Diogo Ribeiro

ESMAD - Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

dfr@esmad.ipp.pt 0009-0001-2022-7072

November 16, 2025

Outline

What is Reinforcement Learning?

Reinforcement Learning

A computational approach to learning from interaction with an environment through trial and error, guided by a reward signal.

Key Components:

- **Agent:** The learner and decision maker
- **Environment:** What the agent interacts with
- **State:** Current situation of the agent
- **Action:** Choices available to the agent
- **Reward:** Feedback signal from environment

Goal: Learn a policy π that maximizes cumulative reward over time.

RL vs Other ML Paradigms

Supervised Learning:

- Learning from labeled examples
- Direct feedback (correct answer)
- i.i.d. data assumption
- Immediate evaluation

Unsupervised Learning:

- Finding patterns in data
- No explicit feedback
- Structure discovery

Reinforcement Learning:

- Learning from interaction
- Delayed, evaluative feedback
- Sequential, correlated data
- Exploration vs exploitation
- Credit assignment problem

Key Difference

RL must discover which actions yield the most reward through trial and error.

Applications of Reinforcement Learning

Game Playing:

- AlphaGo, AlphaZero (board games)
- Atari games, StarCraft II
- Poker (Libratus, Pluribus)

Robotics:

- Robot manipulation and grasping
- Locomotion and navigation
- Autonomous vehicles

Business Applications:

- Recommendation systems
- Resource allocation and scheduling
- Automated trading
- Energy management
- Healthcare treatment optimization

Markov Decision Process (MDP)

MDP

A tuple (S, A, P, R, γ) where:

- S - State space
- A - Action space
- $P(s'|s, a)$ - Transition probability function
- $R(s, a, s')$ - Reward function
- $\gamma \in [0, 1)$ - Discount factor

Markov Property

The future is independent of the past given the present:

$$\mathbb{P}(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots, S_0) = \mathbb{P}(S_{t+1}|S_t, A_t)$$

Policy and Value Functions

Policy

A policy $\pi : S \to \Delta(A)$ is a mapping from states to probability distributions over actions.

- **Deterministic:** $\pi(s) = a$
- **Stochastic:** $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$

Value Functions

State-value function:

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s\right]$$

Bellman Equations

Bellman Expectation Equations

For any policy π:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

$$Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')]$$

Bellman Optimality Equations

For the optimal policy π^*:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Optimal Policy

Existence of Optimal Policy

For any finite MDP, there exists an optimal deterministic policy π^* such that:

$$V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall s \in S, \forall \pi$$

Extracting optimal policy from Q^*:

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Extracting optimal policy from V^*:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')]$$

Note

Dynamic Programming Approach

Requirements:

- Complete knowledge of MDP dynamics (P, R)
- Finite state and action spaces

Two main approaches:

1. **Policy Iteration:**
   - Policy evaluation: Compute V^π for current policy
   - Policy improvement: Update policy greedily w.r.t. V^π
   - Repeat until convergence
2. **Value Iteration:**
   - Iteratively update value function using Bellman optimality equation
   - Extract optimal policy from converged value function

Convergence

Both policy iteration and value iteration converge to optimal policy π^* and value function V^*.

Value Iteration Algorithm

Value Iteration

Input: MDP (S, A, P, R, γ), threshold $\theta > 0$
Output: Optimal value function V^* and policy π^*

```
1. Initialize $V(s) = 0$ for all $s \in S$
2. Repeat:
     • $\Delta \leftarrow 0$
     • For each $s \in S$:
         • $v \leftarrow V(s)$
         • $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
         • $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   Until $\Delta < \theta$
3. Extract policy: $\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
```

Complexity: $O(|S|^2|A|)$ per iteration.

Policy Iteration Algorithm

Policy Iteration

Initialize: Arbitrary policy π

Repeat:

1. **Policy Evaluation:**
   - Solve system of equations: $V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$
   - Or iterate until convergence
2. **Policy Improvement:**
   - $\pi'(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$

Until $\pi' = \pi$

Advantage

Often converges in fewer iterations than value iteration, but each iteration is more ex

Monte Carlo RL

Key idea: Learn from complete episodes of experience.

Advantages:

- Model-free: Don't need to know P or R
- Learn from actual or simulated experience
- Can be used with non-Markov environments

Requirements:

- Episodes must terminate
- Can only update after episode completion

Return

The return from time step t is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Monte Carlo Policy Evaluation

Goal: Estimate $V^\pi(s)$ from episodes following policy π.

First-Visit MC

For each episode:

1. Generate episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T$
2. For each state s appearing in episode (first time only):
    - Compute return G_t from that occurrence
    - Append G_t to list $\text{Returns}(s)$
    - $V(s) \leftarrow \text{average}(\text{Returns}(s))$

Convergence

By the law of large numbers, $V(s) \to V^\pi(s)$ as the number of visits to $s \to \infty$.

Monte Carlo Control

Challenge: Policy improvement requires action values $Q(s, a)$, not just $V(s)$.

Solution: Estimate $Q^\pi(s, a)$ instead of $V^\pi(s)$.

MC Exploring Starts

1. Initialize $Q(s, a)$ arbitrarily and $\pi(s)$ arbitrarily
2. **Repeat forever:**
    - Generate episode with random starting state-action pair
    - For each (s, a) pair in episode (first visit):
        - $G \leftarrow$ return following (s, a)
        - Append G to Returns(s, a)
        - $Q(s, a) \leftarrow$ average(Returns(s, a))
    - For each s in episode: $\pi(s) \leftarrow \arg\max_a Q(s, a)$

Exploration vs Exploitation

Fundamental Dilemma

Exploitation: Choose actions that maximize immediate reward based on current knowledge.
Exploration: Try new actions to discover potentially better options.

ϵ-Greedy Policy:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

Other exploration strategies:

- **Boltzmann (Softmax):** $\pi(a|s) \propto \exp(Q(s, a)/\tau)$
- **Upper Confidence Bound (UCB):** Choose $\arg\max_a [Q(s, a) + c\sqrt{\frac{\ln t}{N(s,a)}}]$
- **Optimistic initialization:** Start with high Q values

Temporal Difference (TD) Learning

Key innovation: Learn from incomplete episodes using bootstrapping.

TD(0) Update Rule

After observing transition (S_t, R_{t+1}, S_{t+1}):

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

where α is the learning rate.

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

Comparison:

- **Monte Carlo:** Uses actual return G_t (unbiased, high variance)
- **TD:** Uses estimate $R_{t+1} + \gamma V(S_{t+1})$ (biased, low variance)

TD vs MC vs DP

Property	DP	MC	TD
Model required?	Yes	No	No
Bootstraps?	Yes	No	Yes
Complete episodes?	No	Yes	No
Converges?	Yes	Yes	Yes
Bias	None	None	Initially biased
Variance	Low	High	Low

Advantages of TD:

- Learn online, without waiting for episode termination
- Can learn from incomplete sequences
- Lower variance than MC
- More efficient for many problems

SARSA: On-Policy TD Control

SARSA

State-Action-Reward-State-Action algorithm learns Q^π for the current policy π.

SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily

For each episode:

1. Initialize S
2. Choose A from S using policy derived from Q (e.g., ϵ-greedy)
3. **Repeat for each step:**
    - Take action A, observe R, S'
    - Choose A' from S' using policy derived from Q
    - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    - $S \leftarrow S'$, $A \leftarrow A'$

Q-Learning: Off-Policy TD Control

Q-Learning

Off-policy TD control that learns optimal action-value function Q^* directly.

Q-Learning Algorithm

Initialize $Q(s, a)$ arbitrarily

For each episode:

1. Initialize S
2. **Repeat for each step:**
   - Choose A from S using policy derived from Q (e.g., ϵ-greedy)
   - Take action A, observe R, S'
   - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$
   - $S \leftarrow S'$

On-Policy vs Off-Policy

On-Policy

Learn about and improve the policy being used for action selection.

- Example: SARSA
- Learns Q^π for current policy π

Off-Policy

Learn about optimal policy while following a different behavior policy.

- Example: Q-learning
- Learns Q^* while following ϵ-greedy policy

Off-policy advantages:

Need for Function Approximation

Problem: Tabular methods don't scale to large state/action spaces.

Examples of large state spaces:

- Backgammon: $\approx 10^{20}$ states
- Go: $\approx 10^{170}$ states
- Continuous state spaces: infinite states

Solution: Approximate value functions using function approximators.

Function Approximation

Represent value function with parameters $\boldsymbol{w}$:

$$\hat{V}(s;\boldsymbol{w}) \approx V^{\pi}(s) \quad \text{or} \quad \hat{Q}(s,a;\boldsymbol{w}) \approx Q^{\pi}(s,a)$$

Common approximators:

- Linear combinations of features

Gradient-Based Value Function Approximation

Objective: Minimize mean squared error between true and estimated values.

Gradient descent update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{2}\alpha \nabla_{\boldsymbol{w}} [V^\pi(S_t) - \hat{V}(S_t; \boldsymbol{w}_t)]^2$$

Semi-gradient TD(0):

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha [R_{t+1} + \gamma \hat{V}(S_{t+1}; \boldsymbol{w}_t) - \hat{V}(S_t; \boldsymbol{w}_t)] \nabla_{\boldsymbol{w}} \hat{V}(S_t; \boldsymbol{w}_t)$$

Semi-gradient Methods

Called "semi-gradient" because we don't differentiate through the target $R_{t+1} + \gamma \hat{V}(S_{t+1}; \boldsymbol{w}_t)$.
This introduces bias but works well in practice.

Linear Function Approximation

Feature representation: $\boldsymbol{x}(s) = [x_1(s), x_2(s), \ldots, x_n(s)]^T$

Linear Value Function

$$\hat{V}(s; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}(s) = \sum_{i=1}^{n} w_i x_i(s)$$

Gradient:

$$\nabla_{\boldsymbol{w}} \hat{V}(s; \boldsymbol{w}) = \boldsymbol{x}(s)$$

Linear TD(0) update:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha[R_{t+1} + \gamma \boldsymbol{w}_t^T \boldsymbol{x}(S_{t+1}) - \boldsymbol{w}_t^T \boldsymbol{x}(S_t)]\boldsymbol{x}(S_t)$$

Advantages:

- Simple and computationally efficient

Deep Q-Networks (DQN)

Breakthrough: Mnih et al. (2015) - Human-level control through deep RL.

Key innovations:

1. **Experience Replay:**
    - Store transitions (s, a, r, s') in replay buffer
    - Sample random mini-batches for training
    - Breaks correlation between consecutive samples
2. **Target Network:**
    - Separate network $\hat{Q}(s, a; \boldsymbol{w}^-)$ for computing targets
    - Updated periodically from main network
    - Stabilizes training

Loss function:

$$L(\boldsymbol{w}) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-) - \hat{Q}(s, a; \boldsymbol{w})\right)^2\right]$$

DQN Algorithm

Deep Q-Network Algorithm

Initialize:

- Replay buffer D with capacity N
- Q-network $\hat{Q}(s, a; \boldsymbol{w})$ with random weights $\boldsymbol{w}$
- Target network $\hat{Q}(s, a; \boldsymbol{w}^-)$ with $\boldsymbol{w}^- = \boldsymbol{w}$

For each episode:

1. Observe initial state s
2. **For each step:**
    - Select action: $a = \begin{cases} \text{random} & \text{with prob. } \epsilon \\ \arg\max_{a'} \hat{Q}(s, a'; \boldsymbol{w}) & \text{otherwise} \end{cases}$
    - Execute a, observe r, s'
    - Store (s, a, r, s') in D

Improvements to DQN

Double DQN (van Hasselt et al., 2016):

- Addresses overestimation bias in Q-learning
- Use online network to select actions, target network to evaluate:

$$y = r + \gamma \hat{Q}(s', \arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w}); \boldsymbol{w}^{-})$$

Dueling DQN (Wang et al., 2016):

- Separate value and advantage streams:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a')$$

Prioritized Experience Replay (Schaul et al., 2016):

- Sample important transitions more frequently
- Priority based on TD error: $p_i = |\delta_i| + \epsilon$

Rainbow DQN: Combines all improvements for state-of-the-art performance.

Policy Gradient Approach

Idea: Directly parameterize and optimize the policy.

Parameterized Policy

Policy with parameters $\boldsymbol{\theta}$:

$$\pi(a|s;\boldsymbol{\theta}) = \mathbb{P}(A_t = a|S_t = s;\boldsymbol{\theta})$$

Objective: Maximize expected return

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_\theta}[G_0] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right]$$

Gradient ascent:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$$

Advantages over value-based methods:

Policy Gradient Theorem

Policy Gradient Theorem (Sutton et al.

For any differentiable policy $\pi(a|s;\boldsymbol{\theta})$:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a Q^\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s;\boldsymbol{\theta})$$

where $\mu(s)$ is the on-policy state distribution.

Equivalent form:

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

REINFORCE algorithm (Williams, 1992):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t;\boldsymbol{\theta}_t)$$

Intuition: Increase probability of actions with high returns, decrease probability of actions

Actor-Critic Methods

Problem with REINFORCE: High variance due to Monte Carlo returns.

Solution: Use learned value function as baseline.

Actor-Critic

Two components:

- **Actor:** Policy $\pi(a|s; \boldsymbol{\theta})$
- **Critic:** Value function $\hat{V}(s; \boldsymbol{w})$ or $\hat{Q}(s, a; \boldsymbol{w})$

Update rules:

- **Critic:** TD learning for value function

$$\delta_t = r + \gamma \hat{V}(s'; \boldsymbol{w}) - \hat{V}(s; \boldsymbol{w})$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_w \delta_t \nabla_{\boldsymbol{w}} \hat{V}(s; \boldsymbol{w})$$

- **Actor:** Policy gradient using TD error

Advanced Policy Gradient Methods

Advantage Actor-Critic (A2C):

- Use advantage function: $A(s, a) = Q(s, a) - V(s)$
- Reduces variance while maintaining unbiased gradient

Asynchronous Advantage Actor-Critic (A3C):

- Multiple parallel agents updating shared parameters
- Improves training stability and speed

Proximal Policy Optimization (PPO):

- Constrains policy updates to trust region
- Clipped surrogate objective:

$$L(\boldsymbol{\theta}) = \mathbb{E}[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

  where $r_t(\boldsymbol{\theta}) = \frac{\pi(a|s;\boldsymbol{\theta})}{\pi(a|s;\boldsymbol{\theta}_{\text{old}})}$
- Currently most popular deep RL algorithm

Deterministic Policy Gradient (DPG)

For continuous action spaces: Deterministic policies can be more efficient.

Deterministic Policy

$$a = \mu(s; \boldsymbol{\theta})$$

Maps states directly to actions (no stochasticity).

Deterministic Policy Gradient Theorem

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_s[\nabla_a Q^{\mu}(s, a)|_{a=\mu(s)} \nabla_{\boldsymbol{\theta}} \mu(s; \boldsymbol{\theta})]$$

Deep Deterministic Policy Gradient (DDPG):

- Actor-critic with deterministic policy

Multi-Agent RL (MARL)

Multi-Agent Setting

Multiple agents learning simultaneously in shared environment.

- Each agent has own policy and objectives
- Actions of one agent affect others
- Environment becomes non-stationary from each agent's perspective

Types of multi-agent settings:

- **Cooperative:** Agents share common goal (team games)
- **Competitive:** Agents have conflicting goals (zero-sum games)
- **Mixed:** Combination of cooperation and competition

Challenges:

- Non-stationarity: Other agents' policies change during learning
- Credit assignment: Which agent contributed to success/failure?

Game Theoretic Concepts

Nash Equilibrium

A joint policy $(\pi_1^*, \ldots, \pi_n^*)$ where no agent can improve by unilaterally changing their policy:

$$J_i(\pi_i^*, \pi_{-i}^*) \geq J_i(\pi_i, \pi_{-i}^*) \quad \forall \pi_i, \forall i$$

Self-play:

- Agent trains against copies of itself
- Used in AlphaGo, AlphaZero, OpenAI Five
- Can converge to Nash equilibrium in two-player zero-sum games

Pareto optimality:

- No agent can improve without hurting another
- Relevant for cooperative settings

MARL Algorithms

Independent Q-Learning:

- Each agent learns independently using Q-learning
- Simple but may not converge due to non-stationarity

Centralized Training, Decentralized Execution (CTDE):

- Training: Use global information (all agents' observations)
- Execution: Each agent acts based only on local observations
- Examples: MADDPG, QMIX, COMA

Communication protocols:

- CommNet: Learn to communicate through backpropagation
- DIAL: Differentiable inter-agent learning
- TarMAC: Targeted multi-agent communication

Mean field methods:

- Approximate effect of many agents with mean field
- Scales to very large numbers of agents

Implementing RL Systems

State representation:

- Design informative features
- Use domain knowledge
- Consider frame stacking for temporal information
- Normalization and scaling

Reward shaping:

- Sparse vs dense rewards
- Avoid reward hacking
- Use reward clipping for stability
- Consider intrinsic motivation

Hyperparameters:

- Learning rate: Often needs decay schedule
- Discount factor γ: Typically 0.95-0.99
- Exploration: ϵ decay schedule
- Network architecture: Depends on state complexity

Debugging and Evaluation

Common Issues

- Policy not learning: Check reward scale, learning rate
- Instability: Reduce learning rate, check target network updates
- Overfitting: Add regularization, increase replay buffer
- Slow learning: Improve state representation, reward shaping

Evaluation metrics:

- Average episodic return
- Success rate (for goal-based tasks)
- Convergence speed
- Sample efficiency
- Robustness to initial conditions

Visualization:

RL Libraries and Frameworks

Python Libraries:

- **OpenAI Gym:** Standard environment interface
- **Stable-Baselines3:** High-quality implementations of RL algorithms
- **RLlib (Ray):** Scalable RL with distributed training
- **TF-Agents, Keras-RL:** TensorFlow-based RL
- **PyTorch RL libraries:** Tianshou, rlpyt

Simulation environments:

- **Atari 2600:** Classic benchmark (57 games)
- **MuJoCo:** Continuous control, robotics
- **PyBullet:** Open-source physics simulation
- **Unity ML-Agents:** 3D environments
- **PettingZoo:** Multi-agent environments

Getting started:

- Start with simple environments (CartPole, MountainCar)
- Use pre-implemented algorithms from Stable-Baselines3

Example: Q-Learning with OpenAI Gym

```python
import gym
import numpy as np

# Create environment
env = gym.make('CartPole-v1')

# Initialize Q-table (discretized state space)
n_states = (6, 12)  # bins for position and velocity
n_actions = env.action_space.n
Q = np.zeros(n_states + (n_actions,))

# Hyperparameters
alpha = 0.1  # learning rate
gamma = 0.99  # discount factor
epsilon = 0.1  # exploration rate

def discretize_state(state):
    # Discretize continuous state
    pos_bins = np.linspace(-2.4, 2.4, n_states[0])
    vel_bins = np.linspace(-3, 3, n_states[1])
    return (np.digitize(state[0], pos_bins),
            np.digitize(state[1], vel_bins))

# Training loop
for episode in range(1000):
    state = discretize_state(env.reset())
    done = False

    while not done:
        # Epsilon-greedy action selection
```

Model-Based RL

Idea: Learn a model of environment dynamics, use for planning.

Advantages:

- More sample efficient
- Can imagine/simulate trajectories
- Transfer learning across tasks

Challenges:

- Model errors compound over time
- Difficult in complex, high-dimensional environments

Approaches:

- **Dyna-Q:** Combine model-free and model-based learning
- **PETS:** Probabilistic ensemble for uncertainty
- **World Models:** Learn compressed representations
- **MuZero:** Model-based planning without explicit dynamics model

Hierarchical RL

Motivation: Break down complex tasks into simpler sub-tasks.

Options Framework

An option consists of:

- Initiation set $I \subseteq S$
- Policy $\pi : S \times A \to [0, 1]$
- Termination condition $\beta : S \to [0, 1]$

Benefits:

- Temporal abstraction
- Reusable skills
- Faster learning
- Better exploration

Inverse RL and Imitation Learning

Problem: Reward engineering is difficult in many domains.

Inverse Reinforcement Learning (IRL)

Given expert demonstrations, infer the reward function being optimized.

Imitation Learning

Learn policy directly from expert demonstrations.

Methods:

- **Behavioral Cloning:** Supervised learning from demonstrations
- **DAgger:** Dataset aggregation for interactive imitation
- **GAIL:** Generative adversarial imitation learning
- **Maximum Entropy IRL:** Probabilistic approach to IRL

Meta-RL and Transfer Learning

Meta-RL (Learning to Learn):

- Train on distribution of tasks
- Learn to quickly adapt to new tasks
- Examples: MAML, RL^2, Meta-Q-Learning

Transfer Learning:

- Reuse knowledge from source tasks
- Fine-tuning pre-trained policies
- Progressive neural networks
- Multi-task learning

Sim-to-Real Transfer:

- Train in simulation, deploy in real world
- Domain randomization
- Reality gap challenges

Current Research Frontiers

Safety and Robustness:

- Safe exploration
- Constrained RL
- Adversarial robustness
- Interpretability

Sample Efficiency:

- Reduce data requirements
- Better exploration strategies
- Offline RL (learning from fixed datasets)

Scaling and Generalization:

- Foundation models for RL
- Decision transformers
- Large-scale distributed training
- Zero-shot generalization

Summary

Key Concepts Covered:

- Markov Decision Processes and Bellman equations
- Dynamic programming methods (value iteration, policy iteration)
- Monte Carlo and temporal difference learning
- Q-learning and SARSA
- Function approximation and deep RL
- Policy gradient methods and actor-critic
- Multi-agent reinforcement learning

The RL Landscape:

- **Model-free vs model-based:** Trade-off between simplicity and efficiency
- **Value-based vs policy-based:** Different approaches to optimization
- **On-policy vs off-policy:** Learning and behavior policies

Practical Takeaways

Best Practices

1. Start simple: Use tabular methods for small problems
2. Understand your problem: Is it episodic? What's the state/action space?
3. Choose appropriate algorithm: DQN for discrete actions, PPO/DDPG for continuous
4. Monitor learning: Plot returns, check convergence
5. Tune hyperparameters: Learning rate, exploration, network architecture
6. Use established libraries: Stable-Baselines3, RLlib

Resources for Further Learning:

- Sutton & Barto: "Reinforcement Learning: An Introduction" (2nd ed., 2018)
- Spinning Up in Deep RL (OpenAI)
- DeepMind x UCL RL Lecture Series

The Future of RL

Exciting developments ahead:

- More robust and safe RL algorithms
- Better sample efficiency through model-based methods
- Integration with foundation models and LLMs
- Real-world deployment in robotics, healthcare, autonomous systems
- Multi-agent systems for complex coordination problems

Final Thought

Reinforcement learning is a powerful paradigm for sequential decision making that continues to push the boundaries of what's possible in AI.

The combination of deep learning and RL has already achieved superhuman performance in games, and is poised to transform many real-world domains.

Acknowledgments

- ESMAD for institutional support
- Mysense.ai for industry applications and insights
- The RL research community for groundbreaking work
- OpenAI, DeepMind, and others for open-source contributions

Generated with LaTeX Beamer

Theme: ESMAD Professional Academic Style

Contact Information

Diogo Ribeiro

ESMAD - Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

dfr@esmad.ipp.pt

0009-0001-2022-7072

github.com/diogoribeiro7

linkedin.com/in/diogoribeiro7