Thesis Seminar: Non-Stationary MDPs and Continual Reinforcement Learning Algorithms

Sandesh Katakam - BS MS 2020

Thesis Supervisor: Dr. Srijith P.K.
Associate Professor, Department of AI
Indian Institute of Technology, Hyderabad
Co-Supervisor: Dr. Seshadri Chintapalli, IISER Berhampur

IIT Hyderabad, 11/11/2024

Outline

- **Motivation**
- **Problem Setup**
- **Taxonomy of Problem settings**
- **Background on Markov Decision Processes:**
  - **Preliminaries**: Markov Property, Markov Reward Process, Markov Decision Processes
  - Bellman Expectation Equation and Bellman Optimality Equation
  - Solving Bellman Optimality Equation
  - Exact Solution Methods: Policy Iteration and Value Iteration Methods
  - Theoretical Guarantees for Exact Solution Methods
  - Sampling-Based Methods:Model-Free Prediction/Control & Function Approx. Methods

Motivation

- In Reinforcement learning (RL), Markov Decision Processes (MDPs) provide a robust framework for modeling **sequential decision-making**. However, most classical RL algorithms assume stationary environments, where the transition dynamics and reward functions remain fixed over time
- Mathematically, this requires developing methods to handle **time-dependent distributions**, **non-stationary dynamics**, and the need for **robust optimization in the face of shifting environment parameters**.

The Problem of Pareto Optimality in Non-stationary MDPs

Setting up the Continual RL Problem(Mathematical Definition)

(General CRL Problem $\mathcal{M}_{CRL}$):[a]

Given a state $\mathcal{S}$, action-space $\mathcal{A}$, an observation space $\mathcal{O}$, a reward function $r : \mathcal{S}x\mathcal{A} \rightarrow \mathcal{R}$, a transition function $p : \mathcal{S}x\mathcal{A} \rightarrow \mathcal{S}$, and an observation function $x : \mathcal{S} \rightarrow \mathcal{O}$, the most general continual reinforcement learning problem problem can be expressed as

$$\mathcal{M}_{CRL} = \langle \mathcal{S}(t), \mathcal{A}(t), r(t), p(t), x(t), \mathcal{O}(t) \rangle$$

where each component of the problem formulation can be considered as a non-stationary function of form $f(i, t)$ where i is the input specific to each component.

[a] Khetarpal et.al 2022, Towards Continual Reinforcement Learning: A Review and Perspectives

Assumptions for Non-stationary Functional Form $f(i, t)$:

- Lipschitz Continuity
- Piecewise Non-Stationarity

Non-Stationary MDPs

Definition(Non-stationary MDPs)

Non-stationary MDP as a special type of CRL Problem:[a]
where $\alpha \subseteq \{\mathcal{S}, \mathcal{A}, r, p\}$, the observation function is an appropriate identity matrix $x = \mathcal{I}$ and the observation space is the state space $\mathcal{O} = \mathcal{S}$

$$\mathcal{M}_{CRL} = \langle \mathcal{S}(t), \mathcal{A}(t), r(t), p(t), x(t), \mathcal{O}(t) \rangle$$

[a] Khetarpal et.al 2022, Towards Continual Reinforcement Learning: A Review and Perspectives

A Taxonomy of Continual RL Formalisms

A categorisation of non-stationarity along two primary dimensions:[1]

- **Scope of Non-Stationarity** (α)
- **Driver of Non-Stationarity** (β)

Definition(scope of Non-stationarity α): defines what elements of the agent-environment interaction proces experience non-stationarity:

$$\alpha \subseteq \{\mathcal{S}, \mathcal{A}, r, p, x, \mathcal{O}\}$$

where $p \in \alpha$ if $\exists t, t' \in R,\ p(t)! = p(t')$, $r \in \alpha$ if $\exists t, t' \in R, r(t)! = r(t')$, etc.

[1] Khetarpal et.al 2022, Towards Continual Reinforcement Learning: A Review and Perspectives

A Taxonomy of Continual RL Formalisms(2)

Definition(Driver of Non-stationarity β): defines the causal assumptions that can be made about the nature of the evolution of non-stationary environment dynamics

$$\beta \subseteq \{stationary, passive, active, hybrid\}$$

where

- **stationary** $\Rightarrow \mathbb{E}[f(i,t)] = \mathbb{E}[f(i,t')]\ \forall t \in \Re,\ \forall t' > t,\ \forall i \in I$
- **passive** $\Rightarrow$ if $\mathbb{E}[f(i,t)] \neq \mathbb{E}[f(i,t')]$, then $|\mathbb{E}[f(i,t)] - \mathbb{E}[f(i,t')]| \not\perp a,\ \forall a \in \mathcal{A},\ \forall t' > t,\ \forall i \in \mathcal{I}$
- **active** $\Rightarrow$ if $\mathbb{E}[f(i,t)] \neq \mathbb{E}[f(i,t')]$, then $|\mathbb{E}[f(i,t)] - \mathbb{E}[f(i,t')]| \perp a,\ \forall a \in \mathcal{A},\ \forall a \in \mathcal{A},\ \forall t' > t,\ \forall i \in \mathcal{I}$
- **hybrid** $\Rightarrow$ if $\mathbb{E}[f(i,t)] \neq \mathbb{E}[f(i,t')]$, then $|\mathbb{E}[f(i,t)] - \mathbb{E}[f(i,t')]| \perp a,\ \exists a \in \mathcal{A},\ \exists t \in \Re, \forall t' > t,\ \exists i \in \mathcal{I}$ and $|\mathbb{E}[f(i,t)] - \mathbb{E}[f(i,t')]| \not\perp a, \exists a \in \mathcal{A},\ \exists t \in \Re,\ \forall t' > t,\ \forall i \in \mathcal{I}$

Markov Property

Definition (Markov Property). Let $(X_n)_{n\geq 0}$ be a stochastic process. The process has the Markov property if for any $n \geq 0$ and any possible states $i_0, \ldots, i_n, j$:

$$P(X_{n+1} = j | X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0)$$
$$= P(X_{n+1} = j | X_n = i_n)$$

This means that the future state depends only on the present state and not on the past states.

State Transition Matrix

For a markov state s and successor state s', the state transition probability is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all successor states s'

$$\mathcal{P}_{ss'} = from \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ . & & . \\ . & & . \\ . & & . \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

where each row of the matrix sums to 1

Markov Processes

Markov Process

A Markov Process (or Markov Chain) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s]$$

Markov Reward Process

Definition

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

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[\mathcal{R}_{t+1} | \mathcal{S}_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Return

Definition

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

$$\mathcal{G}_t = \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k \mathcal{R}_{t+k+1}$$

- The "discount" $\gamma \in [0, 1]$ is the present value of future rewards
- This values immediate reward above delayed reward
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Markov Decision Process

A Markov Decision Process (MDP) is a Markov reward process with decisions

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

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A}_t = a]$
  It's called **four p-argument,** $\mathcal{P}^{a}_{ss'} = p(s', r | s, a)$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[\mathcal{R}_{t+1} | \mathcal{S}_t = s, \mathcal{A}_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

Policy

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a|\mathcal{S}_t = s]$$

- Policies are stationary(time-independent), $\mathcal{A}_t \ \pi(\cdot|\mathcal{S}_t), \forall t > 0$
- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
  - Sequence $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, ...$ is a Markov Process $\langle \mathcal{S}, \mathcal{P}_\pi \rangle$
  - Sequence $\mathcal{S}_1, \mathcal{R}_2, \mathcal{S}_2, ...$ is a Markov Reward Process $\langle \mathcal{S}, \mathcal{P}_\pi, \mathcal{R}_\pi, \gamma \rangle$, where

$$\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

Value Function

State-Value Function:

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

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

Optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s)$$

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

$$q_\pi(s, a) = \mathbb{E}_\pi[\mathcal{G}_t | \mathcal{S}_t = s, \mathcal{A}_t = a]$$

Optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Bellman Expectation Equation

Decomposing state-value function further

$$v_\pi(s) = \mathbb{E}_\pi[\mathcal{R}_{t+1} + \gamma v_\pi(\mathcal{S}_{t+1})|\mathcal{S}_t = s]$$

Decomposing Action-value function further

$$q_\pi(s, a) = \mathbb{E}_\pi[\mathcal{R}_{t+1} + \gamma q_\pi(\mathcal{S}_{t+1}), \mathcal{A}_{t+1}|\mathcal{S}_t = s, \mathcal{A}_t = a]$$

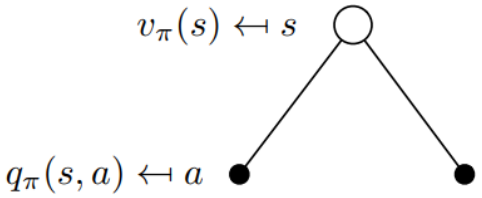
In Matrix form, Bellman expectation equation can be written as

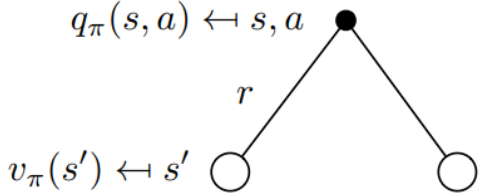
$$v_\pi = \mathcal{R}_\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

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

Bellman Expectation Equations for $\mathcal{V}^{\pi}$ and $\mathcal{Q}^{\pi}$ (1)





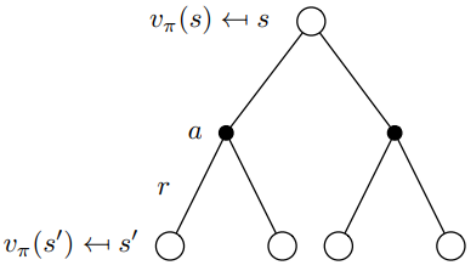
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

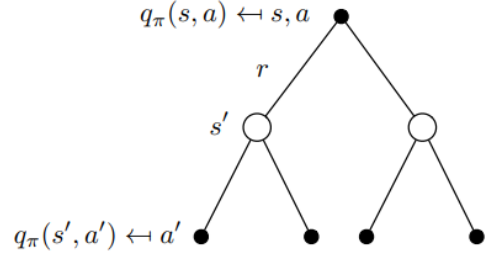
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

2

[2] David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015

Bellman Expectation Equations for $\mathcal{V}^{\pi}$ and $\mathcal{Q}^{\pi}$ (2)





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

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

[3]

[3] David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015

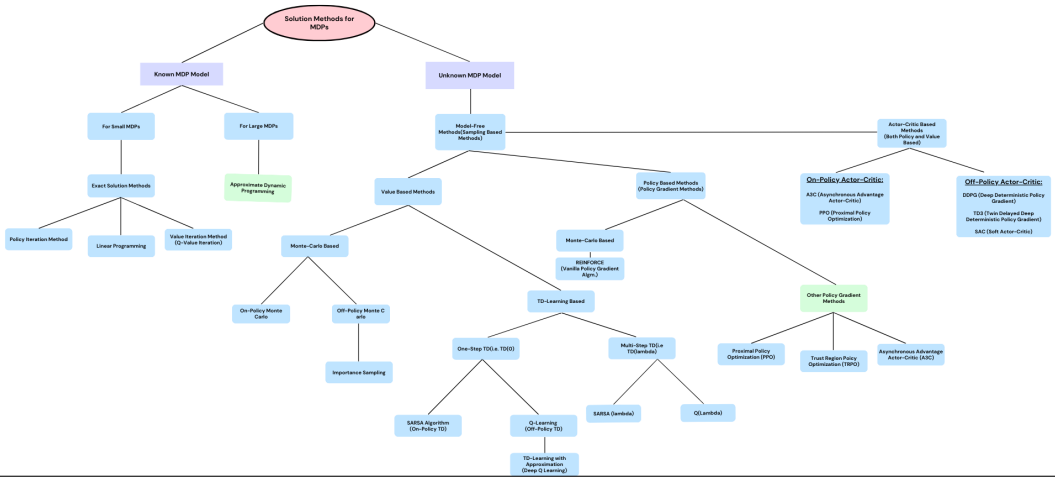
Solving Bellman Optimality Equation

An **optimal policy** can be found by maximizing over $q_*(s, a)$,

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

- Bellman Optimality Equation in **non-linear**
- No closed form solution (in general)
- **Iterative Solution Methods**
  - **Value Iteration** (Exact Solution Method)
  - **Policy Iteration** (Exact Solution Method)
  - **Q-Learning** (Sampling Based Method, Off-Policy TD)
  - **SARSA** (Sampling Based Method, On-Policy TD) etc..

Taxonomy of Solution Methods for MDPs



Exact Solution Methods: Known MDP Model

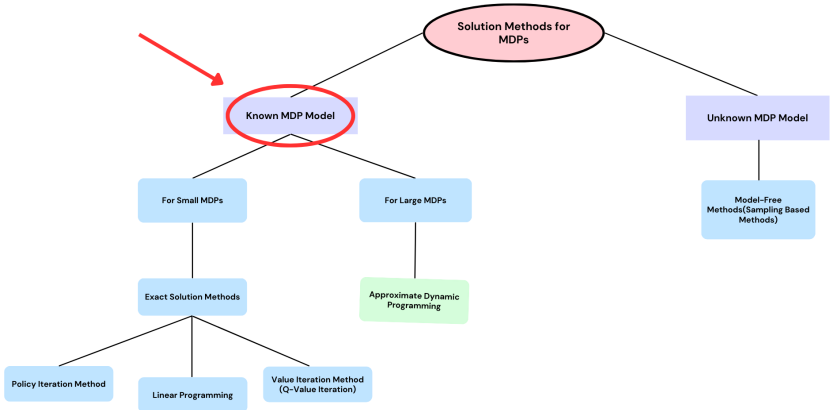


Figure: Taxonomy Exact Solution Methods: Known Model

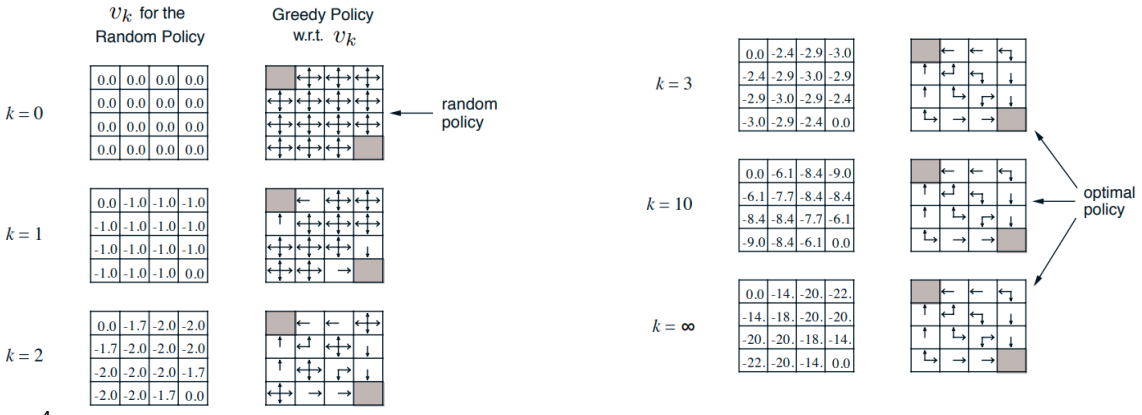
Small Grid World Example

actions

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$r = -1$
on all transitions

Iterative Policy Evaluation in Small Grid World



4

[4] David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015

Exact Solution Methods: Policy Iteration Method (2)

Iterative Policy Improvement (Control)

Given a policy π

- **Evaluate** the policy π

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + ...|S_t = s] \tag{2}$$

- **Improve** the policy by acting greedily with respect to v_π

$$\begin{aligned} \pi^{'} &= greedy(v_\pi) \\ &= \arg\max_a q_\pi(s) \\ &= \arg\max_a \mathbb{E}[\mathcal{R}_{t+1} + \gamma v_\pi(\mathcal{S}_{t+1}|\mathcal{S}_t = s, \mathcal{A}_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3}$$

Note: This Process of Policy Iteration always converges to π^* (By Contraction Mapping Theorem)

Exact Solution Methods: Value Iteration

Question? Why not update policy every iteration instead of waiting for policy evaluation to converge
If we know solution to sub problems $v_*(s')$, then solution to $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{4}$$

Similarly **Q-Value Iteration can be written as:**

$$\mathcal{Q}_{k+1}(s,a) \leftarrow \sum_{s'} p(s', r|s, a)[r + \gamma \max_{a'} \mathcal{Q}_k(s', a')]$$

Idea: Apply The updates iteratively
Intuition: Start with final rewards and work backwards

Theoretical Guarantees

Value Function Space

- Consider the vector space $\mathcal{V}$ over value functions with $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$

Value Function ∞-Norm

- Measure distance between state-value functions u and v by the ∞-norm i.e. the largest difference between state values,

$$\|u - v\|_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

Bellman Expectation Backup Operator

$$\mathcal{T}^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$$

This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}\|\mathcal{T}^\pi(u) - \mathcal{T}^\pi(v)\|_\infty &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi u) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi v)\|_\infty \\ &= \|\gamma \mathcal{P}^\pi (u - v)\|_\infty \\ &\leq \|\gamma \mathcal{P}^\pi\|u - v\|_\infty\|_\infty \\ &\leq \gamma \|u - v\|_\infty\end{aligned}$$

Bellman Optimality Backup Operator

$$\mathcal{T}^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$$

This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\|\mathcal{T}^*(u) - \mathcal{T}^*(v)\|_\infty \leq \gamma \|u - v\|_\infty$$

Contraction Mapping Theorem

Theorem(Contraction Mapping Theorem)
For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $\mathcal{T}(v)$ where $\mathcal{T}$ is a γ-contraction,

- $\mathcal{T}$ converges to a unique fixed point
- At linear convergence rate of γ

5

[5] Reinforcement Leraning: An Introduction, Richard Sutton and Andrew Barto, 1998

Policy Iteration and Value Iteration Convergence

Convergence of Iterative Policy Evaluation and Policy Iteration
Idea: The Bellman expectation operator $\mathcal{T}^{\pi}$ has a unique fixed point i.e. v_{π}(by Bellman Expectation Equation).
By contraction mapping theorem, Iterative Policy Evaluation converges on v_{π} and Policy Iteration converges on v_{*}

Convergence of Value Iteration
Idea: The Bellman Operator $\mathcal{T}^{*}$ has a unique fixed point i.e. v_{*} (by Bellman Optimality Equation).
By Contraction Mapping Theorem, Value Iteration Converges on v_{*}

Limitations of Exact Solution Methods:

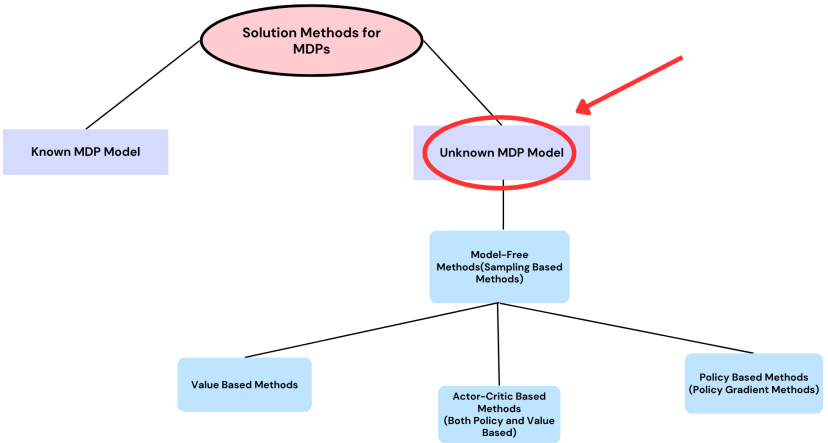
Two Major Limitations of Exact Solution Methods:

- Need to know $\mathcal{P}^{a}_{ss'}$ of the MDP $\rightarrow$ **FIX: Sampling-Based Methods**
  - **Idea:** Using sample rewards and sample transitions i.e. $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{S}' \rangle$ instead of reward function $\mathcal{R}$ and transition dynamics $\mathcal{P}$
  - **Advantages:**
    - **Model-Free:** No advance knowledge of MDP required
    - Breaks the curse of dimensionality theory sampling
    - Cost of backup is constant, independent of $n = \|\mathcal{S}\|$
- Memory Exploding: Storage of values for all states and actions (only possible for small discrete state-action space) $\rightarrow$ **FIX: $\mathcal{Q}/\mathcal{V}$ function fitting** (Note: Although this can be fixed by Approximate Dynamic Programming)

6

[6] Deep RL Bootcamp 2017 by Pieter Abbeel , UC Berkeley

Model-Free Methods(or Sampling Based Methods): Unknown MDP Model



Value Based Methods

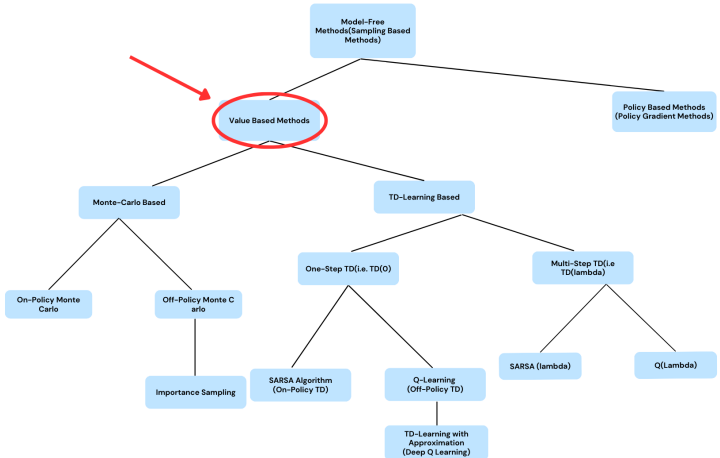


Figure: Taxonomy Model-Free Methods: Value Based Methods

Temporal Difference Based: Prediction

- TD Can learn before knowing the final outcome (i.e. TD can learn online after every step)
- TD Can learn from incomplete sequences
- TD works in continuing (non-terminating) environments
- We start with simplest temporal-difference learning algorithm: **TD(0)**
  - Update value $\mathcal{V}(\mathcal{S}_t)$ toward estimated return $\mathcal{R}_{t+1} + \gamma\mathcal{V}(\mathcal{S}_t)$

$$\mathcal{V}(\mathcal{S}_t) \leftarrow \mathcal{V}(\mathcal{S}_t) + \alpha(\mathcal{R}_{t+1} + \gamma\mathcal{V}(\mathcal{S}_{t+1}) - \mathcal{V}(\mathcal{S}_t))$$

  - $\mathcal{R}_{t+1} + \gamma\mathcal{V}(\mathcal{S}_{t+1}$ → **TD target**
  - $\delta_t = \mathcal{R}_{t+1} + \gamma\mathcal{V}(\mathcal{S}_{t+1}) - \mathcal{V}(\mathcal{S}_t)$ → **TD Error**

7

[7] David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015

Off-Policy TD-Learning (Q-Learning)

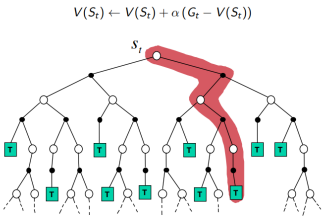
- **Q-Value Iteration can be written as:**
  $\mathcal{Q}_{k+1}(s,a) \leftarrow \sum_{s'} p(s',r|s,a)[r + \gamma \max_{a'} \mathcal{Q}_k(s',a')]$
- **Rewrite as expectation:** $\mathcal{Q}_{k+1} \leftarrow \mathbb{E}_{s' \sim p(s',r|s,a)}[r + \gamma \max_{a'} \mathcal{Q}_k(s',a')]$
- **(Tabular) Q-Learning**: Replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim \mathcal{P}(s'|s,a)$(same as $p(s',r|s,a)$ from before)
  - Consider your old estimate $Q_k(s,a)$
  - Consider your new sample estimate:

$$target(s') = r + \gamma \mathcal{Q}_k(s',a')$$

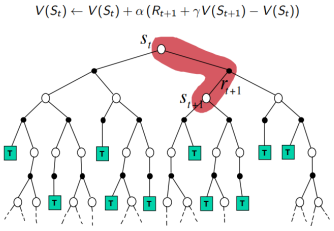
  - Incorporate the new estimate into a running average:

$$\mathcal{Q}_{k+1}(s,a) \leftarrow (1-\alpha)\mathcal{Q}_k(s,a) + \alpha[target(s')]$$

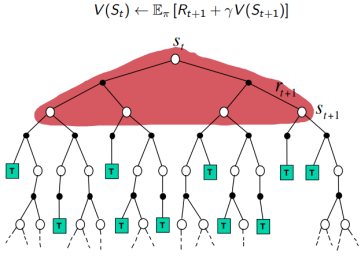
Comparisons: MC, TD, DP



Monte-Carlo Backup



Temporal Difference Backup



Dynamic Programming Backup

8

[8] David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015

Approximation Based Methods for $\mathcal{Q}/\mathcal{V}$ (Deep Q Learning)

Motivation: Tabular methods for $\mathcal{Q}$ ($\mathcal{Q}$-Learning) doesn't scale with large MDPs with many state-action pairs
Solution: Use function approximators (e.g. Neural networks etc.) to approximate $\mathcal{Q}$ values
Catch: TD-Based methods with bootstrapping are good to work with, however when off-policy, nonlinear function approximation and bootstrapping are combined in one RL algorithm, the training could be unstable and hard to converge. (This is known as **Deadly Triad Issue**)

Deep Q Networks (DQN) greatly improve and stabilize the training procedure of Q-learning by two methods:

- **Experience Replay**
- **Periodically updated target**

And the loss function we try to optimize for looks like this:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(r + \gamma \max_{a'} \mathcal{Q}(s', a'; \theta^-) - \mathcal{Q}(s, a; \theta))^2]$$

where $\mathcal{U}(\mathcal{D})$ is a uniform distribution over the replay memory $\mathcal{D}$
θ^- is the parameters of the frozen target $\mathcal{Q}$-network [9]

[9] Lilian Weng's Blog: A (Long) Peek into Reinforcement Learning

Model-Free Setting: Policy Based Methods

In value based we approximated the value or action-value function using parameters θ

$$\mathcal{V}_\theta(s) = \mathcal{V}^\pi(s)$$

$$\mathcal{Q}_\theta(s,a) = \mathcal{Q}^\pi(s,a)$$

And a policy was generated directly from the value function. (e.g. ϵ-greedy) Now, in policy based methods we directly parametrise the policy

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

similar to $\mathcal{Q}_\theta(s,a) = f(\phi(s,a),\theta)$
Goal: Given policy $\pi_\theta(s,a)$ with parameters θ, find best θ that maximises $\mathcal{J}(\theta)$

In Episodic environments:(Use start value)

$$\mathcal{J}_1(\theta) = \mathcal{V}^{\pi_\theta}(s_1) = \mathbb{E}[v_1]$$

In Continuing Environments: (Use average value)

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

Policy Based Methods

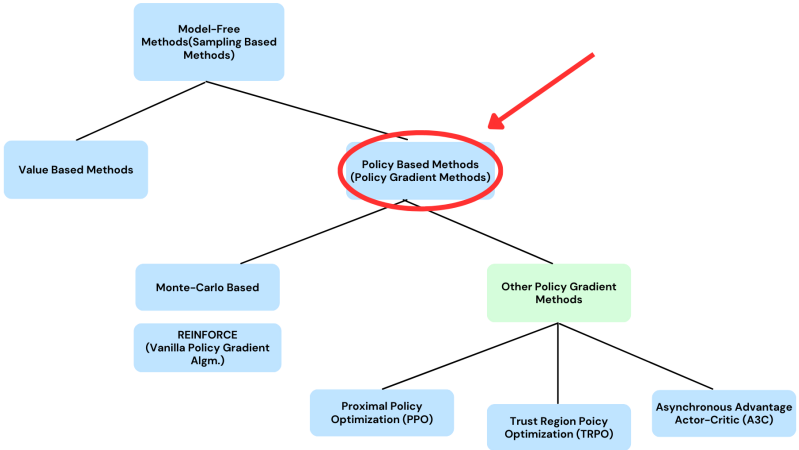


Figure: Taxonomy Model-Free Methods: Policy Based Methods

Policy Gradient Theorem

For any differentiable policy $\pi_\theta(s, a)$, for any of the policy objective functions $\mathcal{J} = \mathcal{J}_1, \mathcal{J}_{avR}$ or $\frac{1}{1-\gamma}\mathcal{J}_{avV}$ the policy gradient is

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

Proof Outline:[10]

- **Express $J(\theta)$ in Terms of the Action-Value Function:**
  Using the action-value function $Q^\pi(s, a)$, we can rewrite $J(\theta)$ as:

$$J(\theta) = \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a),$$

  where $d_\pi(s)$ is the stationary distribution of states under policy π_θ.

[10] Reinforcement Leraning: An Introduction, Richard Sutton and Andrew Barto

Policy Gradient Theorem (Proof Outline)

- **Take the Gradient with Respect to θ:**
  We now take the gradient of $J(\theta)$:

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} \nabla_\theta d_\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) + \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a).$$

- **Approximate $\nabla_\theta d_\pi(s)$ Using the Markov Property:**
  Given that $d_\pi(s)$ depends indirectly on θ through π_θ, we simplify by focusing on the term involving $\nabla_\theta \pi_\theta(a|s)$.

- **Rewrite the Gradient Using the Log Trick:**
  Applying the "log trick" to the second term:

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a).$$

This can be written as an expectation: $\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d_\pi, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right].$

Existing Success in Solving Non-Stationary MDPs

- **Optimizing for the Future:**[11]

  Idea: Stopping the iterative convergence process before reaching optimal policy and using future forecasting to optimize the expectation values over forecasted future

- **Meta-Learning Approaches(Model Agnostic Meta-Learning)** [12]

  find a good initial set of model parameters that can quickly adapt to new tasks with just a few gradient steps by explicitly optimizing the model's performance after one or few gradient updates on a new task, creating a "meta-objective" that maximizes the model's ability to learn quickly rather than its direct performance on any single task

- **Unsupervised Zero-Shot RL with Functional Reward Encodings** [13]
- **Continual Model-Based RL with HyperNetworks:** [14]

[11] Chandak et.al, Optimizing for the Future in Non-Stationary MDPs (ICML 2020)

[12] Chelsea Finn et.al 2017 Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks(ICML 2017)

[13] Abbeel et.al 2024, Unsupervised Zero-Shot Reinforcement Learning via Functional Reward Encodings(ICML 2024)

[14] Huang et.al 2021, Continual Model-Based Reinforcement Learning with Hypernetworks

Methodology of My thesis work

Problem: Learning optimal policies in non-stationary MDPs while not losing performance over the past MDPs
Idea: can be addressed by storing samples of interactions (Markovian transitions E.g.:$s, a, s', a', s'', a'', ..$) and replaying it during the optimization process(**i.e. Experience memory replay methods)**

Bottleneck: Storing Markovian Transitions as the number of tasks grow will result in memory exploding and doesn't scale well

Proposed Solution: Recent works on **Online function approximation methods** for storing memory representations of past sequences from the sequence modelling and SSM (state space models) literature have shown incredible promise for storing very long past sequences for improving context length as **structured matrices (e.g. HiPPO)** [a]

[a] Gu et.al 2020, HiPPO: Recurrent Memory with Optimal Polynomial Projections

We plan to implement this strategy to tackle the **exploding memory problem** in our case across Non-stationary MDPs by treating the Markovian transitions as time-dependent sequences

Experimental Setup and Baselines

Benchmarks:

- **Atari (Bellemare et al., 2013)**, 6 task sequence: A standard, proven benchmark used by Schwarz et al. (2018b) and Rolnick et al. (2019), particularly to demonstrate resilience to catastrophic forgetting
- **Procgen (Cobbe et al., 2020)**, 6 task sequence: Designed to test resilience to forgetting and in-distribution generalization to unseen contexts in procedurally-generated, visually-distinct environments

Baselines:

For Atari:

- EWC (Elastic Weight Consolidation) typically achieves around 50-60 percent
- A3C with Progressive Networks serves as a common baseline, usually maintaining 70-80 percent
- PPO (Proximal Policy Optimization) without any continual learning mechanisms often shows catastrophic forgetting, dropping to 20-30 percent

Note: Metric used to measure the performance is *"Performance Retention"*

For ProcGen:

- Standard PPO baseline achieves around 5-7 mean training level performance across environments
- UCB (Upper Confidence Bound) exploration strategies typically reach 8-10 mean training level performance
- PLR (Prioritized Level Replay) methods achieve approximately 12-15 mean training level performance

References

- Reinforcement Leraning: An Introduction, Richard Sutton and Andrew Barto
- Lilian Weng's Blog: A (Long) Peek into Reinforcement Learning
- Lilian Weng's Blog: Policy Gradient Algorithms
- David Silver UCL Course on Reinforcement Learning(Advanced Topics) 2015
- Deep RL Bootcamp 2017 by Pieter Abbeel , UC Berkeley
- Khetarpal et.al 2022, Towards Continual Reinforcement Learning: A Review and Perspectives
- Chandak et.al, Optimizing for the Future in Non-Stationary MDPs (ICML 2020)
- Towards Safe Policy Improvement for Non-Stationary MDPs (NeurIPS 2020)
- Abbeel et.al 2024, Unsupervised Zero-Shot Reinforcement Learning via Functional Reward Encodings(ICML 2024)
- Huang et.al 2021, Continual Model-Based Reinforcement Learning with Hypernetworks
- Gu et.al 2020, HiPPO: Recurrent Memory with Optimal Polynomial Projections

References

- Powers et.al 2022, CORA: Benchmarks, Baselines, and Metrics as a Platform for Continual Reinforcement Learning Agents(CoLLA 2022)
- Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research (JAIR)
- Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress compress: A scalable framework for continual learning. ICML
- David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In Advances in Neural Information Processing Systems
- Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In International conference on machine learning
- Chelsea Finn et.al 2017 Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks(ICML 2017)

Questions?

"The only stupid question is the one you were afraid to ask but never did"
- Richard Sutton