Reinforcement Learning: Algorithms and Applications

Learning from Interaction

September 9, 2025

Table of Contents

What is Reinforcement Learning?

- Learning through interaction with an environment
- No explicit supervision - learning from rewards and punishments
- Goal: Learn optimal behavior to maximize cumulative reward
- Inspired by behavioral psychology and animal learning
- Different from supervised and unsupervised learning

Key Characteristics

- **Trial-and-error learning**: Agent explores different actions
- **Delayed consequences**: Actions may have long-term effects
- **Exploration vs Exploitation**: Balance between trying new actions and using known good ones
- **Sequential decision making**: Decisions affect future states
- **No labeled examples**: Learning from scalar reward signals

The Reinforcement Learning Framework

- **Agent**: The learner/decision maker
- **Environment**: Everything the agent interacts with
- **State (S)**: Current situation/configuration
- **Action (A)**: What the agent can do
- **Reward (R)**: Immediate feedback from environment
- **Policy (π)**: Strategy for choosing actions

The Agent-Environment Interaction

At each time step t:

1. Agent observes state S_t
2. Agent selects action A_t based on policy π
3. Environment responds with:
   - Next state S_{t+1}
   - Reward R_{t+1}
4. Process repeats...

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Markov Decision Process (MDP)

An MDP is defined by:

- S: Set of states
- A: Set of actions
- P: Transition probabilities $P(s'|s,a)$
- R: Reward function $R(s,a,s')$
- γ: Discount factor $[0,1]$

Markov Property: Future depends only on current state, not history

$$P(S_{t+1}=s'|S_t=s,A_t=a,S_{t-1},A_{t-1},\ldots)=P(S_{t+1}=s'|S_t=s,A_t=a)$$

Return and Value Functions

Return: Total discounted reward from time t

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

State Value Function: Expected return starting from state s

$$V^\pi(s) = E_\pi[G_t | S_t = s]$$

Action Value Function: Expected return from state s, action a

$$Q^\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

Bellman Equations

Bellman Equation for State Values:

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

Bellman Equation for Action Values:

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

Optimal Bellman Equations:

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

Policy-based vs Value-based Methods

Value-based Methods

- Learn value functions
- Derive policy from values
- Examples: Q-learning, SARSA
- Good for discrete actions

Policy-based Methods

- Directly learn policy
- Parameterized policies
- Examples: REINFORCE, Actor-Critic
- Handle continuous actions well

Actor-Critic Methods: Combine both approaches

- Actor: Policy component
- Critic: Value function component

Q-Learning: Off-Policy Temporal Difference

Key Idea: Learn optimal action values $Q^*(s, a)$ directly

Update Rule:

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

Where:

- α: Learning rate
- r: Immediate reward
- γ: Discount factor
- $\max_{a'} Q(s', a')$: Maximum Q-value in next state

Policy: $\pi(s) = \arg\max_a Q(s, a)$ (greedy)

Q-Learning Algorithm

1. Initialize $Q(s, a)$ arbitrarily for all s, a
2. For each episode:
    1. Initialize state s
    2. For each step of episode:
        1. Choose action a using policy derived from Q (e.g., ϵ-greedy)
        2. Take action a, observe reward r and next state s'
        3. Update: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        4. $s \leftarrow s'$
    3. Until s is terminal

Exploration Strategies

ϵ-**greedy**:

- With probability ϵ: choose random action
- With probability $1 - \epsilon$: choose $\arg\max_a Q(s, a)$

Softmax/Boltzmann:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}}$$

Upper Confidence Bound (UCB):

$$a_t = \arg\max_a \left[Q(s, a) + c\sqrt{\frac{\ln t}{N(s,a)}} \right]$$

Policy Gradient Approach

Parameterized Policy: $\pi_\theta(a|s)$

Objective: Maximize expected return

$$J(\theta) = E_{\pi_\theta}[G_t]$$

Policy Gradient Theorem:

$$\nabla J(\theta) \propto \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla \pi_\theta(a|s)$$

REINFORCE Update:

$$\theta \leftarrow \theta + \alpha G_t \frac{\nabla \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}$$

Actor-Critic Methods

Combines:

- Policy gradient (Actor)
- Value function approximation (Critic)

Actor Update:

$$\theta \leftarrow \theta + \alpha\delta \frac{\nabla \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)}$$

Critic Update:

$$w \leftarrow w + \beta\delta\nabla V_w(S_t)$$

Where $\delta = R_{t+1} + \gamma V_w(S_{t+1}) - V_w(S_t)$ is the TD error

Real-World Applications

- **Game Playing**: Chess, Go, Atari games, StarCraft II
- **Robotics**: Robot navigation, manipulation, walking
- **Autonomous Systems**: Self-driving cars, drones
- **Finance**: Algorithmic trading, portfolio management
- **Healthcare**: Treatment recommendations, drug discovery
- **Resource Management**: Traffic control, power grid optimization
- **Natural Language**: Dialogue systems, machine translation
- **Recommendation Systems**: Content recommendation, advertising

Success Stories

- **AlphaGo/AlphaZero**: Mastered Go, Chess, and Shogi
- **DQN**: Human-level performance on Atari games
- **OpenAI Five**: Competed in Dota 2 tournaments
- **AlphaStar**: Achieved Grandmaster level in StarCraft II
- **GPT/ChatGPT**: Large language models with RL fine-tuning
- **Autonomous Vehicles**: Tesla, Waymo self-driving systems
- **Data Center Cooling**: Google's 40% energy reduction

Current Challenges

- **Sample Efficiency**: Need many interactions to learn
- **Exploration**: Finding good strategies in large state spaces
- **Generalization**: Transferring knowledge to new environments
- **Partial Observability**: Dealing with incomplete information
- **Multi-Agent Settings**: Learning with other agents
- **Safety**: Ensuring safe exploration and deployment
- **Interpretability**: Understanding learned policies
- **Reward Engineering**: Designing appropriate reward functions

Advanced Topics and Extensions

- **Deep Reinforcement Learning**: Neural networks as function approximators
- **Multi-Agent RL**: Learning in multi-agent environments
- **Hierarchical RL**: Learning at multiple temporal abstractions
- **Transfer Learning**: Applying knowledge across domains
- **Imitation Learning**: Learning from expert demonstrations
- **Safe RL**: Incorporating safety constraints
- **Meta-Learning**: Learning to learn quickly
- **Offline RL**: Learning from fixed datasets

Future Directions

- **More Sample-Efficient Algorithms**
- **Better Exploration Strategies**
- **Robust and Safe RL Systems**
- **Integration with Other ML Paradigms**
- **Real-World Deployment Challenges**
- **Ethical Considerations and Fairness**
- **Quantum Reinforcement Learning**
- **Continual and Lifelong Learning**

Key Takeaways

- RL enables learning optimal behavior through interaction
- Balancing exploration and exploitation is crucial
- Value-based and policy-based methods offer different advantages
- Deep RL has achieved remarkable successes in complex domains
- Many challenges remain for real-world deployment
- Active area of research with promising future applications

Thank You

Questions?

"The only way to make sense out of change is to plunge into it, move with it, and join the dance."
- Alan Watts

(This quote reflects the essence of reinforcement learning - learning through interaction and adaptation)