Modeling and Analysis of Spacecraft Trajectories using Neural Networks for Implementation into Low-Thrust Control

A Dissertation Proposal

prepared for

the Faculty of College of Engineering & Applied Sciences

Western Michigan University

In Partial Fulfillment of

the Requirements for the Degree

Doctor of Philosophy in Mechanical Engineering

prepared by

Daniel Kolosa

September 27, 2017

Summary

This proposal studies the modeling of proximity maneuvers of a chaser spacecraft relative to a target using a feed-forward neural network. The orbital proximity uses the Clohessy-Wiltshire equations to model the relative position and velocity of the chaser vehicle relative to a target vehicle over a set maneuver time. Using supervised learning, a set of three neural networks are trained and tested against multiple orbit proximity maneuvers to demonstrate using neural networks for modeling orbital proximity problems.

For the dissertation, a neural network will be used as part of modeling and controlling low-thrust trajectories. The neural network structure will be optimized using a genetic algorithm.

Contents

NOMENCLATURE

a	= semi-major axis
a_{thrust}	= thrust acceleration
E	= eccentric anomaly
e	= eccentricity
e_{hidden}	= hidden layer error
e_{out}	= output layer error
e_{weight}	= weight error
F	= thrust acceleration
F_R	= radial thrust acceleration
F_S	= normal thrust
F_W	= circumferential thrust acceleration
G	= gravitational constant
$\boldsymbol{H}$	= angular momentum
I	= Input vector
i	= inclination
$\boldsymbol{K}$	= unit vector
M	= mean anomaly
$\dot{m}$	= mass flow rate
m_1	= mass of central body
$\boldsymbol{N}$	= unit vector points to direction of the ascending node
n	= mean motion

n_{points} = number of data points

O_j = output

$\boldsymbol{r}$ = position vector

T = orbit period

t_n = True value

t_p = time of perigee passage

$\boldsymbol{v}$ = velocity vector

w_{jk} = weight

x = input

y = output

$\hat{y}$ = estimated output

α = learning rate

α^r = thrust direction

Δv = velocity change

δu = relative velocity x

δv = relative velocity y

δw = relative velocity z

δx = relative position x

δy = relative position y

δz = relative position z

ϵ = specific orbital energy

Ω =right ascension longitude of the ascending mode

ω = argument of perigee

μ = standard gravitational parameter of the Earth

ϕ = angle between thrust for orbit transfers

θ = Mean anomaly

θ_r = direction of position relative to a plane

1 Introduction

The fundamental equations of low-thrust spacecraft trajectory dynamics are nonlinear. The goal of this dissertation is to develop an Artificial Neural Network to analyze low-thrust trajectories. A neural network approach was chosen because a properly trained neural network can predict the behavior of nonlinear functions. By predicting nonlinearities, the dynamics can be more accurately predicted than by linearizing the dynamics.

Section 1 will begin by discussing the fundamentals of orbital mechanics and orbital proximity problems will be discussed. There will be a brief discussion about low-thrust models and optimization. The fundamentals of neural networks and their architecture will be discussed. Calculating the output of a neural network as well as the error, and methods to minimize the error and update the weights will be discussed. Then search direction determination methods, and a few standard techniques on how to select weight values will be discussed.

In section 2, a literature review is conducted on various approaches on modeling and solving low-thrust trajectories. There is also a literature review conducted on the use of neural networks for trajectory applications.

Section 3 demonstrates a preliminary design of a Neural Network modeling the dynamics of an orbit proximity problem.

Section 4 describes the future work that will be done in the dissertation.

Section 5 and 6 conclude the proposal by providing a timeline and tasks necessary to complete the dissertation and a brief overview of what was discussed.

1.1 Problem Statement

For the dissertation, the research objective is to create a general mission planning tool and to formulate a neural network to solve general low-thrust orbital targeting problems based on a dynamic model. The dynamic model uses a complex set of equations of motion and other pertubation like solar radiation pressure. A control thrust model will be formulated based on shaped-based methods or Chebyshev polynomials to define the form of the thrust vector and provide an initial guess. The neural network's training dataset will be based on the formulated low-thrust model. Using a genetic algorithm, the parameters of the neural network can be tuned to optimally approximate the low-thrust problem.

For this proposal, a preliminary study was conducted using three neural networks with a feed-forward topology and a backpropagation learning algorithm. The backpropagation learning algorithm used gradient descent to optimize the neural networks' weights.

A set of orbital trajectories were modeled using the Clohessy-Wiltshire equations. The neural networks were trained and tested using the modeled Clohessy-Wiltshire equations to approximate the position of a chaser spacecraft to demonstrate that a neural network can be used to model orbital dynamics.

For the dissertation, as new neural network will be developed to simulate more complex orbital trajectories. The research will focus on two areas: (1) development of the neural network framework based on the polynomial formulation of tow-thrust orbital trajectories, and (2) development of a strategy for selecting and tuning the neural network structure, using a genetic algorithm. The resulting neural network will be tested on example problems such as an interplanetary trajectory from Earth to Jupiter. Another application can be from Jupiter's orbit, the spacecraft can perform a low-thrust targeting maneuver to a Jupiter moon. To determine if the neural network successfully reached the target, the final state of the spacecraft will be compared to already established methods of modeling low-thrust targeting problems.

1.2 Orbital Mechanics Review

Many orbital mechanics problems can be represented as a restricted two-body problem. In a restricted two-body problem, a small object with negligible mass orbits a central body with no other bodies affecting the system. The dynamics of the restricted two body problem can be described by Newton's equation of planetary motion,

$$\ddot{\mathbf{r}} = \tfrac{\mu}{r^3}\mathbf{r} \tag{1}$$

where $\mathbf{r} = \left(\begin{smallmatrix} r_x \\ r_y \\ r_z \end{smallmatrix}\right)$, is the position, μ is the standard gravitational constant of the central body.

1.2.1 Orbital Elements

In orbital mechanics, several different coordinate systems are used. One method to express the position and velocity of an orbiting object with respect to a central body is by using the Cartesian vectors defined below(2).

$$\begin{aligned} r &= \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^T \\ v &= \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T \end{aligned} \tag{2}$$

The state vector at any point in time can be obtained by solving the two-body differential equations for Newton's equation of motion(1).

Another coordinate system commonly used is the Keplerian orbital elements. These orbital elements describe the shape, size, and orientation of the orbit using six elements as shown in the figure below.

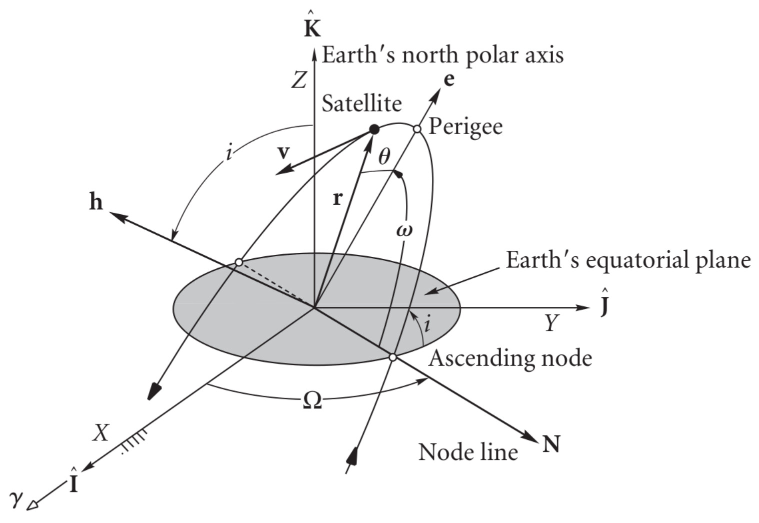


Figure 1: Orbital Elements[1]

The semi-major axis (a), is the average of the periapsis and apoapsis radii. The eccentricity (e), describes the shape of the orbit, with an eccentricity of zero being a circular orbit and as the eccentricity approaches 1 the orbit becomes more elliptical. Beyond an eccentricity of one, the orbit is then hyperbolic. The time of perigee passage (t_p), is the time when an orbit body is at the closest point in the orbit to the central body. The inclination (i) is the vertical tilt of the orbit. The longitude of the ascending node (Ω) is the intersection point between the Earth's equatorial plane and the orbit plane. The argument of perigee (ω) is the angle from the node line to the eccentricity vector in the orbital plane. The true anomaly θ is the angle of the position of a body moving along an orbit measured from periapsis.

An orbit state can be represented as either a state vector [r,v] or a set of six orbital elements [a, e, i, ω, Ω, θ] and one representation and can be converted to the other using mathematical transformations.

1.2.2 Orbit Proximity

Initially, for a spacecraft with high-thrust chemical propulsion systems, a Lambert technique may be used to get a spacecraft close enough to a target spacecraft. A Lambert solver takes a burn-coast-burn approach, which makes it great for getting a spacecraft from point A to point B, but not for proximity maneuvers. Proximity maneuvers are typically done by having a chaser spacecraft perform many small impulsive burns to rendezvous with a target spacecraft. It is assumed the target spacecraft is passive and not performing any maneuvers, and the chaser, is active and performing maneuvers. The relative position between a target and chaser is defined as $\delta r = r - r_0$, where r and r_0 are the position vectors of the chaser and the target, respectively as shown Figure 2 below.

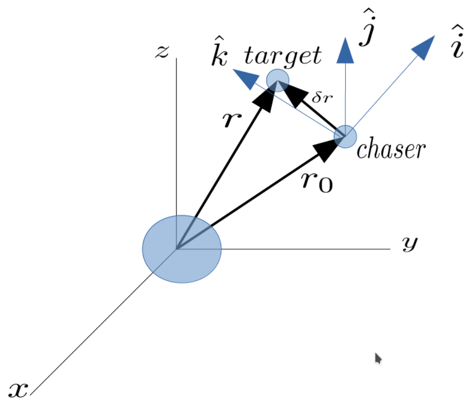


Figure 2: Clohessy-Wiltshire coordinate system

The equation of motion for the chaser relative to the target, and the equation of motion for the target vehicle are expressed in equation (3) and (4) below, respectively[1]

$$\delta\ddot{r} = -\ddot{r_0} - \mu\frac{r_0 + \delta r}{r^3} \tag{3}$$

$$\ddot{r_0} = -\mu\frac{r_0}{r_0^3} \tag{4}$$

For an orbit proximity problem, it is assumed that the target vehicle's orbit is circular, the origin of the reference frame is at the target, gravitation disturbances are negligible, and the distance between the chaser and the target vehicle is relatively small. By making these assumptions and applying some simplifications, equation (3) can be linearized to a set of differential equations[1]

$$0 = \delta\ddot{x} - 3n^2\delta x - 2n\delta\dot{y} \tag{5}$$

$$0 = \delta\ddot{y} + 2n\delta\dot{x} \tag{6}$$

$$0 = \delta\ddot{z} + n^2\delta z \tag{7}$$

The set of coupled differential equations are referred to as the Clohessy-Wiltshire (CW) equations. They can be expressed in a general form in terms of the relative position and velocity.

$$\delta x = (4 - 3\cos nt)\delta x_0 + \frac{\sin nt}{n}\delta u_0 + \frac{2}{n}(1 - \cos nt)\delta v_0 \tag{8}$$

$$\delta y = 6(\sin nt - nt)\delta x_0 + \delta y_0 + \frac{1}{n}(\cos nt - 1)\delta u_0 + \frac{1}{n}(4\sin nt - 3nt)\delta v_0 \tag{9}$$

$$\delta z = \cos nt\delta z_0 + \frac{1}{n}\sin nt\delta w_0 \tag{10}$$

$$\delta u = 3n\sin nt\delta x_0 + \cos nt\delta u_0 + 2\sin nt\delta v_0 \tag{11}$$

$$\delta v = 6(\cos nt - 1)\delta x_0 - 2\sin nt\delta u_0 + (4\cos nt - 3)\delta v_0 \tag{12}$$

$$\delta w = -n\sin nt\delta z_0 + \cos nt\delta w_0 \tag{13}$$

Where δu, δv, and δw are the velocities in the x, y, and z direction, respectively. The CW equations give a good linear approximation to guide a chase vehicle to its target. A applications to the CW equations is a spacecraft docking with the International Space Station(ISS). The major limitation of the CW equations is the accuracy of the equations decrease when there is a large distance between the chaser and the target vehicle. Another limitation is the equations are not applied to spacecraft that are on different orbit planes.

An orbit proximity trajectory model will be shown in the preliminary design section of this proposal.

1.2.3 Dynamic Models and Low-Thrust

Low-thrust maneuvers implement a continuous thrust over an orbit maneuver. For low-thrust maneuvers, thrust is modeled as a continuous vector that changes its magnitude and direction over time. The dynamic model of a low-thrust maneuver for an electric propulsion system can be modeled using a two-body problem,

$$\ddot{r} = \frac{Gm_1}{r^3}\dot{r} + F_t(\alpha_r) \tag{14}$$

where F_t is the thrust magnitude and α_r is the thrusting angle. The equations of motion can be represented as a two-dimensional dynamic model in polar coordinates given below:

$$\begin{aligned} \ddot{r} &= r\dot{\theta_r}^2 - \tfrac{\mu}{r^2} + F(\vec{r})\cos\alpha_r \\ r\ddot{\theta_r} &= -2r\dot{\theta_r} + F(\vec{r})\sin\alpha_r \\ \dot{m} &= \mu(r) \end{aligned} \tag{15}$$

where r is the radial position, θ_r is the direction, $F(\vec{r})$ is the thrust magnitude, α_r is the thrust direction, and $\dot{m}$ is the mass flow rate of the electric propulsion system. A small change in the magnitude or direction of the thrust may result in a large change of the trajectory. To find a continuous thrust control that solves an orbital targeting problem an optimization approach must be taken.

1.2.4 Optimization

For an optimization model, good initial guesses are more important for one type of optimization problem than another to reach a good approximation. Two major approaches to solving an optimization problem are by taking a direct or an indirect approach. A direct approach solves a nonlinear programming problem using a penalty function or augmented Lagrangian functions to find the control parameters. An indirect approach finds an approximate solution using an associated two-point boundary value problem. To obtain a good approximate solution, many people have developed methods that generate good initial guesses.

One approach uses a shape-based approximation method that approximates the shape of low-thrust trajectory with a function. To find an approximate function, parametric data is taken from a known optimal trajectory and passed through a curve-fitting algorithm to return a function that best fits the data[2]. There are many types of shaped-based methods (e.g. exponential sinusoidal, logarithmic spiral, and inverse polynomial).

Another approach uses Chebyshev polynomials to represent the trajectory of the spacecraft. The polynomials are flexible and can handle large changes in state variables and different timescales to create an approximate trajectory. The Chebyshev trajectory is then used to create a feasible trajectory by converting the acceleration profile into a test profile and incorporating the spacecraft's engine and mass parameters[3].

1.3 Artificial Neural Networks

Machine learning is an emerging field where statistical models are applied to datasets for prediction or classification. Machine learning algorithms can used to solve two types of problems, prediction and classification. Prediction is used to predict or estimate a future outcome or value. Classification is used to identify or categorize a pattern given a set of data. Machine learning has two major learning categories, supervised and unsupervised learning. With supervised learning, a statistical model is trained or fit to a dataset and the dataset can be tested with a set of known answers or responses. Some examples of supervised learning are linear and logistic regression. A linear regression example commonly studied predicts the future price of a stock given the opening stock price for a corresponding date. An example of where logistic regression can be applied is handwriting recognition. Unsupervised learning is where the answer or response of a dataset is not known. Some examples of unsupervised learning models are k-means clustering, hierarchical clustering, and neural networks. Image or pattern recognition is a common application for clustering. Supervised and unsupervised learning methods can be used together to form more complex statistical models. Neural networks can be used for supervised learning for function approximation, and unsupervised learning for pattern recognition.

A dataset is any collection of data. The data can be a combination of numeric and string data. A dataset is made up of features, or predictors, labels, and responses. Features or predictors are considered as the independent variables, the labels are the names of the predictors, and the responses are the effects of the predictors.

An individual node has an activation function that

An artificial neural network is a combination of interconnected artificial neurons or nodes that take in an input and produce an output. The output of an artificial neuron is calculated by an activation function which is a non-linear function of the sum of the artificial neuron's inputs. an output is generated and sent the other nodes connected to it. Artificial neural networks were originally inspired by biological neural networks. To have a deeper understanding of an artificial neural network structure, it is beneficial to look at the components of a biological neural network as shown in Figure 3.

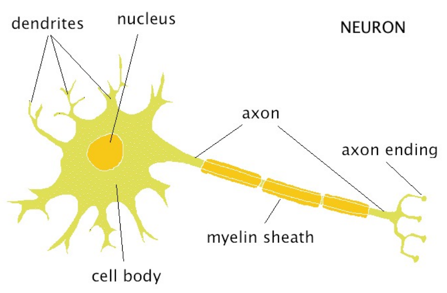


Figure 3: Diagram of a biological neuron `http://webspace.ship.edu/cgboer/theneuron.html`[4]

The dendrite is the component where inputs are accepted from other neurons. The inputs are taken into the cell body in the form of electrical energy. Within the cell body the input signal is processed and when a threshold is achieved, the output is sent from the cell body to the axon. The axon then connects to the dendrites of the other neurons.

1.3.1 Structure

An artificial neural network began as a system that can perform NOT, OR, and AND logical operations. The simplest of neural networks is the McCulloch-Pitts Neuron Model. The output of this model can be expressed in the function and figure below.

$$O^{k+1} = \begin{cases} 1 \quad if \quad \Sigma_{i=1}^{n} w_i x_i^k \geq Threshold \\ \\ 0 \quad if \quad \Sigma_{i=1}^{n} w_i x_i^k \leq Threshold \end{cases} \tag{16}$$

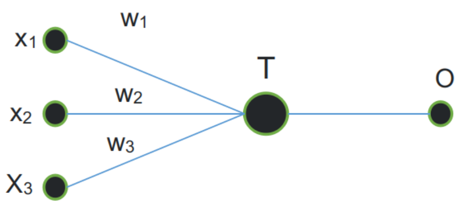


Figure 4: Diagram of McCulloch-Pitts Neuron Model [5]

A neural network is made up of layers connected to each other via weights. A simple neural network is composed of an input layer and an output layer as the one shown below in Figure 5.

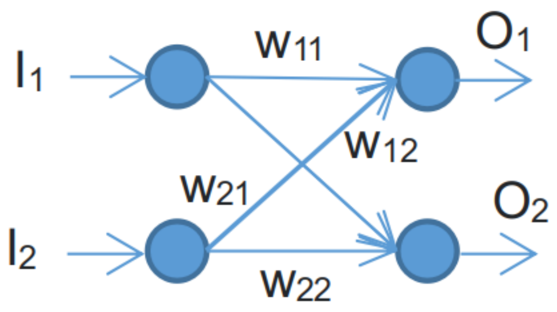


Figure 5: Simple neural network

Starting from the left on Figure 5, the arrow going to the first node is the input value. The circles in the figure represent the nodes. The links interconnecting the nodes are the weights. The arrows exiting to the right from the final circle represent the output.

The outputs of the neural network are determined by the weights of the nodes. The nodes are composed of the sum of the incoming weights of the node and an activation function as seen in Figure 6.

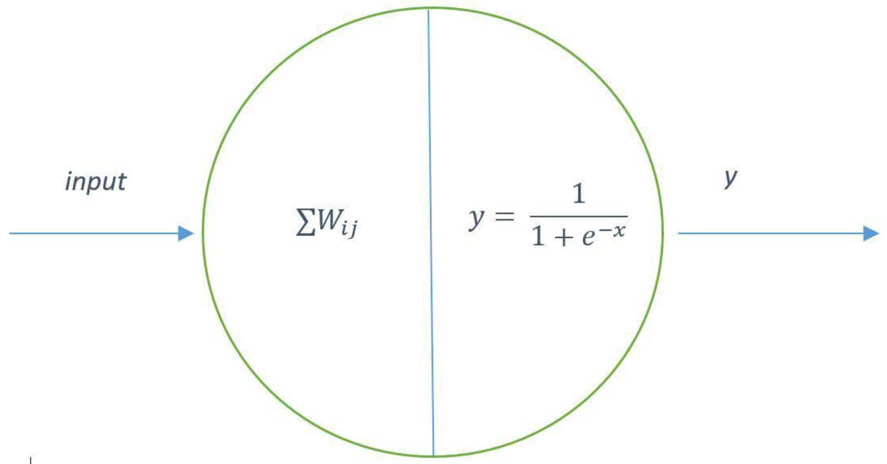


Figure 6: Neural Network Node

where x is the sum of the input weights and y is output of the sigmoid function. For a 2x2 example, the sum of the weights take the matrix form of,

$$\mathbf{x} = \mathbf{WI}$$
$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \tag{17}$$

where **I** is the input vector A common activation function that is used is the sigmoid or logistic function shown below.

$$y(x) = \frac{1}{1+e^{-x}} \tag{18}$$

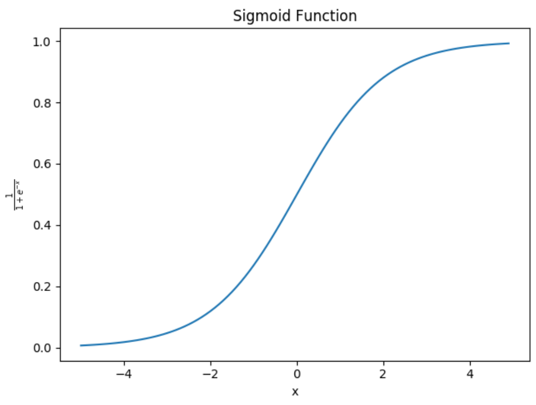


Figure 7: Sigmoid activation function

The sigmoid function, Figure 7, is a popular choice with neural networks because it is asymptotic where outputs can not be either zero or one and differentiable. The output of the nodes connects to the next layer in the network.

A neural network with only two layers is simple to use but it may not give accurate results. In a neural network multiple layers are used to refine the output as seen in the figure below.

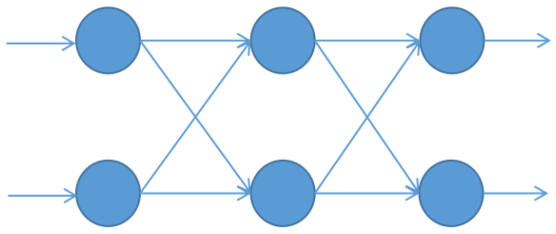


Figure 8: Multiple layer neural network

The nodes between the input and output layers are referred to as the hidden layer. A neural network is not limited to only one hidden layer and not all of the nodes have to be interconnected to each other. By increasing the number of nodes in the hidden layer the computational complexity increases as well.

1.3.2 Topologies

Many different topologies for neural networks have been explained. Different topologies have different advantages and disadvantages. Figure 8 is referred to as a feedforward neural network. A feedforward network will take an input and propagate it forward towards the output. Also all of the nodes are linked from layer to layer. Another topology is known as the skip layer network as seen in Figure 9. For a skip layer neural network, a node can skip connecting to the next layer, the input layer connecting directly to the output layer [6].

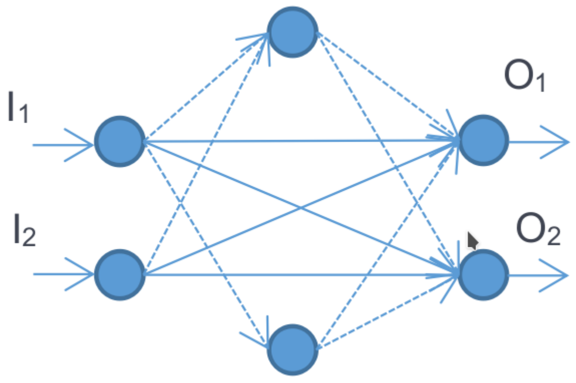


Figure 9: Skip layer neural network[6]

A recurrent neural network works in a similar matter to a feedforward network but has feedback loops through time. With this topology, nodes are only activated for a limited amount of time which allow the nodes to use current and previously encountered neurons. These types of networks are typically used for natural language processing, handwriting analysis, and image recognition, but due to the time component this type of network can be difficult to train [7]. A recurrence network is where a node's output is fed back into its input as seen in Figure 10.

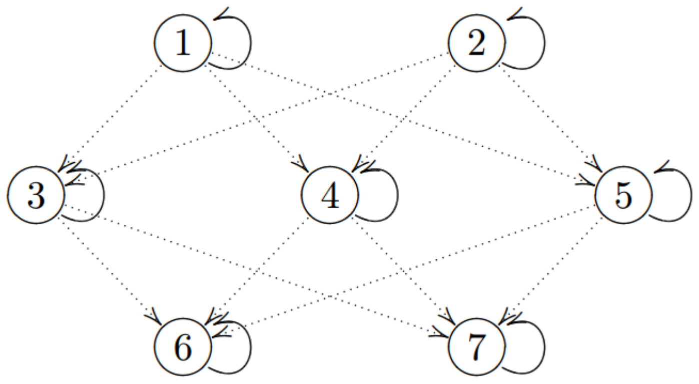


Figure 10: direct recurrent neural network[6]

There are many other types of neural network topologies that will not be discussed in this paper with their own advantages and disadvantages. The preliminary design section of this proposal will use a feed-forward topology.

1.3.3 Error

For a supervised learning approach, a neural network is fed a set of training data and the output of the network is compared to a desired output. Initially random weights are selected for the links to each node, the input data through the input layer is processed through each of the hidden layer nodes to the output layer. The output of the network is then compared to the desired output of the training data, to determine the error.

$$error = O_{training} - O_{Data} \tag{19}$$

To properly train a neural network and update the weights, this error must be minimized. A level error occurs at each node and every input weight contributes a portion of error. A higher weight value contributes more error than a smaller weight.

The error contributed by each weight is determined by the error fraction (20).

$$e_{weight} = \frac{w_{ij}}{\sum_{i,j=1}^{n} w_{ij}} e_{out} \tag{20}$$

Where e_{out} is the output error of the previous node closest to the output layer. A neural network that propagates activation from the input to the output layers and then propagates the error from the output layer to the input layer is called a backpropagating neural network. The change is calculated for each of the weights in a network. The error of the weights in the hidden layer can be expressed in the matrix form[8]

$$\mathbf{e}_{hidden} = \mathbf{w}_{hidden}^{T} \mathbf{e}_{output} \tag{21}$$

After the error is determined, the next step is to change the weights to minimize the error.

1.3.4 Updating Weights

To make a neural network operate properly, the weights must be updated. A brute force approach of updating each weight is too computationally intensive and time consuming. A more efficient approach to update the weights is to use search direction determination methods. Search direction determination is applied to neural networks as a means to minimize the change in error across every layer along the network. A popular first-order method for determining the search direction is steepest descent or gradient descent method. This method uses the gradient to find the descent direction of the cost function. The gradient always points in the direction of maximum increase in the cost function, so the negative of the gradient points in the direction of maximum local rate of decrease of the cost function[9].

$$min(\frac{\partial E}{\partial W_{jk}}) \tag{22}$$

The output error of a neural network can be defined as

$$\frac{\partial}{\partial W_{jk}} = \Sigma_n (t_n - O_n)^2 \tag{23}$$

The summation is squared to prevent any negative values. Since the total is a summation of the error from the weights of the network, Equation (23) can be further expanded[8],

$$\frac{\partial E}{\partial W_{jk}} = -(t_k - O_k) sigmoid(\Sigma w_{jk} O_j)(1 - sigmoid(\Sigma w_{jk} O_j) O_j \tag{24}$$

where $t_k - O_k$ is the error from the output layer, the first sigmoid function is the signal into the final hidden layer, the next sigmoid function is the signal from the previous hidden layer, O_j is the input value to the neural network, and the sigmoid function is the activation function as seen in Figure 7. The updated weight can be expressed as:

$$w_{jk} = w_{ojk} - \alpha \frac{\partial E}{\partial w_{jk}} \tag{25}$$

where w_{ojk} is the original weight and α is a constant learning factor. The learning factor is used to make the network's learning more efficient by preventing the weights from reaching minimum value by overshooting.

Gradient descent is recommended for use with large neural networks with many parameters when a supervised learning approach is possible. There are other search direction determination methods such as conjugate gradient method, Newton's method, quasi-Newton method, Levenberg-Marquardt algorithm, and others that can be implemented into neural networks as well.

1.3.5 Weight and Training Data Selection

When selecting training data and weights there are some rules of thumb that are followed to ensure a well functioning neural network. The first convention is to avoid using large initial weights. Since the output of the weights are based on the sigmoid function, higher weight functions may cause the weight to become saturated. The weight values should be chosen randomly, but the randomization should be in the range of the inverse of the square root of the number of links into a node. For example, if there are two weights going into a node the initial weights should be approximately $\frac{1}{\sqrt{2}}$. Also it is important to avoid setting all of the weights to the same value or setting a weight to zero[8]. By setting the weights to the same value, the error will be equally distributed among the weights. This will make updating the weights inaccurate. By setting a weight to zero, the weight does not contribute anything to the output.

2 Literature Review

The following is a literature review that visits various other works on the topics of low-thrust trajectory optimization, neural networks, and the use of neural networks to solve trajectory optimization problems. Orbital averaging methods have been developed and used in many studies [10, 11]. Neural networks can run efficiently on graphic processing units compared to central processing units, and designed efficiently using genetic algorithms[10, 12]. Neural Networks can be used as controllers to implement control gains for optimizing low-thrust spiral trajectories, generate a thrust direction vector to steer a spacecraft, or to approximate an objective function for a spacecraft trajectory[13–15].

2.1 Orbit Trajectory Analysis

Complex orbit trajectories cannot always be approximated using direct integration methods and may require optimization techniques. One method to solve orbit trajectories is a direct optimization approach developed by Kluever[10]. The author parametrized the control profile u(t) and numerically integrated the state equations. To reduce the computational complexity orbital averaging was used, which is based on the orbital elements time rate of change. An orbital element average rate of change is determined by calculating the incremental change in an orbital element per orbit revolution and dividing it by the orbital period. To parametrize the control profile u(t), the Hamiltonian function for optimal control theory was used. The control law obtained the optimal steering laws from the derivative of the Hamiltonian function. The pitch and yaw steering angles depend on three costate variables (Lagrange multipliers), semi-major axis, eccentricity, and inclination orbital element coefficients that are used as weight functions. The weight functions minimize or maximize the average rate of change for the corresponding orbital elements depending on the value set for the costate variable (higher costate variable give a higher value for the corresponding orbital element).

Solving nonlinear dynamic models can be quite computationally intensive; parameterizing the control profile and or using orbital averaging may allow the neural network to converge to a solution more quickly. Approximation may lead to a significant error, in either the weights of the neural network or the target state of the trajectory.

Different types of targeting problems can be used to provide training data or validation for a neural network. Addis, Cassioli, Locatelli, and Schoen developed a global optimization technique to solve two types of orbit scenarios[11]. The first is the Multiple Gravity Assist problem, where a spacecraft begins at a point and must travel to multiple bodies in a sequence such that the energy used is minimized. The second is a Multiple Gravity Assist Deep Space Maneuver(MGADSM), where a deep space maneuver is done far away from Earth. For a deep space maneuver model, the direction and magnitude of spacecraft's relative velocity at the initial orbiting body and the time instant at which the deep space maneuver takes place must be taken into account[11].

2.2 Neural Network Design

Much of the literature discussing neural networks mentions the issue of time complexity. By increasing the size of a neural network or the complexity of the algorithms used, the time complexity increases as well. Increasing the time complexity means that it will take more time to run trajectories and less time to experiment. Oh and Jung[16] propose implementing a neural network using a graphics processing unit (GPU). Similar to central processing units (CPU), a GPU can be used to perform mathematical operations. These mathematical operations are done on the processing core of the system, which for a CPU is limited to the number of threads a CPU core can support. To do graphical processing requires a lot of computational power, so GPUs were designed with multiple cores so that each core can simultaneously perform mathematical operations. As stated by Oh and Jung, the weight and input matrices of a neural net can be multiplied and the product is then added together with a bias factor. In the paper, the input and weight matrices represent a texture value; this texture value is positioned on row and column values which represent pixel location. An image was input into the neural net for pixel classification to generate a black and white output image. The GPU output was compared to the same image detection program performed on a CPU and the results showed the image generated was the same and computational speed on the GPU increased 20-fold. To create neural networks that are trained and tested using GPUs has been made easier using libraries. Tensorflow and Theano are two popular software libraries that are used for deep learning and neural networks. These libraries easily allow developers to do computations on the GPU.

Many of the papers discussing the use of neural networks to solve various problems provide specific information on the number of nodes a network uses. But determining an adequate number of layers and nodes can be a very tedious task that may require a lot of time to complete but once completeed can give optimal solutions. The work by Bathchis discusses the advantages of using an evolutionary algorithm to determine to structure of a neural network [12]. With a standard neural network, the architecture is fixed and may be suboptimal for a given problem. Bathchis considered many possible approaches to using evolutionary algorithms. One method, was to apply a genetic algorithm to determine to weights of a neural network while keeping the architecture fixed. Another effective method was to use an evolutionary algorithm to find the an optimal neural network and then correct the weights. For the algorithm developed by Bathchis, each generation of ANNs were tested and the best ANN are moved to the next generation. The ANNs developed used direct encoding, meaning all aspects of an ANN are described. The mutations that were introduced can alter the weights of the nodes or the architecture of the ANN. For the learning process of an ANN, each member of a population of an ANN is tested for fitness. The less fit half of the ANN population was destroyed and one was mutated for the next generation. This process continued until one ANN achieved perfect classification or until 500 generations were reached. To test this algorithm the Weka Knowledge Explorer software package was used to generate sample data and classification. The chosen problem was binary classification. The results of the test showed that the evolutionary ANN equaled or outperformed the fixed ANNs. The results showed that the evolutionary algorithm chose 2 to 4 times more nodes than the fixed ANN. Using an evolutionary algorithm may help finding an optimal solution faster than having the researcher manually inspect the results of a trajectory and change the neural network structure and run the trajectory optimization program.

2.3 Neural Networks and Orbit Trajectory Analysis

Artificial neural networks can be used as controllers for optimization problems. When used in conjunction with trajectory modeling, a neural network can be used to model an orbit trajectory. The work done by Yang, Xu, and Zhang[13], modeled low-thrust spiral trajectories of an electric propulsion system using Lyapunov-based guidance and an artificial neural network (ANN) to implement control gains. The ANN used an evolutionary algorithm for learning and training. The dynamic model used the orthogonal radial-horizontal frame (RSW) as the coordinate system. Lyapunov-based guidance was used to model minimum-time and time-fixed minimum-propellant low-thrust orbit transfers. The ANN was a feed-forward network structure, using Particle Swarm Optimization, and differential evolution to determine the optimal parameters. The ANN composed of 12 input layers, 20 hidden layers, and 7 output layer neurons. The time-fixed minimum-propellant transfer was based on an efficiency factor where, a burn was initialized at periapsis and apoapsis. The minimum-time and time-fixed minimum-propellant models used a GTO to GEO transfer and a GTO to GEO transfer with Earth J2 perturbation and Earth-shadow eclipse effect. Attitude and model errors were introduced and the accuracy of the model was determined by the distance to the target orbit. For both minimum-time and time-fixed minimum-propellant models, the satellite was within the threshold, concluding that this model can be used as an autonomous guidance scheme.

Neural networks can be applied to many different low-thrust applications outside of spiral-out maneuvers. To allow for neural networks to generate better approximations genetic algorithms can be used in conjunction with neural networks. The work done by Dachwald modeled interplanetary low-thrust trajectories of spacecraft using evolutionary neural networks [14]. The method proposed uses an artificial neural network and evolutionary algorithms to find an optimal solution without having to give an initial guess. This method only requires a target body and intervals for the initial conditions. Artificial neural networks were combined with evolutionary algorithms to form evolutionary neurocontrollers. Only a feed-forward artificial neural network was considered in this paper. Two cases were modeled, a spacecraft using solar sails and a spacecraft using nuclear electric propulsion. For the solar sail case, the inputs to the ANN were the current and target states of the spacecraft. For the nuclear electric propulsion system case, the current and target states, and fuel consumption were used. The desired output of the ANN for a solar sail is the thrust direction vector, for nuclear electric spacecraft, the thrust direction and engine throttle are the outputs. The dynamic model used the neurocontroller to initially steer the spacecraft and the evolutionary algorithm then examines the parameters generated by the neurocontroller for parameters that would generate an optimal trajectory. These parameters then serve as the output values interpreted as the thrust direction vectors. The spacecraft state and thrust direction vectors were then used in the equations of motion and numerically integrated over the time step, they outputted a new set of spacecraft states that was fed back into the neurocontroller. The trajectory was then rated by a fitness function. This process repeated until the evolutionary algorithm converged to a single steering strategy. The evolutionary neural control algorithm was applied to 3 scenarios. A near-Earth asteroid rendezvous using a solar sail, a MESSENGER-like mission to Mercury, and a Jupiter flyby using nuclear electric propulsion. The algorithm found trajectories that were better than a reference trajectory found using local trajectory optimization methods.

Neural networks can be used to solve a variety of spacecraft trajectories. This allows researchers to experiment with novel trajectories. Ampatzis and Izzo created an artificial neural network(ANN) to approximate an objective function of a spacecraft trajectory[15]. The ANN was integrated into an evolutionary trajectory model. The ANN was adaptively trained by employing the original objective function and collecting input/output data. The neural network was created using feed-forward networks and trained using the Levenberg Marquardt algorithm. The model was then benchmarked using a Multiple Gravity Assist (MGA) Problem, modeled after the Cassini 1 trajectory to Saturn. The objective function was the total deltaV accumulated during the mission. The second benchmark was modeled after the Cassini 2 mission. For this mission, the target planet was Saturn and the planetary flyby sequence was Earth-Venus-Venus-Earth-Jupiter-Saturn and deep space maneuvers(DSM) were allowed in between each one of the planets. The third benchmark used a MGA-DSM mission to Mercury based on the Messenger mission. The results showed that objective functions using an ANN gave accurate results.

Many of the methods used to approximate a control thrust require orthogonal periodic, orthogonal, or shape based functions. Taking a neural network approach to approximate the control thrust will allow for a more general approach. It has been shown that neural networks were used to solve optimization problems. Dachwald, showed that neural networks can be integrated into genetic algorithms to solve interplanetary low-thrust trajectories. Ampatzis and Izzo, demonstrated that neural networks can be trained using an evolutionary algorithm and an objective function to solve a variety of impulsive trajectories. Where there is room to innovate, is to use a genetic algorithm and neural network to solve low-thrust problems other than interplanetary trajectories by creating a general approach to solving low-thrust optimization problems.

3 Preliminary Design

3.1 Statement of work

As an initial demonstration of the use of Artificial Neural Networks to solve orbital dynamics problems, a neural network was created to approximate the dynamics of an orbit proximity problem. The orbit proximity problem uses the Clohessy-Wiltshire (CW) equations to determine the relative position and velocity of a chaser vehicle with respect to a target vehicle. The inputs of the CW problem are the initial state of the chaser vehicle relative to the target, the radius of the circular orbit that the chaser and target vehicles are on, and the time for which to model the dynamics. The output is the solution to the differential CW equation over the specified time span. 200 CW problems are modeled as the training data set for the neural network and 50 CW problems are modeled as the test data set. The initial positions, time span, and the position at time t for the x,y, and z components are used as the training and test data set for a feed-forward neural network.

The feed-forward neural networks consist of an input layer, hidden layer, and output layer. Three neural networks are created to model the x, y, and z dynamics of a CW problem. The neural networks are trained using the time span and initial positions of the chaser vehicle for the input layer. The position of the chaser at a given time input is used for the output layer. The neural networks are then evaluated using a test data set given the same time span and a new set of initial conditions. The trajectory created by the neural networks are then compared to the trajectory calculated by the CW equations and the training and test accuracy of the neural network are compared.

3.2 Implementation and Results

3.2.1 Training and Test Data

This section will discuss the orbit scenario implemented and how results generated by the neural network are used to model the CW equations. The training data is made up of a random set of 200 initial conditions and the test data is made up of a random set of 50 initial conditions from zero to one kilo-meter. The dataset is randomized to prevent the neural network from finding a pattern based on the initial condition inputs. Each CW problem uses the same time span and semi major axis. The time span, semi major axis, and a portion of the relative initial positions and velocities for the training and test data are given in Table 1 below.

Table 1: Examples of training data and test data

	semi major axis(a)	time span	$x_0, y_0, z_0, u_0, v_0, w_0$ (normalized)
training set	6678	$0 \rightarrow \frac{2\pi}{n}$	[0.197, 0.145, 0.110, 0.0, -0.01, 0.56], [0.574, 0.522, 0.648, 0.0, -0.01, 0.56], [0.548, 0.375, 0.815, 0.0, -0.01, 0.56], [0.836, 0.082, 0.260, 0.0, -0.01, 0.56], [0.352, 0.867, 0.882, 0.0, -0.01, 0.56], [0.866, 0.674, 0.558, 0.0, -0.01, 0.56]
test set	6678	$0 \rightarrow \frac{2\pi}{n}$	[0.836, 0.815, 0.669, 0.0, -0.01, 0.56], [0.633, 0.459, 0.208 0.0, -0.01, 0.56], [0.709, 0.374, 0.112, 0.0, -0.01, 0.56], [0.925, 0.733, 0.654, 0.0, -0.01, 0.56], [0.828, 0.617, 0.184, 0.0, -0.01, 0.56], [0.150, 0.125, 0.309, 0.0, -0.01, 0.56]

The mean motion, initial conditions, and time span are passed into an ordinary differential equation solver to calculate the position and velocities from the differential form of the CW equations. The trajectory of the xyz coordinates of the chaser vehicle axis for the first set of initial conditions is shown in Figure 11.

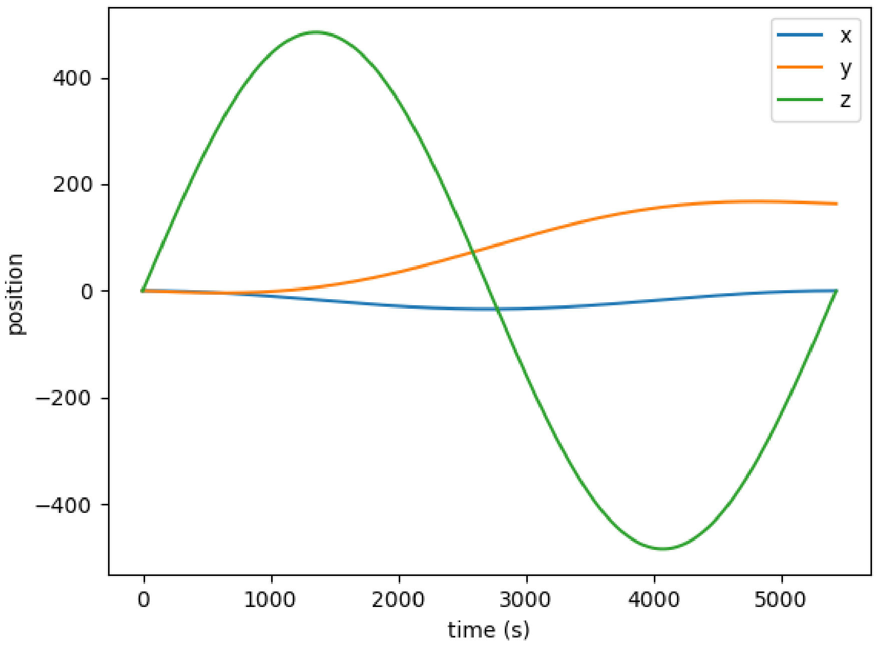


Figure 11: Chaser vehicle trajectory

Before training a neural network, it is good practice to look at a portion of the data set to determine what preprocessing would have to be done to accurately model a neural network. Looking at the training output data, the range for the z components are fairly large relative to the ranges of the activation functions and the x and y components as seen in Figure 11. This may pose a problem for the neural network to accurately optimize the weights and model the trajectory. The z component will be normalized by its maximum value to give a range between -1 and 1. Normalizing the output value does not change the nature of the data and the output can be unnormalized after training and testing. The calculated datasets are passed into a function that processes the data and feeds it into the neural network.

3.2.2 Neural Network Implementation and Results

Within the data preprocessing, the time span is converted from seconds to hours for the neural network input and the initial positions are normalized from zero to one. Three neural networks are created, one representing each component. The time span contains 2716 points, and is combined with the initial conditions as shown in the table below.

Table 2: Input dataset schema for Neural Networks

Neural Network x					Neural Network y					Neural Network z		
time (hr)	x_0	y_0	u_0	v_0	time (hr)	y_0	x_0	v_0	u_0	time (hr)	z_0	w_0
0.000	0.197	0.145	0.0	-0.01	0.000	0.145	0.197	-0.01	0.0	0.000	0.110	0.56
$5.56e^{-4}$	0.197	0.145	0.0	-0.01	$5.56e^{-4}$	0.145	0.197	-0.01	0.0	$5.56e^{-4}$	0.110	0.56
$1.11e^{-3}$	0.197	0.145	0.0	-0.01	$1.11e^{-3}$	0.145	0.197	-0.01	0.0	$1.11e^{-3}$	0.110	0.56
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1.507	0.197	0.145	0.0	0.01	1.507	0.145	0.197	-0.01	0.0	1.507	0.110	0.56

The neural networks representing the x and y components do not have the z component initial conditions because the x and y components are not related to the z components as seen in the dynamic equation (7). After preprocessing the input data set, the three neural networks' parameters are set according to Table 3 below.

Table 3: Parameters of Neural Networks

	epochs	α	input nodes	hidden layers	nodes per hidden layer	activation function	output layer nodes
x	3	0.001	5	2	20	hyperbolic tangent	1
y	3	0.05	5	2	35	hyperbolic tangent	1
z	5	0.03	3	2	50	hyperbolic tangent	1

The weights of the hidden nodes are initialized using random samples from a Gaussian distribution with the mean of the distribution being zero and a scale of $\frac{1}{\sqrt{nodes}}$ where nodes in the number of nodes in the hidden layer. Through trial and error, the hyperbolic tangent, $(y(x) = \frac{2}{1-e^{-2x}} + 1)$, for the x, y, and z components was chosen because the position at certain times are negative for all three components as seen in Figure 11.

The number of nodes per layer, learning rates, and activation functions were selected using trial and error. The trial and error was done by tuning the neural network parameters according to which combination gave the lowest test and training mean squared error and closely modeled the training and test data trajectory.

The training output data for the x and y component remain the same. For the z component, the output is normalized by the maximum z value in the output dataset, and the training data is fed into the neural network. The neural network models the trajectory for the three CW problems. After training, the test data is fed into the neural network by passing the test initial conditions and using the same time span used during training. The trajectory outputs of the neural networks are then compared to the calculated CW test problems by looking at the trajectory and mean squared error. The figure below shows a comparison of one of the test trajectories used with the output of the neural networks.

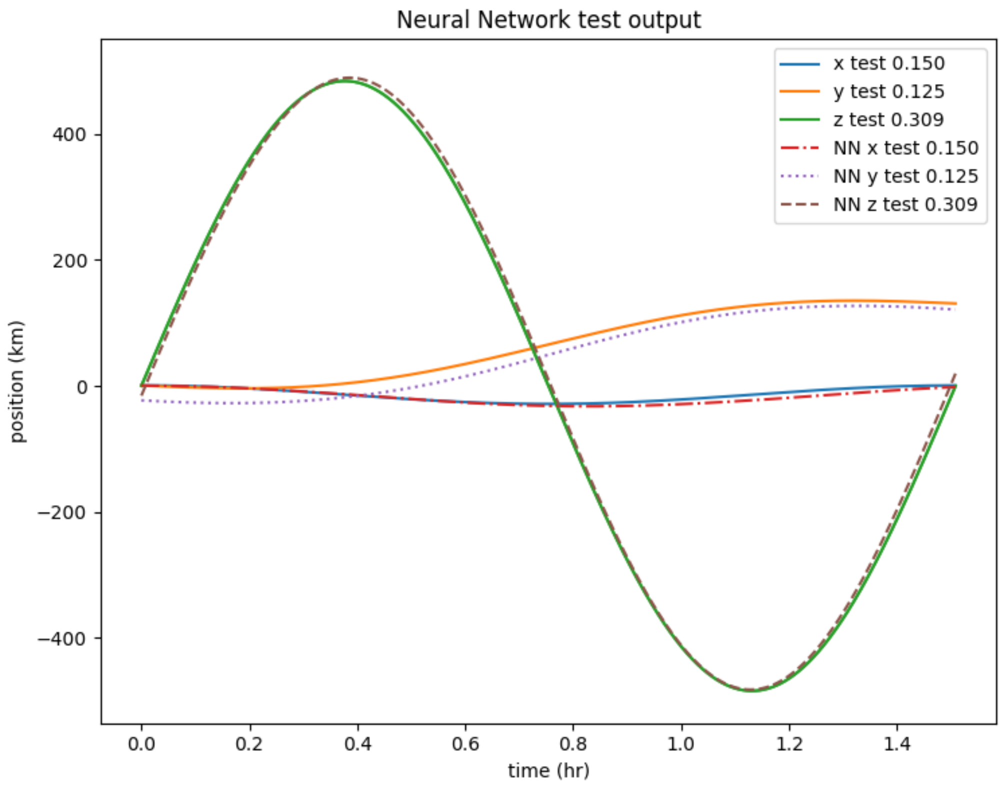


Figure 12: Neural network modeling test data

It can be seen from Figure 12 that the neural networks model the dynamics of the CW equations for test point six. The dotted and dashed lines show the values that the neural networks generate and the solid lines shows the values calculated from the CW equations. With neural networks and other statistical learning models, mean squared error (MSE) is a metric often used to analyze the quality of a predictor,

$$MSE = \frac{1}{n_{points}} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \tag{26}$$

where $\hat{y}_i$ is the estimated value and y_i is the true value. The corresponding training and test MSE for the initial conditions of the x, y, and z positions are given in Table 4 below.

Table 4: Training and test MSE of initial conditions from table 1

Test Points	Component	Training MSE	Test MSE
	x	0.070	21.188
Test Point 1	y	0.0.000180	341.408
	z	0.0543	2178.365
	x	7.095	6.895
Test Point 2	y	0.00235	114.502
	z	0.00868	65.631
	x	4.05425	3.459
Test Point 3	y	0.00170	55.208
	z	0.01380	150.753
	x	6.222	6.847
Test Point 4	y	0.00110	91.617
	z	0.00079	1.804
	x	13.753	16.200
Test Point 5	y	0.00893	251.856
	z	0.01609	35.249
	x	17.296	0.05911
Test Point 6	y	0.00649	1.357
	z	0.00575	291.978

From Table 4 above, the training MSE is higher than the test MSE which is a common occurrence for statistical models. To determine whether an acceptable MSE is obtained, is judged on a per application basis. A small training MSE and a high test MSE may indicate overfitting the training data and therefore, high variance. High variance indicates the flexibility of a model and how sensitive it is to incoming data. Overfitting can be reduced by changing the number of epochs, nodes, and layers of the neural network and would require trial and error to reach an acceptable training and test MSE. Also, it should be noted that the MSE does fluctuate for different initial conditions.

4 Future Work

This proposal demonstrated the use of a neural network to model an orbital mechanics problem with complex dynamics. The results show that by training we can predict the relative position and velocity of a chaser vehicle. Using a similar approach, a neural network can be trained to model more complex orbital mechanics problems.

Low-thrust trajectory optimization is not as simple as solving a CW problem or other impulsive maneuvers. The dynamics of low-thrust trajectories are nonlinear and cannot be solved directly. Typically, low-thrust trajectory problems are solved by simplifying the dynamics and then applying a controller to target minimum fuel usage or mission time. Rather than simplifying low-thrust dynamics, a neural network would be able to approximate the nonlinearities and accurately model the dynamics. This type of approach may create new types of low-thrust trajectory models that may be too complex for humans to create. By using a neural network, it may provide an easier way to relate the control input to the spacecraft's final state making it easier for an optimizer to converge to a minimum-fuel trajectory.

5 Dissertation Approach

For the dynamic model, initially a two-body Sun-centered model will be used. This later can be expanded to a multiple N-body problem. The N-body expansion, will use the Earth, Sun, and Moon system when the spacecraft is within Earth's and Moon's sphere of influence (SOI). When the spacecraft leaves the Earth's SOI, the dynamics will change to a two-body problem Sun spacecraft system. When the spacecraft reaches Jupiter's SOI, the dynamics will change into a three-body Sun Jupiter model.

Solar radiation pressure (SRP) will be applied to the spacecraft as part of the dynamic model. SRP is caused by electromagnetic radiation emitted by the Sun and absorbed or reflected by the spacecraft. This can greatly affect the trajectory of the spacecraft by causing an applied torque on the spacecraft. The SRP model chosen for this mission is the cannonball model.

A neural network was chosen to be used in this interplanetary problem due to their ability of prediction and their approach to problem solving. . Using a conventional optimization method restricts the way in which the interplanetary problem can be solved by restricting the optimization process to following a fixed algorithm. By using neural networks, the optimization model can take a more independent approach to finding an optimal solution. Given the shape of low-thrust trajectories (large spiral orbits), a neural network can predict the shape of a low-thrust trajectory and optimize around it.

Using a heuristic model will create a solution space in which a neural network can find an optimal solution. Within an objective function, a multitude of local minimums may exist but only a few of them may correspond to an acceptable solution. Using a neural network alone may optimize around a minimum that is not an acceptable minimum.

The problem can be written as an optimization problem in the form of:

optimize:

$$J(u) = \psi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$$

Subject to :

$$m_i\ddot{x} = \textstyle\sum_{1\leq i\leq j}^{n} \frac{Gm_i m_j(\overrightarrow{x}_j - \overrightarrow{x}_i)}{|x_i - x_j|^3} + a_{SRP} + u_t(\alpha_r)$$

$$a_{SRP} = -\tfrac{p_{SR}C_r A\overrightarrow{r_{sat}}}{m_{sat}\hat{r}_{sat}}$$

$$x_j, x_i, t_0, t_f \leq 0$$

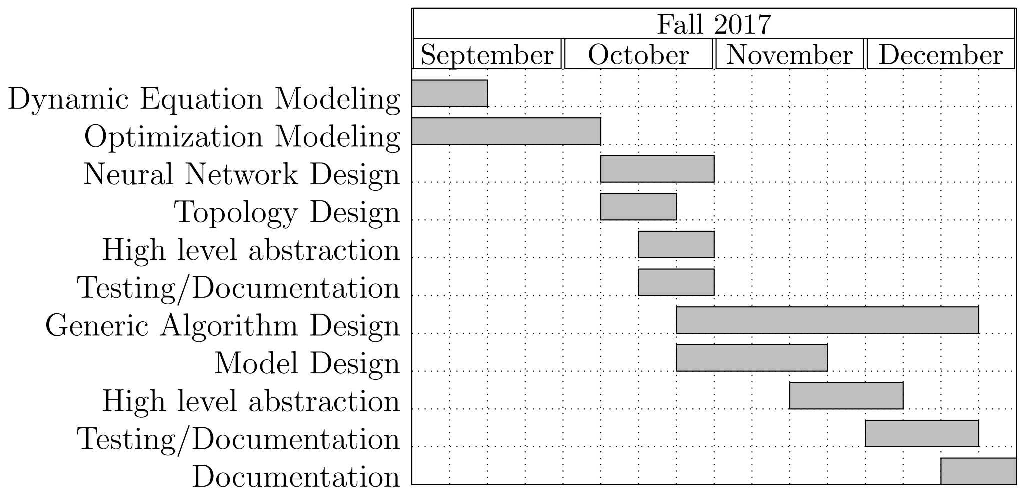
$$\dot{m} = \mu(x)$$

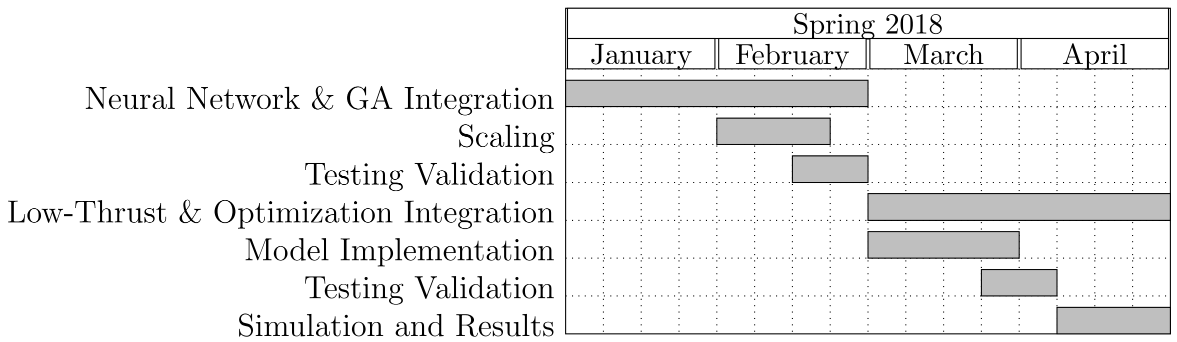
where $J(u)$ is a performance index, ψ is the final spacecraft state error, L is the fuel used, $x(t)$ is the spacecraft state $m_i\ddot{x}$ are the forces from N gravitational bodies, a_{SRP} is the solar radiation pressure, and u is the control and thrust law.

The dissertation will use a neural network and reinforcement learning to solve an interplanetary targeting problem. The first task will be to select the dynamic model and to develop an optimization approach.

The optimization approach will use a reinforcement learning algorithm. A reinforcement learning algorithm will allow the spacecraft to explore new trajectories. The agent for this algorithm is the spacecraft, The optimal policy will be the thrust law, the action is a thrust magnitude and direction, and the reward will be the fuel remaining at the end of a mission. The neural network used will integrate with the reinforcement learning algorithm to predict which action to take at a given time step.

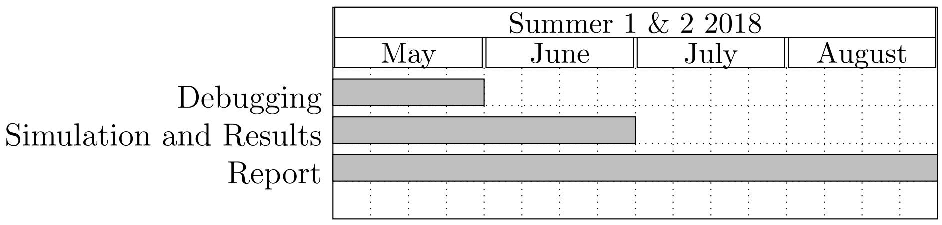
Then the next step will build a neural network as a part of the optimizer for the dynamic model. The neural network will use a popular framework either theano or tensorflow. These frameworks allow complicated neural networks to be designed and scaled more easily than building from scratch. The inputs to the neural network is the spacecraft state and the output of the neural network is the thrust magnitude and direction. The training data set that the neural network and the reinforcement learning algorithm will use will be based on using shaped-based methods to represent low-thrust interplanetary trajectory dynamics. After training, the reinforcement learning algorithm will proceed to further optimize itself for minimum-fuel trajectories by generating trajectories and evaluating the final state error and fuel use. Creating an accurate neural network model will

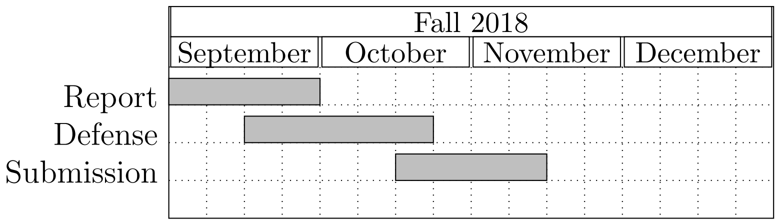




require tweaking many network parameters including learning rate, number of hidden layers, and number of nodes per layer. A good model may be determined by trial and error, but it would ideal to build a genetic algorithm on top of the neural network to create an optimal neural network for a targeting problem.

Also, it would be interesting to test the trained neural network and reinforcement learning algorithm on other types of low-thrust maneuvers to test the robustness of the solver. The following figures give a brief time line and the tasks necessary to complete this dissertation.





6 Conclusion

This proposal began with an introduction into orbital mechanics and orbit proximity. Then neural networks were introduced, describing the history, architecture, and various topologies. The backpropagation algorithm was introduced and how it is implemented with neural networks using training and test data.

This proposal then reviewed key literature on the use of neural networks to model orbital targeting problems. The literature review also discussed some methods used to solve and model low-thrust targeting problems.

The final section proposed a preliminary design. The preliminary design demonstrated using a neural network to predict the dynamics of an orbital proximity problem based on the CW equations. By taking this approach, familiarity with processing a data set and applying it properly to train a neural network was established. The use of neural networks on orbital mechanics problems demonstrated in this proposal will be extended into the dissertation. A dissertation approach was discussed, the approach briefly described low-thrust trajectories and optimization. A schedule was given, showing the tasking needed to be completed and a estimate of how long it will take to complete them.

References

[1] H.D. Curtis. *Orbital Mechanics for Engineering Students.* Elsevier Butterworth-Heinemann, 2 edition, 2005.

[2] B.J. Wall and B.A. Conway. Shape-based approach to low-thrust rendezvous trajectory design. *Journal of Guidance, Control*, 32, 2009.

[3] P.R. Patel, Gallimore A.D., T.H. Zurbuchen, and D.J. Scheeres. An algorithm for generating feasible low thrust interplanetary trajectories. *Journal*, 2009.

[4] G. Boeree. The neuron, 2009. `http://webspace.ship.edu/cgboer/theneuron.html`.

[5] J.M. Zurada. *Introduction to Artificial Neural Systems.* West Publishing Company, 1992.

[6] D. Kriesel. *A Brief Intoduction into Neural Networks.* 2007.

[7] S. Raschka. *Python Machine Learning.* Packt Publishing Company, 2015.

[8] R. Tariq. *Make your own Neural Network.* 1 edition, 2016.

[9] J. Arora. *Introduction to Optimum Design.* Academic Press, 2011.

[10] C.A. Kluever. Low-thrust trajectory optimization using orbital averaging and control parameterization. In *Spacecraft Trajectory Optimization.* Cambridge University Press.

[11] B. Addis, A. Cassioli, Locatelli M., and F. Schoen. Global optimization for the design of space trajectories. *Computational Optimization and Applications*, 48:635–652, 2011.

[12] P. Batchis. An evolutionary algorithm for neural network learning using direct encoding. Technical report, Department of Computer Science, Rutgers University.

[13] Yang D., Xu B., and Zhang L. Optimal low-thrust spiral trajectories using lyapunov-based guidance. *Acta Astronautica*, 2016.

[14] B. Dachwald. Optimization of very-low-thrust trajectories using evolutionary neurocontrol. Technical report, 55th Institutional Astronautical Congress 2004.

[15] Ampatzis C. and Izzo D. Machine learning techniques of approximation of objective functions in trajectory optimization. *European Space Agency*, 2015.

[16] K. Oh and K. Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37:1311–1314, 2004.