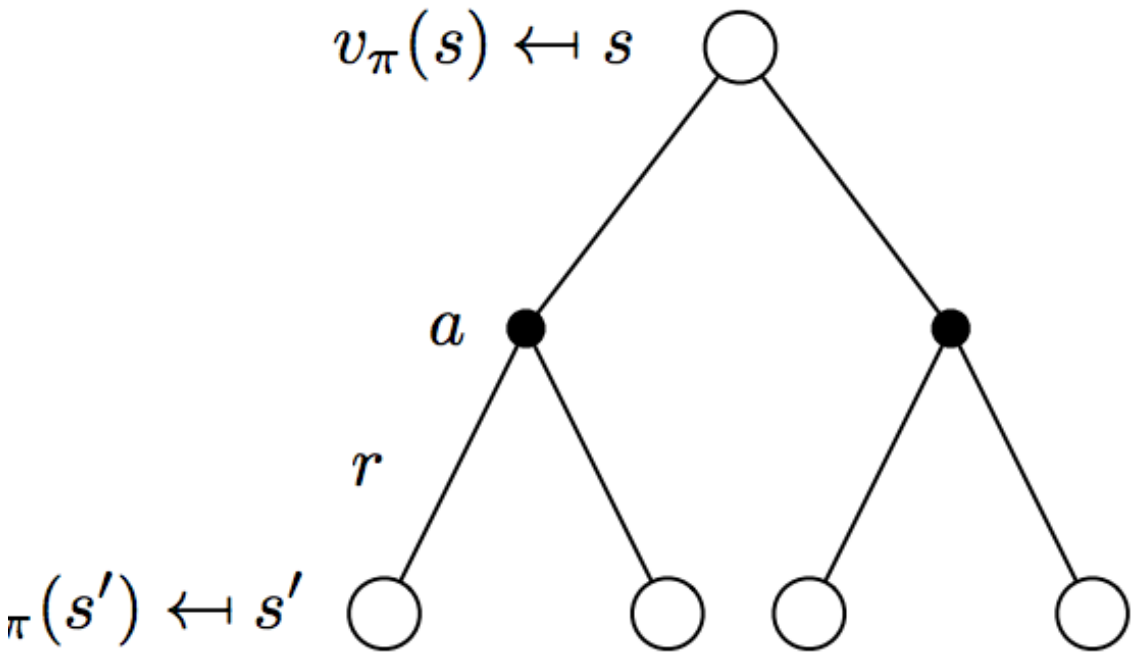


$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a(a|s) v_\pi(s')$$

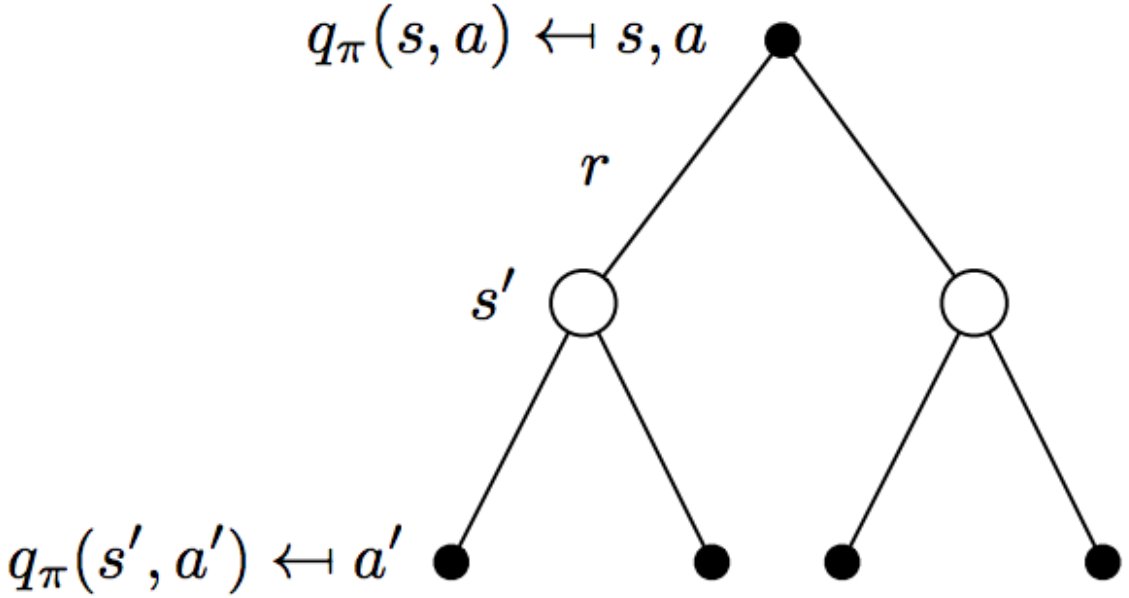
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Bellman Equation for V^{π} (2)



$$v_\pi(s) = \sum_{a \in A} \pi(s|a)(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_\pi(s'))$$

Bellman Equation for and Q^{π} (2)



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \sum_{a' \in A} \pi(s'|a') q_\pi(s',a')$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Optimal Value Function

Optimal Value Function

The optimal state-value function $v_*(s)$ is the maximum function over all policies: $v_*(s) = \max_\pi \; v_\pi(s)$

The action-value function $q_*(s,a)$ is the maximum action-value function over all policies. $q_*(s,a) = \max_\pi \; q_\pi(s,a)$

- Optimal Value Function specifies best possible performance in the MDP. The MDP is solved if we know it.
- An optimal policy can be found by maximizing over $q_*(s,a)$.

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\text{argmax}} \; q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Bellman Optimality equations

- Bellman optimality equation for v$_*$: $v_* = \max_s \; q_*(s,a)$
- Bellman optimality for Q*: $q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s')$
- Bellman optimality V* 2-step case: $v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s')$
- Bellman optimality Equation for Q *: $q_*(s,a) = \mathcal{R}_a^s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a'}(s', a')$
- Bellman optimality equation is non-linear. And it has no general closed form solution
- Many iterative solution methods.

Examples

Q−learning

Sarsa

Learning Vs Planning, Exploration Vs Exploitation

- reinforcement Learning:
  - The environment is initially unknown
  - The agent interact with the environment
  - Agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent perform computation with its model
  - The agent improves its policy
  - a.k.a deliberation, reasoning, search
- Exploration: involves finding more information about the environment
- Exploitation involves exploiting known information to maximise reward
- Balance between Exploitation and exploration should be sought
- Prediction : evaluate future, given a policy
- control : Optimise future , by finding best policy

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Partially Observable Markov Decision Process

A Partially Observable Markov Decision process is an MDP with hidden states. It is a hidden Markov model with actions.

Definition

A Partially Observable Markov Decision process is a tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$ where

- $\mathcal{S}$ is a (finite) set of states.
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{O}$ *is a finite set of observations*
- $\mathcal{P}$ is a state transition probability matrix: $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{Z}$ is an observation function. $\mathcal{Z}^{a}_{s'o} = \mathbb{P}[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$
- $\mathcal{R}$ is a reward function: $\mathcal{R}^{a}_{s} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is discount factor, $\gamma \in [0,1]$

Belief states

Remark

The goal in POMDP is to find a mapping from observations (not states) to actions. POMDPs model the information available to the agent by specifying a function from the hidden state to the observables, just as in an HMM.

History

A History H_t is a sequence of actions, observations, and rewards.
$A_0, O_1, R_i ... A_{t-1}, O_t, R_t$
A belief state b(h) is a probability distribution over states, conditioned on the history h.

$$b(h) = (\mathbb{P}[S_t = s^1, H_1 = h], , ... \mathbb{P}[S_t = s^t, H_t = h])$$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Table of Contents

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Temporal Difference Learning

- Temporal difference learning can be used to estimate the value function of an unknown MDP. TD is a combination of MC and dynamic programming ideas.
- TD methods can learn directly from raw experience without a model of the environment dynamics.
- TD learns from incomplete episodes , by bootstrapping.
- Updates a guess towards a guess

Monte Carlo Policy And Incremental mean

- Goal: Lean v_π from episodes under policy π.
  $S_1, A_1, R_2..., S_k \sim \pi$
- Recall,that the return is the total discounted reward:
  $G_t = R_{t+1} + \gamma R_{t+s} + ...\gamma^{T-1} R_T$
- The value function is the expected return : $V_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$
- Monte Carlo policy evaluation uses empirical mean return instead of expected return. First time vs Every time Visit
- Incremental mean μ_1, μ_2, μ_3 of a sequence can be computed incrementally: $\mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
- In Monte carlo, for each state S_t and return G_t. $N(S_t) \leftarrow N(S_t) + 1$
  $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$
- In non-stationary problem, it can be useful to track a running mean i.e forget old episodes: $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MC and TD

- Goal: Lean v_π online under policy π.
- Increment every visit Monte Carlo.
  - Update value $V(S_t)$ toward actual return G_t.

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

- Simplest temporal difference learning : TD(0)
  - Update value $V(S_t)$ toward estimated return: $R_{t+1} + \gamma V(S_{t+1})$

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

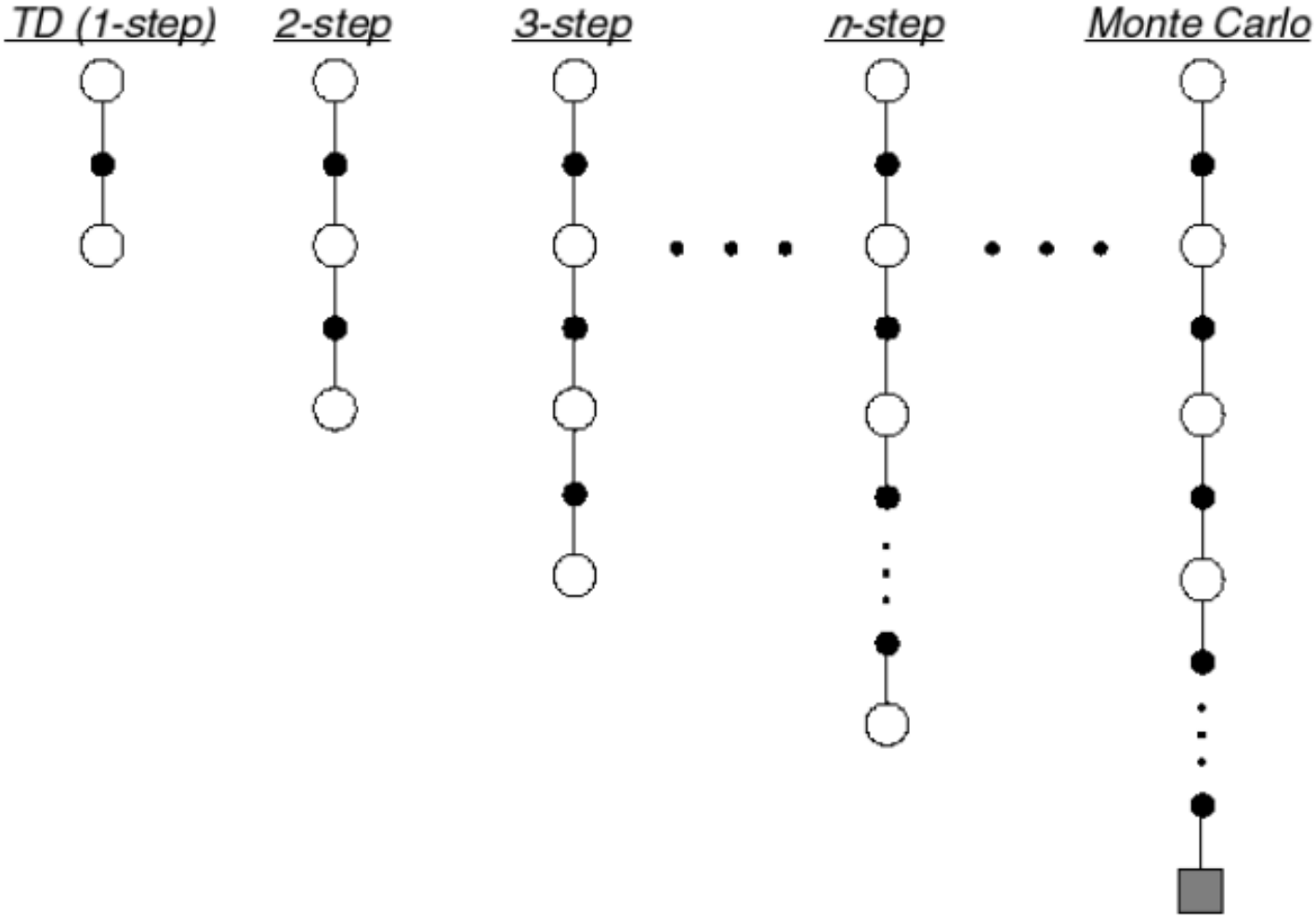
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

Difference MC and TD

- TD can learn online after every step
- MC must wait until the end of episode before return is known.
- TD can learn from incomplete sequences
- MC can only learn from complete sequences
- TD works in continuing environment
- MC works for episodic environment
- TD is usually more efficient than MC. While MC is more simple to understand and use
- TD is sensitive to initial value while MC is not.
- TD has low variance , some bias

n-step Prediction : TD Forward View

- Let TD target look *n* steps into the future

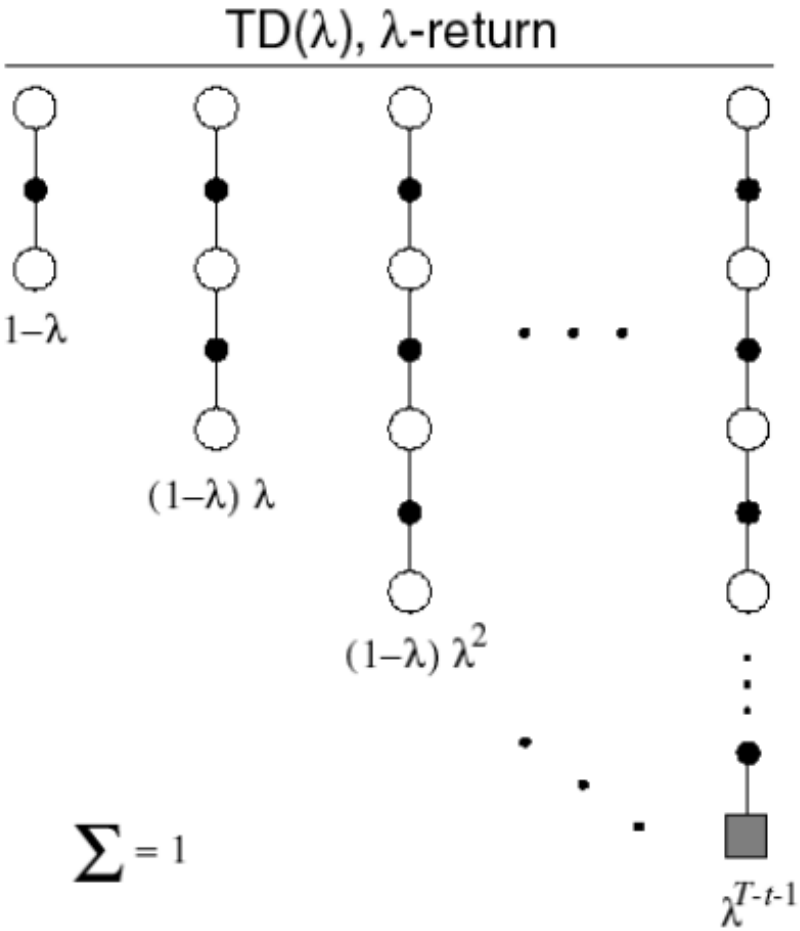


Averaging n-Step return And TD Forward view(λ)

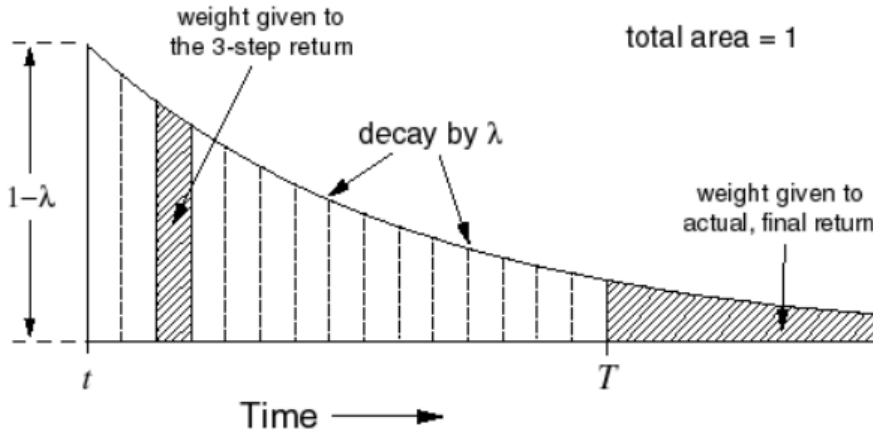
- Define n step return as:
  $G_t^n = R_{t+1} + \gamma R_{t+2} ... + \gamma^n R_{t+n} + \gamma^n V(S_{t+n})$
- define n step temporal-difference learning:
  $V(S_t) \leftarrow V(S_t) + \alpha(G_t^n + V(S_t))$
- We can average n-step returns over different n combining information from two different steps: $\frac{1}{2} G^2 + \frac{1}{2} G^4$
- can we combine information from all time steps?
- The TD(λ) , enable us to to this.
- The λ-return combines all n-steps returns $G_t^{(n)}$ using weight $(1-\lambda)\lambda^{n-1}$: $G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$
- Forward-view TD λ: $V(S_t) = V(S_t) + \alpha(G_t^\lambda - V(S_t))$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

TD (λ) and Weighting Function



Weight



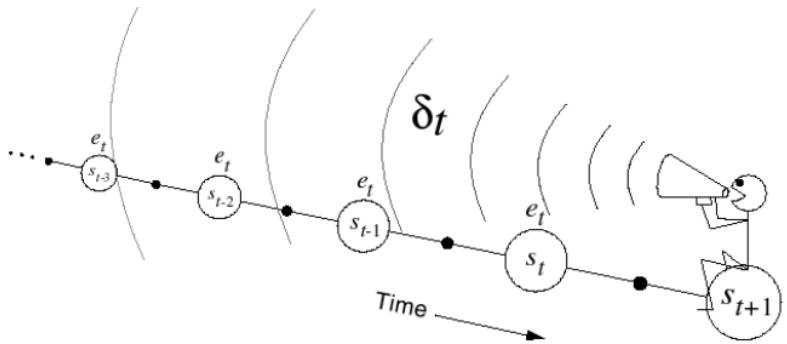
$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

Backward View And Eligibility Trace

- Forward view represent the theoretical conceptualization
- Actual implementation follows the Backward view
- Eligibility trace : : it is a parameter used to control the "memory" of the algorithm, associated to a given state.
  - Frequency heuristic: assign credit to most frequent state
  - Recency heuristic: assign credit to most recent states
  - Eligibility trace combines both heuristics.
    $E_0(s) = 0$
    $E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbb{1}(S_t = s)$
- Backward view keeps an Eligibility trace for every state s
- Update V(s) for every state s in proportion to TD-error δ_t and Eligibility trace: $E_t(s)$
  $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  $V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Backward View TD(λ) And Weighting Function



- When $\lambda = 0$, only current state is updated. $E_t(s) = \mathbb{1}(S_t = s)$
  $V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$
- This is same as TD(0) update:
  $V(S_t) \leftarrow V(S_t) + \alpha \delta_t$
- TD(1) is roughly equivalent to every-state visit Monte-Carlo

Table of Contents

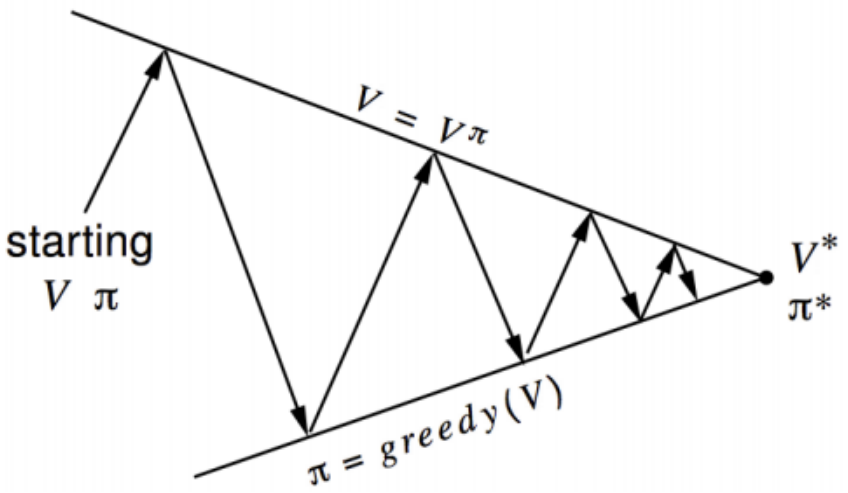
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

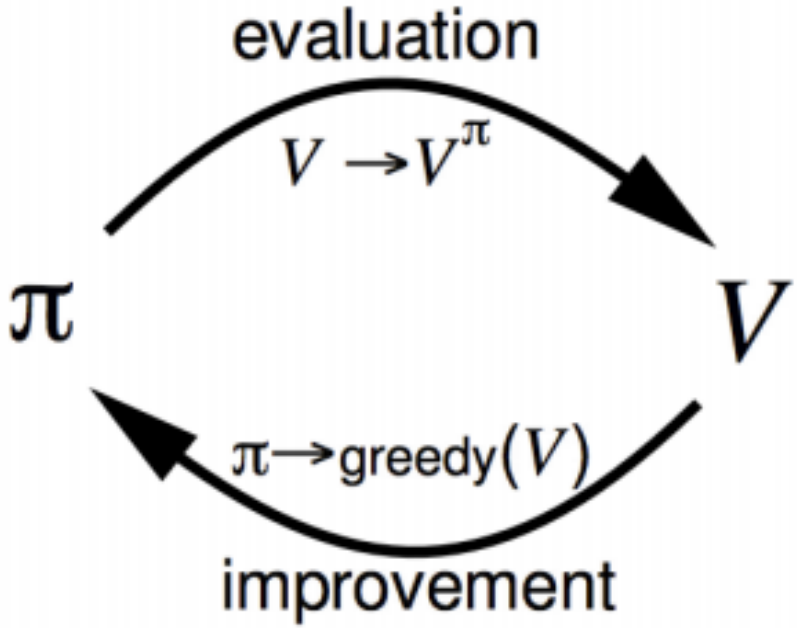
Model Free Control Introduction

Goal: Optimize the value function of an unknown MDP.
Examples: Robot Walking, Ship Steering, Helipcopter, Aeroplane logistics, Portfolio Management. etc

- On policy Learning: learns action value relative to the policy it follows. Or learning about policy π from experience sampled from policy π (sequence of actions) Example SARSA
- Off-policy learning: Learn about policy π from experience sampled from μ. The value assigned to a given state (or state-action pair) is a function of the immediate reward and of the maximum rewards received in the subsequent states during the episode. E.g Q learning.

Policy Iteration Intro





- Policy Iteration : Estimate v_π.
- Policy Improvement : Generate $\pi' \geq \pi$ e.g Greedy policy.

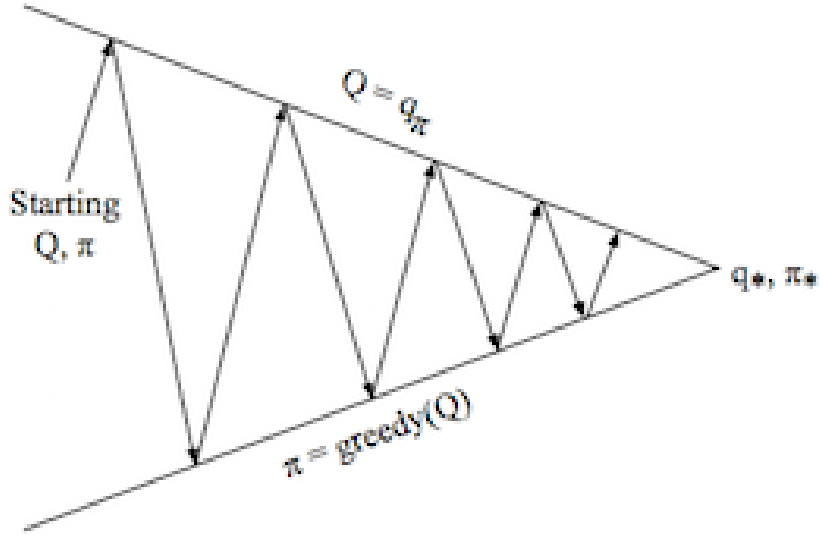
Action-value Function

- Greedy policy improvement over V(s) requires the model of MDP.

$$\pi'(s) = \underset{a \in A}{\text{argmax}}\ \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over Q(s,a) is model-free

$$\pi'(s) = \underset{a \in A}{\text{argmax}}\ Q(s,a)$$



- Policy evaluation: Monte carlo evaluation , Q = q_π
- Greedy policy vs exploration.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

ϵ-Greedy policy

- All m actions are tried with non-zero probability.
- With probability $1 - \epsilon$ choose the greedy action
- With action ϵ choose a random action

Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy with respect to q_π is an improvement , $v'_\pi \geq v_\pi(s)$.

- we can run the Monte Carlo policy improvement every episode.

Greedy in the limit GLIE

Greedy in the limit with infinite exploration (GLIE)

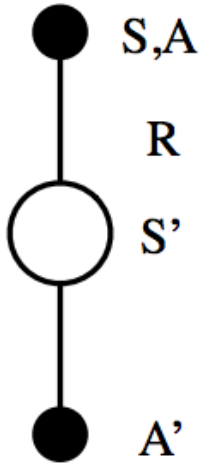
- All states actions are explored infinitely many times
- The policy converges on a greedy policy

GLIE converges to optimal action-value function

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

SARSA And TD control

- Apply TD instead of Q(S,A) instead of MC
- use ϵ-greedy policy improvement
- $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^{\lambda} - Q(S_t, A_t))$



- Every time step
- Policy evaluation, SARSA
- Policy improvement ϵ-greedy.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

n-Step-, Forward View (λ)-, Off-policy(λ)- Sarsa

This is an extension of the corresponding algorithm we have already discussed, but now we use action-values

- n-step Sarsa updates Q(s,a) towards the n-step return:
  $q_t^n \leftarrow R_{t+1} + \gamma R_{t+2} ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^n - Q(S_t, A_t))$
- Forward view Sarsa: $q_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^n$
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t))$
- Sarsa(λ): $E_0(s,a) = 0$;
  $E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbb{1}(S_t = s, A_t = a)$
  $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha \delta_t E_t(s,a)$

Q-Learning

We now consider off-policy learning of action-values Q(s,a).

- Next action is chosen based on behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- We update Q(S_t,A_t) towards alternative action:

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

- Note that in off-policy learning we evaluate the target policy $\pi(a|s)$to compute V_π or q$\pi(s, a)$ while following the behavior policy μ(a—s) :

$$\{S_1, A_1, R_2 ... S_T\} \sim \mu$$

- This might be important in order for agents to learn from other agents, or reuse old policies, or learn more optimal policies, or learn about multiple policies.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Off Policy control with Q-learning

Here we allow both behavior and target policies to improve

- Target policy is greedy w.r.t Q(S,a) $\pi(S_{t+1}) = \underset{a'}{\text{argmax}}\ Q(S_{t+1}, a')$
- The behavior policy μ is e.g ϵ-greedy w.r.t Q(s,a)
- Sarsa control algorithm :

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max'_a\ Q(S', A') - Q(S, A))$$

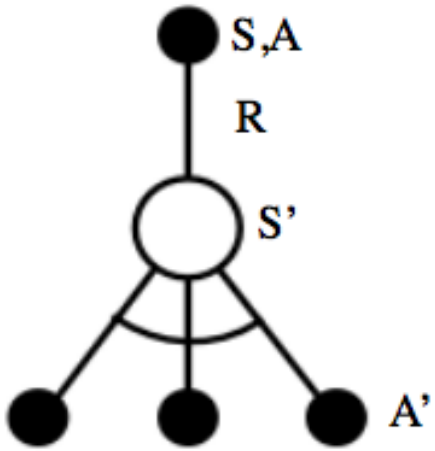


Table of Contents

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Why Value Functions Approximation ?

- Real world problems can present a very large number of states. Thus it impractical to evaluate the value of each state (computational and memory requirement)
- The idea of function approximation is to estimate the value of path visited in the state in the state space and generalize this thus estimate across state space.
- How do we solve large MDP:
  - $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$
  - $\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s,a)$
- We can now Generalise to unseen states and update parameter w using TD learning or MC learning.
- We consider only differential function approximator: Neural nets and Linear Function approximators.

Stochastic Gradient descent

- Define J($\mathbf{w}$) as our objective function. Its gradient is defined as $\Delta_{\mathbf{w}} J(\mathbf{w})$. Then we can adjust $\mathbf{w}$ in direction of negative gradient: $\Delta \mathbf{w} = -\frac{1}{2}\alpha \Delta_{\mathbf{w}} J(\mathbf{w})$
- Define $J(\mathbf{w}) = \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s, \mathbf{w}))^2])$. Then $\Delta \mathbf{w} = \alpha \mathbb{E}_\pi[(v_\pi(s) - \hat{v}_\pi(s, \mathbf{w}))] \Delta_{\mathbf{w}} \hat{v}_\pi(s, \mathbf{w})$
- For stochastic gradient we have that: $\Delta \mathbf{w} = \alpha \mathbb{E}_\pi[(v_\pi(s) - \hat{v}_\pi(s, \mathbf{w})) \Delta_{\mathbf{w}} \hat{v}_\pi(s, \mathbf{w})$
- For a linear approximator we have that: $\hat{v} = \mathbf{x}(S)^T \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$
- We can substitute this into the equation above.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Two-column slide

This is a text in first column.

$$E = mc^2$$

- First item
- Second item

This text will be in the second column and on a second tought this is a nice looking layout in some cases.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Sample frame title

In this slide, some important text will be highlighted because it's important. Please, don't abuse it.

Remark

Sample text

Important theorem

Sample text in red box

Examples

Sample text in green box. "Examples" is fixed as block title.

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE