MODEL-BASED MICRO-DATA REINFORCEMENT LEARNING: WHAT ARE THE CRUCIAL MODEL PROPERTIES AND WHICH MODEL TO CHOOSE?

Balázs Kégl, Gabriel Hurtado, Albert Thomas
Noah's Ark Lab, Huawei Paris
{balazs.kegl,gabriel.hurtado,albert.thomas}@huawei.com

ABSTRACT

We contribute to micro-data model-based reinforcement learning (MBRL) by rigorously comparing popular generative models using a fixed (random shooting) control agent. We find that on an environment that requires multimodal posterior predictives, mixture density nets outperform all other models by a large margin. When multimodality is not required, our surprising finding is that we do not need probabilistic posterior predictives: deterministic models may perform optimally but only if they are trained with a probabilistic goal, allowing heteroscedasticity at training time. Our hypothesis is that heteroscedasticity somehow alleviates long-term error accumulation which often hinders the performance of MBRL. At the methodological side, we design metrics and an experimental protocol which can be used to evaluate the various models, predicting their asymptotic performance when using them on the control problem. Using this framework, we improve the state-of-the-art sample complexity of MBRL on Acrobot by two to four folds, using an aggressive training schedule which is outside of the hyperparameter interval usually considered.

1 INTRODUCTION

Unlike computers, physical systems do not get faster with time (Chatzilygeroudis et al., 2020). This is arguably one of the main reasons why recent beautiful advances in deep reinforcement learning (RL) (Silver et al., 2018; Vinyals et al., 2019; Badia et al., 2020) stay mostly in the realm of simulated worlds and do not immediately translate to practical success in the real world. Our long term research agenda is to bring RL to controlling real engineering systems. Our effort is hindered by slow data generation and rigorously controlled access to the systems.

Micro-data RL is the term for using RL on systems where the main bottleneck or source of cost is access to data (as opposed to, for example, computational power). The term was introduced in robotics research (Mouret, 2016; Chatzilygeroudis et al., 2020). This regime requires performance metrics that put as much emphasis on sample complexity (learning speed with respect to sample size) as on asymptotic performance, and algorithms that are designed to make efficient use of small data. Engineering systems are both tightly controlled for safety and security reasons, and physical by nature (so do not get faster with time), making them a primary target of micro-data RL. At the same time, engineering systems are the backbone of today's industrial world: controlling them better may lead to multi-billion dollar savings per year, even if we only consider energy efficiency.[1]

Model-based RL (MBRL) builds predictive models of the system based on historical data (logs, trajectories) referred to here as *traces*. Besides improving the sample complexity of model-free RL by orders of magnitude (Chua et al., 2018), these models can also contribute to adoption from the human side: system engineers can "play" with the models (data-driven generic "neural" simulators) and build trust gradually instead of having to adopt a black-box control algorithm at once (Argenson & Dulac-Arnold, 2020). Engineering systems suit MBRL particularly well in the sense that most system variables that are measured and logged are relevant, either to be fed to classical control or to a human operator. This means that, as opposed to games in which only a few variables (pixels) are relevant for winning, learning a forecasting model in engineering systems for the *full* set of logged

[1] 1% of the yearly energy cost of the US manufacturing sector is roughly a billion dollar [link, link].

variables is arguably an efficient use of predictive power. It also combines well with the micro-data learning principle of using every bit of the data to learn about the system.

Robust and computationally efficient probabilistic generative models are the crux of many machine learning applications. They are especially one of the important bottlenecks in MBRL (Deisenroth & Rasmussen, 2011; Ke et al., 2019; Chatzilygeroudis et al., 2020). System modelling for MBRL is essentially a supervised learning problem with AutoML (models need to be retrained and, if needed, even retuned hundreds of times, on different distributions and data sets whose size may vary by orders of magnitude, with little human supervision), yet there is little prior work on rigorous comparison of system modelling algorithms. Models are often part of a larger system, experiments are slow, and it is hard to know if the limitation or success comes from the model or from the control learning algorithm. System modelling is hard because i) data sets are non-i.i.d., and ii) classical metrics on static data sets may not be predictive of the performance on the dynamic system. There is no canonical data-generating distribution as assumed in the first page of machine learning textbooks, which makes it hard to adopt the classical train/test paradigm. At the same time, predictive system modelling is a great playground and it can be considered as an instantiation of self-supervised learning which some consider the "greatest challenge in ML and AI of the next few years".[2]

We propose to run a comparison of popular probabilistic models on the Acrobot system to study the model properties required to achieve state-of-the-art performances. We believe that such ablation studies are missing from existing "horizontal" benchmarks where the main focus is on state-of-the-art combinations of models and planning strategies (Wang et al., 2019). We start from a family of flexible probabilistic models, autoregressive mixtures learned by deep neural nets (DARMDN) (Bishop, 1994; Uria et al., 2013) and assess the performance of its models when removing autoregressivity, multimodality, and heteroscedasticity. We favor this family of models as it is easy i) to compare them on static data since they come with exact likelihood, ii) to simulate from them, and iii) to incorporate prior knowledge on feature types. Their greatest advantage is modelling flexibility: they can be trained with a loss allowing heteroscedasticity and, unlike Gaussian processes (Deisenroth & Rasmussen, 2011; Deisenroth et al., 2014), deterministic neural nets (Nagabandi et al., 2018; Lee et al., 2019), multivariate Gaussian mixtures (Chua et al., 2018), variational autoencoders (VAE) (Kingma & Welling, 2014; Rezende et al., 2014), and normalizing flows (Rezende & Mohamed, 2015), deep (autoregressive) mixture density nets can naturally and effortlessly represent a multimodal posterior predictive and what we will call $\boldsymbol{y}$*-interdependence* (dependence among system observables even after conditioning on the history).

We chose Acrobot with continuous rewards (Sutton, 1996; Wang et al., 2019) which we could call the "MNIST of MBRL" for three reasons. First, it is simple enough to answer experimental questions rigorously yet it exhibits some properties of more complex environments so we believe that our findings will contribute to solve higher dimensional systems with better sample efficiency as well as better understand the existing state-of-the-art solutions. Second, Acrobot is one of the systems where i) random shooting applied on the real dynamics is state of the art in an experimental sense and ii) random shooting combined with good models is the best approach among MBRL (and even model-free) techniques (Wang et al., 2019). This means that by matching the optimal performance, we essentially "solve" Acrobot with a sample complexity which will be hard to beat. Third, using a single system allows both a deeper and simpler investigation of what might explain the success of popular methods. Although studying scientific hypotheses on a single system is not without precedence (Abbas et al., 2020), we leave open the possibility that our findings are valid only on Acrobot (in which case we definitely need to understand what makes Acrobot special).

There are three complementary explanations why model limitations lead to suboptimal performance in MBRL (compared to model-free RL). First, MBRL learns fast, but it converges to suboptimal models because of the lack of exploration down the line (Schaul et al., 2019; Abbas et al., 2020). We argue that there might be a second reason: the lack of the approximation capacity of these models. The two reasons may be intertwined: not only do we require from the model family to contain the real system dynamics, but we also want it to be able to represent posterior predictive distributions, which i) are consistent with the limited data used to train the model, ii) are consistent with (learnable) physical constraints of the system, and iii) allow efficient exploration. This is not the "classical" notion of approximation, it may not be alleviated by simply adding more capacity to the function representation; it needs to be tackled by properly defining the *output* of the model. Third, models are

[2] https://www.facebook.com/722677142/posts/10155934004262143/

trained to predict the system one step ahead, while the planners need unbiased multi-step predictions which often do not follow from one-step optimality. Our two most important findings nicely comment on these explanations.

- Probabilistic models are needed when the system benefits from multimodal predictive uncertainty. Although the real dynamics might be deterministic, multimodality seems to be crucial to properly handle uncertainty around discrete jumps in the system state that lead to qualitatively different futures.
- When systems do not exhibit such discontinuities, we do not need probabilistic predictions at all: deterministic models are sufficient but only if they are trained with a loss allowing heteroscedasticity. Allowing heteroscedasticity using a likelihood target at *training* time seems to alleviate error accumulation down the horizon (compared to classical regressors trained to minimize the mean squared error one step ahead).

Note that while our hypotheses and experimental findings are related to the grand debate on how to represent and categorize uncertainties (Deisenroth & Rasmussen, 2011; Gal, 2016; Gal et al., 2016; Depeweg et al., 2018; Osband et al., 2018; Hullermeier & Waegeman, 2019; Curi et al., 2020), we remain agnostic about which is the right representation by concentrating on *posterior* predictives on which the different approaches (e.g., Bayesian or not) are directly empirically comparable. We contribute to the debate by providing empirical evidence on a noiseless system, demonstrating unexplained phenomena even when uncertainties are purely epistemic.

We also contribute to good practices in micro-data MBRL by building an extendable experimental protocol in which we design static data sets and measure various metrics which may correlate with the performance of the model on the dynamic system. We instantiate the protocol by a simple setup and study models systematically in a fast experimental loop. When comparing models, the control agent or learning algorithm is part of the scoring mechanism. We fix it to a random shooting model predictive control agent, used successfully by (Nagabandi et al., 2018), for fair comparison and validation of the models.

2 THE FORMAL SETUP

Let $\mathcal{T}_T = \big((\boldsymbol{y}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{y}_T, \boldsymbol{a}_T)\big)$ be a system trace consisting of T steps of observable-action pairs $(\boldsymbol{y}_t, \boldsymbol{a}_t)$: given an observable $\boldsymbol{y}_t$ of the system state at time t, an action $\boldsymbol{a}_t$ was taken, leading to a new system state observed as $\boldsymbol{y}_{t+1}$. The observable vector $\boldsymbol{y}_t = (y_t^1, \ldots, y_t^{d_\mathrm{y}})$ contains d_y numerical or categorical variables, measured on the system at time t. The action vector $\boldsymbol{a}_t$ contains d_a numerical or categorical action variables, typically set by a control function $\boldsymbol{a}_t = \pi(\mathcal{T}_{t-1}, \boldsymbol{y}_t)$ of the history $\mathcal{T}_{t-1}$ and the current observable $\boldsymbol{y}_t$ (or by a stochastic policy $\boldsymbol{a}_t \sim \pi(\mathcal{T}_{t-1}, \boldsymbol{y}_t)$).

The objective of system modelling is to predict $\boldsymbol{y}_{t+1}$ given the system trace $\mathcal{T}_t$. There are applications where point predictions $\hat{\boldsymbol{y}}_{t+1} = f(\mathcal{T}_t)$ are sufficient, however, in most control applications (e.g., reinforcement learning or Bayesian optimization) we need to access the full posterior distribution of $\boldsymbol{y}_{t+1}|\mathcal{T}_t$ to take into consideration the uncertainty of the prediction and/or to model the randomness of the system (Deisenroth & Rasmussen, 2011; Chua et al., 2018). Thus, our goal is to learn $p(\boldsymbol{y}_{t+1}|\mathcal{T}_t)$.

To convert the variable length input (condition) $\mathcal{T}_t = \big((\boldsymbol{y}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{y}_t, \boldsymbol{a}_t)\big)$ into a fixed length state vector $\boldsymbol{s}_t$ we use a fixed feature extractor $\boldsymbol{s}_t = f_{\mathrm{FE}}(\mathcal{T}_t)$. After this step, the modelling simplifies to classical learning of a (conditional) multi-variate density $p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t)$ (albeit on non-i.i.d. data). In the description of our autoregressive models we will use the notation $\boldsymbol{x}_t^1 = \boldsymbol{s}_t$ and $\boldsymbol{x}_t^j = \big(y_{t+1}^1, \ldots, y_{t+1}^{j-1}, \boldsymbol{s}_t\big)$ for $j > 1$ for the input (condition) of the jth autoregressive predictor $p_j(y_{t+1}^j|\boldsymbol{x}_t^j)$ (see Appendix A for more details on the autoregressive setup).

2.1 MODEL REQUIREMENTS

We define seven properties of the model p that are desirable if to be used in MBRL. These restrict and rank the family of density estimation algorithms to consider. Req (R1) is absolutely mandatory for trajectory-sampling controllers, and Req (R2) is mandatory in this paper for using our experimental toolkit to its full extent. Reqs (R3) to (R7) are softer requirements which i) qualitatively indicate the potential performance of generative models in dynamic control, and/or ii) favor practical usability on real engineering systems and benchmarks. Table 1 provides a summary on how the different models

satisfy (or not) these requirements. We note that depending on the application and the desired control frequency of the system, one may also require models with fast prediction times.

(R1) It should be computationally easy to properly **simulate observables** $\boldsymbol{Y}_{t+1} \sim p(\cdot|\mathcal{T}_t)$ given the system trace to interface with popular control techniques that require such simulations. Note that it is then easy to obtain random traces of arbitrary length from the model by applying p and π alternately.

(R2) Given $\boldsymbol{y}_{t+1}$ and $\mathcal{T}_t$, it should be computationally easy to evaluate $p(\boldsymbol{y}_{t+1}|\mathcal{T}_t)$ to obtain a **likelihood score** in order to compare models on various traces. This means that $p(\boldsymbol{y}|\mathcal{T}_t) > 0$ and $\int p(\boldsymbol{y}|\mathcal{T}_t)\mathrm{d}\boldsymbol{y} = 1$ should be assured by the representation of p, without having to go through sampling, approximation, or numerical integration.

(R3) We should be able to model $\boldsymbol{y}$**-interdependence**: dependence among the d_{y} elements of $\boldsymbol{y}_{t+1} = (y^1_{t+1}, \ldots, y^{d_{\mathrm{y}}}_{t+1})$ given $\mathcal{T}_t$. In our experiments we found that the MBRL performance was not affected by the lack of this property, however, we favor it since the violation of strong physical constraints in telecommunication or robotics may hinder the acceptance of the models (simulators) by system engineers. See Appendix B for further explanation.

(R4) **Heteroscedastic** models are able to vary their uncertainty estimate as a function of the state or trace $\mathcal{T}_t$. Abbas et al. (2020) show how to use input-dependent variance to improve the planning. We found that even when using the deterministic prediction at planning time, allowing heteroscedasticity at *training* time alleviates error accumulation down the horizon.

(R5) Allowing **multi-modal posterior predictives** seems to be crucial to properly handle uncertainty around discrete jumps in the system state that lead to qualitatively different futures.

(R6) We should be able to **model different observable types**, for example discrete/continuous, finite/infinite support, positive, heavy tail, multimodal, etc. Engineers often have strong prior knowledge on distributions that should be used in the modelling, and the popular (multivariate) Gaussian assumption often leads to suboptimal approximation.

(R7) Complex multivariate density estimators rarely work out of the box on a new system. We are aiming at reusability of our models (not simple reproducibility of our experimental results). In the system modelling context, density estimators need to be retrained and retuned automatically. Both of these require **robustness and debuggability**: self-tuning and gray-box models and tools that can help the modeler to pinpoint where and why the model fails. This requirement is similar to what is often imposed on supervised models by application constraints, for example, in health care (Caruana et al., 2015).

2.2 Evaluation metrics

We define a set of metrics to compare system models both on fixed static traces $\mathcal{T}$ (Section 2.2.1) and on dynamic systems (Section 2.2.2). We have a triple aim. First, we contribute to moving the community towards a supervised-learning-like rigorous evaluation process where claims can be made more precise. Second, we define an experimental process where models can be evaluated rapidly using static metrics before having to run long experiments on the dynamic systems. Our methodological goal is to identify static metrics that predict the performance of the models on the dynamic system. Third, we provide diagnostics tools to the practical modeller to debug the models and define triggers and alarms when something goes wrong on the dynamical system (e.g., individual outliers, low probability traces).

2.2.1 Static metrics

We use four metrics on our static "supervised" experiment to assess the models $p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t)$. We define all metrics formally in Appendix C. First we compute the (average) log-likelihood of p on a test trace $\mathcal{T}_T$ for those models that satisfy Req (R2). Log-likelihood is a unitless metrics which is hard to interpret and depends on the unit in which its input is measured. To have a better interpretation, we normalize the likelihood with a baseline likelihood of a multivariate independent unconditional Gaussian, to obtain the **likelihood ratio (LR)** metrics. LR is between 0 (although LR < 1 usually indicates a bug) and ∞, the higher the better. We found that LR works well in an i.i.d. setup but distribution shift often causes "misses": test points with extremely low likelihood. Since these points dominate LR, we decided to clamp the likelihood and compute the rate of test points with a likelihood

less than[3] $p_{\min} = 1.47 \times 10^{-6}$. This **outlier rate (OR)** measures the "surprise" of a model on trace $\mathcal{T}$. OR is between 0 and 1, the lower the better. Third, we compute the **explained variance (R2)**. We prefer using this metrics over the MSE because it is normalized so it can be aggregated over the dimensions of $\boldsymbol{y}$. R2 is between 0 and 1, the higher the better. Fourth, for models that provide marginal CDFs, we compute the **Kolmogorov-Smirnov (KS)** statistics between the uniform distribution and the quantiles of the test ground truth (under the model CDFs). Well-calibrated models have been shown to improve the performance of MBRL algorithms (Malik et al., 2019). KS is between 0 and 1, the lower the better.

All our density estimators are trained to predict the system one step ahead yet arguably what matters is their performance at a longer horizon L specified by the control agent. Our models do not provide explicit likelihoods L steps ahead, but we can simulate from them (following ground truth actions) and evaluate the metrics by a Monte-Carlo estimate, obtaining **long horizon metrics KS(L) and R2(L)**. In all our experiments we use $L = 10$ with 100 Monte Carlo traces, and, for computational reasons, sample the test set at 100 random positions, which explains the high variance on these scores.

All six metrics (LR, OR, R2, KS, R2(10), KS(10)) are averaged over the dimensions to obtain single scores for the environment/model pair, but for debugging we can also evaluate them dimension-wise. LR is the "master" score that combines precision (R2) and calibratedness (KS). R2 is a good single measure to assess the models, especially when iterated to obtain R2(L). OR and KS are excellent debugging tools. The single-target KS and quantile plots are especially useful to spot *how* the models are miscalibrated: e.g., points accumulating in the middle indicate that we overestimate the tails, leading to nonphysical simulations, and vice versa, accumulation at the edges means our model is missing modes. OR is great to detect catastrophic failures or distribution shifts, so monitoring it on the deployed system is crucial. Finally, correlating these metrics to the dynamic performance (Section 2.2.2) for the given system can form the basis of a comprehensive monitoring system which is as important as model performance in practice.

2.2.2 Dynamic metrics

Our ultimate goal is to develop good models for MBRL so we also measure model quality in terms of the final performance. For this, we fix the control algorithm to random shooting (RS) (Richards, 2005; Rao, 2010) which performs well on the true dynamics of Acrobot as well as many other systems (Wang et al., 2019). RS consists in a random search of the action sequence maximizing the expected cumulative reward over a fixed planning horizon L. The agent then applies the first action of the best action sequence. We use $L = 10$ and generate $n = 100$ random action sequences for the random search. For stochastic models we average the cumulative rewards of 5 random trajectories obtained for a same action sequence. We note that one could achieve better results by using a larger n or the cross entropy method (CEM) (de Boer et al., 2004; Chua et al., 2018). One could also consider more complex planning strategies (Wang & Ba, 2020; Argenson & Dulac-Arnold, 2020). However we judge RS with $n = 100$ to be sufficient for our study (see Appendix D for more details). We present here the MBRL loop and notations which will be needed to define the dynamic metrics.

1. Run random policy $\pi^{(1)}$ for $T = 200$ steps, starting from an initial "seed" trace $\mathcal{T}_{T_0}^{(0)}$ (typically a single-step state $\mathcal{T}_1^{(0)} = (\boldsymbol{y}_0, \cdot)$) to obtain a random initial trace $\mathcal{T}_T^{(1)}$. Let the epoch index be $\tau = 1$.
2. Learn $p^{(\tau)}$ on the full trace $\mathcal{T}_{\tau \times T} = \cup_{\tau'=1}^{\tau} \mathcal{T}_T^{(\tau')}$.
3. Run RS policy $\pi^{(\tau)}$ using model $p^{(\tau)}$, (re)starting from $\mathcal{T}_{T_0}^{(0)}$, to obtain trace $\mathcal{T}_T^{(\tau+1)}$.
4. If $\tau < N$, let $\tau = \tau + 1$ and go to Step 2, otherwise stop.

Given the formal algorithm, we can now elaborate what we mean by system modelling for MBRL being essentially a supervised learning problem with AutoML (and why (R7) is important). In Step 2, the chosen model needs to be retrained and, if needed, retuned, on data sets $\mathcal{T}_{\tau \times T}$ of different distribution whose size may vary by orders of magnitude, with little human supervision. This does not mean we need to do full hyperopt in every episode τ, rather that $p^{(\tau)}$ should be robust: trainable without human babysitting over a range of different distributions and data sizes. A single catastrophic learning failure (e.g. getting stuck in initial random function) means the full MBRL loop goes off

[3] As a salute to 5-sigma, using the analogy of the MBRL loop (Section 2.2.2) as the iterated scientific method.

the rail. Models that need to be retuned (because of sensitivity to hyperparameters) must have the retuning (AutoML) feature encapsulated into their training. The models that ended up on the top were not sensitive to the choice of hyperparameters, so we did not need to retune them in every iteration.

MEAN ASYMPTOTIC REWARD (MAR) AND RELATIVE MAR (RMAR). Given a trace $\mathcal{T}_T$ and a reward r_t obtained at each step t, we define the mean reward as $\mathrm{R}(\mathcal{T}_T) = \frac{1}{T}\sum_{t=1}^{T} r_t$.[4] The mean reward in iteration τ is then $\mathrm{MR}(\tau) = \mathrm{R}\left(\mathcal{T}_T^{(\tau)}\right)$. Our measure of asymptotic performance, the mean asymptotic reward, is the mean reward in the second half of the epochs (after convergence; we set N in such a way that the algorithms converge after less than $N/2$ epochs) $\mathrm{MAR} = \frac{2}{N}\sum_{\tau=N/2}^{N} \mathrm{MR}(\tau)$. To normalize across systems and to make the measure independent of the control algorithm we use on top of the model, we define the relative mean asymptotic reward $\mathrm{RMAR} = \mathrm{MAR}/\mathrm{MAR}_{\mathrm{opt}}$, where $\mathrm{MAR}_{\mathrm{opt}}$ is the mean asymptotic reward obtained by running the same control algorithm on the true dynamics ($\mathrm{MAR}_{\mathrm{opt}} = 2.104$ in our experiments; see Table 5 in Appendix D for more discussion on this). This puts RMAR between 0 and 1 (the higher the better).

MEAN REWARD CONVERGENCE PACE (MRCP). To assess the speed of convergence, we define the mean reward convergence pace (MRCP) as the number of steps needed to achieve 70% of $\mathrm{MAR}_{\mathrm{opt}}$ using the running average of 5 epochs $\mathrm{MRCP} = T \times \arg\min_{\tau}\left(\frac{1}{5}\sum_{\tau'=\tau-2}^{\tau+2} \mathrm{MR}(\tau) > 0.7 \times \mathrm{MAR}_{\mathrm{opt}}\right)$. The unit of MRCP is system access steps, not epochs, first to make it invariant to epoch length, and second because in micro-data RL the unit of cost is a system access step.

Out-of-distribution metrics is an important tool for detecting convergence to (possibly local) optima and to feed the debate around the modelling of epistemic uncertainties (Osband et al., 2018) with hard data. Their definition and our subtle preliminary analysis are in Appendix E.

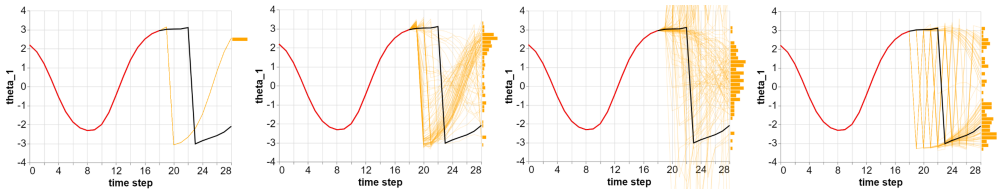
2.3 THE EVALUATION ENVIRONMENT

The Acrobot benchmark system has four observables $\boldsymbol{y} = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]$; θ_1 the angle to the vertical axis of the upper link; θ_2 the angle of the lower link relative to the upper link, both being normalized to $[-\pi, \pi]$; $\dot{\theta}_1$ and $\dot{\theta}_2$ the corresponding angular momenta. The action is a discrete torque on the lower link $a \in \{-1, 0, 1\}$. We use only $\boldsymbol{y}_t$ as the input to the models but augment it with the sines and cosines of the angles, so $\boldsymbol{s}_t = [\theta_1, \sin\theta_1, \cos\theta_1, \theta_2, \sin\theta_2, \cos\theta_2, \dot{\theta}_1, \dot{\theta}_2]_t$. The reward is the height of the tip of the lower link over the hanging position $r(\boldsymbol{y}) = 2 - \cos\theta_1 - \cos(\theta_1 + \theta_2) \in [0, 4]$.

We use two versions of the system to test various properties of the system models we describe in Section 3. In the "raw angles" system we keep $\boldsymbol{y}$ as the prediction target which means that models have to deal with the noncontinuous angle trajectories when the links roll over at $\pm\pi$. This requires multimodal posterior predictives illustrated in Figure 1 and in Appendix G. In the "sincos" system we change the target to $\boldsymbol{y} = [\sin\theta_1, \cos\theta_1, \sin\theta_2, \cos\theta_2, \dot{\theta}_1, \dot{\theta}_2]$ which are the observables of the Acrobot system implementation in OpenAI Gym (Brockman et al., 2016). This smoothes the target but introduces a strong nonlinear dependence between $\sin\theta_{t+1}$ and $\cos\theta_{t+1}$, even given the state $\boldsymbol{s}_t$.

Our aim of predicting dynamic performance on static experiments will require not only score design but also data set design. In this paper we evaluate our models on two data sets. The first is generated by running a random policy $\pi^{(1)}$ on Acrobot. We found that this was too easy to learn, so scores hardly predicted the dynamic performance of the models (Schaul et al., 2019). To create a more "skewed" data set, we execute the MBRL loop (Section 2.2.2) for one iteration using the LinReg_σ model (see Section 3), and generate traces using the resulting policy $\pi^{(2)}_{\mathrm{LinReg}_\sigma}$. On both data sets we use ten-fold cross validation on 5K training points and report test scores on a held-out test set of 20K points. All sets comprise of episodes of length 500, starting from an approximately hanging position: all state variables (the angles and the angular velocities) are uniformly sampled in $[-0.1, 0.1]$.

[4] The common practice is not to normalize the cumulative reward by the (maximum) episode length T, which makes it difficult to immediately compare results across papers and experiments. In micro-data , where T is a hyperparameter (vs. part of the experimental setup), we think this should be the common practice.



(a) deterministic NN_{det} (b) homoscedastic NN_σ (c) heteroscedastic unimodal DARMDN(1) (d) multimodal DARMDN(10)

Figure 1: How different model types deal with uncertainty and chaos around the non-continuity at $\pm\pi$ on the Acrobot "raw angles" system. The acrobot is standing up at step 18 and hesitates whether to stay left ($\theta_1 > 0$) or go right ($\theta_1 < 0$ with a jump of 2π). Deterministic and homoscedastic models underestimate the uncertainty so a small one-step error leads to picking the wrong mode and huge errors down the horizon. A heteroscedastic unimodal model correctly determines the large uncertainty but represents it as a single Gaussian so futures are not sampled from the modes. The multimodal model correctly represents the uncertainty (two modes, each with small sigma) and leads to a reasonable posterior predictive after ten steps. The thick curve is the ground truth, the red segment is past, the black segment is future, and the orange curves are simulated futures. See Section 3 for the definition of the different models and Appendix G for more insight.

3 Models and Results

A commonly held belief (Lee et al., 2019; Wang et al., 2019) is that MBRL learns fast but cannot reach the asymptotic performance of model-free RL. It presumes that models either "saturate" (their approximation error cannot be eliminated even when the size of the training set grows high) and/or they get stuck in local minima (since sampling and learning are coupled). Our research goal is to design models that alleviate these limitations. The first step is to introduce and study models that are learnable with small data but are flexible enough to represent complicated functions (see the summary in Table 1). Implementation details are given in Appendix D.

Table 1: Summary of the different models satisfying (or not) the various requirements from Section 2.1. (R1): efficient simulation; (R2): explicit likelihood; (R3): $\boldsymbol{y}$-interdependence (yellow means "partially"); (R4): heteroscedasticity (yellow means "at training"); (R5): multimodality (yellow means "in principle, yes, in practice, no"); (R6): ability to model different feature types; (R7): robustness and debuggability. The last two columns indicate whether the model is among the optimal ones on the Acrobot sincos and raw angles systems (Section 2.3 and Table 2).

Model	(R1)	(R2)	(R3)	(R4)	(R5)	(R6)	(R7)	sincos	raw angles
(autoregressive) LinReg_σ	✓	✓	✓			✓	✓		
(autoregressive) NN_σ	✓	✓	✓			✓	✓		
GP	✓	✓		✓					
DMDN(1)	✓	✓		✓			✓	✓	
DMDN(10)	✓	✓	✓	✓	✓		✓	✓	✓
DARMDN(1)	✓	✓	✓	✓		✓	✓	✓	
DARMDN(10)	✓	✓	✓	✓	✓	✓	✓	✓	✓
PETS (bagged DMDN(1))	✓	✓		✓	✓		✓	✓	
VAE	✓			✓	✓		✓		
RealNVP	✓	✓		✓	✓				
(autoregressive) NN_{det}	✓		✓			✓	✓		
$\text{DMDN(1)}_{\text{det}}$	✓			✓			✓	✓	
$\text{DARMDN(1)}_{\text{det}}$	✓		✓	✓		✓	✓	✓	

Autoregressive deterministic regressor + fixed variance. We learn d_{y} deterministic regressors $f_1(\boldsymbol{x}^1), \ldots, f_{d_{\text{y}}}(\boldsymbol{x}^{d_{\text{y}}})$ by minimizing MSE and estimate a uniform residual variance $\sigma_j^2 = \frac{1}{T-2}\sum_{t=1}^{T-1}\left(y_{t+1}^j - f_j(\boldsymbol{x}_t^j)\right)^2$ for each output dimension $j = 1, \ldots, d_{\text{y}}$. The probabilistic model is then Gaussian $p_j(y^j|\boldsymbol{x}^j) = \mathcal{N}\big(y^j; f_j(\boldsymbol{x}^j), \sigma_j\big)$. The two baseline models of this type are linear regression (LinReg_σ) and a neural net (NN_σ). These models are easy to train, they can handle $\boldsymbol{y}$-interdependence (since they are autoregressive), but they fail (R5) and (R4): they cannot handle multimodal posterior predictives and heteroscedasticity.

GAUSSIAN PROCESS (GP) is the method of choice in the popular PILCO algorithm (Deisenroth & Rasmussen, 2011). On the modelling side, it cannot handle non-Gaussian (multimodal or heteroscedastic) posteriors and $\boldsymbol{y}$-interdependence, failing Req (R6). More importantly, similarly to Wang et al. (2019) and Chatzilygeroudis et al. (2020), we found it very hard to tune and slow to simulate from. We have reasonable performance on the sincos data set which we report, however GPs failed the raw angles data set (as expected due to angle non-continuity) and, more importantly, the hyperparameters tuned lead to suboptimal dynamical performance, so we decided not to report these results. We believe that generative neural nets that can learn the same model family are more robust, faster to train and sample from, and need less babysitting in the MBRL loop.

MIXTURE DENSITY NETS. A classical deep mixture density net DMDN(D) (Bishop, 1994) is a feed-forward neural net outputting $D(1+2d_{\text{y}})$ parameters $[w^\ell, \boldsymbol{\mu}^\ell, \boldsymbol{\sigma}^\ell]_{\ell=1}^D$, $\boldsymbol{\mu}^\ell = [\mu_j^\ell]_{j=1}^{d_{\text{y}}}$, $\boldsymbol{\sigma}^\ell = [\sigma_j^\ell]_{j=1}^{d_{\text{y}}}$ of a multivariate independent Gaussian mixture $p(\boldsymbol{y}|\boldsymbol{s}) = \sum_{\ell=1}^D w^\ell(\boldsymbol{s})\mathcal{N}\big(\boldsymbol{y}; \boldsymbol{\mu}^\ell(\boldsymbol{s}), \text{diag}(\boldsymbol{\sigma}^\ell(\boldsymbol{s})^2)\big)$. Its autoregressive counterpart DARMDN(D) learns d_{y} independent neural nets outputting the $3Dd_{\text{y}}$ parameters $\big[w_j^\ell, \mu_j^\ell, \sigma_j^\ell\big]_{j,\ell}$ of d_{y} mixtures $p_1, \ldots, p_{d_{\text{y}}}$ (2). Both models are trained to maximize the log likelihood (3). They can both represent heteroscedasticity and, for $D > 1$, multimodal posterior predictives. In engineering systems we prefer DARMDN for its better handling of $\boldsymbol{y}$-interdependence and its ability to model different types of system variables. DARMDN(D) is similar to RNADE (Uria et al., 2013) except that in system modelling we do not need to couple the d_{y} neural nets. While RNADE was used for anomaly detection (Iwata & Yamanaka, 2019), acoustic modelling (Uria et al., 2015), and speech synthesis (Wang et al., 2017), to our knowledge, neither DARMDN nor RNADE have been used in the context of MBRL. DMDN has been used in robotics by Khansari-Zadeh & Billard (2011) and it is an important brick in the world model of Ha & Schmidhuber (2018). Probabilistic Ensembles with Trajectory Sampling (PETS) (Chua et al., 2018) is an important contribution to MBRL that trains a DMDN(D) model by bagging D DMDN(1) models. In our experiments we also found that bagging can improve the LR score (4) significantly, and bagging seems to accelerate learning by being more robust for small data sets (MRCP score in Table 2 and learning curves in Appendix F); however bagged single Gaussians are not multimodal (all bootstrap samples will pick instances from every mode) so PETS fails on the raw angles data.

Table 2: Model evaluation results on the dynamic environments using random shooting MPC agents. $\downarrow$ and $\uparrow$ mean lower and higher the better, respectively. Unit is given after the / sign.

Method	RMAR/$10^{-3}\uparrow$	MRCP$\downarrow$
	Acrobot raw angles system	
LinReg$_\sigma$	215±7	NaN±NaN
NN$_\sigma$	612±9	14630±1930
NN$_{\text{det}}$	703±7	4420±800
DMDN(10)	**968±8**	1760±180
DARMDN(1)	730±7	2840±790
DARMDN(10)	**963±7**	**1460±80**
DARMDN(10)$_{\text{det}}$	709±7	2400±240
PETS	715±7	4420±1700
VAE	668±11	11800±2950
	Acrobot sincos system	
LinReg$_\sigma$	46±3	NaN±NaN
NN$_\sigma$	862±9	2080±280
NN$_{\text{det}}$	862±9	1820±460
DMDN(10)	**981±8**	1580±110
DARMDN(1)	**984±7**	1300±50
DARMDN(10)	**980±7**	1300±70
DARMDN(10)$_{\text{det}}$	**987±7**	1200±70
DARMDN(1)$_{\text{det}}$	**988±7**	1220±70
PETS	**992±7**	960±80
PETS$_{\text{det}}$	**996±7**	**740±70**
VAE	955±10	1510±130
RealNVP	562±26	NaN±NaN

DETERMINISTIC MODELS are important baselines, used successfully by Nagabandi et al. (2018) and Lee et al. (2019) in MBRL. They fail Req (R2) but can be alternatively validated using R2. On the other hand, when used in an autoregressive setup, if the mean prediction represents the posterior predictives well (unimodal distributions with small uncertainty), they may work very well. We implemented deterministic models by "sampling" the mean of the NN$_\sigma$ and DARMDN($\cdot$) models, obtaining NN$_{\text{det}}$ and DARMDN($\cdot$)$_{\text{det}}$, respectively.

VARIATIONAL AUTOENCODERS AND FLOWS. We tested two other popular techniques, variational autoencoders (VAE) (Kingma & Welling, 2014; Rezende et al., 2014) and the flow-based RealNVP (Dinh et al., 2017). VAE does not provide exact likelihood (R2); RealNVP does, but the R2 and KS scores are harder to compute. In principle they can represent multimodal posterior predictives, but in practice they do not seem to be flexible enough to work well on the raw angles system. A potential solution would be to enforce a multimodal output as done by Moerland et al. (2017). VAE performed well (although significantly worse than the mixture models) on the sincos system.

Our results are summarized in Tables 2 and 3. We show mean reward learning curves in Appendix F. We found that comparing models solely based on their performance on the random policy data is a

Table 3: Model evaluation results on static data sets. ↓ and ↑ mean lower and higher the better, respectively. Unit is given after the / sign.

Method	LR↑	OR/10^{-4}↓	R2/10^{-4}↑	KS/10^{-3}↓	R2(10)/10^{-4}↑	KS(10)/10^{-3}↓	trt/min↓	tst/sec↓
	Acrobot raw angles, data generated by random policy							
LinReg_σ	27±1	44±7	9763±0	177±3	8308±485	157±11	0±0	0±0
NN_σ	54±8	171±37	9829±9	171±36	8711±491	212±48	2±0	1±0
DMDN(10)	430±26	0±0	9790±2	124±10	8973±456	129±29	15±0	2±0
DARMDN(1)	424±18	10±2	9784±2	126±6	9267±269	106±17	19±0	2±0
DARMDN(10)	410±8	3±1	9782±2	135±8	9049±375	122±17	18±0	2±0
	Acrobot raw angles, data generated by linear policy after one epoch							
LinReg_σ	3±0	20±5	6832±9	85±1	398±270	87±14	0±0	0±0
NN_σ	25±1	176±31	9574±13	193±16	4844±477	139±23	2±0	1±0
DMDN(10)	137±10	40±11	8449±443	72±9	5659±1086	135±19	15±0	2±0
DARMDN(1)	120±2	56±12	5677±6	47±5	1291±846	114±20	20±1	2±0
DARMDN(10)	143±6	22±6	9571±70	62±5	8065±363	100±11	20±0	2±0
	Acrobot sincos, data generated by random policy							
LinReg_σ	6±0	47±10	8976±1	118±3	5273±320	110±11	0±0	0±0
NN_σ	50±4	188±20	9987±5	176±22	9249±623	257±64	4±0	2±0
GP	88±2	0±0	9999±0	224±11	9750±85	168±29	0±0	9±1
DMDN(10)	361±22	0±0	9957±4	139±15	8963±538	146±35	21±1	1±0
DARMDN(1)	281±5	3±1	9950±5	151±3	8953±337	131±18	27±1	3±0
DARMDN(10)	288±7	1±0	9983±4	153±10	9296±233	140±25	28±1	4±1
	Acrobot sincos, data generated by linear policy after one epoch							
LinReg_σ	2±0	11±4	6652±9	46±1	354±304	127±18	0±0	0±0
NN_σ	32±2	166±34	9986±2	156±16	5529±1061	194±29	4±0	2±0
GP	56±1	6±1	9995±0	113±4	8334±185	133±15	0±0	9±1
DMDN(10)	95±5	29±6	9993±1	85±9	9001±285	128±17	21±0	1±0
DARMDN(1)	125±4	12±4	9991±1	80±4	8693±286	89±13	32±2	3±0
DARMDN(10)	119±4	9±5	9991±2	68±4	8655±269	95±15	30±1	4±0

bad choice: most models did well in both the raw angles and sincos systems. Static performance on the linear policy data is a better predictor of the dynamic performance; among the scores, not surprisingly, and also noted by Nagabandi et al. (2018), the R2(10) score correlates the most with dynamic performance.

Our most counter-intuitive result (although Wang et al. (2019) and Wang & Ba (2020) observed a similar phenomenon) is that $\text{DARMDN}(1)_{\text{det}}$ and PETS_{det} are tied for winning on the sincos system (reaching the optimum MAR_{opt} within statistical uncertainty), which suggests that a deterministic model can be on par with the best probabilistic models if the system requires no multimodality. What is even more surprising is that NN_{det} does not do well, suggesting that the optimal model, even if it is deterministic, needs to be trained for a likelihood score in a generative setup. The low R2(10) score of NN_{det} (and the case study in Appendix G) suggest that classical regression optimizing MSE leads to error accumulation and thus subpar performance down the horizon. Our hypothesis is that heteroscedasticity at training time allows the models to adjust the weights of training points, leading somehow to little error accumulation at a longer horizon.

On the sincos system D(AR)MDN and PETS reach the optimum MAR_{opt} within statistical uncertainty which means that this setup of the Acrobot system is essentially solved. We improve the convergence pace of the PETS implementation of Wang & Ba (2020) by two to four folds (Figure 3 in Appendix F), even though most of the improvement comes from a more ambitious learning schedule (short epochs and frequent retraining). The real forte of D(AR)MDN is the 95% RMAR score on the raw angles system that requires multimodality, beating the other models by about 20%. It suggests remarkable robustness that makes it the method of choice for larger systems with more complex dynamics.

4 Conclusion and Future Work

Our study was made possible by developing a toolbox of good practices for model evaluations and debuggability in model-based reinforcement learning, particularly useful when trying to solve real world applications with domain engineers. We found that heteroscedasticity at *training time* largely alleviates error accumulation down the horizon. Then at *planning time*, we do not need stochastic models: the deterministic mean prediction suffices. That is, unless the system requires multimodal posterior predictives, in which case deep (autoregressive or not) mixture density nets are the only current generative models that work. Our findings lead to state-of-the-art sample complexity (by far) on the Acrobot system by applying an aggressive training schedule. The most important future direction is to extend the results to larger systems and to planning strategies beyond random shooting.

REFERENCES

Zaheer Abbas, Samuel Sokota, Erin J. Talvitie, and Martha White. Selective Dyna-style planning under limited model capacity. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *CoRR*, abs/2008.05556, 2020. URL https://arxiv.org/abs/2008.05556.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. *ArXiv*, abs/2003.13350, 2020.

Christopher M. Bishop. Mixture density networks. Technical report, 1994.

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

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21st International Conference on Knowledge Discovery and Data Mining*, pp. 1721–1730. ACM, 2015.

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *IEEE Transactions on Robotics*, 36(2):328–347, 2020.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pp. 4754–4765. Curran Associates, Inc., 2018.

Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning. In *Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2006.08684.

Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134, 2004.

Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning*, 2011.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

Stefan Depeweg, Jos Miguel Hernndez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of Uncertainty in Bayesian Deep Learning for Efficient and Risk-sensitive Learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1192–1201. PMLR, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Yarin Gal, Rowan McAllister, and Carl E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, April 2016.

Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, 2018.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2450–2462. Curran Associates, Inc., 2018.

E. Hullermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *arXiv: Learning*, 2019.

Tomoharu Iwata and Yuki Yamanaka. Supervised anomaly detection based on deep autoregressive density estimators. *arXiv preprint arXiv:1904.06034*, 2019.

Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. Modeling the long term future in model-based reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkgQBn0cF7.

Balázs Kégl, Alexandre Boucaud, Mehdi Cherti, Akin Kazakci, Alexandre Gramfort, Guillaume Lemaitre, Joris Van den Bossche, Djalel Benbouzid, and Camille Marini. The RAMP framework: from reproducibility to transparency in the design and optimization of scientific workflows. In *ICML workshop on Reproducibility in Machine Learning*, 2018.

S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent Actor-Critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

Ali Malik, Volodymyr Kuleshov, Jiaming Song, Danny Nemer, Harlan Seymour, and Stefano Ermon. Calibrated model-based deep reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4314–4323. PMLR, 2019.

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Learning multimodal transition dynamics for model-based reinforcement learning. In *Scaling Up Reinforcement Learning (SURL) Workshop @ European Machine Learning Conference (ECML)*, 2017.

Jean-Baptiste Mouret. Micro-Data Learning: The Other End of the Spectrum. *ERCIM News*, (107):2, September 2016. URL https://hal.inria.fr/hal-01374786.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018*, pp. 7559–7566. IEEE, 2018.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems 31*, pp. 8617–8629. Curran Associates, Inc., 2018.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems 30*, pp. 2338–2347. Curran Associates, Inc., 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouart Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Anil Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135, 01 2010.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286. PMLR, 2014.

Arthur George Richards. Robust constrained model predictive control. *PhD thesis, Massachusetts Institute of Technology*, 2005.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcemen learning. *arXiv preprint arXiv:1904.11455*, 2019.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3483–3491. Curran Associates, Inc., 2015.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (eds.), *Advances in Neural Information Processing Systems 8*, pp. 1038–1044. MIT Press, 1996.

R. Ueda and T. Arai. Dynamic programming for global control of the Acrobot and its chaotic aspect. In *2008 IEEE International Conference on Robotics and Automation*, pp. 2416–2422, 2008.

Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems 26*, pp. 2175–2183. Curran Associates Inc., 2013.

Benigno Uria, Iain Murray, Steve Renals, Cassia Valentini-Botinhao, and John Bridle. Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4465–4469, 2015.

Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michal Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575, 11 2019. doi: 10.1038/s41586-019-1724-z.

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Xin Wang, Shinji Takaki, and Junichi Yamagishi. An autoregressive recurrent mixture density network for parametric speech synthesis. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4895–4899, 2017.

A AUTOREGRESSIVE MIXTURE DENSITIES

The multi-variate density $p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t)$ is decomposed into a chain of one-dimensional densities

$$p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t) = p_1(y^1_{t+1}|\boldsymbol{s}_t) \prod_{j=2}^{d_\mathrm{y}} p_j(y^j_{t+1}|y^1_{t+1}, \ldots, y^{j-1}_{t+1}, \boldsymbol{s}_t) = p_1(y^1_{t+1}|\boldsymbol{x}^1_t) \prod_{j=2}^{d_\mathrm{y}} p_j(y^j_{t+1}|\boldsymbol{x}^j_t), \quad (1)$$

where, for simplicity, we denote the input (condition) of the jth autoregressive predictor by $\boldsymbol{x}^j_t = \big(y^1_{t+1}, \ldots, y^{j-1}_{t+1}, \boldsymbol{s}_t\big)$. First, p is a proper d_y-dimensional density as long as the components p_j are valid one-dimensional densities (Req (R2)). Second, if it is easy to draw from the components p_j, it is easy to simulate $\boldsymbol{Y}_{t+1}$ following the order of the chain (1) (Req (R1)). Third, Req (R3) is satisfied by construction. But the real advantages are on the logistics of modelling. Unlike in computer vision (pixels) or NLP (words), engineering systems often have inhomogeneous features that should be modeled differently. There exists a plethora of different one-dimensional density models which we can use in the autoregressive setup, whereas multi-dimensional extensions are rare, especially when feature types are different (Req (R6)). At the debuggability side (Req (R7)) the advantage is the availability of one-dimensional goodness of fit metrics and visualization tools which make it easy to pinpoint what goes wrong if the model is not working. On the negative side, autoregression breaks the symmetry of the output variables by introducing an artificial ordering and, depending on the family of the component densities p_j, the modelling quality may depend on the order.

To preserve these advantages and alleviate the order dependence we found that we needed a rich family of one-dimensional densities so we decided to use mixtures

$$p_j(y^j|\boldsymbol{x}^j) = \sum_{\ell=1}^{D} w^\ell_j(\boldsymbol{x}^j) P^\ell_j\big(y^j; \theta^\ell_j(\boldsymbol{x}^j)\big), \quad (2)$$

where component types P^ℓ_j, component parameters θ^ℓ_j, and component weights w^ℓ_j can all depend on j, ℓ, and the input $\boldsymbol{x}^j$. In general, the modeller has a large choice of easy-to-fit component types to choose from given the type of variable y^j (Req (R6)); in this paper all our variables were numerical so we only use Gaussian components with free mean and variance. Contrary to the widely held belief (Papamakarios et al., 2017), in our experiments we found no evidence that the ordering of the variables matters, arguably because of the flexibility of the one-dimensional mixture models that can pick up non-Gaussian features such as multimodality (Req (R5)). Finally a computational advantage: given a test point $\boldsymbol{x}$, we do not need to carry around (density) functions: our representation of $p(\boldsymbol{y}|\boldsymbol{x})$ is a numerical vector concatenating $\big[w^\ell_j, P^\ell_j, \theta^\ell_j\big]_{j,\ell}$.

B $\boldsymbol{y}$-INTERDEPENDENCE

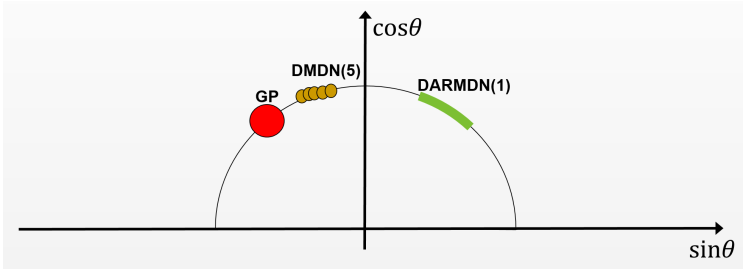


Figure 2: How different models handle $\boldsymbol{y}$-interdependence. GP (and DMDN(1)) "spreads" the uncertainty in all directions, leading to non-physical predictions. DMDN($D > 1$) may "tile" the nonlinear $\boldsymbol{y}$-interdependence with smaller Gaussians, and in the limit of $D \to \infty$ it can handle $\boldsymbol{y}$-interdependence for the price of a large number of parameters to learn. DARMDN, with its autoregressive function learning, can put the right amount of dependent uncertainty on $y^2|y^1$, learning for example the noiseless functional relationship between $\cos\theta$ and $\sin\theta$.

$\boldsymbol{y}$-interdependence is the dependence among the d_{y} elements of $\boldsymbol{y}_{t+1} = (y^1_{t+1}, \ldots, y^{d_{\mathrm{y}}}_{t+1})$ given $\mathcal{T}_t$. Some popular algorithms such as PILCO (Deisenroth & Rasmussen, 2011) suppose that elements of $\boldsymbol{y}_{t+1}$ are independent given $\mathcal{T}_t$. It is a reasonable assumption when modelling aleatoric uncertainty in stochastic systems with independent noise, but it is clearly wrong when the posterior predictive has a structure due to functional dependence. It happens even in the popular AI Gym benchmark systems (Brockman et al., 2016) (think about usual representation of angles: $\cos\theta_{t+1}$ is clearly dependent of $\sin\theta_{t+1}$ even given $\mathcal{T}_t$; see Figure 2), let alone systems with strong physical constraints in telecommunication or robotics. Generating non-physical traces due to not modelling $\boldsymbol{y}$-interdependence may lead not only to subpar performance but also to reluctance to accept the models (simulators) by system engineers.

C STATIC METRICS

We define our static metrics from the decomposition of the multivariate density $p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t)$ into the product of one-dimensional densities (see Appendix A for details):

$$p(\boldsymbol{y}_{t+1}|\boldsymbol{s}_t) = p_1(y^1_{t+1}|\boldsymbol{x}^1_t) \prod_{j=2}^{d_{\mathrm{y}}} p_j(y^j_{t+1}|\boldsymbol{x}^j_t) \quad \text{where} \quad \boldsymbol{x}^j_t = \big(y^1_{t+1}, \ldots, y^{j-1}_{t+1}, \boldsymbol{s}_t\big).$$

LIKELIHOOD RATIO TO A SIMPLE BASELINE (LR) is our "master" metrics. The (average) log-likelihood

$$\mathcal{L}(\mathcal{T}_T; p) = \frac{1}{d_{\mathrm{y}}} \sum_{j=1}^{d_{\mathrm{y}}} \frac{1}{T-1} \sum_{t=1}^{T-1} \log p_j \left(y^j_{t+1}|\boldsymbol{x}^j_t\right) \tag{3}$$

can be evaluated easily on any trace $\mathcal{T}_T$ thanks to Req (R2). Log-likelihood is a unitless metrics which is hard to interpret and depends on the unit in which its input is measured (this variability is particularly problematic when p_j is a mixed continuous/discrete distribution). To have a better interpretation, we normalize the likelihood

$$\mathrm{LR}(\mathcal{T}; p) = \frac{e^{\mathcal{L}(\mathcal{T};p)}}{e^{\mathcal{L}_{\mathrm{b}}(\mathcal{T})}} \tag{4}$$

with a baseline likelihood $\mathcal{L}_{\mathrm{b}}(\mathcal{T})$ which can be adapted to the feature types. In our experiments $\mathcal{L}_{\mathrm{b}}(\mathcal{T})$ is a multivariate independent unconditional Gaussian. LR is between 0 (although LR < 1 usually indicates a bug) and ∞, the higher the better.

OUTLIER RATE (OR). We found that LR works well in an i.i.d. setup but distribution shift often causes "misses": test points with extremely low likelihood. Since these points dominate $\mathcal{L}$ and LR, we decided to clamp the likelihood at[5] $p_{\min} = 1.47 \times 10^{-6}$. Given a trace $\mathcal{T}$ and a model p, we define $\mathcal{T}(p; p_{\min}) = \big\{(\boldsymbol{y}_t, \boldsymbol{a}_t) \in \mathcal{T} : p(\boldsymbol{y}_t|\boldsymbol{x}_{t-1}) > p_{\min}\big\}$, report $\mathrm{LR}\big(\mathcal{T}(p; p_{\min}); p\big)$ instead of $\mathrm{LR}(\mathcal{T}; p)$, and measure the "surprise" of a model on trace $\mathcal{T}$ by the outlier rate (OR)

$$\mathrm{OR}(\mathcal{T}; p) = 1 - \frac{|\mathcal{T}(p; p_{\min})|}{|\mathcal{T}|}. \tag{5}$$

OR is between 0 and 1, the lower the better.

EXPLAINED VARIANCE (R2) assesses the mean performance of the methods. Formally

$$\mathrm{R2}(\mathcal{T}_T; p) = \frac{1}{d_{\mathrm{y}}} \sum_{j=1}^{d_{\mathrm{y}}} \left(1 - \frac{\mathrm{MSE}_j(\mathcal{T}_T; p)}{\sigma_j^2}\right) \text{ with } \mathrm{MSE}_j(\mathcal{T}_T; p) = \frac{1}{T-1} \sum_{t=1}^{T-1} \left(y^j_{t+1} - f_j(\boldsymbol{x}_t)\right)^2, \tag{6}$$

where $f_j(\boldsymbol{x}_t) = \mathbb{E}_{p_j(\cdot|\boldsymbol{x}^j_t)}\{y^j\}$ is the expectation of y^j_{t+1} given $\boldsymbol{x}^j_t$ under the model p_j (point prediction), and σ_j^2 is the sample variance of $(y^j_1, \ldots, y^j_T)$. We prefer using this metrics over the

[5] As a salute to five sigma, using the analogy of the MBRL loop (Section 2.2.2) being the iterated scientific method.

MSE because it is normalized so it can be aggregated over the dimensions of $\boldsymbol{y}$. R2 is between 0 and 1, the higher the better.

CALIBRATEDNESS (KS). Well-calibrated models have been shown to improve the performance of algorithms (Malik et al., 2019). A well-calibrated density estimator has the property that the quantiles of the (test) ground truth are uniform. To assess this, we compute the Kolmogorov-Smirnov (KS) statistics. Formally, let $F_j(y^j|\boldsymbol{x}^j) = \int_{-\infty}^{y^j} p_j(y'|\boldsymbol{x}^j)\mathrm{d}y'$ be the cumulative distribution function (CDF) of p_j, and let the order statistics of $\mathcal{F}_j = \left[F_j\left(y_{t+1}^j|\boldsymbol{x}_t^j\right)\right]_{t=1}^{T-1}$ be s_j, that is, $F_j\left(y_{s_j}^j|\boldsymbol{x}_{s_j}^j\right)$ is the s_jth largest quantile in $\mathcal{F}_j$. Then we define

$$\mathrm{KS}(\mathcal{T}_T; F) = \frac{1}{d_\mathrm{y}} \sum_{j=1}^{d_\mathrm{y}} \max_{s_j \in [1, T-1]} \left| F_j\left(y_{s_j}^j|\boldsymbol{x}_{s_j}^j\right) - \frac{s_j}{T-1} \right|. \tag{7}$$

Computing KS requires that the model can provide conditional CDFs, which further filters the possible models we can use. On the other hand, the aggregate KS and especially the one-dimensional CDF plots ($F_j(y_{s_j}^j|\boldsymbol{x}_{s_j}^j)$ vs. $s_j/(T-1)$) are great debugging tools. KS is between 0 and 1, the lower the better.

All four metrics (LR, OR, R2, KS) are averaged over the dimensions, but for debugging we can also evaluate them dimension-wise.

LONG HORIZON METRICS KS(L) AND R2(L). All our density estimators are trained to predict the system one step ahead yet arguably what matters is their performance at a longer horizon L specified by the control agent. Our models do not provide explicit likelihoods L steps ahead, but we can simulate from them (following ground truth actions) and evaluate the metrics by a Monte-Carlo estimate. Given n random estimates $\mathcal{Y}_L = [\hat{\boldsymbol{y}}_{t+L,\ell}]_{\ell=1}^n$, we can use $f_j(\boldsymbol{x}_t) = \frac{1}{n}\sum_{\hat{\boldsymbol{y}} \in \mathcal{Y}_L} \hat{y}^j$ in (6) to obtain an unbiased R2(L) estimate. To obtain a KS(L) estimate, we order $\mathcal{Y}_L$ and approximate $F_j(y^j|\boldsymbol{x}^j)$ by $\frac{1}{n}|\{\hat{\boldsymbol{y}} \in \mathcal{Y}_L : \hat{y}^j < y^j\}|$ in (7). LR and OR would require approximate techniques so we omit them. In all our experiments we use $L = 10$, $n = 100$, and, for computational reasons, sample the test set at 100 random positions, which explains the high variance on these scores.

D IMPLEMENTATION DETAILS

Note that all experimental code will be put online as soon as the anonymity is lifted. In this section we give enough information so that all models can be reproduced by a moderately experienced machine learning expert.

The sincos and raw angles Acrobot systems are based on the OpenAI Gym implementation (Brockman et al., 2016). The starting position of each episode is the one obtained from the default `reset` function of this implementation: all state variables (the angles and the angular velocities) are uniformly sampled in $[-0.1, 0.1]$. For the linear regression model we use the implementation of Scikit-learn (Pedregosa et al., 2011) without regularization. We use Pytorch (Paszke et al., 2019) for the neural network based models (NN, DMDN and DARMDN) and Gpytorch (Gardner et al., 2018) for the GP models. The hyperparameter search for these models was done in two steps: first using random search over a coarse hyperparameter grid, then using a second step of random search over a finer grid around values of interest. The steps of the coarse grid were defined to contain five values of each hyperparameters (or less where applicable), the finer grid was defined to contain five values of each hyperparameter (or less where applicable) between two interesting spots close in the hyperparameter space. The selected hyperparameters are given in Table 4.

"Nb layers" corresponds to the number of fully connected layers, except for the two following models:

- RealNVP (Dinh et al., 2017): it is the number of coupling layers.
- CVAE (Sohn et al., 2015): it is the total number of layers (encoder plus decoder).

"Nb components" is the number of components in the outputted density mixture. In the GP and deterministic NN cases, it is trivially one.

Table 4: Model hyperparameters.

Method	Learning rate	Neurons per layer	Nb layers	Nb components	Validation size	Nb epochs
			Tried values			
NN_σ	[1e-4, 1e-1]	[20, 300]	[1, 4]	1	[0.05, 0.4]	[10, 300]
DMDN	[1e-5, 1e-2]	[100, 600]	[2, 5]	[2, 20]	[0.05, 0.4]	[50, 500]
DARMDN	[1e-5, 1e-2]	[20, 300]	[1, 4]	[2, 20]	[0.05, 0.4]	[50, 500]
CVAE	[1e-5, 1e-2]	[20, 300]	[4, 10]	NaN	[0.05, 0.4]	[50, 500]
RealNVP	[1e-5, 1e-2]	[10, 300]	[2, 5]	NaN	[0.05, 0.4]	[50, 500]
GP	[1e-3, 1e-1]	NaN	NaN	1	[0.05, 0.4]	[10, 300]
			Best values			
NN_σ	4e-3	200	3	1	0.05	100
DMDN(10)	5e-3	200	3	10	0.1	300
DARMDN(1)	1e-3	50	3	1	0.1	300
DARMDN(10)	1e-3	100	3	10	0.1	300
CVAE	1e-3	60	4	NaN	0.15	100
RealNVP	5e-3	20	3	NaN	0.15	200
GP	5e-2	NaN	NaN	1	0.15	50

For PETS we use the code shared by Wang et al. (2019) for the Acrobot sincos system. Following Chua et al. (2018), the size of the ensemble is set to 5. For the Acrobot raw angles system we use the same PETS neural network architecture as the one available for the original sincos system. Although the default number of epochs was set to 5 in the available code we reached better results with 100 epochs and use this value in our results. Finally, the RS agent is configured to be the same as the one we use: planning horizon $L = 10$, search population size $n = 100$ and 5 particles.

We selected the planning strategy (random shooting with search population size $n = 100$) by evaluating the performance of random shooting and the cross entropy method (CEM) on the true dynamics for different values of n. Results are presented in Table 5. Although for both RS and CEM with $n = 500$ leads to a better performance, $n = 100$ is already sufficient to achieve more than decent mean rewards and outperform the result of Wang et al. (2019) while reducing the total computational cost of the study. CEM was implemented with a learning rate of 0.1, an elite size equal to 50 and 5 iterations. For a fair comparison between RS and CEM n means the total number of sampled action sequences. This means that, for CEM, n means a search population size of $n/5$ for each of the 5 iterations.

Table 5: Comparison of RS and CEM on the true dynamics. The $\pm$ values are 90% Gaussian confidence intervals based on 100 random repetitions of a 200-step rollout.

RS with $n = 100$	RS with $n = 500$	RS with $n = 1000$	CEM with $n = 500$	CEM with $n = 1000$
2.10 ± 0.035	2.27 ± 0.025	2.29 ± 0.024	2.32 ± 0.021	2.30 ± 0.021

We implemented reusable system models and static experiments within the RAMP framework (Kégl et al., 2018).

All $\pm$ values in the results tables are 90% Gaussian confidence intervals based on i) 10-fold cross-validation for the static scores in Table 3, ii) 50 epochs and two to ten seeds in the RMAR and OXX columns, and iii) ten seeds in the MRCP column of Table 2 and 6.

E Out-of-distribution metrics

One of the most interesting aspects of system modelling for control is that the data sets on which we train and test models are rarely coming from an i.i.d. process, so we cannot rely on classical learning theory for model evaluation, model selection, and other activities we usually do in supervised learning, when we learn $p^{(\tau)}$ in Step 2 of the MBRL loop. The non-i.i.d. setup generates heated debates on how to model epistemic and aleatoric uncertainty and how to evaluate models when epistemic uncertainty (the known unknowns due to undersampling some regions of the phase space) is significant (Osband et al., 2018). We feed this debate with experimental data. We define out-of-distribution scores $\text{OXX}(\tau) = \text{XX}\left(\mathcal{T}_T^{(\tau+1)}; p^{(\tau)}\right)$ (XX $\in$ [LR, OR, R2, KS]) and $\text{OXX} = \frac{1}{N}\sum_{\tau=1}^{N}\text{OXX}(\tau)$ in the

most relevant way, that is, how good $p^{(\tau)}$ is for modelling the likelihood of the *subsequent* trace $\mathcal{T}_T^{(\tau+1)}$ of the MBRL loop. Since $\mathcal{T}_T^{(\tau+1)}$ does not exist at the time $p^{(\tau)}$ is learned, we cannot use OXX for training or model selection, but they are relevant out-of-distribution test metrics on comparing various models after learning.

We found that good models all show high OXX scores (Table 6). Sometimes even suboptimal models have high OXX scores which indicates that they converge to local minima where the MBRL loop (data taking and re-learning; Section 2.2.2) becomes stationary; in other words, the models $p^{(\tau)}$ are "happy with" or "unsurprised by" the data $\mathcal{T}_T^{(\tau+1)}$ that the subsequent policy $\pi^{(\tau)}$ generates. It is interesting that those low-RMAR-high-OOX models are also the ones whose RMAR score varies slightly more than it should according to our preliminary ANOVA analysis, also indicating that they are stuck in local minima.

Table 6: Model evaluation results on the dynamic environments using random shooting MPC agents. $\downarrow$ and $\uparrow$ mean lower and higher the better, respectively. Unit is given after the / sign.

Method	RMAR/$10^{-3}\uparrow$	MRCP$\downarrow$	OLR$\uparrow$	OOR/$10^{-4}\downarrow$	OR2/$10^{-4}\uparrow$	OKS/$10^{-3}\downarrow$
			Acrobot raw angles system			
LinReg$_\sigma$	209±21	NaN±NaN	9±4	39±12	9527±78	256±9
NN$_\sigma$	613±20	10000±0	40±2	12±2	9965±8	229±6
NN$_{\text{det}}$	689±15	4700±800	NaN	NaN	9950±13	NaN
DMDN(10)	774±13	2100±500	508±39	1±1	9168±112	129±5
DARMDN(1)	725±14	1900±200	324±22	6±1	7744±90	133±3
DARMDN(10)	946±17	1600±0	282±28	0±0	9608±48	161±6
DARMDN(10)$_{\text{det}}$	709±12	2100±400	NaN	NaN	9644±44	NaN
PETS-RS	708±10	3500±2100	NaN	NaN	NaN	NaN
			Acrobot sincos system			
LinReg$_\sigma$	46±10	NaN±NaN	14±1	43±23	8758±55	195±11
NN$_\sigma$	879±20	2200±0	64±4	6±3	9989±5	216±10
NN$_{\text{det}}$	819±20	2600±1200	NaN	NaN	9985±6	NaN
DMDN(10)	803±9	1500±200	357±26	0±0	9995±2	137±5
DARMDN(1)	976±18	1300±100	366±36	0±0	9970±20	179±5
DARMDN(10)	1000±15	1400±200	333±34	0±0	9989±5	167±5
DARMDN(10)$_{\text{det}}$	1022±16	1400±0	NaN	NaN	9991±3	NaN
PETS-RS	987±10	900±100	NaN	NaN	NaN	NaN

F MEAN REWARD LEARNING CURVES

Figure 3 shows the mean reward learning curves on the Acrobot raw angles and sincos systems. The top four models PETS, DARMDN(10), DARMDN(10)$_{\text{det}}$ and DARMDN(1) converge close to the optimum at around the same pace on the sincos system. PETS converges slightly faster than the other models in the early phase. Our hypothesis is that bagging creates more robust models in the extreme low data regime (100s of training points). Our models were tuned using 5000 points which seems to coincide with the moment when the bagging advantage disappears.

On the raw angles system DARMDN(10) separates from the pack indicating that this setup requires non-deterministic predictors and mixture densities to model multimodal posterior predictives.

The reward is between 0 (hanging) and 4 (standing up). Each epoch starts at hanging position and it takes about 100 steps to reach the stationary regime where the tip of acrobot is above the horizontal line most of the time. This means that reaching an average reward above 2 needs an excellent control policy. Videos will be available as soon as author anonymity is lifted.

G THE POWER OF DARMDN: PREDICTING THROUGH CHAOS

Acrobot is a chaotic system (Ueda & Arai, 2008): small divergence in initial conditions may lead to large differences down the horizon. This behavior is especially accentuated when the acrobot slowly approaches the unstable standing position, hovers, "hesitates" which way to go, and "decides" to fall back left or right. Figures 4 and 5 depict this precise situation (from the test file of the "linear" data, see Section 2.3): around step 18 both angular momenta are close to zero and $\theta_1 \approx \pi$. To make the

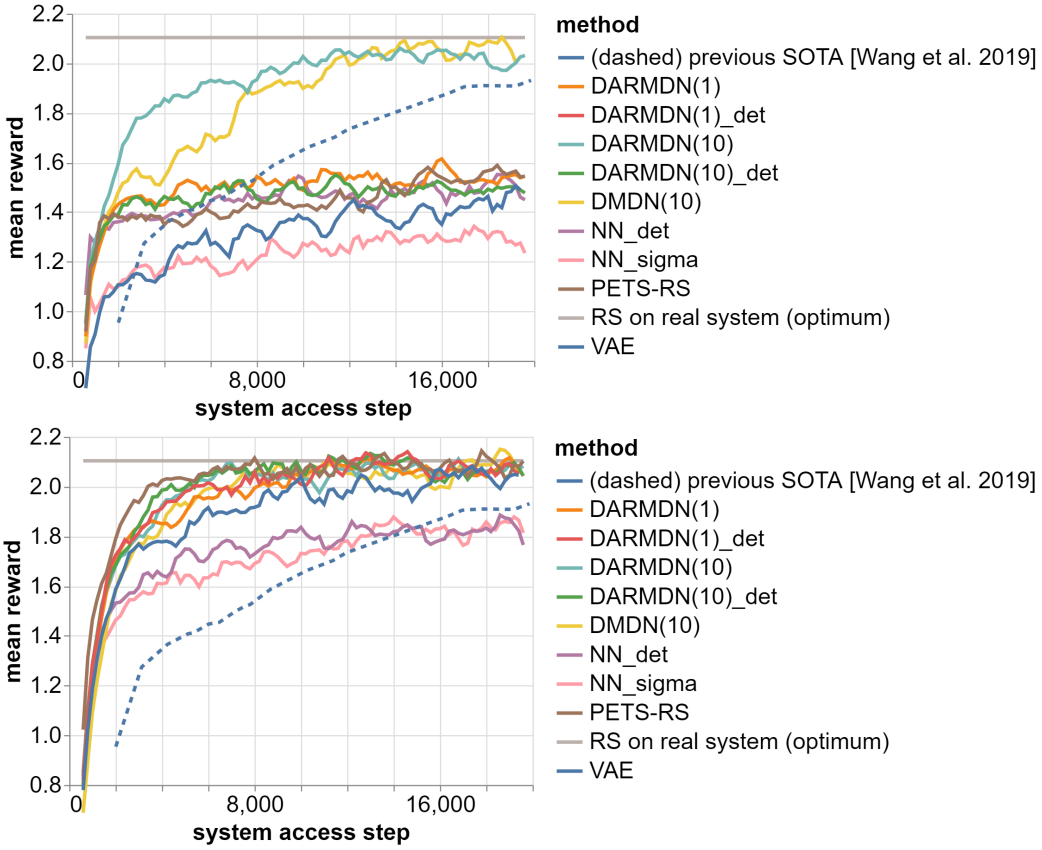


Figure 3: Acrobot learning curves on the raw angles (top) and sincos (bottom) systems. Reward is between 0 (hanging) and 4 (standing up). Episode length is $T = 200$, number of epochs is $N = 100$ with one episode per epoch. Mean reward curves are averaged across three to ten seeds and smoothed using a running average of five epochs, plotted at the middle of the smoothing window (so the first point is at step 600).

modelling even harder, $\theta. = \pi$ is the exact point where the trajectory is non-continuous in the raw angles data, making it hard to model by predictive densities that cannot handle non-smooth traces.

In both figures we show the ground truth (red: past, black: future) and hundred simulated traces (orange) starting at step 18. There is no "correct" solution here since one can imagine several plausible "beliefs" learned using limited data. Yet it is rather indicative about their performance how the different models handle this situation.

First note how diverse the models are. On the sincos data (Figure 4) most posterior predictives after ten steps are unimodal. GP and DARMDN(10) are not, but while GP predicts a coin toss whether Acrobot falls left or right, DARMDN(10) bets more on the ground truth mode. Among the deterministic models, both NN_{det} and $\text{DARMDN(10)}_{\text{det}}$ work well one step ahead (on average, according to their R2 score in Table 3), but ten steps ahead $\text{DARMDN(10)}_{\text{det}}$ is visibly better, illustrating its excellent R2(10) score.

On the raw angles data (Figure 5) we see a very different picture. The deterministic NN_{det} picks one of the modes which happens to be the wrong one, generating a completely wrong trajectory. $\text{DARMDN(10)}_{\text{det}}$ predicts average of two extremem modes (around π and $-\pi$), resulting in a non-physical prediction (θ_1) which has in fact zero probability under the posterior predictive of DARMDN(10). The homoscedastic NN_σ has a constant sigma which, in this situation is too small: it cannot "cover" the two modes, so the model picks one, again the wrong one. The heteroscedastic DARMND(1) correctly outputting a huge uncertainty, but since it is a single unimodal Gaussian, it generates a lot of non-physical predictions between and outside of the modes. This shows that

heteroscedasticity without multimodality may be harmful in these kinds of systems. Finally, DAR-MDN(10) has a higher variance than on the sincos data, especially on the mode not validated by the ground truth, but it is the only model which puts high probability on the ground truth after ten steps, and whose uncertainty is what a human would judge reasonable.

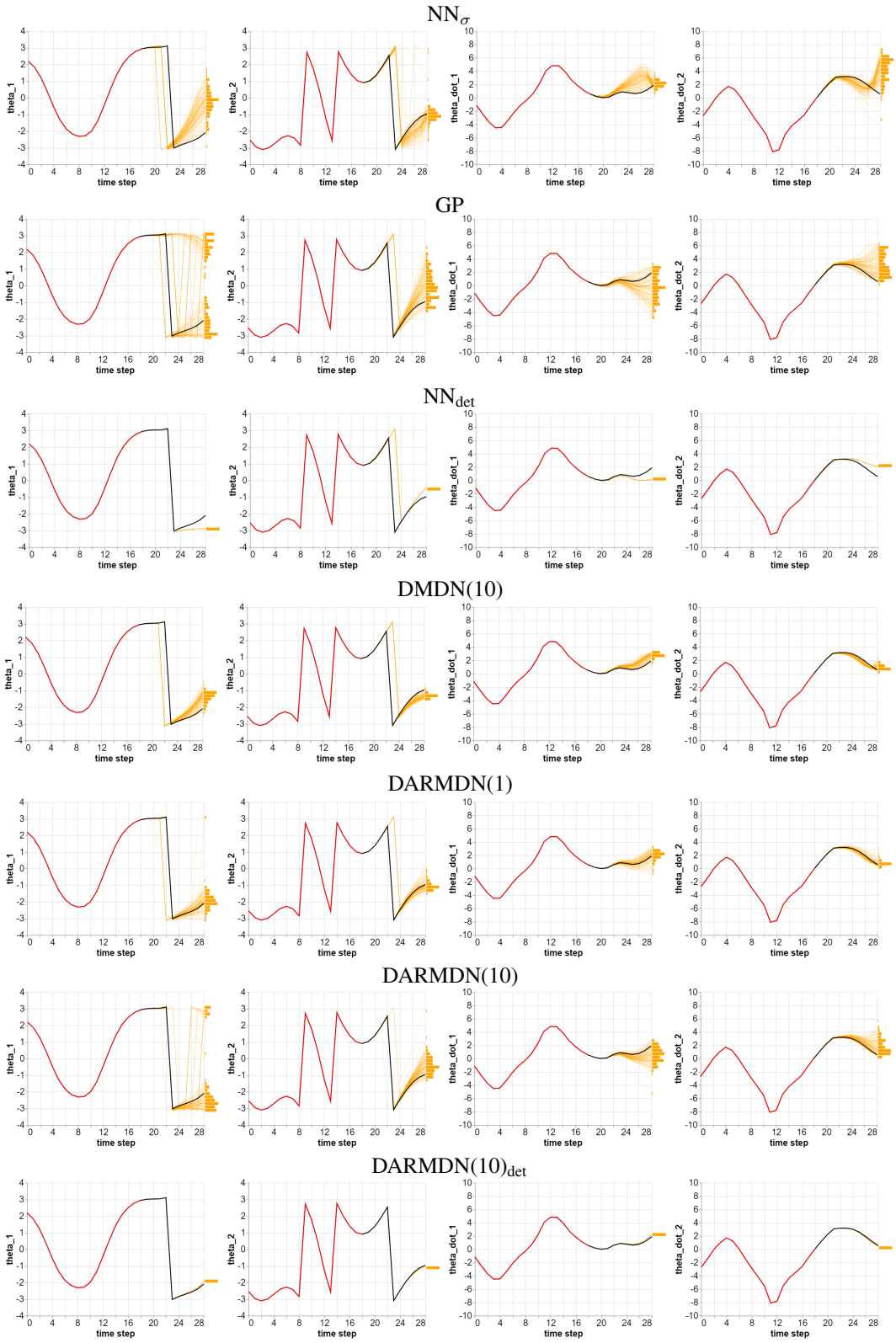


Figure 4: Ground truth and simulation of "futures" by the models trained on the sincos system. The thick curve is the ground truth, the red segment is past, the black segment is future. System models start generating futures from their posterior predictives at step 18. We show a sample of hundred trajectories and a histogram after ten time steps (orange).

.

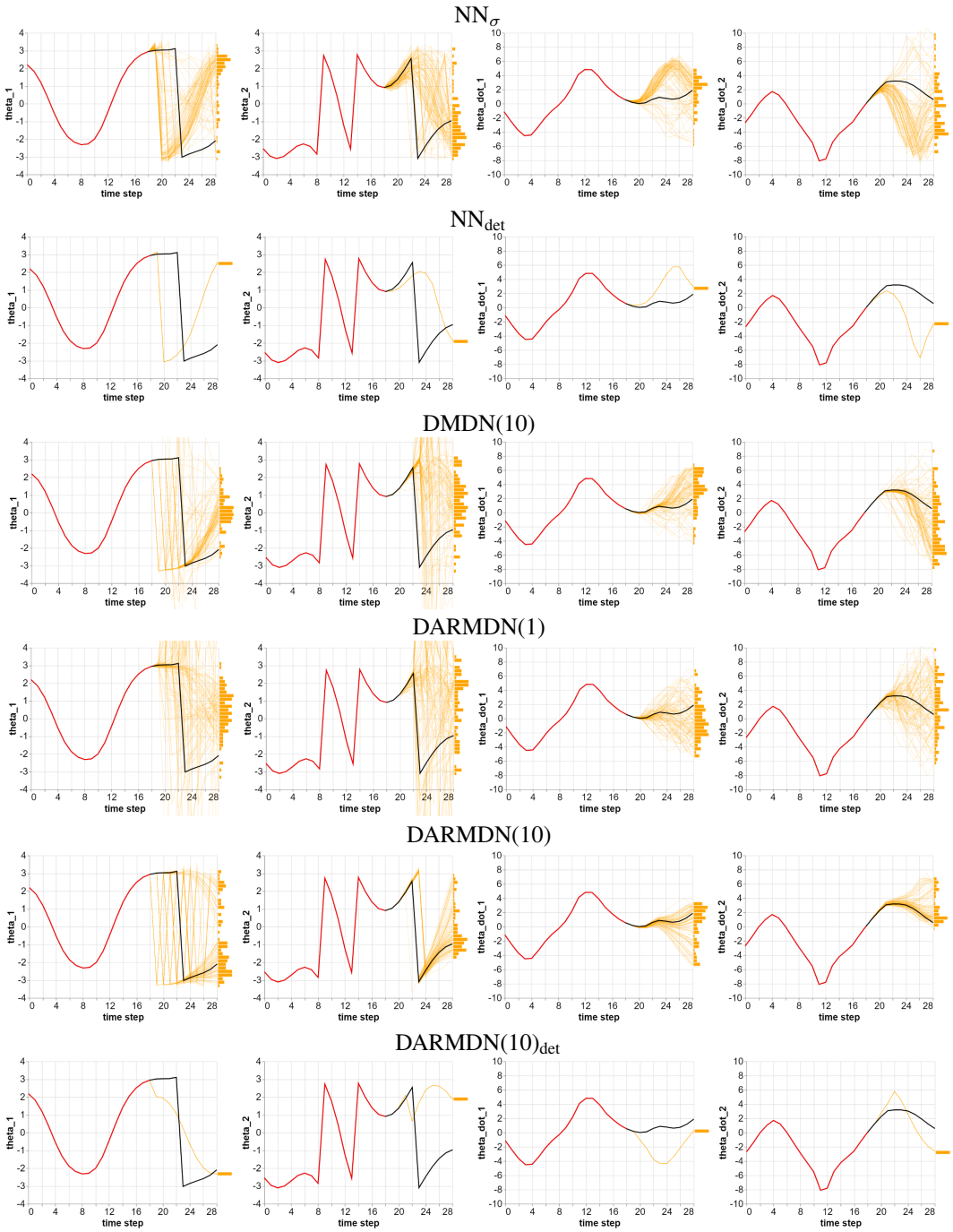


Figure 5: Ground truth and simulation of "futures" by the models trained on the raw angles system. The thick curve is the ground truth, the red segment is past, the black segment is future. System models start generating futures from their posterior predictives at step 18. We show a sample of hundred trajectories and a histogram after ten time steps (orange).