Sequential Bayesian optimal experimental design for non-linear dynamics

Sequential Bayesian optimal experimental design for non-linear dynamics
Master thesis by Markus Semmler
Date of submission: August 23, 2020

1. Review: Boris Belousov
2. Review: Hany Abdulsamad
3. Review: Michael Lutter
4. Review: Prof. Dr. Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Markus Semmler, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 23. August 2020

Markus Semmler

Abstract

This work compares Bayesian models, namely an ensemble with a Bayesian neural network (BNN) and evaluates their performance on different dynamical systems. The environments Pendulum-v0, CartPole-v0, Qube-v0 and MountainCarContinuous-v0 were used for this evaluation. All of these examined systems are one-dimensional. After this, a comparative analysis of experimental design objectives is given and how different methods can estimate them. Two objectives are identified selected for the sequential settings. One objective is the predictive variance, and the other is based on the mutual neural information estimator [1]. Both of them are investigated and therefore plugged into three different optimization problems, one based on discrete optimization, one based on disciplined convex-concave programming (DCCP), and finally merely by using an interior point solver. The learned models are compared on different out-of-sample trajectories to identify the quality of the predictions. It is concluded that the disciplined convex concave programming scheme [2] can efficiently exploit the convexity of the predictive variance. Albeit some good results were obtained, the methods are very sensible to hyper parameter configurations and this topic needs further research. The density of the state space data is evaluated on the two-dimensional Pendulum-v0 environment for the discrete and DCCP algorithm. The resulting predictive models get evaluated exhaustively on trajectories from the validation set. Combining these results with the mutual information estimators yields the full exploration algorithms. They get compared on all environments and additionally are evaluated in a control setting on Pendulum-v0 and CartPole-v0. For the MountainCarContinuous-v0 environment it fails to explore the complete state space. The Qube-v0 is generally hard to solve in model predictive control and thus not considered in the control setup. At the end of the thesis the results are reviewed and an overview of possible further research directions is given. Overall the thesis shows how BNN can be used in active learning, and which properties of the objective might be leveraged to yield an efficient calculation scheme.

Acknowledgments

I want to acknowledge my advisors for supporting me from a technical perspective and that they spend their time on this project. Furthermore I want to thank my family and friends for being patient with me during the thesis.

Contents

1. Introduction

In our current and future society robots represent an integral part of our life. Several subsidiary industries in service, emergency but also entertainment have emerged through the advances Recently robot learning, deep learning and more importantly the intersection of these topics made significant progress. Nevertheless the most crucial areas for our future life stay in active research. One important question arises on how to decide on an data exploration strategy. This is true for sequential decision problem as wells as for really large datasets. Albeit this exploration strategy might work on smaller environments, some setups are even impossible to explore without some prior information about the dynamics. These type of problems are sometimes referred as active learning. In contrast to structure learning approaches, where the goal is to find a network architecture, which learns the given data pretty well. However active learning chooses data points so that the given architecture is learned optimally. The latter is inherently better suited for really big datasets and streaming applications, where it might be not feasible to use all data points. But this affects the model from an operational perspective as it reduces the energy costs needed for finding the correct model configurations.

This work reviews the terms of Bayesian inference and three different Bayesian models. They all have in common that their predictive variance is convex in the inputs. While this would be not problematic when one tries to minimize it, it poses a challenge when the objective has to be maximized. By using a recently introduced quadratic optimizer [2] this maximization problem can be solved. Related to this is the work from [3], but instead of using a linear model different neural network models get compared. One is based on using an ensemble [4] and the other applies the mean-field assumption in variational inference and subsequently reparameterizes the gradient [5]. Afterwards a theoretical overview of the underlying optimization problems is given under the assumption of a convex variance. Therefore a binary formulation is contributed by this thesis. Furthermore these problem are listed in advanced complexity ending with a quadratically constrained quadratic programs (QCQP) [6]. After the comparison these methods get applied to four one-dimensional action space environments. For the Pendulum-v0 a more in-depth

analysis is performed, which also looks at the state distribution. Some combinations perform pretty well and it turns out that it sometimes depends on the environment which method performs good. Despite the results, there needs to be further research to be done, e.g. maybe on could transfer the submodularity results from discrete mutual information problems to these relaxed versions, e.g. they might correspond to symmetries in this continuous space.

Possible applications reach from exploring planar environments, over selecting the most valuable data points from a data set up to the automatic positioning of measurement devices. Consider a plane where for each state and action a next state is given. This state space is two-dimensional and hence the forward model efficiently stores the information which parts of the plane were already explored.. Extracting data points from a dataset are possible applications, where one selects the nearest data points based on an optimal design. This thesis consists of a foundational section, which covers the fundamentals and connects it with related work. Several estimators for design objectives of experimental design are discussed and compared. At the end of the foundational section, the combined exploration algorithms get proposed. Afterwards an evaluation of different estimators is presented and the two most promising objectives, namely predictive variance and MINE, are evaluated in sequential explorative fashion

1.1. Related Work

This type of algorithm is strongly related to the work of [3], with the difference of using a more complex model and a different optimizer. [7] uses Gaussian processes instead of Bayesian neural networks or linear regression. Active learning is closely related to curiosity driven exploration in reinforcement learning. While the former uses a forward model for uncertainty, the latter use reward shaping to make the model curious about it's data. The work [8] is an example of a curiosity driven exploration agent. Usually reinforcement learning explores around a goal trajectory, but also algorithms which incorporate the forward model exist. One example is presented in [9], where the forward model is used inside of the agent. Recently there have been an introduction of a new optimizer [2], which can be generally used for such problems. While these references are considered to be directly related, there are more background references introduced in the text. They form the basis for the algorithm as a whole, but are direct part of this research area.

2. Foundations

This chapter introduces the fundamentals needed for the proposed algorithms later on. This thesis consists of three parts: First of all, the performance of Bayesian neural networks (BNN) and RMS ensemble get evaluated on an extracted dataset from the environment. Afterward, different objectives and estimators used in Bayesian optimal experimental design (BOED) get compared and described. In the third part, an assumption about the form of the design objectives gets leveraged to generate a fast optimizer, related to the iterative linear quadratic regulator (iLQR). Two types of learning get distinguished, namely supervised learning and active learning.

At the beginning of this section, the terms of supervised learning get recapped. Additionally, three practical Bayesian models for obtaining uncertainty estimates get described. Furthermore, the different types of uncertainty are categorized and described from the sequential control perspective this work faces. Afterward, the control perspective gets refined and explained in terms of mathematical optimization. The section closes with a comparison of the predictive variance to the mutual information, and how the mutual information can be estimated.

2.1. Supervised learning of forward model

In supervised learning, the incentive is to try to fit a model to data, usually parameterized by parameters θ. The general setup consists of a dataset and a loss function evaluated over that dataset. It can is expressed in terms of two components:

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \qquad \mathbb{E}\left[\mathcal{L}(\theta)\right]$$

A dataset with tuples of input data x_i and corresponding output y_i as well as the loss function $\mathcal{L}$ over parameters θ. The incentive lies in finding parameters θ, minimizing the expected loss function $\mathcal{L}$ over the complete data set. However, in sequential control, the

output is parameterized as the next state and the input as the current state action tuple. All data samples are grouped into the two data matrices $X \in \mathbb{R}^{Nxd_x}$ and $Y \in \mathbb{R}^{Nxd_y}$. Despite the so-called training set, one usually has an additional validation set to prevent overfitting. Throughout the notation for states and normal data tuples is used exchangeable in the sense that $x_i = s_{i+1}$ and $y_i^\mathsf{T} = (s_i^\mathsf{T}, a_i^\mathsf{T})$. Some chapters feel more natural in one notation and others in the other. The representation as a dataset $\mathbf{X}$ and $\mathbf{Y}$ assumes implicitly that the sequential samples are not dependent on each other. In any way, this assumption is reasonable for Markov decision processes, but might not exactly be true under more exceptional circumstances.

2.1.1. Sequential data as supervised data

A stochastic (Markovian) environment consisting of an (implicit) initial state distribution $\mu(s_1)$ and a (implicit) transition distribution $p(s_{t+1}|s_t, a_t)$. Usually the control policy $\pi(a_t|s_t)$ is independent from the environment and hence needs to be inferred from a specific goal g or an optimization objective $\mathcal{J}$, see [10]. Forward-sampling from the implicit likelihood model yields a trajectory $\xi_t = \{s_1, a_1, \ldots, a_{t-1}, s_t\}$ whereas $\xi = \xi_T$ is a complete trajectory of length T. For each trajectory exists a state $\mathbf{S}$ and action $\mathbf{A}$ matrix with $\mathbf{S}_i = s_i$ and $\mathbf{A}_i = a_i$. The likelihood of a complete trajectory is

$$p(\xi) = \mu(\mathbf{S}_1)p(\mathbf{S}_{2:T}|\mathbf{A}, \mathbf{S}_1)\pi(\mathbf{A}|\mathbf{S}_{1:T-1}) = \mu(s_1)\left[\prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t)\right].$$

The validation set with whom the reference network was trained was generated by setting $\pi(a_t|s_t) = \mathsf{U}(a_{\text{low}}, a_{\text{high}})$ to the uniform distribution, which is independent of the state s_t. By additionally sampling the initial s_1 state uniformely from the complete state space $\mu(s_1) = \mathsf{U}(\mathcal{S})$, a state-action space filling dataset is generated. For a reduction in complexity in the following paragraphs, this is adapted and expressed equivalently as $\mu(s_1) \propto 1$ and $\pi(a_t|s_t) \propto 1$.

2.2. Bayesian inference for supervised data

This section summarizes the Bayesian models used throught. The general framework of Bayesian optimization consists in having a (informative) prior $p(\theta) = p(\theta|\mathcal{D})$ over

parameters as well as a data-likelihood $p(y|x, \theta)$. Both together can be used to calculate the posterior

$$p(\theta|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)}{\int p(\mathbf{Y}|\mathbf{X}, \theta')p(\theta')d\theta'} = \frac{p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)}{p(\mathbf{Y}|\mathbf{X})}$$

by dividing their product through the evidence $p(y|x)$. There are direct training schemes, e.g. in Bayesian linear regression (see section 2.2.2), compared to variational inference scheme, e.g. in Bayesian neural networks (see section 2.2.3). They are explained in more detail in the corresponding section. Often in these settings one is familiar with the posterior predictive distribution

$$\hat{p}(y|x) = \int p(y|x, \theta)p(\theta|x, y)d\theta$$

and is equivalent to the posterior marginalized likelihood. These Bayesian terms are use in two fashions throughout, between the reader should diversify, in the sense that one time it used for learning a Bayesian implicit model for the real-world data. Whereas the so generated model gets subsequently plugged into the optimization for the information gain of this exact distribution. The used estimators for the gradient is introduced in the section of nested inner optimization problems, see section 2.2.2.1, whereas the general framework of optimal experimental design is introduced in section 2.5. For some easier model one can obtain the posterior in closed form but for most neural models there is no closed form distribution for the posterior and one then receives one can't obtain the closed form posterior but instead approximates the posterior $p(\theta|x, y)$ by a variational distribution $q(\theta)$. There are various methods, for example Hamiltonian Monte Carlo, mean field variational inference and laplace approximation to infer the posterior.

2.2.1. Probabilistic inference

This section introduces the two concepts for inferring the optimal parameters used throughout. If the full distribution of the posterior can not be calculated in closed two solutions to that problem are: On one hand the usage of (multiple) point estimates of $\theta_1, \ldots, \theta_J$ are explained like in maximum a-posteriori or MCMC methods and on the other hand additional assumption on the posterior distribution are considered. A prominent example for such an assumption is called the mean-field assumption used throughout variational inference. See section 2.2.1 for more detail on variational inference. Throughout this

chapter the transformation $\theta^{\mathsf{T}} = (\theta_1^{\mathsf{T}}, \ldots, \theta_N^{\mathsf{T}})$ is used to transform the parameters θ in the estimation. As an immediate result the likelihood and prior can be factorized as

$$p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta) = \prod_i p(y_i|x_i, \theta_i)p(\theta_i)$$

over the data. This induces that different parameters are used for each training sample. There are cases where there is no real distribution on θ for the transformation presented above it holds that $\theta_i = \theta_{i'}$.

2.2.1.1. Maximum a-posteriori estimation

The first is called maximum a-posterior estimation, which is deeply related to maximum likelihood. Maximum likelihood maximization just assumes there is a uninformative uniform prior on $\theta \sim 1$. However the details of maximum likelihood are not presented here but can be looked up at [11]. In maximum a -posteriori estimation he main idea is to solve the optimization problem

$$\theta^* = \arg\max_\theta p(\theta|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \theta)p(\theta)$$

built out of the product of likelihood and prior probability. Usually optimization is made in the log-space, e.g. replace $p(\theta|\mathbf{X}, \mathbf{Y})$ by $\log p(\theta|\mathbf{X}, \mathbf{Y})$. Especially for distributions from the exponential family like the Gaussian, Bernoulli or Beta this makes sense as it cancels the exponential function in the probability density function (pdf). It is used in the estimation of th optimal parameters for the RMS-ensemble see section .

2.2.1.2. Approximate variational inference

In variational inference the goal is to fit a parametric distribution q to an arbitrary real distribution p. There are multiple possibilites for the choice p [5, 12, 13]. Usually this is achieved by minimizing the KL-divergence between both q and p, but other distance measures can be used like the Wasserstein distance in GAN's [14] or the f-divergence. At this point posterior inference is described but the other application of variational inference to bounding the mutual information is given in section 2.5.2.3

Most commonly and widely used is approximate posterior inference and consists of fitting a variational distribution $q(\theta)$ to the true posterior $p(\theta|\mathcal{D})$. As presented in [13] the objective can be reformulated

$$\begin{aligned}\mathbb{KL}(q(\theta)|p(\theta|\mathbf{Y},\mathbf{X})) &= \mathbb{E}\left[\log q(\theta) - \log p(\theta|\mathbf{X},\mathbf{Y})\right] \\ &= \log p(\mathbf{Y}|\mathbf{X}) - \mathbb{E}\left[\log p(\theta,\mathbf{Y}|\mathbf{X}) - \log q(\theta)\right] \\ &= \log p(\mathbf{Y}|\mathbf{X}) - \underbrace{\mathbb{E}\left[\log p(\mathbf{Y}|\mathbf{X},\theta)\right] - \mathbb{KL}(q(\theta)|p(\theta))}_{\text{ELBO}_{\mathbf{YX}}(q(\theta))}\end{aligned}$$

in terms of a constant part called the evidence $\log p(\mathbf{Y}|\mathbf{X})$, which only depends on the observed data. The other term is called the evidence lower bound (ELBO), which can be written in terms of the likelihood and the KL between prior and variational posterior. Combining the introduced concepts with the constant nature of the evidence $p(\mathcal{D})$ the original problem of minimizing the KL-divergence 2.1 can be rewritten as

$$\arg\min_q \ \mathbb{KL}(q(\theta)|p(\theta|\mathbf{Y},\mathbf{X})) = \arg\max_q \ \text{ELBO}_{\mathbf{YX}}(\theta) \tag{2.1}$$

maximization of the evidence lower bound [13]. This framework enables the usage of an approximating family (sometime also called hypothesis space) $\mathcal{H}$ with $p(\theta|\mathcal{D}) \notin \mathcal{H}$ not being contained in it. This case is interesting, because for the opposite case $p(\theta|\mathcal{D}) \in \mathcal{H}$ it holds that $\min_q \mathbb{KL}(q(\theta)|p(\theta|\mathcal{D})) = 0$, which reduces the optimization problem from equation 2.1 to maximum likelihood, e.g. $\max \text{ELBO(q)} = \max \mathbb{E}\left[\log p(\mathcal{D}|\theta)\right]$. It is noteworthy at this point that all expectations in this paragraph are from $\theta \sim q(\cdot)$ the variational distribution. A common example of an assumption is the mean-field assumption, which basically treats each parameters with an independet distribution, e.g. $q(\theta) = \prod_i q(\theta_i)$.

2.2.2. Bayesian linear regression

Bayesian linear regression uses as a hypothesis space the classes of linear predictors, e.g. the likelihood is written as $p(y|x,\theta) = \mathcal{N}(y|\theta^\mathsf{T}x, \sigma^2\mathsf{I})$ and the row-independent matrix normal prior on $p(\theta) = \mathcal{N}(\theta|\mu, \mathbf{\Sigma})$. By using the stacked matrices $\mathbf{X}$ and $\mathbf{y}$ the posterior $p(\theta|\mathbf{X},\mathbf{y}) = \mathcal{N}(\theta|\mu^*, \mathbf{\Sigma}^*)$ is given by

$$\mu^* = \mu^*_{\mathbf{X}} = \mathbf{\Sigma}^*(\frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{y} + \mathbf{\Sigma}^{-1}\mu) \qquad \mathbf{\Sigma}^* = \mathbf{\Sigma}_{\mathbf{XX}} =^* (\frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{X} + \mathbf{\Sigma}^{-1})^{-1}$$

is written in dependence of the regularized pseudo-inverse of the data input matrix $\mathbf{X}$. The formulas are easily derivable by linear algebra and have an interesting application

in active learning [3]. An online version of Bayesian linear regression is retrieved by sequentially replacing the prior with the approximated conjugate posterior. This is better known as Bayesian online updating and produces a series of distributions converging to the true posterior.

2.2.2.1. Nested optimization problems

Multiple optimization problems are solved and the solution of one serves as an input of the other. Overall two problems of this type arise throughout the thesis. The purpose of this section is to explain the different types used, how they interact and which approximations are made to make the problem feasible. First of all the Bayesian neural network is trained on standard state action transition dynamics via variational inference. This approximated network is subsequently plugged into the estimator of the information gain or the predictive variance. The other problem arises when in the exeperimental design the updated prior is considered. This paragraph focus on the latter and describes implied properties. By definition

$$\mathbf{X}^* = \arg\min_x \quad \mathcal{L}_V(\mathbf{X}, q^*(\mathbf{X})) \qquad\qquad q^*(\mathbf{X}) = \arg\min_{q(\mathbf{X})} \quad \mathcal{L}_T(q(\mathbf{X}))$$

this is a nested optimization problem as explained in more depth in [15]. These types of problems occur in several areas like multi-objective optimization, active learning, game-theoretic and self-play. For convenience the case of Bayesian linear regression is investigated more deeply and thus we take the negative log-likelihood as training loss e.g. $\mathcal{L}_T(\mu^*, \Sigma^*) = -\log p(\mathbf{y}|\mathbf{X}, \theta) - p(\theta)$ and solve the right subproblem

$$\begin{aligned}\boldsymbol{\Sigma}^*(\mathbf{X}) &= \frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{X} + \boldsymbol{\Sigma}^{-1} \\ \mu^* &= \left(\boldsymbol{\Sigma}^*(\mathbf{X})\right)^{-1}\left(\frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{X}\mu + \boldsymbol{\Sigma}^{-1}\mu\right) = \left(\boldsymbol{\Sigma}^*(\mathbf{X})\right)^{-1}\boldsymbol{\Sigma}^*(\mathbf{X})\mu = \mu\end{aligned}$$

in terms of $\mathbf{X}$. At this point $\boldsymbol{\Sigma}^*$ is differentiable worth $\mathbf{X}$. However due to the propagation of the mean $\mathbf{y} = \mathbf{X}\mu$ the learned mean doesn't change $\mu^* = \mu$, which gets independent of $\mathbf{X}$ by that. Interestingly new input data points with mean propagated outputs only influence the variance. However it is possible to inject noise into the mean using the reparameterization trick using setting $\mathbf{y} = \mathbf{X}(\mu + \sqrt{\boldsymbol{\Sigma}}\varepsilon)$. Returning to the discussion no noise is injected in the mean and by exploiting the independence of μ from $\mathbf{X}$ one can differentiate through both optimization problems and solutions. It is noted that when the

negative predictive variance

$$\begin{aligned}\mathcal{L}_V(\mathbf{X}) &= -\sum_i (x_i - \mu^*)^\mathsf{T} \left(\mathbf{\Sigma}^*(\mathbf{X})\right)^{-1} (x_i - \mu^*) \\ &\propto -\sum_i (x_i - 2\mu)^\mathsf{T} \left(\mathbf{\Sigma}^*(\mathbf{X})\right)^{-1} x_i\end{aligned}$$

is used as the validation objective $\mathcal{L}_V$ it can be differentiated by $\mathbf{X}$. By using gradient descent on the validation loss one can obtain the corrected hypergradient. In the experiments section we make some experiments based on this and the mutual information. It is noteworthy at this point that this discussion is only included for showing the subtle differences between using an updated posterior or the current prior. Although the right subproblem has a closed form solution under Bayesian linear regression, more complex models have to use estimators like presented in [15] for the hypergradient $\nabla\mathcal{L}_V(\mathbf{X})$. The most basic estimator iteratively solves both problems without differentiating through them. However the most complex estimator needs an approximation of the inverse Hessian. The prior is usually set by the previous data matrix with the regular update formulas.

2.2.3. Bayesian neural networks

A Bayesian neural network is an extension to standard neural networks. Typical default neural networks consists of an activation function $\phi(x)$ combined with a multivariate extension of the linear regression model seen in section 2.2.2. Additionally a prior is placed on the weights in a Bayesian neural network to constrain the weight space and allow for (approximated) posterior inference, see section 2.2.1 for more details about variational inference itself. This work focuses on fully connected feed forward neural networks with L different layers. It is straightforward to describe it more formally

$$a_0 = x \qquad v_i = W_i^\mathsf{T} a_{i-1} + b_i \qquad a_i = \phi(v_i) \quad i \in \{1, \dots, L\}$$

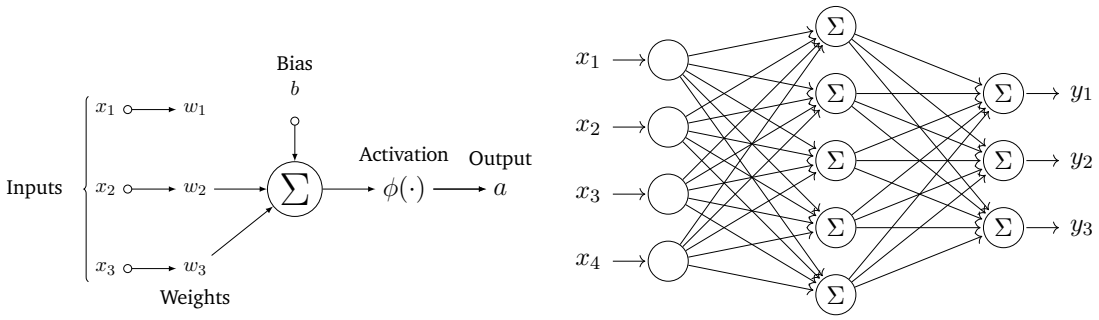
with a recursive relationship. This whole construct is rather unhandy to write out in full detail and thus another layer of abstraction gets introduced. Moreover an abstract function can be designed by abbreviating $\theta^\mathsf{T} = \{\text{vec}(W_1)^\mathsf{T}, \dots, \text{vec}(W_L)^\mathsf{T}\}$ and to write $f_\theta(x) = a_L$, e.g. set the output of the function approximator equal to the output of the last layer. More intuitively the network can be desribed in terms of neurons see also figure 2.1 if you favor this type of description.

2.2.3.1. Bayesian version of standard neural networks

With that in mind the likelihood can be states as $p(y|f_\theta(x), \sigma^2)$ and the prior as $p(\theta)$. In Bayesian inference the goal is obtaining either samples, a closed form solution or an approximated variational posterior of the true posterior $p(\theta|x, y)$. At this point is necessary to introduce shortly kernel density estimation and how it is used to represent the data likelihood of Bayesian neural network (BNN) and the randomized map sampling (RMS) ensemble. Kernel density estimation is used to obtain the approximated marginal likelihod

$$p(y|x) = \mathop{\mathbb{E}}_{\theta \sim p(\cdot)} \left[p(y|f_\theta(x), \sigma^2) \right] \approx \frac{1}{J}, \sum_{j=1}^{J} p(y|f_{\theta_j}(x), \sigma^2) = p(y|x, \theta_1, \ldots, \theta_J) \tag{2.2}$$

which is finite by nature. In terms of section 2.2.1 it is used as $p(\mathcal{D}|\theta)$. While this might seem wrong at first glance, it can be interpreted by setting $\theta = (\theta_1, \ldots, \theta_J)$ the prior doesn't change in number of parameters and the likelihood can again be written in the form $p(y|x, \theta)$. Motivated by this discussion it is totally fine to replace $p(y|x, \theta)$ with an approximated version of $p(y|x)$. Is it noteworthy that in the limit $J \to \infty$ this assumption does not hold and this would correspond to an infinite Bayesian ensemble. By exactly examining the likelihood term at the right of equation 2.2 one notices that either a Gaussian prior can be used to generate θ_j see section 2.2.1 or the method of the RMS ensemble see section 2.2.5.



(a) Sources of uncertainty (b) Architecture of a neural network

Figure 2.1.: Neural network represented as nodes
On the left side a neuron of a neural network is shown. In a Bayesian setting additionally a prior is given over the weights w and bias b. On the right side a complete neural network architecture is shown. It consists of multiple neurons.

2.2.4. Approximation methods for the posterior

The next section covers the relevant topics for the former method based on the mean-field assumption. Besides the approaches mentioned and discussed here, there are also other inference schemes like Hamiltonian Monte Carlo, Posterior sampling or Laplace approximation to the posterior. This work focuses on gradient-based methods and the general objective for training looks like

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \log p(y_i|f_\theta(x_i), \sigma^2) + \Omega(\theta)$$

with the log likelihood subtracted by some regularization term $\Omega(\theta)$. Two variants of this loss are used to optimize either the BNN or the RMS-ensemble. The former method approximates the posterior with an isotropic Gaussian, which is better known as the mean-field assumption in variational inference see section 2.2.1 for more details. This method uses the negative KL-divergence between the prior and the posterior as a regularization term $\Omega(\theta)$. Nevertheless there is an inherent approximation error to the true posterior resulting from a misconfiguration of the hypothesis space by the mean-field assumption. The latter method uses multiple independent networks and a different loss for each network. It simply uses the log probability of the prior as a regularization term

2.2.4.1. Variational mean-field Bayesian neural networks

This method is motivated by using the techniques from section 2.2.1 together with path wise gradients. The uncertainty is injected in parameter space by sampling a value from an approximation of the posterior distribution and gets propagated to the output. By using the reparameterization trick the gradient can be propagated back into the approximation of the posterior and the parameters can be learned using standard gradient descent or more advanced optimizers on the ELBO objective. The general form is given by

2.2.4.1.1. Reparameterization tricks Before being able to use this model in the optimization objective introduced in equation 2.1 two problems arise arise. The first one is rather simple and is the fact that the gradient has to be propagated through the distribution into the hyperparameters. It turns out that by applying th reparameterization trick these hyperparameters can be trained directly via stochastic subgradient descent or more sophisticated methods like Nesterov momentum, RMSProp or ADAM. It can be be applied

by replacing a variable x from a Gaussian distribution $x \sim \mathcal{N}(\cdot|\mu, \sigma)$ with the pathwise reparameterization $x(\mu, \sigma) = \mu + \sigma\varepsilon$. By this transformation the variable and thus the gradient is just depending on noise $\varepsilon \sim \mathcal{N}(\cdot|0, \mathrm{I})$ from a standard Gaussian. This trick is needed in either case, whether the noise is injected in *weight space* or *activation space*.

In weight space the noise gets injected before the samples are multiplied by the weight matrices W_i. With this method the sampled weights have very high dimensions, especially for a neural network. All activations of a neural network have a lower dimension than all weights and thus results in higher variance, when sampling from weight space instead of activation space [16]. The activation space transformation is given by

$$v_i \sim \mathcal{N}\left(\cdot|M_i^\mathsf{T} x, \sqrt{V_i^\mathsf{T}(a_{i-1} \odot a_{i-1})}\right) \qquad W_i \sim \mathcal{N}(\cdot|M_i, V_i)$$

a distribution of much smaller dimension than the right one. From here the default reparameterization trick can be used. Up to certain special cases where access to the weight space noise is needed, the activation space sampling shows better performance [5]. The later introduced algorithm switches between both modes when training the network and when estimating the mutual information.

2.2.4.1.2. Expressiveness As the weights from the variational posterior are assumed to be independent this results in a convex variance [17, 18]. The authors propose to use a different inference schemes like HMC or to change the model to a Gaussian process or an ensemble. However this work focuses on gradient based optimization techniques and a solution approach based on sequential quadratic programming is presented later on.

2.2.5. Randomized map sample ensemble

As an alternative model randomized map sampling is used in an ensemble [4] is used. While their work was motivated by previous work on linear regression models [19] see also section 2.2.2. In [4] they not the difference between the data likelihood and the parameter likelihood. Recalling the terms from section 2.2 is is noteworthy that this was the data likelihood $p(\mathcal{D}|\theta)$ a s opposed to the parameter likelihood $p_\theta(\mathcal{D}|\theta)$. While it is possible for some models to derive them, for the case of neural networks this can not be derived in closed form [4]. Instead the authors propose to use several networks in control

terms $f_i(x)$ a modified loss, e.g. they propose to use

$$-\mathcal{L}_j = \frac{1}{N}\sum_{i=1}^{N} \|y_i - f_\theta(x_i)\|^2 + \|\sqrt{\Sigma_{\text{anc}}}(\theta_j - \theta_{j,\text{anc}})\|^2$$

a slightly varied the loss for each member j of the ensemble. Each neural network in the ensemble represents one sample from the posterior distribution. Note that the left hand side of the equation shows a negative loss $-\mathcal{L}$, because the mean squared error is used instead of log likelihood. Both objective have to be optimized Compared to section 2.2.1 a pattern underlying both method exists. This work uses the following loss

$$\mathcal{L}_j = \frac{1}{N}\sum_{i=1}^{N} \log p(y_i|f_\theta(x_i), \sigma^2) + \Omega(\theta) \qquad \Omega(\theta) = \log \mathcal{N}(\theta|\theta_{j,\text{anc}}, \Sigma_{\text{anc}})$$

in order to make a coherent framework with Bayesian neural networks. The parameter J is used to define the number of ensemble members in the RMS-ensemble, whileas in BNN's it represents the number of samples used to propagate through the network for the kernel density estimation.

2.2.6. Skip connections in neural networks

Recall that when a sequential control setting is used one has $y_t = s_{t+1}$ and $x_t^\mathsf{T} = (s^\mathsf{T}, a^\mathsf{T})$. In robotics and deep learning it is convenient to use skip connections. The technique is practically realized by plugging the abbreviation for the residual $\Delta s_t = s_{t+1} - s_t$ as the output of the network $y_t = \Delta s_t$. After this transformation a parameterized distribution $p(\Delta s_t|s_t, a_t)$ gets obtained and can be used to obtain the next state using the simple relation $s_{t+1} = s_t + \Delta s_t$. This work uses skip connections only for the ensemble and not for the BNN. A clear advantage is that whenever the action space is box-constrained the residual is as well constrained. Noteworthy is also the fact that the weights of the neural network model can be chosen much smaller, e.g. the prior has to be set to smaller values to work.

2.3. Sources of uncertainty

This section covers the main types of uncertainties present in learning a model based upon data. There are different measures of uncertainty whereas usually the entropy $\mathbb{H}$ or the variance $\mathbb{V}$ is used [20, 21]. One of the most basic types is the model uncertainty itself. It is introduced by choosing a hypothesis space $\mathcal{H}$ which can not model the target distribution with high confidence. More related to a model from the hypothesis space are *aleotoric* and *epistemic* uncertainty [20]. While the first one is inherent to the data generating process, e.g. in cases where the data generating process is not deterministic. Both of them have their roots in quantifying uncertainty and distinguishing different causes for the noise. A decomposition for both measure has been proposed in [21] and is recapped in section 2.3.2. Another form of uncertainty is given by the factorization. Each point of the trajectory is viewed independently and thus a factorization over the time is performed. This introduces another error, e.g. areas where the model fails to be represent the correct uncertainty.

2.3.1. Model Uncertainty

Most basically there is uncertainty when the model can't approximate the real hypothesis arbitrarily well. While this can have many reasons for neural networks it is most likely the architecture producing this problem. The model uncertainty or otherwise called expected approximation error of the hypothesis space is the minimum distance

$$\Delta y = \min_{f} \mathbb{E}\left[|y - f(x)|\right]$$

between the learned and the real solution. For neural networks this approximation error is rather small. It gets introduced by a finite sized hidden layer in the neural network. If the number of neurons in a neural network with one hidden layer go to infinity it can be shown that the hypothesis space is dense in the space of continuous functions. By this analogy the approximation error is equivalent to the size of the smallest closed set which contains besides the data and at least one hypothesis.

2.3.2. Aleotoric and Epistemic uncertainty

Aleotoric uncertainty is the uncertainty present in the data or the underlying stochastic process. There are data processes or distributions which have inherent noise, e.g. for one

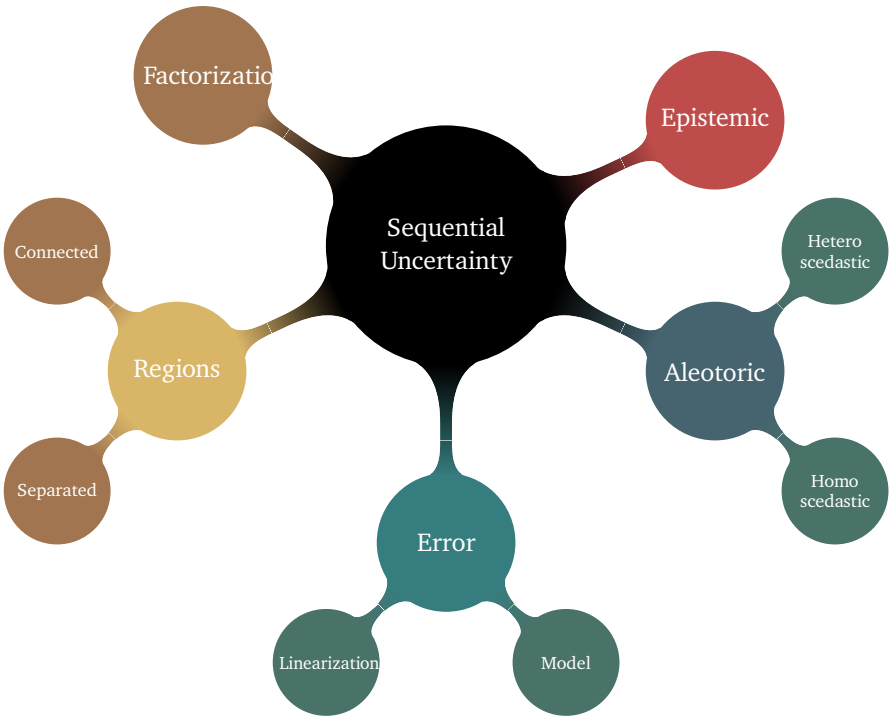


Figure 2.2.: Sources of sequential uncertainty
A categorization of uncertainty. Note that approximation errors induced by the model are also considered as uncertainty. It should be clear that in the case of an approximation error of the model, the model is unconsciously uncertain. This work is about eliminating conscious uncertainty. Aleotoric and epistemic uncertainty fall into this category, but only the latter can be reduced by gathering more/better data. Specific to sequential settings are the region based uncertainty, these include the finite horizon uncertainty and the one introduced by separated state regions.

input x two different outputs $y_1 \neq y_2$ can be obtained. This is uncertainty where the model can't get more certain by increasing the amount of data available. Two decompositions are relevant for this work. They deliver also the motivation for the objectives used later on in section 2.5. For more details on these types of variances a good reference is presented in [21]. Aleotoric uncertainty gets further refined as having heteroscedastic or homoscedastic variance. The first corresponds to a different uncertainty for each sample, whereas the latter corresponds to homoscedastic variance. In all experiments a homoscedastic variance is assumed and thus the variance of the likelihood stays fix.

2.3.2.1. Different measures of uncertainty

The first occurs when using the entropy $\mathbb{H}$ as a measure of uncertainty [21]. Later on the resulting epistemic uncertainty from this decomposition is used for learning. The authors decompose the total uncertainty into an aleotoric part, which is defined in terms of the entropy $\mathbb{H}$. The counterpart of the decomposition is the epistemic uncertainty. Epistemic uncertainty can be reduced by increasing the amount of data. It naturally corresponds to uncertainty in the model itself, e.g. at initialization time the model is rather uncertain. The full decomposition of total variance into aleotoric and epistemic uncertainty is given by

$$\underbrace{\mathrm{MI}(y, \theta)}_{\text{epistemic}} = \underbrace{\mathbb{H}\left[p(y|x)\right]}_{\text{total}} - \underbrace{\mathop{\mathbb{E}}_{\theta \sim q(\cdot)}\left[\mathbb{H}\left[p(y|x, \theta)\right]\right]}_{\text{aleotoric}}$$

the mutual information between output y and parameters θ [21]. The decomposition shows which parts are optimized for optaining optimal experimental designs. This definition clearly motivates the usage of the later on. Using the same scheme the aleotoric uncertainty can be set to the expected variance over θ as compared to the direct variance over θ. The total variance is given by $\mathbb{V}\left[p(y|x)\right]$. Schematically equivalent to the decomposition for the mutual information the epistemic uncertainty under variance

$$\underbrace{\mathbb{V}\left[p(y|x, \theta)\right]}_{\text{epistemic}} = \underbrace{\mathbb{V}\left[p(y|x)\right]}_{\text{total}} - \underbrace{\mathop{\mathbb{E}}_{\theta \sim q(\theta)}\left[\mathbb{V}\left[p(y|x, \theta)\right]\right]}_{\text{aleotoric}}$$

can be written as the difference between total and aleotoric uncertainty. It is noteworthy that one recovers the predictive variance as the epistemic part, see section 2.5.1. This section shortly summarized the motivation for the later used objectives mutual information and predictive variance. These types of uncertainty talk about independent data point. In most real world settings this is usually not the case as they are sequentially dependent. The next section describes these sequential uncertainties.

2.3.3. Sequential uncertainty

Sequential uncertainty occurs when a model is used recursively and can be described by two effects. Consider a forward model $p(s_{t+1}|s_t, a_t)$ which is used to predict multiple time steps into the future, e.g. the output state s_{t+1} serves as an input to the model. The

input uncertainty is directly propagated to the output uncertainty. By this phenomena the model gets increasingly more uncertain when extrapolated into the future. Besides this naive addition of sequential uncertainty two other factors are considered.

2.3.3.1. Finite horizon uncertainty

First there is the fixed horizon T over which the model looks into the future. As finitely many parts get aggregated over time it can not reach all sections of the state space by nature. So it can only be certain around some bounded neighborhood of the current position.

2.3.3.2. Connected and separated state-space regions

This type of uncertainty occurs when a subspace of the state space is separated into two parts. A state space $\mathcal{S}$ is separated if there exist two points s_1, s_1' such that the infinite application of the set operator

$$\mathcal{T}^k(s) = \mathcal{T}^{k-1}(s) \cup \{f_\theta(s', a) \colon s' \in \mathcal{T}^{k-1}(s) \wedge a \in \mathcal{A}(s')\} \qquad \mathcal{T}^0(s) = \{s\}$$

yields two different sets with $\mathcal{T}^\infty(s) \cap \mathcal{T}^\infty(s') = \varnothing$. The action space $\mathcal{A}(s)$ depends on the current state in general but in the used environments later the action space is constant for all states, e.g. $\mathcal{A}(s) = \mathcal{A}$. Whenever the state space $\mathcal{S}$ is bounded this means induces that $\mathcal{T}^\infty(s)$ is closed. However when the state space is not bounded this could possible yield a non closed set. That's why only bounded state spaces are allowed in this thesis.

2.4. Optimization

This section the three different schemes used to solve the underlying optimization problems. Strong evidence is given on the used optimization schemes, which can be divided into a discrete, a disciplined convex-concave and a log-barrier formulation. The most general form of optimization problems from which all of them descent in some scene is

$$\begin{aligned} \min_x \quad & g(x) \\ \text{s.t.} \quad & f(x) = 0 \\ & x_{\min} \leq x \leq x_{\max} \end{aligned}$$

given with an additional state constraint. While the state constraint $x_{\min} \leq x \leq x_{\max}$ is not necessary it is often imposed by the environment. At this point we make the assumption that $g(x)$ is concave which is correct for the negative predictive variance. For the mutual information we empirically verified that at least in the case of Bayesian linear regression the optimal values lie at the boundary.

2.4.1. Sequential quadratic programming

This section outlines how to transform a problem of the first type into one solvable by the disciplied convex and convex-concave program solvers. For the discrete as well as the point method there is no need to make such an approximation. By applying a second-order Taylor approximation to the Lagrangian of the problem

$$\hat{g}(\Delta x) = \frac{1}{2}\Delta x^{\mathsf{T}} \underbrace{\left[\nabla^2 g(x^*) + \sum_i \eta_i \nabla^2 f_i(x^*)\right]}_{Q} \Delta x + \underbrace{\nabla g(x^*)^{\mathsf{T}}}_{c} \Delta x + \underbrace{g(x^*)}_{d}$$

a quadratic approximation to the real function is made. Note that the quadratic matrix not only depends on the Hessian of $g(x)$ but also on the Hessian of the constraints $f(x)$. This is due to the KKT-optimality conditions met in an optimum, and thus. The same technique is applied using a first-order approximation to the constraints $f(x)$ around x^* results in

$$\hat{f}(\Delta x) = \underbrace{J_f(x^*)}_{F} \Delta x + \underbrace{f(x^*)}_{h}$$

with $\Delta x = x - x^*$. At this point it is noteworthy that the variable getting optimized transitions from x to Δx. By repeatingly solving for Δx, updating the parameters $\hat{x} = x + \Delta x$, reapproximating and solving by the above introduced concept, one results in a powerful optimization scheme called sequential quadratic programming. It is used in section 2.4.2 and 2.4.4. It is noteworthy that if there is no constraint on the input space, e.g. $\|x\|_\infty \leq x_{\max}$ the objective can be replaced by the symmetric matrix $\frac{1}{2}(Q + Q^{\mathsf{T}})$ and the resulting optimization problem has the same optimum values. The resulting objective and constraints are

$$\hat{g}(\Delta x) = \frac{1}{2}\Delta x^{\mathsf{T}} Q \Delta x + c^{\mathsf{T}} \Delta x + d \qquad \hat{f}(\Delta x) = F\Delta x + h$$

represented by their parts. Sometimes this transformation is required as this is the case for the disciplined concave convex program solver. The interior point method and discrete

solver use generally no quadratic approximation, so this is only the case for DCP and DCCP programming.

2.4.1.1. Gauss-Newton approximation to the Hessian

Typically a Gauss-Newton approximation to the true Hessian is used. This ensures positive definiteness of the estimated Hessian. Additionally it is far easier to calculate as it depends only on first order gradient information e.g. as $\mathbb{E}\left[\nabla g(x) \nabla g(x)^{\mathsf{T}}\right]$. In the context of a convex function it totally makes sense to approximate it using the Gauss-Newton matrix. For more information on this topic see [].

2.4.2. Convex Programming

The quadratic $g(x)$ is convex, when the matrix Q is positive definite. There are solvers out there which can easily solve this kind of optimization problem. They might can be solved by sequential quadratic programming and can be written

$$\begin{aligned} \min_{x} \quad & g(x) \\ \text{s.t.} \quad & f(x) = 0 \\ & x_{\text{min}} \leq x \leq x_{\text{max}} \end{aligned}$$

as the standard optimization problem. These type of problems can be solved with linear constraints or also non-linear constraints. In the latter case, the constraints get linearized after each solution update step. In control there is the iterative linear quadratic regulator [22], which uses a closed form solution instead of gradient updates.

2.4.3. Discrete Programming

Whenever $g(x)$ is concave the solutions for the optimal actions lie at the input space boundaries. In general this is the case for a concave function if $[x_{\text{min}}, x_{\text{max}}] \subseteq \{x \colon f(x) \leq 0\}$. As we have 1-dimensional actions this clearly motivates the approach of solving the following problem

$$\begin{aligned} \min_{x} \quad & g(x) \\ \text{s.t.} \quad & f(x) = 0 \\ & x_{\text{min},i} = x_i \vee x_i = x_{\text{max},i} \end{aligned}$$

in a discrete fashion. However by using this approach two problems exist. Albeit the solutions are indeed discrete and lie at the boundary, in practice solutions from the interior are also required. Especially if the solutions serve as the data points for the supervised learning of the forward model. We tried using a filter at this point to smooth out the actions, but this showed bad results in general. The other problem is that trying out all solution is inherently slow and in higher dimensions sophisticated approximations and procedures are needed, e.g. branch-and-bound. Due to these facts this approach is not a good choice at least for continuous control problems, however it is concluded as a comparison in the latter experiments.

2.4.4. Convex Concave Programming

At this point we make the assumption that $g(x)$ is concave which is correct for the negative predictive variance. For the mutual information we empirically verified that at least in the case of Bayesian linear regression the optimal values lie at the boundary. Again a quadratic objective $g(x)$ is used as well as a additional convex term $r(x)$. While in general arbitrary convex $r(x)$ can be used, in control often a quadratic regularization function is used. The essence of the so called epigraph transformation lies in replacing the concave function $g(x)$

$$\begin{aligned} \min_{x} \quad & t + \lambda r(x) \\ \text{s.t.} \quad & g(x) \leq t \\ & f(x) = 0 \\ & x_{\text{min}} \leq x \leq x_{\text{max}} \end{aligned} \tag{2.3}$$

with a variable t [2]. By this transformation the objective gets optimized over both x and t and is convex in both variables. However there is an additional non-linear constraint introduced. As it can be inverted, e.g. $-g(x) \geq -t$ it can be either viewed as a convex or a concave constraint. Nevertheless the disciplined convex concave programming approach approximates the concave constraint by a linear constraint in multiple iterations and solves

it by the same techniques presented in section 2.4.2. This results in an effective scheme for solving these types of optimization problems. Besides choosing the actions with the most information gain there are various other important applications.

2.4.4.1. Relaxing the original problem by regularization

By adding regularization to the original objective the problem can be relaxed such that solutions from the complete interval might occur as a result. The final solution depends heavily on the regularization parameter λ. The resulting problem can be states as

$$\begin{aligned} \min_{x} \quad & g(x) + \lambda r(x) \\ \text{s.t.} \quad & f(x) = 0 \\ & x_{\text{min}} \leq x \leq x_{\text{max}} \end{aligned}$$

the following problem. From this approximation two further solution methods can be derived. Consider a quadratic of the form $g(x) = x^{\mathsf{T}}Qx + c^{\mathsf{T}}x + d$, which is the most basic formulation. One has to consider different versions of a quadratic program. Most basically a quadratic program with linear constraints can be written as

$$\begin{aligned} \min_{x} \quad & g(x) = x^{\mathsf{T}}Qx + c^{\mathsf{T}}x + d \\ \text{s.t.} \quad & Fx \leq f \\ & \|x\|_{\infty} \leq x_{\text{max}} \end{aligned}$$

Three main cases can occur for Q namely that Q is *positive definite*, *negative definite* or in the worst case *indefinite*.There are the semi-definite case, but we will not consider them, as they can be regularized by using shrinkage estimators. The next section covers the easiest problems of them when there is a convex program which follows the disciplined convex programming rules [23].

2.4.5. Quadratically constrained quadratic optimization problem

More difficult to solve are quadratically constrained quadratic optimization problems. They are based on the epigraph transformed problem of the convex-concave section but

with general quadratics $r(x)$ and $g(x)$, e.g. Q beeing only symmetric. The resulting form is

$$\begin{aligned} \min_{x} \quad & r(x) \\ \text{s.t.} \quad & g(x) = 0 \\ & f(x) = 0 \\ & \|x\|_\infty \leq x_{\text{max}} \end{aligned}$$

given by the standard form. In general these type of problems are NP-complete and inherently difficult to solve [6]. However there are solvers using heuristics like the semidefinite-relaxation and suggest-and-improve framework [6]. Albeit the approximated problem is of this form, we reduce it further using the Gauss-Newton approximation of the Hessian to get a concave representation of $g(x)$. This approximation transforms this type of problem into a convex concave program.

2.5. Bayesian optimal experimental design

Bayesian optimal experimental design (BOED) is about designing an experiment, from which data is acquired to further improve the underlying model. There is a design $d = x$ an objective function Ψ. The goal is to find a design which maximizes, or by flipping the objective function minimizes the objective function. Several slightly different design criterion's have been proposed. They can be namely divided into A/E/I/V/G-optimality. However this work views takes a look at V-optimality and the expected information gain as an objective function in this framework. Both have inherent problems, when applied to reinforcement learning. The used notation corresponds in the control setting to $d = (\mathbf{S}_1, \mathbf{A})$ and $y = \mathbf{S}_{2:T}$. It is noteworthy that the sequential fashion of the design d can be further exploited and thus the computational burden can be drastically reduced. Usually the first initial state is fixed as well, e.g. $\mathbf{S}_1$, the design is further reduced to $d = \mathbf{A}$ and the initial state is treated as a random variable. However to simplify the notation it is described for $d = x$ and y or in the case of sequential fashion $d = \mathbf{A}_t$ and $y = \mathbf{S}_{t+1}$, whereas $\mathbf{S}_t$ is either given by the environment or a propagated constraint of the sequential optimization problem.

2.5.1. Predictive Variance

The first main objective is better known in the optimal experimental design literature as V-optimality and the objective to maximize can be written as

$$\Psi(d) = \mathop{\mathbb{V}}_{\theta \sim p(\cdot|\mathcal{D})} [f_\theta(d)]$$

and involves sampling from the predictive model and the prior, while using these particles to estimate the variance of the model. Some problems with this objective occur, at least in the setting of Bayesian linear regression and their neural network version, see section 2.2. To solve a convex maximization problem the solver from

2.5.2. Mutual information

The mutual information is usually viewed between between outputs y and parameters θ. It is non-negative and symmetric but doesn't fulfill the triangle inequality. Hence it can not be considered as a proper metric. A different objective called the variation of information additionally fulfills the triangle inequality and is thus a metric. The mutual information can be expressed in different ways however it's raw extended formula is given by

$$\Psi(d) = \mathsf{MI}(y, \theta) = \mathop{\mathbb{E}}_{y,\theta \sim p(\cdot|d)} \left[\log \frac{p(y|d, \theta)}{p(y|d)} \right] = \mathop{\mathbb{E}}_{y,\theta \sim p(\cdot|d)} \left[\log \frac{p(\theta|y, d)}{p(\theta)} \right] = \mathsf{EIG}(d)$$

with its direct relation to the expected information gain. It is noteworthy at this point that the MI is given in terms outcome y and parameters θ, whereas the expected information gain is given in terms of the design d. See section 2.5 for more details on the connections of design, parameters and outcome. Besides that a notion of normalized mutual information is used in some problems like clustering. There is a deep connection to the entropy $\mathsf{MI}(y, \theta) = H(y) - H(y|\theta)$ and as stated earlier it fulfills $\mathsf{MI}(y, \theta) \geq 0$ as well as $\mathsf{MI}(y, \theta) = \mathsf{MI}(\theta, y)$. A straight forward approach is using a nested Monte Carlo estimator

$$\Psi(d) = \mathsf{MI}(y, \theta) \approx \frac{1}{N} \sum_{n=1}^{N} \frac{\log p(y_n|\theta_{n,0}, d)}{\frac{1}{M} \sum_{m=1}^{M} \log p(y_n|\theta_{n,m}, d)} \tag{2.4}$$

for explicitly evaluating the mutual information with $M = \sqrt{N}$. However the problem lies in the sample size which is exponential in the dimension of the parameters θ [24].

Besides the exact estimator from equation 2.4 there are different approximations to the mutual information [25]. The first estimator inspected is called mutual neural information estimation (MINE). Afterwards the modified estimator called information noise contrastive estimation (INCE) from [26] is presented. Finally various estimators from [12] are described. This work focuses on lower bounds to the mutual information, but also other bounds can be considered. All methods get described and compared to each other and the predictive variance. They share the same underlying theoretical aspects and are based on the Barber-Agakov lower bound.

2.5.2.1. Barber & Agakov variational bound

In the Barber-Agakov lower bound the posterior $p(\theta|y,d)$ is approximated by a variational distribution $q(\theta|y,d)$. It is noteworthy that there is also an Barber-Agakov upper bound on the mutual information, but it is not so relevant for this work. Coming back to the lower bound, it can be derived by adding the variational distribution and separating the expected information gain

$$\begin{aligned}\Psi(d) = \mathsf{MI}(y,\theta) &= \mathop{\mathbb{E}}_{y,\theta\sim p(\cdot|d)}\left[\log\frac{p(\theta|y,d)}{p(\theta)}\right] \\ &= \mathop{\mathbb{E}}_{y,\theta\sim p(\cdot|d)}\left[\log\frac{q(\theta|y,d)}{p(\theta)}\right] + \mathop{\mathbb{E}}_{y,\theta\sim p(\cdot|d)}\left[\log\frac{p(\theta|y,d)}{q(\theta|y,d)}\right] \\ &\geq \mathop{\mathbb{E}}_{y,\theta\sim p(\cdot|d)}\left[\log\frac{q(\theta|y,d)}{p(\theta)}\right]\end{aligned} \tag{2.5}$$

into a term comparable to the original objective and the difference between the true expected information gain and the variational information gain. By exploiting the positiveness of the divergence a lower bound can be obtained [25]. Even though the lower bound involves a distribution it gets used to recover the distribution-free as well as the distribution-based lower bounds used later on. In the variational estimators from [12] a neural network is used to represent the variational distribution directly $q(\theta|y,d)$. This stands in contrast to the estimators based on a critic network $T_\psi(y,\theta)$ not representing any kind of distribution. At first glance their connection is not visible, but it is based on a energy-based variational family [25]. This family

$$q(\theta|y,d) = \frac{p(\theta)}{Z(y)}e^{T_\psi(y,\theta)} \qquad\qquad Z(y) = \mathop{\mathbb{E}}_{p(\theta)}\left[e^{T_\psi(y,\theta)}\right]$$

reveals how all estimators rely on some form of distribution. A ton of different estimators can be derived by using these representations. In [25] the complete connections between the estimators are presented. Coming back the variational family can be plugged into the Barber-Agakov lower bound to receive an unormalized version. By applying Jensen's inequality it is easy to derive

$$\Psi(d) = \mathsf{MI}(y, \theta) \geq \mathop{\mathbb{E}}_{p(y,\theta|d)} [T_\psi(y, \theta)] - \log \mathop{\mathbb{E}}_{p(y|d)p(\theta)} \left[e^{T_\psi(y,\theta)} \right]$$

the Donsker-Varadhon lower bound [25]. This bound is the basis for the MINE-f bound and the INCE estimator introduced in the following sections. For more details on the mutual information see also [27, 25, 12].

2.5.2.2. Mutual neural information estimation

Most prominent usage of neural networks in an estimator is done by the mine estimator [28], which is a combination of the Donsker-Varadhan lower bound and a neural network. Instead being a real upper or lower bound to the mutual information it is only correlated with the estimator. However a slightly modified version was presented in [1], which results in a real lower bound to the mutual information. The equation for this estimator is given by

$$\Psi(d) = \mathrm{MI}(y, \theta) \geq \Psi(d, \psi) = \mathop{\mathbb{E}}_{p(y,\theta|d)} [T_\psi(y, \theta)] - e^{-1} \mathop{\mathbb{E}}_{p(y|d)p(\theta)} \left[e^{T_\psi(y,\theta)} \right]$$

difference between the mean under the joint and the exponential mean under the marginals. There are various concerns on the limitations when a distribution-free and high-confidence lower bound is used as an estimator for the mutual information [27]. More specifically the authors conclude that these estimators are upper bounded in terms of the sample size $\mathcal{O}(\ln K)$. Interpreting this results yield in an exponential number of samples to accurately approximate the mutual information. Nevertheless the existence of this upper bound there might be problems with small information gains and thus accurately estimate able.

2.5.2.3. Variational estimators

In [12] several variational estimators are compared. This work focuses on the variational posterior approximation. It is based on the bound from Barber-Agakov lower bound from

equation 2.5. By using a MCMC estimate on the expectation

$$\mathsf{MI}(y, \theta) \geq \mathop{\mathbb{E}}_{y,\theta\sim p(\cdot|d)} \left[\log \frac{q(\theta|y,d)}{p(\theta)} \right] \approx \frac{1}{K} \sum_{i=1}^{K} \log \frac{q(\theta_i|y_i,d)}{p(\theta_i)}$$

a lower bound called variational posterior. It might be used for implicit models where only samples from $p(y|d,\theta)$ can be generated. As in the previous estimators: By using the reparameterization trick from section 2.2.4.1.1 the gradient can be backpropagated into the design d. Their work considers also more sophisticated estimators based on the NMC estimator and an estimator based on two amortized networks. However these are not lower bounds and thus not further considered here.

3. Sequential optimal experimental design

While the optimal experimental design is viewed from a non-sequential perspective, there are two notions how it is sequential. Foremost continuously updating the prior with the learned posterior is a form of sequential updating. In the final algorithm it iterates between phases of exploration, e.g. where trajectories are recorded and phases of learning the new posterior. However if a trajectory is viewed with states $\mathbf{S}$ and actions $\mathbf{A}$ the trajectory itself is sequential. This section describes this sequential nature and how it gets solved inside of the optimization objective. It gets viewed from a control perspective and makes heave use of the optimization problems described in section 2.4. The general form of optimization problem is given

$$\begin{aligned} \min_{\mathbf{A}} \quad & \Psi(\mathbf{S}_{1:T-1}, \mathbf{A}) \\ \text{s.t.} \quad & s_{t+1} = f(s_t, a_t) \\ & a_{\min} \leq a_t \leq a_{\max} \\ & s_1 = s_0. \end{aligned} \tag{3.1}$$

by the following sequential environment. In section 4.2 the environments are explained in more depth. This work focuses only on 1-dimensional action spaces, but the methodology can be extended to the multidimensional case. However scaling the BNN and the estimators is a major challenge to accommodate for when multiple action dimensions are considered. In equation 3.1 a factorization over the trajectory formalized as

$$\Psi(\mathbf{S}_{1:T-1}, \mathbf{A}) = \sum_{t=1}^{T-1} \Psi_t(s_t, a_t)$$

can be used to make the problem tractable. Although this problem is comparable to normal control methods like model predictive control (MPC) or (iterative) linear quadratic regulator (iLQR), it can not be plugged into such a method as these methods solve for optimas where $\nabla_{a_t} \Psi_t(s_t, a_t) = 0$. By the discussion from section 2.4 the optimas lie

at the boundaries of the action space where it might hold that $\nabla_{a_t} \Psi_t(s_t, a_t) \neq 0$. This happens for example whenever each $\Psi_t(s_t, a_t)$ is concave. Most importantly it enables the use of shooting techniques. Multiple shooting exploits the forward model to unroll the trajectory and subsequently optimize the actions. Usually the first state s_1 is defined by the environment. By using the forward model $s_{t+1} = f(s_t, a_t)$ the trajectory ξ can be unrolled and more importantly all other states $\mathbf{S}_{2:T}$ are determined by the learned forward model. This type of data depends on a learned forward model and thus is different from the real result. Either one can differentiate through the constraints and solve this optimization problem completely, e.g. in log-barrier methods for non-linear optimization. However this work uses shooting to make the points independent and glue them together with linear constraints. By unrolling the trajectory the dynamics model can be linearized around the current trajectory and on top of that the objective can be quadratized. In iLQR the dynamics get linearized as well and this method is thus strongly related to iLQR with the focus on convex maximization. This work doesn't repeat the details about iLQR and MPC and more on the exploration methods.

3.1. Binary Exploration

The most basic algorithm for this type of exploration is very simplistic. Recall section 2.4.3 where the optimal solutions lie at the boundary. As this work considers only 1-dimensional action spaces this method is tractable for short horizons T. When scaling up either the action space dimensionality or the horizon this method gets intractable. Short horizons induce also simpler environments. This algorithm gets only evaluated on the Pendulum environment, as the other environments already need big horizons for goal directed control. Nevertheless it uses brute force evaluation to find the best exploration sequence. There might be a more sophisticated tree search producing better results. The complete method is formulated in algorithm 1. In control it's not beneficial to only execute the actions with the most system excitation. While the result of this method is discrete it can be smoothed out by exponential smoothing. However this filtering is extremely difficult to tune, depends highly on the problem and adds a lag to the action execution. Even if there is a more efficient selection scheme of the actions, transforming the discrete actions to continuous ones is a major challenge.

Algorithm 1: $\mathbf{A} = \text{discrete\_exploration}(f, \Psi, s_1, \bar{\mathbf{A}}, T)$

```
// Get all 2^T permutations
```
p = permutations($[a_{\text{min}}, a_{\text{max}}], T$);
v = float[2^T];

```
// Evaluate objective for all permutations
```
for $i \in \{1, \ldots, 2^T\}$ **do**
  $\mathbf{A} = p[i]$;
  $\mathbf{S} = \text{unroll}(f, s_1, \mathbf{A})$;
  $v[i] = \Psi(\mathbf{S}_{1:T-1}, \mathbf{A})$;
end

```
// Find the best permutation and pass back
```
$i = \arg\min v$;
Result: $\mathbf{A} = p[i]$

3.2. Convex-Concave Exploration

This section lists the proposed algorithm used in the learning process. It is assumed that $\Psi_t(s_t, a_t)$ is concave in a_t and this assumption is exploited throughout. While this assumption is not necessary for the following derivation, it is valid at least for the predictive variance. However it motivates the quadratic approximation used for the objective and explains the performance. Each objective part $\Psi_t(s_t, a_t) \approx g_t(s_t, a_t)$ is approximated by a quadratic function using a modified 2nd-order Taylor approximation

$$\Psi_t(s_t, a_t) \approx g_t(s_t, a_t) = (\Delta x_t)^\mathsf{T} Q_t (\Delta x_t)^\mathsf{T} + q_t^\mathsf{T} (\Delta x_t) + c_t$$

with Q_t symmetric and negative definite with $x_t^\mathsf{T} = (s_t^\mathsf{T}, a_t^\mathsf{T})$. Instead of using the complete Hessian in the approximation, this work uses a Gauss-Newton approximation to the real Hessian. See section 2.4.1 for more details on the usage. It is noteworthy at this point that the Hessian of the dynamics constraints is left out of the objective. The boundary conditions for the actions motivates the regularization of Q as it's Hessian is the identity. In practice it is difficult to determine the exact Lagrange multipliers, but this work simply sets it to a fixed value. As evidence suggests for BNN's the negative predictive variance is approximately concave (for shallow networks) and thus we Q_t is negative definite. If the function is highly non concave, this approximation might produce bad results. Note that

this section always talks in terms of minimizing a function as otherwise the terms convex and concave can be mixed up too easily. For each timestep t we have a concave objective $g_t(a_t)$, which we want to minimize. As g_t is concave the minimum only exists if the action space is bounded (e.g. $a_{\text{low}} \leq a_t \leq a_{\text{high}}$) and closed, or short compact. Hence the wanted minimum a_t^* is either a_{low} or a_{high} (in multidimensional case all combinations off edges (active constraints), which means if there is no tunnel between these edges [29], each of them is a local optima and we have $\mathcal{O}(2^d)$ local optima) and more importantly it does not fulfill the constraint $\nabla g_t(a_t^*) = 0$. All x_t in $g_t(x_t, a_t)$ are uniquely determined by the initial state x_1 and the actions $\mathbf{A}_{1:t-1}$ through the dynamic constraints $x_{t+1} = f(x_t, a_t)$.

3.2.1. Forming the complete optimization problem

Besides the quadratic approximation to the objective there are non-linear dynamics constraints remaining in the problem. These constraints can be easily linearized by a 1-st order Taylor approximation

$$f(s_t, a_t) \approx A_t \Delta s_t + B_t \Delta a_t + h_t$$

and integrated into the constrains as well. Obviously the linearized dynamics and the approximated quadratic objective have small approximation errors only in the neighborhood of the current trajectory. To mitigate this problem an additional parameter δ is introduced to account for the change in actions. The idea is to repeat this procedure δ iterations while constraining the size of the action update. This methodology can be compared to trust region methods see also [10, 30]. The complete optimization problem linearized around a trajectory ξ is formulated as

$$\begin{aligned}
\min_{\mathbf{A}} \quad & \sum_{t=1}^{T-1} (\Delta x_t)^\mathsf{T} Q_t (\Delta x_t)^\mathsf{T} + q_t^\mathsf{T} (\Delta x_t) + c_t \\
\text{s.t.} \quad & s_{t+1} = A_t \Delta s_t + B_t \Delta a_t + h_t \\
& a_t = \bar{a}_t + \Delta a_t \\
& a_{\text{max}} \geq \max(a_t, \delta \Delta a_t) \\
& a_{\text{min}} \leq \min(a_t, \delta \Delta a_t) \\
& s_1 = s_0 \\
& x_t^\mathsf{T} = (s_t^\mathsf{T}, a_t^\mathsf{T})
\end{aligned}$$

this sequential optimization problem with variables $\mathbf{S}$ and $\mathbf{A}$. Note that this algorithm cares only about the received actions and not the states. Recall the shooting techniques

were a trajectory is unrolled before the optimization, by that the next states are determined by the solution $\mathbf{A}$ and the current state of the environment. The outline of the problem is given in algorithm 2.

Algorithm 2: $\mathbf{A} = \text{dccp\_exploration}(f, \Psi, s_1, \bar{\mathbf{A}}, T, \delta)$

```
// Repeat δ iterations
```
for $k \in \{1, \ldots, \delta\}$ **do**

```
    // Approximate quadratic optimization problem around ξ
```
    $\mathbf{S} = \text{unroll}(f, s_1, \bar{\mathbf{A}})$;
    $A_t, B_t, h_t = \text{linearize}(f, \mathbf{S}_{1:T-1}, \bar{\mathbf{A}})$;
    $Q_t, q_t, c_t = \text{quadratic}(\Psi, \mathbf{S}_{1:T-1}, \bar{\mathbf{A}})$;

```
    // Solve for action update ΔA and update current actions Ā
```
    $\Delta\mathbf{A} = \text{solve\_dccp}(Q_t, q_t, c_t, A_t, B_t, h_t, \mathbf{S}, \bar{\mathbf{A}})$;
    $\bar{\mathbf{A}} = \bar{\mathbf{A}} + \Delta\mathbf{A}$

end

```
// Pass back the result
```
Result: $\mathbf{A} = \bar{\mathbf{A}}$

3.2.2. Adding regularization to the concave objective.

In control it is important to get actions $a_t \in [a_{\text{low}}, a_{\text{high}}]$ from the complete interval. Usually regularization techniques are employed to draw the optimized variables to zero. In terms of reinforcement learning this can be interpreted as regularizing around the zero policy. Despite these global regularization technique, also local regularization is used in control namely *slew rate penalty*. It implies that the actions are not to far from each other and thus some form of control stability is added. The idea is to add both contraints, e.g. the

slew rate penalty and the regularization, to the objective

$$
\begin{aligned}
\min_{\mathbf{A}} \quad & \sum_{t=1}^{T-1} \lambda \|a_t\|_2^2 + \mu \|a_{t+1} - a_t\|_2^2 + b_t \\
\text{s.t.} \quad & b_t \geq g(s_t, a_t) = (\Delta x_t)^\mathsf{T} Q_t (\Delta x_t)^\mathsf{T} + q_t^\mathsf{T} (\Delta x_t) + c_t \\
& s_{t+1} = A_t \Delta s_t + B_t \Delta a_t + h_t \\
& s_1 = s_0 \\
& a_t = \bar{a}_t + \Delta a_t \\
& a_{\text{max}} \geq \max(a_t, \delta \Delta a_t) \\
& a_{\text{min}} \leq \min(a_t, \delta \Delta a_t) \\
& x_t^\mathsf{T} = (s_t^\mathsf{T}, a_t^\mathsf{T})
\end{aligned}
$$

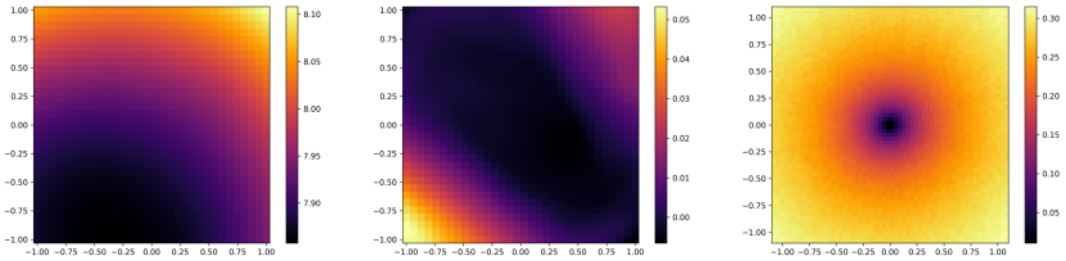
while simultaneously performing a epigraph transformation for each concave part $g_t(s_t, a_t)$ using the variable b_t. See section 2.4.4.1 for more details on this transformation. It doesn't change the solutions but makes the objective convex in it's variables by adding a non-linear (concave) constraint. This problem gets plugged into the DCCP solver, which in turn upper bounds the concave constraint by a linear constraint. For more details on the used optimizer see section 2.4.4.1 and [2].

4. Experiments

The experiment section compares different information gain estimators, in order to select an appropriate one for the latter algorithms. On top of this the model learning capabilities of different neural architectures get compared. Therefore datasets were extracted from the environment using a Wiener process. The BNN and the RMS ensemble are compared. Continuing from that the best model architecture is used to run the active learning part. Active learning is achieved by taking the actions from an algorithm introduced in section 3.

4.1. Bayesian linear regression and information gain estimators

This section compares the estimates by using either Mine-f [28] or the variational posterior approximation [12] to the real mutual information. The real mutual information is calculated using a nested Monte Carlo estimator. Both variation posterior and mine-f estimator are based on using a (normalized) variational distribtuion to lower bound the mutual information. It was applied to Bayesian linear regression and approximately calculates the mutual information. See figure 4.1 for a comparison between the original mutual information and the ones gathered from this estimator. Albeit these approximations go in the right direction. In fact they jump after each optimization step, and thus doesn't converge that stable. Both estimator seem to exhibit the same structural performance. This work focuses on the mine-f estimator for the approximation, but also some cases with the variational posterior are presented.



(a) Mine-f (b) Variational posterior (c) NMC

Figure 4.1.: Mutual Information
This compares the original mutual information of Bayesian linear regression calculated using nested Monte Carlo (NMC) to both estimators. On the left the MINE estimator is presented, whereas in the middle the variational posterior approximation is presented.

4.2. Environments

This section summarizes the used environments. They all have only one single action in their dynamics models. However they share very different characteristics what exactly the action manipulates. All spaces are actually bounded and by normalization they get manipulated to be from the interval $[-1, 1]$.

4.2.1. Pendulum

The easiest environment is the pendulum. It consists of a three dimensional state space as well as a one-dimensional action space. The standard environment from gym is used, whereas the dynamics are integrated by Euler's method. Usually the goal is to either do a swing up or follow a reference trajectory. The state space consists of the angle and the angle velocity, whereas the angle is represented in $\sin$ and $\cos$ space. A state tuple can thus be described as $(\cos(\theta), \sin(\theta), \dot{\theta})$ The only action is applied on the acceleration of the angle θ. The crux lies in bringing the pendulum up, because one from a hanging position a swinging motion is needed.

4.2.2. MountainCar

The mountaincar environment consists of a cart placed in a valley surrounded by two mountains. Ultimately the goal of the agent is to surpass the right peak and reach the goal behind. This environment has the smallest state space, e.g. it has a cart position x as well as a cart velocity $\dot{x}$. One configuration of the state space can be described as the tuple $(x, \dot{x})$. While the cart can not go right directly, it has to fulfill a swinging motion using the left mountain to surpass the right one. In some sense this is similar to the Pendulm, if the goal is to swing it around.

4.2.3. CartPole

The cartpole environment consists of a cart placed on a horizontal rail, in contrast to MountainCar where the cart move on a slope. Attached to the cart is a free moving pendulum similar to the Pendulum environment. The state space is thus a concatenation of the state space of the Pendulum and the MountainCar and a single configuration is given by $(\cos(\theta), \sin(\theta), x, \dot{\theta}, \dot{x})$ and contains five dimensions. The action accelerates the cart, which implicitly can move the Pendulum on it and subsequently stabilize it. This environment uses a Runge-Kutta integration method instead of the default Euler method.

4.2.4. Qube

By far the most advanced environment in this comparison is qube. It uses the environment from the quanser library released by the IAS lab. Basically it consists of two sticks attached to each other. One of them moves horizontally and the other is attached to this end by 90 degrees and thus moves vertically. The horizontal stick is attached to a qube and a motor. The goal of this environment is to swing up the vertical pendulum and stabilize it by only controlling the horizontal stick directly. One configuration of the state space is given by $(\cos(\theta_h), \sin(\theta_h), \cos(\theta_v), \sin(\theta_v), \dot{\theta_h}, \dot{\theta_v})$. The angle θ_h describes the angle of the horizontal stick relative to the qube and analogous the angle θ_v describes the angle of the stick relative to the plane.¸

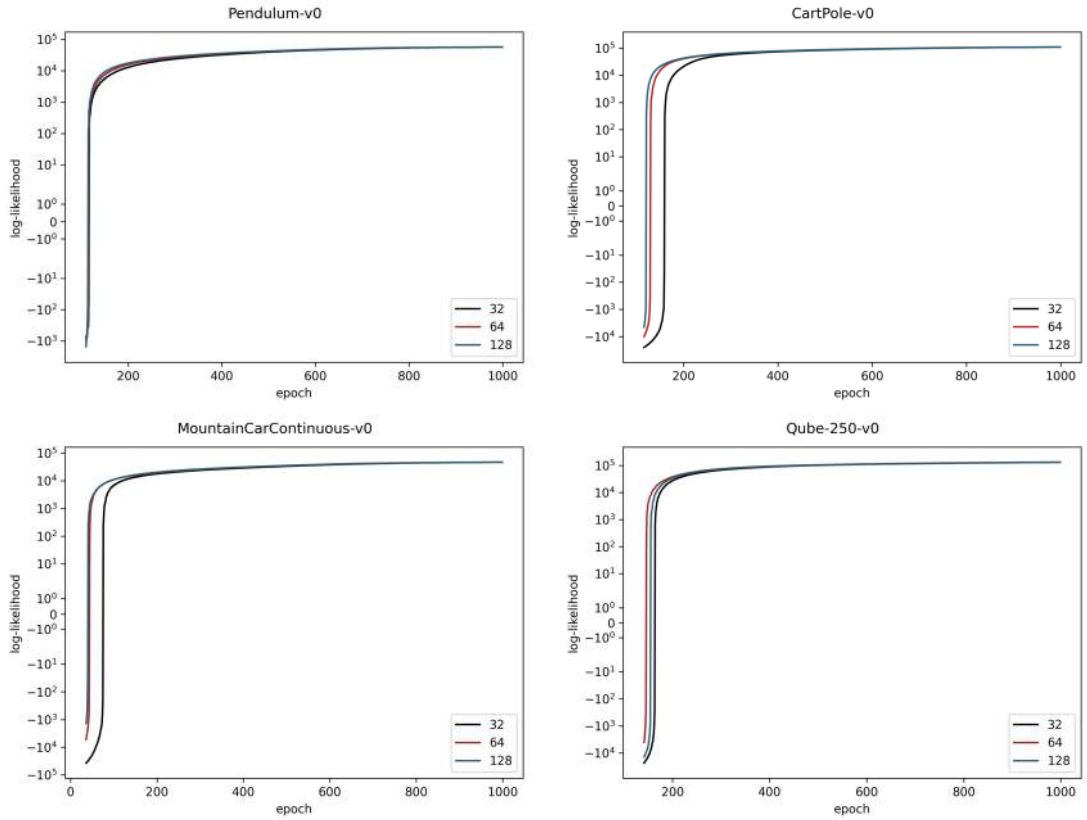


Figure 4.2.: BNN - Hidden Dimension
This compares the different runs with varying hidden dimensions. It can be ssen that the hidden dimension has a minor influence on the progress. Generally the smallest size which can fit the data should be chosen.

4.3. Model learning of $p(s_{t+1}|s_t, a_t)$

Prior to the active learning set up a comparison of the Bayesian neural network from section 2.2.3 are compared to the ensemble from section 2.2.5. The data is extracted from the environment, by sampling the actions from a Wiener process and is normalized before training. For angles only $\sin$ and $\cos$ features are used as described in the environments section in more detail. The reason is that the angle normalization is not differentiable and skip connections should be used with angles as otherwise there exist a convergent series

x_k such that $(\cos(x_k), \sin(x_k))^\mathsf{T}$ converges, but $x \neq \arctan(\cos(x), \sin(x))$ holds.

4.3.1. Model learning for BNN

The base configuration for the BNN is given in figure 4.3, while it was found by manual tuning of the parameters. This section compares the influence the hyper parameters have got onto the performance. Is is noteworthy that the training procedure for the BNN is relatively stable and produces a likelihood graph with minimal variance. If not stated otherwise the variance is obtained by using five different random seeds.

Three different settings for the hidden dimensions are considered. There is a clear trade off between having too much and not enough hidden dimensions. As can be verified in the plots with a hidden dimension of 128 the learning process takes more epochs to start in the beginning. Reducing the dimension improves training at the beginning, but in the end a hidden dimension of 64 gives the best result between the learning progress and the final performance. See figure 4.4 for the runs.

As for the hidden dimension three different settings for the learning rate were considered. See figure 4.4 for the learning graphs. For the learning rates 0.0005 and 0.005 the log likelihood looks good and stable and thus we conclude that both rates are appropriate. However in practice the bigger learning rate 0.005 was used. If the learning rate gets increased furthermore to 0.01 the training gets inherently unstable and from time to

	BNN	RMS
batch size	256	256
hidden dimension	64	64
hidden layers	2	2
prior sigma	0.1	0.1
learning rate	0.005	0.005

	BNN	RMS
k-step objective	5	2
num epochs	1000	1000
steps per epoch	10	10
num samples	32	32
skip connections	False	True

Figure 4.3.: Base configuration for BNN and RMS
This table shows the default configuration used in training. The hyper parameter search alters single parameters of this configuration.

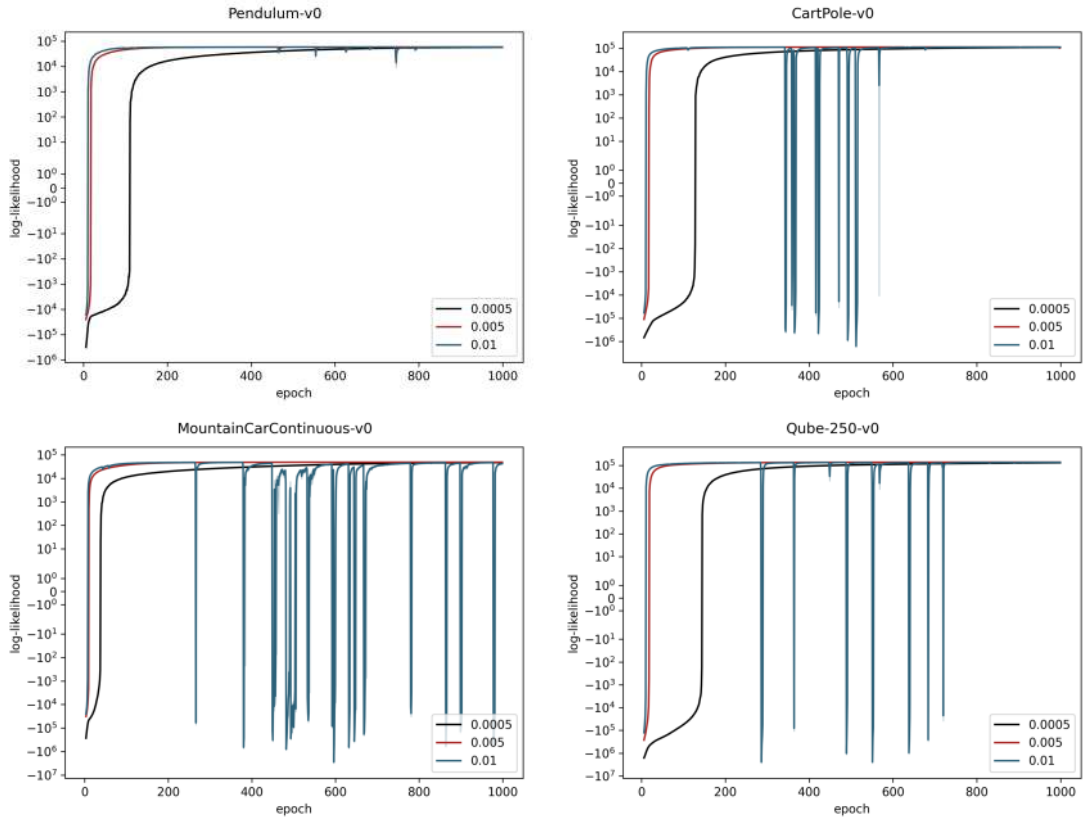


Figure 4.4.: BNN - Learning Rate
Generally the learning rate should not be too big or too small. When it is too big the learning gets unstable and the log likelihood has negative spikes. However if it too small the optimizer takes longer to reach its optimum.

time a very bad update is performed on the network, which reduces the log likelihood tremendously.

While the prior seems to be uninformative and thus the hyper parameters don't change that much during the training. As can be seen in figure 4.5 the only environment where it makes a difference is the Pendulum-v0. In this environment a prior scale of 0.01 results in faster learning at the beginning, but in the end a prior of 0.1 seems beneficial. Usually in neural network training only data tuples are viewed. However with the Bayesian

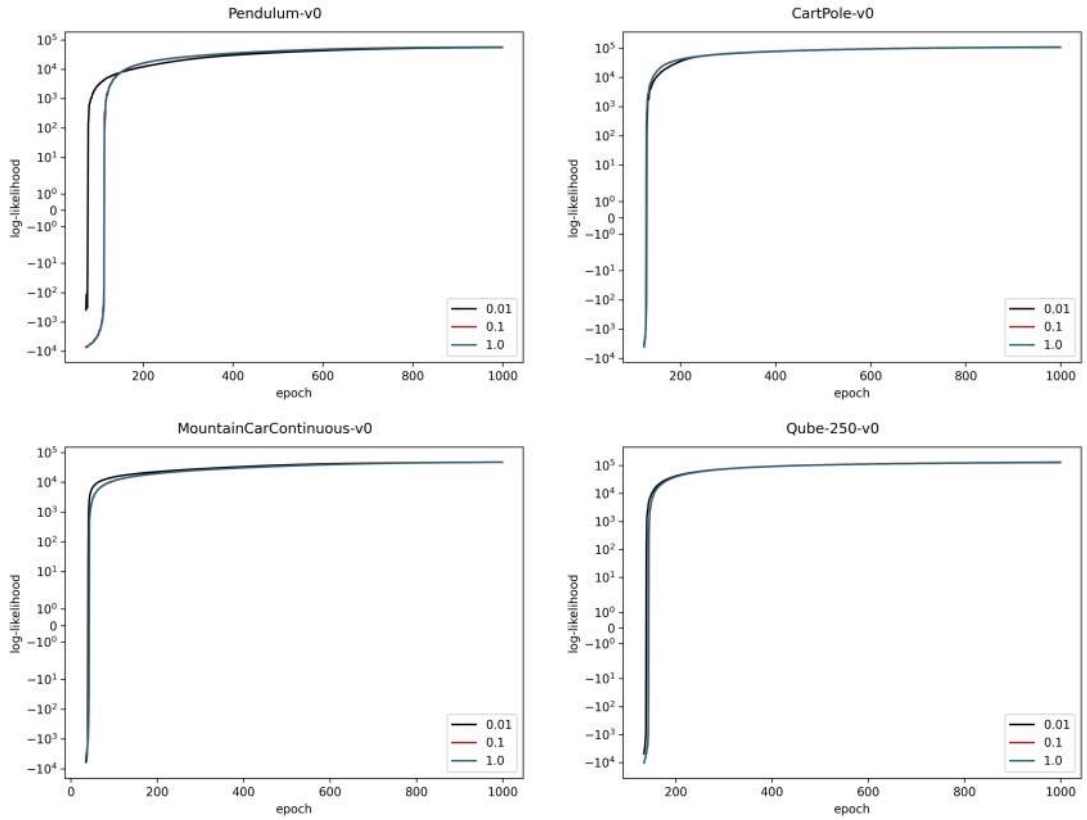


Figure 4.5.: BNN - Prior
In the case of a BNN the prior does not have that much influence. The Bayesian community calls this an uninformative prior and the influence of the prior on the posterior vanishes over time.

framework introduced earlier the alternative

$$\log p(\mathbf{S}_{1:K}, \mathbf{A}_{1:K-1}) = \log \prod_{k=1}^{K-1} p(s_{k+1}|s_k, a_k) = \sum_{k=1}^{K-1} \log p(s_{k+1}|s_k, a_k)$$

likelihood can be used. In figure 4.6 it is clearly visible that a longer K generates better learning graphs. This could also be verified looking at the predicted trajectories. They seem to be much smoother, also in regions where it fails to predict the ground-truth.

4.3.2. Model learning for RMS Ensemble

This section examines the performance of the ensemble. As for the BNN a base configuration is altered, which is given in figure 4.3. The plots corresponding to the hidden dimension and the learning rate are added to the appendix, e.g. figure A.1 and A.2, because they share a similar characteristic as the BNN. The prior was alternated between 0.01, 0.1 and 1.0. It is noteworthy, whenever the prior of the ensemble is too big this could result in divergent behavior. However skip connections were used and thus the targets are much smaller, which implies that the weights and thus the prior should be

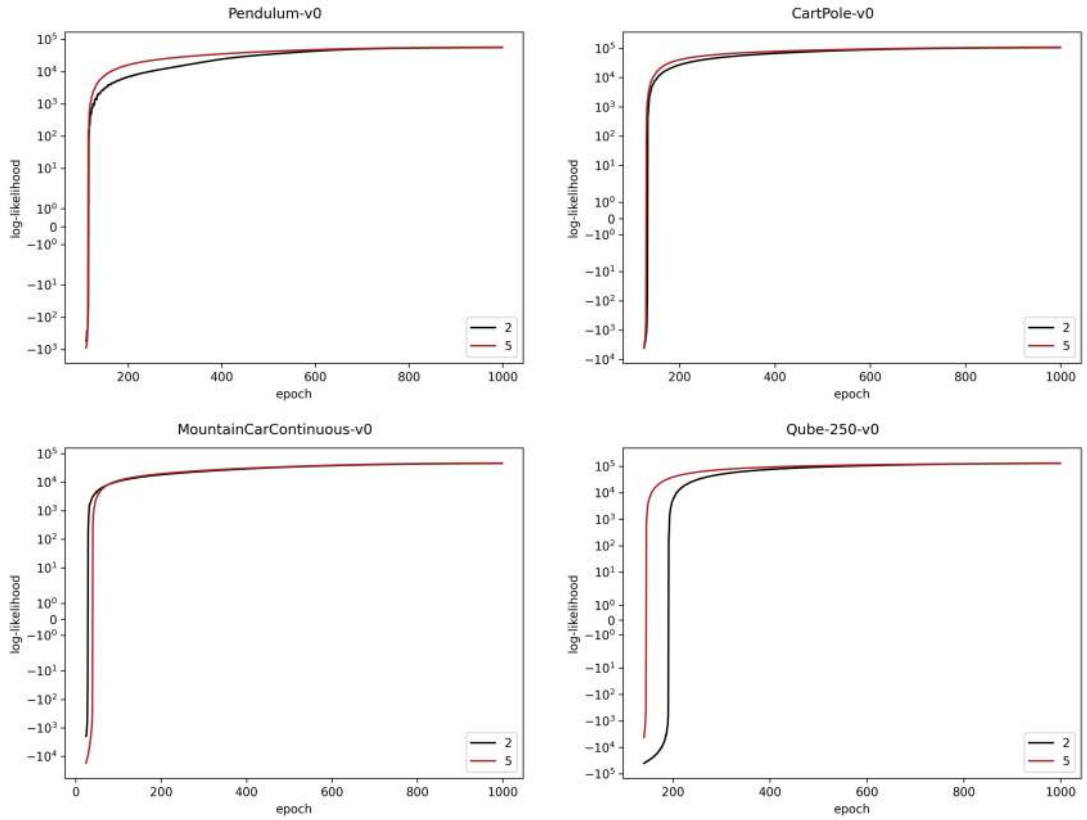


Figure 4.6.: BNN - K-step objective
For the BNN a longer K-step objective yields better and more importantly smoother results. The model seems to smooth out errors in the future, even if it fails for some steps.

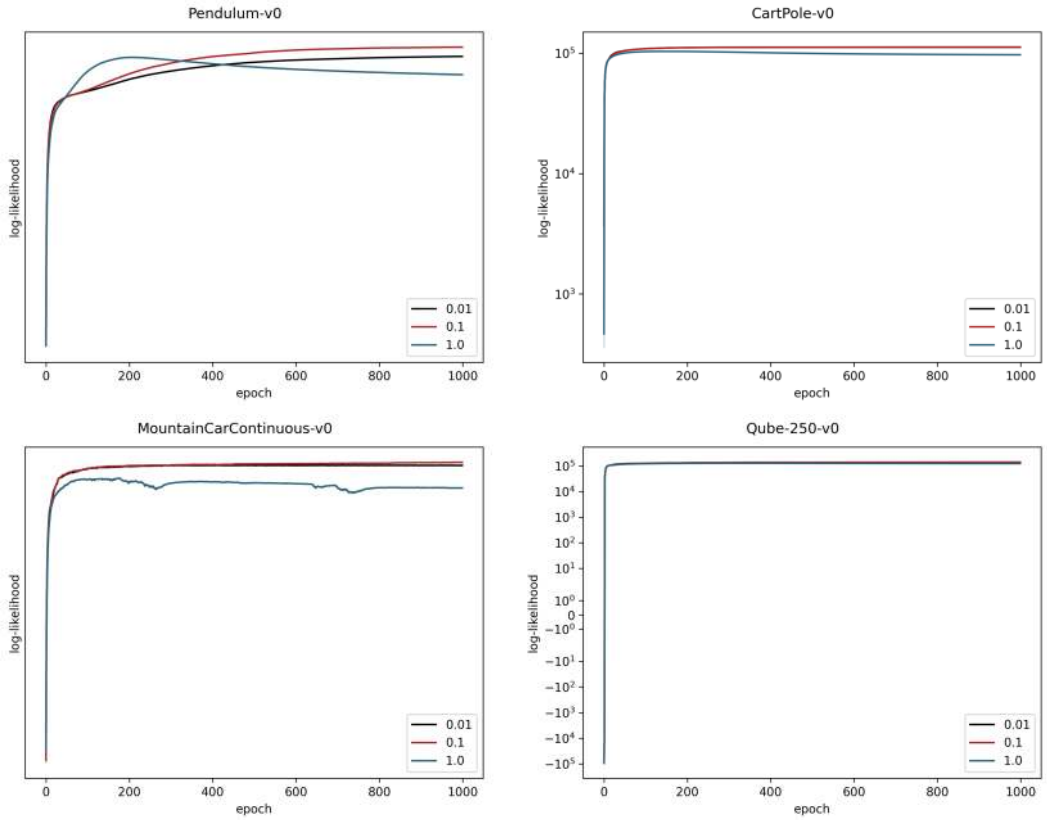


Figure 4.7.: RMS - Prior
On the ensemble the prior has a bigger influence. Choosing a prior with big scales yields in diverging behavior. Hence a smaller prior should be chosen.

chosen much smaller. Not using skip connections drastically reduced the performance.

Another important factor is determined by the number consecutive steps in training. Training the ensemble with a multiple steps likelihood converges much slower. This effect is contrary to the BNN, where increasing the steps in the likelihood improves the convergence speed. When compared to figure 4.6. Modifying the learning rate exhibits the expected behavior in the context of neural networks. The same holds for the hidden dimension and in this context a bigger network size yields slower convergence at the beginning but a larger capacity at the end.

4.3.3. Selecting the optimal run

For the learning part, the best network configuration for each environment is selected automatically. The selection scheme used for selecting the best hyper parameter configuration uses a simple exponential average

$$\hat{v}_{e+1} = \gamma \hat{v}_e + v_{e+1}$$

to smooth out the log likelihood and fold it into one value. After this transformation the best configuration can be selected by taking the maximum. One might also use the integral

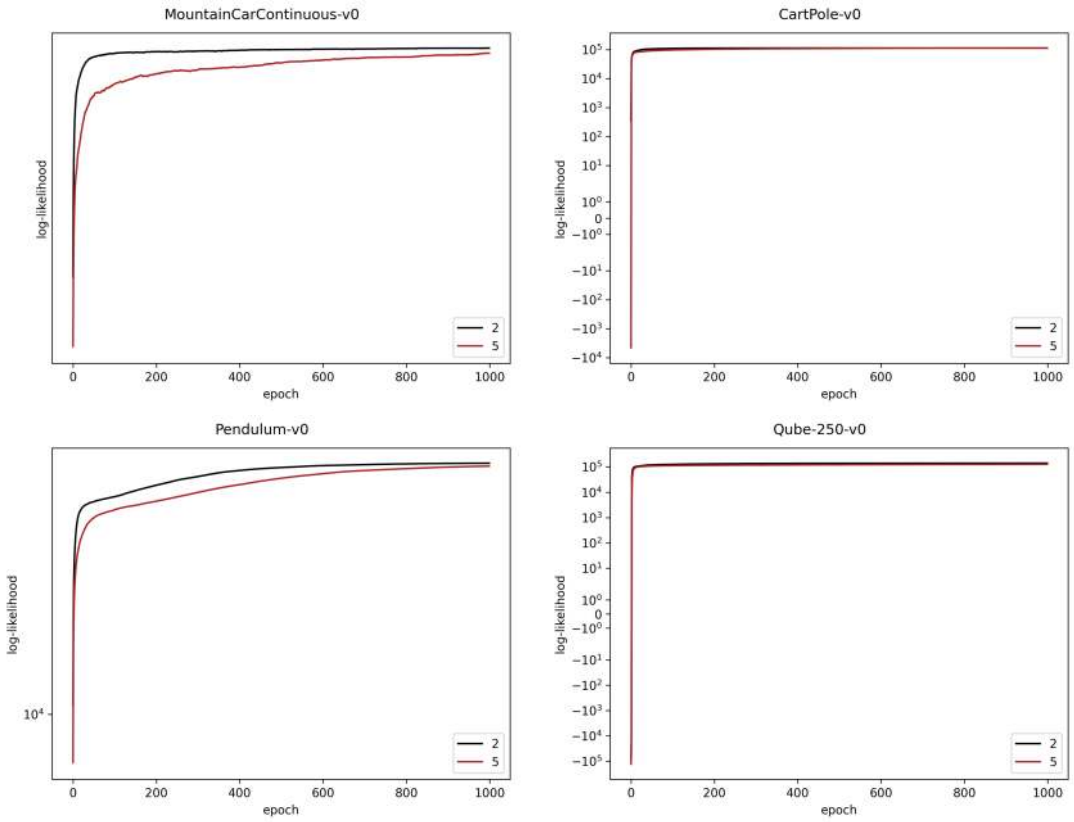


Figure 4.8.: RMS - K-step objective
As compared to the BNN, the RMS ensemble is much more sensitive to the number of steps in the objective. This effect doesn't increase the performance, instead the optimizer takes more iterations to solve this problem.

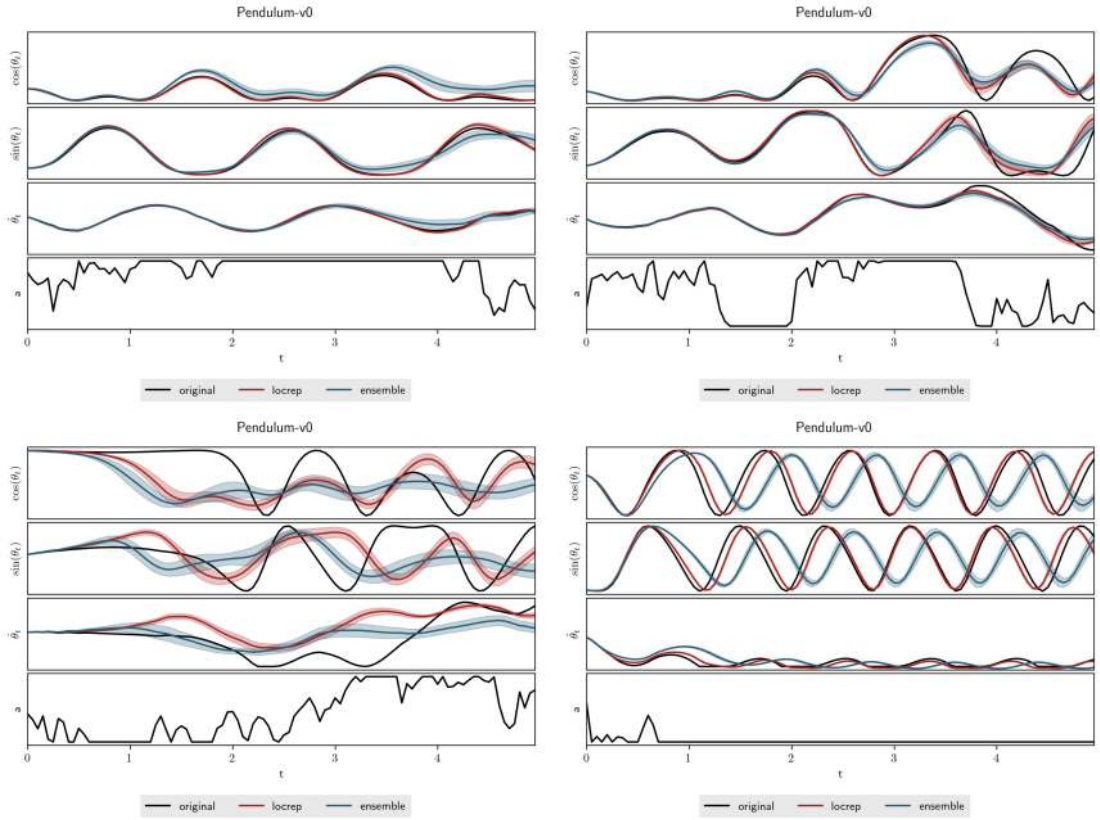


Figure 4.9.: Supervised trained models for Pendulum-v0
BNN and RMS ensemble get evaluated on the validation set trajectories. The trajectories stay the same for all comparisons. It is noteworthy how the BNN achieves a slightly better accuracy.

underneath the curve but in experiments this rule selected semi optimal configurations.

4.3.4. Comparing the predictions of both models

In figure 4.9 the predictive performance of both models, namely the ensemble and the BNN, gets evaluated. The two upper trajectories are represented very well by both networks, whereas the BNN seems to have a slight advantage over the ensemble. Both trajectories at the bottom are harder to predict by the models. Nevertheless the BNN outperforms

the ensemble at the bottom right trajectory. The same analytical results are obtained for MountainCar. However in more complex environments, like CartPole-v0 and Qube-v0 the ensemble seems to be superior over the BNN. The predicted trajectories are presented in section A.1.2. Due to this observation the BNN seems to have a scaling disadvantage, but excels at smaller environments.

4.4. Exploration

This section shows the performance of the algorithms from section 3. The best run configuration is determined by the selection rule from 4.3.3. For the pendulum environment the explored state-space gets compared to the states generated by using a Wiener process. First of all the binary exploration from section 3.1 gets evaluated.

4.4.1. Binary Exploration

The most basic algorithm for performing this kind of exploration is binary exploration from section 3.1. It gets evaluated using two different objectives, e.g. predictive variance and mine-f estimator from section 2.5.1 and 2.5.2.2, respectively. It is noteworthy, that the other estimators are either computationally very expensive, e.g. INCE , or exhibits really large values, e.g. VPOST 2.5.2.3, as evaluated in the first experiments section 4.1.

4.4.1.1. Predictive Variance

This section evaluates how the methods perform using the predictive variance. The predictive variance can be obtained for either the BNN or the RMS ensemble and it's estimation quality depends on the number of samples. As evidence suggest and explained in [18] the variance of shallow Bayesian neural networks is convex. This is exploited in the DCCP algorithm.

While this method finds the global optimum, it is very slow when big horizons T are used. See section 3.1 for details about the method. For all environments a short horizon of $T = 10$ was used. The log likelihood curves are presented in figure 4.10. For the environments with swinging components the method increases fairly well. Note that for MountainCarContinuous-v0 the training gets instable. Due to the short horizon $T = 10$, it is very unlikely that the cart reaches the right goal state. As a result the carts keeps

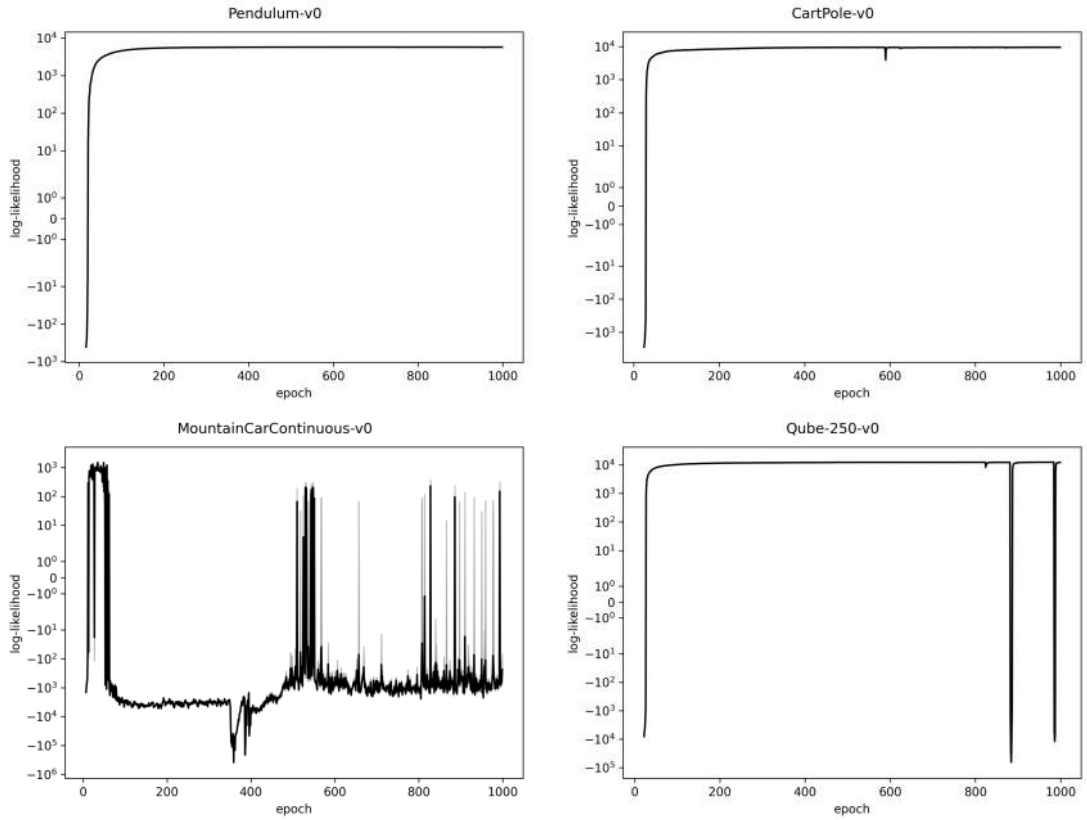


Figure 4.10.: BNN - Binary exploration
The log likelihood graphs for discrete exploration on all environments. Due to the short horizons the MountainCarContinuous-v0 environment does converge at first place, but degrades in performance afterwards.

exploring in between both peaks and the data points in this area are much more dense compared to the outer regions.

In figure 4.11 the explored state space is displayed. A horizon of $T = 10$ is used and only discrete actions are applied. It is noteworthy, that the BNN covers the point $(0, 0)$ very well, but fails to represent the extreme edge cases. For the RMS it also arrives the center, but also gathers some points from the velocity edge cases. However the circles are much more circular and follow a certain pattern. This stands in contrast to the BNN, where the points are distributed more uniformly.

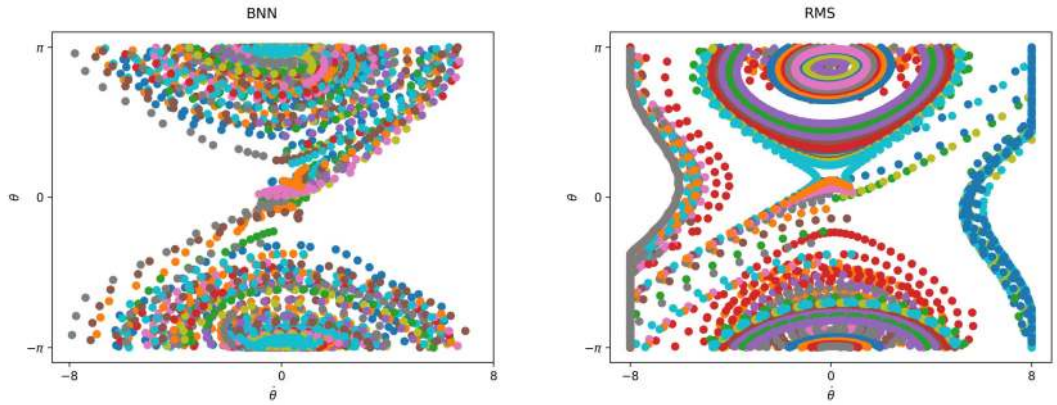


Figure 4.11.: State space for binary exploration with predictive variance.
This plot shows the state space for binary exploration. The predictive variance is used as an objective, whereas each trajectory is printed in a different color.

4.4.1.2. MINE

Instead of using the predictive variance to predict the information gain, a different estimator called the mine estimator is used. The performance gets compared to the variational

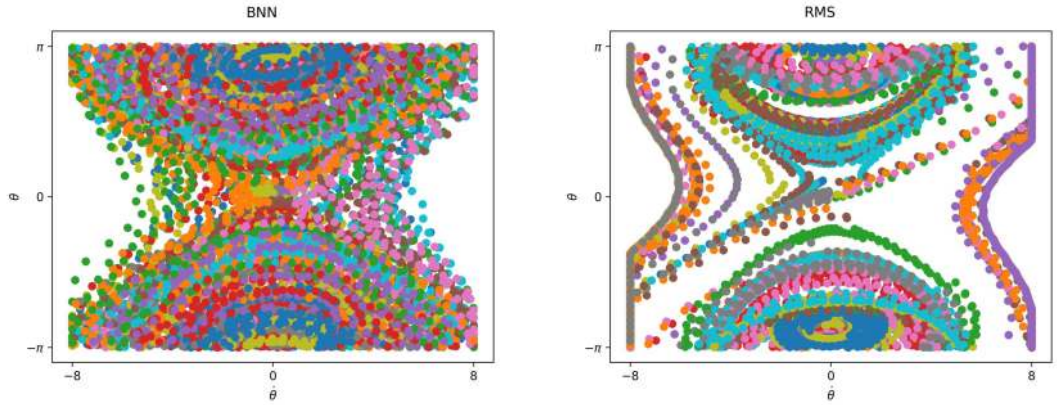


Figure 4.12.: State space for binary exploration with MINE-f.
This plot shows the state space for binary exploration using the MINE-f objective. Each trajectory is printed in a different color.

posterior in section 4.1. It used an additional amortized neural network, to perform an approximation to this quantity. The obtained trajectories, for all methods including the ones from the predictive variance, are presented in figure 4.13 and compared to the ground truth. Although an additional estimator is used the distribution of the state space shows the same characteristic: In figure 4.12 the state space for the BNN with MINE is much better than the combination of the RMS ensemble and the MINE estimator. However it seems that using a MINE estimator is beneficial over the predictive variance. In latter experiments the overall performance of both estimators depends on the problem and thus

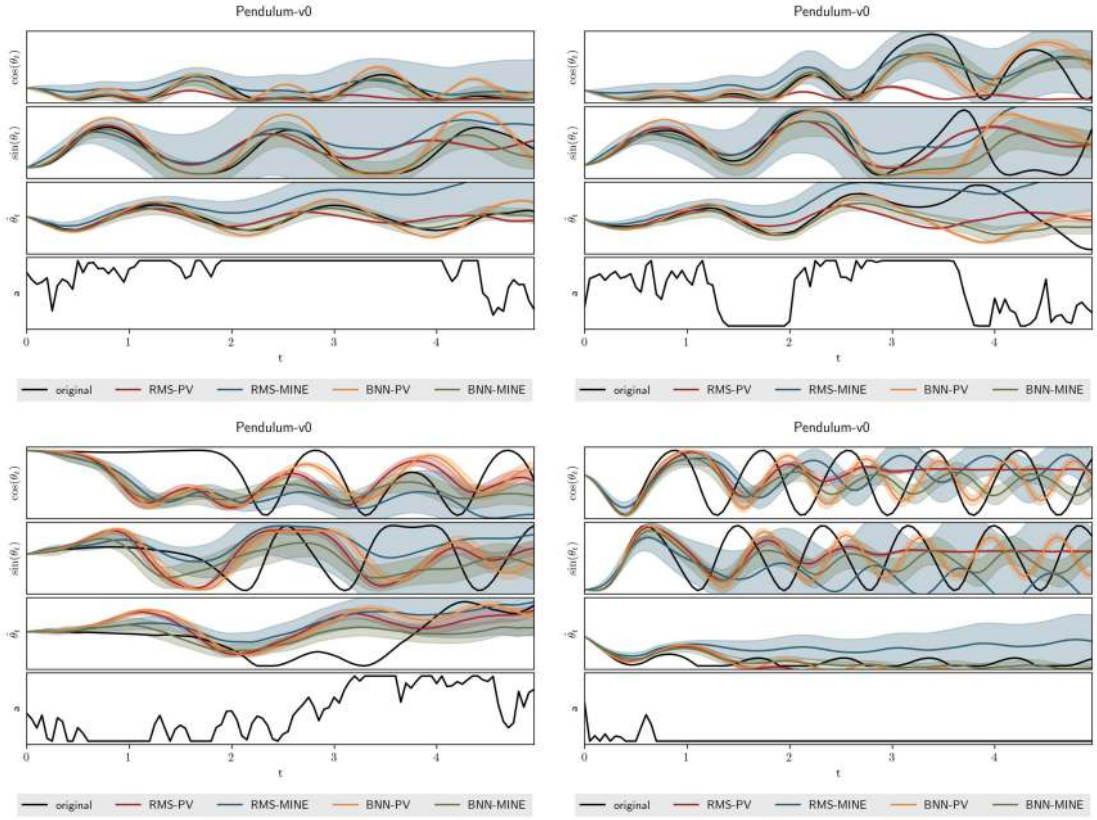
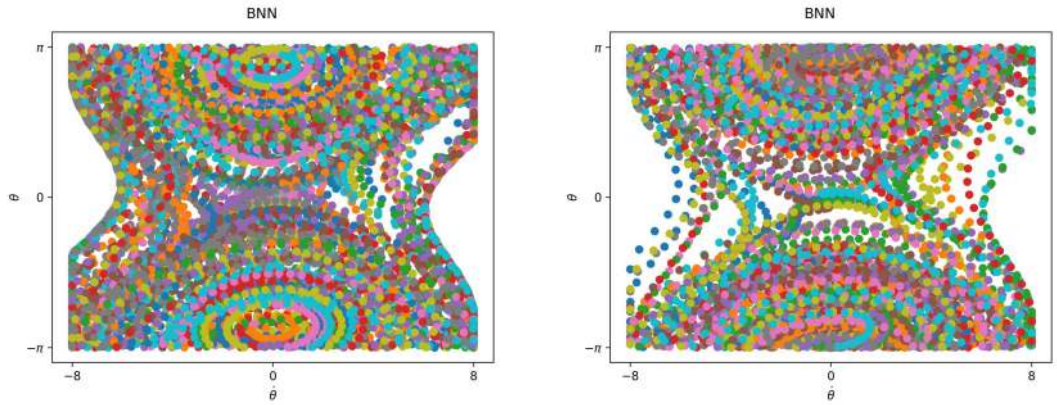


Figure 4.13.: Learned models for discrete exploration
A comparison of the models from the discrete exploration. All combinations of BNN and RMS ensemble together with the predictive variance objective and MINE-f are evaluated on the validation trajectories.



(a) BNN-PV (b) BNN-MINE

Figure 4.14.: State space for DCCP exploration
This plot shows the state space for DCCP exploration. For both plot a BNN was used, whereas each trajectory is printed in a different color. On the left side is the predictive variance and on the right the MINE-f objective.

the right one has to be chosen problem specific.

4.4.2. DCCP Exploration

The following section evaluated the methods using the relaxed optimization problem from section 3.2. They are executed on all presented environments and the resulting models are compared on some trajectories from the validation set. It is noted that the RMS ensemble is left out from this evaluation as it fails to share the same distributional behavior in binary exploration in contrast to the BNN, see section 4.4.1. By this argument this section compares the performance of the estimators in conjunction with the BNN. It was empirically verified, that a set of point estimates in conjunction with the estimators of the mutual information yields bad results.

The predicted trajectories on the Pendulum-v0 can be viewed in figure 4.15. For the Pendulum-v0 the predictive variance and the MINE estimator yield better predictive results than the variational posterior. However for the upper right trajectory from 4.15 the variational posterior method gets a prediction which is not valid in the middle, but at the the end. Nevertheless it is concluded that the variational posterior and the MINE objective

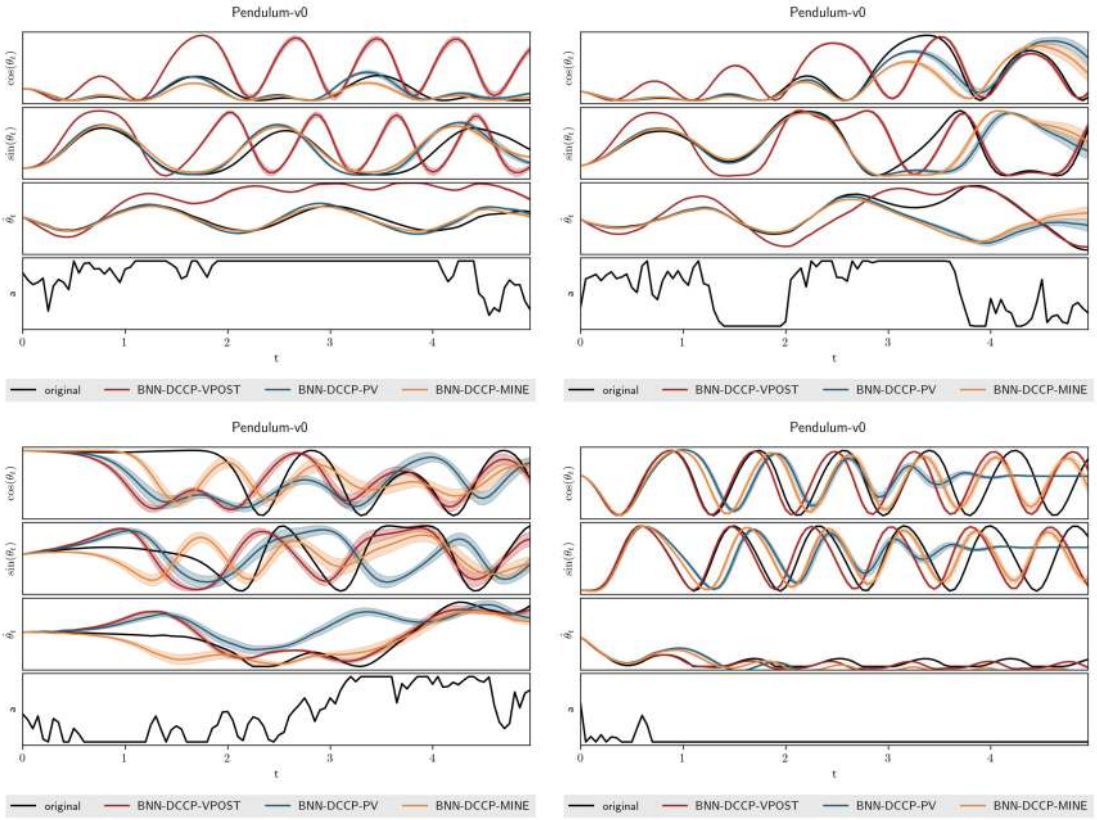


Figure 4.15.: Learned models for DCCP exploration
This plot shows the learned models for the Pendulum-v0. As explained in the section, only the BNN is further considered. Three objectives are considered, namely predictive variance (PV), MINE-f and variational posterior (VPOST) are presented.

deliver comparable exploration results. The trajectories from the other environment are presented in section A.2.2. Especially in the environment MountainCarContinuous-v0 the MINE objective delivers better results. in contrast to the predictive variance, where it fails predict the end of the trajectory. Albeit the differences are not so extreme, the same behavior can be examined for the CartPole-v0 environment. Last but not least both method perform equally well on the Qube-v0.

For the environments Pendulum-v0 and CartPole-v0 the BNN is evaluated using three different objectives. These objectives include the predictive variance, MINE-f estimator

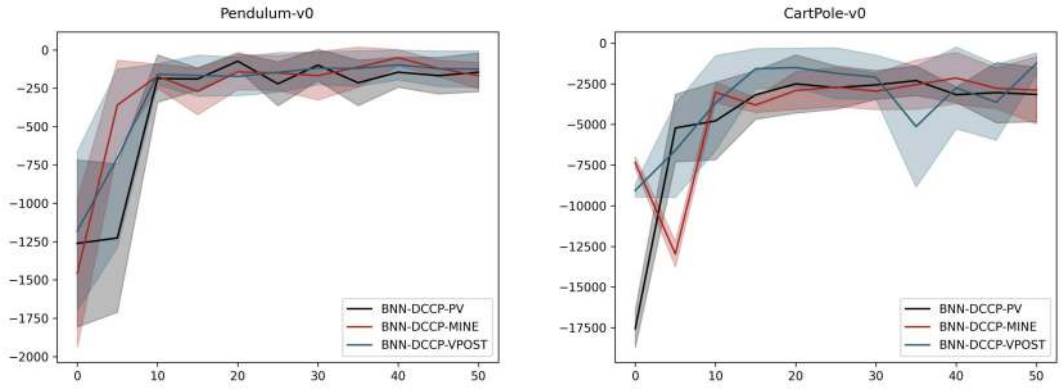


Figure 4.16.: Control using the learned models
The reward graphs of the environments Pendulum-v0 and CartPole-v0. The reward is displayed for the BNN with PV, MINE and VPOST. Note that the reward was evaluated after 5 epochs of active learning. They all share the same overall performance in control.

and the variational posterior approximation. All of these combinations are tested in a control setting in section 4.5.

4.5. Control

The generated models are used in a control setting on all environments except MountainCarContinuous-v0 and Qube-v0. For Qube-v0 the problem lies in the hardness of model predictive control (MPC) itself and with the supervised trained model, it was difficult to get the controller running. For the MountainCarContinuous-v0 environment long horizons are needed, where the iLQR optimizer fails. Hence only Pendulum-v0 and CartPole-v0 are considered in a control setting. After a model is learned, the reward is evaluated over 5 different seeds. After five epochs of learning the distribution of rewards gets evaluated, and the rewards for multiple epochs are plotted in figure 4.16. All control signals were generated using the library from [31]. Further tuning in the exploration process, might also solve Qube-v0 and MountainCarContinuous-v0. However these parts were left out from this thesis.

5. Results

This section summarizes the main results made during the experiments and the foundational analysis of the methodology. First of all the shape of the exploration policy is examined in-depth. Afterwards that both mutual information estimators are compared and it is discussed, which one to choose. Finally a comparison between BNN and RMS is given, and it is explained that the BNN is better suited for sample based estimators of the mutual information.

5.1. Exploration policies

For some environments the exploration policy does the same thing as a goal directed method. As only one-dimensional environments were used, this can not be generalized to multiple action dimensions easily. The effect is most prominent on the MountainCar environment, where optimal exploration ends up in the goal state. The other three environments contain a circular pendulum and are easier to explore, e.g. the horizons can be set much shorter.

5.2. MINE and variational posterior

Two estimator for the mutual information, namely the MINE estimator and the variational posterior are compared. They both utilize an amortized network in their estimation scheme. As presented in figure 4.15 and section 4.1, both estimators show equally good performance. In contrast to the variational posterior, the MINE estimator additionally needs samples from the marginal density. Despite this computational burden, the MINE estimator gives more stable results in the case of Bayesian linear regression. Overall this work favors the MINE-f estimator, because it better fits into the DCCP framework.

5.3. BNN model better suited

Although the ensemble shows good performance and acceptable variance estimates, the number of samples is fixed and they cannot be easily resampled as each ensemble member is a point estimate of the target function. This stands in contrast to the BNN, which is able to change the number of samples dynamically. For the predictive variance both models are equally well suited, while the BNN is better suited for sample-based estimation of the mutual information. By this argument, only the BNN was examined exhaustively in the experiments section. It was shown that the RMS ensemble fails to exhibit the same state distribution as the BNN.

5.4. Horizon for MountainCarContinuous-v0

Sometimes the model fails to explore, as can be seen for the MountainCarContinuous-v0 environment. Investigation yields that the cart tries to explore the left area, e.g. it drives the cart with full speed to the left. This happens, because control on this environment needs big horizons. Simultaneously the introduced method yields better solution with short optimization horizons. It is not easy how to trade off between these effect. This has to be further researched and is an integral part for solving these types of environments. Skip connections might be used, but these have to be integrated in the Bayesian framework.

6. Outlook

A short outlook is given on how to progress with this research. In general three pathways are considered. The first one dives deeper into the underlying problem and is able to cope with indefinite quadratic approximations and the control regularization. Alternatively the submodularity property in the discrete case might be exploited to make a more efficient selection scheme for the discrete actions. Last but not least the method could be extended and tuned on multiple dimensions. Theoretically algorithm 2 could be extended to deal with multiple actions.

6.1. Semi definite programming relaxation

Note that the second derivative of the convex/concave objective gets approximated by the Gauss Newton matrix and is thus either positive or negative definite, respectively. However if the full Hessian is approximated the resulting program is a quadratically constrained quadratic program, see section 2.4.5. Under these circumstances a semidefinite relaxation to the original problem can be made. Details about this can be found in [6]. It is noted that this is only a different approach of solving the underlying problem. However it might reduce the approximation error, introduced when using the Gauss Newton matrix instead of the Hessian.

6.2. Exploit submodularity and smooth actions

While it is not directly clear how this can be applied over time with discrete variables. But the submodularity of the mutual information might be exploited [32]. This would reduce drastically the computational efforts, as computations can be reused. However this goes beyond the scope of this thesis. Another interesting direction, would be to create a super

optimization problem. The upper level problem should take the discrete actions from the lower level problem as input and smooth them out. The goal is to get actions from the complete interval $x \in [-1, 1]$

6.3. Extend to multiple action dimensions

While only environments with one-dimensional actions are considered, the method might be adapted to more complex environments with multiple continuous actions. This optimization problem inherently suffers from the curse of dimensionality, in the sense that there are exponential many local optima. Despite this fact it is not clear how these local optimas can be compared or if they are comparable. It might be not bad, to have exponentially many local optimas, if the gained solutions are well distributed and not from one part of the input space.

Bibliography

[1] S. Kleinegesse and M. U. Gutmann, "Bayesian experimental design for implicit models by mutual information neural estimation," *arXiv preprint arXiv:2002.08129*, 2020.

[2] X. Shen, S. Diamond, Y. Gu, and S. Boyd, "Disciplined convex-concave programming," 2016.

[3] M. Schultheis, B. Belousov, H. Abdulsamad, and J. Peters, "Receding horizon curiosity," in *Conference on Robot Learning*, pp. 1278–1288, 2020.

[4] T. Pearce, F. Leibfried, A. Brintrup, M. Zaki, and A. Neely, "Uncertainty in neural networks: Approximately bayesian ensembling," *arXiv preprint arXiv:1810.05546*, 2018.

[5] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in neural information processing systems*, pp. 2575–2583, 2015.

[6] J. Park and S. Boyd, "General heuristics for nonconvex quadratically constrained quadratic programming," *arXiv*, pp. arXiv–1703, 2017.

[7] M. Buisson-Fenet, F. Solowjow, and S. Trimpe, "Actively learning gaussian process dynamics," in *Learning for Dynamics and Control*, pp. 5–15, PMLR, 2020.

[8] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

[9] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, pp. 1–9, 2013.

[10] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Conference on Artificial Intelligence (AAAI)*, 2010.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[12] A. Foster, M. Jankowiak, E. Bingham, P. Horsfall, Y. W. Teh, T. Rainforth, and N. Goodman, "Variational bayesian optimal experimental design," in *Advances in Neural Information Processing Systems*, pp. 14036–14047, 2019.

[13] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[14] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[15] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International Conference on Artificial Intelligence and Statistics*, pp. 1540–1552, 2020.

[16] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *arXiv preprint arXiv:1906.10652*, 2019.

[17] A. Y. Foong, Y. Li, J. M. Hernández-Lobato, and R. E. Turner, "'in-between'uncertainty in bayesian neural networks," *arXiv preprint arXiv:1906.11537*, 2019.

[18] A. Y. K. Foong, D. R. Burt, Y. Li, and R. E. Turner, "On the expressiveness of approximate inference in bayesian neural networks," 2019.

[19] I. Osband, B. Van Roy, D. J. Russo, and Z. Wen, "Deep exploration via randomized value functions.," *Journal of Machine Learning Research*, vol. 20, no. 124, pp. 1–62, 2019.

[20] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction," *arXiv preprint arXiv:1910.09457*, 2019.

[21] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning," in *International Conference on Machine Learning*, pp. 1184–1193, 2018.

[22] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.,"

[23] M. C. Grant, *Disciplined convex programming*. PhD thesis, Stanford University, 2004.

[24] T. Rainforth, R. Cornish, H. Yang, A. Warrington, and F. Wood, "On nesting monte carlo estimators," in *International Conference on Machine Learning*, pp. 4267–4276, 2018.

[25] B. Poole, S. Ozair, A. v. d. Oord, A. A. Alemi, and G. Tucker, "On variational bounds of mutual grant2004disciplinedinformation," *arXiv preprint arXiv:1905.06922*, 2019.

[26] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[27] D. McAllester and K. Stratos, "Formal limitations on the measurement of mutual information," in *International Conference on Artificial Intelligence and Statistics*, pp. 875–884, 2020.

[28] M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, A. Courville, and R. D. Hjelm, "Mine: mutual information neural estimation," *arXiv preprint arXiv:1801.04062*, 2018.

[29] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," in *Advances in Neural Information Processing Systems*, pp. 8789–8798, 2018.

[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[31] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for End-to-end Planning and Control," 2018.

[32] M. Queyranne, "Minimizing symmetric submodular functions," *Mathematical Programming*, vol. 82, no. 1-2, pp. 3–12, 1998.

A. Appendix

This section contains plots with predicted trajectories and also likelihood computation graphs. They are considered as additional content, which are used in the main text.

A.1. Supervised Model Learning

All graphs regarding the supervised model learning are part of this section, in the sense that the data was extracted by a Wiener process policy and stayed fixed throughout the training. It is noteworthy that the trajectories from this set are used for validation and testing of the active model learning part.

A.1.1. Training runs

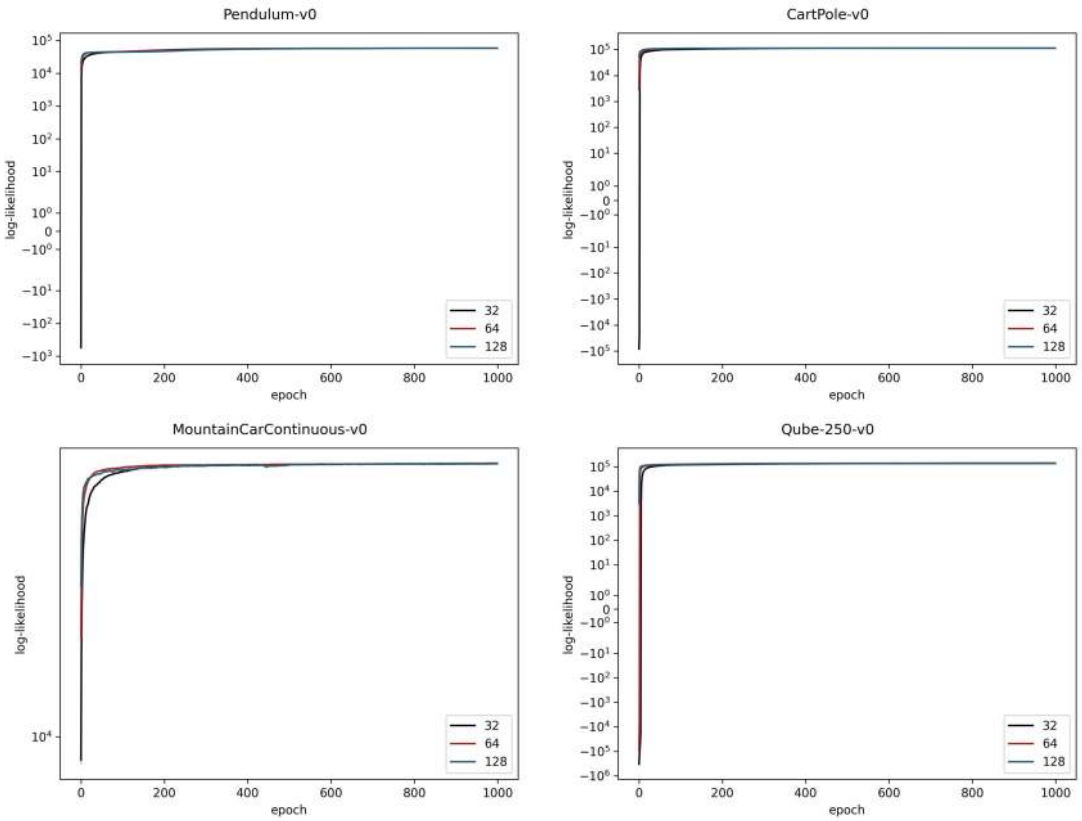


Figure A.1.: RMS - Hidden Dimension

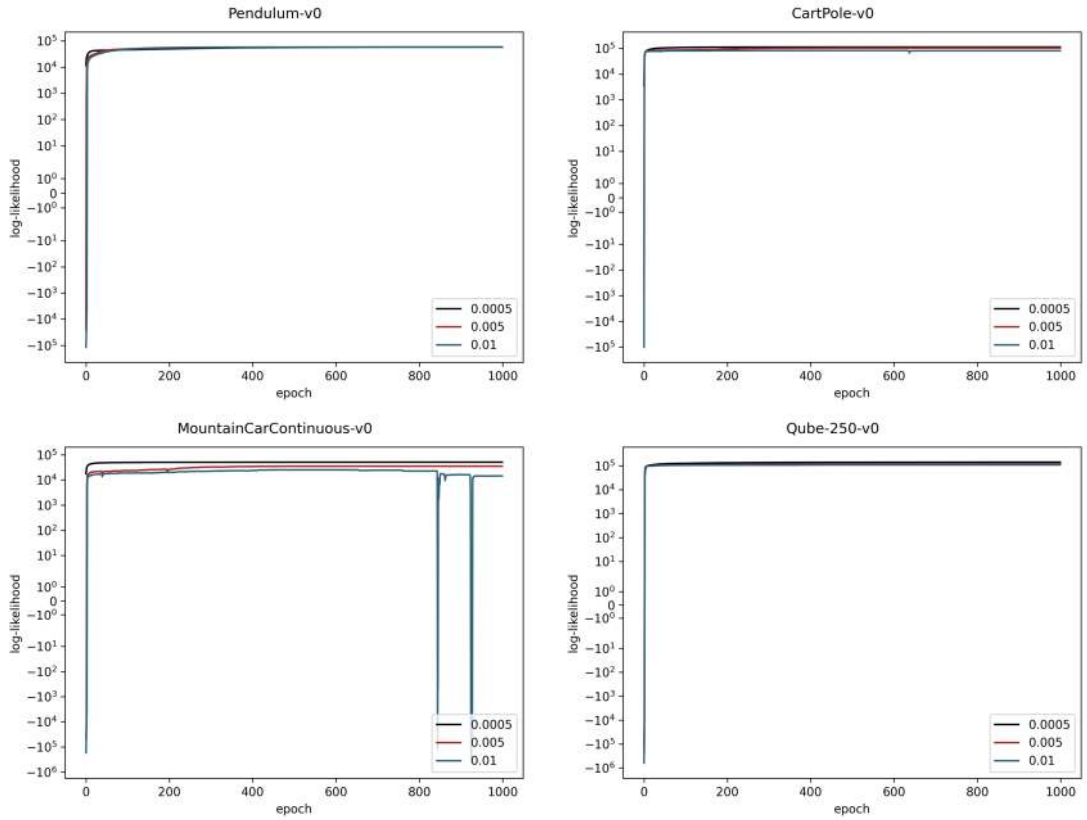


Figure A.2.: RMS - Learning Rate

A.1.2. Predicted Trajectories

A.1.2.1. MountainCarContinuous-v0

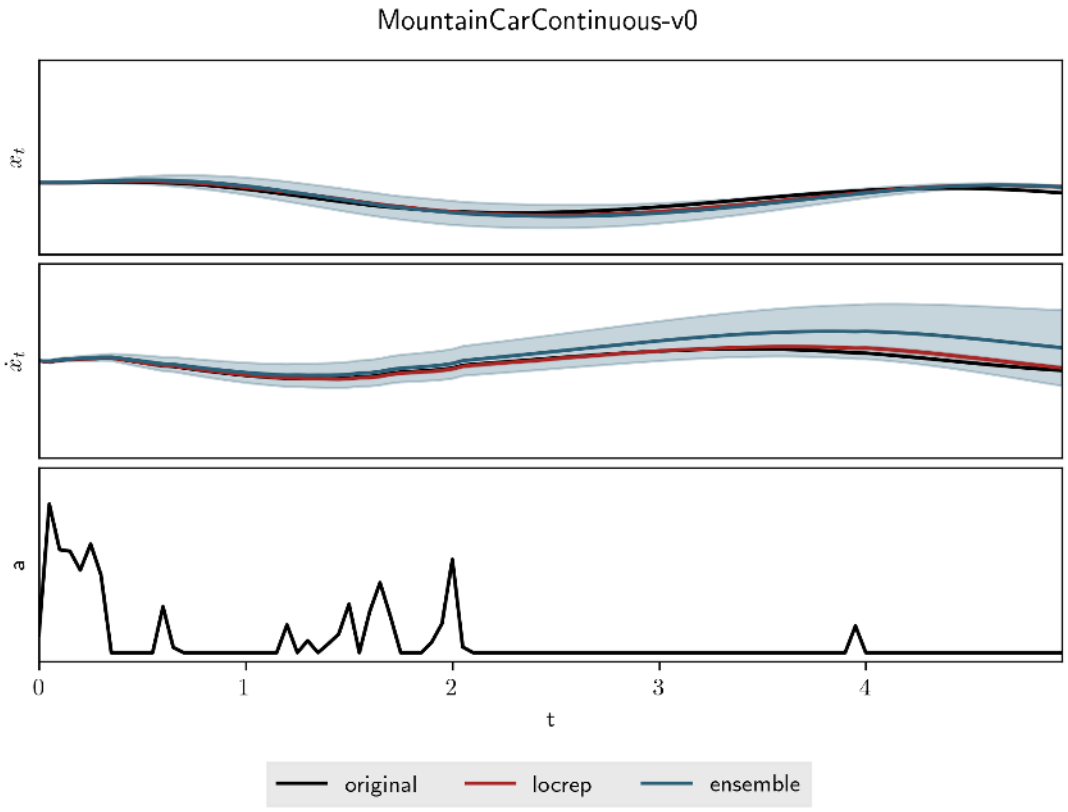


Figure A.3.: Trajectory 0 for MountainCarContinuous-v0

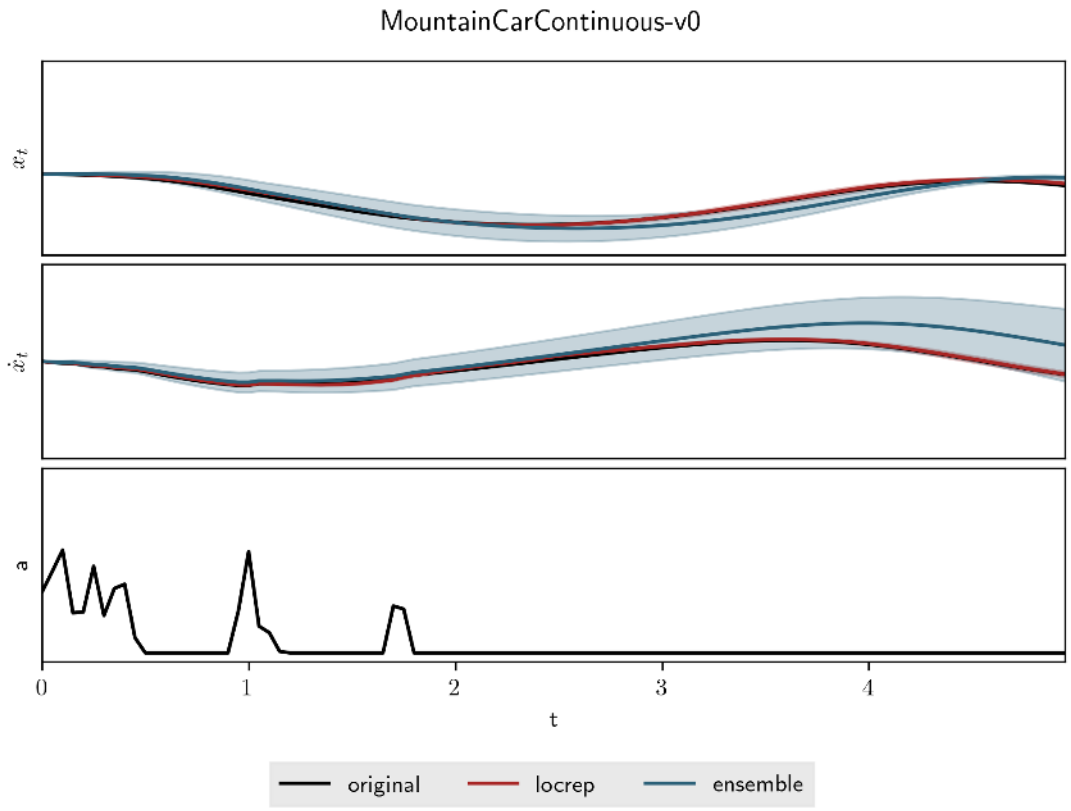


Figure A.4.: Trajectory 1 for MountainCarContinuous-v0

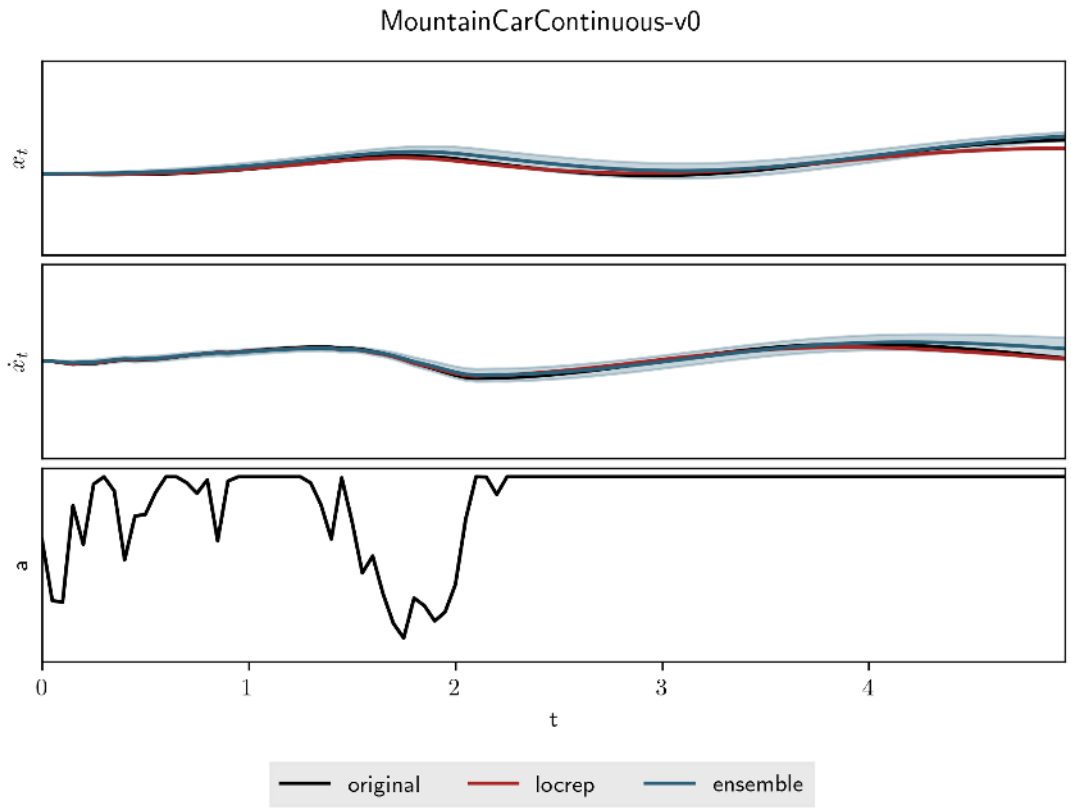


Figure A.5.: Trajectory 4 for MountainCarContinuous-v0

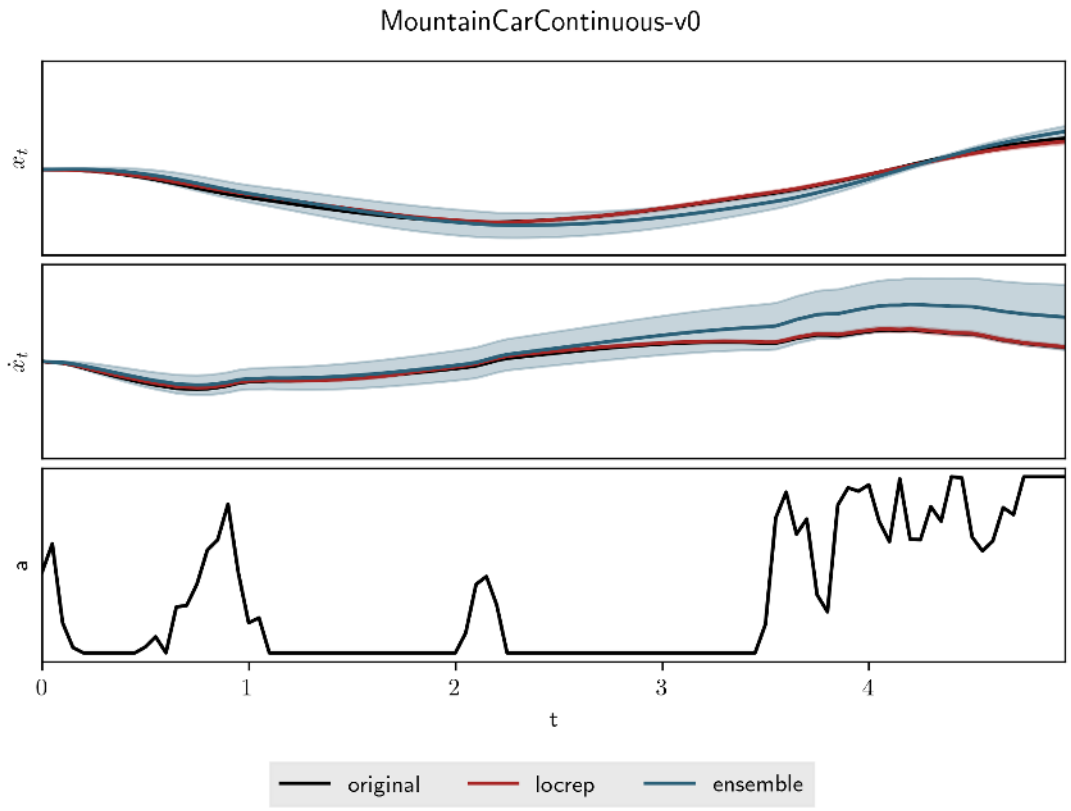


Figure A.6.: Trajectory 7 for MountainCarContinuous-v0

A.1.2.2. CartPole-v0

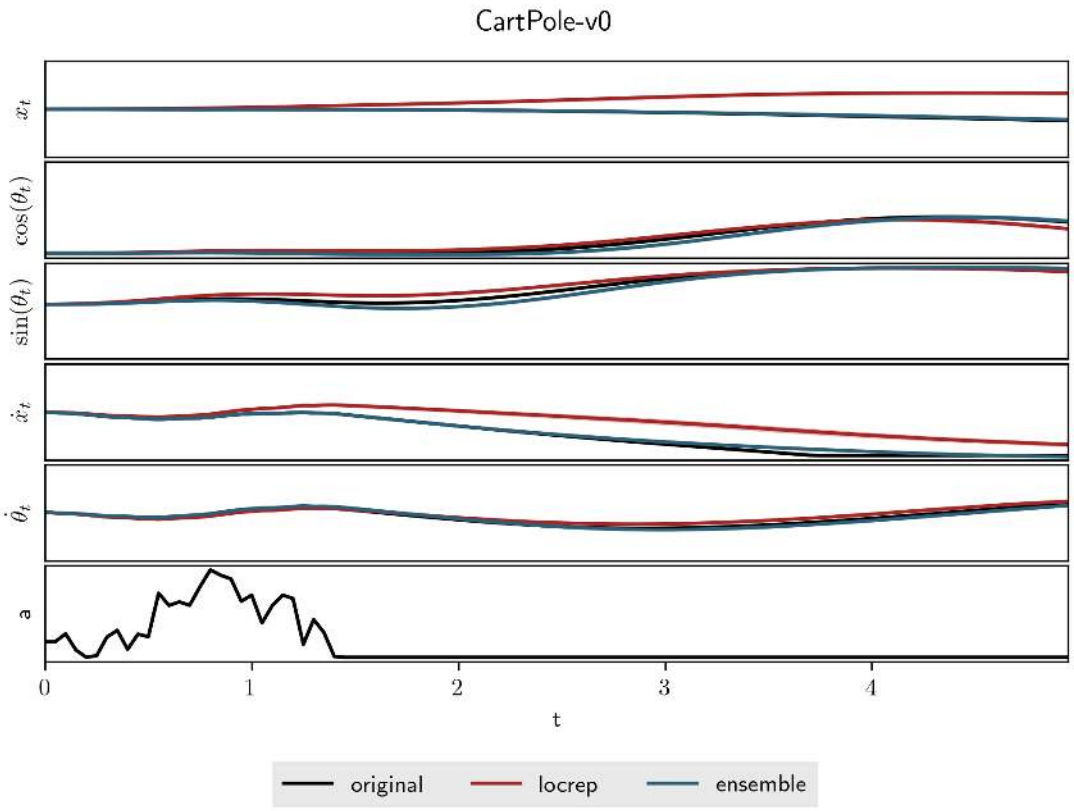


Figure A.7.: Trajectory 0 for CartPole-v0

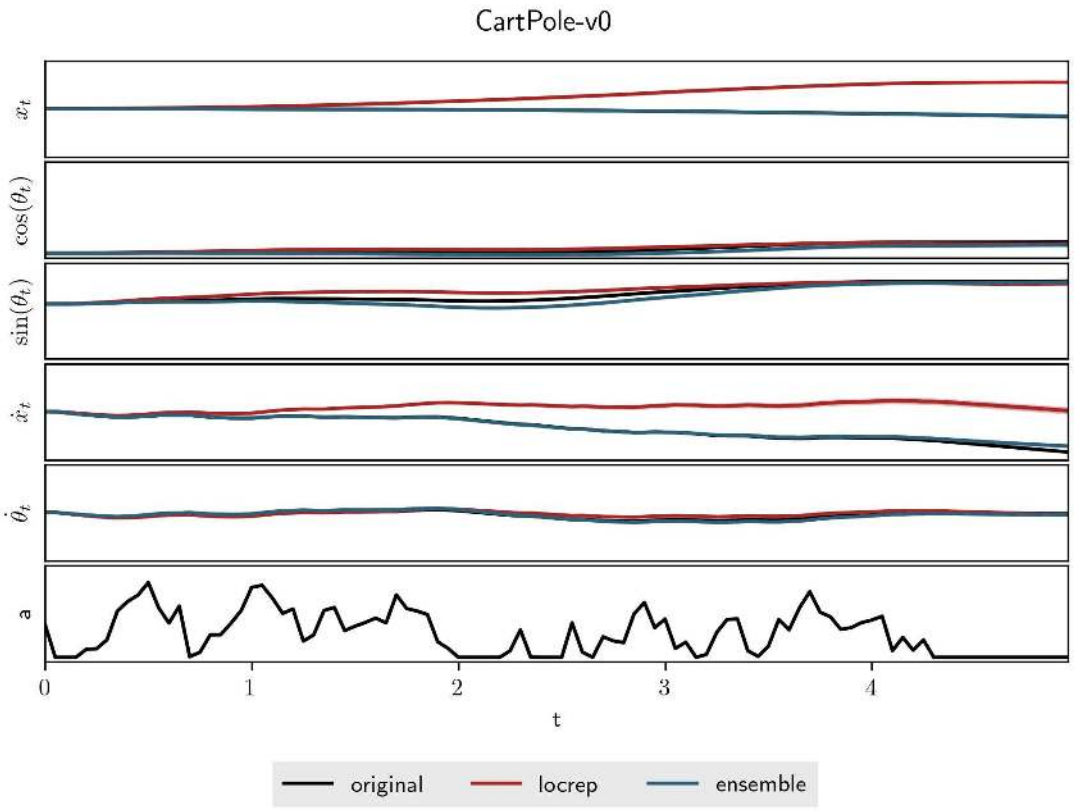


Figure A.8.: Trajectory 1 for CartPole-v0

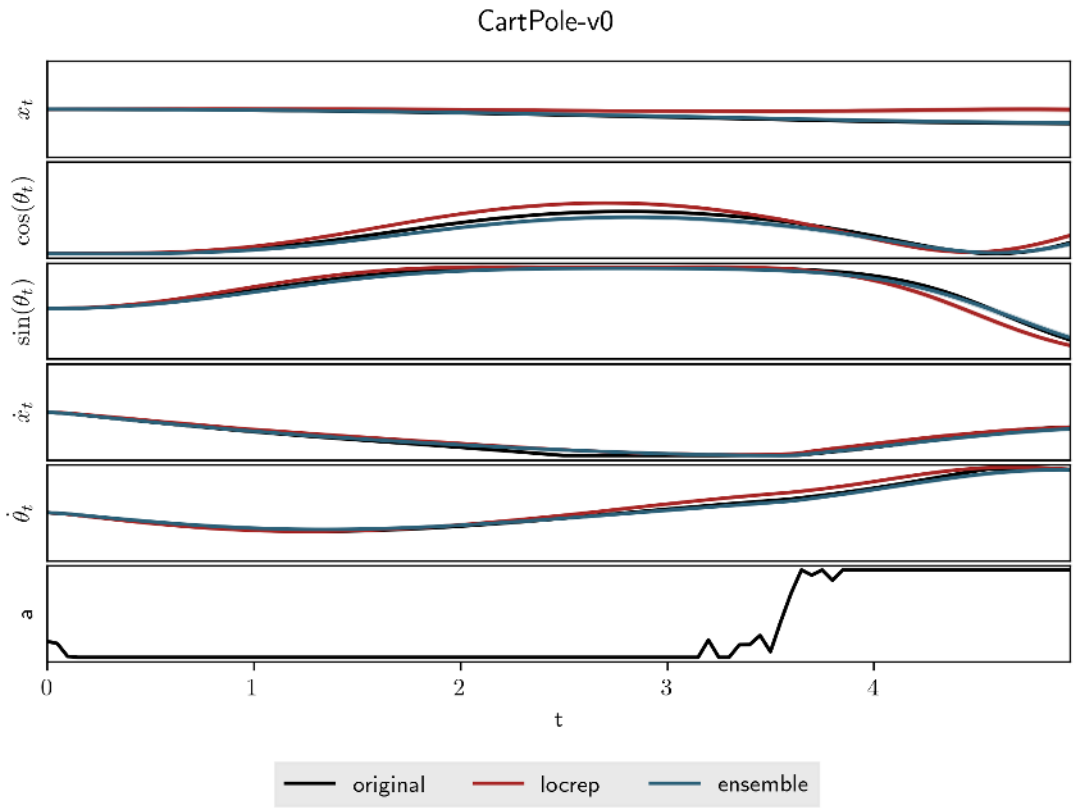


Figure A.9.: Trajectory 4 for CartPole-v0

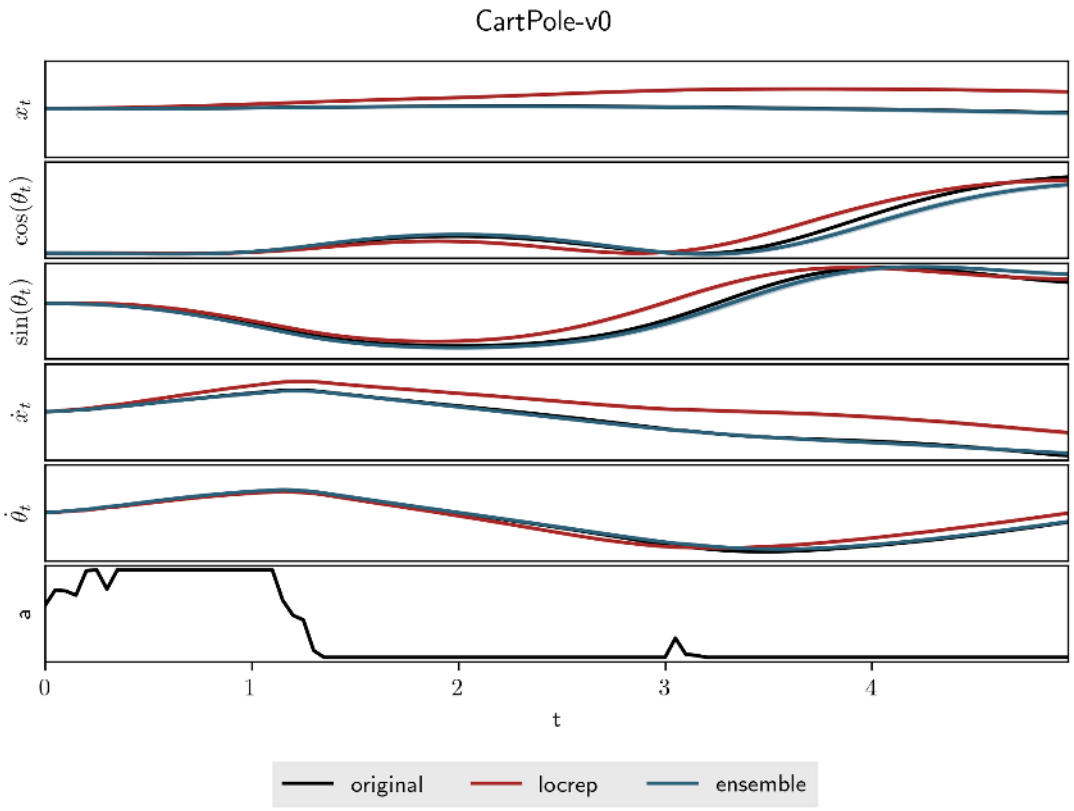


Figure A.10.: Trajectory 7 for CartPole-v0

A.1.2.3. Qube-v0

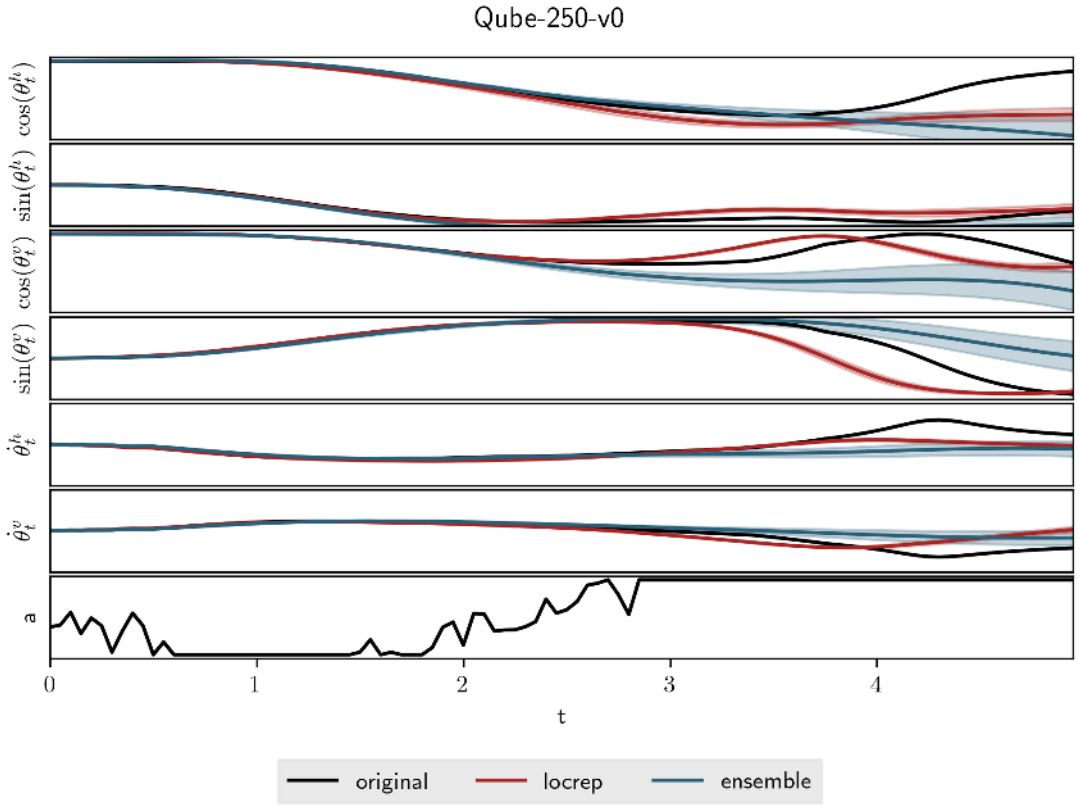


Figure A.11.: Trajectory 0 for Qube-v0

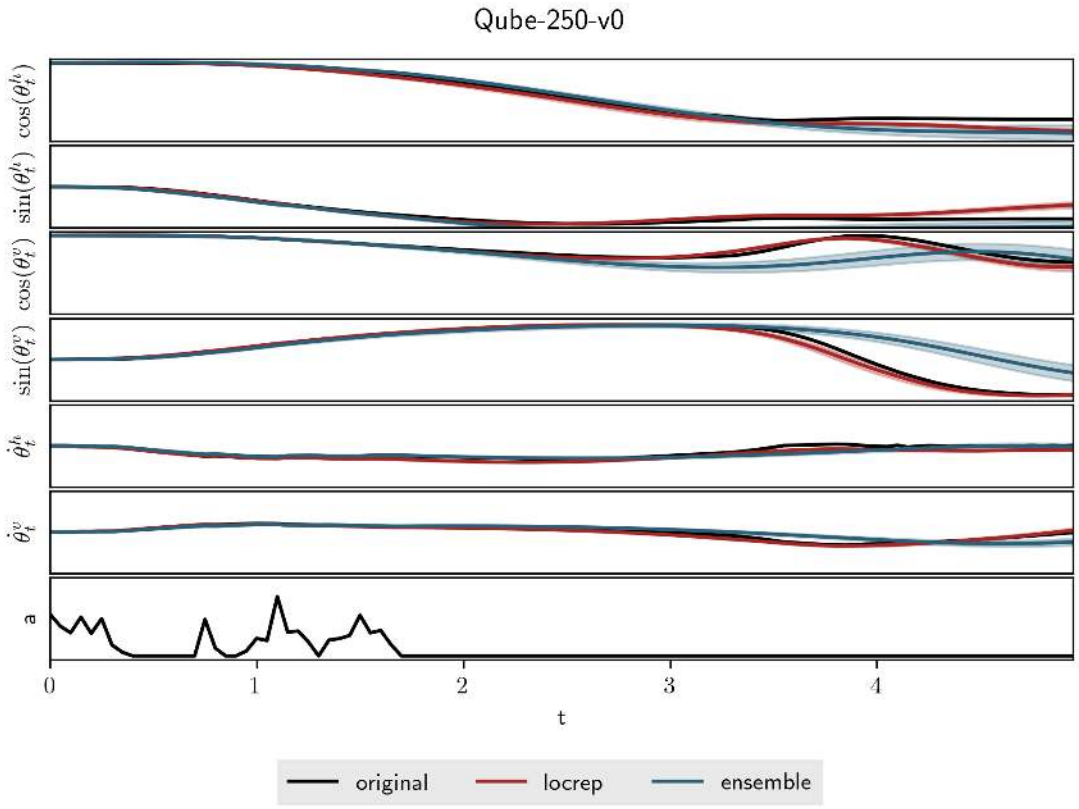


Figure A.12.: Trajectory 1 for Qube-v0

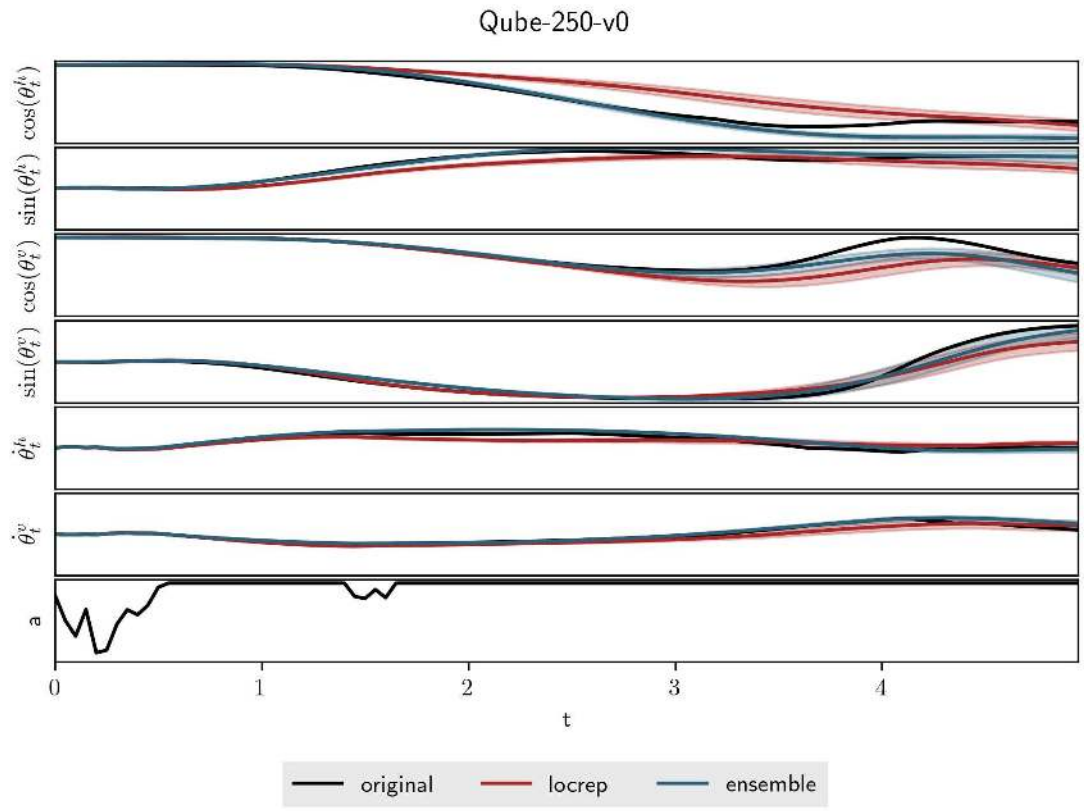


Figure A.13.: Trajectory 4 for Qube-v0

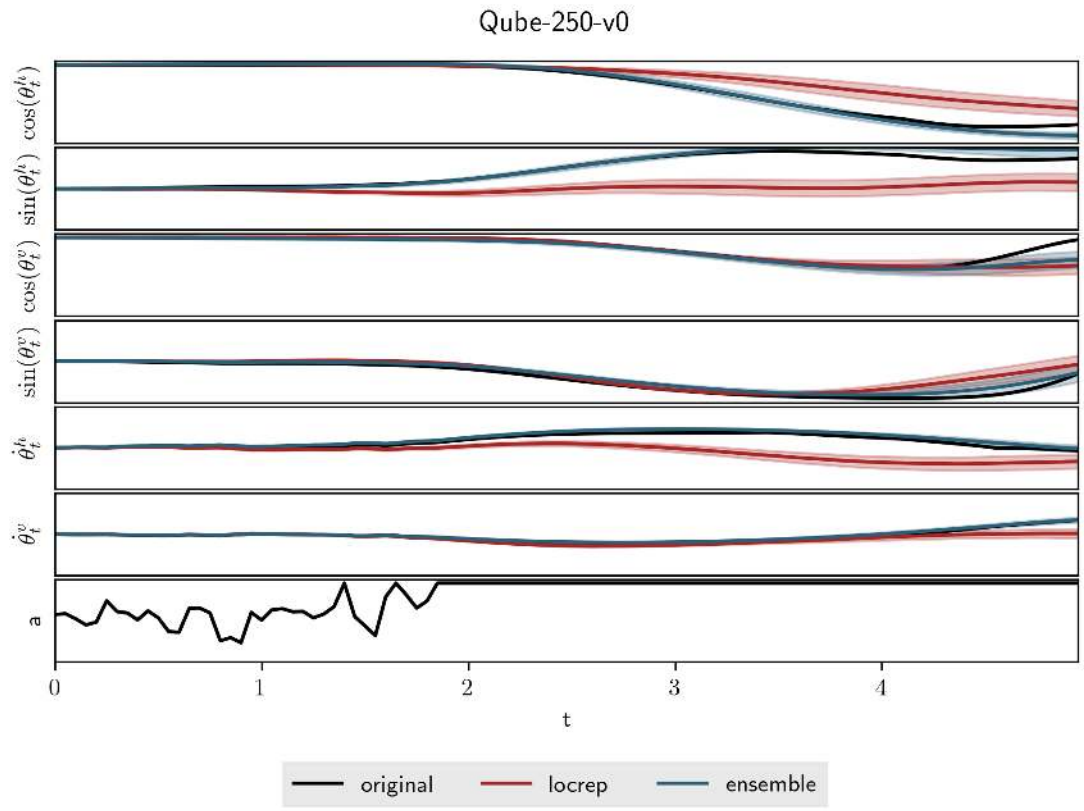


Figure A.14.: Trajectory 7 for Qube-v0

A.2. Active Model Learning

This section contains the graphs from the active learning part. Evaluations of validation trajectories using the predictive model are given.

A.2.1. Training runs

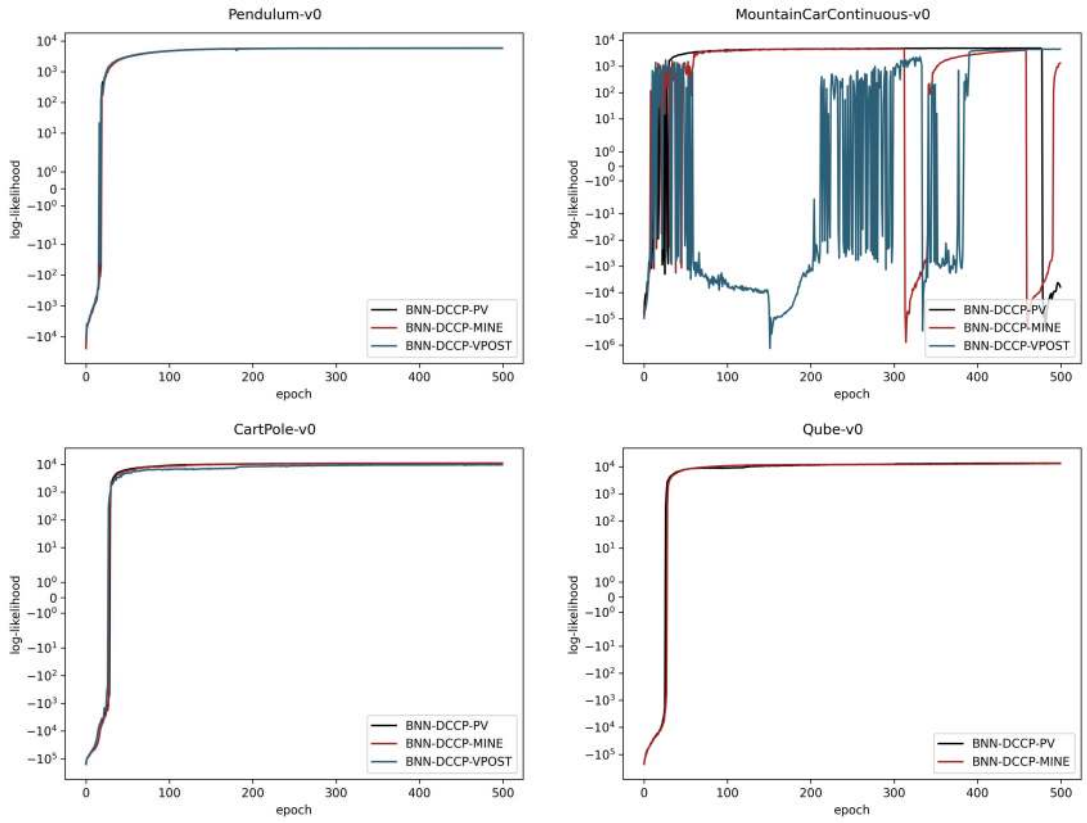


Figure A.15.: BNN - Active learning

A.2.2. Predicted Trajectories

A.2.2.1. MountainCarContinuous-v0

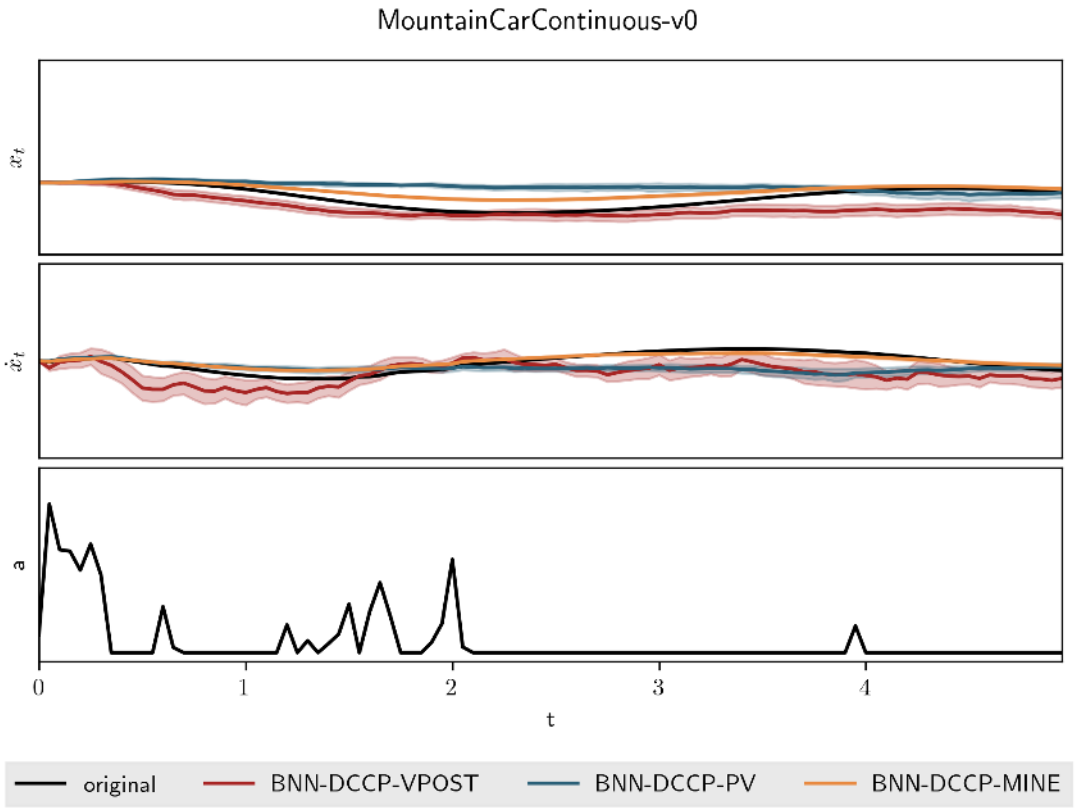


Figure A.16.: Trajectory 0 for MountainCarContinuous-v0

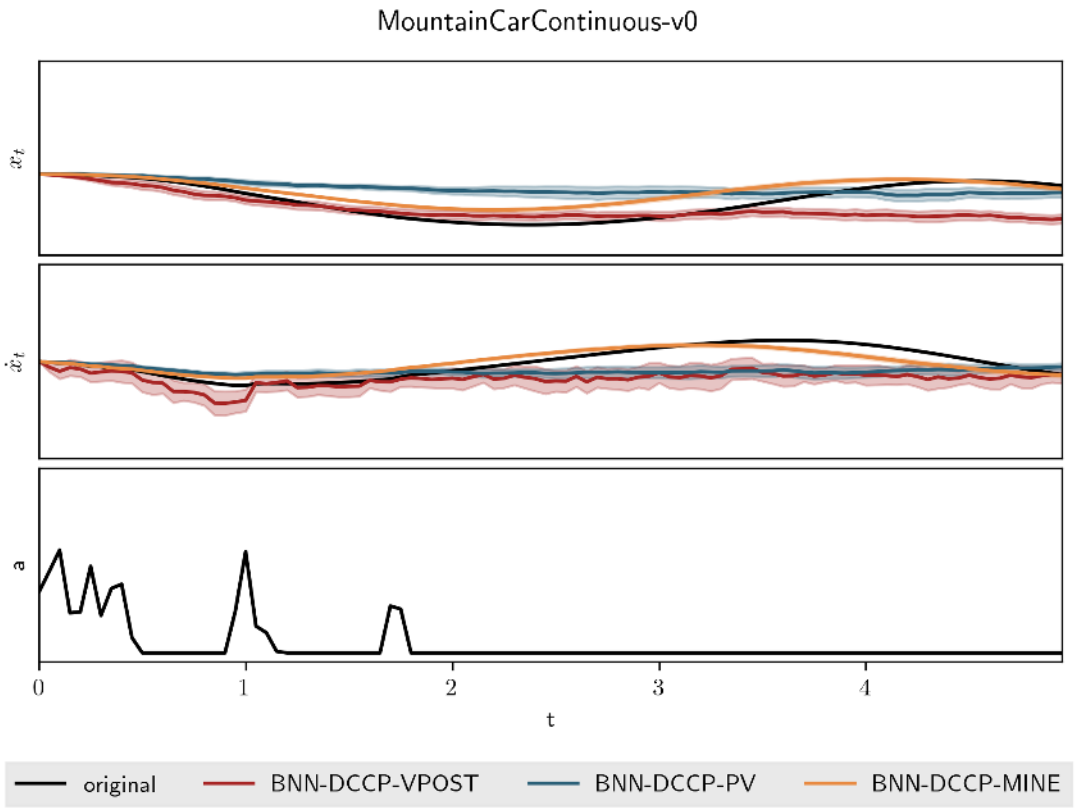


Figure A.17.: Trajectory 1 for MountainCarContinuous-v0

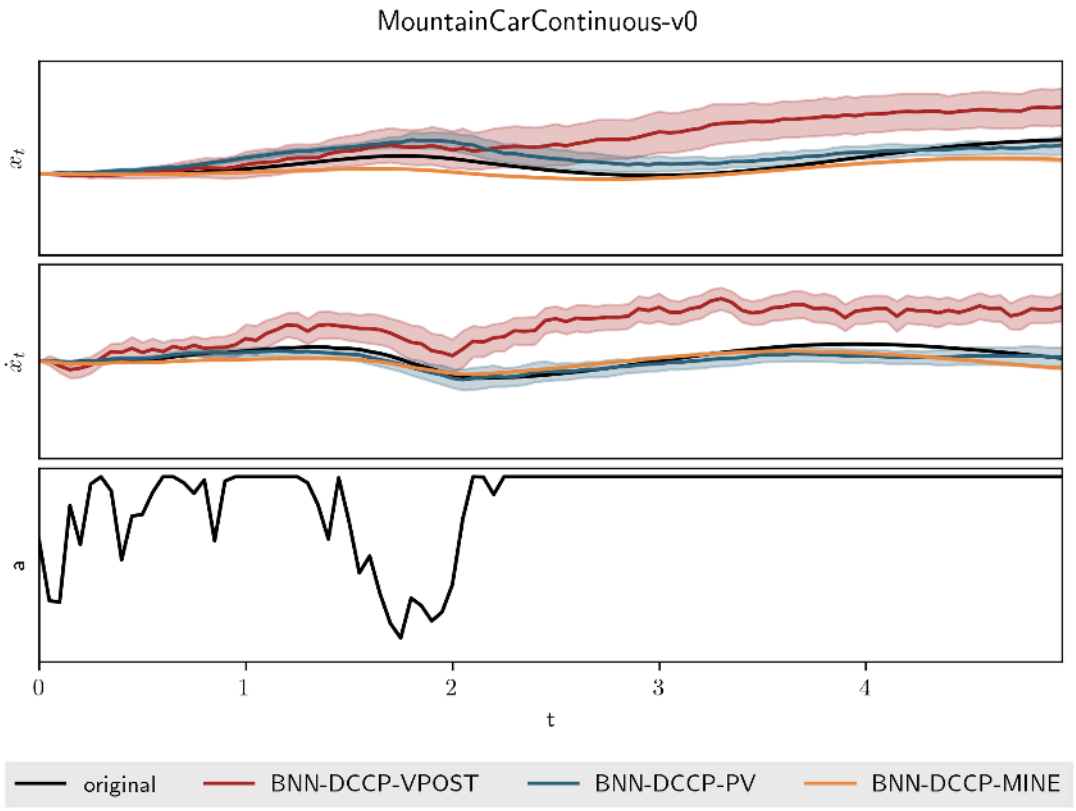


Figure A.18.: Trajectory 4 for MountainCarContinuous-v0

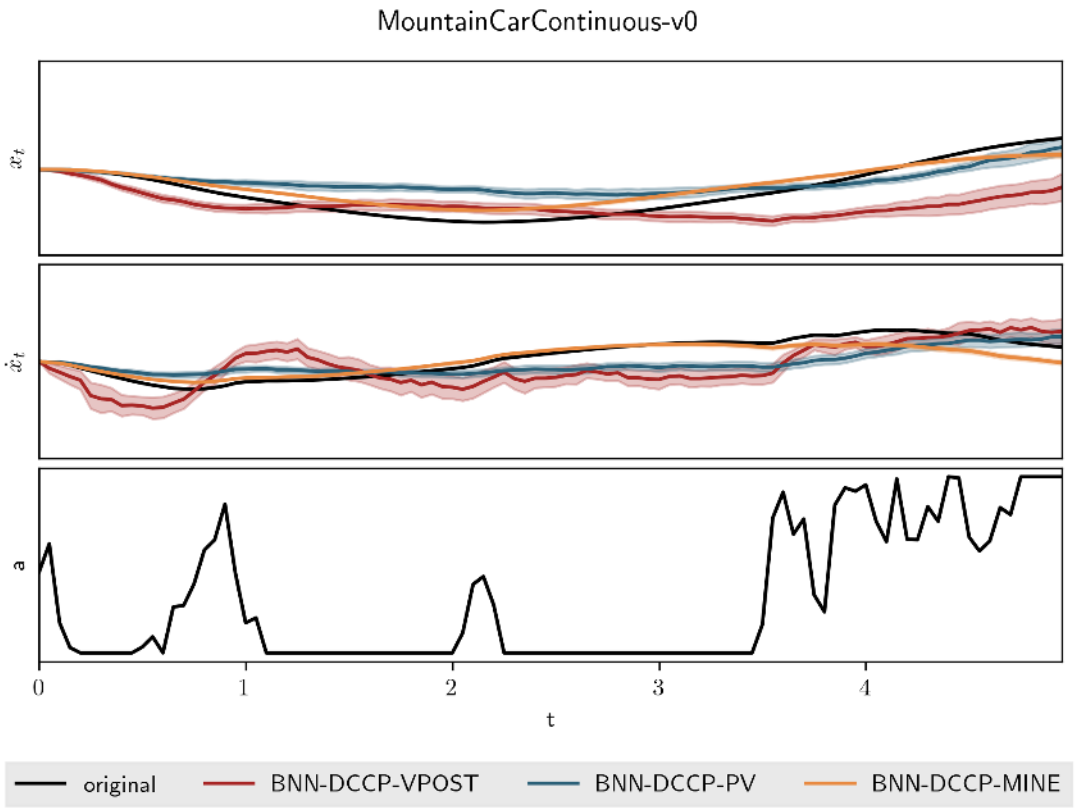


Figure A.19.: Trajectory 7 for MountainCarContinuous-v0

A.2.2.2. CartPole-v0

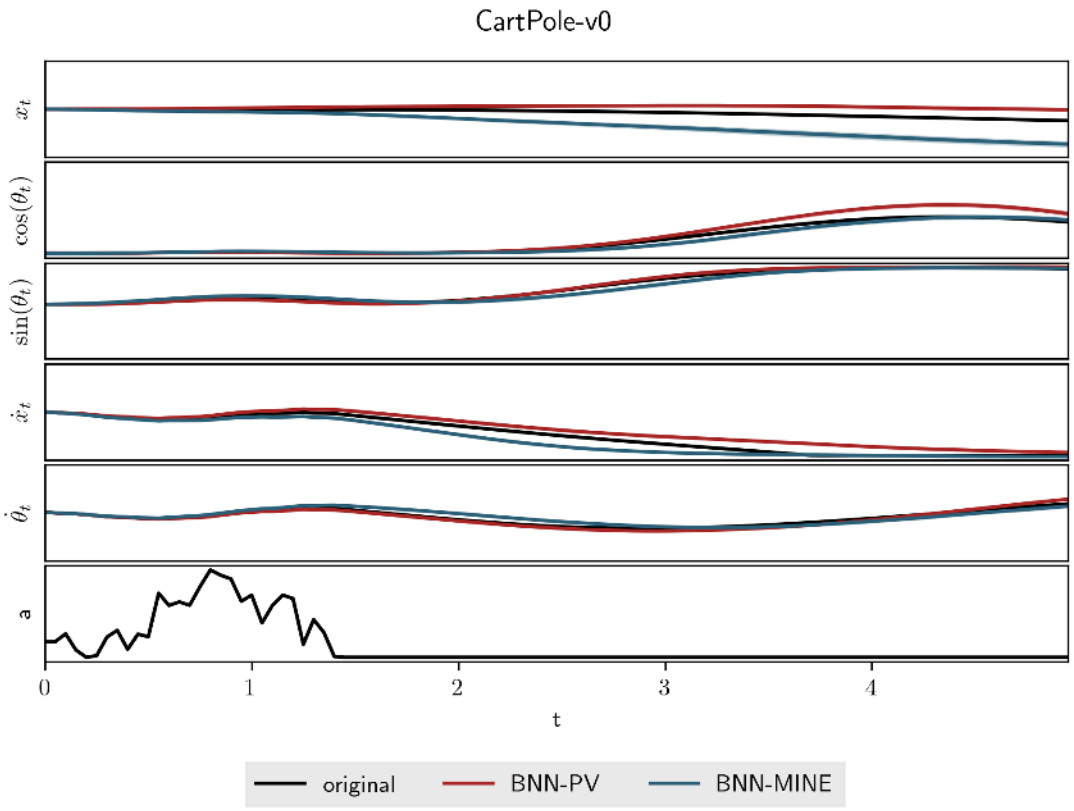


Figure A.20.: Trajectory 0 for CartPole-v0

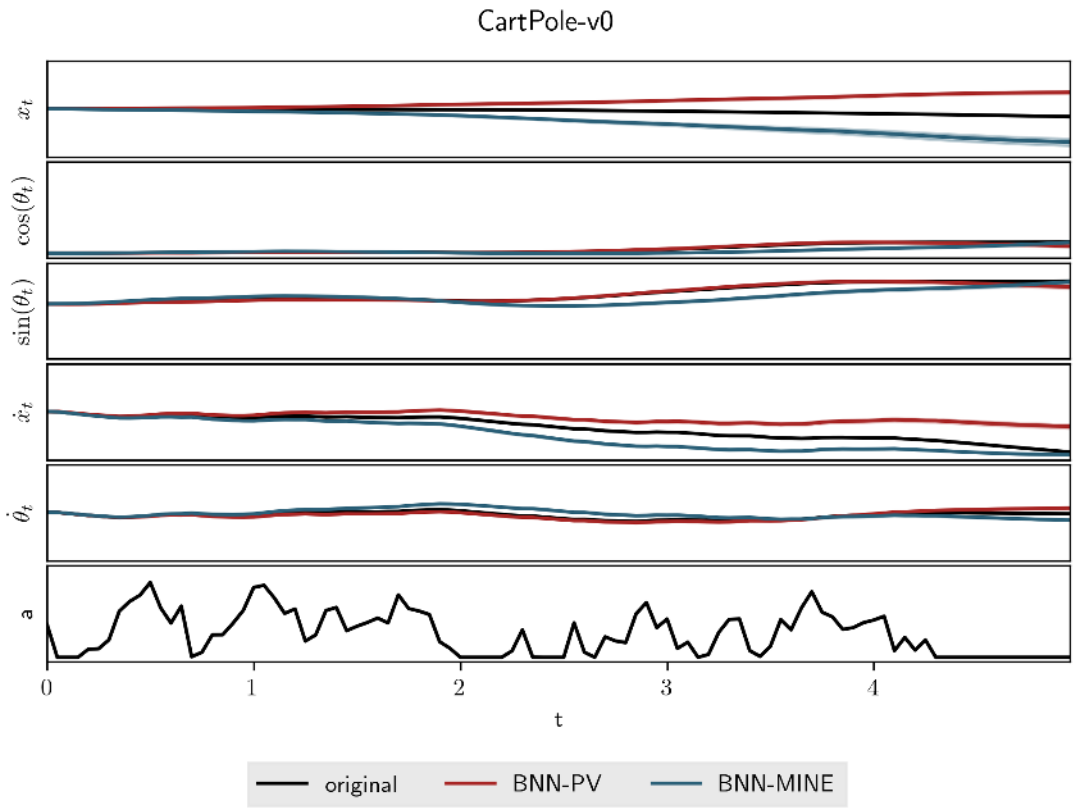


Figure A.21.: Trajectory 1 for CartPole-v0

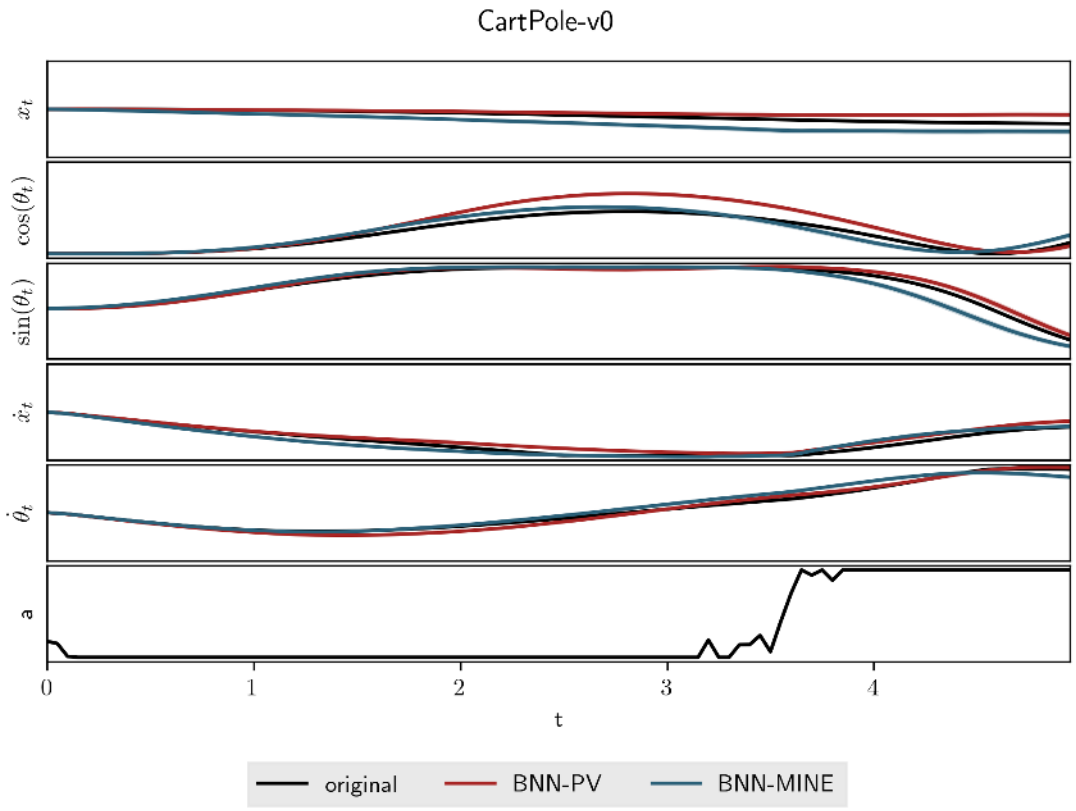


Figure A.22.: Trajectory 4 for CartPole-v0

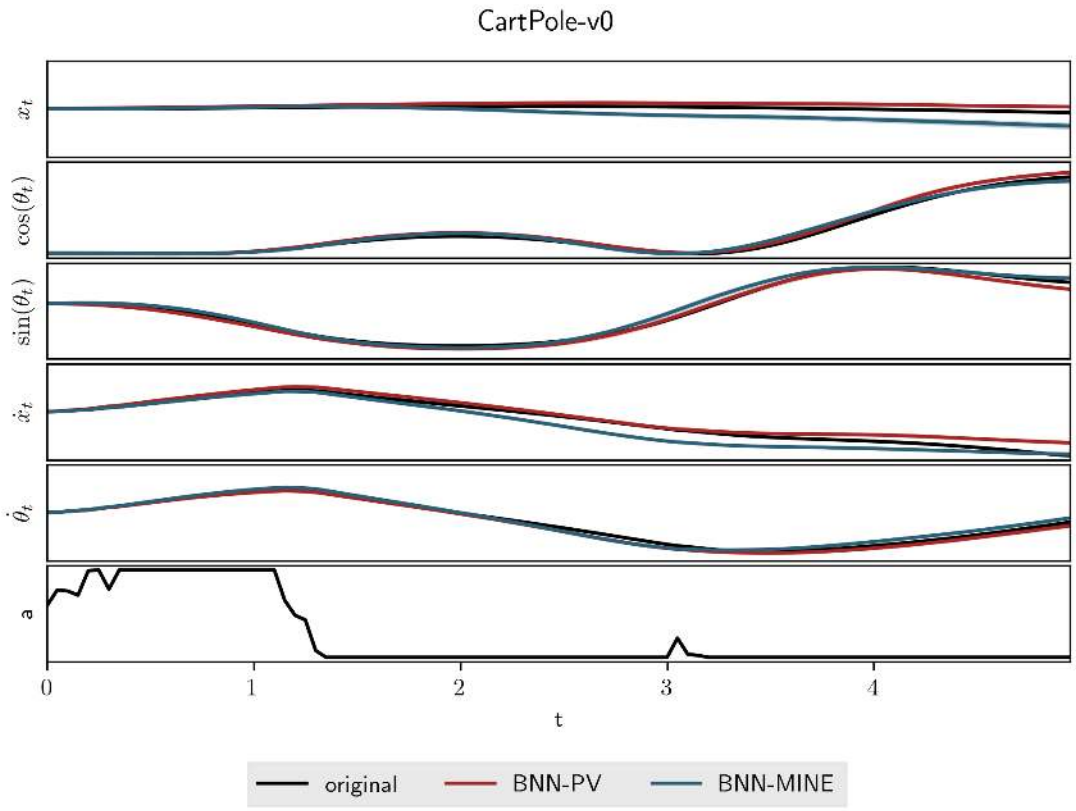


Figure A.23.: Trajectory 7 for CartPole-v0

A.2.2.3. Qube-v0

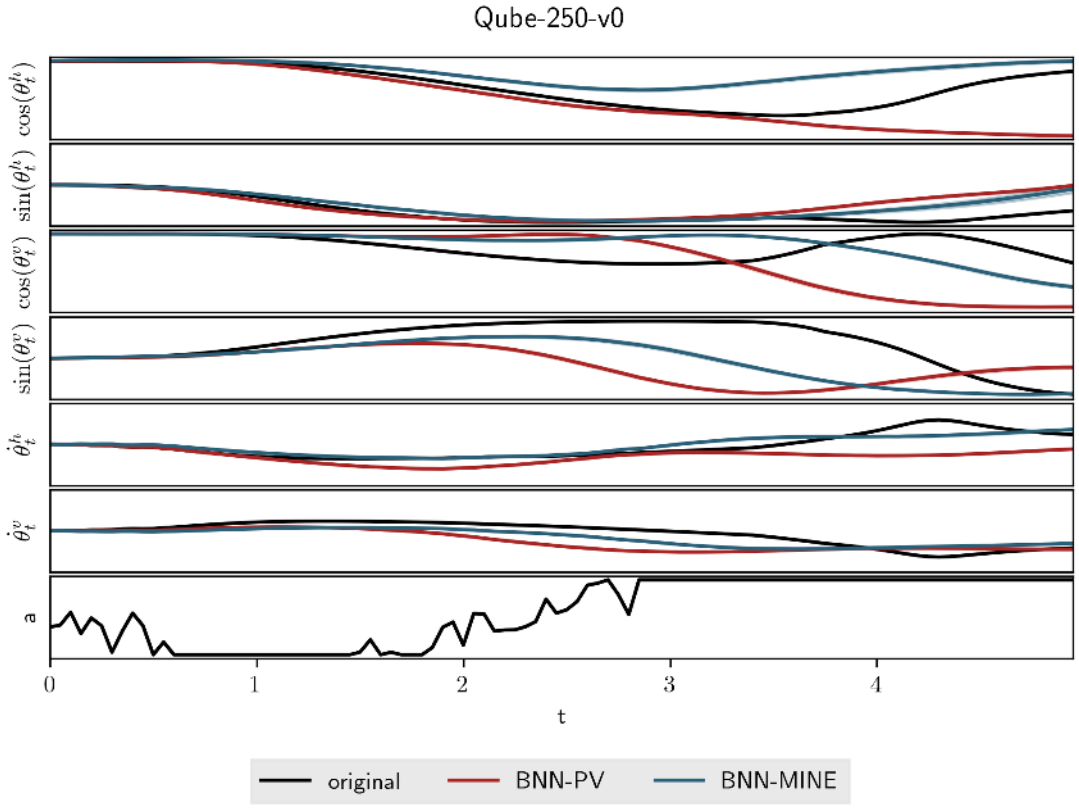


Figure A.24.: Trajectory 0 for Qube-v0

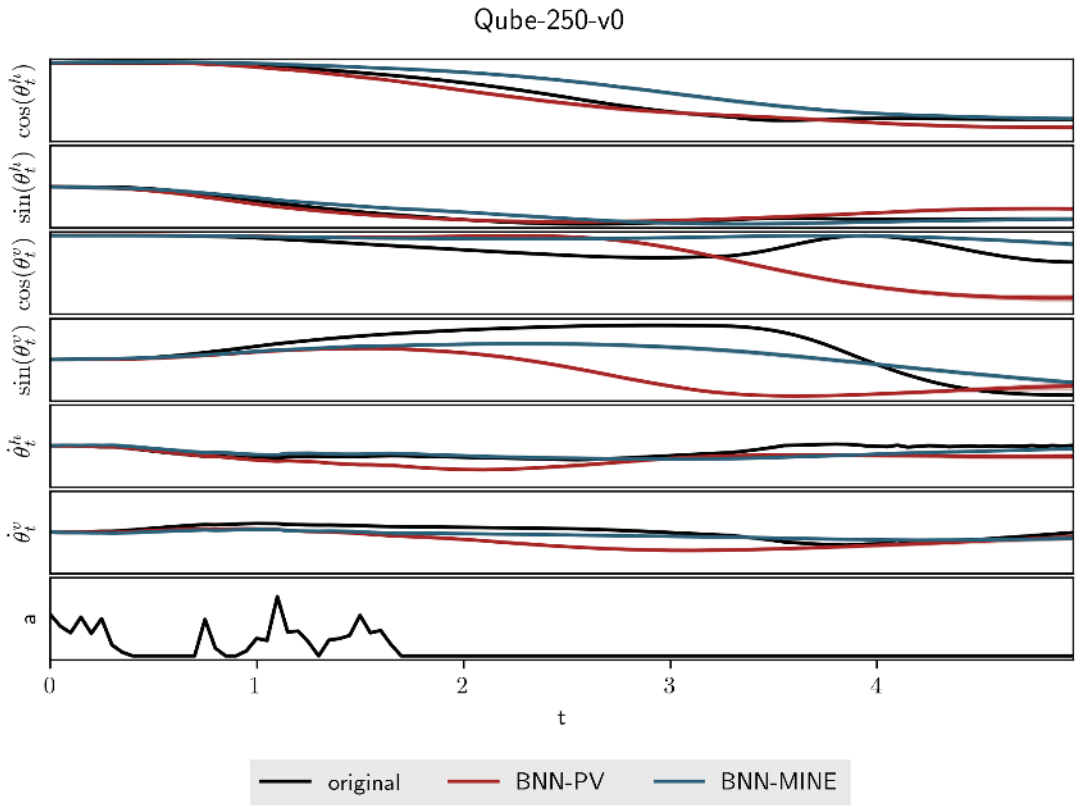


Figure A.25.: Trajectory 1 for Qube-v0

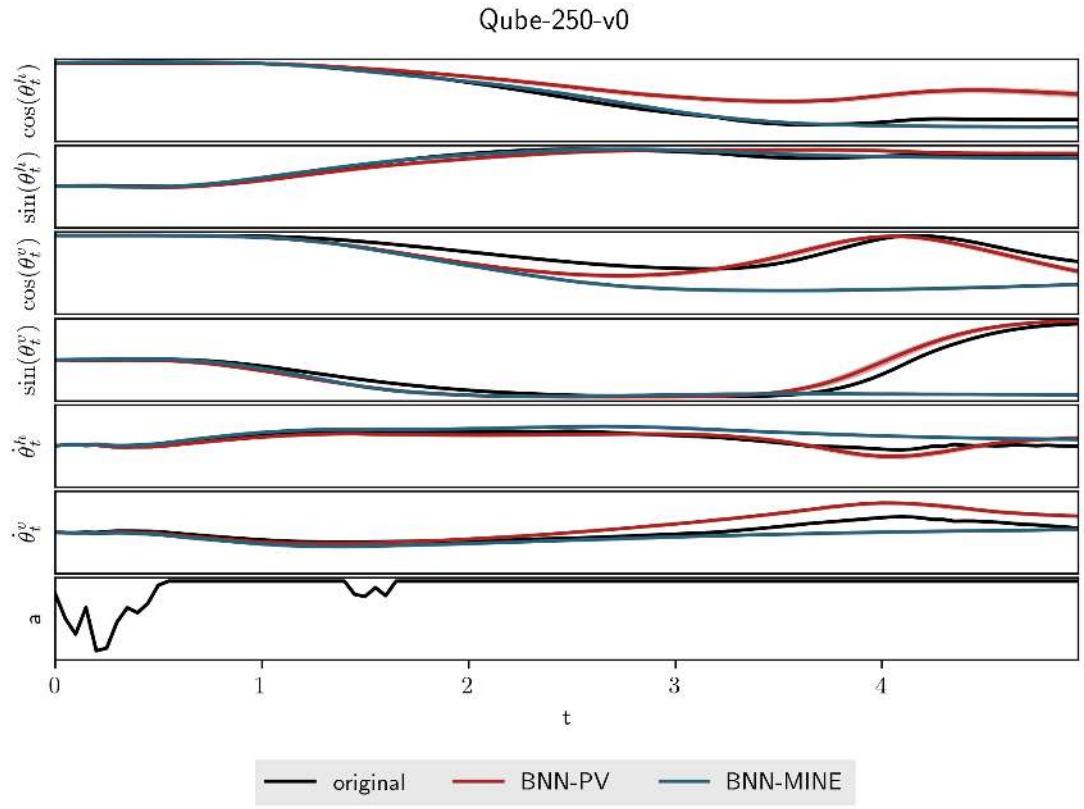


Figure A.26.: Trajectory 4 for Qube-v0

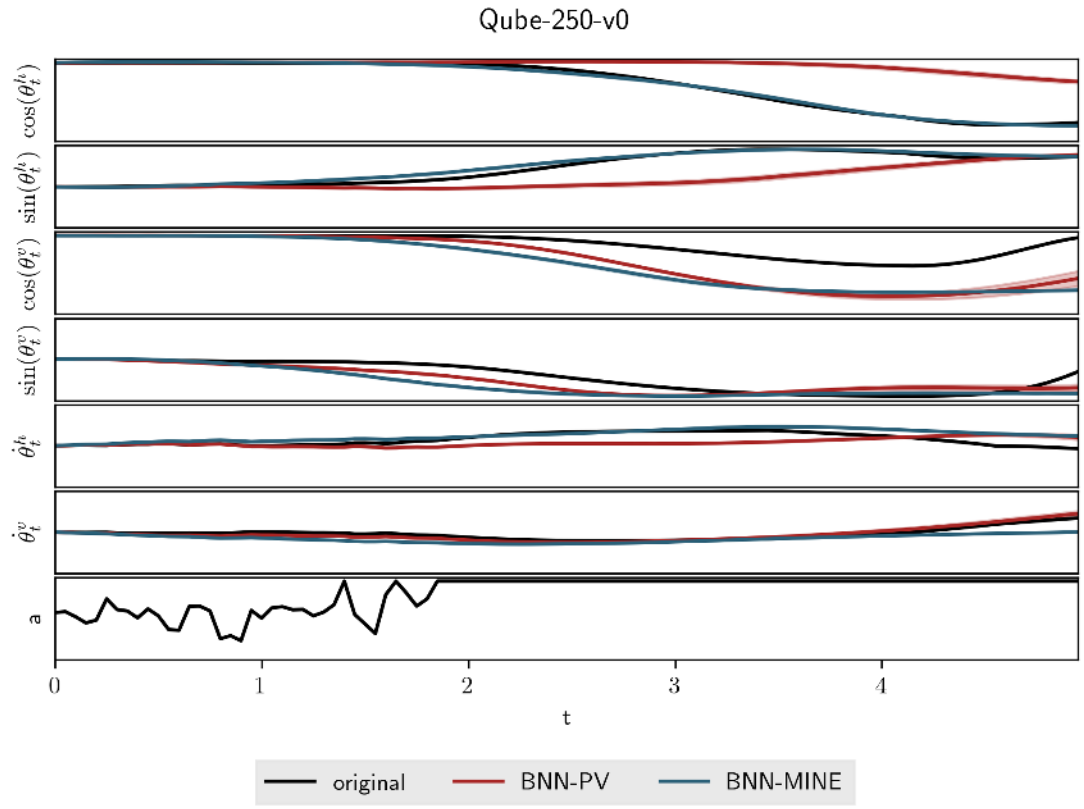


Figure A.27.: Trajectory 7 for Qube-v0