Towards Better Uncertainty Estimation in Model-Based Reinforcement Learning

Joshua M. Smith **Chenqi Zhu**

Abstract

Reinforcement Learning(RL), despite attaining phenomenal performance on simulated tasks in robotics and video games, often fails to resolve practical tasks in real-world environment due to high sample complexity. Model-based RL algorithms are capable of drastically reducing that but also performs sub-par against their model-free counter-parts. The key to uplift model-based RL performance is adopting probabilistic models, which heavily relies on an accurate estimation of the uncertainties within the model itself. In our project, we aim to adopt state-of-the-art methods to uncertainty estimation for better exploration in RL.

1 Problem Background

Current Reinforcement Learning(RL) algorithm are generally split into two major categories. Model-based RL and Model-free RL. While Model-free RL the agent attempts to learn a value function or policy that maximizes cumulative rewards, in Model-based RL the agent directly learns the mapping from state to action. Hence the model-free approach is promised to greatly lower sample complexity in learning practical tasks outside of simulated environments Janner et al. [2021]. However, model-based learning is subject to modelling errors known as model bias in which the algorithm incorrectly assumes the learned model sufficiently reflects the dynamics of the real environment Deisenroth and Rasmussen [2011].

Previous works have shown that probabilistic models and ensembles are efficient in addressing those issues and experiments have revealed that they are able to match the asymptotic performance of model-free algorithms with far less samples Wang et al. [2019]. However, precise characterization of the uncertainties in those models is often required for probabilistic models to work. This probabilistic uncertainty of the model is determined by the aleatoric uncertainty and the epistemic uncertainty. The aleatoric uncertainty refers to uncertainty that cannot be reduced by gathering more training data. This can be caused by a variety of sources, such as imprecise measurements, model representation ability, or inherent stochasticity in the environment, thus this type of uncertainty cannot be resolved by the abundance of data. Epistemic uncertainty describes the uncertainty in the learned model which could be resolved by collecting and training on more training dataGal [2016], or it can be solved much efficiently with better exploration using improved uncertainty estimation. Therefore, an accurate and provable estimation of those uncertainties is crucial to the performance of Model-based RL.

2 Related Work

2.1 Model-based RL

Model-based RL algorithms mostly fall into three categories: Dyna-style, Policy Search, and Shooting algorithms.

Dyna algorithm is a learning scheme where agents observes states and reactively chooses actions to receive rewards, then update its model from observed dataSutton [1991]. Existing algorithms in

Preprint. Under review.

this style includes Model-Ensemble Trust-Region Policy Optimization (ME-TRPO), which uses an ensemble of deep neural networks and employs Trust Region Policy Optimization to reduce gradient computationKurutach et al. [2018]. Stochastic Lower Bound Optimization (SLBO) offers theoretical guarantee to ME-TRPO by lower bounding the expected rewards but not explicitly quantifying uncertaintyLuo et al. [2021]. Model-Based Meta-Policy-Optimzation (MB-MPO) utilizes ensembles but adopts meta-learning rather than depending on accurate models.

Unlike Dyna algorithms that rely on dynamics models, Policy Search uses backpropagation through time to improve policy with model derivatives. Probabilistic Inference for Learning Control (PILCO) models the dynamics of the environment as Gaussian Processes, incorporating uncertanty into long-term planning and policy evaluation Deisenroth and Rasmussen [2011]. Iterative Linear Quadratic-Gaussian (iLQG) provides a linear approximation of the environment dynamics and solve it with linear quadratic regulator, ensuring global convergence through a Levenberg-Marquardt method Todorov and Li [2005]. Guided Policy Search (GPS) pursues asymptotic local convergence instead of a global one by constructing iteratively refitted time-varying linear dynamics models and is shown to be able to handle complex, nonsmooth dynamics Levine and Abbeel [2014].

Shooting Algorithms are designed to solve receding horizon problems which repeatedly solves an optimization problem under constraints. Random Shooting (RS) initializes the RL agent with random sequences of actions typically sampled from uniform distribution, and use the resulting linear model to synthesize a controllerKothare et al. [1995]. Mode-Free Model-Based (MB-MF) replaces the controller in RS with a neural network and fine-tune the policy using model-free algorithm to achieve comparable results to the latterNagabandi et al. [2017]. Probabilistic Ensembles with Trajectory Sampling (PETS) is the most remarkable one among all the model-based algorithm as it combines probabilistic models with ensemble technique to mitigate the sampling-based uncertainty propagation by using uncertainty-aware deep network dynamics models, it is also shown empirically that it achieves better performance overall when comparing to other model-based as well as model-free algorithmsChua et al. [2018].

2.2 Modeling of epistemic uncertainty

One method for estimating epistemic uncertainty in neural networks is by using Bayesian Neural Networks with stochastic gradient Markov Chain Monte-Carlo to sample from the posterior distribution. This method is able to approximate the posterior distribution over weights for a model very accurately Louizos and Welling [2017]. However, it requires expensive computation which is often prohibitive for large deep neural nets Osband et al. [2023].

Another method for estimating epistemic uncertainty is with an ensemble-based approximation. In this method, a distribution of statistically plausible model weights are generated by training a set of identical models on the same training data distribution, but with different batches/batch order and with different random weight initialization Kurutach et al. [2018]. Each of the models in the ensemble represents a particle in the model weight distribution. Using the difference/variance across the probabilistic predictions of these particles, a measure of epistemic uncertainty can be attained. This is because, with infinite training data, all of the models should converge to having identical probabilistic outputs for a given input, but may not as they are convergingLakshminarayanan et al. [2017]. One issue with this method, is that although the uncertainty accuracy generally benefits from over 100 particles, in many modern DNN applications, available computer and memory limits particles to around 10. This is because, in order to run the ensemble in parallel, GPU memory is necessary linearly with number of particles n.

A recent method for estimating epistemic uncertainty in deep neural nets are with epinets, a subset of Epistemic Neural Nets (ENN) Osband et al. [2023]. Epistemic Neural Nets are neural nets consisting of a parameterized function f and an epistemic index z. Bayesian Neural Nets mentioned above are examples of Epistemic Neural Nets. Osband et al. [2023] also introduces the epinet, which is a style of Epistmic Neural Net that randomly generates the index z from a specified prior distribution. Given z and the input x, the epinet model adds an pinet module to an existing base model, $m(x)$. The epinet model is defined as $f(x, z) = m(x) + \sigma(sg[\phi(x)], z)$ where the epinet model, σ is parameterized by z and a feature representation possibly taken from the base model, $\phi(x)$. This feature representation could be the identity or could have a mix of x and deep features from the base model m. Note that it was generally found that epistemic uncertainty was more accurate in Osband et al. [2023] when the gradient of the epinet model loss w.r.t the feature representation $\phi(x)$ was not propagated

back to the base model hence the stop gradient. Since every index z represents a "particle", this formulation allows the base model and feature representation $m(x)$ and $phi(x)$ to be computed once when computing n particles. If we define the epinet model to be much less computationally expensive compared to the base model, then we can produce many particles for a total computational cost less than the cost of computing 2 particles in a traditional ensemble method. Overall, epinets tend to allow better uncertainty estimation at relatively less computational cost and show promising empirical performance.

3 Project Goal

Given the significance uncertainties play in Model-based RL, we would like to develop an algorithm that achieves better estimation of it using the state-of-the-art techniques. In particular, ENN is most favorable as it offers the theoretical guarantee that any ENN with small expected joint log loss can give rise to actions with near optimal expected reward. Due to the fact that Probabilistic Ensembles with Trajectory Sampling (PETS) addresses uncertainty using uncertainty-aware neural networks, it is reasonable to argue that combining it with ENN instead of an ensemble approach would lead to better estimation of uncertainties at similar or less computational cost. Furthermore, the epinet architecture promises seamless integration of any conventional neural network and itself. We hope that better estimation of uncertainties with ENN could realize better performance of the RL agent in achieving higher expected rewards with exceptionally less sample complexity.

We also would like to experiment with Active Sensing using epinet derived epistemic uncertainty quantification. Given that epistemic uncertainty is defined as the error for a given input that could be resolved with more training data representative of this point, we believe that during the learning phase of the algorithm, promoting visiting states which have high epistemic uncertainty should accelerate learning. This is in contrast to visiting states with low or zero epistemic uncertainty, where training on the state, action transition corresponding to this state would not improve the transition model. Similar to work on intrinsic curiosity in RL, we believe that this prediction-error inspired exploration incentive could improve sample efficiency and exploration in sparse reward environments and test it with mountain-car as a toy problem.

4 Approach

Our project aim to include the implementation of both the PETS and ENN using latest Pytorch library instead of the out-dated tensorboard 1 that was used in their original implementation.

We would approach the problem both empirically and theoretically, comparing the performance of our algorithm to that of the original PETS in both Mujoco, such as Cartpole and 7-dof Pusher, which were used in the publication of PETS, and the OpenAI Gym mountain-car environment, which we believe is an uncomplicated environment that evaluate how well the agent explores. As we primarily investigate the convergence of the expected rewards as the metric for agent performance, we would also derive theoretical bounds of our algorithm, possibly finding a lower bound on the expected rewards or an upper bound on the loss.

Moreover, we would also like test out different ideas such as enhancing the exploration and performance with auxiliary rewards. One option, relating to work on intrinsic curiosity similar to Berseth et al. [2021] and Pathak et al. [2017], is to promote exploring states which have yet to minimize their epistemic uncertainty. By using an epinet and minimizing a measure of epistemic uncertainty, we would hope to yield curiosity-inspired directed exploration which is not implicated by the aleatoric uncertainty or informally the noisy t.v. problem. Also, similar to SMiRLBerseth et al. [2021], we may try minimizing both epistemic uncertainty in the Monte Carlo Search portion of the algorithm, as it may allow the Tree Search Algorithm to have less uncertainty and therefore allow a tighter bound on probabilistic performance expectation.

The documentation of our implementations will be hosted on a private GitHub repository which we will be sharing with the class graders at the end of the semester and the details of our algorithm will be written in similar Neurips style to be submitted as our final report.

5 Definitions

We define Reinforcement Learning as a Markov Decision Process formulated in Bellman [1957]. Let $\boldsymbol{s}_t \in \mathbb{R}^n$ denotes the continuous state at time stamp t and $\boldsymbol{a}_t \in \mathbb{R}^m$ denotes the continuous action, $r(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathbb{R}$ be the reward function and $p : \mathbb{R}^{n+m} \mapsto \mathbb{R}^n$ be the transition function such that $s_{t+1} = p(\boldsymbol{s}_t, \boldsymbol{a}_t)$. In Model-based RL, the goal is to learn a transition function $\hat{p}$ that estimates the true transition function given data $\mathcal{D} = \{(\boldsymbol{s}_n, \boldsymbol{a}_n), \boldsymbol{s}_{n+1}\}_{n=1}^N$ sampled from real system.

In order to capture the aleatoric and epistemic uncertainties present in the transition dynamics, Chua et al. [2018] proposes Probabilistic Ensembles with Trajectory Sampling (PETS) to use probabilistic neural networks to measure aleatoric uncertainty and ensembles to measure epistemic uncertainty.

Probabilistic Neural Networks, originally proposed by Specht [1990], is a network based on parametric estimation of a probability distribution function. The use of Gaussian Mixture Model is seen as the key to train and approximate the probability distribution function accurately Tsuji et al. [1995]. Thus we define the loss function as negative log prediction probability w.r.t. a predictive model that outputs a Gaussian distribution as

$$\mathcal{L}_{Gaussian}(\boldsymbol{\theta})$$

$$= \sum_{n=1}^{N} [\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n) - \boldsymbol{s}_{n+1}]^\top \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{s}_n, \boldsymbol{a}_n)[\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n) - \boldsymbol{s}_{n+1}] + logdet\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n)$$

where $\hat{p} = Pr[\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t] = \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n))$

While Bayesian Neural Network can provide accurate estimation of epistemic uncertainty given sufficient training data, Ensemble method provides a simpler way of achieving similar accuracy. Given an ensemble of B-bootstrapped models, where $\boldsymbol{\theta}_b$ denotes the parameter of the b-th model $\hat{p}_{\boldsymbol{\theta}_b}$, the combined probability distribution is simply $\hat{p}_{\boldsymbol{\theta}} = \frac{1}{B} \sum_{b=1}^{B} \hat{p}_{\boldsymbol{\theta}_b}$

Aleatoric uncertainty is defined as the irreducible stochasticity in the environment. In our application, we quantify this as $A = \frac{1}{b} \sum_{i=1}^{b} Var(\theta_i)$ where θ_i represents one of the b probabilistic distributions in the size b ensemble for the predicted future state given current state and action.

Epistemic uncertainty is defined as the uncertainty due to a lack of current training data. We quantify this using the Generalized Jensen-Shannon Divergence for more than 2 distributions. We compute this on the b bootstraps in the ensemble. Formally, $JSD(\theta) = \frac{1}{n} \sum_{i=1}^{n} D(\theta_i, M)$ where D is the KL-Divergence, and M is defined as $M = \frac{1}{n} \sum_{i=1}^{n} \theta_i$

Let f be function class parameterized by θ, and P_Z be a reference distribution which can be either uniform or standard Gaussian; let x be an input vector and z be an integer over P_Z, which we denote as the epistemic index that represents the underlying epistemic uncertainty possibly can be resolved by future data. The Epistemic Neural Network estimates $f_\theta(x, z)$ Osband et al. [2023].

The ENN is split into two parts: learnable network $\sigma_\theta^L(x, z)$ with trainable parameters and prior network $\sigma^P(x, z)$ constituting the prior uncertainty with no trainable parameters; the output is simply the sum of the two. Given some data points $x_1 ... x_N$, ENN offers a probability $\hat{P}_{1:N}(y_{1:N})$ to classes $y_1 ... y_N$ as a joint prediction where $\hat{P}_{1:N}(y_{1:N}) = \int_z P_Z dz \prod_{i=1}^{N} softmax(f_\theta(x_i, z))_{y_i}$

Theorem 2 of ENN guarantees that, for all data distribution $\mathcal{D}$, reward function r, and action $\hat{a} \in \arg\max_a \sum_{y_{1:N}} \hat{P}_{1:N}(y_{1:N}) r(a, y_{1:N})$,

$$\mathbb{E}[r(\hat{a}, y_{1:N})|\mathcal{D}, x_{1:N}] \geq \max_a \mathbb{E}[r(\hat{a}, y_{1:N})|\mathcal{D}, x_{1:N}] - \sqrt{2\,\mathbb{E}[KL(P_{1:N}||\hat{P}_{1:N})|\mathcal{D}, x_{1:N}]}$$

Which essentially tells that how good an action is bounded by the KL divergence between the estimated prediction $\hat{P}_{1:N}$ by Epinet and the optimal $P_{1:N}$.

6 Implementation

Our code base is available at github.com/MatthewCQSZ/cmsc742_uncertainty/.

6.1 Extending PyTorch and JAX Implementation

We worked to extend the PyTorch PETS implementation we are building off of. Previously, for Model Predictive Control (MPC), it only had the $TS\infty$ propagation method implemented. This corresponds for following a single ensemble bootstrap for the entire predicted state trajectory. $TS1$ corresponds to using a random bootstrap's prediction at every timestep.

Because of the fact that existing code base for Epinet is only implemented in JAX, yet the backbone of PETS with MBC control is developed with PyTorch, we adopt a hybrid structure to train our model as illustrated in Figure 1. The overall algorithm is summarized in Algorithm 1.

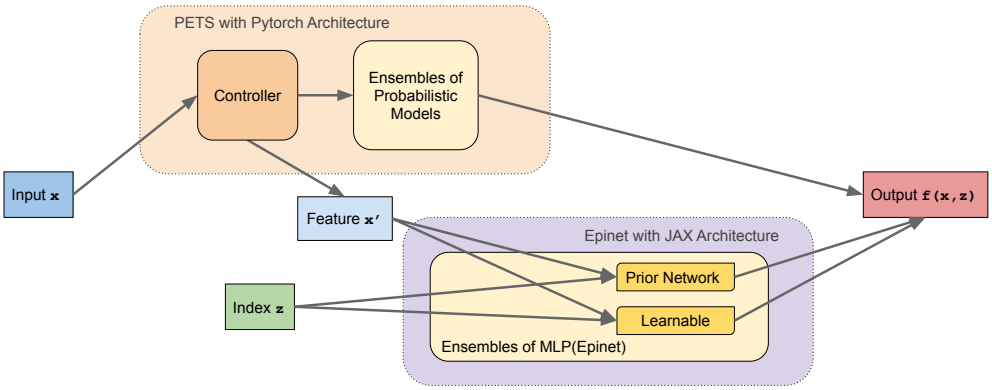


Figure 1: PETS+Epinet Architecture

Algorithm 1 PETS + Epinet

Sample trajectories $\mathcal{D}$ with random controller. ▷ MPC controller
for Trials $k = 1...K$ **do**
Train model $\mathcal{P}$ using PETS + Epinet Architecture
end for
for Time $t = 1...T$ **do**
  for Action $a_i, i = 1...N$ sampled through CEM **do** ▷ CEM stands for Cross Entropy Method
    `Propagate state` s_i `using boot-strapping`
    `Evaluate Action as` $\hat{p}_{\boldsymbol{\theta}} = \frac{1}{B}\sum_{b=1}^{B} \hat{p}_{\boldsymbol{\theta}_b}(s_i, a_i)$ `Update the CEM distribution`
  end for
  `Take only the optimal action` a_t^* `and update` $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t^*, s_{t+1}\}$
end for

7 Problems with Epinet

In many applications in uncertainty-aware ML, Epinet's allow significant speedup by reducing the size of the neural net used for sampling the different bootstraps. However, when applied to PETS uncertainty sampling, as demonstrated in Figure 2, we cannot take advantage of this speedup. During trajectory sampling, at every step, the transition model requires a different state/action pair. In the case of an Epinet, this would require recomputing the main feature net for every sample, so that an Epinet would not speed up sampling of different transitions from the transition model.

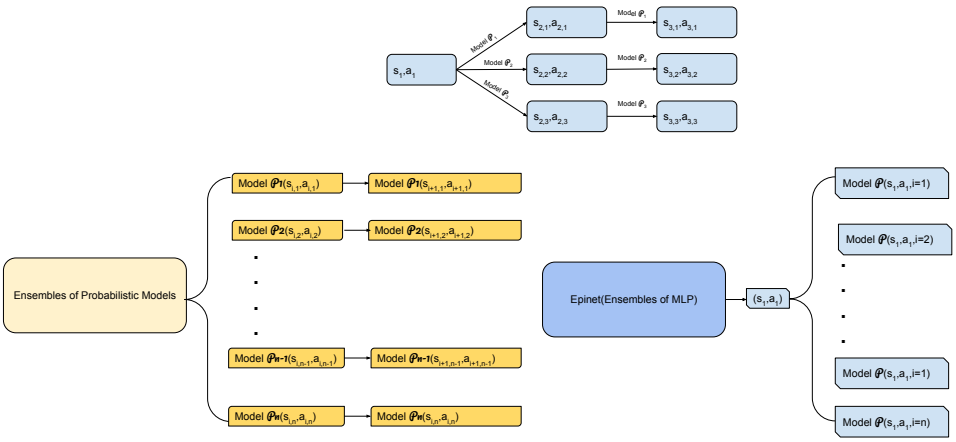


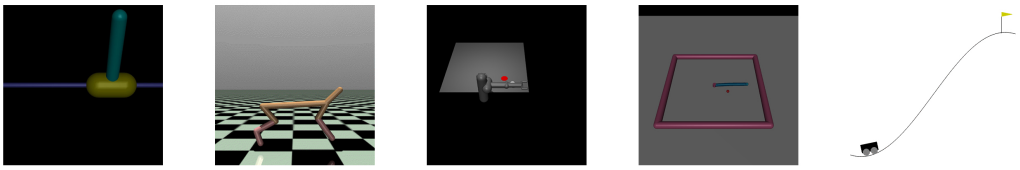
Figure 2: Problems with ensemble in Epinet

8 Experiments

8.1 Matching Baseline Experiments

To start, since we are building off of the unofficial pytorch implementation of PETS, we first made sure that we could match the baseline results in the PETS Chua et al. [2018]. We were able to match the baselines in Mujoco environments, as well as test on new environment of Gym Mountain Car.

Figure3 illustrates the environments we used for our experiments. First four are Mujoco Environments Cartpole, Halfcheetah, Pusher, and Reacher respectively. Mujoco refers to Multi-Joint dynamics with Contact physical engine that is widely used for research in robotics and numerous other areas Todorov et al. [2012]. We used the implementations from Gym, a RL library developed by OpenAI Brockman et al. [2016], to unify the experiments. The last one is Gym Mountain Car, which we believe is suitable for evaluating the agent's ability to explore efficiently.



(a) Cartpole (b) Halfcheetah (c) Pusher (d) Reacher (e) Mountain Car

Figure 3: Environments Our Experiments Run On

- **Cartpole** or Inverted Pendulum involves a cart with a pendulum placed on top which moves along a pole with one end fixed and the other end free. The goal is to balance the pole as long as possible.
- **HalfCheetah** is a simulated quadrupal robot consisting of 9 links and 8 joints, the goal is to make the robot balance itself and run forward as fast as possible.
- **Reacher** is a robot arm consisting of two joints. The robot is fixed at one end and the goal is to move the fingertip of the free end to reach a target at a random location.
- **Pusher** is a multi-jointed robot that is similar to a human arm. The goal is to move a cylindrical target to a designated location using its "hand".
- **Mountain Car** is a classic control environment where a toy car is put in a sinusoidal valley and the goal is to reach the target on the top of the right hill as soon as possible.

In order to demonstrate the validity of our approach, we want to make PETS+Epinet algorithm match the expected return during evaluation of that of vanilla PETS published by Chua et al. [2018], under epistemic rewards, which will be explained further in Section 8.2. However, our implementation fails to converge as demonstrated in Figure 4 through the Cartpole task due to the difficulties mentioned in Section 7. The vanilla PETS achieved high return without epistemic rewards and epistemic rewards worsens the agent's performance, but Epinet fails to learn.

HalfCheetah and Reacher follows a similar pattern to Cartpole, as reflected on Figure 5. It is worth noting that while the overall performance of Epinet pales in comparison to PETS, it started slightly higher for both environments. This could be an outcome of the simple fact that Epinet is a much smaller than the ensemble used in vanilla PETS. Overall, the performance of HalfCheetah and Reacher, along with Cartpole, corroborates our analysis on the ensemble in Section 7.

Pusher and Mountain Car, shown in Figure 6, are two special cases, in both of which PETS fails to make or maintain significant improvement on the expected return. In Pusher, PETS and Epinet performed similarly but in Mountain Car the results are more fascinating. It wouldn't be reasonable to fully conclude from this single observation that the Epinet is making the agent to conduct more efficient exploration. However, it does draws out the peculiarity of PETS starting to have drops in performance after many iterations of training.

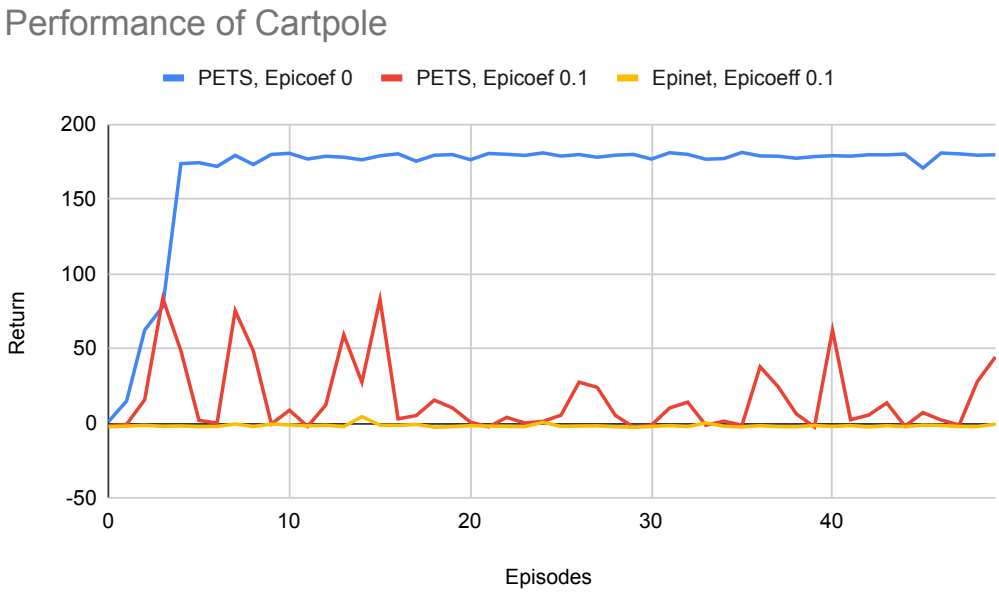
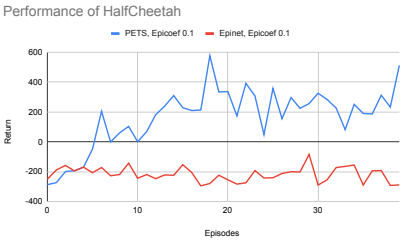


Figure 4: Evaluation return on Cartpole

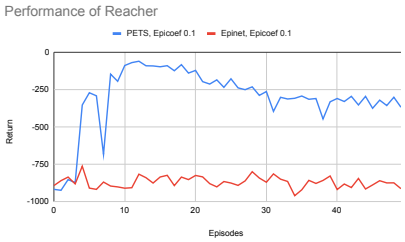
8.2 Epistemic Reward Experiments

Given that epistemic uncertainty is error which can be reduced with additional training data, we hypothesized that incentivising visiting high epistemic error state/action pairs would expedite the learning process for the transition dynamics model. In our experiments so far, however, this reward worsened the learning process. An example of performance with and without the epistemic reward can be seen in figure 7.

We are currently investigating why incentivising exploration of high-epistemic state pairs could cause worsened performance. One hypothesis we are currently investigating is that the epistemic reward causes the training distribution of states to be far from the optimal policy's state distribution which would then cause catastrophic forgetting in the transition model's predictions for the state transitions encountered via the optimal policy. We can see that as training progresses the agent without epistemic reward reaches an asymptotic max performance whereas the agent with epistemic reward continually degrades.

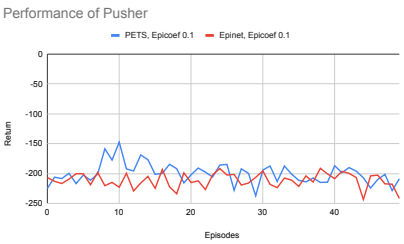


(a) HalfCheetah

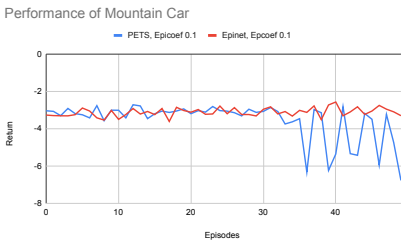


(b) Reacher

Figure 5: Evaluation return on HalfCheetah and Reacher



(a) Pusher



(b) Mountain Car

Figure 6: Evaluation return on Pusher and Mountain Car

Another issue with the epistemic reward formulation is that it requires computing the prediction for every bootstrap in order to get the reward for a given transition. This increases the computational expense of model-based planning with b bootstraps by a factor of b. However, we plan that our implementation of EpiNets for the ensemble in the future will allow us to reduce this expense to significantly less than a proportional relationship. This is because adding b bootstraps in an epinet less than proportionally increases compute cost.

9 Conclusion and Future Work

We extended PETS with epistemic uncertainty quantification and an Epinet. The uncertainty quantification was able to be sped-up with the implementation of the Epinet. However, our experiments did not show performance improvements by implementing these methods in the PETS algorithm.

PETS, while powerful on many tasks, still fails on certain tasks that involves efficiency on agent's ability to explore, which is heavily affected by epistemic uncertainty. Although we intend to tackle this issue by extending Epinet to PETS and introducing epistemic rewards, our model fails to learn decently.

In future work, we believe that certain issues with our implementation could be resolved to ultimately improve the PETS algorithm. First, the method would likely benefit from learning methods to make it less effected by distributional shift from active exploration. Additionally, the uncertainty quantification might be useful outside of improving exploration in the learning phase. For example, the uncertainty quantification may be useful for reducing uncertainty during the evaluation phase when planning. Finally, the intrinsic reward implemented in this work only contained an addition of the uncertainty quantification, when a different method of incorporating uncertainty in the reward function may be more performant.

References

Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

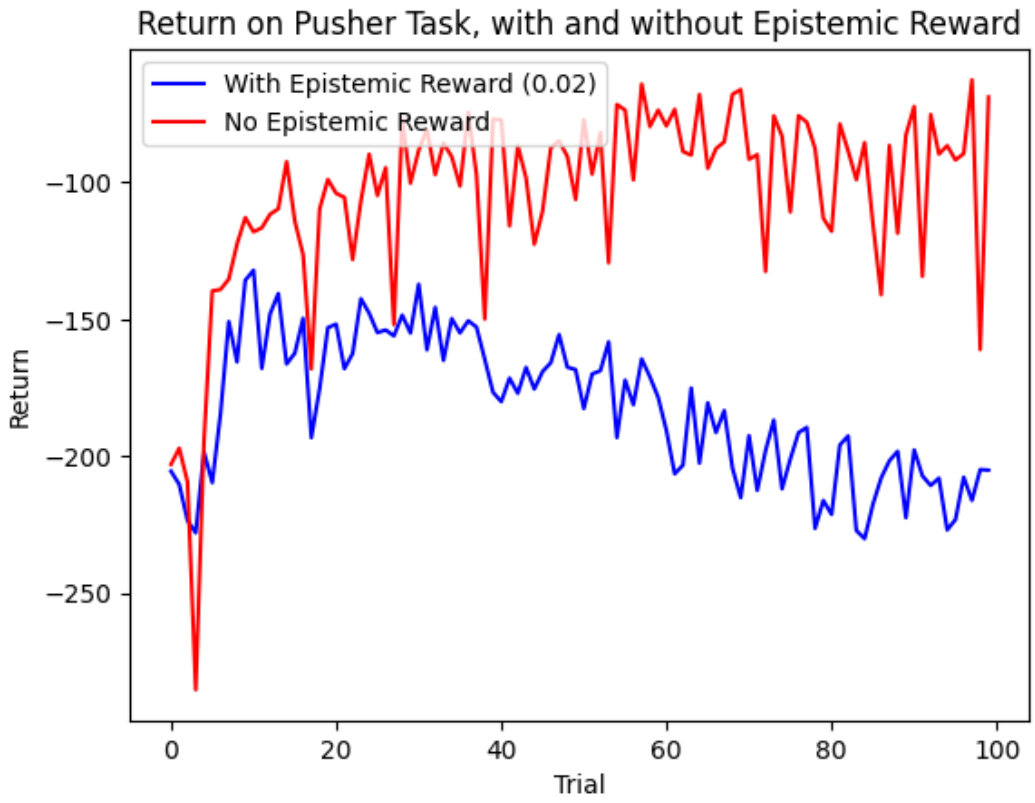


Figure 7: Pusher Task Performance With/Without Epistemic Reward

Glen Berseth, Daniel Geng, Coline Devin, Nicholas Rhinehart, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Smirl: Surprise minimizing reinforcement learning in unstable environments, 2021.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 465–472, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Yarin Gal. Uncertainty in deep learning. 2016.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2021.

M.V. Kothare, V. Nevistic, and M. Morari. Robust constrained model predictive control for nonlinear systems: a comparative study. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 3, pages 2884–2885 vol.3, 1995. doi: 10.1109/CDC.1995.478579.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf`.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 1071–1079, Cambridge, MA, USA, 2014. MIT Press.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks, 2017.

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees, 2021.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

Ian Osband, Zheng Wen, Seyed Mohammad Asghari, Vikranth Dwaracherla, Morteza Ibrahimi, Xiuyuan Lu, and Benjamin Van Roy. Epistemic neural networks, 2023.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Donald F Specht. Probabilistic neural networks. *Neural networks*, 3(1):109–118, 1990.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL `https://doi.org/10.1145/122344.122377`.

E. Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306 vol. 1, 2005. doi: 10.1109/ACC.2005.1469949.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

T. Tsuji, H. Ichinobe, O. Fukuda, and M. Kaneko. A maximum likelihood neural network based on a log-linearized gaussian mixture model. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 3, pages 1293–1298 vol.3, 1995. doi: 10.1109/ICNN.1995.487343.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning, 2019.