A Distributional Perspective on Reinforcement Learning

Krishna Agarwal,Kintan Saha,Gavish Bansal

Indian Institute of Science, Bangalore

June 1, 2025

Motivation

- Traditional reinforcement learning (RL) focuses on maximizing expected returns.
- This approach assumes that the expectation of return provides sufficient information about its Distribution.
- However, in many real-world scenarios involving risks(eg. Stock Market) this underlying distribution is very important in order to get some better insight on data.

Setting

- The agent interacts with an environment through actions and observations over time.
- This interaction is modeled as a Markov Decision Process (MDP):

$$(\mathcal{X}, \mathcal{A}, R, P, \gamma)$$

- Components of the MDP:
  - $\mathcal{X}$: state space
  - $\mathcal{A}$: action space
  - $P(\cdot|x, a)$: transition kernel
  - R: reward function, treated as a random variable
  - $\gamma \in [0, 1]$: discount factor
- A stationary policy π maps each state $x \in \mathcal{X}$ to a distribution over actions $\mathcal{A}$.

Distributional View

- A random variable U is distributed according to the same law as V if $U \overset{D}{=} V$.
- This relation indicates that the two sides of a distributional equation are equal in distribution, not necessarily in value.
- We denote:
  - Cumulative distribution function (c.d.f): $F_U(y) := \mathbb{P}(U \leq y)$
  - Inverse c.d.f (quantile function): $F_U^{-1}(q) := \inf\{y : F_U(y) \geq q\}$
- These ideas are fundamental for defining distance between distributions.

Measuring Distance Between Distributions

- Need a metric to compare probability distributions.
- Total variation, KL divergence, and Kolmogorov distance are unsuitable. (WHY?)
- **Wasserstein metric (a.k.a. Earth Mover's Distance)** is ideal:

$$d_p(F, G) = \inf_{U,V} ||U - V||_p$$

$$d_p(F, G) = \left(\int_0^1 |F^{-1}(u) - G^{-1}(u)|^p du \right)^{1/p}$$

- It captures the cost of transforming one distribution into another.
- Define $\bar{d}_p$ as:

$$\bar{d}_p(Z_1, Z_2) := \sup_{x,a} d_p(Z_1(x, a), Z_2(x, a)).$$

Policy Evaluation

- $P^\pi : \mathcal{Z} \to \mathcal{Z}$

$$P^\pi Z(x,a) \overset{D}{:=} Z(X',A') \tag{4}$$
$$X' \sim P(\cdot \mid x,a), \quad A' \sim \pi(\cdot \mid X'),$$

- $\mathcal{T}^\pi : \mathcal{Z} \to \mathcal{Z}$ as

$$\mathcal{T}^\pi Z(x,a) \overset{D}{:=} R(x,a) + \gamma P^\pi Z(x,a). \tag{5}$$

  - It can be shown that for a fixed policy π, T^π is a γ-contraction under $\bar{d}_p$.
  - This implies convergence to a unique fixed point Z^π.
  - That's what we want in Policy Evaluation setting

Control in Distributional RL

Let Π^* be the set of optimal policies. We begin by characterizing what we mean by an *optimal value distribution*.

Definition 1 (Optimal value distribution). *An optimal value distribution is the v.d. of an optimal policy. The set of optimal value distributions is* $\mathcal{Z}^* := \{Z^{\pi^*} : \pi^* \in \Pi^*\}$.

Note that not all value distributions with expectation Q^* are optimal. They must match the full distribution of the return under some optimal policy.

Control in Distributional RL

Definition 2. *A greedy policy π for $Z \in \mathcal{Z}$ maximizes the expectation of Z. The set of greedy policies for Z is*

$$\mathcal{G}_Z := \left\{ \pi : \sum_a \pi(a \mid x) \mathbb{E}Z(x, a) = \max_{a' \in \mathcal{A}} \mathbb{E}Z(x, a') \right\}.$$

An operator $\mathcal{T}$ is called a distributional Bellman optimality operator if it implements a greedy selection rule, i.e.

$$\mathcal{T}Z = \mathcal{T}^\pi Z$$

for some $\pi \in \mathcal{G}_Z$

Control in Distributional RL

Lemma

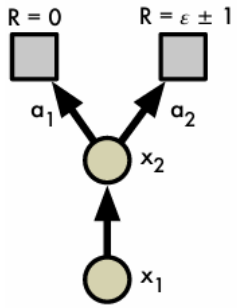
Let $Z_1, Z_2 \in \mathcal{Z}$. Then

$$\|\mathbb{E}\mathcal{T}Z_1 - \mathbb{E}\mathcal{T}Z_2\|_\infty \leq \gamma \|\mathbb{E}Z_1 - \mathbb{E}Z_2\|_\infty ,$$

and in particular $\mathbb{E}Z_k \to Q^*$ exponentially quickly.

- By inspecting This Lemma, we might expect that Z_k converges quickly in d_p to some fixed point in Z . But Unfortunately, convergence is neither quick nor assured to reach a fixed point.
- This is becaause this Optimality Operator as defined above is not a contraction in any standard metric defined above.

Non-Expansion in Distributional RL



	x_1	x_2, a_1	x_2, a_2
Z^*	$\epsilon \pm 1$	0	$\epsilon \pm 1$
Z	$\epsilon \pm 1$	0	$-\epsilon \pm 1$
$\mathcal{T}Z$	0	0	$\epsilon \pm 1$

Figure 2. Undiscounted two-state MDP for which the optimality operator $\mathcal{T}$ is not a contraction, with example. The entries that contribute to $\bar{d}_1(Z, Z^*)$ and $\bar{d}_1(\mathcal{T}Z, Z^*)$ are highlighted.

Figure: Counter Example in Control Setting

Now consider Z as given in Figure, and its distance to Z^*:

$$\bar{d}_1(Z, Z^*) = d_1(Z(x_2, a_2), Z^*(x_2, a_2)) = 2\epsilon,$$

When we apply $\mathcal{T}$ to Z, however, the greedy action a_1 is selected and $\mathcal{T}Z(x_1) = Z(x_2, a_1)$. But

$$d_1(\mathcal{T}Z, \mathcal{T}Z^*) = d_1(\mathcal{T}Z(x_1), Z^*(x_1)) = \frac{1}{2}|1 - \epsilon| + \frac{1}{2}|1 + \epsilon| > 2\epsilon$$

Algorithm might not converge

- Consider the example where $\epsilon = 0$.
- The tie breaker rule is Choose a_2 if $Z(x_1) = 0$ and a_1 Otherwise.
- It can easily be seen that if $\mathcal{T}$ is repeatedly applied on Z^* then it will keep on oscillating between $Z(x_1) = 0$ and $Z(x_1) = \pm 1$.
- This shows that Repeated application of $\mathcal{T}$ need not converge.

Interpretation

- This shows that the undiscounted update is not a nonexpansion as:

$$\bar{d}_1(\mathcal{T}Z, \mathcal{T}Z^*) > \bar{d}_1(Z, Z^*).$$

- From discussion so far, it is very much clear that it is not possible to repeat the algorithms used in Q-value estimation where optimality operator was a contraction in L_{inf} norm in a theoretically sound manner.
- Even though convergence is not guaranteed in some cases as seen already but in general this algorithm works quite well in real world applications.

Parametric Distribution

We model the value distribution using a discrete distribution parameterized by:

- $N \in \mathbb{N}$: number of atoms
- $V_{\text{min}}, V_{\text{max}} \in \mathbb{R}$: range of values

The support is the set of atoms:

$$\{z_i = V_{\text{min}} + i\Delta z : 0 \leq i < N\}, \quad \Delta z := \frac{V_{\text{max}} - V_{\text{min}}}{N - 1}$$

These atoms are the "canonical returns" of the distribution.

The atom probabilities are given by a parametric model:

$$\theta : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^N$$

$$Z_\theta(x, a) = z_i \quad \text{w.p.} \quad p_i(x, a) := \frac{e^{\theta_i(x,a)}}{\sum_j e^{\theta_j(x,a)}}$$

The discrete distribution is expressive and computationally efficient

Visualization of Algorithm

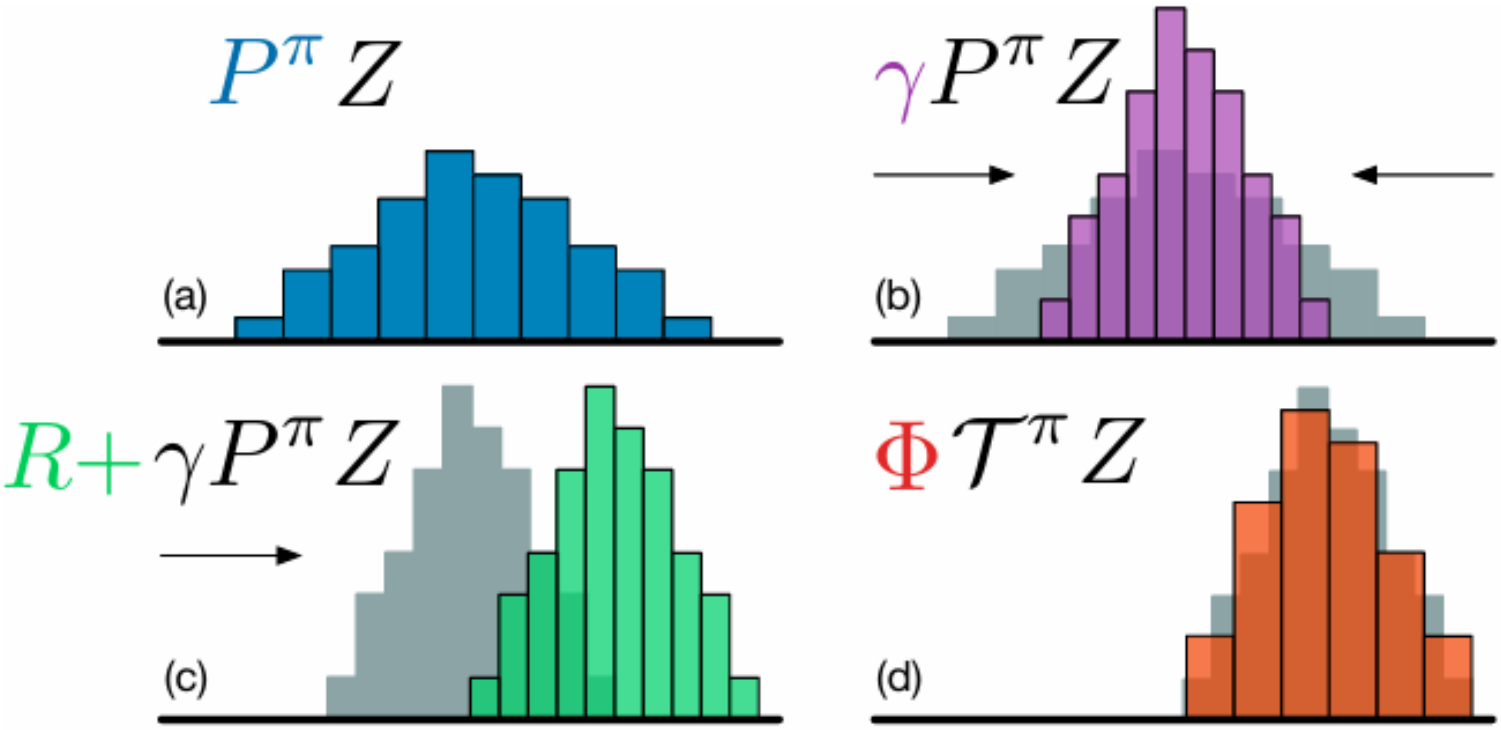


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

Figure: Visualization

Projected Bellman Update

- Problem we will be facing while using Parametric distribution is that in general $\mathcal{T}Z$ have different supports as compared to Z.
- To Solve this issue, we'll introduce this projection operator to project $\mathcal{T}Z_\theta$ onto the supports of Z_θ.
- Let π be the greedy policy with respect to $\mathbb{E}Z_\theta$. Given a sample transition (x, a, r, x'), we compute the Bellman update $\hat{\mathcal{T}}z_j := r + \gamma z_j$ for each atom z_j, then distribute its probability $p_j(x', \pi(x'))$ to the immediate neighbours of $\hat{\mathcal{T}}z_j$.
- Now,if Φ is the projection operator, Then the sample loss $\mathcal{L}_x(\theta)$ is the cross-entropy term of the KL divergence:

$$D_{\mathrm{KL}}(\Phi\hat{\mathcal{T}}Z_{\hat{\theta}}(x, a) \parallel Z_\theta(x, a)).$$

Algorithm 1: Categorical Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \ldots, N-1$

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z \quad \# \, b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

Experiments

We validated the distributional algorithms across a range of environments:

- **LunarLander, CartPole, MountainCar, CliffWalking**

The results were compared with state-of-the-art baselines including:

- **Double DQN**
- **Dueling DQN**

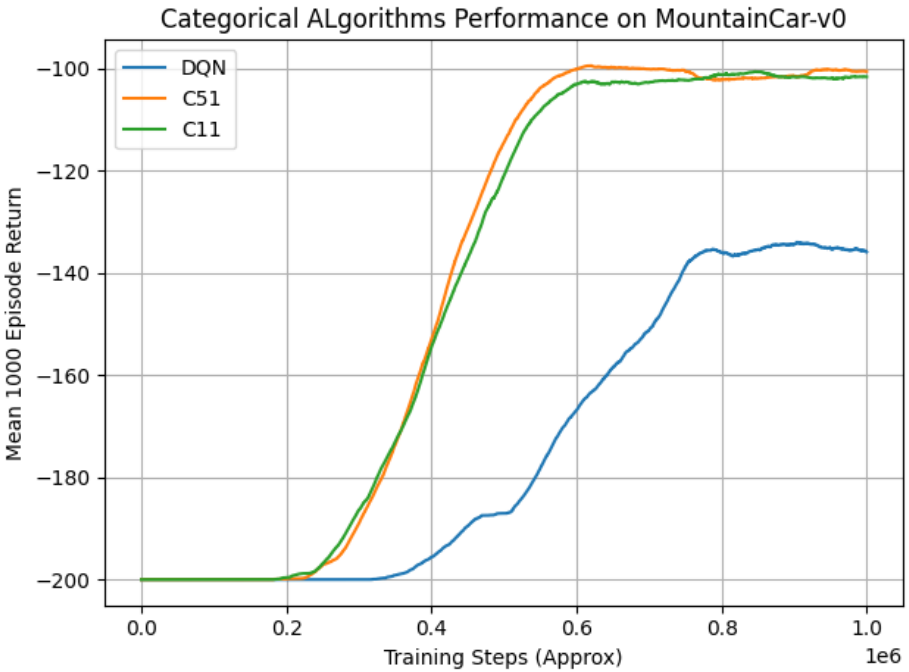
While the original paper reports results on all **57 Atari games**, due to limited compute resources, we focused on environments with simpler reward structures.

Despite this, our categorical algorithms achieved strong performance. Given their ability to model complex reward distributions, we believe similar improvements would extend to the Atari benchmarks as demonstrated in the paper.

Experiment Details

- **Categorical Algorithms** (e.g., C51, C11):
  - Use a feedforward neural network with two hidden layers of sizes **128** and **512**, respectively.
- **DQN, DDQN, and Dueling DQN**:
  - Use a feedforward neural network with two hidden layers of size **128**.
  - A **soft update** rule is applied to update the target network.
  - The soft update parameter is set to `1e-3`.
- **Activation Function**: `ReLU`
- **Optimizer**: `Adam`
- **Exploration Strategy (ϵ-greedy)**:
  - ϵ is linearly annealed from **1.0** to **0.01**.
  - Decay occurs over the first **50% of total timesteps**.

MountainCar-v0



Training Curve: 1M steps averaged over 1000 episodes

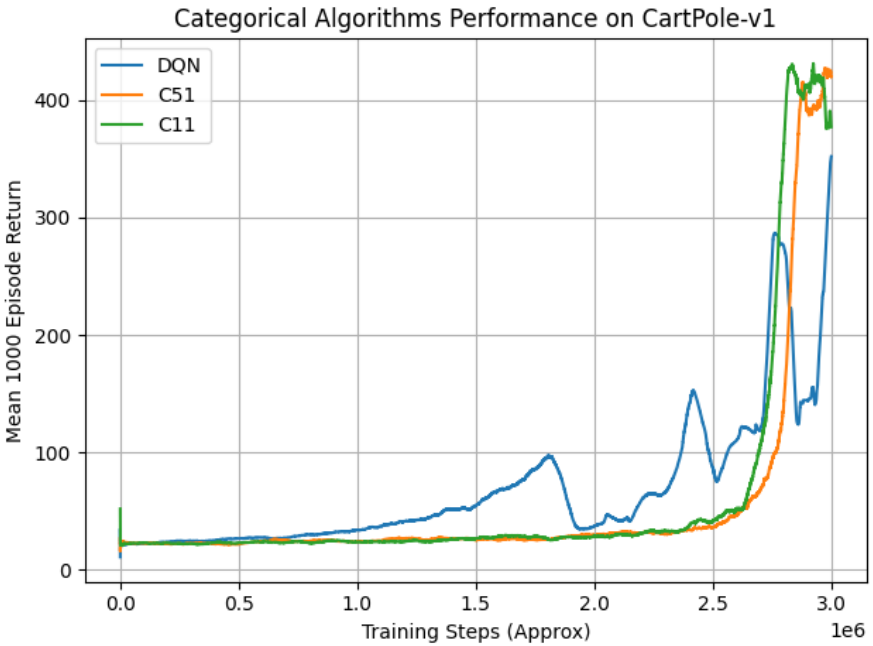
Environment Details

- **Observation Space:** 2D continuous
- **Action Space:** Discrete (3 actions)
- **Reward:** -1 per timestep until goal
- **Max Steps:** 200

Training Config

- $\gamma = 0.99$
- $v_{\text{min}} = -200,\ v_{\text{max}} = 0$
- Learning rate: 1×10^{-4}
- Replay buffer size: 5000
- Batch size: 64
- Target update: every 300 steps

CartPole-v1



Training Curve: 3M steps averaged over 1000 episodes

Environment Details

- **Observation Space:** 4D continuous
- **Action Space:** Discrete (3 actions)
- **Reward:** $+1$ per timestep until falls
- **Max Steps:** 500

Training Config

- $\gamma = 0.99$
- $v_{\text{min}} = 0,\ v_{\text{max}} = 500$
- Learning rate: 1×10^{-4}
- Replay buffer size: 5000
- Batch size: 64
- Target update: every 300 steps

LunarLander-v2

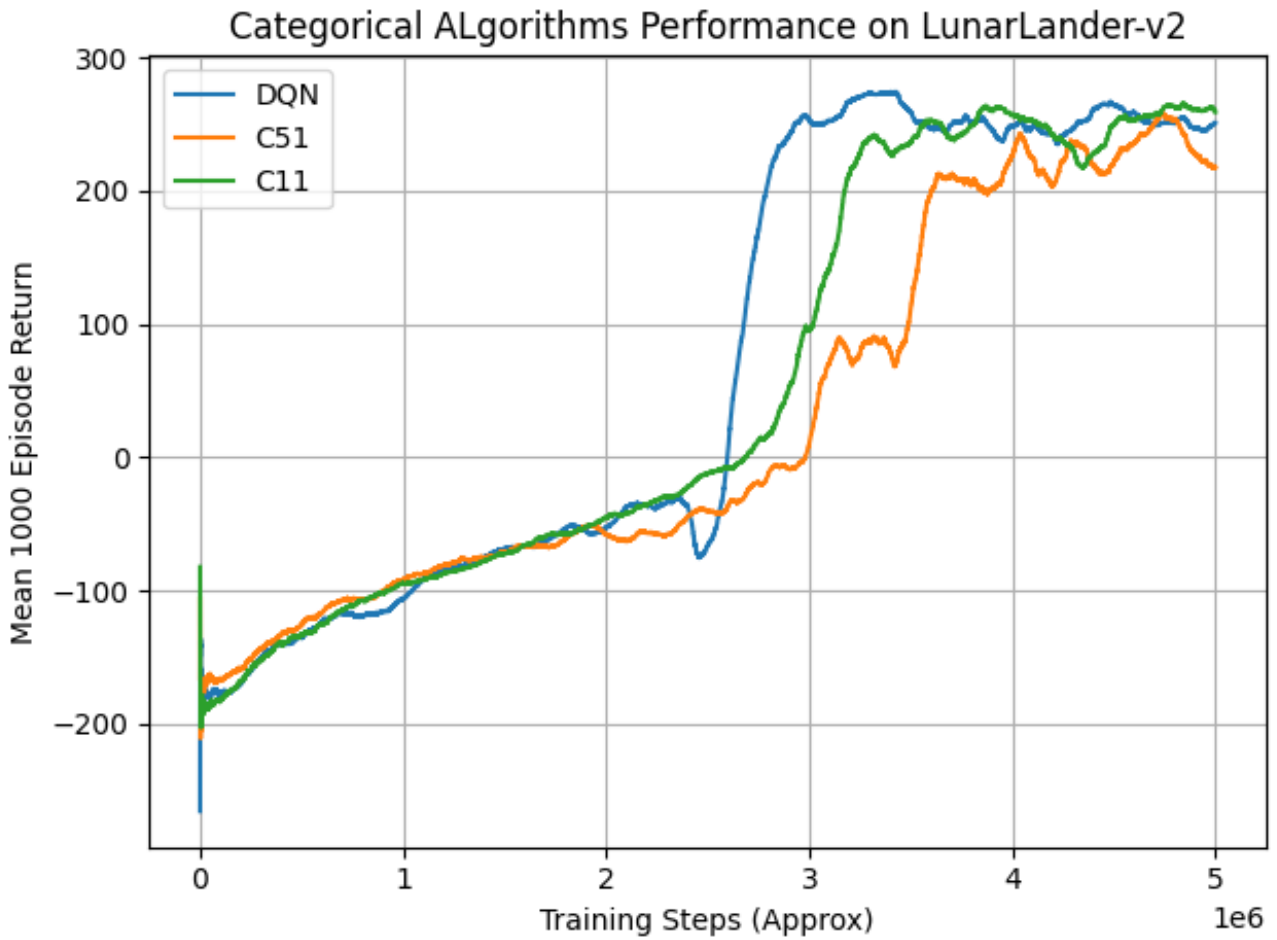


Figure: *Training Curve: 5M steps averaged over 1000 episodes*

LunarLander-v2

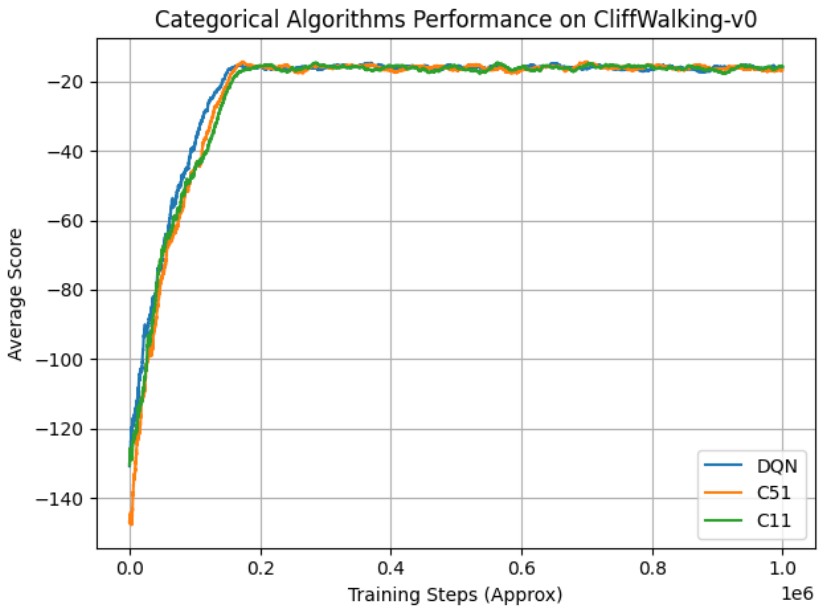
Environment Details

- **Observation Space:** 8D continuous
- **Action Space:** Discrete (4 actions)
- **Reward Structure:**
  - +100 to +140 for successful landing
  - -100 for crashing
  - -0.3 per frame (fuel cost)
  - +10 per leg contact
  - Episode ends on crash, landing, or 1000 steps

Training Config

- $\gamma = 0.99$
- $v_{\text{min}} = -250$, $v_{\text{max}} = 250$
- Learning rate: 5×10^{-5}
- Replay buffer size: 5000
- Batch size: 64
- Target update: every 300 steps

CliffWalking-v0



Training Curve: 1M steps averaged over 1000 episodes

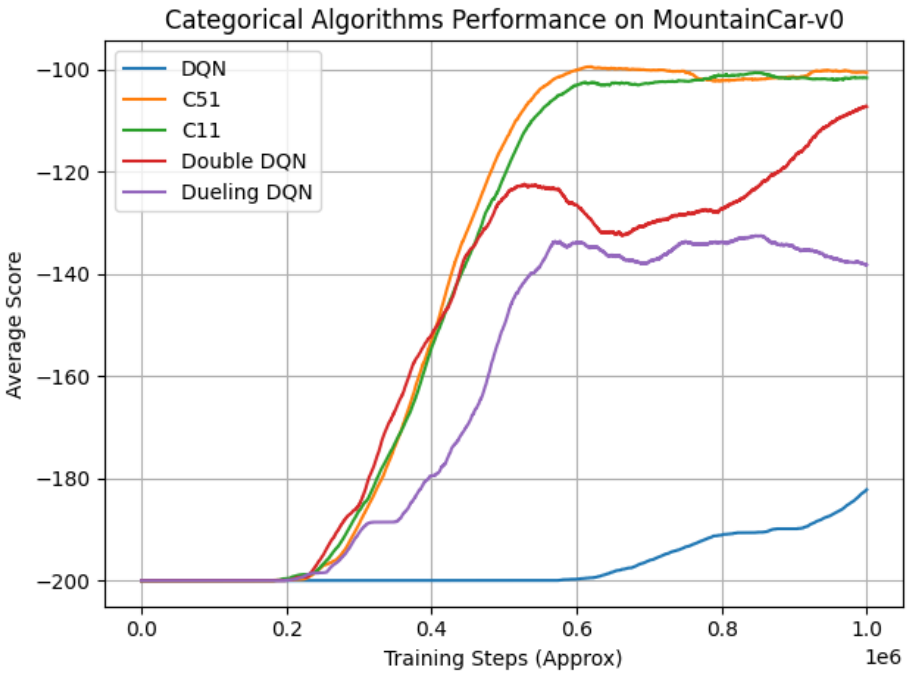
Environment Details

- **Observation Space:** Discrete (48 states)
- **Action Space:** Discrete (4 actions)
- **Reward:** -1 per step, -100 if falls off cliff
- **Goal:** Reach bottom-right safely

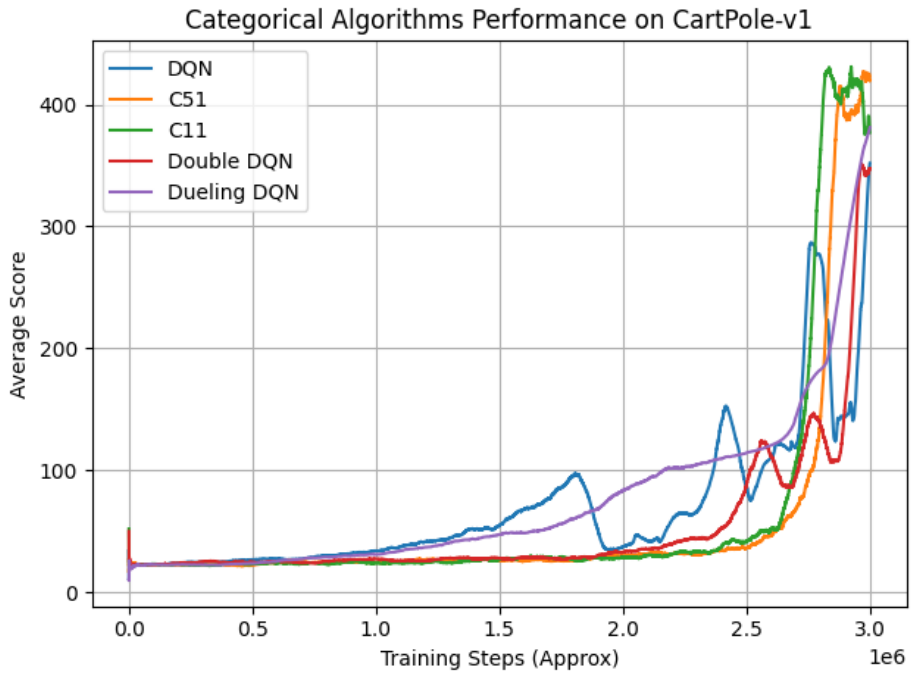
Training Config

- $\gamma = 0.99$
- $v_{\text{min}} = -120,\ v_{\text{max}} = 10$
- Learning rate: 5×10^{-4}
- Replay buffer size: 5000
- Batch size: 64
- Target update: every 300 steps

Comparison with other Baselines

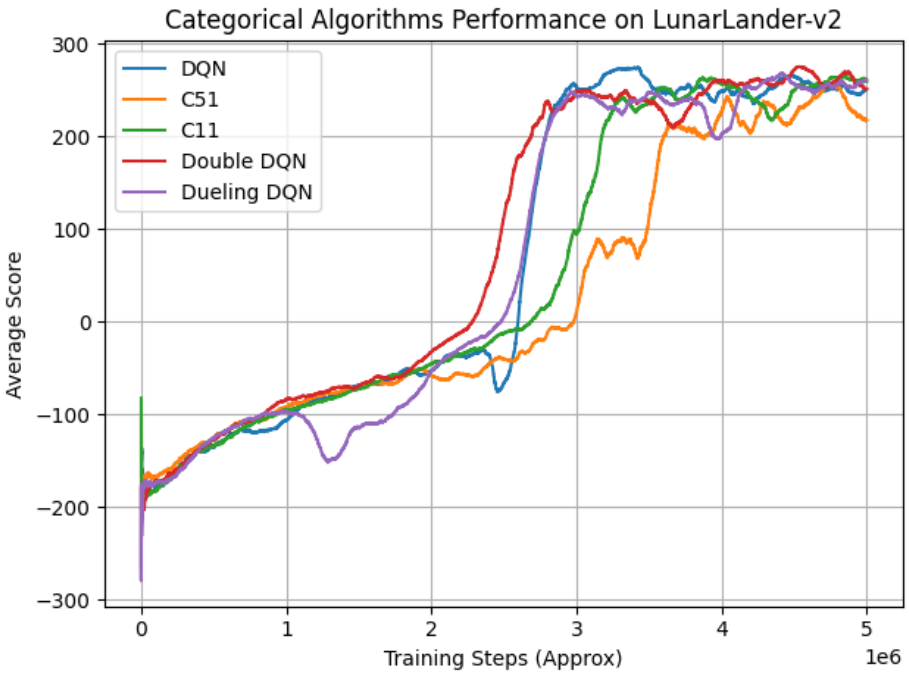


Training Curve over 1M steps

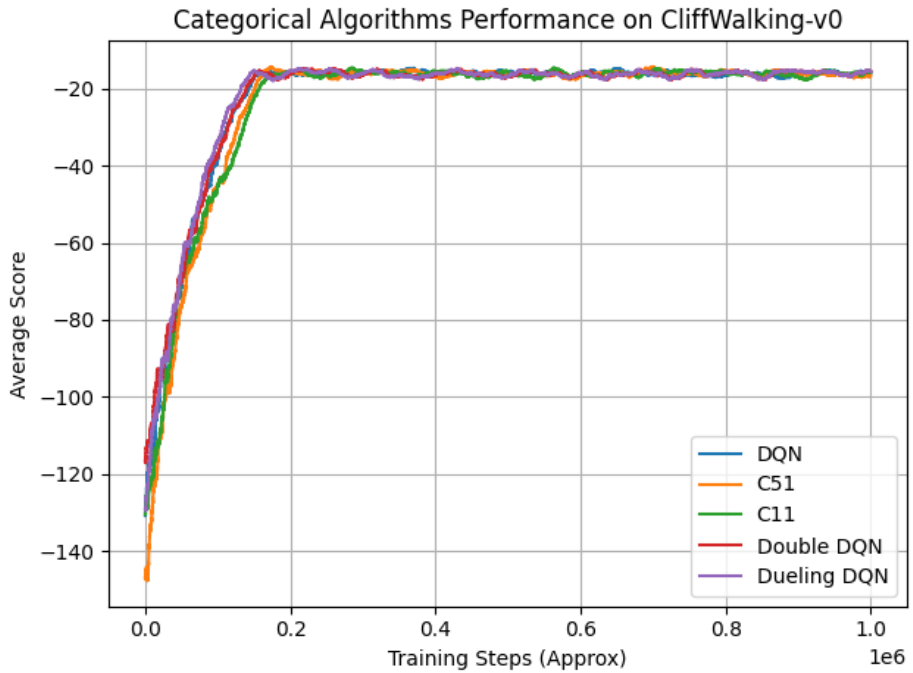


Training Curve over 3M steps

Comparison with other Baselines



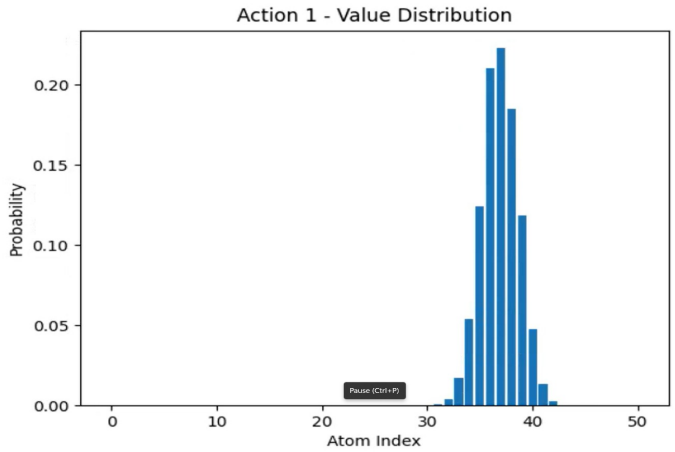
Training Curve over 5M steps

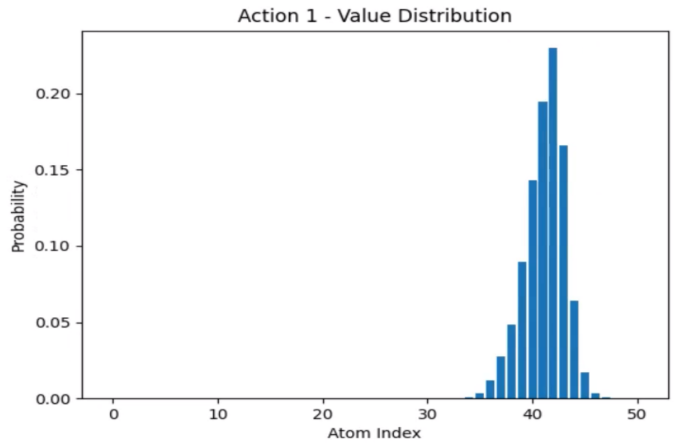


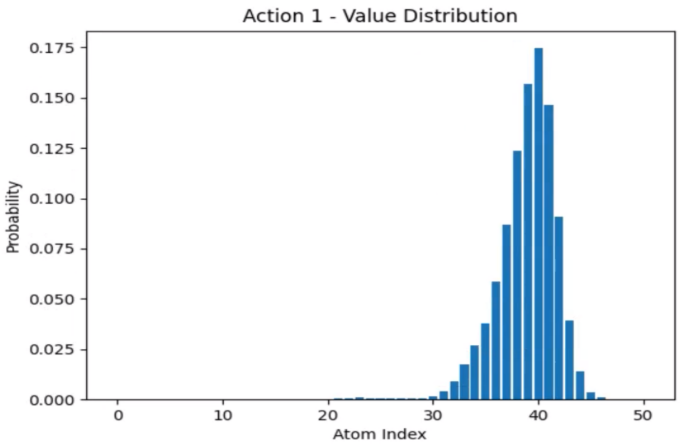
Training Curve over 1M steps

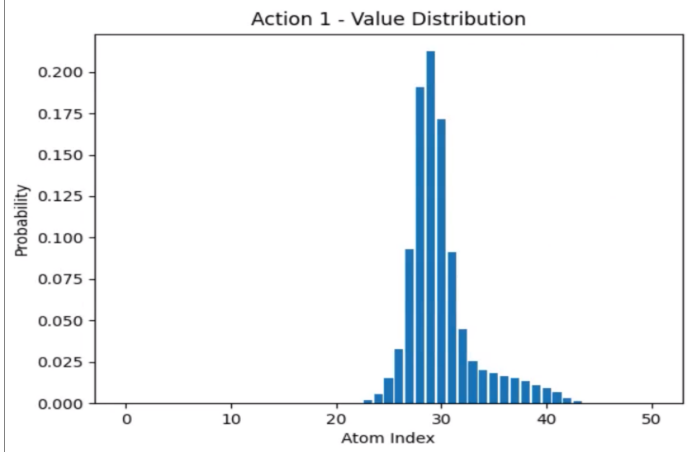
Visualizing the value distribution

We also attempt to visualise the value distribution for LunarLander-v2.









Any Questions?

We're happy to discuss!