Brief Report

A Brief Tutorial on Reinforcement Learning: From MDP to DDPG

Tian Zhang * and Zhirong Su

Posted Date: 20 February 2026

doi: 10.20944/preprints202601.0420.v2

Keywords: reinforcement learning; tutorial

 doi:10.20944/preprints202601.0420.v2

Brief Report

A Brief Tutorial on Reinforcement Learning: From MDP to DDPG

Tian Zhang * and Zhirong Su

Independent Researcher, China

* Correspondence: tianzhang@sdnu.edu.cn

Abstract

Following the evolution from theoretical foundations to advanced deep learning algorithms, a coherent overview of reinforcement learning (RL) is proposed in this tutorial. We begin with the mathematical formalization of sequential decision-making via Markov decision processes (MDPs). In MDP, Bellman equation and Bellman optimality equation play important roles, they provide for policy evaluation and the fundamental condition for optimal behavior, respectively. The movement from these equations to practical algorithms is explored, starting with model-based dynamic programming and progressing to model-free temporal-difference (TD) learning. As a pivotal model-free algorithm, Q-learning directly implements the Bellman optimality equation through sampling. To handle high-dimensional state spaces, function approximation and deep reinforcement learning emerge, exemplified by Deep Q-Networks (DQN). Thereafter, actor-critic methods address the challenge of continuous action spaces. As a typical actor-critic scheme, the deep deterministic policy gradient (DDPG) algorithm is illustrated in detail on how it adapts the principles of optimality to continuous control by maintaining separate actor and critic networks. Finally, the tutorial concludes with a unified perspective, observing the development of RL as a logical progression from defining optimality conditions to developing scalable solution algorithms. Furthermore, future directions are summarized.

Keywords: reinforcement learning; tutorial

1. Introduction: How Do Machines Learn to Make Decisions?

Imagine teaching a robot to walk—you cannot specify the movement of every muscle in detail, but you can tell it "you'll get a reward for walking well, and lose points for falling." Through repeated attempts, the robot gradually figures out a stable walking method. This is the core idea of reinforcement learning (RL), i.e., an agent learns an optimal decision-making policy through interaction with the environment and trial-and-error [1].

RL has made groundbreaking progress over the past decade, from AlphaGo defeating human Go champions to agents surpassing humans in complex games. Behind these achievements lies a series of sophisticated mathematical frameworks and algorithmic advancements. This tutorial guides you from the most fundamental Markov decision process (MDP), step by step, to explore its core equations and algorithm evolution, up to understanding advanced algorithms like the deep deterministic policy gradient (DDP), unveiling the mystery behind intelligent decision-making systems. The logistic structure of the tutorial is illustrated in Figure 1.

 doi:10.20944/preprints202601.0420.v2

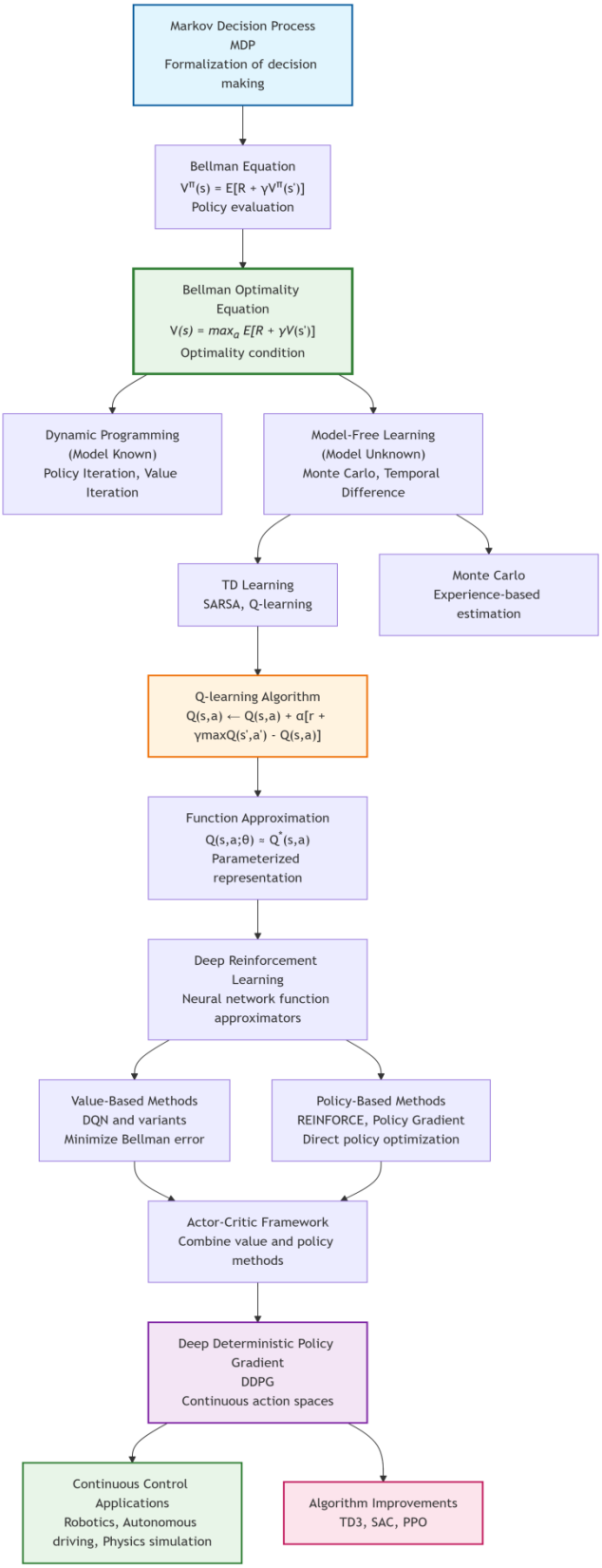

Figure 1. Logistic structure of the tutorial.

 doi:10.20944/preprints202601.0420.v2

2. The Mathematical Foundation of Decision-Making: MDP

In this section, we first introduce the Markov property, and then give the definition of MDP. Finally, the policy and return are explained.

The Markov property is the core assumption of reinforcement learning, it shows that the future state depends only on the current state and not on past states. Mathematically,

$$P(S_{t+1}|S_t)= P(S_{t+1}\,|\,S_t,S_{t-1},...,S_0)$$

where P is the probability, and S_t is the state at time t

This memoryless property greatly simplifies the modeling of decision problems.

Formally, a standard MDP consists of five key elements.

(1) State space (S): The set of all possible situations of the environment.

(2) Action space (A): The set of operations the agent can perform.

(3) Transition probability (P): The probability distribution of state transitions after taking an action.

(4) Reward function (R): The environment's immediate feedback for a state-action pair.

(5) Discount factor (γ): Measures the present value of future rewards ($0 \le \gamma \le 1$).

*Remarks:*The discount factor is similar to the concept of "present value" in economics—100 $ tomorrow is worth less than 100 $ today, and the agent also values immediate rewards more.

A policy π is the agent's decision rule, specifying what action to take in each state. The agent's goal is to maximize the cumulative discounted return as

$$G_t=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+...$$

where G_t is referred to as the return at time t.

3. From Policy Evaluation to Optimal Policy: The Evolution of the Bellman Equation

In this section, we first give the definitions of state-value function and action-value function, respectively. And then the Bellman equation and Bellman optimality equation are discussed successively. Finally, the dynamical programming methods to solve Bellman optimality equation under known model are given.

To evaluate the quality of a policy, RL introduces two core concepts as follows.

1) State-value function $V^{\pi}(s)$: The expected return starting from state s and following policy π.

$$V^{\pi}(s)=\boldsymbol{E}_{\pi}[G_t\,|\,S_t=s]$$

2) Action-value function $Q^{\pi}(s,a)$: The expected return after taking action a in state s and thereafter following policy π.

$$Q^{\pi}(s,a)=\boldsymbol{E}_{\pi}[G_t\,|\,S_t=s,A_t=a]$$

The value functions satisfy an important recursive relationship—the Bellman Equation:

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

This equation reveals a profound insight: the value of the current state equals the immediate reward plus the discounted value of the successor state.

Remarks: The Bellman equation allows us to evaluate the quality of any given policy π, but it does not tell us how to find a better policy.

The ultimate goal of RL is to find the optimal policy π^*. This leads to one of the most central equations in RL theory, i.e., the Bellman optimality equation.

1) Optimality equation for the state-value function:

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

2) Optimality equation for the action-value function:

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

 doi:10.20944/preprints202601.0420.v2

Remarks: The Bellman optimality equation no longer depends on a specific policy π, but directly defines the self-consistent condition that the optimal value functions must satisfy. It tells us: the optimal state value equals the value corresponding to the action that yields the maximum expected return; the optimal action value equals the immediate reward plus the maximum optimal action value of the successor state.

Remarks: Imagine finding the shortest path between cities. The Bellman equation tells you "how long it will take to reach the destination along the current route"; the Bellman optimality equation tells you "the shortest possible time from the current location to the destination, and which direction to go next."

Remarks: From the Bellman equation to the Bellman optimality equation, we complete the conceptual leap from policy evaluation (analyzing a given policy) to policy optimization (finding the optimal policy). This equation is the theoretical foundation for all subsequent optimal control algorithms.

When the environment's dynamics model (P and R) are known, we can solve the Bellman optimality equation using dynamic programming methods [2] as follows.

1)Policy iteration: Alternates between policy evaluation (solving the Bellman equation) and policy improvement.

2)Value iteration: Directly iterates the Bellman optimality equation:

$$V_{k+1}(s)=max_a\Sigma_{s'}P(s'|s,a)[R(s,a,s')+\gamma V_k(s')]$$

Remarks: The value iteration algorithm essentially repeatedly applies the Bellman optimality equation until the value function converges. This verifies that the Bellman optimality equation is not only descriptive but also constructive—it directly provides a solution algorithm.

4. Model-Free Learning: From Theory to Practice

In realistic circumstances, the environment model is often unknown. In this section, the model-free scenario is discussed. Monte Carlo (MC) methods and temporal difference (TD) learning are given at the beginning.

1) *Monte Carlo methods:* Estimate the value function using returns from complete trajectories.

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

2) *Temporal difference (TD) learning:* Combines ideas from Monte Carlo and dynamic planning, using estimated values for updates.

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

where the TD error $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ measures the discrepancy between prediction and actual outcome.

Q-learning is a milestone algorithm in RL history [3]. As a practical breakthrough from the Bellman optimality equation. Its update rule derives directly from the Bellman optimality equation as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

*Remarks:*Comparing the Q-learning update formula with the Bellman optimality equation, you'll find that Q-learning is approximately solving the optimal equation through sampling. $max_a Q(S_{t+1}, a)$ corresponds to $max_{a'} Q^*(s', a')$ in the optimal equation, and the entire update moves towards the optimal Q-value.

Remarks: Q learning is an off-policy learning, i.e., the agent learns the optimal action-value function to find the optimal policy, while potentially exploring other behaviors. This allows it to directly estimate the optimal Q-function without depending on the currently executed policy.

When the state space is huge or continuous, tabular methods become infeasible. The function approximation scheme emerges, that uses parameterized functions (e.g., neural networks) to represent value functions or policies as

$$Q(s,a;\theta)\approx Q^{\pi}(s,a)$$

 doi:10.20944/preprints202601.0420.v2

where θ are learnable parameters. This paves the way for combining deep learning with reinforcement learning.

5. The Rise of Deep Reinforcement Learning

Proposed by DeepMind in 2015, the deep Q-networks (DQN) achieves human-level performance on Atari games [4]. The main innovations include

1) *Experience replay:* Stores transition samples (s,a,r,s') and randomly samples from them to break data correlations.

2) *Target network:* Uses a separate target network to compute the TD target, stabilizing the learning process.

3) *End-to-end learning:* Learns control policies directly from raw pixel input.

The DQN loss function essentially minimizes the error of the Bellman optimality equation as

$$L(\theta) = \boldsymbol{E}[(r + \gamma max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

where θ^- are the target network parameters.

Remarks: DQN can be viewed as a practical implementation that approximates the Q-function with a deep neural network and solves the Bellman optimality equation using stochastic gradient descent.

6. The Deep Deterministic Policy Gradient (DDPG) Algorithm

Many practical problems (e.g., robot control, autonomous driving) require decision-making in continuous action spaces [6]. Discrete-action methods like DQN face the curse of dimensionality, i.e., if each dimension has m possible values, a d-dimensional action space has m^d possible actions. More importantly, the $max_a Q(s, a)$ operation in DQN becomes computationally intractable in continuous spaces.

Unlike value-based methods, policy gradient (PG) methods directly parameterize the policy $\pi(a|s;\theta)$ and optimize the expected return [5]. The fundamental theorem is

$$\nabla_\theta J(\theta) = \boldsymbol{E}_\pi[\nabla_\theta \log \pi(A_t|S_t;\theta) Q^\pi(S_t, A_t)]$$

Remarks: Increase the probability of actions that lead to high returns and decrease the probability of actions that lead to low returns.

The REINFORCE is a typical MC-PG method. The sampling efficiency of REINFORCE is low. Therefore, by combing value-based and policy-based methods, the DDPG emerges.

The DDPG is an actor-critic algorithm [7] designed for continuous control, cleverly avoiding the maximization problem. The characteristic of DDPG is

1) *Deterministic policy:* $a=\mu(s|\theta^\mu)$ directly outputs a deterministic action.

2) *Actor-critic architecture:*

actor network (policy network):$\mu(s|\theta^\mu)$, responsible for selecting actions.

critic network (value function network): $Q(s,a|\theta^Q)$, evaluates the quality of actions.

3) *Soft update mechanism*: Target network parameters slowly track the online network parameters.

DDPG's update rules embody the adaptation of Bellman optimality ideas to continuous spaces.

1) *Critic update* (based on the continuous form of the Bellman optimality equation):

$$y=r+\gamma Q'(s',\mu'(s'|\theta^{\mu'})|\theta^{Q'})$$

Note that the target policy network μ' replaces the max operation here, as the deterministic policy directly gives the optimal action selection.

3) *Actor update(*improving the policy along the gradient of the Q-function):

$$\nabla_{\theta^\mu} J \approx \boldsymbol{E}[\nabla_a Q(s, a|\theta^Q) \nabla_{\theta^\mu} \mu(s|\theta^\mu)]$$

The actor network is trained to output actions that maximize the Q-value, which is essentially the "policy improvement" step in continuous action spaces.

The Pseudocode of DDPG is listed as

 doi:10.20944/preprints202601.0420.v2

Initialize actor network $\mu(s|\theta^{\mu})$ and critic network $Q(s,a|\theta^{Q})$

Initialize corresponding target networks μ′ and Q′

Initialize experience replay buffer R

for each time step do

Select action a = $\mu(s|\theta^{\mu})$ + exploration noise N_0

Execute a, observe reward r and new state s′

Store transition (s,a,r,s′) in R

Randomly sample a minibatch from R

Update critic (based on modified Bellman optimality equation)

$$y_i=r_i+\gamma Q'(s_i',\mu'(s_i'|\theta^{\mu'})|\theta^{Q'})$$

Minimize $L=\frac{1}{N}\Sigma_i(y_i-Q(s_i,a_i|\theta^{Q}))^2$

Update actor (approximate policy improvement)

$$\nabla_{\theta^{\mu}}J \approx \frac{1}{N}\Sigma_i \nabla_a Q(s,a|\theta^{Q})|_{s=s_i,a=\mu(s_i)}\nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s=s_i}$$

Soft update target networks

$$\theta^{Q'}\leftarrow\tau\theta^{Q}+(1-\tau)\theta^{Q'}$$

$$\theta^{\mu'}\leftarrow\tau\theta^{\mu}+(1-\tau)\theta^{\mu'}$$

end for

Remarks: DDPG can be seen as an approximate method for solving the Bellman optimality equation in continuous spaces. The critic learns the optimal Q-function, the actor learns the corresponding optimal policy, and they are optimized alternately, similar to policy iteration in dynamic programming.

The advantages and challenges of DDPG is outlined as

First, we give the advantages.

1）Efficiently handles continuous action spaces, avoiding the curse of dimensionality.

2）Relatively high sample efficiency.

3）Can learn deterministic policies, suitable for tasks requiring precise control.

However, there are challenges.

1）Sensitive to hyperparameters.

2）Exploration efficiency issues (requires carefully designed exploration noise).

3）May converge to local optima.

7. Frontier Developments and a Unified Theoretical Perspective

In this section, we give the logistical evolution and future directions.

Following DDPG, researchers proposed various improved algorithms, all revolving around the core of solving the Bellman optimality equation more stably. Some of them are

1) TD3 (twin delayed deep deterministic policy gradient): Addresses overestimation via double Q-learning, a correction for errors in the max operator of the Bellman optimality equation.

2) SAC(soft actor-critic): Introduces a maximum entropy framework, modifying the Bellman equation to encourage exploration [8].

 doi:10.20944/preprints202601.0420.v2

3) PPO(proximal policy optimization): Stabilizes training by constraining policy updates, ensuring monotonic policy improvement [9].

From MDP to the DDPG, the development of reinforcement learning presents a clear logical thread. Table 1 shows this complete framework of theoretical evolution and algorithmic development.

Table 1. Correspondence between Theoretical Foundations and Algorithm Evolution.

Theoretical Stage	Core Concept	Mathematical Expression	Represent at-ive Algorithms	Applicable Scenarios
Problem Definition	Markov Decision Process	(S,A,P,R,γ)	-	Decision problem formulation
Policy Evaluation	Bellman Equation	$V^{\pi}(s) = \Sigma_a \pi(a\|s)\Sigma_{s'} P(s'\|s,a)[R + \gamma V^{\pi}(s')]$	Policy evaluation algorithms	Analyzing a given policy
Optimal Condition	Bellman Optimality Equation	$V(s) = max_a \Sigma_{s'} P(s'\|s,a)[R + \gamma V(s')]$	Theoretical benchmark	Defining optimal policy standard
Model-Based Solution	Dynamic Programming	$V_{k+1}(s) = max_a \Sigma_{s'} P(s'\|s,a)[R + \gamma V_k(s')]$	Value iteration, Policy iteration	Environments with known model
Model-Free Learning	Temporal Difference Learning	$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)]$	SARSA、Q-learning	Environments with unknown model
High-Dimensional Extension	Function Approximation	$Q(s,a;\theta) \approx Q^{\pi}(s,a)$	Linear function approximation	Large state spaces
Deep Integration	Deep Q-Learning	$L(\theta) = \boldsymbol{E}[(r + \gamma max_{a'} Q(s',a';\theta^{-}) - Q(s,a;\theta))^2]$	DQN and its variants	High-dimensional inputs like images
Policy Optimization	Policy Gradient	$\nabla_\theta J(\theta) = \boldsymbol{E}[\nabla_\theta \log \pi(a\|s;\theta) Q^{\pi}(s,a$	REINFORCE	Direct policy optimization
Integrated Methods	Actor-Critic	actor: $\nabla_\theta J \approx \boldsymbol{E}[\nabla_\theta \mu(s) \nabla_a Q(s,a)]$	A3C、DDPG	Continuous action spaces
Continuous	Determinis	$a = \mu(s\|\theta^{\mu})$	DDPG、	Robot control,

 doi:10.20944/preprints202601.0420.v2

Theoretical Stage	Core Concept	Mathematical Expression	Representat-ive Algorithms	Applicable Scenarios
Control	tic Policy Gradient		TD3、SAC	etc.

We conclude the key turning points in the logical evolution.

1) From evaluation to optimization, i.e., Bellman equation to Bellman optimality equation.

2) From known to unknown, i.e., dynamic programming (model known) to model-free learning.

3) From discrete to continuous, i.e., tabular methods to function approximation to deep neural networks.

4) From value function to policy, i.e., value-based methods to policy gradient to actor-critic.

5) From theory to practice, i.e., theoretical equations to practical algorithms to engineering implementation.

Potential future directions of RL are listed as follows.

1)*Theoretical deepening:* Deeper understanding of the generalization capabilities and convergence properties of deep reinforcement learning.

2)*Algorithmic efficiency:* Improving sample efficiency, reducing the need for environment interaction.

3) *Safety and reliability:* Ensuring safety during exploration.

4) *Multi-task learning:* Enabling transfer of knowledge and skills.

8. Conclusion: The Science of Intelligent Decision-Making Within a Unified Framework

From the mathematical formalization of MDPs, to the establishment of the Bellman equation, and further to the theoretical breakthrough of the Bellman optimality equation, RL has built a complete theoretical system for decision-making. Q-learning transforms the optimality equation into a practical algorithm, deep neural networks solve the problem of high-dimensional representation, and algorithms like DDPG extend it to the domain of continuous control.

This development history reveals a profound scientific methodology. That is from clear problem definition (i.e., MDP), to establishing optimality conditions (i.e., Bellman optimality equation), to developing practical solution algorithms (i.e., from dynamic programming to deep reinforcement learning). Each layer of progress is built upon a solid theoretical foundation while simultaneously pushing the theory further.

Despite fast development, the core of RL revolves around the fundamental problem of how to make optimal decisions in uncertain environments. The Bellman optimality equation continues to guide the development direction of new algorithms. With computing power improvement and theoretical refinement, intelligent systems based on these principles play key roles in more complex decision-making tasks.

The Bellman equation not only became the cornerstone of RL but also opened a new era of enabling machines to learn autonomous decision-making. The path of exploration from theoretical equations to practical algorithms continues, and each step brings us closer to truly intelligent machine decision systems.

 doi:10.20944/preprints202601.0420.v2

References

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
2. Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.
3. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3-4), 279–292.
4. Mnih, V., Kavukcuoglu, K., Silver, et al., (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.
5. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*.
6. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. *International Conference on Machine Learning (ICML)*.
7. Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning (ICML)*.
8. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*.
9. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.