AAS 18-427

LEARNING CAPABILITIES OF NEURAL NETWORKS AND KEPLERIAN DYNAMICS

Damien Guého*, Puneet Singla†, Robert G. Melton‡

Machine learning (ML) tools, especially deep neural networks (DNNs) have garnered significant attention in the last decade; however, it is not clear whether ML tools can learn the inherent characteristics of dynamical model (such as conservation laws) from the training data set. This paper considers the effectiveness of DNNs in learning dynamical system models by considering the Keplerian two-body problem. Training a DNN with data from a single revolution produces poor performance when predicting motion on subsequent revolutions. By incorporating deviations from constancy of angular momentum and total energy into the loss function for the DNN, predictive performance improves significantly. Further improvements appear when a richer training data set (generated from a number of orbits with different in orbital element values) is employed.

INTRODUCTION

Identification and control of nonlinear dynamical systems using neural networks were introduced by Narendra and Parthasarathy in 1990 [1] based on observed simulation results. With rapid progress, neural networks for identification of unknown dynamical systems [2] have been subjected to a broad and comprehensive consideration for long-term prediction capabilities. Data driven models have been used for centuries in dynamical system modeling especially in astrodynamics, where sometimes only input-output observations are available. Kepler's laws, Newton's laws of motion and Newton's gravitational law were developed with critical reliance on observational data. Some earlier efforts were focused on deriving appropriate models from data rather than fitting a model of fixed structure. The advent of ubiquitous computing solutions has led to unprecedented breakthroughs in pattern analysis and machine intelligence. Parallel developments in sensing technologies, microelectronics and embedded systems enable the acquisition of high precision data from physical systems to enable the machine learning (ML) approaches. As soon as large amounts of observed data are available, a model can be learned, similar to human cognition in learning from past experience to predict future events. There are different types of machine learning, with supervised learning, reinforcement learning, and unsupervised learning [3–5]. Machine learning methods have shown great capabilities for a wide range of applications such as improving orbit prediction accuracy through supervised machine learning [6]. While it may appear that the proliferation of data and coupled embedded systems are slated to supplant the physical models and their associated insights with data driven models and control systems that work based upon large training data sets, the stark reality is far from that view point. The main reason is that efforts are concentrated on developing efficient learning algorithms to fit a fixed structure model to a large amount of data rather than finding the optimal model structure. In this work, we study the quintessential question about approximation as well as prediction capabilities of ML tools, especially with Deep Neural Networks (DNN).We consider the unperturbed two-body problem to investigate the approximation and interpolation capabilities of NN-based methods. In this scenario, the underlying nonlinear dynamics will be realized by a

*Graduate Student, Department of Aerospace Engineering, Pennsylvania State University, State College, PA-16802, Email: djg76@psu.edu.

†Associate Professor, AIAA Associate Fellow, AAS Fellow, Department of Aerospace Engineering, Pennsylvania State University, State College, PA-16802, Email: psingla@psu.edu.

‡Professor, AIAA Associate Fellow, AAS Fellow, Department of Aerospace Engineering, Pennsylvania State University, State College, PA-16802, Email: rgmelton@psu.edu.

multi-layer neural network (also known as Deep Neural Network). We investigate whether the learned NN model can reproduce known constants of the motion (e.g. energy and angular momentum). While the training data set is generated by the solution of known two-body problem equations of motion, all the simulations are performed using the TensorFlow API [7] originally developed by researchers and engineers from the Google Brain team within Google's AI organization [8]. Because TensorFlow is an open source software library based on a strong computer algebra system, it allows high performance numerical computation thanks to a high level of abstraction.

It should be emphasized that this paper does not advocate the use of any ML tools to predict orbit states of resident space objects. This paper is an attempt to understand the approximation and prediction capabilities of a multi-layer neural network approach. The two-body problem is used here as an example due to its rich history. This paper is our first attempt to understand how incorporation of prior knowledge about system dynamics such as conservation laws affects the learning of a multi-layer neural network.

The structure of the paper is as follows: first, a brief introduction to neural networks is presented followed by a discussion of different training algorithms and computation of the loss function. Next, an introduction to the Keplerian two-body problem is presented followed by a discussion of neural network learning with an integrator in the loop. Numerical results are presented to test the learning of a multi-layer neural network by considering two different test cases. Finally, the paper concludes with a discussion of the results and some future research directions.

THE NEURAL NETWORK

Overall description

The first step toward artificial neural networks came in 1943 when W. McCulloch, a neurophysiologist, and W. Pitts, a mathematician, wrote an essay on how neurons might work. One approach focused on biological processes in the brain while the other focused on the application of neural networks to artificial intelligence. In recent decades, many types of neural networks have been extensively studied: multi-layer perceptrons, recurrent, convolutional, long-short term memory or modular neural networks. Among the various types of neural network structures that can be implemented, we focus on the most common architecture known as a multi-layer perceptron (MLP) neural network. Based upon two physical components (the processing elements called neurons or perceptrons, and the connection between them called links), the MLP neural network is organized in three different kinds of layers. The input layer is built with neurons that receive data from outside the network, the output layer with neurons whose outputs are used externally, and the hidden layers with neurons that receive and produce data internally. Typically, an MLP neural network consists of one input and one output layer which represent the input and the output of the overall network, respectively, and one or more hidden layers. All the layers are connected with links affected by weights (see Figure 1 for a schematic representation). This specific (and also widely used) structure is known as a feedforward neural network because the connections in the network flow forward from the input layer to the output layer without any internal feedback loops. A neuron is an elementary unit that plays the role of a mathematical function. The neuron receives one or more weighted inputs, sums them, adds a bias (that can be 0) and passes them through a non-linear function known as an activation function or transfer function. This thresholding function (inspired from logic gates in threshold logic) is bounded, differentiable and often monotonically increasing and continuous. For a given neuron α, considering p inputs $x_1, \cdots, x_p$ having weights $w_1, \cdots, w_p$, the output will be

$$y_\alpha = \phi\left(\sum_{i=1}^{p} w_i x_i + b\right), \tag{1}$$

with b the bias and ϕ the activation function. Among the most common activation functions (identity, logistic, radial-basis, tanh, arctan, rectified linear unit, softPlus...), we will choose the sigmoid function, such that

$$\phi(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}. \tag{2}$$

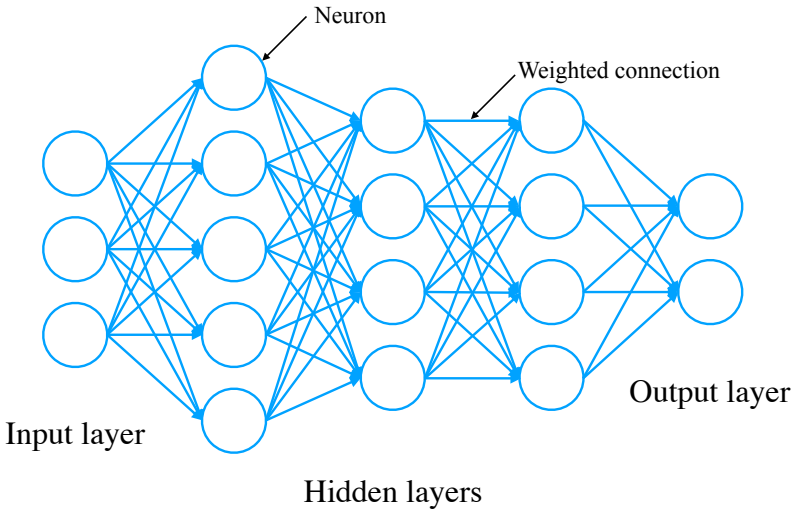


Figure 1: Generic MLP neural network with three hidden layers

For $n \in \mathbb{N}^*$, a *training data set* of size n is a set of pairs $(\mathbf{x}_k, \mathbf{y}_k)_{k\in[\![1,n]\!]}$ where $\mathbf{x}_k \in \mathbb{R}^r$ serves as input for the network and $\mathbf{y}_k \in \mathbb{R}^m$ is compared to the value $\tilde{\mathbf{y}}_k$ produced by the network to measure its fitness capability. The comparison is performed by computing the mean square error (MSE) for each pair k:

$$MSE(k) = \frac{1}{m}\sum_{i=1}^{m}\left(y_k(i) - \tilde{y}_k(i)\right)^2 = \frac{1}{m}\left|\left|\mathbf{y}_k - \tilde{\mathbf{y}}_k\right|\right|_2^2. \tag{3}$$

Finally, the sum of the MSE over the whole training set is called the *loss* and illustrate the proficiency of the network to map two different data sets (namely $\{\mathbf{x}_k\}$ and $\{\mathbf{y}_k\}$, $k \in [\![1, n]\!]$):

$$Loss = \sum_{k=1}^{n} MSE(k). \tag{4}$$

Training a neural network

Although the optimization community has studied the general problem of optimizing non-linear functions for many years, the multilayer neural networks do not represent a typical optimization problem. Gradient descent and its various variants such as conjugate gradient are often used for optimization of smooth nonlinear functions. However, many of these optimizers do not perform very well for the training of multi-layer neural networks as they have a tendency to get stuck in local minimum due to a fixed learning rate. Furthermore, many of these optimizers do not parallelize to run on GPUs or a distributed network and hence are computationally intensive for large networks. For the training of multi-layer neural networks, different variants of gradient descent algorithms have been developed which have an adaptive learning rate to avoid local minima and plateaus in the loss function. The most commonly used optimizers are: Adagrad [9], Adadelta optimizer [10], Adam optimizer [11], FtrlOptimizer [12], RMSprop [13] and NADAM [14]. These algorithms deploy momentum based methods and/or compute average of gradients to adjust the learning rate to avoid local minima. All of these optimizers are available with Google's TensorFlow package and have been optimized for parallel processing. In this paper, all the networks are trained using the Adam optimizer. The main feature of the Adam optimizer is that it computes adaptive learning rates for each parameter and also stores an exponentially decaying average of past gradients similar to Adadelta and RMSprop. Furthermore, it is well suited for problems that are large in terms of data and/or parameters (our training data sets can contain billions of entries).

Designing a qualified network

Creating a MLP neural network architecture therefore means proposing values for the number of hidden layers and the number of neurons in each of these layers*. The question on how to choose the number of hidden layers and neurons in a neural network has been addressed multiple times but there is no pragmatic technique that describes how to design an efficient neural network. It is also obscure whether there exists a unique optimal model for a given problem and there is different consensus regarding the impact on performance from adding additional hidden layers or increasing the number of neurons. However, designing a competent network architecture is relatively easy from the moment one has access to sufficient computation capabilities to test distinct configurations.

THE KEPLERIAN TWO-BODY PROBLEM

History

One of the most important problems in classical mechanics is the well known *two-body problem* which describes the motion of two point particles (or two uniform spherical masses) subject to a central force. Since Johannes Kepler first formulated the laws that describe planetary motion early in the 17$^{\text{th}}$ century, many scientists endeavored to solve for the equation of motion of the planets. Newton worked on the special case of the three-body problem and realized that any longitude on Earth could be determined knowing the Moon's position. Laplace and Poincaré, based on Euler's work, dealt with stability issues using series just as in the time of Newton and Leibniz and the invention of the calculus. While the general process to solve the two-body problem in the three spatial dimensions takes advantage of some of the most essential techniques in classical mechanics (decomposition of the dynamics, use of relative motion, the symmetries, reduction of the order of the system with conservation laws), celestial mechanics are the experimental laboratory for the discovery of new mathematics. Based on Giuseppe Piazzi's observations of Ceres (1801) [15], Gauss calculated the orbit of Ceres originally using only three points. With some approximations, Gauss found a nonlinear equation proportional to $1/r^2$. He initiated the theory of least squares and the premises of data driven models.

Formulation

Let $\mathbf{r}_1$ and $\mathbf{r}_2$ be the position vector of two bodies, and m_1 and m_2 be their mass. If $\mathbf{r} = \mathbf{r}_2 - \mathbf{r}_1$ is the relative position vector between the two bodies, the dynamics of the two-body problem is

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r}, \tag{5}$$

with $\mu = G(m_1 + m_2)$ and G is the universal gravitational constant.
In an inertial reference frame and using Cartesian coordinates, with $\mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ and $r = \sqrt{x^2 + y^2 + z^2}$, Eq. (5) can be written as

$$\begin{aligned} \ddot{x} &= -\frac{\mu x}{r^3}, \\ \ddot{y} &= -\frac{\mu y}{r^3}, \\ \ddot{z} &= -\frac{\mu z}{r^3}. \end{aligned} \tag{6}$$

From Eq. (5) and Eq. (6), we define the function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ as

$$\mathbf{f} : \mathbf{r} \mapsto \ddot{\mathbf{r}} \Leftrightarrow \mathbf{f} : \begin{bmatrix} x \\ y \\ z \end{bmatrix} \mapsto \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -\frac{\mu}{r^3} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{7}$$

*as well as a choice for the activation functions and the optimizer.

that contains the dynamics of the two-body problem. This second-order system of differential equations can be numerically solved using several types of integrators starting from the simple second-order Euler integration setup to fourth-, fifth- or higher-order Runge-Kutta integration schemes. Presently, the Dormand-Prince (RKDP for Runge-Kutta Dormand-Prince) method is the most commonly used explicit method for solving ordinary differential equations [16]. A member of the Runge-Kutta family of ordinary differential equations solvers, the Dormand-Prince algorithm uses six function evaluations to calculate fourth- and fifth-order accurate solutions. In this paper, any generation of a *true solution* is performed with the Dormand-Prince method of integration along with the model dynamics from Eq. (5) and Eq. (6).

Integrated solution from the neural network

The idea of this paper is to investigate whether a certain model of neural network has the potential to approximate the function $\mathbf{f}$ introduced in Eq. (7) and therefore to determinate if such a model has the capacity to interpret and portray the underlying dynamics embedded in some data set.
Consider the initial value problem as follows:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{F}(\mathbf{x}, t); \\ \mathbf{x}(t_0) &= \mathbf{x}_0, \end{aligned} \tag{8}$$

where $\mathbf{x} = \begin{bmatrix} \mathbf{r} & \dot{\mathbf{r}} \end{bmatrix}^T$ is the unknown state vector of time t that is to be approximated. $\mathbf{F} : \mathbb{R}^6 \to \mathbb{R}^6$ contains the dynamics $\mathbf{f}$ of the two-body problem in the six-dimensional space of position and velocity. A solution of the Eq. (8) is

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{F}(\mathbf{x}(\tau))\mathrm{d}\tau, \tag{9}$$

where a numerical integrator is used to compute the second term of Eq. (9).
Consider now $\tilde{\mathbf{F}} : \mathbb{R}^6 \to \mathbb{R}^6$ that contains a neural network to approximate the dynamics instead of the function $\mathbf{f}$ from Eq. (7). The neural network replaces the function $\mathbf{f}$. Denoted with a tilde $\tilde{\mathbf{x}}(t)$ to indicate that the solution comes from a neural network approximation, Eq. (9) becomes

$$\tilde{\mathbf{x}}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \tilde{\mathbf{F}}(\tilde{\mathbf{x}}(\tau))\mathrm{d}\tau. \tag{10}$$

While the neural network takes over the dynamical model, a well chosen integrator will then have the role to finally integrate the general solution. The integrator implemented in this paper is a Runge-Kutta fixed step-size algorithm. Subsequently, the accuracy of the neural network is measured by calculating the difference between the integrated solution using the neural network model and the true solution with the true dynamics $\mathbf{F}$ and a Dormand-Prince integration algorithm. Figure 2 displays how the solution from the neural network and the true solution are generated to compute the loss function.

Expression of the Loss function

Overall, since the Loss is an indicator of the fitness capabilities of the network, it will compare the true value of the states with the approximations provided by the network. Based on Eq. (4), the Loss function is

$$Loss = \sum_{k=1}^{n} \left(\frac{1}{3} ||\mathbf{r}_k - \tilde{\mathbf{r}}_k||_2^2 + \frac{1}{3} ||\mathbf{v}_k - \tilde{\mathbf{v}}_k||_2^2 \right). \tag{11}$$

Taking into account the Constants of the Motion: When approximating the dynamics of the unperturbed two-body problem (e.g. conservative system), the constants of the motion can be seen as constraints. Constraints can be either hard constraints, which set some conditions for the model that are required to be satisfied, or soft constraints, which results in penalizing the loss function if a deviation from constancy is observed. The unperturbed two-body problem is a constrained problem and the phase of training the neural network by taking into account the constants of the motion as soft constraints becomes a constraint optimization problem. In order to evaluate the prediction accuracy gained by considering these constraints, we provide the

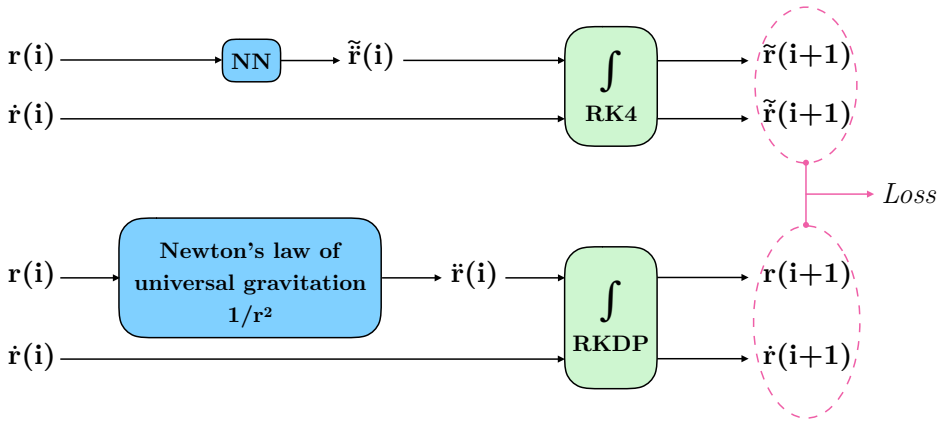


Figure 2: How the neural network is used: the upper part illustrates Eq. (10) with the neural network used to approximate the dynamics along with the Runge-Kutta fixed-size step algorithm while the bottom part is the classical generation of the true solution with the known dynamics and a Dormand-Prince integration algorithm. The loss is calculated (Eq. (3) and Eq. (4)) after generating the two outputs.

network with the context of inherent dynamics by incorporating a penalty term in the loss function corresponding to violations of conservation of angular momentum as well as total energy. Given an initial state $\mathbf{x}_0 = \begin{bmatrix}\mathbf{r}_0 & \mathbf{v}_0\end{bmatrix}^T$, we define initial angular momentum, $\mathbf{h}_0 = \mathbf{r}_0 \times \mathbf{v}_0$ and initial energy, $e_0 = v_0^2/r_0 - \mu/r_0$. Now, the loss function can be modified as follows to account for conservation of angular momentum and energy:

$$Loss = \sum_{k=1}^{n} \left(\frac{1}{3}||\mathbf{r}_k - \tilde{\mathbf{r}}_k||_2^2 + \frac{1}{3}||\mathbf{v}_k - \tilde{\mathbf{v}}_k||_2^2 \underbrace{+\frac{1}{3}||\mathbf{h}_0 - \tilde{\mathbf{h}}_k||_2^2 + ||e_0 - \tilde{e}_k||_2^2}_{\text{soft constraints}} \right), \tag{12}$$

with $n \in \mathbb{N}^*$ the size of the training data set.

Runge-Kutta integration scheme with neural network

For any step-size $h > 0$ and $\forall n \in \mathbb{N}$, we define $\mathbf{x}_{n+1}$ as the RK4 (Runge-Kutta 4$^{\text{th}}$-order) approximation of $\mathbf{x}(t_{n+1})$ by

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right), \tag{13}$$

with

$$\begin{aligned}
\mathbf{k}_1 &= h\mathbf{F}(\mathbf{x}_n, t_n); \\
\mathbf{k}_2 &= h\mathbf{F}\left(\mathbf{x}_n + \frac{\mathbf{k}_1}{2}, t_n + \frac{h}{2}\right); \\
\mathbf{k}_3 &= h\mathbf{F}\left(\mathbf{x}_n + \frac{\mathbf{k}_2}{2}, t_n + \frac{h}{2}\right); \\
\mathbf{k}_4 &= h\mathbf{F}\left(\mathbf{x}_n + \mathbf{k}_3, t_n + h\right).
\end{aligned} \tag{14}$$

One can reformulate the two-body problem and write the dynamics using a pseudo-matrix form

$$\mathbf{F}(\cdot) = \begin{bmatrix} \mathbf{0}_{3\times3}(\cdot) & \mathbf{I}_{3\times3}(\cdot) \\ \mathbf{f}(\cdot) & \mathbf{0}_{3\times3}(\cdot) \end{bmatrix} \tag{15}$$

such that

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, t) \Leftrightarrow \begin{bmatrix} \dot{\mathbf{r}} \\ \ddot{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3}(\cdot) & \mathbf{I}_{3\times3}(\cdot) \\ \mathbf{f}(\cdot) & \mathbf{0}_{3\times3}(\cdot) \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \dot{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{r}} \\ \mathbf{f}(\mathbf{r}) \end{bmatrix}, \tag{16}$$

where the matrix product has to be seen as a composition. Because we want the dynamics to be approximated by a neural network, Eq. (16) becomes

$$\dot{\mathbf{x}} = \tilde{\mathbf{F}}(\mathbf{x}, t) \Leftrightarrow \begin{bmatrix} \dot{\mathbf{r}} \\ \ddot{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3}(\cdot) & \mathbf{I}_{3\times3}(\cdot) \\ \mathbf{NN}(\cdot) & \mathbf{0}_{3\times3}(\cdot) \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \dot{\mathbf{r}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{r}} \\ \mathbf{NN}(\mathbf{r}) \end{bmatrix}, \tag{17}$$

where $\mathbf{f}$ has been replaced by the neural network model ($\mathbf{NN}$). Now, for any $h > 0$ and $n \in \mathbb{N}$, we define the pairs $(\mathbf{p}_i, \mathbf{q}_i)_{1\leq i\leq 4} \in \mathbb{R}^3 \times \mathbb{R}^3$ as

$$\begin{aligned}
\mathbf{p}_1 &= h\dot{\mathbf{r}}_n; \\
\mathbf{q}_1 &= h\mathbf{NN}(\mathbf{r}_n); \\
\mathbf{p}_2 &= h\left(\frac{\mathbf{q}_1}{2} + \dot{\mathbf{r}}_n\right); \\
\mathbf{q}_2 &= h\mathbf{NN}\left(\frac{\mathbf{p}_1}{2} + \mathbf{r}_n\right); \\
\mathbf{p}_3 &= h\left(\frac{\mathbf{q}_2}{2} + \dot{\mathbf{r}}_n\right); \\
\mathbf{q}_3 &= h\mathbf{NN}\left(\frac{\mathbf{p}_2}{2} + \mathbf{r}_n\right); \\
\mathbf{p}_4 &= h\left(\mathbf{q}_3 + \dot{\mathbf{r}}_n\right); \\
\mathbf{q}_4 &= h\mathbf{NN}\left(\mathbf{p}_3 + \mathbf{r}_n\right).
\end{aligned} \tag{18}$$

Finally, the RK4 approximation of $\mathbf{x}(t_{n+1})$ using the neural network model as a substitute for the dynamics (written with a tilde $\tilde{\mathbf{x}}(t_{n+1})$ to specify that it comes from the neural network) is

$$\tilde{\mathbf{x}}(t_{n+1}) = \tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + \frac{1}{6}\left(\begin{bmatrix} \mathbf{p}_1 \\ \mathbf{q}_1 \end{bmatrix} + 2\begin{bmatrix} \mathbf{p}_2 \\ \mathbf{q}_2 \end{bmatrix} + 2\begin{bmatrix} \mathbf{p}_3 \\ \mathbf{q}_3 \end{bmatrix} + \begin{bmatrix} \mathbf{p}_4 \\ \mathbf{q}_4 \end{bmatrix}\right). \tag{19}$$

Figure 3 presents an expanded view of the Runge-Kutta integration scheme implemented along with the neural network. As a fourth-order integrator, the Runge-Kutta algorithm makes use of the neural network at each integration step four times.

NUMERICAL RESULTS

In this section, we discuss the approximation of two-body Keplerian dynamics with a multi-layer NN coupled with a fourth order RK integrator. To assess the performance of the NN in learning Keplerian dynamics, we consider two test cases. The first test case involves the training of the NN with data corresponding to one particular orbit and the second test case corresponds to training with data involving multiple orbits sampled from a range of orbit parameters. For both test cases, we consider the effect of including violations of constants of the motion (e.g. angular momentum and energy conversation) in the loss function during the NN training.

Test Case 1: Training Over a Specific Orbit

This first test case involves training with a reference near-circular Low-Earth Orbit (LEO) with semi-major axis and eccentricity

$$a = 7172490\text{m} \quad \text{and} \quad e = 0.0011.$$

The initial conditions for the reference orbit in Cartesian space are given as follows:

$$\begin{aligned}
&x_0 = 757700\text{m},\ y_0 = 5222607\text{m},\ z_0 = 4851500\text{m}; \\
&\dot{x}_0 = 2213.21\text{m}\,\text{s}^{-1},\ \dot{y}_0 = 4678.34\text{m}\,\text{s}^{-1},\ \dot{z}_0 = -5371.30\text{m}\,\text{s}^{-1}.
\end{aligned}$$

A RK4 integrator with 0.01s time step is used to predict position and velocity vectors over an orbit while using a NN model for two-body dynamics approximation and true initial conditions. True orbit data is assumed to be available every 0.01s to compute the loss function.

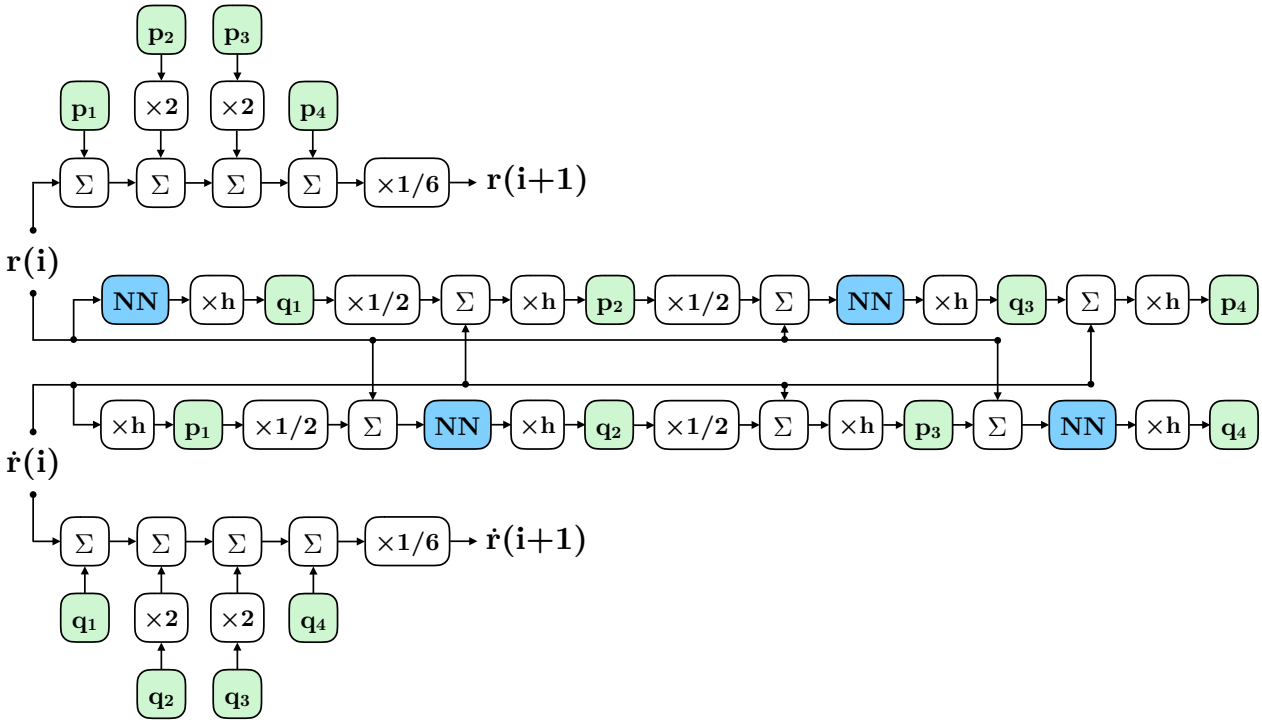
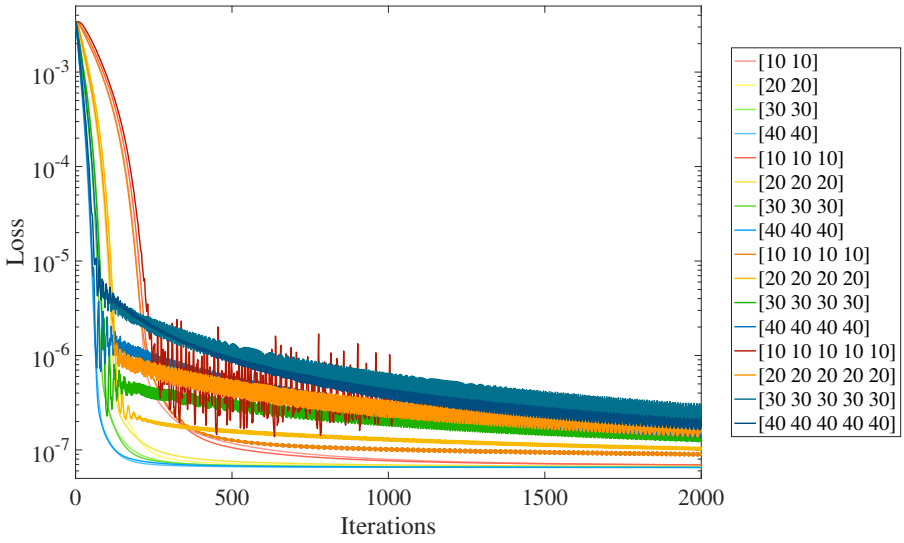


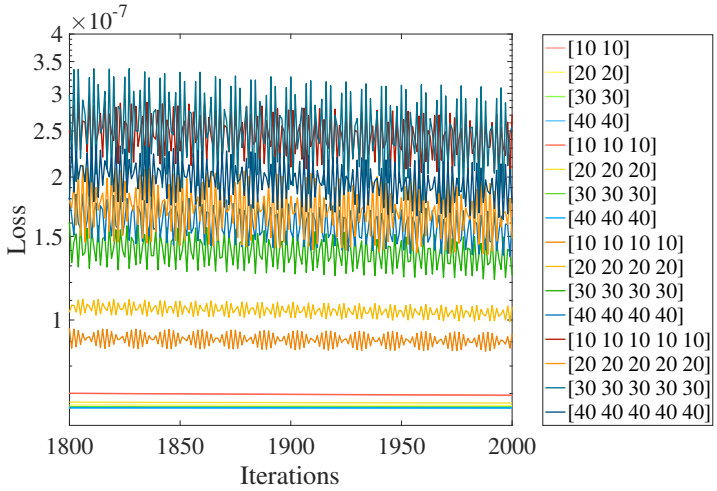
Figure 3: 4[th] order Runge-Kutta integration scheme using a neural network model to approximate the dynamics.

The Neural Network Architecture: To assess the effect of different NN architectures (layers and neurons per layer) on the approximation accuracy, we consider sixteen distinct configurations with two to five layers and the number of neurons varying from ten to forty per hidden layer. For two thousand iterations (or epochs e.g. the number of times the entire training data set will flow through the network), we examine closely the progression of the loss function and more particularly the monotonicity, its final value and the rate of convergence. Figure 4(a) presents the evolution of the loss function for the sixteen different configurations. Although all the architectures manage to provide an acceptable network, the loss function does not converge for all considered NN architectures, and varies from configuration to configuration. Networks with four layers or more (eight configuration total as seen in Figure 4(b)) rapidly start oscillating. This non-monotonicity implies that the optimizer is not able to update the weights at each iteration to continuously decrease the overall value of the loss function. This can happen if there are too many parameters to optimize: some updated weights or bias values result in a smaller loss while others are responsible for an increase. This analysis allows us to draw two conclusions: 1. more layers and/or more neurons per layer don't always mean better accuracy; 2. optimizers with neural networks that are too deep may not converge monotonically (even if they do not diverge), and end up with a fuzzy behavior. Based upon these results, we use a NN with three layers and thirty neurons per hidden layer.

Figure 5(a) shows the evolution of the loss function (from Eq. (11)) for three layers with thirty neurons per layer NN [30-30-30] trained with orbital data for 1.6 orbit time period. It is apparent that the loss function reaches a plateau around 3×10^{-8} after approximately 600 epochs. Figure 5(b) shows the true and NN approximated reference orbit plots for 4 revolutions. Figures 5(c) and 5(d) show the norm of the approximation error for position and velocity variables, respectively. Similarly, Figure 5(e) shows the plot of error in reproducing constants of the motion, i.e., angular momentum and total energy. The vertical red line corresponds to the end of the training time period. From these plots, it is clear that although the approximation error is small for the training time period, the prediction accuracy of the NN approximated orbit dynamic

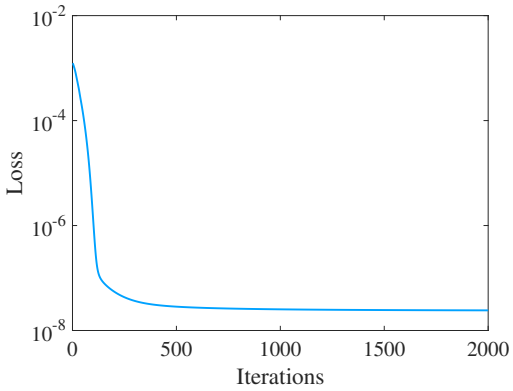


(a) Loss Function Evolution

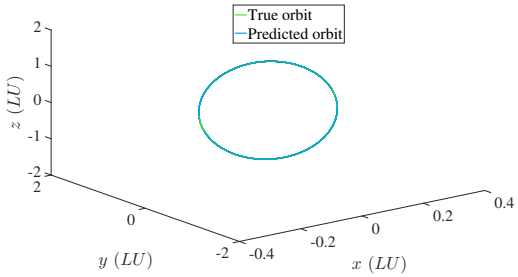


(b) Loss Function Evolution (Zoomed Version between epochs 1800 and 2000)

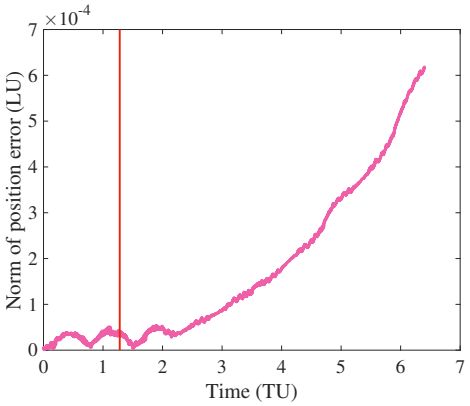
Figure 4: Evolution of the loss function for different network architectures.



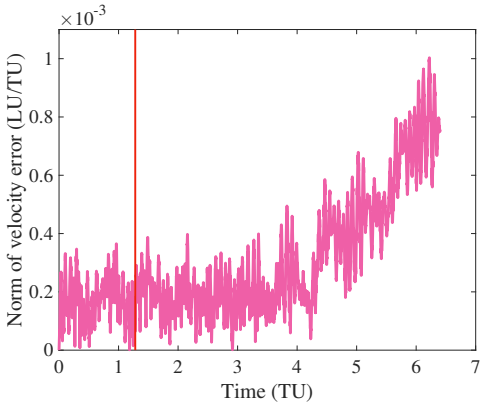
(a) Loss Function Evolution



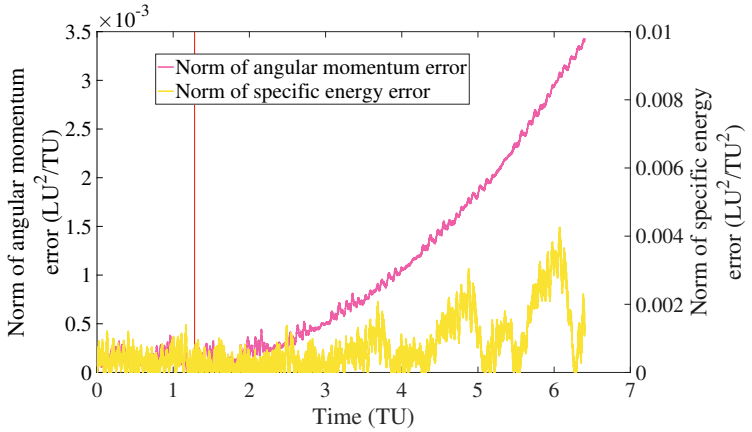
(b) True and NN Approximated Orbits



(c) Norm of position error vs. time



(d) Norm of velocity error vs. time



(e) Constant of the Motion Violation

Figure 5: Performance of NN over Test Data involving Time Prediction for Test Case 1

model deteriorates rapidly over time.

The accuracy of the same NN-based model is even worse when it is tasked to predict the motion of another orbit with semi-major axis and eccentricity

$$a = 7791108\text{m} \quad \text{and} \quad e = 0.0982,$$

(cf. Figure 6). From these results, we can conclude that the NN approximated dynamic model is unable to learn the inherent characteristics of Keplerian dynamics, i.e., conservation of energy and angular momentum and hence resulting in poor prediction capabilities.

The evolution of the loss function (from Eq. (12)) is shown in Figure 7(a). Notice that the converged value of the loss function is larger than the converged loss function in earlier training due to addition of the new terms $||\mathbf{h}_0 - \tilde{\mathbf{h}}_k||_2^2$ and $||e_0 - \tilde{e}_k||_2^2$. However, the larger value of penalty function does not necessarily mean a poor approximation capability of the NN model. Figures 7(c) and 7(d) show the norm of the approximation error for position and velocity variables, respectively. The vertical red line still corresponds to the end of the training data set. Although the approximation accuracy decreases with time prediction, the prediction accuracy of the NN learned model has increased by an order of magnitude as compared to results presented in Figures 5(c) and 5(d). This increase in prediction accuracy can be attributed to inclusion of constants of the motion during the training of NN. This observation is once again confirmed by the plot of constant of the motion violation in Figure 7(e). From these results, we can conclude that inclusion of a penalty term in the loss function corresponding to violation of constants of the motion helps with NN training. However, including constants of the motion as soft constraints in the loss function does not improve significantly the NN-based model accuracy when the prediction is performed on an unknown orbit. Figure 8 proves that this enhanced training has not produced acceptable prediction capabilities for the NN model.

Test Case 2: Training on a Set of Orbits

To further improve the accuracy of the NN-approximated orbit model, we consider the training data comprising one revolution for ten different orbits (Figure 9 shows the ten orbits considered for the NN training).

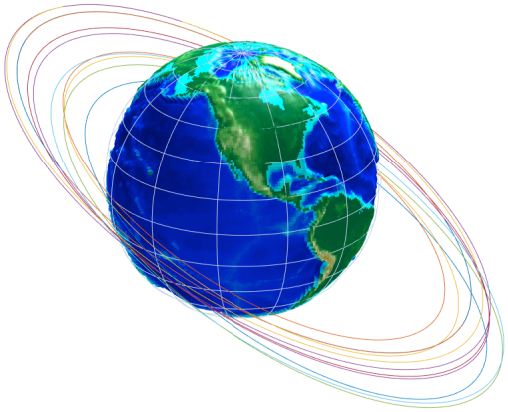
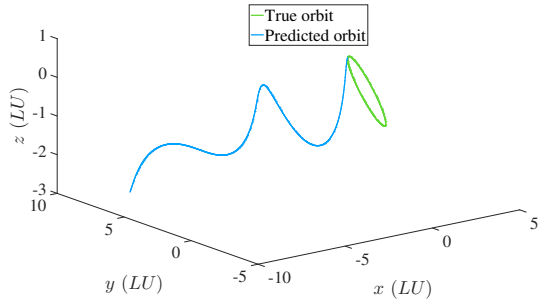


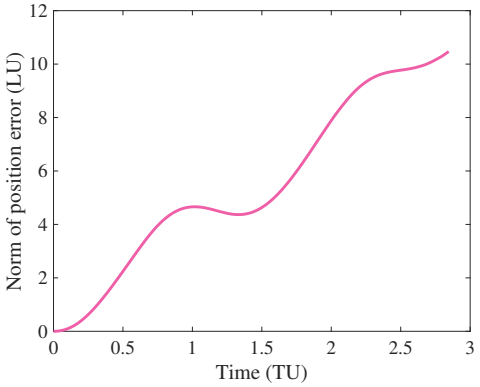
Figure 9: A Schematic of 10 Different Orbits used for NN Training in Test Case 2.

The data gathered from the ten orbits are stacked in a single set to produce a large training array consisting essentially of inputs with ten different initial conditions. Table 1 below lists the orbital elements for all 10 orbits.

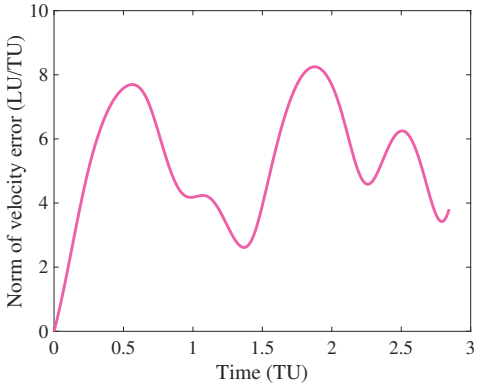
It should be noted that the orbital elements considered correspond to all 10 orbits being coplanar. Also, for validation and testing, we consider three more orbits with the same constraints in the orbital elements as listed in Table 1.



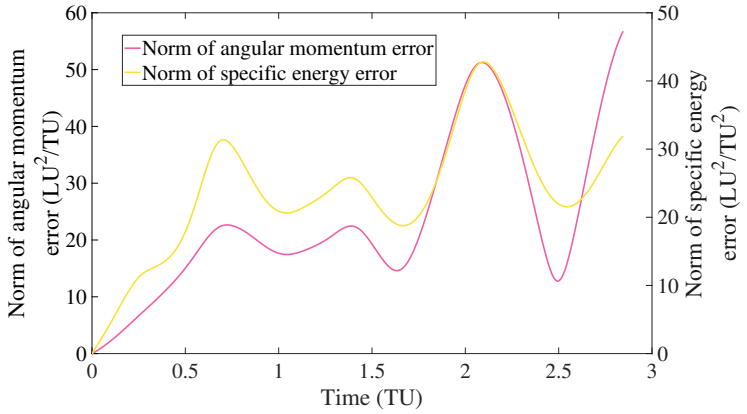
(a) True and NN Approximated Orbits



(b) Norm of position error vs. time

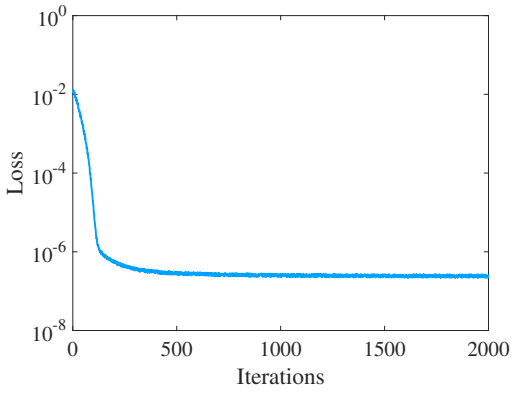


(c) Norm of velocity error vs. time

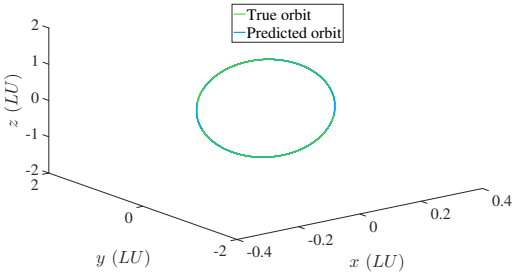


(d) Constants of the Motion Violation

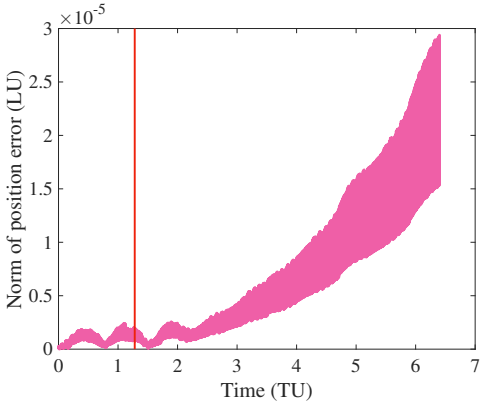
Figure 6: Performance of NN over Test Data Involving Another Orbit for Test Case 1



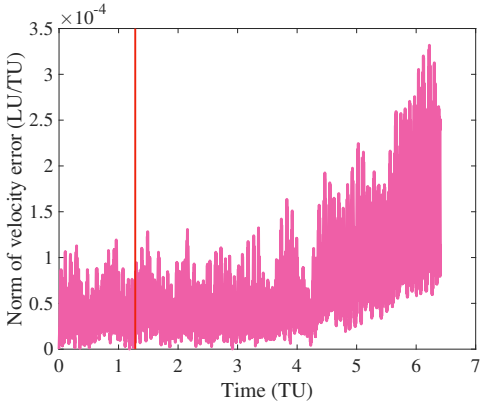
(a) Loss Function



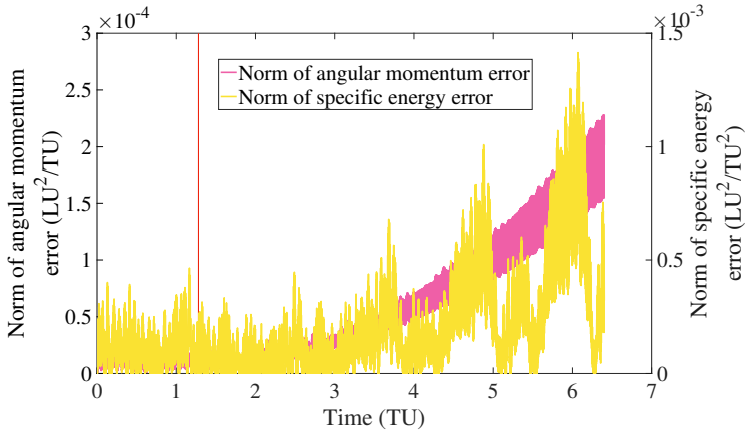
(b) True and NN Approximated Orbit



(c) Norm of Position Error vs. Time

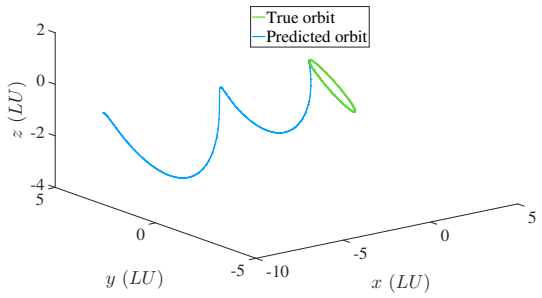


(d) Norm of velocity Error vs. Time

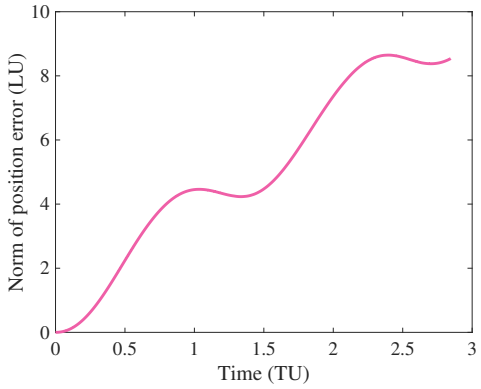


(e) Constant of the Motion Violation

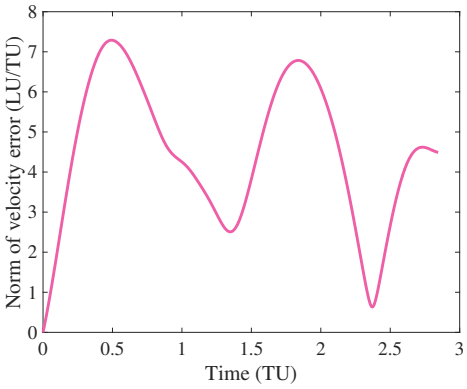
Figure 7: Performance of NN by Including Constants of the Motion in Loss Function for Test case 1.



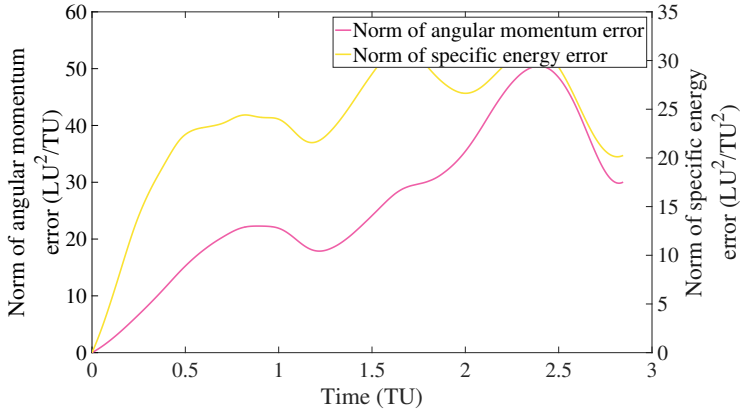
(a) True and NN Approximated Orbits



(b) Norm of position error vs. time



(c) Norm of velocity error vs. time



(d) Constants of the Motion Violation

Figure 8: Performance of NN over Test Data Involving Another Orbit Including Constants of the Motion in Loss Function for Test Case 1

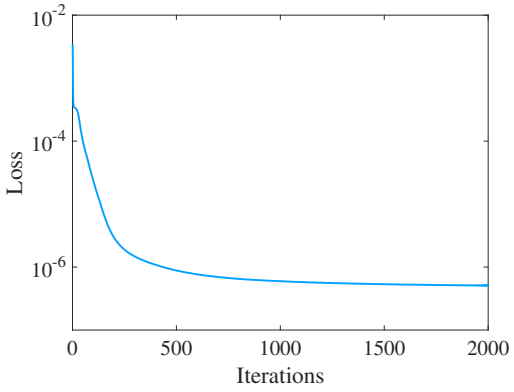
Table 1: Orbital elements for the ten training orbits

Semi-major axis a	$10000\text{km} \leq a \leq 13000\text{km}$
Eccentricity e	$0 \leq e \leq 0.4$
Inclination i	$i = \pi/6$
RAAN Ω	$\Omega = \pi/3$
Argument of perigee ω	$\omega = \pi/4$
True anomaly θ	$\theta = \pi/2$

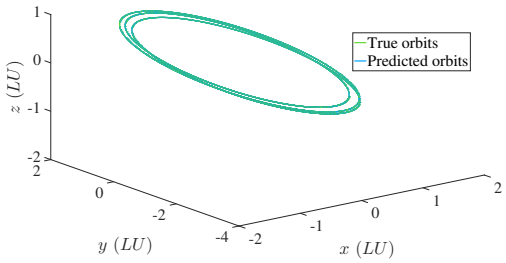
Figure 10(a) shows the plot of loss function evolution during the NN training. As expected, the loss function decreases continuously and converges to approximately 5×10^{-7} after 600 epochs. Because of the very large set of data, the value of the loss function is consequently larger than the converged loss function values in the previous test case. Figures 10(c) and 10(d) show the approximation error for position and velocity states for three orbits not included in the training. In position, the absolute error between the neural network prediction and the true value does not surpass $3 \times 10^{-7} LU$ with a mean below $10^{-7} LU$. Similarly, the norm of velocity approximation error does not exceed $1.2 \times 10^{-6} LU/TU$ with a mean close to $2 \times 10^{-7} LU/TU$. Finally, Figures 10(e) and 10(f) show the norm of error in conservation of angular momentum and energy, respectively. The better performance of the NN in learning the orbit dynamics as compared to the previous Test Case can be attributed to the richness of the training data set. In addition, Figure 11 shows the enhanced training by including constraints violation in the loss function. Similarly as before, the precision accuracy is one order of magnitude better. In position, the absolute error between the neural network prediction and the true value does not exceed $1.5 \times 10^{-8} LU$ with a mean below $4 \times 10^{-9} LU$. The norm of velocity approximation error does not exceed $4 \times 10^{-7} LU/TU$ with a mean around $5 \times 10^{-8} LU/TU$. Because the integration error is close to $10^{-10} LU$ in position and $10^{-9} LU/TU$ in velocity, these plots confirm for the first time the ability of the NN model to learn the orbit dynamics.

CONCLUSION

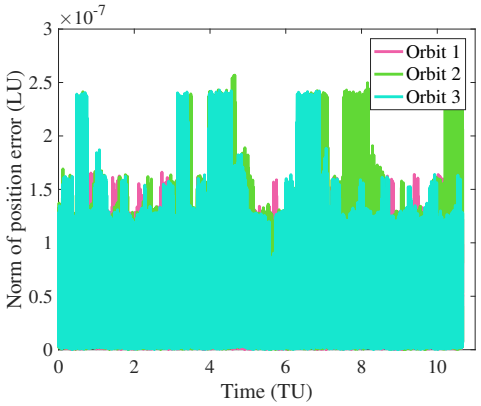
In this paper, we have investigated the learning capabilities of neural networks based on the well known Keplerian two-body problem. The goal was to examine whether the specific structure of neural networks could learn the inherent dynamics of two-body problem and examine whether neural network learned model can reproduce well-known characteristics of Keplerian dynamics such as conservation of energy and angular momentum. We consider two test cases to assess the learning capability of the converged NN. Although NN approximation errors are very small for a training data set involving the prediction of a specific orbit, the approximation accuracy deteriorates rapidly for the test data set. The NN approximation completely breaks down when it is tasked to predict an orbit not included in the training data set. Furthermore, the NN-learned model performs very poorly in reproducing constants of the motion. However, the accuracy of the NN-learned model is increased by an order of magnitude when one includes the penalty term in loss function formulation corresponding to violation of constants of the motion during the NN training. Furthermore, the performance of the NN-learned model increases considerably when the training data set is made richer by training the neural network over a set of orbits rather than a specific orbit. Although it seems that the NN-learned model can be trained to approximate Keplerian dynamics to a good accuracy, the complexity of the learned model is still an issue to be investigated. The resulting NN model is a profligate model for Keplerian dynamics as compared to Newton's law of gravitation. Hence, future research direction will first be concentrated on investigating a different architecture for the NN-based model such as a recursive model to take into account the time dependency of the input data. There are also many other paramters that can be tuned to adjust the training of the NN model such as other activation functions. Moreover, since the key element for the training process is the optimizer, we will also concentrate our efforts to design and select the best optimizer algorithms



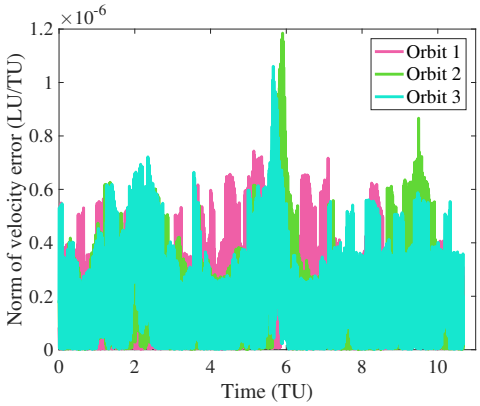
(a) Loss Function Evolution



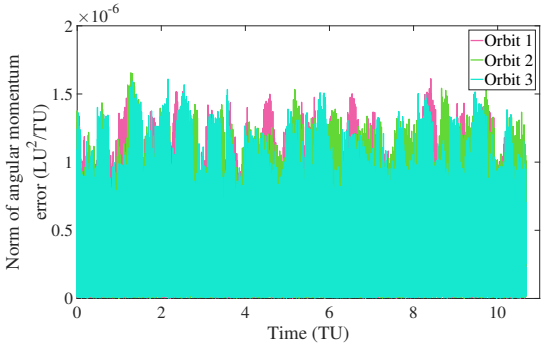
(b) True and NN Approximated Orbit



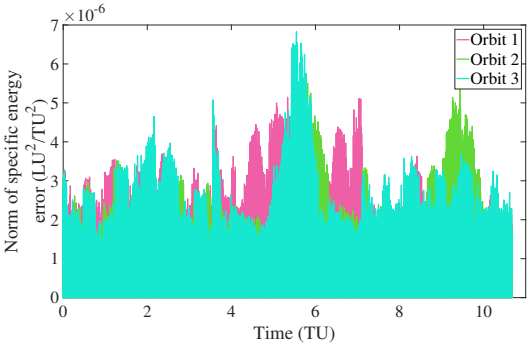
(c) Norm of Position Error vs. Time



(d) Norm of Velocity Error vs. Time

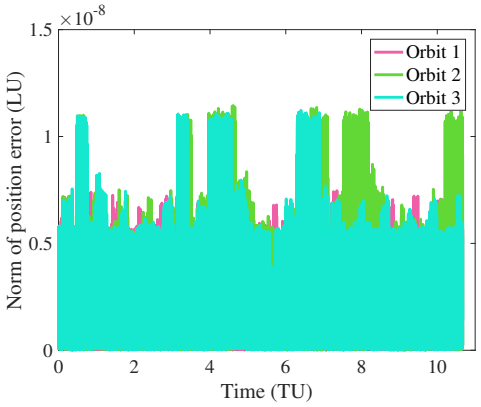


(e) Angular Momentum Constraint Violation

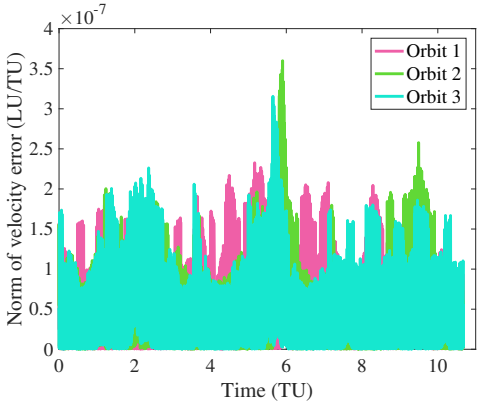


(f) Total Energy Constraint Violation

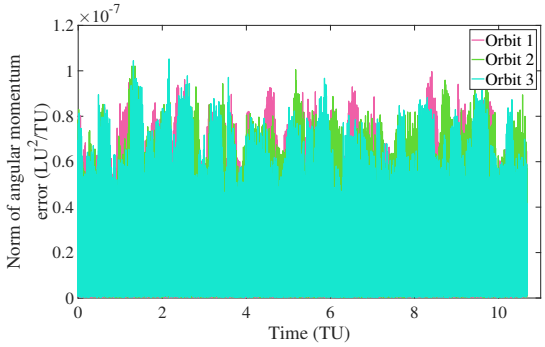
Figure 10: Training and Performance of NN for Test Case 2.



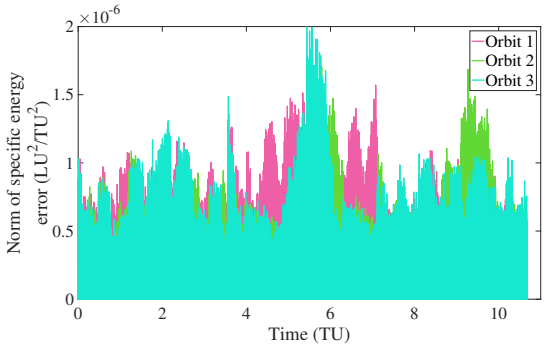
(a) Norm of Position Error vs. Time



(b) Norm of Velocity Error vs. Time



(c) Angular Momentum Constraint Violation



(d) Total Energy Constraint Violation

Figure 11: Performance of NN Including Constants of the Motion in Loss Function for Test Case 2.

for this application. Eventually, the ultimate goal will be to develop a parsimonious NN model of Keplerian dynamics and evaluate the complexity of this model to the two-body dynamics.

ACKNOWLEDGMENT

This material is based upon work supported jointly by the AFOSR grants FA9550-15-1-0313 and FA9550-17-1-0088.

REFERENCES

[1] Kumpati S. Narendra and Kannan Parthasarathy. Neural networks and dynamical systems. *International Journal of Approximate Reasoning*, 1992.

[2] Yi-Jen Wang and Chin-Teng Lin. Runge-kutta neural network for identification of dynamical systems in high accuracy. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 9(2), 1998.

[3] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[4] Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[5] C. M. Bishop. *Pattern Analysis and Machine Intelligence*. Springer, New York, 2004.

[6] Xiaoli Bai Hao Peng. Improving orbit prediction accuracy through supervised machine learning. 2018.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[8] TensorFlow. An open source machine learning framework for everyone.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[10] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[13] G. Hinton, N. Srivastava, and K. Swersky. rmsprop, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[14] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[15] Giorgia Foderà Serio, Alessandro Manara, Piero Sicoli, and William Frederick Bottke. *Giuseppe Piazzi and the discovery of Ceres*. University of Arizona Press, 2002.

[16] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.