Artificial Intelligence 334 (2024) 104132

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

Probabilistic reach-avoid for Bayesian neural networks [☆]

Matthew Wicker [a,*], Luca Laurenti [b], Andrea Patane [a], Nicola Paoletti [c], Alessandro Abate [a], Marta Kwiatkowska [a]

[a] Department of Computer Science, University of Oxford, Oxford, UK
[b] Delft Center for Systems and Control (DCSC), TU Delft, Delft, Netherlands
[c] Department of Informatics, King's College London, London, UK

A R T I C L E I N F O

Keywords:
Reinforcement learning
Formal verification
Certified control synthesis
Bayesian neural networks
Safety
Reach-while-avoid

A B S T R A C T

Model-based reinforcement learning seeks to simultaneously learn the dynamics of an unknown stochastic environment and synthesise an optimal policy for acting in it. Ensuring the safety and robustness of sequential decisions made through a policy in such an environment is a key challenge for policies intended for safety-critical scenarios. In this work, we investigate two complementary problems: first, computing reach-avoid probabilities for iterative predictions made with dynamical models, with dynamics described by Bayesian neural network (BNN); second, synthesising control policies that are optimal with respect to a given reach-avoid specification (reaching a "target" state, while avoiding a set of "unsafe" states) and a learned BNN model. Our solution leverages interval propagation and backward recursion techniques to compute lower bounds for the probability that a policy's sequence of actions leads to satisfying the reach-avoid specification. Such computed lower bounds provide safety certification for the given policy and BNN model. We then introduce control synthesis algorithms to derive policies maximizing said lower bounds on the safety probability. We demonstrate the effectiveness of our method on a series of control benchmarks characterized by learned BNN dynamics models. On our most challenging benchmark, compared to purely data-driven policies the optimal synthesis algorithm is able to provide more than a four-fold increase in the number of certifiable states and more than a three-fold increase in the average guaranteed reach-avoid probability.

1. Introduction

The capacity of deep learning to approximate complex functions makes it particularly attractive for inferring process dynamics in control and reinforcement learning problems [56]. In safety-critical scenarios where the environment and system state are only partially known or observable (e.g., a robot with noisy actuators/sensors), Bayesian models have recently been investigated as a safer alternative to standard, deterministic, Neural Networks (NNs): the uncertainty estimates of Bayesian models can be propagated through the system decision pipeline to enable safe decision making despite unknown system conditions [15,20,46]. In particular, *Bayesian Neural Networks* (BNNs) retain the same advantages of NNs (relative to their approximation capabilities) and also enable reasoning about uncertainty in a principled probabilistic manner [49,51], making them very well-suited to tackle safety-critical problems.

☆ This article belongs to Special Issue: Risk-Aware Autonomy.
* Corresponding author.
E-mail addresses: mattrwicker@gmail.com, matthew.wicker@cs.ox.ac.uk (M. Wicker).

https://doi.org/10.1016/j.artint.2024.104132
Received 15 April 2022; Received in revised form 24 September 2023; Accepted 3 April 2024

Available online 17 April 2024

In problems of sequential planning, time-series forecasting, and model-based reinforcement learning, evaluating a model with respect to a control policy (or strategy) requires making several predictions that are mutually dependent across time [19,42]. While multiple models can be learned for each time step, a common setting is for these predictions to be made iteratively by the same machine learning model [34], where the state of the predicted model at each step is a function of the model state at the previous step and possibly of an action (from the policy). We refer to this setting as *iterative predictions*.

Unfortunately, performing iterative predictions with BNN models poses several practical issues. In facts, BNN models output probability distributions, so that at each successive timestep the BNN needs to be evaluated over a probability distribution, rather than a fixed input point – thus posing the problem of successive predictions over a stochastic input. Even when the posterior distribution of the BNN weights is inferred using analytical approximations, the deep and non-linear structure of the network makes the resulting predictive distribution analytically intractable [51]. In iterative prediction settings, the problem is compounded and exacerbated by the fact that one would have to evaluate the BNN, sequentially, over a distribution that cannot be computed analytically [20]. Hence, computing sound, formal bounds on the probability of BNN-based iterative predictions remains an open problem. Such bounds would enable one to provide safety guarantees over a given (or learned) control policy, which is a necessary precondition before deploying the policy in a real-world environment [55,60].

In this paper, we develop a new method for the computation of probabilistic guarantees for iterative predictions with BNNs over *reach-avoid* specifications. A reach-avoid specification, also known as constrained reachability [57], requires that the trajectories of a dynamical system reach a goal/target region over a given (finite) time horizon, whilst avoiding a given set of states that are deemed “unsafe”. Probabilistic reach-avoid is a key property for the formal analysis of stochastic processes [1], underpinning richer temporal logic specifications: its computation is the key component for probabilistic model checking algorithms for various temporal logics such as PCTL, csLTL, or BLTL [18,40].

Even though the exact computation of reach-avoid probabilities for iterative prediction with BNNs is in general not analytically possible, with our method, we can derive a guaranteed (conservative) lower bound by solving a backward iterative problem obtained via a discretisation of the state space. In particular, starting from the final time step and the goal region, we back-propagate the probability lower bounds for each discretised portion of the state space. This backwards reachability approach leverages recently developed bound propagation techniques for BNNs [63]. In addition to providing guarantees for a given policy, we also devise methods to synthesise policies that are maximally certifiable, i.e., that maximize the lower bound of the reach-avoid probability. We first describe a numerical solution that, by using dynamic programming, can synthesize policies that are maximally safe. Then, in order to improve the scalability of our approach, we present a method for synthesizing approximately optimal strategies parametrised as a neural network. While our method does not yet scale to state-of-the-art reinforcement learning environments, we are able to verify and synthesise challenging non-linear control case studies.

We validate the effectiveness of our certification and synthesis algorithms on a series of control benchmarks. Our certification algorithm is able to produce non-trivial safety guarantees for each system that we test. On each proposed benchmark, we also show how our synthesis algorithm results in actions whose safety is significantly more certifiable than policies derived via deep reinforcement learning. Specifically, in a challenging planar navigation benchmark, our synthesis method results in policies whose certified safety probabilities are eight to nine times higher than those for learned policies.

We further investigate how factors like the choice of approximate inference method, BNN architecture, and training methodology affect the quality of the synthesised policy. In summary, this paper makes the following contributions:

- We show how probabilistic reach-avoid for iterative predictions with BNNs can be formulated as the solution of a backward recursion.
- We present an efficient certification framework that produces a lower bound on probabilistic reach-avoid by relying on convex relaxations of the BNN model and said recursive problem definition.
- We present schemes for deriving a maximally certified policy (i.e., maximizing the lower bound on safety probability) with respect to a BNN and given reach-avoid specification.
- We evaluate our methodology on a set of control case studies to provide guarantees for learned and synthesized policies and conduct an empirical investigation of model-selection choices and their effect on the quality of policies synthesised by our method.

A previous version of this work [65] has been presented at the thirty-seventh Conference on Uncertainty in Artificial Intelligence. Compared to the conference paper, in this work, we introduce several new contributions. Specifically, compared to Wicker et al. [65] we present novel algorithms for the synthesis of control strategies based on both a numerical method and a neural network-based approach. Moreover, the experimental evaluation has been consistently extended, to include, among others, an analysis of the role of approximate inference and NN architecture on safety certification and synthesis, as well as an in-depth analysis of the scalability of our methods. Further discussion of related works can be found in Section 7.

2. Bayesian neural networks

In this work, we consider fully-connected neural network (NN) architectures $f^w : \mathbb{R}^m \to \mathbb{R}^n$ parametrised by a vector $w \in \mathbb{R}^{n_w}$ containing all the weights and biases of the network. Given a NN f^w composed by L layers, we denote by $f^{w,1}, ..., f^{w,L}$ the layers of f^w and we have that $w = \left(\{W_i\}_{i=1}^L\right) \cup \left(\{b_i\}_{i=1}^L\right)$, where W_i and b_i represent weights and biases of the i-th layer of f^w. For $x \in \mathbb{R}^n$ the output of layer $i \in \{1, ..., L\}$ can be explicitly written as $f^{w,i}(x) = a(W_i f^{w,i-1}(x) + b_i)$ with $f^{w,1}(x) = a(W_1 x + b_1)$, where $a : \mathbb{R} \to \mathbb{R}$

is the activation function. We assume that a is a continuous monotonic function, which holds for the vast majority of activation functions used in practice such as sigmoid, ReLu, and tanh [28]. This guarantees that f^w is a continuous function.

Bayesian Neural Networks (BNNs), denoted by $f^{\mathbf{w}}$, extend NNs by placing a prior distribution over the network parameters, $p_{\mathbf{w}}(w)$, with $\mathbf{w}$ being the vector of random variables associated to the parameter vector w. Given a dataset $\mathcal{D}$, training a BNN on $\mathcal{D}$ requires to compute posterior distribution, $p_{\mathbf{w}}(w|\mathcal{D})$, which can be computed via Bayes' rule [51]. Unfortunately, because of the non-linearity introduced by the neural network architecture, the computation of the posterior is generally intractable. Hence, various approximation methods have been studied to perform inference with BNNs in practice. Among these methods, we consider Hamiltonian Monte Carlo (HMC) [51], and Variational Inference (VI) [13]. In our experimental evaluation in Section 6.3 we employ both HMC and VI.

Hamiltonian Monte Carlo (HMC) HMC proceeds by defining a Markov chain whose invariant distribution is $p_{\mathbf{w}}(w|\mathcal{D})$, and relies on Hamiltonian dynamics to speed up the exploration of the space. Differently from VI discussed below, HMC does not make any parametric assumptions on the form of the posterior distribution and is asymptotically correct. The result of HMC is a set of samples that approximates $p_{\mathbf{w}}(w|\mathcal{D})$. We refer interested readers to [35,52] for further details.

Variational inference (VI) VI proceeds by finding a Gaussian approximating distribution $q(w) \sim p_{\mathbf{w}}(w|\mathcal{D})$ in a trade-off between approximation accuracy and scalability. The core idea is that $q(w)$ depends on some hyperparameters that are then iteratively optimized by minimizing a divergence measure between $q(w)$ and $p_{\mathbf{w}}(w|\mathcal{D})$. Samples can then be efficiently extracted from $q(w)$. See [13,38] for recent developments in variational inference in deep learning.

3. Problem formulation

Given a trained BNN $f^{\mathbf{w}}$ we consider the following discrete-time stochastic process given by iterative predictions of the BNN:

$$\mathbf{x}_k = f^{\mathbf{w}}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{v}_k, \quad \mathbf{u}_k = \pi_k(\mathbf{x}_k), \quad k \in \mathbb{N}_{>0}, \tag{1}$$

where $\mathbf{x}_k$ is a random variable taking values in $\mathbb{R}^n$ modelling the state of System (1) at time k, $\mathbf{v}_k$ is a random variable modelling an additive noise term with stationary, zero-mean Gaussian distribution $\mathcal{N}(0, \sigma^2 \cdot I)$, where I is the identity matrix of size $n \times n$. $\mathbf{u}_k$ represents the action applied at time k, selected from a compact set $\mathcal{U} \subset \mathbb{R}^c$ by a (deterministic) feedback Markov strategy (a.k.a. policy, or controller) $\pi : \mathbb{R}^n \times \mathbb{N} \to \mathcal{U}$.[1]

The model in Eqn. (1) is commonly employed to represent noisy dynamical models driven by a BNN and controlled by the policy π [20]. In this setting, $f^{\mathbf{w}}$ defines the transition probabilities of the model and, correspondingly, $p(\bar{x}|(x,u),\mathcal{D})$ is employed to describe the *posterior predictive distribution*, namely the probability density of the model state at the next time step being $\bar{x}$, given that the current state and action are (x,u), as:

$$p(\bar{x}|(x,u),\mathcal{D}) = \int\limits_{\mathbb{R}^{n_w}} \mathcal{N}(\bar{x} \mid f^w(x,u), \sigma^2 \cdot I) p_{\mathbf{w}}(w|\mathcal{D}) dw, \tag{2}$$

where $\mathcal{N}(\cdot \mid f^w(x,u), \sigma^2 \cdot I)$ is the Gaussian likelihood induced by noise $\mathbf{v}_k$ and centred at the NN output [51].

Observe that the posterior predictive distribution induces a probability density function over the state space. In iterative prediction settings, this implies that at each step the state vector $\mathbf{x}_k$ fed into the BNN is a random variable. Hence, a N-step *trajectory* of the dynamic model in Eqn (1) is a sequence of states $x_0, ..., x_N \in \mathbb{R}^n$ sampled from the predictive distribution. As a consequence, a principled propagation of the BNN uncertainty through consecutive time steps poses the problem of predictions over stochastic inputs. In Section 4.1 we will tackle this problem for the particular case of reach-avoid properties, by designing a backward computation scheme that starts its calculations from the goal region, and proceeds according to Bellman iterations [11].

We remark that $p(\bar{x}|(x,u),\mathcal{D})$ is defined by marginalizing over $p_{\mathbf{w}}(w|\mathcal{D})$, hence, the particular $p(\bar{x}|(x,u),\mathcal{D})$ depends on the specific approximate inference method employed to estimate the posterior distribution. As such, the results that we derive are valid w.r.t. a specific BNN posterior.

Probability measure For an action $u \in \mathbb{R}^c$, a subset of states $X \subseteq \mathbb{R}^n$ and a starting state $x \in \mathbb{R}^n$, we call $T(X|x,u)$ the *stochastic kernel* associated (and equivalent [1]) to the dynamical model of Equation (1). Namely, $T(X|x,u)$ describes the one-step transition probability of the model of Eqn. (1) and is defined by integrating the predictive posterior distribution with input (x,u) over X, as:

$$T(X|x,u) = \int\limits_{X} p(\bar{x}|(x,u),\mathcal{D}) d\bar{x}. \tag{3}$$

In what follows, it will be convenient at times to work over the space of parameters of the BNN. To do so, we can re-write the stochastic kernel by combining Equations (2) and (3) and applying Fubini's theorem [21] to switch the integration order, thus obtaining:

[1] We can limit ourselves to consider deterministic Markov strategies as they are optimal in our setting [1,11]. Also, in the following, we denote with π the time-varying policy described, at each step k, by policy $\pi_k : \mathbb{R}^n \to \mathcal{U}$.

$$T(X|x,u) = \int_{\mathbb{R}^{n_w}} \left[\int_X \mathcal{N}(\bar{x}|f^w(x,u), \sigma^2 \cdot I) d\bar{x} \right] p_{\mathbf{w}}(w|\mathcal{D}) dw. \tag{4}$$

From this definition of T it follows that, under a strategy π and for a given initial condition x_0, $\mathbf{x}_k$ is a Markov process with a well-defined probability measure Pr uniquely generated by the stochastic kernel T [11, Proposition 7.45] and such that for $X_0, X_k \subseteq \mathbb{R}^n$:

$$\Pr[\mathbf{x}_0 \in X_0] = \mathbf{1}_{X_0}(x_0),$$

$$\Pr[\mathbf{x}_k \in X_k | \mathbf{x}_{k-1} = x, \pi] = T(X_k|x, \pi_{k-1}(x)),$$

where $\mathbf{1}_{X_0}$ is the indicator function (that is, 1 if $x \subseteq X_0$ and 0 otherwise). Having a definition of Pr allows one to make probabilistic statements over the stochastic model in Eqn (1).

Remark 1. Note that, as is common in the literature [20], according to the definition of the probability measure Pr we marginalise over the posterior distribution at each time step. Consequently, according to our modelling framework, the weights of the BNN are not kept fixed during each trajectory, but we re-sample from $\mathbf{w}$ at each time step.

3.1. Problem statements

We consider two problems concerning, respectively, the certification and the control of dynamical systems modelled by BNNs. We first consider safety certification with respect to probabilistic reach-avoid specifications. That is, we seek to compute the probability that from a given state, under a selected control policy, an agent navigates to the goal region without encountering any unsafe states. Next, we consider the formal synthesis of policies that maximise this probability and thus attain maximal certifiable safety.

Problem 1 *(Computation of Probabilistic Reach-Avoid).* Given a strategy π, a goal region $\mathrm{G} \subseteq \mathbb{R}^n$, a finite-time horizon $[0, N] \subseteq \mathbb{N}$, and a safe set $\mathrm{S} \subseteq \mathbb{R}^n$ such that $\mathrm{G} \cap \mathrm{S} = \emptyset$, compute for any $x_0 \in \mathrm{G} \cup \mathrm{S}$

$$P_{reach}(\mathrm{G}, \mathrm{S}, x_0, [0, N]|\pi) = \\ \Pr\left[\exists k \in [0, N], \mathbf{x}_k \in \mathrm{G} \wedge \forall 0 \le k' < k, \mathbf{x}_{k'} \in \mathrm{S} \mid \mathbf{x}_0 = x_0, \pi\right]. \tag{5}$$

Outline of the approach In Section 4.1 we show how $P_{reach}(\mathrm{G}, \mathrm{S}, x_0, [0, N]|\pi)$ can be formulated as the solution of a backward iterative computational procedure, where the uncertainty of the BNN is propagated backward in time, starting from the goal region. Our approach allows us to compute a sound lower bound on P_{reach}, thus guaranteeing that $\mathbf{x}_k$, as defined in Eqn (1), satisfies the specification with a given probability. This is achieved by extending existing lower bounding techniques developed to certify BNNs [63] and applying these at each propagation step through the BNN.

Note that, in Problem 1, the strategy π is provided, and the goal is to quantify the probability with which the trajectories of $\mathbf{x}_k$ satisfy the given specification. In Problem 2 below, we expand the previous problem and seek to synthesise a controller π that maximizes P_{reach}. The general formulation of this optimization is given below.

Problem 2 *(Strategy Synthesis for Probabilistic Reach-Avoid).* For an initial state $x_0 \in \mathrm{G} \cup \mathrm{S}$, and a finite time horizon N, find a strategy $\pi^* : \mathbb{R}^n \times \mathbb{R}_{\ge 0} \to \mathbb{R}^c$ such that

$$\pi^* = \arg\max_{\pi} P_{reach}(\mathrm{G}, \mathrm{S}, x_0, [0, N] \mid \pi). \tag{6}$$

In Section 5, we will provide specific schemes for synthesizing optimal strategies when π is either a look-up table or a deterministic neural network.

Outline of the approach To solve this problem, we notice that the backward iterative procedure outlined to solve Problem 1 has a substructure such that dynamic programming will allow us to compute optimal actions for each state that we verify, thus producing an optimal policy with respect to the given posterior and reach-avoid specification. With low-dimensional or discrete action spaces, we can then derive a tabular policy by solving the resulting dynamic programming problem. For higher-dimensional action spaces instead, in Section 5.1 we consider (generalising) policies represented as neural networks.

4. Methodology

In this section, we illustrate the methodology used to compute lower bounds on the reach-avoid probability, as described in Problem 1. We begin by encoding the reach-avoid probability through a sequence of value functions.

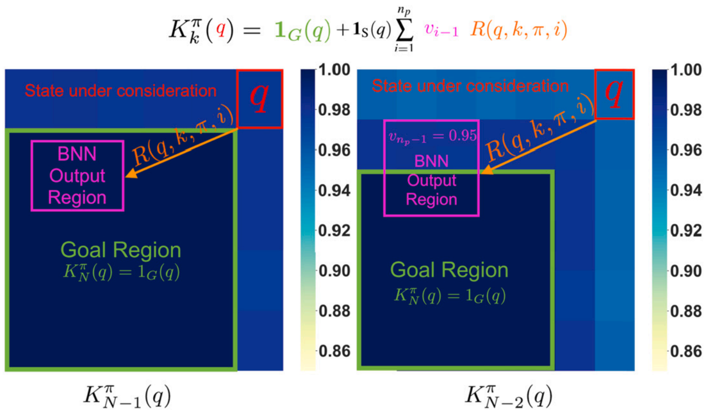

Fig. 1. Examples of functions K_{N-1}^π (left) and K_{N-2}^π (right), which are lower bounds of V_k^π. On the left, we consider the first step of our backward algorithm, where we compute $K_{N-2}^\pi(q)$ by computing the probability that $\mathbf{x}_N \in \mathrm{G}$ given that $\mathbf{x}_{N-1} \in q$. On the right, we consider the subsequent step. We outline the state we want to verify in red and the goal region in green. With the orange arrow, we represent the 0.95 transition probability of the BNN dynamical model, and in pink we represent the worst-case probabilities spanned by the BNN output. On top, we show where each of these key terms comes into play in Eqn (9). (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

4.1. *Certifying reach-avoid specifications*

We begin by showing that $P_{reach}(\mathrm{G},\mathrm{S},x,[k,N]|\pi)$ can be obtained as the solution of a backward iterative procedure, which allows to compute a lower bound on its value. In particular, given a time $0 \le k < N$ and a strategy π, consider the value functions $V_k^\pi : \mathbb{R}^n \to [0,1]$, recursively defined as

$$V_N^\pi(x) = \mathbf{1}_\mathrm{G}(x),$$
$$V_k^\pi(x) = \mathbf{1}_\mathrm{G}(x) + \mathbf{1}_\mathrm{S}(x)\int V_{k+1}^\pi(\bar{x})p\big(\bar{x}|(x,\pi_k(x)),\mathcal{D}\big)d\bar{x}. \tag{7}$$

Intuitively, V_k^π is computed starting from the goal region G at $k = N$, where it is initialised at value 1. The computation proceeds backwards at each state x, by combining the current values with the transition probabilities from Eqn (1). The following proposition, proved inductively over time in the Supplementary Material, guarantees that $V_0^\pi(x)$ is indeed equal to $P_{reach}(\mathrm{G},\mathrm{S},x,[0,N]|\pi)$.

Proposition 1. *For $0 \le k \le N$ and $x_0 \in \mathrm{G} \cup \mathrm{S}$, it holds that*

$$P_{reach}(\mathrm{G},\mathrm{S},x_0,[k,N]|\pi) = V_k^\pi(x).$$

The backward recursion in Eqn (7) does not generally admit a solution in closed-form, as it would require integrating over the BNN posterior predictive distribution, which is in general analytically intractable. In the following section, we present a computational scheme utilizing convex relaxations to lower bound P_{reach}.

4.2. *Lower bound on P_{reach}*

We develop a computational approach based on the discretisation of the state space, which allows convex relaxation methods such as [63] to be used. The proposed computational approach is illustrated in Fig. 1 and formalized in Section 4.3. Let $Q = \{q_1, ..., q_{n_q}\}$ be a partition of $\mathrm{S} \cup \mathrm{G}$ in n_q regions and denote with $z : \mathbb{R}^n \to Q$ the function that associates to a state in $\mathbb{R}^n$ the corresponding partitioned state in Q. For each $0 \le k \le N$ we iteratively build a set of functions $K_k^\pi : Q \to [0,1]$ such that for all $x \in \mathrm{G} \cup \mathrm{S}$ we have that $K_k^\pi(z(x)) \le V_k^\pi(x)$. Intuitively, K_k^π provides a lower bound for the value functions on the computation of P_{reach}.

The functions K_k^π are obtained by propagating backward the BNN predictions from time N, where we set $K_N^\pi(q) = \mathbf{1}_\mathrm{G}(q)$, with $\mathbf{1}_\mathrm{G}(q)$ being the indicator function (that is, 1 if $q \subseteq \mathrm{G}$ and 0 otherwise). Then, for each $k < N$, we first discretize the set of possible probabilities in n_p sub-intervals $0 = v_0 \le v_1 \le ... \le v_{n_p} = 1$. Hence, for any $q \in Q$ and probability interval $[v_i, v_{i+1}]$, one can compute a lower bound, $R(q,k,\pi,i)$, on the probability that, starting from any state in q at time k, we reach in the next step a region that has probability $\in [v_i, v_{i+1}]$ of safely reaching the goal region. The resulting values are used to build K_k^π (as we will detail in Eqn (9)). For a given $q \subset \mathrm{S}$, $K_k^\pi(q)$ is obtained as the sum over i of $R(q,k,\pi,i)$ multiplied by v_{i-1}, i.e., the lower value that K_{k+1}^π obtains in all the states of the $i-th$ region. Note that the discretisation of the probability values does not have to be uniform, but can be adaptive for each $q \in Q$. A heuristic for picking the value of thresholds v_i will be given in Algorithm 1. In what follows, we formalise the intuition behind this computational procedure.

4.3. Lower bounding of the value functions

For a given strategy π, we consider a constant $\eta \in (0,1)$ and $\epsilon = \sqrt{2\sigma^2}\mathrm{erf}^{-1}(\eta)$, which are used to bound the value of the noise, $\mathbf{v}_k$, at any given time. Intuitively, η represents the proportion of observational error we consider.[2] Then, for $0 \le k < N$, $K_k^\pi : Q \to [0,1]$ are defined recursively as follows:

$$K_N^\pi(q) = \mathbf{1}_{\mathrm{G}}(q), \tag{8}$$

$$K_k^\pi(q) = \mathbf{1}_{\mathrm{G}}(q) + \mathbf{1}_{\mathrm{S}}(q) \sum_{i=1}^{n_p} v_{i-1} R(q,k,\pi(q),i), \tag{9}$$

where

$$R(q,k,\pi(q),i) = \eta^n \int_{H_{k,i}^{q,\pi,\epsilon}} p_{\mathbf{w}}(w|\mathcal{D})dw, \tag{10}$$

$$H_{k,i}^{q,\pi,\epsilon} = \{w \in \mathbb{R}^{n_w} \mid \forall x \in q, \forall \gamma \in [-\epsilon,\epsilon]^n, \text{ it holds that:}$$
$$v_{i-1} \le K_{k+1}^\pi(q') \le v_i, \text{ with } q' = z(f^w(x,\pi_k(x)) + \gamma)\}.$$

The key component for the above backward recursion is $R(q,k,\pi,i)$, which bounds the probability that, starting from q at time k, we have that $\mathbf{x}_{k+1}$ will be in a region q' such that $K_{k+1}^\pi(q') \in [v_i, v_{i+1}]$. By definition, the set $H_{k,i}^{q,\pi,\epsilon}$ defines the weights for which the BNN maps all states covered by q into the goal states given action $\pi(q)$. Given this, it is clear that integration of the posterior $p_{\mathbf{w}}(w|\mathcal{D})$ over the $H_{k,i}^{q,\pi,\epsilon}$ will return the probability mass of system (1) transitioning from q to q' with probability in $[v_i, v_{i+1}]$ in one time step. The computation of Eqn (9) then reduces to computing the set of weights $H_{k,i}^{q,\pi,\epsilon}$, which we call the *projecting weight set*. A method to compute a safe under-approximation $\bar{H} \subseteq H_{k,i}^{q,\pi,\epsilon}$ is discussed below. Before describing that, we analyze the correctness of the above recursion.

Theorem 1. *Given $x \in \mathbb{R}^n$, for any $k \in \{0,\dots,N\}$ and $q = z(x)$, assume that $H_{k,i}^{q,\pi,\epsilon} \cap H_{k,j}^{q,\pi,\epsilon} = \emptyset$ for $i \ne j$. Then:*

$$\inf_{x\in q} V_k^\pi(x) \ge K_k^\pi(q).$$

A proof of Theorem 1 is given in the Supplementary Material. Note that the assumption on the null intersection between different projecting weight sets required in Theorem 1 can always be enforced by taking their intersection and complement.

4.4. Computation of projecting weight set

Theorem 1 allows us to compute a safe lower bound to Problem 1, by relying on an abstraction of the state space, that is, through the computation of $K_0^\pi(q)$. This can be evaluated once the projecting set of weight values $H_{k,i}^{q,\pi,\epsilon}$ associated to $[v_{i-1}, v_i]$ is known.[3] Unfortunately, direct computation of $H_{k,i}^{q,\pi,\epsilon}$ is intractable. Nevertheless, a method for its lower bounding was developed by Wicker et al. [63] in the context of adversarial perturbations for one-step BNN predictions, and can be directly adapted to our settings.

The idea is that an under approximation $\bar{H} \subseteq H_{k,i}^{q,\pi,\epsilon}$ is built by sampling weight boxes of the shape $\hat{H} = [w^L, w^U]$, according to the posterior, and checking whether:

$$v_{i-1} \le K_{k+1}^\pi(z(f^w(x,\pi_k(x)) + \gamma)) \le v_i,$$
$$\forall x \in q, \forall w \in \hat{H}, \forall \gamma \in [-\epsilon,\epsilon]^n. \tag{11}$$

Finally, $\bar{H}$ is built as a disjoint union of boxes $\hat{H}$ satisfying the above condition. For a full discussion of the details of this method we refer interested readers to [63]. In order to apply this method to our setting, we propagate the abstract state q through the policy function $\pi_k(x)$, so as to obtain a bounding box $\widehat{\Pi} = [\pi^L, \pi^U]$ such that $\pi^L \le \pi_k(x) \le \pi^U$ for all $x \in q$. In the experiments, this bounding is only necessary when $\pi_k(x)$ is given by an NN controller, for which bound propagation of NNs can be used for the computation of $\widehat{\Pi}$ [25,29].

The results of Proposition 2 and Proposition 3 from Wicker et al. [63] can then be used to propagate q, $\widehat{\Pi}$ and $\hat{H}$ through the BNN. For discrete posteriors (e.g., those resulting from HMC) one can use the method described by Gowal et al. [29] (Eqs. (6) and (7)). Propagation of q, $\widehat{\Pi}$ amounts to using these method to compute values $f_{q,\epsilon,k}^L$ and $f_{q,\epsilon,k}^U$ such that, for all $x \in q, \gamma \in [-\epsilon,\epsilon]^n, w \in \hat{H}$, it holds that:

[2] The threshold is such that it holds that $Pr(|\mathbf{v}_k^{(i)}| \le \epsilon) = \eta$. In the experiments of Section 6 we select $\eta = 0.99$.

[3] In the case of Gaussian VI the integral of Equation (10) can be computed in terms of the *erf* function, whereas more generally Monte Carlo or numerical integration techniques can be used.

Algorithm 1 Probabilistic Reach-Avoid for BNNs.

Input: BNN model $f^{\mathbf{w}}$, safe region S, goal region G, discretization Q of $\mathrm{S} \cup \mathrm{G}$, time horizon N, neural controller π, number of BNN samples n_s, weight margin ρ_w, state space margin ρ_x
Output: Lower bound on V^π

```
1:  For all $0 \leq k \leq N$ set $K_k^\pi(q) = 1$ iff $q \subseteq \mathrm{G}$ and 0 otherwise
2:  for $k \leftarrow N$ to 1 do
3:    for $q \in Q \setminus \mathrm{G}$ do
4:      $v_1 \leftarrow \max_{x \in [q-\rho_x, q+\rho_x]} K_{k+1}^\pi(z(x))$
5:      $\bar{H} \leftarrow \emptyset$ # $\bar{H}$ is the set of safe weights
6:      for desired number of samples, $n_s$ do
7:        $w' \sim P(w|\mathcal{D})$
8:        $\hat{H} \leftarrow [w' - \rho_w, w' + \rho_w]$
9:        $\bar{X} \leftarrow [f_{q,\epsilon,k}^L, f_{q,\epsilon,k}^U]$   # Computed according to Eqn (12)
10:       if $\min_{x \in \bar{X}} K_{k+1}^\pi(z(x)) \geq v_1$ then
11:         $\bar{H} \leftarrow \bar{H} \bigcup \hat{H}$
12:       end if
13:     end for
14:     Ensure $H_i \cap H_j = \emptyset \quad \forall H_i, H_j \in \bar{H}$
15:     $K_k^\pi(q) = v_1 \cdot \eta^n \int_{\bar{H}} p_{\mathbf{w}}(w|\mathcal{D}) dw$   (Eqn (9))
16:   end for
17: end for
18: return $K^\pi$
```

$$f_{q,\epsilon,k}^L \leq f^w(x, \pi_k(x)) + \gamma \leq f_{q,\epsilon,k}^U. \tag{12}$$

Furthermore, $f_{q,\epsilon,k}^L$ and $f_{q,\epsilon,k}^U$ are differentiable w.r.t. the input vector [29,64].

Finally, the two bounding values can be used to check whether or not the condition in Eqn (11) is satisfied, by simply checking whether $[f_{q,\epsilon,k}^L, f_{q,\epsilon,k}^U]$ propagated through K_{k+1}^π is within $[v_i, v_{i+1}]$. We highlight that computing this probability is equivalent to a conservative estimate of $R(q, k, \pi, i)$.

4.5. Probabilistic reach-avoid algorithm

In Algorithm 1 we summarize our approach for computing a lower bound for Problem 1. For simplicity of presentation, we consider the case $n_p = 2$, (i.e., we partition the range of probabilities in just two intervals $[0, v_1]$, $[v_1, 1]$ - the case $n_p > 2$ follows similarly). The algorithm proceeds by first initializing the reach-avoid probability for the partitioned states q inside the goal region G to 1, as per Eqn (8). Then, for each of the N time steps and for each one of the remaining abstract states q, in line 4 we set the threshold probability v_1 equal to the maximum value that K^π attains at the next time step over the states in the neighbourhood of q (which we capture with a hyper-parameter $\rho_x > 0$). We found this heuristic for the choice of v_1 to work well in practice (notice that the obtained bound is formal irrespective of the choice of v_1, and different choices could potentially be explored). We then proceed in the computation of Eqn (9). This computation is performed in lines 5–14. First, we initialise to the null set the current under-approximation of the projecting weight set, $\bar{H}$. We then sample n_s weights boxes $\hat{H}$ by sampling weights from the posterior, and expanding them with a margin ρ_w heuristically selected (lines 6-8). Then, for each of these sets, we first propagate the state q, policy function, and weight set $\bar{H}$ to build a box $\bar{X}$ according to Eqn (12) (line 9), which is then accepted or rejected based on the value that K^π at the next time step attains in states in $\bar{X}$ (lines 10-12). $K_{N-i}^\pi(q)$ is then computed in line 14 by integrating $p_{\mathbf{w}}(w|\mathcal{D})$ over the union of the accepted sets of weights.

5. Strategy synthesis

We now focus on synthesising a strategy that maximizes our lower bound on P_{reach}, thus solving Problem 2. Notice that, while no global optimality claim can be made about the strategy that we obtain, maximising the lower bound guarantees that the true reach-avoid probability will still be greater than the improved bound obtained after the maximisation.

Definition 1. A strategy π^* is called maximally certified (max-cert), w.r.t. the discretised value function K^π, if and only if, for all $x \in \mathrm{G} \cup \mathrm{S}$, it satisfies

$$K_0^{\pi^*}(z(x)) = \sup_\pi K_0^\pi(z(x)),$$

that is, the strategy π^* maximises the lower bound of P_{reach}.

It follows that, if $K_0^{\pi^*}(z(x)) > 1 - \delta$ for all $x \in \mathrm{G} \cup \mathrm{S}$, then the max-cert strategy π^* is a solution of Problem 2. Note that a max-cert strategy is guaranteed to exist when the set of admissible controls $\mathcal{U}$ is compact [11, Lemma 3.1], as we assume in this work. In the

Algorithm 2 Numerical Synthesis of Action for region q at time k.

```
Input: BNN model $f^{\mathbf{w}}$, safe region S, goal region G, action space $\mathcal{U}$, abstract state $q \in Q$, controller $\pi$, number of BNN samples $n_s$
Output: Action maximizing $K^{\pi}$
1: $\Upsilon \leftarrow$ middle points of each region in a partition of $\mathcal{U}$
2: $\kappa^* \leftarrow 0$
3: $u^* \leftarrow 0$
4: for $u \in \Upsilon$ do
5:     $\hat{\kappa} \leftarrow 0$
6:     for $j$ from 0 to $n_s$ do
7:         $w' \sim P(w|\mathcal{D})$
8:         $\bar{X} \leftarrow [f^L_{q,\epsilon,k}, f^U_{q,\epsilon,k}]$ # Computed for $q$ and $u$ via (Eqn (12))
9:         $\hat{\kappa} = \hat{\kappa} + \frac{\min_{x \in \bar{X}} K^*_{k+1}(z(x))}{n_s}$
10:    end for
11:    if $\hat{\kappa} > \kappa^*$ then
12:        $\kappa^* \leftarrow \hat{\kappa}$
13:        $u^* \leftarrow u$
14:    end if
15: end for
16: return $u^*$
```

next theorem, we show that a max-cert strategy can be computed via dynamic programming with a backward recursion similar to that of Eqn (9).

Theorem 2. *For $0 \le k < N$ and $0 = v_0 < \dots < v_{n_p} = 1$, define the functions $K^*_k : \mathbb{R}^n \to [0,1]$ recursively as follows*

$$K^*_k(q) = \sup_{u \in \mathcal{U}} \left(\mathbf{1}_{\mathrm{G}}(q) + \mathbf{1}_{\mathrm{S}}(q) \sum_{i=1}^{n_p} v_i R(q,k,u,i) \right),$$

where $R(q,k,u,i)$ and $H^{q,u,\epsilon}_{k,i}$ are defined as in Eqn (10). If π^* is s.t. $K^*_0 = K^{\pi^*}_0$, then π^* is a max-cert strategy. Furthermore, for any x, it holds that $K^{\pi^*}_0(z(x)) \le P_{reach}(\mathrm{G}, \mathrm{S}, [0,N], x|\pi^*)$.

Theorem 2 is a direct consequence of the Bellman principle of optimality [1, Theorem 2] and it guarantees that for each state $q \in \mathrm{S}$ and time k, we have that $\pi^*(q,k) = \arg\max_{u \in \mathcal{U}} \sum_{i=1}^{n_p} v_i R(q,k,u,i)$.

In Algorithm 2 we present a numerical scheme based on Theorem 2 to find a max-cert policy π^*. Note that the optimization problem required to be solved at each time step state, i.e. $\arg\max_{u \in \mathcal{U}} \sum_{i=1}^{n_p} v_i R(q,k,u,i)$, is non-convex. Hence, in Algorithm 2, in Line 1, we start by partitioning the action space $\mathcal{U}$. Then, in Lines 4–15 for each action in the partition we estimate the expectation of K^*_{k+1} starting from q via n_s samples taken from the BNN posterior (250 in all our experiments). Finally, in Lines 11–14 we keep track of the action maximising K^*_{k+1}.

The described approach for synthesis, while optimal in the limit of an infinitesimal discretization of $\mathcal{U}$, may become infeasible for large state and action spaces. As a consequence, in the next subsection, we also consider when π is parametrised by a neural network and thus can serve as a function over a larger (even infinite) state space. Specifically, we show how a set of neural controllers, one for each time step, can be trained in order to maximize probabilistic reach-avoid via Theorem 2. In Section 6 we empirically investigate both controller strategies.

5.1. An approach for strategy synthesis based on neural networks

In this subsection we show how we can train a set of NN policies $\pi_0, \dots, \pi_{N-1} : \mathbb{R}^n \to \mathcal{U}$ such that at each time step k, π_k approximately solves the dynamic programming equation in Theorem 2. Note that, because of the approximate nature of the NN training, the resulting neural policies will necessarily be sub-optimal, but have the potential to scale to larger and more complex systems, compared to the approach presented in Algorithm 2.

At time k we start with an initial set of parameters (weights and biases) θ_k for policy π_k. These parameters can either be initialized to θ_{k+1}, the parameters synthesised at the previous time step of the value iteration for π_{k+1}, or to a policy employed to collect the data to train the BNN as in Gal et al. [24], or simply selected at random. In our implementation where no previous policy is available, we start with a randomly initialized NN, and then at time k we set our initial neural policy with that obtained at time $k+1$. We then employ a scheme to learn a "safer" set of parameters via backpropagation. In particular, we first define the following loss function penalizing policy parameters that lead to an unsafe behaviour for an ensemble of NNs sampled from the BNN posterior distribution:

$$\mathcal{L}(x, \theta_k) = -\alpha || \sum_{w \in \bar{W}} f^w(x, \pi_k(x)) - \mathbf{A}_k ||_2 + (1-\alpha) || \sum_{w \in \bar{W}} f^w(x, \pi_k(x)) - \mathbf{R}_k ||_2, \quad (13)$$

where $\bar{W}$ are a set of parameters independently sampled from the BNN posterior $p_{\mathbf{w}}(w|\mathcal{D})$, for a probability threshold $0 \le p_t \le 1$, $\mathbf{A}_k = \{x : K^{\pi_{k+1}}_{k+1}(x) \ge p_t\}$ and $\mathbf{R}_k = \{x : K^{\pi_{k+1}}_{k+1}(x) \le 1 - p_t\}$ are the sets of states for which the probability of satisfying the specification at time $k+1$ is respectively greater than p_t and smaller than $1-p_t$. For $X \subset \mathbb{R}^n$, $||x-X||_2 = \inf_{\bar{x}\in X} ||x-\bar{x}||_2$ is the standard L_2 distance of a point from a set, and $0 \leq \alpha \leq 1$ is a parameter taken to be 0.25 in our experiments, that weights between reaching the goal and staying away from "bad" states. Intuitively, the first term in $\mathcal{L}(x,\theta_k)$ enforces θ_k that leads to high values of $K_{k+1}^{\pi_{k+1}}$, while the second term penalizes parameter sets that lead to small values of this quantity.

$\mathcal{L}(x,\theta_k)$ only considers the behaviour of the dynamical system of Equation (1) starting from initial state x. Then, in order to also enforce robustness in a neighbourhood of initial states around x, similarly to the adversarial training case [44], we consider the robust loss

$$\bar{\mathcal{L}}(x,\theta_k) = \max_{x' : ||x-x'||_2 \leq \epsilon} \mathcal{L}(x,\theta_k). \tag{14}$$

Note that by employing Eqn (12) we obtain a differentiable upper bound of $\bar{\mathcal{L}}(x,\theta_k)$, which can be employed for training θ_k.

5.2. *Discussion on the algorithms*

In this section we provide further discussion of our proposed algorithms including the complexity and the various sources of approximation that may lead to looser guarantees. To frame this discussion, we start by highlighting the complexity and approximation introduced by the chosen bound propagation technique shared by both of the algorithms. We then proceed to discuss how discretisation choices made with respect to the state-space, the weight-space, and the observational noise, practically affect the tightness of our probability bounds for both algorithms, and finally how the action-space discretisation affects our synthesis algorithm.

Bound propagation techniques Given that there are currently no methods for BNN certification that are both sound and complete [10,63,65], the evaluation of the R function will always introduce some approximation error. While it is difficult to characterize this error in general, it is known that for deeper networks, BNN certification methods introduce more approximation than shallow networks [62]. The recently developed bounds from Berrada et al. [10], have been shown to be tighter than the IBP and LBP approaches from Wicker et al. [63] at the cost of computation complexity that is exponential in the number of dimensions of the state-space. In contrast, each iteration of the interval bound propagation method proposed in and Wicker et al. [63] requires the computational complexity of four forward passes through the neural network architecture.

Discretization error and complexity While our formulation supports any form of state space discretisation, we can assume for simplicity that each dimension of the n-dimensional state-space is broken into m equal-sized abstract states. This implies that certification of the system requires us to evaluate the R function $\mathcal{O}\big(N(m^n)\big)$ many times, where N is the time horizon we would like to verify. Given that n is fixed, the user has control over m, the size of each abstract state. For large abstract states, small m, one introduces more approximation as the R function must account for all possible behaviours in the abstract state. For small abstract states, large m, there is much less approximation, but considerably larger runtime. Assume the c-dimensional action space is broken into t equal portions at each dimension, then the computational complexity of the algorithm becomes $\mathcal{O}\big(t^c N(m^n)\big)$ as each of the m^n states must be evaluated t^c-many times to determine the approximately optimal action. As with the state-space discretization, larger t will lead to a more-optimal action choice but requires greater computational time.

6. Experiments

We provide an empirical analysis of our BNN certification and policy synthesis methods. We begin by providing details on the experimental setting in Section 6.1. We then analyse the performance of our certification approach on synthesized policies in Section 6.2. Next, in Section 6.3, we discuss how the choice of the BNN inference algorithm affects synthesis and certification results. Finally, in Section 6.4 we study how our method scales with larger neural network architectures and in higher-dimensional control settings.

6.1. *Experimental setting*

We consider a planar control task consisting of a point-mass agent navigating through various obstacle layouts. The point-mass agent is described by four dimensions, two encoding position information and two encoding velocity [4]. To control the agent there are two continuous action dimensions, which represent the force applied on the point-mass in each of the two planar directions. The task of the agent is to navigate to a goal region while avoiding various obstacle layouts. The knowledge supplied to the agent about the environment is the locations of the goal and obstacles. The full set of equations describing the agent dynamics is given in Appendix A.1. In our experiments, we analyse three obstacle layouts of varying difficulty, which we name `v1`, `v2` and `Zigzag` - visualized in the left column of Fig. 3. Obstacle layout `v1` places an obstacle directly between the agent's initial position and the goal, forcing the agent to navigate its way around it. Obstacle layout `v2` extends this setting by adding two further obstacles that block off one side of the state space. Finally, scenario `Zigzag` has 5 interleaving triangles and requires the agent to navigate around them in order to reach the goal.

In order to learn an initial policy to solve the task, we employ the episodic learning framework described in Gal et al. [23]. This consists of iteratively collecting data from deploying our learned policy, updating the BNN dynamics model to the new observations, and updating our policy. When collecting data, we start by randomly sampling state-action pairs and observing their resulting state

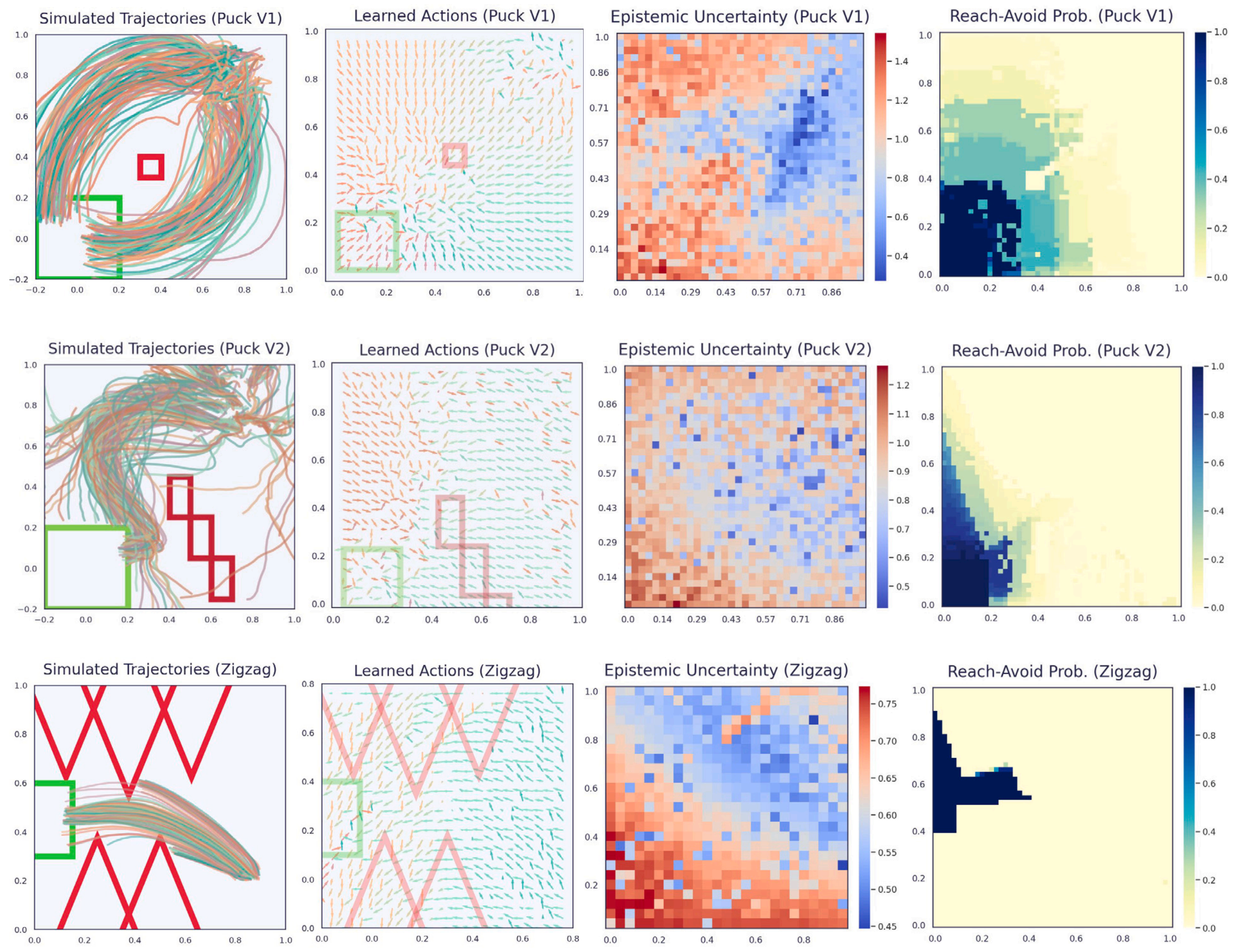

Fig. 2. Left Column: 200 simulated trajectories for the learned policy starting from the initial state. **Centre Left Column:** A 2D visualization of the learned policy. Each arrow represents the direction of the applied force. **Centre Right Column:** The epistemic uncertainty for the learned dynamics model. **Right Column:** Certified lower-bounds of probabilistic reach avoid for each abstract state according to BNN and final learned policy.

according to the ground-truth dynamics. After this initial sample, all future observations from the ground-truth environment are obtained from deploying our learned policy. The initial policy is set by assigning a random action to each abstract state. This is equivalent to tabular policy representations in standard reinforcement learning [59]. We additionally discuss neural network policies in Section 6.4. Actions in the policy are updated by performing gradient descent on a sum of discounted rewards over a pre-specified finite horizon. The reward of an action is taken to be the ℓ_2 distance moved towards the goal region penalized by the ℓ_2 proximity to obstacles as is done in [23,59]. For the learning of the BNN, we perform approximate Bayesian inference over the neural network parameters. For our primary investigation, we select an NN architecture with a single fully connected hidden layer comprising 50 hidden units, and learn the parameters via Hamiltonian Monte Carlo (HMC). Larger neural network architectures are considered in Section 6.4, while results for variational approximate inference are given in Section 6.3.

Unless otherwise specified, in performing certification and synthesis we employ abstract states spanning a width of 0.02 around each position dimension and 0.08 around each velocity dimension. Velocity values are clipped to the range $[-0.5, 0.1]$. When performing optimal synthesis, we discretise the two action dimensions for the point-mass problem into 100 possible vectors which uniformly cover the continuous space of actions $[-1, 1]$. When running our backward reachability scheme, at each state, we test all 100 action vectors and take the action that maximizes our lower bound to be the policy action at that state, thus giving us the locally optimal action within the given discretisation. Further experimental details are presented in Appendix A and code to reproduce all results in this paper can be found at https://github.com/matthewwicker/BNNReachAvoid.

The computational times for each system were roughly equivalent. This is to be expected given that each has the same state space. The following average times are reported for a parallel implementation of our algorithm run on 90 logical CPU cores across 4 Intel Core Xeon 6230 clocked at 2.10 GHz. Training of the initial policy and BNN model takes in the order of 10 minutes, 6 hours for the certification with a horizon of 50 time steps, and 8 hours for synthesis.

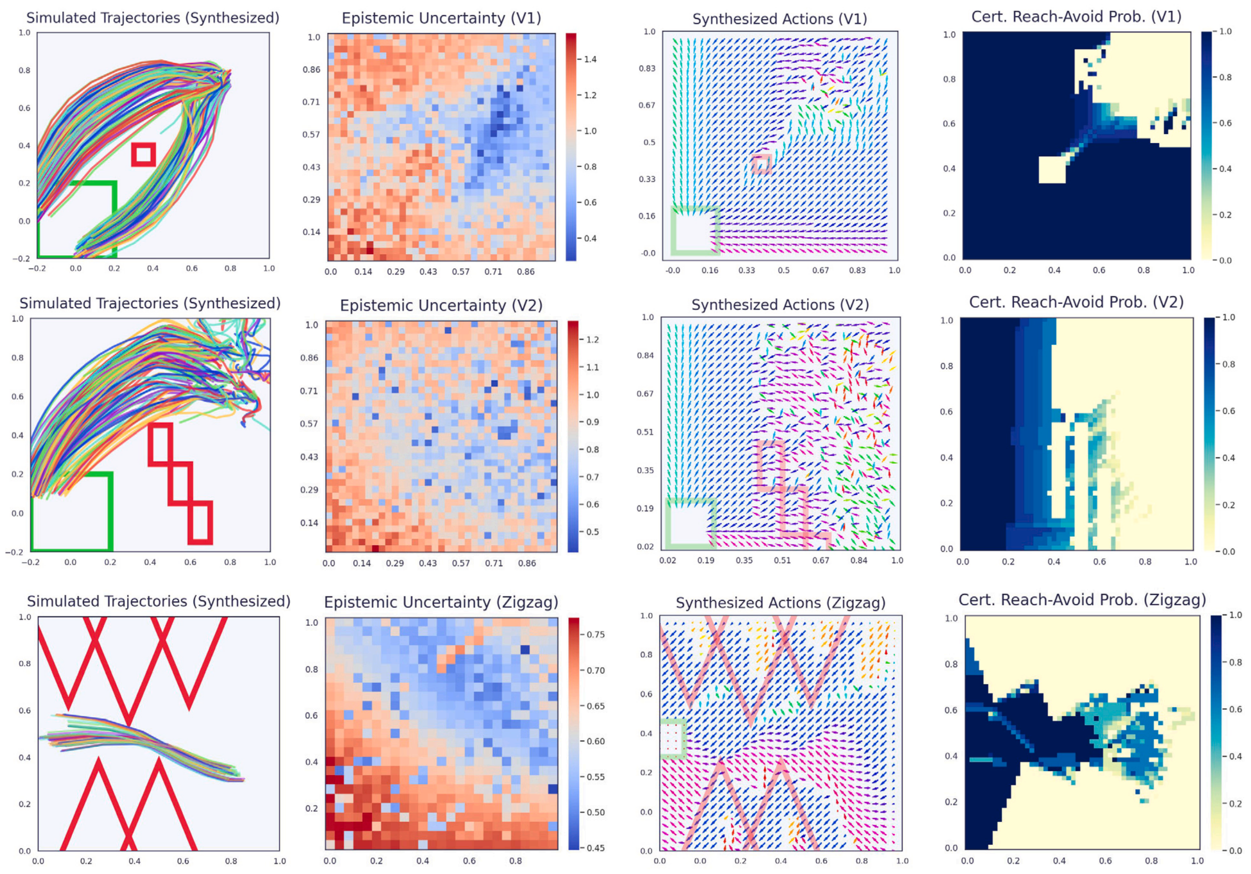

Fig. 3. A version of each learned system after a new policy has been synthesized from the reach-avoid specification. **First column:** 200 simulations of the synthesized policy in the real environment. **Second column:** BNN epistemic uncertainty given as the variance of the BNN predictive distribution. **Third column:** A visualization of the maximally certifiable policies, which demonstrate a clearer tendency to avoid obstacles throughout the state space compared to the policies in Fig. 2. **Fourth column:** Synthesized policies have remarkably higher lower-bounds than learned policies, corresponding plots for learned systems in Fig. 2.

6.2. *Comparing certification of learned and max-cert policies*

In Fig. 2 and Fig. 3 we visualize systems from both learned and synthesized policies. Each row represents one of our control environments and is comprised of four figures. These figures show, respectively, simulations from the dynamical system, BNN uncertainty, the control policy plotted as a gradient field, and the certified safety probabilities. The first column of the Figures depicts 200 simulated trajectories of the learned (Fig. 2) or synthesized (Fig. 3) control policies on the BNN (whose uncertainty is plotted along the second column). Notice how in both cases we visually obtain the behaviour expected, with the overwhelming majority of the simulated trajectories safely avoiding the obstacles (red regions in the figure) and terminating in the goal (green region). A vector field associated with the policy is depicted in the third column of the figures. Notice that, the actions returned by our synthesis method intuitively align with the reach-avoid specification, that is, synthesized actions near the obstacle and out-of-bounds are aligned with moving away from such unsafe states. Exceptions to this are represented by locations where the agent is already unsafe and that, as such, are not fully explored during the BNN learning phase (e.g., the lower triangles in the `Zigzag` scenario), locations where two directions are equally optimal (e.g., in the top right corner of the `v1` environment) and locations which are not along any feasibly optimal path (e.g., the lower right corner of `v2`) and as such are not accounted by the BNN learning.

In Table 1 we compare the certification results of the synthesized policy against the initial learned policy. As the synthesized policy is computed by improving on the latter, we expect the former to outperform the learned policy in terms of the guarantees obtained. This is in fact confirmed and quantified by the results of Table 1, which lists, for each of the three environments, the average reach-avoid probability estimated over 500 trajectories, the average certification lower bound across the state space, and the certification coverage (i.e., the proportion of states where our algorithm returns a non-zero probability lower bound). This notion of coverage only requires a state to be certified with a probability above 0, and so it is most informative when evaluated together with the average lower-bound and visual inspection of Fig. 2 and Fig. 3.

Indeed, the synthesized policy significantly improves on the certification guarantees given by the learned policy, and consistently so across the three environments analysed, with the lower bound improving by a factor of roughly 3.5. This considerable improvement is to be expected as worst-case guarantees can be poor for deep learning systems that are not trained with specific safety

Table 1
Certification comparisons for learned and synthesized policy across the three environments. **Performance** indicates the proportion of simulated trajectory that respect the reach-avoid specification. **Avg. Lower Bound** is the mean certification probability across all states. **Cert. Coverage** is the proportion of states that we are able to certify (i.e., with a non-zero lower bound for the reach-avoid probability).

Learned Policy

	Performance	Avg. Lower Bound	Cert. Coverage
`V1`	0.789	0.212	0.639
`V2`	0.805	0.192	0.484
`Zigzag`	0.815	0.189	0.193

Synthesized Policy (Optimal)

	Performance	Avg. Lower Bound	Cert. Coverage
`V1`	**0.94**	**0.789**	**0.808**
`V2`	**0.94**	**0.597**	**0.624**
`Zigzag`	**1.00**	**0.710**	**0.910**

Table 2
Certification comparisons for learned and synthesized policy between VI and HMC BNN learning on obstacle layout `V1`. **Performance** indicates the proportion of simulated trajectory that respect the reach-avoid specification. **Avg. Lower Bound** is the mean certification probability across all states. **Cert. Coverage** is the proportion of states that we are able to certify with non-zero probability.

Learned Policy

	Performance	Avg. Lower Bound	Coverage
Var. Inference	0.832	0.399	0.696
Ham. Monte Carlo	0.789	0.212	0.639

Synthesized Policy (Optimal)

	Performance	Avg. Lower Bound	Coverage
Var. Inference	**1.00**	**0.762**	**0.851**
Ham. Monte Carlo	**0.94**	**0.789**	**0.808**

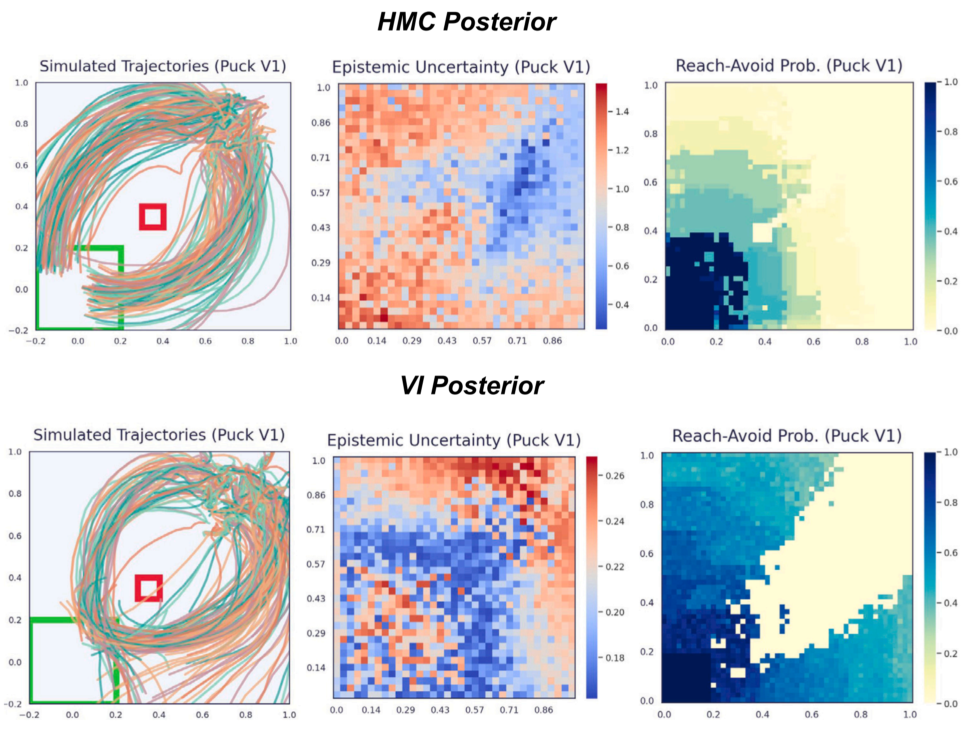

Fig. 4. Top Row: Visualization of the learned system using HMC to approximately infer dynamics. **Bottom Row:** Visualization of the learned system using VI to approximately infer dynamics. We highlight that the VI approximation displays a 5 to 10 times reduction in epistemic uncertainty.

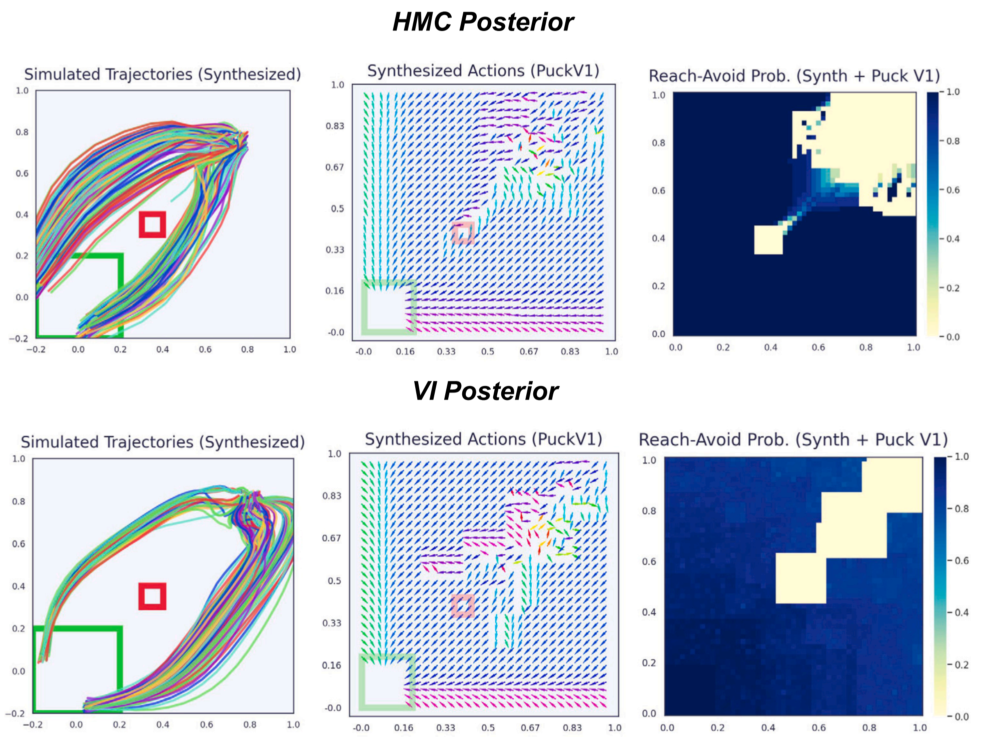

Fig. 5. Top Row: Visualization of the synthesized policy and its performance based on the HMC dynamics model. **Bottom Row:** Visualization of the synthesized policy and its performance based on the VI dynamics model.

objectives [29,48,64]. In particular, for both the `V1` and `Zigzag` case studies, we observe that the average lower bound jumps from roughly 0.2 to greater than 0.7. Moreover, the most significant improvements are obtained in the most challenging case, i.e., the `Zigzag` environment, with the certification coverage increasing of a 4.75 factor. Interestingly, also the average model performance increases for the synthesized models. Intuitively this occurs because while in the learning of the initial policy passing through the obstacle is only penalised by a continuous factor, the synthesized policy strives to rigorously enforce safety across the BNN posterior. A visual representation of these results is provided in the last column of Fig. 2 for the learned policy and in Fig. 3 for the synthesized policy. We note that the uncertainty maps in these figures are identical as the BNN model is not changed, only the policy.

6.3. On the effect of approximate inference

The results provided so far have been generated with BNN dynamical models learned via HMC training. However, different inference methods produce different approximations of the BNN posterior thus leading to different dynamics approximations and hence synthesized policies.

Table 2 and the plots in Figs. 4 and 5 analyse the effect of approximate inference on both learned and synthesized policies, comparing results obtained by HMC with those obtained by VI training on the `v1` scenario. We notice that also in the case of VI the synthesized policy significantly improves on the initial policy. Interestingly, the certification results over the learned policy for VI are higher than those obtained for HMC, but the results are comparable for the synthesized policies. In fact, it is known in the literature that VI tends to under-estimate uncertainty [47,50] and is more susceptible to model misspecification [45]. As such, being probabilistic, the bound obtained is tighter for VI where the uncertainty is lower than that of HMC which provides a more conservative representation of the agent dynamics. For example, we see in the first two rows of Table 2 that the average lower bound achieved for the variational inference posterior is 1.88 times higher than the bounds for HMC posterior. However, our synthesis method reduces this gap between HMC and VI, while still accounting for the higher uncertainty of the former, and hence the more conservative guarantees.

While HMC approximates the posterior by relying on a Monte Carlo estimate of it, VI is a gradient-based technique, where the number of training epochs (i.e., the number of full sweeps through the dataset) is a key hyper-parameter. We thus analyse the effect of training epochs in the quality of the dynamics obtained in Fig. 6, along with the effect on the synthesized policies and the certificates obtained for such policies. The left plot of the figure shows a set of predicted trajectories over a 10 time-step horizon for a varying number of training epochs, with the ground truth behaviour highlighted in red. The BNN trajectories are colour-coded based on the number of epochs each dynamics model was trained for. In yellow, we see that the BNN which has only been trained for 10 epochs displays considerable error in its iterative predictions. This is reduced considerably for a model trained for 50 epochs,

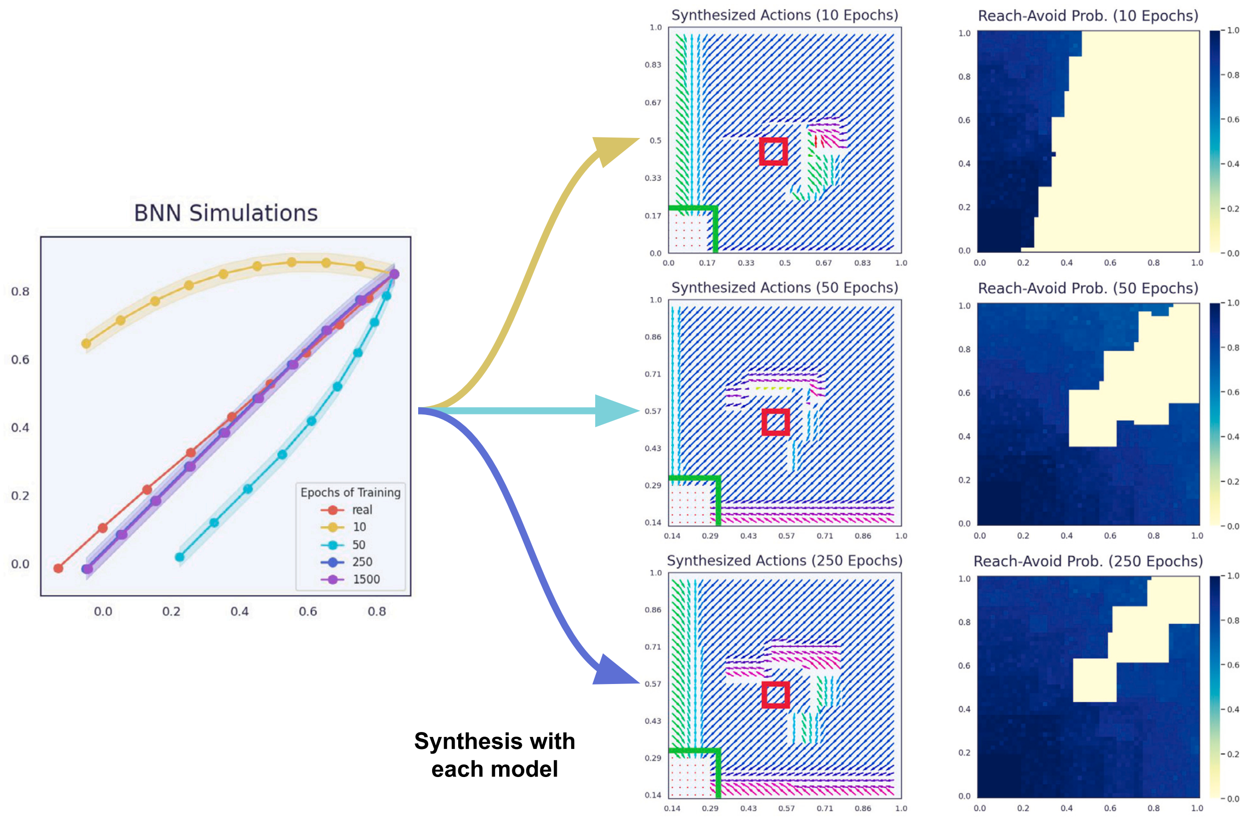

Fig. 6. Analysis for number of training epochs used in performing VI training on the `V1` environment. Left: sample of 10-step agent trajectories obtained with BNNs trained with VI and different number of epochs (red: ground truth trajectory). Right: synthesis and certification results for a selection of training epochs.

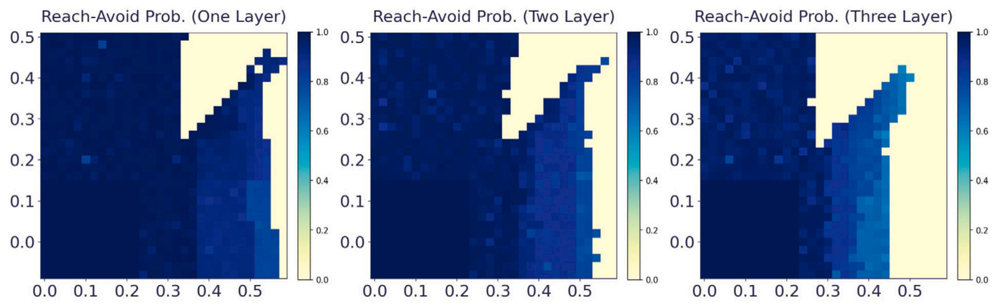

Fig. 7. We vary the depth of the BNN used to learn the dynamics and observe its effects on our certified safety probabilities over the first half of the Puck-V1 state-space. From left to right we plot the lower-bound reach-avoid probabilities for a one-layer BNN dynamics model, a two-layer BNN dynamics model, and a three-layer BNN dynamics model.

but the cumulative error after 10 epochs is still considerable. Finally, as expected, for models trained for 250 and 1500 epochs we empirically observe a trend toward convergence to the ground truth.

We notice that the policy and certifications directly reflect the quality of the approximation. In fact, as we increase the model fit, we see that there are significant improvements in both the intuitive behaviour of the synthesized policy as well as the resulting guarantees we are able to compute.

6.4. Depth experiments

In this section, we evaluate how our method performs when we vary the depth of the BNN dynamics model considered. In Fig. 7, we plot the certified reach-avoid probabilities for a learned policy and a one, two, and three-layer BNN dynamics model where each layer has a width of 12 neurons. Similar architectures are found in the BNNs studied in recent related work [41]. Other than the depth of the BNN, all the other variables in the experiment are held equal (e.g., number of episodes during learning, discretization of state-space, and number of BNN samples considered for the lower bound). The learning procedure results in BNN models with roughly equivalent losses and in policies that are qualitatively similar, see Appendix A.3 for further visualizations. Given that the key factors of the system have been held equal, we notice a decrease in our certified lower bound as depth increases. Specifically, the average lower bound for the one-layer model is 0.811, for the two-layer model it is 0.763, and for the three-layer model it decreases further to 0.621. Fig. 7 clearly demonstrates that as the BNN dynamics model becomes deeper, then our lower bound becomes more conservative. This finding is consistent with existing results in certification of BNNs [10,63] and DNNs [29,48]. We note, however, that when the verification parameters are refined, i.e., more samples from the BNN are taken, we are able to certify the three-layer system with an average certified lower-bound of 0.867 (see Appendix A.3). These additional BNN samples increase the runtime of our certification procedure by 1.5 times.

7. Related work

Certification of machine learning models is a rapidly growing area [25,29,37,66]. While most of these methods have been designed for deterministic NNs, recently safety analysis of Bayesian machine learning models has been studied both for Gaussian processes (GPs) [12,17,30] and BNNs [5,16,63], including methods for adversarial training [43,64]. The above works, however, focus exclusively on the input-output behaviour of the models, that is, can only reason about static properties. Conversely, the problem we tackle in this work has additional complexity, as we aim to formally reason about iterative predictions, i.e., trajectory-level behaviour of a BNN interacting in a closed loop with a controller.

Iterative predictions have been widely studied for Gaussian processes [26] and safety guarantees have been proposed in this setting in the context of model-based RL with GPs [8,9,36,54]. However, all these works are specific to GPs and cannot be extended to BNNs, whose posterior predictive distribution is intractable and non-Gaussian even for the more commonly employed approximate Bayesian inference methods [51]. Recently, iterative prediction of *neural network dynamic models* have been studied [2,61] and methods to certify these models against temporal logic formulae have been derived [2]. However, these works only focus on standard (i.e., non-Bayesian) neural networks with additive Gaussian noise. Closed-loop systems with known (deterministic) models and control policies modelled as BNNs are considered in [41]. In contrast with our work, Lechner et al. [41] can only support deterministic models without noisy dynamics, only focus on the safety verification problem, and are limited to BNN posterior with unimodal weight distribution.

Various recent works consider verification or synthesis of RL schemes against reachability specifications [7,39,58]. None of these approaches, however, support both continuous state-action spaces and probabilistic models, as in this work. Continuous action spaces are supported in [33], where the authors provide RL schemes for the synthesis of policies maximising given temporal requirements, which is also extended to continuous state- and action-spaces in [32]. However, the guarantees resulting from these model-free algorithms are asymptotic and thus of a different nature than those in this work. The work of Haesaert et al. [31] integrates Bayesian inference and formal verification over control models, additionally proposing strategy synthesis approaches for active learning [67]. In contrast to our paper these works do not support unknown noisy models learned via BNNs.

A related line of work concerns the synthesis of runtime monitors for predicting the safety of the policy's actions and, if necessary, correct them with fail-safe actions [3,6,14,22,53]. These approaches, however, do not support continuous state-action spaces or require some form of ground-truth mechanistic model for safety verification (as opposed to our data-driven BNN models).

8. Conclusions

In this paper, we considered the problem of computing the probability of time-bounded reach-avoid specifications for dynamic models described by iterative predictions of BNNs. We developed methods and algorithms to compute a lower bound of this reach-avoid probability. Additionally, relying on techniques from dynamic programming and non-convex optimization, we synthesized certified controllers that maximize probabilistic reach-avoid. In a set of experiments, we showed that our framework enables certification of strategies on non-trivial control tasks. A future research direction will be to investigate techniques to enhance the scalability of our methods so that they can be applied to state-of-the-art reinforcement learning environments. However, we emphasise that the benchmark considered in this work remains a challenging one for certification purposes, due to both the non-linearity and stochasticity of the models, and the sequential, multi-step dependency of the predictions. Thus, this paper makes an important step toward the application of BNNs in safety-critical scenarios.

Declaration of competing interest

The authors declare no financial interests/personal relationships which may be considered as potential competing interests.

Data availability

We have attached a link to the code in the manuscript.

Acknowledgements

This project received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115).

Appendix A. Further experimental details

A.1. Agent dynamics

The puck agent is derived from a classical control problem of controlling a vehicle from an initial condition to a goal state or way point [4]. This scenario is more challenging than other standard benchmarks (i.e. inverted pendulum) due to both the increase state-space dimension and to the introduction of momentum which makes control more difficult. The state space of the unextended agent is a four vector containing the position in the plane as well as a vector representing the current velocity. The control signal is a two vector representing a change in the velocity (i.e. an acceleration vector). The dynamics of the puck can be given as the following system of equations where η determines friction, m determines the mass of the puck, and h determines the size of the time discretization.

$$\dot{q} = Aq + Bc$$
$$A = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 - h\eta/m & 0 \\ 0 & 0 & 0 & 1 - h\eta/m \end{bmatrix}$$
$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ h/m & 0 \\ 0 & h/m \end{bmatrix}$$

The n dimensional extension of the above dynamics is done by simply noting the structure of the matrices and generalizing them. In the upper-left of matrix A we have the 2×2 identity matrix which is extended to $n \times n$. Similarly, the upper-right of A is extended to h times the $n \times n$ identity matrix, and the lower-right is $1 - h\eta/m$ times the $n \times n$ identity matrix. For each environment and including the n dimensional generalizations, time resolution, h, is set to 0.35, the mass of the object, m, is fixed to 5.0, and the friction coefficient, η, is set to 1.0.

A.2. Learning parameters

In this section we provide the hyper-parameters used for learning an initial policy and for synthesizing NN policies. In particular, we give full parameters for the environmental interaction required to learn our policies, BNN parameters to perform approximate inference, and NN parameters for neural policy synthesis.

Episodic parameters In Table A.3 we give the parameters for our episodic learning set up. We provide the duration (number of episodes) and the amount of data collected for each episode (number of trajectories). We highlight that as this is a model-based set up, we require many fewer simulations of the system than corresponding model-free algorithms. Each environment also has an empirically tuned maximum horizon (maximum duration for each trajectory), policy size (discretization of the state-space), and obstacle aversion, c, as discussed in Section 5.

BNN architectures In Table A.4, we report the HMC learning parameters for our initial set up. We give details on NN architecture size, and highlight that our hidden layer uses sigmoid activation functions while the output is equipped with a linear activation function. The burn-in perior of the HMC chain is the number of samples which are automatically not included in the posterior but help initialize the chain prior to use of the Metropolis-Rosenbluth-Hastings correction step Neal [51]. For each environmental set up we employ a leap-frog numerical integrator with 10 steps. The prior for all NN architectures is selected based on 2 times the variance perscribed by Glorot and Bengio [27] which has shown to be an empirically well-performing prior in previous works Wicker et al. [63,64]. The likelihood used to fit the BNN dynamics model is a mean squared error (ℓ_2) loss function.

When using variations inference, we use 1500 epochs to fit a posterior approximated by Stochastic Weight Averaging Guassian (SWAG) which does not admit a weight-space prior. We use a learning rate of 0.025 and a decay of 0.1.

Neural policy parameters In our experiments, we employ an NN policy, π_θ, comprised of a single hidden layer with 36 neurons. This is first trained to mimic actions that are sampled randomly at uniform. This training is done with SGD and is done for 15000 sampled states and actions. The NN policy is trained with 100 epochs of stochastic gradient descent with learning rate 0.00075 every time it is trained. This occurs after a BNN has been fit and used to update the actions according to loss presented in Section 5 save for no adversarial noise is taken into consideration. When we do perform synthesis, all of the same parameters are used: 15000 actions are considered and updated in parallel by SGD w.r.t. the adversarial loss defined in Section 5.

A.3. Further scalability experiments

In this section we provide further analysis and discussion of experiments on BNN depth in our framework and provide further experiments exploring how our method scales with state-space dimensions.

Table A.3
Episodic learning hyperparameter values corresponding to each problem setting we study.

Episodic Learning Parameters

	# Episodes	# Trajectories	Max Horizon	Policy Size	c
`V1`	15	20	25	$35 \times 35 \times 5 \times 5$	0.25
`V2`	10	15	45	$35 \times 35 \times 5 \times 5$	0.25
`Zigzag`	25	15	35	$25 \times 25 \times 3 \times 3$	0.125

Table A.4
Hyperparameters for learning BNN dynamics model with Hamiltonian Monte Carlo.

BNN Learning Parameters

	# Layers	# Neurons	Activations	Samps.	Burn In	LR	Decay
`V1`	1	50	sigmoid	500	25	0.05	0.1
`V2`	1	50	sigmoid	500	5	0.05	0.1
`Zigzag`	1	50	sigmoid	250	15	0.1	0.1

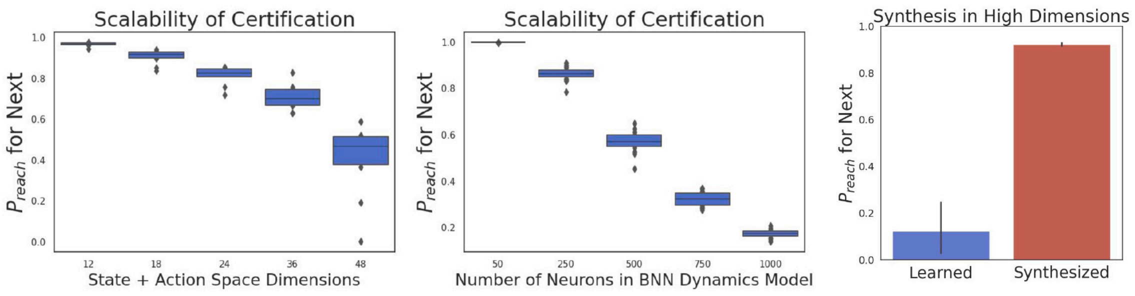

Fig. A.8. Left: Certified lower bound for forward invariance with respect to an increasing number of state-space dimensions. **Centre:** Certified lower bound as a function of the width of the BNN **Right:** Comparison between synthesised policy and learned policy on the 12-dimensional puck problem.

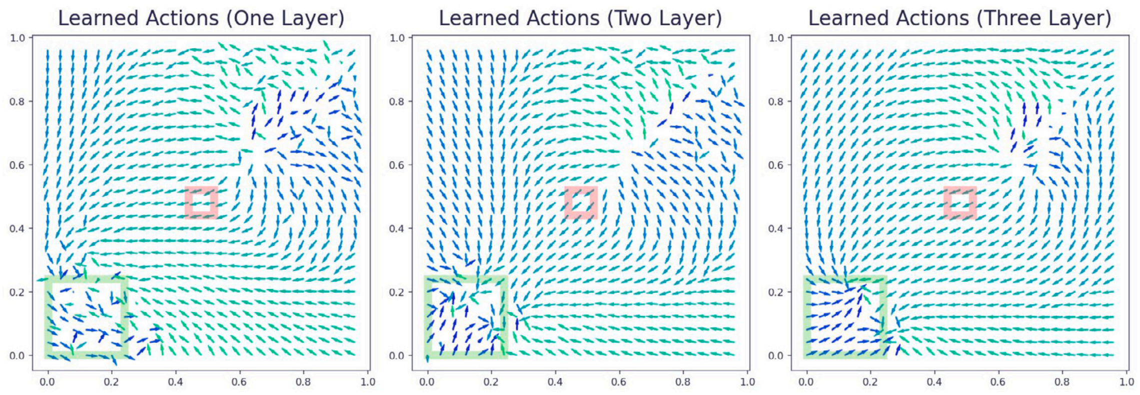

Fig. A.9. We plot the learned policies corresponding to each of the dynamical systems whose certification is visualized in Fig. 7. The left plot is the policy learned along with a one-layer BNN, the centre plot is the policy learned along with a two-layer BNN, and the right plot is the policy learned along with a three-layer BNN.

A.3.1. Further detail on BNN depth experiments

In Fig. A.9 we plot the policies that correspond to each of the BNNs learned for our depth experiments discussed in Section 6.4 and visualized in Fig. 7. We highlight that each of the policies are qualitatively very similar, though they may have slight quantitative differences. In Fig. A.10 we plot the result of the more computationally expensive certification on the three-layer BNN. Though our method struggles to get strong certification for the system with a three-layer BNN in Fig. 7, by tuning the certification parameters we are able to get a much tighter lower-bound (average lower-bound safety probability $0.621 \rightarrow 0.867$). We notice that many of the previously uncertified (i.e., lower-bound probability 0.0) states have lower-bounds above 0.6 when more BNN samples are used in the certification procedure.

A.3.2. Scaling with state-space dimensionality

In this section, we evaluate how our method performs while varying the dimensionality of the environment and the size of the BNN architecture. In particular we perform the analyses using an n-dimensional generalizations of the `v1` layout - described by $3n$ continuous values, n dimensions for position, velocity, and action spaces, respectively. For such high-dimensional state space, full discretisation of the state space becomes infeasible. In order to overcome this, we consider a forward-invariance variant of the

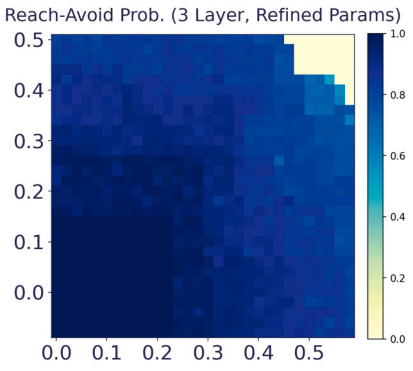

Fig. A.10. Lower-bound reach-avoid probabilities for the three layer BNN after refining the certification parameters from the procedure in Fig. 7.

reach-avoid property from our previous experiments, where the agent goal is iteratively moved at each step in order to guide it to the global goal. In other words, here we consider one-step reachability ($N = 1$), which allows us to significantly reduce the set of discretised states to consider (by restricting to neighbour states that can be reached in one step only).

The results for these analyses are given in Fig. A.8. The left plot of the figure, shows how even for 48 dimensions we are still able to obtain non-vacous bounds at 0.5, but as expected, the quality of the bound decreases quickly with the size of the state space and actions. The centre plot depicts the certified bounds obtained for an increasing number of BNN hidden units, up until 1000 for the 12-dimensional `v1` scenario. Finally, the right plot of Fig. A.8 shows that our synthesis algorithm strongly improves on the initially learned policy, even in higher-dimensional settings. We accomplish this improvement by following the neural policy synthesis method presented in Section 5 where the worst-case ϵ for tuning our actions is set to 0.025.

This analysis of the forward invariance property allows us to understand how our algorithm, particularly the evaluation of the R function, scales to larger NNs and state spaces. However, for state-space dimensions that are greater than the ones analyzed in the prior section of this paper, certification of the entire state-space is computationally infeasible due to the exponential nature of the discretization involved.

Appendix B. Proofs

Proof of Proposition 1. In what follows, we omit π (which is given and held constant) from the probabilities for a more compact notation. The proof is by induction. The base case is $k = N$, for which we have

$$V_N^\pi(x) = \mathbf{1}_{\mathrm{G}}(x) = P_{reach}(\mathrm{G}, \mathrm{S}, x, [N, N]),$$

which holds trivially. Under the assumption that, for any given $k \in [0, N-1]$, it holds that

$$V_{k+1}^\pi(x) = P_{reach}(\mathrm{G}, \mathrm{S}, x, [k+1, N]), \tag{B.1}$$

we show the induction step for time step k. In particular,

$$P_{reach}(\mathrm{G}, \mathrm{S}, x, [k, N] | \pi) =$$

$$Pr(\mathbf{x}_k \in \mathrm{G} | \mathbf{x}_k = x) + \sum_{j=k+1}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_k = x) =$$

$$\mathbf{1}_{\mathrm{G}}(x) + \mathbf{1}_{\mathrm{S}}(x) \sum_{j=k+1}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_k = x)$$

Now in order to conclude the proof we want to show that

$$\sum_{j=k+1}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k, j) + 1, \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_k = x) = \int V_{k+1}^\pi(\bar{x}) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x}.$$

This can be done as follows

$$\sum_{j=k+1}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k+1, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_k = x) =$$

$$Pr(\mathbf{x}_{k+1} \in \mathrm{G} | \mathbf{x}_k = x) +$$

$$\sum_{j=k+2}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k+1, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_k = x) =$$

$$\int_{\mathrm{G}} p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} +$$

$$\sum_{j=k+2}^{N} \int_{\mathrm{S}} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k+2, j), \mathbf{x}_{j'} \in \mathrm{S} \wedge \mathbf{x}_{k+1} = \bar{x} | \mathbf{x}_k = x) d\bar{x} =$$

$$\int_{\mathrm{G}} p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} +$$

$$\sum_{j=k+2}^{N} \int_{\mathrm{S}} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k+2, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_{k+1} = \bar{x}) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} =$$

$$\int \big(\mathbf{1}_{\mathrm{G}}(\bar{x}) +$$

$$\mathbf{1}_{\mathrm{S}}(\bar{x}) \sum_{j=k+2}^{N} Pr(\mathbf{x}_j \in \mathrm{G} \wedge \forall j' \in [k+2, j), \mathbf{x}_{j'} \in \mathrm{S} | \mathbf{x}_{k+1} = \bar{x})\big) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} =$$

$$\int V^{\pi}_{k+1}(\bar{x}) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x}$$

where the third step holds by application of Bayes rule over multiple events.

Proof of Theorem 1. The proof is by induction. The base case is $k = N$, for which we have

$$\inf_{x \in q} V^{\pi}_{N}(x) = \inf_{x \in q} \mathbf{1}_{\mathrm{G}}(x) = \mathbf{1}_{\mathrm{G}}(q) = K^{\pi}_{N}(q).$$

Next, under the assumption that for any $k \in \{0, N-1\}$ it holds that

$$\inf_{x \in q} V^{\pi}_{k+1}(x) \geq K^{\pi}_{k+1}(q),$$

we can work on the induction step: in order to derive it, it is enough to show that for any $\epsilon > 0$

$$\int V^{\pi}_{k+1}(\bar{x}) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} \geq$$

$$F([-\epsilon, \epsilon] | \sigma^2)^n \sum_{i=1}^{n_p} \int_{H^{q,\pi}_{k,i}} v_{i-1} p_{\mathbf{w}}(w | \mathcal{D}) dw,$$

where $F([-\epsilon, \epsilon] | \sigma^2) = \mathrm{erf}(\frac{\epsilon}{\sqrt{2\sigma^2}})$ is the cumulative function distribution for a normal random variable with zero mean and variance σ^2 being within $[-\epsilon, \epsilon]$. This can be argued by rewriting the first term in parameter space (recall that the stochastic kernel T is induced by $p_{\mathbf{w}}(w|\mathcal{D})$) and providing a lower bound, as follows:

$$\int V^{\pi}_{k+1}(\bar{x}) p(\bar{x} \mid (x, \pi_k(x)), \mathcal{D}) d\bar{x} =$$

(By definition of predictive distribution)

$$\int \Big(\int V^{\pi}_{k+1}(\bar{x}) p(\bar{x} | (x, u), w) d\bar{x} \Big) p_{\mathbf{w}}(w | \mathcal{D}) dw \geq$$

(By V^{k}_{k+1} being non negative everywhere and by the Gaussian likelihood)

$$\int \Big(\int_{f^{w}(x, \pi(x,k)) + \epsilon}^{f^{w}(x, \pi(x,k) - \epsilon} V^{\pi}_{k+1}(\bar{x}) \mathcal{N}(\bar{x} | f^{w}(x, \pi(x,k)), \sigma^2 \cdot I) d\bar{x} \Big) p_{\mathbf{w}}(w | \mathcal{D}) dw \geq$$

(By standard inequalities of integrals)

$$\int \inf_{\bar{\gamma} \in [-\epsilon, \epsilon]} V^{\pi}_{k+1}(f^{w}(x, \pi(x,k) + \bar{\gamma}) \Big(\int_{[-\epsilon, \epsilon]^n} \mathcal{N}(\gamma | 0, \sigma^2) d\gamma \Big)^n p_{\mathbf{w}}(w | \mathcal{D}) dw \geq$$

(By the assumptions that for $i \neq j$ $H^{q,\pi}_{k,i}$ and $H^{q,\pi}_{k,j}$ are non-overlapping)

$$\Big(\int_{[-\epsilon,\epsilon]} \mathcal{N}(\gamma|0,\sigma^2)d\gamma\Big)^n \sum_{i=1}^{n_p} \int_{H_{k,i}^{q,\pi,\epsilon}} \inf_{\bar{\gamma}\in[-\epsilon,\epsilon]} V_{k+1}^{\pi}(f^{w}(x,\pi(x,k)+\bar{\gamma})p_{\mathbf{w}}(w|\mathcal{D})dw,$$

(By the fact that $v_i \leq \inf_{x\in q} V_{k+1}^{\pi}(f^{w}(x,\pi(x,k)+\bar{\gamma})\,)$

$$\Big(\int_{[-\epsilon,\epsilon]} \mathcal{N}(\gamma|0,\sigma^2)d\gamma\Big)^n \sum_{i=1}^{n_p} v_i \int_{H_{k,i}^{q,\pi,\epsilon}} p_{\mathbf{w}}(w|\mathcal{D})dw,$$

where the last step concludes the proof because, by the induction hypothesis, we know that for $q' \subseteq \mathbb{R}^n$

$$\inf_{\bar{x}\in q'} V_{k+1}^{\pi}(\bar{x}) \geq K_{k+1}^{\pi}(q')$$

and by the construction of sets $H_{k,i}^{q,\pi}$ for each of its weights $K_{k+1}^{\pi}(f^{w}(\bar{x},\pi(x,k))$ is lower bounded by v_{i-1}.

References

[1] Alessandro Abate, Maria Prandini, John Lygeros, Shankar Sastry, Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems, Automatica 44 (11) (2008) 2724–2734.

[2] Steven Adams, Morteza Lahijanian, Luca Laurenti, Formal control synthesis for stochastic neural network dynamic models, arXiv preprint, arXiv:2203.05903, 2022.

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, Ufuk Topcu, Safe reinforcement learning via shielding, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[4] Karl J. Astrom, Richard M. Murray, Feedback Systems: An Introduction for Scientists and Engineers, Princeton University Press, Princeton, NJ, USA, 2008.

[5] Anish Athalye, Nicholas Carlini, David Wagner, Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples, in: International Conference on Machine Learning, PMLR, 2018, pp. 274–283.

[6] Guy Avni, Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, Bettina Könighofer, Stefan Pranger, Run-time optimization for learned controllers through quantitative games, in: International Conference on Computer Aided Verification, Springer, 2019, pp. 630–649.

[7] Edoardo Bacci, David Parker, Probabilistic guarantees for safe deep reinforcement learning, in: International Conference on Formal Modeling and Analysis of Timed Systems, Springer, 2020, pp. 231–248.

[8] Felix Berkenkamp, Angela P. Schoellig, Andreas Krause, Safe controller optimization for quadrotors with Gaussian processes, in: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 493–496.

[9] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, Andreas Krause, Safe model-based reinforcement learning with stability guarantees, in: NIPS, 2017.

[10] Leonard Berrada, Sumanth Dathathri, Krishnamurthy Dvijotham, Robert Stanforth, Rudy R. Bunel, Jonathan Uesato, Sven Gowal, M. Pawan Kumar, Make sure you're unsure: a framework for verifying probabilistic specifications, Adv. Neural Inf. Process. Syst. 34 (2021) 11136–11147.

[11] Dimitir P. Bertsekas, Steven Shreve, Stochastic Optimal Control: the Discrete-Time Case, Athena Scientific, 2004.

[12] Arno Blaas, Andrea Patane, Luca Laurenti, Luca Cardelli, Marta Kwiatkowska, Stephen Roberts, Adversarial robustness guarantees for classification with Gaussian processes, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 3372–3382.

[13] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, Daan Wierstra, Weight uncertainty in neural network, in: International Conference on Machine Learning, PMLR, 2015, pp. 1613–1622.

[14] Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J. Kochenderfer, Jana Tumova, Reinforcement learning with probabilistic guarantees for autonomous driving, arXiv preprint, arXiv:1904.07189, 2019.

[15] Ginevra Carbone, Matthew Wicker, Luca Laurenti, Andrea Patane', Luca Bortolussi, Guido Sanguinetti, Robustness of Bayesian neural networks to gradient-based attacks, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 15602–15613.

[16] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, Matthew Wicker, Statistical guarantees for the robustness of Bayesian neural networks, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, in: International Joint Conferences on Artificial Intelligence Organization, vol. 7, 2019, pp. 5693–5700, https://doi.org/10.24963/ijcai.2019/789.

[17] Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Andrea Patane, Robustness guarantees for Bayesian inference with Gaussian processes, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 7759–7768.

[18] Nathalie Cauchi, Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Marta Kwiatkowska, Luca Cardelli, Efficiency through uncertainty: scalable formal synthesis for stochastic hybrid systems, in: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, 2019, pp. 240–251.

[19] Marc Peter Deisenroth, Carl Edward Rasmussen, PILCO: a model-based and data-efficient approach to policy search, in: Proceedings of the International Conference on Machine Learning, 2011.

[20] S. Depeweg, J.M. Hernández-Lobato, F. Doshi-Velez, S. Udluft, Learning and policy search in stochastic dynamical systems with Bayesian neural networks, in: 5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings, 2017.

[21] Guido Fubini, Sugli integrali multipli, Rend. Accad. Naz. Lincei 16 (1907) 608–614.

[22] Nathan Fulton, André Platzer, Verifiably safe off-model reinforcement learning, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2019, pp. 413–430.

[23] Yarin Gal, Rowan McAllister, Carl Edward Rasmussen, Improving PILCO with Bayesian neural network dynamics models, in: International Conference in Machine Learning (ICML), 2016.

[24] Yarin Gal, Rowan Thomas McAllister, Carl Edward Rasmussen, Improving PILCO with Bayesian neural network dynamics models, in: Data-Efficient Machine Learning Workshop, vol. 951, 2016, p. 2016.

[25] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, Martin Vechev, Ai2: safety and robustness certification of neural networks with abstract interpretation, in: 2018 IEEE S&P, IEEE, 2018, pp. 3–18.

[26] Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, Roderick Murray-Smith, Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting, in: Advances in Neural Information Processing Systems, 2003, pp. 545–552.

[27] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[28] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.

[29] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, Pushmeet Kohli, On the effectiveness of interval bound propagation for training verifiably robust models, in: Neural Information Processing Systems (NeurIPS), 2018.
[30] Kathrin Grosse, David Pfaff, Michael Thomas Smith, Michael Backes, How wrong am I? - Studying adversarial examples and their impact on uncertainty in Gaussian process machine learning models, arXiv preprint, arXiv:1711.06598, 2017.
[31] S. Haesaert, P.M.J.V.d. Hof, A. Abate, Data-driven and model-based verification via Bayesian identification and reachability analysis, Automatica 79 (5) (2017) 115–126.
[32] M. Hasanbeig, D. Kroening, A. Abate, Deep reinforcement learning with temporal logics, in: Proceedings of FORMATS, in: LNCS, vol. 12288, 2020, pp. 1–22.
[33] Mohammadhosein Hasanbeig, Alessandro Abate, Daniel Kroening, Certified reinforcement learning with logic guidance, arXiv preprint, arXiv:1902.00778, 2019.
[34] Jingyi Huang, Andre Rosendo, Deep vs. deep Bayesian: reinforcement learning on a multi-robot competitive experiment, arXiv preprint, arXiv:2007.10675, 2020.
[35] Pavel Izmailov, Sharad Vikram, Matthew D. Hoffman, Andrew Gordon Gordon Wilson, What are Bayesian neural network posteriors really like?, in: International Conference on Machine Learning, PMLR, 2021, pp. 4629–4640.
[36] John Jackson, Luca Laurenti, Eric Frew, Morteza Lahijanian, Safety verification of unknown dynamical systems via Gaussian process regression, in: 2020 59th IEEE Conference on Decision and Control (CDC), IEEE, 2020, pp. 860–866.
[37] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, Mykel J. Kochenderfer Reluplex, An efficient SMT solver for verifying deep neural networks, in: International Conference on Computer Aided Verification, Springer, 2017, pp. 97–117.
[38] Mohammad Emtiyaz Khan, Håvard Rue, The Bayesian learning rule, arXiv preprint, arXiv:2107.04562, 2021.
[39] Bettina Könighofer, Roderick Bloem, Sebastian Junges, Nils Jansen, Alex Serban, Safe reinforcement learning using probabilistic shields, in: International Conference on Concurrency Theory: 31st CONCUR 2020: Vienna, Austria (Virtual Conference), Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2020.
[40] Marta Kwiatkowska, Gethin Norman, David Parker, Stochastic model checking, in: International School on Formal Methods for the Design of Computer, Communication and Software Systems, Springer, 2007, pp. 220–270.
[41] Mathias Lechner, Đorđe Žikelić, Krishnendu Chatterjee, Thomas Henzinger, Infinite time horizon safety of Bayesian neural networks, Adv. Neural Inf. Process. Syst. 34 (2021).
[42] Faming Liang, Bayesian neural networks for nonlinear time series forecasting, Stat. Comput. 15 (1) (2005) 13–29.
[43] Xuanqing Liu, Yao Li, Chongruo Wu, Cho-Jui Hsieh, Adv-bnn: improved adversarial defense through robust Bayesian neural network, in: 7th International Conference on Learning Representations, ICLR 2019-Conference Track Proceedings, 2019.
[44] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, Towards deep learning models resistant to adversarial attacks, arXiv preprint, arXiv:1706.06083, 2017.
[45] Andres Masegosa, Learning under model misspecification: applications to variational and ensemble methods, Adv. Neural Inf. Process. Syst. 33 (2020) 5479–5491.
[46] Rowan McAllister, Carl Edward Rasmussen, Data-efficient reinforcement learning in continuous state-action Gaussian-pomdps, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.
[47] Rhiannon Michelmore, Matthew Wicker, Luca Laurenti, Luca Cardelli, Yarin Gal, Marta Kwiatkowska, Uncertainty quantification with statistical guarantees in end-to-end autonomous driving control, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 7344–7350.
[48] Matthew Mirman, Timon Gehr, Martin Vechev, Differentiable abstract interpretation for provably robust neural networks, in: International Conference on Machine Learning, PMLR, 2018, pp. 3578–3586.
[49] Kevin P. Murphy, Machine Learning: a Probabilistic Perspective, MIT Press, 2012.
[50] Pavel Myshkov, Simon Julier, Posterior distribution analysis for Bayesian inference in neural networks, in: Workshop on Bayesian Deep Learning, NIPS, 2016.
[51] M. Radford Neal, Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media, 2012.
[52] M. Radford Neal, et al., Mcmc using Hamiltonian dynamics, in: Handbook of Markov Chain Monte Carlo, vol. 2, 2011, p. 2.
[53] T. Dung Phan, Radu Grosu, Nils Jansen, Nicola Paoletti, Scott A. Smolka, Scott D. Stoller, Neural simplex architecture, in: NASA Formal Methods Symposium, Springer, 2020, pp. 97–114.
[54] Kyriakos Polymenakos, Alessandro Abate, Stephen Roberts, Safe policy search using Gaussian process models, in: Proceedings of the 18th International Conference on Autonomous Agents and Multi Agent Systems, IFAAMS, 2019, pp. 1565–1573.
[55] Kyriakos Polymenakos, Luca Laurenti, Andrea Patane, Jan-Peter Calliess, Luca Cardelli, Marta Kwiatkowska, Alessandro Abate, Stephen Roberts, Safety guarantees for iterative predictions with Gaussian processes, in: 2020 59th IEEE Conference on Decision and Control (CDC), IEEE, 2020, pp. 3187–3193.
[56] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al., Mastering Atari, Go, chess and shogi by planning with a learned model, arXiv preprint, arXiv:1911.08265, 2019.
[57] S. Esmaeil Zadeh Soudjani, A. Abate, Probabilistic reach-avoid computation for partially-degenerate stochastic processes, IEEE Trans. Autom. Control 58 (12) (2013) 528–534.
[58] Xiaowu Sun, Haitham Khedr, Yasser Shoukry, Formal verification of neural network controlled autonomous systems, in: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, 2019, pp. 147–156.
[59] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction, 1998.
[60] Julia Vinogradska, Bastian Bischoff, Duy Nguyen-Tuong, Anne Romer, Henner Schmidt, Jan Peters, Stability of controllers for Gaussian process forward models, in: International Conference on Machine Learning, PMLR, 2016, pp. 545–554.
[61] Tianhao Wei, Changliu Liu, Safe control with neural network dynamic models, arXiv preprint, arXiv:2110.01110, 2021.
[62] Matthew Wicker, Adversarial robustness of Bayesian neural networks, PhD thesis, University of Oxford, 2021.
[63] Matthew Wicker, Luca Laurenti, Andrea Patane, Marta Kwiatkowska, Probabilistic safety for Bayesian neural networks, in: Jonas Peters, David Sontag (Eds.), Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), in: Proceedings of Machine Learning Research, vol. 124, PMLR, 03–06 Aug 2020, pp. 1198–1207.
[64] Matthew Wicker, Luca Laurenti, Andrea Patane, Zhuotong Chen, Zheng Zhang, Marta Kwiatkowska, Bayesian inference with certifiable adversarial robustness, in: Arindam Banerjee, Kenji Fukumizu (Eds.), Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 130, PMLR, 13–15 Apr 2021, pp. 2431–2439.
[65] Matthew Wicker, Luca Laurenti, Andrea Patane, Nicola Paoletti, Alessandro Abate, Marta Kwiatkowska, Certification of iterative predictions in Bayesian neural networks, in: Uncertainty in Artificial Intelligence, PMLR, 2021, pp. 1713–1723.
[66] Matthew Robert Wicker, Juyeon Heo, Luca Costabello, Adrian Weller, Robust explanation constraints for neural networks, in: The Eleventh International Conference on Learning Representations, 2022.
[67] V. Wijesuriya, A. Abate, Bayes-adaptive planning for data-efficient verification of uncertain Markov decision processes, in: Proceedings of QEST, in: LNCS, vol. 11785, 2019, pp. 91–108.