Bellman Neural Networks for the Class of Optimal Control Problems with Integral Quadratic Cost

Enrico Schiassi, Andrea D'Ambrosio, and Roberto Furfaro.

***Abstract*—This work introduces Bellman Neural Networks (BeNNs) and employs them to learn the optimal control actions for the class of optimal control problems (OCPs) with integral quadratic cost. BeNNs represent a particular family of Physics-Informed Neural Networks (PINNs) specifically designed and trained to tackle OCPs via applying the Bellman Principle of Optimality (BPO). The BPO provides necessary and sufficient optimality conditions, which result in a nonlinear partial differential equation known as the Hamilton-Jacobi-Bellman (HJB) equation. BeNNs learn the optimal control actions from the unknown solution of the arising HJB equation (i.e., the value function), where the unknown solution is modeled using a Neural Network. Additionally, the paper shows how to estimate the upper bounds on the generalization error of BeNNs while learning the solutions for the OCP class under consideration. The generalization error estimate is provided in terms of the choice and number of the training points as well as the training error. Numerical studies show that BeNNs can be successfully applied to learn the feedback control actions for the class of optimal control problems considered and, after the training is completed, deployed to control the system in a closed-loop fashion.**

***Impact Statement*—The proposed research improves our understanding of how to solve optimal control problems with closed-loop solutions and has potentially a countless number of applications in several different areas. The study is at the intersection between optimal control theory and artificial intelligence connected with mathematical tools for functional interpolation. This advances the ability to implement a higher level of autonomy in decision-making for practical applications with a beneficial impact on our society.**

***Index Terms*—Bellman Neural Networks, Close-loop Control, Extreme Learning Machines, Functional Interpolation, Hamilton-Jacobi-Bellman Equation, Physics-Informed Neural Networks**

I. Introduction

During the last century, optimal control theory has garnered interest in many fields of study. In particular, Optimal Control Problems (OCPs) play a crucial role in several scientific areas, such as trajectory optimization and tracking [1, 2, 3]. In many practical OCPs, it is not trivial and most likely impossible to find an analytical solution. Hence, one has to rely on numerical methods. The solutions of OCPs can be open-loop or closed-loop. One way to derive closed-loop solutions is to exploit the *Bellman's Principle of Optimality (BPO)*, ensuring the optimality sufficient and necessary conditions, which returns the Hamilton–Jacobi–Bellman (HJB) partial differential equation [4]. That is, by solving the HJB equation, feedback optimal control actions can be synthesized and then directly deployed to optimally control the system. In many practical applications, the HJB PDE must be solved numerically. The HJB equation is affected by the *curse of dimensionality*, as the computational cost increases exponentially with the system dimension number. Hence, traditional numerical methods to solve PDE, such as FEM, becomes impracticable [5, 6, 7]. Many methods, relying on the so-called Adaptive Dynamic Programming (ADP), have been proposed to solve HJB PDEs for OCPs [8, 9, 10, 11, 12]. ADP can be directly employed to solve the HJB equation using an actor-critic structure. ADP is generally implemented via two most popular methods: 1) policy iteration and 2) value iteration. Examples include Vrabie and Lewis [13] who devised an adaptive critic design using a policy iteration algorithm, and Al-Tamimi et al. [14] who employed two NNs to model actor and critic and solve a discrete optimal control problem using value iteration. In contrast, the method we propose attempts to directly solve the HJB using Physics-Informed Neural Network (PINN [15]). It builds a loss function with HJB residual. Using PINNs allows learning closed-loop optimal controllers that directly satisfy the HJB PDE. We will specifically solve HJB PDEs with two different PINN frameworks. The first one is the classic PINNs, as introduced by Raissi et al. [15]. The second one is an improved and more robust Physics-Informed Neural Networks (PINNs) framework that combines Neural Networks (NNs) and a functional interpolation technique called *Theory of Functional Connections (TFC)* [16], forming a PINN TFC-based framework [17, 18]. For the problems considered in this work, PINNs are specifically designed and trained to learn optimal control actions by satisfying the BPO. For this reason, we name these PINNs *Bellman Neural Networks (BeNNs)*. More specifically, this paper focuses on the class of OCPs with integral quadratic cost, both for finite and infinite horizon problems. The main contributions and goals of this work are: 1) to show the feasibility in using PINN-based frameworks to directly learn the HJB PDE solutions and thus closed-loop optimal control, 2) the estimation of the generalization error generated via the proposed methodology, and 3) the comparison of three different PINN-based frameworks in learning the HJB PDE solutions, to select the most suitable choice to tackle this class of OCPs. PINNs represent a machine learning framework that exploits the physics as a regularization term in

This paragraph of the first footnote will contain the date on which you submitted your paper for review.

The corresponding author is Roberto Furfaro (e-mail: robertof@email.arizona.edu)

Enrico Schiassi is a PhD candidate at the Systems and Industrial Engineering Department, The University of Arizona, Tucson, AZ 85721 (e-mail: eschiassi@email.arizona.edu).

Andrea D'Ambrosio is a Postdoctoral Associate at the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 (e-mail: andreada@mit.edu).

Roberto Furfaro (corresponding author) is a professor at the Aerospace and Mechanical Engineering Department and Systems and Industrial Engineering Department, The University of Arizona, Tucson, AZ 85721 (e-mail: robertof@email.arizona.edu).

This paragraph will include the Associate Editor who handled your paper.

the loss function for training NNs [15]. PINNs are introduced to consider the DEs describing the physics behind the collected data. Indeed, DEs residuals are embedded within the loss function as additional terms, which penalize the training when the DEs and their constraints (e.g., the initial conditions or ICs and/or boundary conditions or BCs) are violated. In the classic PINN framework the DE constraints are part of the loss function along with the DE residuals within the domain. This represents the major drawback for the classic PINN framework [19]. The PINN TFC-based methods used in this paper substantially improve the classic PINN frameworks, removing their main limitation. The key ingredients of the PINN-TFC methods are the Constrained Expressions (CEs) introduced in the Theory of Functional Connections (TFC) [16]. The general expression of the CE is,

$$f(\mathbf{x}) \simeq f_{CE}(\mathbf{x}, g(\mathbf{x})) = A(\mathbf{x}) + B(\mathbf{x}, g(\mathbf{x}))$$

where $f(\mathbf{x})$ is the unknown function to be interpolated, $f_{CE}(\mathbf{x}, g(\mathbf{x}))$ is the CE, $A(\mathbf{x})$ is a functional that analytically satisfies any given linear constraint, and $B(\mathbf{x}, g(\mathbf{x}))$ projects the free-function $g(\mathbf{x})$, which is a real function that must exist on the constraints, onto the space of functions that vanish at the constraints [18]. TFC is a functional interpolation technique which can be applied to many mathematical problems. In particular, TFC is applied for approximating the solution of DEs [20], as well as for OCPs via the PMP approach. For solving DE, the original (or classic) TFC employs a linear combination of orthogonal polynomials as free-function [20]. Although employing orthogonal polynomials as free-function allows to compute highly accurate results, it affects the original TFC framework with the curse of dimensionality. In particular when solving large-scale PDEs, as the computational cost increases exponentially with the number of bases required to compute accurate solutions [20]. To overcome these issues, PINN TFC-based methods use NNs as free-function. PINN TFC-based methods have been already applied to solve a class of OCPs via indirect method [21, 22].
In this manuscript, first, we explain how BeNNs are modeled and trained to learn control actions for the class of finite and infinite horizon optimal control problems with integral quadratic cost. Then we will provide an estimate on the upper bounds of the generalization error of BeNNs in learning the solutions for the OCP class considered in this work in terms of the training error, number, and choice of the training points. Two benchmarks OCPs are then tackled to show the effectiveness of BeNNs in learning optimal control actions for the class of OCPs considered and to test the theoretical findings. Finally, the results will be discussed in the remaining section.

II. Bellman Neural Networks for Optimal Control Problems

BeNNs represent a particular family of PINNs specifically designed and trained to learn the solutions of OCPs via the application of the BPO. In this work, we will focus on the class of OCPs with integral quadratic cost. The authors here coin the term *"Bellman Neural Networks (BeNNs)"* to refer in a more compact form to PINNs designed and trained to learn optimal control via the application of the BPO. The NN representation of the HJB unknown solution is done via two different PINN frameworks: classic PINNs and PINNs combined with TFC. According to PINN TFC-based frameworks, a generic differential equation's unknown solutions are modeled via the TFC's CEs using NNs as free-functions. When Deep NNs (DNNs) are used as free-function, the PINN TFC-based framework is called *Deep-TFC* [18]. In the this framework, gradient-based method, such as ADAM optimizer [23] or the L-BFGS optimizer [24], are employed to optimize the free-function parameters. When shallow NNs, trained via Extreme Learning Machine (ELM) algorithm [25], are used as free-function, the PINN TFC-based framework is called *Extreme-TFC (X-TFC)* [17]. The ELM algorithm randomly samples input weights and bias. These are kept fixed during the training procedure. Thus, the output weights are the only trainable parameters [25]. Thanks to this feature, the training is performed with a robust and fast least-squares. When X-TFC is used to learn the solution of linear DEs, a single pass least-squares is employed to optimize the free-function parameters (e.g., the NN output weights). Conversely, iterative least-squares are required to optimize the free-function parameters when X-TFC is used to learn the solution of nonlinear DEs (such as the HJB PDEs). The details regarding the iterative least-squares procedure can be found in [17]. This section shows how to design and train BeNN to learn the optimal control actions for two classes of problems with integral quadratic cost: finite horizon problems and infinite horizon problems. Moreover, following Ref. [26], we derive an estimate on the upper bounds of the generalization error of BeNNs in learning the solutions for the OCP class considered in this work.

A. Finite Horizon Problems

The typical finite horizon OCP is posed as follows,

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(t,\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{\phi}(\boldsymbol{x}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u}) \tag{1}$$

subject to

$$\begin{aligned} &\dot{\boldsymbol{x}} = \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) \\ &\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\ &\boldsymbol{x}(t_f) \in \mathcal{C} \\ &t \in [t_0, t_f] \end{aligned} \tag{2}$$

where t, $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^n$, $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$, and $\mathcal{C} \subset \Omega$ are the time variable, states, control, and terminal conditions respectively. In general the final time t_f may be fixed or unknown (e.g. for the time-free problems). The system dynamics $\boldsymbol{f} \in \mathbb{R}^n$ is assumed to be known. The main characteristic of the finite horizon problems is the presence of the Meyer term in the cost function, $\boldsymbol{\phi}(\boldsymbol{x}(t_f))$, which results to be a temporal boundary condition for the arising HJB equation. In this manuscript we will consider only fixed time problems. We consider the value function $V(t,\boldsymbol{x})$ defined as follows,

$$V(t,\boldsymbol{x}) = \inf_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(t,\boldsymbol{x},\boldsymbol{u}) \tag{3}$$

which is the unique solution of the following HJB equation [27],

$$-V_t + \sup_{\boldsymbol{u}\in\mathcal{U}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u})\} = 0, \quad \forall t\in[t_0,t_f], \forall \boldsymbol{x}\in\Omega \tag{4}$$

subject to

$$V(t_f,\boldsymbol{x}) = \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x}\in\Omega \tag{5}$$

The first step is to derive the optimal control, $\boldsymbol{u}_{\text{opt}}$, as a closed analytical form with respect to the value function V. That is,

$$\boldsymbol{u}_{\text{opt}} = \sup_{\boldsymbol{u}\in\mathcal{U}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u})\} \tag{6}$$

Once $\boldsymbol{u}_{\text{opt}}$ is obtained, its closed analytical form is plugged back into (4). Then, (4) reduces to a nonlinear PDE that is solved for V,

$$\mathcal{D}(V(t,\boldsymbol{x})) = \mathcal{F}(t,\boldsymbol{x}), \quad \forall t\in[t_0,t_f], \forall \boldsymbol{x}\in\Omega \tag{7}$$

subject to

$$V(t_f,\boldsymbol{x}) = \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x}\in\Omega \tag{8}$$

where $\mathcal{D}(\cdot)$ is a nonlinear differential operator acting on V, and $\mathcal{F}(t,\boldsymbol{x})$ is the known forcing term. Our goal is to train a BeNN to learn the solution of problem (7)-(8). Two different training algorithms are proposed. One employs classic PINNs and the other one PINN TFC-based. The training algorithms are given below. The main challenge in solving finite horizon problems with PINN-based frameworks is that the nonlinear PDE (7) must be learned simultaneously with the temporal boundary condition (8). The goal of the Algorithm 1 is to train a BeNN to learn the solution of problem (7)-(8) employing classic PINNs. Therefore, V is modeled as,

$$V(t,\boldsymbol{x}) \simeq V(t,\boldsymbol{x};\theta) = V^\theta(t,\boldsymbol{x}) \tag{9}$$

where θ are the NN parameters that are learned during the training. To train the NN, the following residuals need to be defined. The interior (int) residual is,

$$\mathcal{R}_{\text{int},\theta}(t,\boldsymbol{x}) = \mathcal{D}(V^\theta(t,\boldsymbol{x})) - \mathcal{F}(t,\boldsymbol{x}), \quad \forall t\in[t_0,t_f], \forall \boldsymbol{x}\in\Omega \tag{10}$$

The temporal boundary (tb) residual is,

$$\mathcal{R}_{tb,\theta}(\boldsymbol{x}) = V^\theta(t_f,\boldsymbol{x}) - \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x}\in\Omega \tag{11}$$

The following physics-driven loss function, $J(\theta)$, is formed by collocating the residuals on the training points,

$$J(\theta) = \lambda\sum_{n=1}^{N_{\text{int}}}||\mathcal{R}_{\text{int},\theta}(t_n,\boldsymbol{x}_n)||^2 + \sum_{n=1}^{N_{\text{tb}}}||\mathcal{R}_{tb,\theta}(\boldsymbol{x}_n)||^2 \tag{12}$$

where, N_{int} are the interior training points (e.g., the points sampled from $[t_0,t_f]$ and Ω), N_{tb} are the temporal boundary training points, (e.g., the points sampled from Ω), and λ is a hyperparameter for balancing the interior residual over the temporal boundary residual.

The algorithm to train a BeNN to learn the solution of problem (7)-(8) by employing PINN TFC-based is Algorithm 2, where V is modeled with the following CE,

$$V(t,\boldsymbol{x}) \simeq V(t,\boldsymbol{x};\theta) = V^\theta(t,\boldsymbol{x}) = g^\theta(t,\boldsymbol{x}) + \Big(\phi(\boldsymbol{x}) - g^\theta(t_f,\boldsymbol{x})\Big) \tag{13}$$

Algorithm 1: PINN based algorithm for BeNN training

```
Input: Domain $[t_0,t_f]\times\Omega$, differential operator $\mathcal{D}$, forcing
       term $\mathcal{F}$, training points (for interior and temporal
       boundary), hyperparameter for balancing the two
       terms in the total loss function, and the optimization
       algorithm to minimize the loss with respect to the
       parameters $\theta$.
Goal: To find BeNN, $V^* = V^{\theta^*}$, to approximate the true
      solution $V$ of Eqs. (7)-(8) as best as possible via
      solving the following minimization problem
          $\theta^* = \min_{\theta\in\Theta} J(\theta)$
      where $\Theta$ is the search space of the NN parameters,
      whose formal definition is given in Ref. [26]
Step 1: Approximate the real, unknown, $V$ with a NN,
        which is called BeNN, $V^\theta$.
Step 2: For an initial guess of the vector $\theta^0\in\Theta$, evaluate
        $V^{\theta^0}$, the residuals, the loss, and its gradients to
        initialize the BeNN training (e.g., the optimization
        algorithm).
Step 3: Run the BeNN training until the optimal
        parameters, $\theta^*$, are computed. The map $V^* = V^{\theta^*}$
        is the desired BeNN for approximating the true
        solution $V$ of (7)-(8).
```

where $g^\theta(t,\boldsymbol{x}) = g(t,\boldsymbol{x};\theta)$ is a NN with parameters θ, that are learned during the training. To train the NN, the residuals need to be defined. The Interior (int) residual is,

$$\mathcal{R}_{\text{int},\theta}(t,\boldsymbol{x}) = \mathcal{D}(V^\theta(t,\boldsymbol{x})) - \mathcal{F}(t,\boldsymbol{x}), \quad \forall t\in[t_0,t_f], \forall \boldsymbol{x}\in\Omega \tag{14}$$

The temporal boundary (tb) residual is,

$$\mathcal{R}_{tb,\theta}(\boldsymbol{x}) = V^\theta(t_f,\boldsymbol{x}) - \phi(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x}\in\Omega \tag{15}$$

The following physics-driven loss function, $J(\theta)$, is formed by collocating the residuals on the training points,

$$J(\theta) = \sum_{n=1}^{N_{\text{int}}}||\mathcal{R}_{\text{int},\theta}(t_n,\boldsymbol{x}_n)||^2 \tag{16}$$

where, N_{int} are the interior training points (e.g., the points sampled from $[t_0,t_f]$ and Ω),

Algorithm 2: PINN TFC-based algorithm for BeNN training

```
Input: see Alg. 1, but only the interior training points are
       needed.
Goal: see Alg. 1, but the loss to minimize is the one of Eq.
      (16)
Step 1: Approximate the real, unknown, $V$ with the
        functional $V^\theta$ given in Eq. (13), which is the BeNN.
Step 2: see Alg. 1
Step 3: see Alg. 1
```

As previously stated, using a PINN TFC-based framework removes one of the main limitations of the traditional PINN methods. Competing objectives during the PINN training, which are minimizing the DE residuals within the domain (unsupervised learning task) and learning the equation constraints (supervised learning task), arise because the constraints of the equation are not analytically fulfilled. Thus, unbalanced gradients can occur during the PINN training via gradient-based methods. This may cause the PINN failure in learning the correct DE solution [28]. Indeed, when multiple competing objectives are present, gradient-based techniques may diverge or get stuck in limit cycles [29, 30]. Moreover, when using gradient-based methods the unknown NN parameters appear non-linearly. Thus, it becomes prohibitive to provide any meaningful initial guess for nonlinear DEs. Two different PINN TFC-based approaches are employed: Deep-TFC and X-TFC. Within Deep-TFC the CE free-function is a deep NN, whose parameters $\boldsymbol{\theta}$ are learned via gradient-based methods. Deep-TFC removes the issue of having competing objectives in the loss function, but it does not remove the issue of providing the initial guess for sensitive nonlinear DEs. Indeed, as the NN parameters appear non-linearly in the functional V^θ, it is prohibitive to provide an initial guess for nonlinear DEs when it is needed. Within X-TFC the CE free-function is represented by a shallow NN trained with ELM algorithm, which randomly samples input weights and bias. These are kept fixed during the training procedure. Thus, the output weights are the only trainable parameters [25, 31]. Thanks to this feature, the training is performed with a robust and fast least-squares. Hence, X-TFC eliminates all the limitations previously mentioned. In particular, as the unknowns (e.g., output weights) appear linearly in the functional V^θ, it is possible to provide a meaningful initial guess when required. For OCPs, the HJB equation is always nonlinear. In many cases, when needed, a good initial guess would be a quadratic value function dependent on the state vector. As an example, the following expression could be used: $V_{\text{guess}}(\boldsymbol{x}) = \boldsymbol{x}^T P \boldsymbol{x}$, where P is a positive definite matrix. Although X-TFC does not, in principle, need a stabilizing control, the latter can be alternative as an informative initial guess. As X-TFC solves a non-linear optimization via a sequence of approximation that converges toward a (local) minimum, one can select as starting point for the optimization, an initial stabilizing controller with a defined region of attraction and use the theorems from Beard et al. [32] to conclude that the region of attraction is at least as large as the one for the initial stabilizing controller. Although the extension is straightforward, we believe that stability analysis requires further investigation which is out of the scope of this work. However, an informative initial guess is not always needed, especially if the X-TFC framework is used, for which even a random initialization of the value function can allow the BeNN to learn the correct optimal control actions.

B. *Infinite Horizon Problems*

Considering a typical LQR integral cost, these problems are posed as following,

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x},\boldsymbol{u}) = \int_0^\infty \left(Q(\boldsymbol{x}) + \boldsymbol{u}^T R \boldsymbol{u}\right) \mathrm{d}t \tag{17}$$

subject to

$$\begin{aligned}\dot{\boldsymbol{x}} &= \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0\end{aligned} \tag{18}$$

where $\boldsymbol{x}\in\Omega\subseteq\mathbb{R}^n$, $\boldsymbol{u}\in\mathcal{U}\subseteq\mathbb{R}^m$ are the states, and control respectively. The system dynamics $\boldsymbol{f}\in\mathbb{R}^n$ is assumed to be known. The function $Q(\boldsymbol{x})$ is positive definite, and $R\in\mathbb{R}^{n\times m}$ is a symmetric positive definite matrix. The main characteristic of the infinite horizon problems is the presence of the matrices Q and R which can affect and regulate the time when the system converges to equilibrium. If these matrices are not properly set, the desired final condition near the equilibrium will not be reached. The value function V is,

$$V(\boldsymbol{x}) = \inf_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x},\boldsymbol{u}) \tag{19}$$

which is the unique solution of the following HJB equation,

$$\sup_{\boldsymbol{u}\in\mathcal{U}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \left(Q(\boldsymbol{x}) + \boldsymbol{u}^T R\boldsymbol{u}\right)\} = 0, \quad \forall \boldsymbol{x}\in\Omega \tag{20}$$

$$\text{subject to} \quad V(\boldsymbol{0}) = 0 \tag{21}$$

Even for infinite horizon OCPs, the proposed procedure can be applied to a generic dynamical system. However, if the dynamics is affine in the control (i.e., $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$), the optimal control $\boldsymbol{u}_{\text{opt}}$ can be derived in a closed analytical form. Since most of the infinite horizon OCPs of interest for real-world problems belong to this class, affine systems are considered and derived in the following of this work without loss of generality. Hence:

$$\boldsymbol{u}_{\text{opt}} = \underset{\boldsymbol{u}\in\mathcal{U}}{\text{argsup}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \Big(Q(\boldsymbol{x}) + \boldsymbol{u}^T R\boldsymbol{u}\Big)\} = -\frac{1}{2}R^{-1}g^T(x)V_{\boldsymbol{x}} \tag{22}$$

Once $\boldsymbol{u}_{\text{opt}}$ is derived, its closed analytical form is plugged back into (20). Then equation (20) reduces to a nonlinear PDE that is solved for V,

$$Q(\boldsymbol{x}) + V_{\boldsymbol{x}}^T\boldsymbol{f}(\boldsymbol{x}) - \frac{1}{4}V_{\boldsymbol{x}}^T g(\boldsymbol{x})R^{-1}g^T(\boldsymbol{x})V_{\boldsymbol{x}} = 0, \quad \forall\boldsymbol{x}\in\Omega \tag{23}$$

$$\text{subject to} \quad V(\boldsymbol{0}) = 0 \tag{24}$$

Equation (23) can be rewritten in the following form,

$$\mathcal{D}(V(\boldsymbol{x})) = \mathcal{F}(\boldsymbol{x}), \quad \forall\boldsymbol{x}\in\Omega \tag{25}$$

$$\text{subject to} \quad V(\boldsymbol{0}) = 0 \tag{26}$$

where $\mathcal{D}(\cdot)$ is a nonlinear differential operator acting on V, and $\mathcal{F}(t,\boldsymbol{x}) = -Q(\boldsymbol{x})$ is the known forcing term. The problem (25)-(26) is in the same form of problem (7)-(8). Even for the solution of infinite horizon OCPs via PINN-based frameworks, the main challenge is the selection of the matrices Q and R and the fact that the nonlinear PDE (25) must be learned simultaneously with the temporal boundary condition (26). Thus algorithms 1 and 2 are applied to learn $V(\boldsymbol{x})$. For the type of infinite horizon problems considered in this work, the CE for the PINN TFC-based framework is,

$$V^\theta(\boldsymbol{x}) = g^\theta(\boldsymbol{x}) - g^\theta(\boldsymbol{0}) \tag{27}$$

The considerations about the convenience in using PINN TFC-based methods made for the finite horizon problems hold also for the infinite horizon ones.

C. Estimate on the Generalization Error

In this section, following the steps in Ref. [26], we will estimate the error generated by BeNN trained both via algorithm 1 and by algorithm 2 in learning the solution V of the problems (7)-(8) and (25)-(26), respectively. This error is the *total error*, also known as the *generalization error* [26, 33]. For classic PINN we have,

$$\varepsilon_G = \varepsilon_G(\theta^*; \mathcal{S}) = ||V - V^*||, \tag{28}$$

for PINN TFC-based we have,

$$\varepsilon_{G,\text{tfc}} = \varepsilon_{G,\text{tfc}}(\theta^*; \mathcal{S}) = ||V - V^*_{\text{tfc}}|| \tag{29}$$

That is, the generalization error is the absolute error between the real, V, solution and the approximated solution V^* (or V^*_{tfc}). From Eqs. (28) and (29) it can be seen how the generalization error is a function of the training set $\mathcal{S}$ and the trained BeNN with parameters θ^*, obtained via algorithms 1 and 2 respectively. Nevertheless, for convenience, we will omit the generalization error dependency on the trained parameters and the training set.
As the real solution V is not always available, it is not always possible to evaluate the generalization error, neither during nor after the training process. Conversely, the so-called *training* error can be monitored and evaluated a-posteriori once the trained is completed. In the following subsections, we will defined the training error for both classic PINN and PINN TFC-based training algorithms, and show how the generalization error is bounded by the training error and the quadrature error, which is the error associated to the choice of the training points.
For classic PINN, the *training error*, ε_T, is given by,

$$\varepsilon_T = J(\theta^*)^{\frac{1}{2}} = \left(\lambda \sum_{n=1}^{N_{\text{int}}} ||\mathcal{R}_{\text{int},\theta^*}(t_n, \boldsymbol{x}_n)||^2 + \sum_{n=1}^{N_{\text{tb}}} ||\mathcal{R}_{tb,\theta^*}(\boldsymbol{x}_n)||^2 \right)^{\frac{1}{2}} \tag{30}$$

That is, ε_T can be evaluated once the training is completed and ε_G can be estimated in terms of ε_T. Before stating the theorem showing the ε_G estimate in term of ε_T, several hypothesis need to made.
Let X, Y be separable Banach spaces with norms $||\cdot||_X$ and $||\cdot||_Y$, respectively. Let $X^* \subset X$ and $Y^* \subset Y$ be closed sub-spaces with norms $||\cdot||^*_X$ and $||\cdot||^*_Y$, respectively. In problems (7)-(8) and (25)-(26), the differential operator $\mathcal{D}$ is mapping, $\mathcal{D} : X^* \mapsto Y^*$, and the forcing term $\mathcal{F} \in Y^*$, such that,

$$\begin{cases} (H1): & ||\mathcal{D}(V)||_{Y^*} < \infty, \quad \forall V \in X^*, \quad \text{with} \quad ||V||_{X^*} < \infty \\ (H2): & ||\mathcal{F}||_{Y^*} < \infty \end{cases} \tag{31}$$

In addition, we make the assumption that $\forall \mathcal{F} \in Y^*$, $\exists!\ V \in X^*$ such that (7)-(8) and (25)-(26) hold. Furthermore, we assume that the solutions of (7)-(8) and (25)-(26) fulfill the stability bound: let $Z \subset X^* \subset X$ be a closed sub-space with norm $||\cdot||_Z$.

Assumption II.1. *For any $V, U \in Z$, the differential operator $\mathcal{D}$ satisfies*

$$(H3): \quad ||V - U||_X \leq C_{\text{pde}}\left(||V||_Z, ||U||_Z\right) ||\mathcal{D}(V) - \mathcal{D}(U)||_Y, \tag{32}$$

where C_{DE} is a positive constant the explicitly depends on $|\boldsymbol{u}|_Z$ and $|\boldsymbol{v}|_Z$. That is,

$$C_{\text{pde}} = C_{\text{pde}}\left(||V||_Z, ||U||_Z\right) > 0.$$

Based on the assumptions above, that according to Ref. [26] require that both the differential operator and the forcing term are finite and bounded (H1 and H2, respectively), and that the DE is subject to a *conditional stability estimate* (H3), the following theorem holds.

Theorem 1. *Let $V \in Z \subset X^*$ be the unique solution of the DE (7)-(8) (and (25)-(26)) and assume that the stability hypothesis (H3) holds. Let $V^* \in Z \subset X^*$ be the BeNN computed via algorithm 1, based on the chosen training set $\mathcal{S}$. The following estimate on the generalization error holds,*

$$\varepsilon_G \leq C_{\text{pde}} \varepsilon_T + C_{\text{pde}} C_{\text{train}}^{\frac{1}{2}} N^{-\frac{\alpha}{2}} \tag{33}$$

with $C_{\text{pde}} = C_{\text{pde}}(||V||_Z, ||V^*||_Z)$ and $C_{\text{train}} = C_{\text{train}}(||\mathcal{R}_{\theta^*}||^2)$ (with $\mathcal{R}^*_\theta$ including interior and temporal boundary) stemming from (H3). It is worth to notice that the constants C_{pde}, C_{train} depend on the BeNN, V^*, and on the number of training points $N = N_{\text{int}} + N_{\text{tb}}$.

Proof: For this proof, we indicate as $\mathcal{R} = \mathcal{R}_{\theta^*}$ the residual corresponding to the trained BeNN, V^*. Here, in the residual term $\mathcal{R}$ we consider all the components (e.g., interior and temporal boundary). The residual is the following,

$$\mathcal{R} = \mathcal{D}(V^*) - \mathcal{F} \tag{34}$$

As V solves the DE (7)-(8) (and (25)-(26)) the following holds true,

$$\mathcal{D}(V) - \mathcal{F} = 0 \tag{35}$$

Thus, equation (34) can be rewritten as follows,

$$\mathcal{R} = (\mathcal{D}(V^*) - \mathcal{F}) - (\mathcal{D}(V) - \mathcal{F}) = \mathcal{D}(V^*) - \mathcal{D}(V) \tag{36}$$

We have, by (28),

$$\varepsilon_G = ||V - V^*|| \tag{37}$$

Because of the stability bound H3, the following holds true,

$$\varepsilon_G = ||V - V^*|| \leq C_{\text{pde}} ||\mathcal{D}(V^*) - \mathcal{D}(V)|| \tag{38}$$

Therefore, as by (36) $||\mathcal{D}(V^*) - \mathcal{D}(V)|| = ||\mathcal{R}||$, we have,

$$\varepsilon_G \leq C_{\text{pde}} ||\mathcal{R}|| \tag{39}$$

As the residuals are collocated on the training points, we have the following bound (due to the quadrature error, see [26] and references within for further details),

$$||\mathcal{R}||^2 \leq \varepsilon_T^2 + C_{\text{train}} N^{-\alpha}, \tag{40}$$

where the term $C_{\text{train}} N^{-\alpha}$ is the error associated to the choice of the training points. Then,

$$\varepsilon_G^2 \leq C_{\text{pde}}^2 |\mathcal{R}|^2 \leq C_{\text{pde}}^2 \left(\varepsilon_T^2 + C_{\text{train}} N^{-\alpha}\right)$$

Finally,

$$\varepsilon_G^2 \leq C_{\text{pde}}^2 \left(\varepsilon_T^2 + C_{\text{train}} N^{-\alpha}\right) = C_{\text{pde}}^2 \varepsilon_T^2 + C_{\text{pde}}^2 C_{\text{train}} N^{-\alpha}$$

Thus,

$$\varepsilon_G \leq C_{\text{pde}}\varepsilon_T + C_{\text{pde}}C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}}$$

which is the estimate (33). This completes the proof. ■

The estimate (33) suggests that the generalization error when using algorithm 1 is small under several conditions. The BeNN must be *well-trained*. The training error must be sufficiently small (e.g., $\varepsilon_T << 1$). Moreover, the training error can only be computed a posteriori, since we do not have any prior control on it. The error associated with the training points depends on the choice and the number of training points N as well as on the training constant C_{train}. The latter also depends on the residual of the BeNN, V^*, and thus, indirectly, on the number of training points N (where $N = N_{\text{int}} + N_{\text{tb}}$). That is, N needs to be large enough such that $C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}} << 1$. Importantly, the constant C_{train} also depends on the architecture of the underlying BeNN. The C_{train} evaluation depends on the features of the governing DEs, and the training point collocation can not be worked out in the set up of the Theorem 1 [34]. The constant C_{pde} encodes the stability of the DEs and depends on both the exact (unknown) solution V and the trained BeNN, V^*, that needs to be bounded.

The main point of the estimate (33) is to provide an upper bound of the generalization error in terms of the training and quadrature errors, regardless of the choice of the BeNN architecture. The estimate (33) holds for any expansion of V, including NNs (of any types), and so does the algorithm 1. There is no guarantee that the training error is small if NNs are used to approximate V. However, it can be expected that if the DE is stable and the training points collocation scheme is accurate (e.g., there is some control on C_{pde} and C_{train}) the training error will be small, leading to a small generalization error.

For PINN TFC-based, the *training error* ε_T is given by,

$$\varepsilon_T = J(\theta^*)^{\frac{1}{2}} = \left(\sum_{n=1}^{N_{\text{int}}} w_n^{\text{int}}||\mathcal{R}_{\text{int},\theta^*}(t_n, \boldsymbol{x}_n)||^2\right)^{\frac{1}{2}} \tag{41}$$

That is, ε_T can be evaluated once the training is completed and $\varepsilon_{G,\text{tfc}}$ can be estimated in terms of ε_T according to the following theorem, under the same assumptions made for the previous theorem.

Theorem 2. *Let $V \in Z \subset X^*$ be the unique solution of the DE (7)-(8) (and (25)-(26)) and assume that the stability hypothesis (H3) holds. Let $V_{\text{tfc}}^* \in Z \subset X^*$ be the BeNN computed via algorithm 2, based on the chosen training set $\mathcal{S}$. The following estimate on the generalization error holds,*

$$\varepsilon_{G,\text{tfc}} \leq C_{\text{pde}}\varepsilon_T + C_{\text{pde}}C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}} \tag{42}$$

with $C_{\text{pde}} = C_{\text{pde}}(||V||_Z, ||V_{\text{tfc}}^*||_Z)$ and $C_{\text{train}} = C_{\text{train}}(||\mathcal{R}_{\theta^*}^2||)$ (with $\mathcal{R}_\theta^*$ including solely the interior residual) stemming from (H3). Note that these constants C_{pde}, C_{train} depend on the BeNN, V_{tfc}^*, and on the number of training points $N = N_{\text{int}}$.

Proof: The proof of theorem 2 is equivalent to the proof of theorem 1. Thus it will not be repeated. ■

The considerations made for classic PINN holds true for PINN TFC-based.

III. Numerical Results

A finite and an infinite horizon OCPs with integral quadratic cost, both with linear and nonlinear dynamics, have been solved using the proposed algorithms. The problems selected are benchmarks problems with analytical solutions. We selected these problems to allow the readers to better appreciate the performances, understand the utility of the proposed PINN-based framework, and guide them through the step-by-step formulation using all the PINN frameworks. Moreover, having the analytical solutions allows us to compute the generalization error, and therefore to numerically test the theoretical findings. The formulation and the results are showed and analyzed below. The selected OCPs have been coded in Python 3.7 and ran with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM. The training points are evenly spaced along all the dimensions, forming a uniform grid of points.

A. Problem 1: Finite Horizon Problem with Linear Dynamics

Consider the following finite horizon problem with linear dynamics (example 1 from [27]),

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x}) = (x(t_f))^2 + \int_0^{t_f} u^2\, dt \tag{43}$$

$$\text{subject to} \qquad \dot{x} = x + u \tag{44}$$

with t_f given, in particular $t_f = 1$.

The HJB equation for the problem is,

$$-V_t + \sup_u \left\{-V_x(x+u) - u^2\right\} = 0, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{45}$$

subject to,

$$V(1,x) = x^2, \quad \forall x \in [-1,1] \tag{46}$$

The optimal control is,

$$u = -\frac{1}{2}V_x \tag{47}$$

Thus, the resulting HJB PDE, which solution is learned using both algorithms 1 and 2, is,

$$-V_t - V_x x + \frac{1}{4}V_x^2 = 0, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{48}$$

subject to,

$$V(1,x) = x^2, \quad \forall x \in [-1,1] \tag{49}$$

The analytical expression of the value function is,

$$V_{\text{exact}}(t,x) = \frac{2x^2}{1+\exp\{2(t-1)\}} \tag{50}$$

thus the exact optimal control is,

$$u_{\text{exact}}(t,x) = -\frac{2x}{1+\exp\{2(t-1)\}} \tag{51}$$

To generate the results, different PINN frameworks have been considered for comparison, such as classic PINNs, Deep-TFC, and X-TFC. All the NNs employ the hyperbolic tangent as the activation function. For classic PINN and Deep-TFC, the unknown parameters θ have been randomly initialized. For X-TFC, input weights and bias have been sampled from $\mathcal{U}[-1,1]$, and the output weights have been initialized by setting them all equal to zero. When classic PINN and Deep-TFC are employed for the BeNNs training, the ADAM optimizer with initial learning rate equal to 0.001 has been adopted [23]. For the training with X-TFC, iterative least-squares with tolerance equal to 10^{-8} has been used (note that the tolerance is defined according to the losses of two subsequent iterations). The results are reported in Tables I-II, where ε_G and ε_T represent the generalization error and the training error as defined in section 2, respectively. Table I shows the BeNNs training with $N_{\text{int}} = 400$ internal training points, and $N_{\text{tb}} = 200$ for classic PINNs. As can be seen, X-TFC generally outperforms the other two methods, both in terms of accuracy and computational times. Moreover, it is worth to notice that, in order to obtain the same accuracy of classic PINN and Deep-TFC, X-TFC requires just one layer and also a lower number of neurons (as illustrated by the case with $L = 20$). The corresponding training time, in the order of 16 milliseconds, can also suggest a potential online (and real-time) training of the BeNNs. This is very promising for future applications. For Table II, the same hyperparameters of Table I are used, apart from the internal training points ($N_{\text{int}} = 4900$ for Table II), and the temporal boundary training points for classic PINNs ($N_{\text{tb}} = 2450$ for Table II). As expected, the training time for the BeNNs training increases as a function of the higher number of training points. However, especially for classic PINNs and Deep-TFC, more training points do not lead to obtain more accurate solutions for this example. This is also valid for X-TFC, for which an accurate solution was already found for fewer training points. The attentive reader can also observe that the generalization error ε_G is lower than the training error ε_T, proving numerically theorems 1 and 2. For the case of X-TFC with $N_{\text{int}} = 400$ and $L = 200$, another simulation has been performed to test the deployment of the network. Indeed, the optimal control actions obtained from the value function learned during the training have been exploited to propagate (44) forward in time, starting from the initial condition $x_0 = -1$. The corresponding results are reported in Fig. 1. The left panels show the integrated state with the BeNN-based optimal control action. The right panel reports the absolute error with respect to the integrated analytical solution. The absolute error is reported to be in the order of 10^-11, proving that the proposed methodology is capable of providing highly accurate solutions. A Monte Carlo analysis has also been performed, considering 1000 different initial conditions sampled randomly from a uniform distribution within the range [-1,1]. The distribution of the errors between the state at final time, obtained after the propagation, and the corresponding analytical solutions is reported in the histogram of Fig. 2. The error in the order of 10^{-10} demonstrates that BeNNs can robustly and accurately learn the optimal control actions even with perturbed initial conditions.

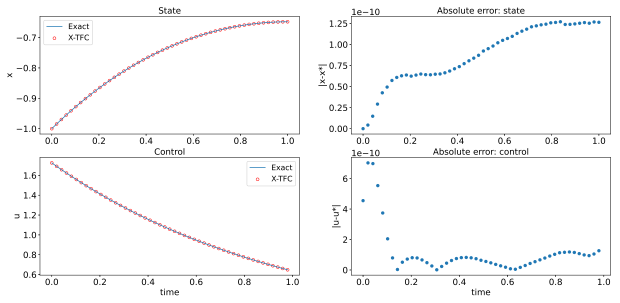

Figure 1. States, control and the corresponding error during the integration for the finite horizon problem ($N_{\text{int}} = 400$, $L = 20, x_0 = -1$). L is the number of neurons.

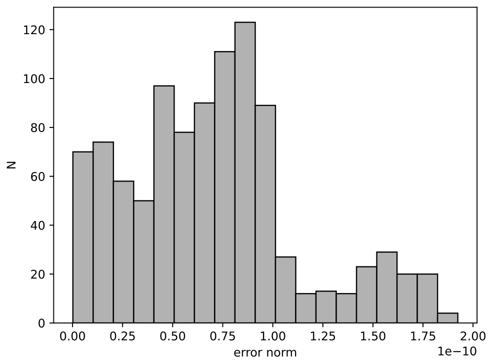

Figure 2. Histogram of the error between the final approximated and true states in the Monte Carlo analysis ($N_{\text{int}} = 400$, $L = 20$)

B. Problem 2: Infinite Horizon Problem with Nonlinear Dynamics

Consider the following infinite horizon problem with nonlinear dynamics (example 2 from [35]),

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J} = \int_0^\infty (\boldsymbol{x}^T\boldsymbol{x} + u^2)\,dt \tag{52}$$

subject to

$$\begin{aligned} \dot{x_1} &= -x_1 + x_2 \\ \dot{x_2} &= -\frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_1^2 x_2 + x_1\,u \end{aligned} \tag{53}$$

Using equation (22), the optimal control is,

$$u = -\frac{1}{2}x_1 V_{x_2} \tag{54}$$

Thus the resulting HJB PDE, which solution is learned using both algorithms 1 and 2, is,

$$x_1^2 + x_2^2 + (-x_1 + x_2)\,V_{x_1} + \left(-\frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_1^2 x_2\right)V_{x_2} - \frac{1}{4}x_1^2 V_{x_2}^2 = 0 \tag{55}$$

subject to the following condition,

$$V(\mathbf{0}) = 0 \tag{56}$$

in $x_1 \in [-1,1]$ and $x_2 \in [-1,1]$.

Table I
RESULTS FOR PROBLEM 1, WITH $N_{\text{int}} = 400$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	348.87	6.89×10^{-4}	2.00×10^{-3}
Classic PINN	4	200	15000	668.80	6.41×10^{-4}	1.17×10^{-2}
Classic PINN	10	200	15000	1647.90	1.22×10^{-3}	4.01×10^{-3}
Deep-TFC	2	200	15000	320.78	2.48×10^{-5}	1.47×10^{-4}
Deep-TFC	4	200	15000	662.10	4.87×10^{-5}	1.60×10^{-4}
Deep-TFC	10	200	15000	1666.71	1.16×10^{-4}	4.16×10^{-4}
X-TFC	1	20	8	0.016	3.09×10^{-4}	1.44×10^{-3}
X-TFC	1	200	4	3.95	2.91×10^{-11}	8.56×10^{-11}
X-TFC	1	2000	4	15.95	2.04×10^{-11}	3.79×10^{-11}

Table II
RESULTS FOR PROBLEM 1, WITH $N_{\text{int}} = 4900$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	1048.82	6.13×10^{-4}	3.35×10^{-3}
Classic PINN	4	200	15000	2356.83	5.40×10^{-4}	2.46×10^{-3}
Classic PINN	10	200	15000	4231.86	3.50×10^{-3}	5.12×10^{-2}
Deep-TFC	2	200	15000	1671.83	3.12×10^{-5}	1.71×10^{-4}
Deep-TFC	4	200	15000	2349.12	4.13×10^{-5}	2.38×10^{-4}
Deep-TFC	10	200	15000	2856.98	6.42×10^{-5}	2.31×10^{-4}
X-TFC	1	20	8	6.05	2.36×10^{-4}	1.23×10^{-3}
X-TFC	1	200	4	9.56	4.78×10^{-11}	5.82×10^{-10}
X-TFC	1	2000	4	292.22	1.80×10^{-11}	1.03×10^{-10}

The exact solution for the value function and the optimal control are,

$$V_{exact}(\boldsymbol{x}) = \frac{1}{2}x_1^2 + x_2^2 \tag{57}$$

$$u_{\text{exact}}(t, x) = -x_1 x_2 \tag{58}$$

For what concerns the numerical results, the same considerations carried out for the previous example are still valid. For this problem, one training point in $\boldsymbol{x} = \boldsymbol{0}$ has been used for classic PINNs, whereas the tolerance for X-TFC iterative least-squares has been set to 10^{-5}. Even for this case, it is possible to observe from Tables III-IV that the best accuracy and computational times are obtained with X-TFC. Moreover, Fig. 3 shows the results of the integration of Eqs. (53) under the application of the deployed optimal control actions learned during the training of the BeNNs. The comparison with respect to the analytical solution and the corresponding absolute errors between the real and approximated states and control prove the accuracy of the BeNNs training. The results of a Monte Carlo analysis for 1000 simulations with initial conditions sampled from a uniform random distribution within the range $[-\mathbf{1}, \mathbf{1}]$ are reported in Fig. 4. The errors between the approximated and analytical final states are on the order of 10^{-10}, proving the robustness of the BeNNs to compute accurate solutions starting from different initial states within the domain.

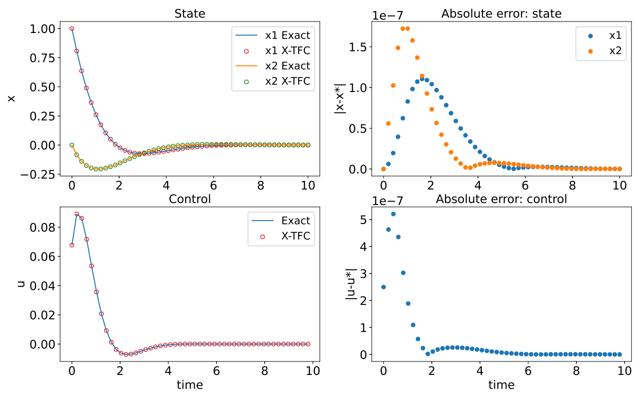

Figure 3. States, control and the corresponding error during the integration for the infinite horizon problem 2 ($N_{\text{int}} = 400$, $L = 20$, $X_0 = -1$)

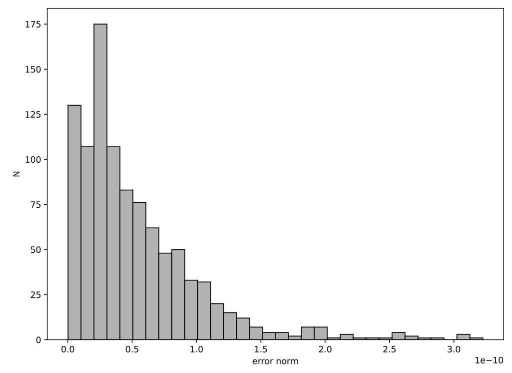

Figure 4. Histogram of the error between the final approximated and true states in the Monte Carlo for the infinite horizon problem 2 analysis ($N_{\text{int}} = 400$, $L = 20$)

IV. CONCLUSIONS

In this paper, we focused on the development of the BeNNs and used them to learn the HJB PDE solutions for OCPs with integral quadratic cost. Using BeNNs allows to retrieve the optimal control in an analytical form, represented by the NNs. Indeed, once the optimal control is learned on the training points, no further interpolation (usually needed for traditional methods) is required to evaluate it on the test/query

Table III
RESULTS FOR PROBLEM 2, WITH $N_{\text{int}} = 400$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	199.69	1.06×10^{-3}	2.03×10^{-3}
Classic PINN	4	200	15000	300.86	1.20×10^{-3}	1.01×10^{-2}
Classic PINN	10	200	15000	553.83	3.74×10^{-3}	2.79×10^{-1}
Deep-TFC	2	200	15000	210.02	2.48×10^{-5}	3.41×10^{-5}
Deep-TFC	4	200	15000	459.02	1.98×10^{-5}	2.03×10^{-5}
Deep-TFC	10	200	15000	1236.61	2.21×10^{-5}	3.38×10^{-5}
X-TFC	1	20	8	0.016	5.30×10^{-4}	6.91×10^{-4}
X-TFC	1	200	5	4.02	1.44×10^{-7}	2.71×10^{-7}
X-TFC	1	2000	5	16.47	8.16×10^{-7}	1.06×10^{-6}

Table IV
RESULTS FOR PROBLEM 2, WITH $N_{\text{int}} = 4900$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	571.16	3.60×10^{-4}	1.24×10^{-3}
Classic PINN	4	200	15000	1074.71	1.84×10^{-3}	1.20×10^{-2}
Classic PINN	10	200	15000	2380.45	3.49×10^{-3}	3.07×10^{-2}
Deep-TFC	2	200	15000	541.1	2.14×10^{-5}	3.08×10^{-5}
Deep-TFC	4	200	15000	1099.01	2.96×10^{-5}	3.14×10^{-5}
Deep-TFC	10	200	15000	2806.87	2.18×10^{-5}	3.14×10^{-5}
X-TFC	1	20	7	4.02	2.35×10^{-4}	5.48×10^{-4}
X-TFC	1	200	5	88.11	2.98×10^{-6}	3.00×10^{-6}
X-TFC	1	2000	5	289.89	9.78×10^{-7}	1.10×10^{-6}

points. This results in a closed-loop optimal control that can be deployed to compute optimal trajectories in real-time, as showed in Fig. 5. Indeed, the deployment time is in order of milliseconds. Three PINN frameworks have been tested and compared. The main goal of this work is to show the feasibility in using PINN-based methods to learn closed-loop optimal control, by directly solving the HJB equation, and to select the best PINN method for the class of OCPs considered. The results show that X-TFC is the best option to tackle this class of OCPs. Our theoretical findings on the estimation of the generalization error generated by BeNNs in learning the solution of the HJB PDEs has been tested through benchmark OCPs. The low computational time required for the training of BeNNs also suggests a possible use for online training. Further investigations will be carried out in the future about this possibility and its applications. The proposed approach can be applied in all the research fields where OCPs with integral quadratic cost must be solved. The authors are currently focusing on the space engineering applications, because of the significant importance that generating real-time closed-loop optimal control brings to spacecraft autonomy. In particular, satellite optimal attitude control and trajectory planning are currently under investigation. In addition, efforts are in progress to solve large-scale OCPs belonging to the class considered here and for other classes of OCPs, such as problems with discontinuous control. Future works will also focus on the stability analysis of the closed-loop system obtained with the control generated from the solution of the HJB equation [36]. Finally, future works will focus on the comparison of BeNNs against some of the most common and used methods to solve HJB PDEs.

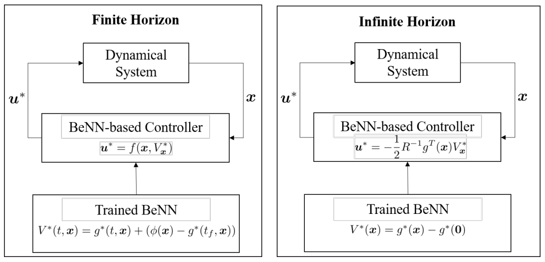

Figure 5. Schematic on how to deploy trained BeNNs to control a dynamical system

REFERENCES

[1] E. Schiassi, A. D'Ambrosio, A. Scorsoglio, R. Furfaro, and F. Curti, "Class of optimal space guidance problems solved via indirect methods and physics-informed neural networks," in *31st AAS/AIAA Space flight Mechanics Meeting*, 2021.

[2] L. Kong, W. He, Y. Dong, L. Cheng, C. Yang, and Z. Li, "Asymmetric bounded neural control for an uncertain robot by state feedback and output feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 3, pp. 1735–1746, 2019.

[3] Z. Li, X. Li, Q. Li, H. Su, Z. Kan, and W. He, "Human-in-the-loop control of soft exosuits using impedance learning on different terrains," *IEEE Transactions on Robotics*, 2022.

[4] B. R. E., "Dynamic programming," *Princeton University, Press, Princeton, NJ*, 1957.

[5] C. M. Chilan and B. A. Conway, "Optimal nonlinear control using hamilton–jacobi–bellman viscosity solutions on unstructured grids," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 1, pp. 30–38, 2020.

[6] E. Cristiani and P. Martinon, "Initialization of the shooting method via the hamilton-jacobi-bellman approach," *Journal of Optimization Theory and Applications*, vol. 146, no. 2, pp. 321–346, 2010.

[7] J. N. Reddy, "An Introduction to the Finite Element Method," *Journal of Pressure Vessel Technology*, vol. 111, no. 3, pp. 348–349, 08 1989. [Online]. Available: https://doi.org/10.1115/1.3265687

[8] H.-G. Zhang, X. Zhang, L. Yan-Hong, and Y. Jun, "An overview of research on adaptive dynamic programming," *Acta Automatica Sinica*, vol. 39, no. 4, pp. 303–311, 2013.

[9] Y. Li, J. Zhang, W. Liu, and S. Tong, "Observer-based adaptive optimized control for stochastic nonlinear systems with input and state constraints," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[10] S. Baldi, I. Michailidis, E. B. Kosmatopoulos, A. Papachristodoulou, and P. A. Ioannou, "Convex design control for practical nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 7, pp. 1692–1705, 2014.

[11] E. B. Kosmatopoulos, "Control of unknown nonlinear systems with efficient transient performance using concurrent exploitation and exploration," *IEEE Transactions on Neural Networks*, vol. 21, no. 8, pp. 1245–1261, 2010.

[12] I. Michailidis, S. Baldi, E. B. Kosmatopoulos, and P. A. Ioannou, "Adaptive optimal control for large-scale nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5567–5577, 2017.

[13] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[14] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Model-free q-learning designs for linear discrete-time zero-sum games with application to h-infinity control," *Automatica*, vol. 43, no. 3, pp. 473–481, 2007.

[15] 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*, vol. 378, pp. 686–707, 2019.

[16] D. Mortari, "The Theory of Connections: Connecting Points," *MDPI Mathematics*, vol. 5, no. 57, 2017.

[17] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, and D. Mortari, "Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations," *Neurocomputing*, 2021.

[18] C. Leake and D. Mortari, "Deep theory of functional connections: A new method for estimating the solutions of partial differential equations," *Machine learning and knowledge extraction*, vol. 2, no. 1, pp. 37–55, 2020.

[19] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[20] C. Leake, H. Johnston, and D. Mortari, "The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations," *Mathematics*, vol. 8, no. 8, p. 1303, 2020.

[21] A. D'Ambrosio, E. Schiassi, F. Curti, and R. Furfaro, "Pontryagin neural networks with functional interpolation for optimal intercept problems," *Mathematics*, vol. 9, no. 9, p. 996, 2021.

[22] E. Schiassi, A. D'Ambrosio, K. Drozd, F. Curti, and R. Furfaro, "Physics-informed neural networks for optimal planar orbit transfers," *Journal of Spacecraft and Rockets*, pp. 1–16, 2022.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[25] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, " Extreme learning machine: Theory and applications ," *Neurocomputing*, vol. 70, no. 2006, pp. 489–501, May 2006.

[26] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics-informed neural networks for approximating pdes," *IMA Journal of Numerical Analysis*, 2022.

[27] M. Mehrali-Varjani, M. Shamsi, and A. Malek, "Solving a class of hamilton-jacobi-bellman equations using pseudospectral methods," *Kybernetika*, vol. 54, no. 4, pp. 629–647, 2018.

[28] D. Y. Wang, *Study Guidance and Control for Lunar Soft Landing (Ph.D. Dissertation).* School of Astronautics, Harbin Institute of Technology, Harbin, China, 2000.

[29] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, "Cycles in adversarial regularized learning," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, pp. 2703–2717.

[30] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games," in *International Conference on Machine Learning*. PMLR, 2018, pp. 354–363.

[31] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, p. 879–892, 2006.

[32] R. W. Beard, G. N. Saridis, and J. T. Wen, "Approximate solutions to the time-invariant hamilton–jacobi–bellman equation," *Journal of Optimization theory and Applications*, vol. 96, no. 3, pp. 589–626, 1998.

[33] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[34] T. De Ryck and S. Mishra, "Error analysis for physics informed neural networks (pinns) approximating kolmogorov pdes," *arXiv preprint arXiv:2106.14473*, 2021.

[35] W. Tang and P. Daoutidis, "Distributed adaptive dynamic programming for data-driven optimal control," *Systems & Control Letters*, vol. 120, pp. 36–43, 2018.

[36] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, "Neural network optimal feedback control with enhanced closed loop stability," *arXiv preprint arXiv:2109.07466*, 2021.