Lecture 3: Policy Optimization in Reinforcement Learning

Dr. Yaodong Yang

Institute for AI, Peking University

08/2023

Reinforcement Learning in Continuous Action Spaces

Introduction

Reinforcement Learning (RL) methods are typically designed for discrete action spaces. However, many real-world problems, such as robotic control or autonomous driving, involve continuous action spaces.

Challenges

Applying RL to continuous action spaces presents several challenges, including:

- Infinite possible actions at each state,
- The need for function approximation, and
- The exploration-exploitation trade-off.

Discretization Approach

One way to handle continuous action spaces is to discretize them into a finite set of actions. However, this approach has limitations:

- Curse of dimensionality: The number of actions grows exponentially with the number of dimensions.
- Loss of granularity: Discretization may lead to inaccurate representations of the action space.

Example

For a 2D robotic arm with continuous joint angles, discretization would create a grid of possible actions, resulting in a limited set of feasible movements.

Policy Parameterization

Policy Parameterization

Another approach is to parameterize the policy directly. The policy $\pi(a|s;\theta)$ is a parametric function of the state s and parameters θ.

Advantages

Parameterization allows for:

- Flexibility: The policy can represent a wide range of continuous functions.
- Smoothness: The policy can produce smooth actions in the continuous space.
- Policy Gradients: Policy gradient methods can optimize the parameters directly.

Actor-Critic Method

Actor-Critic Approach

The Actor-Critic method [GBLB12] combines an actor (policy) network and a critic (value) network.

Advantages

The Actor-Critic approach offers:

- Policy Optimization: The actor network optimizes the policy parameters.
- Value Estimation: The critic network estimates the value function to guide the actor's updates.
- Stability: Actor-Critic can improve sample efficiency and reduce variance in the learning process.

Introduction

- Generalized Advantage Estimation (GAE) [SML+15] is a technique used in reinforcement learning to estimate the advantages of actions in a state.
- It is an extension of the advantage estimation concept, aiming to improve the stability and sample efficiency of policy gradient methods.

Definition

- The advantage function measures how much better (or worse) an action is compared to the average action in a given state.
- It plays a crucial role in policy gradient methods by guiding the policy updates towards better actions.
- One common way to estimate advantages is using the formula:

$$\text{Advantage}(s, a) = \left(\sum_{t=0}^{T} \gamma^t r_t \right) - V(s)$$

where s is the state, a is the action, γ is the discount factor, r_t is the reward at time step t, and $V(s)$ is the estimated state-value function.

Definition

- Accurate advantage estimation can be challenging due to the high variance in rewards and the difficulty of approximating the state-value function.
- Traditional advantage estimation can lead to unstable policy updates and slow convergence in complex environments.
- GAE addresses these challenges by introducing a **discount factor** λ to balance the trade-off between bias and variance in advantage estimation.

Formulation

- GAE computes a **weighted sum** of advantages over multiple time steps, giving more importance to recent rewards.
- It is defined as:

$$\text{GAE}(s, a) = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}$$

where $\delta_{t+l} = r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})$ is the temporal difference error at time step $t + l$.

- The hyperparameter λ controls the trade-off between bias and variance. For $\lambda = 0$, GAE reduces to regular advantage estimation, while higher values introduce more bias but lower variance.

Relationship of GAE with MC and TD Methods

Monte Carlo (MC)

MC methods estimate the value function and advantages by sampling full trajectories and computing the returns from the sampled trajectories.

Temporal Difference (TD)

TD methods estimate the value function and advantages by bootstrapping from the next time step's value function estimate.

GAE

GAE combines features of both MC and TD methods, using a discount factor λ to control the trade-off between bias and variance in advantage estimation.

Application of Deep Reinforcement Learning

- Deep Reinforcement Learning (DRL) has found applications in various fields due to its ability to learn from raw sensory input.
- It has been successfully applied to solve complex tasks and achieve impressive performance in different domains.

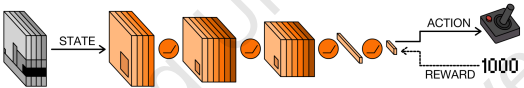

Fig. 1. The network takes the state—a stack of greyscale frames from the video game—and processes it with convolutional and fully connected layers. At the final layer, the network outputs a discrete action, which corresponds to one of the possible control inputs for the game. [ADBB17]

Example: Playing Video Games

Fig. 2. DreamerV3 [HPBL23] is the first algorithm that collects diamonds in Minecraft without human demonstrations or manually-crafted curricula, which poses a big exploration challenge.

- **State**: Raw pixel images.
- **Action**: Control game moves, actions, and strategies.
- **Reward**: Points earned for game objectives achieved, penalties for losing or making mistakes.

Example: Robotic Control

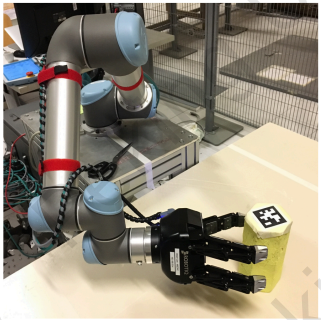

Fig. 3. The Pick and Place experiment. The robot has to pick up the yellow cylinder and bring it in a random place. [FTCG21]

- **State**: Joint positions, velocities, and sensor data.
- **Action**: Control robotic arm movements and manipulate objects.
- **Reward**: Positive for successful tasks, negative for collisions or errors.

Example: Autonomous Driving

Fig. 4. MetaDrive is a new driving simulation platform which can generate an infinite number of diverse driving scenarios from both the procedural generation and the real data importing. [LPF+22]

- **State**: Sensor inputs like camera images, LIDAR, and GPS data.
- **Action**: Control steering, acceleration, and braking.
- **Reward**: Positive for safe driving, reaching the destination, negative for collisions or rule violations.

Introduction to Policy Gradient Methods

Overview

Policy Gradient [SMSM99] Methods are a class of reinforcement learning algorithms that directly optimize the policy's parameters to find the best policy for the task at hand.

Key Idea

The main idea behind policy gradient methods is to use gradient ascent to iteratively improve the policy by maximizing the expected return.

Policy Gradient Theorem

Policy Gradient Theorem

The policy gradient theorem expresses the gradient of the expected return with respect to the policy's parameters θ as:

$$\nabla_\theta J(\theta) \propto \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla log p_\theta(a_t^n | s_t^n),$$

where:

- $J(\theta)$ is the objective function representing the expected return,
- N is the number of sampled trajectories and T_n is the max length of each trajectory,
- R is the total reward of a trajectory τ.
- $\nabla log p_\theta(a_t^n | s_t^n)$ is the policy gradient with respect to θ

Policy Gradient Theorem (Contd.)

Derivation Step 1

Starting with the expression for the objective function $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} = \sum_\tau R(\tau) p_\theta(\tau),$$

we take the gradient with respect to θ to find $\nabla_\theta J(\theta)$.

Derivation Step 2

Using the identity $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau) \nabla_\theta log p_\theta(\tau)$, we rewrite the gradient as:

$$\nabla_\theta J(\theta) = \sum_\tau R(\tau) p_\theta(\tau) \nabla_\theta log p_\theta(\tau).$$

Policy Gradient Theorem (Contd.)

Derivation Step 3

Using Monte Carlo method to sample many trajectories, we have:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla_\theta log p_\theta(\tau^n).$$

Final Result

The final result of the derivation is the policy gradient theorem:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla log p_\theta(a_t^n | s_t^n),$$

Actor-Critic Update

- Actor-Critic is a popular policy gradient algorithm that combines the advantages of both policy-based and value-based methods.
- It maintains an actor (policy) and a critic (value function) to estimate the state-value function $V(s)$.
- The policy is updated using the advantage function $A(s, a) = Q(s, a) - V(s)$, where $Q(s, a)$ is the action-value function.
- The policy update formula is as follows:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta \log \pi_\theta(a|s) \cdot A(s, a)$$

where α is the learning rate.

Related Algorithms

- There are several variations of policy gradient algorithms, including:
  - REINFORCE (Monte-Carlo Policy Gradient)
  - Actor-Critic
  - Proximal Policy Optimization (PPO)
  - Trust Region Policy Optimization (TRPO)
  - etc.
- Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the specific problem and requirements.
- Many modern RL libraries provide implementations of these algorithms, making it easier to experiment and apply them to various tasks.

Example: Cartpole Environment

- Let's consider the classic Cartpole environment, where the agent needs to balance a pole on a moving cart.
- State: $s = (x, \dot{x}, \theta, \dot{\theta})$, representing cart position, cart velocity, pole angle, and pole angular velocity, respectively.
- Action: $a \in \{-1, 1\}$, indicating left or rightward force applied to the cart.
- Reward: $r = 1$ for each time step the pole remains upright, else $r = 0$.
- Objective: Maximize the expected cumulative reward over episodes.

Comparison of Policy Gradient Algorithms

Algorithm	Advantages	Disadvantages	Example
REINFORCE	Simple	High variance	Cartpole
PPO	Efficient	Hyperparameter sensitivity	Atari Games
TRPO	Guaranteed improvement	Computationally expensive	Robotics
DDPG	Continuous action	Overestimation bias	Robotic
TD3	Stable	Hyperparameter tuning	Robotic
SAC	Exploration	Entropy tempera-	Robotic

Motivation for Importance Sampling

- In reinforcement learning, estimating expected values under different policies is challenging.
- In Policy Gradient methods, the agent learns from diverse trajectories gathered by a behavior policy.
- Importance Sampling allows us to reuse this data and estimate the value under a different policy (target policy).

Importance Sampling in Policy Gradient

- Policy Gradient methods use the following update rule for the policy parameter θ:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

  where α is the learning rate and $J(\pi_\theta)$ is the objective function (e.g., expected return).
- However, directly using samples collected from a behavior policy in the gradient estimation can lead to high variance and slow convergence.
- Importance Sampling corrects this by reweighting the samples using the Importance Sampling ratio.

Importance Sampling Ratio

- The Importance Sampling Ratio (ISR) is defined as:

$$\text{ISR}(\theta, \theta_{\text{old}}, \tau) = \frac{\pi_\theta(\tau)}{\pi_{\theta_{\text{old}}}(\tau)}$$

where τ represents a trajectory sampled from the behavior policy, $\pi_\theta(\tau)$ is the probability of obtaining trajectory τ under the target policy π_θ, and $\pi_{\theta_{\text{old}}}(\tau)$ is the probability under the old policy $\pi_{\theta_{\text{old}}}$.

- The objective function with Importance Sampling is:

$$\hat{J}(\pi_\theta) = \frac{1}{N} \sum_{i=1}^{N} \text{ISR}(\theta, \theta_{\text{old}}, \tau_i) \cdot R(\tau_i)$$

where N is the number of trajectories, and $R(\tau_i)$ is the return obtained from trajectory τ_i.

Importance Sampling and KL Divergence

- In reinforcement learning, Importance Sampling allows us to reuse data collected from a behavior policy to estimate values under a different target policy.
- The KL Divergence term ensures that policy updates do not deviate too far from the previous policy.
- When the KL Divergence becomes too large, it can lead to several issues:
  - **Destructive Updates:** Drastic policy changes can destabilize learning and lead to divergent behavior.
  - **Sample Inefficiency:** Large KL Divergence can cause the new policy to deviate significantly from the old policy, resulting in inefficient use of previously collected data.
  - **Stagnation:** A large KL Divergence can hinder exploration, causing the agent to get stuck in local optima.

Trust Region Policy Optimization (TRPO)

- TRPO is a powerful extension of Policy Gradient with additional emphasis on the trust region constraint.
- TRPO directly limits the KL Divergence between the new and old policies.
- TRPO's objective function is as follows:

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_{\boldsymbol{\theta}}} J(\pi) \quad \text{s.t.} \quad D_{KL}(\pi, \pi_k) \leq \delta$$

where $\text{KL}_{\text{target}}$ is a pre-defined KL Divergence threshold.

Trust Region Policy Optimization (TRPO)

- TRPO ensures that the policy update stays within the trust region defined by $\mathrm{KL}_{\text{target}}$.
- To achieve this, TRPO solves an optimization problem subject to a KL Divergence constraint:

$$\theta_{\text{new}} = \underset{\theta}{\text{argmax}} \left[\hat{J}(\pi_\theta) - \beta \cdot \mathrm{KL}(\pi_\theta, \pi_{\theta_{\text{old}}}) \right]$$

  subject to $\mathrm{KL}(\pi_\theta, \pi_{\theta_{\text{old}}}) \leq \mathrm{KL}_{\text{target}}$
- TRPO usually requires a complex optimization process, such as conjugate gradient, to find the policy update within the trust region.

Trust Region Policy Optimization (TRPO)

- TRPO is computationally more expensive than PPO because of the constrained optimization.
- However, TRPO typically provides more stable and conservative policy updates.
- The trust region constraint in TRPO prevents large policy changes and helps avoid policy collapses during training.
- TRPO has been widely used in various domains and has shown promising results in complex reinforcement learning tasks.

Optimization Objective

The objective function of Proximal Policy Optimization (PPO) [SWD+17] is to find a policy that maximizes the expected reward while ensuring that policy updates are not too far from the original policy.

$$\pi_{k+1} = \arg\max_{\pi \in \Pi_{\boldsymbol{\theta}}} J(\pi) \quad \text{s.t.} \quad D_{KL}(\pi, \pi_k) \leq \delta$$

where:

- $\Pi_{\boldsymbol{\theta}} \subseteq \Pi$ denotes the set of parameterized policies with parameters $\boldsymbol{\theta}$.
- $J(\pi)$ is the expected reward, which is the objective function to maximize.
- D_{KL} is the KL distance, and δ is a hyperparameter that controls the trust region constraint.

PPO-Penalty

PPO-Penalty modifies the objective function of Proximal Policy Optimization (PPO) to handle constrained optimization problems:

$$\max_{\theta} \mathbb{E}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} \hat{A}_\pi(s,a) - \beta D_{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s), \pi_\theta(\cdot \mid s)]\right]$$

- $\hat{A}_\pi(s,a)$ is the Generalized Advantage Estimation (GAE) version of the advantage.
- We compute $d = \hat{\mathbb{E}}\left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s), \pi_\theta(\cdot \mid s)]\right]$ to measure the divergence between the old and updated policies.
- If $d < d_{\text{targ}}/1.5$, we decrease β by dividing it by 2.
- If $d > d_{\text{targ}} \times 1.5$, we increase β by multiplying it by 2.
- The updated β is then used for the next policy update.

PPO-Clip

Let $r(\theta)$ denote the probability ratio between the new and old policies:

$$r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)}$$

The PPO-Clip surrogate objective function is given by:

$$J(\pi) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}_\pi(s,a), \text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_\pi(s,a)\right)\right]$$

where:

- $\hat{A}_\pi(s,a)$ is the Generalized Advantage Estimation (GAE) at state s and action a.
- ε is a hyperparameter that determines the clipping range.

Entropy Coefficient

- The entropy coefficient α is a hyperparameter that scales the entropy term in the objective function.
- The objective function with the entropy coefficient is defined as:

$$J(\pi) = \mathbb{E}\left[\sum_a \pi(a \mid s)\hat{A}_\pi(s,a) + \alpha \sum_a \pi(a \mid s) \log \pi(a \mid s)\right]$$

where $\hat{A}_\pi(s,a)$ is the GAE advantage function, and $\pi(a \mid s)$ is the probability of taking action a in state s according to the policy.

Effect of Entropy Coefficient

- A higher entropy coefficient encourages more exploration by increasing the importance of the entropy term in the objective function.
- This promotes a more stochastic policy that explores different actions and states to discover better strategies.
- A lower entropy coefficient promotes exploitation by making the policy more deterministic, focusing on the currently learned optimal actions.

Introduction

- DDPG [LHP+15] is an actor-critic algorithm for continuous action spaces in reinforcement learning.
- It combines the strengths of deep neural networks with policy gradients to achieve high-performance policy optimization.
- **Actor**: Approximates the optimal policy deterministically.
- **Critic**: Evaluates the actor's action using Q-values.

Actor	**Critic**
θ_{actor}	θ_{critic}
Policy $\pi_{\text{actor}}(s;\theta_{\text{actor}})$	Q-Function $Q_{\text{critic}}(s,a;\theta_{\text{critic}})$
Deterministic Policy	Q-value Approximation

Overview

Actor Update: Maximize the Q-value estimated by the critic network.

$$\theta_{\text{actor}} \leftarrow \arg\max_{\theta_{\text{actor}}} Q_{\text{critic}}(s, \pi_{\text{actor}}(s; \theta_{\text{actor}}); \theta_{\text{critic}})$$

Critic Update: Minimize the Mean Squared Bellman Error.

$$\theta_{\text{critic}} \leftarrow \arg\min_{\theta_{\text{critic}}} \mathbb{E}\left[\delta^2\right]$$

$$\delta = Q_{\text{critic}}(s, a; \theta_{\text{critic}}) - Q_{\text{target}}$$

$$Q_{\text{target}} = r + \gamma Q_{\text{critic}}(s', \pi_{\text{actor}}(s'; \theta_{\text{actor}}); \theta_{\text{critic}})$$

Twin Critic Architecture

- TD3 [FHM18] is an off-policy algorithm for continuous action spaces in reinforcement learning.
- It is an extension of DDPG and addresses the overestimation bias issue of Q-value estimates.
- TD3 utilizes twin critics and target policy smoothing to improve stability and convergence.
- **Twin Critic**: Utilizes two Q-value approximators to mitigate overestimation bias.

Twin Critic	**Twin Critic**
$Q_1(s, a; \theta_{Q1})$	$Q_2(s, a; \theta_{Q2})$
Q-value Approximation	Q-value Approximation

Target Policy Smoothing

Target Policy Smoothing: Adds noise to the target policy to prevent policy overfitting.

$$a' = \pi_{\text{target}}(s') + \text{clip}(\mathcal{N}(0, \sigma), -c, c)$$

- $\pi_{\text{target}}(s')$: Target policy action at the next state.
- $\text{clip}(\cdot)$: Clips the noise within a predefined range $[-c, c]$.
- σ: Noise standard deviation.

Overview

Actor-Critic Updates:

Actor Update: Maximize the Q-value estimated by the critic network.

$$\theta_{\text{actor}} \leftarrow \arg\max_{\theta_{\text{actor}}} Q_{\text{critic}}(s, \pi_{\text{actor}}(s; \theta_{\text{actor}}); \theta_{\text{critic}})$$

Critic Update: Minimize the Mean Squared Bellman Error.

$$\theta_{\text{critic}} \leftarrow \arg\min_{\theta_{\text{critic}}} \mathbb{E}\left[\delta^2\right]$$

$$\delta = Q_{\text{critic}}(s, a; \theta_{\text{critic}}) - Q_{\text{target}}$$

$$Q_{\text{target}} = r + \gamma \min_{i=1,2} Q^i_{\text{critic}}(s', \pi_{\text{actor}}(s'; \theta_{\text{actor}}); \theta^i_{\text{critic}})$$

Introduction

- SAC [HZAL18] is an off-policy algorithm for both discrete and continuous action spaces in reinforcement learning.
- It combines actor-critic architecture with the principle of entropy maximization to improve exploration and robustness.
- SAC utilizes the soft Bellman backup to maintain a stochastic policy.

Entropy Regularization

Entropy Regularization: Incorporates entropy term to encourage exploration.

$$\text{Objective:} \quad \max_{\theta_{\text{actor}}} \mathbb{E}_{a \sim \pi_{\text{actor}}} \left[Q_{\text{critic}}(s, a) - \alpha \log \pi_{\text{actor}}(a|s) \right]$$

- $\pi_{\text{actor}}(a|s)$: Policy distribution over actions given state s.
- $Q_{\text{critic}}(s, a)$: Q-value estimate for state-action pair (s, a).
- α: Entropy temperature, controls the trade-off between exploration and exploitation.

Soft Bellman Backup

Soft Bellman Backup: Incorporates the maximum entropy principle into the Bellman backup.

$$Q_{\text{target}} = r + \gamma(\min_{i=1,2} Q^i_{\text{critic}}(s', \pi_{\text{actor}}(s'; \theta_{\text{actor}}); \theta^i_{\text{critic}}) - \alpha \log \pi_{\text{actor}}(a'|s'))$$

- r: Immediate reward for taking action a in state s.
- γ: Discount factor.
- $p(\cdot|s, a)$: Next state distribution given current state-action pair (s, a).

Comparison of DDPG, TD3, and SAC

Algorithm	DDPG	TD3	SAC
Update Rule	Actor-Critic	Actor-Two-Criic	Soft-Actor-Critic
Stability	Unstable, overestimation bias	More stable with twin critics	Stable with entropy regularization
Sample Efficiency	Moderate, more samples needed	Improved with target policy smoothing	Efficient, high sample efficiency
Exploration	Limited, deterministic policy	Better with target policy smoothing	Effective with stochastic policy
Characteristics	Off-policy, model-free, continuous action space		

Table. 2. This table compares DDPG, TD3, and SAC with columns for update rule, stability, sample efficiency, exploration, and characteristics.

Summary

- **DDPG**: DDPG suffers from instability due to overestimation bias in Q-value updates. It requires more samples to achieve good performance, and its deterministic policy limits exploration capabilities.
- **TD3**: TD3 addresses the overestimation bias by using twin critics and achieves better stability. It also improves sample efficiency through target policy smoothing, and its exploration is enhanced compared to DDPG.
- **SAC**: SAC maintains stability with entropy regularization, leading to a more robust training process. It achieves high sample efficiency, making it more data-efficient. The stochastic policy and entropy term in the objective function allow effective exploration, improving performance in complex environments.

Summary and Outlook

In this lecture, we covered traditional reinforcement learning algorithms:

- Prior knowledge.
- On-policy algorithms.
- Off-policy algorithms.

In the next lecture, we will introduce:

- Fundamentals of Human Feedback

Thanks!

References I

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath.
A brief survey of deep reinforcement learning.
arXiv preprint arXiv:1708.05866, 2017.

[FHM18] Scott Fujimoto, Herke Hoof, and David Meger.
Addressing function approximation error in actor-critic methods.
In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[FTCG21] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni.
Robotic arm control and task training through deep reinforcement learning.
In *International Conference on Intelligent Autonomous Systems*, pages 532–550. Springer, 2021.

[GBLB12] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska.
A survey of actor-critic reinforcement learning: Standard and natural policy gradients.
IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(6):1291–1307, 2012.

[HPBL23] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap.
Mastering diverse domains through world models.
arXiv preprint arXiv:2301.04104, 2023.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine.
Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

References II

[LHP+15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra.
Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971, 2015.

[LPF+22] Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou.
Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning.
IEEE transactions on pattern analysis and machine intelligence, 45(3):3461–3475, 2022.

[SML+15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel.
High-dimensional continuous control using generalized advantage estimation.
arXiv preprint arXiv:1506.02438, 2015.

[SMSM99] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour.
Policy gradient methods for reinforcement learning with function approximation.
Advances in neural information processing systems, 12, 1999.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347, 2017.