Tackling Uncertainty in Reinforcement Learning: A Dual Variational Inference Approach for Task and State Estimation

Zhidong Yang
Ohio State University，Computer Science & Engineering，
Columbus, OH 43210, USA
E-mail: yang.6390@buckeyemail.osu.edu

Zongxin Yao
Xi'an Technological University,
Computer Science and Engineering,
Xi'an, 710021, China
E-mail: yaozongxin@xatu.edu.cn

Haoyu Liu
Xi'an Technological University
Computer Science and Engineering
Xi'an, 710021, China
E-mail: 1121227983@qq.com

Hongge Yao
Xi'an Technological University
Computer Science and Engineering
Xi'an, 710021, China
E-mail: yaohongge@xatu.edu.cn

Abstract —**Uncertainty in decision-making processes presents a critical challenge for autonomous agents, often leading to suboptimal or erroneous policies. This paper addresses two prevalent yet distinct types of uncertainty that significantly degrade agent performance: fuzzy uncertainty, stemming from ambiguous task boundaries, and gray uncertainty, arising from noisy or incomplete state observations. To tackle these challenges, we propose the Dual-Task-State Inference (DTS-Infer) method, a novel framework that leverages variational inference within an off-policy reinforcement learning structure. DTS-Infer utilizes a dual-network architecture to explicitly disentangle and resolve these uncertainties: (1) a task inference network learns a latent distribution over tasks from historical data to disambiguate task goals, thereby solving the fuzzy uncertainty problem ; and (2) a state inference network captures robust latent features of the current state to overcome corrupted sensory input, thus addressing gray uncertainty. Extensive experiments on continuous control benchmarks demonstrate that DTS-Infer significantly outperforms state-of-the-art algorithms. For instance, in the Half-Cheetah-Fwd-Back environment, DTS-Infer achieved a final average reward of 1612.61, representing an 18.9% improvement over the PEARL algorithm. Furthermore, ablation studies confirmed that our inference modules contribute to an 80% increase in average reward over a standard TD3 baseline, highlighting the method's effectiveness in enhancing the robustness and adaptability of intelligent agents.**

Keywords-Reinforcement Learning; Meta Learning; Fuzzy Uncertainty Information; Gray Uncertainty Information; Variational Inference

I. INTRODUCTION

During decision-making processes, intelligent agents are frequently influenced by uncertain information, leading to inaccurate decisions [1]. Uncertain information is categorized into random information, fuzzy information, unknown information, gray information, and pan-gray information. This work focuses on two prevalent yet distinct types of uncertainty that significantly degrade agent performance. The first, gray uncertainty, pertains to incomplete or corrupted information at the state level. In RL, this typically manifests as noisy sensor readings or partial observability, where the agent receives a distorted representation of the true environment state, leading to flawed state estimation. The second, fuzzy uncertainty, relates to ambiguity at the task level. This occurs when the objectives or environmental dynamics are ill-defined, making it difficult for the agent to infer the current task's requirements from immediate observations. Such ambiguity prevents the agent from forming a coherent long-term strategy, as it cannot clearly distinguish between different, potentially

conflicting, goals. Figure 1 illustrates these two challenges.

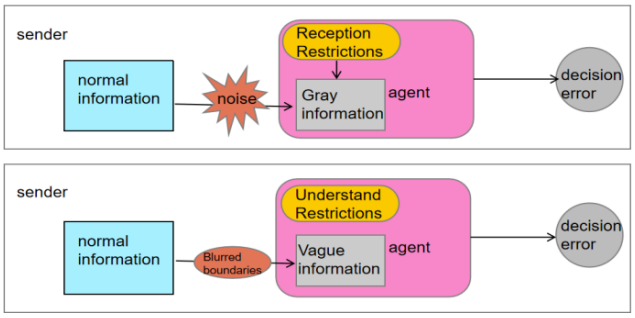


Figure 1. Gray Uncertainty Information and Fuzzy Uncertainty Information

Gray uncertainty fundamentally involves the absence of certain feature information. Therefore, we interpret erroneous decisions made by agents when confronting gray uncertainty as a failure to express unknown information. Fuzzy uncertainty arises from unclear information boundaries, where agents cannot comprehend the information due to their own capability limitations. Erroneous decisions made by agents when facing fuzzy uncertainty can be understood as a lack of corresponding information needed to describe events more accurately.

Our core idea is to treat these uncertainties as inference problems within a probabilistic framework. We propose that the ambiguity from gray and fuzzy uncertainty can be resolved by learning low-dimensional latent representations of states and tasks, respectively. To this end, we leverage the power of variational inference to model the distributions over these latent variables. The primary contributions of this work are twofold: (1) to counter gray uncertainty, we introduce a state inference network that encodes the noisy, high-dimensional state observations into a probabilistic latent variable. This process effectively filters noise and captures the essential underlying state features, providing the policy with a robust state representation. (2) To address fuzzy uncertainty, we design a task inference network that aggregates historical trajectories (context) to infer a latent variable representing the current task. This allows the agent to disambiguate goals and adapt its policy accordingly, a principle central to meta-learning.

Finally, based on the above innovations, this paper proposes a method to address uncertainty issues: the Dual Task Inference (DTS-Infer) approach based on variational autoencoders. We compare DTS-Infer with state-of-the-art algorithms in the MOJUCO environment and conduct systematic ablation experiments to evaluate the necessity of each component. Experimental results demonstrate that our method outperforms existing approaches in both convergence and accuracy.

The following sections comprise the main body (Sections 2 – 6) and appendices. Section 2 reviews existing reinforcement learning research, summarizing the strengths and limitations of various approaches. Section 3 outlines how variational inference is employed to address uncertainty. Section 4 details the algorithm’s operational principles and training methodology. Section 5 presents experimental results, including the overall training workflow, comparative experiments, ablation studies, and noise adversarial experiments. Section 6 concludes with a summary and review.

II. Related Work

Meta-learning The goal of meta-learning is to learn useful inductive biases from relevant tasks, which can then be leveraged for subsequent new tasks. Meta-learning enables parameters to continuously adapt to different tasks, utilizing experience gained from past tasks to learn new ones. When learning new tasks, meta-learning can be viewed as a special case of a Partially Observable Markov Decision Process (POMDP). For the agent, the new task constitutes the unobserved portion. The agent must continuously gather information and adjust itself to adapt to the new task. The RL2 algorithm proposed by Duan et al. [18] in 2016 utilizes the hidden state of an RNN network as memory and experience, propagating it across each episode to enable rapid adaptation to new tasks. Finn et al. [20] introduced the gradient-based meta-learning MAML model-agnostic algorithm in 2017, focusing on policy learning. Gupta et al. [21] proposed the MAESN algorithm in 2018, incorporating noisy exploration into MAML to enhance sample

exploration efficiency. Mishra [21], Wang [23], and others adapted to new tasks by aggregating experience into the latent representations relied upon by the policy network. Such methods are categorized as context-based meta-reinforcement learning algorithms. The PEARL algorithm proposed by Rakelly [24] et al. in 2019 belongs to this context-based category, employing an additional network for task inference to enhance algorithmic performance. This paper employs meta-learning algorithms to enable agents to rapidly adapt to new environments. It collects sufficient samples for both inference networks, accelerating and refining the distribution constructed by the inference networks to address fuzzy uncertainty.

Variational Inference Wingate et al. [25] proposed applying variational inference to the policy exploration problem in reinforcement learning in 2011. This method uses variational inference to estimate prior information about the policy distribution, enabling rapid policy exploration in complex environments. Florensa et al. [26] proposed representing policies as probability distributions in 2017 and employing variational inference to optimize distribution parameters, addressing tasks with multiple hierarchical levels. The VIMCO algorithm proposed by Mnih et al. [27] in 2016 employs variational inference for reinforcement learning, enabling it to handle high-variance unbiased estimators and significantly improving robustness against randomness. The VPG [28] algorithm introduced in 2017 utilizes variational inference to optimize policy networks; however, due to the inherent complexity of POMDPs, VPG remains applicable only to small-scale problems. The MAESN algorithm proposed by Gupta et al. [21] in 2018 enhances agent exploration capabilities by generating latent states for stochastic exploration via variational inference. The PEARL algorithm introduced by Rakelly et al. [24] in 2019 improves sampling efficiency and adaptability to new tasks by applying variational inference to derive historical experience. This paper collects task experience and performs variational inference on tasks to obtain a latent space of task information. Simultaneously, variational inference is applied to the agent's current state to enhance the model's task adaptability. By obtaining a latent space of the agent's state features and deriving a probability distribution over the agent's state, the decision-making capability of the decision network is improved, reducing the gray uncertainty caused by information transmission processes.

Deterministic Policy Gradient Algorithm Deterministic policy-based reinforcement learning features high algorithmic stability, high data efficiency, and broad applicability. In 2016, Lillicrap et al. [29] proposed the DDPG algorithm, which combines policy gradient methods [30] with Q-learning [31] and incorporates Gaussian noise for environmental exploration, demonstrating strong performance across continuous control tasks. The Twin Delayed DDPG (TD3) [32] algorithm, proposed by Fujimoto et al. in 2018 as an enhancement to DDPG, improves stability by incorporating a twin Q-network and a delayed update mechanism. The D4PG algorithm, proposed by Barth-Maron et al. in 2018, builds upon TD3 by incorporating distributed experience revisitation and distributed priority sampling techniques, thereby improving the algorithm's data efficiency and parallelizability. This paper employs the Deterministic Policy Gradient algorithm for agent control, extending TD3 by integrating an inference network to construct our model, DTS-Infer.

III. Variational Inference of Uncertainty Information

A. Fuzzy Uncertainty Information Task Inference

Due to unclear boundaries and insufficient cognitive capabilities, intelligent agents cannot make correct actions based on current information when faced with fuzzy uncertainty. This manifests as follows: when an intelligent agent encounters a crossroads scenario with ambiguous boundary information and limited decision-making capacity, it cannot determine whether to turn left or right. Consequently, it randomly selects an action, significantly increasing the likelihood of erroneous decisions. In response to this scenario, we contend that this fuzzy uncertainty stems from neglecting transitional information and lacking a means to

express it. Therefore, we introduce a variable to describe this transitional information, as illustrated in Figure 3. We construct a deep neural network using extensive historical samples to fit this variable. Since its content is inferred from a large number of task samples, this network is termed the Task Inference Network, as shown in Figure 2.

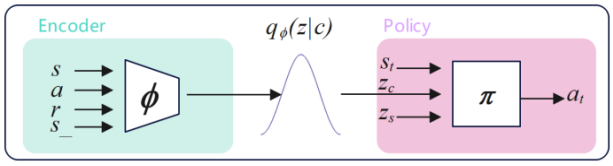


Figure 2. Task Inference Network

In the task inference network, we trained an Encoder1 parameterized by ϕ to fit the task distribution corresponding to the current environment. By feeding historical experiences into the network, it generates a distribution over the task space, where c represents all collected experiences. Drawing from a large number of task samples obtained from past tasks, it continuously refines the distribution of the task space. The variables generated by the network represent information about the current task, also known as task latent variables. These are used as input variables during decision-making, enabling the agent to understand the current task's context. When faced with ambiguous or uncertain information, the agent gains a general target direction and selects actions toward that objective. Furthermore, by leveraging historical experience from previous tasks as prior knowledge, the agent can make informed decisions for new task samples based on past patterns when encountering novel tasks.

After extensive task training, we consider the task distribution generated by the task inference network sufficiently refined. Upon training completion, its parameters ϕ are fixed. For new tasks, Encoder1 directly infers by collecting samples from the current task. For instance, when confronted with an environment involving eating an apple, the agent can leverage previously learned experience of eating a pear—the action most analogous to eating an apple—to execute an action. Furthermore, when presented with identical state information but differing task environments, providing the agent with a specific objective resolves ambiguity and uncertainty. To ensure the generated distribution approximates the task's true posterior distribution as closely as possible, we construct the variational lower bound as follows:

$$ELBO = E_T[E_{z\sim q_\phi(z|c^T)}[R(T,z) + D_{KL}\left(q_\phi\left(z|c^T\right)\|p(z)\right)]] \quad (1)$$

Here, p (z) denotes the unit Gaussian prior probability of Z. In generating latent variables for task inference, the network's input consists of samples collected by the agent, while its output represents the inference for the current task. The loss function for updating parameter ϕ is defined as:

$$Loss\_Encoderl = \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\varphi, p\right) + Loss\_Critic \quad (2)$$

The Loss_Critic in the loss function serves as a constraint term for generating latent variables, using the score assigned by the value network to the current action to measure the accuracy of the inferred action a.

B. Inference of Gray Uncertainty Information States

Gray uncertainty information can be understood as the state information used by an intelligent agent for decision-making, which is affected by noise or sensor failures leading to missing state information. This results in compromised environmental feature information within the state data. For example, when navigating an intersection scenario, an agent receiving normal information would execute a right turn. However, when confronted with noisy information, the incomplete data may cause the agent to execute a straight or left turn instead, leading to erroneous decisions. In such scenarios, we contend that this gray uncertainty arises from neglecting incomplete information and lacking a representation for missing unknowns. Therefore, we introduce a variable to describe this missing unknown information. Since the content of this variable is inferred from a large number of state samples, we refer to this network as a State Inference Network.

Based on this, we construct the State Inference Network as shown in Figure 3 below.

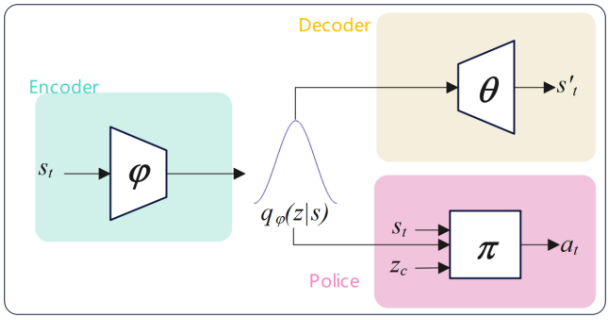


Figure 3. State Inference Network

In the state inference network, a parameterized neural network Encoder 2 is constructed to fit the distribution corresponding to the agent's state space S. By inputting the agent's state information St into the network, a complete distribution of the state space is established. Uncertainty information is also treated as a random variable within the current posterior probability, generating the distribution of the state space. This provides the agent with more feature information about the current state, or more specifically, feature information about the current state in a low-dimensional space. When faced with gray uncertainty and missing state information, variables extracted from the constructed state distribution still contain additional information that can help the agent understand the current true state.

To ensure the accuracy of the trained state distribution, we constructed a decoder whose input is sampled from the posterior distribution obtained by the encoder, and whose output represents the corresponding intelligent weight reconstruction state features. We aim to reconstruct the specific features of the original state based on the sampled data, using these as constraints for generating latent variables to evaluate the accuracy of the generated results.

The generation process depicted in the figure involves compressing state information from high to low dimensions while extracting feature information. After constructing a complete sample distribution, data points that appear random in high-dimensional space become discernible in low-dimensional space. Specifically, consider an agent navigating an intersection environment. The range of latent variable values generated is (0-2), where (0~0.5) indicates the agent should turn left at that data point, (0.5~1) represents proceeding straight, (1~1.5) signifies turning right, and (1.5~2.0) denotes waiting at the intersection. Therefore, incorporating this as an input variable for the policy network enables the agent to discern more feature information, thereby enhancing its decision-making capabilities.

To ensure the generated distribution approximates the true posterior distribution of the state space as closely as possible, we construct the variational lower bound as follows:

$$ELBO = E_T[E_{Zs\sim q_\phi(z_x|s)}[R(T,z_s)+D_{KL}\left(q_\phi\left(z_s|s\right)\|p\left(z_s\right)\right)]] \quad (3)$$

The loss function for the training state inference network is set as follows:

$$Loss\_Encoder2 = \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\varphi, p\right) - \frac{1}{n}\sum_{i=1}^{n}\log p_{\phi'}\left(s_i|z_{si}\right) \quad (4)$$

C. Decision Networks

We utilize the two variables obtained from the inference network to make decisions. By incorporating the current state information as input, we construct a decision network as shown in Figure 4 below. For the decision network, knowing the current state informations itself, possessing latent feature information about the current state, and understanding the current task situation collectively enhance its decision-making capability and ability to handle two types of uncertainty information.

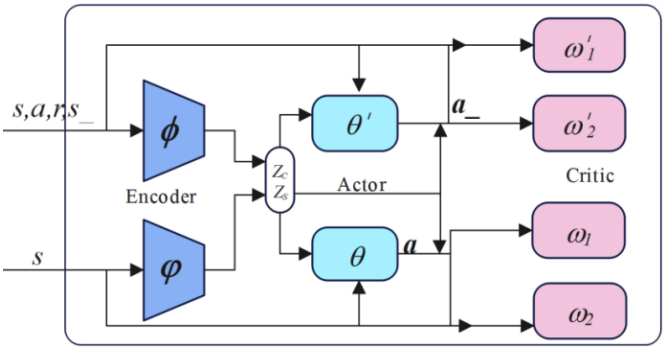


Figure 4. Decision Network Structure

The decision network comprises nine network models: Actor Network, Critic1 Network, Critic2 Network, Target_Actor Network, Target_Critic1 Network, Target_Critic2 Network, Task Inference Network Encoder1, State Inference Network Encoder2, and Decoder.

The Actor network is used to fit the policy network, taking the current state as input and outputting specific actions. During action selection, the criteria incorporate information, significantly reducing the risk of erroneous decisions by the agent. Critic1 and Critic2 are value networks, taking as input and outputting the evaluation value Q. When performing value network evaluations, the action is selected under the guidance of, lowering the error decision rate and making the Q-values output by the Critic more stable.

Encoder1 is the task inference network, while Encoder2 and the decoder form the state inference network. The task inference network takes the loss function of the critic network as a constraint to construct a posteriori distribution based on historical experience, outputting latent variables containing current task feature information. Using historical experience as a benchmark, it assists the agent in making correct decisions when faced with ambiguous and uncertain information. The state inference network takes the agent's current state s as input and outputs latent variables containing the agent's current state feature information. This is then fed into the decoder network, which outputs the agent's reconstructed state. Using the reconstruction error between the reconstructed state and the original state as a constraint, it constructs the distribution of the agent's state space. By benchmarking against the progressively refined state distribution from historical tasks, the agent can still obtain feature information about its current state even when affected by noise or when state information is incomplete due to sensor damage, thereby aiding correct decision-making.

When calculating the target value y, the addition of inputs to the Actor and Critic networks reduces the risk of erroneous actions and stabilizes the Critic network's scoring. Building upon this, we use the minimum value from the outputs of the two Critic networks to compute y, thereby mitigating overestimation issues. The formula is:

To address the high variance and inaccuracy in target estimation inherent in deterministic policy algorithms, we incorporate target policy smoothing regularization. This method emphasizes that similar behaviors should carry similar values. Specifically, it smooths the estimated target value by computing it based on the surrounding area around the target action.

In practice, the expected action is approximated by introducing a small amount of random noise to the target action and averaging across mini-batches.

Considering that the actor network updates by maximizing cumulative expected return, it relies on the critic network. If the critic network is unstable, the actor network will also exhibit oscillations. Therefore, the actor network updates with a delay, meaning it updates only after the critic network has updated multiple times.

IV. Network Framework and Algorithm Flow

A. Network Framework

Based on the two inference networks described above, this paper constructs a model for addressing uncertain information on top of traditional reinforcement learning. This model employs a dual inference approach for task and state based on variational autoencoders, referred to as DTS-Infer. The model comprises three components: ① task inference, ② state inference, and ③ decision-making. The network architecture of the model is illustrated in Figure 5 below.

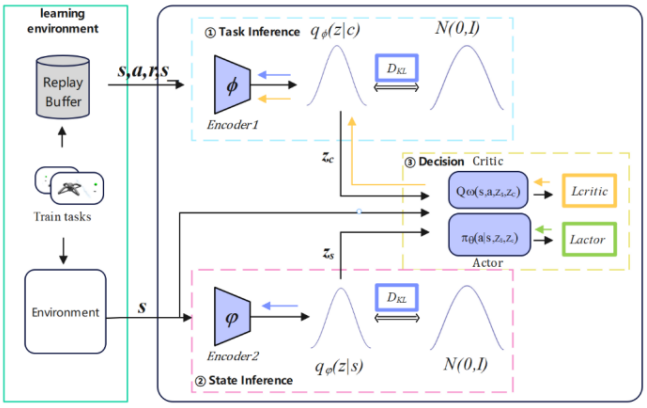


Figure 5. Model Network Architecture

As shown in Figure 5, the task inference network takes samples obtained during the task as input and outputs information relevant to the current task. The state inference network takes the agent's current state S as input and outputs latent features representing the current state. The decision network utilizes the outputs from both inference networks, along with the agent's current state information S, as input to guide the agent in selecting actions.

B. Network Training

The DTS-Infer algorithm comprises pre-training and holistic task training components. During pre-training, a meta-learning approach designs rules for internal and external parameters. Internal network training and external parameter updates alternate at a fixed frequency. The internal network sequentially learns distinct internal parameters by tackling various basic tasks, while external parameter updates optimize different parameters to obtain initial parameters with strong environmental adaptability. During holistic task training, the DTS-Infer algorithm converges rapidly with minimal training sessions across diverse test environments, acquiring optimal action strategies. Moreover, meta-learning inherently possesses the ability to learn how to learn. By training across numerous test tasks, it accumulates relevant experience from past tasks. When confronted with ambiguous or uncertain information, it can guide the agent's exploration based on this learned historical experience, thereby enhancing learning efficiency and reducing the risk of erroneous decisions.

Inner loop the uncertainty module and reinforcement learning module together form the inner loop of the algorithm, as shown in modules ①②③ in the figure above. Within this inner loop, the algorithm focuses on learning the decision-making capabilities of the actor in reinforcement learning, the evaluation criteria of the critic, and the generation and inference capabilities of the inference network under the current task.

The specific workflow is as follows: The environment first initializes the current state s. The state inference network generates latent variables based on the current state s, while the task inference network generates latent variables based on the samples. The actor network processes s and the latent variables as input to generate an action a. After the agent executes action a, the environment provides a reward r and the next state s_2. This information is stored in the experience buffer. Once the buffer is full, N transitions are randomly sampled from it to update the actor, critic, and inference networks, continuing until the algorithm converges.

Outer loop Furthermore, the DTS-Infer algorithm outperforms the unassisted PEARL in model

$$\theta_{meta} = (1-n\tau)\theta_{meta} + \tau(\theta_{T1} + \theta_{T2} + \cdots + \theta_{Tn}) \quad (5)$$

$$w_{meta} = (1-n\tau)w_{meta} + \tau(w_{T1} + w_{T2} + \cdots + w_{Tn}) \quad (6)$$

$$\phi_{meta} = (1-n\tau)\phi_{meta} + \tau(\phi_{T1} + \phi_{T2} + \cdots + \phi_{Tn}) \quad (7)$$

$$\varphi_{meta} = (1-n\tau)\varphi_{meta} + \tau(\varphi_{T1} + \varphi_{T2} + \cdots + \varphi_{Tn}) \quad (8)$$

θ_{meta} represents the meta-parameters of the Actor network, w_{meta} denotes the initial parameters of the Critic network, ϕ_{meta} signifies the initial parameters of the state inference network, and φ_{meta} indicates the initial parameters of the state inference network. n denotes the total number of tasks completed during training. τ is a constant controlling the update rate of hyperparameters. $\theta_{T_j}, \omega_{T_j}, \phi_{T_j}, \varphi_{T_j}$ represents the parameters of the three neural networks in task .After completing all tasks and training, $\theta_{meta}, \omega_{meta}, \phi_{meta}, \varphi_{meta}$ is obtained as the initialization parameters.

Taking meta-task T1 as an example, after DTS-Infer completes its internal update, its initial parameters are also updated. Each time a task is switched, the initial parameters are assigned to the network parameters of the new task, serving as the **initialization** parameters for executing the new task T2. Once all tasks have been traversed, the external initial parameters at this point become the final initialization parameters adapted to the environment.

C. Algorithm Pseudocode

Algorithm Flowchart

Required: Task set $T=\{T_1,T_2,\cdots,T_n\}$

Required: Hyperparameters τ, $l_{actor},l_{critic},l_{vae}$

Randomly initialize actor, critic, and inference network parameters $\theta,\omega,\phi,\varphi$

- Initialize Target network parameters $\theta'\leftarrow\theta,w'\leftarrow w$
- Initialize meta-parameters $\theta_{meta}\leftarrow\theta,w_{meta}\leftarrow w,\phi_{meta}\leftarrow\phi,\varphi_{meta}\leftarrow\varphi$
- for in Task T_j, $j=1,2\ldots,n$ do
- for episode=1,M do
- Initialize Buffer B_{T_j}, Buffer B_C
- Initialization noise N1,N2
- Receive initial state s_1
- for t=1,T do
- State inference network encodes state to generate state z_{st}
- Task Inference Network for Sample Encoding Generates z_{ct}
- Actor make decisions $a_t=\pi_\theta(s_t,z_{ct},z_{st})+N1_t$
- Execute action a_t, and the environment provides reward r_t and the next state S_{t+1}
- Store $(s_t,z_{ct},z_{st},r_t,S_{t+1})$ in the experience pool B_{T_j},
- Store (s_t,a_t,r_t,s_{t+1}) in the experience pool B_C
- Sample N empirical samples $(s_t,z_{ct},z_{st}a_t,r_t,S_{t+1})$ from B_{T_j}.
- Use the Target_Critic network to compute y, and use the Critic network to compute q_t
- Compute $Loss\_Critic=\frac{1}{n}\sum_i(y-q_t)^2$,update the parameters of the Critic network w_i
- Compute $Loss\_Actor=\frac{1}{N}\sum_i q_t$, update the parameters of the Actor network θ
- Computing State Inference

$$Loss\_Encoder2=\frac{1}{N}\sum_i(KL+\mathrm{R}_e\ con)$$

- Compute task inference for

$$Loss\_Encoder1=\frac{1}{N}\sum_i(Loss\_critic)$$

- Update target network parameters $\theta'=\tau\theta+(1-\tau)\theta',w'=\tau w+(1-\tau)w'$,
- End for
- End for
- Set task parameters:
- $\theta'_{T_j}=\theta,w_{T_j}=w,\phi_{T_j}=\phi$
- Update meta parameters:
- $\theta_{meta}=(1-n\tau)\theta_{meta}+\tau(\theta'_{T1}+\theta'_{T2}+\cdots+\theta'_{Tn})$
- $w_{meta}=(1-n\tau)w_{meta}+\tau(w'_{T1}+w'_{T2}+\cdots+w'_{Tn})$
- $\phi_{meta}=(1-n\tau)\phi_{meta}+\tau(\phi_{T1}+\phi_{T2}+\cdots+\phi_{Tn})$
- $\theta\leftarrow\theta_{meta},w\leftarrow w_{meta},\phi\leftarrow\phi_{meta},\varphi\leftarrow\varphi_{meta}$
- $\theta'\leftarrow\theta,w'\leftarrow w$
- End for

V. EXPERIMENTS

To evaluate the performance of the DTS-Infer algorithm proposed in this paper, we designed the experimental content as follows. First, in Section 5.1, we compared the proposed algorithm with previous reinforcement learning algorithms in terms of algorithm performance. In Section 5.2, we assessed the superiority of the variables in our algorithm through ablation experiments. Finally, in Section 5.3, we conducted noise resistance experiments to verify the performance of DTS-Infer when dealing with uncertain information. All experiments were implemented based on Pytorch and NVIDIA CUDA, using an RTX 3060 GPU with 16GB of video memory and a CPU running at 14.4 GHz.

A. Performance Comparison Experiment

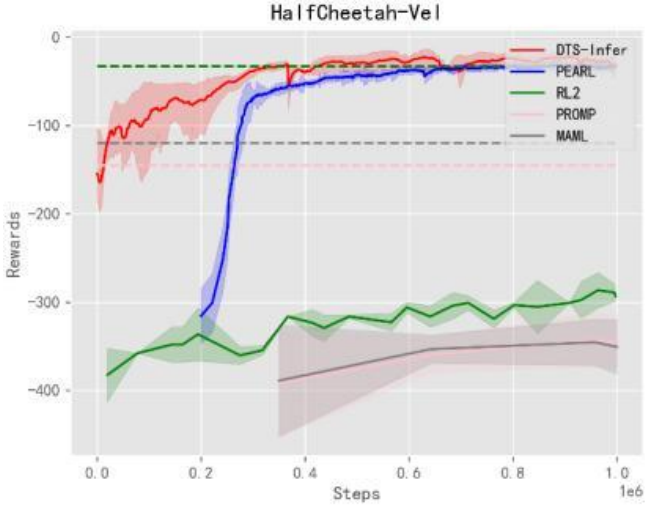
Experimental Environment: To validate the performance of the DTS-Infer algorithm, this paper conducts evaluations across three continuous control environments focused on robotic motion, with experiments performed under the MuJoCo simulator within the Gym environment [34]. These tasks include HalfCheetah-Fwd-Back (controlling robot walking direction), Ant-Goal (controlling

robot movement to reach different target positions), and HalfCheetah-Vel (controlling robot target velocity). These test tasks were introduced by Finn et al. [20] and Rothfuss et al. [35].

In experiments controlling target velocity, the reward equals the absolute value of the difference between the robot's current speed and the target speed. In experiments controlling target walking direction, the reward depends on the speed of the robot's movement toward the target direction. In tasks involving reaching different target positions, the reward depends on the speed of arrival at the target location. All tasks in training have a fixed step size of 200.

Furthermore, comparisons were conducted with gradient-based meta-reinforcement learning algorithms such as MAML [20] and ProMP [35], black-box neural network-based RL2 [18], and task-inference-based PEARL [24]. Results for each algorithm were averaged across three random seeds. The comparative experimental results are shown in Figure 6 below. The horizontal axis represents the number of training steps in millions, while the vertical axis shows the average reward obtained by the model in the meta-test task. The horizontal dashed line indicates the average reward achievable by the corresponding algorithm after final convergence.

Results:



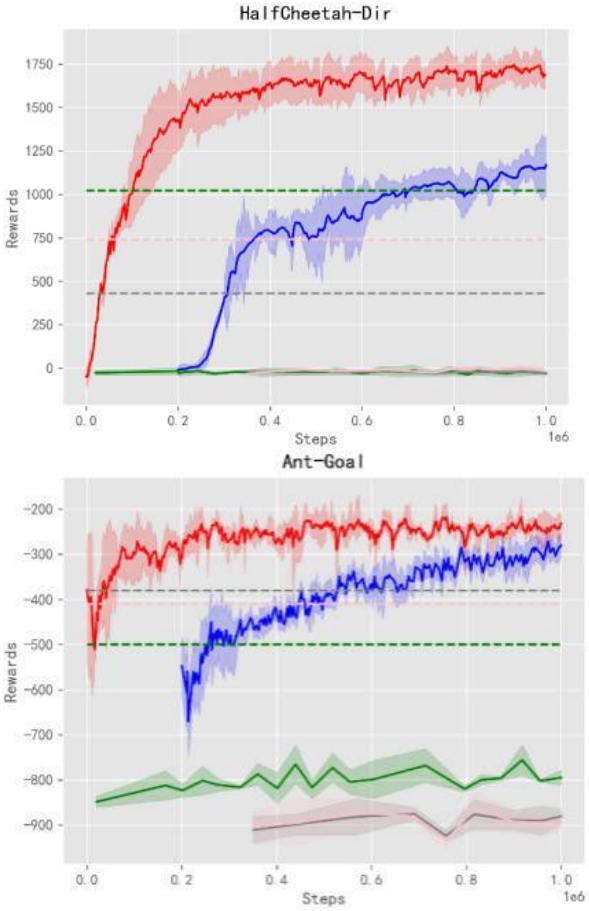


Figure 6. Algorithm Comparison Experiment Diagram Under Different Environments

Compared to the experimental results in Figure 6, both DTS-Infer and PEARL require millions of training steps to converge because they provide agents with more task information than RL2, MAML, and ProMP. Moreover, their converged rewards significantly exceed those of other algorithms. Furthermore, the DTS-Infer algorithm outperforms the unassisted PEARL in model performance.

Compared to PEARL with task inference, DTS-Infer demonstrate significant improvements in convergence efficiency and algorithmic performance in the Half-Cheetah-Fwd-Back and Half-Cheetah-Vel environments. This may stem from the environment's inherently low-dimensional action and state spaces, resulting in fewer inference network parameters. The task-informed outputs generated by DTS-Infer adapt more readily to new tasks, enabling agents to rapidly acclimate to environments. Furthermore,

DTS-Infer incorporates auxiliary features from the state hidden layer, providing agents with richer feature information that enhances algorithmic performance. In the ANT-Goal environment, DTS-Infer converges notably faster than PEARL. However, the convergence curve shows higher variance during early training. This may stem from the fact that in such environments, the agent's objective is to rapidly reach a designated target point, and the target points for each task differ. For instance, in extreme cases, the target locations for two consecutive tasks may lie diagonally opposite on the map. Consequently, the agent's previously learned experience becomes largely ineffective, necessitating the re-exploration of the target.

TABLE I. COMPARATIVE TEST RESULTS

Environment	Model	Convergence Steps	Average Reward
Half-Cheetah-Vel	DTS-Infer	400000+	-21.45
	PEARL	600000+	-35.76
Half-Cheetah-Fwd-Back	DTS-Infer	400000+	1612.61
	PEARL	1000000+	1356.40
Ant-Goal	DTS-Infer	1000000+	-193.36
	PEARL	1000000+	-292.69

The average reward in Table 1 refers to the mean reward obtained by the model during the training task after one million training steps. As shown in Table 1, DTS-Infer outperform PEARL in training efficiency, model convergence speed, and final performance due to its auxiliary mechanisms.

During the evaluation of our algorithm, we set up a series of 2D navigation tasks requiring the agent to move to different target locations. The target was set as a random coordinate point within the range [-1, 1], while the agent's state represented its current 2D position. The task reward was defined as the negative square of the distance between the agent and the target. All tasks had a fixed step size of 200. Comparisons were made with the MAML and PEARL algorithms. Results for each algorithm were averaged across three random seeds. Experimental results are shown in Figure 7. The left and middle panels display the number of navigation trajectories collected during testing on the horizontal axis and the average reward earned by the model on the vertical axis. The right panel illustrates the performance of DTS-Infer versus PEARL after collecting 30 trajectories across three distinct tasks.

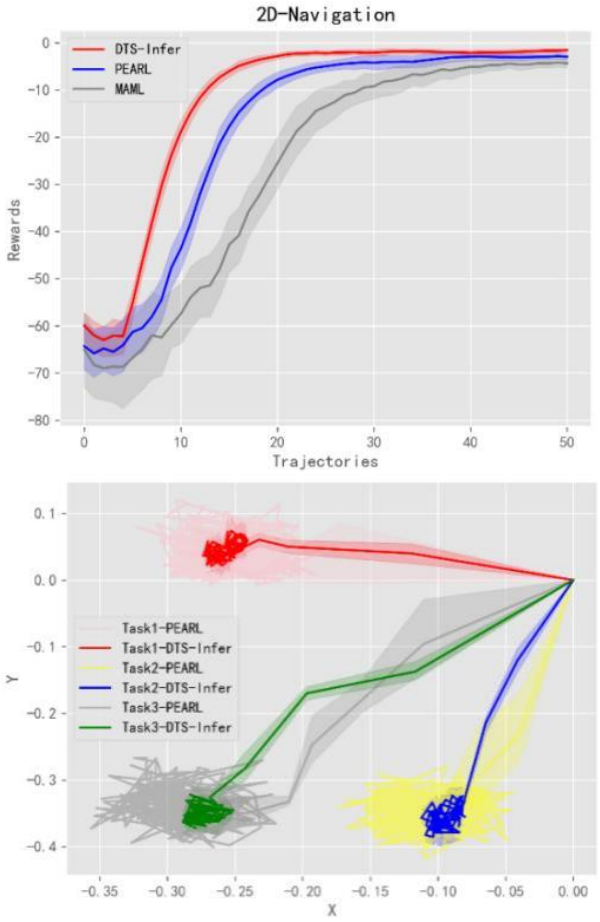


Figure 7. Navigation Comparison Experiment

Results: As shown in the curve diagram in the left-center of Figure 7, both DTS-Infer and PEARL algorithms demonstrate significantly faster exploration capabilities than MAML. With the assistance of both algorithms, adaptation to the task can begin within 10 collected trajectories. This indicates that the task information provided enhances the agent's ability to adapt to new environments. In terms of convergence performance, DTS-Infer slightly outperform PEARL, though the overall difference is minimal. This is because both the state space and task space of the navigation task are relatively uncomplicated. Both PEARL and DTS-Infer effectively guide the agent to rapidly understand the current objective task and enhance its exploration capabilities, resulting in comparable exploration efficiency.

As clearly demonstrated by comparing the task trajectories in the right panel of Figure 7, the path generated by the DTS-Infer algorithm becomes significantly denser as it approaches the target point. This is where its effectiveness becomes evident: by supplementing the task information provided to the agent with hidden layer insights, it genuinely assists the agent in making more optimal decisions. When evaluating the agent's trajectory in terms of covered area, the DTS-Infer algorithm's coverage area is demonstrably more than twice as large as that of PEARL.

B. ZC and ZS Ablation Experiments

In this section, to better validate the impact of key components within the DTS-Infer algorithm on its overall performance, we conducted experiments in the HalfCheetah-Vel environment. Using the traditional TD3 algorithm as the baseline, we progressively incorporated additional components while observing the convergence rate and model performance. The experimental results are shown in Figure 8.

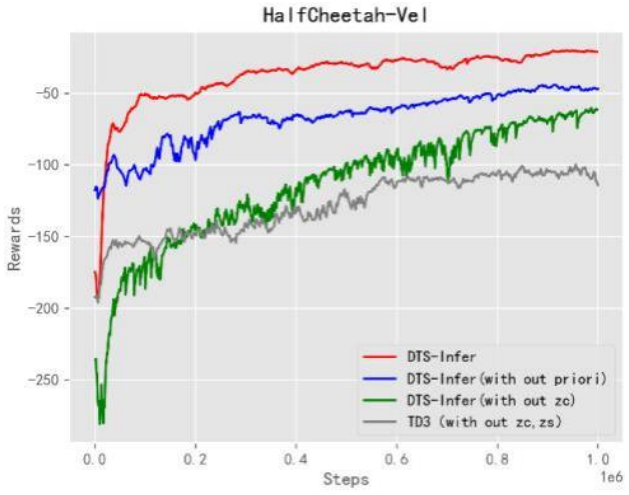


Figure 8. Ablation Experiment

Compared to the experimental results in Figure 8, DTS-Infer demonstrate clear superiority over other algorithms in terms of convergence speed and final model performance. This is because the DTS-Infer algorithm incorporates both and, while also leveraging comprehensive prior knowledge provided by from a large number of historical tasks. Among algorithms lacking prior knowledge, the DTS-Infer (with out priori) variant—which incorporates both—achieves the best final performance. However, this algorithm exhibits significant fluctuations before 200,000 steps. This occurs because the task inference network lacks sufficient prior knowledge, and the state inference network lacks a complete distribution over the sample space. It must learn incrementally through interactions with the environment. The DTS-Infer (without) algorithm, which incorporates only, exhibits significant volatility during early convergence. This occurs because it lacks, preventing the agent from making task inferences when adapting to new environments and hindering rapid identification of task objectives in the initial experimental phase. Furthermore, the training curve indicates that the DTS-Infer (with out) algorithm starts at a relatively low baseline during initial training. However, after forming the state space distribution beyond 200,000 steps, its convergence rate significantly accelerates, surpassing the TD3 algorithm. The TD3 algorithm without the feature exhibits the worst performance in both convergence curve volatility and final convergence quality.

TABLE II. ABLATION TEST RESULTS

Model	Convergence Steps	Average Reward
DTS-Infer	400000+	-21.45
DTS-Infer(with out priori)	1000000+	-37.36
DTS-Infer(with out zc)	1000000+	-62.66
TD3（with out zc,zs）	1000000+	-113.35

The final data in Table 2 shows that the DTS-Infer (with out priori) algorithm achieves an 80% increase in average reward compared to TD3, while DTS-Infer (with out) achieves a 50% increase in average reward over TD3. This demonstrates that the algorithm designed in this paper significantly enhances performance.

C. Noise Countermeasure Experiment

To verify whether the DTS-Infer algorithm can mitigate the impact of uncertainty when confronted with gray-level uncertain information, a noise adversarial experiment was conducted in the HalfCheetah-Vel environment. During the training phase, the algorithm was trained according to the basic DTS-Infer procedure. In the testing process, the simulated environment continuously generated gray uncertainty

information by introducing Gaussian noise to the state generated at each step, observing its noise handling capability. On the other hand, during testing, gray uncertainty information was simulated to occur randomly by adding noise to the state after setting a random stride. Furthermore, the model performance under noisy conditions was compared with the PEARL algorithm. The experimental results are shown in Figure 9.

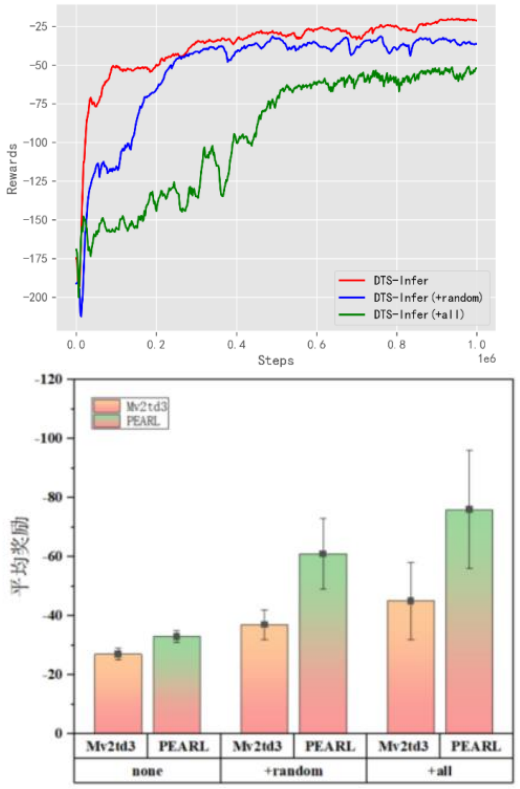


Figure 9. Noise Resistance Experiment

As shown in the curve diagram on the left of Figure 9, DTS-Infer can still accomplish the agent control task even after incorporating two types of noise. Compared to DTS-Infer without noise influence, when random noise is introduced, due to the completeness of the distribution of, noise data is continuously incorporated into the distribution during training, resulting in a final convergence outcome with minimal difference. However, when noise is introduced at every step, the algorithm converges slowly in the early stages and exhibits significant curve fluctuations. This occurs because historical task training lacked noise training. The distribution of large amounts of noisy data differs greatly from the prior distribution, reducing the advantage of prior historical experience provided to the agent. Adaptation to the current noisy data is required. During the mid-to-late testing phase, the model curve becomes relatively smoother due to adaptation. However, the variance remains substantial. This occurs because some noise is overly random, rendering the feature information within the corresponding state data ineffective. It also becomes difficult to retain the original normal information features, leading to erroneous decisions and thus high variance.

As shown in Figure 9 (right), from the final convergence results, the DTS-Infer algorithm achieves similar performance to the PEARL algorithm without noise when random noise is introduced. Moreover, when noise is added at every step, DTS-Infer significantly outperforms PEARL. This demonstrates that the information-based DTS-Infer algorithm can indeed handle gray uncertainty information.

VI. Conclusions

In this paper, we proposed the Dual-Task-State Inference (DTS-Infer) method, a novel framework designed to enhance the robustness of reinforcement learning agents against uncertainty. Our approach explicitly addresses two distinct challenges prevalent in autonomous decision-making: fuzzy uncertainty stemming from ambiguous task goals and gray uncertainty arising from corrupted state observations. By leveraging variational inference, DTS-Infer employs a dual - network architecture to separately infer latent variables for both task context and state representation, providing the agent with a more complete informational basis for decision-making.

The effectiveness of our method was empirically validated through extensive experiments on continuous control benchmarks. The results demonstrated that DTS-Infer not only accelerates convergence but also achieves superior final performance, yielding an 18.9% increase in average reward over the state-of-the-art PEARL algorithm in the Half-Cheetah-Fwd-Back task. Furthermore, ablation studies underscored the critical contribution of our dual-inference mechanism, which was shown to improve

performance by 80% compared to a standard TD3 baseline. These findings confirm that by disentangling and explicitly modeling different sources of uncertainty, we can develop significantly more adaptive and resilient intelligent agents.

For future work, we plan to extend this framework to more complex scenarios, such as partially observable environments where uncertainty is an intrinsic property of the problem. Further research could also explore the integration of more sophisticated generative models to handle other forms of uncertainty information beyond the fuzzy and gray categories addressed herein.

REFERENCES

[1] Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. Science, 185(4157), 1124-1131.

[2] March, J. G. (1994). A primer on decision making: How decisions happen. Free Press.

[3] Wang Qingyin, Liu Zhiyong. The Conception, Sort and Mathematical Expression of Uncertainty Information [J]. Operations Research and Management Science, 2001, (4): 9-15.

[4] Deng, J. L. (1982). Control problems of grey systems. Systems & Control Letters, 1(5), 288-294.

[5] Liu, S., & Lin, Y. (2007). Grey information: Theory and practical applications. Springer Science & Business Media.

[6] Zadeh, L. A. (1965). Fuzzy sets. Information and Control, 8(3), 338-353.

[7] Dubois, D., & Prade, H. (1980). Fuzzy Sets and Systems: Theory and Applications. Academic Press.

[8] Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv preprint arXiv:1312.6114.

[9] Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. arXiv preprint arXiv:1401.4082.

[10] Hoffman, M. D., Blei, D. M., Wang, C., & Paisley, J. (2013). Stochastic Variational Inference. Journal of Machine Learning Research, 14(1), 1303-1347.

[11] Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. Journal of the American Statistical Association, 112(518), 859-877.

[12] Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. The Annals of Mathematical Statistics, 41(1), 164-171.

[13] Efron, B., & Tibshirani, R. J. (1993). An Introduction to the Bootstrap. CRC press.

[14] Bishop, C. M., & Winn, J. M. (2006). Variational Bayesian inference. Journal of Machine Learning Research, 6(Jan), 211-244.

[15] Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-. hook. Ph.D. thesis, Tech. Univ. Munich.

[16] Bengio, Y., Bengio, S., and Cloutier, J. Learning a synaptic learning rule. Universit ´ e de Montr ´ eal, 1990.

[17] Thrun, S., & Pratt, L. (1998). Learning to learn: Introduction and overview. In Learning to learn (pp. 3-17). Springer.

[18] Tan Xiaoyang, Zhang Zhe. A Survey of Meta-Reinforcement Learning [J]. Journal of Nanjing University of Aeronautics and Astronautics, 2021, 53(5): 653-663. DOI: 10.16356/j.1005-2615.2021.05.001.

[19] Duan, Y., et al. "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning." arXiv preprint arXiv:1611.02779 (2016)

[20] Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning (pp. 1126-1135).

[21] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine,S. Meta-reinforcement learning of structured exploration strategies. In Advances in Neural Information Processing Systems, 2018.

[22] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In International Conference on Learning Representations, 2018.

[23] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. arXiv preprint arXiv:1611.05763, 2016.

[24] Rakelly, K., Zhou, A., Finn, C., Levine, S., & Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Advances in Neural Information Processing Systems (pp. 11807-11818).

[25] Wingate D, Goodman N D, Roy D M, et al. Bayesian policy search with policy priors[C]//Twenty-second international joint conference on artificial intelligence. 2011.

[26] Florensa C, Duan Y, Abbeel P. Stochastic neural networks for hierarchical reinforcement learning [J]. arXiv preprint arXiv:1704.03012, 2017.

[27] Mnih, A., & Rezende, D. J. (2016). "Variational inference for Monte Carlo objectives." International Conference on Machine Learning, 2016.

[28] Gu, S., Lillicrap, T., Ghahramani, Z., & Turner, R. E. (2017). Variational Policy Gradient Algorithms. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1194-1203).

[29] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[30] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3-4), 229-256.

[31] Watkins, C.J.C.H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292.

[32] Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In International Conference on Machine Learning (pp. 1583-1592).

[33] Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Muldal, A, ... & Lillicrap, T. (2018). Distributed Distributional Deterministic Policy Gradients. In International Conference on Learning Representations (ICLR).

[34] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physicsengine for model-based control. In International Conference on Intelligent Robots and Systems (IROS), pp.5026-5033, 2012.

[35] Rothfuss, J., Lee, D., Clavera, I., Asfour, T., and Abbeel, P.Promp: Proximal meta-policy search. In International Conference on Learning Representations, 2018.

APPENDIX

Decision Network Algorithm Update

The Actor network update employs policy gradient descent, specifically expressed as:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_i \nabla_{ai} Q\left(s_i, z_c, z_s, a_i \middle| w\right) \cdot \nabla_\theta \pi\left(s_i, z_c, z_s \middle| \theta\right)$$

V2TD3 incorporates random noise N into the decision-making process to enhance the exploration capability of the Actor network.

$$a_i = \pi\left(s_i, z_c, z_s \middle| \theta\right),\ \ a_i = a_i + N$$

The Critic network updates using the mean squared error loss function:

First, use the Target Actor network to compute the action corresponding to the state:

$$a_i^{'} = \pi\left(s_i^{'}, z_c, z_s \middle| \theta'\right)$$

A noise term ϵ is added to the objective smoothing strategy regularization method, resulting in $a_i^{'} = a_i^{'} + \varepsilon$

$$Loss\_Critic = \frac{1}{N}\sum_i \left(y - Q_i\left(s_i^{'}, z_c, z_{s'}, a_i^{'} \middle| \omega_i^{'}\right)\right)^2$$

Among

Them $y = r_i + \gamma \min_{i=1,2} Q_i^{'}\left(s_i^{'}, z_c, z_{s'}, a_i^{'} \middle| \omega_i^{'}\right)$

The specific representation of the task inference network Encoder1 update is as follows:

$$Loss\_Encoder1 = \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\phi, p\right) + Loss\_Critic$$

State inference network Encoder2 and Decoder network are updated simultaneously:

$$Loss\_Encoder2 = \frac{1}{n}\sum_{i=1}^{n} D_{KL}\left(q_\varphi, p\right) - \frac{1}{n}\sum_{i=1}^{n} \log p_\theta\left(s_i \middle| z_{si}\right)$$

The V2TD3 algorithm incorporates Target_ Actor and Target_Critic networks, whose architectures are identical to those of the Actor and Critic networks. These target networks enable agents to learn task policies stably. By integrating soft update principles, they gradually update network parameters, significantly enhancing the stability of agents during learning. The parameter update method is as follows:

$$\theta' = \tau\theta + (1-\tau)\theta'$$

$$w_i^{'} = \tau w_i + (1-\tau) w_i^{'}$$

Decision Algorithm Process

The algorithm's pseudocode is:

Initialize network parameters $\theta,\ \theta',\ \omega_i,\ \omega_i^{'}$

Initialize Buffer B_{T_j} , Buffer B_C

For episode=1，M do

Generate a random noise signal ϵ for action exploration

Obtain the initial observation state s1

For t=1, T do

State Inference Networks for State s1 Code Generation zs

Task Inference Network for Sample Encoding Generation zc

Actor makes decisions

$a_t = \pi_\theta\left(s_t, z_{st}, z_{ct}\right) + N1_t$

Execute action at, the environment provides reward rt and the next state st+1

Store $a_t = \pi_\theta\left(s_t, z_{ct}, z_{st} a_t, r_t, s_{t+1}\right)$ into experience pool B_{T_j}

Store $a_t = (s_t, a_t, r_t, s_{t+1})$ into experience pool BC

Sample N empirical samples $(s_t, z_{ct}, z_{st} a_t, r_t, s_{t+1})$ from B_{T_j}

Use the Target_Critic network to compute y, and use the Critic network to compute qt

Compute Loss_Critic, Update Critic Network Parameters wi

Compute Loss_Actor, Updat Actor Network Parameter θ

Compute Loss_Encoder2 , Update State Inference Network Parameters φ

Compute Loss_Encoder1 , Update State Inference Network Parameters ϕ

Update Tar_Actor, Tar_Critic Network Parameters $\theta', \omega_i^{'}$

$$\theta' = \tau\theta + (1-\tau)\theta'$$

$$w_i^{'} = \tau w_i + (1-\tau) w_i^{'}$$

End For