arXiv:1605.07127v1 [stat.ML] 23 May 2016

Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks

Stefan Depeweg
Siemens AG and Technical University of Munich
stefan.depeweg@siemens.com

José Miguel Hernández-Lobato
Harvard University
jmh@seas.harvard.edu

Finale Doshi-Velez
Harvard University
finale@seas.harvard.edu

Steffen Udluft
Siemens AG
steffen.udluft@siemens.com

Abstract

We present an algorithm for model-based reinforcement learning that combines Bayesian neural networks (BNNs) with random roll-outs and stochastic optimization for policy learning. The BNNs are trained by minimizing α-divergences, allowing us to capture complicated statistical patterns in the transition dynamics, e.g. multi-modality and heteroskedasticity, which are usually missed by other common modeling approaches. We illustrate the performance of our method by solving a challenging benchmark where model-based approaches usually fail and by obtaining promising results in a real-world scenario for controlling a gas turbine.

1 Introduction

In model-based reinforcement learning, an agent uses its experience to first learn a model of the environment and then uses that model to reason about what action should be taken next. We consider the case in which the agent observes the current state $\mathbf{s}_t$, takes some action a, and then observes the next state $\mathbf{s}_{t+1}$. The problem of learning the model corresponds then to learning a transition function that determines the value of $\mathbf{s}_{t+1}$ as a function of $\mathbf{s}_t$ and a. More generally, we will desire a stochastic transition function $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a)$ specifying the conditional distribution of $\mathbf{s}_{t+1}$ given $\mathbf{s}_t$ and a.

When the state space is continuous, popular modes for transition functions include Gaussian processes [16, 12, 3], fixed bases such as Laguerre functions [21], and adaptive basis/neural networks [5]. In the latter two cases, a deterministic transition model is generally made stochastic by the addition of some Gaussian noise to the final output. Thus, while these models can capture complex deterministic functions, they are extremely limited in the kinds of stochasticity—or transition noise—that they can express. In particular, in many real-world scenarios stochasticity may often arise due to some unobserved environmental feature that can affect the dynamics in complex ways (such as unmeasured gusts of wind on a boat). Bayesian neural networks (BNNs) are uniquely suited to capture stochastic transition dynamics with complex structure. Recently, numerous works have developed efficient inference approaches for BNNs, including variational methods [2, 6], probabilistic backpropagation [9] and Markov Chain Monte Carlo methods [1, 25].

We take advantage of a very recent inference advance based on α-divergence minimization [10] to learn BNN transition functions that are both scalable and expressive. We focus on the off-policy batch reinforcement learning scenario, in which we are given an initial batch of data from an already-running system and are asked to find a better (ideally near-optimal) policy. Such scenarios are common in real-world industry settings such as turbine control [17], where exploration is usually restricted to avoid possible damage to the system. We propose an algorithm that uses random roll-outs and stochastic optimization for learning such an optimal policy from the predictions of BNNs. The

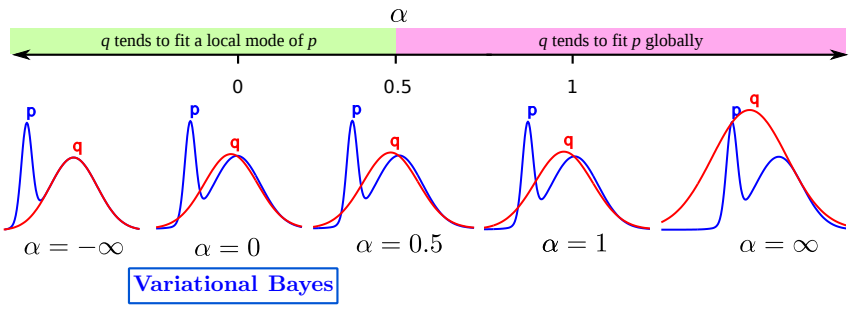

Figure 1: Solution for the minimization of the α-divergence between the posterior p (in blue) and the Gaussian approximation q (in red and unnormalized). Figure source [13].

resulting approach for policy search produces (up to our knowledge) the first model-based solution of a 20-year-old benchmark problem: the Wet-Chicken [20, 8]. Additionally, we also obtain very promising results on a real-world application on controlling gas turbines.

2 Background

Model-Based Reinforcement Learning In this work we consider reinforcement learning problems in which an agent acts in a stochastic environment by sequentially choosing actions over a sequence of time steps, in order to maximise a cumulative reward. We assume that our environment has some true dynamics $T_{\text{true}}(\mathbf{s}_{t+1}|\mathbf{s}, a)$, and we are given some cost function $c(\mathbf{s}_t)$. In the model-based reinforcement learning setting, our goal is to learn an approximation $T_{\text{approx}}(\mathbf{s}_{t+1}|\mathbf{s}, a)$ for the dynamics $T_{\text{true}}(\mathbf{s}_{t+1}|\mathbf{s}_t, a)$ based on collected samples $(\mathbf{s}_t, a, \mathbf{s}_{t+1})$. The agent then tries to solve the control problem in which T_{approx} is assumed to be the true dynamics.

Bayesian Neural Networks. Given data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, formed by feature vectors $\mathbf{x}_n \in \mathbb{R}^D$ and targets $\mathbf{y}_n \in \mathbb{R}^K$, we assume that $\mathbf{y}_n = f(\mathbf{x}_n; \mathcal{W}) + \boldsymbol{\epsilon}_n$, where $f(\cdot; \mathcal{W})$ is the output of a neural net with weights $\mathcal{W}$. The network output is corrupted by additive noise variables $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{\Gamma})$ with diagonal covariance matrix $\boldsymbol{\Gamma}$. The network has L layers, with V_l hidden units in layer l, and $\mathcal{W} = \{\mathbf{W}_l\}_{l=1}^L$ is the collection of $V_l \times (V_{l-1} + 1)$ weight matrices. The $+1$ is introduced here to account for the additional per-layer biases. The activation functions for the hidden layers are rectifiers: $\varphi(x) = \max(x, 0)$. Let $\mathbf{Y}$ be an $N \times K$ matrix with the targets $\mathbf{y}_n$ and $\mathbf{X}$ be an $N \times D$ matrix of feature vectors $\mathbf{x}_n$. The likelihood is then

$$p(\mathbf{Y} \,|\, \mathcal{W}, \mathbf{X}, \boldsymbol{\Gamma}) = \prod_{n=1}^{N} p(\mathbf{y}_n \,|\, \mathcal{W}, \boldsymbol{\Gamma}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(y_{n,k} \,|\, f(\mathbf{x}_n; \mathcal{W}), \boldsymbol{\Gamma}) \,. \tag{1}$$

We specify a Gaussian prior distribution for each entry in each of the weight matrices in $\mathcal{W}$:

$$p(\mathcal{W} \,|\, \lambda) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} \,|\, 0, \lambda) \,, \tag{2}$$

where $w_{ij,l}$ is the entry in the i-th row and j-th column of $\mathbf{W}_l$ and λ is a prior variance. The posterior distribution for the weights $\mathcal{W}$ can then be obtained by applying Bayes' rule: $p(\mathcal{W} \,|\, \mathcal{D}, \boldsymbol{\Gamma}, \lambda) = p(\mathbf{Y} \,|\, \mathcal{W}, \mathbf{X}, \boldsymbol{\Gamma}) p(\mathcal{W} \,|\, \lambda) / p(\mathbf{y} \,|\, \mathbf{X}, \boldsymbol{\Gamma}, \lambda)$, where $p(\mathbf{y} \,|\, \mathbf{X}, \boldsymbol{\Gamma}, \lambda)$ is a normalization constant. Given a new input vector $\mathbf{x}_\star$, we can then make probabilistic predictions for the output $\mathbf{y}_\star$ using the predictive distribution given by

$$p(\mathbf{y}_\star \,|\, \mathbf{x}_\star, \mathcal{D}, \boldsymbol{\Gamma}, \lambda) = \int \mathcal{N}(y_\star \,|\, f(\mathbf{x}_\star; \mathcal{W}), \boldsymbol{\Gamma}) p(\mathcal{W} \,|\, \mathcal{D}, \boldsymbol{\Gamma}, \lambda) \, d\mathcal{W} \,. \tag{3}$$

The exact computation of (3) is intractable in practice and we have to use approximations.

α-divergence minimization. We approximate the exact posterior $p(\mathcal{W}) = p(\mathcal{W} \,|\, \mathcal{D}, \boldsymbol{\Gamma}, \lambda)$ with the factorized Gaussian distribution

$$q(\mathcal{W}) = \left[\prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l} | m_{ij,l}, v_{ij,l}) \right] , \tag{4}$$

where the approximation parameters $m_{ij,l}$, $v_{ij,l}$ are determined by minimizing a divergence (distance) between the true posterior p and the approximation q. After fitting q, we can make probabilistic predictions by replacing p with q in (3). We aim to adjust q by minimizing the α-divergence between p and q [13]:

$$\mathrm{D}_\alpha[p||q] = \frac{1}{\alpha(\alpha-1)}\left(1 - \int p(\mathcal{W})^\alpha q(\mathcal{W})^{(1-\alpha)}\right) d\mathcal{W}\,, \tag{5}$$

which includes a parameter $\alpha \in \mathbb{R}$ that controls the properties of the optimal q. Figure 1 shows for the one-dimensional case that, as α changes from values smaller than 0.5 to values larger than 0.5, q changes from fitting a local mode to covering the whole posterior p. When $\alpha = 0$, the solution obtained is the same as with variational Bayes (VB) [22, 2].

The computation of (5) is infeasible in practice. Instead, we follow [10] and minimize a sum of local α-divergences, with one local α-divergence for each of the N likelihood factors in (1). Since q is Gaussian and $p(\mathcal{W}|\lambda)$ is also Gaussian, we represent q as $q(\mathcal{W}) \propto f(\mathcal{W})^N p(\mathcal{W}|\lambda)$ where f is a Gaussian factor that approximates the geometric mean of the N likelihood factors in (1). We then minimize the sum of local α-divergences given by $\sum_{n=1}^N \mathrm{D}_\alpha[p_n||q]$, where $p_n(\mathcal{W}) \propto p(\mathbf{y}_n \,|\, \mathcal{W}, \mathbf{x}_n, \mathbf{\Gamma}) q(\mathcal{W})/f(\mathcal{W})$ represents the distribution obtained by replacing the n-th copy of f in q with the n-th exact factor in (1).

The energy function

$$E_\alpha(q) = -\log Z(q) - \frac{1}{\alpha}\sum_{n=1}^N \log \mathbf{E}_q\left[\left(\frac{p(\mathbf{y}_n \,|\, \mathcal{W}, \mathbf{x}_n, \mathbf{\Gamma})}{f(\mathcal{W})}\right)^\alpha\right], \tag{6}$$

can be shown to have the same stationary points as the previous sum of local α-divergences [10]. Therefore, in practice we minimize (6), where f is in exponential Gaussian form, that is,

$$f(\mathcal{W}) = \exp\left\{\sum_{l=1}^L \sum_{i=1}^{V_l} \sum_{j=1}^{V_{l-1}+1} \frac{1}{N}\left(\frac{\lambda v_{i,j,l}}{\lambda - v_{i,j,l}} w_{i,j,l}^2 + \frac{m_{i,j,l}}{v_{i,j,l}} w_{i,j,l}\right)\right\} \propto \left[\frac{q(\mathcal{W})}{p(\mathcal{W} \,|\, \lambda)}\right]^{\frac{1}{N}} \tag{7}$$

and $\log Z(q)$ is the logarithm of the normalization constant of the exponential form of q, that is,

$$\log Z(q) = \sum_{l=1}^L \sum_{i=1}^{V_l} \sum_{j=1}^{V_{l-1}+1} \left[\frac{1}{2}\log\left(2\pi v_{i,j,l}\right) + \frac{m_{i,j,l}^2}{v_{i,j,l}}\right]. \tag{8}$$

The hyper-parameters $\mathbf{\Gamma}$ and λ can be tuned by minimizing (6). The scalable optimization of this energy function is done in practice by using using stochastic gradient descent. For this, we subsample the sum over N data points in (6) using minibatches and approximate the expectations over q with an average over K samples drawn from q. We can then use the reparametrization trick [11] to obtain unbiased stochastic gradients from the resulting stochastic approximator to (6).

It can be shown that minimizing (6) when $\alpha \to 0$ is equivalent to adjusting q by VB [10]. Therefore, when we use VB to adjust q we obtain solutions that tend to fit locally only one of the modes in p, as illustrated in Figure 1. This is a well known property of VB. However, as our experiments show, this local fit of VB results in predictive distributions that lack flexibility and are unable to capture complicated patterns, e.g. heteroskedasticity or multi-modality. In model-based reinforcement learning, capturing such complicated patterns in the dynamics can be important for learning optimal control policies. In this work, we empirically show that, by minimizing (6) using $\alpha = 0.5$, we can actually capture the aforementioned complicated patterns with high accuracy. The reason for this is likely to be the fact that $\alpha = 0.5$ achieves a balance between tendencies to fit locally and globally p, as shown in Figure 1. Interestingly, in [10], it is observed that $\alpha = 0.5$ often produces higher test log-likelihood than $\alpha = 0$ or $\alpha = 1$ in regression problems with Bayesian neural networks.

3 Model-based policy search with Bayesian neural networks

We now describe a model-based policy search algorithm that uses the Bayesian neural networks from the previous section. Model-based policy search methods include two key parts [4]. The first part consists in learning a dynamics model from data in the form of state transitions $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$, where $\mathbf{s}_t$ denotes the current state, a_t is the action applied and $\mathbf{s}_{t+1}$ is the resulting state. The second part consists in learning the parameters $\mathcal{W}_\pi$ of a deterministic policy function π that returns the optimal action $a_t = \pi(\mathbf{s}_t; \mathcal{W}_\pi)$ as function of the current state $\mathbf{s}_t$.

We first learn a Bayesian neural network for the dynamics. Afer this, we learn a policy by minimizing the expected cost with respect to the probabilistic predictions of the Bayesian network. The expected cost is obtained by averaging over multiple roll-outs: Given a starting state $\mathbf{s}_0$, we simulate the evolution of the system over a fixed horizon T using the probabilistic predictions of the Bayesian network and the actions produced by the current policy. This procedure allows us to obtain performance estimates for any particular cost function. If model, policy and cost function are differentiable, we are then able to tune $\mathcal{W}_\pi$ by gradient descent over the roll-out average.

In this work, we assume the underlying true dynamics are a stochastic system governed by the predictions of the Bayesian network:

$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) = \int \mathcal{N}(\mathbf{s}_{t+1}|f(\mathbf{s}_t, a_t; \mathcal{W}), \mathbf{\Gamma}) q(\mathcal{W})\, d\mathcal{W}\,, \tag{9}$$

which is obtained by replacing in the right-hand side of (3), $\mathbf{y}_\star$ with $\mathbf{s}_{t+1}$, $\mathbf{x}_\star$ with $\mathbf{s}_t$ and a_t, and the true posterior on $\mathcal{W}$ with its approximation q. In practice, instead of predicting $\mathbf{s}_{t+1}$, we predict $\mathbf{\Delta}_{t+1} = \mathbf{s}_{t+1} - \mathbf{s}_t$ which usually produces better results. We also encode the policy function π using a deterministic neural network parameterized by weights $\mathcal{W}_\pi$.

We assume that the actions are continuous and that we are given a cost function $c(\mathbf{s}_t)$. In the policy search phase, our goal is to minimize the expected cost over a finite horizon T:

$$J(\mathcal{W}_\pi) = \mathbf{E}\left[\sum_{t=0}^{T} c(\mathbf{s}_t)\right]\,. \tag{10}$$

we approximate (10) as follows, by using (9) and replacing a_t with $\pi(\mathbf{s}_t; \mathcal{W}_\pi)$:

$$\begin{aligned}
J(\mathcal{W}_\pi) &= \int \left[\sum_{t=0}^{T} c(\mathbf{s}_t)\right] \left[\prod_{t=1}^{T} p(\mathbf{s}_t|\mathbf{s}_{t-1}, \pi_{\mathcal{W}_\pi}(\mathbf{s}_{t-1}))\right] d\mathbf{s}_0 \cdots d\mathbf{s}_T \\
&= \int \left[\sum_{t=0}^{T} c(\mathbf{s}_t)\right] \left[\prod_{t=1}^{T} \int \mathcal{N}(\mathbf{s}_t|f(\mathbf{s}_{t-1}, \pi_{\mathcal{W}_\pi}(\mathbf{s}_{t-1}); \mathcal{W}_t), \mathbf{\Gamma}) q(\mathcal{W}_t) d\mathcal{W}_t\right] p(\mathbf{s}_0) d\mathbf{s}_0 \cdots d\mathbf{s}_T \\
&= \int \left[c(\mathbf{s}_0) + \sum_{t=1}^{T} c(\mathbf{s}_t^{\{\mathcal{W}_1,\ldots,\mathcal{W}_t\},\{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_t\},\mathcal{W}_\pi})\right] \left[\prod_{t=1}^{T} q(\mathcal{W}_t)\mathcal{N}(\boldsymbol{\epsilon}_t|0, \mathbf{\Gamma}) d\mathcal{W}_t d\boldsymbol{\epsilon}_t\right] p(\mathbf{s}_0)\, d\mathbf{s}_0 \\
&\approx \frac{1}{K}\sum_{k=1}^{K}\left[c(\mathbf{s}_0^k) + \sum_{t=1}^{T} c(\mathbf{s}_t^{\{\mathcal{W}_1^k,\ldots,\mathcal{W}_t^k\},\{\boldsymbol{\epsilon}_1^k,\ldots,\boldsymbol{\epsilon}_T^k\},\mathcal{W}_\pi})\right]\,.
\end{aligned} \tag{11}$$

In the first line in this expression, we use the assumption that the dynamics are Markovian with respect to the current state and the current action. In the second line we use (9). In the third line, $\mathbf{s}_t^{\{\mathcal{W}_1,\ldots,\mathcal{W}_t\},\{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_t\},\mathcal{W}_\pi}$ is the state that is obtained at time t in a roll-out generated by using a policy with parameters $\mathcal{W}_\pi$ and a sequence of deterministic transition functions given by neural networks with sets of weights $\mathcal{W}_1, \ldots, \mathcal{W}_t$ and with noise values $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_t$ added to their output. In the last line we have approximated the integration with respect to $\mathcal{W}_1, \ldots, \mathcal{W}_T$, $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_T$ and $\mathbf{s}_0$ by averaging over K samples of these variables. To sample $\mathbf{s}_0$, we draw this variable uniformly at random from the available transitions $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$. Note that in the second line of (11) we sample a different $\mathcal{W}_t$ for each t instead of sampling a single $\mathcal{W}$ that is reused at each time step. By doing so, we assume that the true dynamics have complicated stochastic patterns that no single $\mathcal{W}$ can describe.

The expected cost (10) can then be optimized by stochastic gradient descent using the gradients of the Monte Carlo approximation used in (11). Algorithm 1 computes this Monte Carlo approximation. The gradients can then be obtained using automatic differentiation tools such as theano. Note that, in Algorithm 1, instead of sampling a new $\mathcal{W}_t^k$ at each roll-out step, we only sample K different values for $\mathcal{W}$ at the beginning of the roll-out. We then choose randomly one of these K samples at each roll-out step. This increases the efficiency of the algorithm implementation in theano and, when using GPUs, it reduces the cost of transferring the random samples for $\mathcal{W}$ to the GPU. Note that the outer loop over K can be trivially parallelized because each roll-out is independent of each other.

4 Experiments

4.1 Flexibility of predictions of BNNs

We evaluate the predictive performance of Bayesian neural networks trained by minimizing (6) with $\alpha = 0.5$ in two simple regression problems. The first one is characterized by a bimodal predictive

Algorithm 1 Model-based policy search with probabilistic neural networks.

Input: $\mathcal{D} = \{\mathbf{s}_n, a_n, \mathbf{\Delta}_n\}$ for $n \in 1..N$
Fit $q(\mathcal{W})$ *and* $\mathbf{\Gamma}$ *by optimizing* (6).
function UNFOLD($\mathbf{s}_0$)
  sample$\{\mathcal{W}_1, .., \mathcal{W}_K\}$ *from* $q(\mathcal{W})$
  $C \leftarrow Kc(\mathbf{s}_0)$
  for $k = 1 : K$ **do**
    for $t = 0 : T$ **do**
      $j \sim \text{Unif}(\{1, .., K\})$
      $\mathbf{\Delta}_t \leftarrow f(\mathbf{s}_t, \pi(\mathbf{s}_t; \mathcal{W}_\pi); \mathcal{W}_j)$
      $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{\Gamma})$
      $\mathbf{s}_{t+1} \leftarrow \mathbf{s}_t + \mathbf{\Delta}_t + \boldsymbol{\epsilon}$
      $C \leftarrow C + c(\mathbf{s}_{t+1})$
      $\mathbf{s}_t \leftarrow \mathbf{s}_{t+1}$
  return C/K
Fit $\mathcal{W}_\pi$ *by optimizing* $\frac{1}{N}\sum_{n=1}^{N}$ UNFOLD($\mathbf{s}_n$)

Table 1: Results on bi-modal problem

Method	RMSE	Log-likelihood
$\alpha = 0.5$	5.23	**-2.11**
$\alpha = 10^{-6}$	**5.10**	-3.05

Table 2: Results on heteroskedastic problem

Method	RMSE	Log-likelihood
$\alpha = 0.5$	**1.87**	**-1.73**
$\alpha = 10^{-6}$	1.88	-2.05

Table 3: Results on WetChicken benchmark

Method	Reward	Log-likelihood
$\alpha = 0.5$	**2.63**	**0.10**
$\alpha = 10^{-6}$	2.25	-2.90

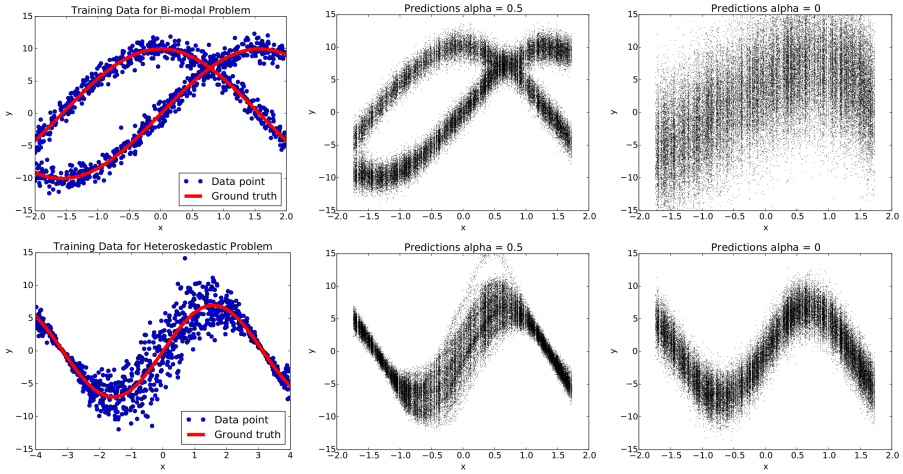

Figure 2: Results on the toy problem. Left, training data (blue points) and ground truth functions (red). Middle, predictions generated with $\alpha = 0.5$. Right, predictions generated with $\alpha = 10^{-6}$.

distribution. The second is characterized by a heteroskedastic predictive distribution. In the latter case the magnitude of the noise in the output changes as a function of the input.

In the first problem $x \in [-2, 2]$ and y is obtained as $y = 10 \sin x + \epsilon$ with probability 0.5 and $y = 10 \cos x + \epsilon$, otherwise, where $\epsilon \sim \mathcal{N}(0, 1)$ and ϵ is independent of x. The plot in the top left of Figure 2 shows a training dataset obtained by sampling 1000 values of x uniformly at random. The plot clearly shows that the distribution of y for a particular x is bimodal. In the second problem $x \in [-4, 4]$ and y is obtained as $y = 7 \sin x + 3|\cos(x/2)|\epsilon$. The plot in the bottom left of Figure 2 shows a training dataset obtained with 1000 values of x uniformly at random. The plot clearly shows that the distribution of y is heteroskedastic, with a noise variance that is a function of x.

We fitted a neural network with 2 hidden layers and 50 hidden units per layer using Adam with its default parameter values, except for the learning rate, which takes value 0.01 in the first problem and 0.002 in the second problem. We used minibatches of size 250 and 1000 training epochs. To approximate the expectations in 6, we draw $K = 50$ samples from q. We only updated the randomness in the posterior samples from q once every 10 minibatches. This reduced the variance in the stochastic gradients and helped the method converge to lower values of the energy. We compared with $\alpha = 10^{-6}$, which is equivalent to variational Bayes.

The plots in the middle of Figure 2 show the predictions obtained with $\alpha = 0.5$. In this case, the predictive distribution is able to capture the bimodality in the first problem and the heteroskedasticity

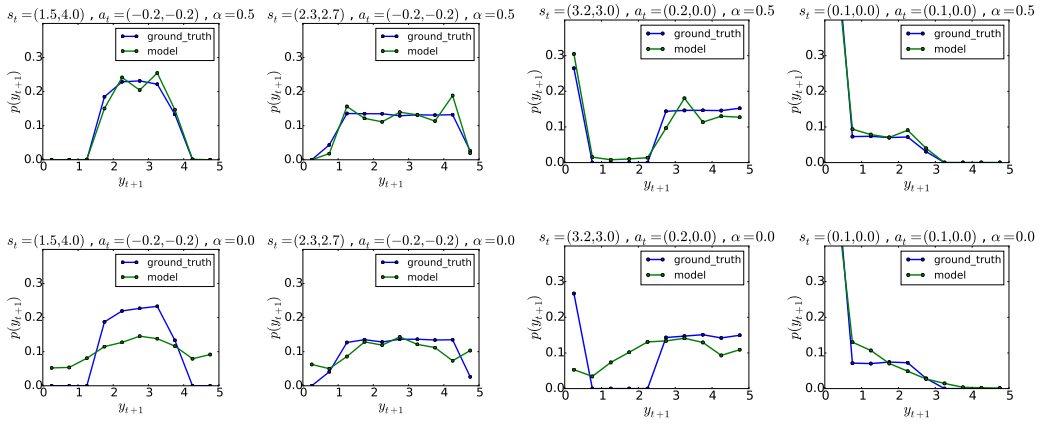

Figure 3: From left to right, predictive distribution of a BNN vs. ground-truth for different values of y_t, one column for each value of y_t. Top row: predictive distribution of a BNN with $\alpha = 0.5$. Bottom row: predictions of a BNN with $\alpha = 10^{-6}$.

pattern in the second problem. The plots in the right of Figure 2 show the predictions obtained with $\alpha = 10^{-6}$, which converges to suboptimal solutions in which the predictive distribution has a single mode (in the first problem) or is homoskedastic (in the second problem). Tables 1 and 2 shows the average test RMSE and log-likelihood obtained by each method on each problem. In the first problem, $\alpha = 10^{-6}$ focuses on minimizing the test error, while $\alpha = 0.5$ produces better log-likelihood values. These results show that Bayesian neural networks trained with $\alpha = 0.5$ can model very complicated predictive distributions, which may be multimodal and heteroskedastic.

4.2 Wet-Chicken benchmark

We now evaluate the performance of policies learned using the approach from section 3 with $\alpha = 0.5$. For this, we consider the Wet-Chicken benchmark [20, 8], a challenging problem for model-based policy search that presents both bi-modal and heteroskedastic transition dynamics. We will use the two-dimensional version and extend it to the continuous case.

In this problem, a canoeist is paddling on a two-dimensional river. The canoeist's position at time t is (x_t, y_t). The river has width $w = 5$ and length $l = 5$ with a waterfall at the end, that is, at $y_t = l$. The canoeist wants to move as close to the waterfall as possible because at time t he gets reward $r_t = y_t$. However, going beyond the waterfall boundary makes the canoeist fall down, having to start back again at the origin $(0, 0)$. At time t the canoeist can choose an action $(a_{t,x}, a_{t,y}) \in [-1, 1]^2$ that represents the direction and magnitude of its paddling. The river dynamics have stochastic turbulences s_t and drift v_t that depend on the canoeist position on the x axis. The larger x_t, the larger the drift and the smaller x_t, the larger the turbulences. Therefore, x_t determines the trade-off between stochasticity and drift. The underlying dynamics are given by the following system of equations. The drift and the turbulence magnitude are given by $v_t = 3x_t w^{-1}$ and $s_t = 3.5 - v_t$, respectively. The new location (x_{t+1}, y_{t+1}) is given by the current location (x_t, y_t) and current action $(a_{t,x}, a_{t,y})$ using

$$x_{t+1} = \begin{cases} 0 & \text{if} \quad x_t + a_{t,x} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ w & \text{if} \quad x_t + a_{t,x} > w \\ x_t + a_{t,x} & \text{otherwise} \end{cases}, \qquad y_{t+1} = \begin{cases} 0 & \text{if} \quad \hat{y}_{t+1} < 0 \\ 0 & \text{if} \quad \hat{y}_{t+1} > l \\ \hat{y}_{t+1} & \text{otherwise} \end{cases}, \tag{12}$$

where $\hat{y}_{t+1} = y_t + (a_{t,y} - 1) + v_t + s_t \tau_t$ and $\tau_t \sim \text{Unif}([-1, 1])$ is a random variable that represents the current turbulence. These dynamics result in rich transition distributions depending on the position as illustrated by the plots in Figure 3. As the canoeist moves closer to the waterfall, the distribution for the next state becomes increasingly bi-modal (see plots in the third column) because when he is close to the waterfall, the change in the current location can be large, if the canoeist falls down the waterfall and starts again at $(0, 0)$. The distribution may also be truncated uniform for states close to the borders (see plots in the fourth column). Furthermore the system has heteroskedastic noise (see plots

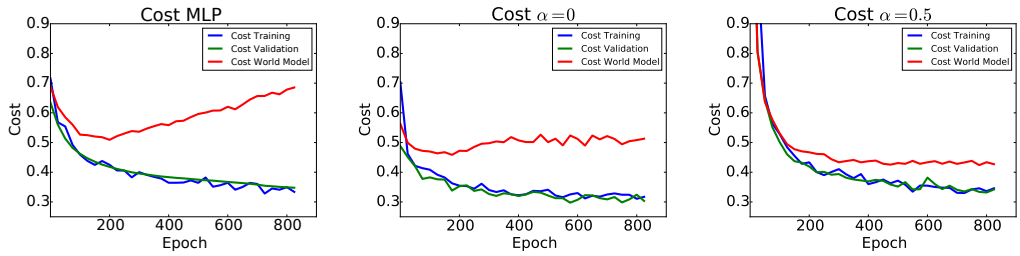

Figure 4: Costs of the policies obtained using different models for the dynamics. Blue curves show cost over time on the training set, green curve on the validation set and red curve when evaluating the policy on the world model. Left: results for MLP. Middle: results for Bayesian network with $\alpha = 0$. Right: results for Bayesian network with $\alpha = 0.5$.

in the first and second columns). The smaller the value of x_t the higher the noise variance. Because of these properties, the Wet-Chicken problem is especially difficult for model-based reinforcement learning methods. To our knowledge it has only been solved using model-free approaches after a discretization of the state and action sets [8].

To solve the problem, we train Bayesian neural networks with 2 hidden layers and 20 hidden units per layer using $\alpha = 0.5$ and $\alpha = 10^{-6}$ on a dataset of size $20,000$. The resulting predictive distributions for x_{t+1} are shown in the top row of Figure 3 for specific choices of (x_t, y_t) and $(a_{x,t}, a_{y,t})$. These plots show that $\alpha = 0.5$ produces distributions that are very close to the ground truth, capturing the bi-modality and heteroskedasticity of the problem. By contrast $\alpha = 10^{-6}$ fails to capture these patterns. The test-loglikelihood, shown in Table 3, also shows that $\alpha = 0.5$ performs better in this problem.

In the next step we train a policy using Algorithm 1. We use a horizon of size $T = 5$ and optimize the policy network for 100 epochs and averaging over $K = 20$ samples in each gradient update. Table 3 shows the average reward obtained when evaluating the policies generated with $\alpha = 0.5$ and $\alpha = 10^{-6}$ on the true dynamics. In this case, $\alpha = 0.5$ performs significantly better.

4.3 Application to the control of a gas turbine

We now use Algorithm 1 to learn a controller for a gas turbine [17]. The data consists of 40,000 observations of a 30 dimensional time-series. We are also given a cost function that evaluates the performance of the current state of the turbine. The features in the time-series are grouped into three sets: a set of environmental variables E_t (e.g. temperature and measurements from sensors in the turbine), a set of variables relevant for the cost function N_t (e.g. the turbines current pollutant emission) and a set of variables that can be manipulated A_t. We assume that the effect of the actions A_t is delayed; it may take up to 5 time steps to take effect on N_t. The same occurs with the environmental variables. We also assume that the actions A_t do not affect the environmental variables. The resulting transition model is $N_t = f(E_{t-5}, .., E_t, A_{t-5}, ..A_t)$. That is, based on past actions and states of the environment, we try to learn a model for the reward-relevant variables.

We evaluate the performance of a particular policy by simulating a task with partial observability. For this, we train a ground truth model f_{truth} on all the available data. We call this the world model. To make fair comparisons, the world model is a non-Bayesian neural network with deterministic weights. After training, we make the world model stochastic by adding output noise that is Gaussian and uncorrelated. The noise variances are fixed by maximum likelihood on some validation data. We then train a Bayesian neural network with $\alpha = 0.5$ on the same data as the world model, but with only half of the environmental features E_t as inputs. After this, we use Algorithm 1 to learn a policy that is evaluated on the stochastic world model by doing roll-outs. The environmental variables E_t change very slowly but are impossible to predict. For simplicity, when we do roll-outs we assume a constant model $E_t = E_0$ for the environmental variables.

The aim in this learning task with partial observability of the environmental variables is to learn a policy that is robust to noise in the dynamics. This noise would be originated by latent factors that cannot be controlled, such as the missing environmental variables.

The world model and the Bayesian neural network have two hidden layers with 100 hidden units. The policy network is a MLP with two hidden layers including 20 hidden units. For policy training and world-model evaluation we do a roll-out with horizon $T = 20$. For learning the policy we use minibaches of size 10 and draw $K = 10$ samples from q. The plot in the right of Figure 4 shows the performance of the resulting policy as a function of the training epochs. The plots in the left and middle of this figure show results for a non-Bayesian MLP and a Bayesian neural network with $\alpha = 10^{-6}$, respectively. The red curves in these plots corresponds to the policy performance in the world model, which is never available to the methods. We can observe that the MLP, and the $\alpha = 10^{-6}$ approach to a lesser extent, produce poor results. While training and validation performance continuously decrease, their performance in the world-model deteriorates. A validation set is in this setting a set of starting states held out during policy training. The Bayesian network trained with $\alpha = 0.5$ produces more robust results, with training, validation and world costs that correlate with each other.

4.4 Comparision with Gaussian processes

We now compare the predictions of Bayesian networks (BNNs) with $\alpha = 0.5$ with those of Gaussian processes (GPs). We considered the data for the Wet-Chicken and Turbine problems. We splited the available data into training and test sets with 75% and 25% of the data. For scalability, we used sparse GPs based on the FITC approximation [19], using a total of 150 inducing points. Table 4 shows the average test RMSE for each method. Overall, BBNs with $\alpha = 0.5$ obtain higher test log-likelihood, which shows that they are better at capturing the statistical properties of the true dynamics.

Table 4: Comparison with Gaussian processes

	Wet-Chicken		**Turbine**	
Method	RMSE	LL	RMSE	LL
$\alpha = 0.5$	1.19	**-0.05**	**0.30**	**0.71**
Sparse GP	**1.15**	-1.73	0.32	0.45

5 Related work

One prominent class of policy search algorithms are policy gradient techniques [15]. The main focus nowadays is an on-line model-free context [14], whereas traditionally, deterministic model-based techniques have been used in discrete state space [23]. Our work can be seen as a monte-carlo model-based policy gradient technique in continuous stochastic systems. Similar work was done using Gaussian processes [3] or with recurrent neural networks [18] . A gaussian process approach, while restricted to a Gaussian state distribution, allows propagating beliefs over the roll-out procedure. More recent work [7] augments a model-free learning procedure with data generated from model-based roll-outs.

There have been few experiments using bayesian neural network for reinforcement learning in general. In [2] a thompson sampling approach is used for a contextual bandits problem. The work in [24] combines variational auto-encoder with stochastic optimal control for visual data. Compared to our approach the first contribution focusses on the exploration/exploitation dilemma, whereas the second one uses a stochastic optimal control approach to solve the learning problem, whereas our work seeks to find a parametrized policy.

6 Conclusion and future work

We have presented a novel algorithm for model-based reinforcement learning that combines two key elements. The first one is a model for the dynamics given by a Bayesian neural network that is trained by minimizing α-divergences. We have shown that these probabilistic networks can capture complicated statistical patterns in the transition dynamics, e.g. multi-modality and heteroskedasticity. Minimizing α-divergences seems to be key for capturing such complicated patterns, since other more common approaches such as variational Bayes often produce less flexible predictive distributions. The second key element is an algorithm that uses random roll-outs and stochastic optimization for learning a parameterized policy. The combination of these two key elements has allowed us to solve a challenging benchmark problem where model-based approaches usually fail and has also shown promising results in a real-world scenario for controlling a gas turbine.

As future work we consider finding structured and interpretable policy representations. Especially for applications where one would ideally monitor and interpret the behavior of a given policy. Another line of research would be to explore how our findings could be applied in a model-free scenario.

Acknowledgements

José Miguel Hernández-Lobato acknowledges support from the Rafael del Pino Foundation. The authors would like to thank Ryan P. Adams, Siegmund Duell, Hans-Georg Zimmermann, Matthew J. Johnson and David Duvenaud for helpful discussions.

References

[1] A. K. Balan, V. Rathod, K. P. Murphy, and M. Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3420–3428, 2015.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1613–1622, 2015.

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

[4] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[5] A. Draeger, S. Engell, and H. Ranke. Model predictive control using neural networks. *Control Systems, IEEE*, 15(5):61–66, 1995.

[6] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[7] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*, 2016.

[8] A. Hans and S. Udluft. Efficient uncertainty propagation for reinforcement learning with limited data. In *Artificial Neural Networks–ICANN 2009*, pages 70–79. Springer, 2009.

[9] J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. *arXiv preprint arXiv:1502.05336*, 2015.

[10] J. M. Hernández-Lobato, Y. Li, M. Rowland, D. Hernández-Lobato, T. Bui, and R. E. Turner. Black-box α-divergence minimization. *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, arXiv preprint arXiv:1511.03243*, 2016.

[11] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

[12] J. Ko, D. J. Klein, D. Fox, and D. Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 742–747. IEEE, 2007.

[13] T. Minka et al. Divergence measures and message passing. Technical report, Technical report, Microsoft Research, 2005.

[14] J. Peters and S. Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006.

[15] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[16] C. E. Rasmussen, M. Kuss, et al. Gaussian processes in reinforcement learning. In *NIPS*, volume 4, page 1, 2003.

[17] A. M. Schaefer, D. Schneegass, V. Sterzing, and S. Udluft. A neural reinforcement learning approach to gas turbine control. In *Neural Networks, 2007. IJCNN 2007*, pages 1691–1696. IEEE, 2007.

[18] A. M. Schaefer, S. Udluft, and H.-G. Zimmermann. The recurrent control neural network. In *ESANN*, pages 319–324. Citeseer, 2007.

[19] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2005.

[20] V. Tresp. The wet game of chicken. *Siemens AG, CT IC 4, Technical Report*, 1994.

[21] B. Wahlberg. System identification using laguerre models. *Automatic Control, IEEE Transactions on*, 36(5):551–562, 1991.

[22] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

[23] X. Wang and T. G. Dietterich. Model-based policy gradient reinforcement learning. In *ICML*, pages 776–783, 2003.

[24] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2728–2736, 2015.

[25] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.