CS-566 Deep Reinforcement Learning

Policy-Based Learning

Nazar Khan
Department of Computer Science
University of the Punjab

Motivation: Continuous Action Spaces

- Deep RL has major successes in **continuous action spaces:**
  - Robotics (e.g., robot arms)
  - Self-driving cars
  - Real-time strategy games
- These environments require actions over **continuous ranges**, not discrete sets.

Examples of Continuous Action Spaces

- **Robotics control:** Joint angles, torque, or velocity.
- **Self-driving cars:** Steering, acceleration, braking.
- **Drone flight:** Continuous pitch, roll, yaw, thrust.
- **Industrial control:** Adjusting temperature or flow rate.
- **Finance:** Portfolio weights as continuous allocations.
- **Healthcare:** Continuous dosage control (e.g., insulin).
- **Gaming and simulation:** Throttle, aim, camera rotation.
- **Locomotion:** Walking, running, or balancing

Limitation of Value-Based Methods

- Value-based RL (e.g., Q-learning, DQN):
  1. Learns $Q(s, a)$ for all actions.
  2. Selects best action via $\arg\max_a Q(s, a)$.
- Works well for **discrete actions**.
- In continuous spaces:
  - $\arg\max$ is hard to compute.
  - Learning becomes unstable.
- Need a method that handles continuous actions directly.

Policy-Based Methods: The Direct Approach

- Skip value estimation – learn the **policy directly**.
- Policy-based methods represent:

$$\pi_\theta(a|s) = P(a|s;\theta)$$

- Model will directly output action probability.
- Improve parameters θ using **gradient ascent**.
- Learn by playing episodes and improving the policy each time.

Why Policy-Based Methods?

- **Advantages:**
  - Work naturally with **continuous actions**.
  - Produce **stochastic** policies (smooth exploration).
  - Applicable to more domains than value-based methods.
  - Integrate well with gradient-based deep learning.
- Some of the most popular deep RL methods are policy-based.
- Form the foundation for modern algorithms:
  - REINFORCE
  - Actor-Critic
  - PPO, A3C, DDPG

Jumping Robots
The Challenge of Locomotion

- One of the most intricate problems in robotics: **learning to walk, run, and jump.**
- Simulated robots have learned to jump over obstacle courses using deep reinforcement learning.
- Video example: `https://www.youtube.com/watch?v=hx_bgoTF7bs`[1].

Human Analogy

Learning to walk takes human infants months, even though the body is optimized for it. Locomotion combines *perception, balance, and continuous control*. Robots face a much harder version of this challenge.

[1] **heess2017emergence**.

Jumping Robots
Why Locomotion Is Hard

- Locomotion is a **sequential decision problem**.
- Each leg has multiple **joints** that must:
  - Actuate in the right order.
  - Apply the right force and duration.
  - Rotate to the right angle.
- These control variables – **angles, forces, durations** – are all **continuous**.
- Algorithms must discover the **optimal continuous policy**.

Relevance

Policy-based deep reinforcement learning is widely used to train locomotion agents in simulation and real-world robotics.

Continuous Policies

From Discrete to Continuous Actions

- Earlier problems: small, **discrete** action spaces (e.g., Grid Worlds, Mazes, Atari: $\{N, E, S, W\}$ or joystick moves).
- Even complex games like Chess have discrete actions.
- In many real-world tasks, actions are instead **continuous**.

Shift in Focus

We now move from **large state spaces** to **continuous action spaces**.

Continuous Policies

Examples of Continuous Actions

- **Self-driving cars:** steering angle, duration, and angular velocity must vary smoothly.
- **Throttle control:** continuous adjustment of acceleration and braking.
- **Robotic joints:** can rotate by 1°, 2°, 90°, or any value in between.

Challenge

An action can take any value in a continuous range (e.g., $[0, 2\pi]$ or $\mathbb{R}^+$), making the space **infinitely large**.

Continuous Policies

Why Policy-Based Methods?

- Searching all combinations of continuous actions is infeasible.
- Discretization can approximate solutions but introduces **quantization errors**.
- In continuous domains, $\arg\max$ can no longer identify "the" best action.
- **Value-based methods fail** when actions are not discrete.

Solution

Policy-based methods learn continuous or stochastic policies *directly*, without needing a value function or $\arg\max$.

Stochastic Policies
Motivation

- Robots operate in **stochastic environments** – sensors and actuators introduce uncertainty.
- Example: a robot misjudges a door's distance or balance, leading to failure.
- Small noise in Q-values can cause **large policy shifts** in value-based methods.
  - **Example:**
  $$Q(s, a_1) = 1.00,\ Q(s, a_2) = 0.99 \Rightarrow a_1$$
  After small noise:
  $$Q(s, a_1) = 0.99,\ Q(s, a_2) = 1.00 \Rightarrow a_2$$
    - Tiny Q perturbation $\Rightarrow$ abrupt action change.
    - Leads to unstable/oscillating policies.
    - Worse in spaces with continuous or similarly beneficial actions.
- Convergence requires **slow learning rates** to smooth randomness.

Stochastic Policies
Advantages of Stochastic Policies

- Stochastic policies output a **distribution over actions** $\pi_\theta(a|s)$.
  - Instead of *choosing* a single best action, the agent *samples* actions according to $\pi_\theta(a|s)$.
- Naturally handle randomness in environment and action execution.
- Enable **built-in exploration** – no need for ϵ-greedy or softmax sampling.
  - Sampling *is* exploration.
- Improve stability and prevent drastic policy oscillations.

Stochastic Policies
Limitations and Extensions

- Purely episodic policy-based methods can have **high variance**.
  - Return G_t depends on **entire trajectory**

$$G_t = r_t + \gamma r_{t+1} + \dots$$

  - Small randomness early in the episode $\Rightarrow$ large change in final return
  - Each episode produces a different G_t $\Rightarrow$ noisy gradient estimate
- May converge to **local optima** rather than global ones.
- Often **slower to converge** than value-based methods.

Solution: Actor–Critic Methods

Newer algorithms combine value and policy learning for stability:

- A3C (Asynchronous Advantage Actor–Critic)
- TRPO (Trust Region Policy Optimization)
- PPO (Proximal Policy Optimization)

Policy-Based RL Environments
Gym and MuJoCo

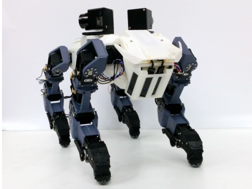
- Real-world robotics experiments are **expensive and slow**.
- Reinforcement learning often relies on **simulated physics environments**.
- Simulators approximate robot dynamics, forces, and interactions with the environment.
- Two popular simulators:
  - **MuJoCo** – Multi-Joint dynamics with Contact[2]
  - **PyBullet** – Open-source physics engine[3]
- Integrated with **OpenAI Gym** for standardized experimentation.

[2] **todorov2012mujoco**.
[3] **coumans2019**.

Robotics Environments

Complexity Beyond Classic RL Tasks

- Unlike Grid World, Mountain Car, or CartPole, robotic tasks have:
  - Multiple **joints and degrees of freedom**
  - **Continuous action spaces** (angles, forces, durations)
  - **Visuo-motor coordination** (e.g., grasping)
  - **Locomotion learning** (walking, running, jumping)
- Environments are partly unpredictable – agents must react to **disturbances**.

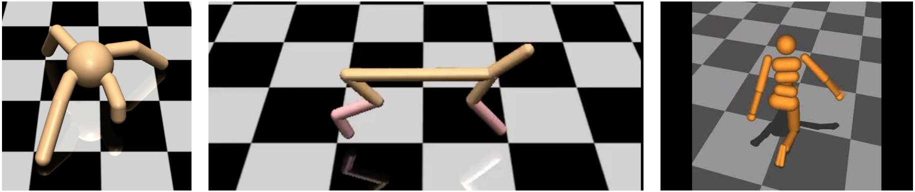


Physics Simulation Models
Why Simulate?

- Model-free RL requires **millions of samples**.
- This makes it infeasible on real robots.
- Physics engines simulate:
  - Forces, acceleration, velocity
  - Mass, elasticity, and friction
  - Grasping, locomotion, and gait
- Goals:
  - **Accuracy** – realistic physical dynamics
  - **Speed** – fast enough for RL training

MuJoCo Environments
Examples

- MuJoCo is deterministic but typically uses **randomized initial states**.
- Resulting environments are **non-deterministic overall**.
- Common benchmark tasks in Gym/MuJoCo:
  - **Ant** – 4-legged locomotion
  - **Half-Cheetah** – 2D running
  - **Humanoid** – full-body walking



Gym MuJoCo: Ant, Half-Cheetah, and Humanoid

Policy-Based Algorithm: REINFORCE

- Policy-based methods learn a parameterized policy π_θ that directly selects actions, without using a value function for action choice.
- Unlike value-based methods (which use $\arg\max$), these can naturally handle **continuous actions**.
- Policies are parameterized by θ (e.g., neural network weights) mapping states S to action probabilities A.

Intuitive Analogy: The Supermarket

The Supermarket Example

- **Value-based:** estimate how close each direction is to the supermarket (Q-values) and follow the shortest path.
- **Policy-based:** ask a local for a full set of directions (a trajectory) and try to improve it.

Policy Optimization Framework

- Basic framework of policy-based algorithms
  1. Initialize policy parameters θ.
  2. Sample a trajectory τ from π_θ.
  3. If τ yields high reward, adjust θ toward τ; otherwise, away.
  4. Repeat until convergence.
- Recall that value function $V^\pi(s_0)$ is the expected cumulative return from initial state s_0.
- Natural to use $V^\pi(s_0)$ as performance objective $J(\theta)$.
- **Goal:** maximize performance objective $J(\theta) = V^\pi(s_0)$.
- Use gradient ascent:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta)$$

Gradient Ascent Optimization (Algorithm Sketch)

Input: $J(\theta)$, learning rate α
Randomly initialize θ
repeat
    Sample trajectory τ
    Compute gradient $\nabla_\theta J(\theta)$
    Update: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
until convergence

Neural Network Policy Representation

- $\pi_\theta(a|s)$: probability of taking action a in state s.
- Represented by a neural network with parameters θ:
  - Input: state s.
  - Output: action probabilities $\pi_\theta(a|s)$.
- Parameters θ define the mapping from states to actions.
- **Goal:** update θ so that π_θ becomes the optimal policy.
- **Intuition:** the better the action a, the more we should increase θ in that direction.

Ideal Update with Known Optimal Action

- Suppose we magically know the optimal action $a^\star$ for each state s.
- Then, we can update parameters toward the gradient of this optimal action:
$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \pi_{\theta_t}(a^\star|s)$$
- *Adjust θ so that probability of best action is maximized*.
- This pushes π_θ in the direction of the best possible action.
- However, in practice we do **not know** $a^\star$.

Using Sample Trajectories Instead

- We can use sampled trajectories to estimate which actions are good.
- Replace the unknown $a^\star$ with a sampled action a and an estimated value:

$$\theta_{t+1} = \theta_t + \alpha \hat{Q}(s,a) \nabla_\theta \pi_{\theta_t}(a|s)$$

- Adjust θ so that probability of sampled action is maximized.
- But scale the adjustment by the quality of that state-action pair.
- $\hat{Q}(s,a)$ can come from:
  - Estimated Q-function,
  - Discounted return, or
  - Advantage function.

Problem: Instability from Double Updates

- The policy $\pi_\theta(a|s)$ is itself a probability.
- In the previous update, high-value actions:
  - are pushed harder (large $\hat{Q}(s,a)$), and
  - occur more often (large $\pi_\theta(a|s)$).
- These actions are **doubly reinforced**, which may cause instability.
- **Fix:** normalize the update by dividing by $\pi_\theta(a|s)$:

$$\theta_{t+1} = \theta_t + \alpha \frac{\hat{Q}(s,a)}{\pi_\theta(a|s)} \nabla_\theta \pi_{\theta_t}(a|s)$$

which can also be written as

$$\theta_{t+1} = \theta_t + \alpha \hat{Q}(s,a) \frac{\nabla_\theta \pi_{\theta_t}(a|s)}{\pi_\theta(a|s)}$$

From Gradients to Log-Gradients

- Use the calculus identity:

$$\nabla \log f(x) = \frac{\nabla f(x)}{f(x)}$$

- Substitute into the previous update:

$$\theta_{t+1} = \theta_t + \alpha \hat{Q}(s, a) \nabla_\theta \log \pi_\theta(a|s)$$

- This is the **core REINFORCE update rule**[4].
- REINFORCE updates similar to **logarithmic cross-entropy loss**.

[4] **williams1992simple**.

Understanding the REINFORCE Update

- $\hat{Q}(s, a)$ acts as a weight – stronger reward $\Rightarrow$ larger parameter push.
- $\nabla_\theta \log \pi_\theta(a|s)$ points in the direction that increases the log-probability of good actions.

The update

$$\Delta\theta = \alpha \hat{Q}(s, a) \nabla_\theta \log \pi_\theta(a|s)$$

increases the probability of actions that yield higher returns.

REINFORCE Algorithm (Monte Carlo Policy Gradient)

Algorithm 1 REINFORCE

Initialize policy parameters θ
for each episode **do**
    Generate trajectory $(s_0, a_0, r_0, \ldots, s_T)$ using π_θ
    for $t = T$ to 0 **do**
        Compute return from step t onwards

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

        Update policy parameters

$$\theta \leftarrow \theta + \alpha \, G_t \, \nabla_\theta \log \pi_\theta(a_t|s_t)$$

    end for
end for

REINFORCE Summary

- Improves policy directly – no intermediate Q-function.
- Works for discrete, continuous, or stochastic actions.
- Known as **Monte Carlo Policy Gradient** since it uses sampled trajectories.
- $\pi_\theta(a|s)$ is a neural policy mapping states to action probabilities.
- The gradient ascent update adjusts θ to favor rewarding actions.
- Instability corrected by normalizing with $\pi_\theta(a|s)$.
- Using $\nabla \log \pi_\theta(a|s)$ yields the elegant and stable update rule

$$\theta_{t+1} = \theta_t + \alpha \hat{Q}(s, a) \nabla_\theta \log \pi_\theta(a|s)$$

Online vs Batch Updates

- Two main ways to update parameters in policy gradient methods:
  1. **Online:** update after each time step.
  2. **Batch:** update after completing the full trajectory.

Online Updates

- Parameters are updated **inside the innermost loop**.
- Each time step immediately affects the policy.
- Suitable for **parallel or streaming** environments.
- Ensures new information is used as soon as it becomes available.

Batch and Mini-Batch Updates

- **Batch:** accumulate all gradients over the trajectory, then update once.
- Reduces computational overhead of frequent updates.
- **Mini-batch:** compromise between online and batch.
- Balances:
  - Information efficiency (like online),
  - Computational efficiency (like batch).

Advantages of Policy-Based Methods

- **Deep learning compatibility:** Policy parameterization fits naturally with neural networks.
- **Stochastic policies:** Naturally discover stochastic behavior (no ϵ-greedy needed).
- **Exploration:** Built-in stochasticity promotes exploration.
- **Continuous actions:** Work well with large or continuous action spaces.
- **Smooth updates:** Small $\Delta\theta \Rightarrow$ small $\Delta\pi$, improving stability.

Disadvantages of REINFORCE (Episodic Monte Carlo)

- **Low bias, high variance:** Full random episodes produce unbiased but noisy estimates.
- **Low sample efficiency:** Many trajectories needed to estimate gradients.
- **Slow convergence:** Few updates per trajectory, learning is slower than value-based methods.
- **Local optima:** May converge to suboptimal policies.

Next Lecture

Improved policy-based methods.