Learning HJB Viscosity Solutions with PINNs for Continuous-Time Reinforcement Learning

Alena Shilova, Thomas Delliaux, Philippe Preux, Bruno Raffin

Learning HJB Viscosity Solutions with PINNs for Continuous-Time Reinforcement Learning

Alena Shilova, Thomas Delliaux, Philippe Preux , Bruno Raffin

RESEARCH REPORT
N° 9541
February 2024
Project-Team Scool

ISSN 0249-6399 ISRN INRIA/RR--9541--FR+ENG

Learning HJB Viscosity Solutions with PINNs for Continuous-Time Reinforcement Learning

Alena Shilova*, Thomas Delliaux*, Philippe Preux †, Bruno Raffin ‡

Project-Team Scool

Research Report n° 9541 — February 2024 — 30 pages

Abstract: Despite recent advances in Reinforcement Learning (RL), the Markov Decision Processes are not always the best choice to model complex dynamical systems requiring interactions at high frequency. Being able to work with arbitrary time intervals, Continuous Time Reinforcement Learning (CTRL) is more suitable for those problems. Instead of the Bellman equation operating in discrete time, it is the Hamilton-Jacobi-Bellman (HJB) equation that describes value function evolution in CTRL. Even though the value function is a solution of the HJB equation, it may not be its unique solution. To distinguish the value function from other solutions, it is important to look for the viscosity solutions of the HJB equation. The viscosity solutions constitute a special class of solutions that possess uniqueness and stability properties. This paper proposes a novel approach to approximate the value function by training a physics informed neural network (PINN) through a specific ϵ-*scheduling* iterative process constraining the PINN to converge towards the viscosity solution and shows experimental results with classical control tasks, where PINNs outperform popular RL algorithms in a nearly continuous-time setting.

Key-words: physics-informed neural networks, viscosity solutions, continuous time reinforcement learning, optimal control

* Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL, Lille, France
† Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 - CRIStAL, Lille, France
‡ Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, Grenoble, France

RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

Apprentissage des solutions de viscosité HJB avec PINNs pour l'apprentissage par renforcement en temps continu

Résumé : Malgré les progrès récents en matière d'apprentissage par renforcement (RL), les processus décisionnels de Markov ne constituent pas toujours le meilleur choix pour modéliser des systèmes dynamiques complexes nécessitant des interactions à haute fréquence. Étant capable de travailler avec des intervalles de temps arbitraires, l'apprentissage par renforcement en temps continu (CTRL) est plus adapté à ces problèmes. Au lieu de l'équation de Bellman fonctionnant en temps discret, c'est l'équation de Hamilton-Jacobi-Bellman (HJB) qui décrit l'évolution de la fonction valeur dans CTRL. Même si la fonction valeur est une solution de l'équation HJB, elle n'en est peut-être pas l'unique solution. Pour distinguer la fonction valeur des autres solutions, il est important de rechercher les solutions de viscosité de l'équation HJB. Les solutions de viscosité constituent une classe particulière de solutions possédant des propriétés uniques et de stabilité. Cet article propose une nouvelle approche pour approximer la fonction de valeur en entraînant un réseau neuronal informé par la physique (PINN) à travers un processus itératif de ϵ-*scheduling* contraignant le PINN à converger vers la solution de viscosité et montre des résultats expérimentaux avec tâches de contrôle classiques, dans lesquelles les PINN surpassent les algorithmes RL populaires dans un contexte de temps presque continu.

Mots-clés : réseaux de neurones basés sur la physique, solutions de viscosité, apprentissage par renforcement en temps continu, contrôle optimal

Contents

1 Introduction

Reinforcement learning (RL) is getting more and more attention. Most of state-of-the-art RL methods are designed to work with Markov Decision Processes (MDPs) and, in particular, rely on a discrete time assumption. Even though this assumption is not that restrictive in some tasks like games, it is no longer valid in complex problems such as driving cars, finance trading or controling a dynamical system. Such problems are usually described by a dynamical system where discrete time RL (DTRL) often struggles to provide accurate control within short time intervals due to several reasons [9, 37, 27]. As DTRL algorithms operate in regular discrete time steps, it becomes challenging to capture the nuances of continuous state dynamics within small intervals. Even when the timestep is set to be small, DTRL may fail to learn the optimal policy as exploration cannot be done efficiently.

Continuous Time Reinforcement Learning (CTRL) derives from the optimal control theory and thus provides a promising theoretical framework to tackle aforementioned shortcomings of DTRL. CTRL is agnostic on the discretization of time and thus can provide a good control at any chosen frequency without the need to retrain the policy. Similar to DTRL, one way to find an optimal policy is to compute it using the value function. Thus, the key ingredient in CTRL is the Hamilton-Jacobi-Bellman (HJB) equation [9, 29], a Partial Differential Equation (PDE), which is a continuous-time counterpart of the Bellman equation. This PDE describes the evolution of the value function with respect to a state of the system, which in turn evolves continuously with time. In principle, the value function is a solution of the HJB equation. However, it is only a necessary condition and not a sufficient one. The HJB equation may have multiple solutions in a class of continuous functions [29] making it a challenging task to find the value function. Distinguishing the value function from the other solutions of the HJB equation relies on the search for the *viscosity* solution that possesses uniqueness and stability properties.

However, there is no guarantee in general that the solution found by existing methods is the good solution, *i.e.* the viscosity solution. Moreover, even verifying if a given solution is a viscosity solution is a hard task, let alone finding them. The existing methods for solving the HJB equation are limited and mainly applicable to specific cases such as Linear Quadratic Regulator (LQR) problems or variational problems [11]. There exist approaches that can be applied in general case, such as Finite Difference (FD), Finite Element Methods (FEM) [12], and dynamic programming methods [29] that transform a CTRL problem back to a DTRL problem thanks to the discretization. However, their effectiveness is hindered by the exponentially growing algorithmic complexity with respect to the state space dimensionality. In addition, they are also affected by discretization error.

Conversely, neural networks (NNs) are emerging as PDE solvers that do not require discretization of the domain and thus they better cope with the curse of dimensionality [32]. Although neural networks have shown a great potential in various domains [23, 18], applying them to solve the HJB equation requires caution because of the aforementioned non-uniqueness issue.

In this paper, we revisit CTRL approaches and analyze how the latest advances in deep learning can be applied and adjusted to solve the HJB equation in the viscosity sense. The main contribution of this paper is to provide a NN based framework to find the viscosity solutions of the general HJB equation. To the best of our knowledge this is the first attempt to do it. We focus on the case of deterministic environments with known dynamics. The extension to stochastic environments and unknown dynamics is possible (see [37, 42]), but left for future work. To find viscosity solutions, our approach is to solve sequentially a series of PDEs so that the solution of the final one is the value function. We use NN solvers for those PDEs and we propose several ways of building the sequence of those equations. This work can be interesting for the optimal control community as we show how to use the neural networks to get the viscosity solutions and

for the reinforcement learning community as our work can be used as a basis for Model Based Continuous Time Reinforcement Learning.

This paper is organized as follows: we start with the analysis of the existing literature in Section 2. Then, we carefully introduce the definitions and notations related to CTRL in Section 3.1, describe the HJB equation in Section 3.2 and define viscosity solutions in Section 3.3. Then, different ways of integrating neural networks in the process of solving the HJB equation are discussed in Section 4. Finally, we demonstrate the performance of our approach in Section 4 on classical control tasks from RL, and show that our method performs significanly better than some popular DTRL algorithms in almost continuous-time setting in Section 5.

2 Related Works

[9, 29, 5] are among the first papers to study CTRL. Those works introduced the HJB equation as a key equation for finding the optimal policy. In [9], different discretization schemes and algorithms are analysed for computing the value function, including continuous TD(λ) and continuous Actor-Critic. In his Ph.D. [5], Coulom studied the applicability of Doya's methods [9] to a large class of control problems. Munos [29] tackled CTRL by the study of viscosity solutions and their properties. He demonstrated the challenges of solving the HJB equation such as the non-uniqueness of solutions and handling inequality boundary conditions. In addition, convergent numerical schemes based on dynamic programming were derived to approximate the value function. One of the problems of numerical schemes is the curse of dimensionality. To mitigate this problem, sparse grids [17] can be used instead of uniform grid. In this article, we take the formalism proposed in [29], but we consider neural network based approaches to find viscosity solutions, while [29] considers tabular algorithms.

As it was first demonstrated in [28], NNs can be applied for solving the HJB equation [22, 4, 34, 24, 14, 1]. In [34], the training is regularized to avoid converging to "bad" solutions. Moreover, the same work raised the problem of falling in a local minimum of the squared HJB residual, and some solutions to avoid it were proposed. Compared to previous papers, [22, 1] have emphasized the robustness of the resulting controller and the stability of the proposed algorithm. Adapting the former methods for more practical cases such as the inverted pendulum and the cartpole was done in [24]. We consider similar approaches as [34, 24], but unlike them we use the formalism proposed in [29] to develop an approach that can converge to viscosity solutions.

Several extensions to the classical HJB equation were also introduced such as HJB for an explorative reward function [37], the soft HJB equation with maximum entropy regularization [20, 13] and the distributional HJB equation [39]. Those works extend the existing theory to other definitions of the value function, however experiments are conducted on a limited set of simple problems. Futhermore, it is also possible to extend the HJB equation to continuous-time partially observable Markov decision processes (CTPOMDPs). The work of [2], proposed a formalism to describe CTPOMDPs, including the CTPOMDP HJB equation. [19] tackle the problem of CTRL by adapting the well-known DQN algorithm to this framework. A definition for the Q-function in the continuous case is given and the "HJB equation for the Q-function" is derived, which results in a DQN-like algorithm for the semi-discrete time setting. However, this approach is limited to Lipschitz continuous control, while there are a lot of cases where only discrete control is available. The HJB equation was used to improve DTRL algorithms like PPO in [27]. This resulted in a significant improvement on MuJoCo tasks, proving that HJB loss is better adapted for learning value functions of dynamical systems.

There were some attempts to propose alternative ways for solving the HJB equation. Another NN approach to approximate the value function based on Pontryagin's maximum principle

has been proposed [30], assuming the concavity of the reward function. They use numerical methods to collect a dataset of trajectories by solving the boundary value problem on short time intervals and then train a NN to fit them. Conversely, [7] considers some special neural network architectures inspired with min-plus algebra to solve some optimal control problems with linear dynamics and quadratic rewards.

Another line of research exploits the continuous time formulation to do more accurate Monte Carlo estimations of value functions [26, 25, 42] rather than using the HJB equation. [26] adapted the fitted value iteration algorithm to the CTRL problem. In [42], a model-based algorithm was introduced, which aims at solving the problem in an actor-critic manner. Bayesian neural ODEs are used in order to learn the dynamics of the system.

3 Hamilton-Jacobi-Bellman Equation

3.1 Reinforcement Learning Formalism in the Continuous Time Case

We consider the optimal control problem for infinite-horizon deterministic dynamical systems. The state dynamics and reward signal are known on the whole domain. The extension to unknown dynamics and rewards is left for future works.

In what follows, we denote with $O \subseteq \mathbb{R}^d$, the set of *controlable* states of our system, and then the *admissible* control $u \in U$ keeps the state trajectory inside the domain O, where $U \subset \mathbb{R}^m$ corresponds to the action/control space. We assume that U is bounded. We use $g \in C(O)$ to denote that g is a continuous function on O, while $g \in C^1(O)$ denotes that g is a continuously differentiable function on O.

The main difference between the continuous and discrete time cases is that transitions depend on time t, which is a continuous variable, generating trajectories of states continuously in time. More formally, in the continuous case the states do not form a sequence $\{x_t\}_{t=0}^{\infty}$ but a trajectory $x : \mathbb{R}_+ \to \bar{O} \subset \mathbb{R}^d$. Similarly, the actions are defined for any t: $u : \mathbb{R}_+ \to U$. The dynamics of the environment (the transition function) is defined through the following ordinary differential equation (ODE):

$$\frac{dx(t)}{dt} = f(x(t), u(t)) \tag{1}$$

where $f : \bar{O} \times U \to \mathbb{R}^d$ is called the state dynamics function. In the discrete case, $x(t)$ is only defined at times $\{0, dt, 2dt, \dots\}$, where dt is a time step. While the state x is inside the control domain O (*i.e.* $x(t) \in \bar{O}$), the reward $r : \bar{O} \times U \to \mathbb{R}$ is received.

Without loss of generality, we focus on the problem of optimal control under state constraints, *i.e.* $x(t) \in O$ for any $t > 0$. Note that constraining the system to stay inside O is a common requirement for dynamical systems. For example, we may want to limit the maximal angular speed in the inverted pendulum to mitigate the risk of wearing off or breaking the mechanism. The interested reader can refer to Appendix A.4 to learn more about other possible cases.

Given the initial state $x(0) = x_0$, we define the cumulative discounted reward for a given control function $u(t)$ as:

$$J(x_0; u(t)) = \int_0^{\infty} \gamma^t r(x(t), u(t)) dt, \tag{2}$$

where $\gamma \in [0, 1)$ is a discount factor. J is called the reinforcement functional. Then, the value function is defined as:

$$V(x) = \sup_{u(t) \in U_x} J(x; u(t)), \tag{3}$$

where $U_x = \{u(t)|x(t) \in O, \forall t > 0 \text{ and } x(0) = x\}$, *i.e.* a control that keeps the state of the system inside the domain O. Our goal is to find an optimal policy $\pi : \bar{O} \to U$, such that $u^*(t) = \pi(x^*(t))$ for any t, where $u^*(t)$ and $x^*(t)$ are the control and state of the optimal trajectory, then $V(x) = J(x; u^*(t))$.

3.2 Hamilton-Jacobi-Bellman Equation

Let $L(x, u, p) = p^T f(x, u) + r(x, u)$ and $H(x, W, \nabla_x W) = -W \ln \gamma - \sup_{u \in \mathcal{U}} L(x, u, \nabla_x W)$, then the HJB equation can be expressed as

$$H(x, W, \nabla_x W) = 0. \tag{4}$$

We can prove that the value function defined in equation 3 satisfies the following result (see [11]):

Theorem 3.1. *If the value function V is differentiable at x, then it should satisfy the Hamilton-Jacobi-Bellman equation.*

If the value function $V(x)$ is known, then we can define a feed-back control policy $\pi : \bar{O} \to U$ such that $\pi(x(t)) = u^*(t)$ by setting:

$$\pi(x) \in \underset{u \in U}{\operatorname{argsup}} \left\{ \nabla_x V(x)^T f(x, u) + r(x, u) \right\}. \tag{5}$$

In practice, if for some values u, $L(x, u, \nabla_x V) \approx \max_{u'} L(x, u', \nabla_x V)$, then it is possible to pick up a wrong control because of numerical instabilities.

One way to bypass this issue is to consider, similarly to [20], a stochastic policy with a probability distribution for a discrete set $\mathcal{U}$:

$$\pi^*(x, u) = \frac{\exp(\alpha L(x, u, \nabla_x V))}{\sum_{u' \in \mathcal{U}} \exp(\alpha L(x, u', \nabla_x V))}, \tag{6}$$

where α is an inverse temperature parameter. Compared to equation 5, equation 6 applies a softmax to compute the probability of the best control. Using π^*, one can compute the soft version of the HJB equation:

$$-W \ln \gamma - \mathbb{E}_{u \sim \pi^*} L(x, u, \nabla_x W) = 0. \tag{7}$$

In the following we mainly focus on the general HJB equation (equation 4), but this soft version can be beneficial for some practical cases as shown in our experiments.

When $O \not\equiv \mathbb{R}^d$, the control in state constrained optimal control problems should also satisfies $f(x, u^*(x))^T \eta(x) \leq 0$ for any $x \in \partial O$, where $\eta(x)$ the external normal vector at point $x \in \partial O$. The optimal policy is not known a priori and thus it is hard to verify this constraint. In [11, 33], it was shown that it can be reformulated as:

$$-H(x, W, \nabla_x W + \alpha \eta(x)) \leq 0 \quad \forall \alpha \leq 0, x \in \partial O. \tag{8}$$

From equation 5, it is crucial to find an efficient way to compute V to get the optimal control u^*. In DTRL, the Bellman equation is traditionally used to find V. The HJB equation can be seen as a continuous-time analog of the Bellman equation. However, solving the HJB equation involves several challenges (see [29]). First, the value function $V \in C(O)$, a continuous solution of the HJB equation, may be non-differentiable on O, therefore it cannot satisfy the HJB equation on the whole domain. Second, the HJB equation may have multiple generalized continuous solutions. Third, it is common that the HJB equation (equation 4) has to be solved under inequality boundary conditions. To address those points, we introduce the viscosity property.

3.3 Viscosity

Here, we present the crucial and yet complex notion of viscosity [11]. Refer to Appendix B for a more intuitive presentation.

Definition 3.2 (Viscosity solution)**.**

- $W \in C(O)$ is a viscosity subsolution of the HJB equation in O if $\forall \psi \in C^1(O)$ and $\forall x \in O$ local maximum of $W - \psi$ such that $W(x) = \psi(x)$, we have:

$$H(x, \psi(x), \nabla_x \psi(x)) \leq 0$$

- $W \in C(O)$ is a viscosity supersolution of the HJB equation in O if $\forall \psi \in C^1(O)$ and $\forall x \in O$ local minimum of $W - \psi$ such that $W(x) = \psi(x)$, we have:

$$H(x, \psi(x), \nabla_x \psi(x)) \geq 0$$

- If $W \in C(O)$ is a viscosity subsolution and a supersolution then it is a viscosity solution.

Viscosity solutions were first introduced in [6] and they are proven to be unique for multiple types of PDEs. Under some additional assumptions, which include continuity of f and r, one can prove that the value function is a *unique viscosity solution* for $O = \mathbb{R}^d$ (See Appendix A.3 for more details). A similar result exists for $O \subseteq \mathbb{R}^d$ and when the value function should satisfy the boundary condition equation 8.

A large class of practical control problems with continuous dynamics, like classical control or MuJoCo tasks [35], needs to find the unique viscosity solution of the HJB equation to get a correct value function. Thus, developping methods converging to the viscosity solution is the key factor for CTRL. However, checking the conditions of Definition 3.2 is not feasible in practice. Instead, we use the next property of viscosity solutions [11].

Lemma 3.3 (Stability)**.** *Let W^ϵ be a viscosity subsolution (resp. a supersolution) of*

$$W^\epsilon(x) + F^\epsilon(x, W^\epsilon, \nabla_x W^\epsilon, \nabla_x^2 W^\epsilon) = 0$$

in O. Suppose that F^ϵ converges to F uniformly on every compact subset of its domain, and W^ϵ converges to W uniformly on compact subsets of $\bar{O}$. Then W is a viscosity subsolution (resp. a supersolution) of the limiting equation.

In our case, we are interested in the equation:

$$H(x, W^\epsilon, \nabla_x W^\epsilon) = \epsilon \Delta_x W^\epsilon(x), \tag{9}$$

where the left hand side is the same as in equation 4, while the right hand side depends linearly on $\epsilon > 0$. Let $\Delta_x W$ be a Laplacian of W, then $F^\epsilon(x, W^\epsilon, \nabla_x W^\epsilon, \nabla_x^2 W^\epsilon) = -\frac{1}{\ln \gamma} H(x, W^\epsilon, \nabla_x W^\epsilon) - W^\epsilon(x) + \frac{\epsilon}{\ln \gamma} \Delta_x W^\epsilon(x)$ and $F(x, W, \nabla_x W, \nabla_x^2 W) = -\frac{1}{\ln \gamma} H(x, W, \nabla_x W) - W(x)$ in Lemma 3.3. In [11], it is shown that equation 9 has a unique smooth solution $W^\epsilon(x)$, *i.e.* it admits a classical solution, which is a viscosity solution at the same time. Therefore, if $W^\epsilon(x)$ converges uniformly to $W(x)$ (convergence of solutions) and $\epsilon \Delta_x W^\epsilon$ converges uniformly to 0 (convergence of equations), then $W(x)$ is a viscosity solution of the original HJB equation (equation 4).

4 How to Reach Viscosity

In what follows, we present two methods to find viscosity solutions of an HJB equation. First, we present the existing method based on dynamic programming. Then, we introduce a new neural approach. Further, we assume that the state dynamics $f(x, u)$ are known and the control space U is discrete. The case of unknown $f(x, u)$ and continuous control space is left for future work.

4.1 Dynamic Programming

Several solvers exist such as Finite Difference method (FD) or Finite Element Method (FEM). These methods require the discretization of the domain (a grid for FD or a triangulation for FEM). The work of [29] establishes the connection between solving the HJB equation and the classical reinforcement learning framework by deriving an MDP from the discretization of the HJB equation, using either FD or FEM schemes (see Appendix C for a short summary of the method). The strong point of this method is that there exist viscosity convergence guarantees [29]. The weak point is that they are mesh-dependent, making them sensible to the curse of dimensionality. For example, in the case of the cartpole problem where $O \subseteq \mathbb{R}^4$, a naive approach that divides all dimensions uniformly in N parts results in N^4 states. Setting $N = 32$, which may not be sufficient to solve the problem, leads to 2^{20} states, which is already too many to process on a single device. In Section 5, we present the performance of the FEM based dynamic programming only on the inverted pendulum environment due to the aforementioned reasons. Despite many efforts, we were not able to make the algorithm based on FD work in our experiments, thus it is not considered.

4.2 Neural Solver

The idea is to solve a series of PDE equations starting from some ϵ_0 and gradually decrease it so that the solution of the final PDE for $\epsilon_\infty = 0$ is the desired viscosity solution. Thus, the general framework to get viscosity solutions is the following: define the sequence of $\{\epsilon_n\}_{n=0}^{\infty}$, choose a PDE solver, then iteratively solve equation 9 so that $W^\epsilon(x)$ form a convergent sequence to $W(x)$ and output $W(x)$ as a final result. In what follows, we choose PINNs as a PDE solver and we define a few ϵ-*schedulers* to generate $\{\epsilon_n\}$.

PINNs The idea of PINNs was proposed in [32] and applied to some simple PDEs like one-dimensional nonlinear Schrödinger equation, but not for optimal control. In PINNs framework, the neural network acts as a solution of the PDE that needs to be solved. Being randomly initialized, the neural network is gradually fit to satisfy the PDE and its corresponding boundary conditions with the help of optimization and automatic differentiation that allows to compute precise derivatives. For example, the solution of the equation $W'_x - W = 0$ for $W \in C^1(O = [0, 1])$ with the boundary condition $W(0) = 1$ can be found by minimizing the loss $\min_W \{\|W'_x - W\|_2^2 + \lambda\|W(0) - 1\|_2^2\}$. The first term of this loss is called a PDE loss and the second term a boundary loss, where λ is a hyperparameter that weighs a boundary loss against a PDE loss. PINNs can be trained in a self-supervised manner as a dataset can be generated by simply drawing random samples from the domain O. Still, if the solution is known at some points of the domain, then the training can be augmented with a data-driven loss. Refer to Appendix D for a more detailed introduction to PINNs.

Further, we denote W_θ^ϵ and W_θ the two neural networks that compute the solutions of equation 9 and equation 4 respectively with θ being its parameters. In PINNs-like mannner, we define losses corresponding to equation 9 and equation 8. Let us define $\mathcal{S}_O \sim \mathcal{U}(O, N_F)$, a sample of

points drawn uniformly from O of size N_F, and $\mathcal{S}_{\partial O} \sim \mathcal{U}(\partial O, N_B)$. We have:

$$\mathcal{L}_O(\theta, \mathcal{S}_O) = \frac{1}{N_F} \sum_{x_i \in \mathcal{S}_O} \left(H(x_i, W_\theta^\epsilon, \nabla W_\theta^\epsilon) - \epsilon \mathrm{Tr}(\nabla^2 W_\theta^\epsilon)\right)^2 \tag{10}$$

$$\mathcal{L}_{\partial O}(\theta, \mathcal{S}_{\partial O}) = \frac{1}{N_B} \sum_{x_i \in \mathcal{S}_{\partial O}} \left([-H(x_i, W_\theta^\epsilon, \nabla W_\theta^\epsilon + \alpha\eta(x_i))]^+\right)^2 \tag{11}$$

where $[f(x)]^+ = \max\{f(x), 0\}$. In addition to PDE-related and boundary-related losses, we introduce an MSE regularization loss that should encourage uniform convergence of solutions:

$$\mathcal{L}_R(\theta, \mathcal{S}_O) = \frac{1}{N_F} \sum_{x_i \in \mathcal{S}_O} \left(W_\theta^\epsilon(x_i) - W_{\theta_{\epsilon_{n-1}}}^{\epsilon_{n-1}}(x_i)\right)^2 \tag{12}$$

where $\theta_{\epsilon_{n-1}}$ is the best parameters found for ϵ_{n-1}. The final loss is:

$$\mathcal{L}(\theta, \mathcal{S}_O, \mathcal{S}_{\partial O}) = \mathcal{L}_O(\theta, \mathcal{S}_O) + \lambda \mathcal{L}_{\partial O}(\theta, \mathcal{S}_{\partial O}) + \lambda_R \mathcal{L}_R(\theta, \mathcal{S}_O). \tag{13}$$

For simplicity, we use the same amount of points to sample from O and ∂O, *i.e.* $N_F = N_B$.

ϵ-schedulers Training is performed using multiple epochs t, where each epoch starts from generating datasets $\mathcal{D}_O$ and $\mathcal{D}_{\partial O}$ of samples from O and ∂O respectively. To ensure the uniform convergence of equation 9 to equation 4, we need to define a sequence of $\{\epsilon_n\}_{n=0}^\infty$, such that $\epsilon_n \to 0$ and $\epsilon_n \Delta_x W_\theta^{\epsilon_n}(x) \to 0$ uniformly for all $x \in \bar{O}$ when $n \to \infty$, or equivalently $\max_{x \in O} |\epsilon_n \Delta_x W_\theta^{\epsilon_n}(x)| \to 0$ ($\max_{x \in O} |\epsilon_n \Delta_x W_\theta^{\epsilon_n}(x)|$ also denoted as $\|\epsilon_n \Delta_x W_\theta^{\epsilon_n}\|_\infty$). Each ϵ_n requires multiple epochs to get the best θ_{ϵ_n} so that $W_{\theta_{\epsilon_n}}^{\epsilon_n} \approx W^{\epsilon_n}$. An update of ϵ can happen either at regular frequency, *i.e.* after each N_u epochs, or it can be triggererd by a special condition at the end of some epochs. Thus, we distinguish two timelines, where one is indexed by t and another one, slower, by n.

We propose three ways to define $\{\epsilon_n\}_{n=0}^\infty$. The first one is the naive *non-adaptive scheduler*:

$$\epsilon_{n+1} = \epsilon_n \rho \tag{14}$$

where $\rho \in (0, 1)$ is the rate of ϵ_n decay. We keep the same ϵ value for N_u epochs, thus $n = \left\lfloor \frac{t}{N_u} \right\rfloor$.

The second scheduler is called the *adaptive scheduler*. Let $\delta_n = \delta(\epsilon_n) = \max_{x \in \mathcal{D}_O} \left|\epsilon_n \Delta_x W_{\theta_{\epsilon_n}}^{\epsilon_n}(x)\right| = \left\|\epsilon_n \Delta_x W_{\theta_{\epsilon_n}}^{\epsilon_n}\right\|_\infty^{\mathcal{D}_O}$. Given ϵ_0, we get all the consecutive elements with the update rule:

$$\epsilon_{n+1} = \begin{cases} \frac{\rho\delta_{n-1}}{\delta_n}\epsilon_n & \text{if } \rho\delta_{n-1} \le \delta_n \\ \epsilon_n & \text{otherwise,} \end{cases} \tag{15}$$

where ρ has the same purpose as before. To compute the best θ_{ϵ_n} for an estimation of W^{ϵ_n}, we only update ϵ if $L(\theta, \mathcal{S}_O, \mathcal{S}_{\partial O})$ does not improve for n_ϵ consecutive epochs, *i.e.* $\mathcal{L}(\theta_i) \ge \mathcal{L}(\theta_{i-1})\ \forall i : t - n_\epsilon + 1 \le i \le t$. Thus, n_ϵ specifies the minimum number of epochs with fixed ϵ to obtain $W_\theta^\epsilon(x) \approx W^\epsilon(x)$, a solution of equation 9.

The intuition behind the adaptive scheduler is to use actual values of δ_n to regulate the speed with which ϵ_n goes to zero so that δ_n decreases making $\|\epsilon_n \Delta_x W^{\epsilon_n}\|_\infty$ also decrease. Indeed, assume that we have decreasing sequences $\epsilon_0, \epsilon_1, \ldots, \epsilon_{n-1}$ and $\delta_0, \delta_1, \ldots, \delta_{n-1}$ and we want to find a new ϵ_n such that $\delta_n < \delta_{n-1}$. If a new candidate value ϵ_n', *s.t.* $\epsilon_n' < \epsilon_{n-1}$, corresponds to

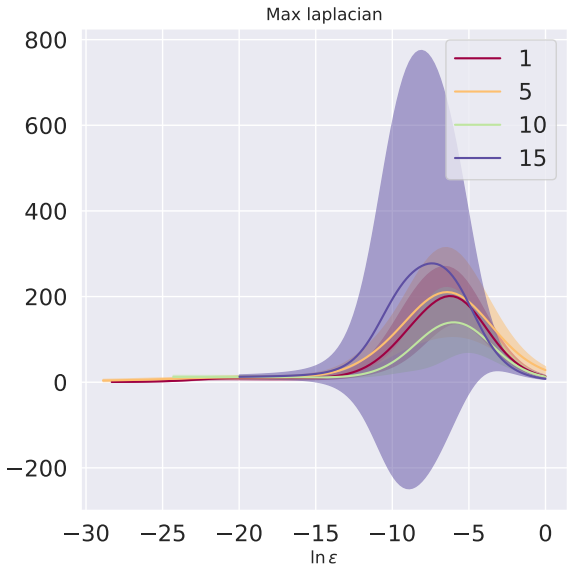


Figure 1: Evolution of $\|\Delta_x W_\theta^\epsilon\|_\infty$ with respect to $\ln \epsilon$ shown for different n_ϵ values, computed for the Pendulum environment (see Section 5).

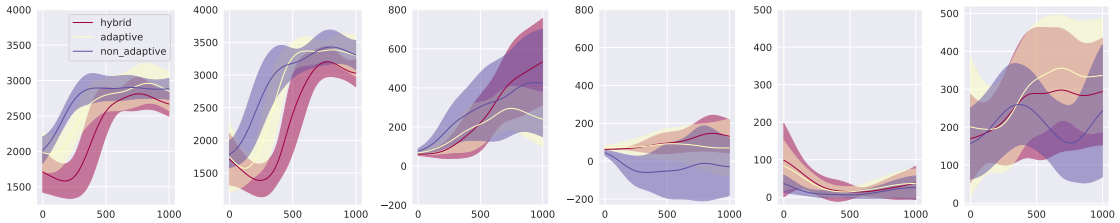
δ'_n and $\delta'_n \geq \delta_{n-1}$, then recalculate $\epsilon_n = \rho \frac{\delta_{n-1}}{\delta'_n} \epsilon'_n < \epsilon'_n$. Let us also assume that the Laplacian of W^ϵ does not change much with ϵ: $C\|\Delta_x W^{\epsilon'}_{\theta_{\epsilon'}}\|_\infty \geq \|\Delta_x W^{\epsilon}_{\theta_{\epsilon}}\|_\infty$ for all $\epsilon < \epsilon'$ and empirically it holds, especially for small ϵ (see Figure 1). In this case, $\delta_n = \delta(\epsilon_n) = \epsilon_n \left\|\Delta_x W^{\epsilon_n}_{\theta_{\epsilon_n}}\right\|_\infty = \rho \frac{\delta_{n-1}}{\delta'_n} \epsilon'_n \|\Delta W^{\epsilon_n}_{\theta_{\epsilon_n}}\|_\infty \leq \rho C \frac{\delta_{n-1}}{\delta'_n} \epsilon'_n \|\Delta W^{\epsilon'_n}_{\theta_{\epsilon'_n}}\|_\infty = \frac{\delta_{n-1}}{\delta'_n} \delta'_n = \rho C \delta_{n-1}$. If $\rho \leq \frac{1}{C}$ then $\delta_n < \delta_{n-1}$, thus it encourages the uniform convergence of equations.

We also propose a *hybrid scheduler*, mixing the two previous schedulers by starting with the non-adaptive scheduler for several ϵ-updates and then using the adaptive scheduler until the end of the training. The nonadaptive scheduler serves as a "warm-start" at the beginning of the training allowing us to do more regular updates for large ϵ for which training is easier (see Appendix E.3). Then, it is better to use the adaptive scheduler for smaller ϵ to ensure the convergence $\delta(\epsilon, \theta) \to 0$.

Putting everything together gives Algorithm 1.

Algorithm 1 ϵ-HJBPINNs

Set $\epsilon = \epsilon_0$, $\theta = \theta_0$, initialize W_θ^ϵ
for epoch t in $\{1, \ldots, \text{NB\_ITER}\}$ **do**
  Generate datasets $\mathcal{D}_O(N_D)$ and $\mathcal{D}_{\partial O}(N_D)$ of N_D states uniformly sampled from O and ∂O respectively
  for batches $\mathcal{S}_O \in \mathcal{U}(\mathcal{D}_O, N_F)$ and $\mathcal{S}_{\partial O} \in \mathcal{U}(\mathcal{D}_{\partial O}, N_F)$ **do**
    Update $\theta_t := \theta_t - \nu \nabla_\theta \mathcal{L}(\theta, \mathcal{S}_O, \mathcal{S}_{\partial O})$, where $\mathcal{L}(\theta, \mathcal{S}_O, \mathcal{S})$ is computed with equation 13
  end for
  Update ϵ using one of the ϵ-schedulers
end for



(a) (b) (c) (d) (e) (f)

Figure 2: Cumulative reward for different ϵ-schedulers along the training: (a) Pendulum, HJB, (b) Pendulum, soft HJB, (c) CartPole Swingup, HJB (d) CartPole Swingup, soft HJB (e) Acrobot, HJB (f) Acrobot, soft HJB.

5 Experimental Results

Environments We use continuous-time adaptations of inverted pendulum, cartpole and acrobot[1]. Those are challenging benchmarks for continuous time, similar to [24], where PINNs are also applied to solve the HJB equation.

Pendulum ($dt = 0.001$)[2] The state space consists of the angle ϕ and the angular speed $\dot{\phi}$. We consider $O = [-\pi, \pi] \times [-10, 10]$ (reducing the domain makes the comparison fairer with respect to exploration-based algorithms that do not compute value function on the whole domain) and $U = \{-2, 0, 2\}$.

CartPole Swingup ($dt = 0.005$) The state space consists of the pole angle ϕ, the pole angular speed $\dot{\phi}$, the cart coordinate y and its speed $\dot{y}$. We consider a difficult version: the problem of swinging up the pole with $O = [-\pi, \pi] \times [-10, 10] \times [-5, -5] \times [-5, 5]$ with $U = \{-3, 0, 3\}$.

Acrobot ($dt = 0.005$) The state space consists of angles ϕ_1 and ϕ_2 and their corresponding angular speeds $\dot{\phi}_1$ and $\dot{\phi}_2$. The control is the torque applied to the extreme tip. We consider $O = [-\pi, \pi] \times [-\pi, \pi] \times [-12.57, 12.57] \times [-28.27, 28.27]$ and $U = \{-5, 0, 5\}$.

We use cumulative rewards to compare the obtained policies, reported as mean cumulative rewards across several rollouts (typically 5 during the training) in the discrete-time environments. Each rollout is made of 5000 timesteps.

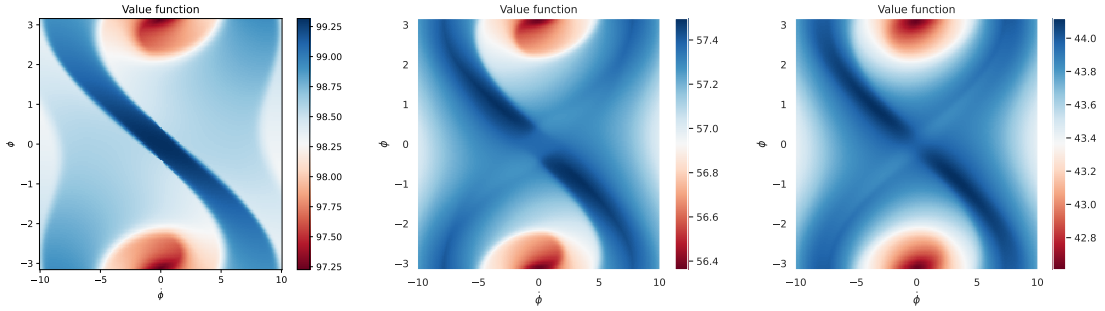
Analysis of ϵ-schedulers In this section we evaluate the performance of Algorithm 1[3] for different ϵ-schedulers.

Each training was executed with 1000 epochs (NB_ITER = 1000) and repeated for 8 different seeds. We have tested several neural architectures and obtained the best performance with a Fourier-Feature Network (FFN) [41], consisting of 3 layers, where the first layer is of size $d \times 40$ (as recommended in the original article, where d is the dimensionality of state space) and other layers contain 100 neurons each. The best performing activation function is `tanh`. When working with PINNs, it is important to use smooth activation functions as using non-smooth activations like `relu` may cause the training to fail. Indeed, the PDE loss $\mathcal{L}_O$ requires computing second order derivatives, and even the third derivative during the backward propagation, but those derivatives do not exist for `relu`. We have also observed that the training is more stable if we *standardize* the output of the neural networks, *i.e.* $W^{\epsilon}(x_i, \theta)$ and $W^{\epsilon_{n-1}}(x_i, \theta_{\epsilon_{n-1}})$, across the

[1] We used the environments taken from `https://github.com/cagatayyildiz/oderl` and slightly modified them to define O explicitly in each case.

[2] In the gym inverted pendulum environment dt is set to 1/20.

[3] Placeholder for the repository link. The code will be available for the final version.



(a) (b) (c)

Figure 3: Value function of Pendulum with ϕ as Y axis and $\dot{\phi}$ as X axis: (a) Ground Truth (computed with dynamic programming [29]) (b) PINNs for HJB, equation 4 (c) PINNs for soft HJB, equation 7.

samples in the batch $\mathcal{S}_O$ just before computing the regularization loss (equation 12). Indeed, the scale of the value function computed with the neural network is constantly changing during the training, therefore standardization helps to enforce uniform convergence of solutions without restraining the neural network training too much.

We have experimented with different ϵ-schedulers from Section 4.2 and their hyperparameters for both equation 4 and equation 7. The comparison of schedulers with the best hyperparameters are shown in Figure 2 and more can be found in Appendix E.2. In case of Pendulum, all three schedulers can learn a good policy. We also observe that using the soft HJB equation is better than the original HJB equation to find a policy with higher rewards. Indeed, the former one yields better results when the best control only marginally outperforms other controls. It can be seen when comparing value maps in Figure 3, where the value at $(\phi, \dot{\phi}) = (0, 0)$ is higher for the soft HJB equation. Figure 3 also indicates that neural architectures for PINNs still struggle to approximate the non-smooth regions, thus further research in neural architectures for PINNs is required. The soft HJB equation leads to higher cumulative rewards for Acrobot as well, but it is the opposite for Cartpole Swingup. Depending on the problem, one ϵ-scheduler can perform better than the others, thus we cannot conclude that one scheduler is strictly better or worse than the other two.

However, training is very sensitive to hyperparameters, especially for the non-adaptive scheduler. Moreover, all schedulers can fail if λ_R is too small, showing the importance of the regularization loss equation 12, see Appendix E.2.

Comparison with DTRL Algorithms In this section, we compare the performance of Algorithm 1 on different classical RL control tasks with well-studied DTRL algorithms such as PPO and A2C implemented in `stable-baselines3` [31][4]. To compare PINNs and DTRL agents, we used the same total number of training samples. However, note that this setting is less advantageous for PINNs training as it requires to sample uniformly across the whole domain to guarantee the viscosity, while DTRL agents can learn more from the trajectories that bring the highest outcome. For each algorithm, we take the best trained agent and we report its evalution mean and standard deviation over 100 rollouts in Table 1.

[4]It contains reliable implementations of RL agents. However, the same algorithms from other packages may perform differently.

For all reported environments, DTRL algorithms struggle to learn a good policy for so small dt and even completely fail in case of Acrobot. PINNs is able to achieve nearly optimal behaviour on Pendulum (stabilizing the pole). It performs less well in the case of Acrobot and CartPole, but its learnt policies still manage to swing up the pole in both environments outperforming DTRL competitors.

environment	method	mean	std
Pendulum	A2C	961.27	955.83
	PPO	1048.02	755.56
	PINNs	4006.17	562.97
CartPole Swing-Up	A2C	79.16	10.35
	PPO	79.16	10.35
	PINNs	865.50	209.50
Acrobot	A2C	0.0	0.0
	PPO	0.0	8.21
	PINNs	524.80	133.98

Table 1: Mean and standard deviation of the cumulative reward for different methods.

Discussion on PINNs scalability Even if PINNs can be executed for high-dimension problems, their precision can be stronly degraded. This is because PINNs are difficult to train with existing methods, which limits the complexity of problems currently considered, in particular high-dimensional ones. The work of [38] analyses this effect and propose a stochastic gradient descent (SGD) algorithm taking into account the neural tangent kernel (NTK) eigenvalues. Others have addressed this issue through adaptive sampling of colocation points [8, 40], reformulation of the loss to get a single term [10], NTK adaptive eigenvalue selection [21]. The work of [36] focuses on PINNs for HJB. They prove that for the HJB equation (under certain assumptions), PINNs require a high order loss to converge to a stable solution. They propose to rely on an l_∞ norm computed through an adversarial scheme to ensure convergence stability. In [15], an extension of SGD to dimentionality sampling to solve the HJB equation is proposed. But these papers assume linear-quadratic-Gaussian (LQG) control guaranteeing a unique solution, and thus disregard the difficulty of having a training scheme converging towards viscosity solutions.

We also tried to apply the R3 method [8] to our algorithm (see Appendix E.4), but it did not give any noticeable improvement and it can even lead to worse results for some tasks. Therefore, since adaptive sampling is a promising way to increase scalability, a further study on how to use adaptive sampling for viscosity solutions is required.

6 Conclusion

In this article, we consider the problem of finding the viscosity solutions of the deterministic HJB equation with neural network solvers. We propose a general scheme, which relies on solving a series of different PDE equations depending on ϵ. This framework gives flexibility on how ϵ are updated. In our experiments, we have shown that our scheme is able to learn the value function with different ϵ-schedulers and thus to find control policies that significanly outperform DTRL approaches on classical control environments when dt is small. Further work is required to improve the scalability of our method. Integrating the adaptive sampling methods can help to improve sample efficiency and considering more sophisticated neural networks can help with

an approximation of non-smooth areas. One limitation of our work is that it assumes that the dynamics are continuous ($f \in C(O)$), which is an important assumption for proving uniqueness of a viscosity solution. Thus, the approach considered in this paper cannot be applied to the case of non-continuous dynamics in a straightforward way. As the latter case is very important in real life applications, it should be studied in the future work. Finally, an interesting research perspective is to add model learning and actor/critic paradigm into our algorithm to explore the unknown dynamics case and enable the training based on trajectories.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

[1] Dipak M Adhyaru, IN Kar, and Madan Gopal. Bounded robust control of nonlinear systems using neural network–based hjb solution. *Neural Computing and Applications*, 20:91–103, 2011.

[2] Bastian Alt, Matthias Schultheis, and Heinz Koeppl. POMDPs in continuous time and discrete spaces. *Advances in Neural Information Processing Systems*, 33:13151–13162, 2020.

[3] Piermarco Cannarsa, Fausto Gozzi, and Halil Mete Soner. A boundary-value problem for Hamilton-Jacobi equations in Hilbert spaces. *Applied Mathematics and Optimization*, 24(1):197–220, 1991.

[4] Tao Cheng, Frank L Lewis, and Murad Abu-Khalaf. Fixed-final-time-constrained optimal control of nonlinear systems using neural network HJB approach. *IEEE Transactions on Neural Networks*, 18(6):1725–1737, 2007.

[5] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Theses, Institut National Polytechnique de Grenoble - INPG, June 2002.

[6] Michael G Crandall and Pierre-Louis Lions. Viscosity solutions of hamilton-jacobi equations. *Transactions of the American mathematical society*, 277(1):1–42, 1983.

[7] Jérôme Darbon, Peter M Dower, and Tingwei Meng. Neural network architectures using min-plus algebra for solving certain high-dimensional optimal control problems and Hamilton-Jacobi PDEs. *Mathematics of Control, Signals, and Systems*, 35(1):1–44, 2023.

[8] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Mitigating propagation failures in physics-informed neural networks using retain-resample-release (r3) sampling. In *Proceedings of the 40th International Conference on Machine Learning*, pages 7264–7302. PMLR, 2023. ISSN: 2640-3498.

[9] Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[10] Vasiliy A. Es'kin, Danil V. Davydov, Ekaterina D. Egorova, Alexey O. Malkhanov, Mikhail A. Akhukov, and Mikhail E. Smorkalov. About optimal loss function for training physics-informed neural networks under respecting causality, 2023.

[11] Wendell H Fleming and Halil Mete Soner. *Controlled Markov processes and viscosity solutions*, volume 25. Springer Science & Business Media, 2006.

[12] Christian Grossmann, Hans-Görg Roos, and Martin Stynes. *Numerical treatment of partial differential equations*, volume 154. Springer, 2007.

[13] Igor Halperin. Distributional offline continuous-time reinforcement learning with neural physics-informed pdes (sciphy rl for doctr-l). *arXiv preprint arXiv:2104.01040*, 2021.

[14] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[15] Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, and Kenji Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks, 2023.

[16] Hitoshi Ishii. Uniqueness of unbounded viscosity solution of Hamilton-Jacobi equations. *Indiana University Mathematics Journal*, 33(5):721–748, 1984.

[17] Wei Kang and Lucas C. Wilcox. Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations, 2016.

[18] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[19] Jeongho Kim, Jaeuk Shin, and Insoon Yang. Hamilton-Jacobi deep Q-Learning for deterministic continuous-time systems with Lipschitz continuous controls. *The Journal of Machine Learning Research*, 22(1):9363–9396, 2021.

[20] Jeongho Kim and Insoon Yang. Hamilton-Jacobi-Bellman equations for maximum entropy optimal control. *arXiv preprint arXiv:2009.13097*, 2020.

[21] Gregory Kang Ruey Lau, Apivich Hemachandra, See-Kiong Ng, and Bryan Kian Hsiang Low. Pinnacle: Pinn adaptive collocation and experimental points selection. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023.

[22] Derong Liu, Ding Wang, Fei-Yue Wang, Hongliang Li, and Xiong Yang. Neural-network-based online hjb solution for optimal robust guaranteed cost control of continuous-time uncertain nonlinear systems. *IEEE transactions on cybernetics*, 44(12):2834–2847, 2014.

[23] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[24] Michael Lutter, Boris Belousov, Kim Listmann, Debora Clever, and Jan Peters. HJB optimal feedback control with deep differential value functions and action constraints. In *Conference on Robot Learning*, pages 640–650. PMLR, 2020.

[25] Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Robust value iteration for continuous control tasks. *arXiv preprint arXiv:2105.12189*, 2021.

[26] Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Value iteration in continuous actions, states and time. *arXiv preprint arXiv:2105.04682*, 2021.

[27] Amartya Mukherjee and Jun Liu. Bridging physics-informed neural networks with reinforcement learning: Hamilton-Jacobi-Bellman Proximal Policy Optimization (HJBPPO). *arXiv preprint arXiv:2302.00237*, 2023.

[28] R. Munos, L.C. Baird, and A.W. Moore. Gradient descent approaches to neural-net-based solutions of the Hamilton-Jacobi-Bellman equation. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 3, pages 2152–2157 vol.3, 1999.

[29] Remi Munos. A Study of Reinforcement Learning in the Continuous Case by the Means of Viscosity Solutions. *Machine Learning*, 40:265–299, 2000.

[30] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.

[31] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[32] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[33] Halil Mete Soner. Optimal control with state-space constraint i. *SIAM Journal on Control and Optimization*, 24(3):552–561, 1986.

[34] Yuval Tassa and Tom Erez. Least squares solutions of the HJB equation with neural network value-function approximators. *IEEE transactions on neural networks*, 18(4):1031–1041, 2007.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[36] Chuwei Wang, Shanda Li, Di He, and Liwei Wang. Is l^2 physics informed loss always suitable for training physics informed neural network? *Advances in Neural Information Processing Systems*, 35:8278–8290, 2022.

[37] Haoran Wang, Thaleia Zariphopoulou, and Xun Yu Zhou. Reinforcement learning in continuous time and space: A stochastic control approach. *The Journal of Machine Learning Research*, 21(1):8145–8178, 2020.

[38] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[39] Harley E Wiltzer, David Meger, and Marc G Bellemare. Distributional Hamilton-Jacobi-Bellman equations for continuous-time reinforcement learning. In *International Conference on Machine Learning*, pages 23832–23856. PMLR, 2022.

[40] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023.

[41] Ge Yang, Anurag Ajay, and Pulkit Agrawal. Overcoming the spectral bias of neural value approximation. In *International Conference on Learning Representations*, 2022.

[42] Çağatay Yıldız, Markus Heinonen, and Harri Lähdesmäki. Continuous-time model-based reinforcement learning, 2021.

A Optimal Control Background

We use the same set-up as in Section 3.1. Further, we generalize the problem to the situations when exiting the domain O is possible and we state some known theoretical results from the literature.

A.1 General Formalism

In addition to the notions defined in Section 3.1, we also introduce some additional notations related to exiting the domain. Let τ denote the exit time. At time τ, we have $x(\tau) \in \bar{O}$. Thus, let us define the exit reward $R : \partial O \to \mathbb{R}$, which is obtained at the boundary points when control is pushing the system out of $\bar{O}$. Under these notations, we redefine the reinforcement functional:

$$J(x_0; u(t)) = \int_0^\tau \gamma^t r(x(t), u(t))dt + \gamma^\tau R(x(\tau)) \tag{16}$$

and the value function:

$$V(x) = \sup_{u(t)\in U} J(x; u(t)). \tag{17}$$

Note that, when $R(x) \to -\infty$, then $\tau \to \infty$, which brings us back to the problem considered in Section 3.1

A.2 Hamilton-Jacobi-Bellman Equation

Similar to Section 3.2, the similar result holds for the value function defined with equation 17:

Theorem A.1. *(Hamilton-Jacobi-Bellman). If the value function V is differentiable at x, then the Hamilton-Jacobi-Bellman (HJB) equation holds at any $x \in O$:*

$$V(x)\ln(\gamma) + \sup_{u\in U}\left\{\nabla_x V(x)^T f(x,u) + r(x,u)\right\} = 0. \tag{18}$$

When $O \subset \mathbb{R}^d$, V also satisfies the following boundary conditions:

$$V(x) \geq R(x) \quad \text{for } x \in \partial O \tag{19}$$

A.3 Uniqueness of Viscosity Solutions

In this section, we state more formally the uniqueness result that holds for viscosity solutions and the additional assumptions under which it is verified. This section presents the short summary of the main theoretical results from [11].

Assumption A.2.

(i) U is bounded,

(ii) f, r are bounded, f is continuous on $\mathbb{R}^d \times U$ and r is uniform continuous on $\mathbb{R}^d \times U$,

(iii) there exists L_f, such that $\|f(x,u) - f(y,u)\| \leq L_f\|x - y\|$ for any $x, y \in \mathbb{R}^d$.

Theorem A.3. *Given Assumption A.2, the value function V is uniformly continuous and bounded in $\mathbb{R}$ and then it is a unique viscosity solution of the HJB equation 4.*

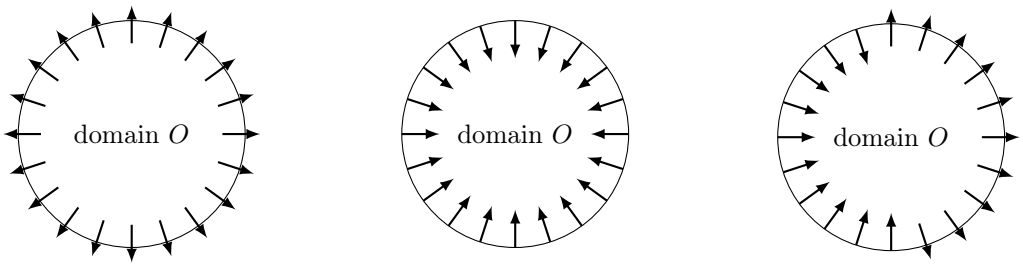


Figure 4: Boundary conditions. Case 1(left), Case 2 (middle) and Case 3(right).

This theorem is given for the case when $O = \mathbb{R}^d$ and therefore there is no boundary condition. The other cases are discussed in the next sections.

The proof of Theorem A.3 consists of several parts. First, one can prove that under Assumption A.2, the value function is indeed uniform continuous and bounded. Then, one can show that it is a viscosity solution due to the dynamic programming principle that holds in continuous time case as well. The uniqueness comes from the comparison principle. It states that under Assumptions A.2, if W and V are viscosity subsolution and supersolution respectively and are bounded and uniformly continuous functions, then $W \leq V$. The comparison principle implies that if such W and V are viscosity solutions (both subsolutions and supersolutions), then $W \leq V$ and $W \geq V$, therefore $W \equiv V$. Thus, V is a unique viscosity solution of the HJB equation in $\mathbb{R}^d$.

A.4 Viscosity for Different Boundary Conditions

When $O \not\equiv \mathbb{R}^d$, the additional assumptions on O are required.

Assumption A.4. For any $x \in \partial O$ and its normal vector $\eta(x)$

(i) $\exists u(x) \in U$ with $f(x, u(x))^T \eta(x) < 0$;

(ii) $\exists u(x) \in U$ with $f(x, u(x))^T \eta(x) > 0$.

Assumption A.5 (Regularity condition)**.** There exist $\epsilon_0, r > 0$ and $\hat{\eta}(x)$, a bounded and uniform continuous map of $\bar{O}$, satisfying

$$B(x + \epsilon\hat{\eta}(x), r\epsilon) \subset O, \quad \forall x \in O, \epsilon \in (0, \epsilon_0] \tag{20}$$

with $B(x, r) = \{y \in \mathbb{R}^d : \|x - y\| < r\}$.

In this section, we cover three different cases of boundary conditions that appear in control problems when $O \subset \mathbb{R}^d$, which are illustrated in Figure 4.

Case 1 If the system exits at any boundary point $x \in \partial O$ once the boundary is reached, *e.g.* it is the case when the exit reward $R(x)$ is sufficiently high to prefer to leave the area, *e.g.* when $R(x) \equiv R \geq r$ for any $x \in \partial O$ and $r \geq r(\bar{x}, u(\bar{x}))$ for any $\bar{x}, u(\bar{x})$. Then, the boundary condition is described with the following equation:

$$V(x) - R(x) = 0 \quad \forall x \in \partial O. \tag{21}$$

The uniqueness result holds due to the comparison principle that states that under Assumptions A.2 and provided that W and V are bounded and uniform continuous functions, if W

and V are viscosity subsolution and supersolution respectively, then $\sup_{x\in\bar{O}}(W(x) - V(x)) \leq \sup_{x\in\partial O}(W(x)-V(x))$. The existence of such value function is assured with Assumption A.4-(ii).

Case 2 If the system never exits the control domain. Let us denote the external normal vector at point $x \in \partial O$ as $\eta(x)$, then this boundary can be expressed as $f(x, u^*(x))^T\eta(x) \leq 0$ for any $x \in \partial O$. The optimal policy is not known a priori and thus it is hard to verify this constraint. In [11, 33], it was shown that it can be reformulated as:

$$-H(x, W, \nabla_x W + \alpha\eta(x)) \leq 0 \quad \forall\alpha \leq 0, x \in \partial O. \tag{22}$$

This allows to extend Definition 3.2.

Definition A.6 (Constrained viscosity solution)**.** $W \in C(\bar{O})$ is called a constrained viscosity solution of the HJB equation equation 18 if it is a viscosity subsolution in O and a viscosity supersolution in $\bar{O}$, *i.e.* if $\forall\psi \in C^1(\bar{O})$ and $\forall x \in \bar{O} \cup \arg\min\{(W - \psi)(x) : x \in \bar{O}\}$ with $W(x) = \psi(x)$, we have:

$$H(x, \psi(x), \nabla_x\psi(x)) \geq 0.$$

It is also possible to prove that there exists a continuous value function provided that Assumption A.4(i) holds and the set of admissible actions is not empty for any state of the system. Under the additional Assumption A.5, there exists a unique constrained viscosity solution (see [33]).

Case 3 If there exists a subset of points of the boundary at which the system exits the control domain. This is the same boudary condition considered in [29]. This boundary is formulated as follows:

$$R(x) - V(x) \leq 0 \quad \forall x \in \partial O. \tag{23}$$

However, this boundary is not sufficient to have uniqueness, therefore we redefine viscosity for this inequality constraint equation 23.

Definition A.7 (Viscosity solution with the boundary condition equation 23)**.**

- $W \in C(\bar{O})$ is a viscosity subsolution of the HJB equation in O with the boundary condition equation 23 if it is a viscosity subsolution in O and $\forall\psi \in C^1(\bar{O})$ and $\forall x \in \partial O$ local maximum of $W - \psi$ such that $W(x) = \psi(x)$, we have:

$$\min\{H(x, \psi(x), \nabla_x\psi(x)), R(x) - W(x)\} \leq 0$$

- $W \in C(\bar{O})$ is a viscosity supersolution of the HJB equation in O with the boundary condition equation 23 if it is a viscosity supersolution and $\forall\psi \in C^1(O)$ and $\forall x \in \partial O$ local minimum of $W - \psi$ such that $W(x) = \psi(x)$, we have:

$$\max\{H(x, \psi(x), \nabla_x\psi(x)), R(x) - W(x)\} \geq 0$$

- If $W \in C(\bar{O})$ is a viscosity subsolution and a supersolution with the boundary condition equation 23 then it is a viscosity solution with the boundary condition equation 23.

It is easy to check that when equation 23 is verified then a viscosity subsolution $W(x)$ in O is a viscosity subsolution with the boundary condition equation 23. However, when $W(x) > R(x)$ for some point $x \in \partial O$ then definition A.7 imposes an additional constraint that $W(x)$ should be a viscosity supersolution at such boundary points. Then similarly to Case 2, boundary condition equation 8 should be also satisfied, which can be interpreted as the system not being able to exit at those points. Similarly to Case 2, there is a uniqueness result:

Theorem A.8. *Let us assume that Assumptions A.2-A.5 hold, then the value function V is in $C(\bar{O})$ and it is the unique viscosity solution of the HJB equation in O with the boundary condition equation 23.*

The proof of this theorem can be found in [11, 3].

We choose to distinguish 3 different cases as it creates 3 different ways of approaching boundary conditions in practice. Indeed, equation 21, equation 22 and equation 23 produce different boundary losses for PINNs, *i.e.* $\mathcal{L}_{\partial O}$. However, note that Case 1 and Case 2 are subcases of Case 3.

Finally, some of the assumptions can be relaxed and it is possible to obtain more general uniqueness results (see [11, 3, 16]). However, the assumptions mentioned earlier are verified for the large class of control problems that appear in practice, like classical control or MuJoCo problems with no contacts [35]. Dealing with more general dynamics should be tackled in the future works.

B Intuition for Viscosity Solutions

In this section, we aim at providing the intuition behind the viscosity solutions. For that, we draw some parallels between DTRL and CTRL.[5]

Let us consider the DTRL formulation of the problem. We know that the optimal value function V in DTRL should satisfy the Bellman equation

$$V(x) = \max_u \{r(x,u) + \gamma \sum_{x'} p(x'|x,u)V(x')\}. \tag{24}$$

From that, we can introduce the Bellman operator as

$$T(\psi)(x) = \max_u \{r(x,u) + \gamma \sum_{x'} p(x'|x,u)\psi(x')\} \tag{25}$$

where ψ is an arbitrary function defined on the state space. This operator is known to be monotonic, *i.e.* for any functions ψ, ψ' we have

$$\psi \geq \psi' \Rightarrow T(\psi) \geq T(\psi'). \tag{26}$$

Moreover, from Eq. equation 24 follows that V should satisfy $V = T(V)$. Therefore, from Eq. equation 24-equation 26 we get the alternative definition for the solution of the Bellman equation 24.

Definition B.1. Let $V \in C(O)$, then V is the optimal value function if and only if

- $\forall \psi \in C^1(O)$ such that $\psi \geq V$

$$\max_u \{r(x,u) + \gamma \sum_{x'} p(x'|x,u)\psi(x')\} \geq V(x), \forall x \in O,$$

- $\forall \psi \in C^1(O)$ such that $\psi \leq V$

$$\max_u \{r(x,u) + \gamma \sum_{x'} p(x'|x,u)\psi(x')\} \leq V(x), \forall x \in O.$$

[5] This section is based on `https://benjaminmoll.com/wp-content/uploads/2020/02/viscosity_for_dummies.pdf`.

This definition can be seen as the discrete-time version of the viscosity solution definition. Therefore, in the discrete-time case, satisfying the fixed point equation is equivalent to satisfying the "discrete-time" viscosity solution definition.

As mentionned in the paper, $V \in C(O)$ can be non differentiable at some points of O, thus it is impossible to verify whether HJB equation is satisfied everywhere. Therefore, the main idea behind viscosity solutions is to replace V by some smooth functions where V is non differentiable.

In the following, first, we suppose that V is differentiable everywhere and we show a connection between Hamilton-Jacobi-Bellman equation and Bellman equation. Then, for the case when V is non smooth, we replace V by a smooth function and we show that it is possible to derive the notions of viscosity super/subsolutions.

Let us discretize our continous-time problem with a time-step dt. For simplicity, we consider that for any $x \in O$ there exists an optimal control $u^* = \pi(x) \in U$ so that $V(x) = J(x; u^*)$. Therefore, we replace sup with max in the definition of the value function, though it is possible to show that the next results also hold in case of sup. From the definition of the value function, we get

$$\begin{aligned}
V(x(t)) &= \max_u \left\{ \int_t^{t+dt} \gamma^{(s-t)} r(x(s), u(s)) ds + \gamma^{dt} V(x(t+dt)) \right\} \\
&= \max_u \left\{ \int_t^{t+dt} \gamma^{(s-t)} r(x(s), u(s)) ds + e^{dt \ln(\gamma)} V(x(t+dt)) \right\} \\
&\approx \max_u \left\{ r(x(t), u(t)) dt + e^{dt \ln(\gamma)} V(x(t+dt)) \right\} \\
&\approx \max_u \left\{ r(x(t), u(t)) dt + (1 + \ln(\gamma) dt) V(x(t+dt)) \right\}
\end{aligned}$$

So we derive this discrete-time dynamic programming problem:

$$V(x_t) = \max_u \left\{ r(x_t, u) dt + (1 + \ln(\gamma) dt) V(x_{t+dt}) \right\}, \tag{27}$$

where $x_{t+dt} = f(x_t, u) dt + x_t$.

Let us suppose that V is differentiable for all $x \in O$ and that $dt \in \left(0, -\frac{1}{\ln(\gamma)}\right)$. By subtracting $(1 + \ln(\gamma) dt) V(x_t)$ from both sides of Eq. equation 27 and then dividing by dt, we obtain

$$-\ln(\gamma) V(x_t) = \max_u \left\{ r(x_t, u) + \left(\frac{1}{dt} + \ln(\gamma) \right) \left(V(x_{t+dt}) - V(x_t) \right) \right\}.$$

If dt goes toward 0, we have

$$\ln(\gamma) V(x_t) = -\max_u \left\{ r(x_t, u) + \nabla_x V(x_t)^T f(x_t, u) \right\}.$$

This is exactly the Hamilton-Jacobi-Bellman equation, the continuous time equivalent of the Bellman equation.

Now, let us assume that V is non differentiable. As mentionned before, V should be replaced by a smooth function at the points where $\nabla_x V$ does not exist. Let ψ be an arbitrary smooth function on O such that $V - \psi$ has a local maximum at x_t and $V(x_t) = \psi(x_t)$. Therefore, $V \leq \psi$ in a neighborhood of x_t. If $1 + \ln(\gamma) dt > 0$, then

$$\begin{aligned}
V(x_t) &= \max_u \left\{ r(x_t, u) dt + (1 + \ln(\gamma) dt) V(x_{t+dt}) \right\} \\
&\leq \max_u \left\{ r(x_t, u) dt + (1 + \ln(\gamma) dt) \psi(x_{t+dt}) \right\}
\end{aligned}$$

Let us subtract $(1+\ln(\gamma)dt)\psi(x_t)$ from both sides and use $\psi(x_t)=V(x_t)$, as a result we have

$$-\ln(\gamma)V(x_t)dt \leq \max_u \{r(x_t,u)dt + (1+\ln(\gamma)dt)(\psi(x_{t+dt})-\psi(x_t))\}$$

Then, let us divide by dt and let dt goes toward 0, we have

$$\begin{aligned} & -\ln(\gamma)V(x_t) \leq \max_u \left\{r(x_t,u)+\nabla_x\psi(x_t)^T f(x_t,u)\right\} \\ \Leftrightarrow \quad & \ln(\gamma)V(x_t) - \max_u \left\{r(x_t,u)+\nabla_x\psi(x_t)^T f(x_t,u)\right\} \leq 0 \\ \Leftrightarrow \quad & H(x_t,\psi(x_t),\nabla_x\psi(x_t)) \leq 0. \end{aligned}$$

This gives us the definition of a viscosity subsolution.

It is possible to obtain the definition of a viscosity supersolution in a like manner, by performing the same derivations for an arbitrary $\psi \in C^1(O)$ such that $V-\psi$ has a local minimum in x_t and $V(x_t)=\psi(x_t)$. In both cases, we use the monotonicity of $\max_u \{r(x_t,u)dt + (1+\ln(\gamma)dt)\psi(x_{t+dt})\}$ in the function ψ, which is a counterpart of the Bellman operator in Definition B.1.

Thus, we recover the definition of a viscosity solution. The intuition is whenever a solution V of the HJB equation is non differentiable at some point $x \in O$, it should also satisfy other conditions imposed by viscosity for it to be a proper value function. In this way, the viscosity property serves as a regularizer to help to eliminate "bad" solutions of the HJB equation.

C Dynamic Programming

Further, we consider only FEM based dynamic programming proposed in [29]. In the FEM case, we use a triangulation Σ^δ to cover the state space. It is also possible to discretize the control space, denoted by U^δ. The vertices of the triangulation Σ^δ are denoted $\{\xi_1,\xi_2,...,\xi_{N_\delta}\}$ with $N_\delta \in \mathbb{N}$. In this setting, V is approximated by a piecewise linear function V^δ. Thus, for $x \in Simplex(\xi_0,..,\xi_d)$, we have

$$V^\delta(x) = \sum_{i=0}^{d} \lambda_{\xi_i}(x) V^\delta(\xi_i)$$

where $\lambda_{\xi_i}(x)$ is the barycentric coordinates inside the simplex $(\xi_0,...,\xi_d)$.

By using a FEM approximation scheme, the HJB equation is transformed into:

$$V^\delta(\xi) = \sup_{u\in U^\delta} [\gamma^{\tau(\xi,u)} V^\delta(\eta(\xi,u)) + \tau(\xi,u) r(\xi,u)]$$

where $\eta(\xi,u)=\xi+\tau(\xi,u)f(\xi,u)$ and $\tau(\xi,u)$ is a time discretization function that should satisfy:

$$\exists k_1,k_2>0, \forall \xi \in \Sigma^\delta, \forall u \in U^\delta, k_1\delta \leq \tau(\xi,u) \leq k_2\delta$$

If F^δ is defined as $F^\delta[\phi](\xi) = \sup_{u\in U^\delta}[\gamma^{\tau(\xi,u)} \sum_{i=0}^{d} \lambda_{\xi_i}(\eta(\xi,u))\phi(\xi_i) + \tau(\xi,u)r(\xi,u)]$, it is possible to show that F^δ satisfies a contraction property, and since $V^\delta(\xi) = F^\delta[V^\delta](\xi)$ holds, dynamic programming techniques can be applied to compute V^δ. Moreover, it can be proved that $V^\delta \underset{\delta\to 0}{\longrightarrow} V$ uniformly on any compact of the state space. With this method, one can derive algorithms that converge towards V, without even knowing the dynamics of the system. Thus, this is one of the approaches that allows us to find a viscosity solution of the HJB equation (see [29] for more details).

D Introduction to PINNs

Here, we provide a short introduction to PINNs for those readers who are not familiar with this method. The adaptation of PINNs to solving the HJB equation in the viscosity sense is covered in Section 4.2.

To solve a differential equation

$$F(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) = 0, \quad W : \bar{O} \to \mathbb{R}, x \in O, \tag{28}$$

with K_1 equality boundary conditions

$$B_i(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) = 0, \quad x \in \partial O, i \leq K_1, \tag{29}$$

and K_2 inequality boundary conditions

$$G_i(x, W(x), \nabla_x W(x), \nabla_x^2 W(x)) \leq 0, \quad x \in \partial O, i \leq K_2. \tag{30}$$

one can assume that $W(x)$ lies in the class of functions $\mathcal{F}_\theta = \{f_\theta(x) = NN(x, \theta) : \theta \in \Theta\}$ represented by neural networks of a fixed architecture and parametrized with weights $\theta \in \Theta$. If it is the case then there exists θ such that $W_\theta[x]$ should satisfy equation 28 and thus minimize the loss

$$\mathcal{L}_{PDE}(\theta) = \frac{1}{N_F} \sum_{i=1}^{N_F} \left(F(x_i, W_\theta[x_i], \nabla_x W_\theta[x_i], \nabla_x^2 W_\theta[x_i])\right)^2 \quad \forall x_i \in \mathcal{S}_u(O, N_F), \tag{31}$$

with $\mathcal{S}_u(O, N_F)$ denoting a sample of N_F points drawn uniformly from O. If the solution $W_\theta[x]$ should satisfy some additional boundary constraints then it should also minimize the boundary losses for all $k \leq K_1$ and $k' \leq K_2$

$$\mathcal{L}_{B_k}(\theta) = \frac{1}{N_B^k} \sum_{i=1}^{N_B^k} \left(B_k(x_i, W_\theta[x_i], \nabla_x W_\theta[x_i], \nabla_x^2 W_\theta[x_i])\right)^2 \quad \forall x_i \in \mathcal{S}_u(\partial O, N_B^k) \tag{32}$$

$$\mathcal{L}_{G_{k'}}(\theta) = \frac{1}{N_G^{k'}} \sum_{i=1}^{N_G^{k'}} \left(\left[G_{k'}(x_i, W_\theta[x_i], \nabla_x W_\theta[x_i], \nabla_x^2 W_\theta[x_i])\right]^+\right)^2 \quad \forall x_i \in \mathcal{S}_u(\partial O, N_G^{k'}), \tag{33}$$

where $[f(x)]^+ = \max\{f(x), 0\}$.

To put everything together, when solving a PDE in a PINNs-like manner, one should train a neural network $W_\theta[x]$ that minimizes:

$$\mathcal{L}(\theta) = \mathcal{L}_{PDE}(\theta) + \sum_{k=1}^{K_1} \lambda_k \mathcal{L}_{B_k}(\theta) + \sum_{k=1}^{K_2} \lambda'_k \mathcal{L}_{G_k}(\theta). \tag{34}$$

where $\lambda_k, \lambda'_k > 0$ are some mixing coefficients for different boundary conditions.

E Experimental Results. Supplementary

E.1 Dynamic Programming Experimental Results

In this section, we present the results obtained with one of the algorithms proposed in [29]. First, a grid is built by dividing each axis by N points. Then, we use the Delaunay's triangulation over the grid and apply the Value Iteration algorithm (VI) to the FEM-MDP derived in [29].

We set $\delta = \frac{1}{N}$, δ being the discretization step. The stopping criterion used at the step n is $\|V_n - V_{n-1}\|_\infty \leq \epsilon$ where ϵ is a chosen tolerance. In our experiments we work with $\epsilon = 10^{-5}$.

When δ goes towards 0, our approximated value function, V^δ, converges towards the true value function. In our case, $\delta \to 0$ is equivalent to $N \to +\infty$. Empirically, we can see in Figure 5 that this property is satisfied. Indeed, as we increase N, we obtain a more accurate V^δ, and as a result, a better control that leads to a higher cumulative reward.

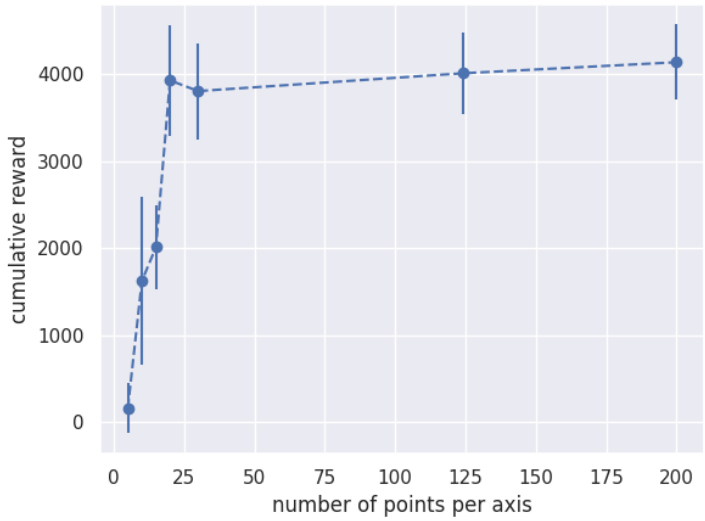


Figure 5: The cumulative reward obtained on the inverted pendulum for different grid sizes N.

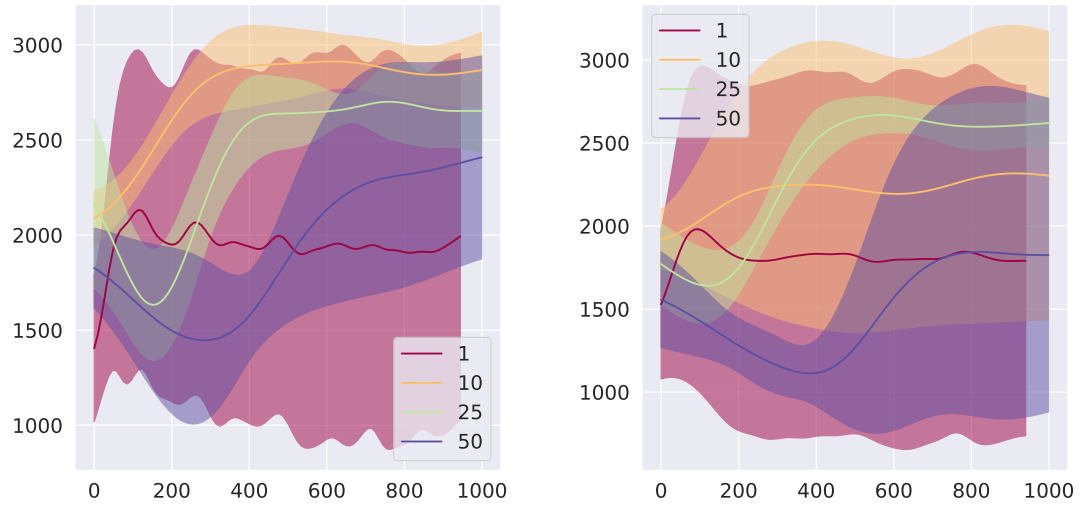
E.2 Comparison of ϵ schedulers

As mentioned in Section 4.2, we tested three kind of ϵ-schedulers. All scheduler experiments have been performed with the parameters described in Section E.5, except mentioned so.

Figure 6 depicts the performance of non-adaptive scheduler on Pendulum. The non-adaptive scheduler fails if ϵ is updated too fast and the NN is not able to adjust to a new ϵ (see Figure 8a), but also it may fail if ϵ is updated too slow, as the NN starts to overfit to the given ϵ. The adaptive scheduler shown in Figure 7 demonstrates a more robust performance with respect to the choice of n_ϵ. Both schedulers fail more often when the contribution of the regularization loss, equation 12, is too low (*e.g.* Figure 8b shows that the adaptive scheduler can produce the wrong value function).

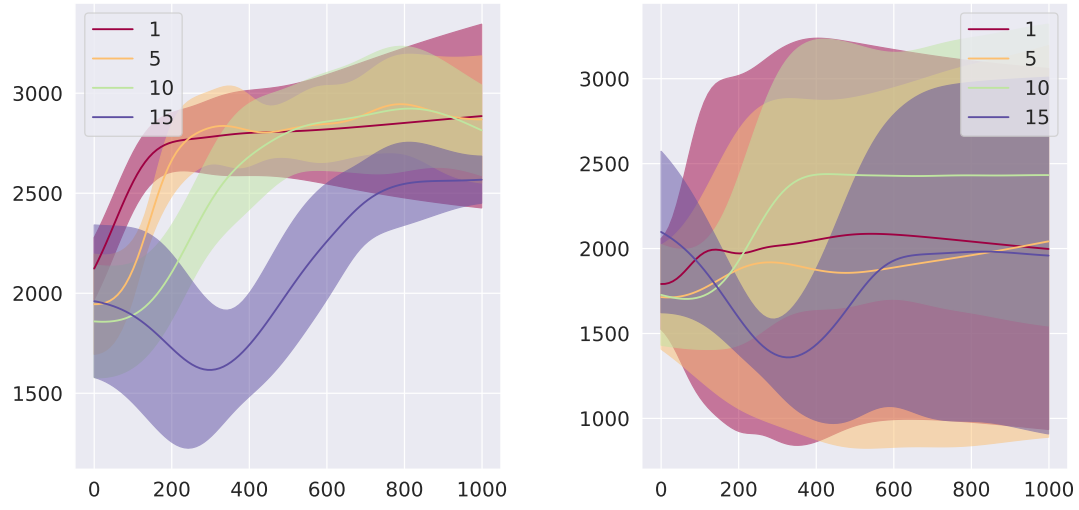
E.3 Convergence of PINNs for a fixed ϵ

In this section, we provide the results that we have obtained for different fixed ϵ. We have used the same parameters as in table 2. In Figure 9, it is clear that it is easier for PINNs to approach W^ϵ when ϵ is high enough. Therefore, we assume that starting our ϵ scheduler from $\epsilon_0 = 1$ leads to a more stable convergence.



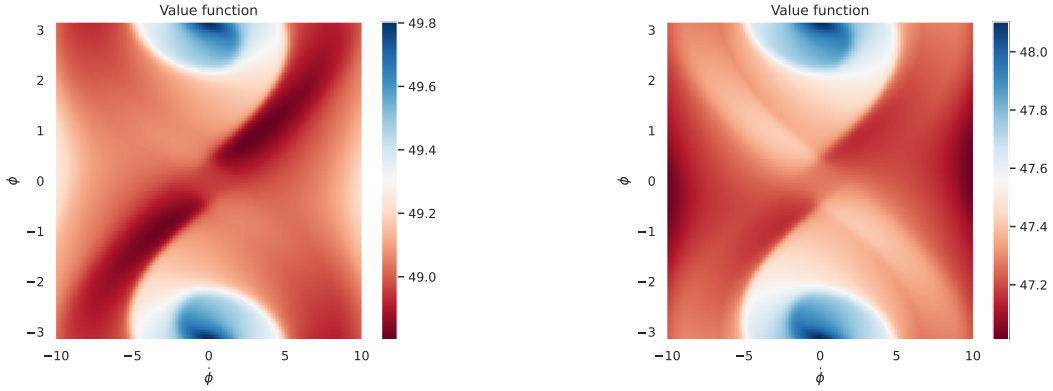
(a) (b)

Figure 6: Cumulative rewards on Pendulum with non-adaptive scheduler shown for different N_u (a) with strong regularization $\lambda_R = 10^{-3}$ (b) with weak regularization $\lambda_R = 10^{-6}$.



(a) (b)

Figure 7: Cumulative rewards on Pendulum with adaptive scheduler shown for different n_ϵ (a) with strong regularization $\lambda_R = 10^{-3}$ (b) with weak regularization $\lambda_R = 10^{-6}$.



(a) (b)

Figure 8: Value maps on Pendulum with the failed schedulers: (a) a non-adaptive scheduler with $N_u = 1, \lambda_R = 10^{-3}$ (b) an adaptive scheduler $n_\epsilon = 1, \lambda_R = 10^{-6}$.

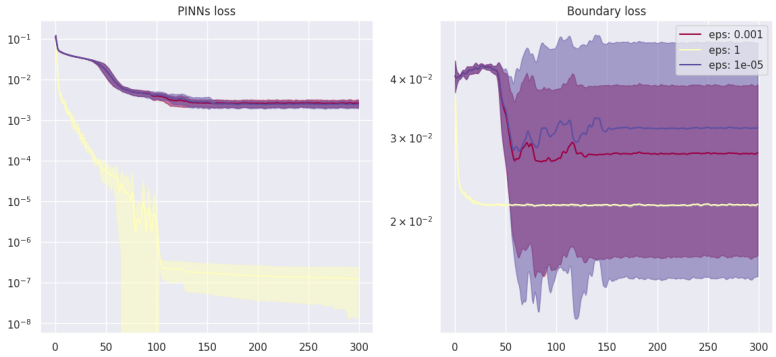
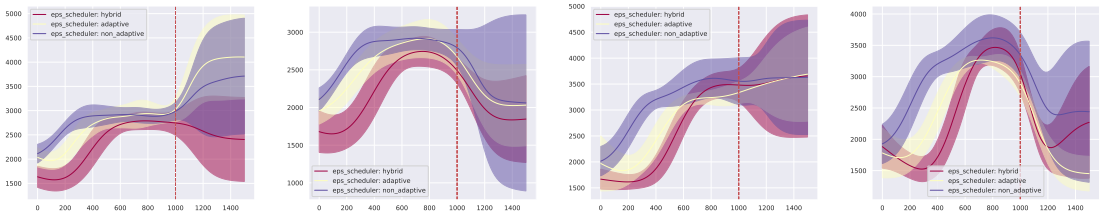


Figure 9: W^ϵ pinns loss and boundary loss for $\epsilon = 1, 10^{-3}, 10^{-5}$.

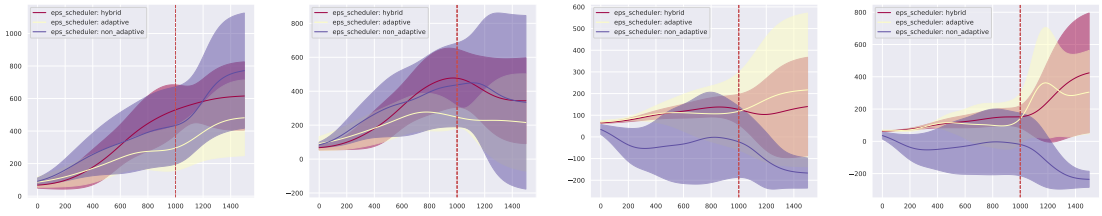
E.4 Adaptive Sampling

This section shows our experiments with adaptive sampling techniques. We considered the R3 method from [8]. First, we have observed that using it together with ϵ-schedulers can be detrimental to the uniform convergence. Therefore, we use this method only at the end of the training. After NB_ITER epochs, we execute R3 for the additional 400 epochs with the fixed ϵ. It can be seen as the way to finetune the final result. Figures 10-12 demonstrate the effect of such procedure on the classical control environments described in Section 5. The vertical dash line shows the start of the final stage with the adaptive sampling. The results are shown for different dataset sizes used for adaptive sampling. One can observe that adaptive sampling helps to improve the results for some tasks and some schedulers, *e.g.* it provides a noticeable improvement for Acrobot with the adaptive and hybrid schedulers. Nevertheless, it can also lead to worse results, especially when the big datasets are used. The degraded performance can be explained with the fact that the adaptive sampling of R3 can *break* viscosity by overfitting the model on difficult non-smooth areas. Thus, futher research on how to combine adaptive sampling with viscosity is necessary.



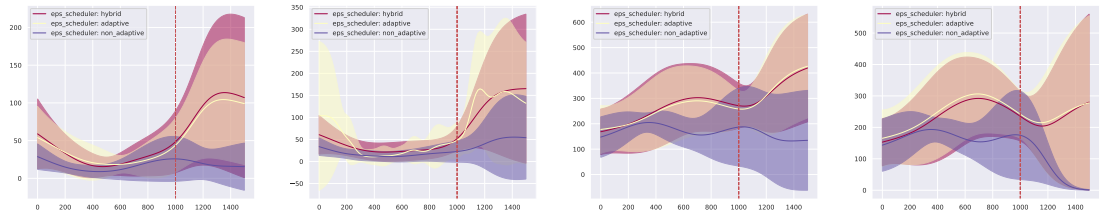
(a) (b) (c) (d)

Figure 10: Cumulative reward for different adaptive sampling setting (pendulum environment): (a) HJB, dataset of size 1000, (b) HJB, dataset of size 10000, (c) Soft HJB, dataset of size 1000, (d) Soft HJB, dataset of size 10000.



(a) (b) (c) (d)

Figure 11: Cumulative reward for different adaptive sampling setting (cartpole swingup environment): (a) HJB, dataset of size 1000, (b) HJB, dataset of size 10000, (c) Soft HJB, dataset of size 1000, (d) Soft HJB, dataset of size 10000.



(a) (b) (c) (d)

Figure 12: Cumulative reward for different adaptive sampling setting (acrobot environment): (a) HJB, dataset of size 1000, (b) HJB, dataset of size 10000, (c) Soft HJB, dataset of size 1000, (d) Soft HJB, dataset of size 10000.

Environment	Names	Hyperparameters	values
Shared	batch size	N_F	64
	learning rate	ν	0.00085
	patience adaptive scheduler	n_ϵ	10
	boundary loss coefficient	λ	10^{-1}
	starting ϵ	ϵ_0	1
	number of epochs between ϵ updates	N_u	10
	non-adaptive scheduler coefficient	ρ	0.5
	adaptive scheduler coefficient	ρ'	0.99
	number of ϵ updates with non-adaptive scheduler	N_ϵ	5
	reg loss coefficient	λ_R	10^{-3}
Pendulum	number of sampled points	N_D	5000
Cartpole	number of sampled points	N_D	5000
Acrobot	number of sampled points	N_D	5000

Table 2: Hyperparameters for Algorithm 1.

E.5 Best Hyperparameters

The best performing hyperparameters are gathered in Table 2.

RESEARCH CENTRE
LILLE – NORD EUROPE

Parc scientifique de la Haute-Borne
40 avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399