Correlated Trajectory Uncertainty for Adaptive Sequential Decision Making

Ian Char[1*], **Youngseog Chung**[1*], **Rohan Shah**[1], **Willie Neiswanger**[2,3], and **Jeff Schneider**[1,4]

[1]*Machine Learning Department, Carnegie Mellon University*
[2]*Computer Science Department, Stanford University*
[3]*Department of Computer Science, University of Southern California*
[4]*Robotics Institute, Carnegie Mellon University*

1. Introduction

One of the great challenges with decision making tasks on real world systems is the fact that data is sparse and acquiring additional data is expensive. In these cases, it is often crucial to make a model of the environment to assist in making decisions. At the same time, limited data means that learned models are erroneous, making it just as important to equip the model with good predictive uncertainties. In the context of learning sequential decision making policies, these uncertainties can prove useful for informing which data to collect for the greatest improvement in policy performance (Mehta et al., 2021, 2022) or helping the policy identify and avoid regions of state and action space that are uncertain during test time (Yu et al., 2020). Additionally, assuming that realistic samples of the environment can be drawn, an adaptable policy can be trained that attempts to make optimal decisions for any given possible instance of the environment (Ghosh et al., 2022; Chen et al., 2021).

In this work, we examine the so-called "probabilistic neural network" (PNN) model that is ubiquitous for learning a transition function in model-based reinforcement learning (MBRL) works. We argue that while PNN models may have good marginal uncertainties, they form a distribution of non-smooth transition functions. Not only are these function samples unrealistic and may hamper adaptability, but we also assert that this leads to poor uncertainty estimates when predicting multiple step trajectories. To address this, we propose a simple sampling method that can be implemented on top of pre-existing models. We evaluate our sampling technique on a number of control environments, including a realistic nuclear fusion task. Not only do smooth transition function samples produce more calibrated uncertainties, but they also lead to better downstream performance for an adaptive policy.

2. Method

Preliminaries. In this work, we focus on finding optimal policies for infinite-horizon Markov Decision Processes (MDPs). We define the following MDP, $\mathcal{M} := (\mathcal{S}, \mathcal{A}, r, T, T_0, \gamma)$. $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the set of actions that can be played at any state; $r : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}$

$*$. Equal contribution author.

is the reward function over current state, action, and next state; $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function; $T_0 \subset \Delta(\mathcal{S})$ is the initial state distribution; and γ is the discount factor. $\Delta(\mathcal{S})$ and $\Delta(\mathcal{A})$ denotes the class of probability distributions over the state and action space, respectively, and we assume that $\mathcal{S} \subset \mathbb{R}^D$. Also, in this work we constrain the transition function to be deterministic. Our goal is to learn a policy function, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the objective $J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1})\right]$, where $a_t \sim \pi(s_t)$, $s_{t+1} = T(s_t, a_t)$, and the expectation is over randomness in initial state and policy actions. We focus on deep reinforcement learning algorithms in which π is a neural network.

One difficulty that arises with deep reinforcement learning methods is the large number of samples needed to learn a good policy. Model-based reinforcement learning (MBRL) methods alleviates this by also learning a model of the environment, $\hat{T}$. One can then reduce the number of samples needed from the true MDP by supplementing the data with fictitious samples generated using $\hat{T}$ and π. For notational convenience, let $\mathcal{X} := \mathcal{S} \times \mathcal{A}$ be the space of concatenated state-action pairs. A trajectory in the MDP of length H can then be written as $(x_1, x_2, \ldots, x_H)$, where $x_t := (s_t, a_t) \in \mathcal{X}$, $s_t = T(x_{t-1})$, $a_t \sim \pi(s_t)$, and $s_1 \sim T_0$. Instead of learning the transition to the next state $x_t \to s_{t+1}$, in practice one usually learns the state delta: $x_t \to s_{t+1} - s_t$, such that the learned model $f_\theta : \mathcal{X} \to \Delta(\mathbb{R}^D)$ can predict the next state as $\hat{T}(x_t) = s_t + f_\theta(x_t)$, where θ is the parameters of the model. One can then use the learned model to "rollout" a fictitious trajectory $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_H)$, where $\hat{x}_t = (\hat{s}_t, \hat{a}_t)$, $\hat{s}_t \sim \hat{T}(\hat{x}_{t-1})$, $\hat{a}_t \sim \pi(\hat{s}_t)$, and $\hat{s}_1 \sim T_0$. As a last piece of notational convenience, we use superscripts to denote particular dimensions of vectors. For example, $s_t^{(d)}$ is the d^{th} dimension of s_t and $f_\theta^{(d)}(x)$ is the d^{th} dimension of the model's output.

2.1 The "Probabilistic Neural Network"

One of the first works to emphasize the importance of uncertainty in MBRL was Chua et al. (2018). To model the transition function, this work proposes using an ensemble of so-called "probabilistic neural networks" (PNN), which are models that output predictive distributions instead of point predictions. In particular, they propose learning a PNN that outputs a mean vector and diagonal covariance matrix which parameterize a multivariate Gaussian distribution. In the context of our work, when f_θ is such a PNN, we write $f_\theta(x) = (\mu_\theta(x), \sigma_\theta(x))$, where $\mu_\theta : \mathcal{X} \to \mathbb{R}^D$ and $\sigma_\theta : \mathcal{X} \to \mathbb{R}^D$ are the mean and standard deviation functions respectively. We will always assume this form of predictive distribution when referring to a PNN. Since this work, PNN's have been used in many MBRL works, including works from both the online (Janner et al., 2019) and offline (Yu et al., 2020; Chen et al., 2021; Yu et al., 2021) settings.

In their paper, Chua et al. (2018) motivate the ensemble of PNNs by claiming that the predicted variance of a PNN, σ_θ, captures aleatoric uncertainty (i.e. the randomness inherent in the true dynamics of the MDP) and ensembling accounts for epistemic uncertainty (i.e. estimation uncertainty stemming from a finite number of datapoints from the system). However, we claim that in reality this breakdown is not so clear. For one thing, the main experiments presented in Chua et al. (2018) as well as subsequent works that leverage this architecture (Janner et al., 2019; Yu et al., 2020; Chen et al., 2021) test their methods in environments with no aleatoric uncertainty. Despite this absence of aleatoric uncertainty, σ_θ seems to play an important role and is even used for penalization in Yu et al. (2020)

and Chen et al. (2021) to indicate that the policy is leaving the support of the data. It seems that PNNs can play an important role in helping capture epistemic uncertainty, and we demonstrate this empirically in Appendix B with a toy example.

That being said, a PNN can only capture *marginal* uncertainty (i.e. the uncertainty for any single input $x \in \mathcal{X}$), it has no notion of *joint* uncertainty over the input space $\mathcal{X}$. As a result, it is unable to draw smooth samples of the transition function (see Figure 1). Why does this matter? Consider predicting a full trajectory and assume that the true dynamics, T, and the predicted mean function, μ_θ, are Lipschitz smooth. Then, if consecutive inputs x_{t-1} and x_t are close (i.e. $\|x_{t-1} - x_t\|$ is small), we expect the residuals $T(x_{t-1}) - \mu_\theta(x_{t-1})$ and $T(x_t) - \mu_\theta(x_t)$ to also be close. In other words, we often expect there to be temporal correlations in the residuals that stem from the smoothness in the dynamics and policy (we empirically show these correlations exist in Appendix H). This may be problematic for two reasons: first, these correlations in residuals can lead to miscalibrated uncertainty predictions (see Appendix G) and overconfident behavior in downstream policy learning. Second, assuming that an adaptive policy is being trained, these temporal correlations may be key in distinguishing and adapting to different environments.

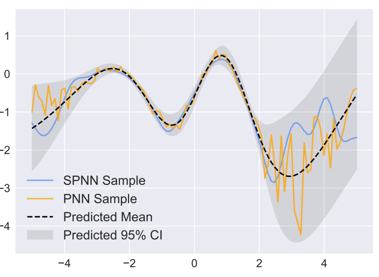

Figure 1: **A visual example of function samples**. The vanilla PNN sample can be seen in orange and a sample from our proposed SPNN method can be seen in blue. Crucially both samples stem from the same mean and standard deviation predictions.

Learning Residual Correlation An ideal model should therefore model uncertainty jointly over $\mathcal{X}$ space, and as a result, be able to sample smooth transition functions that can be used to generate trajectory predictions. For the following, we fix an output dimension d and a time step in the dynamics rollout t. To motivate our method, we first note that the predictive distribution of a PNN can be rewritten as $\mu_\theta^{(d)}(x_t) + \sigma_\theta^{(d)}(x_t)Z_t$ where all of the stochasticity comes from $Z_t \sim \mathcal{N}(0, 1)$. In the vanilla PNN, $Z_1, \ldots, Z_t$ are i.i.d random variables; however, one can instead learn a joint distribution over these random variables. Intuitively, this joint distribution should be such that, when $\|x_i - x_j\|$ is small, the correlation between Z_i and Z_j is high. To achieve this, for each dimenions d, we leverage a kernel function $\kappa_d : \mathcal{X} \times \mathcal{X} \to [0, 1]$. This function indicates the similarity between two points, and we assume that $\kappa_d(x, x) = 1$ in our work. Then, $Z_1, \ldots, Z_t$ are distributed as a multivariate Gaussian distribution with mean 0 and covariance matrix Σ, where $\Sigma_{i,j} = \kappa_d(x_i, x_j)$. The parameters of the kernel function can then be tuned by optimizing the likelihood of the data via a gradient-based optimizer. Because this technique results in smooth transition function samples, we refer to it as the Smooth Probabilistic Neural Network (SPNN).

There are several important details to note about this method. First, because of our assumption on the kernel function, the marginal distribution remains the same as the vanilla PNN; only the smoothness of the function samples will change. Second, since every collection of $Z_1, \ldots, Z_t$ is a multivariate Gaussian random variable, this implies that we are modelling the function of standardized residuals (i.e. $\frac{T(x) - \mu_\theta(x)}{\sigma_\theta(x)}$) as a Gaussian Process

(GP) parameterized by a constant mean function of 0 and kernel function κ_d. Although GPs are often used as a prior over functions, our method is purely frequentist as the GP is fit to the residuals in the dataset by optimizing kernel parameters via maximum likelihood, and a posterior is never computed. That being said, we can take advantage of GP computational efficiencies in our method. As discussed in Rahimi and Recht (2007) and Wilson et al. (2020), function samples from a GP can be approximated as Fourier series. Following this, for every trajectory, we can sample a function $g : \mathcal{X} \to \mathbb{R}^D$ where $g^{(d)}(x) = \sqrt{\frac{2}{B}} \sum_{b=1}^{B} \cos(\phi_{b,d}^T x + \tau_{b,d})$. Here, B is the number of bases in the approximation, $\tau_{b,d} \sim \mathcal{U}(0, 2\pi)$, and $\phi_{b,d} \sim p_{\kappa_d}$ where p_{κ_d} is the spectral density corresponding to kernel κ_d. One can then simply use $g^{(d)}(x_t)$ in place of Z_t. See Appendix C for more details.

3. Experiments

We empirically evaluate the impact of drawing smooth transition function samples. We measure these effects in two ways: first, by assessing how smooth samples affect modeling metrics such as likelihood and calibration under a test set, and secondly, by assessing how smooth samples affect the downstream training of policies.

Environments. We test our method on a number of different environments where there exists some safety critical limit that the agent must avoid. We give brief descriptions of each environment here, but more details can be found in Appendix D. **(Fusion)** We first consider controlling a tokamak for nuclear fusion, an application that has gained interest in the RL community (Degrave et al., 2022; Char et al., 2023; Seo et al., 2021, 2022). In our environment, adapted from Char and Schneider (2023), the goal is to push β_N (the normalized ratio between plasma and magnetic pressure) to be as close to a fixed limit as possible by adjusting the amount of power injected; however, if the limit is exceeded, the episode ends and the agent is given a large negative reward. The static dataset is comprised of five trajectories in which a PID controller was used to hit a static β_N target well below the limit. **(Mountain)** Next, we consider an environment where an agent is tasked with traversing a mountain ridge in a two-dimensional space. Not only can the agent fall off either side of the ridge, but there is also a cliff at the end of the ridge that the agent must get as close as possible to without falling over. To move, the agent has a thruster which can be angled to accelerate the agent in different directions. To form the dataset for this environment, we train an agent on the true environment and collect data at different stages of performance. In total, we consider three variants of datasets which we name Random, Medium, and Expert. Note that Medium is a supser set of Random, and Expert is a super set of Medium. **(Cart Pole)** Finally, we consider the standard environment in which an agent controls a cart with a pole attached and is tasked with balancing the pole while ensuring the cart does not go out of bounds. For this environment, we collect 250 data points with a poor performing policy.

Training For each of the environments listed above, we train an ensemble of five PNNs. Although forming an ensemble by itself does not give the ability to sample smooth transition functions, it can introduce correlation in residuals assuming one chooses and fixes an ensemble member for each trajectory prediction. We found that for all environments (except the mountain ridge environment), each trained PNN was miscalibrated. Thus, we

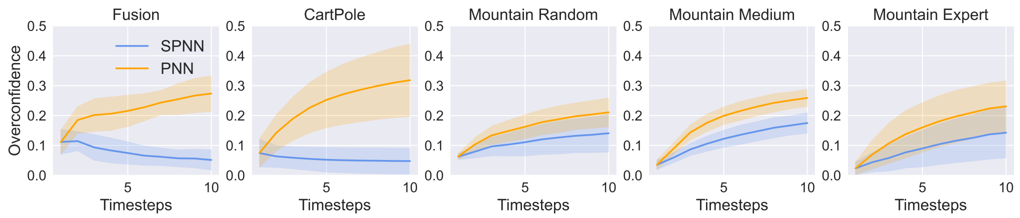

Figure 2: **Overconfidence vs. Timestep**. The plots above show the computed overconfidence by timestep in the rollouts when a single PNN or SPNN is used. The solid line shows the mean overconfidence computed over five random seeds, and the shaded region shows the standard error. We use the in-distribution dataset to compute the metrics for the plots shown here.

use the Uncertainty Toolbox (Chung et al., 2021a) to recalibrate each member by finding a constant scaling for the standard deviation of each output dimension. To learn a policy, we use the Soft Actor Critic (SAC) (Haarnoja et al., 2018) algorithm. Following Ni et al. (2021); Chen et al. (2021), we use a recurrent network for the policy to enable the policy to adapt to different instances of the dynamics. See Appendix E for more details.

Modelling Metric Results We first use uncertainty metrics to evaluate the quality of multi-step trajectory samples generated using our sampling technique. Specifically, we measure the *average calibration of centered prediction intervals at each timestep*. Average calibration (also referred to as probabilistic or quantile calibration) (Gneiting et al., 2007; Kuleshov et al., 2018; Song et al., 2019; Chung et al., 2021b) is a standard metric in uncertainty quantification which measures the average discrepancy between the expected and observed proportion of data covered by a predictive interval. We also report the component of miscalibration (i.e. error in average calibration) stemming from overconfidence - where prediction intervals are too narrow and the observed proportion within the interval is thus less than the expected proportion. We call this metric the *overconfidence*, and we focus on it since it is often preferable to be underconfident, especially in safety-critical tasks.

To compute these metrics, we start by splitting a test set of data into sub-trajectories of length 10 (a sub-trajectory is a contiguous segment of a full trajectory in the dataset that need not include the start of the trajectory). Following this our evaluation procedure is as follows: first, a number of "replay" samples are drawn for each sub-trajectory (i.e. the model is used to predict the sub-trajectory using the same action sequence), these samples are then used to form centered prediction intervals for each sub-trajectory, and finally miscalibration and overconfidence metrics can be computed for each step of the replays. We compute these metrics for both an in-distribution (ID) dataset, collected using the same policy as the train set, and an out-of-distribution (OOD) dataset, collected using an expert policy. Concrete definitions of metrics and more details on evaluation procedure can be found in Appendix F.

Table 1 shows the miscalibrations and overconfidences averaged across time steps for each environment. Especially in the single PNN case, it is clear that adding smoothness dramatically decreases the overconfidence in the PNN. This is apparent in Figure 2, where it is clear that without smoothness, overconfidence grows much faster over time. We also observe that ensembling often helps with uncertainty, especially when evaluating on OOD datasets. While adding smoothness to the samples can cause miscalibration due to underconfidence, we see that the least overconfident models are the ones which couple ensembling and smoothness.

	Method	Fusion	Cart Pole	Mountain Random	Mountain Medium	Mountain Expert
ID	PNN	0.22 $\pm$ 0.01 (0.22)	0.27 $\pm$ 0.01 (0.24)	0.16 $\pm$ 0.01 (0.16)	0.19 $\pm$ 0.01 (0.18)	0.16 $\pm$ 0.02 (0.15)
	SPNN	**0.10 $\pm$ 0.01** (0.08)	**0.14 $\pm$ 0.02** (0.05)	0.11 $\pm$ 0.01 (0.11)	0.12 $\pm$ 0.01 (0.12)	0.12 $\pm$ 0.01 (0.09)
	PNN Ensemble	0.13 $\pm$ 0.01 (0.03)	0.23 $\pm$ 0.01 (0.02)	0.06 $\pm$ 0.00 (0.04)	0.10 $\pm$ 0.01 (0.08)	**0.10 $\pm$ 0.01** (0.05)
	SPNN Ensemble	0.17 $\pm$ 0.01 (**0.01**)	0.29 $\pm$ 0.01 (**0.00**)	**0.05 $\pm$ 0.00** (**0.02**)	**0.09 $\pm$ 0.00** (**0.05**)	0.11 $\pm$ 0.01 (**0.03**)
OOD	PNN	0.35 $\pm$ 0.02 (0.35)	0.32 $\pm$ 0.01 (0.29)	**0.15 $\pm$ 0.02** (0.08)	0.37 $\pm$ 0.01 (0.37)	0.23 $\pm$ 0.02 (0.23)
	SPNN	0.27 $\pm$ 0.02 (0.26)	0.18 $\pm$ 0.02 (0.13)	0.17 $\pm$ 0.01 (0.05)	0.33 $\pm$ 0.01 (0.33)	0.16 $\pm$ 0.02 (0.15)
	PNN Ensemble	0.18 $\pm$ 0.01 (0.16)	**0.16 $\pm$ 0.01** (0.03)	0.18 $\pm$ 0.01 (0.06)	0.28 $\pm$ 0.00 (0.28)	0.14 $\pm$ 0.02 (0.11)
	SPNN Ensemble	**0.17 $\pm$ 0.01** (**0.14**)	0.19 $\pm$ 0.01 (**0.01**)	0.20 $\pm$ 0.01 (**0.04**)	**0.26 $\pm$ 0.00** (**0.26**)	**0.12 $\pm$ 0.01** (**0.07**)

Table 1: **Miscalibrations and Overconfidences.** The table shows the miscalibration averaged over time steps for each method of sampling and environment. In addition we show the standard error over the five seeds and the proportion of the miscalibration due to overconfidence (in parentheses). All of these quantities are rounded to two digits. We bold the lowest mean miscalibration and overconfidence for each of the environments. The top and bottom blocks show the metrics computed over in-distribution (ID) and out-of-distribution (OOD) datasets, respectively.

Method	Fusion	Cart Pole	Mountain Random	Mountain Medium	Mountain Expert	Average
NN	65.95 $\pm$ 1.63	**100 $\pm$ 0.00**	63.57 $\pm$ 9.80	26.79 $\pm$ 0.62	49.73 $\pm$ 2.50	61.21
PNN	65.34 $\pm$ 12.94	99.98 $\pm$ 0.02	64.29 $\pm$ 4.01	23.39 $\pm$ 1.34	44.34 $\pm$ 13.45	59.47
SPNN	**91.46 $\pm$ 2.50**	98.78 $\pm$ 1.22	**65.93 $\pm$ 1.72**	**39.08 $\pm$ 8.24**	**84.72 $\pm$ 4.73**	**75.99**

Table 2: **Normalized Policy Returns.** Each of the reported numbers is averaged over the last 20% of recorded evaluation episodes during training and five random seeds. We also report the standard errors from the five seeds, and we bold the result with the highest mean. All numbers are normalized using the performance of a poor and expert policy (i.e. a normalized score of 100 is the same performance as the expert policy).

Policy Performance We now turn to examining the performance of an adaptive policy trained with and without smooth samples. For this section we include an additional baseline: an ensemble of neural networks outputting a point prediction (we refer to this as NN). Table 2 shows the returns achieved by the policy averaged over the last 20% of training steps. For the fusion environment and all configurations of the Mountain Ridge environment, we see that it is beneficial to have smooth samples and that this better sampling prevents the final policy from crossing the β_N limit or falling off a cliff for each respective environment. Interestingly, we see that the medium version of the mountain ridge environment is the most difficult to optimize. We hypothesize this may be because the medium version of the dataset has a greater spread of data so the dynamics are more confident yet do not have enough data to fully model the dynamics. This is therefore a case where it is especially important that there are smooth samples of the dynamics, and we visually show the difference in policy performance in Figure 21 in the Appendix. Lastly, while there seems to be little difference between the final scores in Cart Pole, in Figure 17, we show that the returns while training are much more reliable when using smooth dynamics samples. We hypothesize this may be due to the smooth dynamics being more controllable and easier to adapt to.

Discussion. To summarize, in this work we highlighted the existence of correlated errors in dynamics models and how sampling smooth trajectory functions is key to capture this phenomenon in uncertainty estimates. Experimentally, we show that with more intelligent sampling we can achieve less overconfident uncertainty predictions and better performing policies. In future works, we hope to extend our results to more complex and higher dimensional environments.

References

MD Boyer, KG Erickson, BA Grierson, DC Pace, JT Scoville, J Rauch, BJ Crowley, JR Ferron, SR Haskey, DA Humphreys, et al. Feedback control of stored energy and rotation with variable beam energy and perveance on diii-d. *Nuclear Fusion*, 59(7):076004, 2019.

Ian Char and Jeff Schneider. Pid-inspired inductive biases for deep reinforcement learning in partially observable control tasks. *arXiv preprint arXiv:2307.05891*, 2023.

Ian Char, Joseph Abbate, László Bardóczi, Mark Boyer, Youngseog Chung, Rory Conlin, Keith Erickson, Viraj Mehta, Nathan Richner, Egemen Kolemen, et al. Offline model-based reinforcement learning for tokamak control. In *Learning for Dynamics and Control Conference*, pages 1357–1372. PMLR, 2023.

Xiong-Hui Chen, Yang Yu, Qingyang Li, Fan-Ming Luo, Zhiwei Qin, Wenjie Shang, and Jieping Ye. Offline model-based adaptable policy learning. *Advances in Neural Information Processing Systems*, 34:8432–8443, 2021.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Youngseog Chung, Ian Char, Han Guo, Jeff Schneider, and Willie Neiswanger. Uncertainty toolbox: an open-source library for assessing, visualizing, and improving uncertainty quantification. *arXiv preprint arXiv:2109.10254*, 2021a.

Youngseog Chung, Willie Neiswanger, Ian Char, and Jeff Schneider. Beyond pinball loss: Quantile methods for calibrated uncertainty quantification. *Advances in Neural Information Processing Systems*, 34:10971–10984, 2021b.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602 (7897):414–419, 2022.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Markus Deserno. How to generate exponentially correlated gaussian random numbers. *Department of Chemistry and Biochemistry UCLA, USA*, 2002.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, pages 7513–7530. PMLR, 2022.

Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. *arXiv preprint arXiv:1807.00263*, 2018.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Viraj Mehta, Biswajit Paria, Jeff Schneider, Stefano Ermon, and Willie Neiswanger. An experimental design perspective on model-based reinforcement learning. *arXiv preprint arXiv:2112.05244*, 2021.

Viraj Mehta, Ian Char, Joseph Abbate, Rory Conlin, Mark Boyer, Stefano Ermon, Jeff Schneider, and Willie Neiswanger. Exploration via planning for information about the optimal trajectory. *Advances in Neural Information Processing Systems*, 35:28761–28775, 2022.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

JT Scoville, DA Humphreys, JR Ferron, and P Gohil. Simultaneous feedback control of plasma rotation and stored energy on the diii-d tokamak. *Fusion engineering and design*, 82(5-14):1045–1050, 2007.

Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. *arXiv preprint arXiv:2203.09168*, 2022.

J Seo, Y-S Na, B Kim, CY Lee, MS Park, SJ Park, and YH Lee. Development of an operation trajectory design algorithm for control of multiple 0d parameters using deep reinforcement learning in kstar. *Nuclear Fusion*, 62(8):086049, 2022.

Jaemin Seo, Y-S Na, B Kim, CY Lee, MS Park, SJ Park, and YH Lee. Feedforward beta control in the kstar tokamak by deep reinforcement learning. *Nuclear Fusion*, 61(10): 106010, 2021.

Hao Song, Tom Diethe, Meelis Kull, and Peter Flach. Distribution calibration for regression. In *International Conference on Machine Learning*, pages 5897–5906. PMLR, 2019.

ITER Physics Expert Group on Confinement Transport, , ITER Physics Expert Group on Confinement Modelling Database, , and ITER Physics Basis Editors. Chapter 2: Plasma confinement and transport. *Nuclear Fusion*, 39(12):2175–2249, 1999.

James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from gaussian process posteriors. In *International Conference on Machine Learning*, pages 10292–10302. PMLR, 2020.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

Appendix A. Related Works

Reinforcement Learning The experimental setting considered in this work falls under the so-called "offline" reinforcement learning setting (Levine et al., 2020). In this setting, a policy must be learned from a fixed dataset of environment interactions and additional environment queries cannot be made. In this setting, Yu et al. (2020) the MOPO algorithm that uses an ensemble of PNNs and relies on penalties to make sure the policy does not steer out of the offline dataset's support. Chen et al. (2021) extend this idea in the algorithm MAPLE, which incorporates an adaptive policy into the learning procedure. Our reinforcement learning experiments are similar to the set up in MAPLE, except we do full episodes from start states (drawn from the start distribution which is assumed to be known). In contrast, MAPLE does short rollouts of 10 steps starting states randomly selected in the offline dataset.

Gaussian Processes in RL Hypothetically, one could use a GP for the dynamics model and draw posterior samples when generating rollouts with the policy, and this has been done for several low dimensional tasks by previous works (Deisenroth and Rasmussen, 2011; Mehta et al., 2021, 2022) However, the non-parametric nature of GPs makes scaling up to higher dimensional tasks a challenge. Perhaps an even greater problem comes from the fact that these posterior samples are much more computationally expensive than their PNN alternative. Because algorithms such as MBPO, MOPO, and MAPLE require millions if not billions of model samples to train a neural network policy to convergence, GPs are often too big of a computational burden to use.

Appendix B. PNN Toy Example

To help guide intuition on why the PNN is useful even in deterministic environments, consider a toy regression problem in which we wish to model the function $f(x) = \cos(3x)$. Our training dataset consists of 100 (X, Y) data points where X is distributed as an exponential random variable. As such, there will be a high concentration of X data around 0, but the concentration of training data quickly tapers off. We train a PNN on this toy problem and show the results in Figure 3. As one would hope, the PNN is confident in regions where data is plentiful, and the model produces wide predictive distributions in regions lacking data. Why does this happen instead of the network producing highly confident predictive distribution where the mean goes through each training point? We hypothesize that this is due to both the capacity of the network and the property of neural networks to

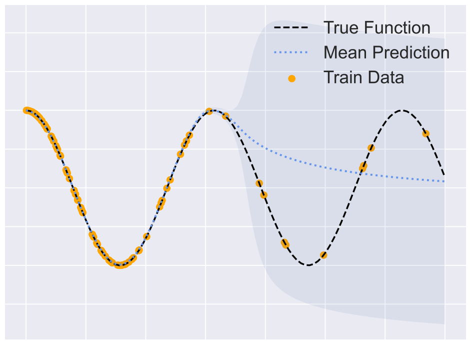

Figure 3: **PNN Trained on a Toy Example**. The orange points make up the training dataset, the black dashed line is the true function, and the dotted blue line shows predicted mean. The blue shaded region shows three standard deviations of the predicted Gaussian distribution. We also use a validation set with 20 points to know when to stop training the model.

produce generally smooth solutions. Similar observations were also made in Seitzer et al. (2022), although they consider the setting in which aleatoric noise truly does exist.

Appendix C. Algorithm Details

Algorithm 1 SPNN Trajectory Sampling

```
1: **Input**: Policy $\pi$, initial state $s_1$, kernels $\{\kappa_d\}_{d=1}^{D}$, horizon $H$, and number of bases $B$.
2: Sample function $g$ by sampling $\phi_{b,d} \sim p_{\kappa_d}$ and $\tau_{b,d} \sim \mathcal{U}(0, 2\pi)$ for $b \in \{1, \dots, B\}$ and $d \in \{1, \dots, D\}$.
3: **for** $t \leftarrow 1, \dots H$ **do**
4:     $a_t \sim \pi(s_t)$
5:     $x_t \leftarrow (s_t, a_t)$
6:     $s_{t+1} \leftarrow s_t + \mu_\theta(x_t) + \sigma_\theta(x_t) g(x_t)$
7: **end for**
8: **Return** $(x_1, \dots, x_H)$
```

Appendix D. Environment Details

Nuclear Fusion Environment As stated in the main body of the paper, this environment is adapted from Char and Schneider (2023). This environment uses equations described in Boyer et al. (2019) and Scoville et al. (2007). In particular, we use the following relations for stored energy, E, and rotation, v_{rot}:

$$\begin{aligned} \dot{E} &= P - \frac{E}{\tau_E} \\ \tau_E &= C_E I^{0.95} B^{0.15} P^{-0.69} \\ \beta_N &= C_\beta \left(\frac{aB}{I}\right) E \end{aligned}$$

where P is the total power, τ_E is the energy confinement time, I is the plasma current, a is the minor radius, B is the magnetic field, and C_E, C_β are constants set to 200 and 5, respectively. The second of these equations is ITERH-98 scaling (Transport et al., 1999). For our version of the environment, we also fix $I = 10^6$, $a = 0.589$, and $B = 2.75$. Similar to Char and Schneider (2023), we include momentum in the energy update. The equation describing the evolution of the energy is

$$\dot{E}_t = 0.5 \left(P_t - \frac{E_t}{\tau_E}\right) + 0.5\dot{E}_{t-1}$$

The observation space for the environment is three dimensional and consists of the current β_N measurement, the rate of change of β_N, and the current amount of power being injected into the system. The action space is one-dimensional and is simply the change in the power.

We set the β_N limit to be 2.2 , and the reward function is

$$r(\beta_N, a) := \begin{cases} \left(\frac{2.5-|\beta_N-2.2|}{2.5}\right)^2 - \frac{\|a\|}{10} & \beta_N \leq 2.2 \\ -100 & \beta_N > 2.2 \end{cases}$$

where a is the action scaled to be between -1 and 1. We use a horizon length of 100 for each episode.

Mountain Ridge Environment The mountain ridge environment has five-dimensional observations space: $s_t = (x_t, \dot{x}_t, y_t, \dot{y}_t, \theta)$, where x_t is the x position, y_t is the y position, and θ is the angle of the thruster used to propel the agent. The action space is two-dimensional and consists of a^{thrust} and a^{angle}, which controls the amount of thrust and the change in angle of the thruster, respectively. The updates are as follows:

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t \Delta t \\ \dot{x}_{t+1} &= \dot{x}_t + \left(\text{sign}(x_t)x_t^2 + a^{\text{thrust}} \sin(\theta_t)\right) \Delta t \\ y_{t+1} &= y_t + \dot{y}_t \Delta t \\ \dot{y}_{t+1} &= \dot{y}_t + \left(a^{\text{thrust}} \cos(\theta_t) + 0.05 \exp(y_t)\right) \Delta_t \\ \theta_{t+1} &= \text{clip}\left(\theta_t + \frac{\pi}{6} f(a^{\text{angle}}), -\pi, \pi\right) \end{aligned}$$

where the function f is defined as

$$f(x) = \begin{cases} 1 + \exp\left[-12.5\,(x - 0.5)\right] & x > 0 \\ 1 + \exp\left[12.5\,(x + 0.5)\right] & x \leq 0 \end{cases}$$

The reward function is simply $\frac{6+y_t-\|a^{\text{thrust}}\|}{10}$ while the agent is on the cliff. The episode ends and the agent recieves a reward of -100 if $|x_t| > 3$, $y_t < -6$, or $y_t > 5$. We use a horizon length of 200 for each episode.

Appendix E. Additional Training Details

Model Training For each of the dynamics models, we use a network with 2 hidden layers, each with 512 units. We use two separate heads for the mean and standard deviation predictions. We find that we can get better uncertainty by adding an additional hidden layer with 256 units to the standard deviation head. Besides the change in architecture, the learning procedure follows what is done in Chua et al. (2018), and we use the Adam (Kingma and Ba, 2014) optimizer with a learning rate of 3×10^{-4} and a batch size of 64. Lastly, we use 10% of the data as a validation set and early stop based on MSE (although we pick the checkpoint that achieves the best negative log likelihood).

Reinforcement Learning Training Our implementation of SAC with recurrent policies closely follows the implementation given by Ni et al. (2021) and uses a Gated Recurrent Unit (GRU) (Cho et al., 2014). We give hyperparameter settings in Table 3. We train for 100, 250, and 1000 epochs for the Cart Pole, Fusion, and Mountain environments, respectively.

Following other offline model-based reinforcement learning works (Yu et al., 2020; Chen et al., 2021), we add a penalty to the reward to indicate when the policy is going out of distribution. When using PNN models, we use the maximum predicted standard deviation among the ensemble members, and, when using neural networks with point predictions, we use the standard deviation among the mean predictions. As per (Chen et al., 2021) we scale this uncertainty by 0.25.

When simulating episodes with the model during training time, the trajectory can sometimes blow up and predict large values. While this is rare, we find that it helps training stability to cap the velocity components of the state space to reasonable values. Finally, for all methods of sampling, we choose a member of the ensemble to make predictions each episode, and we fix this member for the entire episode.

Hyperparameter	**Value**
Discount Factor	0.99
Learning Rate	3×10^{-4}
Batch Size	256
Target Soft Update Weight	5×10^{-3}
History Lookback Size	64
Exploration Steps per Epoch	1000
Gradient Steps per Epoch	1000

Table 3: **Reinforcement learning hyperparameters.**

Appendix F. Model Metric Details

A widely accepted metric in uncertainty quantification to evaluate the validity of distributional predictions is *average calibration*. Given input covariates X, target variables Y, a predictive distribution with CDF $F_X : \mathcal{X} \to (\mathcal{Y} \to [0,1])$ and its corresponding quantile function $F_X^{-1} : \mathcal{X} \to ([0,1] \to \mathcal{Y})$, F is said to be average calibrated if

$$P\left(Y \leq F_X^{-1}(p)\right) = p, \forall p \in [0,1]. \tag{1}$$

Note that Eq. 1 assesses the validity of the predictive quantile function F_X^{-1}, which is identical to a prediction interval between the probabilities $[0,p]$. We note that centered prediction intervals (e.g. a 95% prediction interval that spans the probabilities $[0.025, 0.975]$) can be more useful in practice, and we assess the average of centered prediction intervals, which is defined as:

$$P\left(F_X^{-1}(0.5 - p/2) \leq Y \leq F_X^{-1}(0.5 + p/2)\right) = p, \forall p \in [0,1]. \tag{2}$$

Miscalibration, i.e. error in average calibration of centered intervals, is then measured as

$$\int_0^1 \mid P\left(F_X^{-1}(0.5 - p/2) \leq Y \leq F_X^{-1}(0.5 + p/2)\right) - p \mid dp. \tag{3}$$

Given a dataset $\{x_i, y_i\}_{i=1}^N$, and a uniform draw of probabilities $\{p_k\}_{k=1}^K \in [0,1]$, miscalibration of centered intervals can be estimated as

$$\frac{1}{K}\sum_{k=1}^{K}\left|\texttt{(empirical coverage at } p_k\texttt{)} - p_k\right|, \tag{4}$$

where $\texttt{(empirical coverage at } p_k\texttt{)}$ is defined as $\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\{F_{x_i}^{-1}(0.5 - p_k/2) \le y_i \le F_{x_i}^{-1}(0.5 + p_k/2)\}$ and $\mathbb{I}$ is the indicator function.

We compute miscalibration of centered intervals at each timestep of a trajectory, where the inputs are the current state-action pairs, and the targets are the state delta: i.e. from Eq. 4, x_i would be the tuple $(s_{i,t}, a_{i,t})$ and y_i would be $s_{i,t+1} - s_{i,t}$. We used 19 equi-spaced probabilities: $\{p_k = \frac{k}{20}\}_{k=1}^{19}$. Since an ensemble does not provide a closed form quantile function, we use empirical quantiles for $F_{x_i}^{-1}$ by generating many trajectories for a single test sequence of states and actions.

To measure overconfidence, we performed the outer summation over probabilities in Eq. 4 only if the empirical coverage was lower than p_k:

$$\frac{1}{K}\sum_{k=1}^{K}\min\left(0, \texttt{(empirical coverage at } p_k\texttt{)} - p_k\right), \tag{5}$$

Appendix G. Ignoring Error Correlation Can Lead to Overconfidence

In this section, we show that under certain assumptions, ignoring the correlation between consecutive residuals leads to overconfident predictions. While these assumptions make major simplifications to the problem, this result still gives insight into why overconfidence may grow over time. In what follows, assume that there is a fixed action sequence $a_1, \ldots, a_N$. The corresponding rollout using the true transition function, T, is then $x_1 \ldots, x_{N+1}$.

Ideally, we would compare this to the distribution of rollouts created by sampling autoregressively from $\hat{T}$, which we assume to be a PNN. However since this distribution is difficult to characterize, we focus on analyzing one-step errors. Towards this end, let $\delta_t := \mu_\theta(x_t) - T(x_t)$ and let $\Delta_N := \sum_{t=1}^N \delta_t$. Although the true amount of error after N steps is hard to reason about because of the predicted sequence's autoregressive nature, Δ_N can be thought of as a proxy. We also make the following assumptions:

1. The sequence of residuals is Markovian, i.e. $p(\delta_t|\delta_1, \ldots, \delta_{t-1}) = p(\delta_t|\delta_{t-1})$.

2. The distribution between consecutive residuals is

$$\begin{bmatrix}\delta_t \\ \delta_{t-1}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}0 \\ 0\end{bmatrix}, \begin{bmatrix}1 & \rho \\ \rho & 1\end{bmatrix}\right)$$

Note that these are assumptions that are made to the true underlying transition function (i.e. that there are correlations between residuals). We will now see that the standard deviation of Δ_N grows faster if positive correlation is present versus if an independence assumption is made (as is usually done with sampling procedure in PNNs).

Proposition 1 *(Deserno (2002)) Following the above assumptions*

$$\Delta_{N+1} = \sqrt{\frac{1+\rho}{1-\rho}} G_{N+1} - \frac{\rho}{1-\rho}\delta_{N+1} + \frac{1-\sqrt{1-\rho^2}}{1-\rho} g_1$$

where $N > 1$, $g_1, \dots, g_{N+1} \overset{i.i.d}{\sim} \mathcal{N}(0,1)$, *and* $G_{N+1} := \sum_{t=1}^{N+1} g_t$.

Note that the first term of Δ_N has standard deviation $\sqrt{\frac{1+\rho}{1-\rho}N}$, and the standard deviation for the rest of the terms does not grow as N increases. In contrast, if one were to model residuals as independent, the standard deviation of the sample trajectories from that model after N steps would be $\sqrt{N}$. Again, the assumptions made here prevent any statement from being made in the actual setting in which trajectories are autoregressively predicted; however, it does give intuition as to why PNNs may produce overconfidence predictive distributions over time.

We now restate the proof from Deserno (2002) for completeness.

Proof

Using the fact that $p(\delta_t|\delta_{t-1} = d) \sim \mathcal{N}(\rho d, 1-\rho^2)$, we can express the sequence in terms of IID samples as follows:

$$\delta_1 = g_1; \qquad \delta_t = \rho\delta_{t-1} + \sqrt{1-\rho^2} g_t$$

We can expand the right definition of δ_t to get the following,

$$\delta_t = \rho^{t-1} g_1 + \sqrt{1-\rho^2} \sum_{i=2}^{t} g_i \rho^{t-i}$$

Summing this up to get Δ_N,

$$\begin{aligned}\Delta_{N+1} &= \sum_{t=1}^{N+1}\left[\rho^{t-1} g_1 + \sqrt{1-\rho^2}\sum_{i=2}^{t} g_i \rho^{t-i}\right] \\ &= g_1 \frac{1-\rho^{N+1}}{1-\rho} + \sqrt{1-\rho^2}\sum_{i=2}^{N+1} g_i \sum_{n=i}^{N+1} \rho^{n-i} \\ &= g_1 \frac{1-\rho^{N+1}}{1-\rho} + \frac{\sqrt{1-\rho^2}}{1-\rho}\left(\sum_{i=2}^{N+1} g_i - \rho \sum_{i=2}^{N+1} g_i \rho^{N+1-i}\right).\end{aligned}$$

The first term in the bracket is $G_{N+1} - g_1$, and the second term can be rewritten with δ_{N+1}.

$$\Delta_{N+1} = \sqrt{\frac{1+\rho}{1-\rho}} G_{N+1} - \frac{\rho}{1-\rho} x_N + \frac{1-\sqrt{1-\rho^2}}{1-\rho} g_1$$

■

Method	**Fusion**	**Cart Pole**	**Mountain Random**	**Mountain Medium**	**Mountain Expert**
Dimension 0	0.69 ± 0.13	0.96 ± 0.04	0.95 ± 0.02	0.94 ± 0.01	0.93 ± 0.02
Dimension 1	0.70 ± 0.04	0.95 ± 0.05	0.93 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
Dimension 2	0.85 ± 0.08	0.96 ± 0.04	0.62 ± 0.10	0.57 ± 0.07	0.51 ± 0.11
Dimension 3	-	0.93 ± 0.07	0.45 ± 0.03	0.66 ± 0.02	0.68 ± 0.07
Dimension 4	-	-	-0.02 ± 0.03	0.13 ± 0.02	0.13 ± 0.03

Table 4: **Empirical Correlations** Each value is the average over five seeeds and five ensemble members. Note entries in the table that are entry are due to the environment being lower dimensional (e.g. Fusion only has three dimensions).

Appendix H. Empirical Correlations

We empirically compute temporal correlation between the residuals for models, specifically ensembles of PNNs trained on all our environments. Consider a rollout in the true dynamics - $(s_0, a_0, r_0), \ldots, (s_n, a_n, r_n)$. For a given ensemble member, let $b_0 = \frac{T(s_0,a_0)-\mu_\theta(s_0,a_0)}{\sigma_\theta(s_0,a_0)}, \ldots, b_n = \frac{T(s_n,a_n)-\mu_\theta(s_n,a_n)}{\sigma_\theta(s_n,a_n)}$ be the corresponding sequence of standardized residuals. We make the assumption that successive residuals b_i, b_{i+1} are sampled from a bivariate gaussian with correlation coefficient ρ, that is $\begin{bmatrix} b_i \\ b_{i+1} \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$. Then, we compute the maximum likelihood estimator, $\hat{\rho}$, based on the observed residuals $b_1, \ldots, b_n$. The estimates in Table 4 are averaged over five seeds and five ensemble members in each corresponding ensemble.

Appendix I. Additional Experimental Results

Additional Calibration Results To better understand how uncertainty estimates change with different sampling procedures, we provide additional plots of the miscalibration and overconfidence metrics. Figures 4- 11 show how both metrics change with respect to time for single models and an ensemble. Figures 12- 15 show how the metrics change with respect to ensemble size. In all of these, it is clear that smooth samples mitigate against overconfidence over time; however, this can also cause uncertainty predictions to be slightly underconfident.

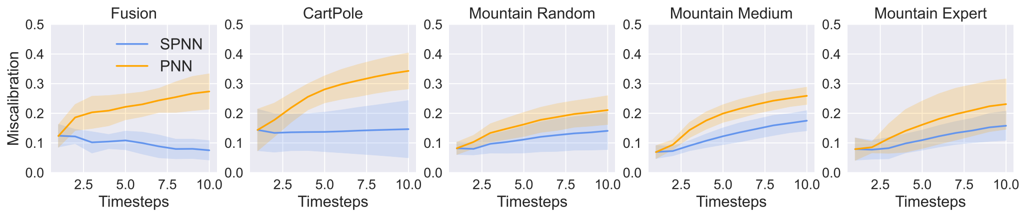

Figure 4: **Average miscalibration for ID data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

Reinforcement Learning Training Curves We also provide plots of the average returns during training of the policy in Figures 16- 20. In general, we see that training with SPNN is less prone to overfitting and often more stable.

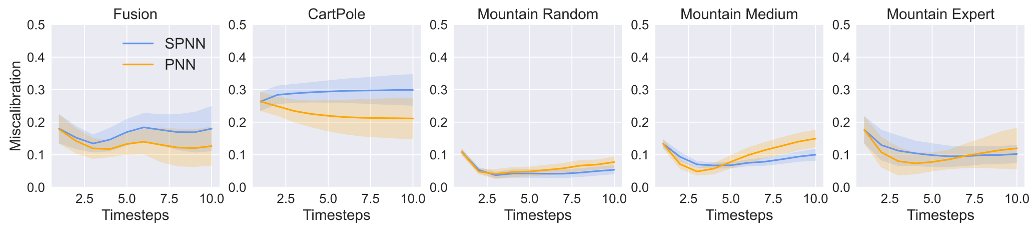

Figure 5: **Average miscalibration for ID data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

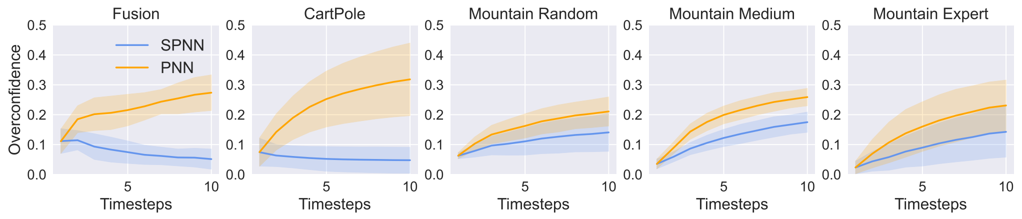

Figure 6: **Average overconfidence for ID data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

Can additional penalty help in the Mountain environment? To prevent the agent from falling off the cliff, it is possible that a higher scale on the penalty could be beneficial. In Table 5, we test what happens when the penalty scaling is changed to 0.0 or 1.0. We find that greatly increasing the penalty 4× does not have same affect on policy performance as intelligent sampling.

Method	Mountain Random	Mountain Medium	Mountain Expert	Average
SPNN Penalty=0.0	61.41 ± 4.47	29.67 ± 0.87	**88.00 ± 1.60**	59.69
PNN Penalty=0.0	63.64 ± 11.41	23.13 ± 1.15	68.77 ± 11.97	51.84
NN Penalty=0.0	31.26 ± 5.68	27.25 ± 0.67	40.71 ± 5.11	33.07
SPNN Penalty=0.25	65.93 ± 1.72	39.08 ± 8.24	84.72 ± 4.73	**63.24**
PNN Penalty=0.25	64.29 ± 4.01	23.39 ± 1.34	44.34 ± 13.45	44.01
NN Penalty=0.25	63.57 ± 9.80	26.79 ± 0.62	49.73 ± 2.50	46.7
SPNN Penalty=1.0	62.05 ± 1.07	**40.45 ± 8.91**	81.78 ± 3.63	61.42
PNN Penalty=1.0	53.06 ± 7.93	30.57 ± 2.33	46.28 ± 4.63	43.3
NN Penalty=1.0	**67.47 ± 10.56**	25.71 ± 0.07	55.07 ± 12.17	49.42

Table 5: **Normalize policy performances for different penalties on the Mountain environment.** Each result is averaged over the last 20% of evaluations during training. Five seeds were used to compute the average scores, and we show the standard errors.

Mountain Environment Visualization To better understand what is happening in the Mountain environment, we plot the average path taken by each type of policy (see Figure 21). While all policies are overconfident and have episdoes where the agent falls off the cliff, on average policies trained with SPNN stay within the support of the dataset and avoid falling off the cliff.

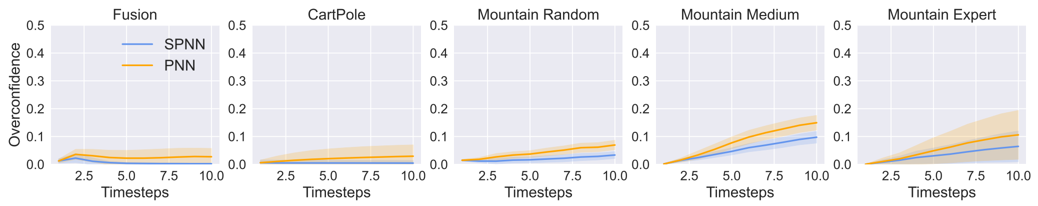

Figure 7: **Average overconfidence for ID data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

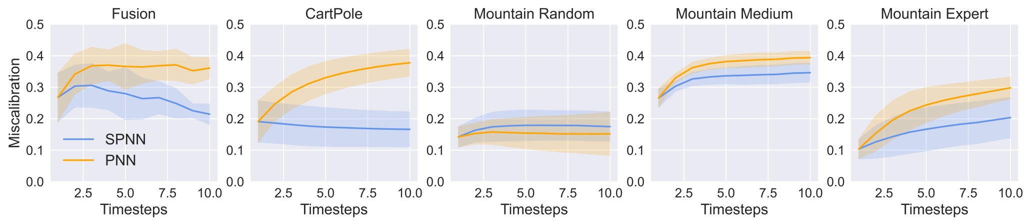

Figure 8: **Average miscalibration for OOD data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

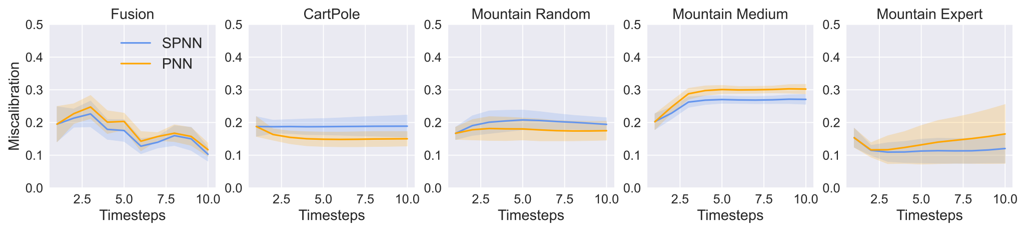

Figure 9: **Average miscalibration for OOD data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

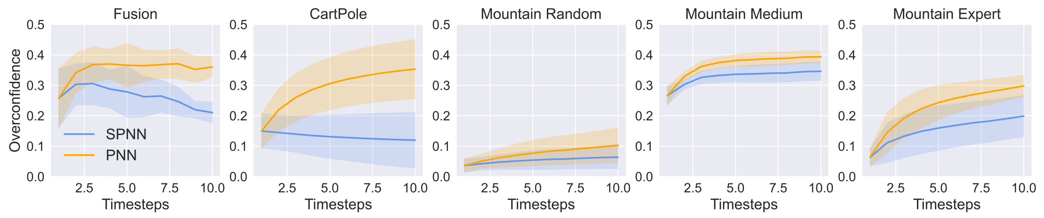

Figure 10: **Average overconfidence for OOD data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

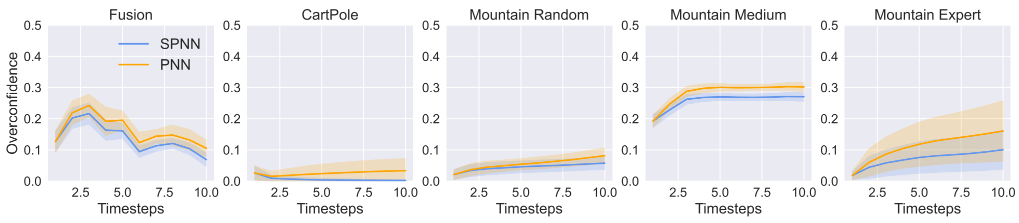

Figure 11: **Average overconfidence for OOD data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

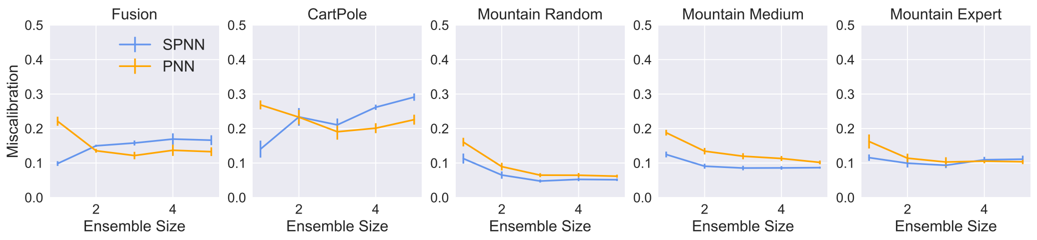

Figure 12: **Average miscalibration over rollout for ID data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

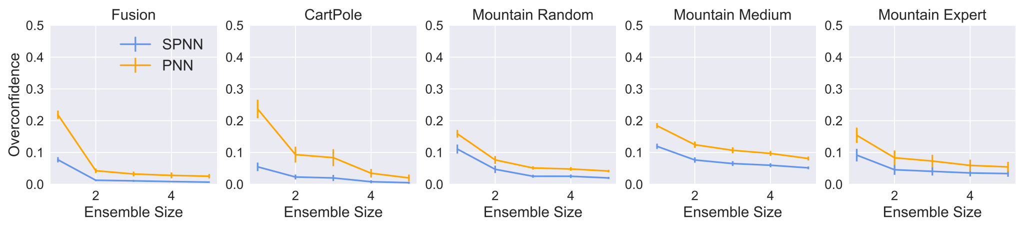

Figure 13: **Average overconfidence over rollout for ID data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

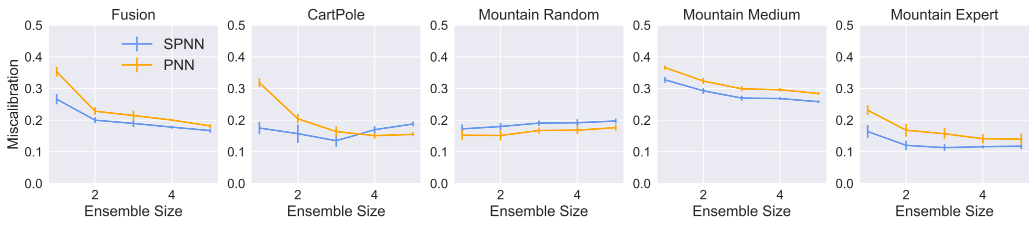

Figure 14: **Average miscalibration over rollout for OOD data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

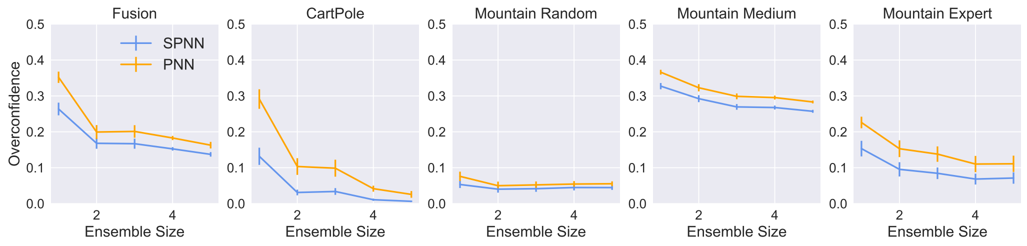

Figure 15: **Average overconfidence over rollout for OOD data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

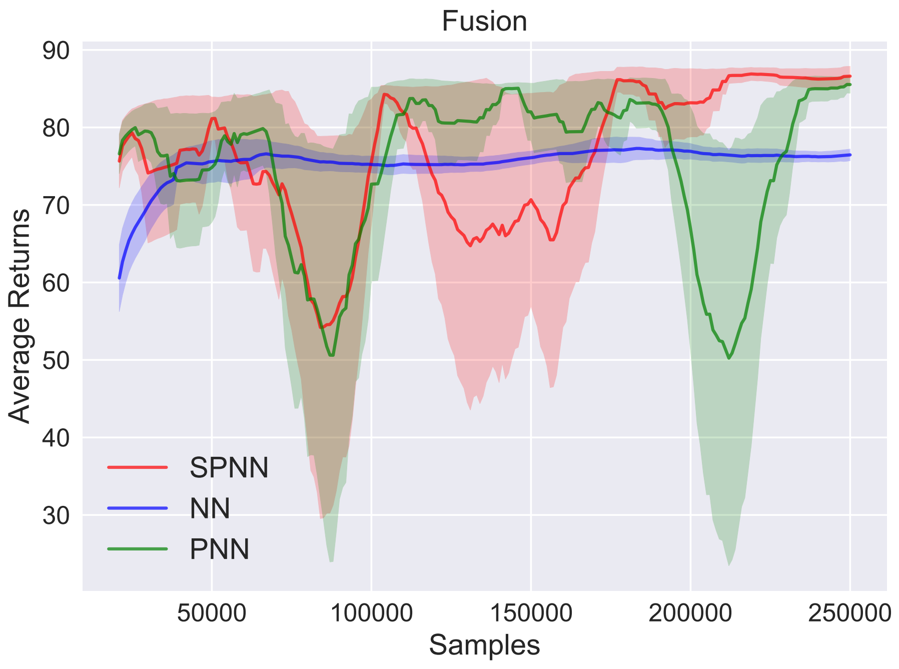

Figure 16: **Average returns for the Fusion environment during training.** The regions shows the standard error over the five seeds.

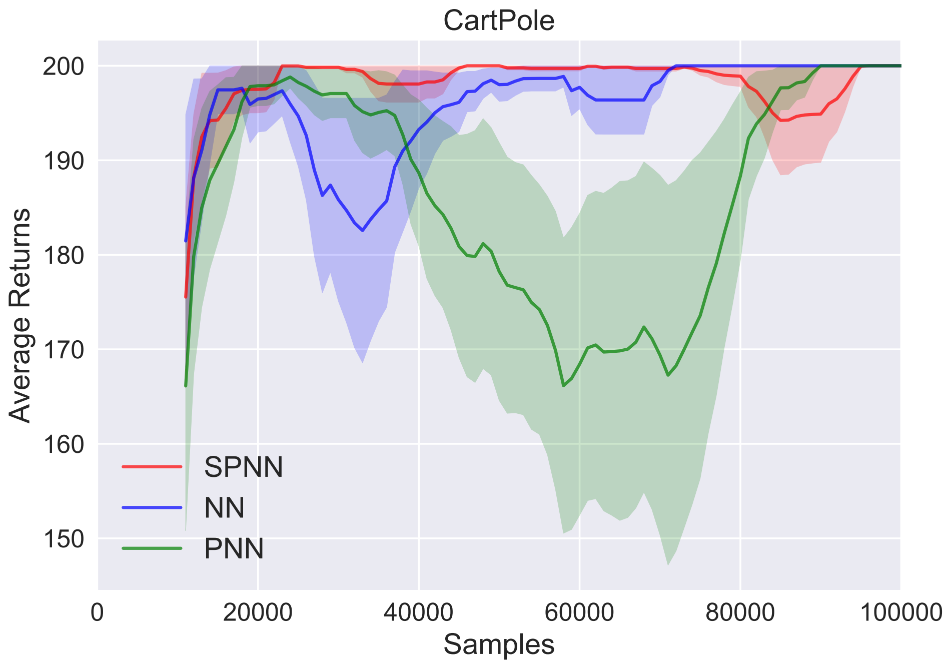

Figure 17: **Average returns for the Cart Pole environment during training.** The regions shows the standard error over the five seeds.

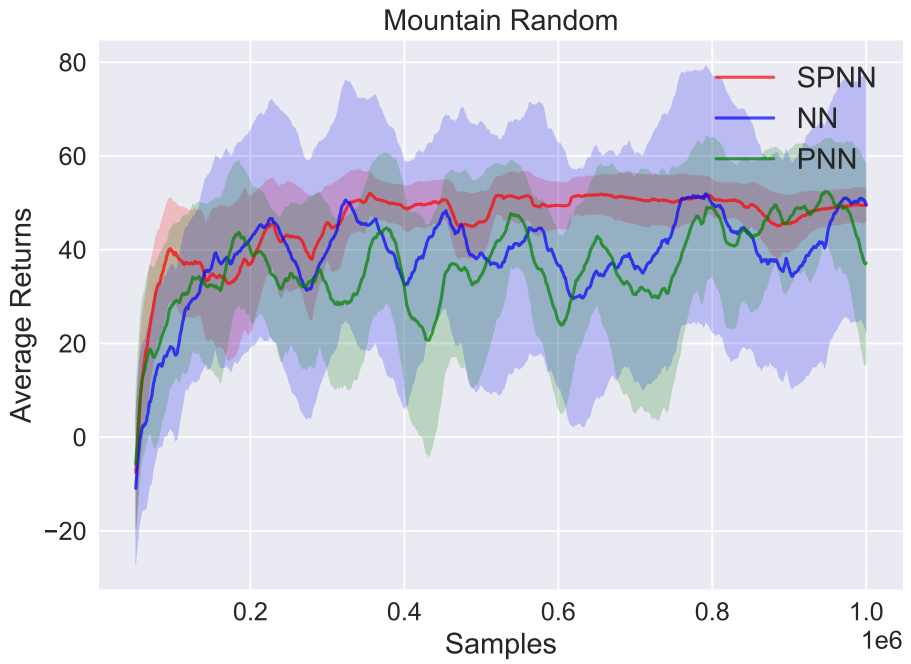

Figure 18: **Average returns for the Mountain Random environment during training.** The regions shows the standard error over the five seeds.

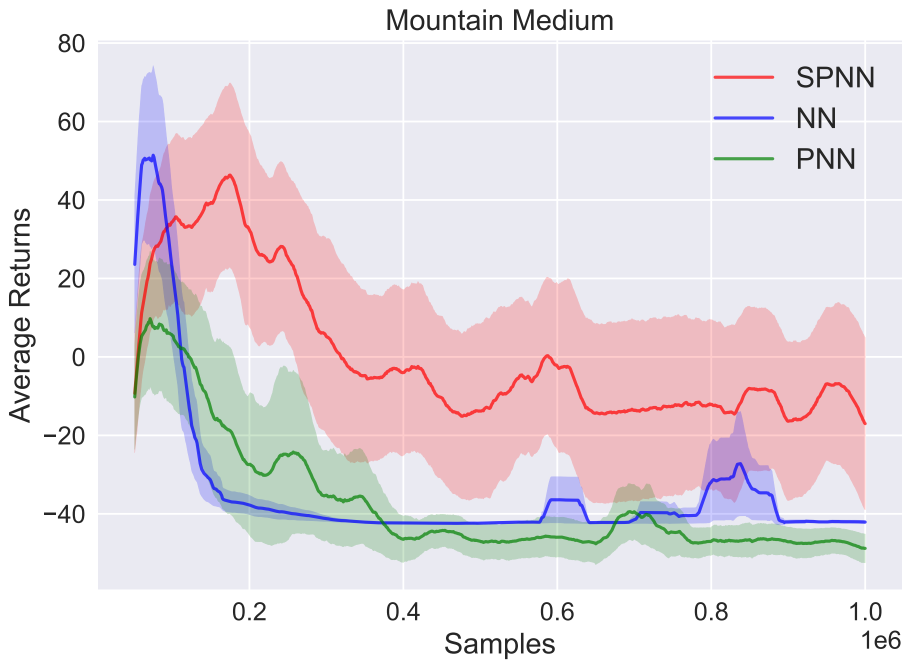

Figure 19: **Average returns for the Mountain Medium environment during training.** The regions shows the standard error over the five seeds.

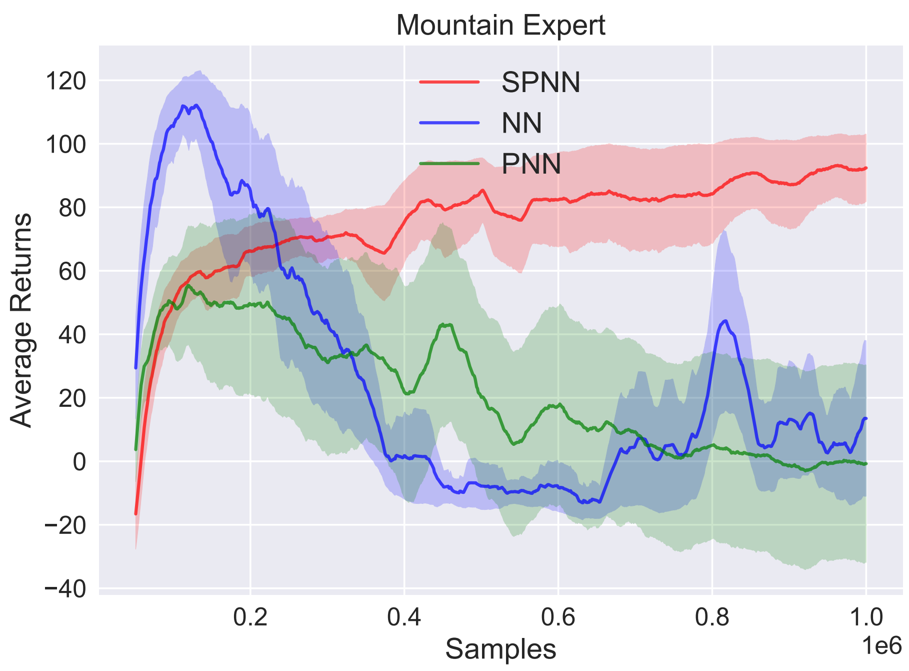

Figure 20: **Average returns for the Mountain Expert environment during training.** The regions shows the standard error over the five seeds.

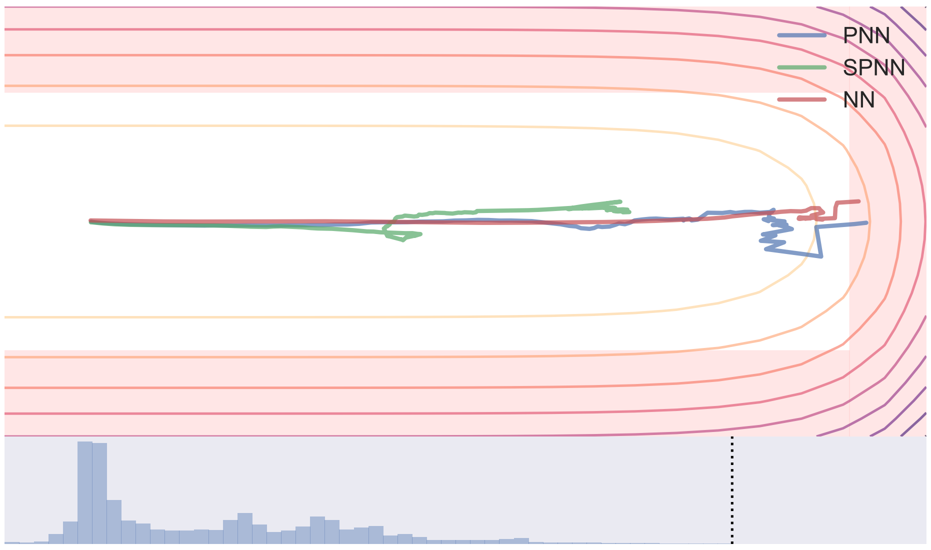

Figure 21: **Average trajectory in medium mountain ridge environment.** To create this figure, we collect 100 paths in the true environment with each of the five policies corresponding to the random seeds. We then average each path together, and the average path can be seen in the top plot. The red regions represent terminal regions where the agent falls off the cliff, and the contour lines show the contours of the mountain ridge. Here, the x-axis shows the y position of the agent as described in the environment definition (Appendix D). The bottom plot shows a histogram of the y data in the Medium dataset, and the dashed black line shows the most extreme recorded y value.