Mastering Robotic Arm Manipulation with Deep Reinforcement Learning

Arjun Bansal, Ruchi Patel, Kausar Patherya, Ayushi Rajpoot

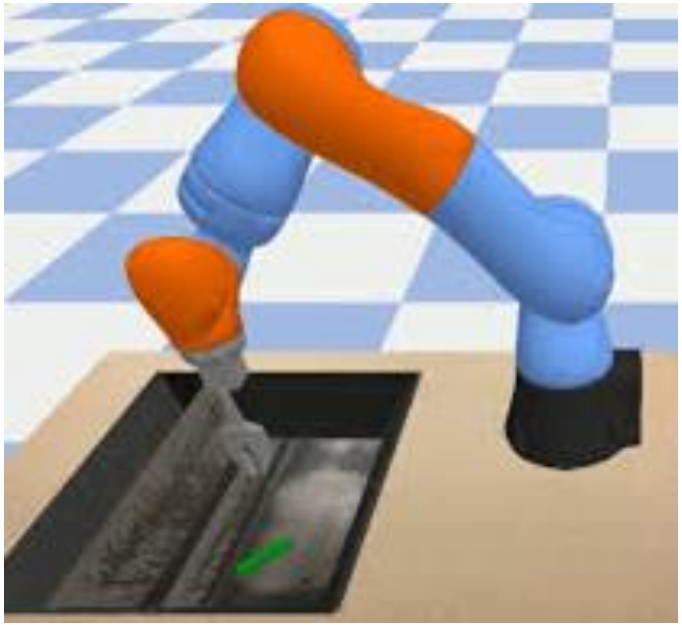
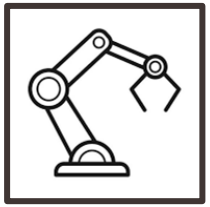


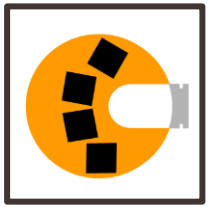
Table of Contents

Motivation



Robotic Manipulation & RL

This field merges robotics and machine learning to develop adaptive systems, enabling robots to learn complex skills like grasping, picking, placing, and assembling through trial and error, enhancing flexibility and generalization.



Kuka PyBullet Environment

The Kuka PyBullet environment simulates a robotic arm for pick-and-place tasks with realistic physics. It is open-source, models real-world systems, and balances complexity with tractability for RL research.

Problem Definition

This environment serves as a testbed to compare RL approaches like A2C, PPO, and DQN (value vs. policy, on vs. off-policy), helping researchers choose algorithms for similar robotics tasks and inform future development.

Table of Contents

Environment

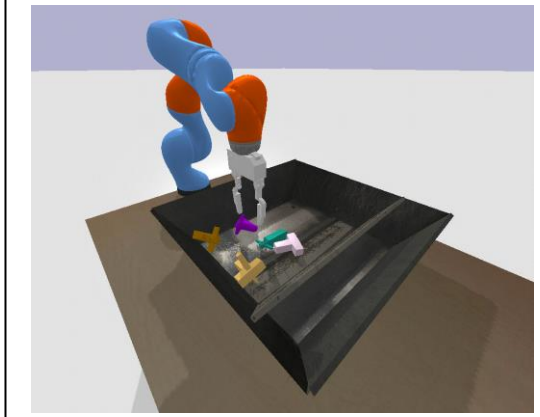
1. The agent has to decide between seven (2 + 2 + 2 + 1) actions so that the manipulator can grasp an object.

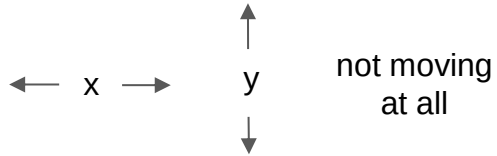
2. The velocity for each direction is equal. There is a “height hack” where the gripper automatically moves down for each action.

3. After picking an action, the agent transitions to a new state from the current one, enjoying/suffering rewards in the process.

4. In this task, the reward is 1 if one of the objects is above height 0.2 at the end of the episode.

5. The inputs to the agent are (48,48,3) images of the environment state. Convolutional neural networks are suited to the task.

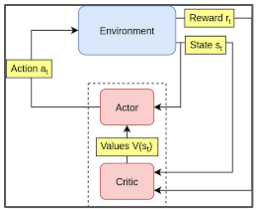




✓ ✗ vertical angle offset?

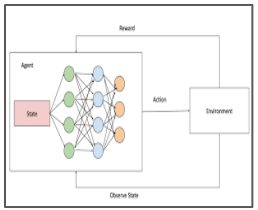
Algorithms

Reinforcement Learning Algorithms



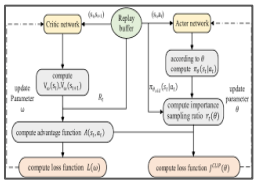
Advantage Actor-Critic

Simultaneously learns a value function and policy. Employs a shared network, enabling efficient learning. Includes entropy regularization to prevent premature convergence.



Deep Q-Networks

Learns control policies from high-dimensional input. Uses a replay buffer and target network to address stability issues. Adaptive decay schedule could lead to adequate exploration.



Proximal Policy Optimization

Extracts more information from each batch of experiences, improving sample efficiency. Generalized Advantage Estimation reduces variance in policy gradient estimates.

Framework

Which algorithm performs best when trained in similar conditions?

Goal

Compare the performance of three RL algorithms ─ A2C, DQN, PPO ─ on robotic arm manipulation.

H1: PPO is expected to excel due to its effectiveness in high-dimensional continuous action spaces.

H2: DQN is robust in discrete action spaces and effective with visual state representations but may fall short in achieving the fine-grained control required for precise robotic manipulation, making it unlikely to outperform PPO.

H3: A2C may underperform compared to PPO and DQN due to its synchronous nature, which could slow learning in high-dimensional, visually driven tasks.

The environment setup was standardized across algorithms for fair comparison. Consistent hyperparameters were used where applicable. Each algorithm was tested across multiple episodes to account for variability.

Table of Contents

Actor-Critic Framework for Kuka Robot Simulation

Bridging Policy and Value-Based Reinforcement Learning

1

Algorithm Overview

Actor: Learns the policy $\pi(a|s;\theta)$ to select actions.
Critic: Estimates the value function $V(s;w)$ to evaluate actions.
Key Insight: Actor improves using feedback from Critic's value estimates.

2

Why Actor-Critic?

Combines the strengths of policy gradient and value-based methods.
Suitable for continuous action spaces like robotic control.
Improves sample efficiency and policy stability.

3

Application in Kuka Simulation

Goal: Train the robotic arm to grasp diverse objects in a simulated environment.
Inputs: Screen observations processed into tensors.
Outputs: Continuous actions for motor control.

Parallelized environments were created, allowing for simultaneous experience collection.

Implementation Pipeline

Steps in Training the Actor-Critic Model

Environment Setup
Kuka environment: discrete timesteps, continuous action space.
Observations: Resized and normalized image frames (84x84x3).

Neural Network Architecture
Actor: Policy network outputting actions.
Critic: Value network estimating $V(s)$.
Shared backbone for feature extraction.

Training Loop
Step 1: Collect experience in parallelized environments.
Step 2: Compute advantages A(s,a) using TD-error.
Step 3: Update actor via policy gradient.
Step 4: Update critic to minimize the loss.

$$\hat{A}(s,a) = r + \gamma V(s') - V(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}[\hat{A}(s,a)\nabla_\theta \log \pi(a|s;\theta)]$$

$$L(w) = \frac{1}{2}\left(r + \gamma V(s') - V(s)\right)^2$$

Monitor & Optimize
Track cumulative rewards and analyze success rate of object manipulation.
Parallel experience collection, Adam optimizer, learning rate scheduling.

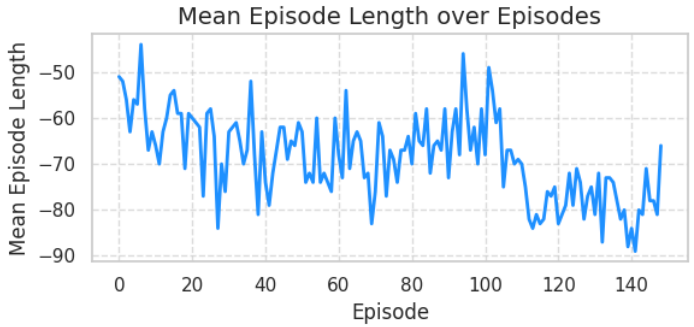
Visualization of Actor-Critic Metrics

Steady improvements across key metrics

1

Mean Episode Length

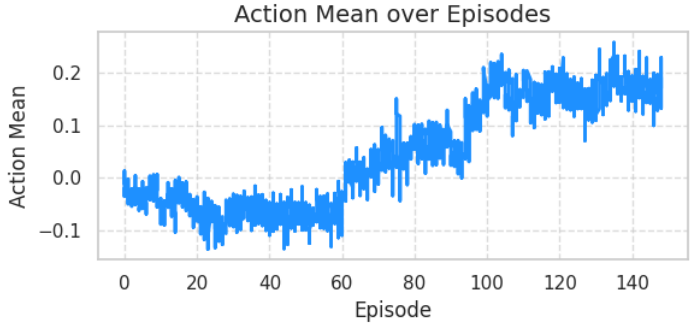
Tracks the average length of episodes as training progresses.
A decrease and eventual plateau in episode lengths reflect consistent task success.



2

Action Mean

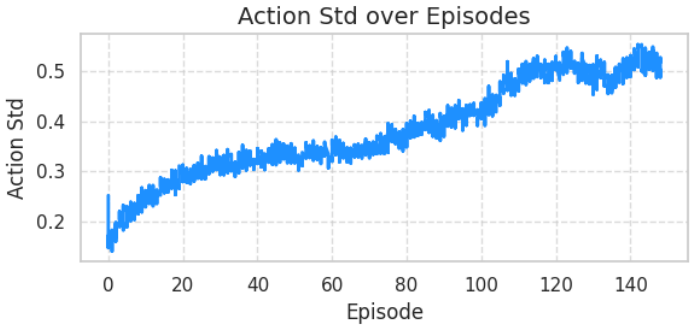
Tracks the average of action outputs.
Increases and stabilizes, centering on preferred action outputs.



3

Action Standard Deviation

Tracks variance of actions, reflecting policy behavior over training
Despite improving action means, the increase in variance hints at a bit of instability.



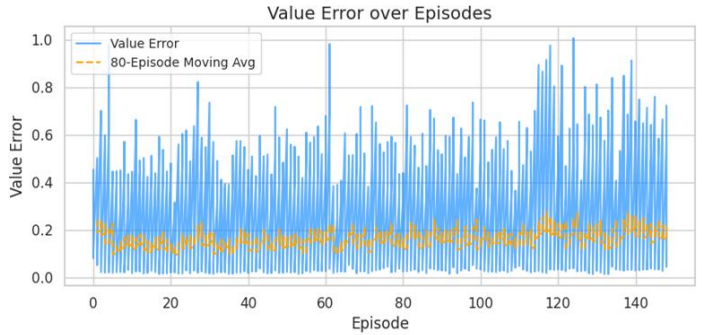
Visualization of Actor-Critic Metrics

Constant fluctuation, no noticeable trends emerge

4

Value Error

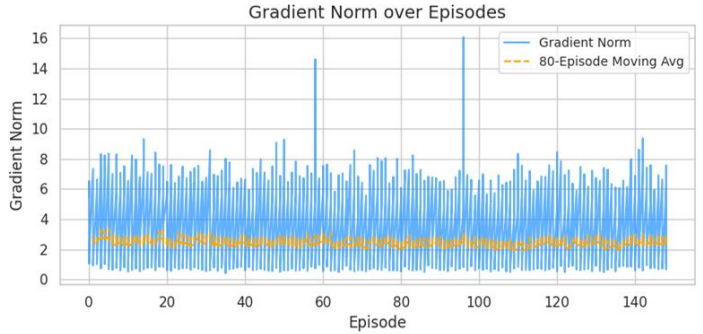
Measures difference between predicted and actual value estimates.
No distinct pattern emerges. Small errors reflect a well-trained critic.



5

Gradient Norm

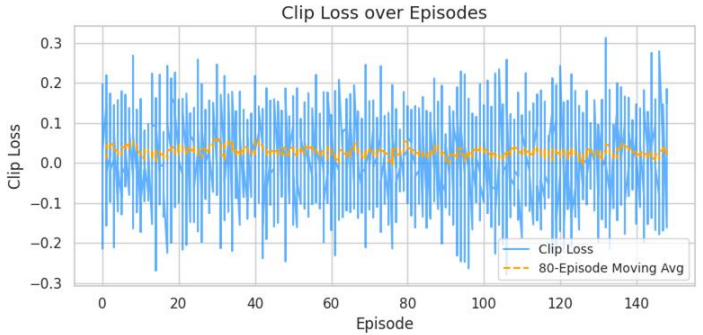
Tracks magnitude of gradients during backpropagation.
Spikes correspond to major policy updates.
Stagnant moving average indicative of stable actor and critic networks.



6

Clip Loss

Constrains how much the policy can change during training.
Fluctuations = tension between old and new policy distributions.
Stable moving average = policy ratio oscillates around the policy boundary but does not move too far.



Deep-Q-Network (DQN) Architecture

Training a DQN agent for vision-based control

1

Network Design

Approximates the Q-function Q(s,a) to estimate the expected return for each action given a state.
Convolutional layers capture the spatial relationships critical for robotic tasks.

Input: 5x5x48 stacked frames (channels-first format).

Convolutional Layers: extract spatial features, batch norm.

Linear Layer: flattened features pass by last layer.

2

Replay Memory

Stores transitions (state, action, reward, next state, reward) observed during interactions with environment.
Random sampling enables decorrelated updates, improving training stability and efficiency.
Cyclic buffer maintains a fixed memory capacity, overwriting old transitions.

3

TD-Learning and Loss Function

As part of the Bellman equation, Q-values are updated to minimize the temporal difference error.
Huber loss is used to handle noisy Q-value estimates. Robust to outliers.

Implementation Pipeline

Steps in Training the DQN Model

Environment Setup

Actions are discretized for simplicity, reducing complexity of learning in continuous action space.
Observations: Resized and normalized image frames (5,5,48)

Neural Network Architecture

Approximates the Q-function to estimate the expected return for each action given a state.
A vector size of 7 is outputted, corresponding to the Q-values for all the possible actions.
3 convolutional layers are able to capture the spatial relationships of the image-like inputs.

Training Loop

Step 1: Batches of transitions sampled randomly from replay memory.
Step 2: Target network stabilizes training, providing fixed Q-values.
Step 3: Epsilon-greedy policy encourages agent to try new actions.
Step 4: Gradient descent is used to minimize Huber loss over targets.

$$\delta = Q(s,a) - \left(r + \gamma \max_{a'} Q(s',a')\right)$$

Monitor & Optimize

Track performance metrics and analyze success rate of object manipulation.
Adjust hyperparameters such as learning rate, replay buffer size, etc

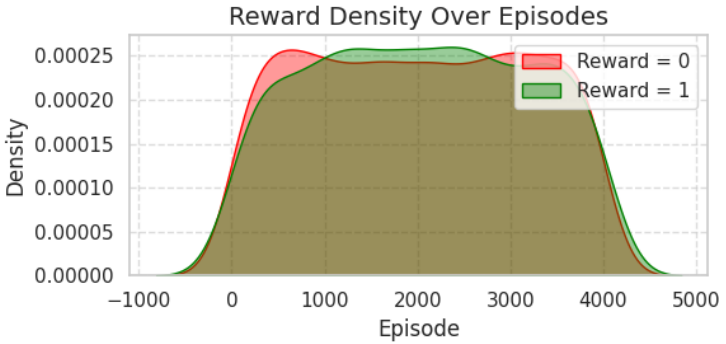
Visualization of DQN Metrics

Learning is gradual

1

Reward Density Over Episodes

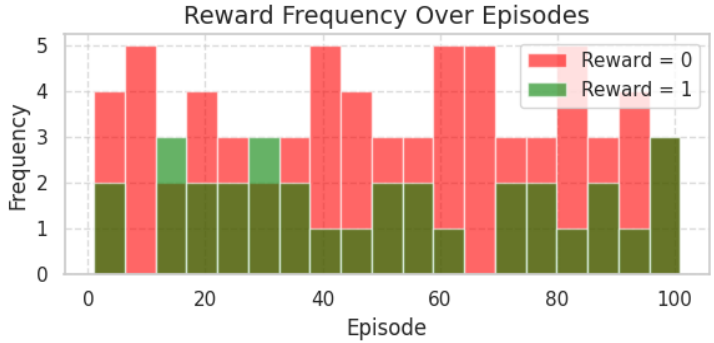
Represents how concentrated the rewards are over episodes.
Distribution of rewards (0 and 1) are relatively equal.



2

Reward Frequency Over Episodes

Tracks how frequently each reward value is received in an episode.
High rewards do tend to occur as training progresses.



3

Mean Reward Over Episodes

Average reward obtained by the agent per episode.
This plateaus and shows light fluctuations, stalled learning progress.

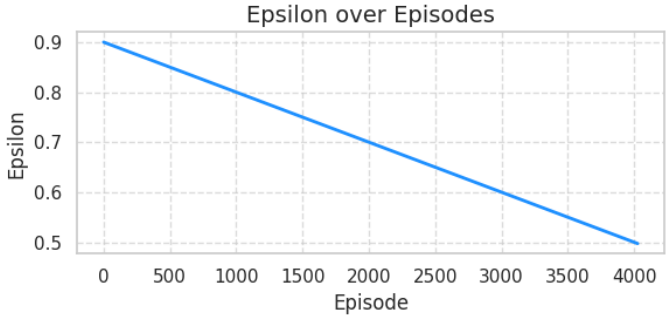
Visualization of DQN Metrics

Exploration yields higher action values.

4

Epsilon Over Episodes

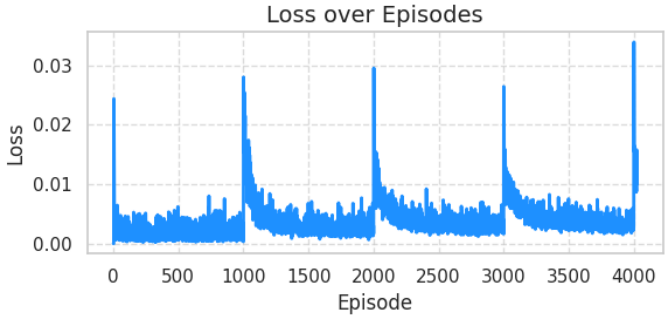
Controls the exploration-exploitation tradeoff (epsilon-greedy).
Shows how exploration decreases over time, arithmetic decay.



5

Loss Over Episodes

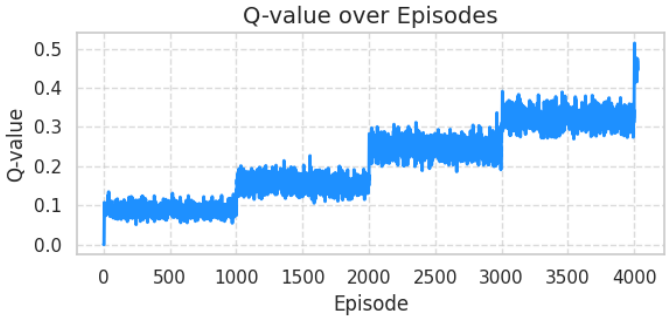
Refers to the error between predicted and target Q-values.
Generally low values for loss but periodic spikes display some instability in training



6

Q-value Over Episodes

Understand how agent's estimates of action values evolve over time.
Increases steadily, agent learns higher action-values across episodes.
Slight fluctuations indicate minor instability.



PPO Framework for Kuka Robot Simulation

Bridging Policy and Value-Based Reinforcement Learning

1

Algorithm Overview

Combines policy gradients and value-based learning with a policy network and value network to optimize agent's policies over multiple updates/seasons

2

Why PPO?

Utilizes **clipped surrogate objectives** to prevent large policy updates and **entropy regularization** to prevent premature convergence
Extracts more information from each batch of experiences, improving sample efficiency
Generalized Advantage Estimation reduces variance in policy gradient estimates.

3

Application in Kuka Simulation

Goal: Apply PPO to train a robotic arm in continuous space and optimize decision making
Inputs: Screen observations processed into tensors.
Outputs: Continuous actions for motor control.

Parallelized environments were created, allowing for simultaneous experience collection.

Implementation Pipeline

Steps in Training the PPO Model

Environment Setup
Kuka environment: discrete timesteps, continuous action space.
Observations: Resized and normalized image frames

Neural Network Architecture
Policy Network: Determines agent's actions by outputting probability distribution for actions
Value Network: Estimates the value of a state which represents expected total future reward

Training Loop
Step 1: Collect trajectories (state, action, reward)
Step 2: Compute advantage using Generalized Advantage Estimation
Step 3: Update Policy Network with clipped objective function
Step 4: Update Value Network by minimizing mean square error

$$L^{CLIP} = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

$$L_{value} = \text{Mean Squared Error}(V(s), R)$$

Monitor & Optimize
Track cumulative rewards and analyze success rate of object manipulation.
Monitor policy loss and value loss & Adjust hyperparameters accordingly

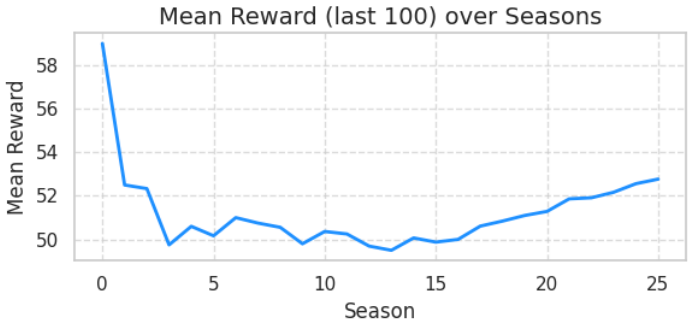
Visualization of PPO Metrics

Steady improvements across key metrics

1

Mean Reward Over Seasons

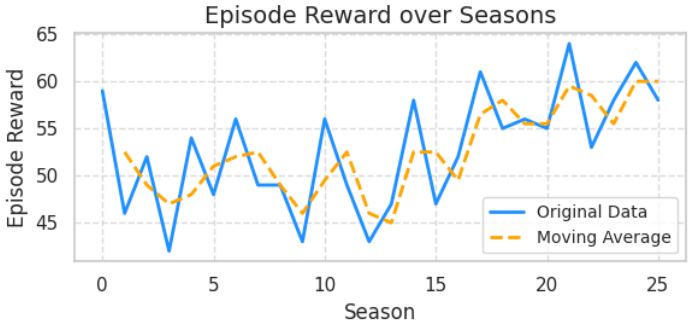
High-level indicator of the agent's performance.
Initial mean reward decreases due to possible exploration, but then recovers and move upwards as policy stabilizes



2

Episode Reward Over Seasons

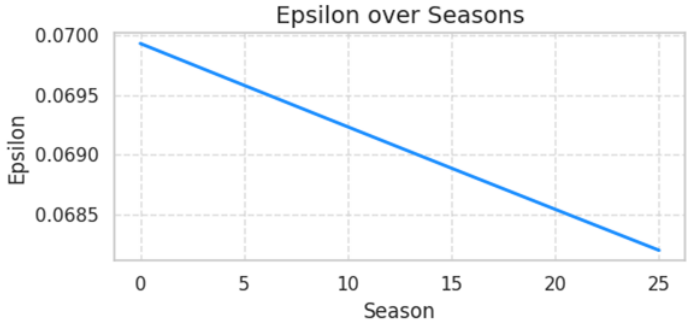
Provides insight into the immediate performance of the agent.
There is a steady improvement in task execution.



3

Epsilon Over Seasons

Controls the exploration-exploitation tradeoff (epsilon-greedy).
Shows how exploration decreases over time, arithmetic decay.



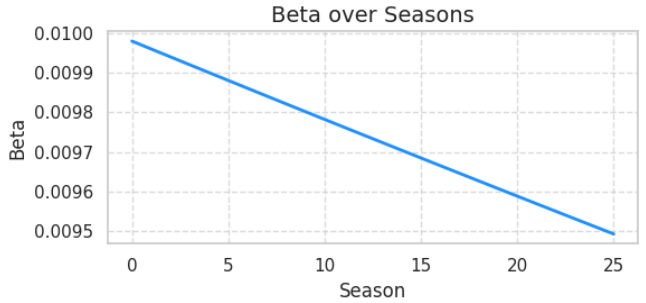
Visualization of PPO Metrics

Loss decreases with convergence issues

4

Beta Over Seasons

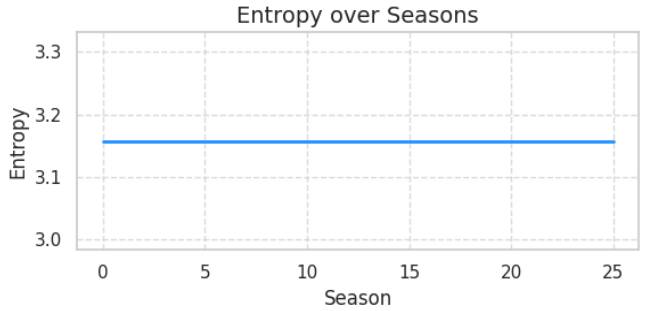
Entropy regularization manages exploration-exploitation balance.
Decreasing beta = agent is focusing on exploiting its policy.



5

Entropy Over Seasons

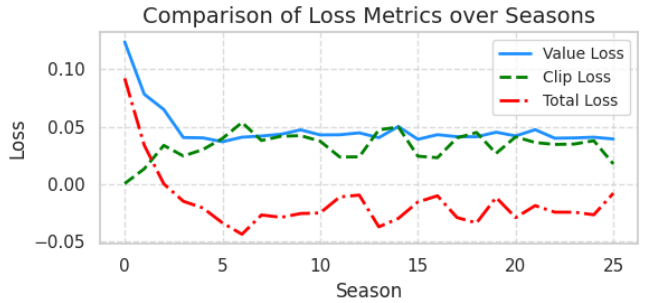
Measures the randomness of the agent's policy.
Remains high = is not converging on a stable policy.



6

Loss Comparison Over Seasons

Indicates how well agent's value function is predicting future rewards.
Clip tracks the stability of the policy update.
A decreasing overall loss = agent is improving across all components



Synthesizing it all

Key takeaways from these comparisons

1

PPO Performs Best

Adaptability to Continuous Action Spaces: Leverages its policy optimization mechanism to handle high-dimensions.
Stable Learning Dynamics: Clipped objective function and adaptive trust region control ensures stable updates.
Metrics Analysis: Outperforms - higher mean rewards, smoother convergence trends, lower variance in training loss.

2

DQN Performance Review

Strength in Discrete Spaces: Less suited for fine-grained control due to reliance on discretization.
Challenges in Precision: Due to discretization, suboptimal performance in continuous domains.
Metrics Fluctuation: Reward density and Q-value over episodes exhibit significant oscillations, reflecting instability.

3

A2C Underperforms

Slow Convergence: Due to its architecture, A2C struggles to converge on optimal policies, particularly in visually driven tasks requiring rapid learning adaptations.
Metrics Analysis: High variability in loss metrics (value error, gradient norm, clip loss) suggests instability.

Thanks

Do you have any questions?