Neural Network-Based Optimization of LEO Transfers

aerospace

MDPI

Article

Neural Network-Based Optimization of LEO Transfers

Andrea Forestieri *,† **and Lorenzo Casalino** †

Dipartimento di Ingegneria Meccanica e Aerospaziale, Politecnico di Torino, Corso Duca d egli Abruzzi 24, 10129 Torino, Italy; lorenzo.casalino@polito.it

* Correspondence: andrea.forestieri@polito.it

† These authors contributed equally to this work.

Abstract: This study investigates the application of neural networks to the evaluation of minimum-time low-thrust transfers in low Earth orbit. The findings demonstrate the effectiveness of utilizing costates to regularize the training loss, significantly enhancing the accuracy of the predictions of the neural networks, even when working with limited datasets. Remarkably precise estimates of transfer times are achieved by training the regularized networks on datasets comprising one million samples. The incorporation of a warm-started guess strategy, involving simpler neural networks to provide transfer time and costates predictions for new transfers, accelerates the data collection process, making this approach highly practical for real-world applications. Overall, the methodology employed in this research study holds significant promise for low-thrust space missions, particularly when the evaluation of multiple minimum-time transfers is necessary in mission planning. In fact, the trained neural networks significantly speed up convergence when solving optimal control problems with indirect optimization methods. Furthermore, the remarkable accuracy in estimating both minimum transfer times and costates provides the flexibility of relying entirely on neural networks for determining minimum time.

Keywords: neural network; machine learning; trajectory optimization; LEO orbit

Citation: Forestieri, A.; Casalino, L. Neural Network-Based Optimization of LEO Transfers. *Aerospace* **2024**, *11*, 879. https://doi.org/10.3390/aerospace11110879

Academic Editor: Fanghua Jiang

Received: 26 September 2024
Revised: 21 October 2024
Accepted: 22 October 2024
Published: 25 October 2024

1. Introduction

Low-thrust electric propulsion is a favored choice for numerous space missions due to its significant benefits in terms of propellant consumption [1]. Mission designers often seek to evaluate time-optimal and fuel-optimal transfer trajectories. The optimization of low-thrust trajectories is typically formulated as an optimal control problem (OCP), which can be typically solved by using numerical approaches, classified as direct or indirect methods.

Direct methods are known for their flexibility and robustness, although they require increased computational resources to achieve high accuracy. In contrast, indirect methods offer several advantages, including high numerical accuracy and the ability to ensure that any solution satisfies the necessary optimality conditions. These methods employ Pontyagin's Maximum Principle (PMP) and result in a boundary value problem (BVP). However, the challenge with indirect methods lies in solving the BVP by using shooting methods, which heavily rely on accurate initial guesses for the costates. Since the costates may not possess physical meaning, providing precise initial guesses is rarely a simple task. This limitation reduces the robustness of indirect methods and makes their implementation time-consuming, especially when evaluating numerous transfers.

To address these limitations, recent advancements have introduced methods such as the enhanced smoothing technique described in [2] and the Uniform Trigonometrization Method detailed in [3,4]. These approaches offer significant improvements in the solution process for indirect methods, even when dealing with control and state constraints. Their contributions underscore that this is a very active area of research, with continuous efforts to improve the practicality and robustness of indirect methods.

Despite these advancements, challenges remain, particularly in multitarget missions, where the objective is to find the optimal sequence of transfers that minimizes an overall mission cost index. In these scenarios, the reliable evaluation of transfer legs is crucial, but the vast number of potential transfers makes classical optimization methods impractical for early mission planning stages. Instead, there is a need for the rapid estimation of transfer costs with sufficient accuracy, without delving deeply into the optimal control law of each individual trajectory.

Active debris removal (ADR) missions in low Earth orbit (LEO) and on-orbit servicing (OOS) missions of satellites represent classical examples of this kind of problems. A key challenge in these missions is the ability to rendezvous with multiple objects within a limited timeframe. As a matter of fact, the time that a servicer spacecraft spends in transfer orbits plays a pivotal role in determining the overall duration of a servicing mission. Therefore, the aim of this study is to explore the use of neural networks as function approximators to enhance the efficiency of designing multitarget missions in LEO by using electric propulsion.

Recently, interest in the application of machine learning (ML) techniques, in particular deep neural networks (DNNs), to different fields of optimal control has grown [5,6]. In the existing literature, the use of DNNs has been explored in several contexts which are relevant for space trajectory optimization problems.

First, some studies have applied DNNs for real-time guidance applications [7–13], highlighting their potential for immediate decision making in dynamic environments. In particular, a computationally efficient DNN is presented in [7], demonstrating its near-optimal performance as a controller across different scenarios. It tackles a significant issue in low-thrust propulsion, i.e., missed thrust events, and shows how the DNN can correct the trajectory when such events occur. In [8], the authors propose an approach that integrates optimal control theory with DNNs to tackle various challenges encountered in spacecraft missions. These challenges include navigating unknown deep space environments, managing limited communication capabilities, and handling complex dynamics. The proposed method involves training a DNN by using state–control and state–costate pairs derived from a high-fidelity algorithm. Specifically, the authors apply this method to address two specific spacecraft missions: the hypersonic reentry problem and the fuel-optimal moon landing problem. The trained DNN plays a pivotal role in enhancing the real-time performance of their algorithm: it is utilized to provide accurate guesses for the initial costates, thereby improving the efficiency of the indirect method during mission execution.

A second application domain of DNNs in space trajectory optimization is as a replacement of traditional methods for solving the BVP produced by indirect methods. Recent studies [14,15] have successfully applied this approach, demonstrating the capability of DNNs to simplify and speed up complex trajectory optimization tasks. In [14], the authors proposed an approach that utilizes artificial DNNs to approximate the solution of optimal control problems by minimizing an error function incorporating the PMP conditions.

An additional application domain, which is of specific interest for the present article, is the use of DNNs to accelerate global optimization processes, particularly those that require frequent evaluation of transfer costs [16–19]. The effectiveness of this approach stems from the representational power of DNNs, as they can learn the functional mapping between initial and final states to the objective function. However, obtaining a large and dense dataset for OCPs can be challenging. DNN models can be viewed as complex geometric transformations, and their generalization capability heavily relies on the ability to smoothly interpolate the dataset points. Therefore, having a large and dense dataset is crucial to achieving better models. In the framework of OCPs solved with indirect methods, the collection of samples requires solving BVPs, making dataset generation laborious.

To address this issue, researchers have explored the utilization of adjoint variables, introduced by the indirect formulation of optimal control problems, as a means to regularize the loss function of DNNs [12,13]. Adjoint variables, also known as costates, represent the gradient of the objective function with respect to variations in the initial states [20]. There-

fore, a term can be added to the DNN loss function that ensures that the derivative of the DNN's output (i.e., the predicted objective function) with respect to the initial states given as inputs to the DNN is equal to the initial value of the costates. This regularization aims at enhancing the generalization power of the DNN by constraining it to learn the underlying mathematical law governing the OCP. Consequently, the parameter space (weights and biases) that minimizes the DNN's loss function is restricted to configurations adhering to this property. This strategy, inspired by physics-informed neural networks (PINNs) [21], aims to enhance the DNN's performance even when dealing with small datasets.

This study seeks to leverage DNNs building on these advancements and develop a more efficient framework for planning multitarget missions, specifically for ADR and OOS operations. This work investigates the ability of DNNs to predict minimum transfer times (the value function) for low-thrust transfers in LEO. In [22], the focus was on building an approximation of optimal transfers for fast evaluations of minimum-time transfers in LEO, but the BVP solution was still required. In contrast, the approach presented here offers a significant advantage: while it can accelerate the convergence of the BVP, its greater benefit lies in its ability to directly estimate the transfer cost, thereby avoiding the need to solve the BVP and reducing computation time. While there is research on using DNNs to estimate the time of flight in LEO transfers [16,17], such studies are typically confined to impulsive transfers and do not use the PINN framework. Our research extends this concept to low-thrust transfers, which also involve constraints on state variables, such as maintaining a spacecraft above a certain altitude limit. This often results in complex optimal trajectories with a three-arc structure, where the middle arc is flown at the altitude limit. This significantly complicates the application of indirect methods, as they require the precise handling of boundary conditions and adjoint variables across multiple phases, making the convergence to an optimal solution more challenging.

In the field of space applications, previous research has applied similar PINN methods to OCPs. The authors in [15] proposed a novel framework for solving BVPs using PINNs in a purely physics-driven manner. Their approach focuses solely on the residuals of the differential equations governing the BVPs. While this method has proven effective for learning optimal controls in intercept problems, it still needs the solution of the BVP and is not suited to obtain rapid estimation costs for multiple transfers. The present work shifts the focus to these rapid estimations; in this context, the regularization of a DNN using adjoint variables to approximate the value function of an OCP has been successfully employed only by the authors in [12]. Their work, which concerned satellite attitude control without state or control constraints, demonstrated better value function approximations with regularization compared with non-regularized DNNs. In the context of low-thrust trajectories, however, the only prior attempt to use a regularized network for this purpose was made in [13], specifically in an Earth–Venus transfer. The analysis concerned a set of trajectories in a narrow beam around a nominal path. That work focused on retrieving, for real-time guidance, the optimal controls from the DNN as derivatives of the represented value function with respect to the initial states. While the regularized DNN offered improved control approximations, it failed to outperform the non-regularized network in value function approximation. These limitations highlight a significant gap in the space trajectory optimization literature.

The present study aims at extending and improving the application of regularized DNNs to space trajectory optimization. First, the selection of LEO transfers as an application explores the method's capabilities to deal with state constraints (in addition to the obvious practical interest for LEO missions). Also, this study identifies key hyperparameters enabling a regularized DNN to outperform a non-regularized one for low-thrust orbital transfers in a wide range of the state space. As discussed in [13], the benefits of regularization were either limited or not noticeable when estimating the value function. However, the results presented here will demonstrate the effectiveness of the proposed approach and will prove that regularization can improve both the accuracy of value function estimation and the model's ability to be generalized to new data.

The first step in this study consists in building three datasets with different characteristics. In order to speed up the collection of new samples, namely, time-optimal trajectories in LEO between given initial and final states, partial data are used to train non-optimized DNNs. In turn, these coarse DNNs provide warm-started guesses for the unknowns of new transfers. Once the datasets are collected, a thorough optimization of the DNNs' hyperparameters is performed by using a Bayesian optimization approach. Then, an assessment of the use of costates as a means to regularize the DNNs' loss is conducted, and the results are compared to those obtained without regularization. Finally, the method is compared to other state-of-the-art algorithms. Section 2 provides a description of the optimal control problem and the DNN approach. Section 3 describes the collection strategy for the datasets and outlines the fine-tuning process of the DNNs' hyperparameters. The results are presented in Section 4 and are validated in Section 5 considering multitarget missions. Finally, the conclusions are drawn in Section 6.

2. Background

2.1. Optimal Control Problem

In this study, the OCP is formulated following the approach presented in [22]. This section provides a concise summary of that work. The motion of the spacecraft is governed by the perturbed two-body problem equations, accounting for the secular effects of J2 and low-thrust acceleration. The analysis focuses on nearly circular orbits and long-duration transfers, disregarding variations in eccentricity, the argument of periapsis, and mean anomaly. This approximation is deemed suitable for preliminary evaluations in the context of sequence planning. Since the solutions of interest perform a large number of revolutions around the Earth, one can reasonably expect that eccentricity and phase corrections can be spread along the trajectory with small variations in total cost, without remarkably affecting the DNN's estimation accuracy. In the multitarget missions in Section 5, five days are reserved for rendezvous operations and are included in the transfer cost; they could also be used for small orbit corrections. Gauss' planetary equations [23] are employed. The differential equations for the orbital elements are expressed as functions of the orthogonal components of the perturbing acceleration, specifically the thrust-to-mass ratio (T/m), in the radial–tangential–normal reference frame. Edelbaum's approximation [24] assumes a constant angle β between the thrust vector and the orbit plane with a sign switch every half revolution and gives the element changes over one revolution. These values are divided by the time required to complete that revolution in order to approximate the time derivatives of the orbital elements. Consequently, the resulting state equations can be expressed as follows:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = 2\frac{T}{m}\sqrt{\frac{a^3}{\mu}}\cos\beta \tag{1}$$

$$\frac{\mathrm{d}i}{\mathrm{d}t} = \frac{2}{\pi}\frac{T}{m}\sqrt{\frac{a}{\mu}}\sin\beta\cos\vartheta_0 \tag{2}$$

$$\frac{\mathrm{d}\Omega}{\mathrm{d}t} = \frac{2}{\pi}\frac{T}{m}\sqrt{\frac{a}{\mu}}\sin\beta(\sin\vartheta_0/\sin i) + \dot{\Omega}_{J2} \tag{3}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = -\frac{T}{g_0 I_{sp}} = -\frac{T}{c} \tag{4}$$

where a represents the semimajor axis, i denotes the inclination, Ω represents the right ascension of the ascending node (RAAN), g_0 represents the gravitational acceleration on Earth's surface, and I_{sp} is the specific impulse (assumed to be constant). Furthermore, $c = g_0 I_{sp}$ denotes the effective exhaust velocity, and $\dot{\Omega}_{J2}$ represents the secular variation in the RAAN due to the J2 perturbation, given by

$$\dot{\Omega}_{J2} = -\frac{3}{2}J_2\left(\frac{r_E}{a}\right)^2\sqrt{\frac{\mu}{a^3}}\cos i$$

with r_E and μ the Earth's radius and gravitational parameter. The controls are the thrust angle β and the angle ϑ_0, which determines the argument of the latitude ϑ, where the sign switch occurs (at $\vartheta = \vartheta_0 + \pi/2 + k\pi$ for any integer k). To determine the optimal control law, the theory of optimal control is applied. The Hamiltonian takes the following form:

$$\begin{aligned} H &= \frac{2}{\pi}\frac{T}{m}\sqrt{\frac{a}{\mu}}\left(\pi\lambda_a a\cos\beta + \lambda_i \sin\beta\cos\vartheta_0 + \lambda_\Omega \sin\beta\frac{\sin\vartheta_0}{\sin i}\right) \\ &\quad -\lambda_\Omega J_2\frac{3}{2}\left(\frac{r_E}{a}\right)^2\sqrt{\frac{\mu}{a^3}}\cos i - \lambda_m\frac{T}{c} \end{aligned} \tag{5}$$

The Euler–Lagrange equations are

$$\begin{aligned} \frac{\mathrm{d}\lambda_a}{\mathrm{d}t} &= -3\lambda_a a\frac{T}{m}\sqrt{\frac{1}{\mu a}}\cos\beta - \lambda_i\left[\frac{1}{\pi}\frac{T}{m}\sqrt{\frac{1}{\mu a}}\sin\beta\cos\vartheta_0\right] \\ &\quad -\lambda_\Omega\left[\frac{1}{\pi}\frac{T}{m}\sqrt{\frac{1}{\mu a}}\sin\beta(\sin\vartheta_0/\sin i) + J_2\frac{21}{4}\left(\frac{r_E}{a}\right)^2\sqrt{\frac{\mu}{a^5}}\cos i\right] \end{aligned} \tag{6}$$

$$\frac{\mathrm{d}\lambda_i}{\mathrm{d}t} = \lambda_\Omega\left[\frac{2}{\pi}\frac{T}{m}\sqrt{\frac{a}{\mu}}\sin\beta\frac{\sin\vartheta_0}{\sin^2 i}\cos i - J_2\frac{3}{2}\left(\frac{r_E}{a}\right)^2\sqrt{\frac{\mu}{a^3}}\sin i\right] \tag{7}$$

$$\frac{\mathrm{d}\lambda_\Omega}{\mathrm{d}t} = 0 \tag{8}$$

$$\frac{\mathrm{d}\lambda_m}{\mathrm{d}t} = 2\frac{T}{m^2}\sqrt{\frac{a}{\mu}}\left[\lambda_a a\cos\beta + \lambda_i\frac{1}{\pi}\sin\beta\cos\vartheta_0 + \lambda_\Omega\frac{1}{\pi}\sin\beta\frac{\sin\vartheta_0}{\sin i}\right] \tag{9}$$

In agreement with PMP, the optimal controls maximize the Hamiltonian (since the problem is formulated to maximize the opposite of the final time: $J = -t_f$). The algebraic equations for the controls are, therefore,

$$\tan\vartheta_0 = \frac{\lambda_\Omega}{\lambda_i \sin i} \tag{10}$$

$$\tan\beta = \frac{\lambda_i\cos\vartheta_0 + \frac{\lambda_\Omega}{\sin i}\sin\vartheta_0}{\pi a\lambda_a} \tag{11}$$

The Hamiltonian is linear with respect to the thrust magnitude, and bang–bang control is required in agreement with PMP. Introducing the switching function

$$S_F = \frac{2}{\pi}\frac{1}{m}\sqrt{\frac{a}{\mu}}\sqrt{(\pi a\lambda_a)^2 + \lambda_i^2 + \left(\frac{\lambda_\Omega}{\sin\imath}\right)^2} - \frac{\lambda_m}{c}$$

maximum thrust is used when the switching function is positive and null thrust when S_F is negative. The optimal control problem is formulated by specifying the boundary conditions that define the desired transfer and the performance index. The initial orbital elements a_0, $\imath_0$, and Ω_0 are prescribed. The target orbit to be reached at the end of the transfer is characterized by its elements at the initial time $t_0 = 0$, represented by a_T, $\imath_T$, and Ω_{T0}. The time-derivative of the target RAAN $(\dot{\Omega}_{J2})_T$ is a function solely dependent on a_T and i_T. At the final time, the following conditions must be satisfied: $a_f = a_T$, $\imath_f = \imath_T$, and $\Omega_f = \Omega_{T0} + (\dot{\Omega}_{J2})_T t_f$.

The theory of optimal control also provides the boundary conditions for optimality. For a minimum-time trajectory with free final mass, the boundary conditions are given by $H_f - \lambda_\Omega(\dot{\Omega}_{J2})_T = 1$ and $\lambda_{mf} = 0$. From these conditions and from the observation that the mass adjoint derivative is always positive, it follows that S_F is always positive, and the engine operates at maximum thrust for the entire duration of the transfer. The five boundary conditions at t_f determine the five unknowns of the problem: t_f and the initial values of the adjoint variables. In some cases, a decrease in orbital altitude (necessary to

amplify the effect of J2 perturbation on the spacecraft) may lead to transfers that penetrate the atmosphere. A constrained optimization problem must be defined by fixing a minimum altitude $h_{\lim}$ to avoid such occurrence. In such cases, a three-arc structure becomes optimal. The spacecraft follows the optimal control law during the initial and final arcs, from t_0 to t_1 and from t_2 to t_f, respectively. It maintains constant altitude (i.e., $\beta = 90$ deg) during the intermediate arc from t_1 to t_2. The altitude constraint is enforced at point 2, where an additional boundary condition, $a_2 = r_E + h_{\lim}$, is introduced. The boundary conditions for optimality state that the adjoint variables and the Hamiltonian are continuous at t_1 and t_2, except for λ_{a2}, which exhibits a free discontinuity. Based on the continuity of the Hamiltonian at t_1, it follows that $\lambda_{a1-} = 0$ (the subscripts $-$ and + indicate the values just before and after the relevant point where the discontinuity occurs). Similarly, at t_2, $\lambda_{a2+} = 0$. The three boundary conditions on a_2, λ_{a1-}, and λ_{a2+} determine the three additional unknowns: t_1, t_2, and λ_{a2+}. The formulation of both the unconstrained and the constrained problems is summarized in Table 1.

Table 1. Optimization problem formulation.

	Unconstrained	**Constrained**
State equations	Equations (1)–(4)	Equations (1)–(4)
Euler–Lagrange equations	Equations (6)–(9)	Equations (6)–(9)
Boundary conditions	$a_f = a_T$ $i_f = i_T$ $\Omega_f = \Omega_{T0} + (\dot{\Omega}_{J2})_T t_f$ $\lambda_{mf} = 0$ $H_f - \lambda_\Omega (\dot{\Omega}_{J2})_T = 1$	$a_f = a_T$ $i_f = i_T$ $\Omega_f = \Omega_{T0} + (\dot{\Omega}_{J2})_T t_f$ $\lambda_{mf} = 0$ $H_f - \lambda_\Omega (\dot{\Omega}_{J2})_T = 1$ $a_2 = r_E + h_{\lim}$ $\lambda_{a1-} = 0$ $\lambda_{a2+} = 0$
Unknowns	$t_f, \lambda_{a0}, \lambda_{i0}, \lambda_{\Omega 0}, \lambda_{m0}$	$t_1, t_2, t_f, \lambda_{a0}, \lambda_{a2+}, \lambda_{i0}, \lambda_{\Omega 0}, \lambda_{m0}$

The order of the BVPs presented in Table 1 can actually be reduced by one. The condition on the final value of the Hamiltonian is essentially a scaling condition, since the problem is homogeneous in the adjoint variables. It can be replaced by specifying one of the initial values; here, we pose $\lambda_\Omega = \pm 1$. By fixing λ_Ω (which remains constant), the adjoint variables and the Hamiltonian are scaled relative to the solution that satisfies $H_f - \lambda_\Omega (\dot{\Omega}_{J2})_T = 1$. However, as only ratios of adjoint variables appear in Equations (10) and (11), the optimal controls remain unaffected. In the constrained problem, the BVP order is further reduced by one, as the unknown λ_{a2+} is actually specified (equal to 0); the proper value is imposed during integration at the beginning of the third phase. It is important to select the appropriate sign of λ_Ω to avoid solutions with negative time of flight. Its sign is determined by the difference between the perturbed RAAN values of the final and initial orbits, evaluated at the final time of Edelbaum's transfer (which neglects RAAN correction) for the same changes in the semimajor axis and inclination.

2.2. Neural Networks

In the context of the optimal control problem at hand, a minimum transfer time is associated with every possible set of initial state, target state, specification of the propulsion system, and altitude constraint. That is, the minimum transfer time, denoted by t_f, is a function of the initial state (a_0, i_0, $\Delta\Omega_0$, m_0), the target state (a_T, i_T), the parameters of the propulsion system (T, c) and the altitude constraint ($h_{\lim}$). Since fixed values of T, c, and $h_{\lim}$ are used in this analysis, t_f can be expressed mathematically as

$$t_f = g_{t_f}(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T) \tag{12}$$

For any Ω_0 and Ω_{T0}, one can select the reference axis such that $\Omega_0 = 0$ and $\Omega_{T0} = \Delta\Omega_0$, where $\Delta\Omega_0$ is the difference between the target and the initial RAAN at start time. Therefore, the function depends only on the initial RAAN difference. Similarly, for the initial costate λ_{x0} associated with variable x and for the intermediate time t_i, the following relationships hold:

$$\lambda_{x0} = g_{\lambda_{x0}}(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T) \quad x = a, i, \Omega, m \tag{13}$$

$$t_i = g_{t_i}(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T) \quad i = 1, 2 \tag{14}$$

In this analysis, fully connected feed-forward DNNs are designed to interpolate the functions g_{t_f}, $g_{\lambda_{x0}}$, and g_{t_i}. The inputs to each DNN are, therefore, the initial states and target states. All DNNs have a single-output neuron (Figure 1), so the mapping defined by the DNNs has the general form $y = f^*(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T, \boldsymbol{\theta})$, where y is the scalar output and $\boldsymbol{\theta}$ denotes the weights and biases of the DNN.

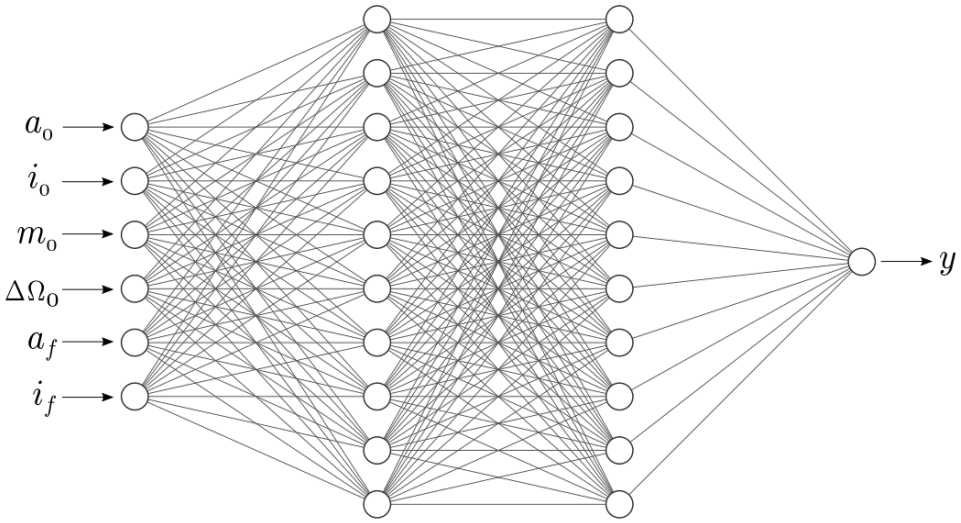

Figure 1. Neural network with two hidden layers. The number of layers and neurons per layer define $\boldsymbol{\theta}$.

2.2.1. Loss Functions

Two types of DNNs are trained to interpolate the minimum transfer time function. The first type, denoted by N_t, utilizes the mean squared error loss. The loss associated with a sample in the dataset is defined as

$$L_t = \left[t_f(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T) - t_f^N(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T, \boldsymbol{\theta})\right]^2 \tag{15}$$

Here, t_f represents the true minimum time found by the indirect method and t_f^N denotes the DNN's prediction. The second type of DNN for minimum-time interpolation, denoted by $N_{t_{reg}}$, utilizes a modified loss function. By the theory of optimal control, the derivative of the objective function with respect to an initial state corresponds to the costate associated with that state. Consequently, by using the initial values of the costates from solved transfers, a regularization term is defined as

$$L_{reg} = \left|\boldsymbol{\lambda}_{\boldsymbol{x_0}\,\mathrm{opt}} - \frac{d}{d\boldsymbol{x_0}} {t_f}^N(a_0, i_0, \Delta\Omega_0, m_0, a_T, i_T, \boldsymbol{\theta})\right|^2 \tag{16}$$

Here, $\boldsymbol{\lambda}_{\boldsymbol{x_0}\,\mathrm{opt}} = \left(\lambda_{a0_{\mathrm{opt}}}\,,\,\lambda_{i0_{\mathrm{opt}}}\,,\,\lambda_{\Omega 0_{\mathrm{opt}}}\,,\,\lambda_{m0_{\mathrm{opt}}}\right)^T$ is the costate vector at the initial time, with the subscript 'opt' denoting that it corresponds to the optimal solution. Moreover, $\boldsymbol{x_0} = (a_0\,,\,i_0\,,\,\Omega_0\,,\,m_0)^T$ is the initial state vector. The modified loss function is defined as

$$L_{t_{reg}} = L_t + rL_{reg} \tag{17}$$

In the above equation, r represents a regularization factor controlling the strength of the regularization term in updating the trainable parameters of the DNN. The regularization term forces the DNN to learn information about the physics of the problem, restricting the domain of possible configurations of $\boldsymbol{\theta}$ that minimize L_t. By leveraging TensorFlow's automatic differentiation capabilities, retrieving the gradient of t_f^N with respect to $\boldsymbol{x_0}$ is computationally efficient, as it only requires the forward pass through the computation graph. Moreover, the gradient is exact.

Additional single-output DNNs, denoted by N_{λ_a}, N_{λ_i}, and N_{λ_m} are used to approximate Equation (13), while N_{t1} and N_{t2} approximate the intermediate times. The mean squared error loss is employed for these five DNNs. Notably, the derivatives of the output of $N_{t_{reg}}$ with respect to the initial states are also used to approximate the costates. In summary, a total of seven DNNs were employed in this work: N_t, $N_{t_{reg}}$, N_{λ_a}, N_{λ_i}, N_{λ_m}, N_{t1}, and N_{t2}. All DNNs were implemented by using Python, with Keras and Tensorflow. In addition, training was performed on NVIDIA GeForce RTX 3070 Ti GPUs and the optimal control problems between the initial, and final states were solved by using a Fortran program that implemented PMP and solved the BVPs, relying on an Intel(R) Core(TM) i5-8400 CPU. The optimal control problems were solved in dimensionless units , and no additional normalization of the data was performed before feeding them to the DNNs.

2.2.2. Learning Rate Schedulers

All DNNs in this paper were trained by using the Adam algorithm [25]. The exponential decay rates for the moment estimates was kept fixed at default values ($\beta_1 = 0.9$ and $\beta_2 = 0.999$), weight decay was not applied, and $\hat{\epsilon}$ was set to 10^{-7}. The stepsize α (i.e., the learning rate upper bound for each parameter in the DNN) was either kept fixed at a given value or updated at the end of every batch according to a scheduler. A total of five schedulers were implemented. The cosine annealing scheduler described in [26] was implemented in two variants. The first one, which will be referred to as *cosine annealing* (CA), set the learning rate η at each batch iteration t according to the following:

$$\begin{cases} \eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{t}{T}\right)\right) & \text{if } t \leq T \\ \eta_t = \eta_{\min} & \text{if } t > T \end{cases} \tag{18}$$

where $\eta_{\min}$ and $\eta_{\max}$ define the range for the learning rate and t is the number of batches processed since the beginning of training. When $t = 0$, $\eta_t = \eta_{\max}$, whereas $\eta_t = \eta_{\min}$ is reached at $t = T$. The learning rate stays constant at $\eta_{\min}$ thereafter. The second variant, which will be referred to as *restarted cosine annealing* (RCA), restarted the learning rate value every time the lower boundary was reached according to the following:

$$\eta_t = \eta_{\min} + \frac{1}{2}\left(\eta_{\max}^i - \eta_{\min}\right)\left(1 + \cos\left(\frac{t}{T_i}\right)\right) \tag{19}$$

Here, $\eta_{\max}^i$ denotes the upper bound of the learning rate during the i-th cycle, t is the number of batches processed since the last restart, and T_i is the i-th cycle's length. At every restart, $\eta_{\max}^i$ is updated according to $\eta_{\max}^i = \gamma_c \eta_{\max}^{i-1}$, where γ_c is a decay rate between 0 and 1. Analogously, the cycle length is updated according to $T_i = k_{\mathrm{mult}} T_{i-1}$, where k_{mult} is a multiplication factor greater than 1.

The other three schedulers, described in [27], adopted a triangular policy. The first triangular scheduler, denoted hereafter as *triangular* 1 (T1), set the learning rate according to the following:

$$\eta_t = \eta_{\min} + (\eta_{\max} - \eta_{\min}) \max\left(0, 1 - \left|2\frac{t}{T} - 2\left\lfloor 1 + \frac{t}{T}\right\rfloor + 1\right|\right) \tag{20}$$

Here, all symbols have the same meaning as in Equations (18) and (19), and $\lfloor x \rfloor$ denotes the floor function of x. The learning rate starts at $\eta_{\min}$ and cyclically varies linearly between $\eta_{\min}$ and $\eta_{\max}$, and T represents the number of batches until the learning rate returns to the initial value. The second triangular scheduler, denoted by *triangular 2* (T2), is a variation of the first one where the maximum learning rate is halved at the end of each cycle:

$$\eta_t = \eta_{\min} + \left(\eta^i_{\max} - \eta_{\min}\right) \max\left(0, 1 - \left|2\frac{t}{T} - 2\left\lfloor 1 + \frac{t}{T}\right\rfloor + 1\right|\right) \tag{21}$$

The upper bound for the learning rate during the i-th is $\eta^i_{\max} = \frac{1}{2}\eta^{i-1}_{\max}$. The last triangular scheduler, denoted by *triangular 3* (T3), is described by

$$\eta_t = \eta_{\min} + \gamma^t_e(\eta_{\max} - \eta_{\min}) \max\left(0, 1 - \left|2\frac{t}{T} - 2\left\lfloor 1 + \frac{t}{T}\right\rfloor + 1\right|\right) \tag{22}$$

Here, γ^t_e is an exponential factor that causes the upper bound to decay after every batch.

3. Methods

3.1. Dataset Collection Strategy

All datasets in this study contain 1 million samples. Each sample in the dataset consists of the initial and final states of the transfer (the input of the DNNs), the optimal time t_f, the intermediate times t_1 and t_2 (only for the constrained transfers), and the costate vector $\boldsymbol{\lambda_{x0}}$. The logic of the strategy adopted to collect samples is illustrated in Figure 2.

At the beginning of dataset collection, the Fortran program randomly selects the costates from a uniform distribution within the range $[-5, 5]$. The transfer time guess is chosen randomly between upper and lower bounds, which are defined with reference to simplified transfers. The lower bound t_{Edel} represents the transfer duration in the absence of RAAN changes, i.e., the original problem treated in Edelbaum's paper [24]. It is a lower bound for the actual transfer, which must, in addition, achieve the required RAAN. The upper bound $t_{\max} = t_{\text{Edel}} + t_{\text{RAAN}}$ is the total duration of the transfer that first changes the semimajor axis and inclination and then uses thrust at constant altitude and inclination to actively achieve the required RAAN change. It is worthwhile to note that these bounds only hold for the initial guess, but the indirect method may actually converge to a minimum-time value outside of this range (even though never below t_{Edel}).

The program first attempts to solve the problem without the altitude constraint. A maximum of 10 random guesses per transfer are attempted. If the problem converges, the trajectory is analyzed. If the converged trajectory falls below the specified altitude limit, the program attempts to solve the constrained transfer. In this case, an initial guess t_1 is selected from a uniform distribution within the range between t_{in} and $1.1t_{\text{in}}$, where t_{in} represents the time at which the spacecraft penetrates the atmosphere in the unconstrained problem. Similarly, t_2 is guessed from a uniform distribution between t_{out} and $1.1t_{\text{out}}$, where t_{out} represents the time at which the satellite rises above the altitude limit in the unconstrained problem.

Since any point on the optimal path between the initial and the target states is actually an initial state for the same optimal controls [20], 10 points along the trajectory of each transfer are sampled and collected in the dataset. This strategy helps obtain a dense dataset, which is crucial for the DNN to learn the underlying function.

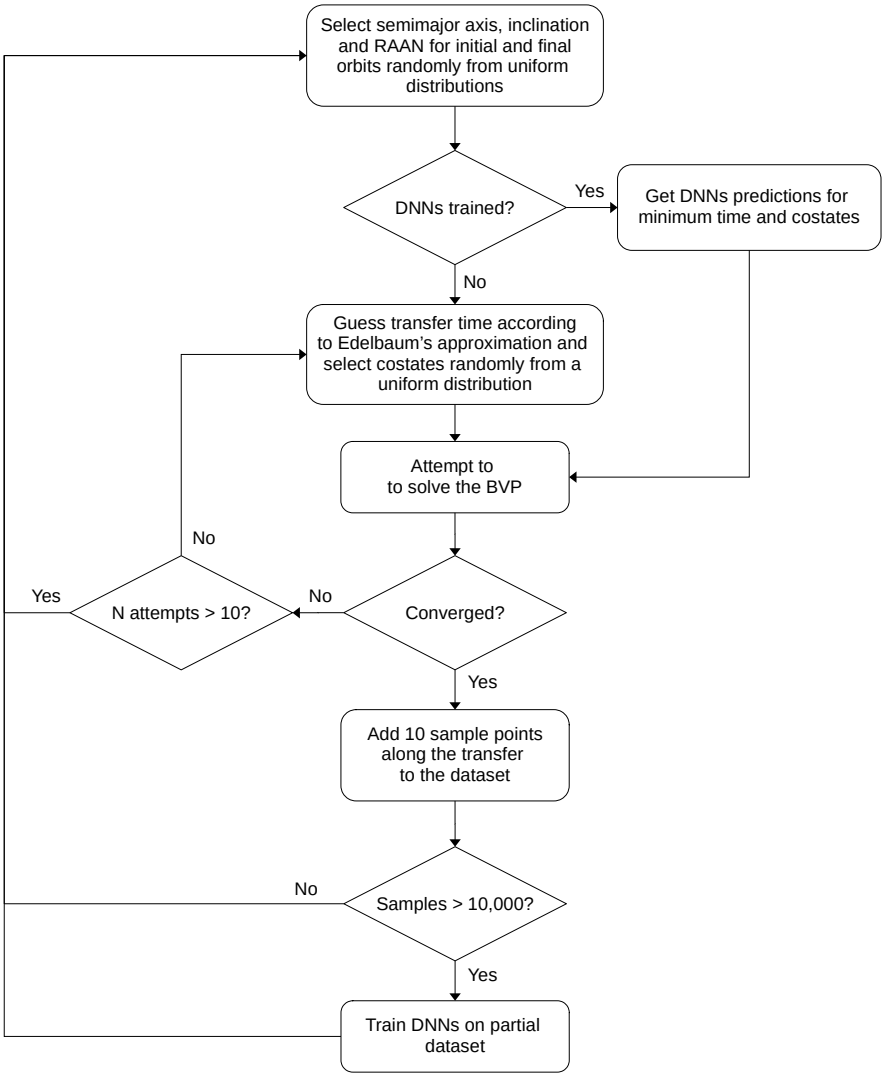

Figure 2. Warm-started guess strategy adopted for the collection of the datasets.

Once 10,000 samples are collected (i.e., 1000 transfers are solved), all seven DNNs are trained in parallel on the partial dataset while collecting more data. The partial dataset is split in three in a 8:1:1 ratio, for training, validation, and testing, respectively. After training, the DNN that best approximate the unknowns are used to provide a warm-started guess for new transfers. As already mentioned, $N_{t_{reg}}$ is employed to also predict $\boldsymbol{\lambda_{x0}}_{\text{opt}}$. Since the Fortran program uses scaled costates by fixing $|\lambda_\Omega| = 1$, the scaled costate vector is given by $\boldsymbol{\lambda_{x0}} = \text{sign}\lambda_\Omega\left(\boldsymbol{\lambda_{x0}}_{\text{opt}}/\lambda_{\Omega\text{opt}}\right)$. The hyperparameters for all DNNs used for the warm-started guesses are fixed a priori to reduce the training time. The DNN architecture consists of 10 hidden layers, 1000 neurons per layer, softplus activation functions for the hidden layers, and linear activation function for the output. The weights are initialized by using the He normal algorithm [28]. The CA scheduler is employed, with $\eta_{\max} = 10^{-4}$, $\eta_{\min} = 10^{-7}$, and $T = 0.9\, b\, \text{n}_{\max}$, where b is the number of batches per epoch and $\text{n}_{\max}$ is the maximum number of epochs for training. In order to train the DNNs in a matter of a few minutes on the available hardware, $\text{n}_{\max}$ is set to 1000, and the batch size is set to 256 samples. The regularization factor r from Equation (17) is set to 10^{-4}, as the gradient of L_t with respect to the DNNs' trainable parameters is observed to be smaller than the one of $L_{t_{reg}}$, on average, by roughly 4 orders of magnitude.

Three different datasets are collected, denoted by $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. For each transfer in the datasets, the initial semimajor axis a_0 is randomly picked from a uniform distribution in the range between 200 km and 2000 km. The same range is used for the target semimajor

axis a_T. Therefore, roughly the entire spectrum of altitudes in LEO is considered. The initial RAAN difference between the final and initial orbit is randomly selected from a uniform distribution in the range between -30 deg and 30 deg. To explore the impact of the initial inclination i_0 on the effectiveness of the DNNs, dataset $\mathcal{D}_1$ encompasses a narrow range of initial inclinations, ranging from 50 to 55 degrees. The second dataset, $\mathcal{D}_2$, covers the range from 45 to 60 degrees, while the third dataset, $\mathcal{D}_3$, covers a broader range, from 40 to 65 degrees. Qualitatively, one can expect that by broadening the initial inclination range, the DNN interpolation of the objective function becomes more challenging. The target inclination is randomly picked from a uniform distribution in the range from $i_0 - 1$ deg to $i_0 + 1$ deg. The maximum available thrust is set to 1 N for all experiments, with a specific impulse of 2500 s. The initial mass for each transfer is randomly selected within a range from 800 kg to 1500 kg. Finally, the altitude limit is set to 200 km.

3.2. Hyperparameters Fine Tuning

After the collection of the datasets, $N_{t_{reg}}$ DNNs are trained with fine-tuned hyperparameters. The hyperparameter search is performed with the partial dataset containing 10^4 samples by using KerasTuner [29]. The *Bayesian Optimization tuner* is employed, which uses Bayesian optimization with the Gaussian process to choose new hyperparameter values by computing a distribution of the objective function based on the models already trained. The search space grows combinatorially with the number of hyperparameter choices and each trial involves training a new model. In order to minimize the search time, only a few crucial hyperparameters are optimized. In particular, the hyperparameters are divided into two groups: those that define the architecture of the DNNs and those related to the learning rate scheduler. A two-step tuning strategy is implemented. The first step involves fixing the scheduler hyperparameters and tuning the architecture ones. Then, the scheduler hyperparameters are tuned for the best three architectures. For both steps, the batch size is set to 128 samples. To reduce the impact of the initialization of $\boldsymbol{\theta}$, every trial is performed three times, and the results are averaged. The number of trials test for architecture tuning is 100. The same figure is used to tune the three best architectures, giving a total of 400 trials to test per model.

During the first step (Table 2), the Bayesian optimizer is tasked with finding the best combination of the number of hidden layers, the number of units per layer and the type of activation function for the hidden units. The number of hidden layers can range from 1 up to 20, and the number of units for layer ranges from 100 to 2000 with a discrete step size of 100. The activation function choices are Softplus, ReLU, and Exponential Linear Unit (ELU). The regularization factor r is also optimized during this step, ranging from 10^{-1} to 10^{-5} with a logarithmic step size of 10. The CA scheduler is employed, with $\eta_{\max} = 10^{-4}$, $\eta_{\min} = 10^{-7}$, and T = 128,000 (corresponding to 1000 epochs). The maximum number of epochs per trial is generously set to 20,000 with an early stopping policy that tracks the best epoch and interrupts training if no improvement occurs after 500 epochs.

Table 2. Architecture hyperparameter search space.

	Min	**Max**	**Step**	**Choices**
N layers	1	20	1	-
Units	100	2000	100	-
Activation	-	-	-	Softplus, ReLU, and ELU
Regularization factor	10^{-5}	10^{-1}	10 log	-

For each of the three best architectures, the Bayesian optimizer is then tasked to search for the best scheduler. When a scheduler is selected, its hyperparameters are searched according to Table 3. For the schedulers CA, RCA, T1, T2, and T3, $\eta_{\min}$ and $\eta_{\max}$ are sampled in the range between 10^{-3} and 10^{-9} with a logarithmic step size of 10. For CA, the

search space for T is between 1000 and 10,000 epochs, with a step size of 1000 epochs. For RCA, T1, T2, and T3, the hyperparameter T is picked in the range from 100 to 1000 epochs, with a step size of 100 epochs. The search spaces for γ_c and k_{mult} are in the ranges from 0.1 to 1 and from 1 to 5, respectively. For T3, the range between 0.1 and 1 is searched for γ_e.

Table 3. Scheduler hyperparameter search space.

	Min	Max	Step	Scheduler Type
η	10^{-7}	10^{-3}	10 log	Constant
η_{min}	10^{-9}	10^{-3}	10 log	CA, RCA, T1,T2,T3
η_{max}	10^{-9}	10^{-3}	10 log	CA, RCA, T1,T2,T3
T_{epochs}	1000	10,000	1000	CA
	100	1000	100	RCA, T1, T2, T3
γ_c	0.1	1	-	RCA
k_{mult}	1	5	-	RCA
γ_e	0.1	1	-	T3

In addition to the learning rate schedulers, the Bayesian optimizer is also given the choice of keeping the learning rate fixed, in the range between 10^{-3} and 10^{-7}. The maximum number of epochs per trial is set to 20,000. For trials with fixed learning rate, the patience of the early stopping policy is set to 500 epochs. For the CA scheduler, the patience is set to $T/4$ epochs, and for the other schedulers, the patience is set to $2T$ epochs, to allow for 2 cycles of non-improvement before stopping.

4. Results

4.1. Dataset Collection

The implementation of the warm-started guess strategy shows a significant improvement in the speed of the collection of the three datasets, as shown in Table 4.

Table 4. Dataset collection statistics.

	$\mathcal{D}_1$		$\mathcal{D}_2$		$\mathcal{D}_3$	
	Partial	**Complete**	**Partial**	**Complete**	**Partial**	**Complete**
Samples	10^4	10^6	10^4	10^6	10^4	10^6
Tested transfers	1022	100,266	1024	100,224	1022	100,264
% converged	97.85	99.74	97.66	99.78	97.85	99.74
Average guesses	1.681	1.041	1.623	1.042	1.691	1.041
Collection time	4 m	59 m	4 m	58 m	4 m	59 m
Average time per converged transfer	0.198 s	0.035 s	0.194 s	0.035 s	0.193 s	0.035 s
% warm-started converged	-	99.62	-	99.69	-	99.65

The results are comparable and consistent for all datasets: on average, 1.7 guesses for a transfer to converge with random guesses. With the help of the DNNs, this figure drops very close to 1, as on average, only 3 in 1000 warm-started transfers do not converge. The average convergence time also improves by a factor of almost 6. Notably (Table 5), these results are obtained by using the $N_{t_{reg}}$ DNN to estimate both t_f and $\boldsymbol{\lambda_{x0}}$, demonstrating superiority over N_t, N_{λ_a}, N_{λ_i}, and N_{λ_m} across all datasets (with the sole exception of N_{λ_a} performing better on D_3).

This finding is intriguing, as it indicates that $N_{t_{reg}}$ has better generalization capabilities, not only for t_f but also for the costates. The $N_{t_{reg}}$ DNN outputs only t_f but also uses its derivatives with respect to the DNN inputs that represent the initial state. This additional

information allows for a better approximation of the costates for unseen scenarios in comparison to DNNs specifically trained to solely predict a single costate. This is not entirely obvious a priori. The difference between N_t and $N_{t_{reg}}$ is that N_t can adjust its weights and biases freely to approximate t_f as accurately as possible. In contrast, $N_{t_{reg}}$ is constrained by the requirement that the derivatives of t_f with respect to the initial state must approximate the costates. This fact limits the weights and biases of the DNN to a smaller set of configurations. Therefore, N_t tends to overfit the training dataset and its performance is inferior in unseen scenarios. However, it is not obvious that for $N_{t_{reg}}$ the derivatives of t_f would be approximated so well that they are even better than those of DNNs trained to approximate only one specific costate. This could strongly indicate that the map learned by $N_{t_{reg}}$ closely replicates the real behavior.

The results on the test split of the partial datasets for $N_{t_{reg}}$, in terms of root mean squared error (RMSE) and mean absolute error (MAE), are shown in Table 6. As expected, the results on $\mathcal{D}_1$ show the best performance, while those on $\mathcal{D}_2$ and $\mathcal{D}_3$ are relatively comparable. These results are quite promising, with errors of just a few hours of transfer time compared with an average optimal transfer time of approximately 10 days and a standard deviation of around 8 days across all test splits.

Table 5. Test RMSE on partial datasets of costates predictions. Comparison of $N_{t_{reg}}$ and the dedicated DNNs.

		$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$
λ_{a0}	$N_{t_{reg}}$	0.859	0.241	3.189
	N_{λ_a}	1.320	0.562	2.089
λ_{i0}	$N_{t_{reg}}$	0.520	1.106	1.183
	N_{λ_i}	0.895	1.524	1.239
λ_{m0}	$N_{t_{reg}}$	0.071	0.032	0.089
	N_{λ_m}	0.105	0.055	0.130

Table 6. $N_{t_{reg}}$ test results on partial datasets.

		$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$
t_f	RMSE	2 h 22 m	4 h 56 m	4 h 06 m
	MAE	1 h 23 m	2 h 11 m	1 h 51 m

4.2. Hyperparameter Search

The $N_{t_{reg}}$ DNN with hyperparameters fine-tuned on the 10,000-samples dataset also achieves better results in estimating t_f and $\boldsymbol{\lambda_{x0}}$ than any of the other fine-tuned DNNs. Table 7 showcases the comparison between $N_{t_{reg}}$ and N_t for the results obtained from the test sets of $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. Interestingly, there is no clear architectural pattern in the fine-tuned $N_{t_{reg}}$ models across these datasets. Notably, $N_{t_{reg}}$ models trained on $\mathcal{D}_1$ and $\mathcal{D}_2$ tend to have considerably deeper architectures compared with the model trained on $\mathcal{D}_3$. This indicates that the optimal DNN depth may vary depending on the characteristics of the data. Nevertheless, despite architectural differences, a consistent trend emerges regarding the choice of activation functions for the hidden layers. In fact, the ELU activation function consistently stands out as the best performer across all datasets. This choice of hidden activations aligns with the unique characteristic of $N_{t_{reg}}$, as it is regularized with the gradient of t_f with respect to $\boldsymbol{x_0}$. ELU, known for its smoothness, differentiability, and adaptability, appears to be particularly well suited for this task. Its smooth transition across positive and negative inputs, along with robustness against issues like vanishing gradients, makes it a strong candidate for DNNs subjected to such regularization. When these factors are considered, it is plausible to speculate that the superior performance of ELU in the

context of $N_{t_{reg}}$ may be attributed, at least in part, to its compatibility with the gradient-based regularization applied to this DNN. Further investigations into the interplay between ELU activations and gradient-based regularization could provide valuable insights into optimizing similar DNN architectures for control problems.

Table 7. Results on partial test sets after hyperparameter search.

	$\mathcal{D}_1$		$\mathcal{D}_2$		$\mathcal{D}_3$	
	$N_{t_{reg}}$	N_t	$N_{t_{reg}}$	N_t	$N_{t_{reg}}$	N_t
t_f RMSE	1 h 51 m	3 h 05 m	2 h 47 m	5 h 16 m	1 h 52 m	4 h 42 m
t_f MAE	50 m	1 h 50 m	1 h 38 m	3 h 01 m	57 m	2 h 15 m
λ_a RMSE	0.077	2.532	0.235	20.0	0.077	4.93
λ_i RMSE	0.126	0.827	0.285	4.05	0.130	4.80
λ_m RMSE	0.020	0.141	0.018	1.75	0.018	1.15
Units per layer	300	100	500	1000	600	600
Layers	12	14	15	5	7	10
Hidden activations	ELU	ELU	ELU	softplus	ELU	softplus
Output activation	linear	linear	linear	linear	linear	linear
Regularization factor	10^{-3}	-	10^{-2}	-	10^{-2}	-
Scheduler	RCA	T	CA	CA	RCA	RCA
$\eta_{\min}$	10^{-6}	10^{-6}	10^{-8}	10^{-7}	10^{-5}	10^{-4}
$\eta_{\max}$	10^{-4}	10^{-4}	10^{-5}	10^{-5}	10^{-4}	10^{-3}
T_{epochs}	400	500	5000	7000	400	500
γ_c	0.2	-	-	-	0.1	0.2
k_{mult}	1.6	-	-	-	1.2	2.1

4.3. Full Dataset Test Results

Since no clear pattern emerges from the partial test sets, $N_{t_{reg}}$ is trained on the full datasets with all three types of hyperparameter sets, $\mathcal{H}_1$, $\mathcal{H}_2$, and $\mathcal{H}_3$, obtained from the searches on the partial $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$, respectively. Table 8 summarizes the test results. Training is repeated three times for all hyperparameter sets, and the results shown are the averages over the three trials. The results are remarkable for all DNNs, as the minimum transfer time is predicted with incredible accuracy. Notably, a hyperparameter set always performs slightly better on the same full dataset when compared with the other hyperparameter sets. Therefore, it is safe to assume that the best architecture and learning parameters depend largely on the specific characteristics of the data. As for the initial costate prediction, all hyperparameter sets perform very similarly and demonstrate an excellent generalization capability.

Table 8. Test results on full datasets.

Dataset	$\mathcal{D}_1$			$\mathcal{D}_2$			$\mathcal{D}_3$		
Hyperparameters	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$	$\mathcal{H}_1$	$\mathcal{H}_2$	$\mathcal{H}_3$
t_f RMSE	27 m	47 m	33 m	46 m	38 m	42 m	46 m	52 m	40 m
t_f MAE	17 m	32 m	21 m	30 m	24 m	26 m	27 m	32 m	25 m
λ_a RMSE	0.23	0.27	0.24	0.21	0.12	0.29	0.33	0.20	0.4
λ_i RMSE	0.37	0.42	0.42	0.22	0.17	0.33	0.40	0.33	0.62
λ_m RMSE	0.02	0.02	0.03	0.02	0.01	0.04	0.03	0.01	0.07

5. Validation

The fine-tuned $N_{t_{reg}}$ DNN trained on $\mathcal{D}_1$ is compared with traditional state-of-the-art ML algorithms (namely, bagging, random forest, decision tree, and extra tree) to assess the performance of the DNNs and the overall methodology. All algorithms are trained on the same dataset. The hyperparameters are fine-tuned according to Table 9 by using random search with 5-fold cross-validation, where the training and validation datasets used to train the DNN are combined. The scikit-learn library [30] is used to implement the algorithms and perform hyperparameter tuning. Each algorithm is allowed a maximum of 100 search iterations. The results of the fine-tuned models on the test dataset are presented in Table 10, evaluated by using RMSE, MAE, and mean relative error (MRE). MRE is calculated as

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|t_{fp} - t_f\right|}{t_f}$$

where t_f is the true minimum-time value, t_{fp} is the predicted value, and n is the number of test samples. The results clearly demonstrate the superior performance of the DNN, which is capable of predicting optimal minimum-times with remarkable accuracy. Although all ML algorithms perform reasonably well, high accuracy is crucial when these models are integrated into global optimization loops, as transfer time prediction errors can accumulate across multiple transfers.

Table 9. Hyperparameter search space for the ML algorithms.

Algorithm	Hyperparameter Space
Decision tree	splitter: [best, random]; max_depth: [2, 100, None]; min_samples_split: [2, 10]; min_samples_leaf: [1, 4]
Extra tree	splitter: [best, random]; max_depth: [2, 100, None]; min_samples_split: [2, 10]; min_samples_leaf: [1, 4]
Random forest	n estimators : [10, 100]; max_depth: [2, 100, None]; min_samples_split: [2, 10]; min_samples_leaf: [1, 4], bootstrap: [True, False]
Bagging	n estimators : [10, 100]; max_samples: [0.2, 0.8]; bootstrap: [True, False]

Table 10. Test result comparison of the fine-tuned models on $\mathcal{D}_1$.

Model	RMSE [Hours]	MAE [Hours]	MRE [%]
DNN	0.45	0.28	0.21
Decision tree	19.16	9.86	5.70
Extra tree	19.13	9.85	5.69
Random forest	11.37	5.46	3.65
Bagging	11.58	5.53	3.76

To further validate these observations, the DNN is applied to a multitarget on-orbit servicing (OOS) mission, where the goal is to minimize the total time required to service 50 satellites (this large value is chosen to highlight the method's capabilities). The servicer has the same parameters as the spacecraft described in Section 3.1. The target spacecraft states are randomly generated, with altitudes uniformly sampled between 600 km and 1000 km, inclinations between 50 deg and 55 deg, and initial RAANs uniformly distributed between 0 deg and 360 deg. The objective is to service all spacecraft in the shortest possible time. The transfer times between targets are time-optimal low-thrust transfers, with the costs estimated by the DNN. A fixed service time of 5 days is added after each transfer to account for rendezvous and servicing operations.

The total number of possible sequences is 50!, which is in the order of 10^{64}. A beam search algorithm with beam width set to 5000 is used to explore this large solution space. For the sake of comparison, the same beam search analysis is also carried out by replacing the regularized DNN with the other state-of-the-art ML algorithms. After the beam search, the best sequence found by each method is optimized by using the indirect method to find the true total mission time, which is compared to the estimated time. The results for the sequence found with the DNN are presented in Table 11.

Table 11. The best sequence of the minimum-time multitarget mission.

True Total Time [Days]	Estimated Time [Days]	Error
524.74	525.03	0.14%

The total mission time error is below 0.2%, which is an impressive result considering that each sequence consists of 50 low-thrust transfers. Figure 3 compares the performance of the DNN with other ML algorithms on this sequence. The DNN shows a strong ability to estimate transfer times across all 50 legs, producing results that closely align with those obtained by using the indirect method. This high level of accuracy demonstrates that the DNN can reliably approximate transfers, making it a robust and effective surrogate for the indirect method. The other ML algorithms also perform reasonably well in estimating the single transfers that make up the sequence, achieving comparable estimates to those of the DNN. However, slight variations are observed in certain legs, leading to marginal differences in their accuracy. Nonetheless, the overall cost of the sequence is estimated with good accuracy by all methods, with a tendency to underestimating the costs.

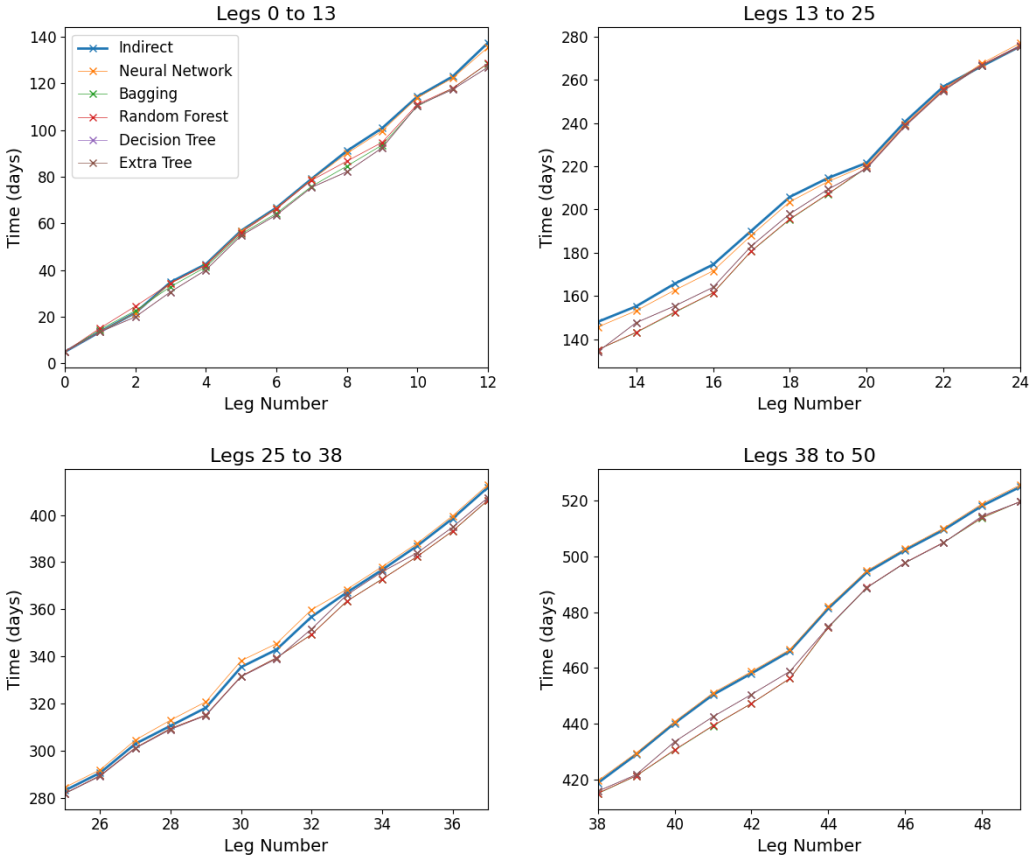

Figure 3. Performance comparison between the DNN and the other ML algorithms on the sequence identified by the DNN during the beam search for the OOS mission.

The beam search finds the best sequences with an apparently lower time of flight when other ML methods are used to estimate the transfer times. However, verification with the indirect method shows that these trajectories are actually unfeasible. As an example, Figure 4 shows the sequence found by using the random forest algorithm, which represents the best alternative ML method tested in terms of accuracy (similar results are found with the other algorithms). While the initial transfers are well approximated, random forest begins to significantly underestimate the transfer time by the fifth transfer. This error accumulates, and by the 15th transfer, the true solution shows that actual duration and propellant consumption are four times larger than the random forest estimation. At this point, 75% of the spacecraft initial mass has already been consumed. Additionally, even the DNN estimate begins to diverge slightly, as the sequence ventures outside the DNN's training domain for mass (and thus thrust acceleration). By the 24th transfer, the sequence found by the random forest model would actually deplete the entire mass of the spacecraft if used as fuel, making mission completion impossible.

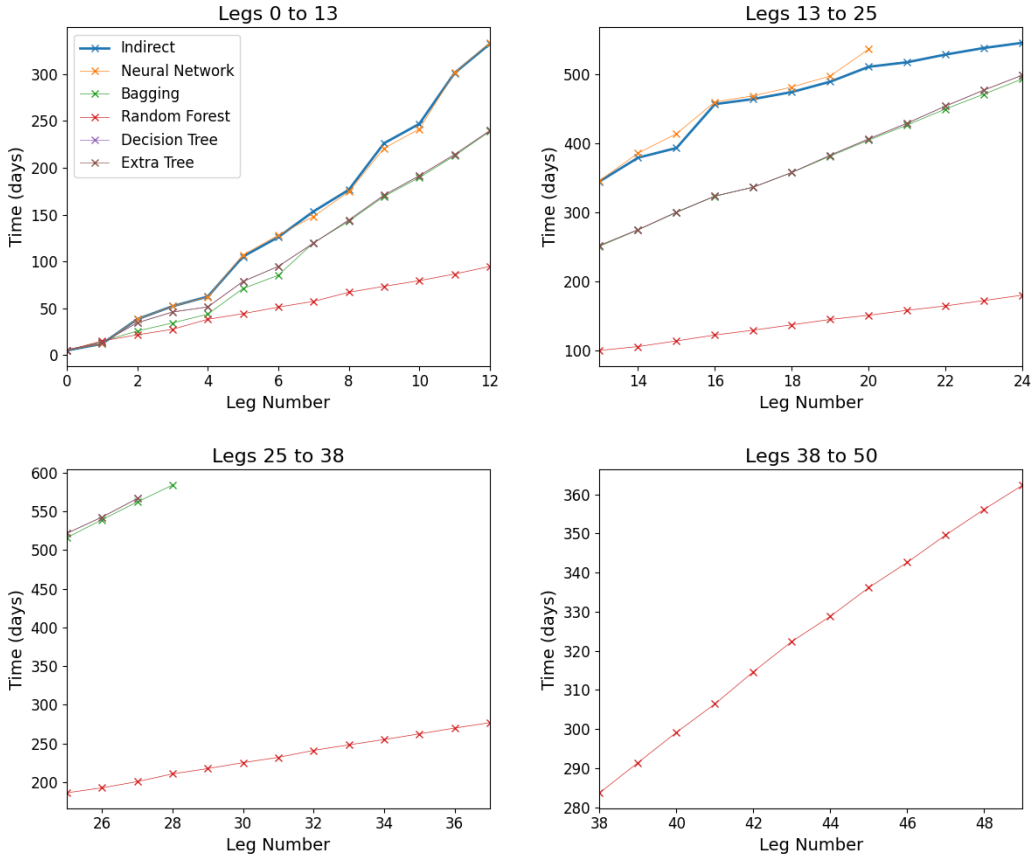

Figure 4. Performance comparison between the DNN and the other ML algorithms on the sequence identified by random forest during the beam search for the OOS mission.

This pattern of divergence is consistent across all alternative ML algorithms. Typically, these algorithms handle short transfers fairly well and approximate sequences accurately for such cases. However, for longer transfers, they occasionally misestimate to the extent that a long transfer is estimated as short. As a result, the beam search saves sequences that are based on poorly estimated long transfers, leading to significant cumulative errors. For the DNN, this behavior is not observed: it consistently provides robust estimates across all transfer lengths, ensuring feasible mission sequences.

The consistency and performance of the DNN are further validated through additional beam searches involving shorter missions. A total of 100 sets, consisting of 10 satellites with random elements, are defined, and the optimal 10-leg sequence is sought for each set. The beam width is reduced to 1000. For each trial, the altitudes are uniformly sampled

between 200 km and 2000 km (in a range broader than the 50-leg mission), and the initial inclinations and RAANs are uniformly distributed between 50 deg and 55 deg and 0 deg and 360 deg, respectively. The results of these trials (Table 12) reaffirm the DNN's accuracy and reliability. The true mission time (mean value of 237 days) shows large variations and ranges from 185 to 295 days, depending on the random elements of the satellites in each set. Notwithstanding, the DNN's absolute error averages just 1.13 days, with a standard deviation of 1.00 day. The maximum error is below 6 days. The low variance in error demonstrates that the DNN not only provides precise transfer time estimates but does so reliably across different mission conditions. Furthermore, the results highlight the DNN's ability to generalize well, even in more constrained mission settings. Shorter sequences imply more demanding maneuvers and longer leg durations, as targets are few and far apart. Despite the reduced beam width, which could increase the likelihood of suboptimal sequences being selected, the DNN maintains high levels of accuracy and consistently estimates feasible mission solutions.

Table 12. Statistical data on 100 trials for 10-leg sequences.

	Mean	Max	Min	Standard Deviation
True total time [days]	237.82	295.74	184.89	20.81
DNN absolute error [days]	1.13	5.89	0.03	1.00

To better evaluate the performance of the proposed methodology, the computation time of the DNN framework is also compared to the estimated computation time of the same beam search procedure if the indirect method were used to calculate transfer costs at each step. On average, the indirect method needs approximately 0.2 s to converge to a solution. Given that the beam search requires evaluating 5,760,050 transfers, this results in an estimated total computation time of about 13 days. In comparison, collecting the dataset, tuning the hyperparameters, and training the DNN takes around 2 days, while the beam search procedure using the DNN is completed in just 1 h. This demonstrates that the proposed method significantly reduces optimization time while maintaining accuracy.

However, the suitability of the DNN method depends on the specific case. In scenarios where only a few transfer evaluations are required, generating the dataset and training the network can be more computationally expensive than solving the transfers directly with the indirect method. Therefore, the proposed approach is particularly advantageous for large-scale problems requiring numerous evaluations, while for smaller problems, traditional methods may be more efficient.

6. Conclusions

This study explored the application of DNNs to predict minimum transfer times for LEO transfers in the context of global optimization and frequent evaluation of transfer costs. The handling of state constraints and the use of the PINN framework are introduced. The results highlight that using costates to regularize the loss during training significantly enhances the DNN's accuracy, even with limited datasets. Specifically, DNNs with ELU activation functions in their hidden layers demonstrate exceptional performance when combined with this regularization approach. Further exploration is needed to understand the role of DNN depth, as this study did not reveal clear patterns in this regard.

Training the regularized DNNs on datasets comprising one million samples achieved impressively accurate results in estimating transfer times. A warm-started guess strategy, which involves using simpler DNNs to predict transfer times and costates for new transfers, greatly expedites the process of collecting training datasets. This approach proves highly practical for real-world applications, particularly in LEO low-thrust space missions. The indirect optimization method can deal with both minimum-fuel and minimum-time trajectories. In this article, only the minimum-time case is treated in order to maintain the focus on the DNN's approximation capabilities. The same approach adopted here could be

used to train the DNN on datasets of minimum-fuel solutions with different durations; no changes in accuracy should be expected, since the two problems are substantially equivalent and only differ in two boundary conditions at the final time. However, in this case, the optimization of long sequences becomes a daunting task for the beam search (rather than for the DNN), as the length of each leg becomes an additional unknown. The analysis of minimum-fuel sequences will be the subject of future work.

The comparison of the DNN model with other state-of-the-art ML algorithms shows that the use of costates for regularization significantly improves prediction accuracy. Validation using beam search to optimize a sequence of transfer of a multitarget OOS mission shows excellent accuracy, with the DNN achieving errors in total mission time below 0.2%. This is notable, given that each sequence involves 50 low-thrust transfers. In contrast, the other ML algorithms show significant errors, leading to impractically high propellant requirements.

Finally, the comparison of computation time of the beam search procedure using either the DNN framework or the indirect method to calculate the transfer costs shows the enormous time savings offered by the DNN approach. While the indirect method is estimated to take around 13 days to evaluate all transfers, the proposed DNN approach reduces the optimization time to just 2 days, demonstrating substantial efficiency improvements. This efficiency, coupled with high accuracy, underscores the practical advantages of the proposed method, especially for scenarios requiring extensive transfer evaluations.

Overall, the methodology proves highly effective for LEO low-thrust missions and holds significant potential for global trajectory optimization, where it can provide rapid and accurate predictions of minimum transfer costs.

Author Contributions: A.F. and L.C. contributed equally to the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research study received no external funding.

Data Availability Statement: Dataset available upon request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lev, D.; Myers, R.M.; Lemmer, K.M.; Kolbeck, J.; Koizumi, H.; Polzin, K. The technological and commercial expansion of electric propulsion. *Acta Astronaut.* **2019**, *159*, 213–227. [CrossRef]
2. Taheri, E.; Kolmanovsky, I.; Atkins, E. Enhanced Smoothing Technique for Indirect Optimization of Minimum-Fuel Low-Thrust Trajectories. *J. Guid. Control Dyn.* **2016**, 39, 2500–2511. [CrossRef]
3. Mall, K.; Grant, M.J.; Taheri, E. Uniform Trigonometrization Method for Optimal Control Problems with Control and State Constraints. *J. Spacecr. Rocket.* **2020**, 57, 995–1007. [CrossRef]
4. Mall, K.; Nolan, S.M.; Levin, W.C.; Risany, L.; DeLaurentis, D.A. Using Uniform Trigonometrization Method for Aviation Based Optimal Control Problems. In Proceedings of the AIAA AVIATION 2023 Forum, San Diego, CA, USA, 12–16 June 2023. [CrossRef]
5. Izzo, D.; Märtens, M.; Pan, B. A survey on artificial intelligence trends in spacecraft guidance dynamics and control. *Astrodyn* **2019**, *3*, 287–299. [CrossRef]
6. Shirobokov, M.; Trofimov, S.; Ovchinnikov, M. Survey of machine learning techniques in spacecraft control design. *Acta Astronaut.* **2021**, *186*, 87–97. [CrossRef]
7. Rubinsztejn, A.; Sood, R.; Laipert, F.E. Neural network optimal control in astrodynamics: Application to the missed thrust problem. *Acta Astronaut.* **2020**, *176*, 192–203. [CrossRef]
8. Shi, J.; Wang, J.; Su, L.; Ma, Z.; Chen, H. A Neural Network Warm-Started Indirect Trajectory Optimization Method. *Aerospace* **2022**, *9*, 435. [CrossRef]
9. Izzo, D.; Origer, S. Neural representation of a time optimal, constant acceleration rendezvous. *Acta Astronaut.* **2023**, *204*, 510–517. [CrossRef]
10. Mughal, A.H.; Chadalavada, P.; Munir, A.; Dutta, A.; Qureshi, M.A. Design of deep neural networks for transfer time prediction of spacecraft electric orbit-raising. *Intell. Syst. Appl.* **2022**, *15*, 200092. [CrossRef]
11. Li, H.; Baoyin, H.; Topputo, F. Neural Networks in Time-Optimal Low-Thrust Interplanetary Transfers. *IEEE Access* **2019**, *7*, 156413–156419. [CrossRef]
12. Nakamura-Zimmerer, T.; Gong, Q.; Kang, W. Adaptive Deep Learning for High-Dimensional Hamilton–Jacobi–Bellman Equations. *SIAM J. Sci. Comput.* **2021**, *43*, A1221–A1247. [CrossRef]

13. Izzo, D.; Öztürk, E. Real-Time Guidance for Low-Thrust Transfers Using Deep Neural Networks. *J. Guid. Control. Dyn.* **2021**, *44*, 315–317. [CrossRef]
14. Effati, S.; Pakdaman, M. Optimal control problem via neural networks. *Neural Comput. Appl.* **2013**, *23*, 2093–2100. [CrossRef]
15. D'Ambrosio, A.; Schiassi, E.; Curti, F.; Furfaro, R. Pontryagin Neural Networks with Functional Interpolation for Optimal Intercept Problems. *Mathematics* **2021**, *9*, 996. [CrossRef]
16. Huang, A.; Wu, S. Neural Network-Based Approximation Model for Perturbed Orbit Rendezvous. *Mathematics* **2022**, *10*, 2489. [CrossRef]
17. Li, H.; Chen, S.; Izzo, D.; Baoyin, H. Deep networks as approximators of optimal low-thrust and multi-impulse cost in multitarget missions. *Acta Astronaut.* **2020**, *166*, 469–81. [CrossRef]
18. Guo, X.; Ren, D.; Wu, D.; Jiang, F. DNN estimation of low-thrust transfer time: Focusing on fast transfers in multi-asteroid rendezvous missions. *Acta Astronaut.* **2023**, *204*, 518–30. [CrossRef]
19. Song, Y.; Gong, S. Solar-sail trajectory design for multiple near-Earth asteroid exploration based on deep neural networks. *Aerosp. Sci. Technol.* **2019**, *91*, 28–40. [CrossRef]
20. Bryson, A.E.; Ho, Y.-C. *Applied Optimal Control*, rev. ed.; Hemisphere: Washington, DC, USA, 1975; pp. 42–89.
21. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
22. Casalino, L.; Forestieri, A. Approximate optimal LEO transfers with J2 perturbation and dragsail. *Acta Astronaut.* **2022**, *192*, 379–389. [CrossRef]
23. Cornelisse J.W.; Schöyer H.F.R.; Wakker K.F. *Rocket Propulsion and Spaceflight Dynamics*, 1st ed.; Pitman: London, UK, 1979; pp. 407–435.
24. Edelbaum, T.N. Propulsion Requirements for Controllable Satellites. *ARS J.* **1961**, *31*, 1079–1089. [CrossRef]
25. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015 .
26. Loshchilov, I.; Hutter, F. SGDR: Stochastic Gradient Descent with Warm Restarts. In Proceedings of International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
27. Smith, L.N.; Cyclical Learning Rates for Training Neural Networks. In Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision, Santa Rosa, CA, USA, 24–31 March 2017. [CrossRef]
28. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the 2015 IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015. [CrossRef]
29. KerasTuner. Available online: https://github.com/keras-team/keras-tuner (accessed on 23 June 2023)
30. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.