2023

Investigating Shallow Neural Networks for Orbit Propagation Deployed on Spaceflight-Like Hardware

Hunter Quebedeaux
University of Central Florida

INVESTIGATING SHALLOW NEURAL NETWORKS FOR ORBIT PROPAGATION DEPLOYED ON SPACEFLIGHT-LIKE HARDWARE

by

HUNTER QUEBEDEAUX
B.S. University of Central Florida, 2021

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Mechanical and Aerospace Engineering
in the College of Engineering and Computer Science
at the University of Central Florida

Fall Term
2023

Major Professor: Tarek Elgohary

ABSTRACT

Orbit propagation is the backbone of many problems in the space domain, such as uncertainty quantification, trajectory optimization, and guidance, navigation, and control of on orbit vehicles. Many of these techniques can rely on millions of orbit propagations, slowing computation, especially evident on low-powered satellite hardware. Past research has relied on the use of lookup tables or data streaming to enable on orbit solutions. These solutions prove inaccurate or ineffective when communication is interrupted. In this work, we introduce the use of physics-informed neural networks (PINNs) for orbit propagation to achieve fast and accurate on-board solutions, accelerated by GPU hardware solutions now available in satellite hardware. Physics-informed neural networks leverage the governing equations of motion in network training, allowing the network to optimize around the physical constraints of the system. This work leverages the use of unsupervised learning and introduces the concept of fundamental integrals of orbits to train PINNs to solve orbit problems with no knowledge of the true solution. Numerical experiments are conducted for both Earth orbits and cislunar space, being the first time a neural network integrator is implemented on flight-like hardware. The results show that the use of PINNs can decrease solution evaluation time by several order of magnitude while retaining accurate solutions to the perturbed two-body problem and the circular restricted three-body problem for deployment on spaceflight-like hardware. Implementation of these neural networks aim to reduce computational time to allow for real-time evaluation of complex algorithms on-board space vehicles.

For Madison, for everything.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Elgohary for his invaluable insight and knowledge throughout our time working together. As well, I thank my cohorts in the Astrodynamics Space and Robotics Lab for many fruitful discussions and their great friendships. I'd like to thank the wonderful scientists and engineers at Lockheed Martin for their support, namely Dr. Bennie Lewis and Dr. Andrew Coats. Finally to my family; I owe every achievement to you. To my parents Charles and Norlita, thank you for your unconditional support. I'm especially thankful for my sister Ashley who has helped me become the person I am today.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Rising commercialization of space exploration has increased the amount of resident space objects (RSOs) in Earth orbit. With the ever growing population or active and inactive vehicles, complex situations arise with enabling active vehicles algorithms like guidance, navigation, and control, or uncertainty quantification of the inactive satellites. On-board spacecraft line of sight measurement models is one method for knowing where an RSO may be, however during loss of vision or communication, estimations based on a-priori data will be needed in order to determine the probability of states. Major issues in the space industry revolve around understanding how space debris will affect future launches. Ground based sensors like Lockheed Martin's FireOPAL can provide estimations for where an RSOs is in orbit. FireOPAL is a wide field of view sensor that can track more than one hundred objects in all orbit regimes; however RSOs are not always continuously trackable or have active tracking devices [1]. Therefore, the need to estimate where an RSO is in a reasonable time remains an important topic of research [2, 3]. Especially in cases of on flight decision making, the need to accurately and quickly estimate uncertainty is important to prevent potential disasters. While real-time execution of complex algorithms like uncertainty propagation have been achieved, high cost, desktop grade hardware was required. It has been shown by Liu et. al. that a mixed multi-threading and multi-streaming GPU and CPU system was needed in order to achieve this result [4]. Currently, substantial work is being done to mitigate the computational cost of uncertainty propagation. One such algorithm, Orthogonal Probability Approximation (OPA), in its construction aims to reduce computation time compared to traditional Monte Carlo simulations, but is still affected by the curse of dimensionality. Researchers have been working to remove the restriction of the curse of dimensionality in propagations, but the need for a high fidelity dynamics propagation model remains, especially for deployment on flight. The ultimate goal is to develop techniques that allow OPA, and similar algorithms, to maintain their computational efficiency even

as problem scales in higher dimensions and deployed onto lower powered hardware. The aforementioned issues will prohibit complex algorithms from achieving real-time computation.

The goal of this work is to identify neural networks that can be integrated into complex algorithms to enable real-time numerical integration, and the primary focus for application lies in the space domain where the demands for quick and precise locations of space based objects are paramount to mission safety. In this work, two shallow neural networks were identified and tested for their ability to approximate solutions to orbital dynamics problems. The use of a unsupervised neural network, as well as novel additional physical constraints and piecewise network construction allowed for a set of neural networks to perform motion prediction at a fast rate enabling real-time performance. This work is organized as follows; the next chapter reviews the use of nonstandard numerical integration for dynamical propagation and the networks used in similar applications. The following chapter then describes the methodology for developing parallel approaches to solving numerical integration using a parallelized implementation of neural networks. Both supervised and unsupervised shallow feedforward networks were deployed and validated against a CUDA implemented numerical integrator for a nonlinear pendulum problem, which mimics the oscillatory behavior of orbit trajectories. Results are generated by comparing solution accuracy and evaluation speeds of thousands of numerical integration performed both by a standard numerical integrator and the neural network approaches. Based on these results, the unsupervised network provided better accuracy and was then demonstrated on two planar orbit problems, the perturbed two body problem with J2 perturbation and drag, and the circular restricted three body problem in the cislunar domain.

CHAPTER 2: NONSTANDARD SOLUTIONS TO DYNAMICAL PROPAGATION

In order to reach the goal of real-time calculation for real-time spacecraft algorithms, continued parallel optimizations of numerical integration is one way to increase computational effectiveness [8]. In the example of uncertainty quantification, GPU parallalized numerical integration provides large speed increases only when the number of required integrations is relatively low. In scaling to higher dimensions or increased model fidelity, bottlenecks occur due to data transfer speeds from GPU to CPU. As well, the performance of GPU parallel computation in lower nodal amounts scales similarly to the same task CPU parallelized, and as the dimensions of the problem increases, issues occur in achieving real-time predictions. One way to achieve large improvements to run-time is the deployment of alternate parallel techniques such as trained neural networks. The main motivation of replacing propagation operations with a neural network is a greatly reduced amount of floating point operations during network deployment.

Neural networks have been proven to possess universal approximation abilities and more recently have been able to replace numerical integrators for complex propagation of dynamical systems [9]. Several researchers have attempted to solve neural network numerical integration; a variety of solutions exist, such as using the network as your integration step, solving an envelope of solutions beforehand to then train a network on the converged data set of three body problems, solving an N-body problem for collisionless disks by minimizing a cost function composed of the dynamics and boundary conditions of the system, or even using a network to learn Kepler's equation [9, 10, 11, 12, 13]. Usually, recurrent neural networks (RNNs) are developed because of the ability to learn temporal data series which enable these networks to better learn dynamics defined by discrete events over time [14]. In certain application however, we are only interested in a state at only

some later time, the intermediate states generated by the RNN is computation that is not needed. Potential computational increases can be made through the development of such a network that can predict an event within a temporal series independent from adjacent time points. Flamant et. al. recognized this and demonstrated a neural network in a similar vein to the application goal of the targeted neural network [16]. Motivated by being interested in the state of a system at a specific time, they aimed to reduce the amount of computational effort by circumnavigating the calculation of determining the states at intermediate time points. By leveraging unsupervised learning with a multilayer perception with 8 hidden layers and 128 neurons per layer, Flamant et. al. developed a network that can solve a family of solutions; the system parameters, numerical integration time, and region of initial conditions are all independent inputs into the network. As well, important to this application, using a neural networks can bring increased efficiency compared to look-up tables when deployed on low-power embedded hardware, and the broader implications that a neural network can bring to reduce energy usage. However, there are a few drawbacks to this application. Namely, unsupervised learning requires human intervention for validating output in order to ethically develop and deploy such a system. Moreover, unsupervised learning increases computation time and complexity for training [17]. With these drawbacks, it is important to identify the potential advantages unsupervised networks have compared to the supervised.

While unsupervised networks have increased computational requirements for training, data set generation for supervised networks can also be a computationally and memory intensive task. Generally, unsupervised neural networks identify patterns in data sets that are not classified or labeled, removing potential bias from a curated and labeled data set. In the case of dynamical propagation, unsupervised networks differ from supervised networks because they do not rely on a solution to the differential equation before training occurs. One specific unsupervised algorithm first pioneered by Lagaris demonstrated approximations of ordinary and partial differential equations [18, 19]. The approach relies on constructing a trial solution approach which takes a form

to satisfy any boundary conditions and is a function of a neural network output. As well, this solution is unsupervised: no knowledge of an existing solution is needed, the underlying dynamics of the system drives training to adhere to the physics of the problem. Perhaps the most interesting result of this method is the ability to find a state at some arbitrary time as long as that time lies within the domain of training. Continuing this work, several researchers have expanded this work to construct trial solutions over an entire domain of initial conditions, using networks to solve an optimal planar orbit transfer, [16, 20, 21, 22] or incorporating the Hamiltonian or Lagrangian into error calculation. Unfortunately, research examining the ability for these networks to learn orbit solutions with complex perturbations or multiple bodies has been lacking.

Based on the literature, there is a clear gap in the ability for real-time dynamical propagation to be done for on orbit devices. Due to this, it is important to examine both supervised and unsupervised networks in their ability to approximate numerical integration of complex differential equations to determine which network structure is best for deployment on low-powered hardware in terms of network accuracy and speed. First, the most common type of shallow network is examined, the standard supervised learning model with the construction of large data sets to train the network. Then, an unsupervised model is examined, and additional augmentations are applied to aid training. The following chapter details the hardware targeted for this study, the structure of the supervised network, and the structure and training metrics used for the Physics-Informed Neural Network.

CHAPTER 3: METHODOLOGY

Flight-like Hardware

With the ultimate goal of improved computational speeds in numerical integration, unrestricted computational power is one way to circumnavigate slow computation; work done on propagating multiple RSOs is just one example of this, utilizing an NVIDIA Quadro K2000, an Intel core i7-860 and 4G of memory, it was possible to propagate 50 RSOs simultaneously, along with visualizing paths [4]. However, the device uses high end enterprise grade components, many of which would not fly because of restrictions on heat generation or radiation tolerance [23]. In general, both GPU parallel computation and GPU evaluated neural networks have been shown in real-time applications such as simulations of fire and solid burnings, raycastings, and gesture-based quadrocopter control [5, 6, 7]. Again, these complex algorithms have relied on computational power that does not exist for devices approved for spaceflight. In this work, restrictions to computational ability is paramount to emulate flight deployment. One such device used in the commercial flight is the NVIDIA Jetson TX2 [24]. This device is an embedded system-on-module that utilizes the CUDA toolkit in order to accelerate performance in parallel tasks [4]. Using a carrier board, this system-on-module is one of the top components of the Jetson family. We opted to deploy the integration on similar "flight-like" hardware: the NVIDIA Jetson Nano, seen in Figure 3.1. The Jetson Nano offers the same development cycle a TX2 module offers while being more performant and energy efficient.

Considering hardware implementations that use GPUs for computation requires thought be taken because of the inherent limitations hardware poses. For example, suppose a CUDA kernel is written to perform matrix multiplication. Suppose Figure 3.2 details this operation. If the kernel requires 2 GBs of data to perform a calculation, the GPU cannot receive all data at once. As a

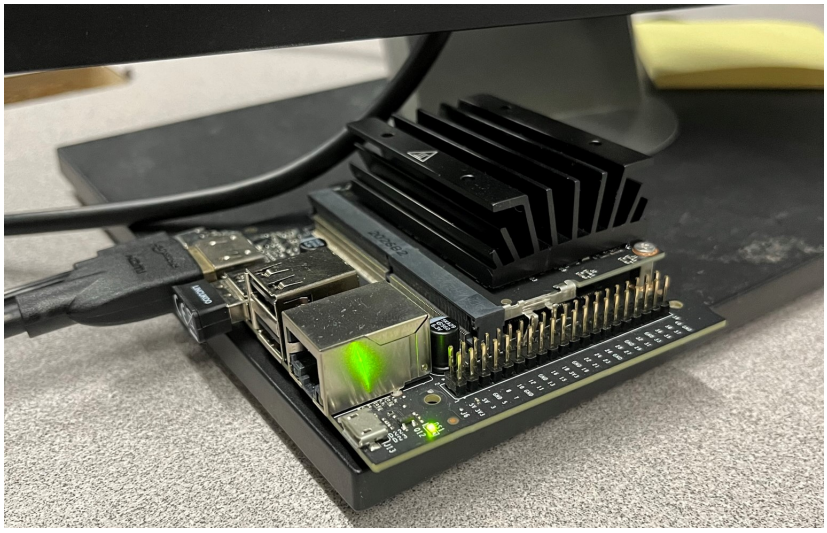

Figure 3.1: NVIDIA Jetson Nano

result, the incoming data must be split into smaller blocks that are then transferred. The GPU kernel performs the operation on the smaller blocks of data, and sends the output back to the CPU [25]. By performing this way, in some cases the input data does not have a critical size and evaluation on the GPU is slower than evaluation on the CPU even if the GPU kernel can perform the computation orders of magnitude faster.

Consider the simple harmonic oscillator described by Equation (3.1). Solving the differential equation using an RK4 integrator, the runtime for a single solution of the RK4 run performed on the GPU before and after transfer off of the GPU is performed on the Jetson Nano. Table 3.1 demonstrates the speed of transfer needed for a single solution.

$$\ddot{x} + \frac{k}{m}x = 0 \tag{3.1}$$

With this limitation, it is important to keep in mind that standard numerical integration techniques

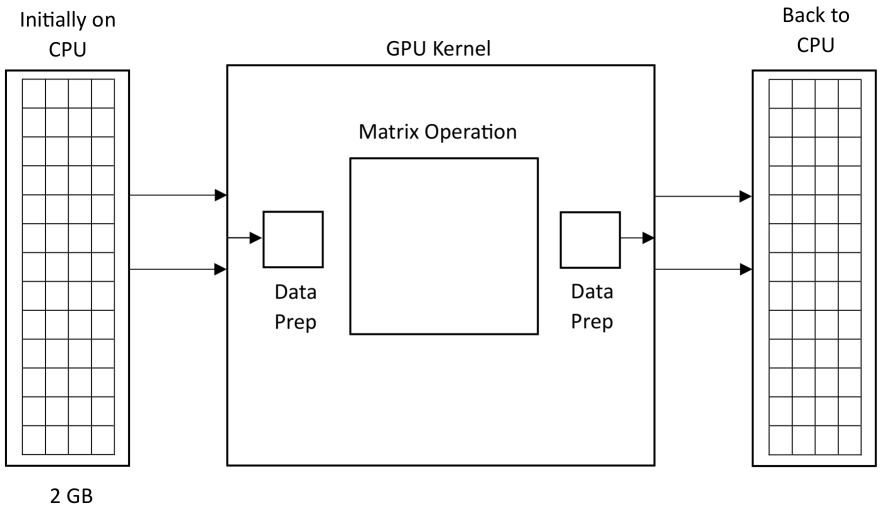

Figure 3.2: Example GPU Kernel

Table 3.1: Average Time to Integrate Single Solution Before and After Transfer to CPU

RK4 solution (before)	RK4 solution (after)	GPU Transfer Speed
4.3840e-2 sec	4.4080e-2 sec	2.4000e-04 sec

have been thoroughly examined, optimized, and designed for computational speed and accuracy. A neural network numerical integrator, or numerical integration performed by CUDA performs best when the amount of integrations exceeds some critical amount, which is dependent on the hardware power. Below this critical amount, standard numerical integration techniques are best.

Supervised Neural Network

Great success was found in the development of a neural network integrator using a shallow, feed-forward network, as in Figure 3.3 [26]. Mathematically, a shallow network is defined by Equation (3.2) where a single activation function σ is used on the summation of the weights, W, of the

first layer and the input values, with a bias term b. Usually, the activation function is defined as a sigmoid function, Equation (3.3).

$$\mathbf{y} = W_2\sigma(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{3.2}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

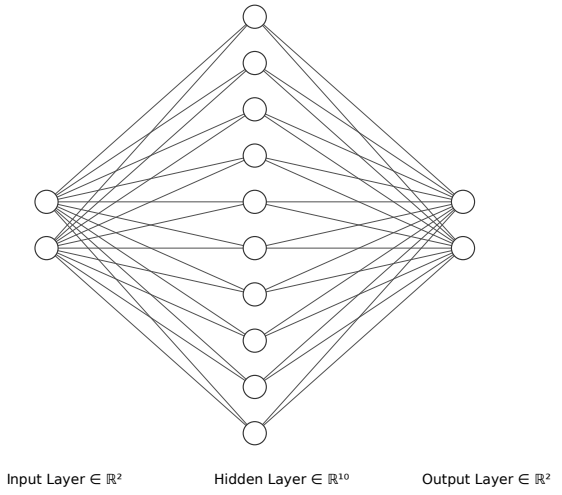

Figure 3.3: Shallow Neural Network

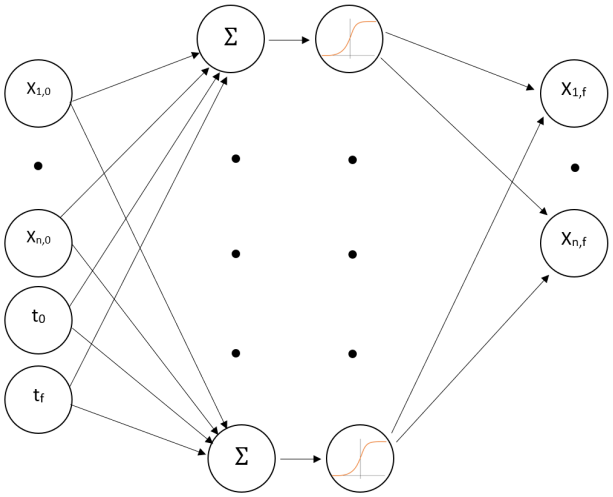

Figure 3.4: Proposed Neural Network Structure

Using this network structure as the base for the first network we will examine, the shallow network layout is described by Figure 3.4. For this specific application, implementing the additional two nodes in the first layer of the network encodes the ability for the network to learn integration over a region of time. Training this network necessitates a large input-output data set. To train this network, a large set of bounds must be generated per initial condition along with the interval for propagation, with the output set being the states at the desired time. Figure 3.4 demonstrates along

side Equation (3.4). To generate this data set, offline numerical integration of the initial set $\mathbf{X}$ must be done to generate the output set $\mathbf{Y}$. This process, while computationally ineffective because of the required generation of large data sets that necessitate solving the numerical integration anyway, can be viewed as precompiling a solution to a dynamical system for deployment, akin to a lookup table.

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} \mathbf{x}_0 \\ t_0 \\ t_f \end{bmatrix} \\ \mathbf{Y} &= \begin{bmatrix} \mathbf{x}_f \end{bmatrix} \end{aligned} \tag{3.4}$$

Implementation Notes

For a shallow network, there have been several heuristics to develop an "optimal" size for the hidden layer size [26].

$$k = \left\lfloor \log_{10}(N)^{-K_1 \frac{K_2 N}{n+2}} \right\rfloor \tag{3.5}$$

Where N is the size of training points given to the system, n is the input size to the network, and K_1 and K_2 are tunable parameters to prevent k from growing too large at larger values of training points. This was suggested for good general performance as this prevents k from growing too large, yet was not given any analytical justification, however, this heuristic provided better network performance than others [27]. For the network, the parameters have been tuned to $K_1 = 4$,

$K_2 = 20$.

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\left[\hat{y} - y\right]^2 \tag{3.6}$$

Network evaluation is performed using the mean-squared error metric, Eq 3.6, and network training is performed using the Adaptive Moment Estimation (Adam) backpropagation algorithm [28]. The Levenberg-Marquardt backpropagation is another algorithm that is one of the fastest training methods for small to medium-sized networks, however in our testing we found that the Adam optimizer captured the nonlinearities of estimating a numerical integrator better than Levenberg-Marquardt. The use of Adam was motivated by the results found in solving the chaotic three-body problem using feedforward networks [29, 11].

Validation

Driving the discussion of the validity of the neural network numerical integrator is a nonlinear pendulum problem trained around the 6σ bound around an arbitrary mean propagated for different times. The short propagation time of 5 seconds was chosen to capture smaller nonlinearities, while the longer propagation time of 15 seconds was chosen to capture the accumulation of the nonlinearities. Equations (3.7) describes the system dynamics, and Equations (3.8) and (3.9) describe the uncertainty used for generating a 6σ bound and the different propagation times.

$$\ddot{\theta} + \frac{g}{l}\sin\theta = 0 \tag{3.7}$$

Short Propagation Time

$$\begin{aligned} \mu &= \begin{bmatrix} 30^\circ & 0 \end{bmatrix}^T \\ \sigma &= \begin{bmatrix} 0.03 & 0.05 \end{bmatrix}^T \\ g &= 9.81 \quad l = 1 \\ t &= 5sec \end{aligned} \tag{3.8}$$

Long Propagation Time

$$\begin{aligned} \mu &= \begin{bmatrix} 30^\circ & 0 \end{bmatrix}^T \\ \sigma &= \begin{bmatrix} 0.03 & 0.05 \end{bmatrix}^T \\ g &= 9.81 \quad l = 1 \\ t &= 15sec \end{aligned} \tag{3.9}$$

Llyod et. al. used the heuristic, Eq (3.10), to quantify the ability of their neural network integrator, taking the number of average correct digits to be the measure of performance [26]. Adopting this, all results that demonstrate decimal accuracy will use this heuristic to estimate the amount of correct digits that a neural network estimates.

$$\log_{10} || \frac{I(f)}{I(f) - \hat{I}(f)} || \tag{3.10}$$

Using Equation (3.7) and the parameters in Equations (3.8) and (3.9), the mean decimal accuracy for this dynamical system was calculated based on the numerical integration results from the neural network integrator as well as a Runge-Kutta 4th order solver (RK4). Because the nonlinear

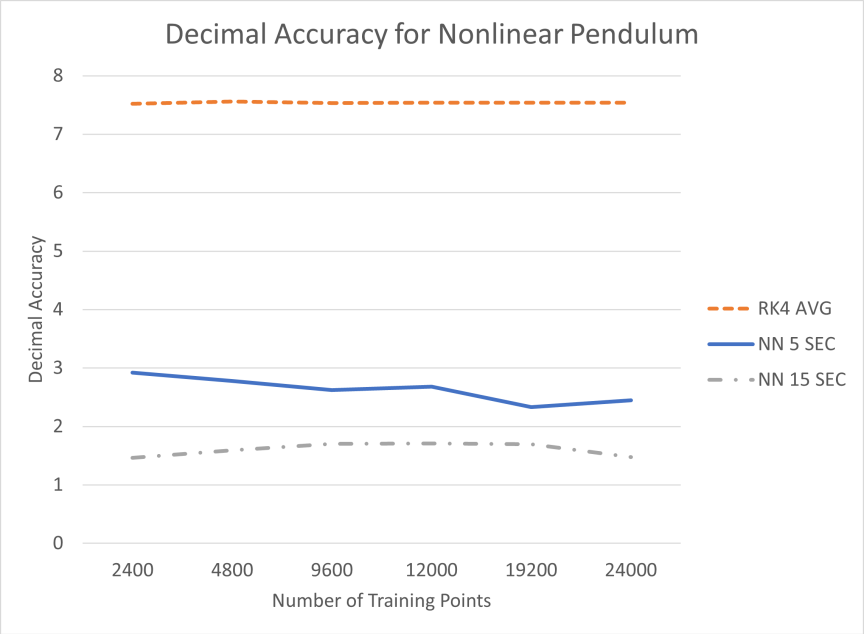

Figure 3.5: Mean Decimal Accuracy for Nonlinear Pendulum System

pendulum system does not have a simple closed-form solution, the numerical solution from an 8th order Runge-Kutta routine (RK8) is taken as the truth. The neural network and RK4 are then used to estimate the true solution and compared with Equation (3.10) and plotted in Figure 3.5. The neural network predicted both a 5 second and 15 second propagation.

Compared to RK4, the neural network performs with around two fifths of the total accuracy to that of RK4 for a short propagation time, and around one fifth of the total accuracy for a longer propagation time; as the nonlinearities propagate through the motion of the system, longer propagation times will further increase the complexity the network must learn. As the number of integrations increased, we can see that the neural network solver remains constant in it's accuracy, pointing towards the optimal hidden layer size heuristic provides good neuron scaling as the number of training points increases. However, the loss in accuracy to the standard numerical integration truths give reason to suggest that a standard shallow feedforward network using supervised

learning cannot capture the complexities of nonlinear differential equations like deep feedforward networks can.

Jetson Nano Evaluation Speeds

The effect that the neural network has on run time of solving integrations was compared with the same process using CUDA parallelization, as seen in Figure 3.6. After 100,000 integrations, both the neural network and the CUDA parallelization are observed to scale linearly with the growing number of numerical integrations, with the neural network scaling 3.48 times slower. Additionally, Table 3.2 further demonstrates the average time to evaluate a numerical integration between a CUDA parallelized task and the evaluation of a neural network model deployed on a GPU. The average execution time for a trained model versus a numerical integration using CUDA parallelization can improve computation speed by up to four orders of magnitude before transfer off of the GPU.

Table 3.2: Average Time to Integrate Single Solution Before and After Transfer to CPU

RK4 solution (before)	RK4 solution (after)	Model Evaluation (before)	Model Evaluation (after)
4.3840e-2 sec	4.4080e-2 sec	6.5970e-6 sec	1.2647e-3 sec

The most major drawback to this solution type is raw amount of data required to train a network over an entire time regime. As a result, this network solution is best when trained for a single time point of interest, instead of an entire trajectory. Implementing a network approximation like this is useful in certain applications where the entire time trajectory is not needed, like uncertainty quantification, where the calculation of a probability density function is performed for a time point of interest. While this network is not effective over large time trajectories, studying a network of this ability is useful in determining overall computational speed of neural networks as well as

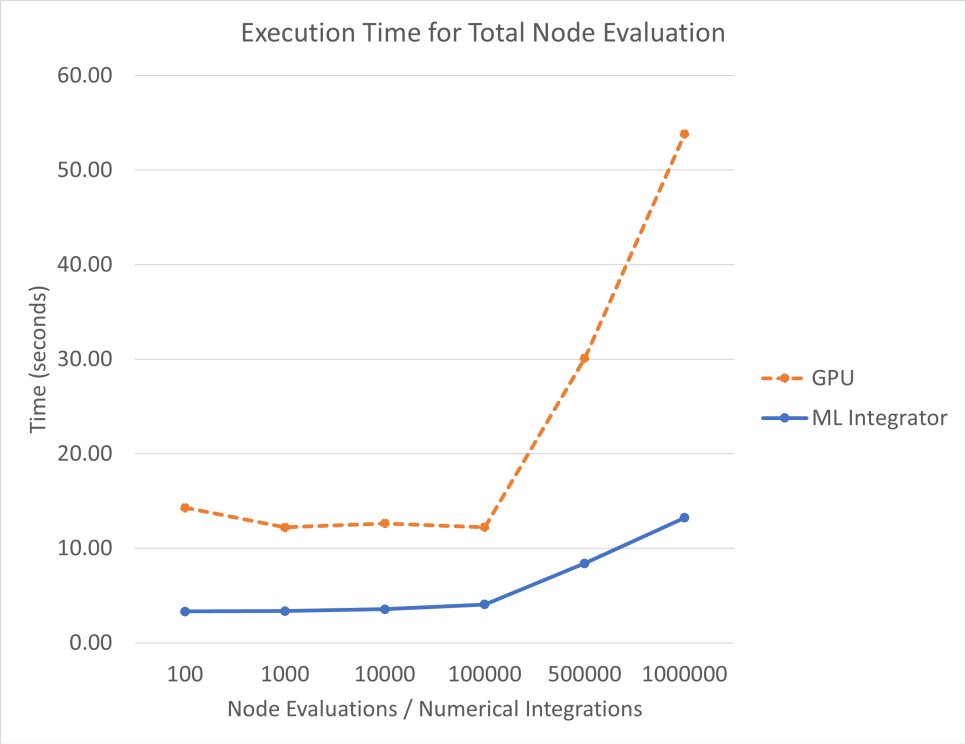

Figure 3.6: Parallelization of Forward Propagation on GPU - Flight Hardware

approximate accuracy for solutions using a neural network approximation. In these early results, the major drawback of this solution type is accuracy, but significant computational speed increases can be seen. However, improvements to the accuracy must be made, especially because of the drastic decrease in accuracy. Lower accuracy in neural networks models are not uncommon, but restricting to the shallow network model requires investigating additional network structures, such as an unsupervised model. In the following section, a review of the general Lagaris method for constructing an unsupervised physics informed neural network is explained, and improvements to the loss function are developed.

Physics Informed Neural Network

Lagaris first detailed a method to construct an unsupervised approach to solve differential equations [18]. Starting, consider a system of j-coupled first order differential equations:

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \mathbf{p}) \tag{3.11}$$

The equations of motion can be written in a general form:

$$G(t, \Psi(t), \Delta\Psi(t), \Delta^2\Psi(t)) = 0, t \in D \tag{3.12}$$

where $\Psi(t)$ is the solution to be computed. This approach relies on minimizing this form with respect to the network parameters w.

$$\min_{\mathbf{w}} \sum_{t \in \hat{D}} G(t, \Psi(t, \mathbf{w}), \Delta\Psi(t, \mathbf{w}), \Delta^2\Psi(t, \mathbf{w}))^2 \tag{3.13}$$

Continuing, we construct j trial solutions of an arbitrary form, which is learned with j neural networks, referred to as a network bundle, which must satisfy the original differential equation. From Equation (3.14), $\mathbf{A}$ and F are chosen to satisfy the boundary conditions and act as another parameter of the system that can affect network training. The network input consists of the domains of time and parameters of the differential equation, $\mathbf{p}$; these can be initial conditions or physical parameters of the system. Additionally, the shallow network consists of a single bias b, and two weight matrices w_1 and w_2. The network structure is illustrated in Figure 3.7.

$$\mathbf{\Psi}(t, \mathbf{p}) = \mathbf{A}(t) + F(t, \mathbf{N}(t, \mathbf{p})) \tag{3.14}$$

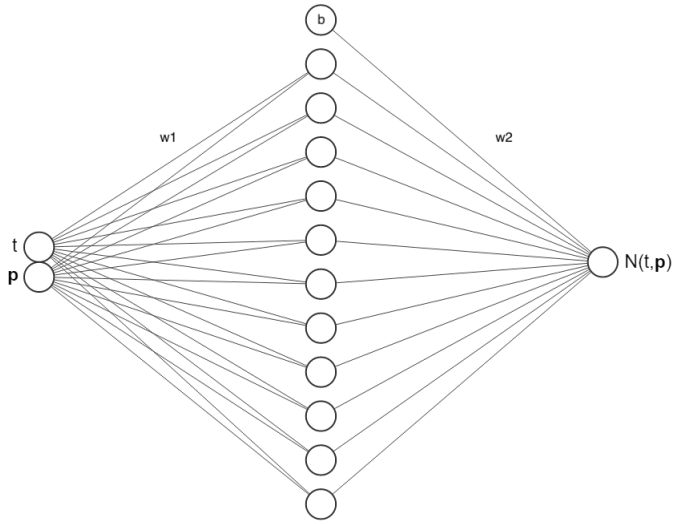

Figure 3.7: PINN Network Layout

$$\frac{d\mathbf{\Psi}(t,\mathbf{p})}{dt} = \mathbf{f}(t, \Psi(t,\mathbf{p}), \mathbf{p}) \tag{3.15}$$

Deviating from the original Lagaris technique, we can introduce several improvements to network training by first exploring the idea of kinematic consistency between the coupled network solutions. Suppose we want to approximate a simple harmonic oscillator defined by the two coupled differential equations in Equation (3.16). By approximating the solutions with two trial solutions, we can define a Ψ_1 and Ψ_2 that approximates the position and velocity of the system as a function of time. It is reasonable to assume that the time derivative of the trial solution Ψ_1 must equal the trial solution Ψ_2. By exploiting this, we can introduce both another metric to inform the network solution bundle of the physical properties of the system, as well as define the activation functions of the networks to have this inherent property. Therefore, in our example, network 1, used to approximate the position, uses a sinusoid activation function while network 2, used to approximate

the velocity, uses a cosine activation function.

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{k}{m} x_1 \end{aligned} \tag{3.16}$$

Finally, the loss function is defined to satisfy the equations by reducing the loss defined by three major metrics. Firstly, the residual of the governing differential equation defined by the mean square error (MSE) in Equation (3.17), which was included in the original Lagaris design. From the introduction of kinematic consistency, enforcement of this can be continued by providing another metric for learning as constructed as such in Equation (3.18). Finally, in the cases where a known parameter of a system remains constant, enforcing the conservation of that known quantity such as in Equation (3.19). In many cases, the Hamiltonian, H, of a conservative dynamical system can take this form [21]. For orbital systems, this can take the form of the integrals of motion. These loss metrics are then summed and an optimization method is used to find the optimal weights the network, such as the stochastic gradient descent method, Equation (3.21).

$$\mathcal{L}_{MSE} = \sum_i \left(\frac{d\mathbf{\Psi}(t_i, \mathbf{p})}{dt} - f(t_i, \mathbf{\Psi}(t_i, \mathbf{p}))\right)^2 \tag{3.17}$$

$$\mathcal{L}_{KC} = \sum \sum_i \left(\frac{d\Psi_k(t_i, \mathbf{p})}{dt} - \Psi_j(t_i, \mathbf{p})\right)^2 \tag{3.18}$$

$$\mathcal{L}_H = \sum_i \left(\mathcal{H}(\mathbf{\Psi}(t_i, \mathbf{p})) - \mathcal{H}_0\right)^2 \tag{3.19}$$

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_H + \mathcal{L}_{KC} \tag{3.20}$$

$$w^{i+1} = w^i - \nu \frac{\partial \mathcal{L}}{\partial w} \tag{3.21}$$

Piecewise Defined Network Solution

Improvement to numerical accuracy can be a difficult challenge training only one network bundle. Over one domain, the solution accuracy of a network deteriorates as the time domain of the solution grows very large. Using only one network bundle can yield good results, but we forsee network scaling to become intractable for highly complex differential equation solutions especially in higher dimensional problems like a perturbed three dimensional orbit problem. A remedy to this that retains network simplicity and adequate network scaling is the use of piecewise network solutions.

Suppose we have a complex solution to a differential equation over a long time period, such as Figure 3.8. Over this region, there are three major nonlinearities in which a shallow network would have a hard time approximating a function that fits this complex shape completely over a single interval. Splitting this function into three regions, separated by color, will allow for a network bundle to approximate the specific regions easier due to the inherent simplicity resulting from segmenting a more complex solution into smaller regions. Supposing we construct multiple trial solutions over a finite domain with each trial solution domain being a unique set. By splitting a time domain into i sections bounded by a time domain defined as $\Big[a_i, b_i\Big]$. For each section i a unique bundle of networks are created each with its own trial solution. Every trial solution region is trained over its domain with the resulting overall solution to the differential equation being a

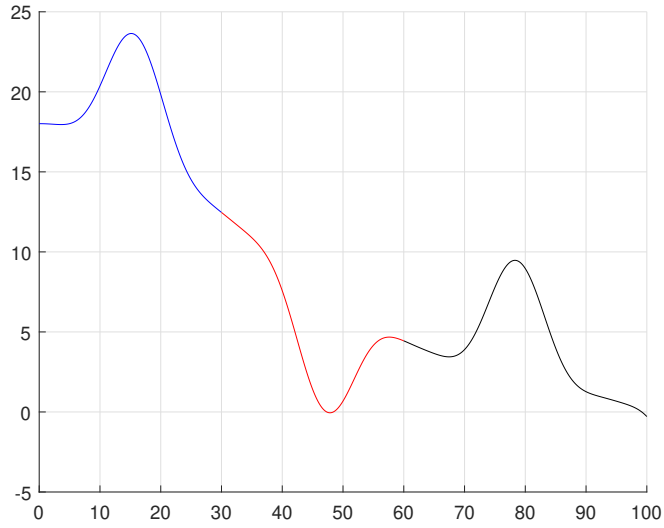

Figure 3.8: Example Complex Solution

piecewise function of networks, illustrated in Equation (3.22).

$$\begin{aligned} \mathbf{N}(t,\mathbf{p}) &= \begin{cases} \mathbf{N}_i(t,\mathbf{p}) & a_i \leq t \leq b_i \end{cases} \\ \mathbf{\Psi}(t,\mathbf{p}) &= \begin{cases} \mathbf{A}_i(t) + F(t,\mathbf{N}_i(t,\mathbf{p})) & a_i \leq t \leq b_i \end{cases} \\ b_{i+1} &= a_i \quad \text{where} \quad i \neq 0 \end{aligned} \tag{3.22}$$

By applying a piecewise solution approach, we can assure that finer learning can be applied to each region of the solution, with reduced training time per network and less nodes required per network. This technique can be utilized when a single network bundle cannot provide accurate solution approximation. However, applying this solution method does have a few downsides, mainly memory requirements. By segmenting the solution into several piecewise network solutions, the number of weights required to be store for deployment increases significantly. Care must be taken when

deploying this approach on low-powered hardware.

Validation

Consider the same nonlinear pendulum example problem from the supervised learning network, for both the short and long propagations. For each propagation, the trajectory solution is constructed with the same number of piecewise network approximations per second of propagation time. Beginning with the short propagation time, the network solution provides an arbitrary function for a trajectory on the initial condition $\begin{bmatrix} 30^\circ & 0 \end{bmatrix}$. Because of this, the following results generate a full trajectory in which none of the generate solution points are affected by its neighboring point. This will become more evident as the propagation time increases. Figures 3.9 and 3.10 showcase a fully generated solution to the nonlinear pendulum over a 5 second propagation time. On average, this solution has an accuracy of approximately 2.877 decimal points.

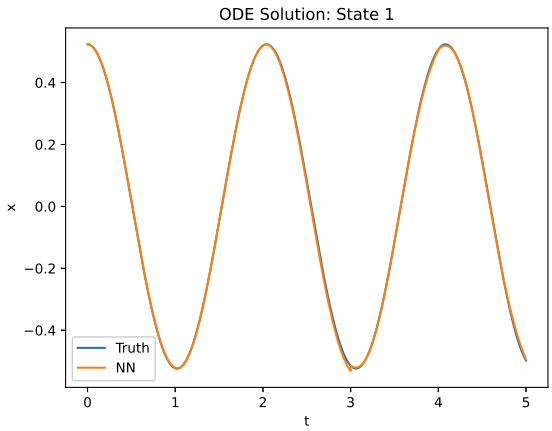

Figure 3.9: 5 Second Propagation - State 1

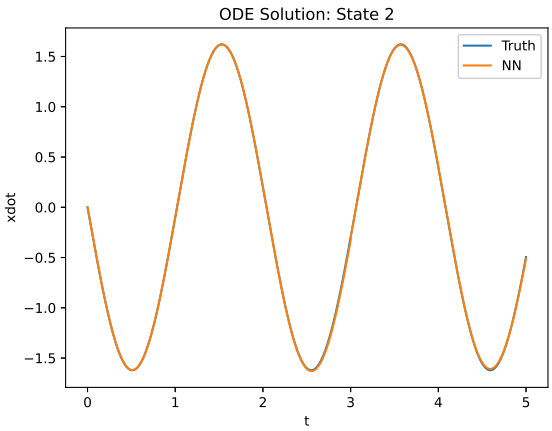

Figure 3.10: 5 Second Propagation - State 2

As we increase the propagation time, we can see that the unsupervised network displays the same issues with propagation time scaling that the supervised network demonstrated. As we increase the

propagation time we can see in both figures 3.11 and 3.12 that the solution approximation decreases in accuracy as the time increases. On average, this solution has an accuracy of approximately 1.994 decimal points. Figure 3.13 demonstrates the state error between the truth and the neural network approximation. We can see that for the nonlinear pendulum, as the nonlinearity of the solution grows a segmented network approach cannot fully capture this.

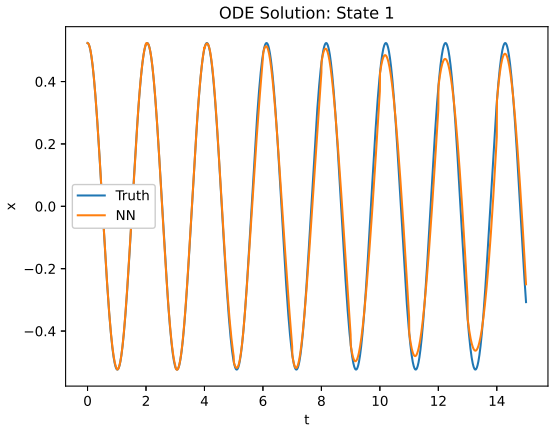

Figure 3.11: 15 Second Propagation - State 1

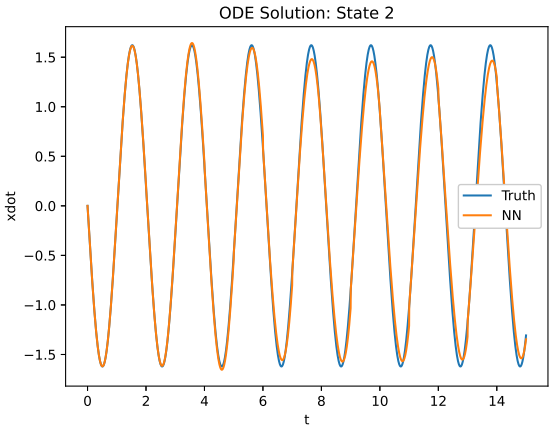

Figure 3.12: 15 Second Propagation - State 2

While both network structures demonstrate issues in long propagation times, the physics informed network provides a better approach to solving the solution of a dynamical propagation. Without needing a large training dataset, the PINN can generate time point independent solutions to trajectories regardless of equation parameters or initial conditions.

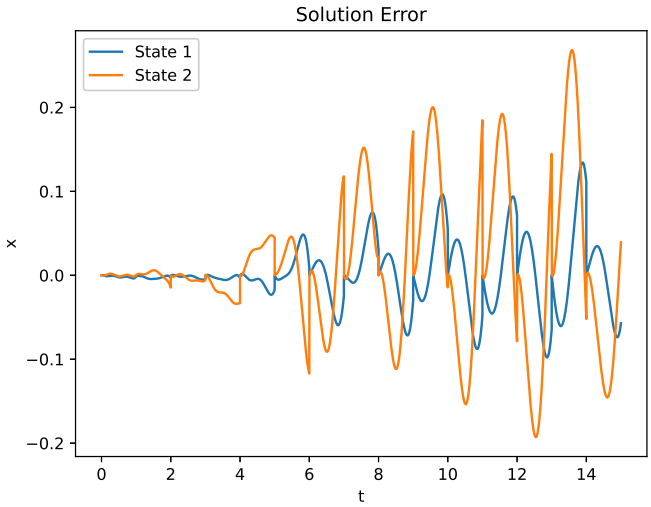

Figure 3.13: 15 Second Propagation Error

CHAPTER 4: PHYSICS INFORMED SOLUTION TO PLANAR ORBIT PROBLEMS

Two-Body Problem

Consider a planar two-body problem subject to perturbing J2 and drag accelerations:

$$\ddot{r} = -\frac{\mu}{\|\mathbf{r}\|^3}\mathbf{r} + \mathbf{a}_p \tag{4.1}$$

This equation can be rewritten as 4 coupled first order differential equations, expanding the terms in to include the J2 and drag perturbations:

$$\begin{aligned}
\frac{dx}{dt} &= v_1 \\
\frac{dy}{dt} &= v_2 \\
\frac{dv_1}{dt} &= -\frac{\mu}{\|\mathbf{r}\|^3}x + 1.5J2\frac{R_E^2}{\|\mathbf{r}\|^5}x - \frac{1}{2}\rho\|\dot{\mathbf{r}}\|\frac{C_D A}{m}x \\
\frac{dv_2}{dt} &= -\frac{\mu}{\|\mathbf{r}\|^3}y + 1.5J2\frac{R_E^2}{\|\mathbf{r}\|^5}3y - \frac{1}{2}\rho\|\dot{\mathbf{r}}\|\frac{C_D A}{m}y
\end{aligned} \tag{4.2}$$

Constructing 4 trial solutions, we can augment the final loss function by tweaking Equation(3.18) to enforce consistency with the trial solutions:

$$\mathcal{L}_{KC} = \sum_i (\frac{d\Psi_1(t_i, \mathbf{p})}{dt} - \Psi_3(t_i, \mathbf{p}))^2 + (\frac{d\Psi_2(t_i, \mathbf{p})}{dt} - \Psi_4(t_i, \mathbf{p}))^2 \tag{4.3}$$

and finally the trial solution can be constructed in the following form:

$$\mathbf{\Psi}(t, \mathbf{N}) = sin(\frac{\pi}{2t_f}t)\mathbf{N}(t, \mathbf{p}) + \mathbf{x}_0 \tag{4.4}$$

Note that when $t = t_0$, the trial solution satisfies the initial conditions. We can train these networks simultaneously using PyTorch with the hyperparameters in Table 4.1.

Table 4.1: Two Body Problem Hyperparameters

Time Domain	t $\in [0, T_{\text{unperturbed}}]$
Data Set Size	1000
Training Epochs	10000
Learning Rate	0.002
Hidden Size Per Network	35
Optimizer	Adam

An elliptical orbit was chosen as the dynamics to learn, with orbital parameters of $a = 8578$ km, $e = 0.2098$, and initial conditions of $r = \begin{bmatrix} 6778 & 0 \end{bmatrix}$km and $v = \begin{bmatrix} 0 & 8.435 \end{bmatrix}$ km/s, J2 = $1.08263e-3$, $\rho_0 = 1.29$kgm^3, $C_D = 2.1$, and $A = 1.0013$ m^2. Equation (4.5) details a simple Jacchia drag model with $H = 50$km Normalizing the orbit by DU $= 6378$ km and TU $= 806.80415$ sec reduces the magnitude of the input data being given to the network, which aids network training stability. The network was trained over an equispaced time domain of 1000 points, and test data was chosen from the reported time variable output from RK45. Figure 4.1 demonstrates the fit that the neural network solution has on an orbit solved by RK45, with Figure 4.16 and 4.17 comparing the accuracy of the network to the RK45 solution. Note that the decimal accuracy is calculated using Equation (4.6), which is a modified heuristic from Lloyd to estimate the number of accurate

digits [26].

$$\rho = \rho_0 e^{-\|\mathbf{r}\|/H} \tag{4.5}$$

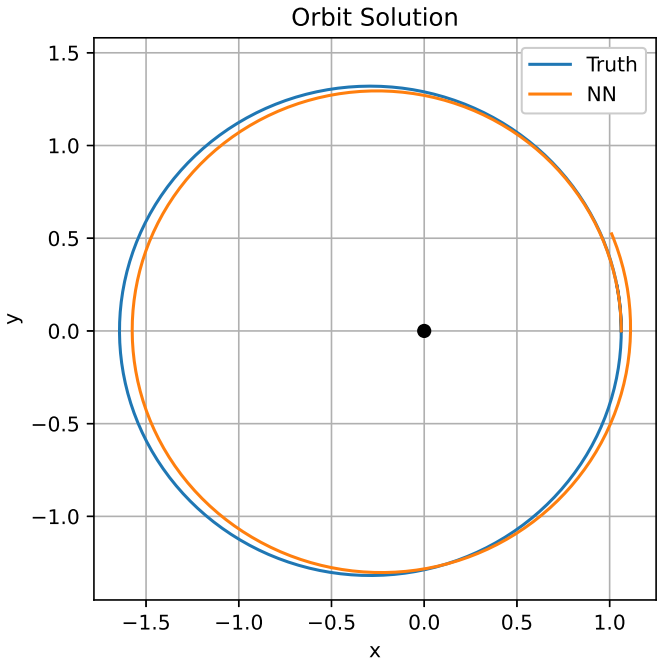

Figure 4.1: 1 Period Orbit Propagation - Solution Along Trajectory

$$\epsilon = \left| \log_{10} \left(\left| \frac{1}{\mathbf{x}_i - \Psi_i} \right| \right) \right| \tag{4.6}$$

Propagation of Multiple Orbit Periods

Using a single network bundle, the solution of a single orbit of a perturbed two body problem is difficult to solve. Tweaks to the hyperparameters could remedy this, but for longer propagation times, the accuracy will continue to fall. For a long propagation period, like four periods, using a

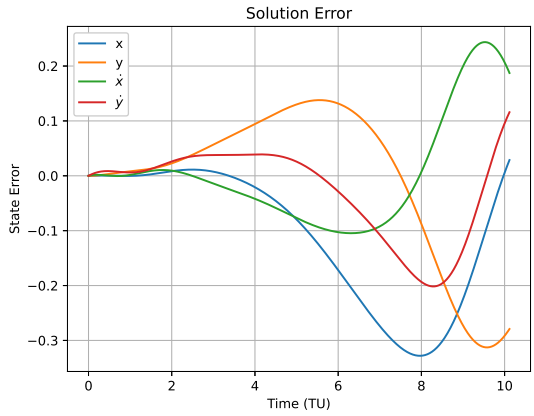

Figure 4.2: Solution Error

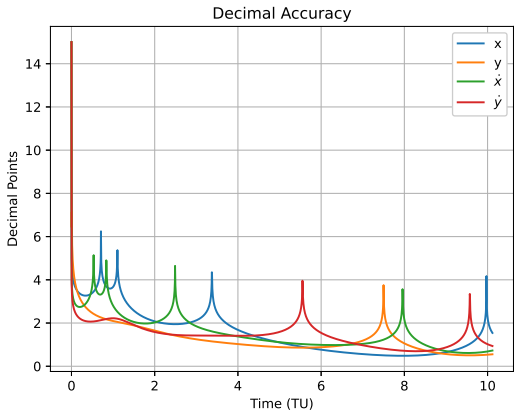

Figure 4.3: Decimal Accuracy

piecewise approach to construct the solution is the best approach to ensure high accuracy. Consider the previous problem now propagating for four orbit periods with each orbit period being subdivided into 10 subtrajectories for a total of 40 subtrajectories. With each subtrajectory, a unique trial solution is constructed over that domain and the total numerical solution of the differential equation is defined using a piecewise function of the trial solutions. Training the networks with the same hyperparameters in Table 4.1, we can see the orbit solution over the four orbit periods, in figs 4.4 - 4.7. An interesting phenomenon occurs when using the piecewise approach and is found across all simulations using this approach. For piecewise solutions, the jumps in accuracy are due to reinitializing the trial solution of the subtrajectory to its initial conditions. These jumps in accuracy are purely a byproduct of evaluating the trial solution at that initial time. By averaging the accuracy calculations, these discontinuities are dismissed as outliers.

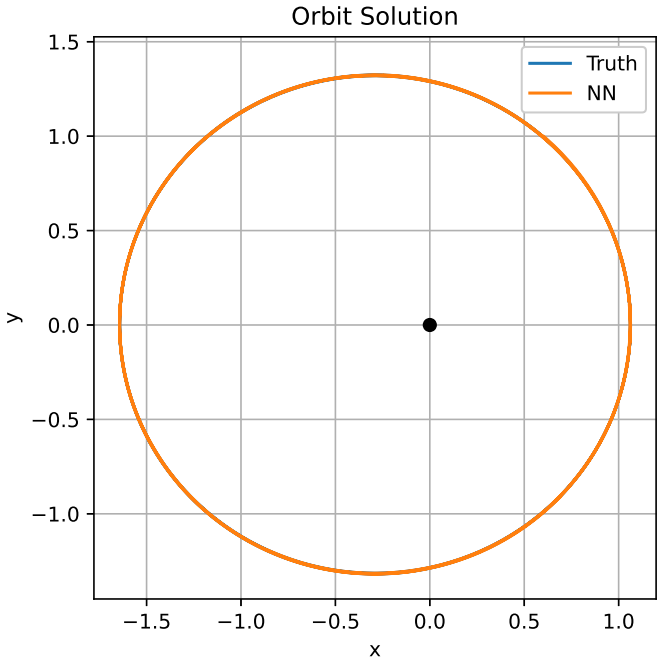

Figure 4.4: 4 Period Orbit Propagation - Solution Along Phase Space

Circular Restricted Three-Body Problem

The circular restricted three-body problem (CR3BP) is an interesting problem that presents challenges in evaluation speed; the highly chaotic and complex system requires higher computational cost to that of the two-body problem. As a direct result, the computational cost increases substantially when evaluated on low-powered flight hardware. Therefore, we wish to extend the neural network trial solution method to solve the CR3BP.

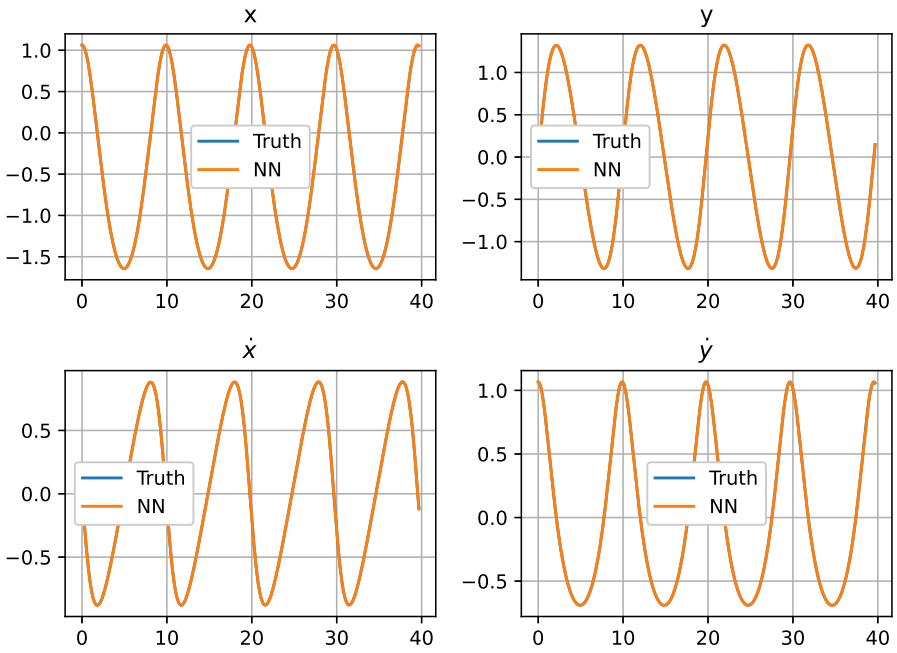

Figure 4.5: 4 Period Orbit Propagation - Solution Along States

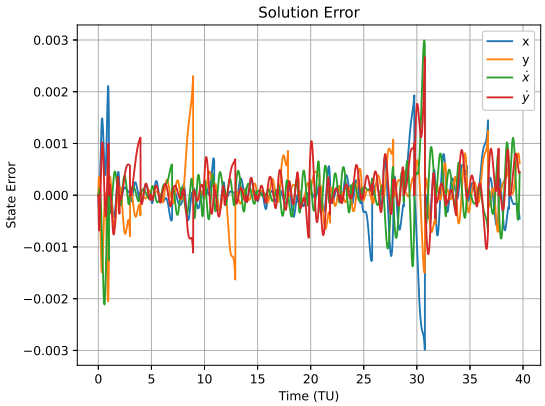

Figure 4.6: Solution Error

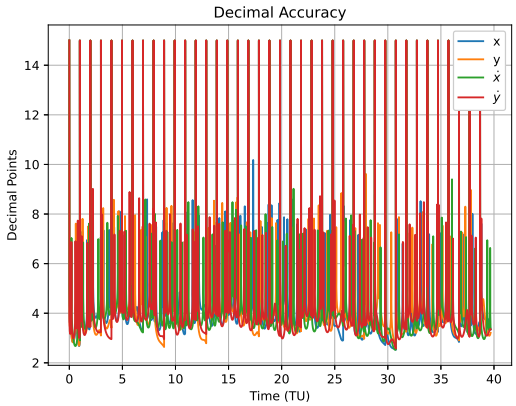

Figure 4.7: Decimal Accuracy

Consider the nondimensional form of the CR3BP with the defining nondimensional constraints:

$$\begin{aligned}\ddot{x} - 2\ddot{y} &= x - (1-\mu)\frac{x+\mu}{r_1^3} - \mu\frac{x-1+\mu}{r_2^3} \\ \ddot{y} + 2\ddot{x} &= y - (1-\mu)\frac{y}{r_1^3} - \mu\frac{y}{r_2^3}\end{aligned} \tag{4.7}$$

$$\begin{aligned}\mu &= \frac{m_2}{m_1+m_2} \\ r_1 &= \sqrt{(x+\mu)^2+y^2} \\ r_2 &= \sqrt{(x-1+\mu)^2+y^2} \\ x_1 &= -\mu \\ x_2 &= 1-\mu\end{aligned} \tag{4.8}$$

The equations of motion can be transformed to 4 coupled first order ODEs.

$$\begin{aligned}\frac{dx}{dt} &= v_1 \\ \frac{dy}{dt} &= v_2 \\ \frac{dv_1}{dt} &= 2v_2 + x - (1-\mu)\frac{x+\mu}{r_1^3} - \mu\frac{x-1+\mu}{r_2^3} \\ \frac{dv_2}{dt} &= -2v_1 + y - (1-\mu)\frac{y}{r_1^3} - \mu\frac{y}{r_2^3}\end{aligned} \tag{4.9}$$

Once again, the final loss function can be constructed by tweaking Equation (3.18) to enforce consistency with the trial solutions. Because this is a planar problem, we can enforce the conservation

of the Jacobi constant:

$$\mathcal{L}_{KC} = \sum_i (\frac{d\Psi_1(t_i, \mathbf{p})}{dt} - \Psi_3(t_i, \mathbf{p}))^2 + (\frac{d\Psi_2(t_i, \mathbf{p})}{dt} - \Psi_4(t_i, \mathbf{p}))^2 \tag{4.10}$$

$$\mathcal{L}_H = \sum_i (\mathcal{C}_J(\mathbf{\Psi}(t_i, \mathbf{p})) - \mathcal{C}_{J0})^2 \tag{4.11}$$

Consider a short period cislunar orbit around liberation point L4 with the following parameters from Table 4.2 trained on the hyperparameters from Table 4.3. Using a single network bundle to solve this differential equation, the network solution diverges drastically as seen in Figures 4.8 and 4.9. Note in the figures plotting the CR3BP, the grey diamond indicates liberation point L4, the black circle indicates the Earth, and the green circle indicates the Moon.

Table 4.2: Parameters - Short Period

x_0	0.487849413
y_0	1.471265959
v_{x0}	1.024841387
v_{y0}	-0.788224219
t	6.2858346244258847
C_0	2.018102532939865

Table 4.3: Hyperparameters - Short Period

Time Domain	$t \in [0, T]$
Data Set Size	500
Training Epochs	10000
Learning Rate	6×10^{-4}
Hidden Size Per Network	10
Optimizer	Adam
Number of Trial Solutions	40

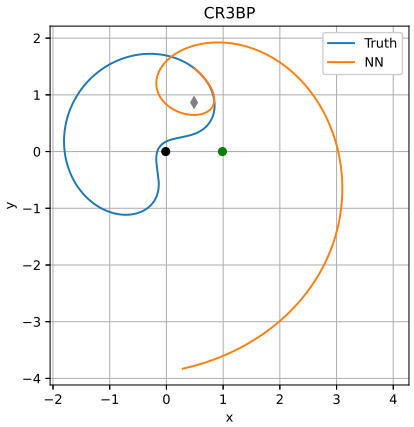

Figure 4.8: CR3BP Phase Space Solution

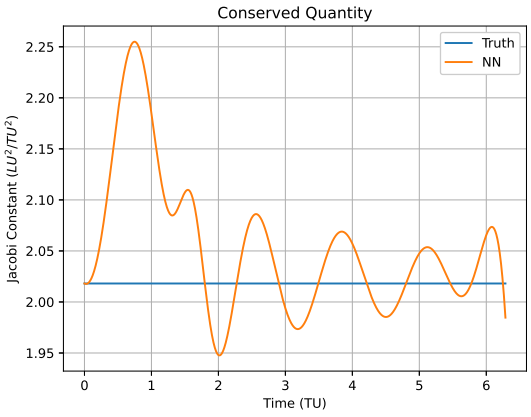

Figure 4.9: Conservation of Jacobi Constant

Over a single large trajectory, the quality of approximation is drastically reduced due to a large amount of complexity a network approximation must capture, to a point where the network get trapped in a local minima and believes that it found a solution. However, utilizing a piecewise defined approach, we can approach problem by learning specific time domains to approximate a much less complex problem. Using the same hyperparameters, but splitting the network learning

into 10 equispaced time regions over the entire domain, we can approximate the same problem with improved accuracy, as seen in Figures 4.10 - 4.13.

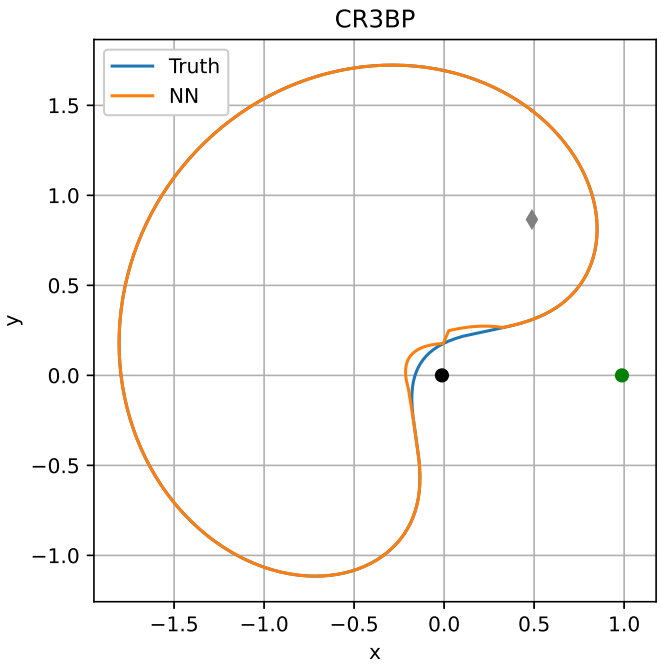

Figure 4.10: CR3BP Phase Space Solution

Consider a long period cislunar orbit around liberation point L4 with the following parameters from Table 4.4 trained on the hyperparameters from Table 4.5. Using multiple network bundles to solve this differential equation, we can see similar accuracy results as the short period orbit, in Figures 4.14 - 4.18.

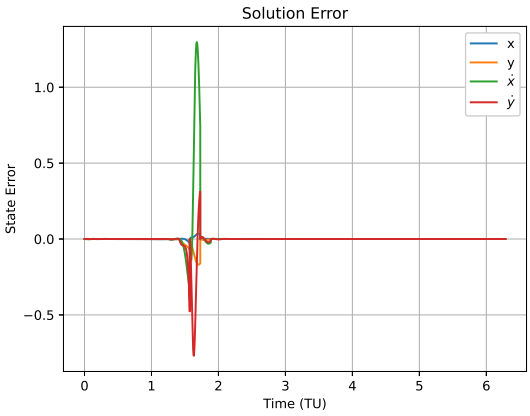

Figure 4.11: Solution Error

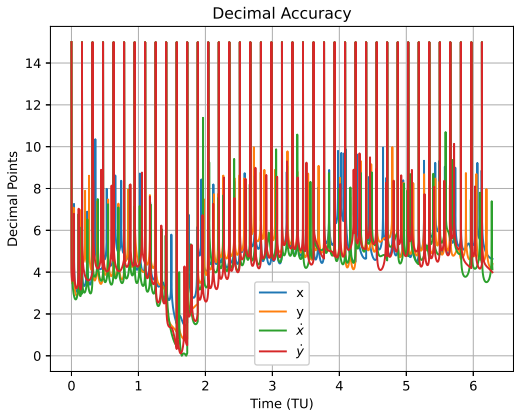

Figure 4.12: Decimal Accuracy

Table 4.4: Parameters - Long Period

x_0	4.8784941344943100E-1
y_0	7.9675359028611403E-1
v_{x0}	-7.4430997318144260E-2
v_{y0}	5.6679773588495463E-2
t	2.1134216469590449E1
C_0	2.9902575598429295

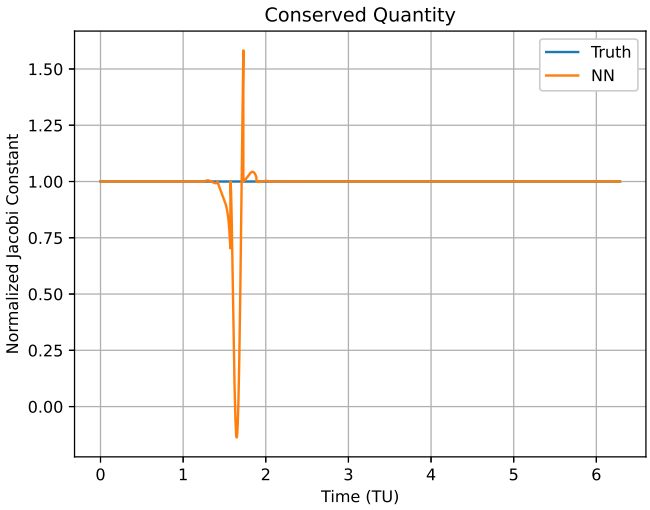

Figure 4.13: Conservation of Jacobi Constant

Table 4.5: Hyperparameters - Long Period

Time Domain	$t \in [0, 4T]$
Data Set Size	500
Training Epochs	10000
Learning Rate	6×10^{-4}
Hidden Size Per Network	10
Optimizer	Adam
Number of Trial Solutions	40

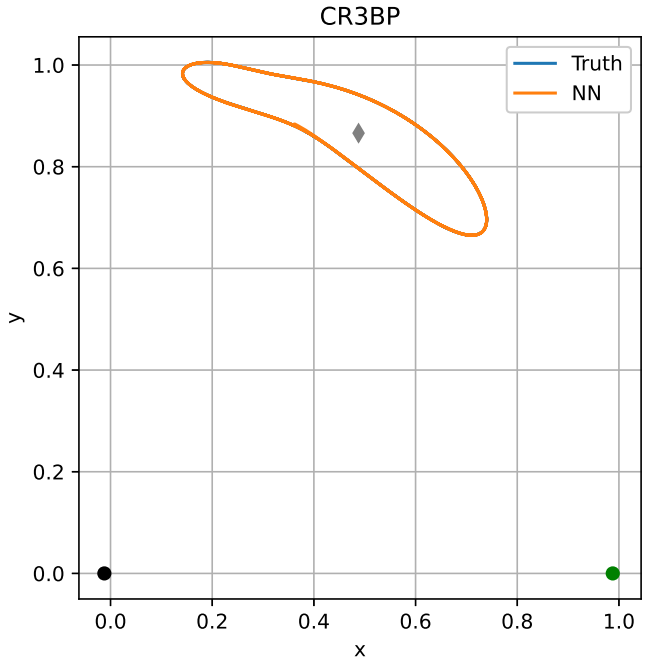

Figure 4.14: CR3BP Phase Space Solution

Jetson Nano Evaluation Speeds

Evaluation time to compute a series of independent dynamical propagations is an important metric that this propagation method excels in. For the perturbed two-body problem and CR3BP over four orbit periods, example problems from Figures 4.4 and 4.14, several forward propagations were performed and compared with a parallel RK45 integration and the network bundle solution. In order to get an accurate representation of total evaluation, the benchmark includes the time it takes for memory to be shared from the GPU to the CPU. Tables 4.6 and 4.7 compare the GPU parallelized RK45 to a network approximated solution evaluated on the GPU. In these tables, speed ratio refers to the ratio of evaluation time of RK45 to the network approximation. Finally, the solution accuracies of the neural network trial solution is compared to RK45 at max tolerance calculated by the Lloyd heuristic. These results over the entire orbit propagation for the individual

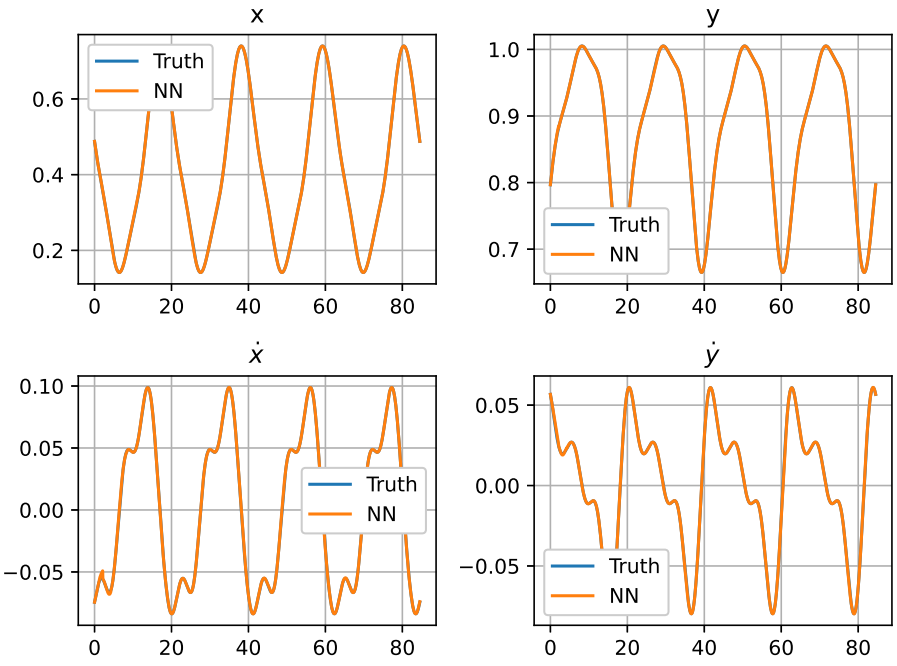

Figure 4.15: CR3BP - Solution Along Phase Space

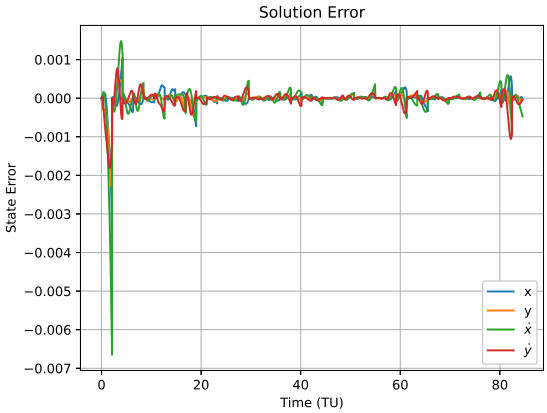

Figure 4.16: Solution Error

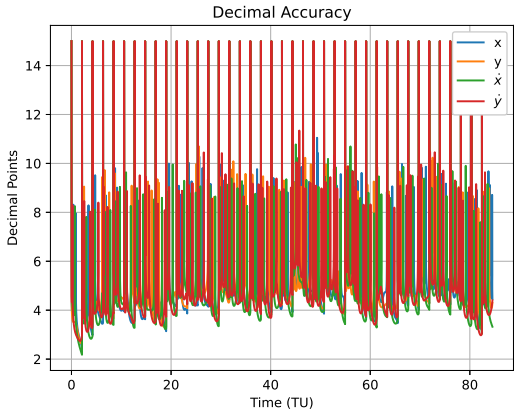

Figure 4.17: Decimal Accuracy

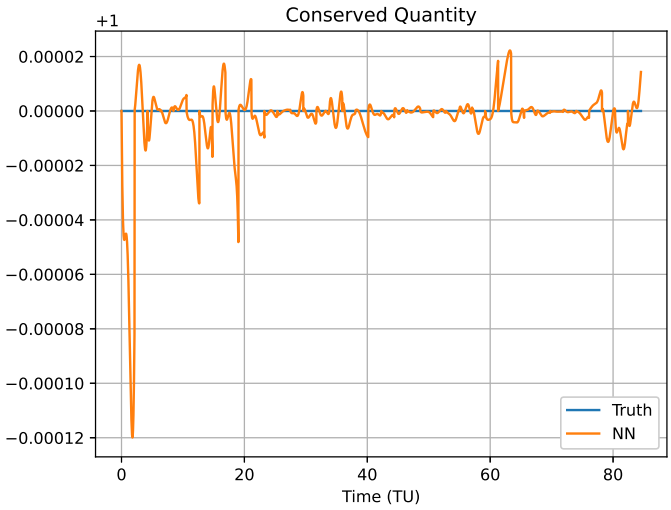

Figure 4.18: Conservation of Jacobi Constant

problems are shown in table 4.8.

Table 4.6: Parallelization of Forward Propagation on Jetson Nano - P2BP

ODE Propagations	RK45 Parallelized	Network Approximation	Speed Ratio
10	6.997697114944458 sec	0.06692218780517578 sec	104.5646794351
100	10.7991680702 sec	0.03540325164794922 sec	305.03322625806
1000	105.696928223 sec	0.06094169616699219 sec	1734.3942632212
10000	654.911970218 sec	0.12562012672424316 sec	5213.4318544005
100000	6711.9176555872 sec	0.15697121620178223 sec	42758.9071295671

One concern that this solution has is in the ability for a set of networks to learn the trajectory close to a central body. For example, the short period CR3BP example has large areas of inaccuracy when the solved trajectory approaches the central body. One possible explanation for this is due to how highly sensitive to the calculation for the Jacobi constant is. This high sensitivity creates instability in the loss calculation over the training epochs, which affects the ability for the network to learn the problem correctly. However, the use of the conservation of the Jacobi constant is

Table 4.7: Parallelization of Forward Propagation on Jetson Nano - CR3BP

ODE Propagations	RK45 Parallelized	Network Approximation	Speed Ratio
10	10.964469194412231 sec	1.8366117477416992 sec	5.9699439514
100	16.5226102273 sec	0.284290075302124 sec	58.1188429098
1000	156.2830433846 sec	0.15202951431274414 sec	1027.9783112581
10000	1621.6082319419 sec	0.15697121620178223 sec	10330.6088286745
100000	16759.9323411385 sec	0.2991485595703125 sec	56025.4489114436

Table 4.8: Parallelization of Forward Propagation on Jetson Nano - P2BP

Orbit Solution	Network Accuracy Compared to RK45
P2BP	4.188735704926352
CR3BP	4.528947255518326

integral to approximating the rest of the solution trajectory with high accuracy. This implies that for trajectories in which the relative distance between the two central bodies and the orbiting body is small, using this method is not best for calculating accurate approach trajectories. A few solutions to this problem could be increasing the complexity of the chosen network in these approach regions or subdividing the approach regions many more subdomains.

Comparing evaluation speeds of the neural network implementation with a parallel RK45 routine, an increase of up to 5 orders of magnitude was measured for up to 100000 numerical integrations. One reason evaluation speed of this technique is so low is because evaluating the network approximated solution does not require any intermediate state knowledge to reach a desired solution at a certain time period. Unlike a standard numerical integration routine in which the intermediate states of a solution are required to be calculated to find the solution, the neural network solution offloads this intermediate step as a pretraining computation, leaving a trained series of networks that can be evaluated at any time of interest near instantaneously.

One area that should be explored with more depth is the use of larger networks in solving problems

of this domain. This work uses shallow networks because these networks have a small amount of learnable parameters which in turn reduces the network size when deployed on low-powered hardware. A network that is comparable to this accuracy would require a much more complex structure. Due to the intended application of low-powered flight hardware, smaller and less complex network structures allow for faster forward evaluation speeds, but a more complex network structure will allow for great approximation accuracy.

CHAPTER 5: CONCLUSION AND FUTURE WORK

In this work, shallow neural networks were examined to identify network structures training parameters to solve orbit propagation faster when deployed on low-powered hardware mimicking the compute power of onboard satellite hardware. Identifying two major types of network learning, the supervised learning and unsupervised learning models, both network types are examined for their ability to approximate the solution to a nonlinear pendulum problem. The physics informed unsupervised model provided more accurate solutions and was demonstrated to solve the trajectories for the perturbed two body problem subject to drag and J2 perturbations, as well as the circular restricted three body problem. The Lagaris method of solving differential equations using unsupervised learning was augmented by introducing kinematic consistency among the neural networks as well as using integrals of motion during network training. In both cases, the PINN was able to approximate the trajectory of a satellite of approximately 4.5 decimals places in canonical units over multiple orbit periods with evaluation time decreases of up to 5 orders of magnitude.

The development of this research was motivated to improve complex on flight algorithms, and future work involves implementing a PINN to reduce the computational requirement of uncertainty quantification for deployment on spaceflight-like hardware. Early explorations into this using the supervised network model show that for simple harmonic oscillators, real calculation of the final probability density function (PDF) shows little deviation from the average PDF values evaluate at the same points, little deviation in the final cumulative distribution function of the calculated PDF, and execution of the uncertainty quantification script less than the time for propagation of the dynamical model, demonstrating the ability for neural network integrators to enable real time probability calculations through a physics based model. Table 5.1 and figures 5.1, 5.2, and 5.3 demonstrate the early results using Orthogonal Probability Approximation to quantify uncertainty. The goal is to apply the unsupervised network model to this algorithm due to its increased accuracy

and ability to approximate more complex differential equations.

Table 5.1: OPA Algorithm – Duffing Oscillator Evaluated with Different Integrators

Integrator	Execution Time	Final CDF Value	Average Error in PDF values
RK4	14.8432 sec	0.99999999724	–
NN Integrator	3.4668 sec	0.99255165928	0.009426

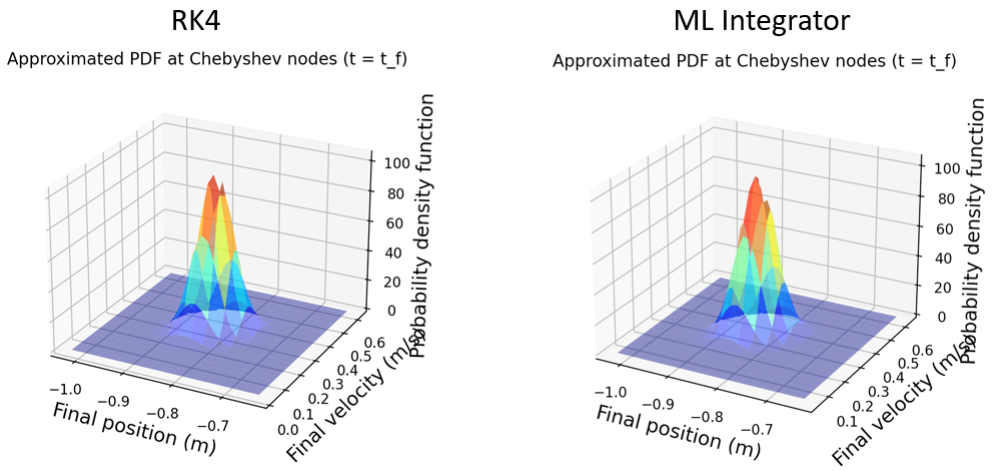

Figure 5.1: Comparison of PDFs Propagated with Different Integrators

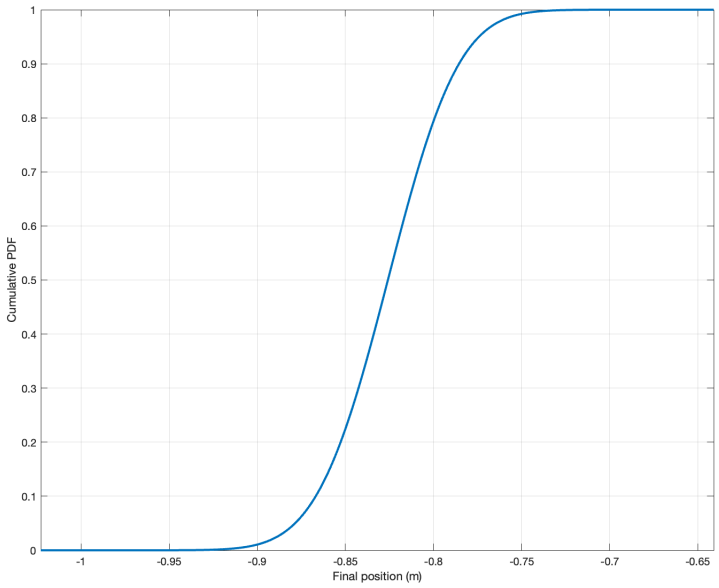

Figure 5.2: CDF Calculated by RK4

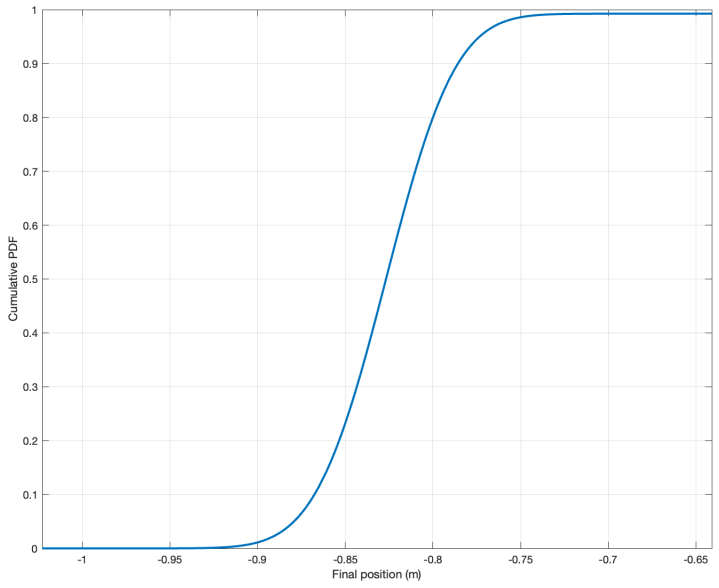

Figure 5.3: CDF Calculated by NN Integrator

LIST OF REFERENCES

[1] McCormack, D. & Drury, R. FireOPAL: Toward a Low-Cost, Global, Coordinated Network of Optical Sensors for SSA Phil Bland1, Greg Madsen2, Matt Bold3, Robert Howie1, Ben Hartig1, Trent Jansen-Sturgeon1, James Mason3. (2018)

[2] Vallado, D. Fundamentals of astrodynamics and applications. (Springer Science & Business Media,2001)

[3] Chesley, S. & Chodas, P. Asteroid close approaches: analysis and potential impact detection. *Asteroids III*. **55** pp. 69 (2002)

[4] Liu, K., Jia, B., Chen, G., Pham, K. & Blasch, E. A real-time orbit SATellites Uncertainty propagation and visualization system using graphics computing unit and multi-threading processing. *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. pp. 8A2-1 (2015)

[5] Zhu, J., Liu, Y., Bao, K., Chang, Y. & Wu, E. Realtime simulation of burning solids on GPU with CUDA. *2010 10th IEEE International Conference On Computer And Information Technology*. pp. 1219-1224 (2010)

[6] Patidar, S. & Narayanan, P. Ray casting deformable models on the gpu. *2008 Sixth Indian Conference On Computer Vision, Graphics & Image Processing*. pp. 481-488 (2008)

[7] Wodziński, M. & Krzyżanowska, A. Sequential classification of palm gestures based on A* algorithm and MLP neural network for quadrocopter control. *Metrology And Measurement Systems*. **24** (2017)

[8] Tichy, W., Münch, D. & Pankratius, V. Parallel programming with CUDA.

[9] Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*. **2**, 359-366 (1989)

[10] Guého, D., Singla, P. & Melton, R. Investigation of Different Neural Network Architectures for Dynamic System Identification: Applications to Orbital Mechanics. *Spaceflight Mechanics 2019*. pp. 1789-1803 (2019)

[11] Breen, P., Foley, C., Boekholt, T. & Zwart, S. Newton versus the machine: solving the chaotic three-body problem using deep neural networks. *Monthly Notices Of The Royal Astronomical Society*. **494**, 2465-2470 (2020)

[12] Quito Jr, M., Monterola, C. & Saloma, C. Solving N-body problems with neural networks. *Physical Review Letters*. **86**, 4741 (2001)

[13] Zheng, M., Luo, J. & Dang, Z. Machine learning-based solution of Kepler's equation. *Third International Conference On Computer Science And Communication Technology (ICCSCT 2022)*. **12506** pp. 1577-1583 (2022)

[14] Karam, M. & Zohdy, M. Modeling a simple inverted pendulum using a model-based dynamic recurrent neural network. *Proceedings Of The Thirty-Seventh Southeastern Symposium On System Theory, 2005. SSST'05.*. pp. 78-82 (2005)

[15] Nascimento, R., Fricke, K. & Viana, F. A tutorial on solving ordinary differential equations using Python and hybrid physics-informed neural network. *Engineering Applications Of Artificial Intelligence*. **96** pp. 103996 (2020)

[16] Flamant, C., Protopapas, P. & Sondak, D. Solving differential equations using neural network solution bundles. *ArXiv Preprint ArXiv:2006.14372*. (2020)

[17] Géron, A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. (" O'Reilly Media, Inc.",2019)

[18] Lagaris, I., Likas, A. & Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions On Neural Networks*. **9**, 987-1000 (1998)

[19] Lagaris, I., Likas, A. & Papageorgiou, D. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions On Neural Networks*. **11**, 1041-1049 (2000)

[20] Schiassi, E., D'Ambrosio, A., Drozd, K., Curti, F. & Furfaro, R. Physics-informed neural networks for optimal planar orbit transfers. *Journal Of Spacecraft And Rockets*. **59**, 834-849 (2022)

[21] Greydanus, S., Dzamba, M. & Yosinski, J. Hamiltonian neural networks. *Advances In Neural Information Processing Systems*. **32** (2019)

[22] Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D. & Ho, S. Lagrangian neural networks. *ArXiv Preprint ArXiv:2003.04630*. (2020)

[23] Lentaris, G., Maragos, K., Stratakos, I., Papadopoulos, L., Papanikolaou, O., Soudris, D., Lourakis, M., Zabulis, X., Gonzalez-Arjona, D. & Furano, G. High-performance embedded computing in space: Evaluation of platforms for vision-based navigation. *Journal Of Aerospace Information Systems*. **15**, 178-192 (2018)

[24] Yost, B., Weston, S., Benavides, G., Krage, F., Hines, J., Mauro, S., Etchey, S., O'Neill, K. & Braun, B. State-of-the-art small spacecraft technology. (2021)

[25] Guide, D. Cuda c programming guide. *NVIDIA, July*. **29** pp. 31 (2013)

[26] Lloyd, S., Irani, R. & Ahmadi, M. Using neural networks for fast numerical integration and optimization. *IEEE Access*. **8** pp. 84519-84531 (2020)

[27] Baum, E. & Haussler, D. What size net gives valid generalization?. *Advances In Neural Information Processing Systems*. **1** (1988)

[28] Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*. (2014)

[29] Kişi, Ö. & Uncuoğlu, E. Comparison of three back-propagation training algorithms for two case studies. (CSIR,2005)