# The Actors and The Critics: Function Approximations and Policy Gradients in Reinforcement Learning

**Debabrota Basu**
debabrota-basu.github.io

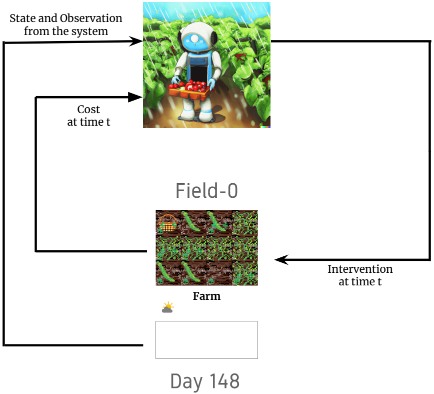
Spring School on Control Theory and Reinforcement Learning
CWI Amsterdam, 2025

# PART 1

## Revisiting MDPs and RL Algorithms

## Reinforcement Learning: The Philosophy

RL: sequentially learning to take optimal decisions under uncertainty.




The goal of the agent is to compute a policy or strategy that maximises the reward accumulated over a time horizon.

# Formalism of the Environment: Markov Decision Processes

A Markov Decision Process (MDP) is a tuple $\mathcal{M} \triangleq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- **State:** $s \in \mathcal{S} \subseteq \mathbb{R}^d$
- **Action/Intervention/Control:** $a \in \mathcal{A} \subseteq \mathbb{R}^d$
- **Transition function/dynamics:** $\mathcal{P}(.|s,a)$ induces a distribution over $s_{t+1}$ for $s_t, a_t$ (previously $f$)
- **Reward Function:** $\mathcal{R}(.|s,a)$ induces a distribution over $\mathbb{R}$ measuring goodness of an action $a$ at state $s$ (negative of cost function)

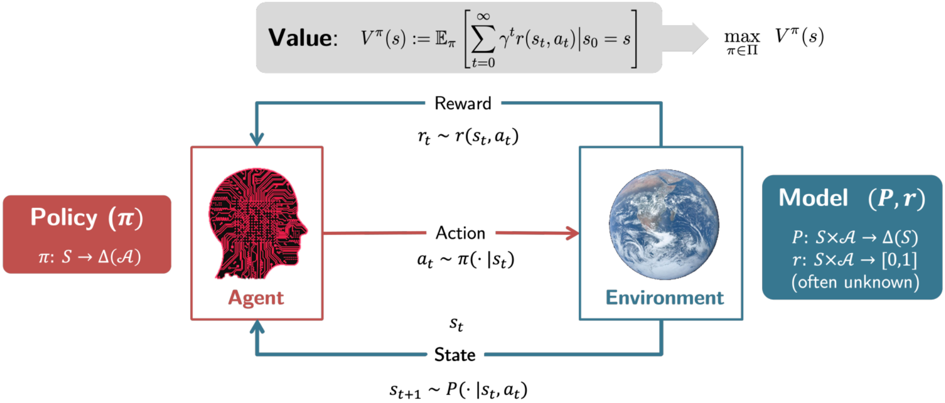
- **Policy:** A deterministic or stochastic map $\pi(\cdot|s_t)$ from present state $s_t$ to actions
- **How good or bas is your policy?** Value Function (Negative of cost of control)

$$V_\pi(s_0) \triangleq \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_t(s_t, \pi(s_t))$$

Or, action-value functions or Q values

$$Q_\pi(s_0, a_0) \triangleq \mathcal{R}(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t \mathcal{R}_t(s_t, \pi(s_t))$$

Goal: Find an optimal policy $\pi^*$ maximising $V_\pi(s_0)$.




Courtesy: Niao He, RLSS 2023

# A Generic Template of RL Algorithms: Generalised Policy Iteration

**Algorithm** Generalised Policy Iteration

**Input:** Initial Policy $\pi_0$
**for** episode $k = 1, 2, \ldots$ **do**
  Observe an initial state $s_0^k$
  **Rollouts:** Collect trajectory data $\{r(s_h^k, a_h^k), s_{h+1}^k\}_{h=0}^H$ and state $s_{h+1}^k$ by playing policy $\pi_k$
  **Policy Evaluation:** Compute the value function of the policy $V_{\pi_k}(s_0^k)$
  **Policy Optimisation:** Use $V_{\pi_k}(s_0^k)$ to compute a better policy $\pi_{k+1}$
**end for**
**return** policy $\pi_K$.

# A Taxonomy of RL Algorithms

1. Level of interaction with the environment:
   - Online: Sequentially learn while collecting data by interacting with the environment
   - Offline: Use data collected in advance by some behavioural policy (e.g. Monte-Carlo methods)

# A Taxonomy of RL Algorithms

1. Level of interaction with the environment:
   - Online: Sequentially learn while collecting data by interacting with the environment
   - Offline: Use data collected in advance by some behavioural policy (e.g. Monte-Carlo methods)
2. Approach to optimal solution:
   - Value-based RL (Off-policy): Find optimal value function $V^*_{\mathcal{M}}$, use it to compute $\pi^\star$
   - Policy-based RL (On-policy): Evaluate $\pi$, improve $\pi$, and repeat

# A Taxonomy of RL Algorithms

1. Level of interaction with the environment:
   - Online: Sequentially learn while collecting data by interacting with the environment
   - Offline: Use data collected in advance by some behavioural policy (e.g. Monte-Carlo methods)
2. Approach to optimal solution:
   - Value-based RL (Off-policy): Find optimal value function $V^*_{\mathcal{M}}$, use it to compute $\pi^\star$
   - Policy-based RL (On-policy): Evaluate $\pi$, improve $\pi$, and repeat
3. Knowledge of the model:
   - Planning: $\mathcal{R}$ and $\mathcal{P}$ are known (dynamic programming)
   - Model-based/model-predictive/model-learning: estimate $\mathcal{R}$ and $\mathcal{P}$ from data yielded through interactions
   - Model-free: no knowledge of $\mathcal{R}$ and $\mathcal{P}$ is used– only transition data

# (Iterative) Policy Evaluation in MDPs with Discrete State-Actions

**Algorithm** Iterative Policy Evaluation

1: **Input:** A policy $\pi$, steps $K$
2: **Initialise:** value function $V_0(s) = 0$ for all $s \in \mathcal{S}$
3: **for** steps $k = 1, 2, \ldots, K$ **do**
4: **for** states $s \in \mathcal{S}$ **do**
5: Using collected dataset or by rolling out $\pi$, compute the Bellman update equation

$$V_k(s) \leftarrow \mathcal{T} V_{k-1}(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \pi(s)) V_{k-1}(s') \tag{1}$$

6: **end for**
7: **end for**
8: **return** estimated value function $V_\pi(s) \leftarrow V_K(s)$ for all $s$.

# Policy Improvement in MDPs with Discrete State-Actions $\rightarrow$ Q-value Iteration

**Greedy improvement:** Bellman optimality equations

Value function

$$V^*_{\mathcal{M}}(s) = \max_a Q^*_{\mathcal{M}}(s,a)$$

Q-Value function

$$Q^*_{\mathcal{M}}(s,a) = \mathcal{T}^\star Q^*_{\mathcal{M}}(s,a) = \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*_{\mathcal{M}}(s)$$

**Greedy improvement:** Bellman optimality equations

Value function

$$V^*_{\mathcal{M}}(s) = \max_a Q^*_{\mathcal{M}}(s,a)$$

Q-Value function

$$Q^*_{\mathcal{M}}(s,a) = \mathcal{T}^\star Q^*_{\mathcal{M}}(s,a) = \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) V^*_{\mathcal{M}}(s)$$

**Algorithm** Q-value iteration (Off-policy, Planning with full information)

1: **Input:** Steps $K$
2: **Initialise:** Q-value function $Q_0(s,a) = 0$ for all $s, a \in \mathcal{S} \times \mathcal{A}$
3: **for** episodes $k = 1, 2, \dots, K$, and state-action pairs $(s,a) \in (\mathcal{S}, \mathcal{A})$ **do**
4: **Compute Q-table:** Evaluate the greedy policy using the Bellman update equation

$$Q_k(s, \pi(s)) \leftarrow \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \pi(s)) \max_b Q_{k-1}(s', b) \tag{2}$$

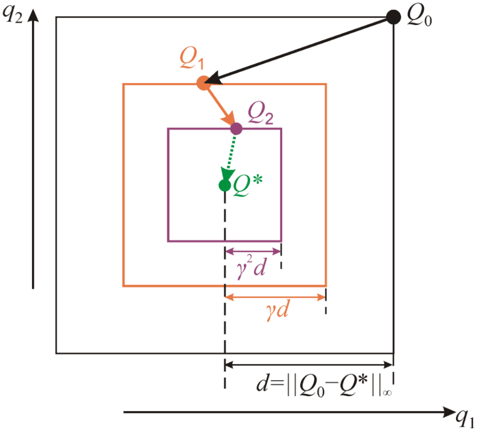
5: **end for**
6: **return** the **greedy policy** for all $s$

$$\pi_K(s) \in \arg\max_{a \in \mathcal{A}} Q_K(s,a) \tag{3}$$

# Why Does Q-iteration Work?

Q-iteration is a fixed point contraction through stochastic approximation.

$$\|Q_k - Q^*_{\mathcal{M}}\|_\infty = \|\mathcal{T}^\star Q_{k-1} - Q^*_{\mathcal{M}}\|_\infty \leq \gamma \|Q_{k-1} - Q^*_{\mathcal{M}}\|_\infty$$




Alternative update of Equation (2)

$$Q_k(s) = (1-\alpha)Q_{k-1}(s) + \alpha \mathcal{T}^\star Q_{k-1}(s)$$

This works as
(1) Bellman operator is a contraction, and
(2) $\mu_k = (1-\alpha)\mu_{k-1} + \alpha \times$ new sample
is a consistent stochastic approximation of mean.

# Limitations of GPIs in MDPs with Discrete State-Actions

Dynamic programming algorithms require an exact representation of value functions and policies.

# Limitations of GPIs in MDPs with Discrete State-Actions

Dynamic programming algorithms require an exact representation of value functions and policies.

Thus, GPI with tabular MDPs suffer from:

- Discrete States
- No generlisation and only look-up tables
- Computationally expensive to handle large state-action spaces
- Discrete Actions

# Limitations of GPIs in MDPs with Discrete State-Actions

Dynamic programming algorithms require an exact representation of value functions and policies.

Thus, GPI with tabular MDPs suffer from:

- Discrete States
- No generlisation and only look-up tables
- Computationally expensive to handle large state-action spaces
- Discrete Actions

### Approximate RL

Can we approximately learn good representations of transitions and rewards or directly the Q-value functions, and use them to find a good policy $\pi$?

**Goal**: Find a policy $\pi$ and functional representation $f$ such that the performance loss $\|V^*_{\mathcal{M}} - V^f_\pi\|$ is as small as possible

# Four Key Challenges in RL

- *Planning in Large Spaces (Curse of Dimensionality)*
  – How to optimise the policy when the number of reachable states and decidable actions are big?
- *Succinct Representation of Information*
  – How to succinctly represent the available information regarding states, actions, dynamics and policies?
- *Exploration–Exploitation Trade-off (Effect of Incomplete Information)*
  – Should you try out new decisions which may prove to be beneficial or play as best as you can with your existing knowledge?
- *Planning under Incomplete Information (Exploration + Planning)*
  – How to estimate the effect of an action and how to predict the future state reached from a state through the action?

# Four Key Challenges in RL and Solutions

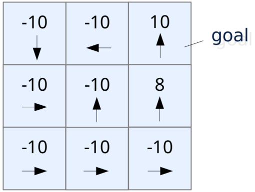
- *Planning in Large Spaces (Curse of Dimensionality)*
  $\rightarrow$ Approximate RL
- *Succinct Representation of Information*
  $\rightarrow$ Abstractions (Frans) $+$ Approximate RL $++$
- *Exploration–Exploitation Trade-off (Effect of Incomplete Information)*
  $\rightarrow$ Bandits (Bert) $+$ Q-learning (Sean) $+$ Optimism (Friday)
- *Planning under Incomplete Information (Exploration + Planning)*
  $\rightarrow$ Approximate RL $+$ Optimism (Friday)

# PART 2

## Function Approximations in RL
## From Discrete to Continuous States: Dynamic Programming with Learning

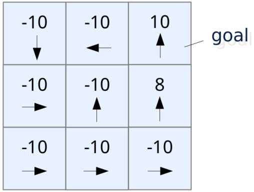
# From Discrete to Continuous World: From Q-tables to Q-functions

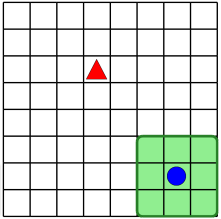
Q-table



## From Discrete to Continuous World: From Q-tables to Q-functions

Q-table

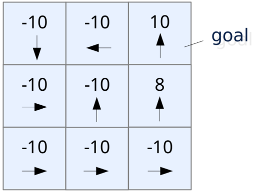


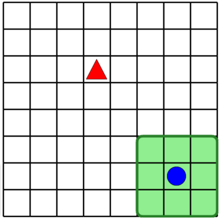



No generalisation across discretisations.

# From Discrete to Continuous World: From Q-tables to Q-functions

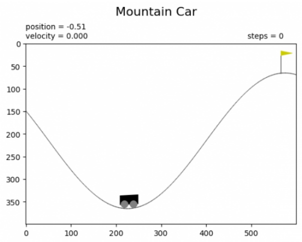
## Q-table

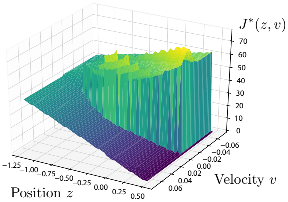


No generalisation across discretisations.

## Q-function




Learn the Q-values as a function of $(s, a)$.

# Approximate RL

### Problem

Exactly computing a Q-function across a continuous state-action space while looking into trajectories is not possible.

### Question

Can we approximately learn good representations of transitions and rewards or directly the Q-value functions, and use them to find a good policy $\pi$?

### Problem

Exactly computing Q-function across continuous state-action space while using trajectories is not possible.

### Question 1

Can we approximately learn generalisable and accurate representations of Q-value?

**Solution:**

Turn the Q-function computation into a learning problem.

$$Q_k(s,a) \leftarrow \mathcal{T}^\star Q_{k-1} = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,\pi(s)) \max_b Q_{k-1}(s',b)$$

a. Write a parametric Bellman update:

$$Q_\theta(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,\pi(s)) \max_b Q_\theta(s',b) \, .$$

# Approximate RL: Planning with Learning (Step 1)

### Problem

Exactly computing Q-function across continuous state-action space while using trajectories is not possible.

### Question 1

Can we approximately learn generalisable and accurate representations of Q-value?

**Solution:**

a. Write a parametric Bellman update:

$$\boxed{Q_\theta(s,a)} = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,\pi(s)) \max_b \boxed{Q_\theta(s',b)} .$$

b. Sample trajectories of $\{(s_i, a_i, r_i, s_i')\}_{i=1}^n$, and solve the regression problem to learn $\theta^\star$:

$$\theta^\star = \arg\min_\theta \boxed{\sum_{i=1}^n \left( Q_\theta(s_i,a_i) - r_i - \gamma \max_b Q_\theta(s_i', b) \right)^2}$$

# How to Choose a Function Approximation Space?

### Linear Models

$$Q^*_{\mathcal{M}} \in \{\theta^\top \phi(s,a), \theta \in \mathbb{R}^d\},$$

where $\phi : \mathcal{S} \times \mathcal{A} \to [0, M]$.

# How to Choose a Function Approximation Space?

### Linear Models

$$Q^*_{\mathcal{M}} \in \{\theta^\top \phi(s,a), \theta \in \mathbb{R}^d\},$$

where $\phi : \mathcal{S} \times \mathcal{A} \to [0, M]$.

### Kernel Models

$$Q^*_{\mathcal{M}} \in \{\mathrm{GaussianProcess}(\mu_Q, K_Q) \quad \text{s.t. } \mu_Q(s,a) = \vec{k}(s,a)^\top H^\top (H K_Q H^\top + \lambda I)^{-1} \vec{R}\}.$$

### Neural Models

$$Q^*_{\mathcal{M}} \in \{f_\theta(s,a), \theta \in \mathbb{R}^d\}.$$

Tutorials will shed further light on how to choose the functions.

# Approximate RL: Planning with Learning (Step 2)

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \Big( Q_\theta(s_i, a_i) - r_i - \gamma \max_b Q_\theta(s_i', b) \Big)^2$$

**Problem**

Unlike in standard approximation schemes (e.g. supervised learning), we have only limited access to the target function, i.e. $Q^*_{\mathcal{M}}$.

**Question 2**

What would be a good way to generate data to directly learn optimal Q-value function?

### Problem

We have only limited access to the target function, i.e. $Q^*_{\mathcal{M}}$.

### Question 2

What would be a good way to generate data to directly learn optimal Q-value function?

**Solution:**

GPI (e.g. Q-value iteration) tends to iteratively learn functions which are close to the optimal value function. Leverage the contraction to generate data.

$$\theta_{k+1} = \arg\min_{\theta} \boxed{\sum_{i=1}^{n} \Big( Q_{\theta}(s_i, a_i) - r_i - \gamma \max_b Q_{\theta_k}(s'_i, b) \Big)^2} \quad \text{for } k = 1, 2, \ldots$$

# Approximate RL: Planning with Learning (Step 3)

**Question 3**

How to use the learned Q-function $\hat{Q}_\pi$ to find a policy close to optimal $\pi^\star$?

# Approximate RL: Planning with Learning (Step 3)

**Question 3**

How to use the learned Q-function $\hat{Q}_\pi$ to find a policy close to optimal $\pi^\star$?

**Solution:**

If $\hat{Q}_\pi$ is a good approximation of $Q^*_{\mathcal{M}}$, use it to compute the greedy policy.

$$\pi_K(s) = \arg\max_{a \in \mathcal{A}} \hat{Q}_{\theta_K}(s, a)$$

Why would it work?

# An Approximate RL Algorithm: Fitted Q-iteration (FQI)

### Three Components

1. Sample trajectories of $\{(s_i, a_i, r_i)\}_{i=1}^{n}$, and solve the regression problem with $r_i + \gamma \max_b Q_\theta(s_i', b)$ as data and $Q_\theta$ as the parametric function to learn $\theta^\star$.
2. Use the $Q_{\theta_{k-1}}$ as the target function to generate data and apply regression iteratively.
3. If $\hat{Q}_{\theta_K}$ is a good approximation of $Q^*_{\mathcal{M}}$, use it to compute the greedy policy.

**Algorithm** Fitted Q-Iteration (FQI)

1: **Input:** Steps $K$ , number of samples $n$, (an initial state distribution $\rho$ and initial policy $\pi_0$) (alternatively, a sampling distribution)
2: **Initialise:** parameter of Q-value function $\theta_0$
3: **for** episodes $k = 1, 2, \ldots, K$ **do**
4: Draw $n$ samples $(s_i, a_i) \sim \rho\pi_0$.
5: Draw $n$ next states $s_i' \sim \mathcal{P}(\cdot|s_i, a_i)$ and rewards $r_i \sim \mathcal{R}(s_i, a_i)$
6: Create a dataset $\mathcal{H}_k = \{(s_i, a_i), y_i\}$ such that $y_i \triangleq r_i + \gamma \max_b Q_{\theta_k}(s_i', b)$
7: Solve the regression problem and compute $\hat{Q}_{\theta_k}$

$$\theta_{k+1} = \arg\min_{\theta} \sum_{i=1}^{n} \Big(Q_\theta(s_i, a_i) - r_i - \gamma \max_b Q_{\theta_k}(s_i', b)\Big)^2$$

8: **end for**
9: **return** the **greedy policy** for all $s$

$$\pi_K(s) \in \arg\max_{a \in \mathcal{A}} \hat{Q}_{\theta_K}(s, a)$$

# Why Does FQI Work?

**Performance Loss to Learning Error: Contraction of Greedy Policy**

$$\left\| V^*_{\mathcal{M}} - \hat{V}_{\pi_K, \theta_K} \right\|_\infty \leq \frac{2\gamma}{1-\gamma} \left\| V^*_{\mathcal{M}} - \hat{V}_{\theta_K} \right\|_\infty$$

# Why Does FQI Work?

**Performance Loss to Learning Error: Contraction of Greedy Policy**

$$\left\|V^*_{\mathcal{M}} - \hat{V}_{\pi_K,\theta_K}\right\|_\infty \leq \frac{2\gamma}{1-\gamma}\left\|V^*_{\mathcal{M}} - \hat{V}_{\theta_K}\right\|_\infty$$

**From Learning Error to Estimation and Approximation Errors (Lazaric et al., 2012)**

$$\left\|V^*_{\mathcal{M}} - \hat{V}_{n,\theta_K}\right\|_{\mathcal{P}\pi} \leq \left\|V^*_{\mathcal{M}} - \hat{V}_{n}\right\|_{\mathcal{P}\pi} + \left\|\hat{V}_{n} - \hat{V}_{n,\theta_K}\right\|_{\mathcal{P}\pi}$$

- Estimation Error: Depends on the complexity of the function class and the coverage of samples collected
- Approximation Error: How good is the function class to approximate the optimal value function and generalise across state-action space.

# Limitations of FQI

- FQI is an Offline RL algorithm.
- FQI loops over all possible actions to get next best action $a_{t+1}$:

$$\arg\max_{a \in \mathcal{A}} Q_\theta^k(s_t, a)$$

- FQI encounters instability (target depends on $Q_\theta^{k-1}(s_{t+1}, a)$).
- Collects data at every episode and forget them in the next one.

# From FQI to DQN: Baby Steps to Deep RL

- DQN is an Online RL algorithm
- One forward pass to get all $Q_{\theta_k}(s_t, a)$
- Use a target network $Q_{\theta_{k-1}}(s_{t+1}, a)$ to ensure stability
- Uses replay buffer to reuse the data

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
epsilon-greedy exploration
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
step in the environment
replay buffer and sampling
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
regression target
gradient step
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  **end for**
**end for**

Courtesy: Antonin Raffin, RLSS 2023

# The Deadly Triad of RL

If we use FQI or DQN with experience replay, we face the deadly triads of RL (Hasselt et al., 2018).

1. **Function approximation:** Using a neural network or linear model to fit Q-values.
2. **Bootstrapping:** Using $\max_a Q_\theta(s, a)$ to construct the target data.
3. **Off-policy learning:** Replay buffer holds data from a mixture of past policies.

# The Deadly Triad of RL

If we use FQI or DQN with experience replay, we face the deadly triads of RL (Hasselt et al., 2018).

1. **Function approximation:** Using a neural network or linear model to fit Q-values.
2. **Bootstrapping:** Using $\max_a Q_\theta(s, a)$ to construct the target data.
3. **Off-policy learning:** Replay buffer holds data from a mixture of past policies.

The Silver Lining or Myopia?

Empirically, we rarely see the deadly triad appearing destructively, while some explosions of Q-value that recover after an initial phase are common (soft divergence).

Do we need a better and new approach to approximate RL theory?

# Intermediate Summary: Value-based Approximate RL

- Leverage Bellman operator's contraction to iteratively approximate $Q^*_{\mathcal{M}}$.
- Parametrise the Q-value functions and turn learning it into an iterative regression problem.

$$\theta_{k+1} = \arg\min_{\theta} \boxed{\sum_{i=1}^{n} \left( Q_\theta(s_i, a_i) - r_i - \gamma \max_b Q_{\theta_k}(s'_i, b) \right)^2} \quad \text{for } k = 1, 2, \ldots, K\,.$$

- Use a "good" data-generating policy to cover the state-action space and/or reuse the old collected data "smartly".
- Use greedy policy once a good approximation $Q_{\theta_K}$ is computed.

# PART 3

## Policy Gradient Algorithms
## From Dynamic Programming to Parametric Policy Optimisation

**Step 1: Policy Parametrisation.** Represent the probability distribution over actions, i.e. a stochastic policy $\pi : \mathcal{S} \rightarrow \Delta_{\mathcal{A}}$, as a parametric family $(\pi_\theta)$.

# Policy-based Algorithms

Step 1: Policy Parametrisation. Represent the probability distribution over actions, i.e. a stochastic policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, as a parametric family $(\pi_\theta)$.

## Discrete Actions

1. Direct Parametrisation:

$$\pi_\theta(a|s) = \theta_{s,a} \text{ such that } \sum_{s,a} \theta_{s,a} = 1 \text{ and } \theta_{s,a} \geq 0.$$

2. Log-linear Policy:

$$\pi_\theta(a|s) = \frac{\exp(\theta^\top \phi(a,s))}{\sum_{s,a} \exp(\theta^\top \phi(a,s))}.$$

3. Neural Softmax Policy:

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(a,s))}{\sum_{s,a} \exp(f_\theta(a,s))}.$$

Step 1: Policy Parametrisation. Represent the probability distribution over actions, i.e. a stochastic policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, as a parametric family $(\pi_\theta)$.

### Discrete Actions

1. Direct Parametrisation
2. Log-linear Policy
3. Neural Softmax Policy

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(a,s))}{\sum_{s,a} \exp(f_\theta(a,s))} .$$

### Continuous Actions

1. Gaussian:

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\, \sigma_\theta^2(s)}} \exp\left( \frac{(a - \mu_\theta(s))^2}{2\, \sigma_\theta^2(s)} \right) .$$

2. Beta (for Bounded Actions) (Chou et al., 2017):

$$\pi_\theta(a|s) = \mathrm{Beta}\left( \frac{a + A_{\max}}{2A_{\max}} ; \alpha_\theta(s), \beta_\theta(s) \right) .$$

# Policy-based Algorithms

**Step 1: Policy Parametrisation.** Represent the probability distribution over actions, i.e. a stochastic policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, as a parametric family $(\pi_\theta)$.

**Step 2: Policy Optimisation.** Find the parameter $\theta^\star$ that maximises the long-term expected reward

$$\begin{aligned}\theta^\star &= \arg\max_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma \mathcal{R}(s_t, a_t) \mid s_0 \sim \rho, a_t \sim \pi_\theta(\cdot|s_t)\right] \\ &= \arg\max_\theta \mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]\end{aligned}$$

Here, $\rho$ is the initial state distribution.

# Policy-based Algorithms

**Step 1: Policy Parametrisation.** Represent the probability distribution over actions, i.e. a stochastic policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, as a parametric family $(\pi_\theta)$.

**Step 2: Policy Optimisation.** Find the parameter $\theta^\star$ that maximises the long-term expected reward

$$\begin{aligned}\theta^\star &= \arg\max_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma \mathcal{R}(s_t, a_t) \mid s_0 \sim \rho, a_t \sim \pi_\theta(\cdot|s_t)\right] \\ &= \arg\max_\theta \mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]\end{aligned}$$

Here, $\rho$ is the initial state distribution.

**The Hill Ahead**

$\mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]$ is non-concave in $\theta$.

Apply gradient ascent on $J(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]$

$$\theta_{k+1} \leftarrow \theta_k + \eta_k \nabla_\theta J(\pi_\theta)\,.$$

# Policy Gradient Algorithms

Apply gradient ascent on $J(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]$

$$\theta_{k+1} \leftarrow \theta_k + \eta_k \nabla_\theta J(\pi_\theta)\,.$$

**Issue**

We cannot exactly compute the gradient of $J(\pi_\theta)$.

Apply gradient ascent on $J(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho}\left[V^{\pi_\theta}(s_0)\right]$

$$\theta_{k+1} \leftarrow \theta_k + \eta_k \nabla_\theta J(\pi_\theta)\,.$$

**Issue**

We cannot exactly compute the gradient of $J(\pi_\theta)$.

**Solution: Stochastic Approximation**

Construct a stochastic estimate of $\nabla_\theta J(\pi_\theta)$ from data collected by playing $\pi_{\theta_k}$.

**Research Question**

How to construct a "good" estimator of $\nabla_\theta J(\pi_\theta)$?

**Theorem (Policy Gradient Theorem (Williams, 1992))**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[Z(\tau)\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

**Theorem (Policy Gradient Theorem (Williams, 1992))**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[ Z(\tau) \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- $\tau$ is a trajectory $\{s_1, a_1, \ldots, s_t, a_t, \ldots\}$ generated by the probability distribution induced by policy $\pi_\theta$ and transition function $\mathcal{P}$:

$$\mathcal{P}_\theta(\tau) = \rho(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) \mathcal{P}(s_{t+1}|s_t, a_t)$$

# The Policy Gradient Theorem

**Theorem (Policy Gradient Theorem (Williams, 1992))**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[ Z(\tau) \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- $\tau$ is a trajectory $\{s_1, a_1, \ldots, s_t, a_t, \ldots\}$ generated by the probability distribution induced by policy $\pi_\theta$ and transition function $\mathcal{P}$:

$$\mathcal{P}_\theta(\tau) \triangleq \rho(s_0) \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t)\mathcal{P}(s_{t+1}|s_t, a_t)$$

- $Z(\tau)$ is the return from the trajectory $\tau$: $Z(\tau) \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$.

# The Policy Gradient Theorem

**Theorem (Policy Gradient Theorem (Williams, 1992))**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[ Z(\tau) \sum_{t=0}^{\infty} \boxed{\nabla_\theta \log \pi_\theta(a_t|s_t)} \right]$$

- $\tau$ is a trajectory $\{s_1, a_1, \dots, s_t, a_t, \dots\}$ generated by the probability distribution $\mathcal{P}_\theta(\tau)$ induced by policy $\pi_\theta$ and transition function $\mathcal{P}$.
- $Z(\tau)$ is the return from the trajectory $\tau$: $Z(\tau) \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$.
- The gradient $\nabla_\theta \log \pi_\theta(a_t|s_t) = \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ is called the score function and exists for differentiable parametric policies.

**Example: Score Function of log-linear Policies**

If $\pi_\theta(a|s) = \frac{\exp(\theta^\top \phi(a,s))}{\sum_{s,a} \exp(\theta^\top \phi(a,s))}$, then $\nabla_\theta \log \pi_\theta(a|s) = \phi(a,s) - \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\phi(a,s)]$.

**Algorithm** REINFORCE

1: **Input:** Learning rate $\eta$, episode number $K$
2: **Initialise:** Initial policy parameter $\theta_0$
3: **for** episodes $k = 0, \ldots, K$ **do**
4: Generate a trajectory $\tau_K$ from policy $\pi_{\theta_k}$
5: Estimate the gradient at $\theta = \theta_k$:

$$\nabla_\theta^{REINFORCE} J(\pi_\theta) \leftarrow \left(\sum_{t=0}^{\infty} \gamma_t r_t\right) \left(\sum_{t=0}^{\infty} \gamma_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right)$$

6: Apply policy gradient ascent

$$\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta^{REINFORCE} J(\pi_\theta) \mid_{\theta=\theta_k}$$

7: **end for**

**Algorithm** REINFORCE

**Input:** Learning rate $\eta$, episode number $K$
**Initialise:** Initial policy parameter $\theta_0$
**for** episodes $k = 0, \ldots, K$ **do**
    Generate a trajectory $\tau_K$ from policy $\pi_{\theta_k}$
    Estimate the gradient at $\theta = \theta_k$:

$$\nabla_\theta^{REINFORCE} J(\pi_\theta) \leftarrow \left(\sum_{t=0}^{\infty} \gamma_t r_t\right) \left(\sum_{t=0}^{\infty} \gamma_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right)$$

    Apply policy gradient ascent

$$\theta_{k+1} \leftarrow \theta_k + \eta \nabla_\theta^{REINFORCE} J(\pi_\theta) \mid_{\theta=\theta_k}$$

**end for**

- A single *infinite-length* trajectory is enough to create an unbiased estimate without learning transition $\mathcal{P}$.
- Estimator has high variance due to correlation of $Z(\tau)$ and $\{\pi_\theta(a_t|s_t)\}_t$.

How to leverage the Markovianity in estimation?

**Q-value Function version of Policy Gradient Theorem**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\boxed{\left(\sum_{t=0}^{\infty} \gamma^s r_i\right)} \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \boxed{\left(\sum_{i=t}^{\infty} \gamma^i r_i\right)} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

$$= \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \boxed{\gamma^t Q_{\pi_\theta}(s_t, a_t)} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

How to decrease variance of the estimator?

Baseline Function version of Policy Gradient Theorem

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \boxed{\gamma^t Q_{\pi_\theta}(s_t, a_t)} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

$$= \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \left(\boxed{Q_{\pi_\theta}(s_t, a_t) - b(s_t)}\right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

if the baseline function satisfies $\mathbb{E}_{a \sim \pi_\theta}[b(s)\nabla_\theta \log \pi_\theta(a|s)] = 0$.

How to decrease variance of the estimator?

**Baseline Function version of Policy Gradient Theorem**

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \left( Q_{\pi_\theta}(s_t, a_t) - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{4}$$

if the baseline function satisfies $\mathbb{E}_{a \sim \pi_\theta}[b(s)\nabla_\theta \log \pi_\theta(a|s)] = 0$.

A good choice of $b(s)$ is $V_{\pi_\theta}(s)$.

**Advantage Function version of Policy Gradient Theorem**

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \left( Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\ &= \mathbb{E}_{\tau \sim \mathcal{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)\right]\end{aligned}$$

## Natural Policy Gradient (NPG): The Covariance Corrected Estimator

**Algorithm** Natural Policy Gradient (NPG)

1: **Input:** Learning rate $\eta$, episode number $K$
2: **Initialise:** Initial policy parameter $\theta_0$
3: **for** episodes $k = 0, \ldots, K$ **do**
4: Generate a trajectory $\tau_K$ from policy $\pi_{\theta_k}$
5: Estimate the gradient at $\theta = \theta_k$
6: Estimate the covariance $\Sigma_{\theta_k}$ at $\theta = \theta_k$
7: Apply covariance/curvature-calibrated policy gradient ascent

$$\theta_{k+1} \leftarrow \theta_k + \eta \left(\Sigma_{\theta_k}\right)^{\dagger} \hat{\nabla}_\theta J(\pi_\theta) \,|_{\theta=\theta_k}$$

8: **end for**

Here, $\Sigma_{\theta_k} = \mathbb{E}_{\tau \sim \mathcal{P}_{\theta_k}} \left[ \nabla_\theta \log \pi_\theta(a|s) \left(\nabla_\theta \log \pi_\theta(a|s)\right)^\top \right]$

# Different Proximal Estimators and Optimisers: PPO, TRPO,...

We can reinterpret the NPG as a proximal gradient ascent step:

$$\theta_{k+1} = \arg\max_{\theta} J(\pi_\theta) \quad \text{s.t.} \quad \boxed{\mathrm{KL}\left(\mathcal{P}_{\theta_k} || \mathcal{P}_\theta\right) \leq \epsilon}\,.$$

where we do a second order approximation of KL: $\boxed{\frac{1}{2}(\theta - \theta_k)^\top \Sigma_{\theta_k} (\theta - \theta_k) \leq \epsilon}$.

We can reinterpret the NPG as a proximal gradient ascent step:

$$\theta_{k+1} = \arg\max_{\theta} J(\pi_\theta) \quad \text{s.t.} \quad \boxed{\mathrm{KL}\left(\mathcal{P}_{\theta_k} || \mathcal{P}_\theta\right) \leq \epsilon}\,.$$

where we do a second order approximation of KL: $\boxed{\frac{1}{2}(\theta - \theta_k)^\top \Sigma_{\theta_k}(\theta - \theta_k) \leq \epsilon}$.

Success of this approach motivates development of different proximal policy gradient algorithms.

### TRPO (Schulman et al., 2015)

$$\max_{\theta} \mathbb{E}_{\tau \sim \mathcal{P}_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_\theta}(s,a)\right] \quad \text{s.t.} \quad \mathbb{E}_{s \sim \mathcal{P}_{\theta_k}}\left[\mathrm{KL}\left(\pi_\theta(\cdot|s) || \pi_{\theta_k}(\cdot|s)\right)\right] \leq \epsilon\,.$$

### PPO (Schulman et al., 2017)

$$\max_{\theta} \mathbb{E}_{\tau \sim \mathcal{P}_{\theta_k}}\left[\min\left\{\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_\theta}(s,a), \mathrm{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}; 1+\delta, 1-\delta\right) A_{\pi_\theta}(s,a)\right\}\right]\,.$$

### Optimisation Perspective

When $J(\pi_\theta)$ is a "concave-like" function, the stochastic gradient ascent would work.

$$J(\pi_{\theta^\star}) - J(\pi_\theta) = \mathcal{O}\left(\|\nabla_\theta J(\pi_\theta)\|\right) .$$

[1] 1. Lin Xiao. On the convergence rates of policy gradient methods. JMLR, 2022.
2. Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. JMLR, 2021.
3. Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. COLT, 2020.

# When Does Policy Gradient Work?: Convergence Analysis

### Optimisation Perspective

When $J(\pi_\theta)$ is a "concave-like" function, the stochastic gradient ascent would work.

$$J(\pi_{\theta^\star}) - J(\pi_\theta) = \mathcal{O}\left(\|\nabla_\theta J(\pi_\theta)\|\right) .$$

### Statistical Perspective

If we have data with enough "coverage" of the state-action space and we have an unbiased estimator, we can apply the policy gradient theorems almost surely.

$$\|\nabla_\theta J(\pi_\theta) - \widehat{\nabla}_\theta J(\pi_\theta)\| \leq \epsilon(\#samples, \delta) \quad \text{with probability } 1-\delta.$$

For details, check some interesting works below.[1]

[1] 1. Lin Xiao. On the convergence rates of policy gradient methods. JMLR, 2022.
2. Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. JMLR, 2021.
3. Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. COLT, 2020.

# Challenges in Policy-based RL Algorithm Design

1. Hard to choose the good stepsize
   - Use clipping, hyperparameter tuning
2. High sample complexity if we cannot use the samples collected from previous policies
   - Use importance sampling, replay buffer
3. The stochastic gradient estimators suffer high variance
   - Use baseline functions with actor-critic methods

# PART 4

## What's ahead?
## Actor-Critic Algorithms, Exploration–Exploitation Trade-offs, and ....

# Actors vs. Critics: Value-based RL vs. Policy-based RL

Value-based RL (Critics)

**Approach:** Learn the optimal Value or Q-value function

**Algorithms:** Value/Q-value Iteration, Q-learning, Fitted Q-iteration, DQN

Pros: Low variance, good convergence guarantees

Cons: Scales badly with dimensions

# Actors vs. Critics: Value-based RL vs. Policy-based RL

Value-based RL (Critics)

**Approach:** Learn the optimal Value or Q-value function

**Algorithms:** Value/Q-value Iteration, Q-learning, Fitted Q-iteration, DQN

Pros: Low variance, good convergence guarantees

Cons: Scales badly with dimensions

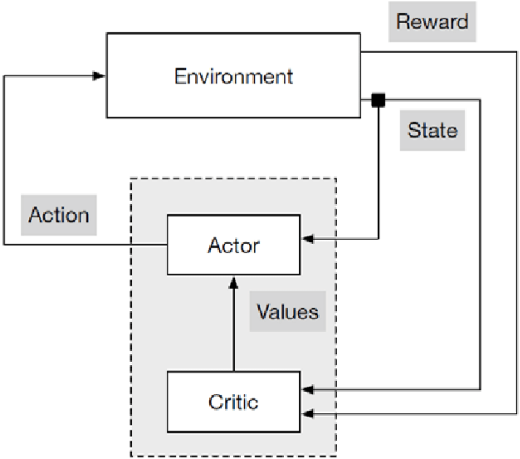
Policy-based RL (Actors)

**Approach:** Learn the optimal (parametrised) policy directly

**Algorithms:** Policy Iteration, REINFORCE, NPG, TRPO, PPO

Pros: Scales for large state-action spaces

Cons: High variance and sample complexity


Reward
Environment
State
Action
Actor
Values
Critic

# What We Have not Covered?

- What are the theoretical guarantees of RL algorithms? How to derive them?
  $\rightarrow$ Convergence analysis
  $\rightarrow$ Regret upper bounds
  $\rightarrow$ Stability analysis
  $\rightarrow$ Sample-complexity bounds

- How to understand generalisation ability of the learned function approximators and corresponding RL policies?
  $\rightarrow$ Learning theory and generalisation errors meet RL

- How to explore either while collecting data for RL training or while running the RL algorithm itself?
  $\rightarrow$ Exploration–exploitation trade-offs

- How to be robust and safe while learning and execution?
  $\rightarrow$ Robust MDPs and Safe RL

**Thanks to our collaborators, teachers, and the audience!**

Questions?

# References I