2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Distributional Reinforcement Learning with Sample-set Bellman Update

Weijian Zhang[1], Jianshu Wang[2], Yang Yu[2]

***Abstract*— Distributional Reinforcement Learning (DRL) not only endeavors to optimize expected returns, but also strives to accurately characterize the full distribution of these returns, a key aspect in enhancing risk-aware decision-making. Previous DRL implementations often inappropriately treat statistical estimations as concrete samples, which undermines the integrity of learning. While several studies have addressed this issue, they frequently give rise to new complications, including computational burdens and diminished stochastic behavior. In our work, we present a novel DRL framework that leverages the Gaussian mixture model to adeptly depict the distribution of returns. This approach ensures precise, authentic sampling critical for robust learning, while also preserving computational tractability. Through extensive evaluation on a diverse array of 59 Atari games, our method not only surpasses the efficacy of prior DRL algorithms but also presents formidable competition to contemporary top-tier RL algorithms, signifying a substantial advancement in the field.**

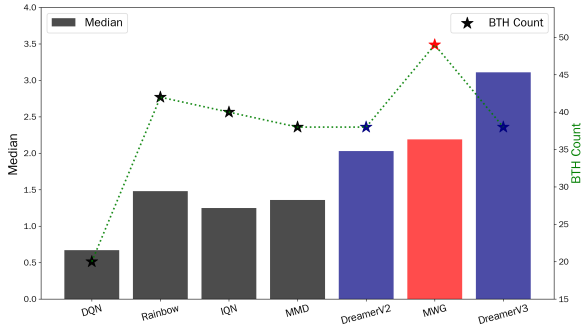

Fig. 1: Median scores (bars, left-axis) and Count of Better than Human (BTH Count) scores (stars, right-axis) of the compared baseline and state-of-the-art algorithms over 59 Atari games

I. INTRODUCTION

The concept of return, defined as the cumulative reward, is pivotal and typically serves as the common objective in Reinforcement Learning (RL). Traditional RL primarily concentrates on estimating the mathematical expectation of the return, often neglecting the return's stochasticity. In contrast, the emerging Distributional Reinforcement Learning (DRL) paradigm endeavors to model this return distribution. Such a return distribution offers a more nuanced basis for real-world decision-making, particularly in fields sensitive to risk [1], [2].

DRL algorithms typically follow a two-stage framework. In the first stage, a neural network generates statistics, such as quantiles, to represent the return distribution. In the second stage, these statistics are updated using a Bellman operator to form the target for the neural network's training. However, not all statistics are amenable to updates via the Bellman operator [3]. Inappropriate handling of statistics in DRL algorithms can lead to biased models. For example, directly updating quantiles using the Bellman operator, as in QR-DQN [4], can underestimate the true variance and converge to a degenerate distribution [3].

The capacity for statistics to be directly updated by the Bellman operator is known as *Bellman closeness* [3]. This property suggests that a set of statistics can be analytically transformed to represent the outcome following a Bellman update, reflecting the advantages of these statistics in terms of update efficiency and accuracy. While conventional moments can achieve Bellman closeness, alternative approaches have been developed to address this theoretical limitation by applying the Bellman update to actual return samples instead of derived statistics, resulting in what is termed *approximate Bellman closure*.

The ER-DQN approach introduces an imputation strategy that generates a set of samples from expectiles [3]. However, this strategy requires solving a system of multivariate nonlinear equations iteratively, which is computationally demanding and reduces the practicality of ER-DQN. The MMD algorithm circumvents the Bellman closeness issue by maintaining a consistent sample set throughout the learning process [5]. Nevertheless, the lack of randomness in the fixed sample set undermines the essential characteristics of sampling.

To balance approximate Bellman closure with computational efficiency, this paper proposes employing a Gaussian Mixture Model (GMM) to represent the return distribution. GMMs are known for their sampling efficiency and reliable parameter estimation through the Expectation-Maximization (EM) method. To alleviate the computational burden inherent in the iterative EM process, we introduce an advanced single-iteration parameter estimation technique. However, due to the non-linear nature of the EM method with respect to samples, updating the neural network based on random samples may introduce bias. We address this by proposing a practical sample-set augmentation strategy that preserves unbiased learning. The proposed Mixed Weighted Gaussian (MWG) DRL algorithm integrates these techniques within a DQN framework.

Empirical studies on 59 Atari games indicate that the MWG algorithm exhibits significant performance improve-

[1]Weijian Zhang is affiliated with Polixir.ai, Nanjing, China, email: weijian.zhang@polixir.ai
[2]Jianshu Wang and Yang Yu are affiliated with National Key Laboratory for Novel Software Technology, School of Artificial Intelligence, Nanjing University and Polixir.ai, Nanjing, China

ments over current state-of-the-art DRL models. As shown in Figure 1, the median score of MWG on the Atari game platform exceeds that of DreamerV2 [6], and it outperforms human-level performance in 49 out of the 59 games, surpassing DreamerV3 [7]. These results highlight MWG's effectiveness in addressing specific challenges and its broad applicability.

The primary contributions of this paper are threefold: first, it employs a GMM for modeling the return distribution; second, it proposes a single-step EM algorithm tailored for reinforcement learning, addressing the computational efficiency issues of traditional EM algorithms; and third, it introduces a sample-set mixing technique to estimate the expected distribution, thereby overcoming the non-linear transformation problem associated with the EM algorithm.

II. Background

In a Markov Decision Process, described by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, the environment initially presents a state s. An agent then decides to select an action a to execute. This action induces a state transition in the environment, leading to a new state s' and providing a reward r to the agent. The agent's objective is to maximize the cumulative reward over several interactions (i.e., the return). To accomplish this, the agent persistently optimizes its policy π in response to the received rewards. In indeterministic environments, the return is a random variable due to the randomness of the state transitions T and the rewards R, denoted by $Z_\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t R_t$, and its distribution by $\eta_\pi(s,a)$.

Similar to the value-based reinforcement learning (RL) theory, the core concept of distributional RL (DRL) is the distributional Bellman operator:

$$\eta_\pi(s,a) := (\mathcal{T}^\pi \eta_\pi)(s,a) = \mathbb{E}[(B_{R,\gamma})_{\#}\eta_\pi(S',A')s,a], \quad (1)$$

where $B_{R,\gamma} : \mathbb{R} \to \mathbb{R}$ is defined by $B_{R,\gamma}(x) = R + \gamma x$, and $g_{\#}\eta \in \mathscr{P}(\mathbb{R})$ denotes the pushforward operation of the distribution η by the function g, such that for any Borel set $A \subseteq \mathbb{R}$, $g_{\#}\eta(A) = \eta(g^{-1}(A))$. Due to the stochastic nature of environment transitions, a variety of subsequent state-action pairs (s',a') can emerge from a given pair (s,a). These pairs form a set $\{(s',a')\}$, with each element corresponding to a return distribution $(B_{R,\gamma})_{\#}\eta_\pi(s',a')$, referred to as a **sample distribution**. The average of these distributions, $\mathbb{E}\left[(B_{R,\gamma})_{\#}\eta_\pi(S',A')\right]$, termed the **expected distribution**, defines the return distribution $\eta_\pi(s,a)$.

The goal of DRL algorithms is to model the return distribution $\eta_\pi(s,a)$. Since the return distribution is an infinite-dimensional function, it cannot be accurately represented with a finite number of parameters. The common approach is to use a set of statistics to approximate it. Existing DRL algorithms are categorized based on the statistics used for approximation. Algorithms such as C51 [8], D4PG [9], and Rainbow [10] are based on a discrete distribution model, while QR-DQN, IQN [11], and FQF [12] utilize a quantile model. ER-DQN employs an expectile model, and MoG [13] is based on the Gaussian mixture model.

A. QR-DQN: Typical Case

QR-DQN is a representative DRL algorithm. Given a transition tuple (s,a,r,s',a'), it predicts a set of quantiles $q = [q_1,\ldots,q_K]$ to approximate the return distribution $\eta_\pi(s,a)$ and another set $q' = [q'_1,\ldots,q'_K]$ for $\eta_\pi(s',a')$. These quantile sets are associated with quantile fractions $\tau = [\tau_1,\ldots,\tau_K]$. The algorithm updates the q' quantiles with a Bellman operation:

$$\hat{q} = [r + \gamma q'_1, \ldots, r + \gamma q'_K]$$

A loss function is then constructed using the quantile regression framework. The temporal-difference error δ_{ij} for each pair of quantiles is computed as:

$$\delta_{ij} = \hat{q}_j - q_i$$

The neural network parameters θ are updated by minimizing the following objective:

$$\arg\min_\theta \sum_{i=1}^{K}\sum_{j=1}^{K} \left|\tau_i - \mathbb{I}_{(\delta_{ij}<0)}\right| |\delta_{ij}| \quad (2)$$

where θ denotes the network parameters.

B. ER-DQN: Separate Statistics from Samples

QR-DQN faces a conceptual challenge: the network's outputs are treated as both samples and statistics during updates and loss evaluation, leading to potential biases. Rowland et al. [3] proposed a method to reconcile this by converting statistical outputs into samples for consistent updates. ER-DQN utilizes expectiles as a robust statistical representation, where for a set of expectiles $[e_k]$, the loss function is:

$$\mathcal{L}(e_k;\tau_k) = \mathop{\mathbb{E}}_{\psi\sim\eta_\pi}\left[(\tau_k \mathbb{I}_{\psi>e_k} + (1-\tau_k)\mathbb{I}_{\psi\le e_k})\,(\psi - e_k)^2\right] \quad (3)$$

However, the iterative computation required to infer the sample set from expectiles is computationally intensive and sensitive to initial conditions, which can lead to non-representative sample sets.

C. MMD: Deterministic Sample-Set

To address the computational inefficiencies and inaccuracies in ER-DQN, Nguyen et al. [5] proposed a direct sampling approach. They define the discrepancy between two sample-sets using Maximum Mean Discrepancy (MMD):

$$\begin{aligned}\mathcal{L}_{MMD^2}(\{m_i\},\{n_i\};\mathcal{K}) = \frac{1}{M^2}\sum_{i,j}\mathcal{K}(m_i,m_j) + \\ \frac{1}{N^2}\sum_{i,j}\mathcal{K}(n_i,n_j) - \frac{2}{MN}\sum_{i,j}\mathcal{K}(m_i,n_j)\end{aligned} \quad (4)$$

where $\{m_i\}$ and $\{n_i\}$ are two sample-sets and $\mathcal{K}$ is a kernel function measuring sample distances. Although MMD ensures the neural network outputs a deterministic sample-set, it neglects the beneficial stochasticity in learning and exploration, revealing an area for improvement in sampling methodologies.

III. Mixed Weighted Gaussian Algorithm

In the preceding sections, we highlighted the prevalent issue of misinterpreting statistical outputs as samples in DRL algorithms and reviewed two approaches aimed at rectifying this problem, each with its own set of challenges. Motivated by the need for efficiency, accuracy, and stochasticity in the sampling process, we posited that the Gaussian Mixture Model (GMM) could be an ideal representation of the return distribution. Hence, we introduce a new DRL algorithmthe Mixed Weighted Gaussian (MWG) algorithmthat adeptly addresses the sampling issues and effectively resolves the misuse of statistical outputs as samples.

A. GMM in DRL

A GMM approximates a given distribution through a combination of several Gaussian distributions, each characterized by a mean μ, a standard deviation σ, and a weight ω:

$$\eta_\pi(s,a) \approx \sum_{k=1}^{K} \omega_k \mathcal{N}(\mu_k, \sigma_k^2), \text{ where } \sum_{k=1}^{K} \omega_k = 1. \quad (5)$$

Here, K is the number of Gaussian components used. Theoretically, GMM can approximate any distribution precisely with a sufficiently large K.

We design a neural network, denoted as $\mathcal{F}$, to output the GMM parameters. For a state-action pair (s,a), the network $\mathcal{F}$ provides the statistics $\Gamma_\pi(s,a) = [(\mu_1,\sigma_1,\omega_1),\ldots,(\mu_K,\sigma_K,\omega_K)]$ representing the return distribution.

Sampling Operation $\mathscr{I}$

As outlined in study [3], the sampling operation $\mathscr{I}$ maps the statistics $\Gamma_\pi(s,a)$ to a sample-set $\Psi_\pi(s,a)$:

$$\mathscr{I}(\Gamma_\pi(s,a)) \mapsto \Psi_\pi(s,a) = \{\psi_i\}_{i=1}^{M}, \quad (6)$$

where ψ_i denotes a sampled return. The sampling for a GMM is straightforward: for each Gaussian component, we draw $m_k = \omega_k M$ samples, which are then concatenated to form $\Psi_\pi(s,a) = \{\psi_{1,1},\cdots,\psi_{1,m_1},\cdots,\psi_{K,1},\cdots,\psi_{K,m_K}\}$.

Projection Operation Π

Directly determining the likelihood function of a GMM for use as a loss function in neural network updates is not feasible. Instead, we employ the EM algorithm to estimate the GMM parameters from the samples and then update the neural network using the MSE loss. The Jensen-Tsallis Divergence (JTD) has been used in the past to measure the discrepancy between mixture models, but it assumes non-Maxwell-Boltzmann statistics, which do not hold for the stochastic returns in reinforcement learning. Consequently, we do not consider the JTD as a suitable loss function.

We refer to the computation of statistics from a sample-set as the *Projection Operation* Π:

$$\begin{aligned}\Pi(\Psi_\pi(s,a)) \mapsto &\Gamma_\pi(s,a) = \\ &[(\mu_1,\sigma_1,\omega_1),\ldots,(\mu_K,\sigma_K,\omega_K)].\end{aligned} \quad (7)$$

However, to avoid the computational complexity of the traditional EM algorithm, we introduce a *single-step EM method*, detailed in steps 7 to 11 of Algorithm 1.

Our single-step EM method improves upon the standard method by optimizing the initialization of Gaussian components. Since reinforcement learning is iterative by nature, this approach leverages the previous iteration's output from the neural network $\mathcal{F}$ as initial values for the EM algorithm, effectively integrating the EM iteration into the reinforcement learning process. This not only reduces computational demands but also enhances the stability of the learning procedure.

B. Expected Distribution Estimation

Adhering to Equation 1, a distributional Bellman update entails initially estimating the expected distribution. Subsequently, we adapt the neural network parameters by employing this expected distribution as the benchmark in our loss function computation. However, present DRL algorithms do not perform these operations due to two primary constraints: the first is the sheer magnitude of the state space in practical applications, which makes the recurrence of identical state-action pairs exceedingly rare, thus limiting the availability of next state samples. The second is that in many DRL algorithms, the loss function's gradient with respect to the distribution is affine. This affineness indicates that the expected value of the stochastic gradient, when derived from a sample distribution, coincides with the gradient obtained from the expected distribution:

$$\mathbb{E}\left[\frac{\partial \mathcal{L}((B_{R,\gamma})_\# \eta_\pi(S_i',A_i'))}{\partial\theta}\right] = \frac{\partial \mathcal{L}(\mathbb{E}[(B_{R,\gamma})_\# \eta_\pi(S_i',A_i')])}{\partial\theta}.$$

Unfortunately, the EM algorithm used to generate target statistics for GMM is not affine relative to the sample-set, which implies that employing GMM as the distribution representation model leads to biased learning during stochastic gradient updates for the neural network.

As it is impractical to compute the expected distribution by accumulating transition samples from (s,a), we resort to *stochastic update* techniques to incrementally estimate it. In this approach, the current distribution $\eta_\pi(s,a)$ produced by the neural network serves as our initial estimate. By integrating the sample distribution from the Bellman update $(B_{R,\gamma})_\#\eta_\pi(s',a')$ with the current estimate, we refine our estimation of the expected distribution. This refined estimate more accurately reflects the true expected distribution, albeit indirectly. However, directly merging these distributions as probability density functions is not feasible:

$$\eta_\pi^{(t+1)}(s,a) = (1-\beta)\eta_\pi^{(t)}(s,a) + \beta(B_{R,\gamma})_\#\eta_\pi^{(t)}(S',A').$$

Nonetheless, by obtaining sample-sets that represent each distribution, we can combine these sets to approximate the average of the distributions:

$$\begin{aligned}\hat{\Psi}_\pi^{(t+1)}(s,a) &= \Psi_\pi^{(t)}(s,a) \cup [R + \gamma\Psi_\pi^{(t)}(S',A')] \\ &= \{\psi_1^{(t)}(s,a),\ldots,\psi_{M_1}^{(t)}(s,a), \\ &\quad R+\gamma\psi_1^{(t)}(S',A'),\ldots,R+\gamma\psi_{M_2}^{(t)}(S',A')\}.\end{aligned} \quad (8)$$

Here, the ratio $M_1 : M_2$ is defined by $(1-\beta) : \beta$, where β is the proportion of the newly obtained sample-set in the blending process. The value of β modulates the

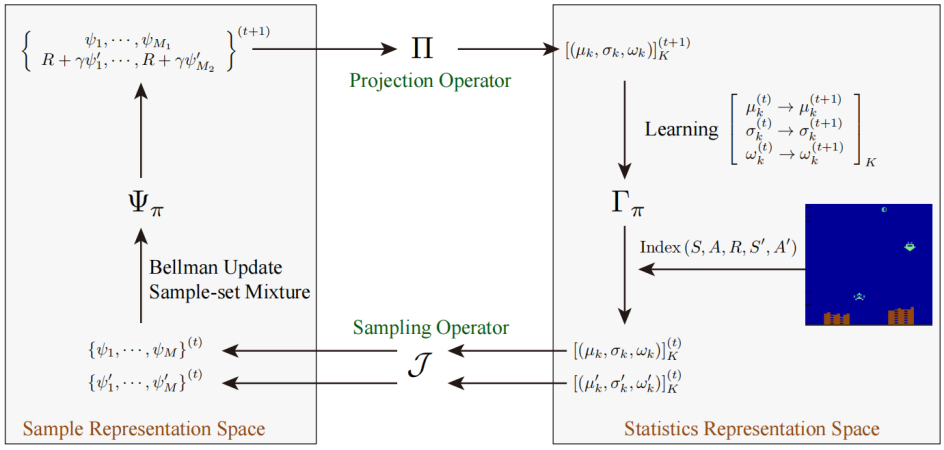

Fig. 2: The Structure of the MWG algorithm

balance between the speed of convergence and the stability of the estimated distribution. The sample-set $\hat{\Psi}_\pi^{(t+1)}(s,a)$ thus serves as an effective surrogate for the expected distribution.

C. Mixed Weighted Gaussian Algorithm

We combine these enhancements to propose a novel Deep Reinforcement Learning (DRL) algorithm, the Mixed Weighted Gaussian (MWG) algorithm, as outlined in Algorithm 1.[1]

Given a transition tuple (s, a, r, s', a'), the MWG algorithm follows these steps:

1) **Indexing**: Generate the statistics $\Gamma_\pi(s,a)$ and $\Gamma_\pi(s',a')$ using the neural network $\mathcal{F}$.
2) **Sampling**: Create the sample sets $\Psi_\pi(s,a)$ and $\Psi_\pi(s',a')$ from the computed statistics through the sampling operation $\mathscr{I}$.
3) **Distributional Bellman Update**: Update each return sample in $\Psi_\pi(s',a')$ via the Bellman equation to obtain a new sample set $\Psi_\pi^B(s',a')$, corresponding to $(B_{R,\gamma})_\# \eta_\pi(s',a')$.
4) **Mixing Sample Sets**: Combine $\Psi_\pi(s,a)$ and $\Psi_\pi^B(s',a')$ based on the mixing coefficient β, producing the blended sample set $\hat{\Psi}_\pi(s,a)$ that approximates the expected distribution $\mathbb{E}[(B_{R,\gamma})_\# \eta_\pi(S',A')]$.
5) **Projection**: Calculate the statistics $\hat{\Gamma}_\pi(s,a)$ from $\hat{\Psi}_\pi(s,a)$ using the Expectation-Maximization (EM) method and sort the components by their means.
6) **Neural Network Update**: Adjust the neural network parameters θ to minimize the Mean Squared Error (MSE) loss.

Figure 2 demonstrate the structure of the MWG algorithm.

D. Analysis of Contraction Mapping properties of MWG

Bellemare et al. [14] have asserted that while the distributional Bellman optimality operator is not a contraction mapping, the existence of an *action gap* in the optimal value function can stabilize the optimal policy after sufficient iterations. This stabilization effectively transforms the operator into the distributional Bellman operator, as stated in Theorem 7.9. For convergence to a unique fixed point, the projection operation Π must preserve the mean, despite approximation errors introduced by Π.

The term *mean-preserving* refers to an operation that maintains the mean of the distribution's statistics equal to that of the sample set prior to projection, thus not affecting the optimal policy. The EM method in the MWG algorithm classifies samples in the sample set and then computes the mean within each class. Since the weighted sum of these means equals the overall sample set's mean, this projection is mean-preserving. Moreover, the neural network's inherent approximation error when estimating the return distribution ensures the presence of an action gap in the value function. Consequently, the projected distributional Bellman optimality operator for MWG will, after enough iterations, become a contraction mapping.

IV. Experiments

To illustrate the performance characteristics of the Mixed Weighted Gaussian (MWG) algorithm, we employ the Dopamine framework [15] for benchmarking against standard algorithms in the Atari 2600 gaming environment [16] under the "Sticky action" setting. This setting, along with parameter configurations, reward shaping functions, and scoring methodology, align with the protocols established by Dopamine, reflecting contemporary best practices within the domain of DRL.

A. Comparative Performance Analysis

We study the effectiveness of the MWG algorithm with established benchmarks, including Deep Q-Network (DQN), Rainbow, Implicit Quantile Networks (IQN), and Mixture Model-based Distributions (MMD). Figure 3 presents the learning curves of these algorithms across a suite of 12 Atari games, revealing the superior performance of MWG.

The mean and median normalized scores aggregated over 59 games serve as pivotal benchmarks for algorithmic assessment. We introduce *Best Count*, the tally of games where

[1]The corresponding project code, resources, and configurations can be accessed at `https://github.com/xiaojianyang820/Dopamine/tree/main`.

Algorithm 1 Mixed Weighted Gaussian Algorithm

Input: a transition sample (s, a, r, s', a')

Index: $\Gamma_\pi(s,a) = [(\mu_1, \sigma_1, \omega_1), \cdots, (\mu_K, \sigma_K, \omega_K)]$; $\Gamma_\pi(s',a') = [(\mu'_1, \sigma'_1, \omega'_1), \cdots, (\mu'_K, \sigma'_K, \omega'_K)]$

Sample: $\Psi_\pi(s,a) = \mathscr{I}(\Gamma_\pi(s,a)) = \{\psi_1, \cdots, \psi_M\}$; $\Psi_\pi(s',a') = \mathscr{I}(\Gamma_\pi(s',a')) = \{\psi'_1, \cdots, \psi'_M\}$

Distributional Bellman Update: $\Psi^B_\pi(s',a') = \{r + \gamma\psi'_1, \cdots, r + \gamma\psi'_M\}$

Sample-set Mix:

$$\hat{\Psi}_\pi(s,a) = \{\psi_1, \cdots, \psi_{M_1}, r + \gamma\psi'_1, \cdots, r + \gamma\psi'_{M_2}\}, M_1 + M_2 = M;\ M_1 : M_2 = (1-\beta) : \beta$$

Project (*Single-Step EM Method*):

Set Initial Parameters: $[\mathcal{N}_1, \cdots, \mathcal{N}_K] = \Gamma_\pi(s,a) = [(\mu_1, \sigma_1, \omega_1), \cdots, (\mu_K, \sigma_K, \omega_K)]$

E Step:

$$\hat{\gamma}_{mk} = \frac{\phi(\psi_m|\mathcal{N}_k)}{\sum_{k=1}^K \phi(\psi_m|\mathcal{N}_k)},\ \forall\psi_m \in \hat{\Psi}_\pi(s,a),\ \forall\mathcal{N}_k \in [\mathcal{N}_1, \cdots, \mathcal{N}_K]$$

M Step:

$$\hat{\mu}_k = \frac{\sum_{m=1}^M \hat{\gamma}_{mk}\psi_m}{\sum_{m=1}^M \hat{\gamma}_{mk}},\ \hat{\sigma}_k^2 = \frac{\sum_{m=1}^M \hat{\gamma}_{mk}(\psi_m - \hat{\mu}_k)^2}{\sum_{m=1}^M \hat{\gamma}_{mk}},\ \hat{\omega}_k = \frac{\sum_{m=1}^M \hat{\gamma}_{mk}}{M},\ k = 1, 2, \cdots, K$$

Sort: $\hat{\Gamma}_\pi(s,a) = [(\hat{\mu}_1, \hat{\sigma}_1, \hat{\omega}_1),\ \cdots,\ (\hat{\mu}_K, \hat{\sigma}_K, \hat{\omega}_K)|\hat{\mu}_1 \leq \cdots \leq \hat{\mu}_K]$

Network Update:

$$\arg\min_\theta \sum_{k=1}^K [(\mu_k - \hat{\mu}_k)^2 + (\sigma_k - \hat{\sigma}_k)^2 + (\omega_k - \hat{\omega}_k)^2]$$

TABLE I: Summaries of scores of the compared algorithms using 5 random seeds

	Mean	Median	Best Count	BTH Count
DQN	2.66	0.67	0	20
Rainbow	5.22	1.48	7	42
IQN	5.56	1.25	5	40
MMD	4.79	1.36	11	38
MWG	8.51	2.19	36	49

an algorithm secures the top score among all contenders, and *BTH Count* (Better Than Human Count), indicative of the number of games where the algorithm outperforms the human baseline. These indicators are crucial for a holistic evaluation of algorithmic prowess, as summarized in Table I.

Table I reveals the escalated challenge of control tasks under the sticky mode. While Rainbow, IQN, and MMD exhibit comparable performance without a clear frontrunner, the MWG algorithm demonstrates a pronounced edge, outstripping its peers across all key statistical measures. Notably, even with a sample size reduced to a mere quarter, MWG surpasses the median score of 2.03 attained by the state-of-the-art model-based algorithm DreamerV2, underscoring the importance of precision in sample selection and the intrinsic randomness within the sample set.

B. Abalation Study

Number of Gaussian Components K

The representational capacity of a Gaussian Mixture Model (GMM) is significantly influenced by the number of Gaussian components. We investigate the impact of this parameter on the MWG algorithm by experimenting with 3, 5, and 7 components, denoted as mwg_3g, mwg_5g, and mwg_7g, respectively. Figure 4 illustrates the learning curves, where the experiments with 3 and 5 components each employed five random seeds, while the 7-component experiment utilized a single seed.

TABLE II: Comparison of 3 and 5 Gaussian components across 59 Atari games with 5 random seeds

	Mean	Median	Best Count	BTH Count
MWG_3g	7.26	2.15	13	49
MWG_5g	8.51	2.19	46	49

In an evaluation across 12 games, it appears that a greater quantity of Gaussian components enhances the learning ability of the algorithm, as evidenced by improved performance across a broader array of games. However, the recorded median scores for 3, 5, and 7 components are 1.56, 1.98, and 1.94, respectively, indicating that the approximation capabilities of a GMM with 5 to 7 components are sufficiently robust for reinforcement learning tasks.

Table II summarizes the comparative performance of using 3 versus 5 Gaussian components across 59 Atari games, each tested with 5 random seeds. The results clearly demonstrate the superiority of the 5-component configuration over the 3-component one. Nevertheless, the findings also suggest that while additional Gaussian components may improve scores in games where the algorithm already performs well, they do not necessarily overcome existing challenges.

Sample-set Mixture Ratio β

The mixing ratio of sample-sets, denoted by β, is a critical factor in the performance of the MWG algorithm. As Figure 5 reveals, the optimal β value is between 0.5 and 0.8, which

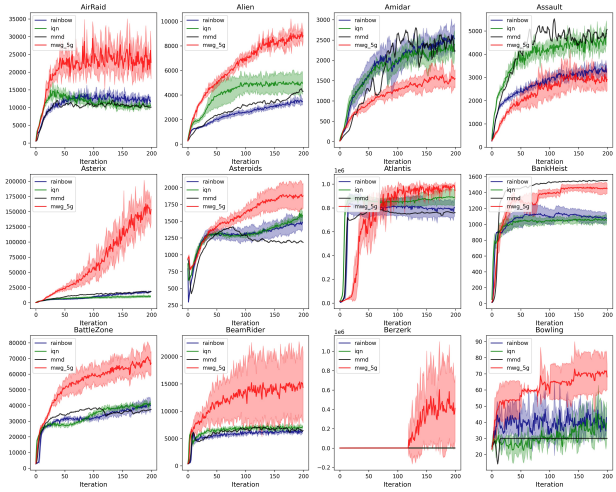

Fig. 3: Learning curves of algorithms on 12 Atari games using 5 random seeds.

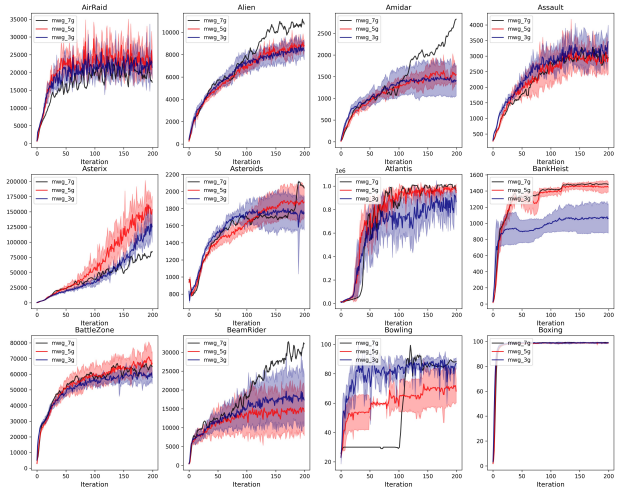

Fig. 4: Impact of varying Gaussian components: 3, 5, and 7 on performance.

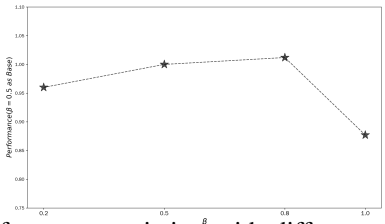

Fig. 5: Performance variation with different mixture ratios β, normalized to the standard MWG algorithm ($\beta = 0.5$)

TABLE III: The Impact of Sample Randomness on algorithm performance using 5 random seeds)

Standard	NR in Action	NR in Learning	NR in Both
4.13	3.65	3.82	3.45

indicates a preference for fresher target sample-sets. This finding contrasts with the minimal update ratios commonly used in stochastic updates, and is attributed to the dynamic nature of the learning targets in RL, where newer targets should be weighted more heavily to prevent early stagnation.

The Randomness of Return Samples

The stochastic nature of reward distribution samples plays a dual role in the learning and action selection processes of DRL algorithms. Stochasticity in action selection enables directed exploration by allowing probabilistic choice among actions with similar expected values. Concurrently, in the learning process, it introduces a variety of target signals that contribute to the robustness of the algorithm. The significance of this randomness is underscored by our experiments on six Atari games, repeated five times, as summarized in Table III. The abbreviation **NR** denotes the absence of randomness in the samples.

REFERENCES

[1] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone, "Risk-constrained reinforcement learning with percentile risk criteria," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6070–6120, 2017.

[2] W. R. Clements, B. Van Delft, B.-M. Robaglia, R. B. Slaoui, and S. Toth, "Estimating risk and uncertainty in deep reinforcement learning," *arXiv preprint arXiv:1905.09638*, 2019.

[3] M. Rowland, R. Dadashi, S. Kumar, R. Munos, M. G. Bellemare, and W. Dabney, "Statistics and samples in distributional reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5528–5536.

[4] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[5] T. T. Nguyen, S. Gupta, and S. Venkatesh, "Distributional reinforcement learning with maximum mean discrepancy," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2020.

[6] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[7] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," *arXiv preprint arXiv:2301.04104*, 2023.

[8] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 449–458.

[9] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.

[10] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[11] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 1096–1105.

[12] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, and T.-Y. Liu, "Fully parameterized quantile function for distributional reinforcement learning," *Advances in neural information processing systems*, vol. 32, 2019.

[13] Y. Choi, K. Lee, and S. Oh, "Distributional deep reinforcement learning with a mixture of gaussians," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 9791–9797.

[14] M. G. Bellemare, W. Dabney, and M. Rowland, *Distributional Reinforcement Learning*. MIT Press, 2023, http://www.distributional-rl.org.

[15] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A Research Framework for Deep Reinforcement Learning," 2018. [Online]. Available: http://arxiv.org/abs/1812.06110

[16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents,"

Journal of Artificial Intelligence Research, vol. 47, pp. 253–279, jun 2013.

[17] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[18] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. Van Hasselt, J. Quan, M. Večerík, *et al.*, "Observe and look further: Achieving consistent performance on atari," *arXiv preprint arXiv:1805.11593*, 2018.