2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Uncertainty-Guided Active Reinforcement Learning with Bayesian Neural Networks

Xinyang Wu[1], Mohamed El-Shamouty[2], Christof Nitsche[1], Marco F. Huber[1,3]

***Abstract*— Recent advances in Reinforcement Learning (RL) have made significant contributions in past years by offering intelligent solutions to solve robotic tasks. However, most RL algorithms, especially the model-free RL, are plagued by low learning efficiency and safety problems. In this paper, we propose using the Bayesian Neural Networks (BNNs) to guide the agent exploring actively to enhance the learning efficiency in RL and investigate the potential of recognizing safety risks in working environments with uncertainty information. We compare two types of uncertainty quantification methods in both action and state spaces. To validate our method, we visualize the quantified uncertainty in robot environments with or without safety hazards. Moreover, we evaluate the learning efficiency and safety performance of the RL agents learned with BNNs on different robotic tasks.**

I. Introduction

Machine Learning has gained immense popularity in various fields, such as computer vision, autonomous control [1], and medical prognoses [2]. Reinforcement Learning (RL), in particular, has received increasing attention for solving complex tasks in diverse fields, such as games [3], [4], autonomous driving [5], and robotics [6]. While deep Neural Networks (NNs) have played a key role in enabling RL's success, their predictions are often too confident and lack the ability to quantify uncertainty. For example, even when the input data is out of distribution, the NN still tends to calculate a prediction without informing the users "I do not know" or "I am not sure". Out of the demand for uncertainty quantification, Bayesian Neural Networks (BNN) has recently gained attention, which can provide trustful predictions by quantifying their uncertainty, especially in safety-critical cases like medical prognosis and Human-Robot Collaboration (HRC).

Despite the growing interest in BNNs, few researchers have explored their use in RL. [7] proposes to enhance the stability and performance of existing model-free RL methods using uncertainty estimates from dropout-based BNNs [8]. While model-free RL algorithms can learn complex tasks by interacting with the environment, they often require a large number of interactions to gather enough informative experience.

Inspired by active learning, where a learning algorithm can interactively query for new data points, we apply the idea of uncertainty-based sampling [9] in RL. The intuition is that the RL agent shall utilize the uncertainty information to actively explore the most uncertain areas in state spaces, because higher uncertainty means higher entropy, which indicates potentially more information gain [10]. Figure 1 illustrates the learning process of the RL agent. Well-explored areas in state space will have low uncertainty and thus be considered less informative than the high uncertainty areas. Meanwhile, areas, which keep causing high uncertainty to BNNs even after enough explorations, will be considered risky and shall be avoided by the agents.

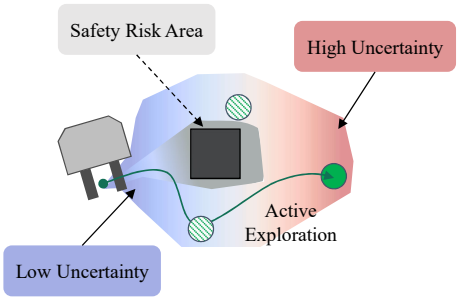


Fig. 1. Guided by the uncertainty quantification of BNNs, agents actively explore the high uncertainty areas (marked in red) and tend to reduce collecting experience from the low uncertainty areas (marked in blue), which are already sufficiently explored. The areas that make BNNs still uncertain, even after many explorations, will be marked as unsafe risks (marked in gray).

In this paper, we first present the RL formalism and the learning efficiency problem in model-free RL in Section II. Then we introduce the state-of-the-art about BNNs in Section III. In Section IV, we introduce our approach with uncertainty-based sampling and safety-aware exploration by utilizing BNNs. In Section V, we evaluate qualitatively and quantitatively the learning efficiency of RL agents with or without BNNs, and their safety performance in hazardous environments with uncertainty quantification.

II. Problem Statement

A. Reinforcement Learning

RL aims to find optimal actions in decision-making processes that maximize immediate and future rewards [11]. An RL problem contains four basic elements: the current state s_t, the action a_t taken at time-step t, the next state s_{t+1} with transition probability $P(s_{t+1}|s_t, a_t)$, and the immediate

[1] X. Wu, C. Nitsche and M. F. Huber are with the Cyber Cognitive Intelligence Department, Fraunhofer IPA. [xinyang.wu, christof.nitsche] @ipa.fraunhofer.de, marco.huber@ieee.org.
[2] M. El-Shamouty is with the Robot and Assistive Systems Department, Fraunhofer IPA. mohamed.el-shamouty@ipa.fraunhofer.de.
[3] M. F. Huber is also with the Institute of Industrial Manufacturing and Management IFF, University of Stuttgart.

reward $R(s_{t+1}|s_t, a_t)$ received from the environment. A trajectory or rollout refers to one complete process from start to end. The Markov property implies that the transition probability depends only on the current state and action [12]. The learned decision maker is called an agent, which collects experience from the environment by interacting with it.

The Actor-Critic structure [13], [14] is a popular approach in RL, where the policy, or the so-called actor, aims to learn an optimal policy

$$\pi^\star(s_t) = \arg\max_{a_t} Q^\star(s_t, a_t) \ , \tag{1}$$

by selecting actions that maximize the expected return estimated by the critic

$$Q^\star(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1}\sim P} \left[r_t + \gamma \max_{a_{t+1}} Q^\star(s_{t+1}, a_{t+1}) \right] \ , \tag{2}$$

following the Bellman equations [11]. The critic $Q(s_t, a_t)$ is represented as an NN and the network is updated by performing gradient descent on the *Temporal Difference Error* (TD-Error) loss function

$$\mathrm{Error}_{\mathrm{TD}} = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \ . \tag{3}$$

A famous algorithm based on the Actor-Critic structure is Deep Deterministic Policy Gradient (DDPG) [15], which is a model-free RL algorithm and utilizes deep NNs with target networks to generalize for continuous state and action spaces.

B. Sampling and Learning Efficiency

RL can be classified as model-based and model-free RL depending on whether to learn the model for the transition $P(s_{t+1}|s_t, a_t)$. Model-free RL has gained attention in recent years due to its simplicity and flexibility in different environments and tasks. However, the sampling and learning efficiency is a key limitation, because model-free RL usually needs to perform a large number of random actions to reach a successful shot during the exploration process, so that a positive reward is triggered and this success will be learned. Many researchers have worked on encapsulating domain knowledge in the form of shaped reward functions [16], [17] or by dividing the task into simpler tasks with auxiliary rewards [18] to guide the exploration process, which, however, requires domain-specific expertise and can lead to local optima. Off-policy algorithms introduce the notion of replay buffers [19] to utilize experience during the whole learning process rather than experience generated in a rollout only by the current policy. Variants exist which apply prioritized sampling on the replay buffer [20] to bias learning of important samples or apply memory augmentation [21] to increase samples with positive rewards. This paper mainly considers the model-free RL and focuses on improving its learning efficiency in robotic tasks with BNNs.

III. Related Work

A. Bayesian Neural Networks

Compared to normal NNs following frequentist approaches, a BNN is characterized by a joint probability distribution $p(\theta)$ over all the model parameters. Moreover, the prediction of a BNN with given inputs results in a probabilistic inference from the weight distributions, as shown in Figure 2. The training of a BNN is namely finding the *posterior distribution* $p(\theta|\mathcal{D})$ based on the learned data $\mathcal{D}$. During the training process, the famous Bayesian rule is followed to adjust the distribution $p(\theta|\mathcal{D})$ with the new information from unseen training data. The Bayesian rule for a BNN can be formulated as

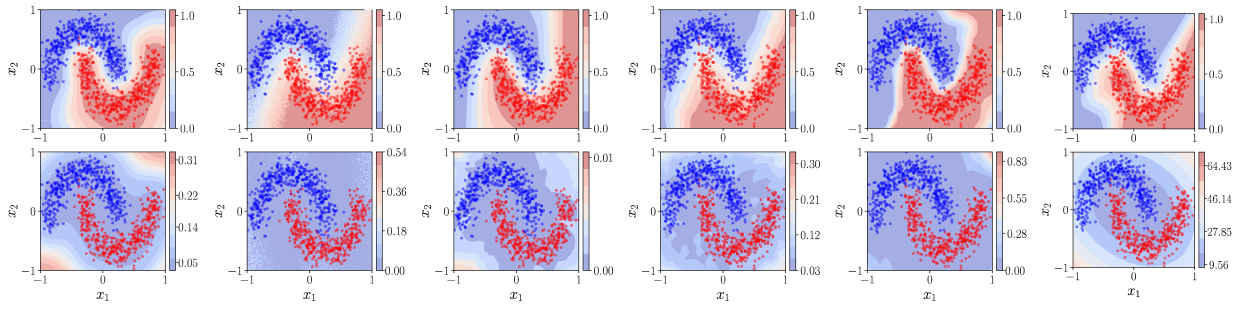
$$p(\theta|\mathcal{D}) = \frac{p(\mathbf{Y}|\mathbf{X}, \theta) \cdot p(\theta)}{p(\mathbf{Y}|\mathbf{X})} \tag{4}$$

with the training data $\mathbf{X} \triangleq [\mathbf{x}_1 \ \dots \ \mathbf{x}_N]$, $\mathbf{Y} \triangleq [\mathbf{y}_1 \ \dots \ \mathbf{y}_N]$. Here $p(\theta)$ denotes the *priori distribution*, which is assumed to be the initial distribution of weights. This can either contain prior knowledge about the problem or be as general and uninformative as possible. $p(\mathbf{Y}|\mathbf{X}, \theta)$ is called *likelihood*. It describes how well the training data can be modeled with the available model parameters θ. $p(\mathbf{Y}|\mathbf{X})$ is the so-called *evidence*. It serves as a normalization factor and describes the principal probability over the training data independent of the parameters. $p(\mathbf{Y}|\mathbf{X})$ cannot be calculated exactly in general, which is why equation (4) can only be solved approximately in practice, especially for a large number of parameters θ, where a high-dimensional integration arises. The following paragraphs introduce different methods for approximately calculating the posterior distribution of the parameters and training the BNNs.

B. Approximation Methods

Dropout is famous for suppressing overfitting in deep NNs as an L2 regularizer [23]–[25]. With dropout layers in front of each layer, *Monte Carlo Dropout* (MC Dropout) [8] trains the deep NN as a BNN to learn the posterior distribution of the underlying Gaussian Process (GP) [26], by minimizing the Kullback-Leibler divergence (KL divergence). Based on MC Dropout, Concrete Dropout [27] can automatically adjust the dropout rates as more data are observed. On the other hand, *Deep Ensemble* [28] uses an ensemble of randomly initialized NNs with the same architecture as an approximation to the BNNs. Although this method is intuitively reasonable, the authors do not offer mathematical proof as in the MC Dropout method. An extension, namely *Anchored Ensembling* [29], is proposed to regularize the parameters of the deep NNs with assumed prior distributions.

Another approach for learning Bayesian inference models is the Variational Inference (VI), which approximates the complicated posterior distribution $p(\theta|\mathcal{D})$ through a simple distribution like a Gaussian. By minimizing the empirical lower bound to the reverse KL divergence using gradient descent, VI formulates the Bayesian inference as an optimization problem, which is equivalent to maximizing the *evidence lower bound* (ELBO) loss function [30]. To improve the scalability of VI on big datasets, Stochastic Variational Inference (SVI) [31] computes a scaled gradient on randomly sampled subsets of data to update the parameters instead of the entire dataset.



(a) Gaussian Process (b) MC dropout (c) Deep Ensemble (d) Variational Inference (e) MCMC (f) KBNN

Fig. 2. Comparison between different BNNs on the "Moon" binary classification dataset [22]. The data points are generated synthetically and labeled as red and blue points in the moon's shape. The first row represents the decision boundary, and the second row is the uncertainty quantification. (a) represents the Gaussian Process (GP) as a baseline; (b)–(f) are different BNNs with the same network architectures. As an online learning algorithm, KBNN is trained only for three epochs.

Another import variant is *Markov Chain Monte Carlo* (MCMC). MCMC combines various sampling methods with the construction of Markov chains for the desired probability distribution, such as the Metropolis-Hastings algorithm [32], [33], which is, however, computationally inefficient. Many improvements have been proposed, such as Gibbs sampling [34], Hamiltonian-Monte-Carlo (HMC) [35], and the No-U-Turn Sampler (NUTS) [36].

Many efforts in past years have extended the classic Kalman Filter to the training of normal NNs [37]–[39]. *Kalman Bayesian Neural Network* (KBNN) [40], [41] offers a closed-form solution to treat the training of BNNs as a nonlinear filtering problem. Such Kalman-filter based methods have proven to be much more effective regarding the learning efficiency, compared to the gradient-based and batch-based methods [42]. Figure 2 shows a comparison between a GP and different types of BNNs on a binary classification dataset. Similar to a GP, BNNs tend to have high uncertainties in the areas where no data is available and low uncertainties in high data-intensity areas.

C. Different Uncertainties

In the context of uncertainty quantification, two sources of uncertainties are usually discussed. The *aleatoric* uncertainty describes the intrinsic uncertainty within the data, also often referred to as measurement noise. The *epistemic* uncertainty represents the lack of knowledge in models, which is reflected in the probability distribution over the model parameters. In our work, we mainly focus on handling the epistemic uncertainty.

IV. Method

In this section, we introduce the approach of using BNNs for uncertainty-based sampling in the Actor-Critic structure, which instructs the agent to actively explore the most uncertain areas in state space with two different types of BNNs. Besides that, we further explore the possibility of utilizing uncertainty information for improving safety in robotic tasks.

A. Uncertainty-Based Sampling

Inspired by the application of Uncertainty Sampling in Active Learning [9], we propose to use a similar concept in the exploration phase of RL agents. As explained in Section II-A, the RL-Setting has four basic elements $(s_t, a_t, P(s_{t+1}|s_t, a_t), R(s_{t+1}|s_t, a_t))$, and exploration can happen in either action or state spaces in traditional RL algorithms: (1) action space: the agent takes a random action at current state, and (2) state space: the agent moves into a random state. In our uncertainty-guided exploration process, the agent does not sample the action or state randomly. Instead, it will take the most uncertain action in the current state, or choose to move into the most uncertain state. In the Actor-Critic structure, we implemented BNNs in both actor and critic networks to make such uncertainty-based sampling of action and state, respectively, possible. However, a timestep-wise sampling of an action in each state is not practical regarding computational time. In our experiments, we found that each variant of BNNs will introduce computational overheads in both training and inference phases to different degrees, as shown in Table I. Therefore, we adopt Deep Ensemble and MC Dropout due to least additional computation overheads. Since the actor network needs to make the prediction for each single state and take the action at each timestep, using a BNN as the actor will bring more cost in training time than the benefits in learning efficiency. Compared to that, the critic network is used only when updating the parameters of the networks, which usually takes place every several episodes with data batches from the replay buffer as inputs. Therefore, the additional time cost for using a BNN as the critic network is much less and thus acceptable. Hence, we mainly focus on using BNNs with the critic network and applying uncertainty-based sampling in the state space.

As explained in Section II-A, we employed the Actor-Critic structure to let agents learn the robotic tasks, where we have the actor network $\pi(s)$ and the critic network $Q(s, a)$. With dropout applied before every layer and activated in the inference phase, the forward pass of the critic network is

TABLE I

COMPUTATIONAL TIME FOR DIFFERENT BNNS IN ACTOR NETWORK

	MC Dropout	Concrete Dropout	Deep Ensemble	SVI	NN
Inference	0.021s	0.075s	0.017s	0.194s	0.005s
Training	0.022s	0.117s	0.089s	0.066s	0.009s

TABLE II

RELATIONSHIP BETWEEN UNCERTAINTY AND SAFETY IN RL

	Low Risk	**High Risk**
Low Uncertainty	Safe area to work in	Avoided due to low rewards
High Uncertainty	Active exploration	Avoided due to high uncertainties

mathematically equivalent to a GP. By performing moment-matching, the first two moments of the predictive distribution of the critic network trained with MC Dropout are given by

$$\mathbb{E}_{q(Q|s,a)}(Q) \approx \frac{1}{N}\sum_{n=1}^{N}\widehat{Q}(s,a,\theta_n)\ , \tag{5}$$

$$\sigma^2_{q(Q|s,a)}(Q) \approx \frac{1}{N}\sum_{n=1}^{N}\widehat{Q}(s,a,\theta_n)^T\widehat{Q}(s,a,\theta_n) - \mathbb{E}_{q(Q|s,a)}(Q)^T\mathbb{E}_{q(Q|s,a)}(Q)\ , \tag{6}$$

where θ represents the random variables for the parameters of the dropout model, $\{\theta_n\}_{i=1}^N$ are N samples from θ, $\mathbb{E}_{q(Q|s,a)}(Q)$ is the first moment, i.e., the mean value, and the prediction of the BNN, while $\sigma^2_{q(Q|s,a)}(Q)$ stands for the model's predictive variance. In practice this is equivalent to calculating the mean and sample variance of N stochastic forward passes through the network [8], [24]. In this case, the TD-Error for training the critic network with MC Dropout will be

$$\mathrm{Error}_{\mathrm{TD}}^{\mathrm{Dropout}} = r_t + \gamma \max_{a_{t+1}} \mathbb{E}_{q(Q|s_{t+1},a_{t+1})}(Q) - \mathbb{E}_{q(Q|s_t,a_t)}(Q)\ . \tag{7}$$

Differently from MC Dropout, Deep Ensemble randomly initializes multiple NNs with the same network architectures and trains them separately on the entire dataset, along with random shuffling of the data points. Thus, Deep Ensemble trains the ensemble of multiple critic networks as a uniformly-weighted mixture model and the predictive distribution is given by

$$\mathbb{E}_{q(Q|s,a)}(Q) \approx \frac{1}{M}\sum_{m=1}^{M}Q_m(s,a,\theta_m)\ , \tag{8}$$

$$\sigma^2_{q(Q|s,a)}(Q) \approx \frac{1}{M}\sum_{m=1}^{M}Q_m(s,a,\theta_m)^T Q_m(s,a,\theta_m) - \mathbb{E}_{q(Q|s,a)}(Q)^T\mathbb{E}_{q(Q|s,a)}(Q)\ , \tag{9}$$

where $\{Q_m|\theta_m\}$ stands for the m-th NN in the ensemble, while $\mathbb{E}_{q(Q|s,a)}(Q)$ and $\sigma^2_{q(Q|s,a)}(Q)$ represent the mean and variance of the mixture model, respectively. The loss function for training BNNs with Deep Ensemble remains the same as Equation (3), since the networks are trained separately, and the calculation of mean and variance is not required.

In each training episode, classic RL algorithms will randomly explore and uniformly sample from the whole space with given noise. With our uncertainty-based sampling in the state space, we first sample N candidate states from the state space. For each of them, we let the actor network predict the action $a_i = \pi(s_i)$, where $i = 1, ..., N$. For the N state and action pairs $\{s_i, a_i\}_{i=1}^N$, we can estimate the predictive distributions from the probabilistic critic network trained with BNNs. The candidate state that has the highest predictive variance $s_i = \mathrm{argmax}_i\, \sigma^2_{q(Q|s_i,a_i)}(Q)$ will be chosen as the next state to explore.

A previous work proposed using multiple neural networks as critic networks and utilizing the disagreements between them to sample the new learning position [43], which is equivalent to using Deep Ensemble as a BNN variant to learn uncertainty quantification in the state space. However, Deep Ensemble lacks mathematical proof, and using multiple networks requires significantly more computational time for both training and inference phases. In contrast, recent research has demonstrated that MC Dropout incurs much less computational overheads in RL models and is thus a more practical and effective alternative [7].

B. Safety-Aware Exploration

Safety has always been challenging in using RL algorithms in safety-critical application, such as collaborative robots in HRC scenarios where the robot can harm the human coworker. While learning to finish given tasks, the RL agent should inform the robots about the risk in state space and avoid risky areas. In this paper, we use a training environment with obstacles [44], where the robot needs to learn not only efficient but also safe behaviors to finish the task.

With uncertainty quantification in RL, the state space observed by the agent can be divided into four categories, as shown in Table II. As explained in Section I, the agent will actively explore the high uncertainty areas during the learning process. The low risk areas will be learned as safe areas without negative rewards, i.e., punishments. The areas, where the agent sufficiently explored with low uncertainties and continuously received punishments due to violation of soft safety constraints, will be avoided in the future due to low rewards. The areas close to hard constraints, such as obstacles, can lead to fluctuating rewards due to the possibility of collision with the obstacle, which further results in high uncertainties around the obstacle. The agent can ensure its safety by avoiding such risky areas if uncertainties remain high after sufficient exploration.

a) Reward Function: To make the robot capable of learning to avoid collisions with the obstacle, we adopt the reward function from [44], which is formulated as

$$r = -\alpha_1 \cdot [d(x_{\mathrm{target}}, x_{\mathrm{robot}}) > \delta] - \alpha_2 \cdot [\max(f_{\mathrm{contact}}) > 0]\ , \tag{10}$$

where $d(\cdot)$ denotes the Euclidean distance, x_{target} and x_{robot} are the positions of target and the robot's end effector, respectively, δ is a threshold distance, f_{contact} is the contact force measured by the robot due to hitting obstacle and α_i

are non-negative trade-off coefficients between the risks of hitting the obstacle and benefits from reaching the target. The conditions, such as $(f_i > 0)$, are evaluated as either 0 or 1 to represent no-contact or contact situations, respectively. Thus, the reward function is sparse, e.g., outputting $-\alpha_2$ in case of reaching the target but having collisions.

b) Loss Function: As illustrated in Figure 2, GP and BNNs tend to have lower uncertainties in the high data-intensity areas while higher uncertainties where data is rare. In RL environments, the BNNs show the reduction of uncertainty in frequently explored areas. However, the collision with the obstacle and the punishment in the reward function, as in (10), causes reward fluctuations and keeps the uncertainty high. From this intuition, we define the areas already visited many times but that still cause high uncertainties in BNNs as risky areas. To suppress the uncertainty and make the RL agent learn towards safe behaviors, we add the uncertainty term in the form of the Upper Confidence Bound (UCB) to the loss function in Equation (7) for MC Dropout

$$\begin{aligned}\mathrm{Error}_{\mathrm{TD}}^{\mathrm{Dropout}} =& r_t + \gamma \max_{a_{t+1}} \mathbb{E}_{q(Q|s_{t+1},a_{t+1})}(Q) \\ & - \left[\mathbb{E}_{q(Q|s_t,a_t)}(Q) + \rho \cdot \sigma_{q(Q|s_t,a_t)}(Q)\right] ,\end{aligned} \tag{11}$$

where ρ is the non-negative confidence value for suppressing the uncertainty. Worth mentioning that the involvement of uncertainty in loss function during training is only possible for MC Dropout. Deep Ensemble trains each NN of the ensemble independently, and the mean and variance are only calculated in the inference phase.

V. Experiments

In this section, we validate and evaluate our approach in three aspects by means of the DDPG algorithm: (1) exploration process, i.e., a visualization of the learned uncertainties during the exploration process of the RL agent, (2) learning efficiency, i.e., an evaluation of our approach on four robotic tasks and comparison of the learning efficiencies with other methods, (3) safety, i.e., we evaluate the impact of safety risks on uncertainty quantification and the safety performance of RL agents. All the experiments are repeated with five random seeds to get statistical results. For Deep Ensemble, we use five NNs in the ensemble.

A. Visualization of the Exploration

a) Experiment Setup: In this experiment, we utilize the *FetchReach-v1* environment from OpenAI Gym [45], [46]. The task for the robot is to reach a given end-effector position. We constrain the three-dimensional Cartesian space to the two-dimensional horizontal plane with a fixed height to visualize the uncertainty during exploration and validate the learned uncertainty quantification.

b) Result: Figure 3(a) shows our approach's learning and exploration process, where the robot actively chooses the relatively more uncertain areas to explore. By comparing the first and the second rows, MC Dropout can reasonably quantify the uncertainties in several episodes, while Deep Ensemble needs more episodes to reach a stable quantification. To evaluate the learned behaviors of agents on the given task, we use *test success rates* to denote the rates of successful achievements of tasks by the agents in test environments. Figure 3(b) shows the test success rates of the NN and the other two BNNs on the two-dimensional Reach task, where we could observe an improvement by MC Dropout in terms of learning efficiency in the early phase.

B. Learning Efficiency

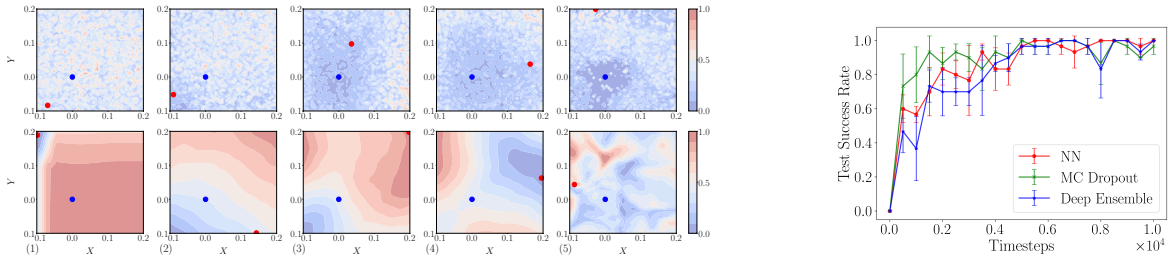
a) Experiment Setup: In this part, we evaluate the learning efficiency and focus on comparing the learning curves of the test success rates of the agents that are trained with normal NNs, MC Dropout and Deep Ensemble, on the four classic robotic tasks from OpenAI Gym: (1) Reach, i.e., the robot arm learns to reach a given position. (2) Push, i.e., the robot arm learns to push one box to a given position. (3) Slide, i.e., the robot arm learns to hit one ball and let it reach a given position. (4) PickAndPlace, i.e., the robot arm picks one box and places it in a given position.

b) Result: As shown in Figure 4, we found MC Dropout with a dropout rate of 0.1 can achieve the best performance compared to other dropout rates on Reach and Slide tasks. However, the learning efficiency is not improved compared to the normal NN. On Push and PickAndPlace tasks, MC Dropout achieves its best results with a dropout rate of 0.2 and shows higher learning efficiency than normal NNs. Although Deep Ensemble performs similarly on Push and PickAndPlace tasks, its computational time is roughly three to four times more than MC Dropout. In our test on the Reach task for $10,000$ timesteps, normal NN has a runtime of approximately 3 minutes and MC Dropout of nearly 4 minutes. In contrast, the runtime of Deep Ensemble is more than 15 minutes.

C. Safety

a) Experiment Setup: In this part, we add one obstacle into the Reach environment, which can have contact forces with the robot arm and stop it from finishing its task [44]. The goal position is randomly sampled within the robot's reachability. Moreover, the obstacle is put in a position near to the robot arm. Any collision with the obstacle will cause negative rewards as punishments to the agent, as in (10). The maximum length of each trajectory is increased from 50 to 100, considering the task complexity.

b) Result: Figure 5 compares the uncertainty by both BNNs in the early and late phases of the learning process and also shows the learning curves of different NNs in both two- and three-dimensional environments with the obstacle. Figure 5(a) shows the different uncertainty quantifications learned by BNNs in the hazardous environment, compared to Figure 3(a). The metric in Figure 5(b) is the *safe success rate*, meaning that the agent must reach the target position without colliding with the obstacle in the entire trajectory. In both two-dimensional and three-dimensional cases, MC Dropout with a dropout rate of 0.2 converges faster than normal NN and Deep Ensemble.



(a) Exploration Process (b) Learning Curve

Fig. 3. (a) The visualized uncertainty during the learning process for both BNNs in the two-dimensional space, with blue and red dots representing the initial and selected target end-effector positions, respectively. The first row represents the uncertainty quantified by MC Dropout, and the second row stands for Deep Ensemble. (1)–(4) show the quantified uncertainties in the first four episodes, and (5) shows the uncertainty after $10,000$ timesteps. From (1) to (4), we can see that the selected positions, namely the red points, are mostly picked from the high uncertainty areas of the previous episodes; and MC Dropout tends to have lower uncertainties in areas visited in the previous episodes. After a sufficient number of timesteps, the uncertainties of both BNNs converge to stable distributions in (5). Both BNNs have lower uncertainties in areas near the initial position of the robot arm and higher uncertainties in edge areas. (b) The learning curve of three types of networks in this two-dimensional environment.

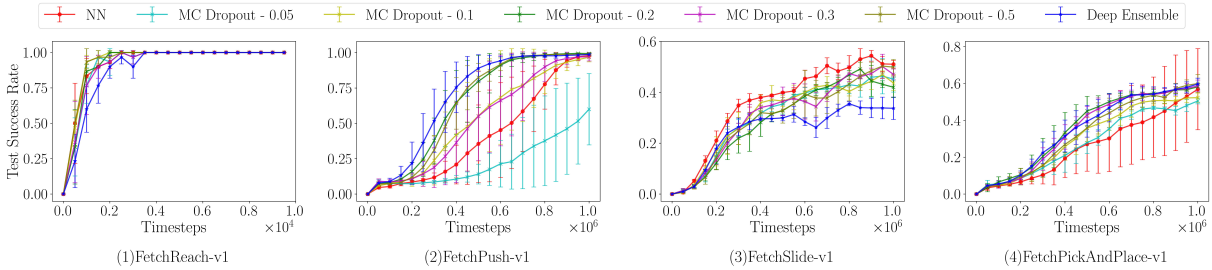
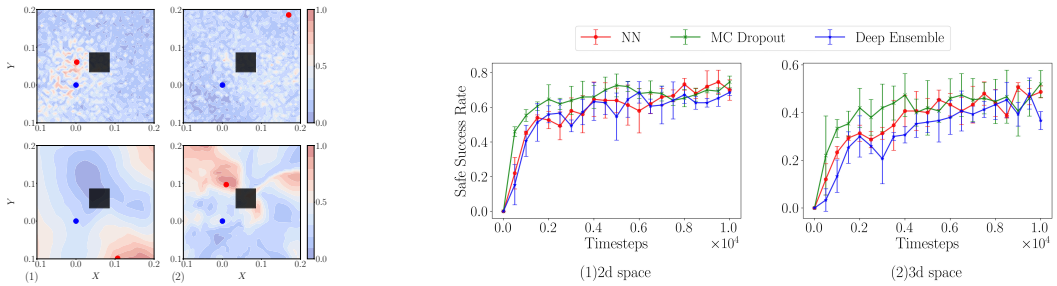


Fig. 4. Test success rates of the three kinds of networks on the four robotic tasks. Except for the Reach task, the results of the other three environments are presented as moving averages over the last 50 episodes. For MC Dropout, we also show the comparison of different dropout rates from 0.05 to 0.5.



(a) Uncertainty Quantification (b) Learning Curve

Fig. 5. (a) The uncertainty quantification of MC Dropout (first row) and Deep Ensemble (second row) in the Reach environment with the obstacle, which is placed near the robot arm and plotted as a black square in the figure. (a)(1) shows the uncertainty of both BNNs after five episodes, where MC Dropout already has recognized the risks of collision with the obstacle because of the punishments in reward function (10). Meanwhile, Deep Ensemble is still not aware of the obstacle. (a)(2) shows the stabilized uncertainty quantification by both BNNs after $10,000$ timesteps, where both BNNs have higher uncertainties in the areas near the obstacle, compared to Figure 3(a). Compared to Deep Ensemble, MC Dropout calculates lower uncertainties near the obstacle. (b) shows the obstacle environment's safe success rates of the three different networks. (b)(1) stands for the learning curve in two-dimensional space, with which we visualized the uncertainty in (a). (b)(2) represents the safe success rates in three-dimensional space.

VI. Conclusion

Sampling and learning efficiency is always a bottleneck to model-free RL. One solution is combining uncertainty quantification from BNNs with model-free RL so that the agent can actively explore the regions with most information gain and avoid wasting efforts in well-explored areas. We propose the uncertainty-based sampling and safety-aware exploration approaches with two variants of BNNs that bring the least additional computational cost, namely MC Dropout and Deep Ensemble, and evaluate them in classic robotic tasks. Results show that MC Dropout can improve the learning efficiencies and the safe success rates with much less computational overheads compared to Deep Ensemble. In the future, we plan to combine BNNs with other safety assurance methods in robotics to make RL more widely used in industry.

Acknowledgment

This work was partially supported by the Baden-Württemberg Ministry of Economic Affairs, Labor, and Tourism within the KI-Fortschrittszentrum "Lernende Systeme und Kognitive Robotik" under Grant No. 017-180036 and the Fraunhofer-Gesellschaft within ML4Safety.

REFERENCES

[1] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to Throw Arbitrary Objects with Residual Physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[2] P. M. Amisha, M. Pathania, and V. K. Rathaur, "Overview of artificial intelligence in medicine," *Journal of family medicine and primary care*, vol. 8, no. 7, p. 2328–2331, 2019.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[5] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[6] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[7] Y. Wu, S. Zhai, N. Srivastava, J. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh, "Uncertainty weighted actor-critic for offline reinforcement learning," *arXiv preprint arXiv:2105.08140*, 2021.

[8] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[9] B. Settles, "Active learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 6, no. 1, pp. 1–114, 2012.

[10] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] A. A. Markov and N. M. Nagorny, *The Theory of Algorithms*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[13] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[14] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Icml*, vol. 99, 1999, pp. 278–287.

[17] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017.

[18] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing solving sparse reward tasks from scratch," in *International conference on machine learning*. PMLR, 2018, pp. 4344–4353.

[19] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[25] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, 2013.

[26] A. Damianou and N. D. Lawrence, "Deep gaussian processes," in *Artificial intelligence and statistics*. PMLR, 2013, pp. 207–215.

[27] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," *Advances in neural information processing systems*, vol. 30, 2017.

[28] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[29] T. Pearce, F. Leibfried, and A. Brintrup, "Uncertainty in neural networks: Approximately bayesian ensembling," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 234–244.

[30] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[31] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, 2013.

[32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[33] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

[34] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 721–741, 1984.

[35] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics letters B*, vol. 195, no. 2, pp. 216–222, 1987.

[36] M. D. Hoffman, A. Gelman *et al.*, "The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo." *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1593–1623, 2014.

[37] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended kalman algorithm," *Advances in neural information processing systems*, vol. 1, 1988.

[38] K. Watanabe and S. G. Tzafestas, "Learning algorithms for neural networks with the kalman filters," *Journal of Intelligent and Robotic Systems*, vol. 3, no. 4, pp. 305–319, 1990.

[39] G. V. Puskorius and L. A. Feldkamp, "Parameter-based kalman filter training: Theory and implementation," *Kalman filtering and neural networks*, pp. 23–67, 2001.

[40] M. F. Huber, "Bayesian perceptron: Towards fully bayesian neural networks," in *Proceedings of the 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 3179–3186.

[41] P. Wagner, X. Wu, and M. F. Huber, "Kalman bayesian neural networks for closed-form online learning," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, 2023.

[42] S. Haykin, *Kalman filtering and neural networks*. John Wiley & Sons, 2004.

[43] Y. Zhang, P. Abbeel, and L. Pinto, "Automatic curriculum learning through value disagreement," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7648–7659, 2020.

[44] M. El-Shamouty, X. Wu, S. Yang, M. Albus, and M. F. Huber, "Towards safe human-robot collaboration using deep reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4899–4905.

[45] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[46] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.