a.i. solutions®

Machine Learned Atmospheric Force Model Trained with Two Line Elements

Written by: Clark P. Newman, Fabio Chiappina, Christina Reid

MACHINE LEARNED ATMOSPHERIC FORCE MODEL TRAINED WITH TWO LINE ELEMENTS

Clark P. Newman[(1)], Fabio Chiappina[(2)], Christina Reid[(3)]

[(1)]a.i. solutions, Inc., 4500 Forbes Blvd, Suite 300. Lanham, MD, USA, Email: clark.newman@ai-solutions.com
[(2)]a.i. solutions, Inc., 4500 Forbes Blvd, Suite 300. Lanham, MD, USA, Email: fabio.chiappina@ai-solutions.com
[(3)]a.i. solutions, Inc., 4500 Forbes Blvd, Suite 300. Lanham, MD, USA, Email: christina.reid@ai-solutions.com

ABSTRACT

Thousands of objects orbiting in Low Earth Orbit (LEO) are catalogued and tracked by US Space Force (USSF) in a database of Two-Line Elements (TLEs) that is available online. A time series of TLE states for a particular object will show its orbital decay over time due to atmospheric drag. The decay data across multiple objects can form a training set to create a Machine Learning (ML) model for the atmospheric drag force. This paper investigates a demonstration of this process by training a ML atmosphere model with decay data from a historical object. Regression tests of the ML model against different propagation models are performed. The process to add force modeling context to the algorithm is described.

1. INTRODUCTION

For objects orbiting in LEO (and in the absence of maneuvers), the atmospheric drag force is the dominant perturbation force that affects the trajectory over time. Thus, accurately modeling a time- and state-dependent atmosphere has been a high priority to enable high accuracy propagations and predictions of orbiting objects in LEO. Temporal changes in the atmosphere are largely driven by solar weather. In particular, it has been found that the atmosphere is sensitive to radio solar flux of 10.7 cm wavelength.[1] This radio frequency is observed and modeled and predictions are made from the models, which are published by various sources at various time resolutions and lengths.

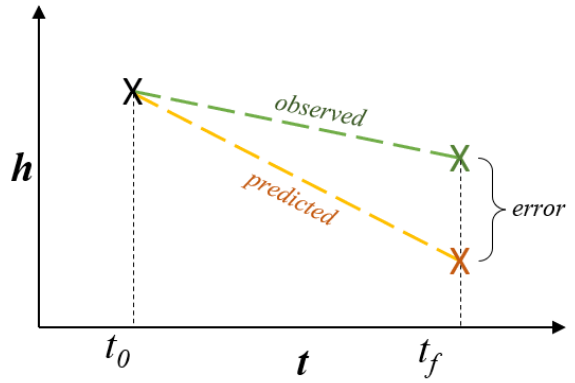

Figure 1. Comparison of observed vs predicted orbital decay of a vehicle over time.

However, the effect of drag upon an orbiting vehicle is observable by the changes in orbital parameters of that vehicle over time. In Fig. 1, this process is notionally plotted, an initial state is observed to decay some amount, as compared to a prediction from the initial state which calculates a different orbital decay. The difference between the observed and predicted orbital states is due to atmospheric modeling errors and forms the basis for a stochastic update to the atmospheric model to improve its predictive value.

Over each time span and across multiple missions, these errors can be organized to be the training data to a ML process that refines the atmospheric model stochastically to reduce prediction errors. In order to train such an ML process, a sufficiently sized training data set is necessary in the form of state histories of multiple LEO vehicles.

The USSF tracks and reports on the states of many orbiting objects, which are available in databases hosted online by organizations such as space-track.org and celestrack.org. The state histories of orbiting objects are formatted as TLEs. This paper investigates the feasibility of the training process and analyzes the performance of the resulting model. The performance is analyzed through a regression test comparing predicted and recorded TLE states for historical missions.

Prior work in the area of machine learning and atmospheric modeling include an analysis into a Long Short-Term Memory NN to forecast the 10.7 cm solar flux[2] and an analysis to estimate thermosphere density during quiet times and during geomagnetic storms using Gaussian processes and neural networks[3]. This work is inspired by and extends these analyses by utilizing training data of space vehicle states to model orbit decay from atmospheric drag.

2. ATMOSPHERIC DRAG

Objects in LEO are subject to a drag force by the upper reaches of the atmosphere. This drag force is nonconservative and reduces the total orbital energy of the object, which reduces its semi-major axis. The reduced semi-major axis causes the object to travel

through higher atmospheric density which further increases the drag force. This causes an exponential orbital decay that eventually will cause the object to de-orbit and re-enter the atmosphere. The object typically is incinerated by atmospheric friction heat.

Atmospheric density as a function of altitude can approximately be modeled as

$$\rho(h) \approx \rho_0 e^{-\frac{h}{H_0}} \quad (1)$$

Where ρ_0 is the atmospheric density at sea level, and has a value of ~1.225 $\frac{\text{kg}}{m^3}$, and H_0 is the height scale, which for Earth's atmosphere has a value of 7.99 km.[4]

This simplistic model ignores temperature variations in altitude, which cause deviations from this ideal exponential decay. The temperature variations are constantly changing and are largely driven by solar activity. Solar weather is thus a driver of atmospheric temperature and thus density, and as such is the subject of considerable observation, modeling, and predictions.

Modern atmospheric models utilize a time series input of solar radio flux values (called F10.7 flux) to accurately model variations in the atmosphere from solar weather. Observations of F10.7 are published, as well as multiple models of predictions. There are short term predictions available from the National Oceanic and Atmospheric Agency (NOAA) which have hourly resolution and predict for 28 days, and there are long term predictions such as Schatten predictions, which have monthly resolution and predict forward for three solar cycles beyond the current cycle (~40 years).[5][6]

Nominal solar weather is chaotic, poorly understood, and is complicated further by geomagnetic storms triggered by Coronal Mass Ejections (CMEs). The seemingly random nature of CMEs and resulting geomagnetic storms makes predictions through these time spans especially difficult. A geomagnetic storm can be indirectly observed by a notable spike in F10.7 flux that deviates from the running average.

A method to accurately observe, model, and predict F10.7 flux would have high value in improving the accuracy of LEO propagations and predictions. Improvements in prediction accuracy can improve performance of collision avoidance, remote sensing, and re-entry predictions. This paper investigates the use of TLE time series histories as training data to craft a ML atmospheric model. The next section describes TLEs and their use in the training process.

3. TWO LINE ELEMENTS (TLEs)

TLEs are a legacy data format for quickly and robustly transmitting vehicle state data. A single TLE is 140 bytes of data arranged in two rows of 70 columns of characters. The end of each row contains a checksum to verify the validity of the transmitted data. A single TLE is displayed in Fig. 2, with each field highlighted and labeled.

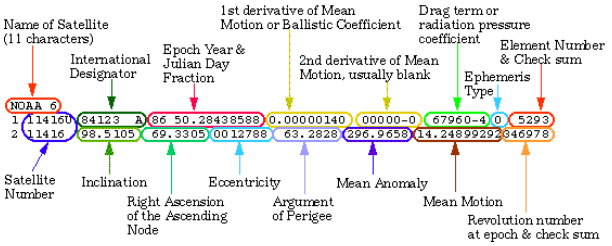

Figure 2. A single TLE with each field highlighted and labeled.

The pertinent fields for this analysis are, in the same order as in Fig. 2:

- Epoch Year & Julian Day Fraction
- 1st derivative of Mean Motion
- Inclination
- Right Ascension of the Ascending Node
- Argument of Perigee
- Mean Anomaly
- Mean Motion

The 2nd derivative of Mean Motion is usually zero, and the drag term is used only within the SGP4 propagator that is designed for TLE use.[7] The object parameters are normalized and the time series orbit decay signal is used to craft a ML atmosphere model.

The USSF tracks objects with a radar cross section of 10 cm^2 or larger, and the states of these objects are maintained in a publicly available database hosted online by space-track.org and celestrak.org. The database of LEO object states over time indirectly observes the atmosphere through the decaying orbital elements over any time span.

TLE time series for multiple contemporaneous LEO objects can form the basis of training data that can be processed by ML algorithms to produce a stochastically trained model that "learns" from the training data. The next section describes Neural Networks (NNs) and the process of backpropagation.

4. NEURAL NETWORKS (NNs)

Neural Networks (NNs) are a mathematical model that roughly emulates the natural processes of neurons in the brains of animals. A diagram of a simplified NN is depicted below in Fig. 3.

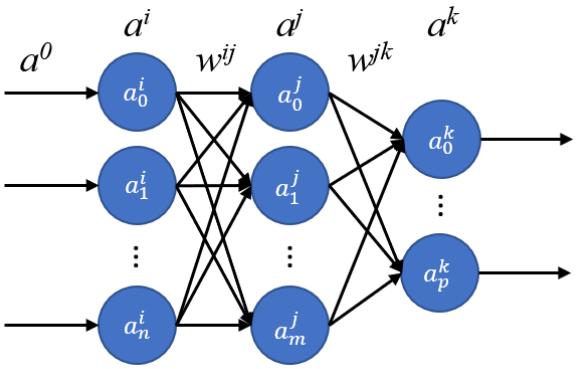

Figure 3. Diagram of a representative NN

In the diagram of Fig. 3, there is an input layer of n neurons labeled the i^{th} layer, a hidden layer of m neurons labeled the j^{th} layer, and an output layer of p neurons labeled the k^{th} layer. Each neuron takes as input the weighted activations from the previous layer, a bias, and processes the resulting value through a function. The weighted activation inputs to the neuron above takes the form

$$a_0^j = \mathrm{ReLU}\left(a_0^i W_{00}^{ij} + a_1^i W_{10}^{ij} + \cdots + a_n^i W_{n0}^{ij} + b_0^j\right) \quad (2)$$

Typically for modern NN applications, the activation function is a Rectified Linear Unit (ReLU) activation function that follows

$$\mathrm{ReLU}(x) = \max(0, x) \quad (3)$$

The ReLU function is different from the more classic Sigmoid function which has a maximum value of unity regardless of the input activation value. ReLU allows unbounded positive activation value which can improve its performance with regression problems.[8]

The NN has two basic processing modes: forward propagation and backward propagation. Forward propagation can be interpreted as 'executing' the model, whereas backward propagation can be considered as 'training' the model. Forward propagation is the processing of input data through the weighted activations across the NN to return the output layer values.

Backward propagation (or simply backpropagation) combines the solution of a forward propagation with the expected or desired solution from that set of inputs, and adjusts the weighting coefficients W's to minimize the error between the observed and expected output. Backpropagation requires a training data set with desired solutions paired with training input cases. Backpropagation attempts to minimize a cost function defined as

$$C(\mathbf{y}, \mathbf{o}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - o_i)^2 \quad (4)$$

Where **y** is a vector of computed outputs, **o** is a vector of observed outputs, and N is the total number of training data points.

Each execution of a backpropagation adjusts the weights of the NN to be more accurate when exposed to the training problem again. This adjustment is called Stochastic Gradient Descent, as each execution of backpropagation slightly adjusts the weights to be more accurate in the future to that input but can only consider the data it's been trained with. Thus, the quality and organization of training data becomes a primary driver of NN model accuracy, and additional techniques exist to add context to the NN, so it is not entirely empirical in nature.

Two methods for adding physical context to a NN model include Physics Influenced Neural Networks (PINN) and Discrepancy Modeling. PINNs add equations of motion to the cost function, which can incur a high cost to network solutions that are not physically feasible.[9] Discrepancy Modeling focuses the cost function on the difference between an analytic force model and observed outputs of a real system controlled with analytic equations of motion.[10]

4.1. TRAINING, VALIDATION, TESTING

As NN are entirely data-driven, their resulting capabilities are heavily influenced by training quality and quantity. Given a fixed set of training data, the process by which that data is processed in training can have an impact on the resulting performance. I.e. it is possible to increase system performance (or decrease) by adjusting the order and methods that training data is utilized in a training process.

The largest driver of system performance is training quantity. On a basic level, more training typically results in better performance, with caveats. A central concern of training is overfitting to the training data. A system that is "overfit" to its training data will perform well on test data that is in-family with the training data, but that performance falls off quickly if exposed to novel inputs that are not in-family with training data. It is important then to have training data that spans the problem space in which the system is expected to perform.

Given a set of training data, a common approach is to split the training data into three groups: training, validation, and testing. Occasionally the validation and testing groups are combined or referred to interchangeably. Of the three subsets of data, the

training set is the largest and first used to train the system. The system's performance cannot be directly evaluated by its performance on the training data, which would result in an unobserved bias and can lead to overfitting, so the performance of the system is evaluated on the validation set. It is at this point that the hyperparameters (e.g. number of hidden layers, number of neurons per layer) are tuned to increase performance.

Once a system has been trained on the training set and adjusted through evaluation of the validation set, it is again trained on the training and validation set with the updated hyperparameters. Finally, a new "zero bias" evaluation is calculated by evaluating performance on the final Test set of training data. The performance of the model on the test data set is used for comparison and evaluation to external models.

5. TLE TRAINING PROCESS

The process to take TLE state inputs, organize ML training data from them, train a ML model, and perform regression testing is shown below in Fig. 4.

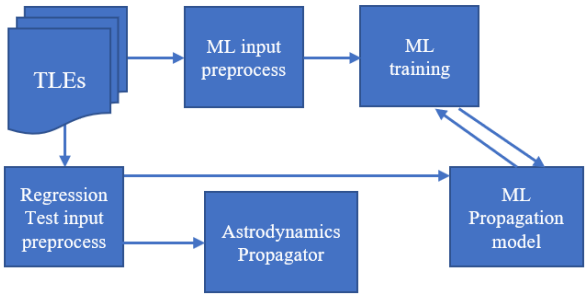

Figure 4. ML Training and Testing Diagram.

Considering a single object, a time history of TLE states is compiled, as shown in the top left of Fig. 4. The TLE states are preprocessed to create the training data. Each training datum contains a pair of TLEs from the same vehicle, separated in time (called the *a priori* state and the *a posteriori* state). The training data is then utilized by the ML Propagation model, which processes the inputs to produce *a posteriori* states. The error between the observed and computed *a posteriori* states drives the training of the ML model.

After training and validation, the ML model is subjected to regression testing on objects in the validation and/or test set. Regression testing on the validation set allow for tuning the model, and regression testing on the test object is for final performance evaluation and comparison.

The training data for this system is naturally broken into time series inputs from various objects. An intuitive method of crafting a training, validation, and test set of data is to name specific objects to populate each subset of training data. E.g. the time series histories of ten objects can be used to train the model, then validation can be evaluated on three different objects, and finally tested on an additional object. In this example the model is trained, validated, and tested on fourteen objects.

5.1. DATA DRIVEN MODEL

The prototype model utilized to explore this approach to orbital decay characterization is called the Data Driven Model (DDM). The DDM is able to "propagate" orbits forward in time completely separated from numerical integration and force modeling.

The inputs and outputs of the prototype NN that models the decay of a LEO object's orbit is displayed below in Fig. 5.

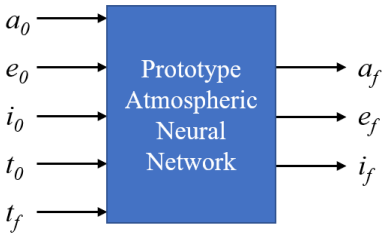

Figure 5. Inputs and Outputs of Atmospheric NN

The prototype NN takes five values as input and three values as output. The input values are:

1. a_0: initial semi-major axis
2. e_0: initial eccentricity
3. i_0: initial inclination
4. t_0: initial epoch
5. t_f: final epoch

The inputs are normalized to be between 0 and 1. The three outputs are:

1. a_f: initial semi-major axis
2. e_f: initial semi-major axis
3. i_f: initial semi-major axis

The output states are likewise normalized to be between 0 and 1. The state vector has been reduced to defining the shape of the orbit but loses information about orbit placement: Right Ascension of the Ascending Node, Argument of Periapsis, and the True (or Mean) Anomaly. For this version, orbit shape is emphasized over explicit position of the vehicle over time. Reducing the state size in a neural network is called *encoding* and can accelerate processing and accuracy by reducing the number of correlations that need to be trained.

This prototype NN accepts as input the shape of a LEO trajectory as defined by semi-major axis, eccentricity, and inclination. The semi-major axis and eccentricity are both expected to decrease as the time span increases, while the inclination is expected to remain the same. It remains as an output to a) enforce that inclination is

invariant with time and b) establish sensitivity to objects with varying inclination. The motivation is that the atmosphere is an oblate spheroid, which would have low-inclination objects encounter higher densities than high-inclination objects.

5.2. TRAINING DATA ORGANIZATION

From a trove of TLE state vectors across multiple objects and timespans, it is necessary to properly collect, organize, and format the state vectors and labels to train the NN. Considering a single object to train with, a time series history of state vectors in TLE format are collected. For this analysis, focus is given to the first quarter of 2020. TLEs from multiple objects that span the first three months of 2020 are downloaded from online databases.

Given a time series of TLE states from a single object in LEO, a training pair can be made from any two TLE states separated by positive time. A chart depicting valid TLE pairs (in blue) for training data input is shown below in Fig. 6.

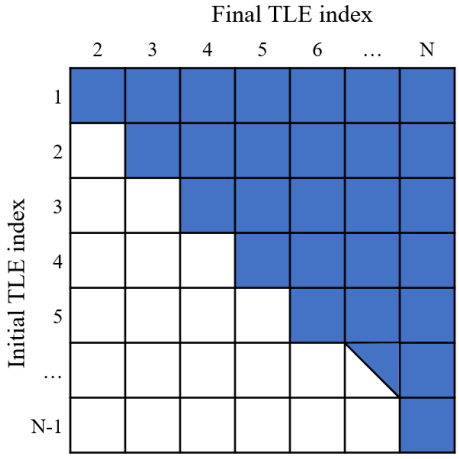

Figure 6. Valid TLE pairs for training data highlighted in blue.

s shown in Fig. 6, each TLE can be paired with any/every TLE that has a higher epoch. The first TLE can be paired with every subsequent TLE while the N-1th TLE can only be paired with the Nth TLE. The maximum number of training pairs from N TLEs of the same object in time is thus

$$\max\left(N_{train}\right) = \frac{1}{2}\left(N_{TLE} - 1\right)^2 \tag{5}$$

E.g., given 101 TLEs from a single object, a maximum of 5000 training pairs can be collected. To increase training speed and adjust the time distribution of training pairs, this maximum set of training pairs is reduced by a thinning coefficient to randomly utilize a subset of possible combinations. This thinning coefficient can be a tuning factor to simultaneously reduce computation time and improve distribution of training pairs.

The model is affected by the order of the training data processed by the model. It is possible to encounter "training saturation", a flavor of early-onset overfitting, by poorly ordering the training data so that the model becomes overfit to a subset of data before training has completed. It is important then to properly randomize the input data order to be robust against this phenomenon. For this application this means randomizing both training pairs in time and across training objects.

5.3. INITIAL TRAINING OBJECTS

For the initial training of the data-driven system, a subset of objects from the vast database of tracked objects needs to be designed. If each TLE is considered a noisy observation of the atmosphere, intuition would guide that a robust training set would span a range of orbital geometries to "observe" the atmosphere with geometric diversity.

To have predictive value within orbital geometries of interest, the model must be trained on objects that inhabit those orbital geometries. The altitude range of 300-1000 km is considered for this exercise. The set of objects to train from is thus limited to this altitude range. Additionally, it is desirable to have objects with diversity of orbital inclination to capture asymmetric properties of the atmosphere. Finally, it is important to capture objects that are not maneuvering, or to disallow training pairs across a maneuver. By only selecting rocket body objects, the chance of a maneuver within the training time span is eliminated. Additionally, rocket bodies have large radar cross sections which improve their tracking characteristics and sensitivity to atmospheric drag.

Below in Fig. 7 is a list of candidate objects which are utilized in a training exercise again focusing on the timespan of the first quarter of 2020.

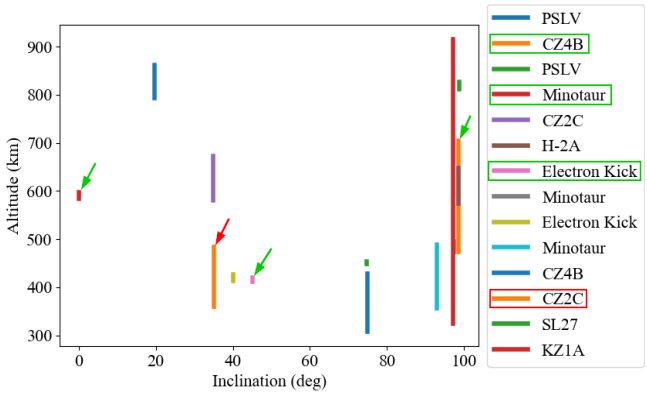

Figure 7. Training objects Inclination and Altitude range. Validation objects are highlighted in green, and test object highlighted in red.

These candidate objects are all rocket bodies and do not contain maneuvers over the training timespan of the first quarter of 2020. Note the variation in inclination and heights of apogee and perigee, respectively. With proper training, the resulting model should have predictive value for any object whose orbit occupies the space spanned by the training objects.

6. MODEL TRAINING

A common approach to training a model has three steps mentioned previously: training, validation, and testing. The first step of training is called "training" because it constitutes the bulk of backpropagation calculations. In this exercise, the majority of the training objects are used in the training step. Their TLEs over a common time span are collected, and training pairs of starting-ending states are generated. Similarly, a subset of objects is used for validation and testing, and training pairs are generated from their respective TLE states.

From the ten test objects over the time span of the first quarter of 2020, there are 149,228 training pairs. The inputs from the testing objects are randomized before processing through the backpropagation routine. The inputs are grouped into batches that are processed sequentially to generate stochastic gradient steps that update the system parameters. The training is "complete" when all batches have been backpropagated through the system.

As shown in Fig. 7, the three objects whose names are boxed in green are dedicated for the validation set, while the one object boxed in red is dedicated to be the test set. Backpropagation is performed on the training set, and then evaluated on the validation set. The evaluation of the validation set allows the model to be tuned on the training data without overfitting to that specific training data.

With the training and validation sets, a sensitivity analysis is run on the hyperparameters (i.e. number of layers and neurons per layer) to ascertain the optimal architecture for this application. Special care must be taken to organize the initial weights and training data order to return consistent solutions from the same set of training data. Below in Fig. 7 is a correlogram depicting the performance of each model architecture on the same training data and validation data. Each model is represented with a colored circle whose color corresponds to the mean error, and the radius corresponds to the variance of error. Not captured in this analysis is the training speed and number of calculations for each model. There is a positive correlation between training clock time and number of trainable parameters (weights between successive neurons), so larger systems with more layers and neurons per layer take longer to train. This will have more of an impact on larger training data sets.

The results from Fig. 8 suggest a model in the interior of the plot could be expected to be more reliable than models at the extremes of hyperparameter range. The results also suggest that the training is still in a data-poor regime, as the ordering of training data and initial weights of the model have a strong influence on the resulting performance. This is a symptom of the limitations to manually collecting training data, for which automation could alleviate by significantly expanding the training set. From these results, an architecture of 4 layers of 64 neurons each (circled below in Fig. 8) will be utilized for the remaining tests. This analysis shall be repeated as the training and validation data sets grow.

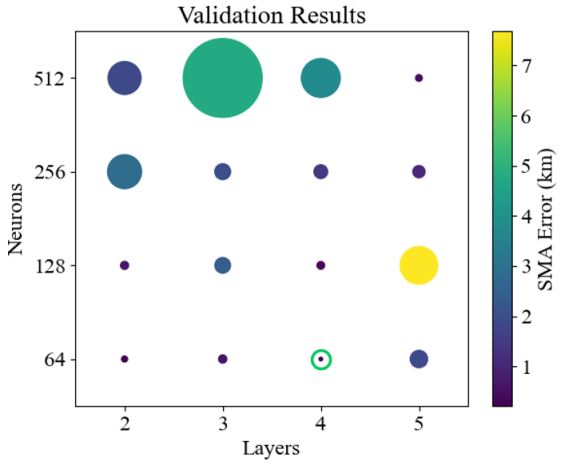

Figure 8. Correlogram depicting the error of each model in a color scale with the variance of error as the radius of the data points. Chosen architecture is circled in green.

7. MODEL TESTING

With the chosen architecture, the model is then trained on the training set and tested more deeply on the test vehicle CZ2C. To further stress the model and explore its predictive performance beyond the training data time

span, the test object's valid time span is extended from the first quarter of 2020 to the first six months of 2020. Below in Fig. 9 is a histogram of SMA error on the extended time span test object.

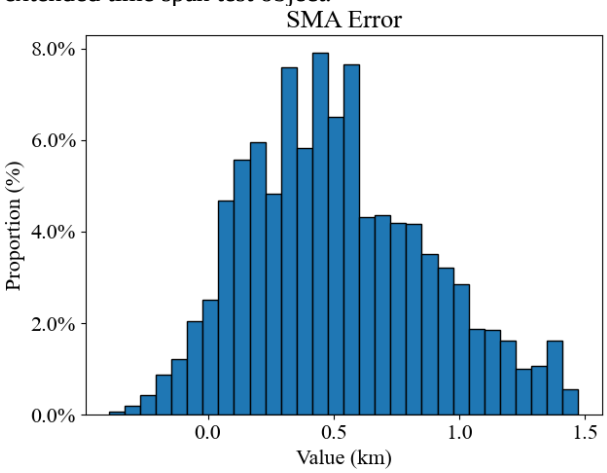

Figure 9. Histogram of SMA error on the extended time span test object after training.

The SMA errors roughly follow a gaussian curve centered at 0.5 km, showing a slight positive bias. Below in Fig. 10 is a histogram of eccentricity error on the extended time span object after training.

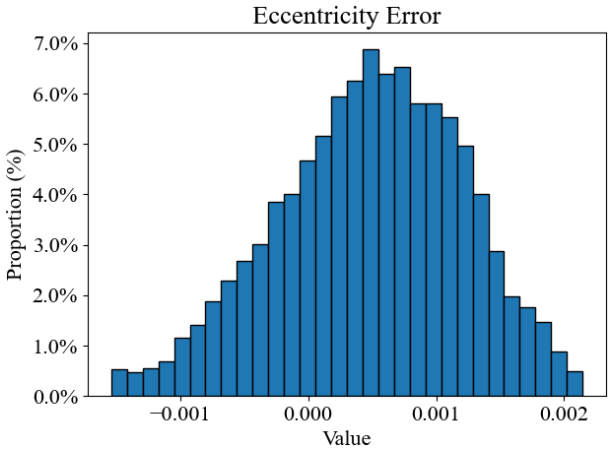

Figure 10. Histogram of Eccentricity error on the extended timespan test object after training.

The eccentricity errors carry a slight positive bias and resembles a rough gaussian curve with slight positive skew. Finally, the histogram of inclination error on the extended time span object after training is below in Fig. 11.

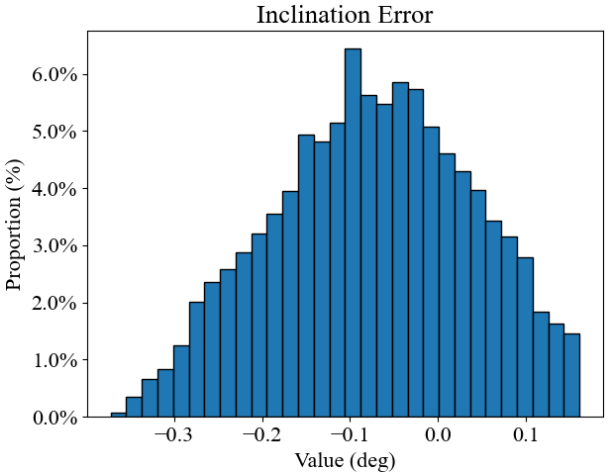

Figure 11. Histogram of Inclination error on the extended timespan test object after training.

While the test object offers a unique set of orbital parameter inputs for the model and test pairs that are beyond the training time span of the model, the model shows good performance in predicting orbital parameters.

To capture the performance across a wider range of orbital parameters, the evaluation is repeated on the training objects, but limiting the evaluation to the three months after the training time span. I.e., the model is trained on ten objects in the first quarter of 2020 and then tested on the same objects in the second quarter of 2020.

Below in Fig. 12 is a histogram of SMA error on the training objects in the 2nd quarter of 2020.

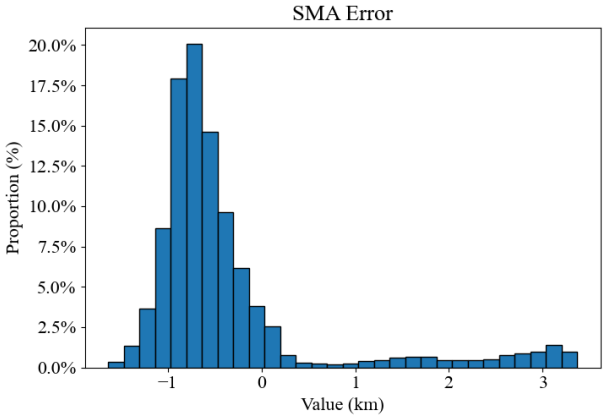

Figure 12. Histogram of SMA error on the training objects within the 2nd quarter of 2020.

There seem to be two distinct trends from likely different objects within the test set. There is a strong gaussian curve centered between -1 and 0 km error, with a "fat tail" that lingers above 3km of error. Below in Fig. 13 is a histogram of eccentricity error on the training objects in the 2nd quarter of 2020.

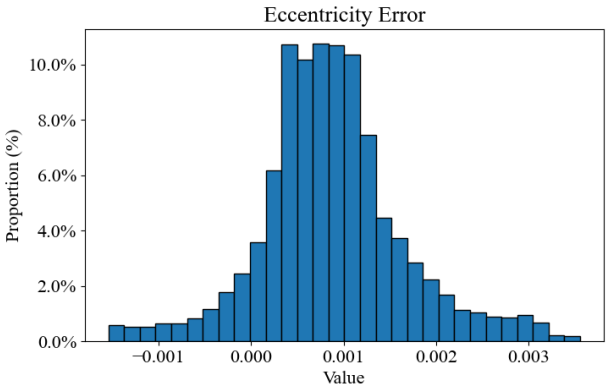

Figure 13. Histogram of Eccentricity Error on the training objects within the 2nd quarter of 2020.

The eccentricity errors on the 2nd quarter objects show a rough gaussian curve with a distinct plateau around 0.001. Finally, a histogram of inclination errors on the training objects in the 2nd quarter of 2020 is shown below in Fig. 14.

These tests show that the model has value in propagating a subset of orbital elements in time with a bounded set of errors. When the model is tested on a time span beyond when it is trained it performs with similar accuracy. The next section compares the data driven model to the numerical integration of an astrodynamics propagation engine.

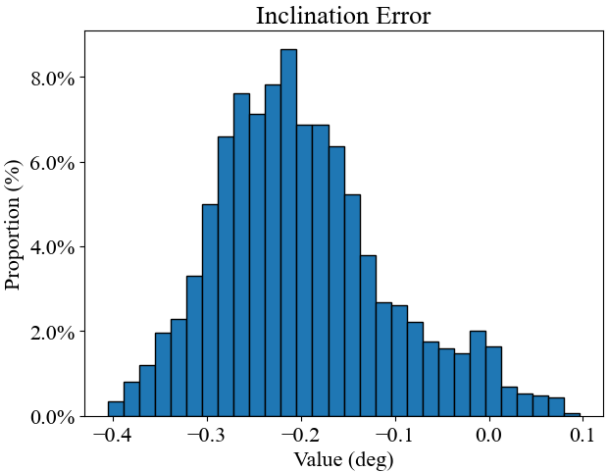

Figure 14. Histogram of inclination error on the training objects within the 2nd quarter of 2020.

8. TESTING VS NUMERICAL INTEGRATION

To investigate the performance capability of the NN approach more fully, it must be placed in context against numerical integration methods more commonly used to predict the future state of vehicles in LEO. The model is evaluated against the astrodynamics propagator in FreeFlyer, which will propagate TLE states forward in time using numerical integration through a complex gravity and atmospheric model.

The most complex numerical integration is called the FreeFlyer (FF) numerical model and is a numerical integration with spherical harmonics gravity model and analytic atmospheric model. Specifically, the FF numerical model uses an Earth gravity model with 4 zonals and 4 tesserals. It also includes gravity from the Sun and Moon and utilized an 8th order Runge-Kutta numerical integrator with 9th order error threshold checking. The analytic atmospheric model in the FF numerical model is

$$\rho(h) = \rho_0(1 + \rho_1)e^{S_{eff}h} \tag{6}$$

Where S_{eff} is the effective scale height that is interpolated from a table to account for temperature variations, and ρ_1 is a correction term that is zero by default.

Within FreeFlyer are also a Keplerian (Two-Body) and SGP4 propagator. The Keplerian propagator assumes a point mass Earth with no atmosphere, and the SGP-4 propagator is a Simplified Perturbation model that is specific to the propagation of TLE data.

A notable difference operationally between the two models is the computer clock time required to execute either the data driven or FF model. To illustrate, the FF models are evaluated on the test object over the first 180 days of 2020. Below in Fig. 15 is a plot of execution time for different models vs propagation time. Included are the DD model, the full force model numerical integrator, the SGP-4 integrator, and the Keplerian (Two-Body) model.

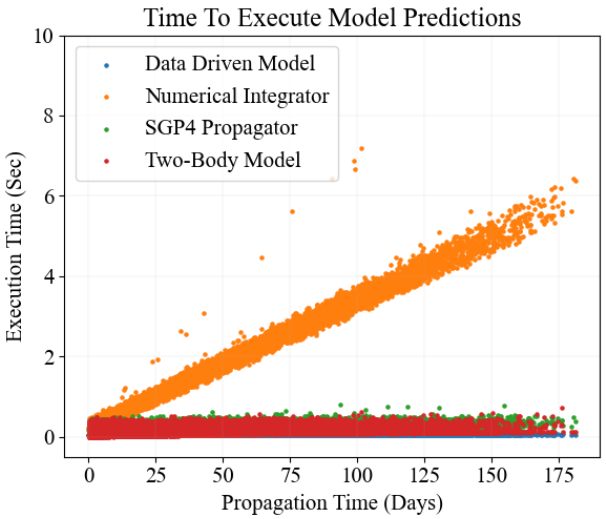

Figure 15. Clock time to execute model predictions between FF and data driven models.

It is clear from Fig. 15 that the data driven model execution is insensitive to propagation time span. An 80-day propagation takes the same amount of clock time

as a 10-day propagation. This is not true for the FF numerical model, which takes longer clock time to execute propagations of longer time spans. Both the Keplerian and SGP-4 integrators are considerably faster than the FF numerical model, but the DD model is still slightly faster in execution time. This capability can be important for highly parallelized applications that could benefit from rapid calculations of propagations.

While the data driven model can be executed on any propagation time with the same execution time, its accuracy is less than that of the FreeFlyer model. To illustrate, the errors from each model are extracted from a test against the extended timespan test object. Below in Fig. 16, the SMA errors from each model are plotted as a function of propagation time.

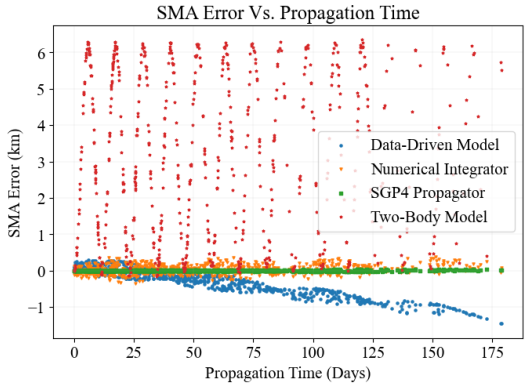

Figure 16. SMA error vs Propagation time for each model.

Each model shows unique behavior: the Two-Body model has the highest errors, and the Data-Driven model shows a negative trend. The SGP-4 propagator shows the smallest SMA errors over any time span. Below in Fig. 17 are the eccentricity errors from each model plotted as a function of time.

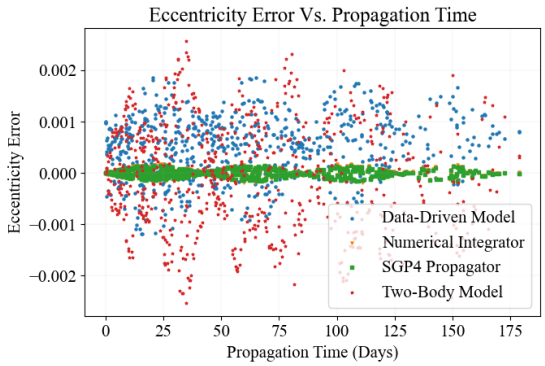

Figure 17. Eccentricity Error vs propagation time for each model.

For eccentricity, the Data-Driven model is in-family with the Two-Body model and has slightly lower errors. The Numerical and SGP-4 integrators have nearly identical performance and behavior. Finally, the inclination errors from each model as a function of time are shown below in Fig. 18.

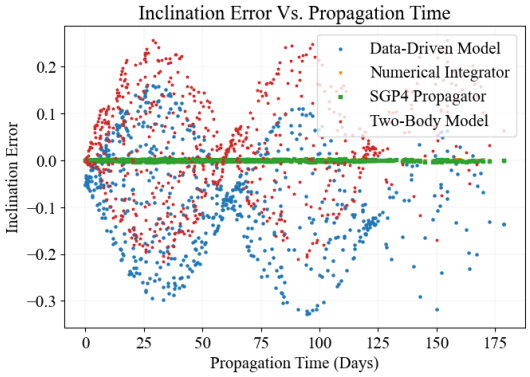

Figure 18. Inclination error vs propagation time for each model.

Similarly, the Data-Driven and Two-Body models are in-family and show similar behavior while the SGP-4 and Numerical integrators both show smaller errors and similar behavior. To summarize the performance comparisons over 6790 TLE pairs, the aggregate statistics of the errors of each model are shown below in Table 1.

Table 1. Propagation error statistics from each model over 6790 test TLE pairs.

		Data Driven	Numerical	SGP4	Two-Body
SMA error (km)	mean	-3.281E-01	3.750E-02	1.451E-04	3.155E+00
	1-σ	3.611E-01	1.239E-01	1.102E-02	2.190E+00
Ecc. Error	mean	4.530E-04	-2.993E-08	-3.857E-06	5.534E-06
	1-σ	6.297E-04	7.787E-05	7.203E-05	1.011E-03
Inc. Error (deg)	mean	-6.850E-02	-4.261E-05	-1.532E-04	3.632E-02
	1-σ	1.105E-01	8.092E-04	1.304E-03	1.101E-01
Execution time (sec)		0.40	15066.33	1221.96	896.46

Some caveats are worth considering. The FF models all propagate a full set of state parameters while the DD model encodes the state information to only "orbit shape" as defined by SMA, eccentricity, and inclination. The DD model could benefit from additional training on more objects, which would be enabled with automated TLE querying. Regardless, the stark calculation speed difference is notable, three orders of magnitude faster than the next quickest method (Keplerian).

9. DISCREPANCY MODELING

The model described thus far in this paper are fully empirical or data-driven models. No equations of motion are utilized, and no insight into the machinations

of the model is available beyond the matrix of weights within the system itself. While the data driven model has shown predictive value when properly orchestrated and trained, it lacks the intuition of an analytical force model and by itself not as accurate as numerical integration.

A method to enhance numerical integration with force modeling from first principles is to reformulate the problem to isolate the scope of the NN to model the discrepancy between numerical integration and True states. Currently the data-driven model acts as an incomplete replacement to numerical integration, and the predictive value of propagating orbit "shape" as it decays in Earth's atmosphere has been established. By refactoring to discrepancy modeling it may be possible to enhance prediction accuracy beyond current capabilities with numerical integration.[10]

10. CONCLUSIONS / FORWARD WORK

A neural network approach to predicting the "shape" of a Low Earth Orbit (excluding specific true or mean anomaly) over time due to orbital decay from atmospheric drag is described. By leveraging the large set of TLE data available online, it is proposed that such a system could be trained on TLEs from across multiple vehicles over a common time span. The process to develop a neural network and organize its training data for this task is described.

The analyses performed show that a neural network has value in propagating orbital states forward in time. A sensitivity analysis was performed on the validation data to optimize the network hyperparameters. The neural network is then trained on the training data to produce a data driven model. That model is tested to ascertain its performance in accuracy and speed. These analyses are also performed on a numerical astrodynamics propagator as a comparison against a typical spacecraft propagation. The astrodynamics propagator in FreeFlyer with default settings is utilized as the benchmark.

The data driven model is tested on the test object and against multiple propagators available in FreeFlyer, namely the Keplerian model, the SGP4 propagator for TLEs, and a full force model numerical integration. The data driven model's execution speed is invariant to propagation time and is faster than any of the FreeFlyer propagation models. The data driven model returned comparable error performance to the Keplerian model, but with larger errors than the SGP4 and numerical model.

The next steps are to greatly increase the size of the training data through automation, incorporate discrepancy modeling, and develop a system that is continually trained on new publications of TLEs. Furthermore, analyses on possible applications of this system shall be investigated, such as collision avoidance analysis, orbital decay prediction, maneuver detection, and onboard computing.

11. REFERENCES

1. Tapping, K. F. The 10.7 cm Solar Radio Flux. Space Weather Journal Vol 11, Issue 7, July 2013
2. Fry, C. A., McLaughlin, C. A. Optimizing a Long Short-Term Memory Neural Network to Forecast Solar Flux. AAS Astrodynamics Specialists Conference, August 2022
3. Wang, Y., Bai, X. Comparison of Gaussian Processes and Neural Networks for Thermospheric Density Predictions During Quiet Times and Geomagnetic Storms. AAS Astrodynamics Specialists Conference, August 2022
4. U.S. Standard Atmosphere, 1976. National Aeronautics and Space Administration. NASA-TM-X-74335
5. 27-Day Outlook of 10.7 cm Radio Flux and Geomagnetic Indices. Online: https://www.swpc.noaa.gov/products/27-day-outlook-107-cm-radio-flux-and-geomagnetic-indices
6. Pesnell, W. D., Schatten, K. H. An Early Prediction of the Amplitude of Solar Cycle 25. Solar Physics Vol 293, July 2018
7. Two Line Orbital Element Set File. Online: https://ai-solutions.com/_help_Files/two-line_element_set_file.htm
8. Rasamoelina, A. D., Adjailia, F., Sinčák, P. A Review of Activation Function for Artificial Neural Network. IEEE International Symposium on Applied Machine Intelligence and Informatics, Jan 2020.
9. Raissi, M., Perdikaris, P., Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Volume 378, 2019, pp 686-707
10. Brunton, S. L. (2022, August 5), Discrepancy Modeling with Physics Informed Machine Learning