Mapping and Correcting for Solar Gravity Perturbations via Artificial Neural Nets for Flyby Trajectories

AE295B Final Report
Presented to
The Faculty of the Department of Aerospace Engineering
San José State University (SJSU)

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

by

Takoua E. Bejaoui

July 27 , 2021

Approved by

Dr. Jeanine Hunter

Faculty Advisor

ABSTRACT

This study lays a foundation for the next generation of in-flight orbital determination using artificial intelligence. Artificial neural networks (ANN), a branch of recurrent neural networks (RNNs), are used to detect solar perturbations to optimize trajectories by locating the positions and magnitudes of minimum burns for flyby missions. The major findings are: how to establish a proper working NN model for a small orbital dynamics data sample, choice and reasoning of open source platform used, NN model, and a defined process to initiate the necessary contour mapping of the solar perturbations. A long-short term memory (LSTM) is used for the NN model. NASA's General Mission Analysis Tool (GMAT), using the Runge-Kutta89 numerical integrator, is used for orbit propagation. Open source platforms Google Colab (using Tensorflow/Keras) and Jupyter notebook are used for NN training. The necessary contour mapping is done via the weights captured for every feature and time step at each epoch during training. The example in this paper is a flyby of Venus followed by a trajectory to Saturn.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude for the following people, for if it were not for their collaboration, this project would not have been possible:

Dr. Hunter and Dr. Mourtos, for supporting and maintaining the establishment of the Aerospace Engineering (AE) department at SJSU, as well as, providing opportunities for students to embrace their diverse set of interests.

Dr.Hunter (chair advisor of project) for her constant support and collaboration, as well as, asking the right questions to better guide the direction of the project.

Mayra Lopez, Grad student at the AE Dept. for her second helpful hindsights and analytical attention to detail within the realms of, both , astrodynamics and NN's.

Samneet Singh, Grad Alumni from the AE Dept. at SJSU, for his insightful support with respect to the initial strategy, as well as. set up with respect to the orbital dynamics of the project.

Dr. Fabio Di Troia (from the Computer Science (C.S) department at SJSU), who is a committee member of the project, and we thank him for his additional insight into hard coding and optimization techniques to be used within the NN framework.

Nelson Wong and Gaurav Kupta (SJSU Graduate students from the Computer Engineering (C.E) and C.S departments), leaders of the Robotics Team, who initiated the first machine learning summer workshop (using the CS231n Stanford class' materials as its base), which has established my foundation in the realm of NN, and their continuous support and guidance throughout the duration of this project.

Dr. Mark Stamp, Dr. Wendy Lee and Dennis Dang, also from the C.S department, who were kind enough to spend some time in further directing my research and share invaluable documentation.

Theodore Hendricks, a SJSU alumni from the AE department, who aided whenever I sought him out for additional clarification and guidance into the inner workings of GMAT, orbital mechanics, second opinion on problem set ups, edited versions of the Lambert's problem via Matlab, as well as, miscellaneous info sheets in regards to the space industry.

Lakhveer for her patience and continued interest in solving random problems with respect to the mathematics of the LSTM model.

Last, but not least, my mother, Sallouha El Bejaoui, who was always a firm and continuous supporter and long-term motivator throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS & SYMBOLS

HEO	high earth orbit
LEO	low earth orbit
TESS	Transiting Exoplanet Survey Satellite
TOI	Transfer Orbit Insertion
SOI	sphere of influence
IDE	Integrated Development Environment
API	Application Programming Interface
CUDA	Compute Unified Device Architecture, parallel API created by Nvidia
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field Programming Gate Array
FBGA	Field Ball Grid Array
SVM	Support Vector Machine (Multiclass)
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long-Short Term Memory
GRU	Gated Recurrent Unit
ESN	Echo State Networks
ResNet	Residual Network
aNN	Artificial Neural Network (also ANN)
AI	Artificial Intelligence
G	Gravitational constant ($6.67x10^{-20}\ \frac{km^3}{kg\ s^2}$)
M	Celestial mass (kg)
μ	Gravitational parameter (G*M)
E	Eccentricity
r	Radial distance
i	Inclination angle
ω	Argument of perigee
Ω	Right ascension of the ascending node (RAAN)
H	Specific angular momentum
B	Semi-minor axis
Δv	Burn magnitude
B	B-plane vector
γ	Flight path angle

Chapter 1 - Introduction

1.1 Motivation

1.1.1 Background

The 20th century marked the beginning of space exploration and the development of the computerized mathematical methods required to accomplish it. Past missions, such as the Apollo 11 through 17, depended on these methods and laid the foundation for future space missions' characterization and trajectory determination. Automated math models, such as simulated propagators, use computational numerical methods to determine future trajectories for spacecraft missions.

One such example is the Hill sphere approximation, which is the volume of space that contains the stable orbit on which a satellite will find itself [1]. The Hill sphere radius is the location where the radial force, between a spacecraft and a secondary mass, transforms to a tidal force that is at hydrostatic equilibrium, and can be further affected by the gravitational perturbations introduced by a larger mass [2]. The Hill sphere radius determines whether a problem can be considered a two-body or three-body problem. This type of approximation can be visually mapped via vector or contour plotting, as shown in Figure.1, where the Lagrange points and the space bodies with different potential fields are also shown. Other popular methodologies within astrodynamics include, but not limited to, the Lambert's problem and orbital phasing.

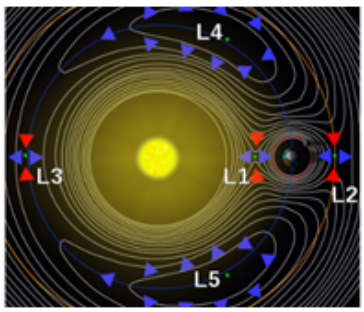


***Figure 1**: Contour plot of the Hill sphere mapping (often discussed in association with the Roche limit) about the Sun and Earth as well as the five Lagrange points[3] .*

The Age of Globalization & Technology in the 20th century, facilitated the Age of Information in the 21st century, and the combination of both gave birth to the internet, autonomous design, and 'smart' commercial applications. The establishment of the Internet of Things (IOTs) generated unlimited and continuously streaming data from, both, sensor and cloud based intelligent embedded systems. Thus, it was only a matter of time before engineers and scientists found creative solutions and were able to use large amounts of data for visualization,

communication and preprocessing. In regards to the aerospace industry, porkchop plots (Fig.2) are an example of data visualization, where large amounts of historical data was used to generate contour plots to determine launch and arrival dates, as well as the minimum burns, or Δv 's, for interplanetary flights. This in turn brought forth machine learning (ML) and brought back artificial neural networks (ANN) (which originated from the 1940's) to the forefront of the political, economic and social spheres of human life.

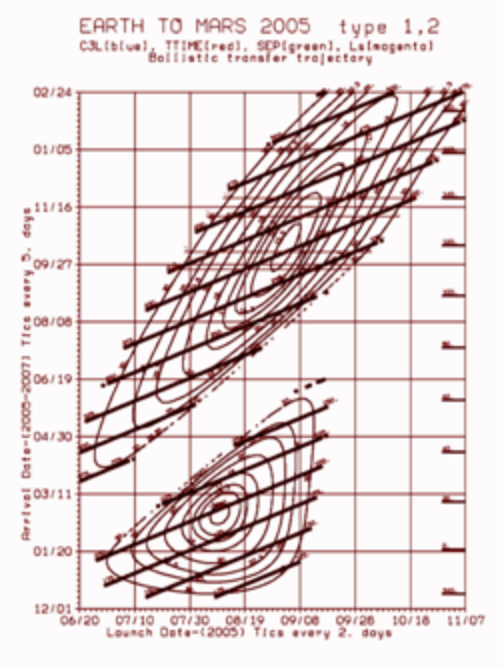


***Figure 2**: Porkchop plots by NASA [4].*

A neural network (NN) is a type of deep learning (DL) algorithm. DL is a subset of ML, and these two, DL and ML, are in turn subsets of, what is commonly referred to as, artificial intelligence (AI). NNs are so named because their models resemble the neural structure of the human brain (as shown in Figure 3). NN is divided into three main methodologies: supervised, unsupervised and reinforced learning. The difference between supervised and unsupervised models is that the former requires output training labels, provided by an operator, while the latter classifies raw clustered data without any output labels to guide its decision-making process. Reinforced learning is often administered with live machines, such as robots, with an operator to guide the machine via reward (similar to the B.F Skinner psychological/operative conditioning) to distinguish between what is correct and what is not. Regardless of the methodology selected, all three require data and can be in the form of statically stored or live feed, as the machine 'learns' to adapt to its surroundings.

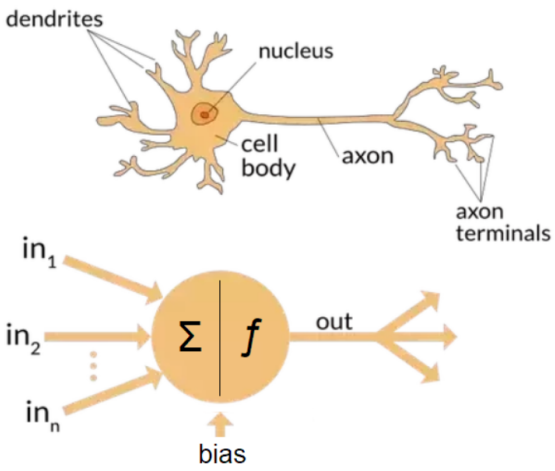


***Figure 3**: Comparison between a biological and artificial neural structure [5].*

The current trend in dynamic and energy-based systems is combining classical/modern control system design with ML and NN infrastructure. In the aerospace industry, this is evident with the creation of unmanned air vehicles (UAVs) and in-situ planetary exploration, such as quad copter drones and the Mars land rovers, respectively. However, not much has been done in the way of autonomizing spacecraft in flight or in mission planning (in contrast to the attitude, determination and control system (ADC)). Currently trajectories are monitored and controlled by ground stations or JPL's deep space network (DSN) using radar technology to track interplanetary travel [6] and the ISS's telecommunication centers. The risk associated with such a communication link is a delay in feedback causing unprecedented loss of the spacecraft itself. Examples of where loss of signal (LOS) has occurred between spacecraft and ground stations are Cassini, Voyagers 1 and 2, as well as one of the most recent missions: JPL's Surveyor II. The loss of communication is usually due to: malfunctioning hardware and mishandling of data at the ground stations with respect to the spacecraft's command frame or unprecedented perturbations within space (such as magnetic or solar perturbations) [7]. Figure 4 displays a table of the most recent manual, semi-autonomous and autonomous Navigation, Guidance and Control (NGC) units aboard certain spacecraft, but not all have been put into practice, as indicated in Figure 5. The autonomous design is not commonly used due to its additional weight, hardware and cost requirements. Thus, there is much to be explored in the area of 'smart' technologies in orbital determination to capture real time data for mission optimization.

System	Advantages	Disadvantages
Ground Tracking	Traditional approach Methods and tools well established	Accuracy depends on ground-station coverage Can be operations intensive
TDRS Tracking	Standard method for NASA spacecraft High accuracy Same hardware for tracking and data links	Not autonomous Available mostly for NASA missions Requires TDRS tracking antenna
Global Positioning System (GPS); GLONASS	High accuracy Provides time signal as well as position	Semi-autonomous Depends on long-term maintenance and structure of GPS Orbit only (see text for discussion) Must initialize some units
Microcosm Autonomous Navigation System (MANS)	Fully autonomous Uses attitude-sensing hardware Provides orbit, attitude, ground look-point, and direction to Sun	First flight test in 1993 Initialization and convergence speeds depend on geometry
Space Sextant	Could be fully autonomous	Flight-tested prototype only—not a current production product Relatively heavy and high power
Stellar Refraction	Could be fully autonomous Uses attitude-sensing hardware	Still in concept and test stage
Landmark Tracking	Can use data from observation payload sensor	Still in concept stage Landmark identification may be difficult May have geometrical singularities
Satellite Crosslinks	Can use crosslink hardware already on the spacecraft for other purposes	Unique to each constellation No absolute position reference Potential problems with system deployment and spacecraft failures
Earth and Star Sensing	Earth and stars available nearly continuously in vicinity of Earth	Cost and complexity of star sensors Potential difficulty identifying stars

Figure 4*: List of NGC systems used in various spacecraft along with their dis/advantages [6].*

System	Basis	Status	Determines	Typical Accuracy (3σ)	Operating Range	Comments	Manufacturer
Global Positioning System (GPS)	Network of navigation satellites	Operational	Orbit*	15 m–100 m in LEO	LEO only	Semi-autonomous	Motorola, Rockwell
Microcosm Autonomous Navigation System (MANS)	Observations of Earth, Sun, and Moon	Flight tested in 1993	Orbit, attitude, ground look point, Sun direction	100 m–400 m in LEO (using only Earth, Sun and Moon)	LEO to GEO, lunar and planetary orbits	Can use other instruments (GPS receiver, star sensor, IMUs) to improve accuracy	Microcosm
Space Sextant	Angle between stars and Moon's limb	Flight tested	Orbit and attitude	250 m	LEO to GEO	Not being actively marketed for space at the present time	Lockheed Martin
Stellar Refraction	Refraction of starlight passing through the atmosphere	Proposed, some ground tests done	Orbit and attitude	150 m–1 km	Principally LEO	Could use attitude sensor data	
Landmark Tracking	Angular measurements of landmarks	Proposed, observability conditions are uncertain	Orbit and attitude	Several kilometers	Principally LEO	Could, in principle, use observation payload data	
Satellite Crosslinks	Range and range rate or angle measurements to other satellites in a constellation	Proposed; may be used on communication s constellations	Orbit†	Theoretically as good as 50 m	Principally LEO	Operation with less than full constellation can be a problem; has no absolute position reference	
Earth and Star Sensing	Observe direction and distance to Earth in inertial frame	Proposed	Orbit and attitude	100 m–400 m in LEO	LEO to GEO, planetary orbits	Similar to MANS with higher accuracy and availability	

* Attitude determination using GPS receivers has been demonstrated. Multipath limits accuracy to ~ 0.3 to 0.5 deg.
† Could, in principle, be used for attitude determination as well.

Figure 5*: In-depth characterizations of the different types of NGCs for spacecrafts [6].*

1.1.2 Significance

The significance of this project is that it paves the way for combining past classical control systems, established by the Fly-by-wire projects in the early 1970's [8-9], with 'smart' technologies, such as ML and DL, to autonomous navigation. Current automated control system design in spacecraft is limited for attitude control of the spacecraft's propulsion systems, electrical supply maintenance, and antenna orientation, due to the additional weight and cost of past projects that tried to bring on board automated versions for the NGC unit [6]. Thus, integrating NN to create an onboard intelligent control system module that would minimize potential losses with respect to the energy usage, cost of the overall spacecraft (especially concerning weight), is the next step in futurizing aerospace flight. The NN mapping of the solar perturbation field will be used specifically for interplanetary flyby trajectories. The spacecraft will use the NN to minimize trajectory corrections maneuvers (burns) by 'free-riding' on the solar perturbation field when transitioning from one space phase to another.

1.2 Literature Review

The initial concept of the artificial neuron began during the 1940's by McCulloch and Pitts , but this logical rendition of the early NN was constrictive, and the first perceptron was not discovered until the 1950's by Frank Rosenblatt [10]. Rosenblatt's perceptron ran under a binary formulation to determine whether or not the neuron will activate instead of being dependent on the dot product of the weights and inputs (as was done with Pitts'). Figure 6 depicts the early framework of the artificial neuron. It was not until 1998 that neural networks made a reappearance in the industry, and it was called the LeNet-5 and is considered to be the first 'famous' Convolutional Neural Network (CNN) architecture [11]. The CNNs are categorical classifiers and so all the later developments of this model, such as the AlexNet (2012) and the VGG-16, examined images and text-based data. Figure 7 presents a few of the commonly known CNN architectures that helped shape the neural network algorithms that are used today.

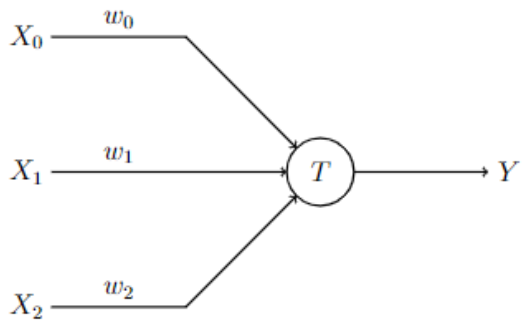


***Figure 6**: The first artificial neuron model developed during the 1940's by McCulloch and Pitts [10].*

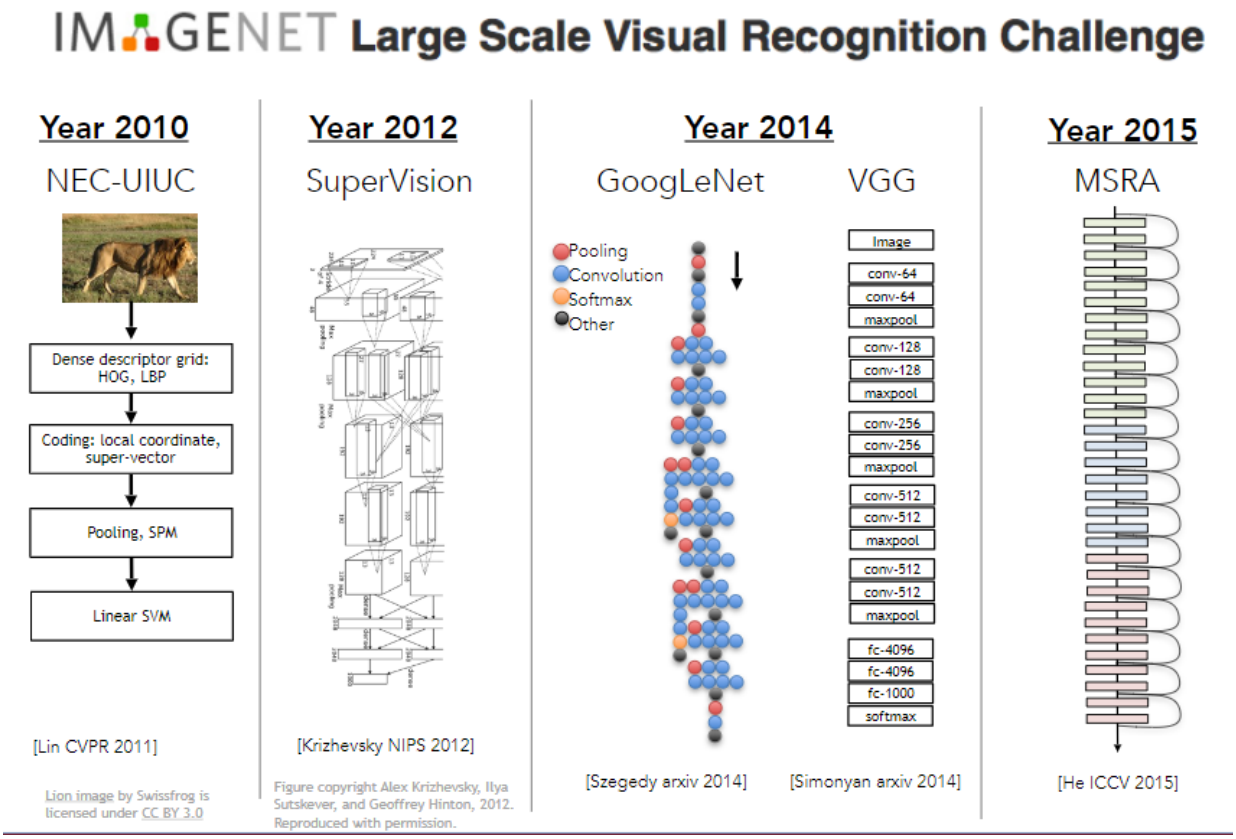


***Figure 7**: Few of the earlier popular CNN models and their architecture [12].*

In 2018, Dr. Stijn De Smet published two research papers that discuss how to use neural nets for solar gravity driven orbital transfers in the Martian and other planetocentric spheres of influence [13-14]. His research established the foundation for a NN based propagator for space missions.This analysis compares NN models that used data generated by the two numerical approaches, the Circular and Elliptical Hill sphere methods, to complete the Poincare mapping. The error was calculated by using GMAT data to verify the results, which indicated a high level of accuracy. The type of NN model created by Dr. Stijn was a fully condensed layer that had a 15 x 15 matrix (15 neural inputs and outputs, along with 15 hidden neurons). The NN was used to determine the initial conditions with respect to the expected final destinations, as well as, to identify imperfect maneuvers on the transfers [13].

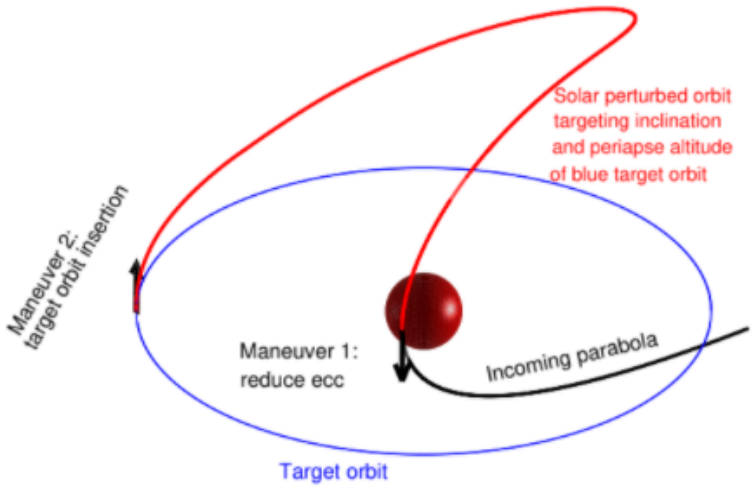


***Figure 8**: Schematic of the transfers by D.Stijn et al [13-14].*

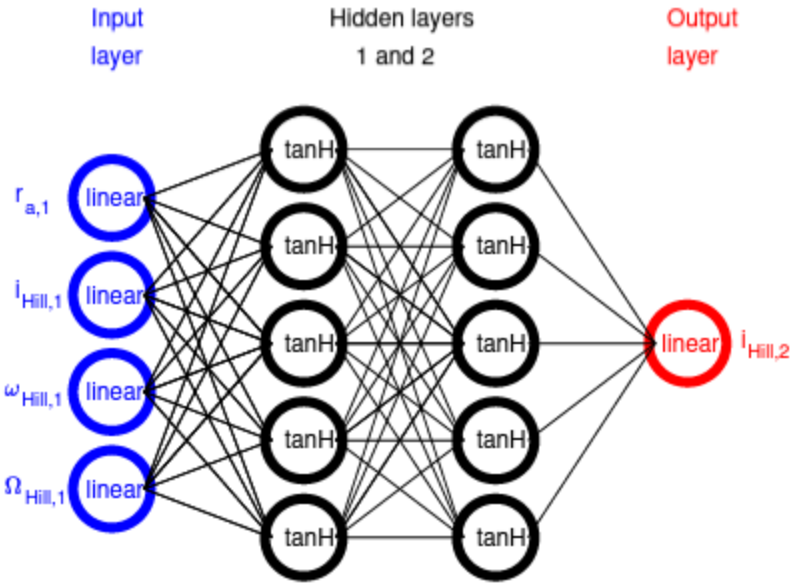


***Figure 9**: NN architecture from Dr. Stijn's research. Used a combination of linear and tanh activation functions within the cells. The five Keplerian elements that are used as inputs are designated on the left hand side of the model; while the output is at the far right. [13]*

There are two general categories of deep learning: supervised and reinforcement. While Dr. Stijn's research was in supervised learning, the incorporation of reinforcement learning with classical/modern control system design was done by Dr. Howell et al [15]. This study was to provide a 'lightweight' solution [15] with respect to the NGC subsystem's physical weight requirements due to the heavy computational algorithm that is necessary for non-linear dynamic modeling. The test subject, a spacecraft which is addressed as the 'agent', undergoes several trial runs through different ML models, such as K-Nearest-neighbor (or K-dimensional tree), Hidden Markov Model (HMM) and Monte Carlo. The astrodynamics model consists of two planetary bodies (Earth and Moon) and a spacecraft which transfers from one closed orbit to another. The ML algorithms receive historical data to train the agent to determine the optimal locations of the burns for the expected transfers without the aid of the original controller. Perturbations, in position and velocity, are introduced into the model. The reward and penalty signals are administered to guide the NN's training process (Figure 10). The outcome of the study indicated that the agent does complete the expected transfer, however it is offset by 30 km and does not meet the defined criteria [15].

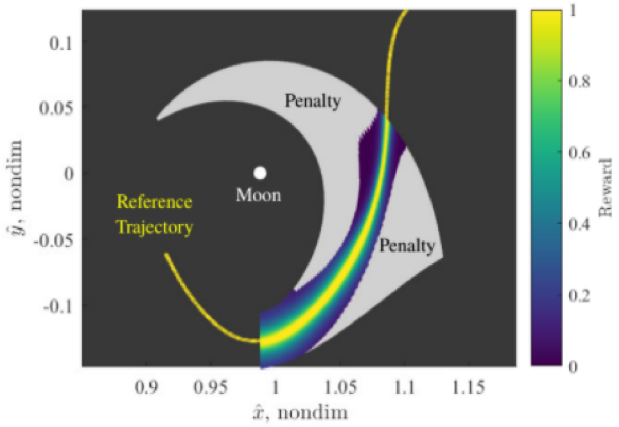


Figure 10*: Penalty and Reward plot as agent learns the expected trajectory via RL [15].*

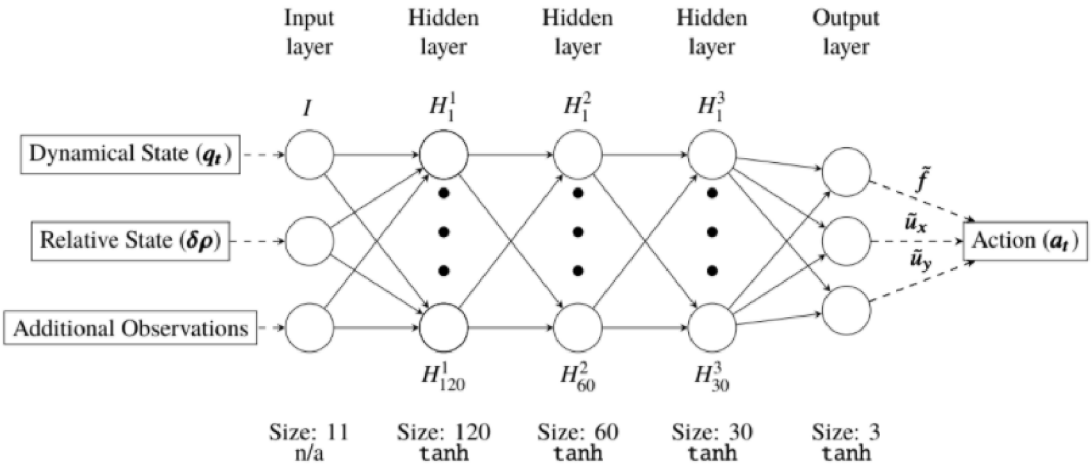


***Figure 11**: NN architecture for the RL modeling. The input contains 11 values and is mapped to 120 neurons [13]. Output values are thrust components [15].*

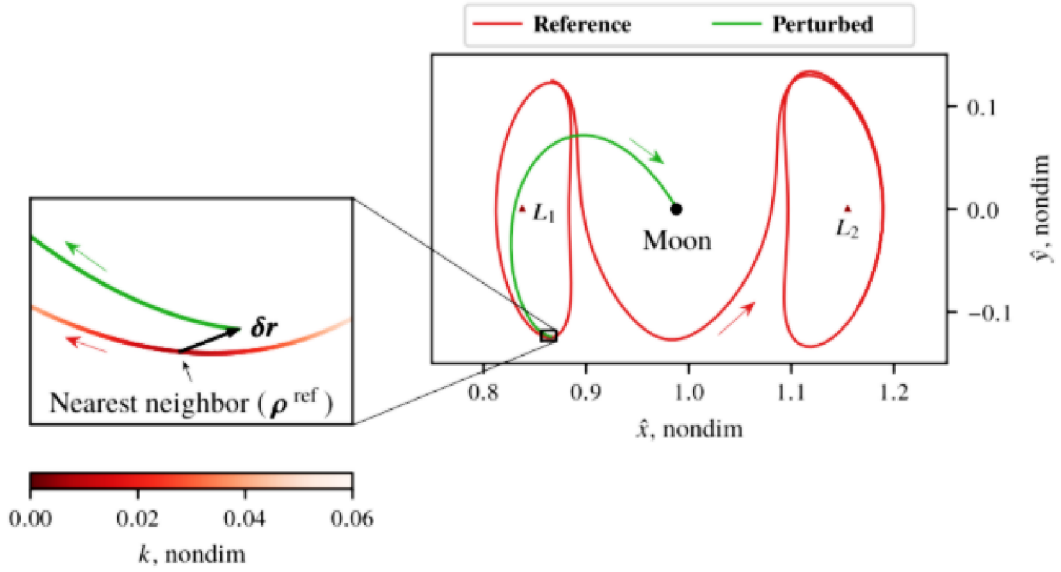


***Figure 12**: Diagram of agent's reference trajectory along with the Moon's perturbation between two liberation points about closed orbits[15].*

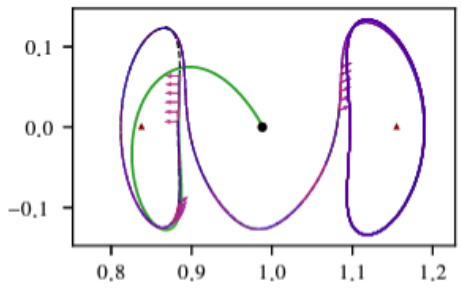
Arrival condition not triggered		Agent completes the transfer, but does not reach the defined arrival criteria of 30 km and 50 cm/s within the maximum number of time steps. For the sample case, in 150 time steps (≈ 130 days), the agent is within 2 km and 2 cm/s of arrival. Given more time steps, arrival criteria is eventually met.

***Figure 13**: Last run of agent as it completes expected transfer but does not arrive at the expected arrival spot at the designated time step [15].*

Since automation is popularly used for in-situ exploration, Dr. Bassil [16] created a supervised learning approach to guide the Mars rover across obstacles that could have prevented it from completing its sampling mission. The NN, with cells that are linearly activated, is made up of three layers (input, hidden and output). Results indicated that the simulated rover was able to avoid being deterred, however, parallelization computation was found necessary for robust computation and execution time. This means that a GPU (and perhaps even incorporating CNN or Transformers) is essential in training the model. However, the current CPUs and FPGAs are inadequate for such exercises.

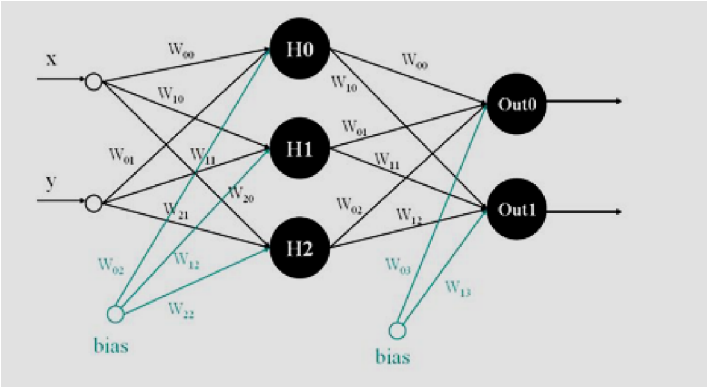


***Figure 14**: Three layered NN model with linear activation [16].*

1.3 Project Proposal

1.3.1 Objective

The objective of this project is to incorporate ANN to create a mapping of solar perturbations for cost optimization of the spacecraft while it conducts interplanetary flyby missions to the outer planets. This supervised learning model will guide the spacecraft in the path that will minimize burns (Δv), or the maximum alignment to any desired trajectory, needed to correct for the influences of the solar gravity perturbations. The analysis for the flybys will be done at different radii of approach to determine the trajectory that will require the least amount of corrections (or burns). The flybys will primarily be conducted about Venus, Mars, Jupiter and Saturn.

1.4 Methodology

1.4.1 Approach

This project will be carried out in three stages:

- **Establishment of a recurrent neural network (RNN) model:** that takes in temporal based data, and creates a sequence by sequence analysis using long-short term memory (LSTM) layers.
- **GMAT propagator to collect data sample(s):** will be used for batch test runs to finalize the type of NN architecture for training the NN. The software is used to collect the perturbed and unperturbed data for the particular spacecraft mission.
- **Astrodynamics theory:** Kepler's Laws and patched conics (two-body problem) to calculate the appropriate ephemerides and epoch.

The NN predicts perturbed data of six features (the three translational vectors of position [X, Y, and Z] as well as for the velocities [Vx, Vy, Vz]). In the first stage, a small data sample will be collected for a batch test run to overfit (exact or a linear regression fit) a 'baby' NN model, so as to finalize the type of architecture for the actual (much larger) data that will be used for training the NN.

In the second stage, all samples and data to be collected are done through the GMAT propagator. The software is used to collect the perturbed and unperturbed data for the particular spacecraft mission.

In the third stage, the spacecraft will conduct a gravity assist maneuver about Venus and use the heliocentric velocity gained to propel it to Saturn. However, before data collection can ensue, the correct epoch must be selected, for the celestial bodies are not stationary objects and have their own orbital periods and angular velocities. The epoch selected must have Earth trailing

Venus and both trailing Saturn, for it ensures that the spacecraft will not be traveling against the retrograde (counter-clockwise) spin of the Sun.

In-depth discussion about data collection and preparation, as well as, the astrodynamics involved can be found in Chapter 2.

In-depth discussion of the NN can be found in Chapter 3.

1.4.2 Theory and Principles

Kepler's Laws and patched conics are defined in Curtis' book [23] and the SJSU astrodynamics course readers SJSU [17]. Frame of reference diagrams for inner to outer planet flyby trajectories, as depicted in Curtis' book, will illustrate the v_inf approach ($v_{\infty a}$) of the spacecraft traveling towards Venus, as well as, v_inf departure ($v_{\infty d}$) , as the spacecraft leaves Venus. Lambert's problem defines the initial and final position and velocity components of interplanetary transit. Kepler's Laws are defined in Chapter 2, while Lambert's problem is presented via Matlab in Chapter 4.

1.4.3 Resources Needed

Resources to complete deliverables for the project are listed below.

Table 1.1 Potential resources and its expected acquisition date for each.

Item	**Expected Acquirement Date (and Cost)**
Google Colab (Jupyter Notebook)	Free, Online, Requires Google Drive account
Pycharms	Free download
Keras/TensorFlow (v.2.3)/Pytorch (3.2)	Free, Online, With Google Colab
GMAT v. 2018	Free download

1.4.4 Preliminary Work

Work completed so far:

- The tentative epoch selected is July 21, 2020. Further calculations will be done to provide a basis. An online propagator was used to make such a selection.
- Collected a small sample size, for a single batch test run for perturbed and unperturbed data in .csv format.
- Loaded the data into a jupyter notebook on Google Colab, successfully split the data into training and testing, normalized the data, created a basic LSTM and dense layer model with single nodes, and successfully trained the data.

- Updated and refined the objective.
- Strategized on the set up of GMAT, refined the context of the problem, and did some review on the background of GMAT and astrodynamics.

1.4.5 Anticipated Challenges and Alternative Strategies

1. Autoencoding the weights to capture solar potential vector fields may require a propagation, at which the solar potential field data is to be collected as the spacecraft maneuvers in its prescribed trajectory.
2. Difficulty in weight interpretation, will require methodology to be reformulated:
    a. Incorporating the Hidden Markov Model for direct weight interpretation.
    b. Rely on hard coding for customization of ready made wrappers given by Keras.
    c. Use the Hamiltonian or the Transformer or incorporate the decoder-encoder NN model within the LSTM.
    d. Instead of comparing position and velocity between unperturbed and perturbed data, will instead compare the force vectors that make up the solar potential gravity field.
    e. May have to use the unsupervised learning approach instead of supervised.
    f. Increase the amount of ephemerides as variables of input and/or the overall architecture of the NN.

1.4.6 Deliverables

The executable files for this project will be as follows:

- .iPynb files for NN framework
- . csv file for storing captured data
- Mathematical models
- .script and .bak files from GMAT.

1.4.7 Evaluation Metrics

The associated evaluation metrics for this project, and its future versions, will be as follows:

- Ability to maintain the ideal trajectory with flight path angle (γ) error of $\pm 5^{\circ}$.
- Ability to maintain the ideal trajectory by calculating the correct aiming radius (Δ) error of ± 1 *km* .
- Ability to calculate the ideal epoch for a transfer orbit insertion (TOI) towards a particular planet ($\pm$ 300 *Julian days*).
- Faster system response time (± 0.5 *ms*), with respect to the evaluation and decision making process of the NN controller.
- Validation loss greater than the training loss.
- Training and validation accuracy at 98 percent.

1.4.8 Timelines

Timelines are conventionally shown using a Gantt chart. The Gantt charts are seasonal, hence the first, shown below and specifically for this proposal, will be for the Summer of 2020 and is referred to as Phase A. Next, is Phase B, which is for the Fall of 2020, and last is Phase C for Spring of 2021.

Solar Perturbation Mapping with NN	Start Date	End Date	Status
Concept	30-May	31-May	Completed
Significance selection	30-May	31-May	Completed
Idea selection	30-May	31-May	WIP
Research	1-Jun	1-Jul	WIP
Literary Review	1-Jun	12-Jul	
Challenges with the current design	7-Jun	14-Jul	
Define project requirements and objective	31-May	13-Jul	
Project Planning	30-May	7-Jul	WIP
Task planning	30-May	7-Jun	
NN model defined and initiated development with Google Colab	14-Jun	1-Jul	
Last verification of draft I of NN model and objective	1-Jun	12-Jul	
Last verification of draft 2 of NN model	28-Jun	26-Jul	
Report Creation	30-May	30-Aug	WIP
Project Proposal	30-May	30-Aug	WIP
Project Progress Report - 1	7-Jun	1-Jul	
Project Progress Report - 2	13-Jul	1-Aug	
Final Project Proposal	5-Aug	22-Aug	

***Figure 15**: Gantt chart for phase A.*

Solar Perturbation Mapping with NN	Start Date	End Date	Status
Research (for phase I)	1-Jul	5-Dec	Completed
Literary Review	1-Jul	5-Nov	Completed
Addressed challenges with the current Astrodynamic model	7-Jun	14-Nov	Completed
Defined project requirements and Prep for Phase II	1-Dec	7-Dec	Completed
Project Planning	30-Aug	18-Dec	Completed
Task planning	30-Aug	7-Dec	Completed
NN model defined and initiated development with Google Colab	14-Jun	15-Aug	Completed
Last verification of draft I of NN model and objective	1-Sep	12-Sep	Completed
Last verification of draft 2 of NN model	12-Sep	26-Sep	Completed
Last verification of draft 3 of NN model	1-Oct	27-Oct	Completed
Astrodynamic/GMAT Setup for Phase II	30-Oct	7-Dec	Completed
Starting GMAT Setup and Actual Testing of NN	10-Dec	29-Dec	WIP
Report Creation (for phase I)	30-May	18-Dec	Completed
Project Proposal	30-May	28-Aug	Completed
Project Progress Report - 1	7-Jun	30-Sep	Completed
Project Progress Report - 2	13-Jul	30-Oct	Completed
Project Progress Report - 3	5-Nov	9-Dec	WIP/Completed

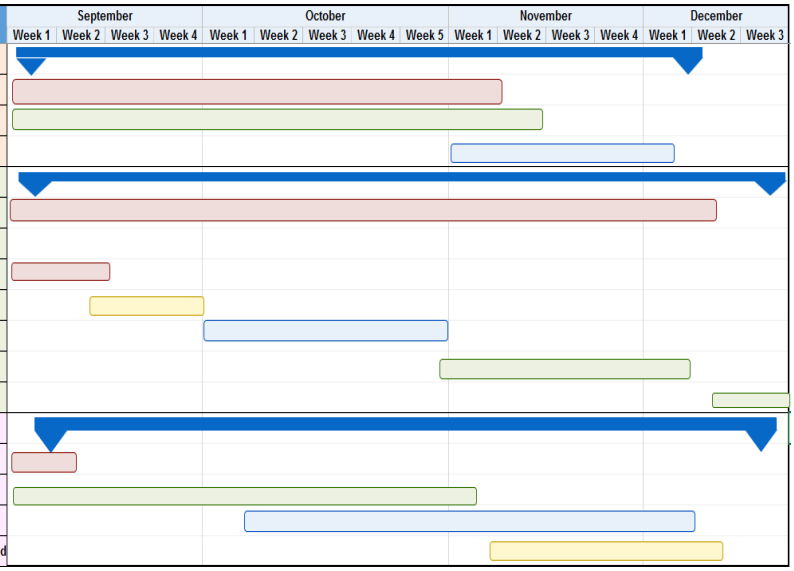
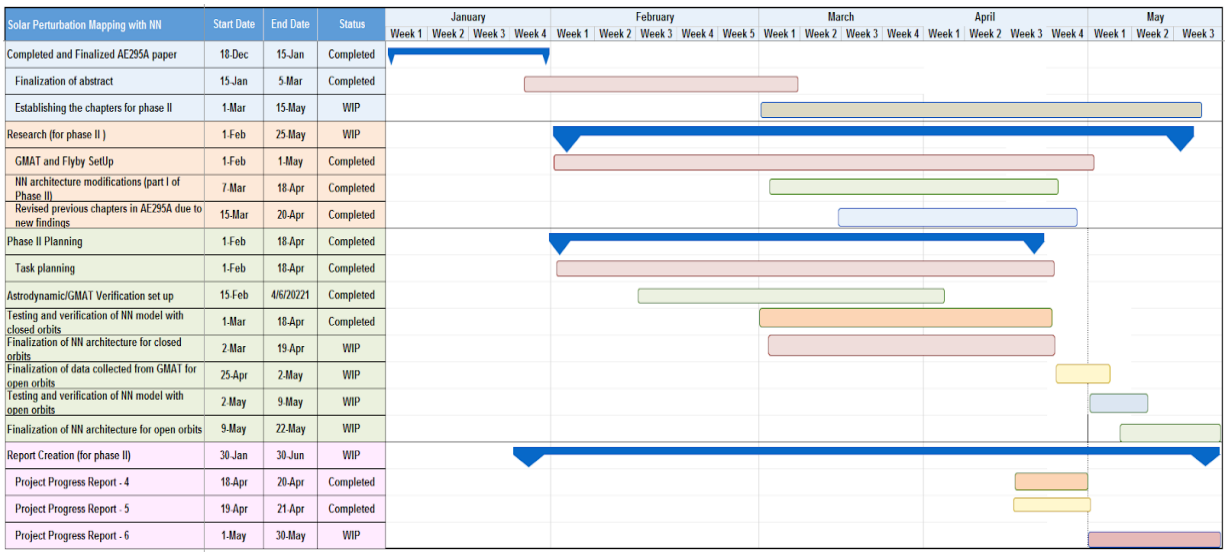


***Figure 16**: Gantt Chart B.*



Solar Perturbation Mapping with NN	Start Date	End Date	Status
Completed and Finalized AE295A paper	18-Dec	15-Jan	Completed
Finalization of abstract	15-Jan	5-Mar	Completed
Establishing the chapters for phase II	1-Mar	15-May	WIP
Research (for phase II)	1-Feb	25-May	WIP
GMAT and Flyby SetUp	1-Feb	1-May	Completed
NN architecture modifications (part I of Phase II)	7-Mar	18-Apr	Completed
Revised previous chapters in AE295A due to new findings	15-Mar	20-Apr	Completed
Phase II Planning	1-Feb	18-Apr	Completed
Task planning	1-Feb	18-Apr	Completed
Astrodynamic/GMAT Verification set up	15-Feb	4/6/20221	Completed
Testing and verification of NN model with closed orbits	1-Mar	18-Apr	Completed
Finalization of NN architecture for closed orbits	2-Mar	19-Apr	WIP
Finalization of data collected from GMAT for open orbits	25-Apr	2-May	WIP
Testing and verification of NN model with open orbits	2-May	9-May	WIP
Finalization of NN architecture for open orbits	9-May	22-May	WIP
Report Creation (for phase II)	30-Jan	30-Jun	WIP
Project Progress Report - 4	18-Apr	20-Apr	Completed
Project Progress Report - 5	19-Apr	21-Apr	Completed
Project Progress Report - 6	1-May	30-May	WIP

***Figure 17**: Gantt Chart for Phase C*

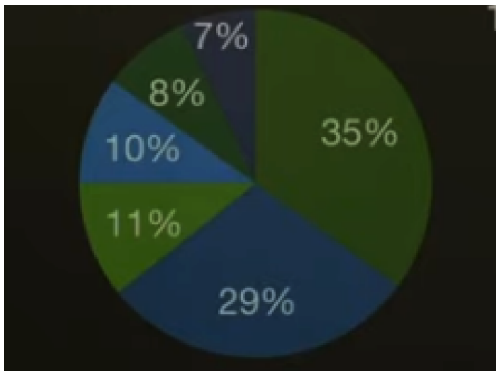
Chapter 2 - Astrodynamics Theory and Data Pre-processing

2.1 Data Visualization

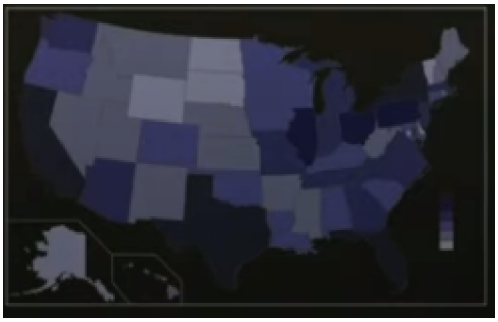
2.1.1 Background and Data Preparation

The importance and methods of data preparation for an AI model, the necessary fundamentals of astrodynamics, as well as, initial setup of GMAT that is integral for data collection are covered in this chapter. The intent of this chapter (and Chapter 3) is to allow both technical and non-technical majors to understand the setup and theory of both NNs and astrodynamics.

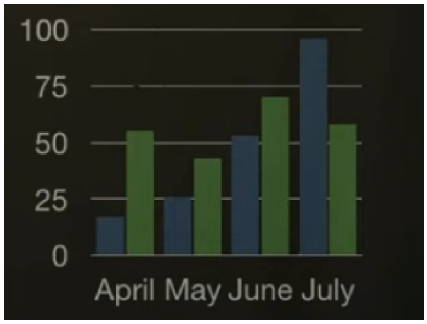
Finding a better approach to using a large data set and saving large amounts of computational loads and time is the purpose behind the concept (and acquired skill) called data visualization. Data visualization helps in determining the critical and limiting factors within a dataset, recognizing potential patterns to formulate certain conclusions, and is the precursor and defining factor to how the NN will be trained. There are seven different data types [21] and each requires using modern methods of analysis. The data types are as follows: categories and fractions, maps, distributions, high-dimensional data, written text, network data and mixed, multimodal data. Each data type has a plot type that guides in telling the story (Figure 18). The type of data that will be used throughout the course of this project is high-dimensional data, because the data samples that are extracted from the simulator will have multiple time steps and at each time step there are multiple qualitative (also called categorical) and quantitative (also known as regressional) features. However, for this project, to conserve time and resources, only a few of the quantitative features (of the overall features that the propagator provides) have been selected as the data sample to train and test the NN model.



(a)



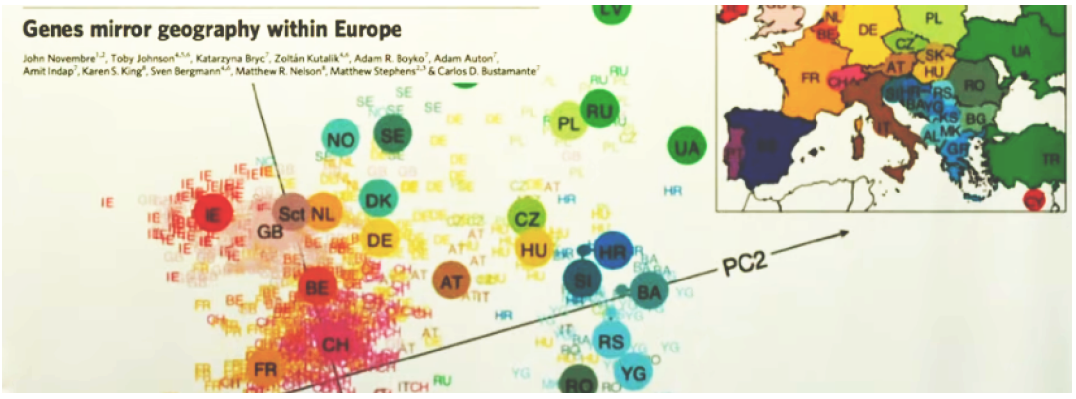
(b)


100
75
50
25
0
April May June July

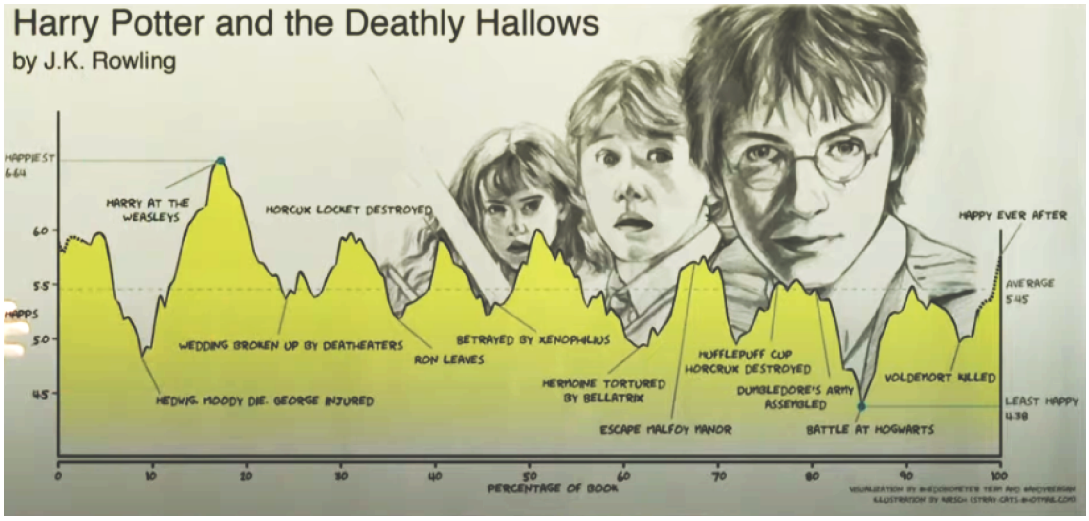
(c)

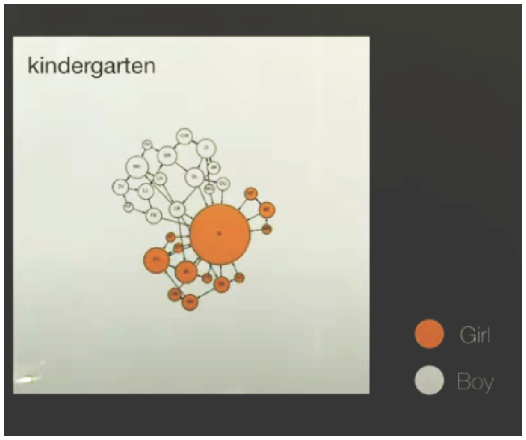

90
60
30
0
0
1
2
3
4
"Poisson" Distribution

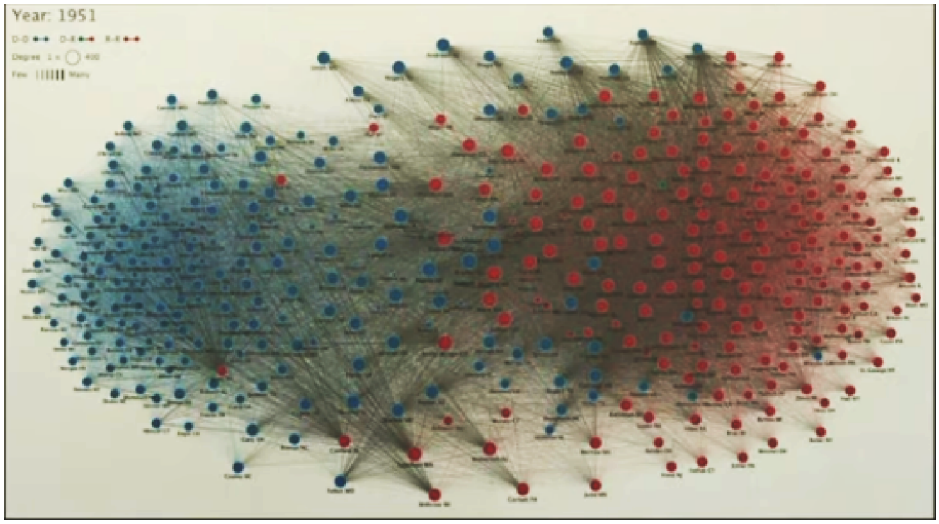
(d)


Genes mirror geography within Europe
John Novembre, Toby Johnson, Katarzyna Bryc, Zoltán Kutalik, Adam R. Boyko, Adam Auton, Amit Indap, Karen S. King, Sven Bergmann, Matthew R. Nelson, Matthew Stephens & Carlos D. Bustamante
PC2
NO
SE
PL
RU
UA
DK
IE
Sct
NL
GB
DE
CZ
HU
HR
BE
AT
BA
CH
RS
FR
YG

(e)


Harry Potter and the Deathly Hallows
by J.K. Rowling
HAPPIEST
6.64
HARRY AT THE WEASLEYS
HORCUX LOCKET DESTROYED
HAPPY EVER AFTER
AVERAGE
5.45
WEDDING BROKEN UP BY DEATHEATERS
RON LEAVES
BETRAYED BY XENOPHILIUS
HUFFLEPUFF CUP HORCRUX DESTROYED
HERMIONE TORTURED BY BELLATRIX
DUMBLEDORE'S ARMY ASSEMBLED
VOLDEMORT KILLED
LEAST HAPPY
4.38
HEDWIG, MOODY DIE. GEORGE INJURED
ESCAPE MALFOY MANOR
BATTLE AT HOGWARTS
PERCENTAGE OF BOOK
(f)


kindergarten
Girl
Boy
(g)


Year: 1951
(h)

(i)

Figure 18: *Examples of data types.* ***(a)****fractional,* ***(b)****map,* ***(c)*** *categorical,* ***(d)*** *distribution,* ***(e)*** *high dimensional data,* ***(f)*** *words,* ***(g)*** *networks,* ***(h)*** *clustering,* ***(i)*** *moving and interactive. [21]*

By selecting a propagator to extract the necessary data, the risk of retrieving 'unclean' (anomalies) data is minimized and makes data shaping, normalization and analysis that much faster. If the data collected has outliers or mismatching data types, then it is up to the operator to delete such anomalies by setting them to zero, removing them from the dataset, or dividing the data in such a way that the NN is being fed sectionally (or in parts/batches).

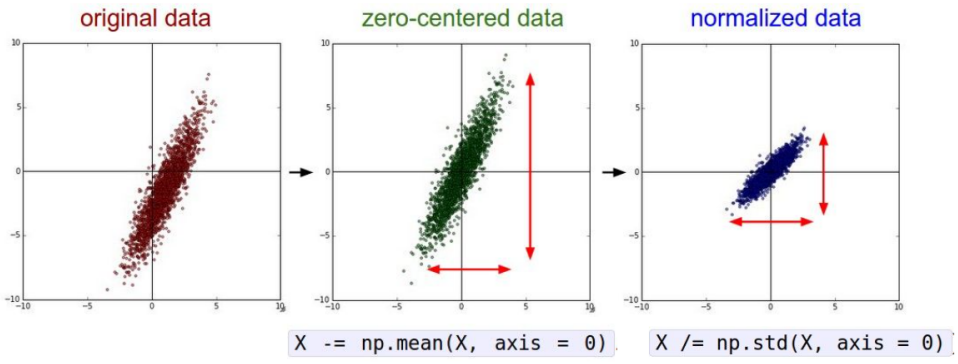


Figure 19: *Pre-processed data is expected to be 'cleaned out' (as explained earlier) and normalized before being fed into the NN model. Be aware that there are different types of normalization tools out there that can be used (for instance, if an operator wants to normalize the whole data set instead of per feature, which is the default setting for most python libraries, then one must hard code the correct*

formulations in order to accomplish that goal).[12]

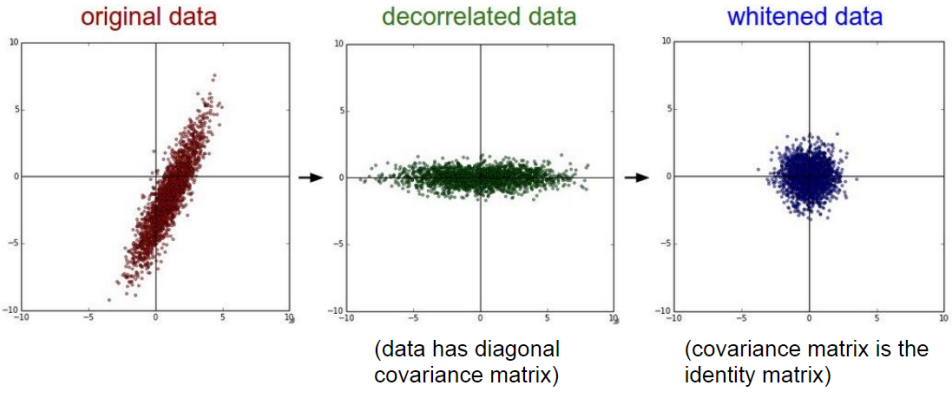


***Figure 20**: Data pre-processing that is commonly used in a reinforcement learning model is Principal Components Analysis (PCA) and data whitening. PCA is dimensional reduction (scaling) with respect to the features in a [MxN] matrix data sample; while whitening is reduction of redundancy within the sampled data.- where features have small correlation to one another and have the same variance and a statistical mean of 0.[12]*

Once data type has been confirmed and pre-processing has been completed (including conducting statistical inferences in the data), then, it is time for data preparation. The questions one needs to ask are:

- How should this data be presented to the NN model? (the expected file format with respect to the open source library, platform selected, and data library based on the language and open sourced platform selected [for this study it was Panda]...etc.)
- What are the expected input sizes (dimensions, such as 3-D or 2-D) and output sizes of the type of the selected NN model? (a 3-D input shape means that it has 3 elements that make up its matrix dimensions - width [row], length [column] and depth; while 2-D means that the input shape is made up of 2 elements/dimensions: row(n), column(m), n x m...etc.)
- How does each layer that is created in, either, the functional or sequential API, processes the input(s) and proposes the output(s)? In other words, what are the calculations done in each cell (based on either activation function or the type of NN model selected)?

Chapter 4 will present the collected clean high dimensional (also defined as multivariate time series) data sample propagated by GMAT.

2.2 Data Setup

2.2.1 Background & Assumptions

Position and velocity vectors, instead of the Keplerian elements, will be the inputs for the NN model to train. This is because using vectors instead of the actual magnitudes will aid in contour mapping of the results. Thus, since a supervised learning model is used for this project the outputs are also the position and velocity vectors instead of thrust factors. Neither thrust nor atmospheric drag will be considered in this project. The only force that will be modeled is due to the n-body effect. Orbital dynamics, such as the Lambert's problem, Hohmann transfer and impulse burns are considered for the simulations and hand calculations.

Due to real time constraints, the spacecraft simulations will begin at a phase point near Venus. The flybys can be modeled about the concept of 'rideshare', where a smaller satellite (such as a microsat) takes off from a larger satellite that is already in a closed orbit about a celestial body; or a spacecraft that has already done its transfer orbit insertion (TOI) - at an earlier phase in time - and is in transit towards Venus. This is done to justify the initial position and velocity vectors to be at different phase points near Venus.

The initial and final state vectors of the planet's ephemerides can be calculated by Jet Propulsion Laboratory's (JPL's) HORIZONS' web interface. The values pertaining to infinite velocity of approach ($V_{\infty a}$) as well as right ascension angle (Ω) was, initially, dependent on the Russian spaceflight Venera-D mission that is to take place, either, in 2016 or 2017. However, it turns out that those values were not intended for an actual flyby to take place. Thus, a random initial ephemerides date was selected, along with an offset to determine the spacecraft's position. The Cassini, as well as the Voyagers 1 & 2, missions will be used as a point of reference to study the alignment of the planets with respect to each other, specifically Earth, Venus, Jupiter and Saturn, at particular times where a burn or a flyby trajectory occurs.

2.3 GMAT & Astrodynamics

2.3.1 GMAT

GMAT is a NASA based orbital propagator and was commercially released in September, 2009. The mathematics and scientific approach behind GMAT are Runge-Kutta89, astrophysics dynamics, and 2-body, as well as, 3-body orbital mechanics. To simplify the scope of the problem, the only force modeling considered when gathering data from GMAT is the n-body effect (how the gravitational acceleration of point masses impact the spacecraft). In GMAT documentation it is called the 'n-Body Point Mass Gravity' [22], and is defined as follows:

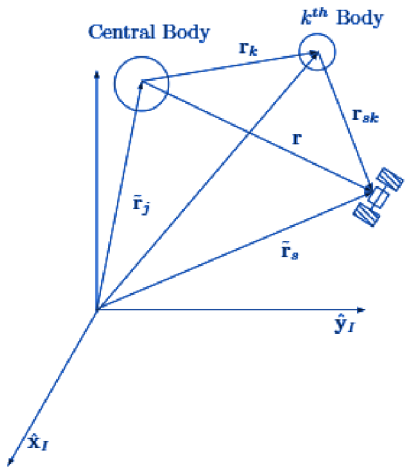


Figure 21*: N-body effect diagram between a central body, spacecraft and kth point mass. [22]*

Equation 1, below, is the initial formulation for the n-body effect modeling in GMAT. Below are the list of terms for Figure 21 :

- r_s : position of spacecraft with respect to a hypothesized inertial frame.
- r_j : position of central body with respect to hypothesized inertial frame.
- r_k : position of the kth body with respect to the hypothesized inertial frame.
- r : position of spacecraft with respect to central body of integration (j^{th} body).
- r_k : position of the k^{th} gravitational body with respect to the central body.
- $r_j + r = r_s$: defining the position of spacecraft with respect to the central body.
    - $r'' = r''_s - r''_j$: second derivative with respect to time.

$$m_s r''_s = \sum_{k=1}^{n} F_k = G \sum_{k=1}^{n} \frac{m_s m_k}{|r_k - r|^3}(r_k - r) \qquad (1)\ [22]$$

Equation 1 is achieved by applying Newton's 2nd Law, where:

- $r_k - r$: is vector from spacecraft to the k^{th} body.
- m_s : mass of the spacecraft
- m_k : mass of the k^{th} body.

r''_s can be re-written as follows:

$$r''_s = G \sum_{k=1}^{n} \frac{m_k}{|r_k - r|^3}(r_k - r) \qquad (2)\ [22]$$

Newton's second law to the j^{th} body to obtain:

$$m_j r_j'' = \frac{Gm_s m_j}{r^3} + G\sum_{k=1, j\neq k}^{n} \frac{m_j m_k}{|r_k|^3} r_k \qquad (3)\ [22]$$

- The first term is the influence of the spacecraft on the central body
- The second term is the influence of the k point mass
- Thus, r_j'' can be simplified as follows:
  - $r_j'' = \frac{Gm_s}{r^3} r + G\sum_{k=1, j\neq k}^{n} \frac{m_k}{|r_k|^3} r_k \qquad (4)\ [22]$

Substituting in equations 2, 3 and 4 into $r'' = r''_s - r''_j$:

- $r'' = G\sum_{k=1}^{n} \frac{m_k}{|r_k - r|^3}(r_k - r) - \frac{Gm_s}{r^3} r - G\sum_{k=1, j\neq k}^{n} \frac{m_k}{|r_k|^3} r_k \qquad (5)\ [22]$
- Collecting terms yields:
  - $r''_{pm} = -\frac{\mu_j}{r^3} + G\sum_{k=1, j\neq k}^{n} \left(\frac{r_k - r}{|r_k - r|^3} - \frac{r_k}{|r_k|^3}\right) \qquad (6)\ [22]$
    - The first term: acceleration on the spacecraft due to central body
    - The second (direct) term: accounts for the force of the k^{th} body on spacecraft.
    - The third (indirect) term: accounts for the force of the k^{th} body on the central body.

Since, force is conservative, velocity and mass partials equate to zero:

$$\frac{\partial r''_{pm}}{\partial v} = 0_{3x3}$$
$$\frac{\partial r''_{pm}}{\partial m} = 0_{3x1}$$

(7) [22]

A vector identity, refer to Equation 8, is then used to determine the partials with respect to position (Equation 9) :

$$\frac{\partial}{\partial x}\frac{a}{a^3} = \frac{1}{a^3}\frac{\partial a}{\partial x} - 3\frac{aa^\intercal}{a^5}\frac{\partial a}{\partial x}$$

(8) [22]

$$\frac{\partial}{\partial \mathbf{r}}(\frac{-\mu_j}{r^3}\mathbf{r}) = \frac{-\mu_j}{r^3}\mathrm{I}_3 + 3\mu_j\frac{\mathrm{r}\mathrm{r}^\intercal}{r^5}$$

(***9***) [22]

Applying the vector identity (Equation 8) to the direct terms of Equation 9, then taking the derivative of the indirect terms, which are zero, and combining like terms, finally get the following result:

$$\frac{\partial r''_{pm}}{\partial r} = -(\frac{\mu_j}{r^3} + \sum_{k=1,k\neq j}^{n}\frac{\mu_k}{||r_k - r||^3})\mathrm{I}_3 + 3(\mu_j\frac{rr^\intercal}{r^5} + \sum_{k=1,k\neq j}^{n}\mu_k(\frac{(r_k - r)(r_k - r)^\intercal}{||r_k - r||^5}))$$

(***10***) [22]

The time Jacobian form of the n-body force modeling is as follows:

$$\frac{\partial r''_{pm}}{\partial t} = \sum_{k=1}^{n}\mu_k(\frac{1}{r^3_{rel}}(\mathrm{I} - 3\hat{r}_{rel}\hat{r}^\intercal_{rel}) - \frac{1}{r^3_k}(\mathrm{I} - 3\hat{r}_k\hat{r}^\intercal_k))\mathrm{v}_k$$

(***11***) [22]

There are several types of numerical integrators, but the Runge-Kutta89 is the integrator of choice for this project. The Runge-Kutta89 is defined via the following equation:

$$\frac{dr^{\,i}}{dt} = f(t, r) \qquad (\textit{12}) [22]$$

As the equation suggests, it calculates the integration at each $i^{\,th}$ of a step and considers the later states as the initial states for the upcoming step.
The series of formulation that the propagator undergoes as it simulates the trajectories are as follows:
Estimates the next stage via time multiplier α_i and time interval $\alpha_i \delta t$:

$$k_i^{(n)} = \delta t\, f(t + a_i \delta t, r^{(n)}(t) + \sum_{j=1}^{i-1} b_{ij} k_j^{(n)}) \qquad (\textit{13}) [22]$$

Total integration step can be calculated by another set of coefficients:

$$r^{(n)}(t + \delta t) = r^{(n)}(t) + \sum_{j=1}^{stages} c_j k_j^{(n)} \qquad (\textit{14}) [22]$$

Estimating the accuracy of the step by comparing the steps at different orders of integration:

$$r^{t(n)}(t+\delta t) = r^{(n)}(t) + \sum_{j=1}^{stages} c_j^* k_j^{*(n)} \qquad (\textbf{15})\ [22]$$

The error estimation:

$$\Delta^{(n)} = \left| \sum_{j=1}^{stages} (c_j - c_j^*) k_j^{(n)} \right| \qquad (\textbf{16})\ [22]$$

The new step size to optimize the accuracy are as follows:

$$\delta t_{new} = \sigma \delta t \left(\frac{\alpha}{\epsilon}\right)^{\frac{1}{m-1}} \qquad (\textbf{17})\ [22]$$

$$\delta t_{new} = \sigma \delta t \left(\frac{\alpha}{\epsilon}\right)^{\frac{1}{m}} \qquad (\textbf{18})\ [22]$$

Where m is the order of truncation being solved and σ is a factor of safety error to prevent unnecessary over iterations [22].

2.3.2 Astrodynamics: Basic Overall Theory and Principles

The Hohmann transfer from Earth to Venus will be modeled with 2-body-problem orbital dynamics. The encounter at Venus will be approximated using patched conics for the 3-body aspects of the flyby transfers so that Kepler laws may be used. Two body orbit problems are often theorized in calculations to simplify the amount and type of forces considered with respect to the spacecraft and its surrounding celestial environment. Thus, for a two body problem the only force modeled is the gravitational force of the celestial body, during the time when the spacecraft is in its sphere of influence (SOI).

Gravitational force along the 'r' component:

$$F^{Q} = -\frac{GMm}{r^2} q_r \qquad (\textbf{19})\ [17]$$

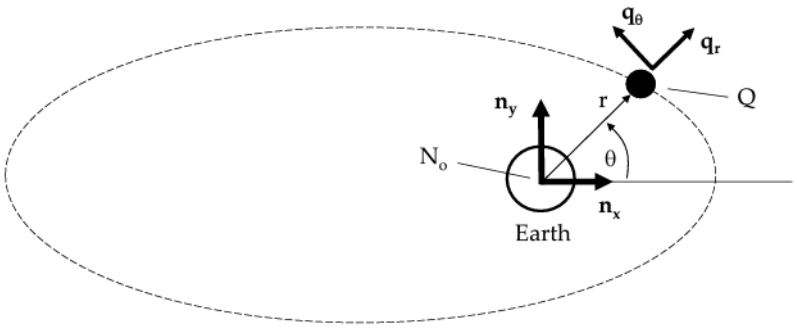
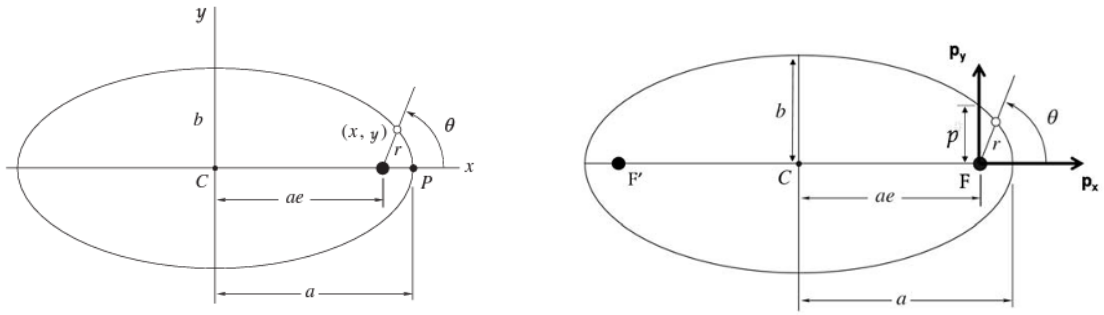


Figure 22*: Basic orbital diagram. The spacecraft's position is defined with respect to the 'q' components, as well as the polar coordinates, 'r' and θ [17].*

.



F, F′ are foci
a = semi-major axis
b = semi-minor axis
e = eccentricity
p = semi-latus rectum
C = center

Figure 23*: Basic characteristics of a two-body problem; where a closed elliptical 2-D orbit is depicted. The angle (θ) and the distance (r) indicate the location, in polar coordinates, of the spacecraft with respect to the foci (emboldened period). The foci is where the planet, or celestial body of interest is located. The point of perigee (P) is the closest point on the closed orbit to the foci or the planet of interest, and is made up of vector components P_x and P_y. The semi- latus rectum is the shortest perpendicular distance between the foci and the closed orbit at which the spacecraft is on. 'C' is the center of the closed elliptical orbit and 'a' is the semi-major axis of the closed elliptical orbit. [17].*

The orbit equation is used to define the polar position of the spacecraft, as well as the orbit's eccentricity and angular momentum in any restricted 2-body problem. It is the following:

$$r = \frac{h^2}{GM}\left(\frac{1}{1+e\cos\theta}\right) \quad \text{where e > 1 for a hyperbola} \qquad (\textbf{20}) \text{ [17]}$$

$$r = \frac{a(1-e^2)}{1+e\cos\vartheta} \qquad (21)\ [17]$$

where a = semi major axis, e = eccentricity of the orbit [e = 0 is a circle, while 0<e<1 is an ellipse and e=1 is a parabola], θ = angle between line of apsides and the spacecraft's location depicted by 'r'. Note that since both, Equation 20 and Equation 21, are the orbit equations, $\frac{h^2}{GM} = a(1 - e^2)$. Where 'h' is angular momentum per mass, 'G' is the gravitational constant, and 'M' is the mass of the celestial body of interest located at the foci.

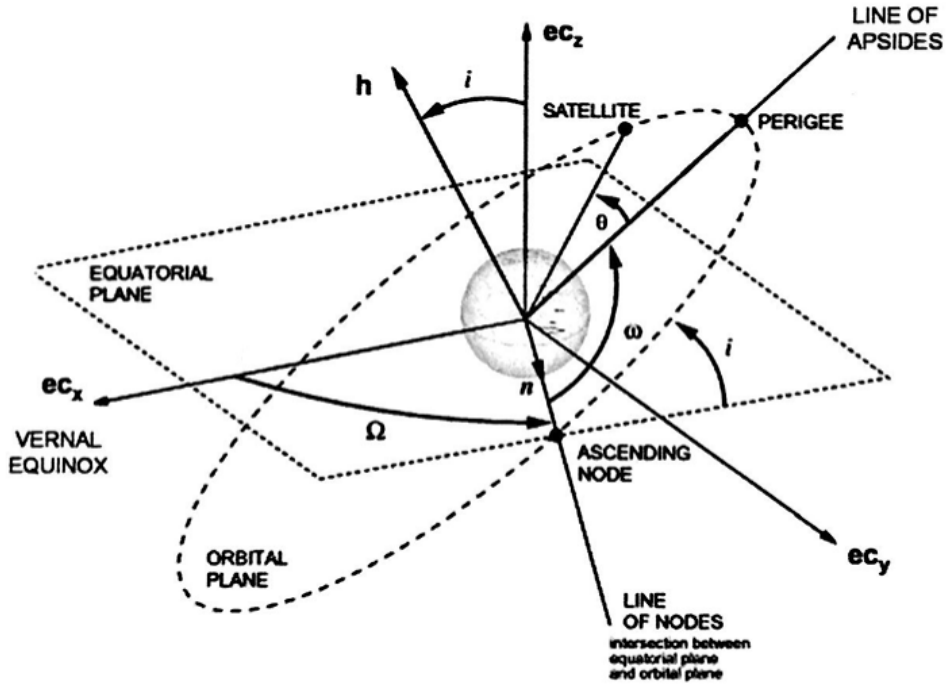


Figure 24*: The Keplerian elements, where the Newtonian reference frame (ecx, ecy, ecz) is at a fixed inertial point at the center of the celestial body. Note that the orbital plane is not always at the equatorial plane. [17]*

As depicted in Figure 24, the Keplerian elements that are used to define any orbit (closed or open) are as follows:

Ω = Right ascension angle; measured between ec_x and the line of the nodes. The line of nodes is where the orbital plane intersects with the equatorial plane. Note that often times there are two types of line of nodes: ascending (spacecraft is traveling up along the elliptical perimeter of the orbital plane) and descending (spacecraft is traveling downward, sometime after the point of perigee, along the elliptical perimeter of the orbital plane).

i = Inclination angle of the orbit.

h = Specific angular momentum. Always normal to the orbital plane.
ω = Argument of perigee; measured between the line of nodes and the line of apsides.
e = Eccentricity of the orbit.
r = Spacecraft location (magnitude of the spacecraft's position vector).
θ = Spacecraft's location in polar coordinates, measured between the line of apsides and the vector form of 'r'.

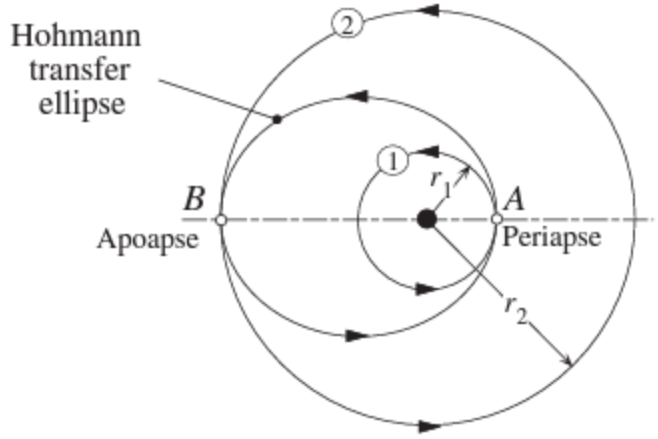


Figure 25*: The Hohmann transfer is an elliptical transfer that is considered to be the shortest and most efficient form of interplanetary travel. The transfer is made between two circular closed orbits either about one particular celestial body of interest or two. The state vector change with respect to position and velocity, as well as the flight path angles, are done through pre-calculated impulsive burns.* r_1 *and* r_2 *are radii for circular orbit 1 and orbit 2, respectively. Since the orbits are circular their calculations with respect to velocity are simplified, as well. The velocity equations are also considered the energy equations for the difference,* Δv*, is what constitutes as the amount of 'burn' or fuel a spacecraft must use to get from point A to point B. [17]*

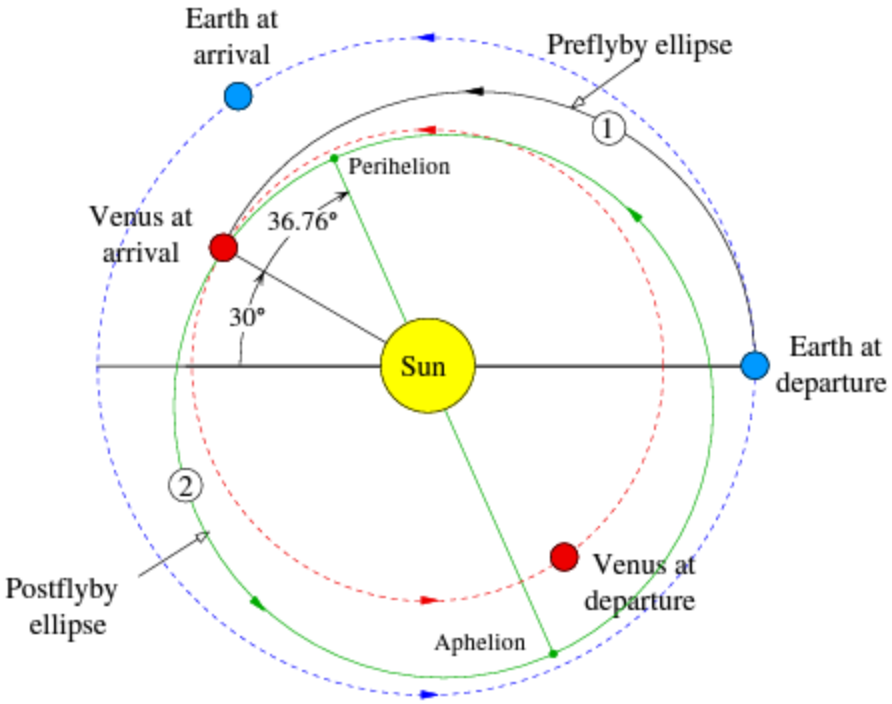


Figure 26: *Sun centered, or heliocentric, flyby trajectories between Earth and Venus [23]. A flyby can be viewed as a power push for the spacecraft, thus making it an ideal choice to reduce the amount of fuel usually used for impulsive burns when changing the flight path angle. This in turn prolongs the life cycle of the spacecraft as it conducts its interplanetary travels. The flyby is also considered as an open orbit, for it is either parabolic or hyperbolic in nature.*

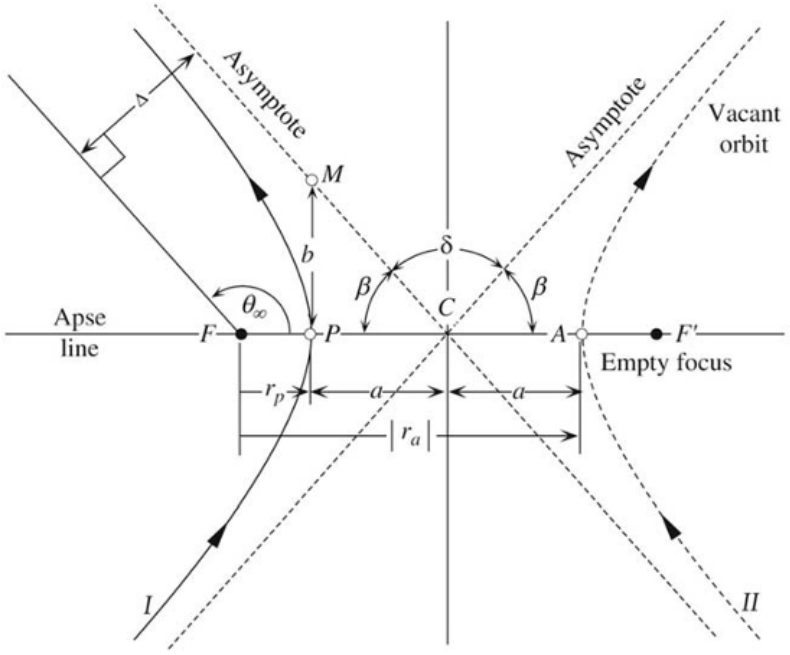


Figure 27*: on the left hand side is where the full focus is (a.k.a planet of interest). The approaching arrow from the bottom (below the Apse line) is where the spacecraft is approaching at a velocity defined as* $V_{\infty a}$. *The arrow above the Apse line is the departing velocity of the spacecraft defined as* $V_{\infty d}$. *[17]*

The hyperbolic trajectory's elements are the following:

a = semi-major axis length, which is negative for a hyperbola since it is outside of the conic section

b = semi-minor axis length

β = asymptote angle

θ_∞ = true anomaly as r → ∞, the angle of asymptote w.r.t. apse line direction

δ = turn angle

Δ = aiming radius

r_p = radius of periapsis, consistent with the definition of r_p for an ellipse: the distance from the occupied focus to the point of closest approach (periapsis)

r_a = radius of apoapsis, also negative for a hyperbola

[17]

The arrows in Figure 27 are traveling in a counterclockwise motion, and depending on the

direction of the planet and where the spacecraft is approaching, the flyby is determined to be either a leading or trailing side flyby. A leading side flyby is where the spacecraft happens to come in front of the planet while the trailing edge flyby is where the spacecraft travels behind the planet. The trailing side flyby is most desirably sought in interplanetary travel, due to the spacecraft having more push from the planet's retrograde motion as it (spacecraft) travels behind it.

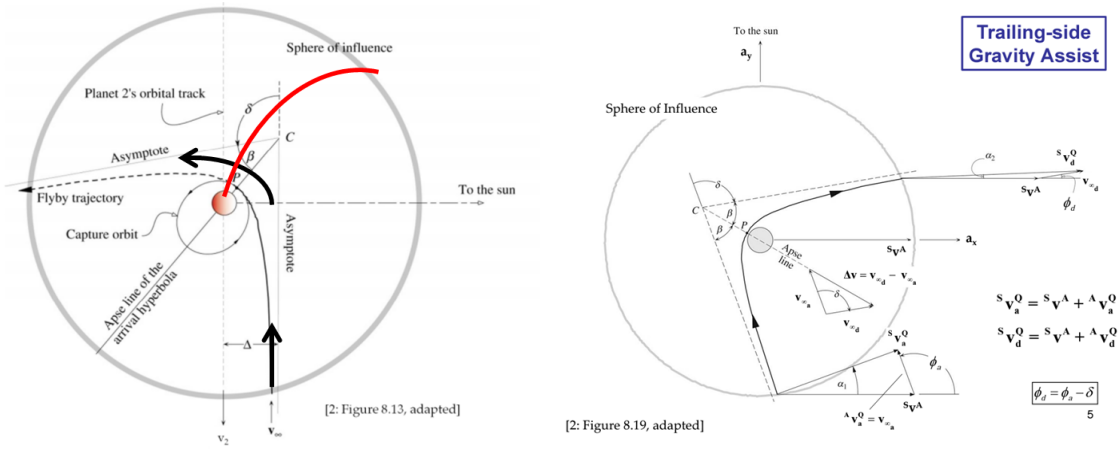


Figure 28*: This is an example of a trailing side flyby. Note that the red arrows and lines indicate the planet's pathway while the black arrows indicate the fly by direction of the spacecraft. The spacecraft's velocity of approach is opposite to the direction of the planet's orbital pathway and the spacecraft conducts the flyby behind the planet's pathway, as well. [17][23]*

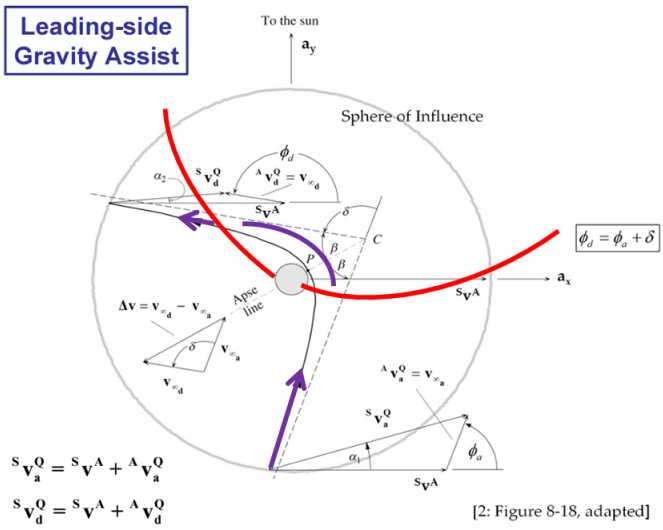


***Figure 29**: Leading side flyby. Note that the pathline that runs parabolically across the diagram is the planet's orbit while the spacecraft is marked by the purple pathway and the blue arrows. The spacecraft is indeed flying in front of the planet's pathway as it conducts its flyby trajectory. [17] [23]*

The energy equations mentioned previously are as follows:

Velocity required to maintain within a closed circular orbit:

$$v_{circular} = \sqrt{\frac{GM}{r}} \qquad (\textbf{22})\ [17]$$

Velocity required to escape the SOI of a celestial body, like in a flyby trajectory. The orbit is usually open and can either be parabolic or hyperbolic in geometry.

$$v_{escape} = \sqrt{\frac{2GM}{r}} \qquad (\textbf{23})\ [17]$$

Velocity required to maintain an elliptical closed orbit or the energy required to complete a Hohmann transfer:

$$v = \sqrt{\frac{2GM}{r} - \frac{GM}{a}} \qquad (\textbf{24})\ [17]$$

2.3.3 Astrodynamics Methodology

The overall process of calculating the required input for GMAT is as follows:

- Since, there are no hyperbolic elements ($V_{\infty a}$, Ω_a , $V_{\infty d}$) from the 2016 Venera-D mission that defines a flyby - will instead use JPL's Horizons Ephemeris data and a Matlab code of Lambert's problem.
- All data collected via JPL's Horizons and GMAT are heliocentric.
- Lambert's problem (will be discussed more in depth in Chapter 4) requires an initial and final position state vector, as well as the duration of flight. Since the Cassini mission had a flyby from Venus to Saturn, the value for the time of flight (of five years) was used.
- Input necessary parameters (start date and position and velocity vectors of spacecraft) into GMAT to retrieve data samples for one flyby (particularly between Venus and Saturn), and create different sets of data at different radii of approach (for that particular flyby). These different sets of data will be listed as perturbed or unperturbed based, and will be divided into training, validation and test data sets for the NN.
- If time permits, then conduct multiple flybys towards different planets, starting from Venus (this is to create a more diversified contour mapping of the solar perturbations and this will allow the NN to create better predictions).
- The only force modeling to be considered is gravitational amongst bodies (n-body effect). All others (atmospheric drag, non-spherical, spacecraft thrust, relativistic corrections, solar radiation pressure, and percent shadow partial derivatives) will not be considered due to simplification and real time constraints.

2.4. GMAT SETUP

2.4.1 Few GMAT Initial Pointers

The data that is to be sampled will have a six features for the unperturbed states: X_U, Y_U, Z_U , V_{xU}, V_{yU}, V_{zU} and six perturbed states: X_P, Y_P, Z_P , V_{xP}, V_{yP}, V_{zP} at every simulated time step. The important step that must be done before running the simulator is making sure for which six features are the desired output (is it for the perturbed set? Or the unperturbed set?), and to do that, one must navigate to the 'Propagator' window and select or deselect (depending which set, respectively), as shown below in Figure 30, inputting the sun as a considered point mass (please, take note that solar radiation pressure (SRP) is in reference to the solar influence on the solar panels of the spacecraft. Hence is not the focus of this study.) :

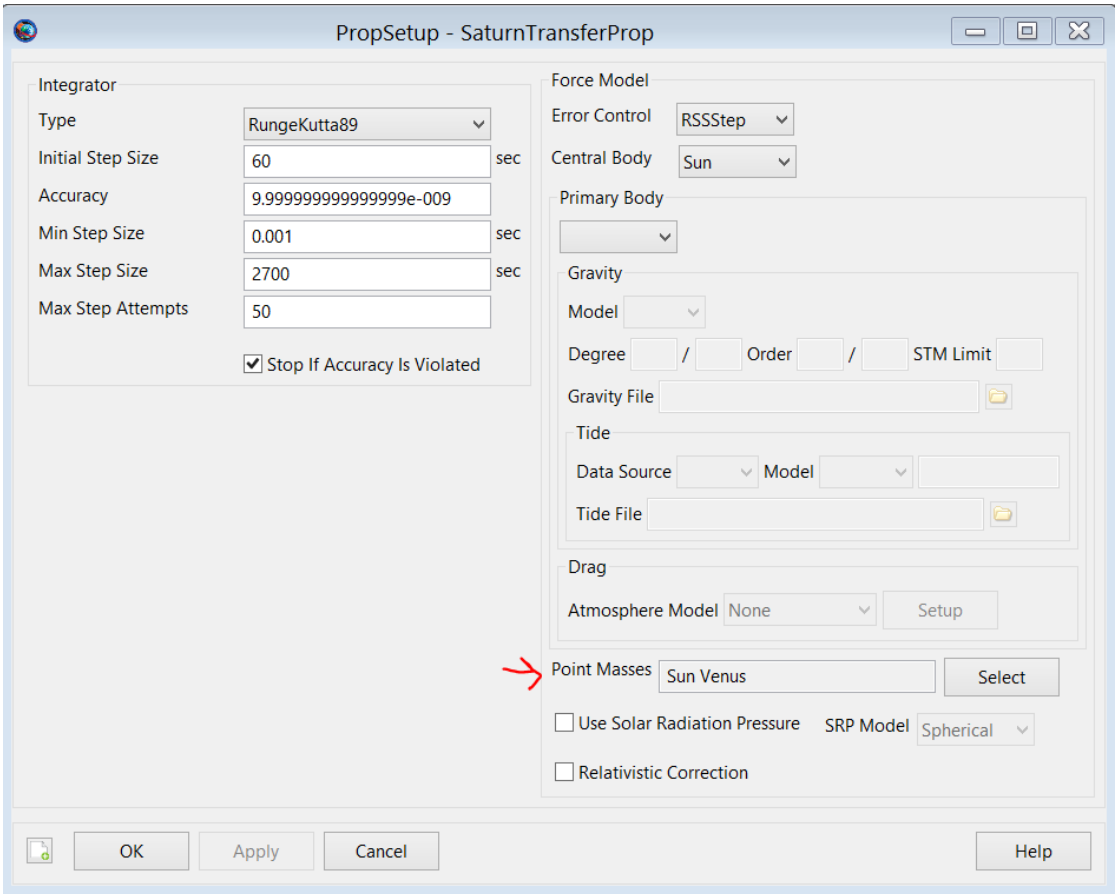


***Figure 30**: Propagator window at GMAT. The point masses are adjustable. Since a transfer to Saturn is to be done, Sun and Venus are selected as the critical bodies.*

Must also toggle on the ReportFile before the start of your 'Mission' sequences, as shown below in Figure 31 , this way all the necessary data is captured for all iterations:

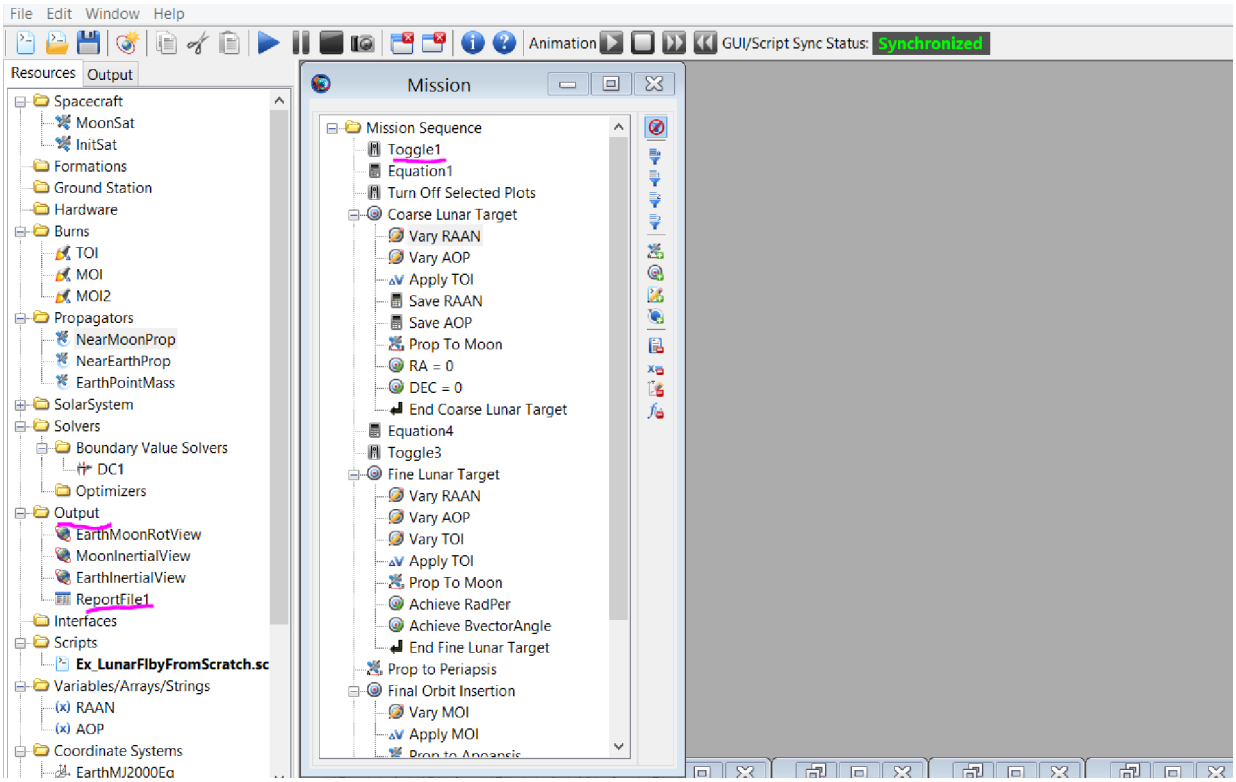


***Figure 31**: GMAT's mission and Resources tab. Notice that the 'ReportFile1' is an output option under the Resources menu and is defined to turn on/off, 'Toggle1', under the Mission tab. Note that the name 'ReportFile1' is modifiable, as well as, the selection of what type of data outputs the simulator is expected to give.*

The launch dates and the overall orbital phasing is strategized after the Cassini, the Venera-D missions, as well as, Voyagers 1 & 2. More of the overall GMAT setup and final astrodynamic calculations, as well as, data collection will be covered in Chapter 4.

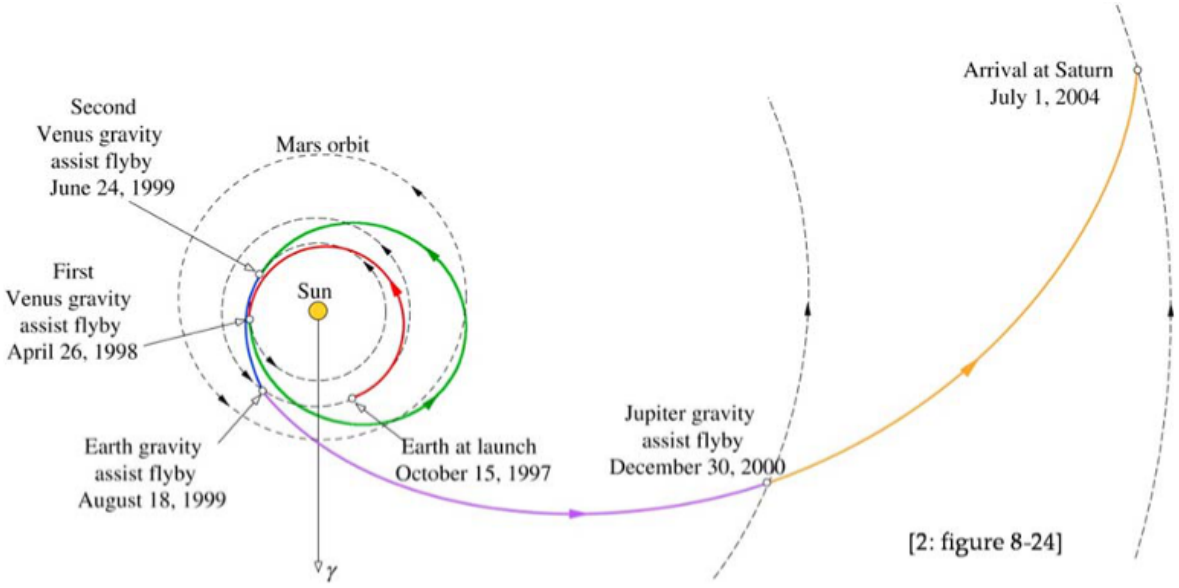


***Figure 32**: Cassini mission timelines and trajectories [17].*

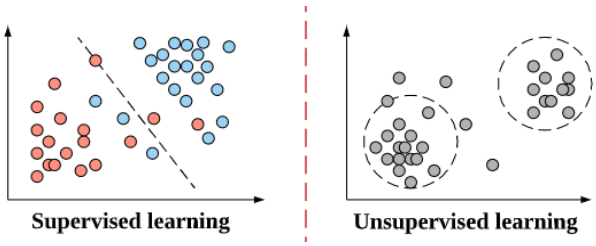
Chapter 3 - Back Propagation Method & RNNs

3.1 NNs & Back Propagation

3.1.1 Background

Chapter 1 discussed briefly the different categories of neural networks, and this chapter will delve a bit deeper and beyond that scope. Chapter 3 is all about Artificial Neural Networks (ANNs): its building blocks, its main categories, the different types that exist, as well as how to train one. This section will focus on the main categories of all NN models, as well as the classical ML algorithms, connected in the overall paradigm that is AI.

The diagram in Figure 33 presents the overall categorization between supervised and unsupervised learning, where the former requires input and output data to create a predicted model and the latter requires only input data for grouping and interpretation.



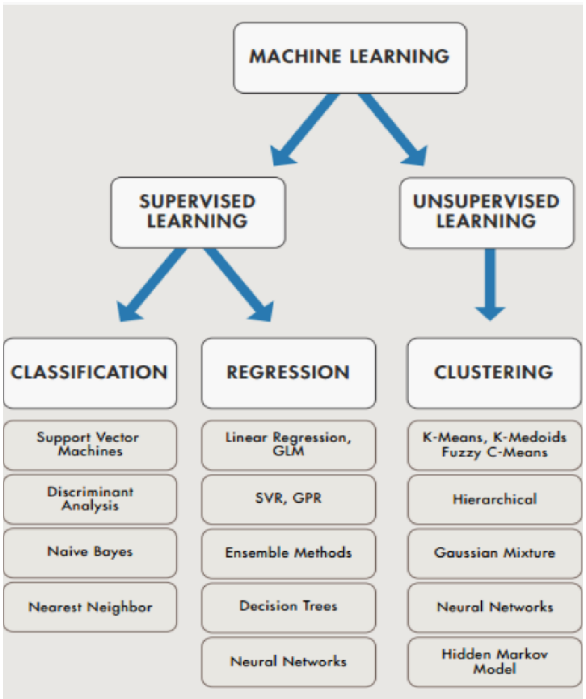


***Figure 33**: Models used for supervised vs unsupervised learning types. This study falls under supervised learning and is a regression type of problem due to the fact that the data sampled is numerical. Thus, the NN model is a viable option. [24]*

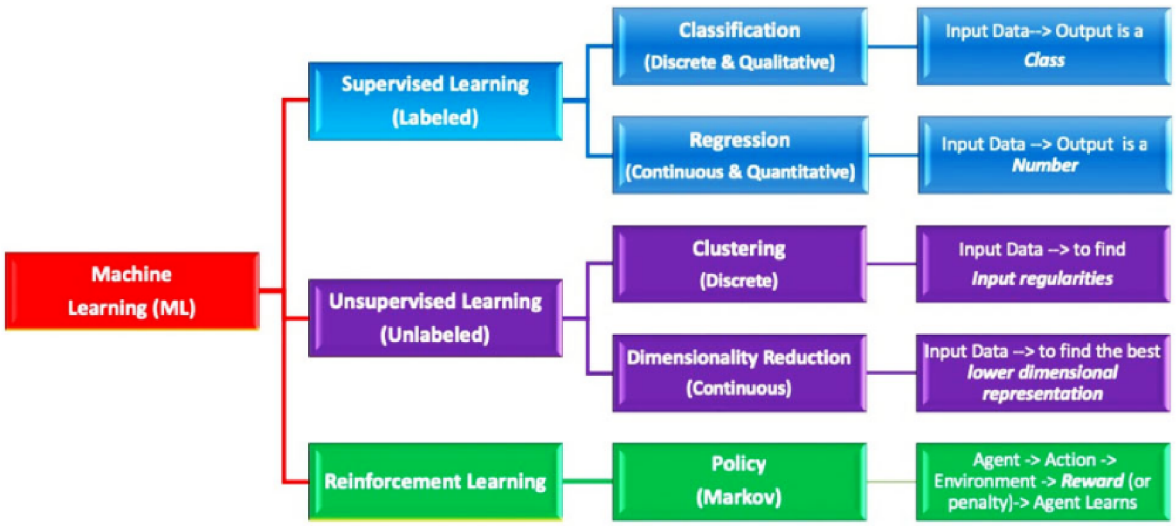


***Figure 34**: General ML diagram. This diagram shows all three general categories (2nd column) pertaining to ML and the type of models (last column) that can be used based on the data type (3rd column) that an operator is dealing with. Note that reinforcement learning can also be defined as semi-supervised. [25]*

The thinking process to decide between whether or not to select the three categories can best be described as the following diagram. Consideration of the data type will determine the best model to use. The diagram, shown in Figure 35, contains a general question and answer pathway to figuring out what model to eventually use.

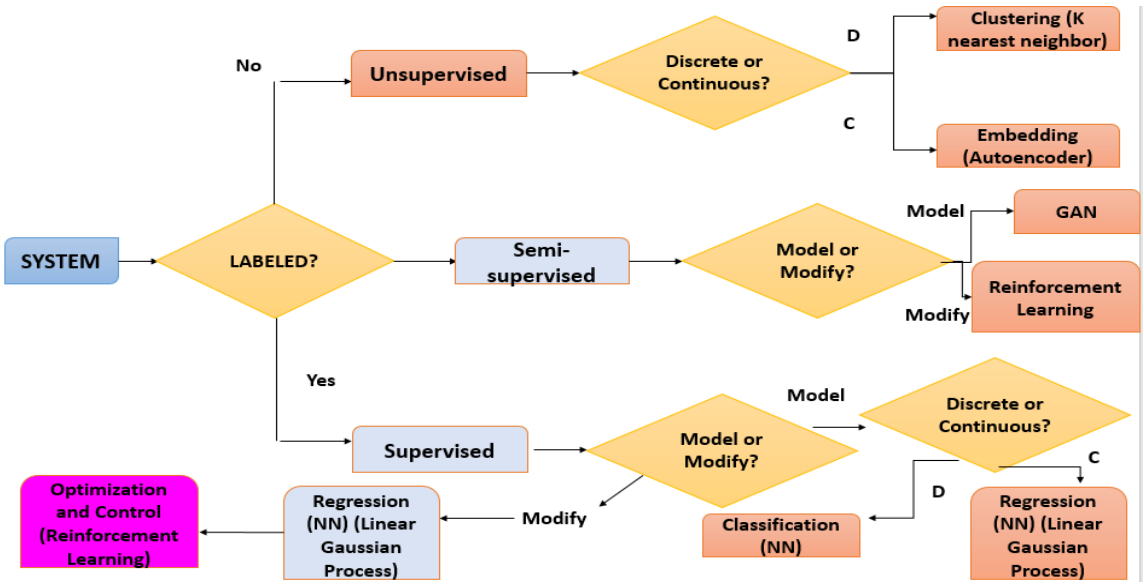


***Figure 35**: Dr. Brunton's ML diagram. This diagram allows one to ask the right questions in figuring out what model to use for their data type. [21]*

Once the type of data has been defined and potential models have been selected, then one must decide whether the purpose of the data is to model or modify as the diagram in Figure 35 suggests). For example, the purpose of the data collected in this study is to allow a machine to predict where the solar perturbations are and which can be used for fuel effective maneuvers. Thus, not only is the numerically labeled data (supervised regression) used for prediction, but is also expected to allow a machine to 'learn' (model) from it, thus, the ideal type of model (or tool) to use is a NN. Section 3.2 discusses different types of NN models an operator can choose from based on the application and data type at hand.

Other fields at which NN was used was in predicting the stop times for maritime transit at different sea ports, as depicted in Figure 36. This is a regression high-dimensionality type of data set, since it is both numerical and contains multiple features, respectively. Another research topic that adopts a similar platform to this thesis, as well as that shown in Figure 36, was in anomaly detection for temporal data derived from machines, such as electrocardiograms (ECGs) (Figure 37).

Distance to port	Distance covered since last signal	Minutes since last signal	Distance to next signal	Minutes to next signal	ATT to port
203	4	15	23	76	631
180	23	76	3	8	555
177	3	8	177	547	547

Input variables

Target

***Figure 36**: Data sample used for RNN. The data was shaped in such a way where the NN was expected to predict the amount of time it takes to travel to the next signal, and it achieves that by being fed the input variables from the previous times and travel distances the ship has traveled. [26]*

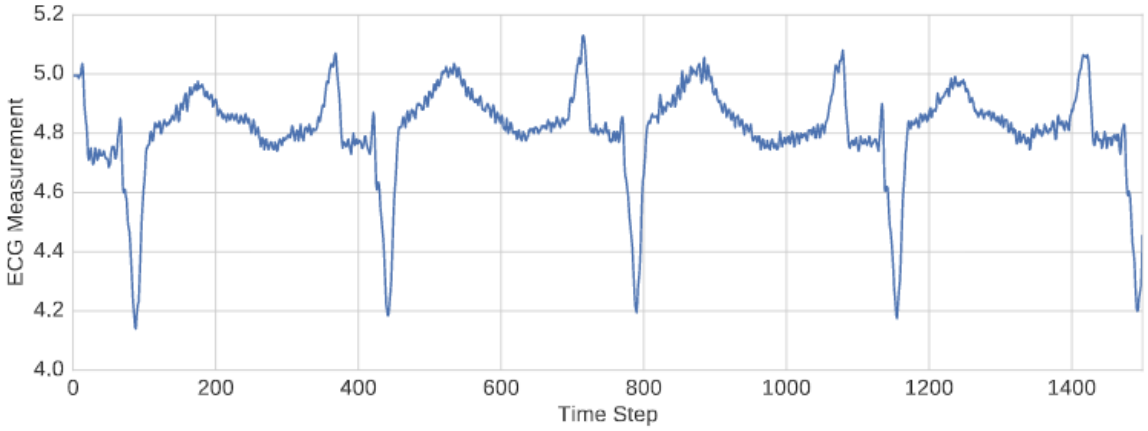


***Figure 37**: Time Series ECG dataset. Note that this is a regression data type and is time dependent, as well. Yet, it is not high dimensionality because it only focuses on one feature which is just the heartbeat or pulse/ms readings at every time step. [27]*

Data preparation usually includes plotting and statistical analysis to observe the behavior of the features of interest. This tactic aids the operator in determining what features to use as inputs and outputs and what the machine is expected to predict.

In short, if the data are images, this is treated as a categorical or classification or a discretized type; while sentences and numbers that have a continuous form are treated as a regression type of data.

3.1.2 Backprop & How Does it Work?

In general, once the data have been collected and prepared the operator must select the type of model to use. If in this case, a NN was selected due to the context of the application, then the following explains how NNs function. In Chapter 1 the structure of the NN was briefly explored, but this section will get more involved in the basic architecture or building blocks that make up the NN. It is important to also note that NN was initially designed (and commonly used) for language processing, image classification, text or audio or temporal sequential predictions, as well as object detection.

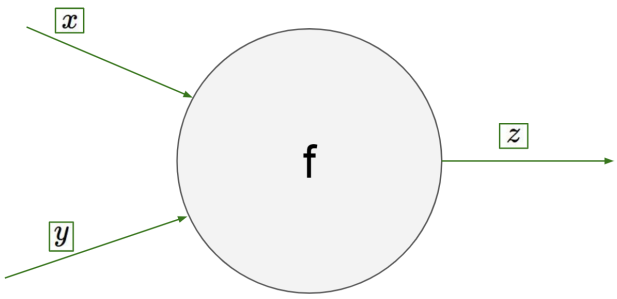


***Figure 38**: A basic model of a node (also interchangeably called 'cell' or 'neuron'). The NN is made up of these little balls called nodes, just like the brain is made up of neurons (as depicted in Figure 35). Note that the node has two inputs (x and y) and one output (z). The arrows are called branches and are designating the forward propagation, since it's moving in one direction, from input to output. The 'f' designated at the center of the node is the activation function that each node has (similar to the way a brain's neuron is 'activated' once a signal is introduced). The different types of activation functions are shown later in this section. Note that for a perceptron, where there is only one node used, the activation function is linear, where f = W*x + b. Where 'W' is the weight(s), 'x' is the input(s), and 'b' is the bias; each input has a weight and a bias associated with it. [12]*

There are several different types of activation functions programmed for each layer of nodes. Typically the hidden layers contain non linear activation functions while the output layer's cells do (cannot have different designated activation functions within a single layer). Figure 39 presents the commonly used types.

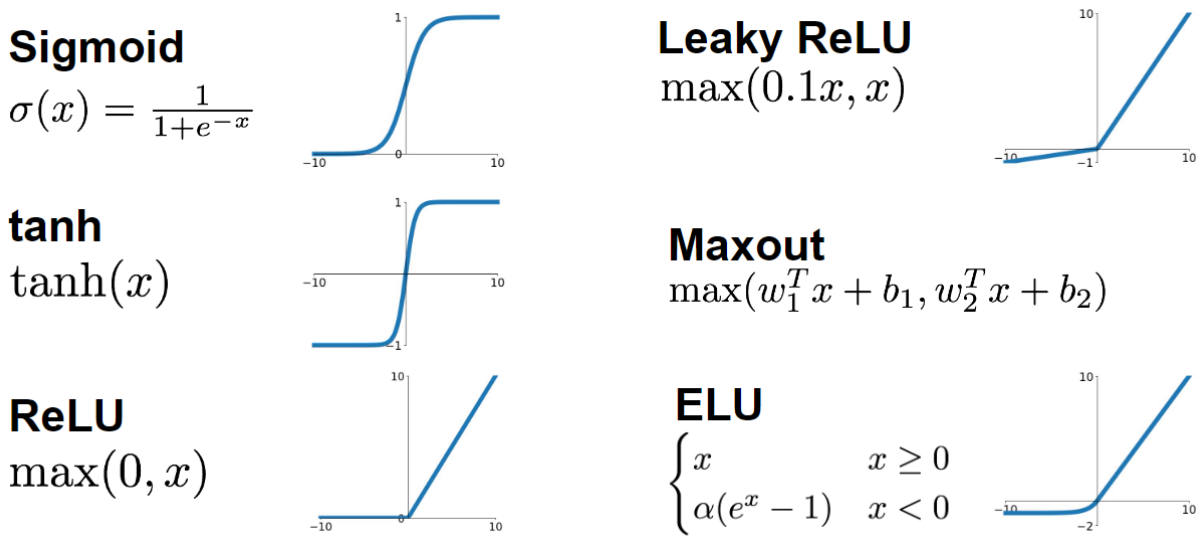


***Figure 39**: Activation functions [12]. Sigmoid, tanh, ReLU and Leaky ReLU are popularly used activation functions in current NN's hidden/output layer's cells'. Keep in mind that the activation functions only work with normalized data where the input values typically range from 0 to 1 (but there are cases where it's between -1 to 1, as the Leaky ReLU, tanh, and ELU functions point out). Keep in mind that due to the nature of the function, sigmoid and tanh functions have issues with saturation (values dying out).[12]*

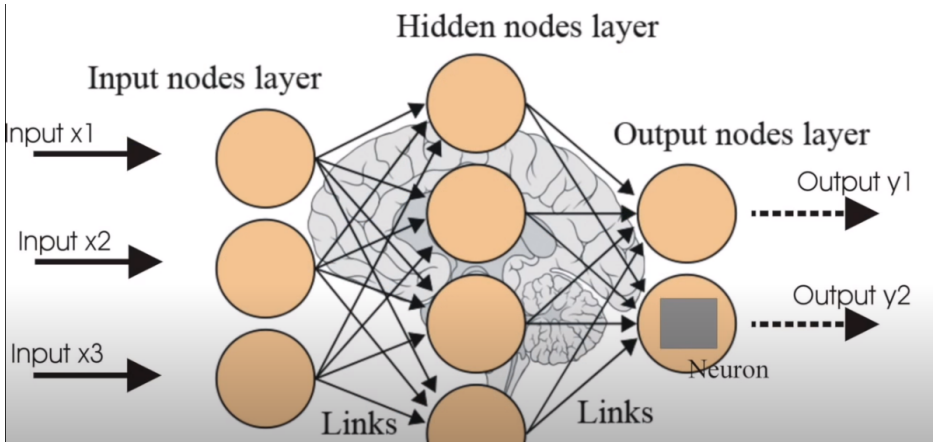


***Figure 40**: General overall architecture of a NN. In contrast to what was shown in Figure 34, multiple nodes are depicted and are connected to each other. Any set of nodes (or units) that exists between the input column of nodes and the output column of nodes are called the hidden nodes. Each node, however, has an internal activation function that is expected to be specified when creating the model (regardless if either the sequential or functional API is selected) in either PyTorch, Keras/Tensorflow and other machine/deep learning open source platforms. Furthermore, depending on the type of NN selected, the overall number of parameters (total connections/branches) will be calculated differently. [28]*

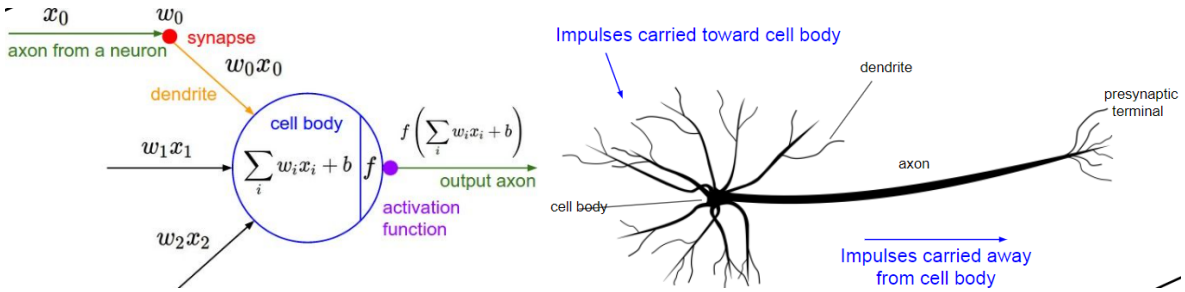


***Figure 41**: Diagram of a perceptron initiated by Frank Rosenblatt around 1957. Similarities between a cell body in a NN vs that of a brain. The parameters depicted in the left image are fully explained in the next figures. [12]*

In a NN, every node that is connected to another node has an associated weight (that's multiplied) and a bias that's added, as depicted below in Figure 38. The weight is also defined as the 'strength'[29] an input has on the output. Weights are randomly initialized (automatically by the software or open sourced platform) and are updated based on the error between the estimated output of the NN and the actual target value. The update is done via gradient descent (a series of partial derivations between the nodal connections) optimization tactics such as stochastic gradient descent (SGD) or the Adam optimizer, as will be shown later in this section.

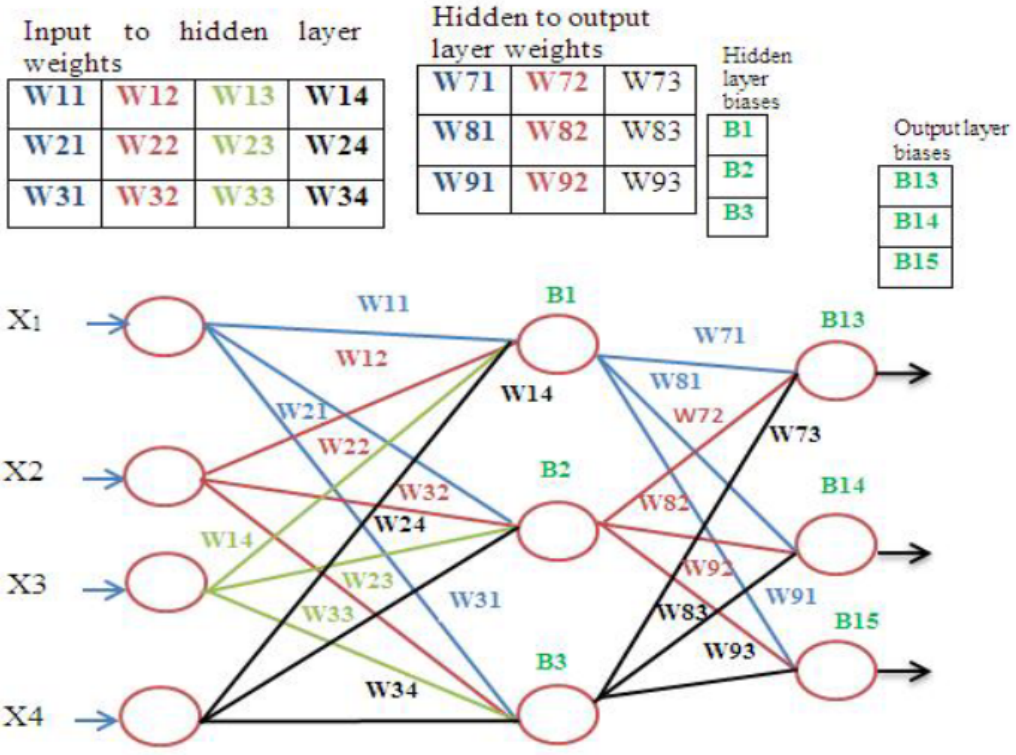


***Figure 42**: Nodal representation of weights and biases. All the 'W's' correspond to the weights and the subscripts correspond to which nodes it's in between. Note that the weights are labeled along the*

branches and the biases (denoted by 'B' and associated subscript) are at the nodes. The middle column or 'layer' is called the 'hidden', while the first layer is the input and the last is the output layer. Since data is represented in vectorized format (or matrices) so are weights, biases and the outputs of the NN. [30]

Understanding back propagation requires the operator to understand how forward propagation works. In order to simplify the initial understanding of the process, this will be done in a general sense where the type of activation function that exists within the node will be specified later, and the type of NN model selected isn't specified and/or considered. Furthermore, the diagrams presented as examples will be using actual numbers before presenting the general mathematical diagram in order to facilitate a better understanding of the process.

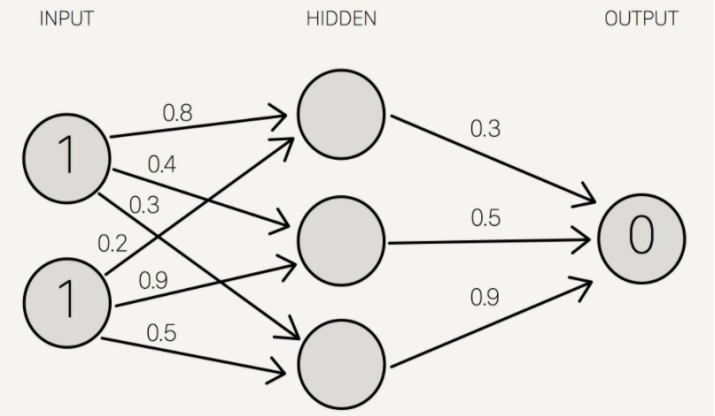


***Figure 43**: Weights are initialized within this three layered NN. Note that the two inputs are just a numerical integer of '1' and the estimated output is just a '0'. Note that this is just the initial stage where the operator has only just declared his/her model with respect to the amount of input features that the NN model should dissect, as well as, the amount of cells that the hidden and output layers should have. It is a rule of thumb to have more hidden layer cells than that of either the input or output layers' (usually by a factor of 2, with respect to the number of input cells and the hidden layer after a prior hidden layer). [29]*

Weight initialization is automated by the open source platform (Tensorflow/Keras, Pytorch...etc.) and the values are usually between 0 and 1; because it mimics a Gaussian distribution. In fact, a lot of the data analysis and tools used in the NN or deep learning online platforms tend to follow such a statistical distribution. This is why it is important to consider whether or not the online tools (usually in the form of extended Python libraries, such as scikit-learn) are in alignment with the type of data and prediction model the operator wants to generate.

Next, once the model has completed the first forward propagation pass, Figs. 41 - 43 presents how the resulting values would look, as well as the mathematics behind it:

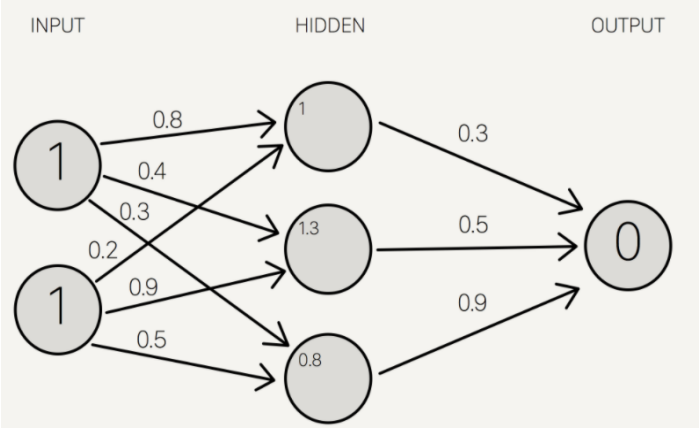
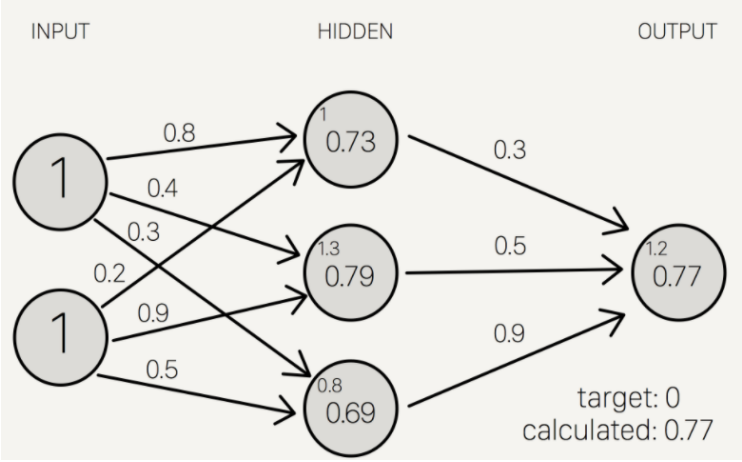


Figure 44: $f = \Sigma(W * x)$, *where 'x' is the perceived input with respect to the node of interest; but with regards to this single hidden layer model, the input is from the actual input layer. Thus, the mathematical model will be input layer's value * initialized weight + the 2nd input layer's value* it's initialized weight = value for the linearly activated node in the first hidden layer. [29]*

Once the values for the first hidden layer are complete, including the calculation done for the sigmoid activation function programmed within the hidden layer's cells (the values that are in larger font size in the hidden/middle layer), the next set of calculations will be for the estimated output. S() is the sigmoid function and its inputs, 'x', are the linear values calculated in Figure 40.

```
S(1.0) = 0.73105857863
S(1.3) = 0.78583498304
S(0.8) = 0.68997448112
```

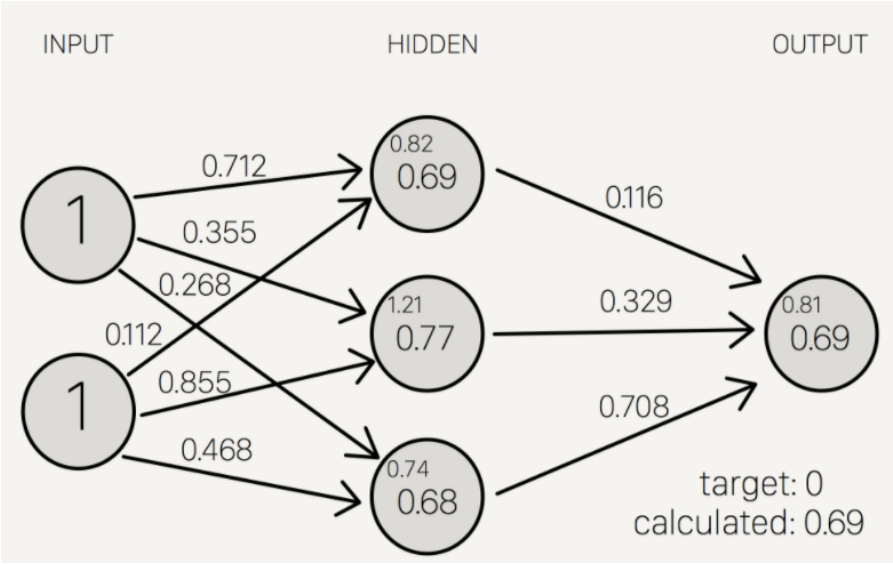


```
0.73 * 0.3 + 0.79 * 0.5 + 0.69 * 0.9 = 1.235
```

```
S(1.235) = 0.7746924929149283
```

***Figure 45**: Calculating the sigmoid function's values and the estimated output. Note that in the context of this particular model, the estimated value turned out to be '0'. The calculation goes from the input value of the hidden cell * it's initialized weight + the 2nd hidden cell's value * it's initialized weight + the 3rd hidden cell's value * it's initialized weight = estimated value (at the single cell output, note that the output layer can have multiple cells, depending on how many features the operator wants to be considered as a targeted label). The calculations shown above the diagram is for the sigmoid activation function, where the input 'x', in the sigmoid equation, are the linear values calculated via the weights and inputs as described earlier. [29]*

Once the forward propagation pass is complete note that the calculated or estimated value is off from the expected targeted value. Thus, this is where backpropagation comes to play, it basically adjusts the weights based on the error between the estimated/calculated output (0.77)value(s) to that of the referenced/targeted value(s) (0). In fact, based on how many epochs (1 epoch = the full forward prop + full backprop pass through all sampled data) an operator has designated the model to run for, the back propagation passes will continue to iterate (1 iteration = 1 forward pass + 1 backward pass over a batched sample, thus 1 epoch can have several iterations depending on how many batches - or batch_size), and the accuracy should increase if the model is learning, thus decreasing the error.




```
Target = 0
Calculated = 0.77
Target - calculated = -0.77
```

, $S'(sum) = \frac{\mathrm{d}sum}{\mathrm{d}result}$, where $sum = \Sigma(W * x)$,

$$\frac{\mathrm{d}sum}{\mathrm{d}result} \times (\text{target result} - \text{calculated result}) = \triangle \text{sum}$$

, thus, (S(1.23) = 0.77).

```
Delta output sum = S'(sum) * (output sum margin of error)
Delta output sum = S'(1.235) * (-0.77)
Delta output sum = -0.13439890643886018
```

, next,

$$H_{result} \times w_{h \to o} = O_{sum}$$

, now we know that the hidden layer result, H_{result}, is the sigmoid function value at a particular hidden node, is *it's forward connecting weights = output sum for each hidden cell node, so can transform it to the following:

$$\mathrm{d}w_{h \to o} = \frac{\mathrm{d}O_{sum}}{H_{results}}$$

, where it states that the greater change, or difference, between output sum (O_{sum}) and input hidden sigmoid result (H_{result}) will cause a greater shift/change in the new weight values, dW, as shown :

```
hidden result 1 = 0.73105857863
hidden result 2 = 0.78583498304
hidden result 3 = 0.68997448112

Delta weights = delta output sum / hidden layer results
Delta weights = -0.1344 / [0.73105, 0.78583, 0.69997]
Delta weights = [-0.1838, -0.1710, -0.1920]

old w7 = 0.3
old w8 = 0.5
old w9 = 0.9

new w7 = 0.1162
new w8 = 0.329
new w9 = 0.708
```

And this calculation continues on to the next layer of nodes until all the weights are upgraded and a new forward propagation pass occurs to calculate a new estimated output value (which shown earlier was 0.69). And another margin of error is calculated, or loss, and then another back prop pass occurs until the margin of error, or loss, is acceptable to the operator's application.

$$\mathrm{d}H_{result} = \frac{\mathrm{d}O_{sum}}{w_{h \to o}}$$

So, since we know that: , then change in the hidden result is also

$$S'(H_{sum}) = \frac{\mathrm{d}H_{sum}}{\mathrm{d}H_{result}}$$

defined as: , thus can do the following:

$$dH_{result} \times \frac{dH_{sum}}{dH_{result}} = \frac{dO_{sum}}{w_{h \to o}} \times \frac{dH_{sum}}{dH_{result}}$$

$$dH_{sum} = \frac{dO_{sum}}{w_{h \to o}} \times S'(H_{sum})$$

, thus numerically finding delta hidden sum:

```
Delta hidden sum = delta output sum / hidden-to-outer weights * S'(hidden sum)
Delta hidden sum = -0.1344 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])
Delta hidden sum = [-0.448, -0.2688, -0.1493] * [0.1966, 0.1683, 0.2139]
Delta hidden sum = [-0.088, -0.0452, -0.0319]
```

Note that the original weights that are situated between the hidden and output layers, as well as the original hidden sums, are used in calculating the change in the hidden sums. This delta hidden sum is then used to calculate the new weights between the input and hidden layers. The following set of equations derives the relationship between I = *input*, ΔH_{sum} = *change or delta in hidden sum*, and dw = *delta or change in weights*:

$$I \times w_{i \to h} = H_{sum}$$

$$\frac{dH_{sum}}{dw_{i \to h}} = I$$

$$dw_{i \to h} = \frac{dH_{sum}}{I}$$

thus,

```
input 1 = 1
input 2 = 1

Delta weights = delta hidden sum / input data
Delta weights = [-0.088, -0.0452, -0.0319] / [1, 1]
Delta weights = [-0.088, -0.0452, -0.0319, -0.088, -0.0452, -0.0319]

old w1 = 0.8
old w2 = 0.4
old w3 = 0.3
old w4 = 0.2
old w5 = 0.9
old w6 = 0.5

new w1 = 0.712
new w2 = 0.3548
new w3 = 0.2681
new w4 = 0.112
new w5 = 0.8548
new w6 = 0.4681
```

new weights, between the input and hidden layers are calculated. The new weights = old weights + delta weights.

Figure 46*: Calculated output value after the first back prop. Note how the accuracy increased and the error decreased when compared to the 1st forward pass's result. The math indicates the steps taken to calculating the calculated output after each epoch. Note that the backprop takes the derivative of the activation function in the calculations (S'()), and goes backwards along the connections of the NN until it reaches the input layer. Keep in mind that there's a difference between sum and activation function's result throughout the calculations. [29]*

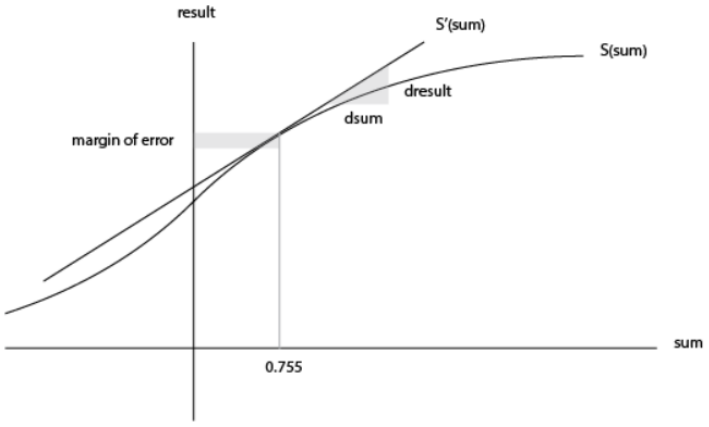


Figure 47*: Plot to help visualize how the derivative and the actual sigmoid curve play a part in backpropagation. [29]*

The margin of error (or loss as was initially defined by earlier/classical ML algorithms) is the difference between the targeted and the actual value that's calculated by the sigmoid function, S(), (as shown in previous figures). Once the derivative of the sigmoid function, S'(), is taken and it's input is the original sum* margin error itself, one gets to see that the sign of the value indicates whether or not the loss that's calculated by the output layer has to travel upwards (+) or downwards (-) to reach the intended targeted value. For instance, in Figure 42, delta output sum was calculated to be -0.13, thus the machine understands that it needs to adjust the weights so that it can reduce the calculated value, 0.77, to its intended targeted value, 0.

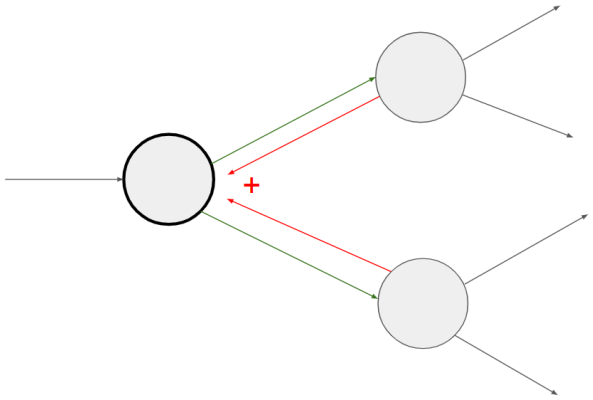


***Figure 48**: General ebb and flow of forward prop and backprop. Note that values, whether it be linear, non-linear or gradient based (derivatives) calculations, add up where there are multiple connections (or branches) at a node. This is evident in the previous figures when conducting the summations in each hidden or output node.[12]*

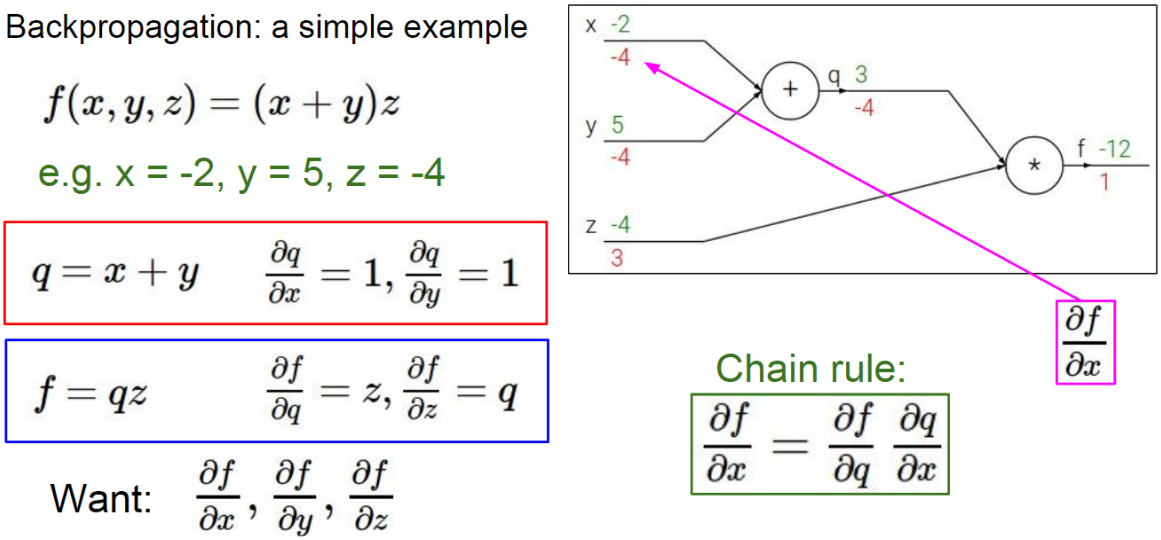
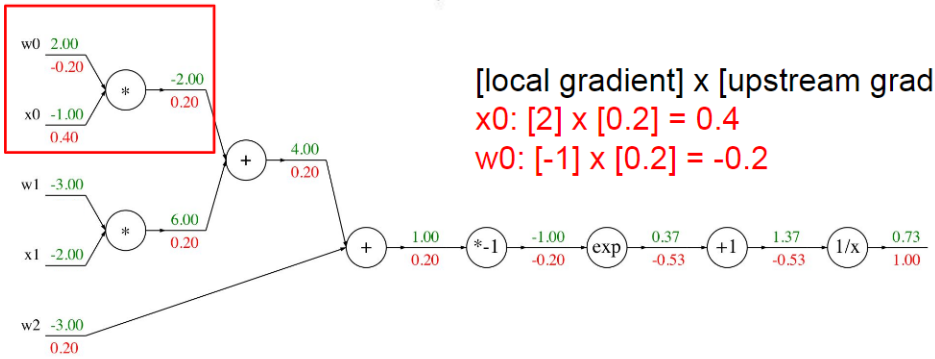


***Figure 49**: the partial derivations that take place throughout the entire nodal connections for back prop. Note that $\frac{df}{dy} = -4$ and $\frac{df}{dz} = 3$ (both values in red at the bottom side of their respective input branch ends). [12]*

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[local gradient] x [upstream gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

$\sigma(x) = \frac{1}{1+e^{-x}}$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

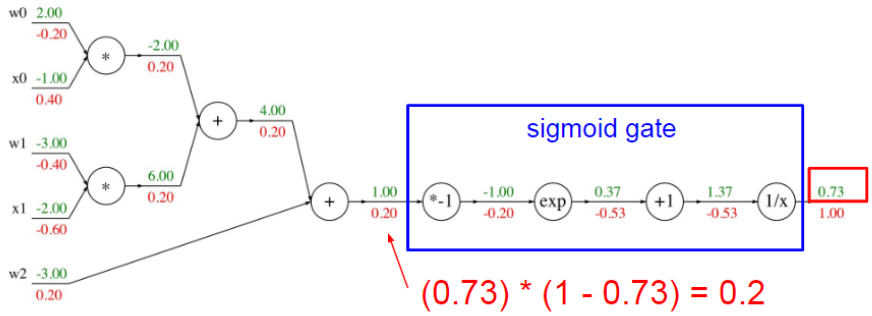


Figure 50: *Another example where the sigmoid function is the activation function for the last output result. Note, that it (sigmoid function) has been broken down to show how the mathematics between the derivative of the local gradient (current node value) and upstream gradient (previous node value) works out in back prop. The value (0.2) is the partial derivative of the sigmoid function with respect to it's input 'x'. [12]*

In conclusion, back propagation is a process that flows backward from the output layer nodes to the input layer nodes so as to update the weights to reduce the overall margin error loss; and it consists of partial derivations or Jacobian matrices, because as witnessed in the prior mathematics, since data is in matrix/vector format, elemental wise derivation was administered at each branch.

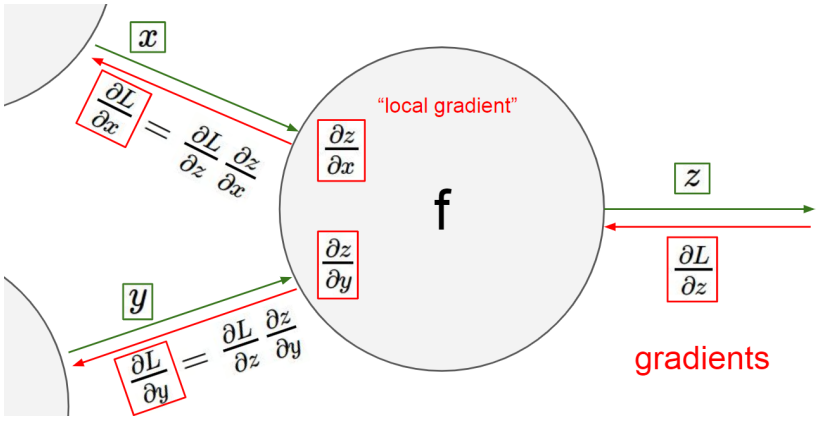


***Figure 51**: Jacobian matrices in backpropagation. [12]*

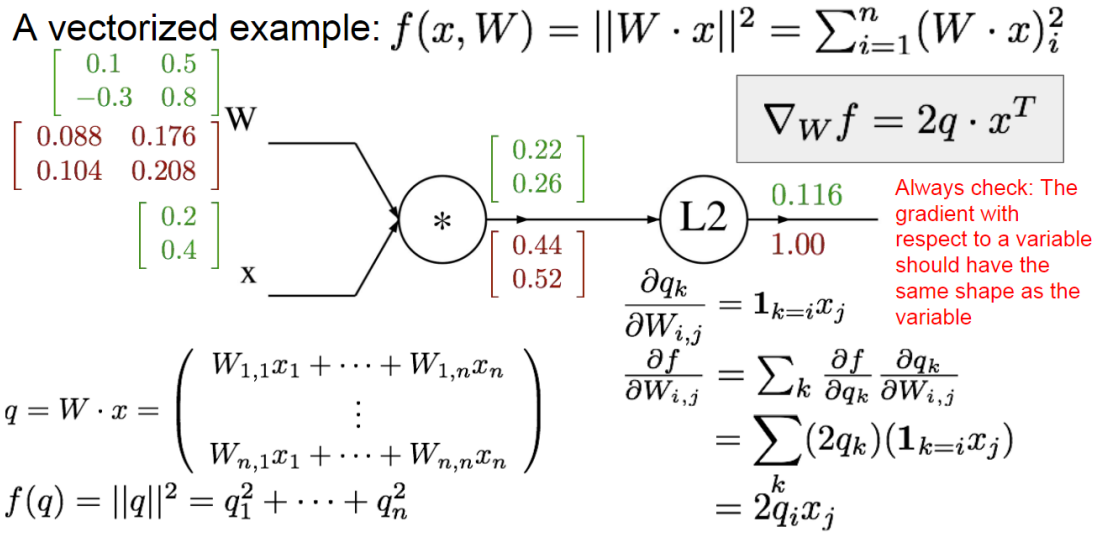


***Figure 52**: A vectorized example of back prop. [12]*

add gate: gradient distributor

max gate: gradient router

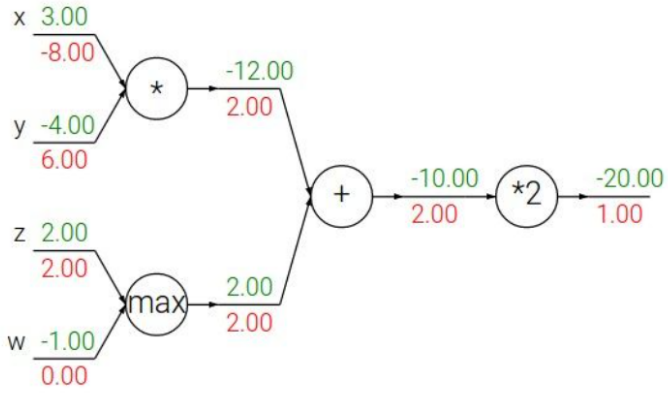


***Figure 53**: Another simplified version of forward (green) and back prop (red) using mathematics. Note that the addition gate is considered to be the gradient distributor and the max gate as the gradient router (determines which nodal value will cause a greater impact to updated weights during back prop).[12]*

3.1.3 Training a NN

Once data type (as discussed in section 1 in both Chapter 2 and 3) and NN model (the different types will be covered in section 3.2) has been selected (which is based on the data type and what the operator expects the machine to predict within the context of the application) then it's time to consider what resources to use to train the NN. In terms of hardware, most operators either end up using a GPU or a FPGA/CPU open source platform. Most individuals favor parallel computation (CUDA) and so opt for the GPU. Fortunately, due to the growing popularity of AI, Google has publicly provided Google Colab that contains Jupyter Notebooks (.ipynb) for operators to write their code freely in the Cloud with the option of using a GPU based server. Although there are multitudes of languages/libraries to use (ranging from Python Numpy, Javascript, C++, Caffe...etc.), this project has finalized on using Keras library with TensorFlow, as a backend engine (another choice would have been PyTorch or Theano), using Google Colab as the choice of IDE. The reasons for these choices were centered around budget, resource accessibility (everything python related is readily found online) and aid from forums and experts around the world who develop new packages for ease of use considering the application at hand, as well as, it's compatibility for the AI noob. Furthermore, due to the flexible and growing nature of python (due to ongoing support from global software/language developers), an operator can use a mixture of TensorFlow and Pytorch, or even a combination of different NN models in one API. An example of the latter would be a LSTM + Transformer + Autoencoder all in one sequential/functional API model.

In section 3.1.2 the basic flow of how an aNN works and the primary elements that make up that architecture was graphically discussed, including the role back prop plays on weight update and how error (or loss) is calculated and considered in the overall process. Error is calculated after the output node and is the comparison/difference between the estimated calculated value to a sampled data called the target or expected output. There are two general branches of error calculations, and they are : classification and regression loss (as shown below). Thus, in training the NN, depending on the data type and what the NN model is expected to predict, either the classification or regression loss type will be used to calculate the error between the output result and the targeted (or referenced) value. The error found determines the way the back prop process will flow and which weights are largely or less likely impacted to mitigate it (that error).

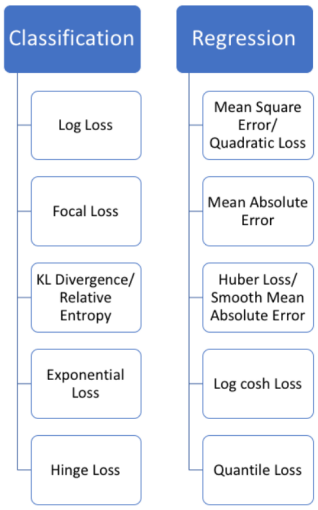


***Figure 54**: List of the different types of classification and regression loss used in NN error calculation. Note that the terminology used is the same with respect to the data type categorization where classification refers to discrete, qualitative data types (such as images) and regression refers to continuous , quantitative data types (such as numerical). [31]*

The commonly used types of error for classification data are: SVM, hinge loss, softmax (also called log loss). Since this project is using a regression data type for analysis, the squared mean error loss was selected (as will be evident later in Chapter 5 and the associated Appendices).

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x;W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss

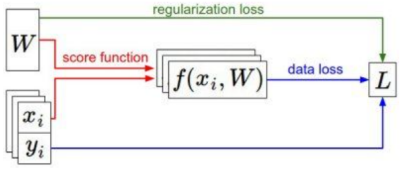
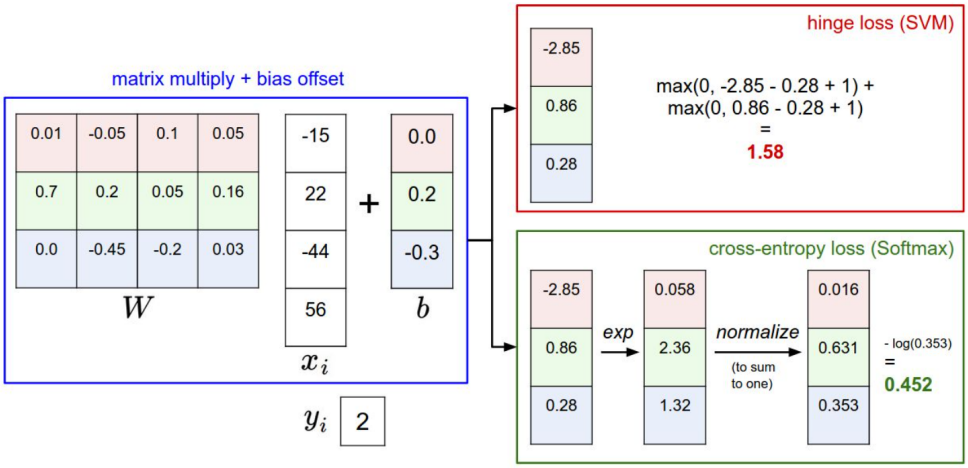


***Figure 55**: List of error functions (or data loss functions) for a classification data type set . Note that the linear 'sum' function discussed in an earlier section, where f= W*x (W=weight, x = input), is also called the score function, and that the last function labeled as 'Full loss', is called as such due to the Regularization loss term (R(W)). The Regularization term prevents the NN model, during training, from overfitting to the data, thus ensuring flexibility in the overall prediction curve of the model. [12]*



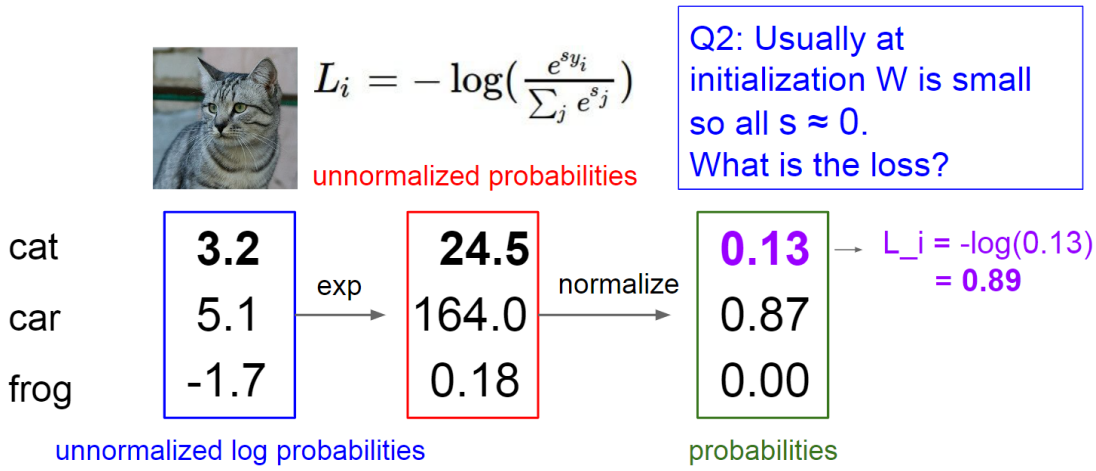


***Figure 56**: Block diagram of how SVM differs from Softmax. Note that both are data loss functions with respect to the activation function's output result. 'W' is the weight matrix, 'x_i' is the input (in this case flattened vector of an image's pixels), and 'b' is the bias. Note that SVM loss calculations go by the maximum score values per feature while softmax is a log function that uses probabilities; thus a probability that is closer to 1 indicate that the machine (or NN model) estimates that class to be the targeted, while a probability of '0' indicates that the model 'thinks' that class is not the targeted label. Example for SVM is shown later in section 3.2.[12]*

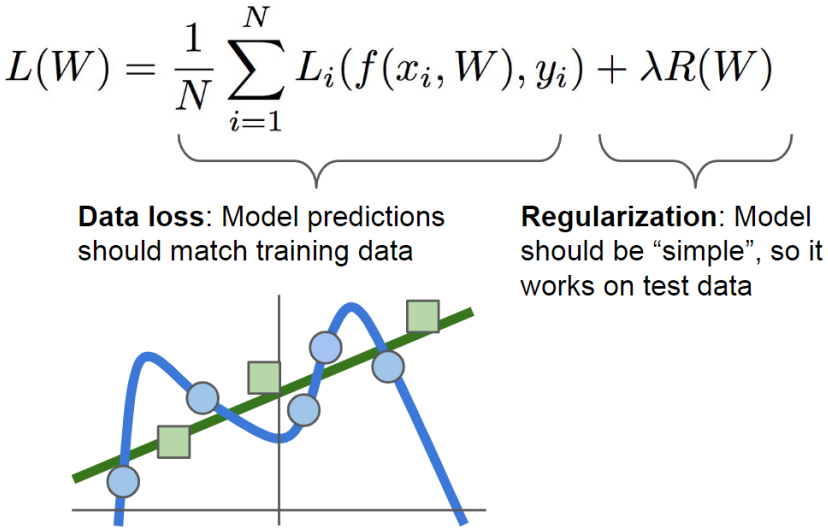


***Figure 57**: Full loss equation includes the regularization loss term. This term prevents the NN model from overfitting (predicting only what's given instead of trying to come to different conclusions, or 'learning'). Note that λ is a hyperparameter defined as the regularization strength (thus, can be modified). There are L1, L2 and L1 + L2 regularizers that can be used. [12]*

Besides the regularization term to modify the way the NN 'learns' the data, there are other methods where the data is modified in efforts to 'tune up' the learning process, such as: batch normalization (is done within the model where it re-normalizes the data - recreating that Gaussian distribution - after a layer that uses a nonlinear activation function, during the training process), max-pooling (downsampling an input by only considering the max values), and dropout (certain nodes that end up having zero results or very little impact after the back prop are dropped from the overall architectures during both forward and back prop, or nodes removed by operator to limit the type of features that the NN uses to 'learn' from the given data sample to reduce training time and memory usage).

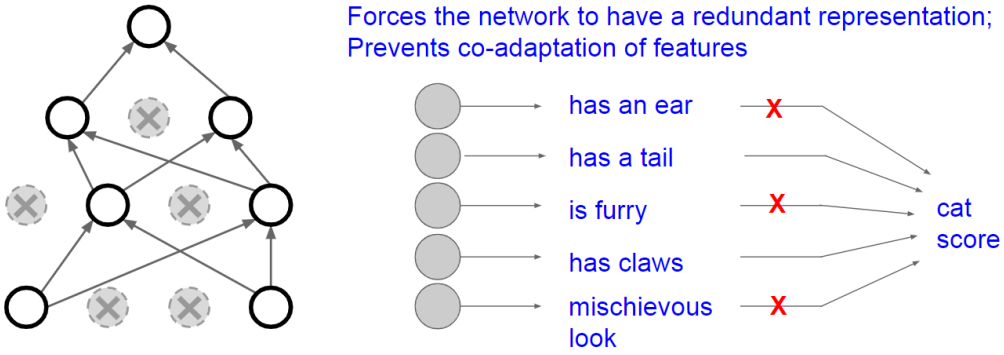


***Figure 58**: Example of dropout in a NN model that has images as inputs, and in this case the input is a cat. Note that some of the features that make up the cat image are removed to improve training efficiency. [12]*

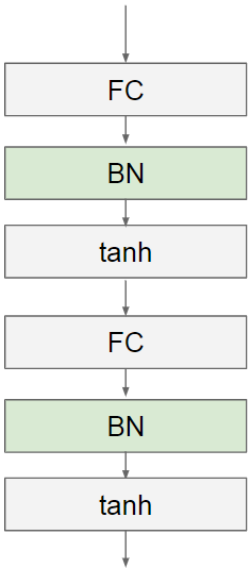


***Figure 59**: Batch normalization (BN) layer usually occurs after a fully connected (or dense) layer*

(FC), or a CNN layer, but before a layer that uses nonlinear activation function (tanh). [12]

To visualize the error calculation process, besides the back prop examples, a colored gradient of local minimas is often used to offer visual aid of the way the NN 'learns' as the weights are updated during the training process. The behavior in the way the NN converges to a target value, through the reduction of the data loss (error) and weight updates, is called optimization, and there are several types of this that are used to train a NN (and gradient descent is one of them). The hyperparameter for any optimizer is the learning rate (which will be evident in the compiled code). The other types of optimizers are:

- Stochastic Gradient Descent (SGD)
- 1st Order Optimization (instantaneous linear slope)
- 2nd Order Optimization (parabolic slope, very handy in finding local minima and saddle points)
- SGD + Momentum (because SGD happens to be a very noisy optimizer as it tries to reach the ideal point of convergence for every data sample point, this is because it has a tendency to get 'stuck' at local minimas and saddle points along the way.)
- Adagrad
- RMSProp
- Nesterov Momentum
- Adam (most ideal for it embodies Adagrad, RMSProp, bias corrections, and Nesterov Momentum attributes)

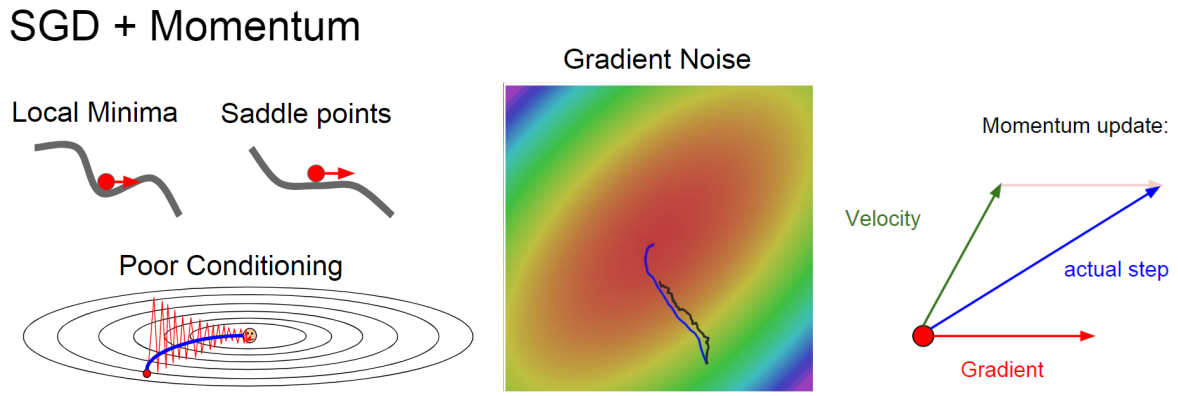
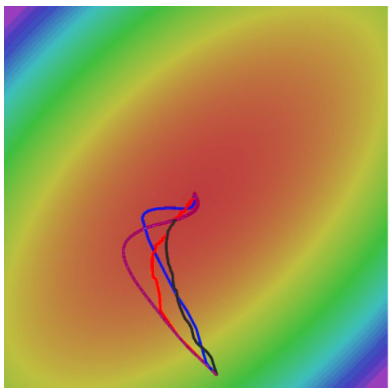


***Figure 60**: SGD + Momentum Optimizer [12]. Note that although the Momentum update prevents the SGD optimizer from getting stuck in local minimas and saddle points, it still is noisy. This is due to the behavior of the SGD as it completes an iteration (1 forward pass + 1 backward pass) for each data point.*




```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

***Figure 61**: Visual comparison between the optimizers. Note that Adam and the SGD + Momentum optimizers share similar characteristics - less noise as it converges. The full Adam formulation contains the three optimizers and a bias correction. [12]*

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

***Figure 62**: Full batch (or Vanilla) gradient descent. Gradient descent uses the concept of 'local derivative' (or 'slope') at every particular 'step' or interval in time to converge to a desired minima; as visually expressed via the varying 'slopes' of a valley or the color plot where the training curve travels to the minima that's displayed in red. Remember that a gradient ($\nabla f(x_t)$), in mathematics, is the vector that sums all the partial derivatives along each dimension, and that a slope is, in any direction, the dot product of the direction with the gradient (or in short, the derivative). The direction of steepest descent is the negative gradient [12].*

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up "velocity" as a running mean of gradients
- Rho gives "friction"; typically rho=0.9 or 0.99

***Figure 63**: Numerical calculations behind SGD and Momentum. [12]*

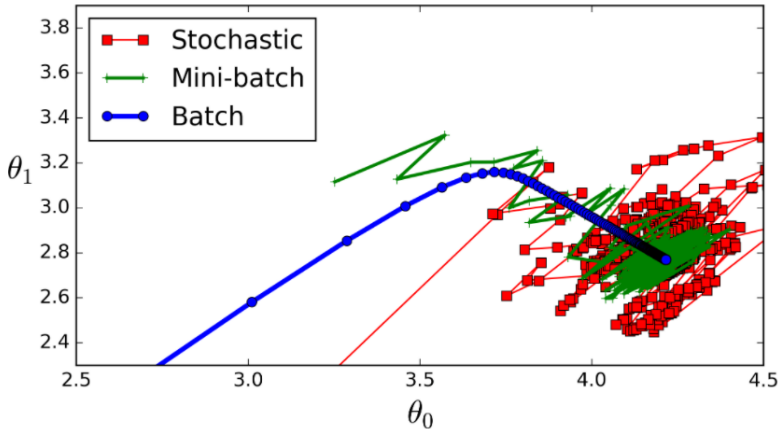


***Figure 64**: SGD (batch size = 1), Mini- batch (gradient descent where batch size >1), and Batch (also called Vanilla or Full gradient descent where batch size = 0). Note that noise is reduced with either Full batch and Mini- batch as the NN model converges to the actual or desired minima.[32]*

As one is training the NN model, certain plots are expected to be embedded in the code to ensure that the model is indeed learning properly.

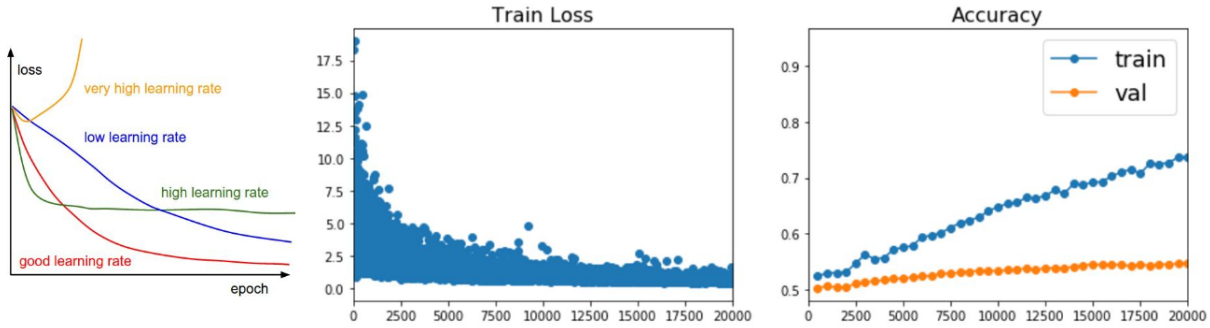


***Figure 65**: The training loss (error) curve indicates whether or not the model is learning appropriately with respect to the learning rate. As the plot on the far left suggests that the ideal learning rate is the red curve, where it is a negative exponential curve that eventually (gradually) plateaus; unlike the green curve where it plateaus early or the blue curve where it is linear or the yellow parabolic curves are high, low and very high learning rates respectively. The accuracy plot, on the far right, compares the training and validation accuracy values; typically it is ideal to have the validation curve larger than that of the training curve for the loss, but vice versa for the accuracy); however, if the gap between the two curves is large than that is a case of overfitting and thus a regularizer or the regularization term's strength (λ), if used, needs to be strengthened (increase its value). Also, overfitting occurs when there's a gap between the training and validation curves in the loss plot and underfitting occurs when the training is below the validation curve in the accuracy plot. Either case, the data needs to be re-evaluated (reshaped differently or include more features), the NN model needs to be simplified (less number of layers and less number of neurons), learning rate adjusted, more data needs to be presented for all: training, validation and test, and regularizers (may) need to be added. [12]*

Be aware that oftentimes there are applications that can be solved using classical ML algorithms (such as linear regression, HMM, Monte Carlo, SVM...etc.), and , as useful as they are in predicting the expected outcomes, these models don't 'learn'. Classical ML algorithms have laid the foundation for NN's, as will be discussed in the next section.

3.2 Other Architectural NN Models vs. RNNs

3.2.1 Background

Understanding the different types of NN models, as well as the backbone of its architecture, will allow the operator to create a NN model that will meet the objective of their study.

Popular classical machine learning models that laid the foundation for NNs:

- Linear regression: Similar to that of the linear trendline regression tool used in data plotting in excel, and in the context of NN a perceptron (where its activation function is a linear equation, y = m*x + b or f = W*x + b, and has no hidden layers - so just input and output nodes).

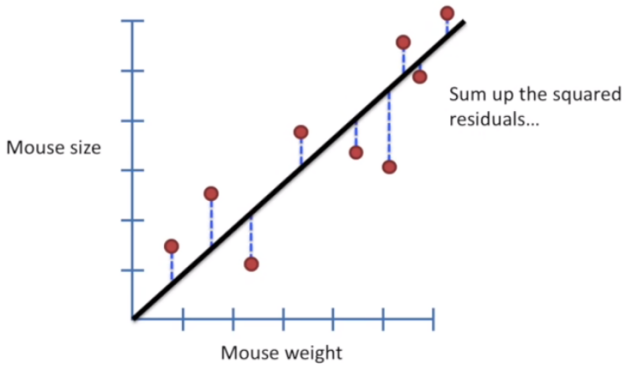


***Figure 66**: Linear Regression; can be used for either classification or regression problems [33].*

- Decision Trees : Split complex datasets into a tree like structure and can be used for both regression and classification type of problems. Decision trees can work with many variables but have issues with overfitting.

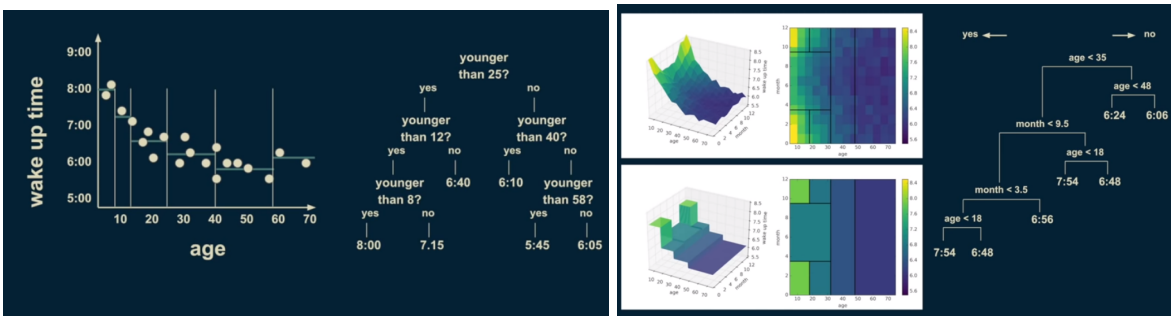


Figure 67*: Decision Tree modeling. Alternative to decision tree modeling are random forests, Naive bayes and linear/polynomial regression modeling. [33]*

- SVM: Type of classification error calculation where the score (or activation function's result) is compared to the target value. It is embodied by NNs specifically for supervised learning models that use classification (or image) based data. (Not to be confused with support vector regression) It is also referred to as the 'hinge loss'.

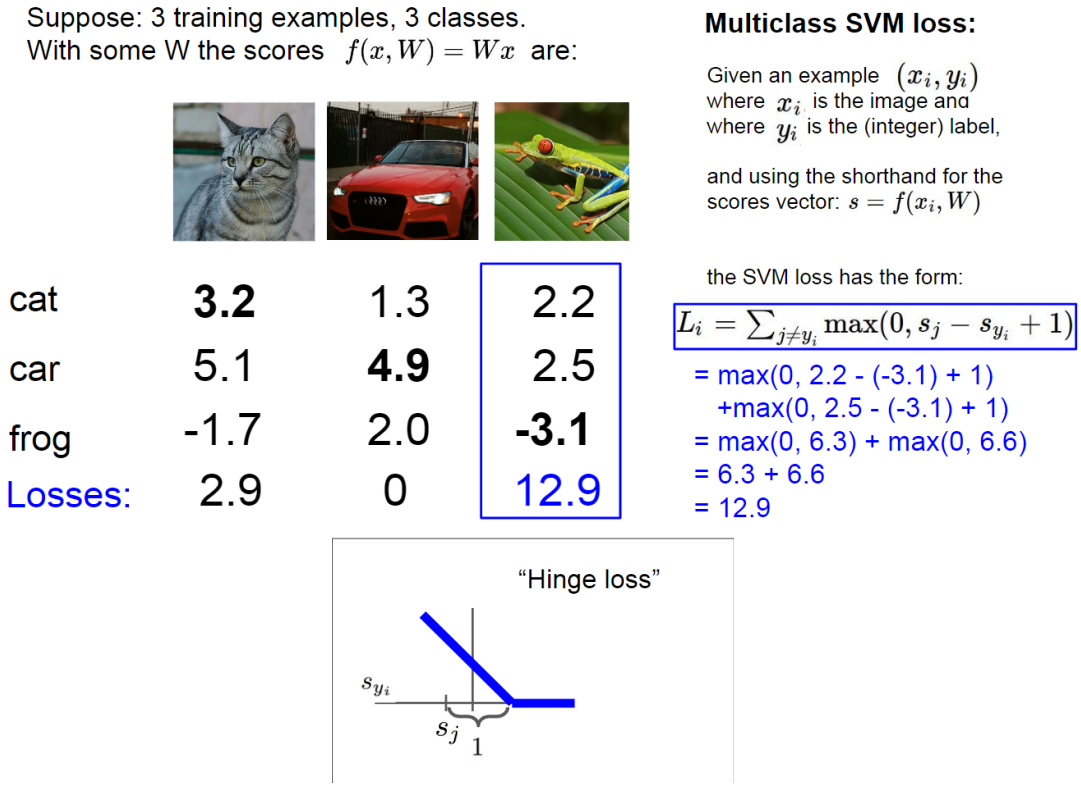


Figure 68*: Grid displaying how SVM works with classification type of data (images). The class labels on the far left (cat, car, frog) are the labels the NN model predicts the input (top images) to be. SVM is a classification error ('Losses' in blue at the bottom far left corner) used for supervised learning NN models. Note that the score value (s_j) calculated from the NN (in this case, f= W*x + b) for each class*

image presented, are all the values except for those that are bolded. The values that are bolded are the true label values (s_{yi}), or score values that are also calculated by the NN model but with respect to the image that's fed as an input (images at top) and the expected correct label. All the score values, calculated per class, are compared at every run (column) to find the max score value. If the max score value coincides with the label that the NN predicted and it matches to the input image, then the SVM loss value (L_i) will equate to 0. If it is completely opposite, however, where the max value isn't the expected true label value (s_{yi}) then the SVM loss value (L_i) will be higher than the ideal 0. Looking at the grid, the third run, or third column, the NN gave a negative value for the frog label when the input was actually a frog, thus the SVM loss calculated was pretty high. This is an indicator that the weights need to be updated and so back prop will update the weights so that the NN model will make better predictions (score values). Note that the second run (or column), the SVM loss is 0, this is because the max score value happens to be the actual true label of the input image. [12]

- HMM : Hidden Markov Model, is the precursor to RNN's and Autoencoders. This introduced the idea of maintaining the sequential order of data, thus time series data were the inputs and outputs.

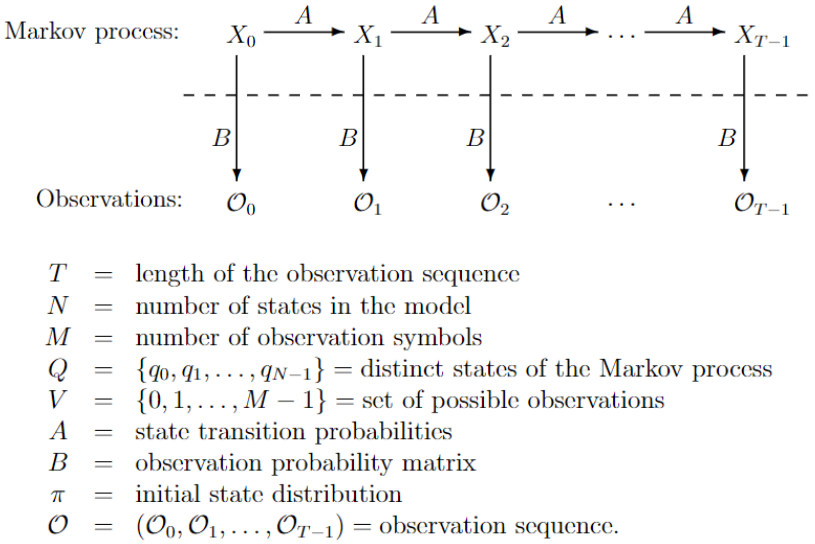


***Figure 69**: Hidden Markov Model (HMM) diagram; where 'O' is the observable outputs, 'B' is the matrix that connects the hidden observations to probabilities (or the observation probability matrix), 'A' is the state transition probabilities, and 'X' is the hidden state sequence. The Markov process is hidden behind the dashed line and is determined by the current state and the 'A' matrix. The term 'observation' is akin to that of 'variables' or 'features' as in a NN model. 'N' is the number of hidden states in the model, however it is important to keep in mind that 'hidden states' in a HMM model does not equate to 'hidden layers' in a NN model. The term 'hidden' in the context of HMMs means that the state is not directly observable because, in most cases, they are values pertaining to past data. Also, the matrices for π, A and B are row stochastic, meaning that the values within the matrices are all probabilities and each row sums up to a total value of '1'; and they behave as 'weights' are in a NN*

model (especially to that of a LSTM). [34]

An example of a HMM at work:

$$\begin{array}{c} \\ H \\ C \end{array}\begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array} \quad \text{equals to} \quad A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

$$\begin{array}{c} \\ H \\ C \end{array}\begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}. \quad \text{equals to} \quad B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix}$$

$$\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix} \text{ is the initial state distribution.}$$

***Figure 70**: For example the A matrix is the hidden state matrix, where in this situation, is the temperature of the environment, where 'H' = hot and 'C' = cold. The 'B' matrix has both hidden (H,C) and observable states (Small = 'S', Large = 'L', and Medium = 'M', which are attributes of tree rings), thus indicating the correlational relationship between two distinguishable states, in this case, temperature and tree ring sizes, to predict hidden states in the near future. π is the initial state matrix, also row stochastic, for the hidden states. [34]*

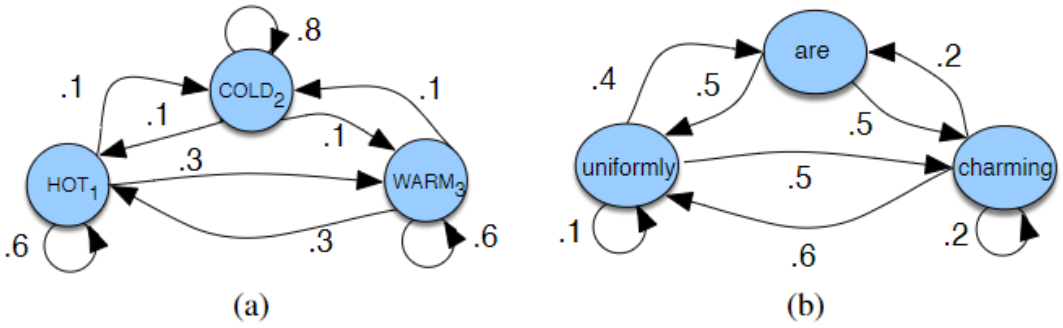
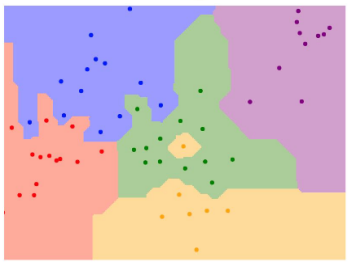
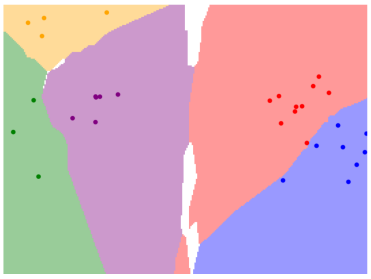


***Figure 71**: Another way of visually explaining the Markov process. These diagrams, (a) and (b), are called Markov chains where the hidden states (similar to that of the 'H' and 'C' shown in the prior figure) and the transitions (similar to 'A' matrix in earlier example) are displayed as balls and chains, respectively. The π matrix is not presented in the diagram and should be a given 1x3 matrix, since there are a total of 3 states in either diagram ((a) and (b)). The values along the chains are the row stochastic probabilities that would be found on a 3x3 'A' matrix. The parameters of the HMM are the A and B matrices and can be trained via the Baum-Welch or the forward-back prop algorithms. The process of finding the sequence of hidden states from the observable states is called decoding or inference and the Viterbi algorithm is usually used. [35]*

- K-Nearest Neighbor (K-NN): Classification method of multivariate data, where especially in applications containing object detection (a.k.a images), the machine is able to better classify different types of pictures during the training process by increasing the number of k-fold validations. K is the hyperparameter and can be adjusted and is attributed to the number of validation folds the training data set is tested against.



K = 1



K=5

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

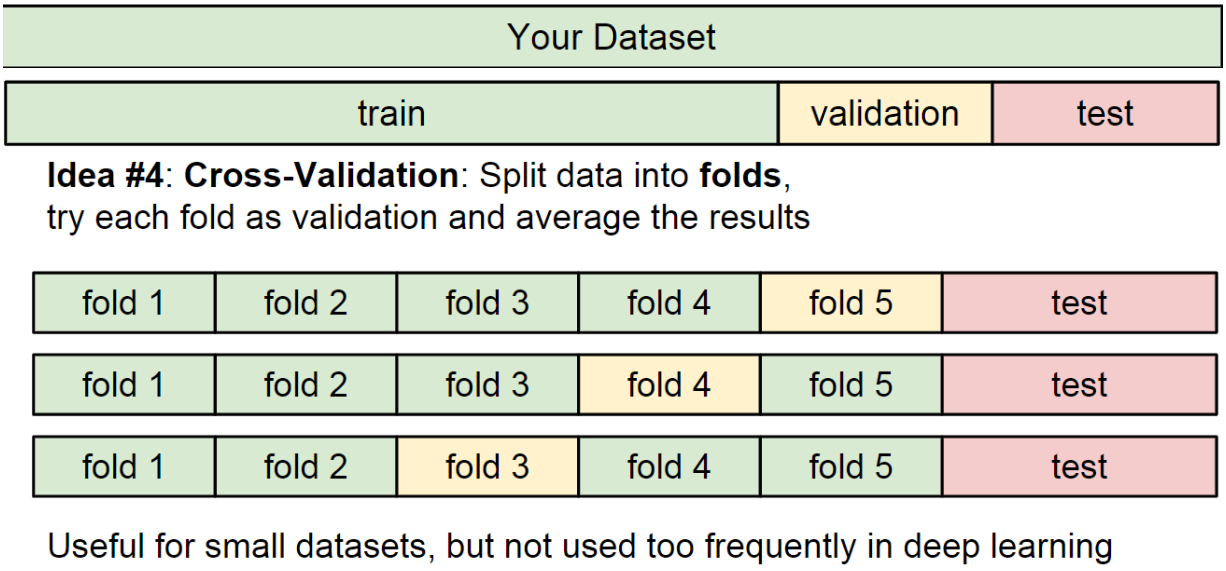


***Figure 72**: K-NN, where 'k' (or the number of validation folds) is the hyperparameter. 'L1' and 'L2' are the different distance calculations (between pixels within an image) that are done when classifying certain pixels for each image category (or class). L2 tends to create smoother divisions amongst the data set's categories/classes. Cross-validation is where there is more than one validation set (k >1) to test the training data against before comparing it with the final test set. k=1 is where there is only one validation set for the training data set to compare against. [12]*

3.2.2 CNN

Although this study uses a regression NN model, CNNs are a great segway for beginners to visually understand how a NN model processes data.

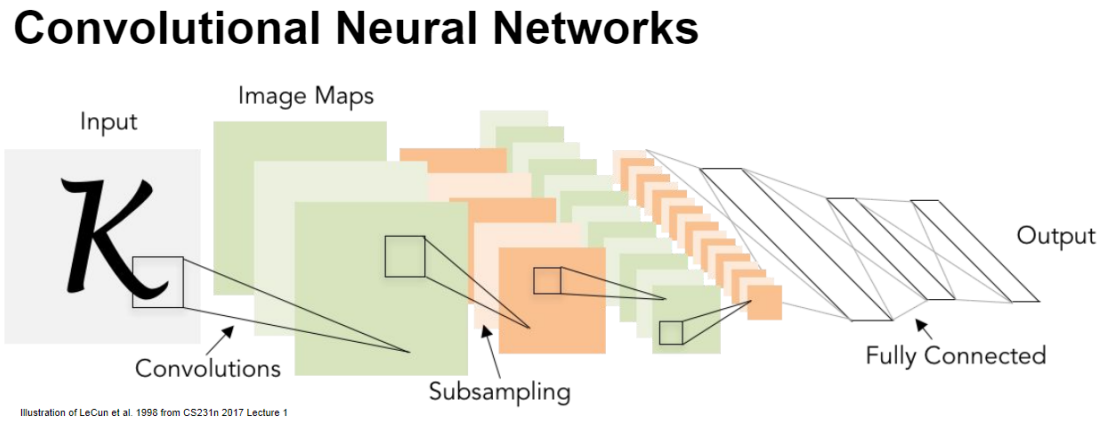


Figure 73*: CNNs are a series of activation maps created by weight matrices that act as filters. These filters, or activation maps, extract certain features of the overall input, and are then connected to fully connected layers and eventually to an output layer (which, also, happens to be a dense layer). [12]*

CNNs, or Convolutional Neural Networks, is a NN model that extracts features, using a filter layer - or convolutional layer, from an input. The inputs are typically an image or text because such data can be presented in grid form so that the filter layer can easily slide (or convolve) over and conduct vector and matrix multiplication (or element-wise dot products). CNNs are found to be used in a variety of applications (such as object and anomaly detection and image captioning) and are at times used in combination with RNNs or Transformers. AlexNet and VGG-16 are such examples of CNNs. Furthemore, their hard code flexibility in parallel computations, ability to work with other NN models and backend engines (i.e. Theano, Pytorch, TensorFlow) and the capabilities to have split branches to train on several types of features is what makes them the 'go to' tool to be used for multiple applications.

However, in order to fully understand the internal operations of how a CNN operates, an introduction to how a typical fully connected (or 'dense') layer computes is necessary.

First off, any NN model that isn't specified (such as CNN, RNN, GAN, GRU, Autoencoder...etc) is typically a linearly activated fully connected layer.

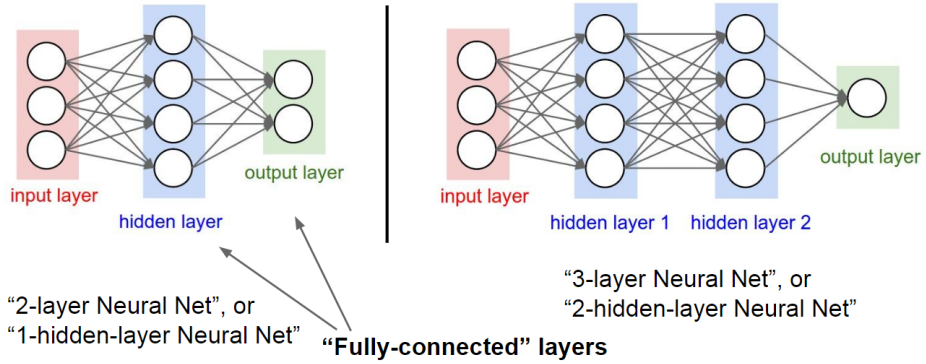


***Figure 74**: Fully connected (or dense) layer. Note how the labeling of a NN is done, the input layers are not considered when calling out the number of layers in a NN model. The dense layer has a linear activation function (f = W*x + b) [12]*

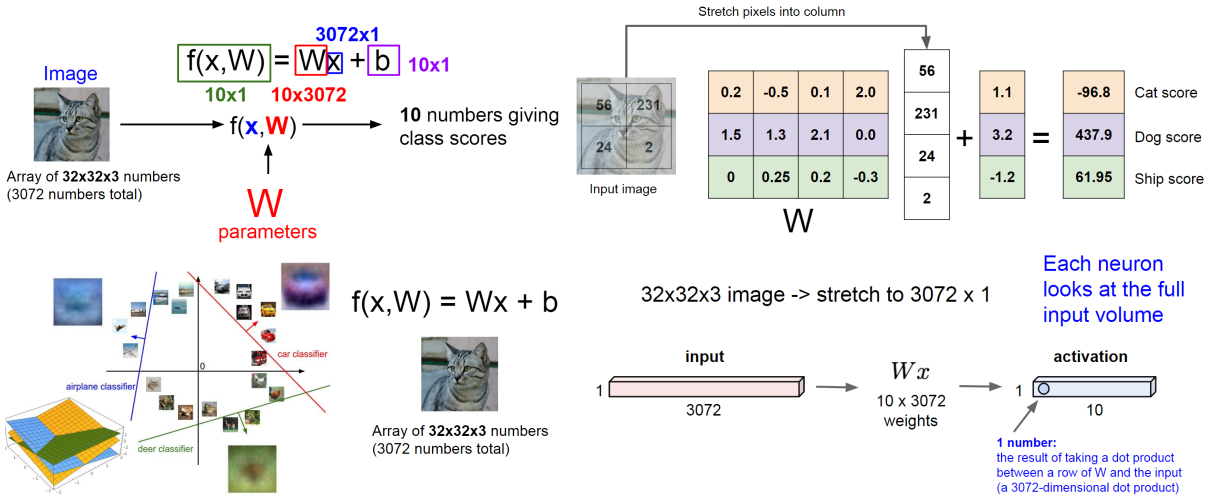


***Figure 75**: Dense, or fully connected, layer operates by the linear activation function, f = W*x + b, and looks at the input volume as a whole. The far upper left image depicts the size of the W, x and b matrices when an input image, such as a cat, is used. Note that 'x' is the flattened vector of the overall input dimensions, where 32 x 32 x 3 = 3072. Since, in that particular problem, there are 10 images or classes, and each input must have its designated weight, W. Thus, there are a total of 10 weights used, in which each has a size of 1x3072. Therefore, able to complete the matrix multiplication or element wise dot product before adding the bias, which is a 10x1, b/c there are 10 inputs. The picture on the upper right hand corner has the input image of a cat, and it's 4 pixel values have been flattened to create the 'x' vector that's multiplied to the initialized 'W' (weight) matrix, and it's added to the bias vector. The lower left hand corner image depicts how the 10 categories of images are classified in a linear fashion when using the linear activation function (very similar to the 'color' plot of the K-NN algorithm. At the lower right hand corner is how the fully connected layer, graphically, processes the element-wise dot product, of the whole input volume, and equates each 'W' * 'x' to a node. Thus, at the end of the dot product for the 1st dense layer, there will be a total of 10 nodes. [12]*

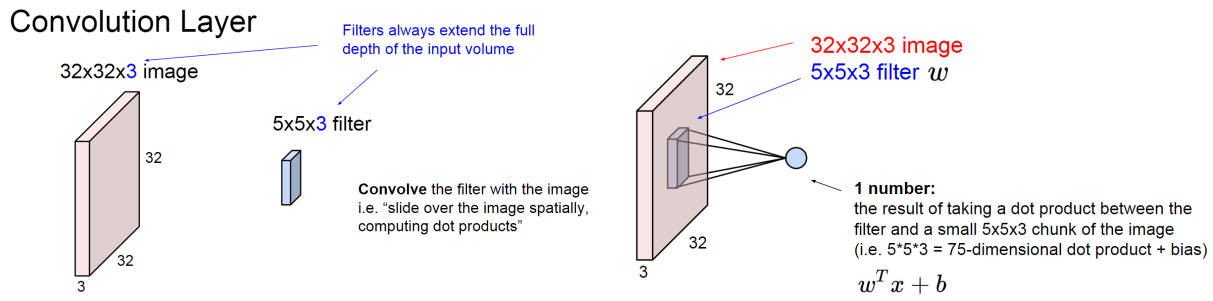


***Figure 76**: The CNN layer is created by a filter, weight (ω), of smaller size that is multiplied to the input. Note that the filter depth is the same as the input image's depth size. One activation map, is created by said filter, and equates to a node. [12]*

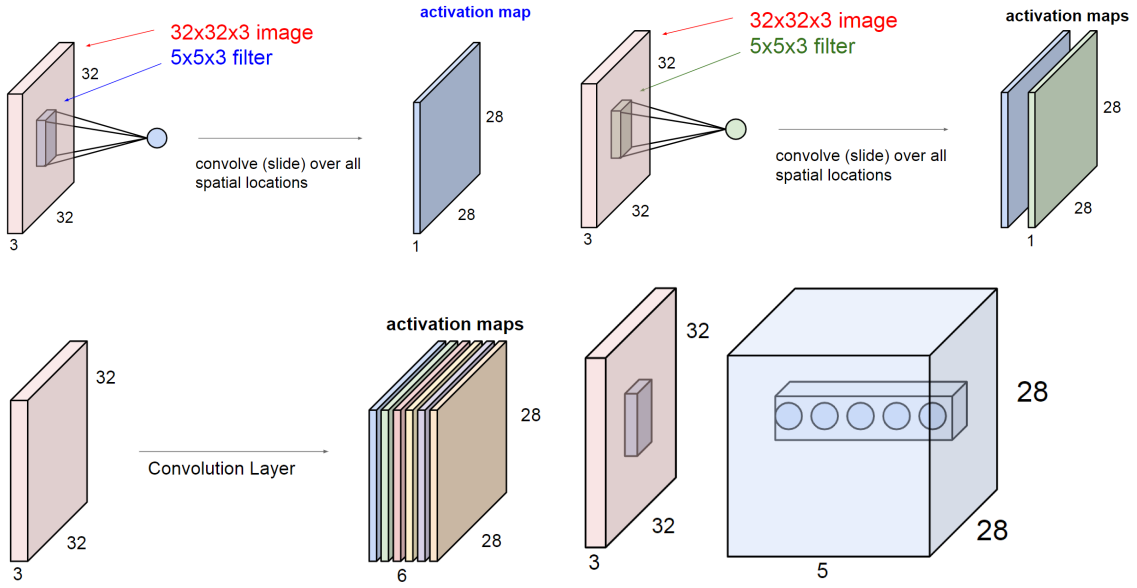


***Figure 77**: The two top images show how two different activation maps are created by two filters. The lower left picture shows an example of one convolutional layer that has 6 filters (activation maps). Remember that 1 convolutional weight = 1 filter = 1 activation map = 1 node, all within a convolutional layer; as indicated in the image at the lower right hand corner. Note that in this example, there are 5 nodes within a convolutional layer, thus, there are 5 filters extracting the same region of an image but analyzing/capturing different features/aspects (of that region). [12]*

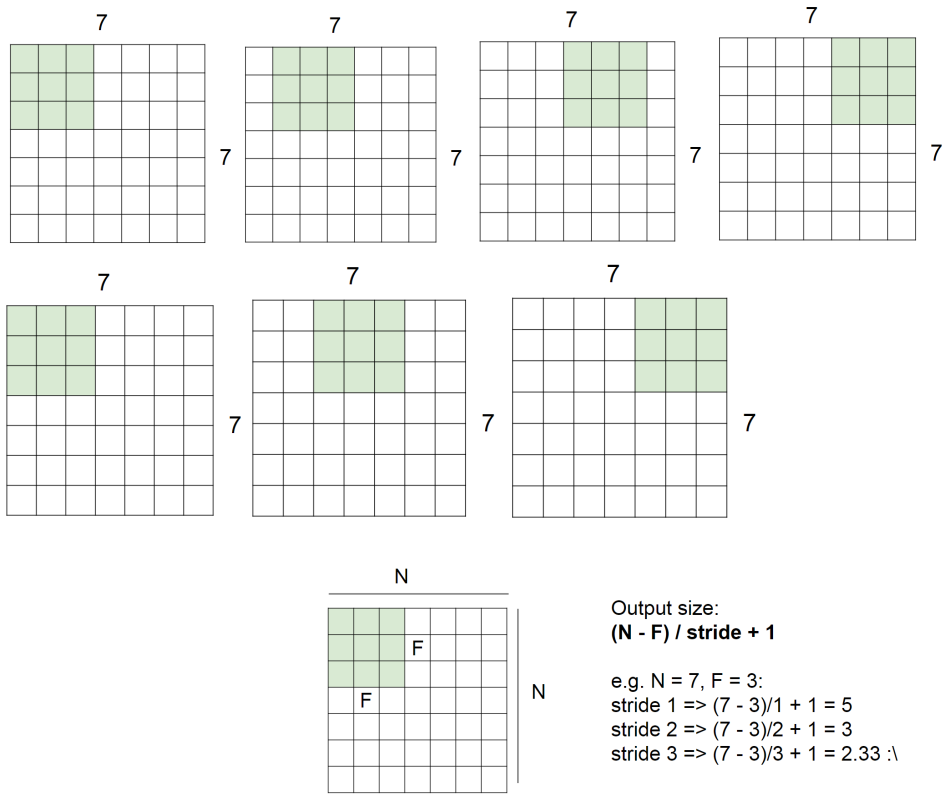


Figure 78: *3x3 filter, ω, slides over the overall, 7x7, input image. Note that the first row has a stride of 1 and the second row has a stride of 2. Stride means the amount of columns the filter moves across one set of rows that the filter covers. In the first row the overall output is 5x5 (has a total of 5 slides) and the second row has a 3x3 output (because it has a total of 3 slides). Note that 3 strides would not work for the given input image. The last image shows the calculations for the overall activation map output due to the filter size, input image size, and the number of strides selected [12]*

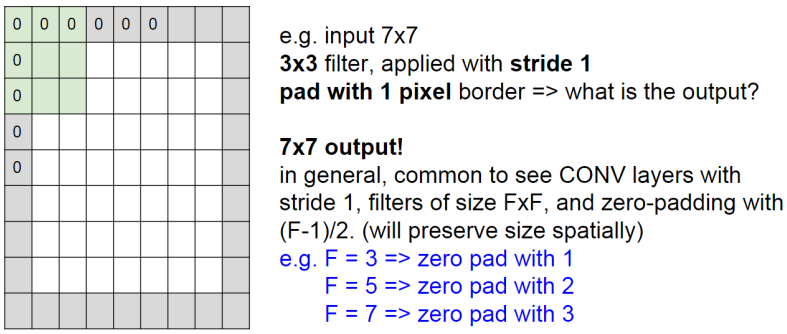
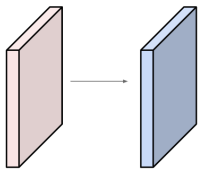


Figure 79: *Zero padding is a technique to ensure that the overall output size equates to that of the original input image size, despite the filter size and number of strides selected. [12]*

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Figure 80: *Overall calculation of the output convolutional layer. The total number of filters, the filter size, and the amount selected for strides and pads are given. Note that the calculation of parameters (weights + bias for the linearly activated and convolutional networks) is different for each NN model.[12]*

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Figure 81: *Summary of CNNs with respect to the hyperparameters and outputs. [12]*

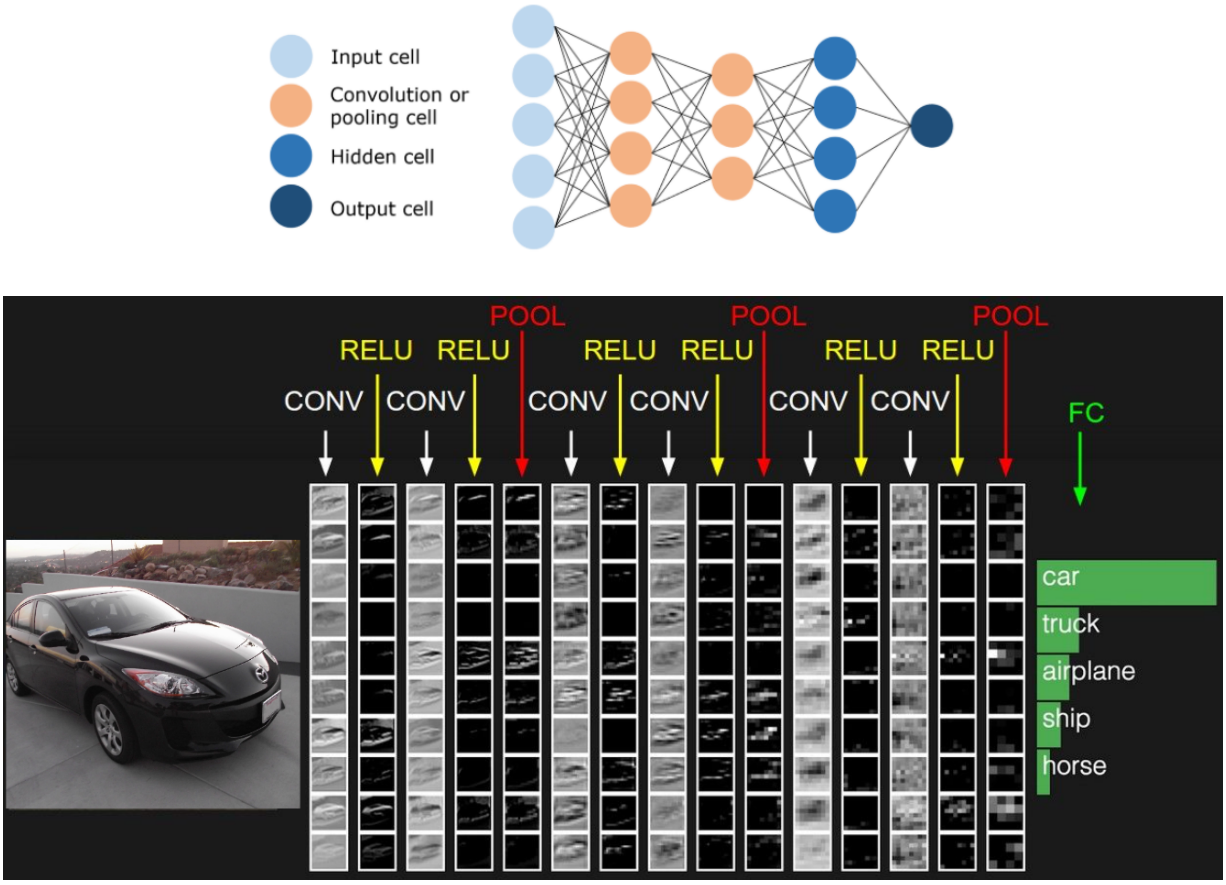


***Figure 82**: Typical layering of a CNN. The top image shows a nodal and branch CNN architecture [36]. The bottom image depicts the CNN architecture layering [12] via actual visuals that the model is rendering as inputs and outputs at each layer. 'POOL' = max pooling, 'RELU' = ReLU activation function based nodal layer, 'CONV' = Convolutional layer, 'FC' = Fully Connected layer.*

3.2.3 RNN

RNNs (or Recurrent Neural Networks) are NN models that are typically used for regression time-series type of data, or text-by-text language and temporal/audio processing, that follow a sequential order. RNNs have outputs that loop back to prior nodes, as well as, to future nodes; hence, the 'recurrence' term used [36]. The purpose of a RNN is to make informed decisions and better prediction models from previous data [36].

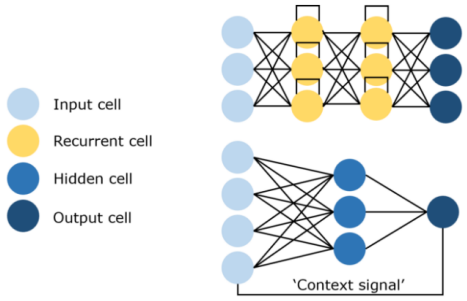


***Figure 83**: RNN nodal and branch architecture [36].*

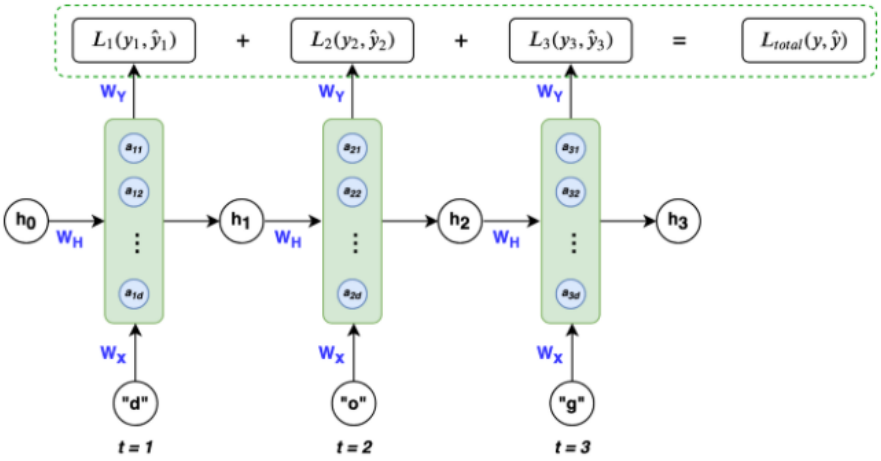


***Figure 84**: Internal spatial process of a RNN model as it learns the word 'dog'. Note that 'L' = losses calculated, 'w' = weights, 'H' and 'h' = hidden, and 't' = time. Note that the model is showing 3 hidden layers and each hidden layer is processing a letter from the word 'dog'.[37]*

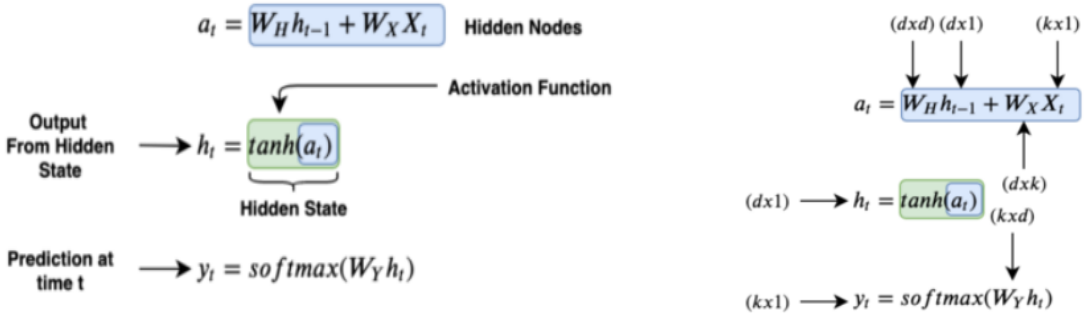


***Figure 85**: Equations used within the RNN processes and their dimensions. Note the equation 'a $_t$' is the transitional linear calculations that take place on the branches in between nodes, 'h $_t$' is the activation function within the hidden layer and takes 'a $_t$' as its input. 'y $_t$' is the output loss function - in this case it is the softmax logarithmic function. In the right hand image, 'k' is the dimension of the input vector and 'd' is the number of the hidden nodes. Inputs tend to be in flattened vector form. [37]*

Other types of RNNs are: LSTMs, GRU, ResNet and ESN, and just like CNNs, can be combined with other NNs, such as: C-RNN-GAN (Continuous Recurrent Neural Network with Generative Adversarial Networks.

3.2.3 Other Types of NN's & LSTM

Comparing and contrasting different types of NN models with respect to their contextual use.

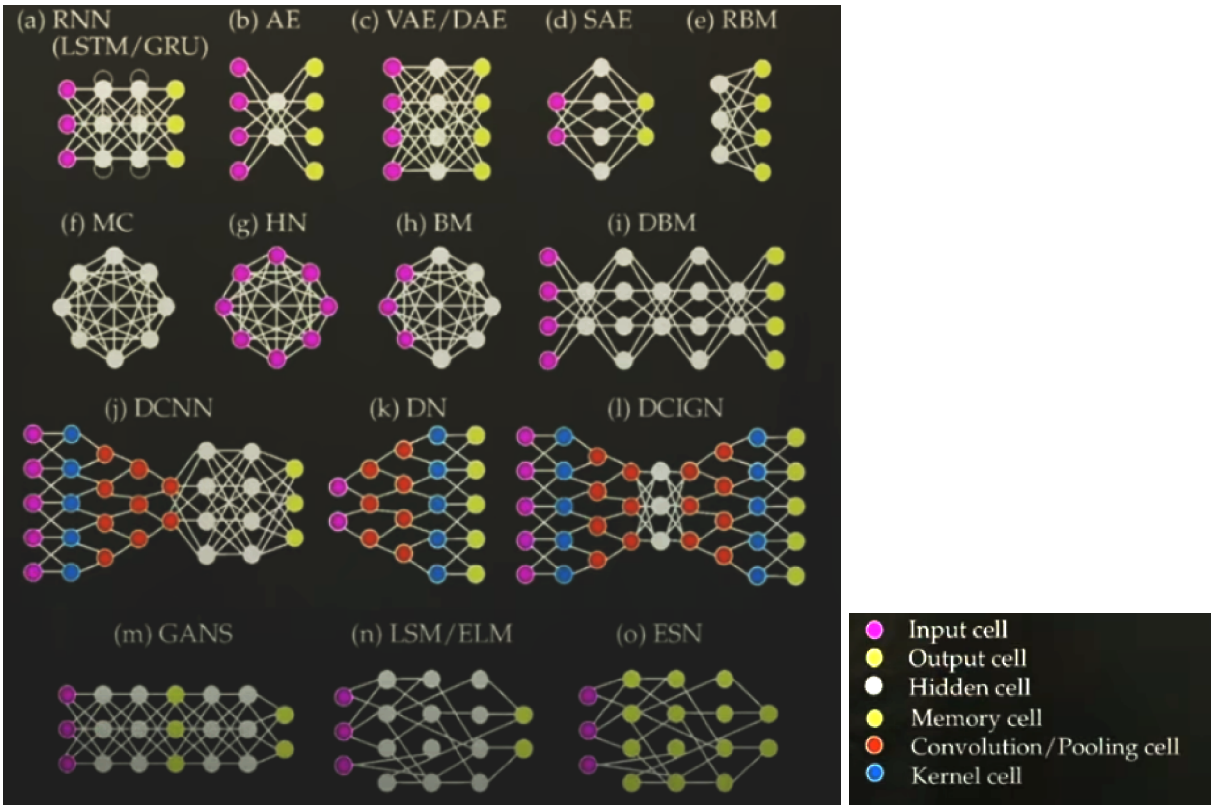


***Figure 86**: Other types of NN models and their typical ball and stick architecture. Note that 'kernel' means weights. [21]*

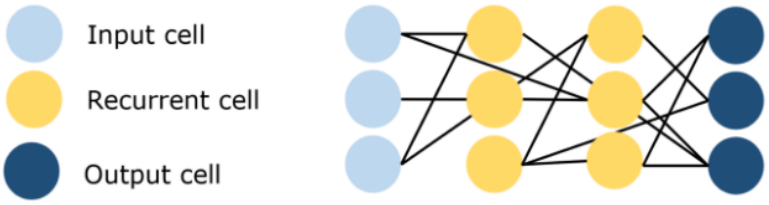


***Figure 87**: Echo State Networks (or ESN) are a type of RNN that have sparse connections in between their hidden layers due to temporal/audio learning. [36]*

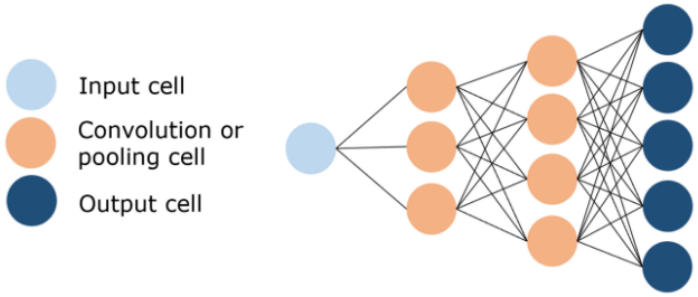


***Figure 88**: The Deconvolutional Neural Network (DNN). As the title suggests, DNN is opposite to that of CNN in that it takes 'noise' or 'bare foundation' as input, and creates the whole context (or image) rather than deconstructing that image. For instance, it creates the whole plot of a story based on just three sentences. [36]*

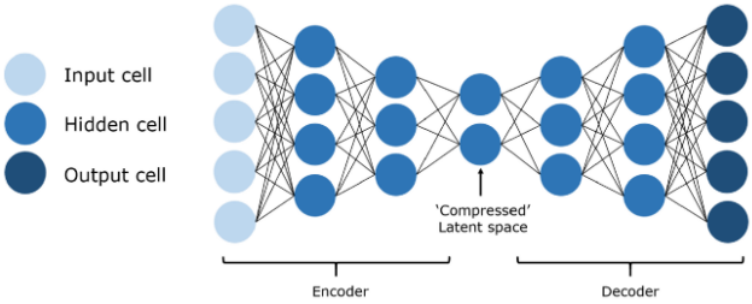


***Figure 89**: Autoencoders (AEs) are a combination of encoders + decoders, where it operates oppositely to that of a CNN, where it takes an image, compresses it, and then produces a somewhat replica of it or it predicts a set of outcomes (such as for sequence to sequence prediction LSTM modeling). The replication only embodies the critical core of the original image - in other words it is low dimensionality but high information and is also called the 'learning representation' of the model. AEs are primarily used for decompressing images, de-noising, and image generation, and recommendation (or prediction) systems for time series or regression based data. There are also VAEs (or Variational Auto-Encoders).[36]*

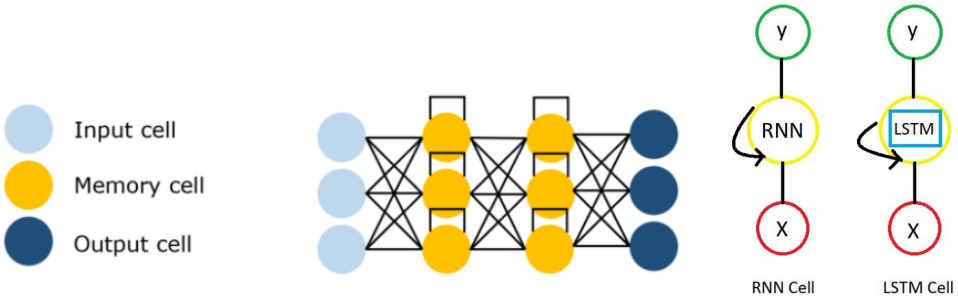


***Figure 90**: LSTMs (or Long-Short Term Memory) are RNNs (as shown in image at the left hand corner [38]) that solve the vanishing and exploding gradient problems found in other RNNs (except for Gated Recurrent Units or GRUs). LSTM cells in the hidden layers have unique gate configurations that decide what to store and what to let go with respect to the hidden inputs and outputs. Each layer is connected to a memory channel (c- channel) and a hidden state (h channel). LSTMs are most popularly used for applications that have time series based data because they are not affected by retaining information and learning from large volumes of time steps (like 1,000). [36]*

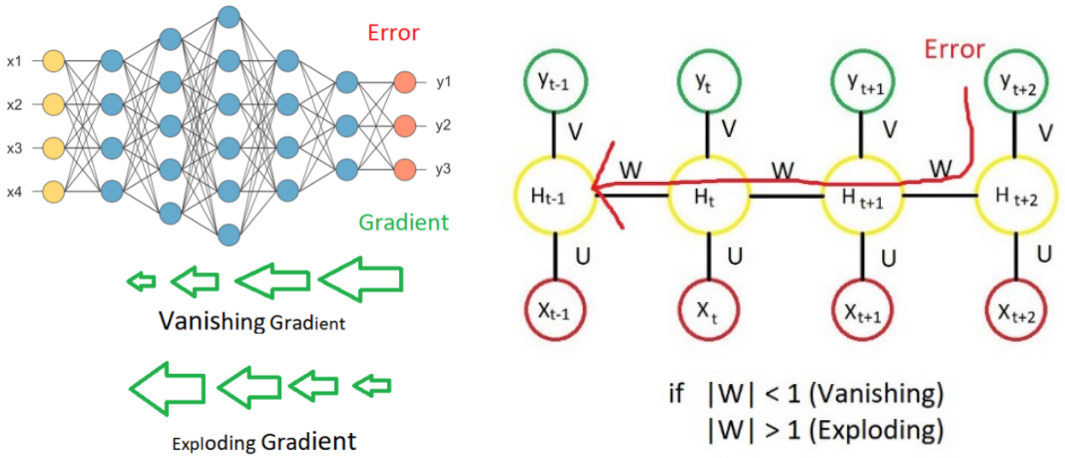


***Figure 91**: Spatial descriptions on vanishing and exploding gradients. [38]*

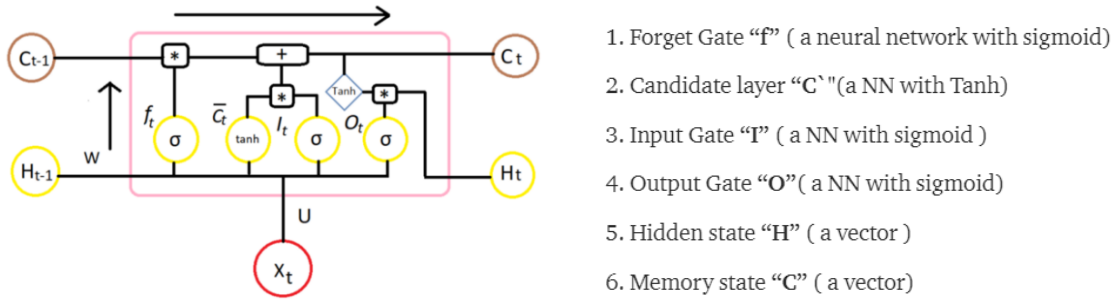


***Figure 92**: Composition of a LSTM cell and its key components (or gates) that control the flow of inputs and outputs. Note the memory channel (C channel) runs across the cell's gates. [38]*

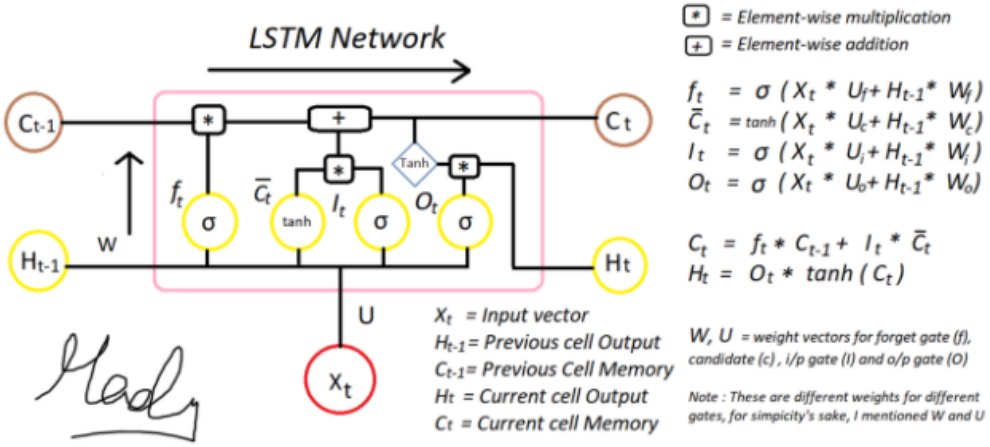


***Figure 93**: LSTM cell and equations for each gate and channel. W is the LSTM weights (for each gate) and U is the weights for the RNN part of the LSTM layer. The C channel is activated when setting statefulness and return_state = True, and the H channel (or the hidden states) is turned on - return_sequences = True, when stacked LSTM (more than one LSTM layer and are atop one another) is used. Also, when using stacked LSTM, the last layer, right before a Dense layer, should be set to False. [38]*

Once the data has been collected, typically, it will be normalized or scaled (depending on how the raw data is distributed) and then split into training and testing. The way LSTMs work is that it requires a 3D input, as can be graphically presented below. Thus, the data input must be defined (or reshaped from 2D to 3D) as follows: (number of samples, number of timesteps, number of features), where 'number of features' = input_dim (in the diagram below).

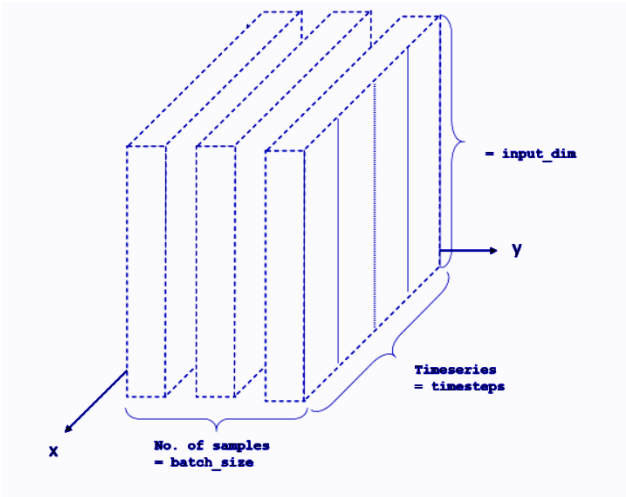


***Figure 94**: LSTM input 3D size spatially defined [18].*

Keep in mind that the number of samples does not equate to batch_size as shown in the diagram in Figure 94, but rather the overall number of samples defined can affect what the batch_size will be. Batch_size is defined in the model.fit line in the code and is dependent on the overall number of samples in both inputs and outputs. Thus, shaping the data affects both the number of samples and batch_size.

More on how the data shape plays a role in training the model is presented in the next section.

Chapter 5 will provide a more in depth look at the finalized version of the NN model and the thought processes that went on in establishing it.

3.3 Basic Setup in Training a NN

3.3.1 Training Sample of a NN with Colab

The lines of code shown below is the preliminary work that was done to provide proof of concept, and will be used to demonstrate all that has been learned in prior sections with respect to NN modeling in Keras/Tensorflow using Google Colab/Jupyter Notebooks as the open sourced library, backend engine, and IDE, respectively.

When using the Google Colab, where Jupyter Notebook is the default blank space for code, it is important to select Edit → Notebook Settings → and select either CPU or GPU as the source for all computations. The setting selected was GPU.

Keep in mind that since the snippets of code are long, only the critical parts are shown; and the code shown is actually the eight trial run in a continual series of trial runs.

The steps are as follows:

- List and import all libraries that are necessary for the study:

▾ import libraries

```
[ ] # %pip install tensorflow
    import tensorflow as tf
    import tensorflow_addons as tfa
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras import initializers
    from tensorflow.keras.layers import Dense, Activation, Flatten, Conv1D, Conv2D, SimpleRNN, GRU, LSTM, LSTMCell, MaxPooling2D, Dropout
    from tensorflow.keras import Model
```

```
[ ] from tensorflow_addons.metrics import RSquare, r_square
```

```
[ ] import sys
    from scipy.stats import randint
```

```
[ ] import numpy as np    #linear algebra
    from numpy import concatenate
    from numpy import split
    from numpy import array
```

Note the brackets on the left hand corner of each set of code. Each section is called a cell, and one must select that bracket to run each section of code.

```
from random import randint
from math import sqrt
import sklearn
from sklearn import preprocessing
from sklearn.utils import shuffle    #can't use this b/c data is correlated
from sklearn.model_selection import train_test_split  #to split data into 2 parts
from sklearn.model_selection import KFold   #used for cross-validation
from sklearn.preprocessing import StandardScaler  #for normalization
from sklearn.preprocessing import Normalizer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline  #data pipeline making
```

```
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics   #check the error and accuracy of model
```

```
import keras
# %pip install --upgrade keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import LSTM, Lambda
from keras.layers.wrappers import TimeDistributed
from keras.layers import RepeatVector
```

```
from keras.layers.core import Dense
from keras.layers import LeakyReLU
from keras.optimizers import Adam
from keras.layers.normalization import BatchNormalization
from keras.metrics import categorical_crossentropy, sparse_categorical_crossentropy
from keras.utils import to_categorical
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
from tensorflow.keras.regularizers import L1, L2
import itertools
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional_recurrent import ConvLSTM2D
```

```
import matplotlib.pyplot as plt   #from matplotlib import pyplot
import seaborn as sns #interactive graphing
import pandas as pd    #data processing, CSV file, data manipulation as in SQL
import pandas.util.testing as tm
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
import functools
```

- If curious, can check out the version and amount of GPUs that are in use:

```
[ ] !nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
```

```
[ ] physical_devices = tf.config.experimental.list_physical_devices('GPU')
    print("Num GPUs Available:", len(physical_devices))
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
    #tf.config.experimental.set_memory_growth(
        #physical_devices, True
    #)
```

```
Num GPUs Available: 1
```

- Next, is to upload the data. In this study, since Google Colab is used, the data is saved in a .csv format in a separate folder in google drive. (It is also important to note that if one is using Google Colab, one must have only folders in their drive (Google Colab's server is sensitive to files that are not in folders).

```
[ ] drive.mount('/content/drive', force_remount=True)

    # enter the foldername in your Drive where you have saved the unzipped
    # 'cs231n' folder containing the '.py', 'classifiers' and 'datasets'
    # folders.
    # e.g. 'MLTRACK/colab/2020/module1/cs231n/assignments/assignment1/cs231n/'
    FOLDERNAME = 'AEDATA/'

    assert FOLDERNAME is not None, "[!] Enter the foldername."

    %cd drive/My\ Drive
    %cp -r $FOLDERNAME ../../
    %cd ../../
    #%cd cs231n/datasets/
    #!bash get_datasets.sh
    #%cd ../../
```

```
Mounted at /content/drive
/content/drive/My Drive
cp: cannot open 'AEDATA/TestNNData3.gsheet' for reading: Operation not supported
/content
```

```
[ ] path="AEDATA/TestNNData4.csv"
    df=pd.read_csv(path)
```

- Next, are a series of codes that serve to verify the data and create statistical inferences (only some are shown). Data is collected from the orbital simulator: GMAT. Panda library is used. :

```
[ ] print(df)
```

```
           U.X           U.Y          U.Z          P.X
0 -61893.02247  17217.967960 -35733.95319 -61893.66006
1 -64475.18781  15505.769780 -37224.76704 -64476.32437
2 -66709.79605  13709.790730 -38514.91871 -66711.50941
3 -68614.25716  11845.816550 -39614.45984 -68616.63228
4 -70202.40493   9927.124463 -40531.37739 -70205.53373
5 -71485.12635   7965.162276 -41271.95694 -71489.10759
6 -72470.81249   5970.044388 -41841.04310 -72475.75188
7 -73165.68030   3950.928257 -42242.22522 -73171.69061
8 -73573.99671   1916.312596 -42477.96681 -73581.19809
9 -73698.76182      0.000482 -42549.99997 -73707.20062
```

```
[ ] df.head()
```

	U.X	U.Y	U.Z	P.X
0	-61893.02247	17217.967960	-35733.95319	-61893.66006
1	-64475.18781	15505.769780	-37224.76704	-64476.32437
2	-66709.79605	13709.790730	-38514.91871	-66711.50941
3	-68614.25716	11845.816550	-39614.45984	-68616.63228
4	-70202.40493	9927.124463	-40531.37739	-70205.53373

```
[ ] df.shape
```

```
(10, 4)
```

U.X = Unperturbed input in the x direction (position of spacecraft without solar perturbations)
U.Y = Unperturbed input in the y direction (position of spacecraft without solar perturbations)
U.Z = Unperturbed input in the z direction (position of spacecraft without solar perturbations)
P.X = Perturbed output in the x direction (position of spacecraft with solar perturbations)

(In later runs planning to use three outputs instead of just one, and maybe eventually six where both position and velocities are considered in every run/iteration.)

- Next, make sure there are no anomalies in the data:

```
[ ]  df.isnull().sum()   #sanity check to make sure there are no nulls
```

```
U.X    0
U.Y    0
U.Z    0
P.X    0
dtype: int64
```

- Reshaping the data - to clarify what is input, X, and output, Y:

```
[ ]  X=dataset[:,0:3]  # Out of the matrix called dataset you are capturing all the rows (is the first element -colon only).
     #[from n row : to m row, from n column : to m column]
     X
```

```
array([[-6.18930225e+04,  1.72179680e+04, -3.57339532e+04],
       [-6.44751878e+04,  1.55057698e+04, -3.72247670e+04],
       [-6.67097961e+04,  1.37097907e+04, -3.85149187e+04],
       [-6.86142572e+04,  1.18458165e+04, -3.96144598e+04],
       [-7.02024049e+04,  9.92712446e+03, -4.05313774e+04],
       [-7.14851264e+04,  7.96516228e+03, -4.12719569e+04],
       [-7.24708125e+04,  5.97004439e+03, -4.18410431e+04],
       [-7.31656803e+04,  3.95092826e+03, -4.22422252e+04],
       [-7.35739967e+04,  1.91631260e+03, -4.24779668e+04],
       [-7.36987618e+04,  4.81501000e-04, -4.25500000e+04]])
```

```
[ ]  X.shape
```

```
(10, 3)
```

```
[ ]  Y=dataset[:,3].reshape(10,1)
     Y
```

```
array([[-61893.66006],
       [-64476.32437],
       [-66711.50941],
       [-68616.63228],
       [-70205.53373],
       [-71489.10759],
       [-72475.75188],
       [-73171.69061],
       [-73581.19809],
       [-73707.20062]])
```

- Next, is normalizing or scaling the data. Now usually most libraries (such as scikit-learn) conduct normalization or scaling on each feature (also called feature scaling), but due to the nature of the data, this was not an option; thus classical scaling/normalization tactic was used. (There is a distinct difference between scaling and normalization when cleaning the data. Scaling uses the min/max values to reduce the overall range of values between, for instance, 0 and 1 , while normalization changes the overall distribution of the data to a Gaussian bell curve. Either case, the title of this process was labeled as

normalization even though scaling was actually used. This was in part due to the ambiguity of either term used in mathematics and in data science and ML articles.) :

Normalize Data

```
[ ] #Input Normalization for whole of dataset instead of having X and Y separately scaled
    #scaler= MinMaxScaler(feature_range=(0,1)) #-1,1 ? b/c 0,1 is slower learning but is ok to use for small sample size
    #scaled_dataset=scaler.fit_transform(data1)  #didn't need to resize at all
    datasetmax, datasetmin = dataset.max(), dataset.min()
    dataset2 = (dataset - datasetmin)/(datasetmax - datasetmin)
    print("After normalization:")
    print(dataset2)
```

```
After normalization:
[[1.29932980e-01 1.00000000e+00 4.17631862e-01 1.29925968e-01]
 [1.01534184e-01 9.81169150e-01 4.01235809e-01 1.01521684e-01]
 [7.69578399e-02 9.61416874e-01 3.87046650e-01 7.69389963e-02]
 [5.60124720e-02 9.40916784e-01 3.74953836e-01 5.59863503e-02]
 [3.85459356e-02 9.19814903e-01 3.64869527e-01 3.85115249e-02]
 [2.44384949e-02 8.98237135e-01 3.56724592e-01 2.43947090e-02]
 [1.35978646e-02 8.76294719e-01 3.50465751e-01 1.35435409e-02]
 [5.95567023e-03 8.54088368e-01 3.46053528e-01 5.88956851e-03]
 [1.46498392e-03 8.31711554e-01 3.43460829e-01 1.38578275e-03]
 [9.28103861e-05 8.10635848e-01 3.42668605e-01 0.00000000e+00]]
```

```
[ ] scaled_X = dataset2[:,0:3]
    scaled_X
```

```
array([[1.29932980e-01, 1.00000000e+00, 4.17631862e-01],
       [1.01534184e-01, 9.81169150e-01, 4.01235809e-01],
       [7.69578399e-02, 9.61416874e-01, 3.87046650e-01],
       [5.60124720e-02, 9.40916784e-01, 3.74953836e-01],
       [3.85459356e-02, 9.19814903e-01, 3.64869527e-01],
       [2.44384949e-02, 8.98237135e-01, 3.56724592e-01],
       [1.35978646e-02, 8.76294719e-01, 3.50465751e-01],
       [5.95567023e-03, 8.54088368e-01, 3.46053528e-01],
       [1.46498392e-03, 8.31711554e-01, 3.43460829e-01],
       [9.28103861e-05, 8.10635848e-01, 3.42668605e-01]])
```

```
[ ] scaled_X.shape
```

```
(10, 3)
```

```
[ ] scaled_Y = dataset2[:,3].reshape(10,1)
    scaled_Y
```

```
array([[0.12992597],
       [0.10152168],
       [0.076939  ],
       [0.05598635],
       [0.03851152],
       [0.02439471],
       [0.01354354],
       [0.00588957],
       [0.00138578],
       [0.        ]])
```

- Next, is dividing the scaled data to training and test, and reshaping the data (from 2-D to 3-D) so that the LSTM model would accept it.

Reshape and resplit data for LSTM is 3D where number of batch samples and number of timesteps need to match.

```
[ ] #train to test size ratio 0.8:0.2 and will only work in 2D
    scaled_x_train, scaled_x_test, scaled_y_train, scaled_y_test = train_test_split(scaled_X, scaled_Y, test_size=0.2, random_state=0, shuffle = False)
```

Double-click (or enter) to edit

```
[ ] scaled_x_train   # the third time step for both scaled x and scaled y datasets were taken out for validation testing
```

```
array([[0.12993298, 1.        , 0.41763186],
       [0.10153418, 0.98116915, 0.40123581],
       [0.07695784, 0.96141687, 0.38704665],
       [0.05601247, 0.94091678, 0.37495384],
       [0.03854594, 0.9198149 , 0.36486953],
       [0.02443849, 0.89823713, 0.35672459],
       [0.01359786, 0.87629472, 0.35046575],
       [0.00595567, 0.85408837, 0.34605353]])
```

```
[ ] scaled_x_train.shape
```

```
(8, 3)
```

```
[ ] scaled_y_train
```

```
array([[0.12992597],
       [0.10152168],
       [0.076939  ],
       [0.05598635],
       [0.03851152],
       [0.02439471],
       [0.01354354],
       [0.00588957]])
```

```
[ ] scaled_y_train.shape
```

```
(8, 1)
```

```
[ ] scaled_x_test
```

```
array([[1.46498392e-03, 8.31711554e-01, 3.43460829e-01],
       [9.28103861e-05, 8.10635848e-01, 3.42668605e-01]])
```

```
[ ] scaled_x_test.shape
```

```
(2, 3)
```

```
[ ] scaled_y_test
```

```
array([[0.00138578],
       [0.        ]])
```

```
[ ] scaled_y_test.shape
```

```
(2, 1)
```

```
[ ] scaled_X_train=np.resize(scaled_x_train,(8,1,3))  #Only resizing the X scaled training parts
    scaled_X_train
```

```
array([[[0.12993298, 1.        , 0.41763186]],

       [[0.10153418, 0.98116915, 0.40123581]],

       [[0.07695784, 0.96141687, 0.38704665]],

       [[0.05601247, 0.94091678, 0.37495384]],

       [[0.03854594, 0.9198149 , 0.36486953]],

       [[0.02443849, 0.89823713, 0.35672459]],

       [[0.01359786, 0.87629472, 0.35046575]],

       [[0.00595567, 0.85408837, 0.34605353]]])
```

```
[ ] scaled_Y_train=np.resize(scaled_y_train,(8,1,1))  #Only resizing the X scaled training parts
    scaled_Y_train
```

```
array([[[0.12992597]],

       [[0.10152168]],

       [[0.076939  ]],

       [[0.05598635]],

       [[0.03851152]],

       [[0.02439471]],

       [[0.01354354]],

       [[0.00588957]]])
```

```
[ ] scaled_Y_train.shape
```

```
(8, 1, 1)
```

Must resize the test sample, as well, for both X and Y, as well

```
[ ]  scaled_X_test=np.resize(scaled_x_test,(2,1,3))
     scaled_X_test
```

```
array([[[1.46498392e-03, 8.31711554e-01, 3.43460829e-01]],

       [[9.28103861e-05, 8.10635848e-01, 3.42668605e-01]]])
```

```
[ ]  scaled_Y_test=np.resize(scaled_y_test,(2,1,1))
     scaled_Y_test
```

```
array([[[0.00138578]],

       [[0.        ]]])
```

- Next, is creating the sequential API of the NN model (one can also use functional API, it is a matter of preference and if one needs to hard code more - which functional is more flexible towards):

▾ Creating Model

```
#lstm
#def create_model():
model=tf.keras.models.Sequential([
                            #there's a diff btwn LSTMCell() and LSTM(), the latter uses CuDNN whilst the other doesn't.
              #1st layer is the input layer, which shape is defined in the next layer
                keras.layers.LSTM(1, input_shape=(1,3), activation='relu', bias_initializer='zeros', kernel_initializer='glorot_uniform', stateful=False, return_sequences= True, r
                #keras.layers.BatchNormalization(trainable=True), #(training=True) NEED TO USE FUNCTIONAL STRATA
                #keras.layers.Dense(1, activation='relu'),
                #keras.layers.Dropout(0.2),  #regularizer
                keras.layers.Dense(1) # activation='softmax')  #Output, softmax is loss probability between estimated and truth label btwn layers
                #default weight initializer in Keras is Xavier or glorot_uniform and bias initialized to zeros, and cmd init is same as kernel_initializer
])

#model.compile(Adam(lr=.0001),loss='mean_squared_error',metrics=['accuracy'])
#return model
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU kernel as fallback when running on GPU
```

```
[ ]  model.compile(Adam(lr=.01),loss='mean_squared_error', metrics=['mse']) #r2_score(scaled_Y_train, model.predict(scaled_X_train)
```

```
[ ]  #Creates a basic model instance
     #model= create_model()
```

```
[ ]  #Display's model architecture
     model.summary()
```

Note that all the green texts are comments. The value for metrics in the model.compile line can be changed to: metrics = [‘accuracy’]. (It was not initially used, but will later be in Chapter 5). The image below is an extension of the model that is created, this is because there are a lot of attributes for the LSTM cell.

▾ Creating Model

```
DNN whilst the other doesn't.

eros', kernel_initializer='glorot_uniform', stateful=False, return_sequences= True, return_state=False), #kernel_regularizer=L2(0.001)), #1st hidden layer, so this is 2nd layer
 FUNCTIONAL STRATA


ility between estimated and truth label btwn layers
ialized to zeros, and cmd init is same as kernel_initializer
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU kernel as fallback when running on GPU
```

```
[ ] model.compile(Adam(lr=.01),loss='mean_squared_error', metrics=['mse']) #r2_score(scaled_Y_train, model.predict(scaled_X_train)
```

```
[ ] #Creates a basic model instance
    #model= create_model()
```

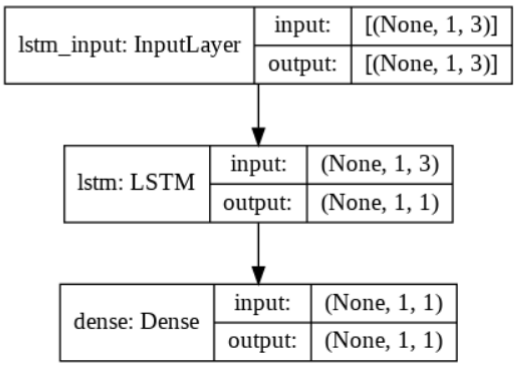
```
[ ] #Display's model architecture
    model.summary()
```

(Note that for the first layer, return_sequences = True, which shouldn't be the case if using a non-stacked LSTM model. Since, only one LSTM layer is used and a dense layer right after, return_sequences = False.)

Note the type of optimizer (Adam) and learning rate (lr), as well as the type of loss function for the output (mse = mean squared error) selected.

```
[ ] Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 1)              20
_________________________________________________________________
dense (Dense)                (None, 1, 1)              2
=================================================================
Total params: 22
Trainable params: 22
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] #can visualize network topology
    keras.utils.plot_model(model, show_shapes=True, show_layer_names=True)
```



Note that parameters (total number of weights + biases) are calculated under model.summary.

Note that when using Keras, biases are defined as part of the weights and when using the get_weights[] command, biases are the last array in each layer. LSTM layers will generally yield three arrays when the get_weights[] command is issued: weights per gate (W), recurrent weights per gate (U) and biases per gate (b).

This model has only an input layer, hidden layer (LSTM) and an output (dense) layer. The number of parameters calculated is via the LSTM cell based formula: 4*[(#of features + 1)*#of units) + (#of units)^2], or can also be defined as follows: 4*[(n x n) + (n x m) + n]. Note that the number 4 in the formulas constitutes as gates for the LSTM cell (as explained in section 3.2.3 LSTMs have four gates. Units designates the number of nodes or cells that the operator defines for a particular LSTM layer, and in this case, it is three. The ‘n x m’ is the matrix of rows x columns or timesteps x features of the input shape defined in the LSTM layer, which in this case is ‘ 1 x 3’.

```
history=model.fit(scaled_X_train,scaled_Y_train, batch_size=8, epochs=300, validation_split=0.2, shuffle=False, verbose=2)
```

```
Epoch 105/300
1/1 - 0s - loss: 0.0010 - mse: 0.0010 - val_loss: 0.0031 - val_mse: 0.0031
Epoch 106/300
1/1 - 0s - loss: 0.0010 - mse: 0.0010 - val_loss: 0.0031 - val_mse: 0.0031
Epoch 107/300
1/1 - 0s - loss: 9.9879e-04 - mse: 9.9879e-04 - val_loss: 0.0030 - val_mse: 0.0030
Epoch 108/300
1/1 - 0s - loss: 9.8883e-04 - mse: 9.8883e-04 - val_loss: 0.0030 - val_mse: 0.0030
```

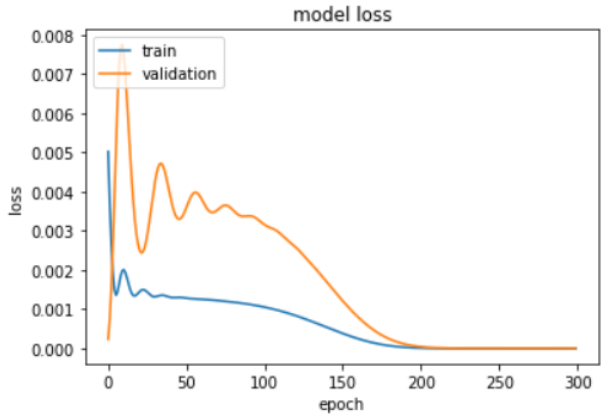
Note this is where the actual training is done (cmd line: model.fit), and the batch size (# of samples), epoch and validation split is specified.

```
[ ] history.history.keys()
```

```
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```

Can use the above line of code to view what the attributes are to the variable ‘history’; which will be later used for the plots:

```
[ ] plt.plot(history.history['mse'])
    plt.plot(history.history['val_mse'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    # "Loss"
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```




```
] model.evaluate(scaled_X_test,scaled_Y_test, batch_size=1, verbose=2 )

2/2 - 0s - loss: 1.3616e-06 - mse: 1.3616e-06
[1.361590193482698e-06, 1.361590193482698e-06]
```

Note that the validation curve is higher than that of the training curve (in the loss plot) so the NN model is learning. The last line tests the NN model's level of loss and accuracy by using the scaled test samples created earlier. Since $10^{-6} > 10^{-10}$ the model is not yet ready to use. Remember that the expected evaluation metric for accuracy was an error (or loss) with a minimum of 10^{-10}. (The accuracy plot is not shown, but will be elaborated on in Chapter 5.)

Chapter 4 - GMAT and Data Collection

4.1 GMAT Runs and Setup

4.1.1 Background

Chapter 4 discusses the setup of, and data collection from, GMAT. The Cassini mission example was primarily used to determine the time ranges for the overall propagation towards Saturn. The overall GMAT setup is divided into three different categories of analysis: changing the initial position and velocity vectors, perturbed vs unperturbed data (within a range of 157 days), and the different delta v requirements after a two year mark (as the spacecraft travels from Venus to Saturn). The spacecraft is modeled at a certain distance, in transit, before it enters the sphere of influence of Venus and does a swingby about the sun as it is hyperbolically propelled towards Saturn. This particular type of trajectory is unique to analyze the impact of the solar perturbations on the correctional burn (TOI) at the two year mark, especially since the swingby will be where the sun's perturbations are at its strongest.

GMAT is an orbital simulator that allows an operator to give certain initial inputs, either via Keplerian elements or position and velocity vectors. The latter option was selected since the former option would require timely efforts in hand calculations to determine energy (velocity) and phasing requirements within different reference frames. The initial position and velocity vector coordinates are found via a combination of JPL's Horizons ephemeris data and Lambert's problem solver in Matlab. Since Lambert's problem is reference frame agnostic, it is important to configure the ephemeris data, in JPL's Horizons website, within a sun centered reference frame. The Horizons web interface has six settings: ephemeris type, target body, coordinate origin, time span, table settings and display/output.

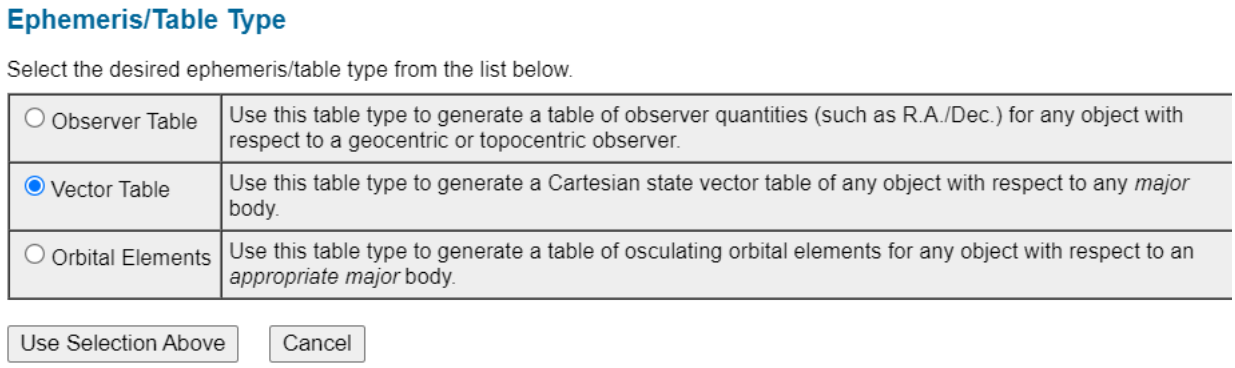


***Figure 95**: Ephemeris settings for Venus to Saturn Transfer on JPL's Horizons web interface.*

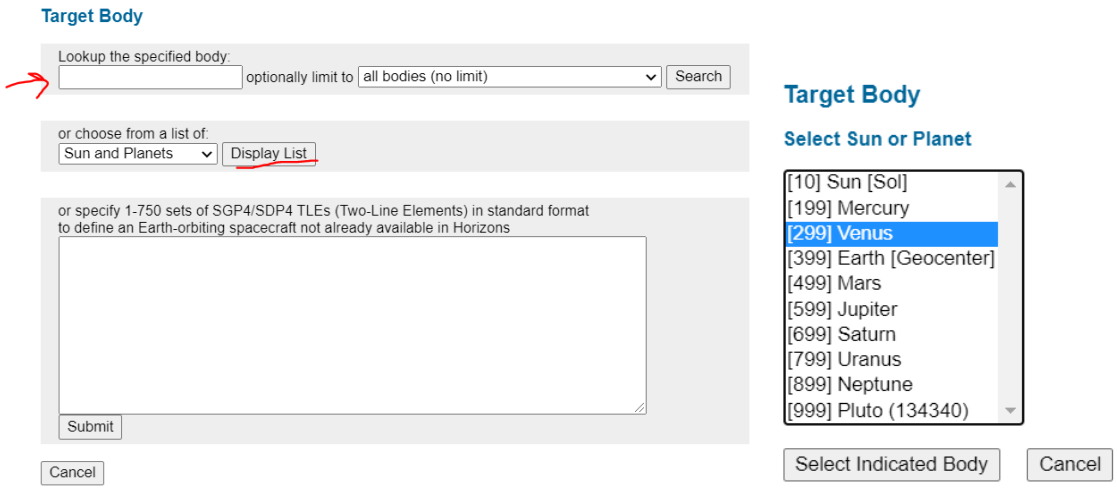


Figure 96*: Target body settings forVenus to Saturn transfer on JPL's Horizons web interface. If the number of the designated body is unknown, please select the 'display list' option, underlined in red, and it will take you to the target body lists within the category, 'Sun and Planets'. Notice that Venus has a designated number of 299 and Saturn is 699. Then click 'Select indicated Body'.*

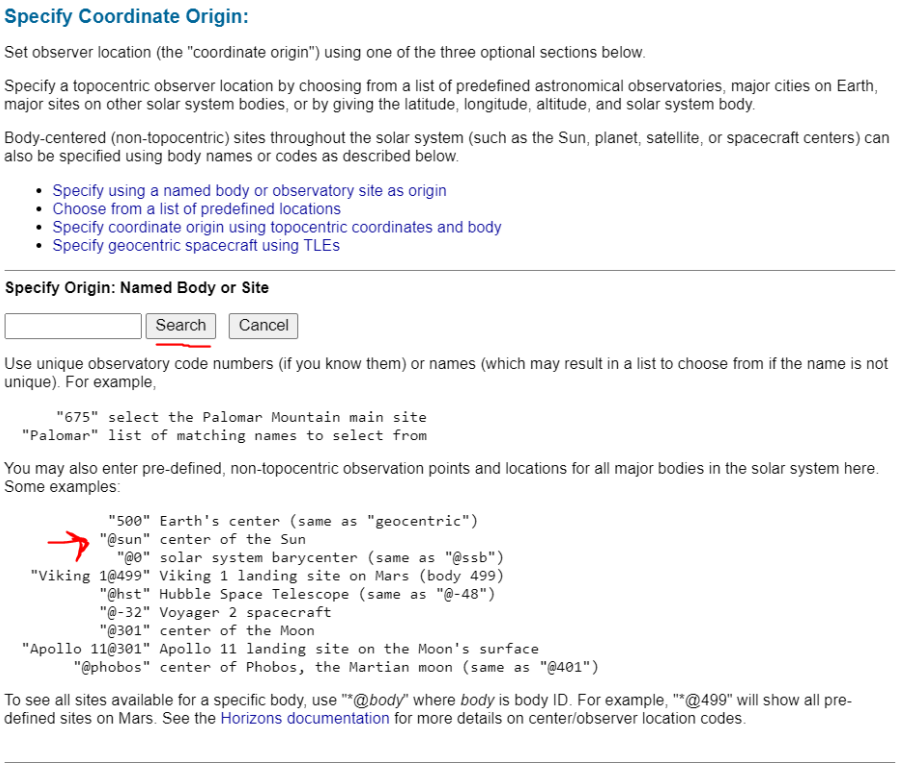


Figure 97*: The first category in specifying the coordinate origin via JPL's Horizons web interface. The other categories below it are : choosing from a list of predefined sites on Earth, specifying origin*

coordinates via latitude and longitude, or specifying global reference frame via an earth orbiting spacecraft. Fortunately, the first category, shown, is all that is necessary. Typing '@sun' in the white box next to the option 'Search', underlined in red, automatically the system understands that a sun reference frame is expected for the data collected. For instance, this project required a sun centered reference frame so Sun (body center)[500@10] was what the system yielded after typing '@sun'.

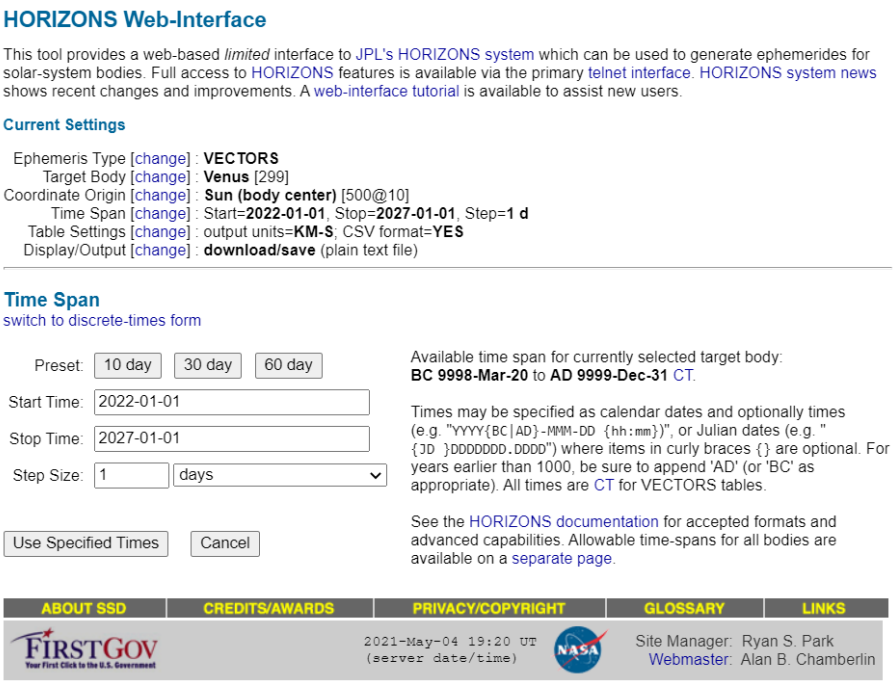


***Figure 98**: Time span settings on JPL's Horizons web interface. The start time was selected at random and stop time was based on the 5 year span that the Cassini mission had between Venus and Saturn transfer. Step size is kept for a day.*

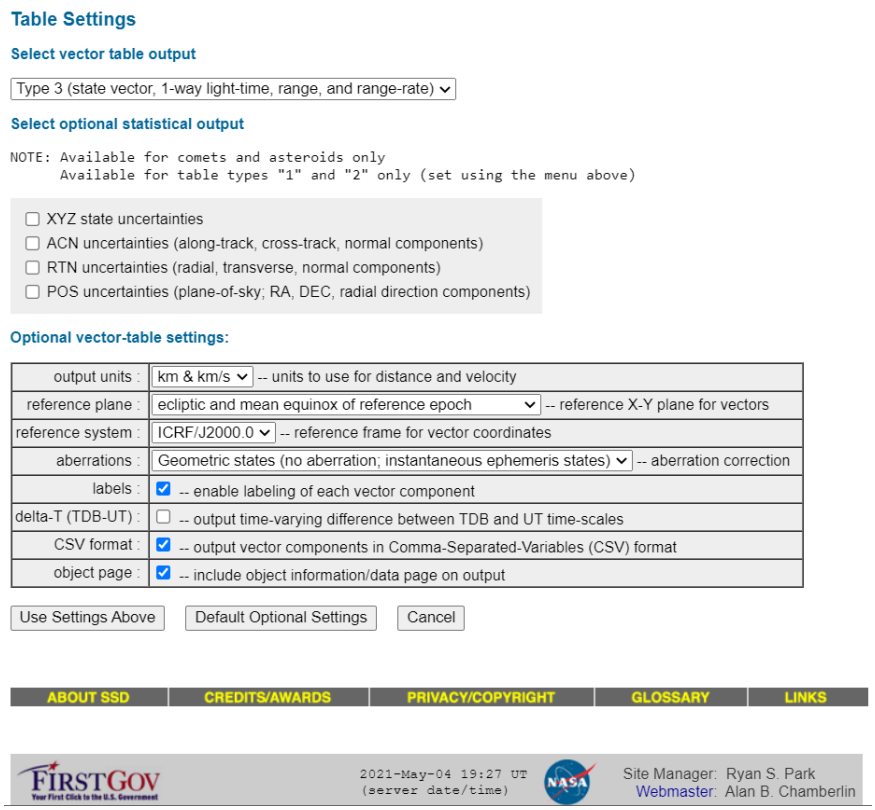


Figure 99: *The table settings configured on JPL's Horizons web interface.*

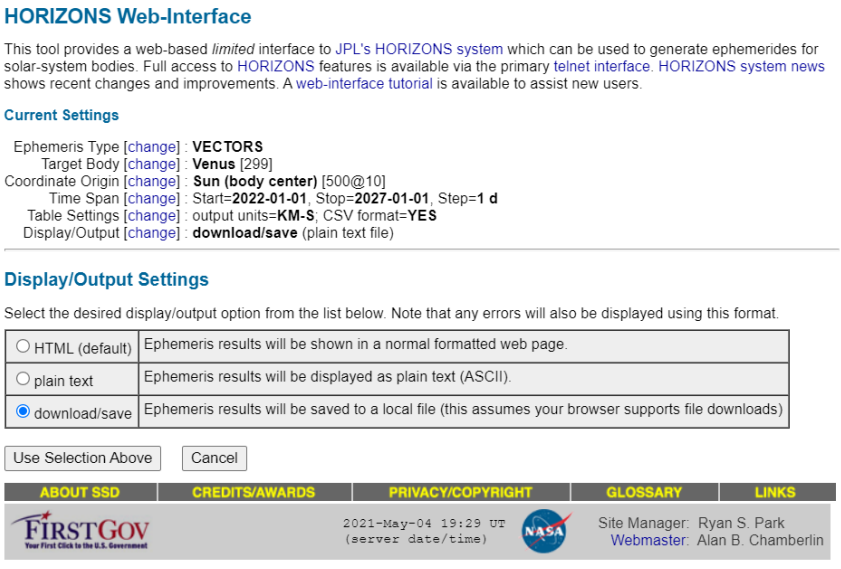


Figure 100: *The display/output settings configured on JPL's Horizons web interface.*

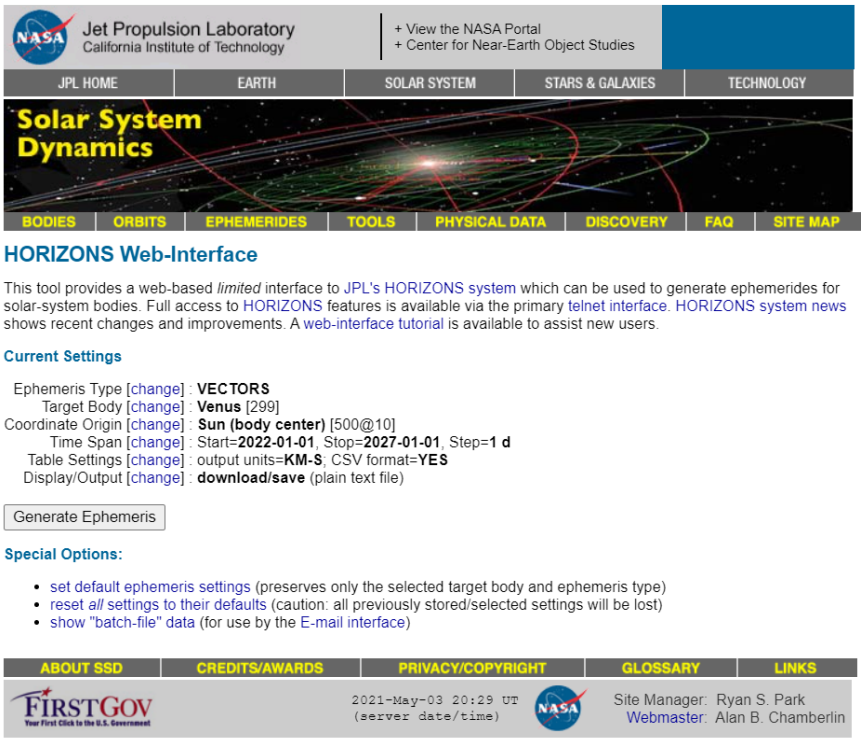


***Figure 101**: Final configuration of JPL's Horizons settings before selecting the option to 'Generate Ephemeris'. Since, the display/output and table settings were configured to be a download and csv format, then by selecting the 'Generate Ephemeris' option, a download will appear that is in csv format.*

Note that JPL's Horizons generates ephemeris data one planet at a time.

Once this data has been retrieved, the initial position vector (at Venus on Jan 01, 2022) and the final position vector (at Saturn on Jan 01, 2027) are used as inputs in the main Lambert's problem code below (note that the code below is calling the Lambert's problem function, and that the function itself has eight code files attached to it. This information can be viewed in the Appendix section of this report.):

```
1 -   clc, clear all, close all;
2
3     %normalization constants used in Lambert solver
4 -   TU = 5.0226757e6;
5 -   DU = 1.4959965e8;   %position 1 and 2 are divided by it
6 -   VU = 29.784582;      %velocities 1 and 2 are divided by it
7 -   GU = 1.3271544e11;    % u = GM, is divided by it
8
9 -   TOFinDays = 5 * 365;  %subject to change; right now it's 5 years cuz the Cassini mission
10                          %for Flyby 2 from Venus to Saturn took the same
11                          %amount of time.
12                          %June 24 1999 to July 1 2004
13
14 -  SecondsInDay = 60*60*24;
15 -  Mu = 1.9885e30 * 6.67*10^-20; %mass of sun (M_s) * gravitational constant (G)
16
17 -  TOF = TOFinDays * SecondsInDay;  %units in (total) seconds
18
19    %Introduce Offsets for propogator, so that s/c isn't at the center of mass
20    %of planet
21 -  VenusOffset = [6500, 0, 0];  %(R + alt = r) where R = radius of Venus = 6051.84 +/- 0.01 km
22 -  SaturnOffset = [120000, 0, 0];          %(R + alt = r) where R = radisu of Saturn = 60268 +/- 4 km
23
24    %Ephimeredes data below retrieved from JPL
25    % Position of Venus on Jan 1 2022 at 00:00:00.000
26 -  Pos1 = [-1.015245439803183E+07,  1.071173286419414E+08,  2.056027777301520E+06] + VenusOffset;
27    % Position of Saturn on Jan 1 2027 00:00:00.000
28 -  Pos2 = [1.364440141772364E+09, 3.390839717138447E+08,  -6.021867286262363E+07] + SaturnOffset;
29
30    %Lambert code that will give you the velocities at position 1 and position
31    %2 respectively ;  remember thst the Lambert Solver is focused on
32    %interplanetary travel.
33 -  temp = Lambert( Pos1, Pos2, TOF, 'Mu', Mu);
34
35    %temp = temp * VU ; b/c Lambert values can either be normalized or
36    %non-normalized. Depending on what values were inputted as the arguments of
37    %the Lambert function.
39    %Magnitudes of the pos1 and pos2 vectors:
40 -  mag_pos1 = sqrt(sum(Pos1.^2));
41 -  mag_pos2 = sqrt(sum(Pos2.^2));
```

***Figure 102**: Calling Lambert's problem function in Matlab. Line 33 is the function itself being called. Note that the required variables are Pos1 (initial position or start position vector - which is at some random offset from the center of Venus), Pos 2 (final position or end position vector - which is at some random offset from the center of Saturn), TOF (time of flight or the total duration of the flight between the two positions (here 5 years was selected due to the time span, between Venus and Saturn, from the Cassini mission), and Mu (which is, $\mu = GM$, or the gravitational acceleration of a body (G) multiplied by its mass (M)). The output of the solver are the velocities at position 1 (Pos 1) and position 2 (Pos 2), in which the Pos 1 will be an input to GMAT's initial spacecraft conditions.*

Lambert's problem is reference frame agnostic and all it cares about are: the two positions and duration of flight (TOF). Thus, it is important to remain consistent with the coordinate system,

for instance: sun reference frame, when generating data from JPL's Horizons, and inputting into Matlab and GMAT.

4.1.2 GMAT Setup for Sun Centered Elliptical Transfer (to and from Saturn)

This section (and the upcoming ones within chapter 4) is visually instructed to guide a beginner to navigate through GMAT. Thus, modifications and explanations for each particular setup and how it affects results will all be shown and discussed.

It is critical to note that it is advisable to not use other applications, nor to even click on anything in GMAT, when a simulation is running in GMAT. Also, F2 key will allow the operator to rename any file; which is handy if the right-click menu does not have that option.

Upon starting GMAT, there are three main tabs : Resources, Outputs and Mission. This section will delve into the options selected, per tab, for the complete transfer (to and from Saturn). This transfer will, also, have an initial back propagation (of 10 days) configured, due to initial expectation that the spacecraft is modeled mid transit as it approaches Venus, rather than starting at a parked orbit about Venus.

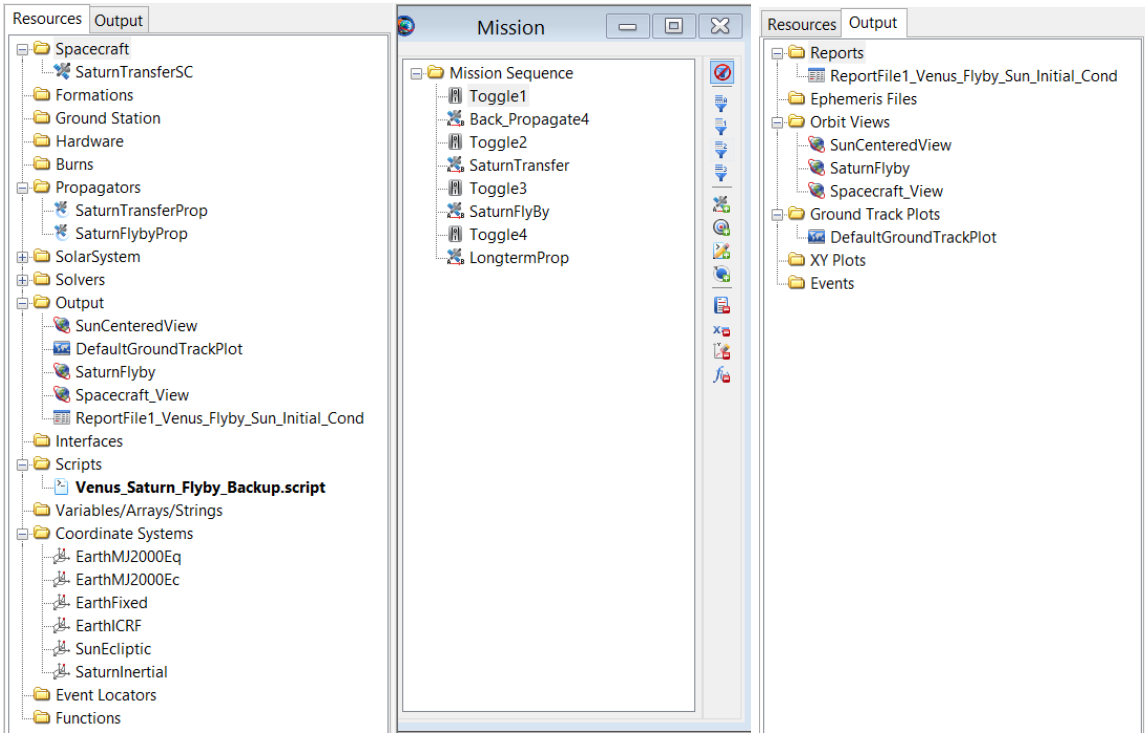


***Figure 103**: Resources, Output and Mission tabs. Resources is where one defines the spacecraft*

trajectory, burns (and even type of burns), fuel, planetary bodies, orbital views during propagation, propagators, coordinate systems, solvers...etc. 'Output' tab has the 'Outputs' defined under Resources and allows the operator to view (after a mission run is complete) the report file(s) and the orbital views, as well as ephemeris data (uploaded files from an accredited planetary database). Right clicking on outputs allows an operator to select what type of output to add.

There were a total of two coordinate systems and two propagators for the hyperbolic trajectory between Venus and Saturn, as well as, the full elliptical trajectory back to Venus.

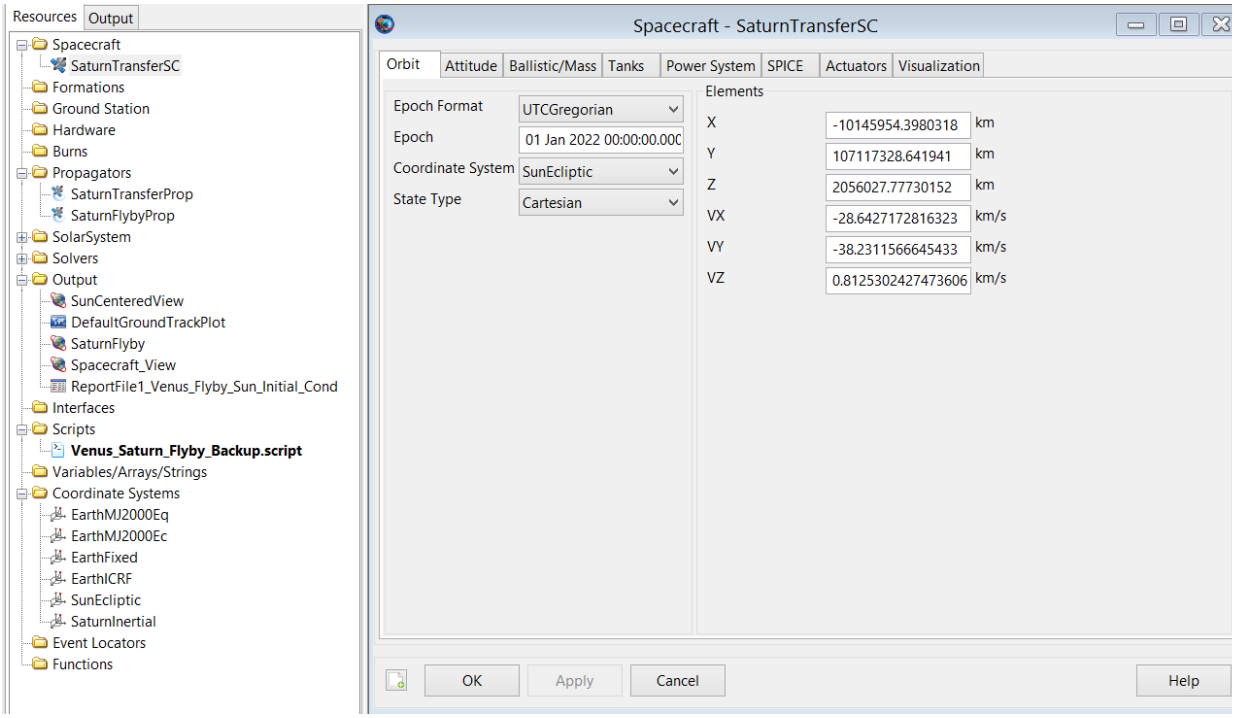


***Figure 104**: Spacecraft settings. Note the name is modifiable (just press F2 or right click). UTC Gregorian time is selected to make time input easier. Coordinate system is defined as sun centered and the cartesian coordinates have been used. Note that X, Y, and Z are the initial POS 1 vector defined via Matlab and VX, VY, and VZ are the velocity components of POS 1 (one of the outputs of Lambert's solver).*

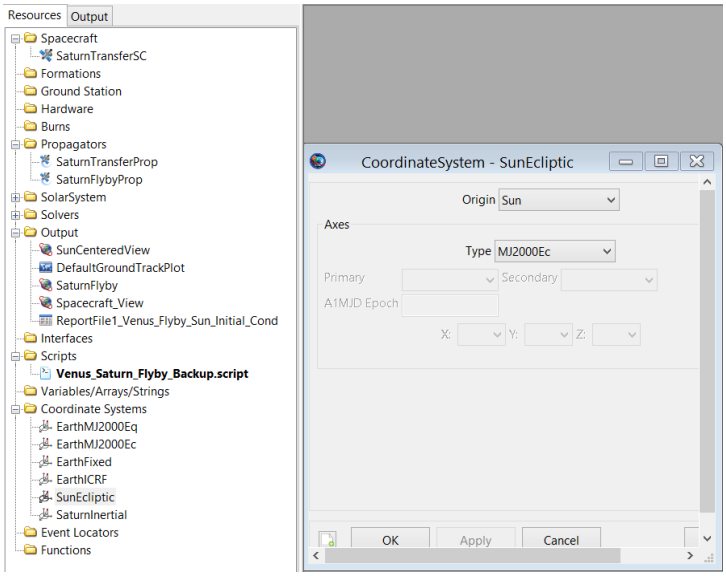


***Figure 105**: Sun ecliptic coordinate system (used during initialization of spacecraft) defined in GMAT.*

The ecliptic plane is the central plane of an orbit, while the equatorial plane is the plane at the center of any space body (such as a planet, sun, asteroid..etc.). Every space body tilts and wobbles along its central axis. For this project, the ecliptic plane is defined with respect to the sun's orbit. Thus, the trajectory in a sun centered view is unaffected by a mass bodies' movements and will have a constant reference frame.

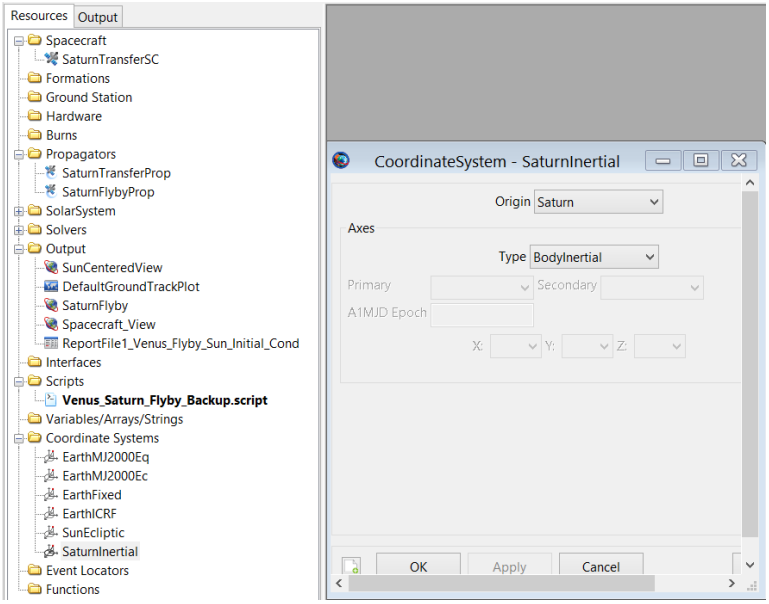


***Figure 106**: The second coordinate system is with respect to the central body of Saturn. Note that there*

is a difference between body fixed (the central reference frame remains fixed as the body rotates) and body inertial (the reference frame rotates with the body).Understand that these reference frames affect the orbital views, so the type of orbital view one needs will play a part in what type of coordinate system one selects.

The propagators are defined once the spacecraft's parameters for its initial position and velocity, as well as the coordinate systems have been defined. Note that the amount of propagators is determined by the number of TOIs (regardless if the burns done are free or not). There will be a total of two distinct propagators. The first propagator is the hyperbolic transfer between Venus and Saturn, while the second propagator is a Saturn flyby. Since Saturn's potential field changes the spacecraft's initial trajectory to a leading flyby about itself, another propagator with its respective coordinate system must be created.

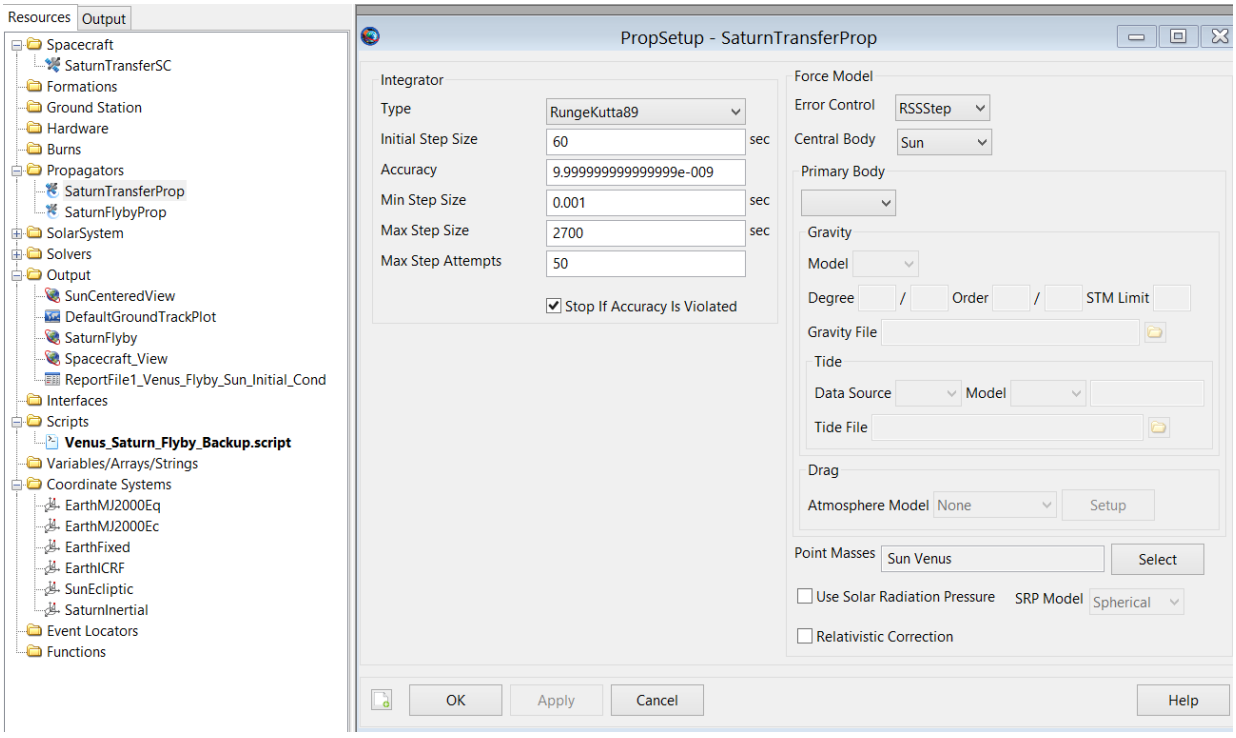


Figure 107*: First propagator : Venus to Saturn hyperbolic transfer. Note that RungeKutta 89 is defined as well as its corresponding accuracy (which is usually between* 10^{-9} *and* 10^{-12} *). The more accurate the propagator is, the more computational power it takes to complete a full run. Error control is kept at default but the central body was changed to the Sun, and the point masses (critical potential influence(s)) are Sun and Venus.*

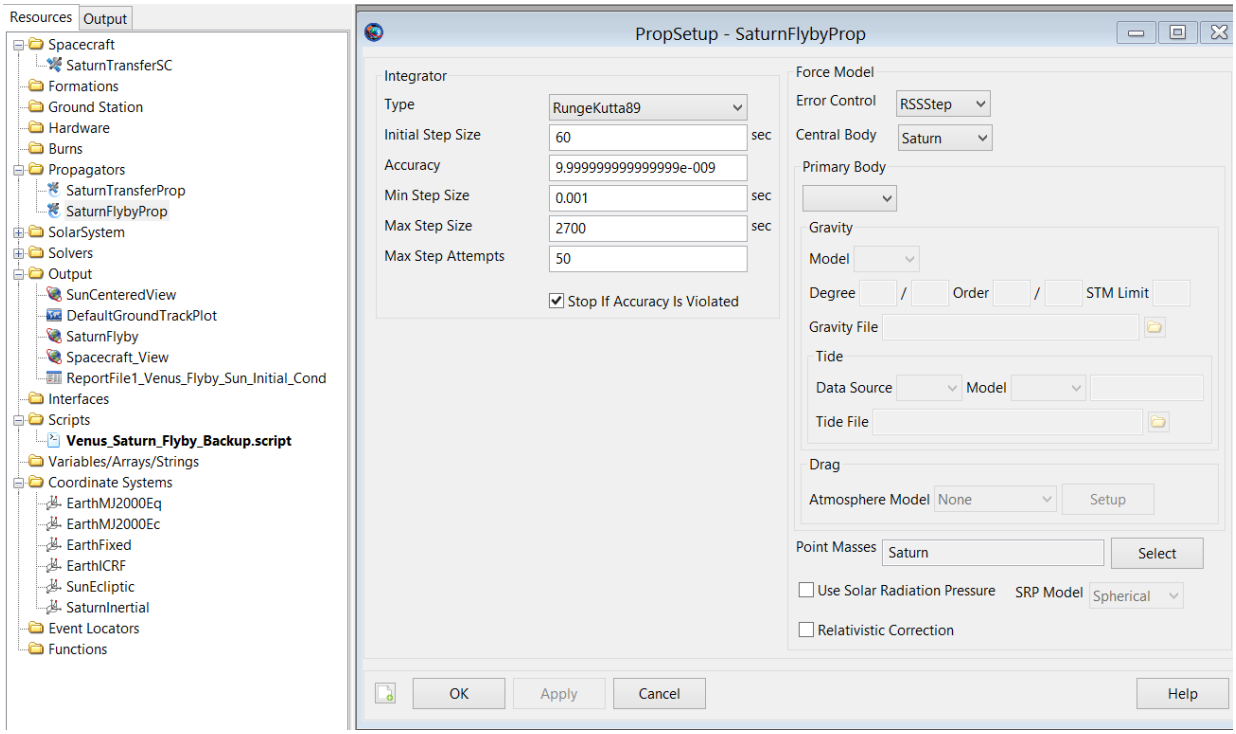


***Figure 108**: Second propagator: Saturn flyby. Note that only Saturn was selected as the point mass.*

Note that 'SolarSystem' (contains data for planets, Sun and its bary center) and 'Solvers' were not used for this analysis. The latter will be used when calculating the delta_v requirement after the two year mark and will be used for unperturbed trajectories (since without it will cause the spacecraft to continuously loop in a normal trajectory after a stopping point once it leaves Venus' sphere of influence).

It is critical to note that the orbital views will not show the complete trajectory if the number of points it is expected to plot is less than what the trajectory requires. A rule of thumb is to increase the default value of 20,000 (in the max number of data points to plot, shown below) to 150,000. Also, the console at the very bottom of the user interface will indicate any potential errors it encounters while the simulation is running. Thus, once the simulation is done, the operator can scroll up or down to check the receipts of the operation.

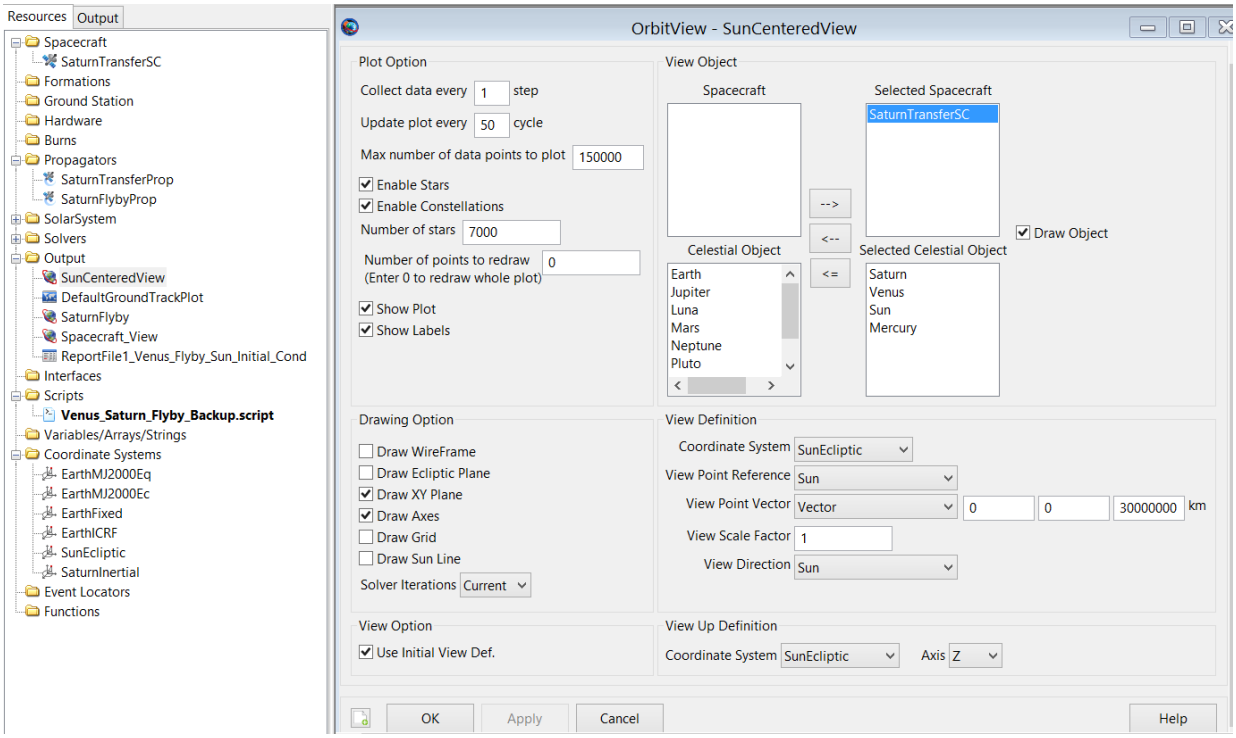


***Figure 109**: Sun centered Orbital view configured in GMAT. The critical considerations are: the max number of data points (if this is not correctly set up there will be an error message indicating there are not enough data points and no trajectory will be plotted), what celestial objects to view, and view/up definitions. It is ideal to define a view point vector so that you do not end up zooming out of the center of the celestial object for quite some time.*

The 'DefaultGroundTrackPlot' is only necessary if conducting closed orbits about a celestial body (preferably a planet). Thus, it is not discussed nor shown for any of the trail runs within the scope of this project.

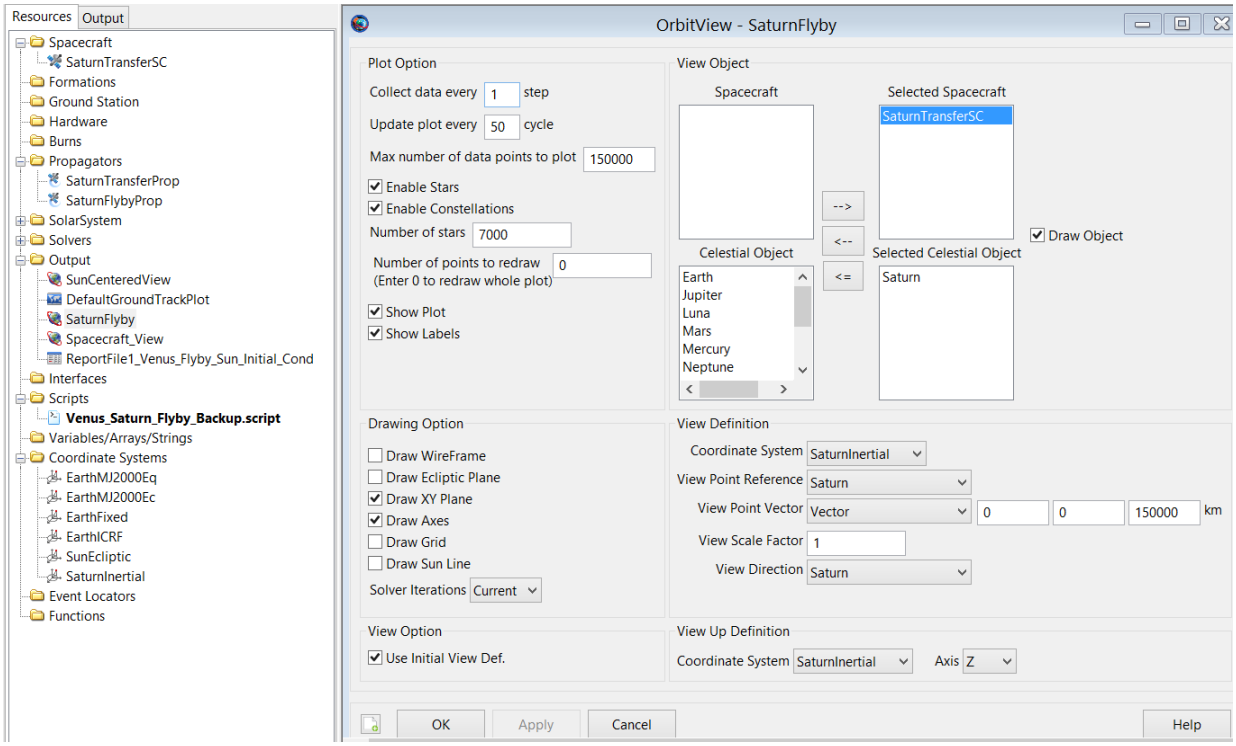


***Figure 110**: Saturn flyby orbital view configuration in GMAT.*

Sun centered orbital view is for the hyperbolic swing by trajectory between Venus and Saturn. Saturn flyby orbital view is the leading edge flyby with respect to Saturn. The next orbital view is the most critical and is with respect to the spacecraft itself. This orbital view allows the operator to see the whole mission regardless of coordinate system and trajectory type.

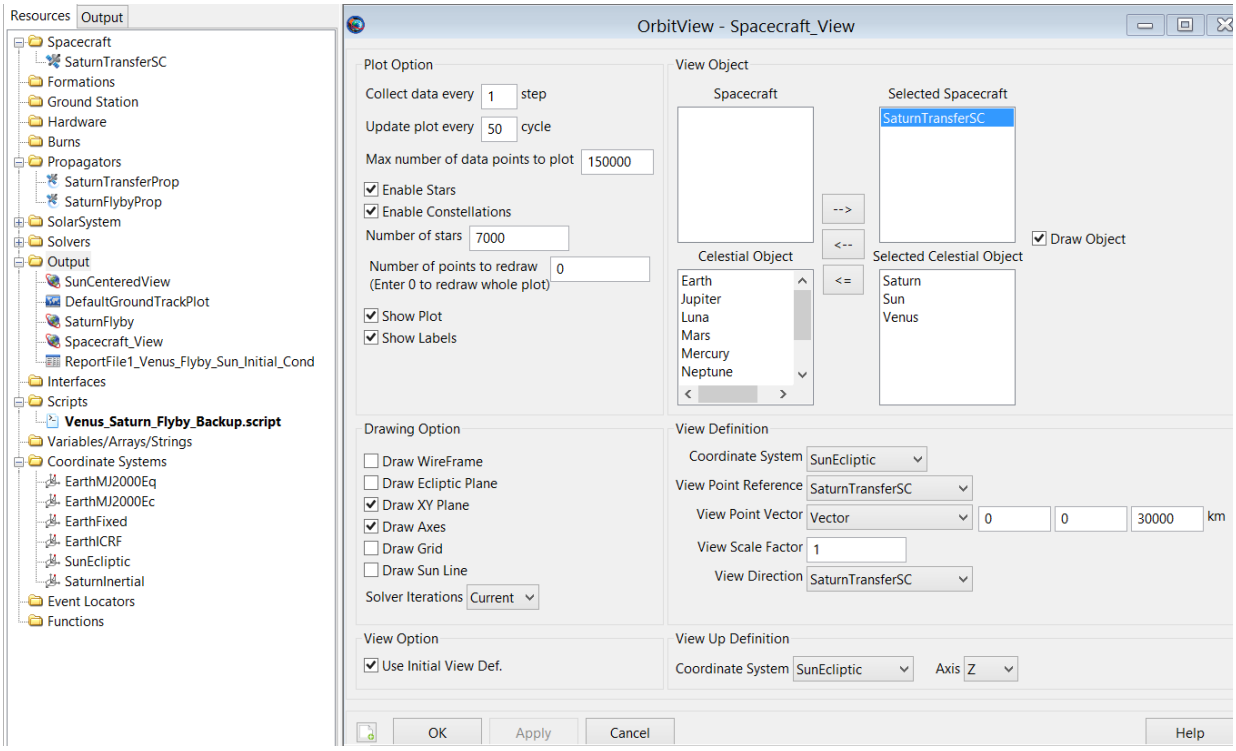


***Figure 111**: Spacecraft orbital view configuration in GMAT.*

'Outputs' option has more than one type of output to add (right-clicking on it will give a list to select from). The other type added is the report file itself (Note to rename any item press the 'F2' key).

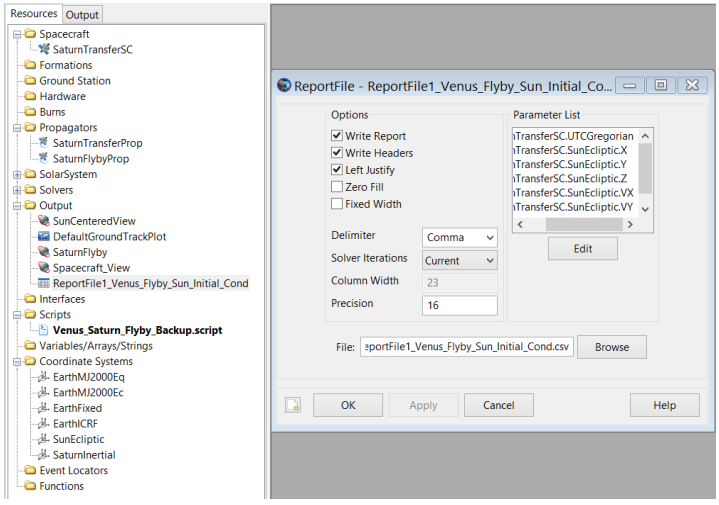


***Figure 112**: Report file configuration in GMAT. Note that the file is saved in .csv format, and the operator can edit the contents of the report file. Also, note that the Parameter can be empty if the*

'ReportFile' is considered as an action item in the mission sequence (under 'Mission' tab). An example of such can be found in section 4.1.5.

Once the attributes of the mission have been defined, the mission sequence (or the steps the script is expected to follow as it completes the simulation's iterations) is to be configured. The overall mission sequence was earlier displayed in Figure 103.

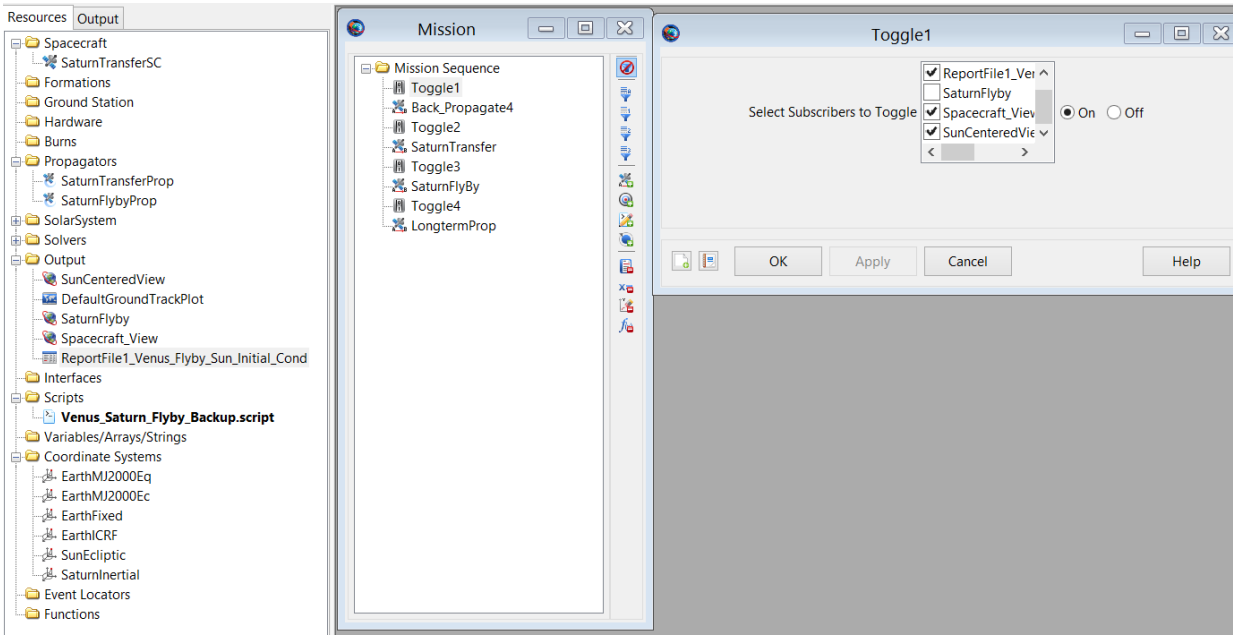


***Figure 113**: Mission sequence's Toggle 1 configuration in GMAT. Toggle is exactly what its name implies : a switch to turn report files or views on or off. The first toggle turns on the report file, spacecraft and sun centered orbital views. The spacecraft's orbital view remains on throughout the whole mission run.*

Mission tab allows the operator to insert anything after or before by right clicking any object type within it. For instance, right clicking on 'Toggle1' allows 'Back_Propagate4' the option to be inserted after it. The purpose of back-propagating was to go backwards in time so as to grab the actual initial conditions of the spacecraft (preferably 10 days before the offset point at periapsis defined in Matlab as Pos 1) as it is in transit towards Venus. This is to fit the narrative of having the spacecraft in transit as it approaches Venus rather than start at a parked orbit (where a burn -TOI - would have been considered). The 'Toggles' in the 'Mission' sequence gives the operator control over what to switch on or off throughout the script (or sequences of actions the simulator is expected to complete). The processes above and below present how to conduct such interference.

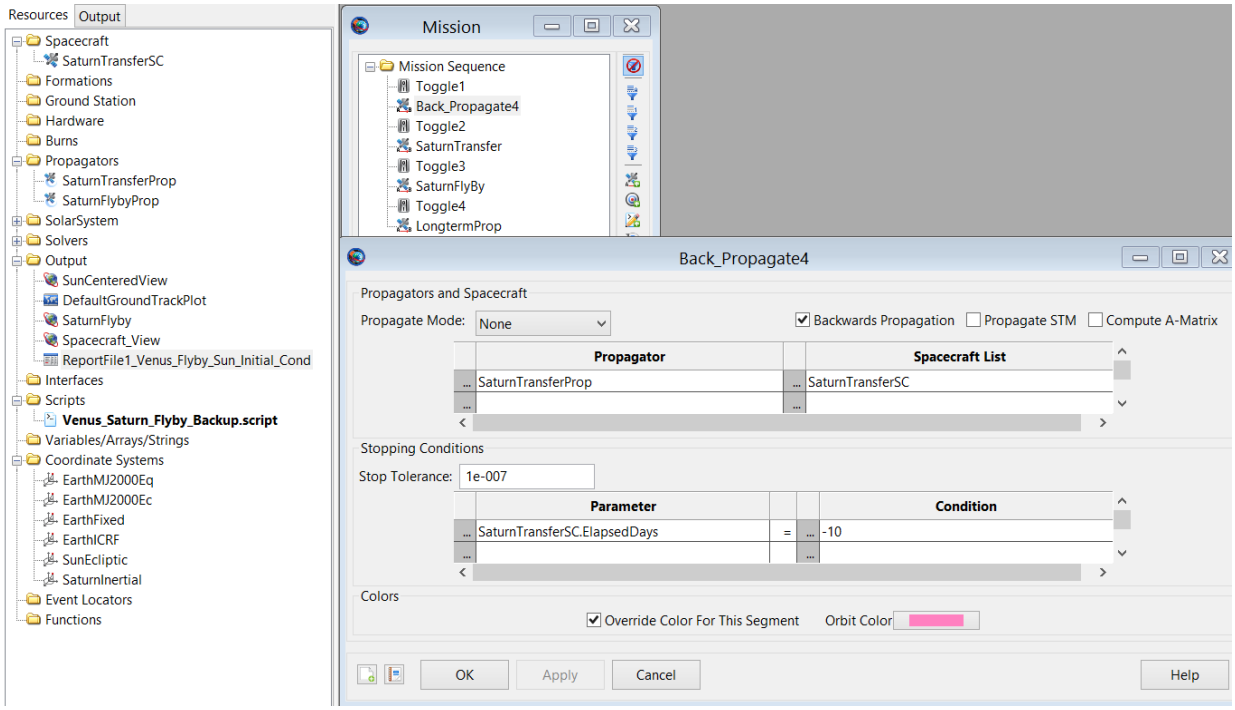


***Figure 114**: Back propagation configuration in GMAT. Note the three dots before each white text box. Those three dots are the menu lists for each designated column. Thus, the operator can select the preferred propagator, spacecraft, parameter and conditions. Conditions tend to be a numerical data type. The parameter selected is 'ElapsedDays' with respect to the spacecraft with the Saturn transfer propagator selected.*

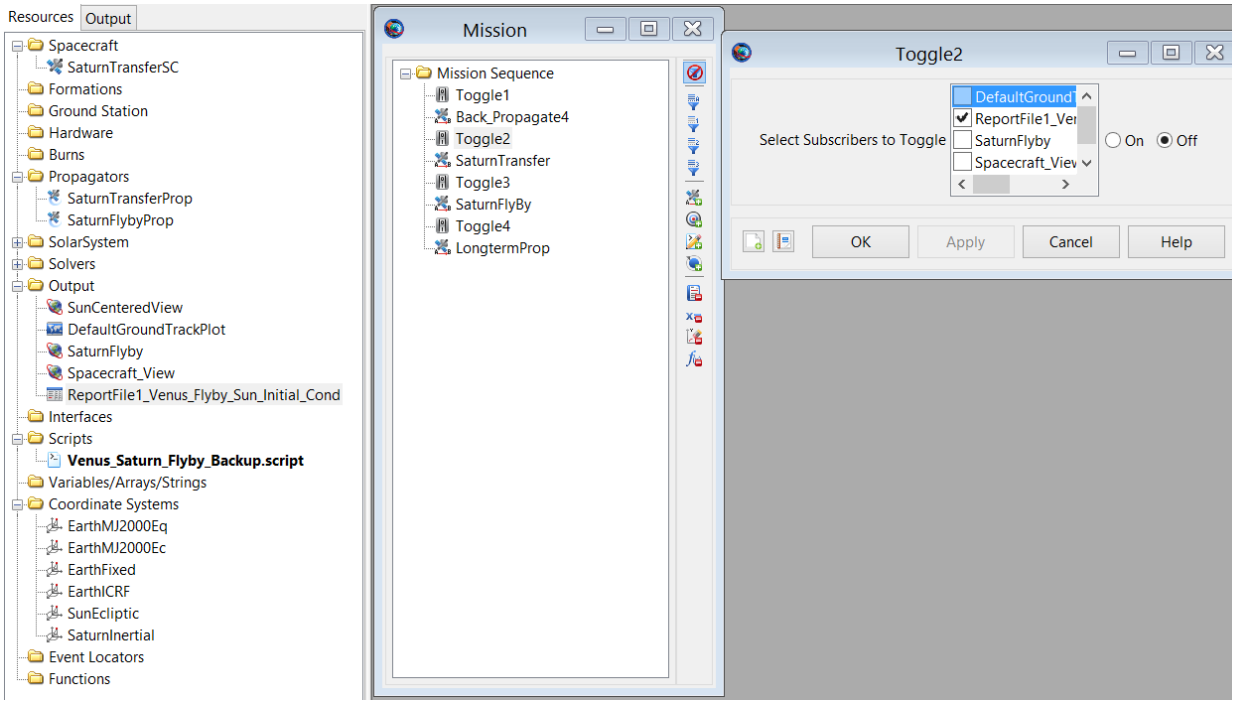


***Figure 115**: 'Toggle2' turns the output for data generation for a particular report file off. The toggles turn on/off any of the outputs defined under the Resources tab.*

Since, the purpose of the sun centered elliptical transfer is to collect the initial velocity and position vectors after the 10 day back propagation (as well as, to ensure that the spacecraft reaches Saturn and how its trajectory looks as it comes back towards Venus) the report file output was toggled off after the back propagation in the mission sequence.

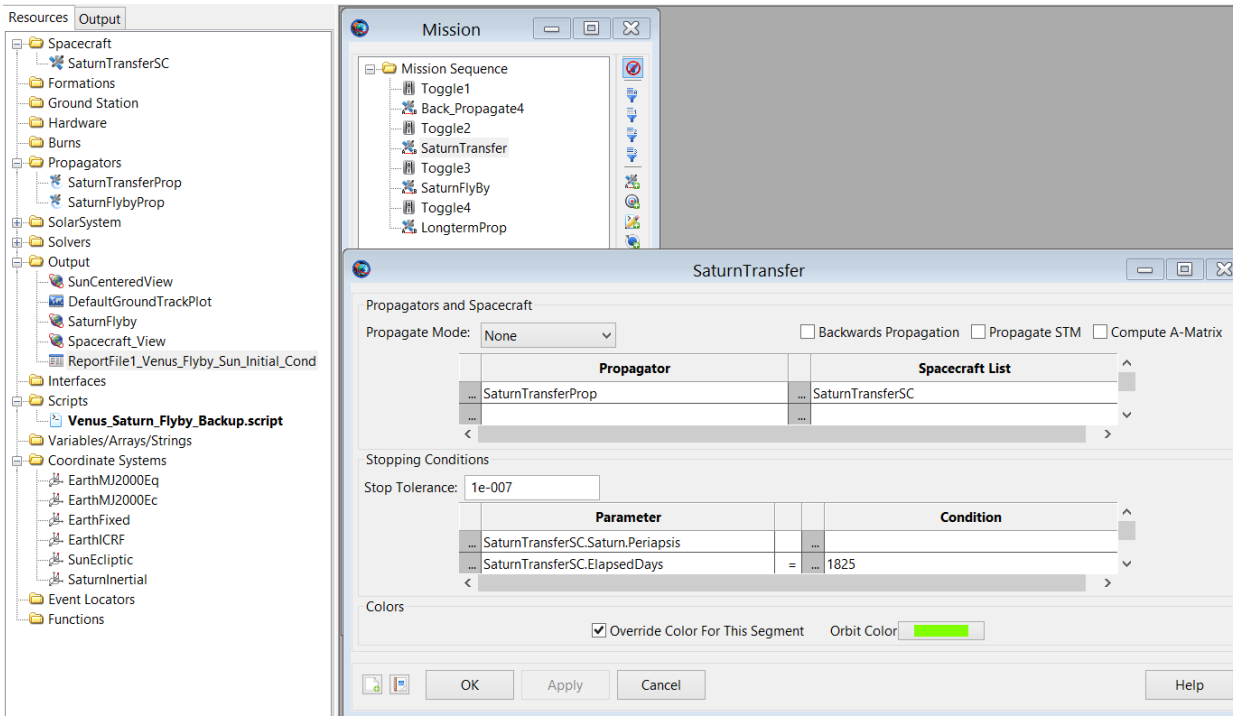


***Figure 116**: After the back propagation is done, the spacecraft is expected to propagate towards Saturn's periapsis. Note that although Pos 2 was defined for Lambert's problem in Matlab, it is not necessary to define it in GMAT - instead it is expected to indirectly solve that value for the operator. Thus, under the column parameter, 'SaturnPeriapsis' has been selected and after it a parameter that dictates the duration of achieving such a parameter (which is 1825 days or 5 years). Another way for GMAT to achieve certain parameters is through setting up targets in the mission sequence or solvers under the 'Resources' tab.*

It is advisable to use different colors for each propagation defined under the 'Missions' tab, for it will help distinguish between different trajectories or TOIs.

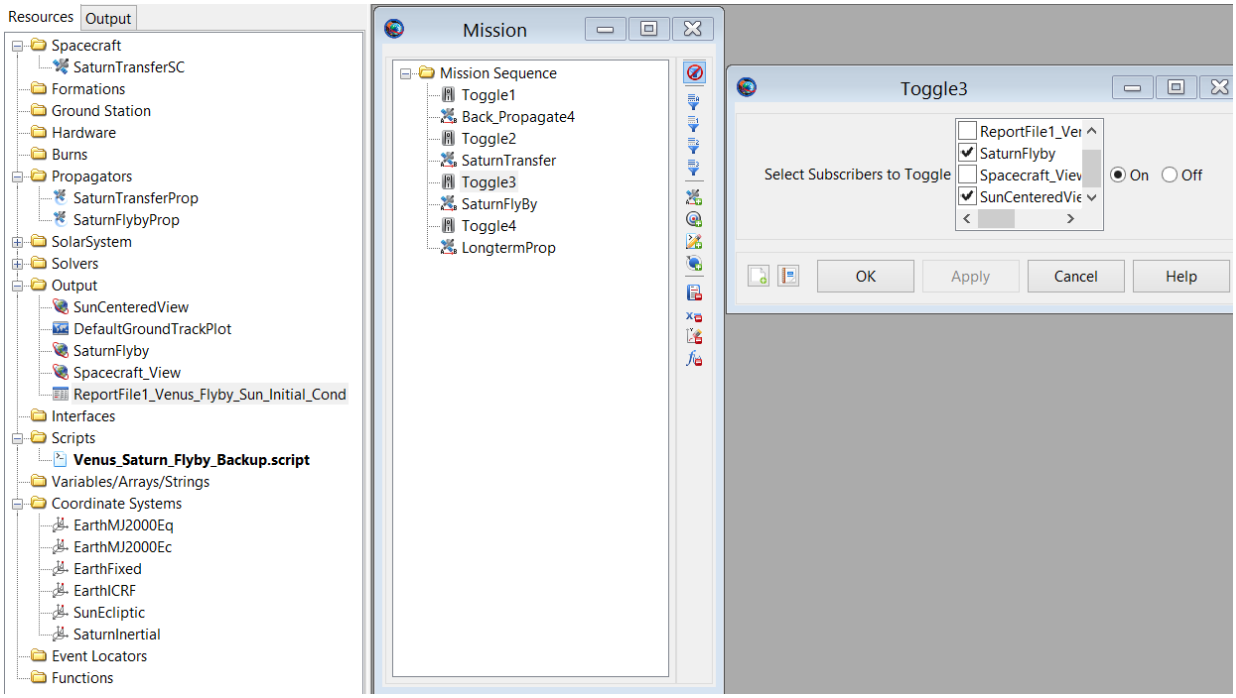


***Figure 117**: After the spacecraft transfers to Saturn, before propagating the flyby about Saturn, must turn on the Saturn centered orbital view and keep the sun centered view on via 'Toggle3'.*

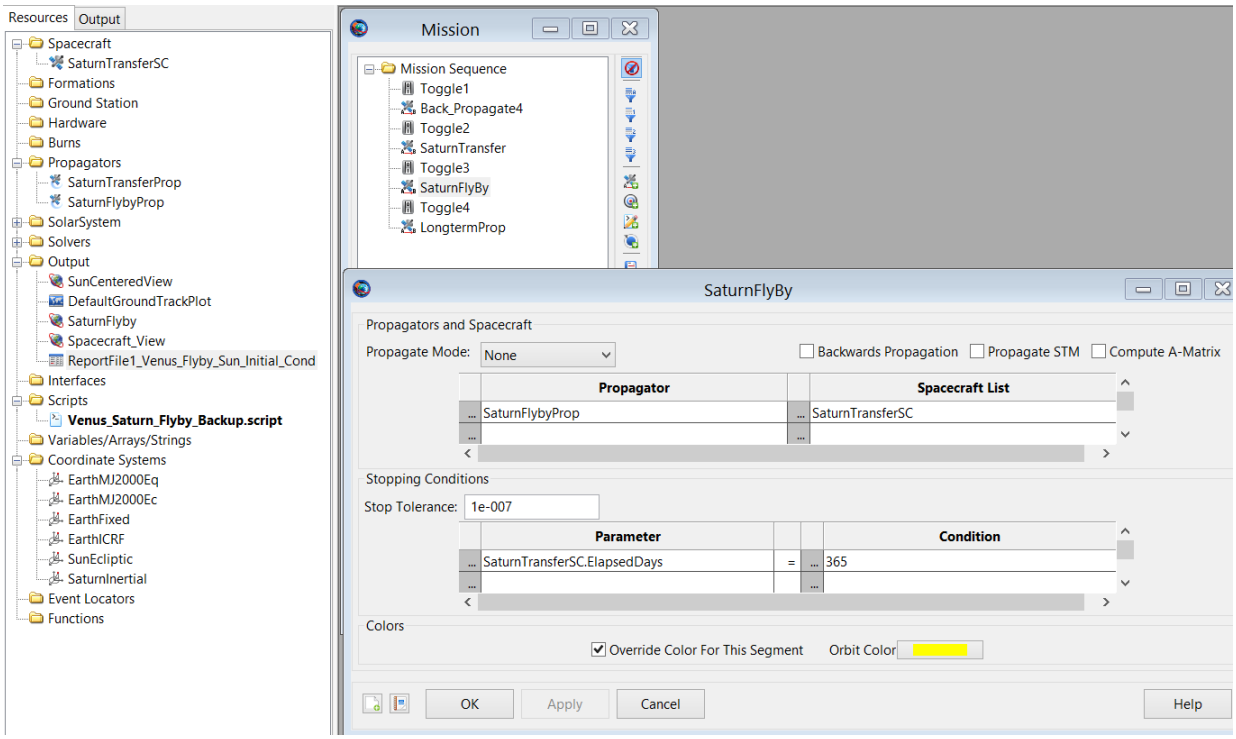


***Figure 118**: Saturn flyby propagation, for one year, under the 'Missions' tab in GMAT.*

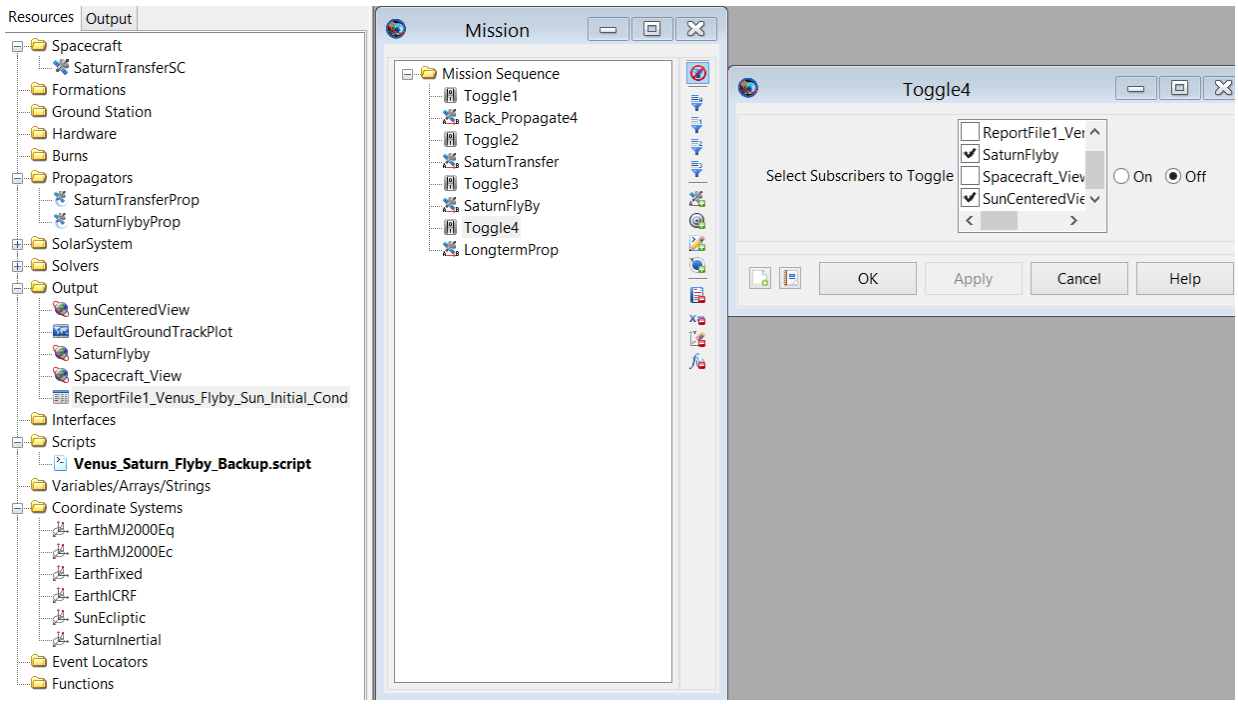


Figure 119: *'Toggle4' turns off the Saturn flyby and sun centered orbital views, after the Saturn flyby propagation is done.*

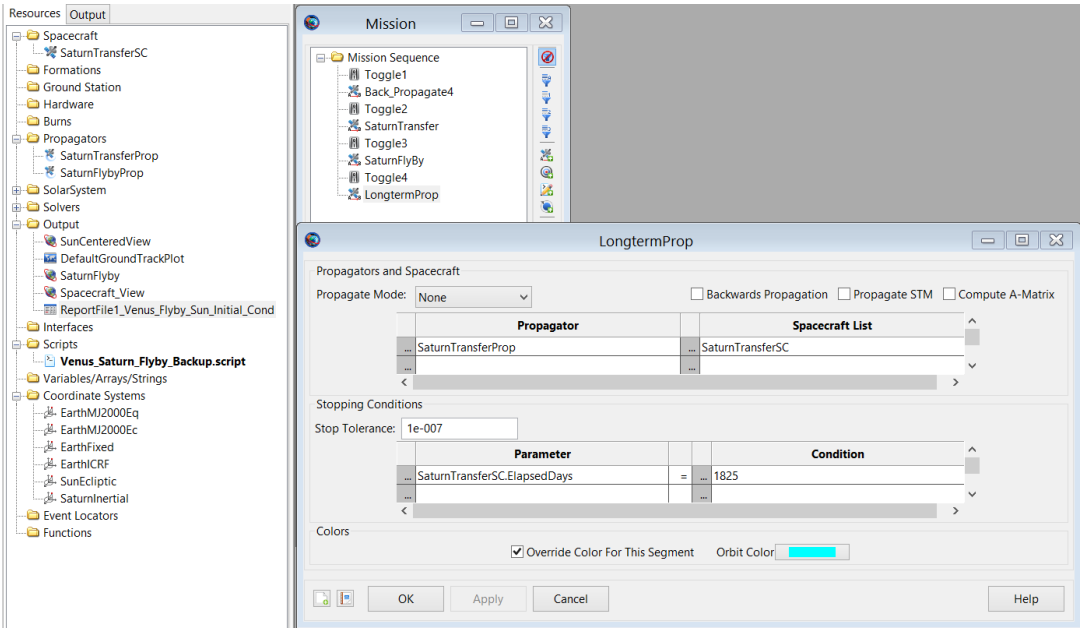


Figure 120: *The last propagation brings the spacecraft back towards Venus and an assumed time of flight (TOF) was set at five years.*

Note that scripts, located after ‘Output’ under the ‘Resources’ tab, can be selected and is the code version of the mission sequences defined under the ‘Missions’ tab. Scripts are modifiable and are automatically generated from the beginning to the end of the overall setup before running a mission (one script per file). Double clicking on the script file leads the operator to a window where manual modification of the script can be done. However, it is strongly advised to do all modifications via ‘Resources’ and ‘Mission’ tabs, and then make sure to synchronize (save) the GUI before each run. Thus, any changes made, under the ‘Resources’ and ‘Mission’ tabs, are saved to the script file as well.

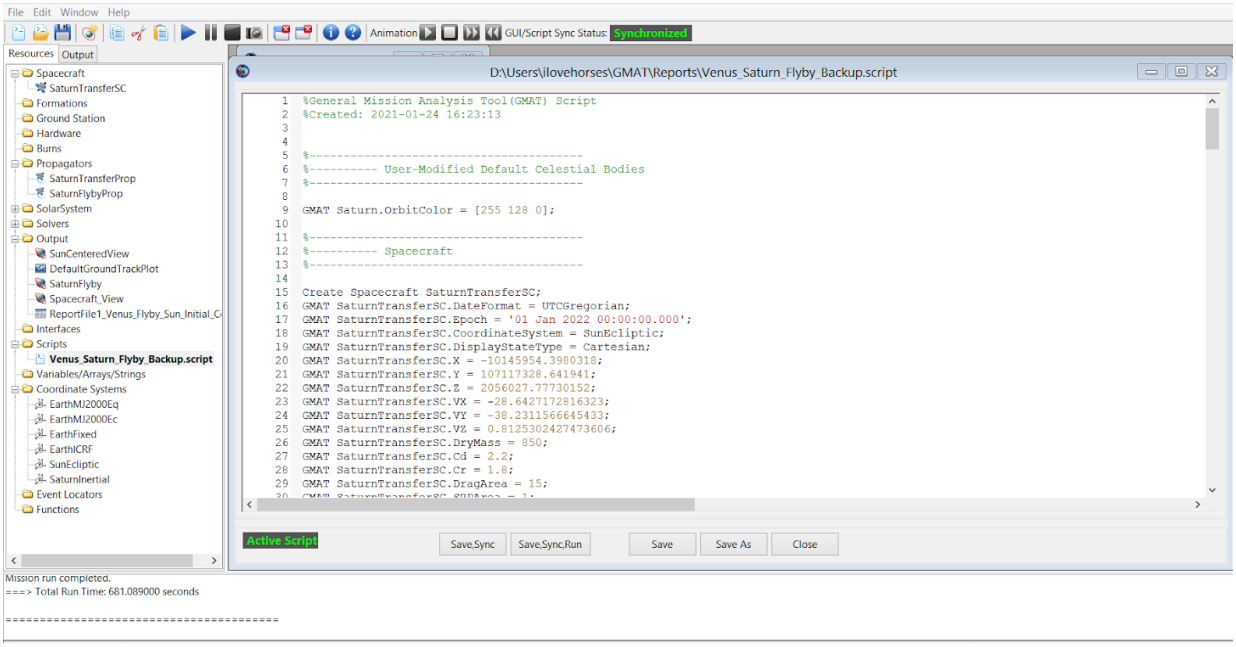


***Figure 121**: Sample of a script once opened in GMAT. Note the blue play button on the top task bar, which is the ‘Run’ button. The ‘Synchronized’ text highlighted in green indicates that the script is up to date (basically the file has been saved). Close the script file before running the mission. Once it’s done successfully, the console will have text stating that the ‘Mission run completed’ and the total time it took for it to be completed (it’s shown at the bottom).*

In GMAT, the orbital views created (under ‘Resources’ tab) will pop up while the mission is running and will have incremental time stamps shown as it runs to completion. There is an animation toolbar (shown in the above figure) that can be used to play back the trajectories once the simulation is complete.

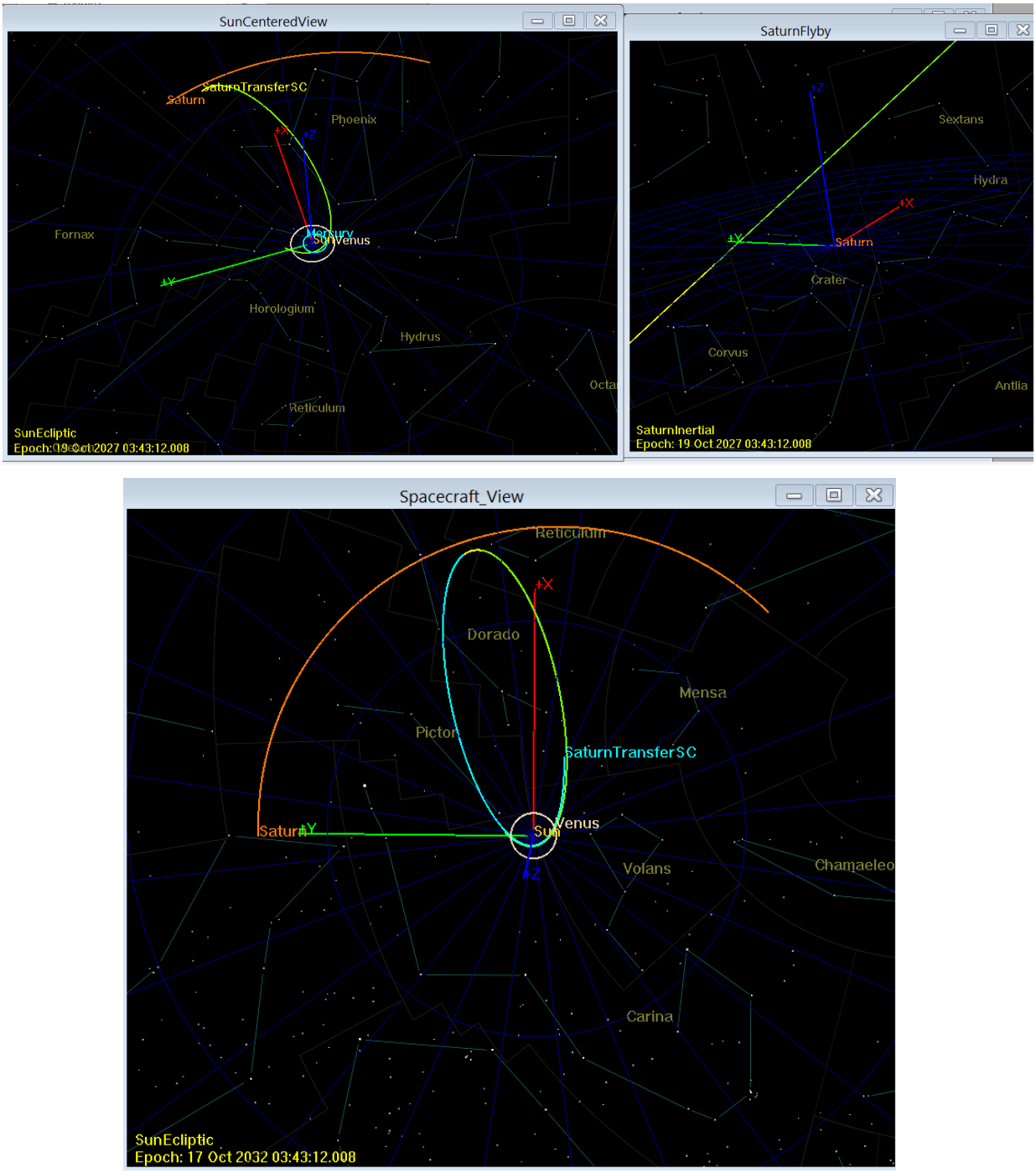


***Figure 122**: Sun centered , Saturn flyby and spacecraft orbital views, respectively from left to bottom, in GMAT. Note the swing by about the sun (the lime trajectory) in the sun centered view (the blue orbit is Mercury's, the white/beige orbit is Venus' and the dark orange trajectory up top is Saturn's). Note in the 'SaturnFlyby' orbital view, the over-exaggerated coordinate system shown is centred at Saturn and what is shown is the spacecraft's projectile as it begins a leading edge flyby (yellow) about Saturn. The orbital view pops up automatically once the simulation starts running.*

A leading edge, near-light or sun side , flyby about Saturn and back towards Venus are the last two parts that make up the complete mission for the elliptical sun centered orbit. (The first two parts are the back propagation from Venus' boundary and the hyperbolic trajectory - in lime-towards Saturn.) The 'SaturnFlyby' orbital view is a close up of the spacecraft's projectile as it reaches Saturn's periapsis before it commits to a free TOI (flyby in yellow) about Saturn. The 'SunCentered' and spacecraft orbital views are consistent that a leading edge flyby has taken place. As the spacecraft conducts a second swingby about the Sun (the aqua trajectory), it ends up closer to the Sun than the previous (lime) trajectory. Since the flyby about Saturn was a leading edge, the momentum (on the spacecraft by Saturn) was not as strong as a trailing edge (or dark side) would have been, and, thus, ended up flying closer to the Sun than its initial trajectory.

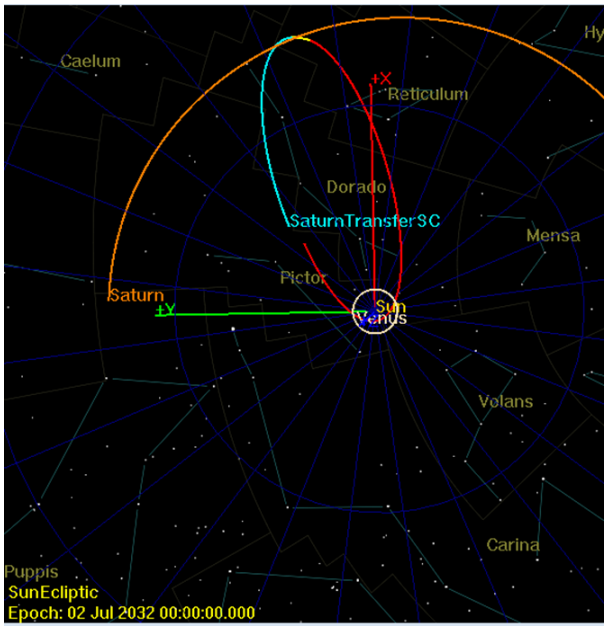


***Figure 123**: Trailing edge flyby about Saturn in GMAT. Backpropagation was set at 180 days from the same initial conditions in this section's setup. The mission sequence was also the same shown, for the leading edge setup, earlier. This means that if the actual full forward propagation started on 28 December 2022, a leading edge flyby would have occurred.*

The report file has captured the end conditions (position and velocity vectors) of the back propagation, which will be used (for both unperturbed and perturbed trajectories) as the initial points within the data for the NN modeling. However, before using this data, the next two sections will discuss the setups of the unperturbed trajectory, as well as the delta_v solver at the end of the two year mark.

4.1.3 GMAT Setup for Unperturbed Trajectory with Burn

The unperturbed trajectory does not include the potential influence of the Sun as the spacecraft attempts to re-enact the same swingby introduced in section 4.1.2. This section will cover the setup of the unperturbed trajectory in GMAT.

The ‘Resources’ tab conditions are the same as the complete elliptical Sun centered orbit, with the exception for the script and file names, as well as, inserting a burn (TOI) and modifying the propagators. (This burn will later on be removed since the current case scenario involves a spacecraft in transit as it arrives at Venus rather than starting from a parked orbit as was initially presumed.) The mission sequence under the ‘Missions’ tab is quite different from what was shown in section 4.1.3. Thus, more effort will be spent on the ‘Missions’ tab rather than the ‘Resources’ tab for this section (and the upcoming ones as well).

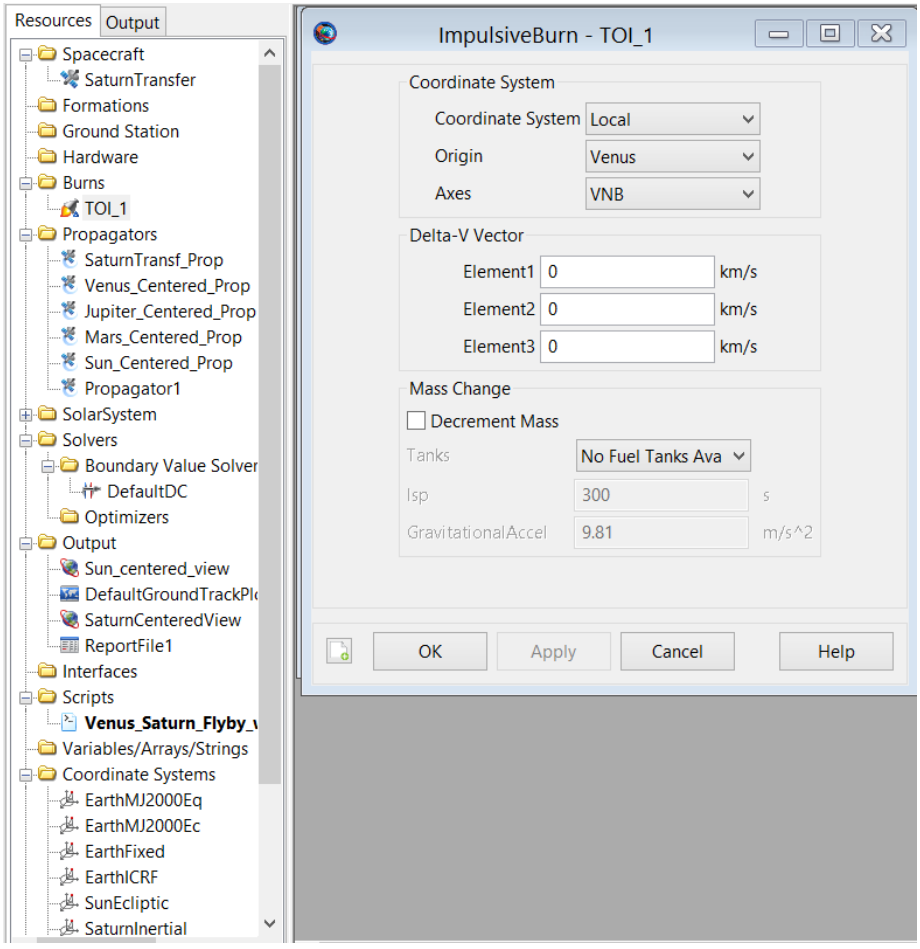


***Figure 124**: Burn defined via VNB (velocity - normal - binormal vector) local coordinate system about Venus. This coordinate system is at the center of the spacecraft and is with respect to the planet (or celestial body) selected. There are other options besides VNB, but since the critical body of \influence is*

Venus, using a Sun centered coordinate system is not necessary. Note that 'Delta-V' vector is set at zero for each 3-D element, and this is because a solver is expected to find the value for any (or all - depending on what is expected to be achieved in the mission sequence under the 'Missions' tab) of the three elements.

Note that there are many propagators in the above figure; this is due to the many trial runs and are not necessary for this mission. The only required propagators are the 'SaturnTransf_Prop' and 'Venus_Centered_Prop'.

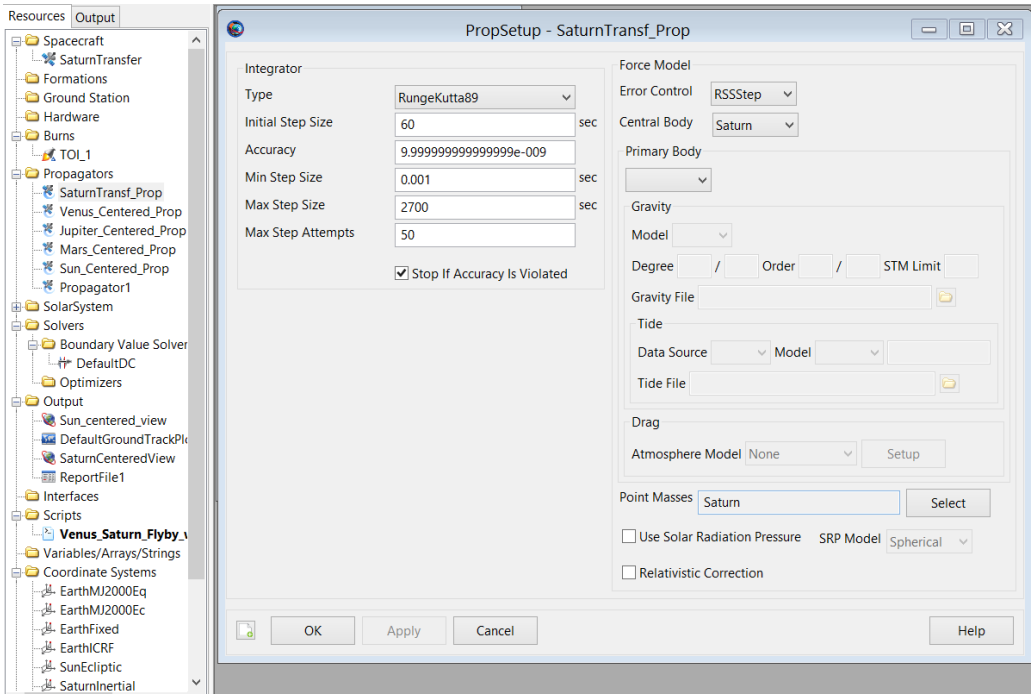


***Figure 125**: 'SaturnTransf_Prop' for the spacecraft as it attempts to commit to a trajectory towards Saturn as it escapes Venus' sphere of influence without the aid of the Sun. Note that the only point mass selected is Saturn.*

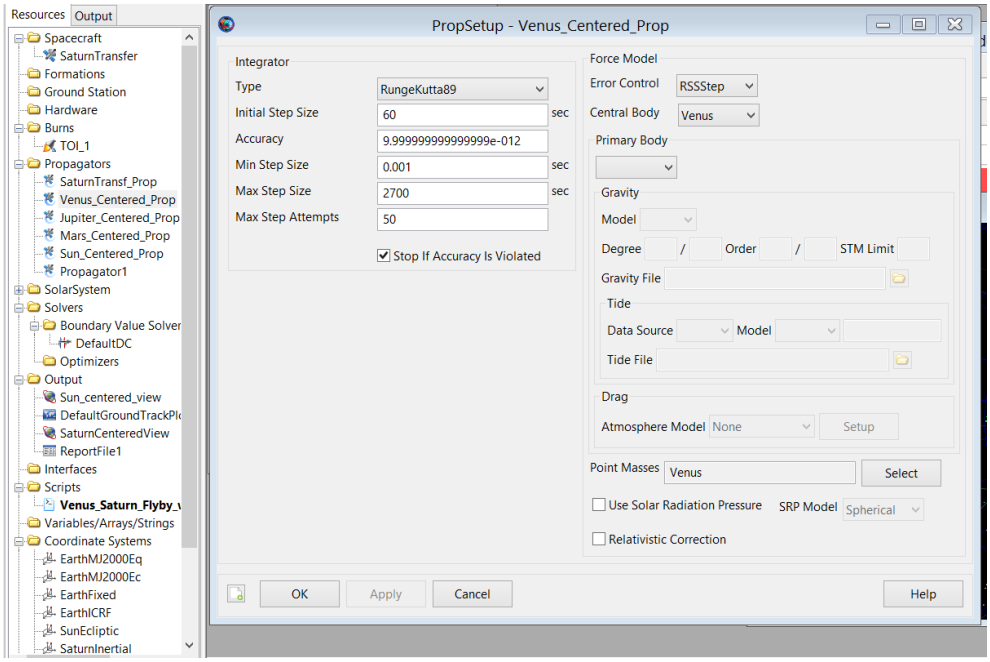


***Figure 126**: 'Venus_Centered_Prop' propagator in GMAT.*

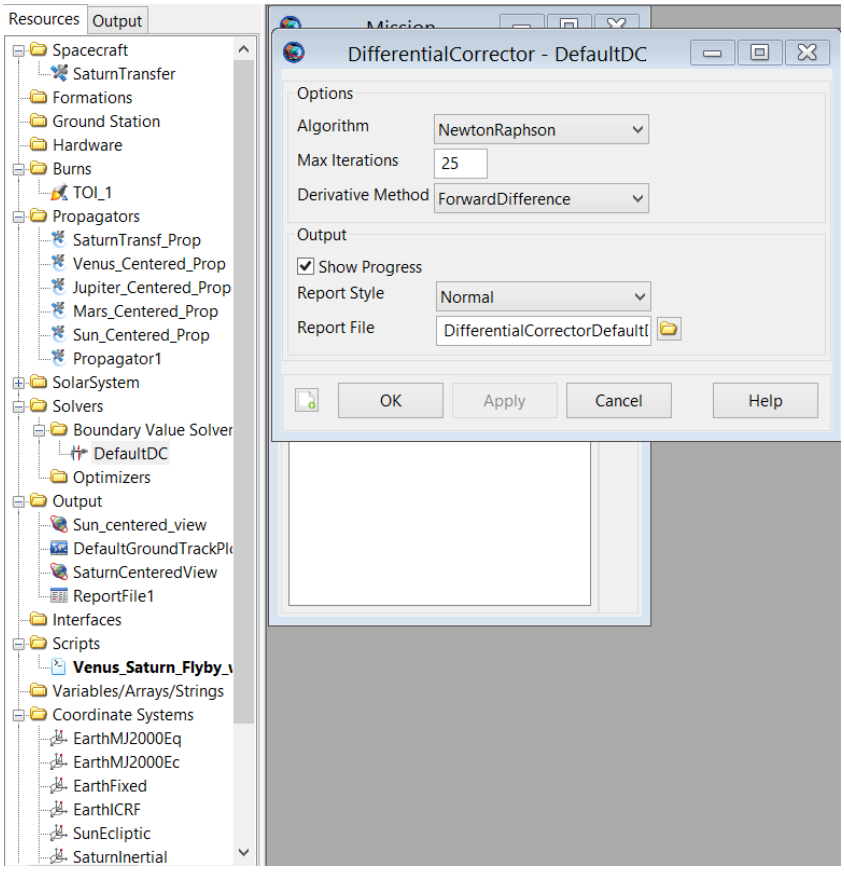


***Figure 127**: The solver is kept at a default but the computational power has been reduced from the original 50 to 25 iterations. This differential solver will be used to solve for the value of the TOI and stops the spacecraft from spiraling normally at a stopping point due to the absence of the Sun's influence.*

Since, a TOI has been defined with a value of zero for each of its elements, under the 'Resources' tab, then in the mission sequence, under the 'Mission' tab, it needs to be set up to solve it. This will require 'Target' and 'Achieve' markers as will be shown in the following figures.

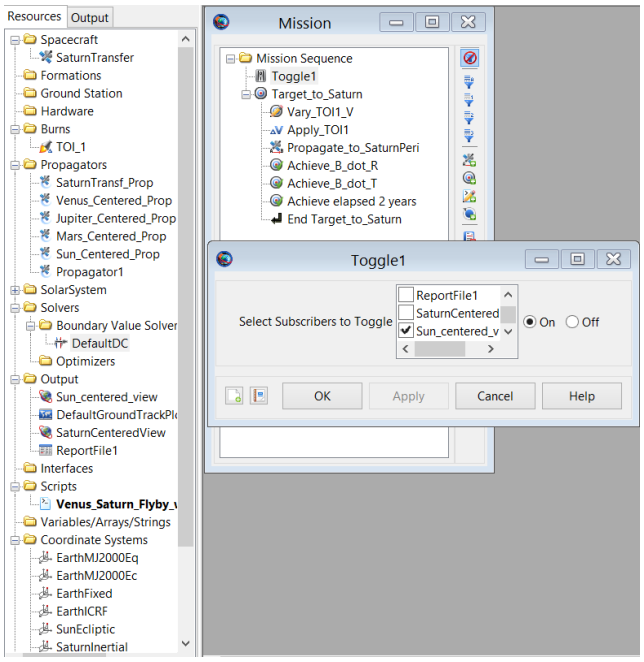


***Figure 128**: It is always handy to start with a toggle for a mission sequence in GMAT. 'Toggle1' is turning on the 'Sun_centered' orbital view.*

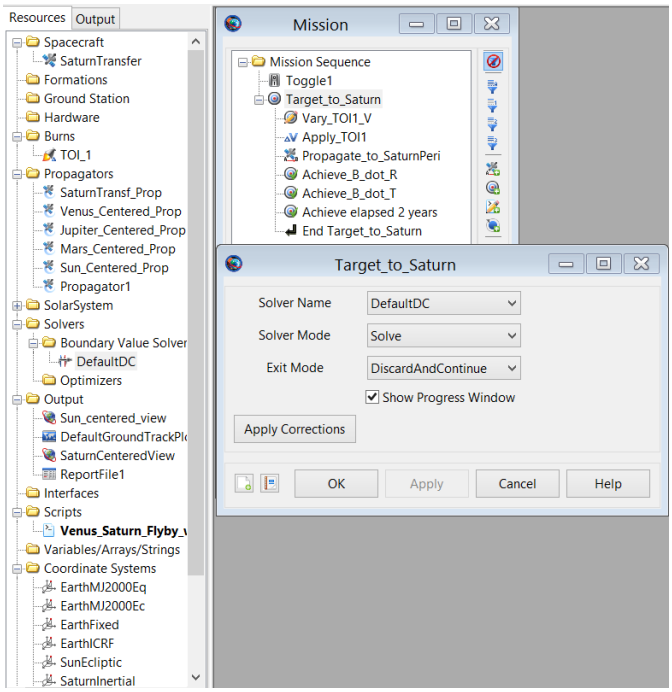


***Figure 129**: Establishing a 'Target' in GMAT is where the solver is used. Keep in mind that in order to insert action items within 'Target', one must right-click and select 'Append' first.*

The purpose of 'Target' is to use the solver to find the value(s) for the burn's element(s). However, in order for it to work certain items need to be appended as shown below.

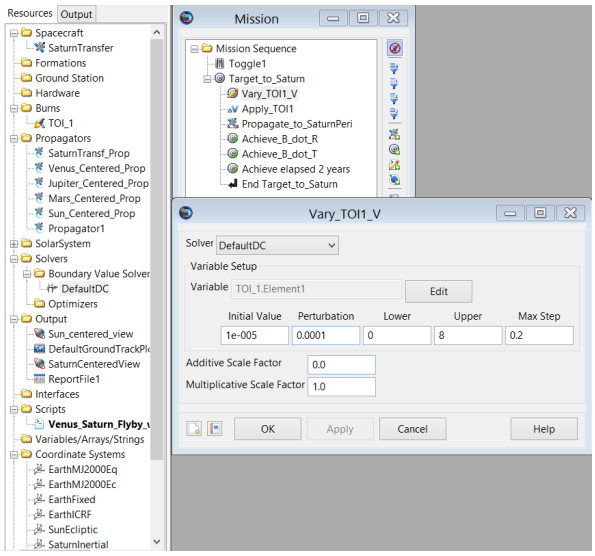


***Figure 130**: 'Vary' must be used to let the solver know its maximum and minimum constraints, with predetermined maximum steps, as it determines the burn value, in this case, with respect to only*

Element 1 of the TOI. Keep in mind that GMAT follows the mission sequence in order, so the burn is done at the very beginning since 'Vary' is the first action item.

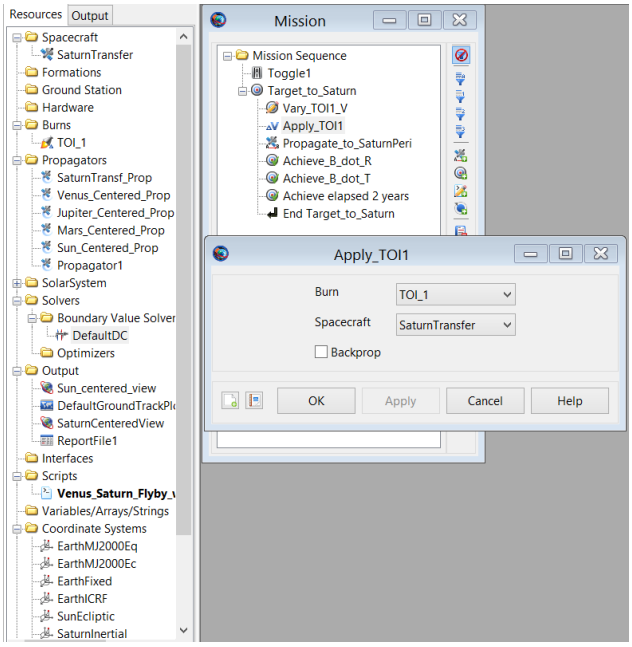


***Figure 131**: Insert a 'Maneuver' action item after a 'Vary' under 'Mission' tab in GMAT.*

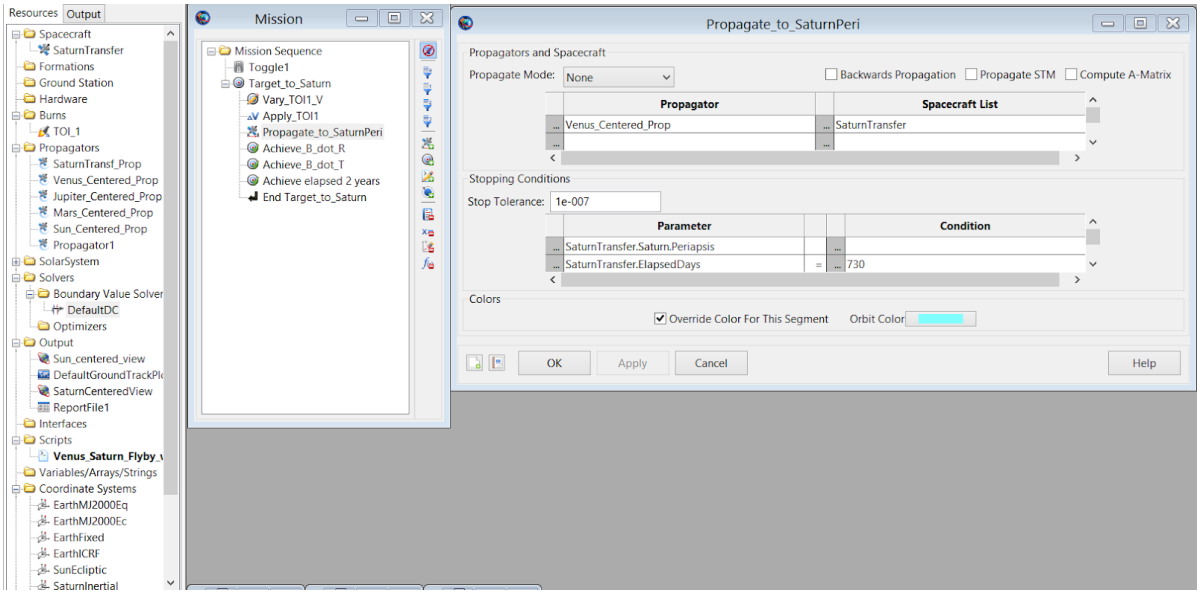


***Figure 132**: Insert 'Propagate' after 'Maneuver' in GMAT. Note the propagator used (which was initially defined under the 'Resources' tab), the parameters and the associated condition.*

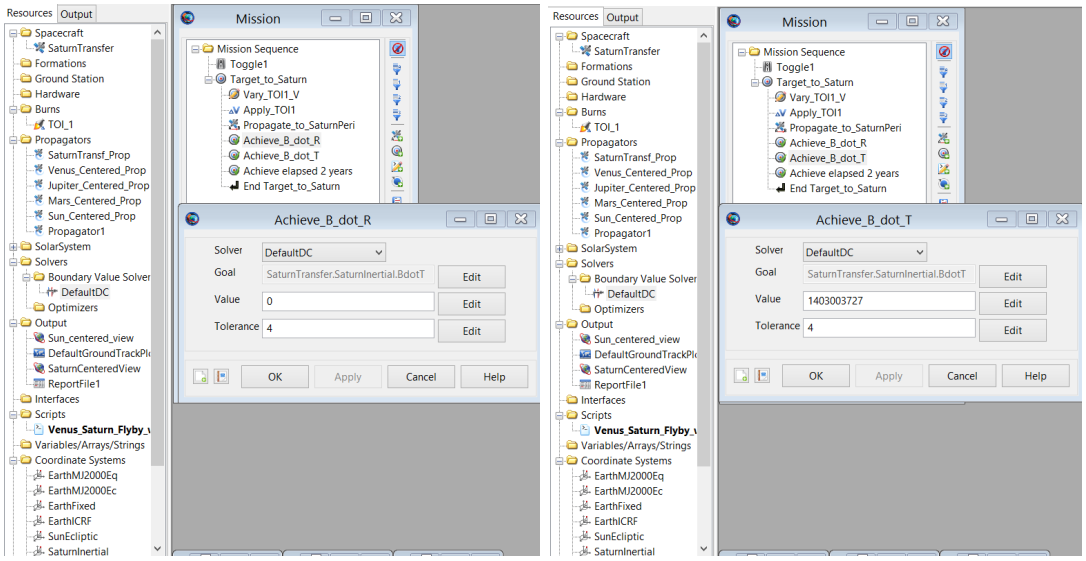


***Figure 133**: 'Achieve' marker after 'Vary' and 'Maneuver' action items. The 'Achieve' marker is for the solver to determine what is the goal and at what tolerance value. The 'B dot R' and 'B dot T' planes have been selected due to ease of constraints concerning a flyby - also called B-plane targeting. The B plane is composed of the vectors : R and T ; where T vector is the radial/tangential vector from the center of mass of the celestial body and R vector is the vector perpendicular to it. 'R' and 'T' determine the position of the spacecraft, with respect to Saturn, as it conducts a flyby about Saturn.*

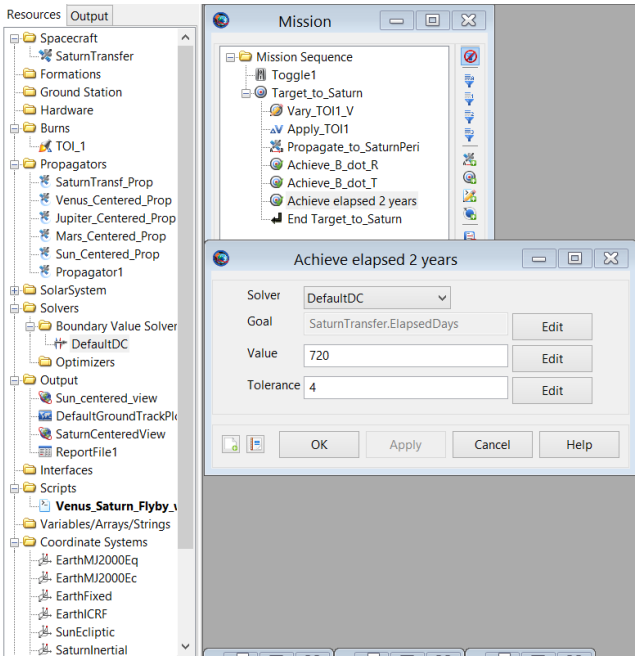


***Figure 134**: 'Achieve' marker for time it takes to transfer to Saturn in GMAT under the 'Mission' tab.*

Note that an operator may use multiple ‘Achieve’ markers after ‘Vary’ and ‘Maneuver’ action items have been defined.

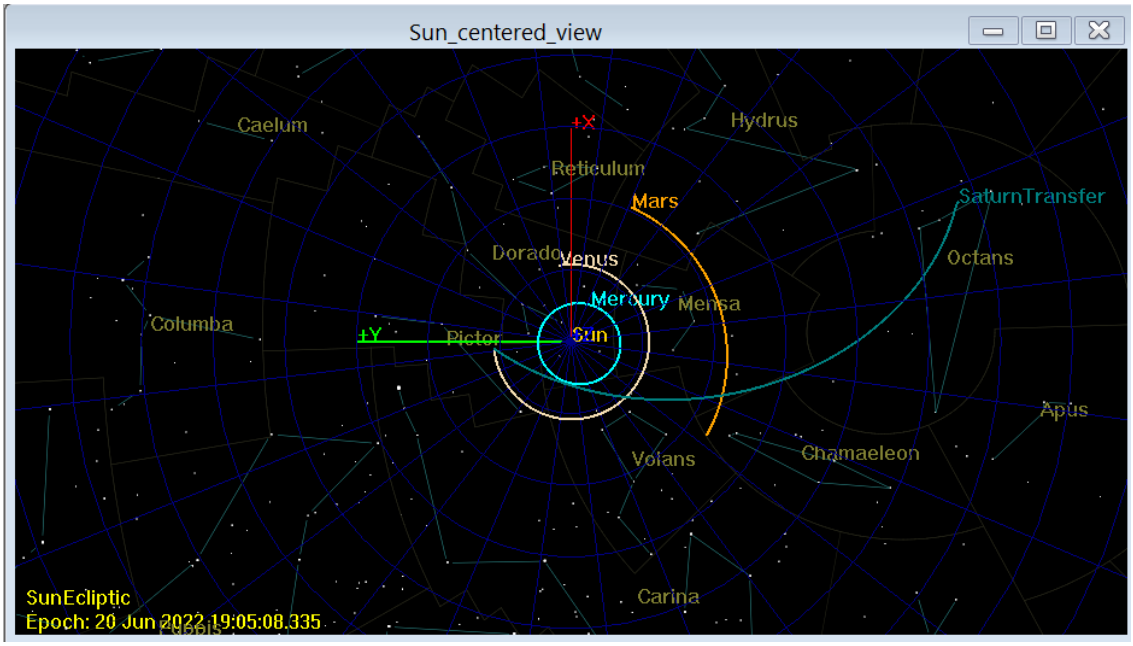


***Figure 135**: Sun centered orbital view of the unperturbed swingby trajectory of the spacecraft.*

Note that the spacecraft stops before the expected two year mark. Thus, the solver gives an output of non convergence since the desired output was at 720 days and the spacecraft can not reach the desired distance along Saturn’s periapsis. However, the non convergence is mainly due to the non existent presence of the Sun’s influence; the spacecraft stops after an initial TOI of 4.8 km/s and 180 days. If the solver was not used for this analysis, the spacecraft would have spiraled continuously normal to where it had stopped.

Solver Window - Target 'Target_to_Saturn' DefaultDC {SolveMode = Solve, ...

Control Variable	Current Value	Last Value	Difference
TOI_1.Element1	4.800010000000002	4.800010000000002	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SaturnTransfer.SaturnInc	1403003727	1005059597.480337	-397944129.5196629
(==) SaturnTransfer.ElapsedD	720	170.7952353579058	-549.2047646420942

NO CONVERGENCE

***Figure 136**: Solver window in GMAT. A solver has been defined under the ‘Resources’ tab. Note the TOI (burn or magnitude of the burning rate of the thruster) value for Element 1 that has been calculated after 25 iterations, and the actual days it has achieved (180 days) in contrast to what was*

desired (720 days).

The NN will require data from the unperturbed and perturbed trajectories between the final time stamp of the back propagation and end of the 180 day mark (as shown in this section). This setup will be discussed in section 4.1.5, and the burn will not be necessary.

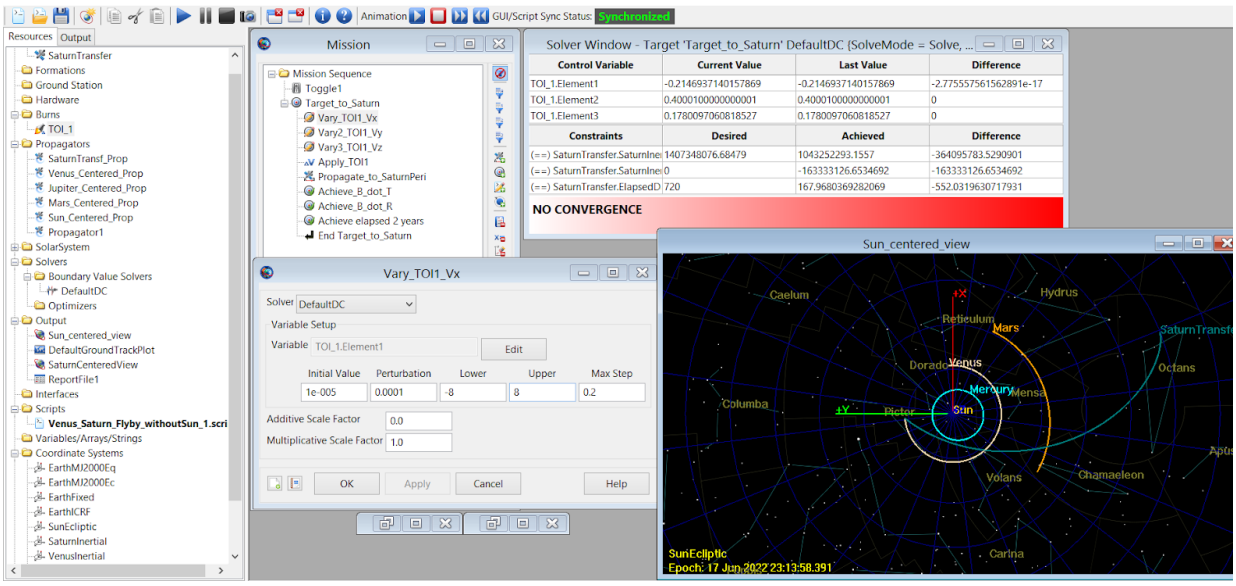


***Figure 137**: Experimental comparison when incorporating more than one element to vary in the mission sequence. Note that the limits have changed, and the trajectory remains the same. The no convergence warning is not critical since the analysis is not dependent on a particular distance along Saturn's periapsis and the Sun's influence is not considered. Thus, no convergence is expected.*

4.1.4 GMAT Setup for Trajectory with Burn at Two Year Mark

The purpose of this analysis is to determine how the Sun's perturbations (for the same swingby trajectory about the Sun towards Saturn) affects a correctional burn at the end of the two year mark. Some of the setup is the same as what was defined in the prior sections of this chapter, while others have been modified. The Sun, as a point mass of potential influence, is considered throughout this study.

Note that the contents of this setup are susceptible to changes throughout the duration of the project (and will be documented).

Most of the ‘Resources’ tab conditions are similar to what has been used in prior sections of this chapter. The spacecraft, propagators, coordinate systems (with the exception of a ‘VenusInertial’) and orbital views (with the exception of a Venus centered view) are all the same as discussed in section 4.1.1. The DC solver setup is the same as discussed in section 4.1.2 and 4.1.3 (with six instead of 25 iterations). The report file outputs have the same setup as discussed in section 4.1.1 but with different file names.

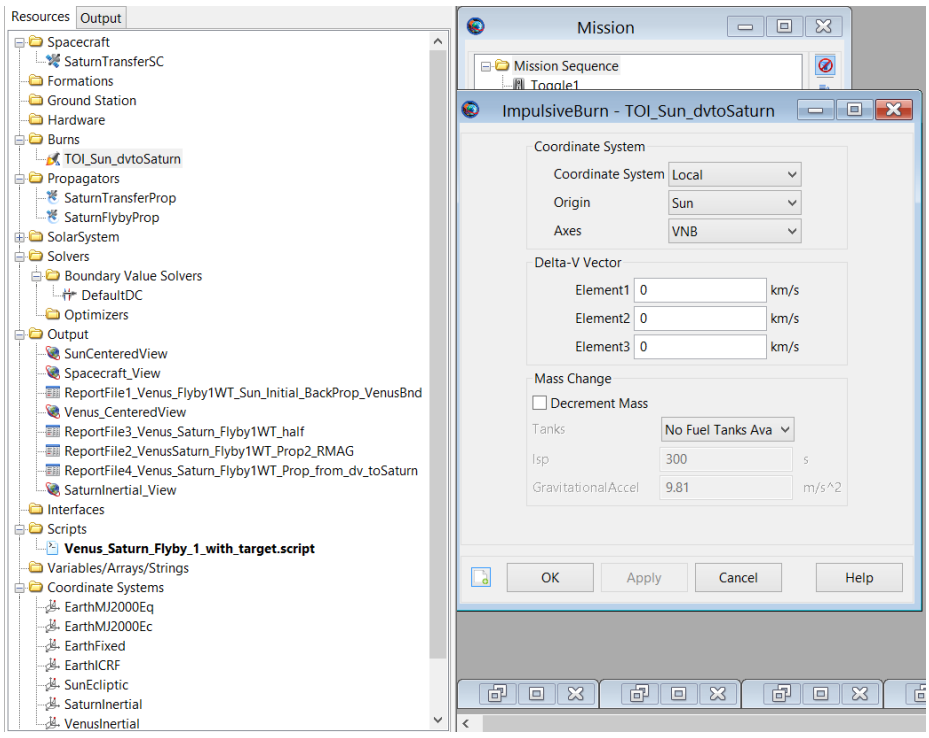


Figure 138*: Since the Sun is the central body at which the correctional burn is to take place, the spacecraft’s thrusters (VNB or Velocity-Normal-Binormal axis or X-velocity, Z - up or down and Y- left or right of the spacecraft’s central body) are with respect to a non-inertial and local Sun’s coordinate system instead of a planet’s.*

Note the multiple ‘ReportFiles’ as outputs under the ‘Resources’ tab. Each report is responsible for capturing data within a particular time interval of the mission sequence. This helps capture the position and velocity vectors at the two year mark, as well as, scope out the maximum radial magnitude (RMAG) that makes up the boundary of the sphere of influence of Venus. There are two methods to find the delta_v (burn) : manually calculating the difference between the

magnitudes of two velocity vectors at a particular time stamp, or using a solver to determine the burn value at a particular point in time. This study started with the former then ended up using the latter due to time constraint. Thus, multiple file outputs were used to capture such data and can be seen under the ‘Outputs’ branch under the ‘Resources’ tab. However, only the latter method (using a solver to determine the burn value) will be shown.

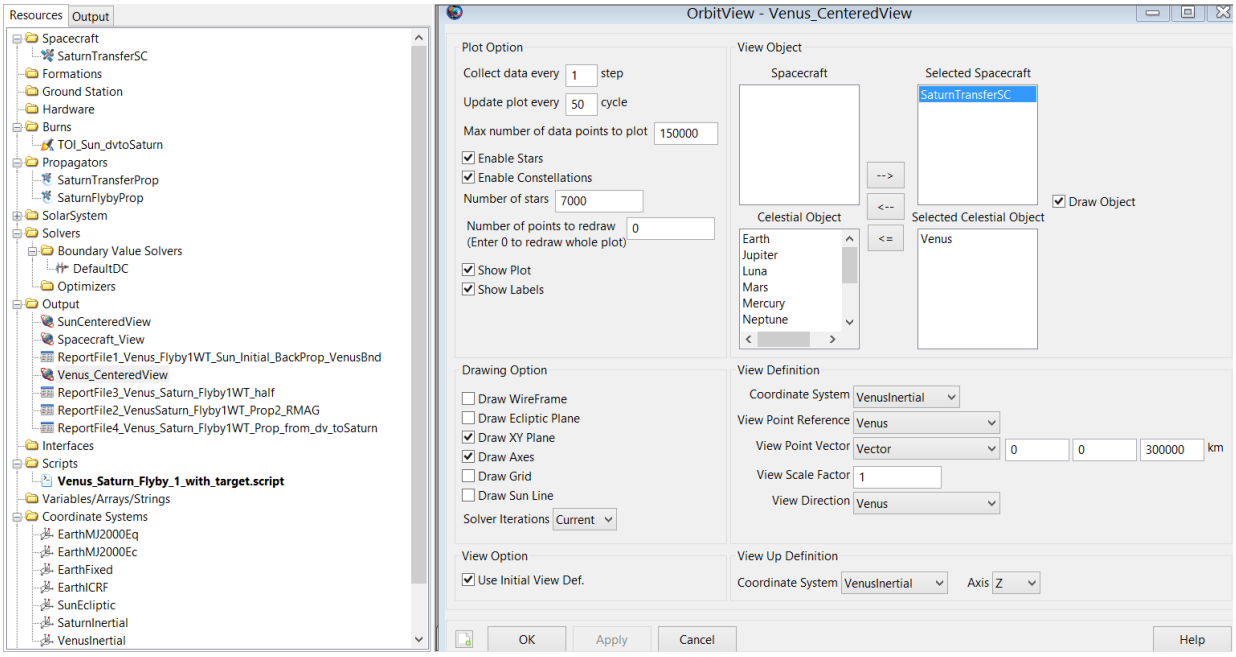


***Figure 139**: Venus centered orbital view settings in GMAT.*

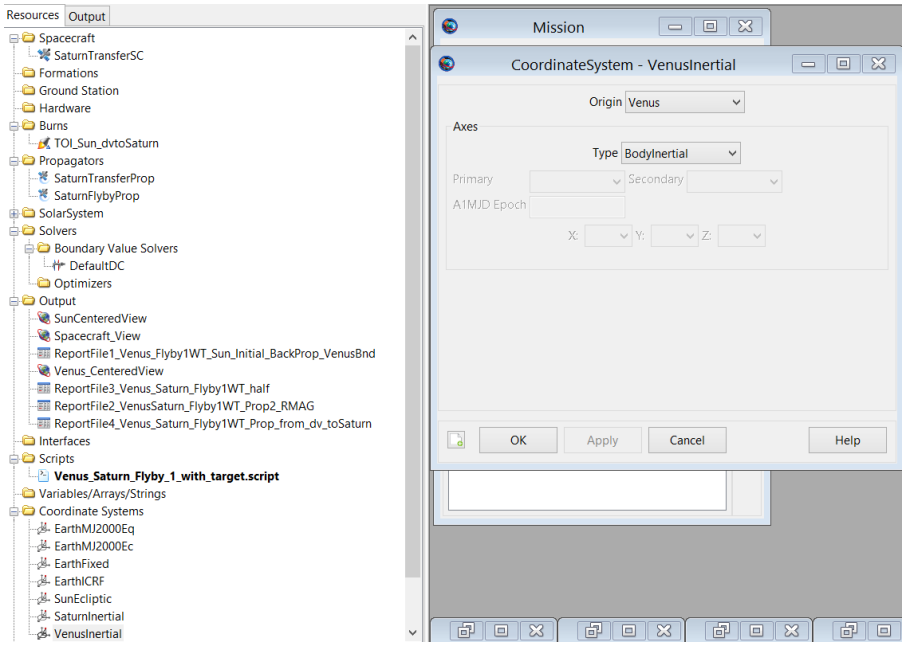


***Figure 140**: Venus inertial coordinate system setup in GMAT.*

The upcoming figures for this section will cover the action items for the mission sequence under the ‘Mission’ tab.

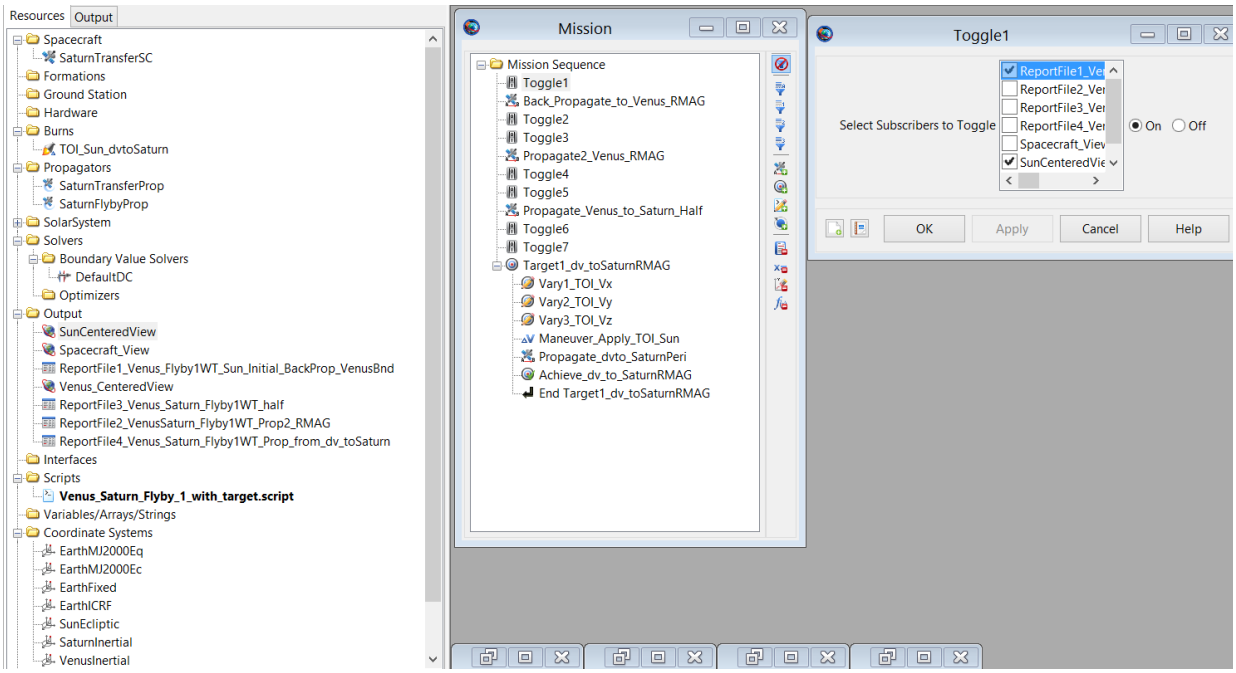


***Figure 141**: 'Toggle1' under 'Mission' tab in GMAT for perturbed swingby with target. ‘ReportFile1’ and the sun centered orbital view have been turned on.*

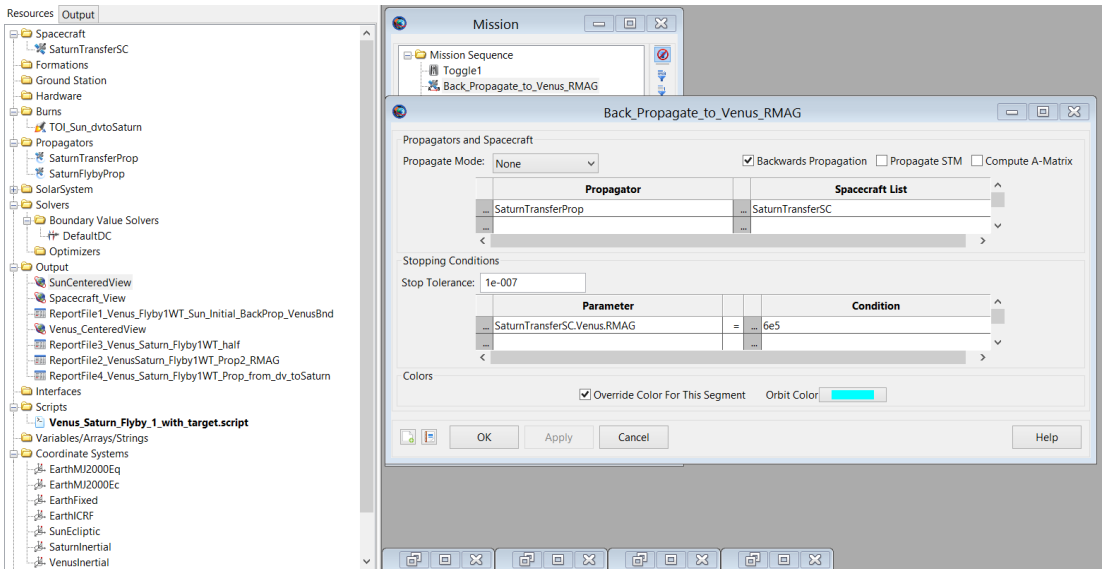


***Figure 142**: Back propagation to Venus' end boundary in GMAT. (This was just an exploratory part).*

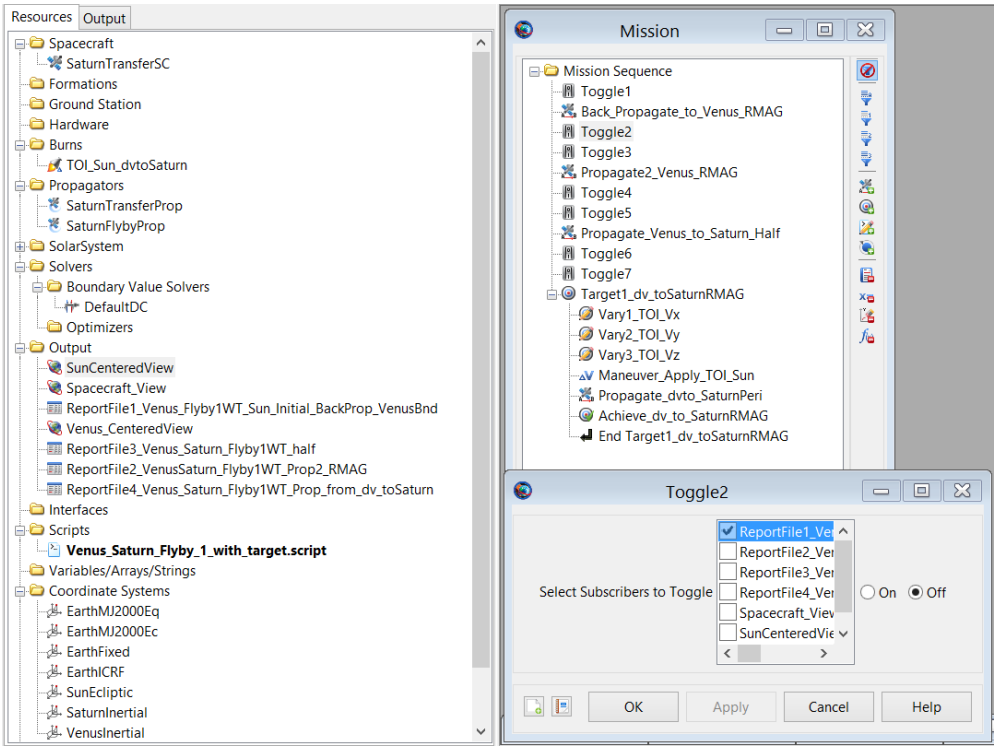


***Figure 143**: Second toggle is to turn 'ReportFile1' off.*

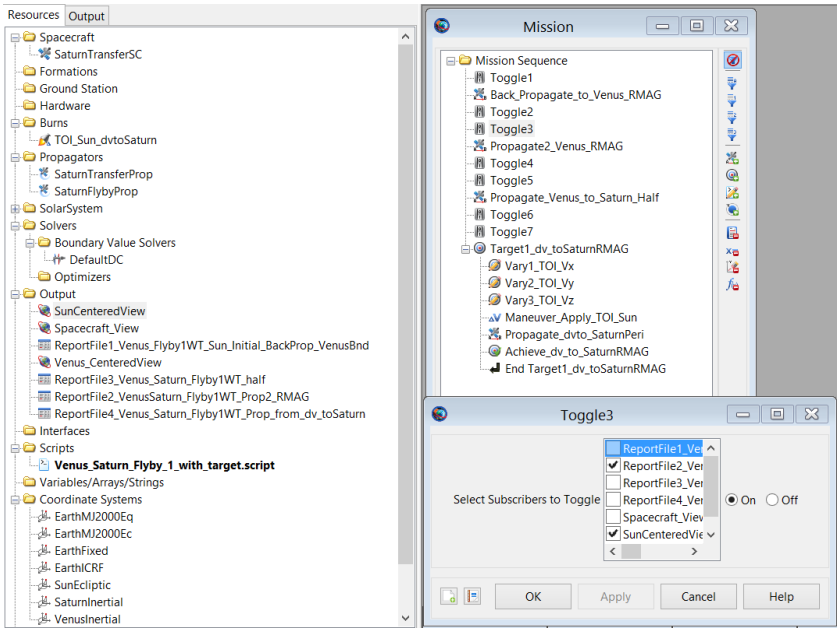


***Figure 144**: 'Toggle3' turns on 'ReportFile2' and the sun centered orbital view.*

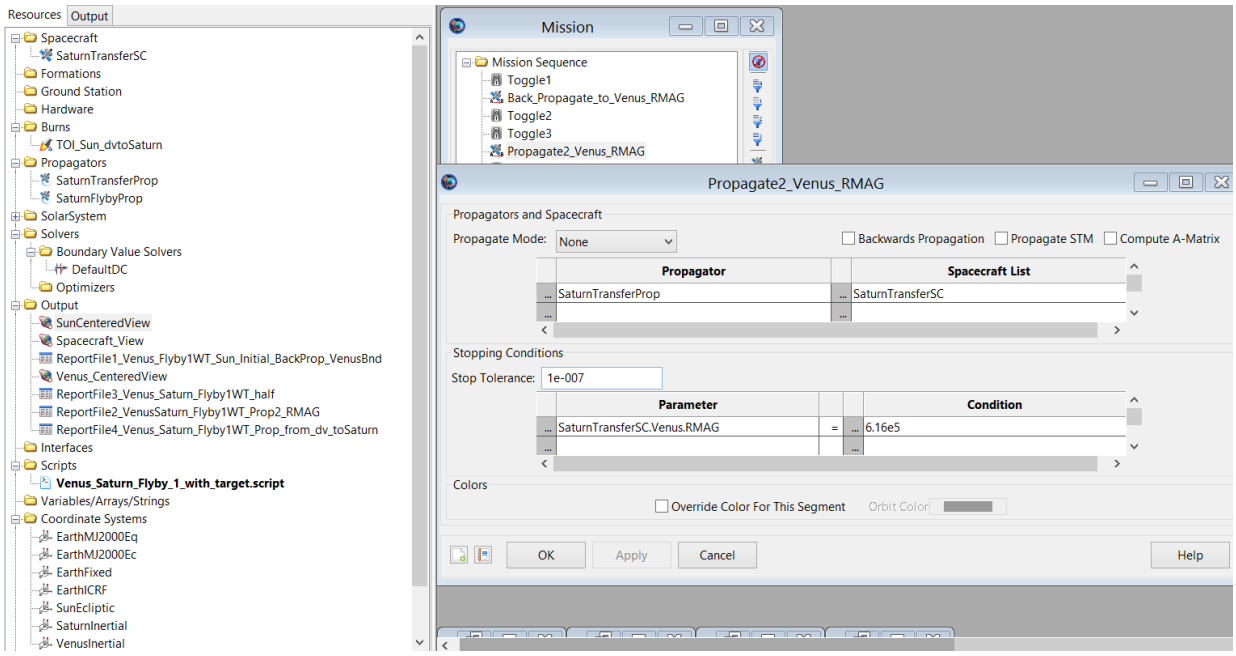


***Figure 145**: Forward propagating to the other end of Venus' boundary.*

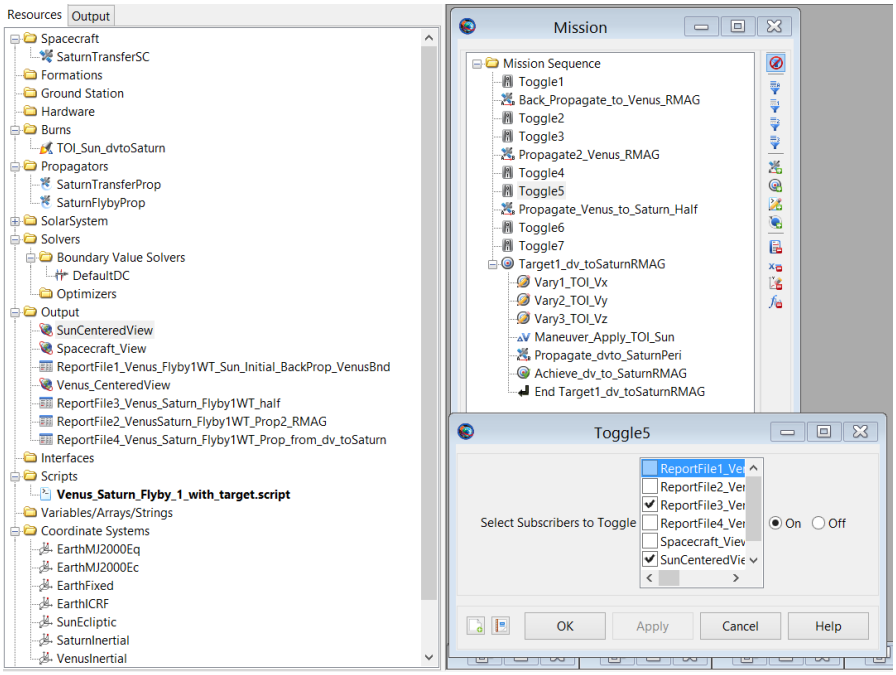


***Figure 146**: 'Toggle5' turns on 'ReportFile3' and sun centered view.*

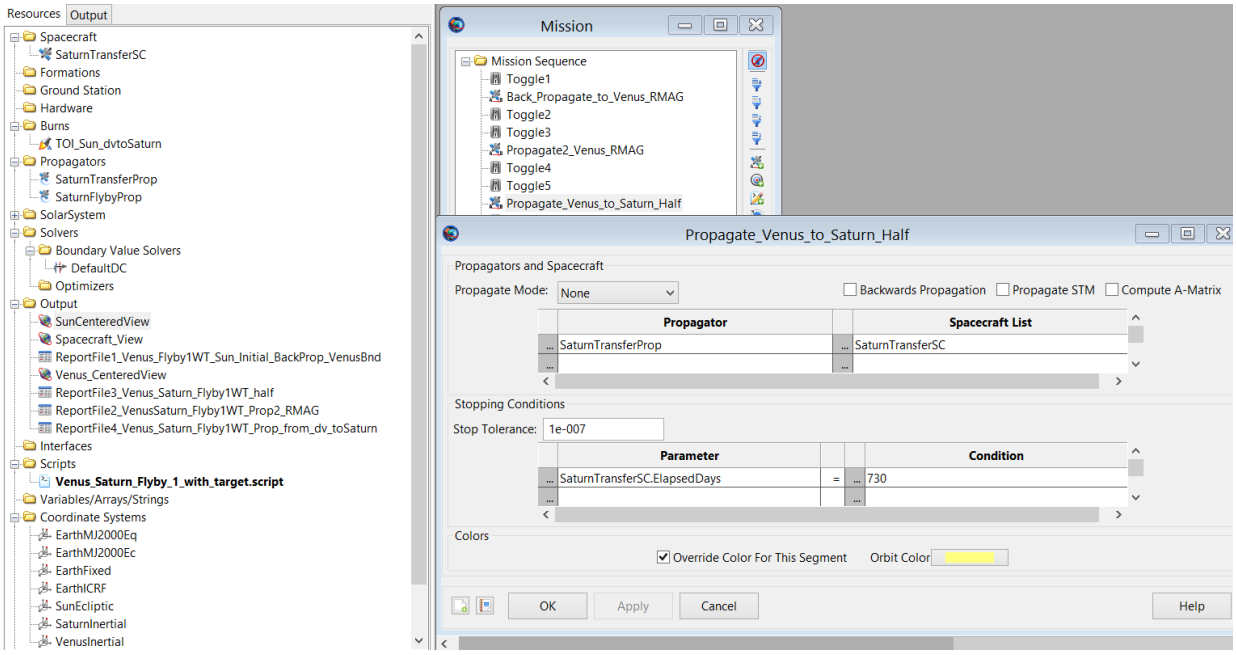


***Figure 147**: Propagating to the two year mark, from Venus to Saturn.*

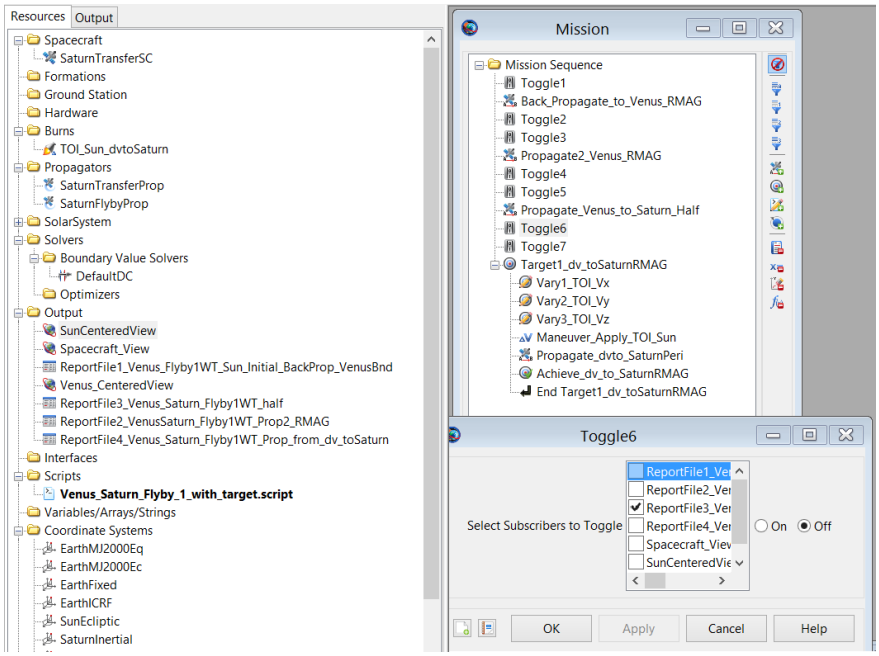


***Figure 148**: Turned 'ReportFile3' off.*

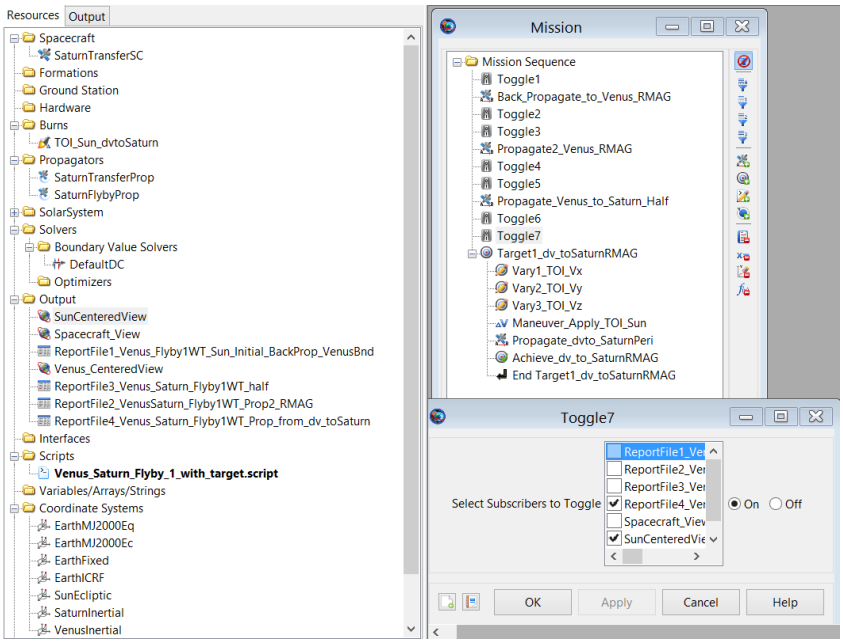


***Figure 149**: Turning on 'ReportFile4' and sun centered orbital view.*

The target part of the mission sequence (under ‘Mission’ tab) has been modified (from what is previously shown in the figure(s) above) and will be shown following this statement. However, it is important to note that the target tends to have a common pattern of : ‘Vary’ (varies the variable - in this case it is the burn - with stated limits and step size), ‘Maneuver’(applies or performs the burns or whatever stated as a variable), and ‘Achieve’ (basically stating the variable [in this case it was Element 1 of the burn), and its associated desired value with allowable tolerances). It is also critical to note that the Keplerian elements can be considered as variables (when doing the ‘Vary’ command) before applying a burn through a ‘Maneuver’. Keplerian elements can also be used as variables to be achieved (under the ‘Achieve’ command).

B-plane targeting’s elements are used as the method of choice for the ‘Achieve’ stage of the mission sequence. This is because B-plane targeting is ideal for gravity-assisted flybys. The B-plane consists of three vector components: ‘T’, ‘R’ and ‘S’. ‘T’ and ‘R’ make up, visually, the ‘X’ and ‘Y’ axis on a 2D-cartesian coordinate plane, while ‘S’ is pointing in or out of the page - just like the ‘Z’ axis in a 3D-cartesian coordinate system. Thus, when doing the B-plane targeting, the operator has the option to use ‘BdotT’ and ‘BdotR’ as variables to be achieved.

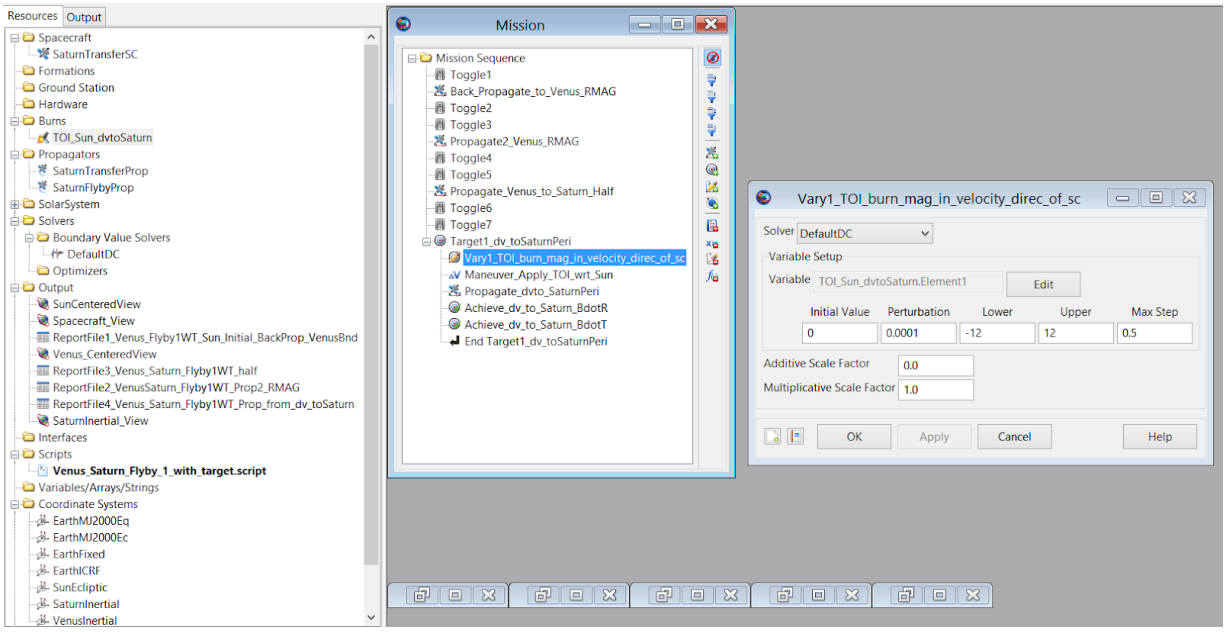


Figure 150*: Vary action item for the first element of the correctional burn (at the two year mark). Note the lower and upper limits. It is critical to first start with a wide range to allow the solver not be too constrained. It is advisable to use max steps between 0.2 and 0.5. No need to specify coordinate axes for burns.*

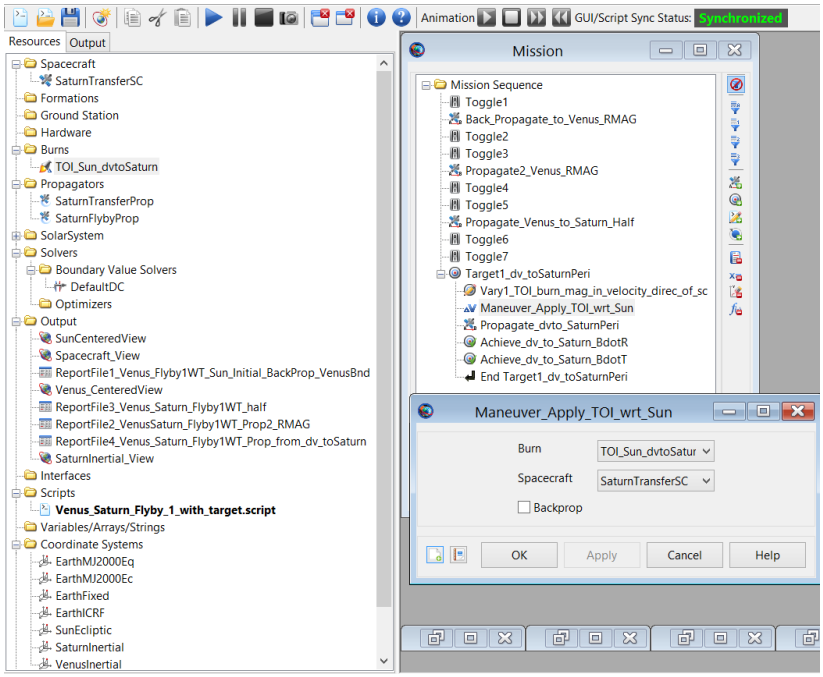


***Figure 151**: Applying the 'Maneuver' (or burn) that has just been varied.*

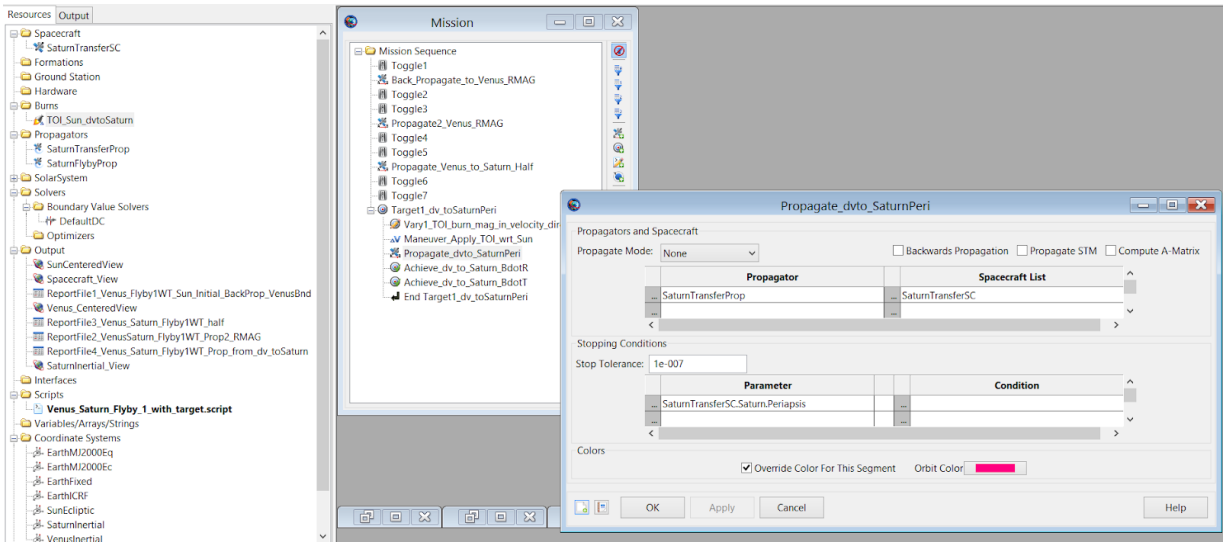


***Figure 152**: Propagate from the stop marker (two year time stamp) and continue onwards towards Saturn's periapsis. Although, in MATLAB, when using the Lambert's solver, a POS 2 was defined. Understand that POS 2 is defined for the sole purpose of ensuring that the two velocity outputs that the solver gives does indeed cause a rendezvous to happen between the spacecraft and Saturn. In GMAT, it is sufficient, since only the burn is critical, to just define the expectant parameter as Saturn's periapsis.*

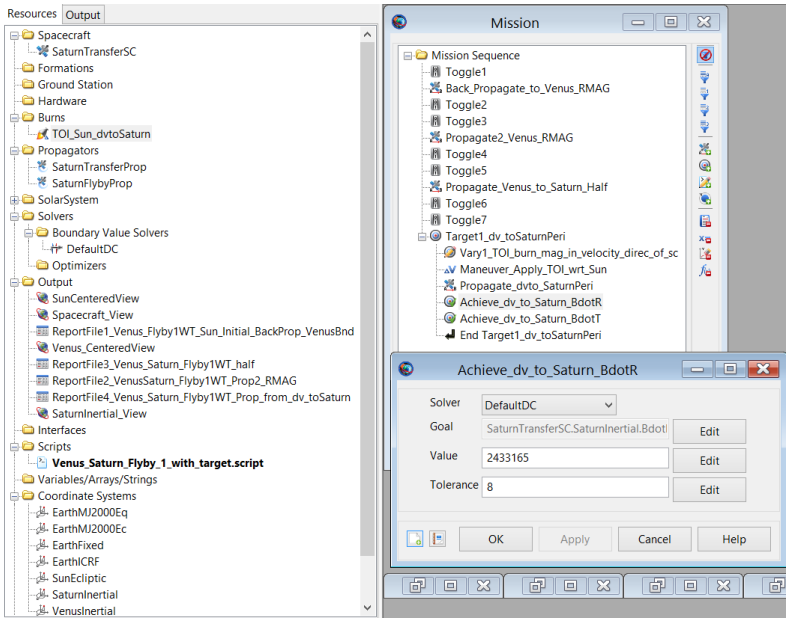


Figure 153: *'Achieve' action item, which was switched from a desired RMAG to the two elements that make up B-plane targeting. B-plane targeting is an ideal method to use for flybys. It consists of two vectors : 'T' (tangential) and 'R' (normal component to 'T'). Started off with a value of '0' but changed it to the value stated in the figure after the first run achieved it.*

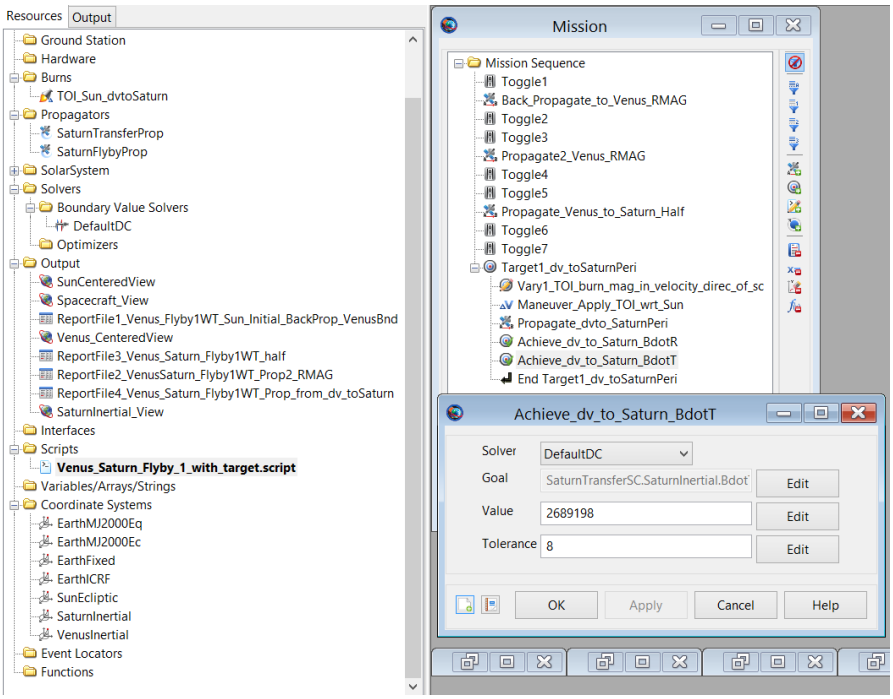
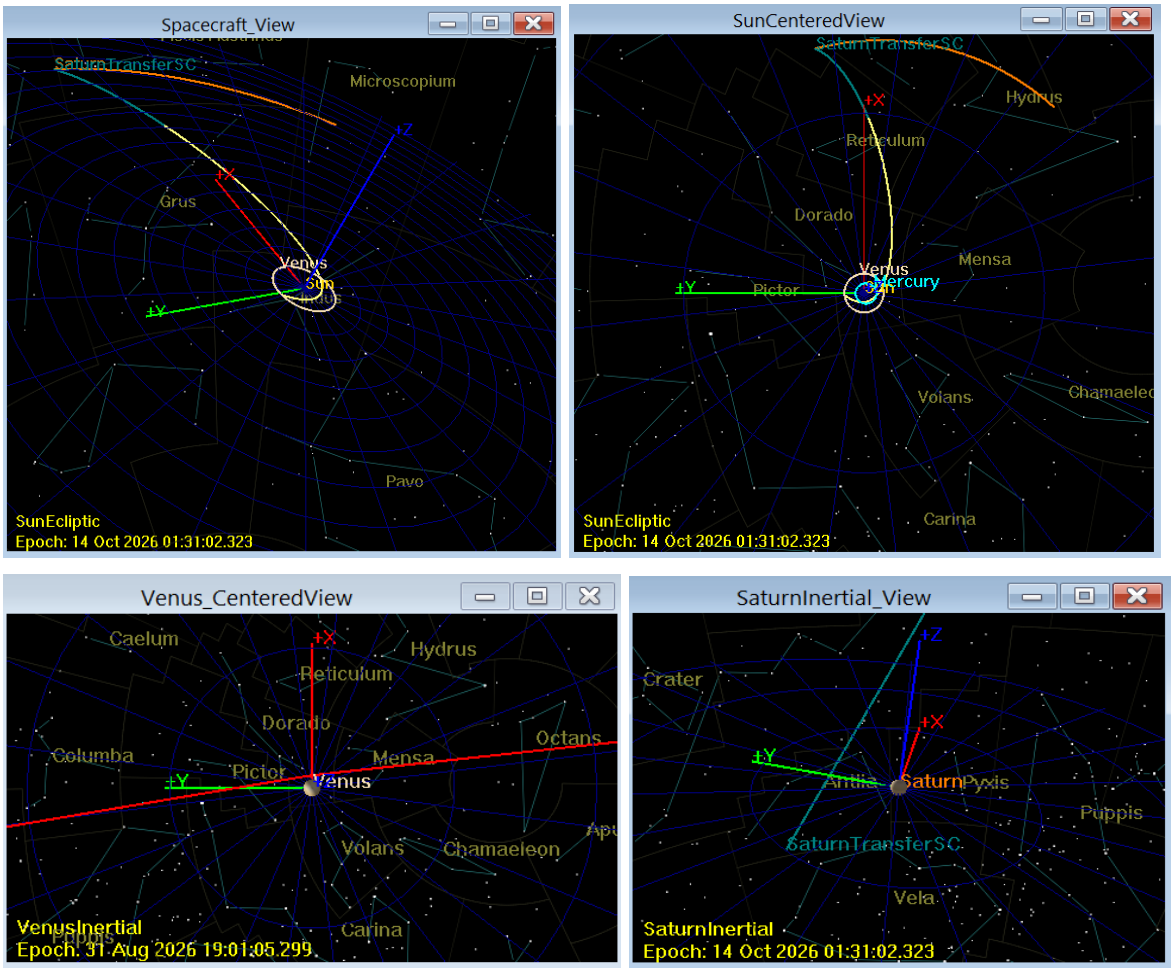


***Figure 154**: 'BdotT' component to be achieved in GMAT. Also, started off with a value of '0', then after the first run changed it to the value the solver has achieved.*

Below are the results of the perturbed swingby with a two year correctional burn setup. A no convergence warning is due to the large difference between the desired versus the achieved variable values that the solver calculates at the end of the prescribed iterations. If unsure what desired value is to be expected, the operator may leave it at '0' at the 'Achieve' setup, then use the value at what the solver achieves for the next run to get the solver to converge (within the same number of iterations). This method is acceptable only if the desired value is not critical to the analysis, as is such in this study; where the spacecraft is expected to achieve some arbitrary coordinate point along Saturn's periapsis. It is important to note that the 'Pos1' value calculated via Matlab was without burns considered. Lambert's problem gives the velocities that the spacecraft is expected to be at 'Pos1' and 'Pos2' only (it is burn and reference frame agnostic).



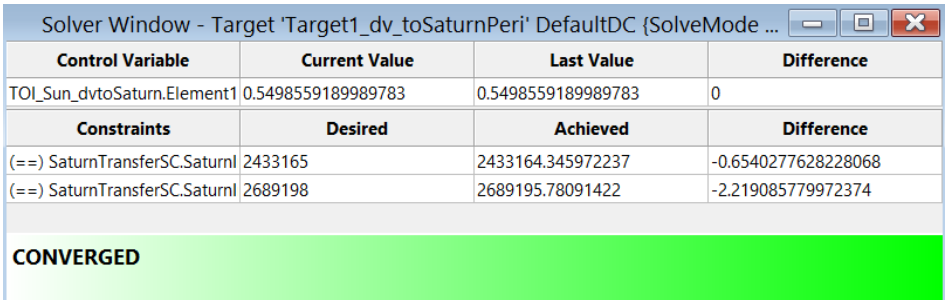


***Figure 155**: Orbital views of the perturbed target run in GMAT. The blue grid is the sun ecliptic plane (as initially defined under coordinate systems and when setting up the spacecraft under the 'Resources' tab). Note that elements 1, 2 and 3 (burn magnitudes in the X, Z and Y directions respectively) may be used in order for the spacecraft to attempt to achieve Saturn's periapsis. Note that the desired value of 2433165 km was found from initially running the setup with a desired value of '0' for both 'BdotT' and 'BdotR' 'Achieve' variables. Thus, allowing the solver to show at what values the spacecraft would most likely converge at. The Venus centered view captures the spacecraft's initial back and forward propagation to determine Venus' sphere of influence (one can also see it if zooms into the sun centered and spacecraft orbital views). Note the location of the trajectory in the Saturn inertial orbital view. The total or absolute burn magnitude is 0.55 km/s.*

The solver, for this problem, is adjusting the thrusters in the VNB (Velocity - Normal -Binormal) directions of the spacecraft, with respect to the non-inertial Sun's coordinate system (or reference frame). The total amount of iterations needed for the solver to calculate the final burns (thruster's coordinate system of X (velocity - forward or backward directions), Y (binormal - left or right) and Z (up or down) to reach an arbitrary point along Saturn's periapsis is dependent on the setup. It is a good rule of thumb to start low, such as 10 or 25 iterations, instead of the default 50. The solver arrived at a value of 2.7e6 km along the tangential component, 'T', and 2.4 e6 km along the 'R' component of the B-plane within six iterations. Furthermore, due to time constraint and solver being computational heavy, max solver iterations will be kept at 25 regardless if it converges or not. Note that only Element 1 (X directional thrust) was considered for this analysis due to it being along the direction of the spacecraft's velocity. However, the operator has the freedom to select any combination of the thrusters (i.e Elements 1 and 2 (X and Z) or Elements 1 and 3 (X and Y) or Elements 1, 2 and 3 (X, Z and Y). The type of combination of elements selected is dependent on the spacecraft's spatial location with respect to its target (i.e. what plane or thrust axis the spacecraft needs to be on). If the combination selected does not work with operating boundary conditions then a 'QNAN' error will be displayed at the end of the iterations and/or the spacecraft will be completely off from its intended target. To resolve that error one must change the 'Vary' limit conditions, change the burn conditions (if applicable for instance change VNB to LVLH [local vertical and local horizontal] or the central body or the coordinate system...etc.) under the 'Resources' tab, or the 'Achieve' conditions (for instance had to switch from RMAG [with respect to Saturn] to 'BdotT' and 'BdotR').

The purpose of this analysis, moving forward, is to compare different swingby trajectories' (starting at different initial conditions, with respect to velocity and position coordinates) correctional burns after the two year mark. More of the latter will be observed in section 5.1.2.

4.1.5 GMAT Setup for Data Used in NN Model

As discussed in section 4.1.3, since the unperturbed trajectory stopped at around 170 days, the perturbed data will also stop within the same time frame. Thus, this section will delve into the setup used in capturing data, perturbed (with Sun) and unperturbed (without Sun), for the NN model. JPL Horizons web page setup and Matlab (as shown in section 4.1.1) will be used for this analysis.

After a series of tests, it was clear that the simulation should run for 157 days and that a burn was not necessary (as was initially shown in section 4.1.3). The NN model requires three sets of data: training, validation and testing. Since Venus' orbital period was 225 days it was preferable to capture the ephemeris data after every 10 days. Thus, every run, with the NN model, will have three sets of data that have a time difference of 10 days with respect to the initial conditions (position and velocity).

The process, with data setup for the NN model, will be as follows:

- Use JPL Horizons web page to capture ephemeris data for both Venus and Saturn, at different time stamps (10 day difference).
- Insert the position component values of Venus, of day one, into the 'Pos1' variable in Matlab, and the last time stamp's position component values of Saturn's into the 'Pos2' variable in Matlab. (Also modify the comments and change script names accordingly.) The code will add in the randomly selected offsets along Venus' and Saturn's periapsis.
- Insert the 'Pos1' and the first row of values (which are the initial velocity components associated with 'Pos1') in the 'temp' variable (solved by Lambert's problem in Matlab) to GMAT's spacecraft setup.
- Run the GMAT script to backprop for 10 days. (This is consistent with the setting of the overall problem, where the spacecraft is in interplanetary transit before committing to the swingby trajectory about the Sun and Venus.)
- Then in two separate scripts (with and without Sun) use the position and velocity values at the end of the backprop and use it as the initial conditions for the spacecraft setup.
- The data is captured, split and titled appropriately to be fed to the NN model.

The figures below depict only one of the runs (the third ephemeris data set) of this repetitive process.

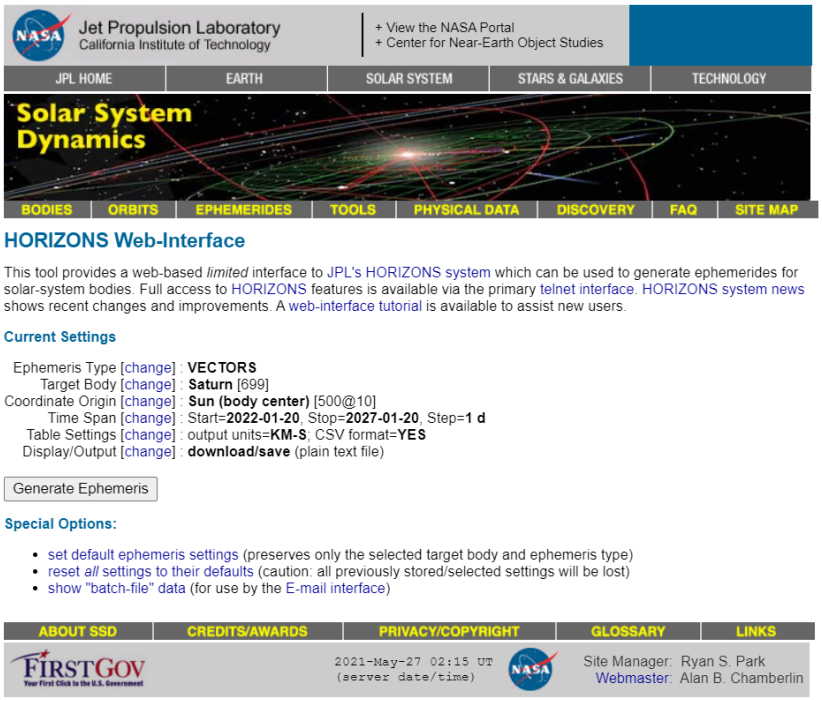


***Figure 156**: JPL Horizons web page complete setup for Venus' and Saturn's ephemeris data. Note that the date is set at 2022-01-20 instead of 2022-01-01 as stated in section 4.1.1. This is the third data set to be used for training the NN model. The 1st data set was between 2022-01-01 and 2027-01-01 and the second data set was between 2022-01-10 and 2022-01-10. Note that the only items that change are the planet (in 'Target Body') and the time span.*

Once the excel files are downloaded with the ephemeris data, then the first time stamp of Venus' and the last time stamp of Saturn's (from each data set) is used for the Lambert's problem in Matlab ('Pos1' and 'Pos2' respectively). Keep in mind that one script must be created for each data set, thus for this first run, there are a total of three Matlab scripts. Please, refer to section 4.1.1 where Lambert's problem in Matlab has been discussed. The only objects that will be modified are the three values within the array for 'Pos1' and 'Pos2' of that script (besides the obvious comments and title of the saved file).

Next, in GMAT, a backprop script for each data set is set up to find the actual initial conditions of the spacecraft before it enters the sphere of influence of Venus. However, due to time constraints, only one of the runs will be discussed throughout this section.

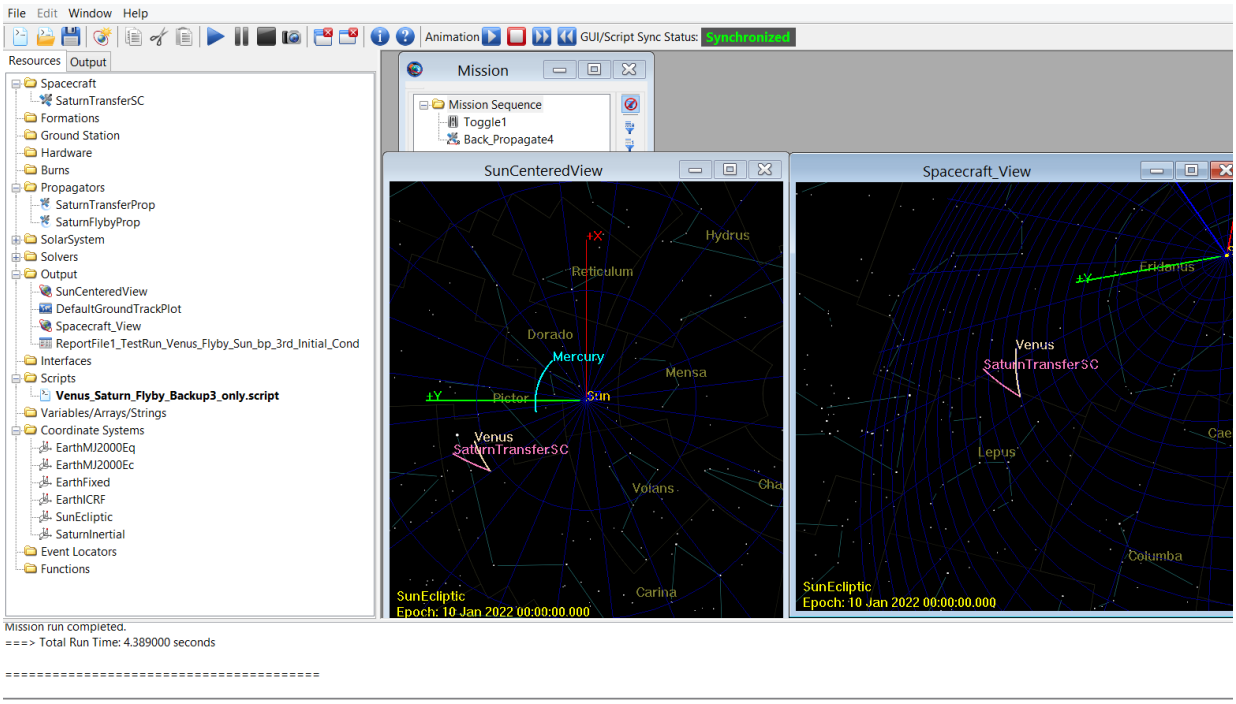


***Figure 157**: Backpropagation for the third data set (pertaining to the ephemeris dates of 2022-01-20 to 2027-01-20). Sun is used for this part of the analysis for consistency sake, and is selected as a point mass in the 'SaturnTransferProp' under the 'Propagators' branch in the 'Resources' tab. The 'Toggle1' has the 'ReportFile1', sun centered and spacecraft orbital views turned on. 'Back_Propagate4' uses the 'SaturnTransferProp' as its propagator, the back propagation option is selected, 'Elapsed Days' is selected as the parameter and its associated condition is '-10'.*

The final position and velocity component values, from the back propagation run, are used in another GMAT script's spacecraft's initial conditions; for which its data will be used for the NN model training. There will be two scripts for each of the backprops' final values - with and without Sun's influence. Thus, a total of six runs will have been completed for the first training of the NN model.

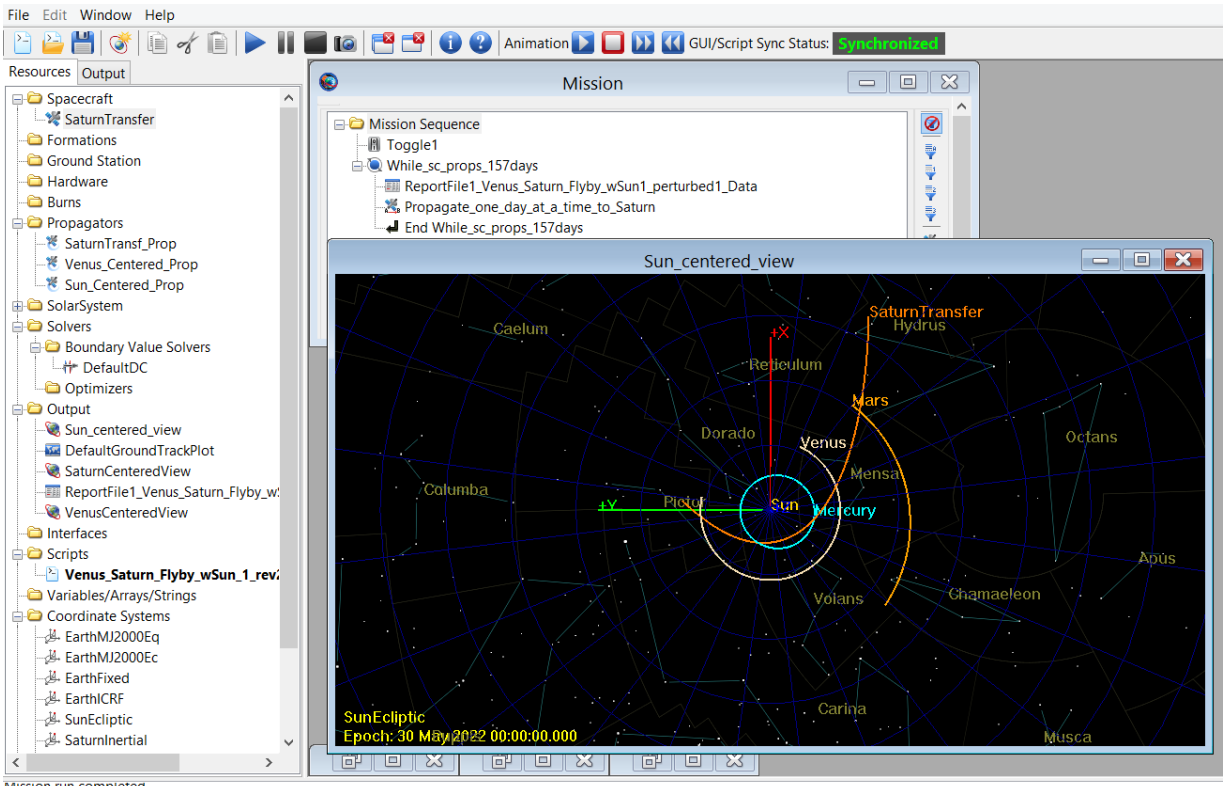


***Figure 158**: Propagation for 157 days, with Sun and Venus, in GMAT. Propagator selected is the 'Venus_Centered_Prop'. Note the control logic (while loop) used in the mission sequence. The condition statement has 'Elapsed Days < 157 days' and within it has the output report file and the propagation set at 'Elapsed Days = 1'. Thus, each day's velocity and position component values are documented in the output file for 157 days. Note that the operator must specify what the 'ReportFile' must have as outputs (as shown in section 4.1.2).*

Don't forget to change the associated date with the initial conditions at the end of the 10 day back-prop. For example: After back-propping from 01 Jan 2022 for 10 days, the end date is 22 Dec 2021. The latter will be the start date (epoch) for the initial conditions of the spacecraft in the script presented in Figures 158 and 159.

The same is done for the simulation without the Sun, but the only difference is unselecting the Sun as a point mass (but keeping Venus) in the 'Venus_Centered_Prop' propagator.

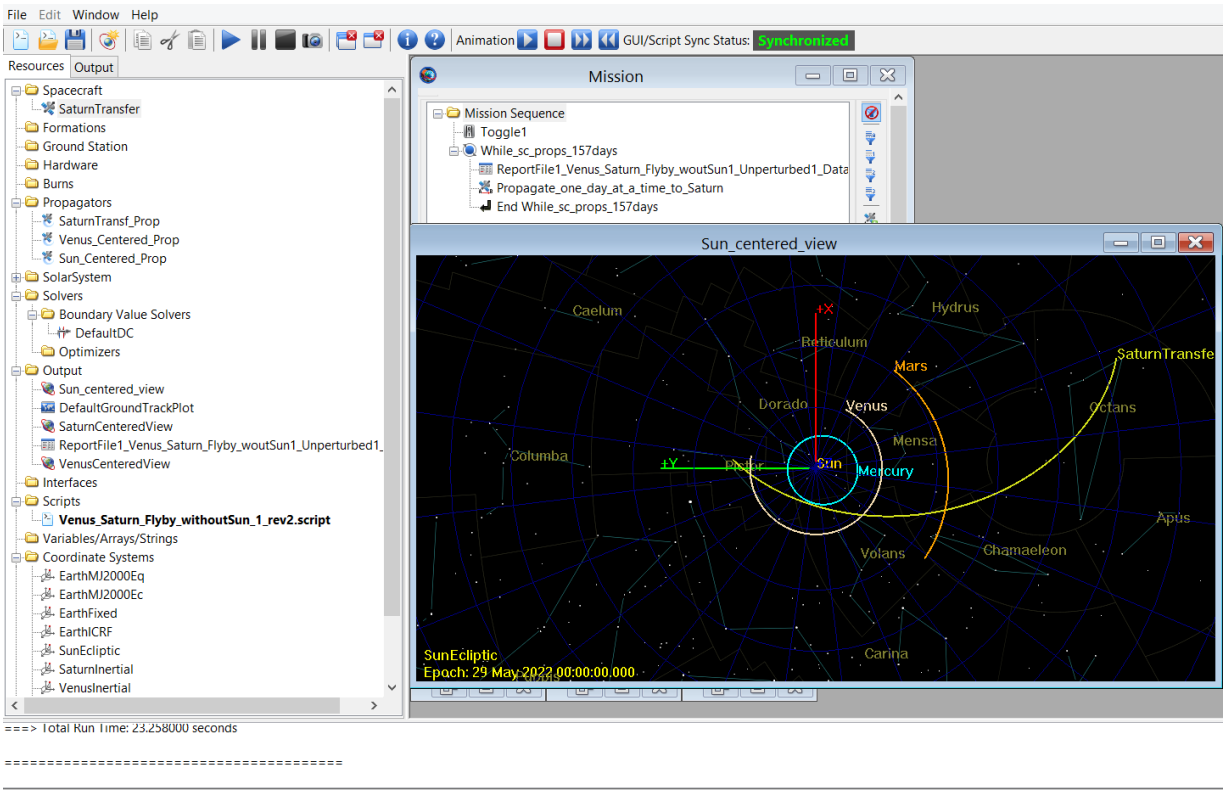


***Figure 159**: Third simulation run without Sun in GMAT. This is after using JPL Horizons web page to gather ephemeris data on 2022-01-20 to 2027-01-20 (for both Venus and Saturn), incorporating Lambert's solver in Matlab, and, eventually, backpropagating (for 10 days) in GMAT to gather the actual initial conditions for this simulation. Note that the spacecraft starts at some point outside the sphere of influence of Venus. Note the animation panel in the taskbar above - can be used to run the trajectory in real time after the simulation run is complete.*

Once all six simulation runs are complete (with and without Sun simulation runs for each of the three different initial conditions with respect to the ephemeris data), it is time for data pre-processing and NN training.

Chapter 5 - NN Training and Burn Analysis

5.1 GMAT Data and NN Setup and Training

5.1.1 Background

Chapter 5 presents the revised setup of the NN model prematurely established in Chapter 3, section 3.3.1. The final setup of the NN model was divided into the following sections (also visibly structured as such in the .ipynb files.):

- Importing all necessary libraries
- Mounting Google Colab to Google Drive to upload data (data was stored as .csv files in Google Drive's folders)
- Describing and visualizing data (used Panda and plotting libraries)
- Splitting data between input(X) and output(Y). (There are three datasets, so each was split accordingly to represent : unperturbed (X) and perturbed (Y).)
- Normalizing (or scaling, depending on application) data's features. (There are many techniques so choose with respect to whether or not covariance of each feature must be preserved and/or whether or not a gaussian distribution is necessary).
- Reshaping the split data from 2D to 3D (due to LSTM layers in model).
- Creating, training, and evaluating the model.
- Interpreting the weights
- Plotting the weights
- Plotting the perturbations (for both position and velocity)

Chapter 5 includes a more thorough discussion of scaling and normalizing data, as well as the optimization techniques used to enhance the overall performance of the LSTM model. For example, due to contradicting articles, there was a misunderstanding that there was no difference between normalization and scaling of data. A thorough description is provided in section 5.1.2, of how based on data visualization and whether preserving co-variance is critical, can one select between the two options. Another misunderstanding was the irreproducibility of the NN model using a Keras wrapper. This is because Keras's backend engine (Tensorflow) has a random seed initializer that affects the training process of the kernels (weights). Initially, it caused a lot of frustration and anxiety but once it was understood that as long as the model retained 90th percentile accuracy for training, validation and testing, a precise and robust model was preferred. Using an API, like Keras, may seem convenient at first when starting out, but proves to be constraining depending on the problem. For instance, a sequential model lacks the flexibility that a functional model has (also further discussed in section 5.1.2), and stating the activation functions of an LSTM model only affects the input gates. Fortunately, other means were discovered to improve the performance results of the NN model, but such considerations were stated under section 6.1.2 as future works.

Chapter 3 provided an initial setup of the model, however many changes have been done due to the further understanding of certain concepts.

5.1.2 Modifications and Considerations done for NN model

Many changes have been made since the first trial run in the last section of chapter 3; they are as follows:

- The addition of data visualization techniques (to better understand whether the data has a Gaussian shape or not). Skewed and Gaussian distributions are analyzed via the central tendency and median, respectively. Most ML algorithms and NNs prefer data to be Gaussian distributed. It is also a good indicator on which normalization or scaling method to use based on how the data's spread displays after the use of such tools.
- Introduced accuracy for the regression model. (Using the default accuracy in Keras; which is fine for both categorical or regression based data. This is determined by investigating its formulation.)
- Tested various optimizers (Adam, SGD - with momentum and velocity aspects, cosine decay, exponential decay and inverse time decay) and found it was effective to stick with Adam. Adam combines RMSProp, Momentum, AdaGrad, and bias corrections to prevent the convergence of the learning model to a local minima instead of the desired minima. Furthermore, the conclusion of the tests proved that the optimizer did not significantly improve the performance of the model, but rather introducing a drop rate regularizer, reshaping the data (one-to-one versus many-to-many), introducing a 'Time-Distributed' layer, and reducing the number of LSTM stacked layers and cells did. The later sections of this chapter will look into the performance of the model with different conditions and when more features are added.
- Keras/Tensorflow algorithms have random seed initialization for the weights after resetting the Jupyter notebook. Thus, every run, despite the same conditions, would produce a slightly different variation than its predecessor. Since a robust model is preferred, it was kept as is.
- Scaling the data between -1 and 1 instead of between 0 and 1. (This is because the LSTM's forget gate will toss values that render the weights 0; for they are perceived as uncritical to the overall learning process. Also, the raw values of positions, and velocities, do go in the negative range.)
- Normalizing the data instead of just scaling it. (It was initially perceived that the formulation used for the 'Normalization' section of the first code, shown in section 3.3.1, was for normalization. This confusion was due to the fact that oftentimes in mathematics, normalization and scaling formulations were used interchangeably. Oftentimes, certain ML articles would place normalization and scalers under one umbrella called: 'Scaling Tools'. However, in data science and statistics, they are different and are selected per the data's visuals. For instance, if the data had a Gaussian shape, then a normalization technique was used. However, in this study, regardless of the data's shape, it will end up normalized. This is because the kernel (weight) initializers are Gaussian shaped and co-variance within each feature needs to be preserved. One of the sklearn tools for centralized 'feature scaling' is normalization via Z-score (or standard score) [39]. Popular

forms of feature scaling data, from sklearn library, are: StandardScaler (or standard normalization per feature), MinMaxScaler (scales every feature between any specified range - usually (0,1) and (-1,1)) and RobustScaler. Furthermore, normalization changes the overall shape of the data, while scaling it adjusts the range of the domain (and dataset). Features that contain a higher magnitude of variance tend to dominate the learning process; hence, it is ideal to ensure that all the features' covariance is preserved even when scaled within a range of (-1,1). Regularizers and initializers within any ML package assume that the data given has undergone some form of normalization or scaling in order to speed up the convergence and learning process. Due to the activation functions used (especially since LSTM gates have tanh and sigmoid functions), the maximum range (of the normalized values) acceptable is between -1 and 1. The basic forms of scaling and normalization are:

- Standardization (StandardScalar): feature based sklearn tool and it normalizes with respect to a data's z-score. Also, called standard normalization and seeks to centralize the data set (mean = 0 and unit variance). Thus, the original variance of the dataset may be altered.
  - Formula: $x' = \frac{x - \bar{x}}{\sigma}$ (***25)*** [40]
- Mean normalization : Data distribution will be transformed to a gaussian distribution and a range of (-1, 1). The covariance of the features is preserved.
  - Formula : $x' = \frac{x - \bar{x}}{xmin - xmax}$ (***26)***
- Min-max scaling (MinMaxScaler) : feature based scaling sklearn tool, where the feature range is flexible (such as between [0,1] or [-1,1]) and covariance is not preserved.
  - Formula : $x' = \frac{x - xmin}{xmin - xmax}$ (***27)***
- Unit vector : scaling done with respect to a feature's unit length and covariance is preserved.
  - Formula : $x' = \frac{x}{||x||}$ (***28)***

- Instead of dividing a whole data set between just tests and training with the scikit-learn tools (as was shown in section 3.3.1). Three separate data sets for all: training, validation and testing are used. This is to ensure the same level of complexity in each, thus improving the overall performance of the model.
- Initially it was presumed that, due to the seq2seq comparisons between input training and label (output training), and the result of the MinMaxScalar giving equal values (per feature) at certain time steps, that normalizing the whole dataset was the ideal solution. However, upon visualizing the different scaling and normalization techniques, it became clear that per feature normalization was the correct method to use. Afterall, the central

tendency of each feature must be preserved to render a more accurate 'understanding' from the NN model.

- May have to forgo the 'relu' (or 'leaky relu') activation functions and replace them with the default activations of LSTMs or using 'tanh' instead. The 'activation' and 'recurrent_activation' parameters are the input gates' activation functions. There is a concern that the 'relu' and 'leaky relu' activations will explode or vanish the input values at an accelerated pace. Thus, preventing the LSTM model from actually learning. Furthermore, there is a growing uncertainty on whether or not the LSTM input gates change to 'relu' when using Keras to modify their activations (remember there are four activation gates total but two are used simultaneously as 'inputs', and the other two are 'forget' and 'output'). This will be further explored in the chapter.
- Increasing the number of features eventually does increase the performance of the NN model. Thus, may have to update the number of features from three (position components) to six (velocity and position components).
- May or may not end up using autoencoders to capture the 'learned representation' of the solar perturbations.
- Overfitting is also where the NN model successfully learns the pattern of a particular dataset and would use it as a reference when exposed to other data. This should be avoided since the NN model is expected to be flexible and have a more generalized and continuous approach when exposed to different types of data.
- Can only use the 'stateful' or 'return_state' attributes of a LSTM layer when using a functional model setup (not a sequential - as was done for this project). This is because the sequential model only accepts single array outputs in between layers - not multiple. The 'return_sequences' may be used for a sequential model.
- Adding a layer in a model impacts the learning rate and batch size for training.
- The basic steps to prevent overfitting (training loss much larger than the validation loss) and underfitting (validation accuracy higher than training accuracy) are the following:
    - Increase the number of data points (for all training, testing and validation).
    - Decrease the number of layers and total number of cells per layer (start small first).
    - Modify the data structure. For instance, since LSTM is used, the data must be represented in 3D (number of samples, number of timesteps, number of features). However, when dealing with prediction time series based data there are different ways in which the data is structured that controls when and how the NN is expected to predict its outputs. Those ways are: one-to-one, one-to-many, many-to-one and many-to-many [41]. Due to the context of the data, the only options are : one to one and many to many. Each time step has its associated position and velocity vectors, and are dependent on one another. One-to-one is where every sample of input and output will only have one time step (for predictive comparisons), and many-to-many is where every sample, for input and

output, has many time steps. (The terms input and output used in this paragraph are X and Y, respectively; where X is the input and Y is the label or desired value within all training, validation and test datasets.) In section 3.3.1 the training sample shown used one-to-one. This section, as well as the upcoming ones, will be using many-to-many. This is because the NN must understand that the time steps are dependent on one another in every iteration (back + forward propagation) it completes for every epoch. Thus, the flow of information will flow across the time steps instead of just understanding one particular time step at every epoch. The latter is the nature of what a one-to-one setup is, while the former is the many-to-many.

- Modify the learning rate and/or optimizer selected. (Rule of thumb, start off with Adam (as the optimizer) and a learning rate of 0.001 or 0.0009). Increase or decrease the learning rate when the training loss curves are linear or plateaus early, respectively.
- Increase or decrease the batch size.
- Introduce a regularizer. There are many to choose from so select one based on the type of NN model. LSTM is this study's model, so drop out rate and kernel regularizers within the LSTM layers can be used. However, start slow by introducing one layer of drop out and adjust its hyperparameter. Furthermore, the forget gate in the LSTM cell behaves as a regularizer.

5.1.3 GMAT Data Preparation

This section will show how the data looks before being fed to the NN model. The integrated development environment (IDE), in which the Keras/Tensorflow packages are being used, is Jupyter Notebooks via Google Colabs. Thus, all the data and the code used to establish the NN model and its training are hosted within one's own Google Drive. Google Colab is a feature of Google Drive and uses Jupyter Notebooks as its IDE.

Note that there are three different sets of GMAT data that will be fed to the NN model (via Google Colab and Google Drive). Each data set has perturbed and unperturbed values for all cartesian components for position and velocity. This setup is done in this manner in hopes of retrieving the perturbation values indirectly. The while loop used in the mission sequence in section 4.1.5 ensures that data collected in equal time steps (which is important in LSTM models).

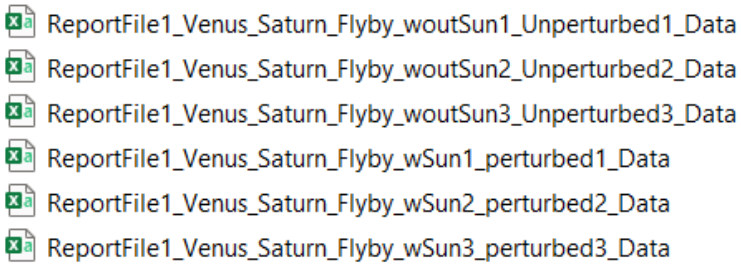


***Figure 160**: Files found in the 'outputs' folder after being created in a GMAT script. These output files can be found under the 'GMAT' folder on one's PC. The GMAT scripts can be found under the 'Reports' folder in the 'GMAT' directory. Note that there are six files for the three time stamps selected with respect to the ephemeris data.*

Each file will have all six features: X-position, Y-position, Z-position, X-velocity, Y-velocity, and Z-velocity, as well as time. In the earlier figures for GMAT setup, throughout chapter 4, the time parameter, UTC Gregorian, was added to the list of outputs within the report. The time parameter ensures that the timesteps are consistent and equal across all data sets. The differences between the vector components will be based on its initial conditions (of their associated GMAT script), and whether or not the Sun's influence is present. The training, validation and testing data sets require an input (unperturbed) and output (perturbed) for every time step (in an LSTM time series sequence to sequence prediction NN modeling). Thus, three data sets that have unperturbed and perturbed values for each time step will be uploaded to Google Colab.

Initially, due to the fear of exploding and vanishing gradients, position and velocity components were trained separately (as shown in section 3.3.1). However, it became clear that the model requires more training data, as well as features to improve performance. Also, normalization techniques, as well as LSTM cell gates and its memory (c-channel) and hidden (h-channel) states prevent the occurrence of exploding and vanishing weights.

Yet, for consistency's sake, a comparison with respect to the NN model's performance when training with position only versus position and velocity will be done. All files are saved and uploaded (to a folder in Google Drive) in .csv format, and the Google Colab files (.ipynb) are saved in a separate folder. It is important to have only folders at the opening home page of one's Google Drive; it helps to reduce complications, in data retrieval, when running the NN model.

For simplicity's sake, it is advisable to use the same number of time steps for unperturbed and perturbed data sets. Next, each initial condition, with respect to the ephemeris data, will have its own .csv file of perturbed and unperturbed data. Thus, there will be a total of three datasets (for every NN run) to accommodate the required training, validation and testing phases of the learning process.

	A	B	C	D	E	F
1	UX	UY	UZ	PX	PY	PZ
2	-47921190.25	120244160.9	4663872.233	-47921190.25	120244160.9	4663872.23
3	-49595395.04	117146385.9	4693616.272	-49609785.18	117153054.3	4694482.60
4	-51217043.63	113980964.2	4719398.909	-51274549.12	114005697.1	4722838.37
5	-52784240.92	110749415	4741131.645	-52913350.17	110800640.2	4748803.2
6	-54295136.09	107453311.3	4758729.282	-54523882.71	107536417.1	4772231.59
7	-55747924.14	104094278.9	4772109.984	-56103652.63	104211551.1	4792967.98
8	-57140847.45	100673994.5	4781195.353	-57649961.55	100824562.7	4810846.36
9	-58472197.22	97194184.81	4785910.495	-59159889.87	97373980.27	4825689.39
10	-59740314.96	93656624.72	4786184.078	-60630278.89	93858352.44	4837307.8
11	-60943593.81	90063135.97	4781948.396	-62057711.95	90276262.88	4845499.76
12	-62080479.96	86415585.55	4773139.425	-63438494.78	86626348.57	4850049.92
13	-63149473.91	82715884.09	4759696.87	-64768635.23	82907321.51	4850729.09
14	-64149131.67	78965984.24	4741564.222	-66043822.87	79117994.62	4847293.46
15	-65078066.05	75167879.01	4718688.8	-67259408.69	75257312.38	4839484.21

***Figure 161**: Excel sample image of the third dataset to be used for NN training. Only the first 15 points are shown. Note the labeling of (UX = Unperturbed in the X position, UY = Unperturbed in the Y position, and UZ = Unperturbed in the Z position, PX = Perturbed in the X position, PY = Perturbed in the Y position and PZ = Perturbed in the Z position) the perturbed and unperturbed based data for positions only.*

Next, save the files in .csv format and upload them to a folder in Google Drive. Create a folder for the Google Colab notebooks, and import all necessary libraries. Before running the code make sure that the hardware accelerator selected, under ‘Notebook Settings’ and ‘Edit’ tab, is ‘GPU’ and not ‘CPU’. This allows the operator to use the cloud’s GPU to run the training of the NN model. After a run is complete, an operator can select ‘Factory Reset Runtime’ or ‘Restart Runtime’, under the tab ‘Runtime’, and reload/refresh the page to erase the previous run’s data. Also, before refreshing/reloading the page, it is advisable to access one’s browser’s settings and delete all cache and cookies (found under security). This technique starts over the training with a different random seed initializer, and prevents interference from the previous run’s results.

There are times, due to high traffic in the cloud, that Google Colab cannot use a GPU on standby. In such a case, it is acceptable to allow the Jupyter notebook to select another source by default or one can select ‘TPU’ instead under the ‘Notebook Settings’.

The full code can be viewed in Appendix B.

5.1.4 Data Visualization of the Data Frame

Pandas, seaborn, plotly, and matplotlib are some of the main libraries used to visualize any parts of the data. Visualizing the training and test datasets determine whether or not the data has a normal (Gaussian) distribution or not. If the training or testing data sets have a normal distribution then use the standardization method if not, then scaling the data (such as the

MinMaxScalar method) is advisable. However, the only way to determine which method works best is to test using all applicable methods and view how each affects the data's distribution. The latter can be standardized but may not produce reliable results [42]. Combining scaling and normalization of a dataset indicates that whichever process was last would dominate the overall shape of the data's distribution. Sometimes certain features will have higher variance than others, and those will dominate the training process due to higher valued weights. For this study, the data's spread is important and must be preserved. Thus, normalization, specifically either mean or unit norm normalization became the ideal choices [43].

```
fig1, axes = plt.subplots(1,6, figsize=(8,2))
#sns.barplot(data=df['UX'], ax=axes[0] )
sns.histplot(data=df['UX'], ax = axes[0], kde = True)
sns.histplot(data=df['UY'], ax = axes[1], kde = True, color = 'sandybrown')
sns.histplot(data=df['UZ'], ax = axes[2], kde = True, color = 'g')
sns.histplot(data=df['PX'], ax = axes[3], kde = True, color = 'lightcoral')
sns.histplot(data=df['PY'], ax = axes[4], kde = True, color = 'mediumpurple')
sns.histplot(data=df['PZ'], ax = axes[5], kde = True, color = 'rosybrown')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49b0a31490>
```

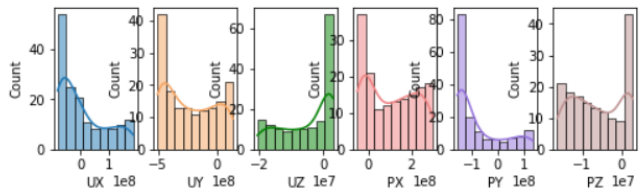


***Figure 162**: Training dataset distribution of every feature using Seaborn library. Note the skewed behavior of the features.*

```
sns.displot(df, kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f49be3a8050>
```

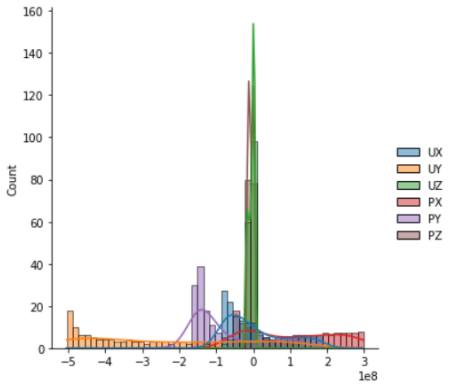


***Figure 163**: Overall distribution of all the features in the training dataset, also using seaborn library.*

```
fig4 = plt.figure()
ax4 = fig4.gca(projection = '3d')
ax4.plot(df['UX'], df['UY'], df['UZ'])
ax4.plot(df['PX'], df['PY'], df['PZ'])

ax4.set_xlabel('x')
ax4.set_ylabel('y')
ax4.set_zlabel('z')

plt.show()
```

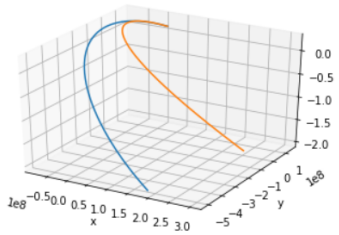


Figure 164*: 3-D plotting of the training dataset using matplotlib. The blue and orange curves are the unperturbed and perturbed trajectories, respectively.*

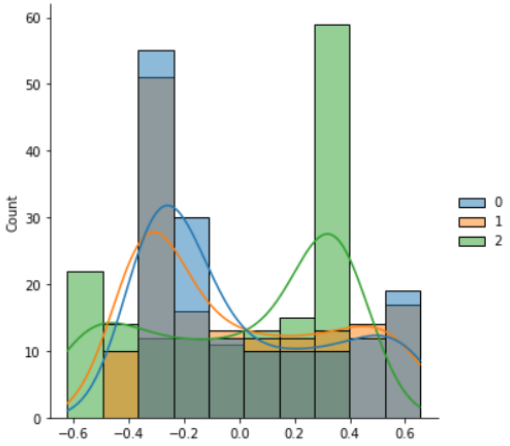
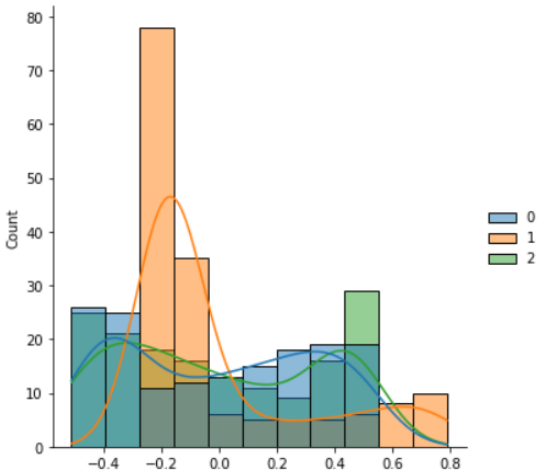
5.1.5 Data Scaling and Normalization

The sklearn libraries are used to normalize and scale features of any numerical based dataset (keep in mind that an ‘axis = 0’ are the columns/features and an ‘axis = 1’ are the rows/samples - only in this context). Matrices or multi-dimensional arrays and Pandas dataframes have indices (index per row) and so do lists (non-homogenic 1-D set of data). Indices are what is used in ‘for’ loops. Scaling and normalizing data speeds up the learning process (digestible to the learning model) and promotes convergence. The tricky part is figuring out what and how to apply the scale and/or normalization tools. This dataset is unique because the unperturbed components (UX, UY, UZ, UVX, UVY, and UVZ) are interdependent on one another, and same with the perturbed components (PX, PY, PZ, PVX, PVY, and PVZ). The unperturbed and perturbed components are input(X) and output/label(Y), respectively. Initially, the MinMaxScalar() was used, which scales each feature (column) of the given dataframe, without normalizing the scaled data (which was a mistake). Furthermore, it is advisable to not normalize and scale any whole dataset before splitting them. For this analysis, there are three datasets : training, validation and testing, and each was mean-normalized per feature after splitting the data between input(X) and output/label(Y). The visuals are shown below in Table 2.

The results for each can be viewed after running code, shown in Appendix B, in a Jupyter Notebook/Google Colab platform.

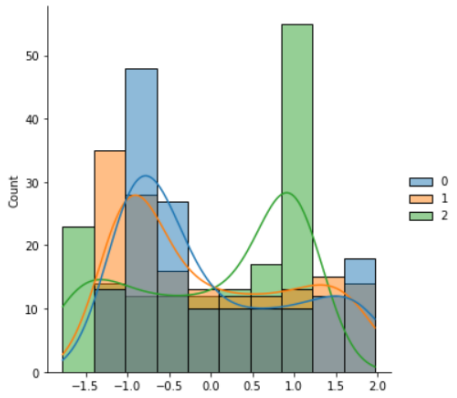
Table 5.2. Types of scaling and normalization methods and sample plots.

Plots	Notes

`sns.displot(mnm_X1, kde=True)`
`<seaborn.axisgrid.FacetGrid at 0x7f830c754090>`
 
`sns.displot(mnm_Y1, kde=True)`
`<seaborn.axisgrid.FacetGrid at 0x7f830c862f90>`
 
	• X1 (input training data)
	• Normalization
	• Method: Mean normalization
	• Not provided by sklearn
	• Bi-modal behavior
	• Range is within (-1, 1)
	• Co-variance is preserved
	• Optimal choice to use
	
	• Y1 (output training data)
	• Normalization
	• Method: Mean normalization
	• Not provided by sklearn
	• Bi-modal behavior
	• Range is within (-1, 1)
	• Co-variance is preserved
	• Optimal choice to use

```
sns.displot(scaler_X1, kde = True)
```

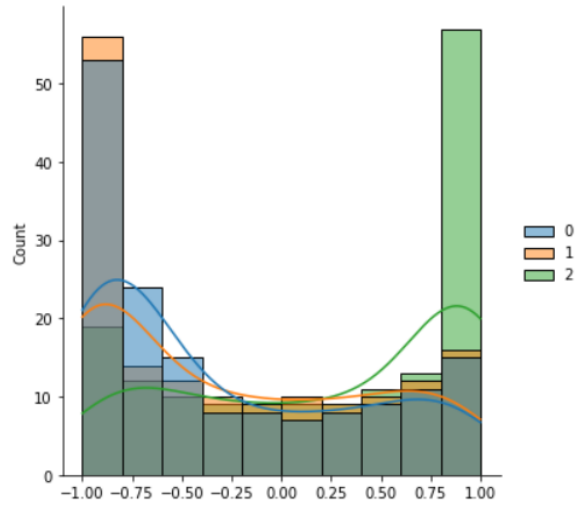
```
<seaborn.axisgrid.FacetGrid at 0x7f830ce19e50>
```



- X1(input training data)
- Scaler
- Method: Scale
- Offered by sklearn
- Range is higher than (-1, 1)
- Bi-modal behavior
- Co-variance has been modified
- Not an optimal choice

```
sns.displot(mmscaled_X1, kde = True)
```

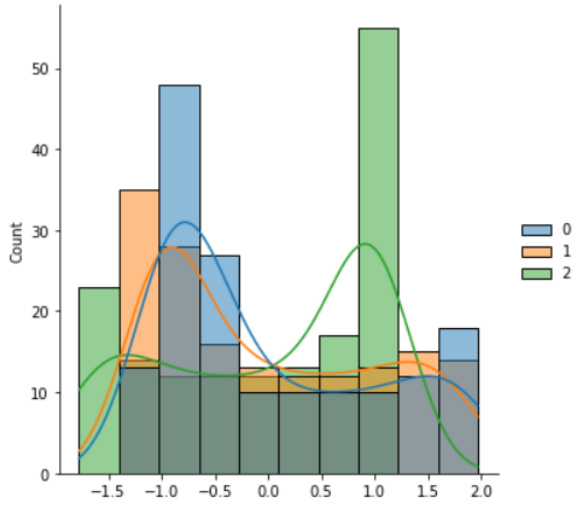
```
<seaborn.axisgrid.FacetGrid at 0x7f830c5356d0>
```



- X1 (input training data)
- Scaler
- Method: MinMaxScalar
- Offered by sklearn
- Range is set at (-1, 1)
- Co-variance has been modified
- Bi-modal behavior (but more skewed than others)
- Not an optimal choice, despite being within the range due to having equal values to that of some of the features in Y1(output training data). This is due to its formulation.

```
sns.displot(scaler_stdnorm_X1, kde = True)
```

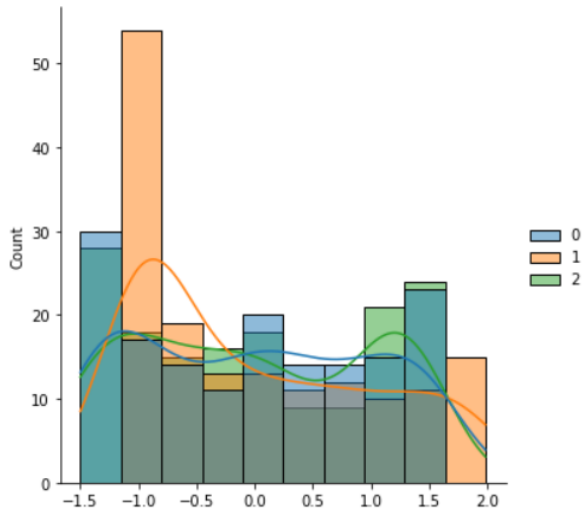
```
<seaborn.axisgrid.FacetGrid at 0x7f830c335290>
```



- X1 (input training data)
- Normalization
- Method: StandardScalar
- Offered by sklearn
- Bi-modal behavior
- Co-variance is semi-preserved
- Range is outside (-1, 1)
- Not an optimal choice

```
sns.displot(pt_X1, kde = True)
```

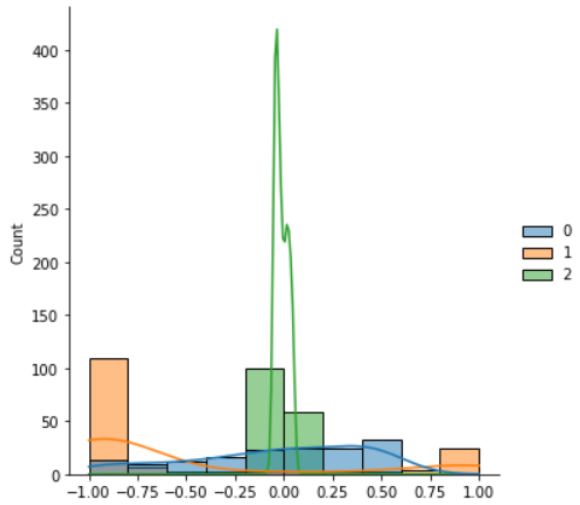
```
<seaborn.axisgrid.FacetGrid at 0x7f830c408650>
```



- X1 (input training data)
- Normalization
- Method: PowerTransformer (‘yeo-johnson’ and not ‘box-cox’ because the former accepts negative and positive values, while the other only accepts positive data.)
- Offered by sklearn
- Bi-modal behavior
- Co-variance is semi-preserved (since it has a standardization attribute that is set to ‘True’ by default.)
- Range is outside (-1, 1)
- Not optimal choice

```
sns.displot(norm_X1, kde = True)
```

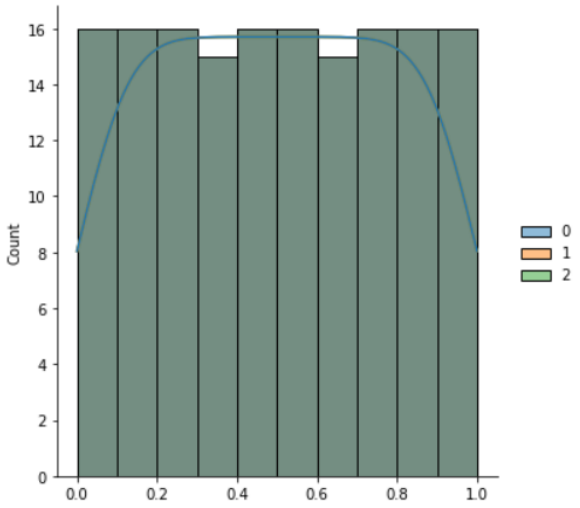
```
<seaborn.axisgrid.FacetGrid at 0x7f830bf306d0>
```



- X1 (input training data)
- Normalization
- Method: Normalizer
- Offered by sklearn
- Normalizes only the samples (rows) - but study needs normalization per feature (columns).
- Range is within (-1,1)
- Not optimal choice

```
sns.displot(qt_X1, kde = True)
```

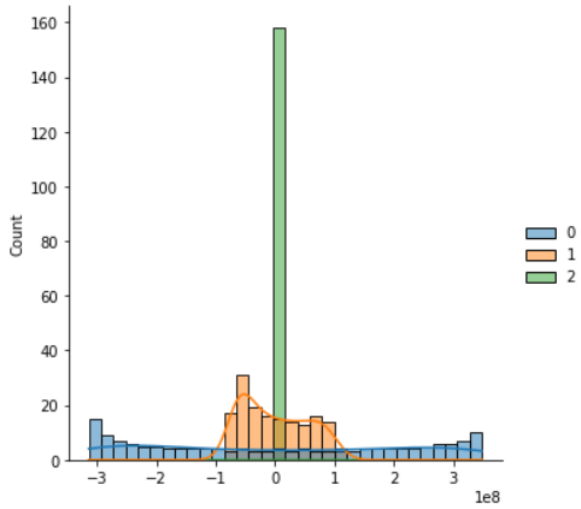
```
<seaborn.axisgrid.FacetGrid at 0x7f830bda0250>
```



- X1 (input training data)
- Normalization
- Method: QuantileTransformer
- Offered by sklearn
- Range is (0,1)
- Uniform distribution for all features
- Not an optimal choice.

```
sns.displot(pca_X1, kde = True)
```

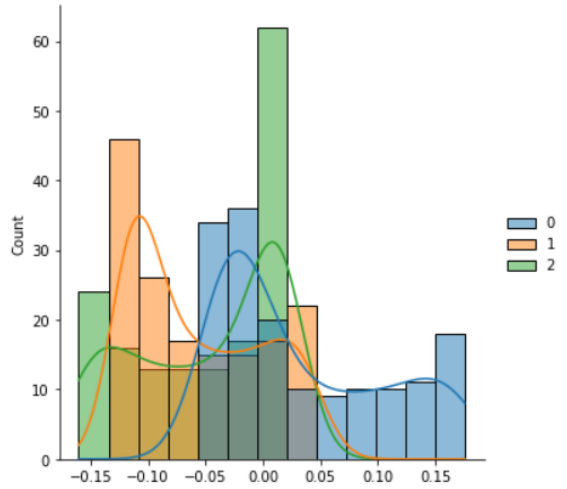
```
<seaborn.axisgrid.FacetGrid at 0x7f4968a3a6d0>
```



- X1 (input training data)
- Normalization
- Method: PCA
- Also called ‘data whitening’ method, for it centralizes data but loses the variance distribution (spread) of the overall data. It erases outliers.
- Can be computational heavy and take up a lot of RAM space.
- Offered by sklearn
- Range is outside (-1,1)
- Uniform distribution for all features
- Not an optimal choice.

```
sns.displot(normer_X1, kde = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f830bbe6b10>
```

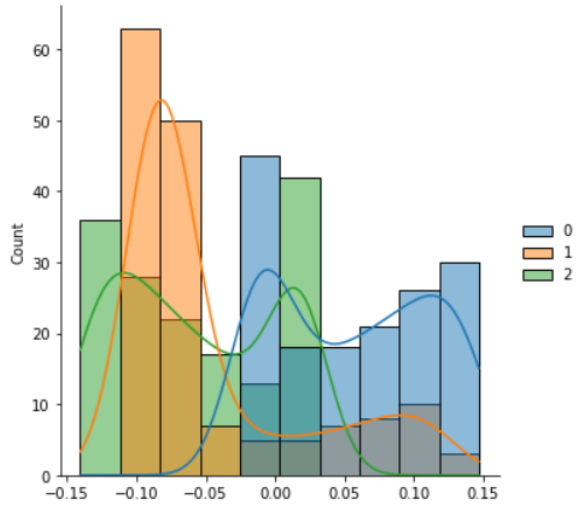


- X1 (input training data)
- Normalization
- Method: Normalize
- Also called ‘unit normalization’ or ‘unit norm’ in statistics.
- Offered by sklearn
- Range is outside (-1,1)
- Bi-modal behavior
- Co-variance is preserved
- Is an optimal choice. (But will keep it in mind for future works).

- Y1 (output training data)
- Normalization
- Method: Normalize
- Also, called ‘unit norm’ by staticians.
- Offered by sklearn
- Bi-modal behavior
- Range is within (-1, 1)
- Co-variance is preserved
- Optimal choice to use (But will keep it in mind for future works).

```
sns.displot(normer_Y1, kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f830bb22810>
```



Mean normalization is the method used to normalize the datasets. Input and outputs must be split before normalizing. Since, the sklearn library did not offer a mean normalization equivalent, a 'for loop' was implemented in code to ease the per feature normalization process.

The whole code can be viewed in Appendix B.

5.1.6 NN Model and Training Results of Three Features

The data, after normalization, must be reshaped from 2D to 3D. The LSTM model only accepts 3D shaped data.

```
norm_X1_in_train=np.resize(mnm_X1,(79,2,3))  #Only resizing the X normalized training parts (samples, time step, features)
norm_X1_in_train
```

***Figure 165**: Sample of code for shaping data for LSTM consumption. This is done in GoogleColab/Jupyter Notebook IDE. The data has been divided into 79 samples. Each sample has 2 time steps. And each time step has 3 features.*

The activation parameters in the NN model were changed to the 'tanh' (hyperbolic tangent). This is because after every epoch, the training loss values would fluctuate between high and low values (i.e. 600 to 0.005). Thus, proving that the 'relu' (and most likely 'leaky relu' as well) would cause the gradients to explode and vanish at an accelerated pace and overfitting to occur.

```
#lstm
#def create_model(): -this is for the functional model-
#using the sequential model instead
model=keras.models.Sequential([
                                #there's a diff btwn LSTMCell() and LSTM(), the latter uses CuDNN whilst the other doesn't.
            #1st layer is the input layer, which shape is defined in the next layer
              #keras.layers.LSTM(15, input_shape=(1,3), activation='relu', return_sequences = True),
              keras.layers.LSTM(4, input_shape=(2,3), activation='tanh', recurrent_activation= 'tanh', bias_initializer='zeros', kernel_initializer='glorot_uniform', rec

              keras.layers.LSTM(2, activation='tanh', recurrent_activation='tanh', bias_initializer='zeros', kernel_initializer='glorot_uniform', recurrent_initializer='

              keras.layers.TimeDistributed(Dense(3)) # activation='softmax'  #Output, softmax is loss probability between estimated and truth label btwn layers

])

#model.compile(Adam(lr=.0001),loss='mean_squared_error',metrics=['accuracy'])
#return model
```

```
model.compile(Adam(learning_rate=.0009), loss='mse', metrics=['accuracy']) # using optimizer = Adam; metrics=[tf.keras.metrics.Accuracy()])
```

Figure 166*: NN sequential model and compiler. Note that it is slightly cut off due to its length (the rest can be seen in Appendix B or an associated github link - also found in Appendix B). The learning rate, optimizer and loss/accuracy metrics are defined in the 'compile' command.*

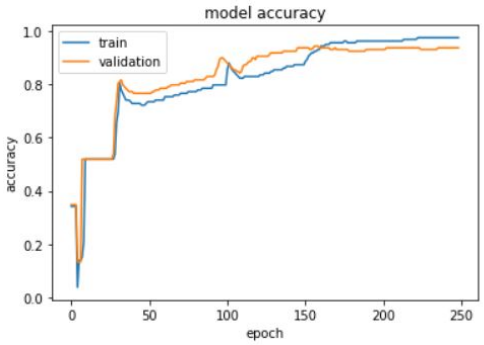
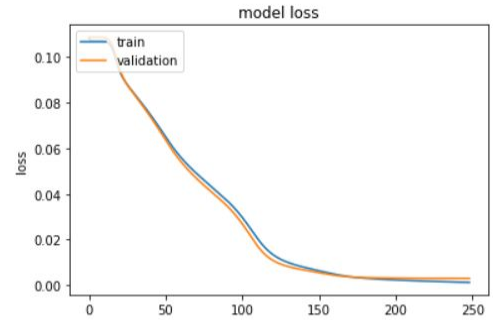
Since this is a many-to-many seq2seq prediction model using a TimeDistributed dense final layer, the 'return_sequences = True' for every LSTM layer. If the last layer did not have the 'TimeDistributed' wrapper then the last layer should have 'return_sequences = False'. The 'return_states' and 'stateful' are used together and can only be set to true when the 'batch_input_shape' attribute, of the first layer in the sequential model, is used (instead of the 'input_shape'). The former ('return_sequences) is responsible for h-state (hidden) while the latter ('stateful' and 'return_states') is for the c-state (or memory channel) outputs in between LSTM layers.

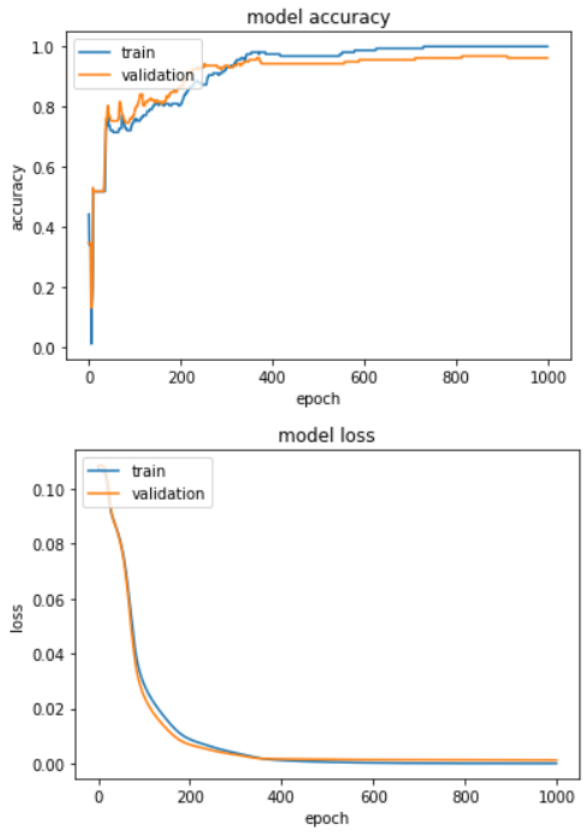
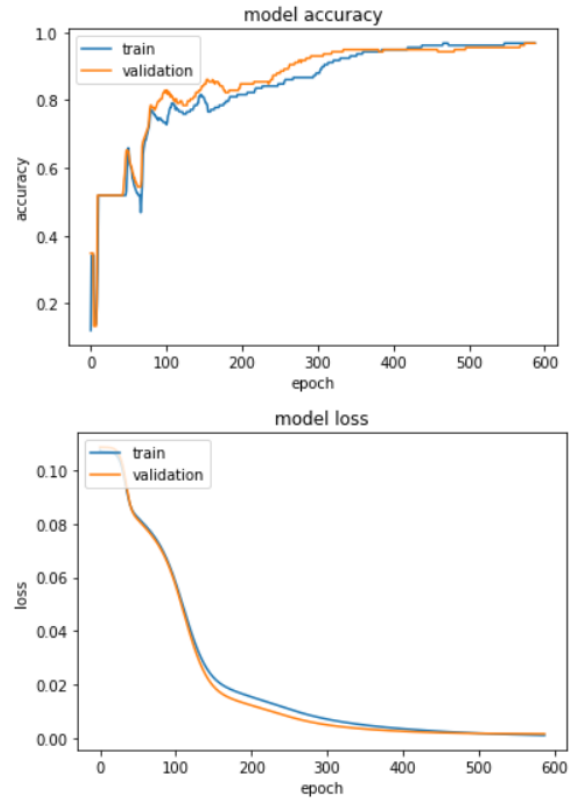
```
es_callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)    #can change what it monitors; stops training once overfitting occurs
```

```
history=model.fit(norm_X1_in_train,norm_Y1_out_train, batch_size= 7 , epochs=1000, callbacks=[es_callback], validation_data=(norm_X2_in_val, norm_Y2_out_val), validation_fre
```

Figure 167*: Sample of code that dictates the training process of the NN model. Note code cuts off, full version can be viewed in Appendix B or in associated github link). The 'callbacks' code shown above can be used to stop the model's training when overfitting occurs. Thus, it will stop at an earlier epoch than initially defined (i.e. epoch = 1000 but training stops at around 200 epochs instead).*

Table 5.3. Training and validation accuracy and loss plots for positions only.

Accuracy and Loss Plots:	Conditions and Notes:
 	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):4, 2 and 3. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY and PZ only. • Data shape:(79, 2, 3).(for both input and output, so many-to-many) • Batch_size = 7 (of the 79 samples) • # of iterations per epoch: 12. • Training stopped at 249 epochs. • Training loss = 0.0013 • Training accuracy = 97% • Validation loss = 0.003 • Validation accuracy = 94% • Testing accuracy = 92%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):4, 2 and 3. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY and PZ only. • Data shape:(79, 2, 3).(for both input and output, so many-to-many) • Batch_size = 12 • # of iterations per epoch: 7. • Training stopped at 1000 epochs. • Training loss = 2.95e-05 • Training accuracy = 100% • Validation loss = 0.0012 • Validation accuracy = 96% • Testing accuracy = 90% • Exact match between predicted and true labels : 82%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):4, 2 and 3. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY and PZ only. • Data shape:(79, 2, 3). (for both input and output, so many-to-many) • Batch_size = 17 • # of iterations per epoch: 5. • Training stopped at 588 epochs. • Training loss = 9.96e-04 • Training accuracy = 97% • Validation loss = 0.0015 • Validation accuracy = 97%

One can evaluate the accuracy of the algorithm (NN model's weights) using the testing data via the following code:

```
e = model.evaluate(
    x=norm_X3_in_test,
    y=norm_Y3_out_test,
    verbose=1
)

e = {out: e[i] for i, out in enumerate(model.metrics_names)}
```

```
3/3 [==============================] - 0s 7ms/step - loss: 0.0076 - accuracy: 0.9241
```

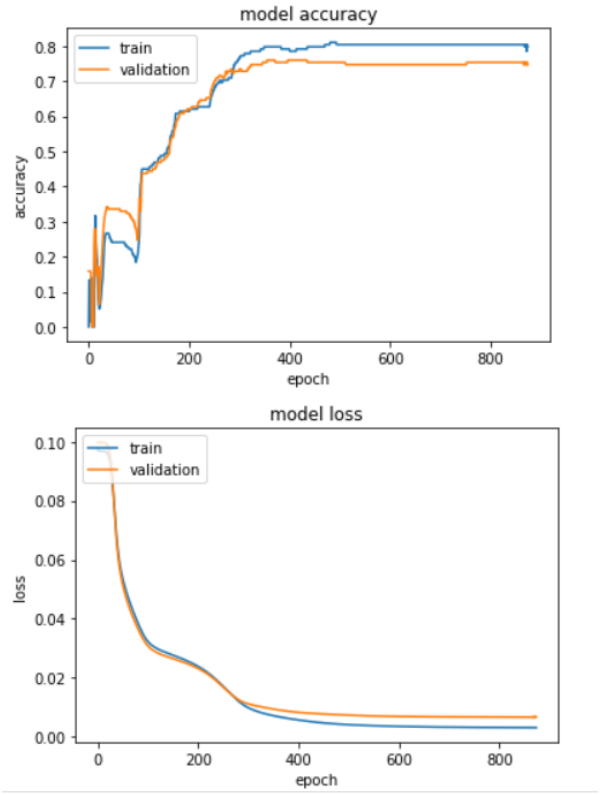
***Figure 168**: Evaluating the accuracy of the model using Testing data.*

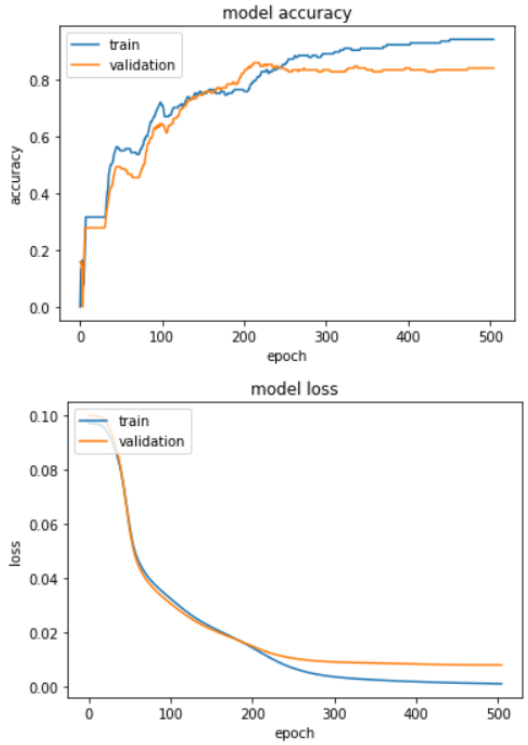
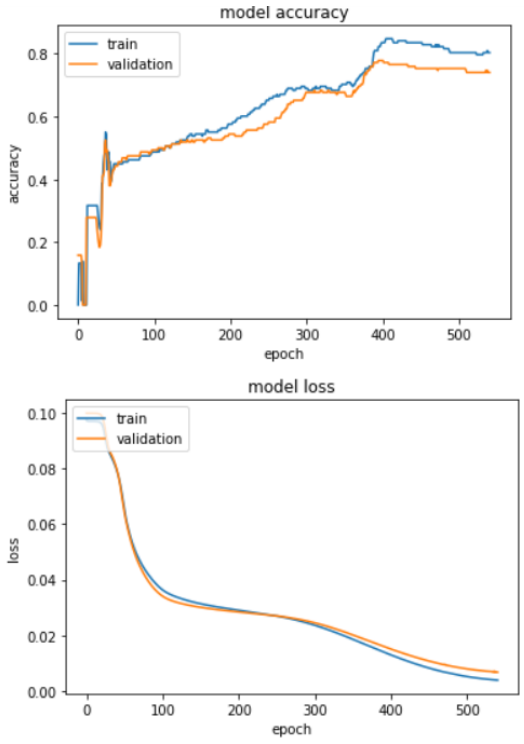
Objective of the training of the NN model for 3 features has been achieved: attaining the 90th percentile for training, validation and testing accuracies.

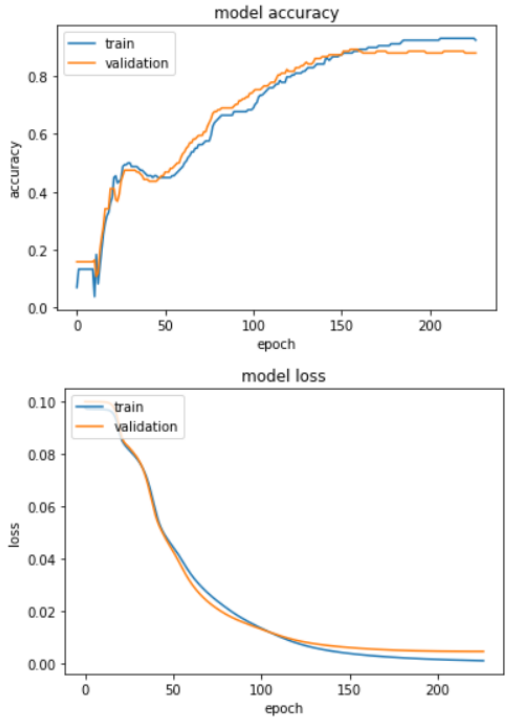
5.1.7 NN Model and Training Results of Six Features

This section examines NN training of all six states features (position and velocity) instead of just three (position). The data used can be viewed in Appendix C. Data was also mean-normalized per feature. The goal is to attain training, validation and testing in the 90th percentile. (Note: Testing accuracy was between 55 and 65% for any output where the testing accuracy was not mentioned.)

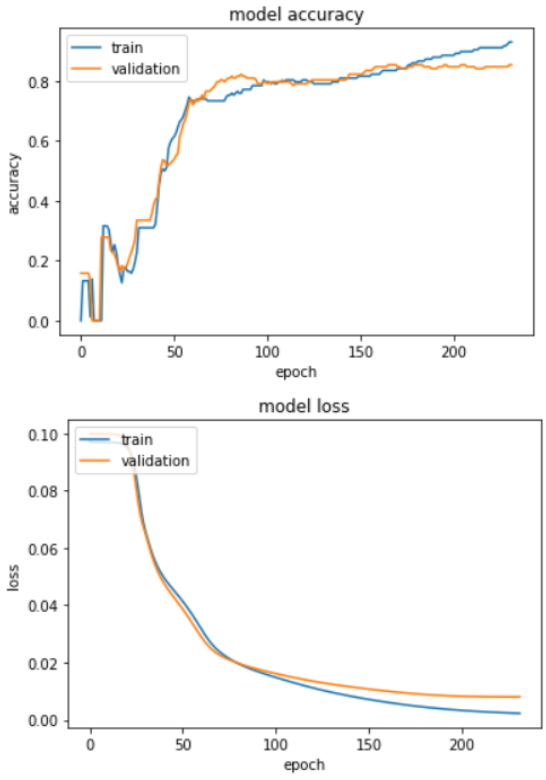
Table 5.4. Training and validation accuracy and loss plots for position and velocity.

Accuracy and Loss Plots:	**Conditions and Notes:**
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):4, 2 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 12 (of the 79 samples) • # of iterations per epoch: 7. • Training stopped at 874 epochs. • Training loss = 0.0031 • Training accuracy = 80% • Validation loss = 0.0067 • Validation accuracy = 75%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):5, 4 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 505 epochs. • Training loss = 0.0010 • Training accuracy = 94% • Validation loss = 0.008 • Validation accuracy = 84% • Testing accuracy = 72%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):7, 2 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 12 (of the 79 samples) • # of iterations per epoch: 7. • Training stopped at 541 epochs. • Training loss = 0.0040 • Training accuracy = 80% • Validation loss = 0.0068 • Validation accuracy = 74% • Testing accuracy = 75%

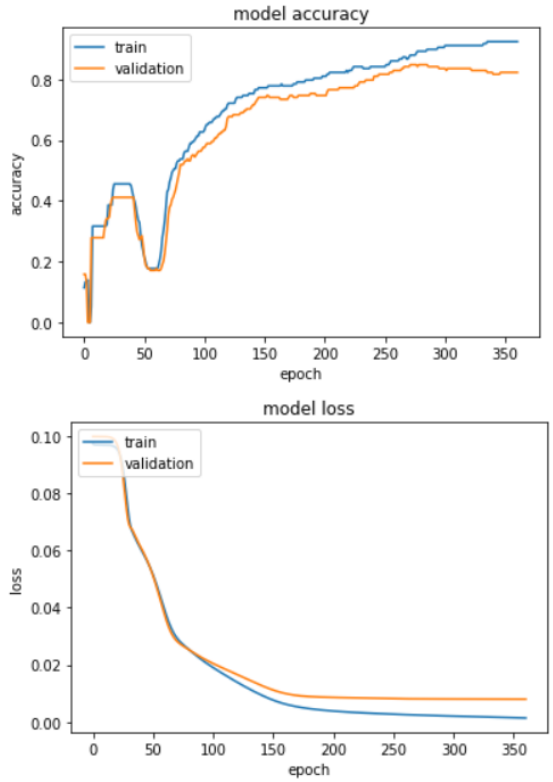
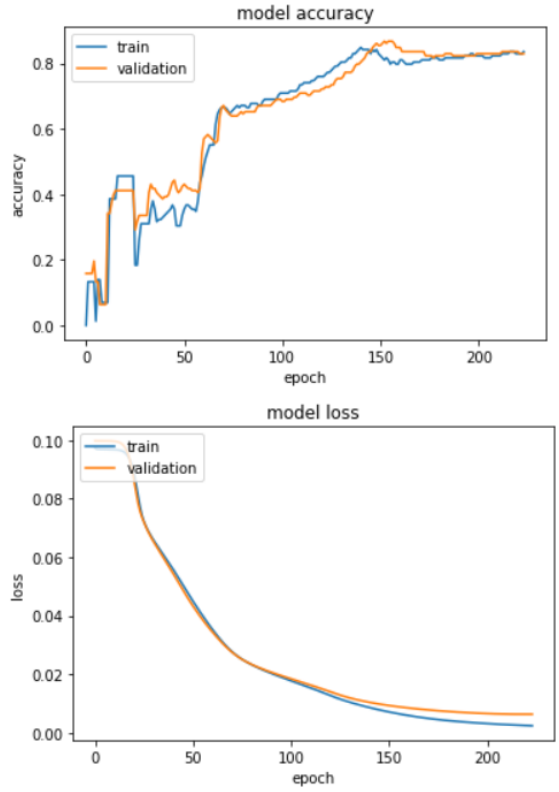
<table>
<tr>
<td> 
</td>
<td>

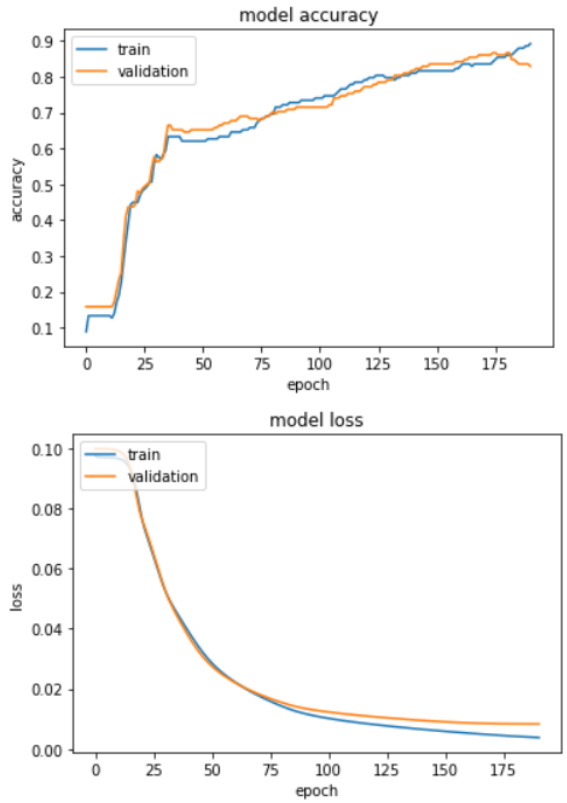
Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):9, 7 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 9 (of the 79 samples)
# of iterations per epoch: 9.
Training stopped at 227 epochs.
Training loss = 0.0012
Training accuracy = 92%
Validation loss = 0.0047
Validation accuracy = 88%
Testing accuracy = 68%

</td>
</tr>
<tr>
<td> 
</td>
<td>

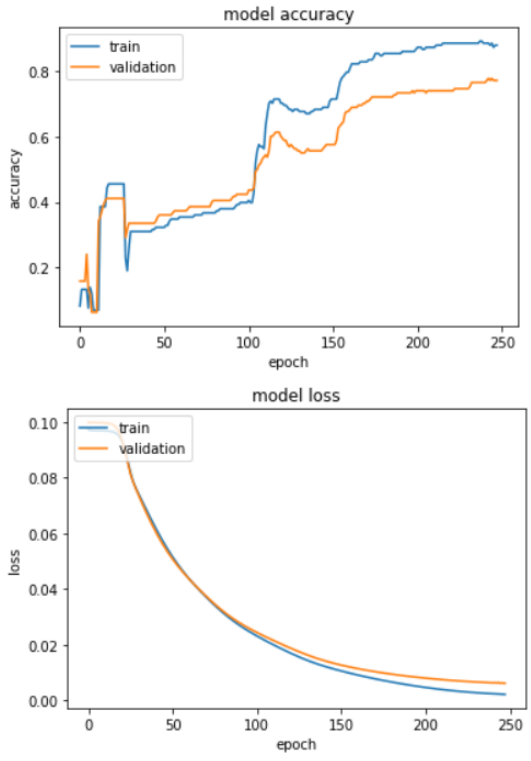
Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):9, 7 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 12 (of the 79 samples)
# of iterations per epoch: 7.
Training stopped at 232 epochs.
Training loss = 0.0023
Training accuracy = 93%
Validation loss = 0.0080
Validation accuracy = 85%
Testing accuracy = 72%

</td>
</tr>
</table>

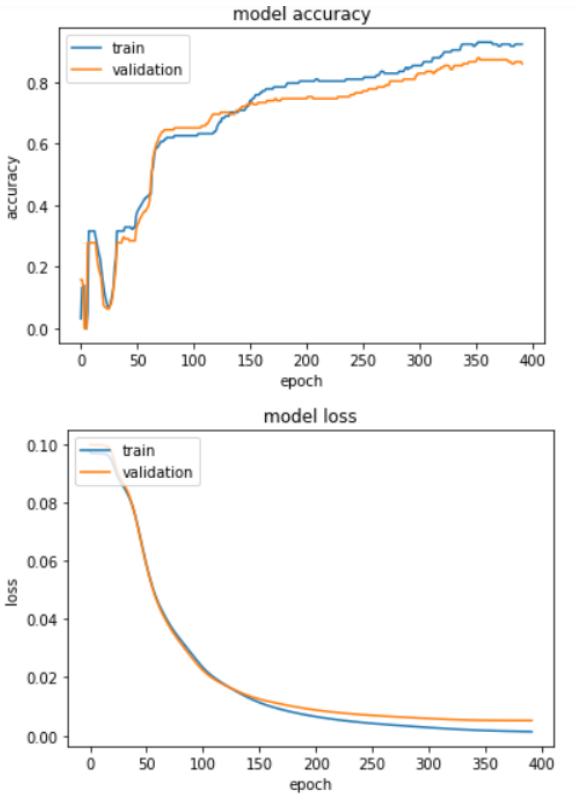
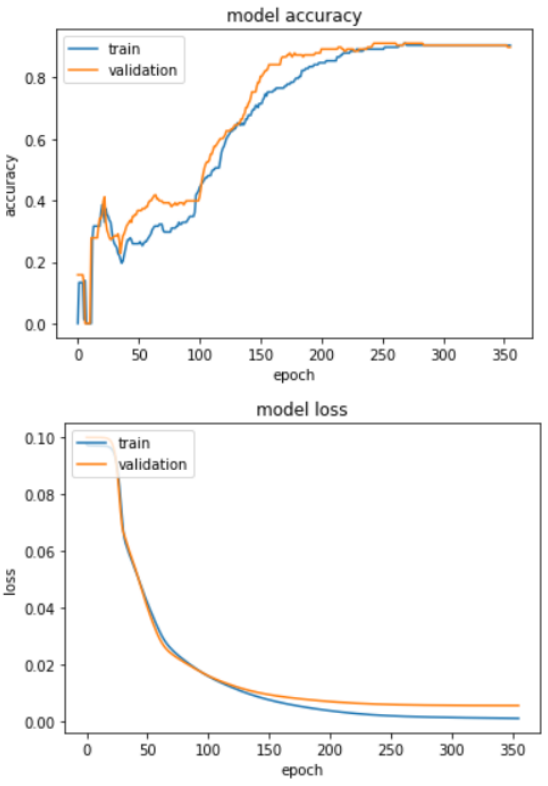
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):9, 7 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 361 epochs. • Training loss = 0.0014 • Training accuracy = 93% • Validation loss = 0.0080 • Validation accuracy = 82% • Testing accuracy = 71%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):9, 6 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 12 (of the 79 samples) • # of iterations per epoch: 7. • Training stopped at 224 epochs. • Training loss = 0.0024 • Training accuracy = 84% • Validation loss = 0.0064 • Validation accuracy = 83% • Testing accuracy = 84%

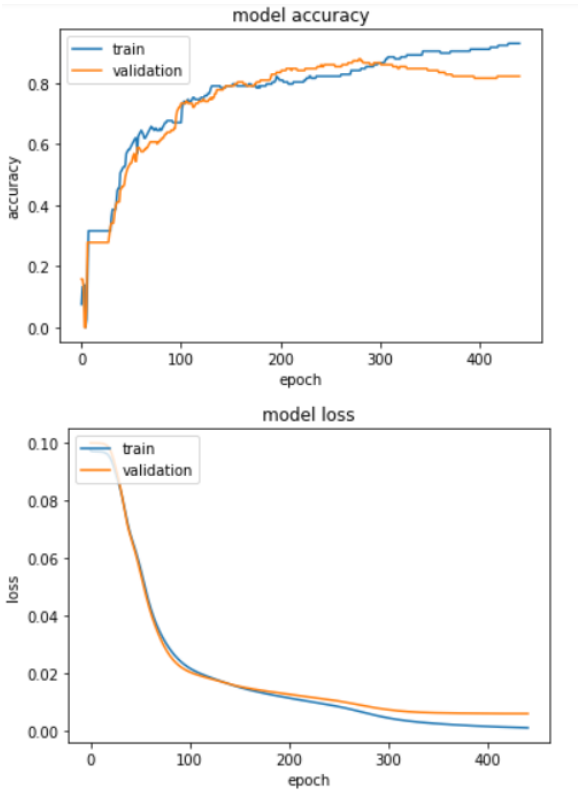


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):9, 5 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9 (of the 79 samples)
- # of iterations per epoch: 9.
- Training stopped at 191 epochs.
- Training loss = 0.0038
- Training accuracy = 89%
- Validation loss = 0.0083
- Validation accuracy = 83%
- Testing accuracy = 74%

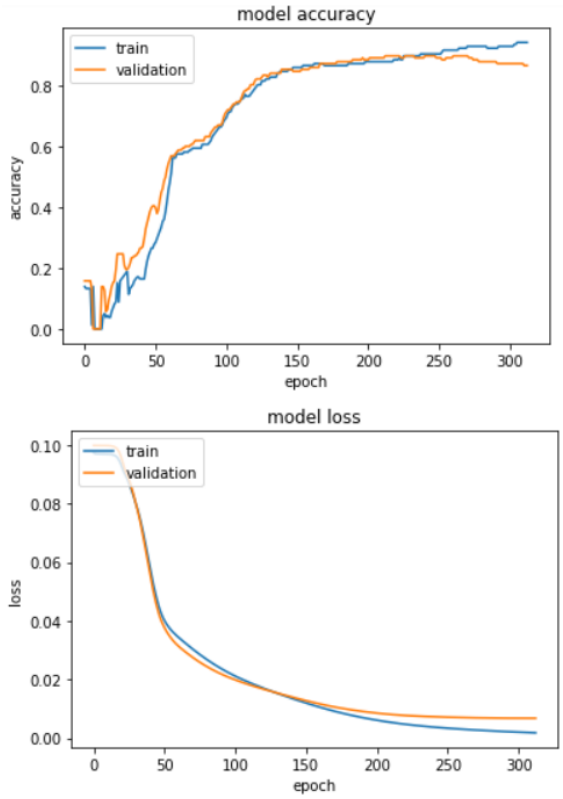


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):9, 5 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 12 (of the 79 samples)
- # of iterations per epoch: 7.
- Training stopped at 248 epochs.
- Training loss = 0.0022
- Training accuracy = 88%
- Validation loss = 0.0063
- Validation accuracy = 77%
- Testing accuracy = 67%

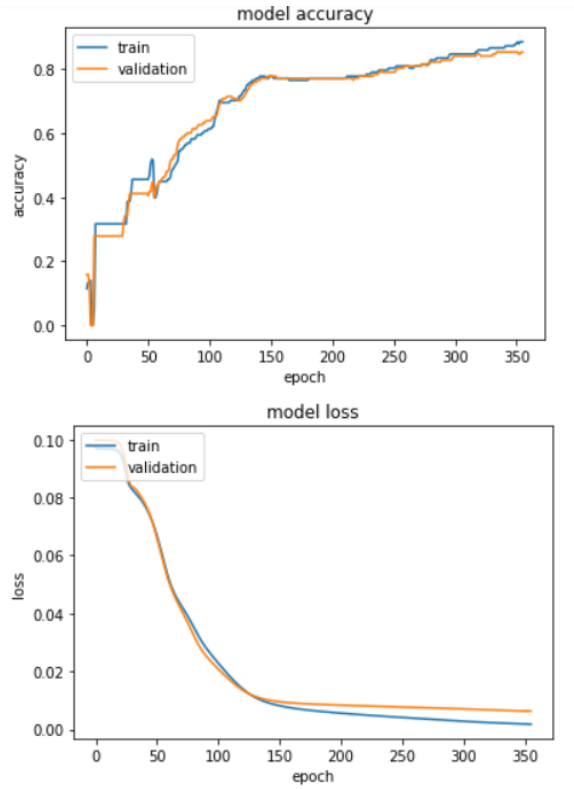
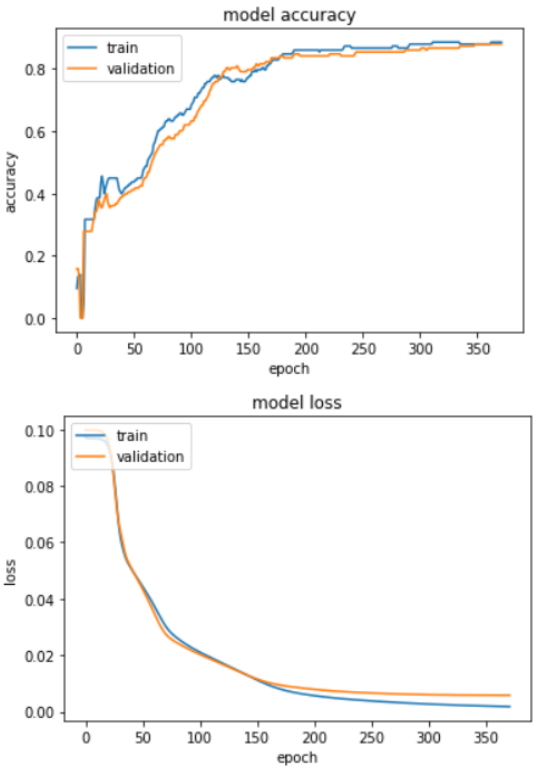
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):9, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 392 epochs. • Training loss = 0.0014 • Training accuracy = 92% • Validation loss = 0.0052 • Validation accuracy = 86% • Testing accuracy = 77%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):8, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 12 (of the 79 samples) • # of iterations per epoch: 7. • Training stopped at 356 epochs. • Training loss = 0.0013 • Training accuracy = 91% • Validation loss = 0.0058 • Validation accuracy = 90% • Testing accuracy = 78%

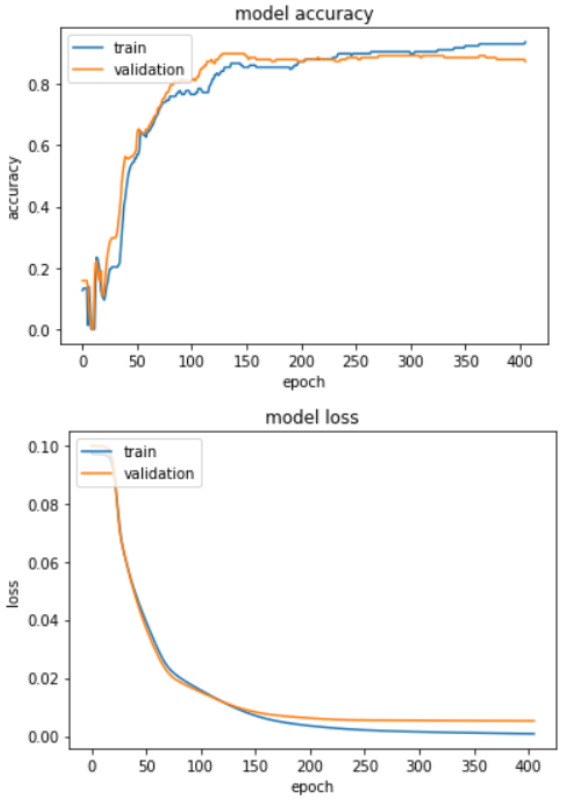


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):8, 5 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 15 (of the 79 samples)
- # of iterations per epoch: 6.
- Training stopped at 441 epochs.
- Training loss = 0.0012
- Training accuracy = 93%
- Validation loss = 0.0061
- Validation accuracy = 82%
- Testing accuracy = 70%



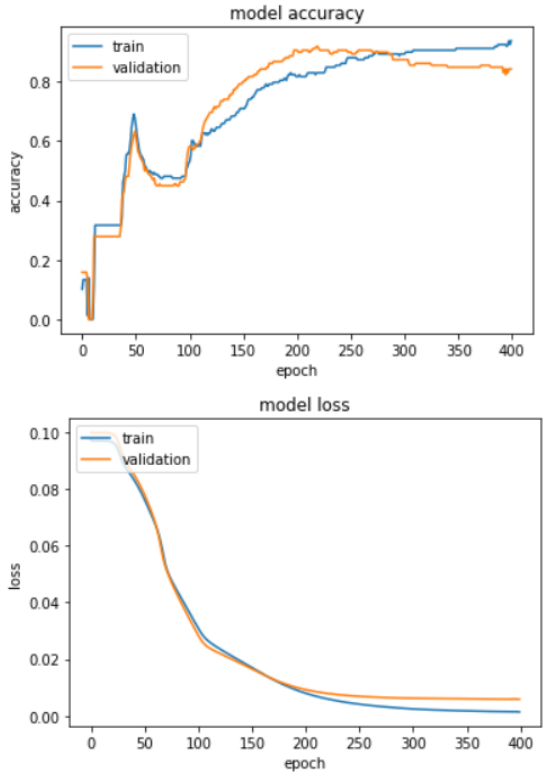
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):8, 4 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 12 (of the 79 samples)
- # of iterations per epoch: 7.
- Training stopped at 313 epochs.
- Training loss = 0.0018
- Training accuracy = 94%
- Validation loss = 0.0068
- Validation accuracy = 87%
- Testing accuracy = 66%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):8, 6 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 356 epochs. • Training loss = 0.0018 • Training accuracy = 89% • Validation loss = 0.0063 • Validation accuracy = 85% • Testing accuracy = 72%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):8, 7 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 372 epochs. • Training loss = 0.0018 • Training accuracy = 89% • Validation loss = 0.0058 • Validation accuracy = 88% • Testing accuracy = 70%

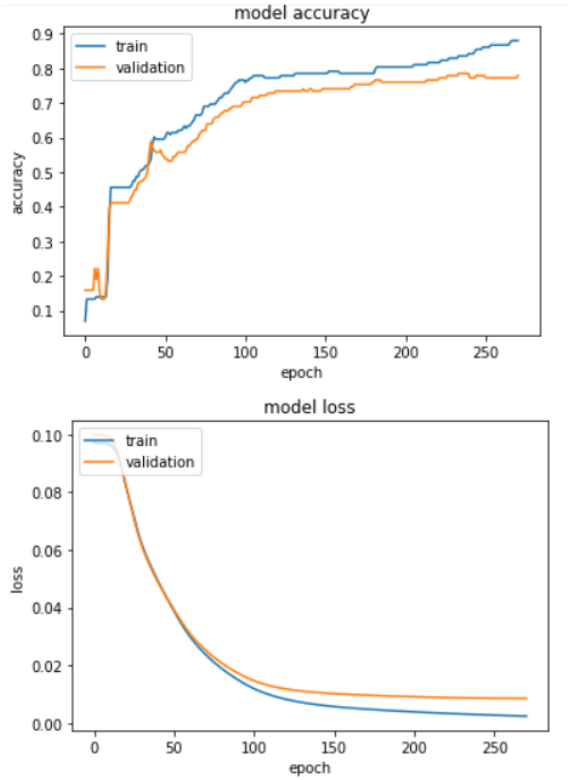
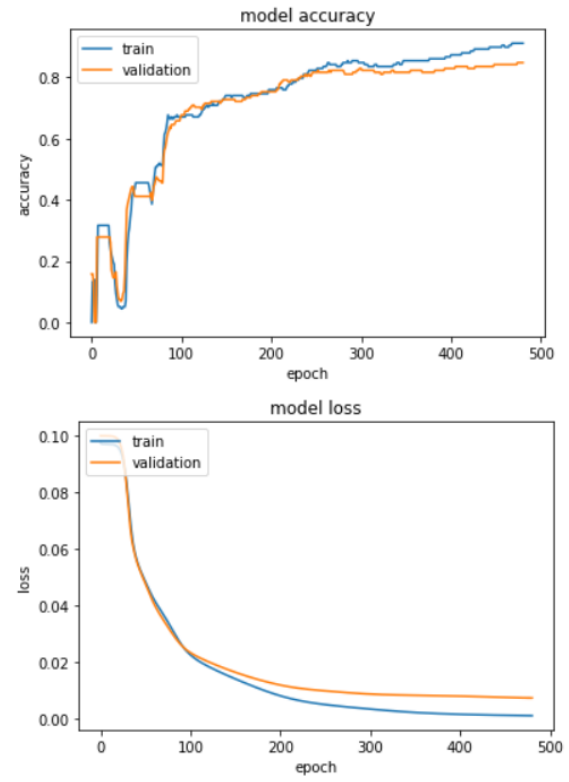


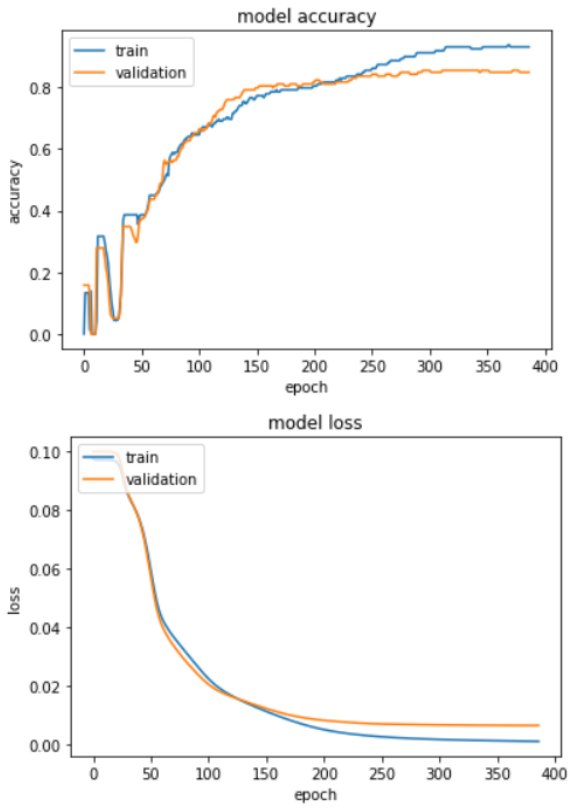
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):8, 7 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 12 (of the 79 samples)
- # of iterations per epoch: 7.
- Training stopped at 406 epochs.
- Training loss = 8.014e-04
- Training accuracy = 94%
- Validation loss = 0.0052
- Validation accuracy = 87%
- Testing accuracy = 77%

  [Predictive accuracy = 51%]

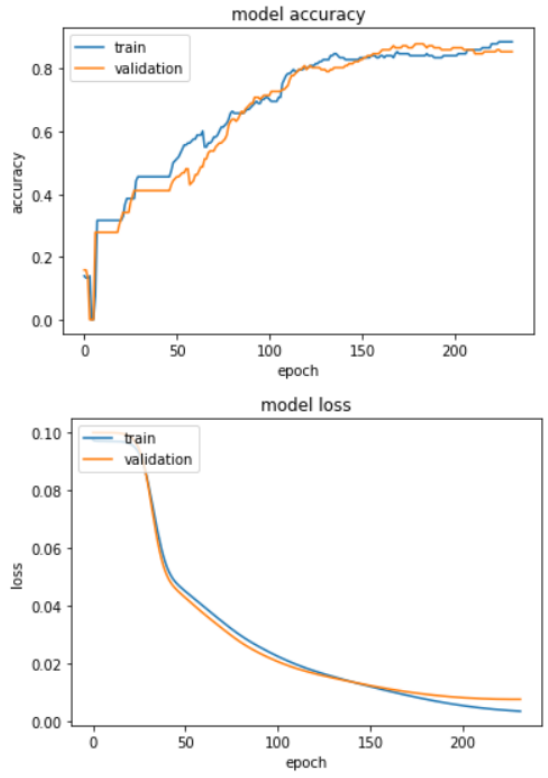


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):7, 4 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 12 (of the 79 samples)
- # of iterations per epoch: 7.
- Training stopped at 400 epochs.
- Training loss = 0.0014
- Training accuracy = 94%
- Validation loss = 0.0059
- Validation accuracy = 84%
- Testing accuracy = 68%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):7, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 (of the 79 samples) • # of iterations per epoch: 9. • Training stopped at 271 epochs. • Training loss = 0.0024 • Training accuracy = 88% • Validation loss = 0.0085 • Validation accuracy = 78% • Testing accuracy = 66%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):7, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 481 epochs. • Training loss = 0.001 • Training accuracy = 91% • Validation loss = 0.0073 • Validation accuracy = 85% • Testing accuracy = 72%

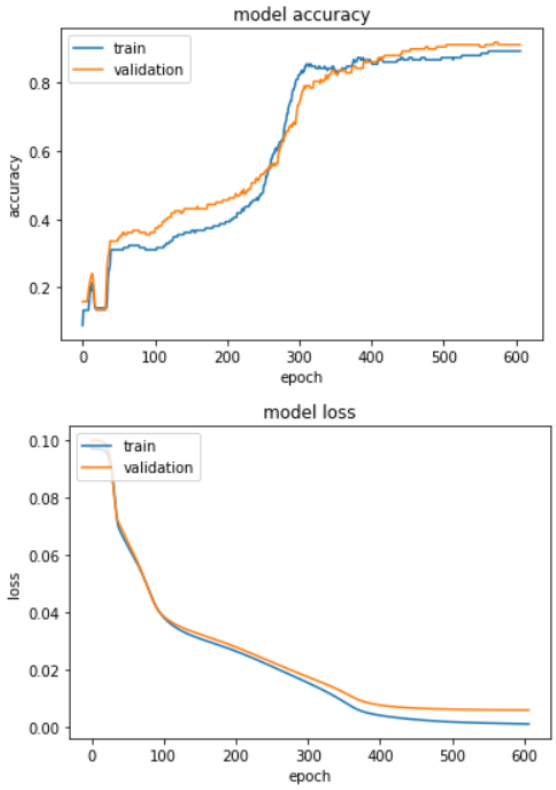
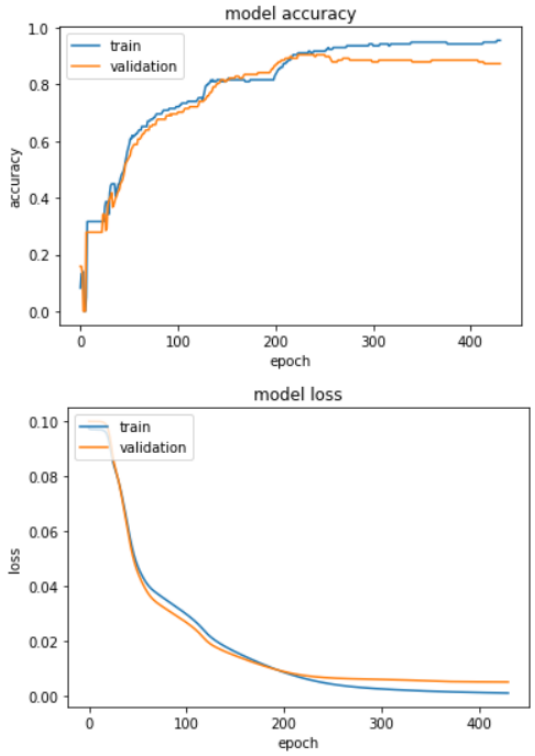
<table>
<tr>
<td> 
</td>
<td>

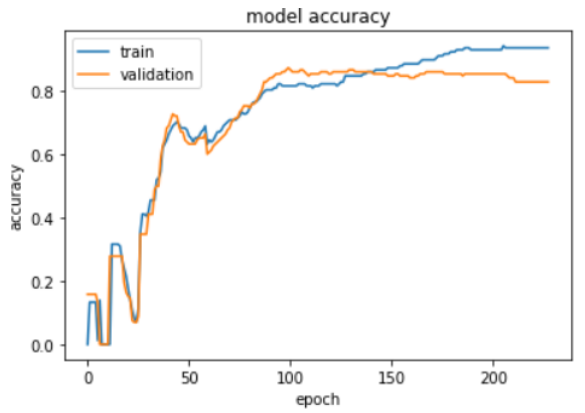
Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):7, 5 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 12 (of the 79 samples)
# of iterations per epoch: 7.
Training stopped at 387 epochs.
Training loss = 9.13e-04
Training accuracy = 93%
Validation loss = 0.0063
Validation accuracy = 85%
Testing accuracy = 66%

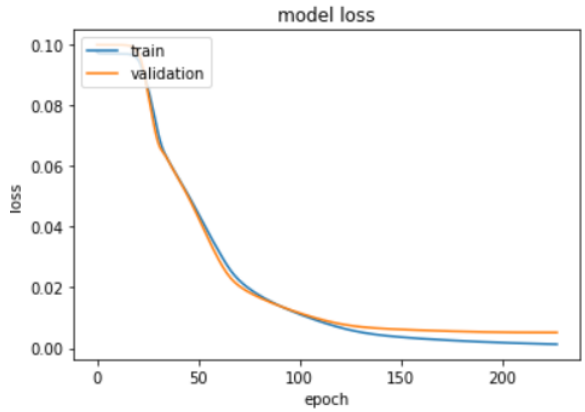
</td>
</tr>
<tr>
<td> 
</td>
<td>

Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):6, 5 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 15 (of the 79 samples)
# of iterations per epoch: 6.
Training stopped at 232 epochs.
Training loss = 0.0036
Training accuracy = 89%
Validation loss = 0.0077
Validation accuracy = 85%
Testing accuracy = 75%

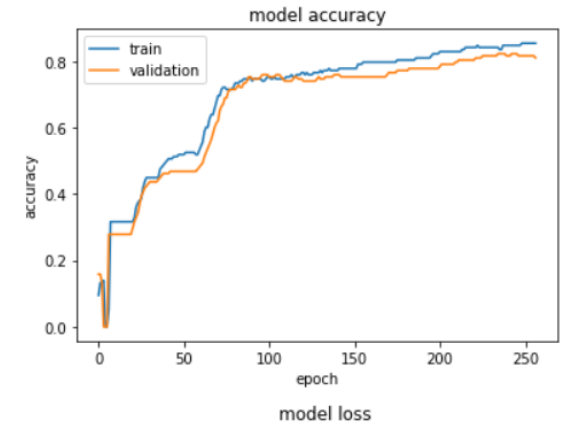
</td>
</tr>
</table>

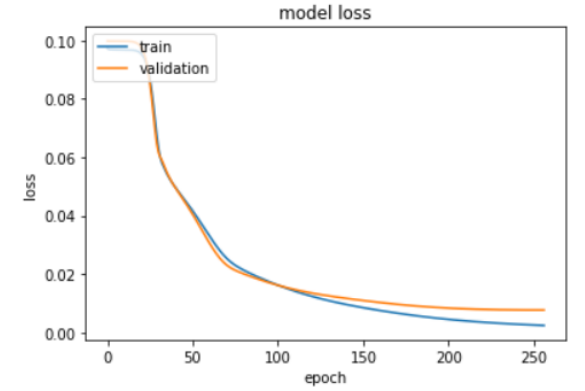
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):6, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 16 (of the 79 samples) • # of iterations per epoch: 5. • Training stopped at 607 epochs. • Training loss = 0.0011 • Training accuracy = 89% • Validation loss = 0.0059 • Validation accuracy = 91% • Testing accuracy = 72%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):10, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 15 (of the 79 samples) • # of iterations per epoch: 6. • Training stopped at 431 epochs. • Training loss = 0.0011 • Training accuracy = 96% • Validation loss = 0.0051 • Validation accuracy = 87% • Testing accuracy = 75%



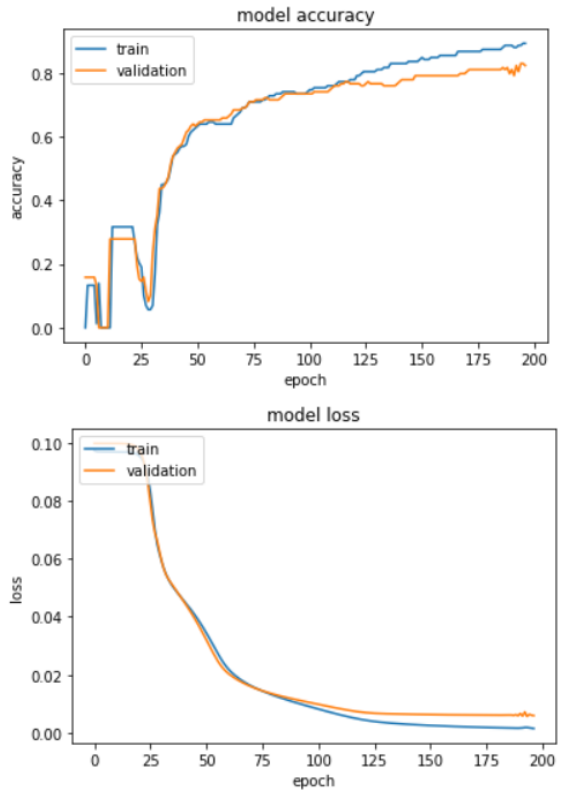


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):12, 8 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6).(for both input and output, so many-to-many)
- Batch_size = 12
- # of iterations per epoch: 7 .
- Training stopped at 228 epochs.
- Training loss = 0.0014
- Training accuracy = 94%
- Validation loss = 0.0052
- Validation accuracy = 83%

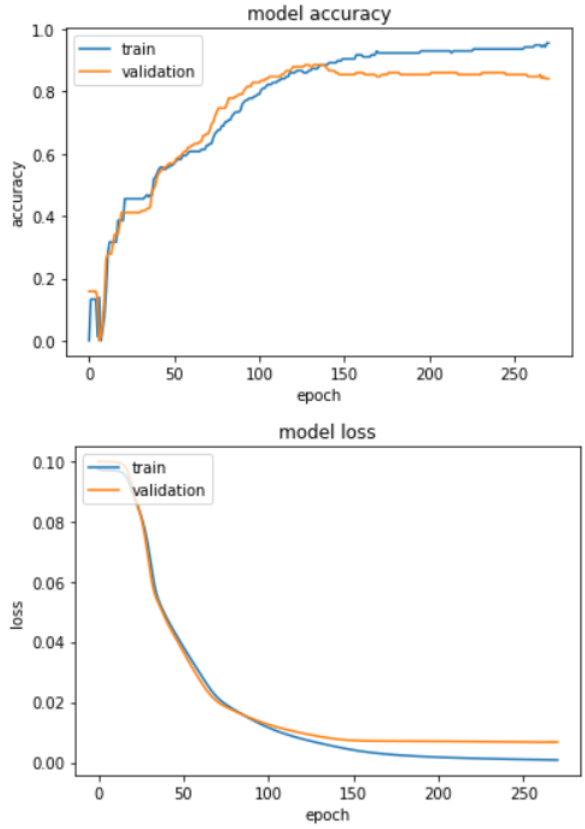




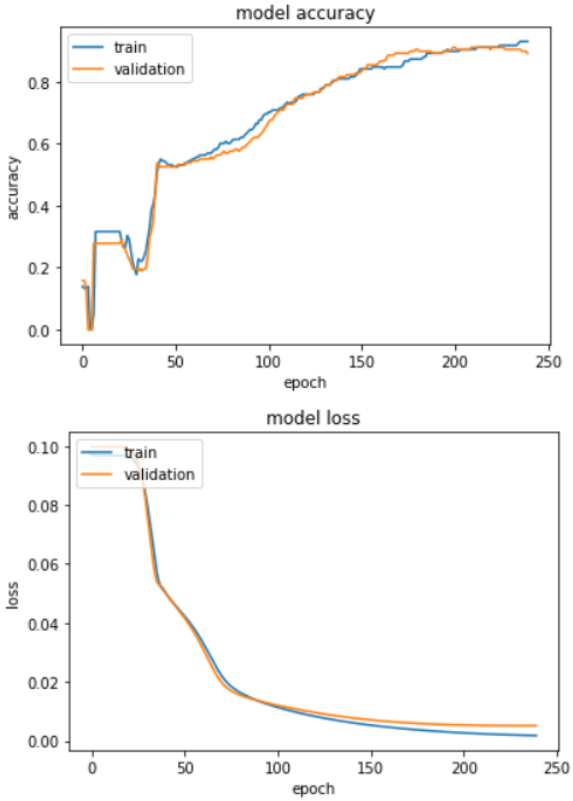
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):12, 8 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6).(for both input and output, so many-to-many)
- Batch_size = 15
- # of iterations per epoch: 6 .
- Training stopped at 257 epochs.
- Training loss = 0.0025
- Training accuracy = 85%
- Validation loss = 0.0077
- Validation accuracy = 81%
- Testing accuracy = 72%



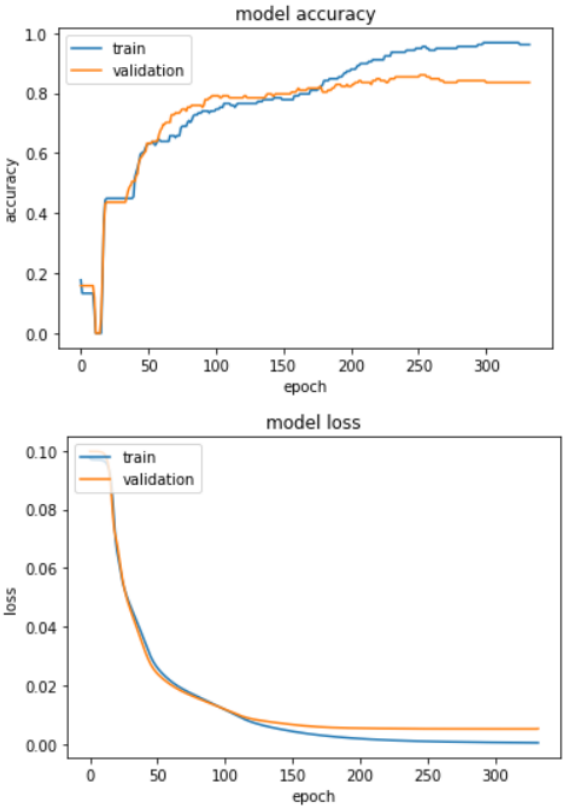
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):18, 10 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6).(for both input and output, so many-to-many)
- Batch_size = 12
- # of iterations per epoch: 7.
- Training stopped at 197 epochs.
- Training loss = 0.0015
- Training accuracy = 89%
- Validation loss = 0.0059
- Validation accuracy = 82%



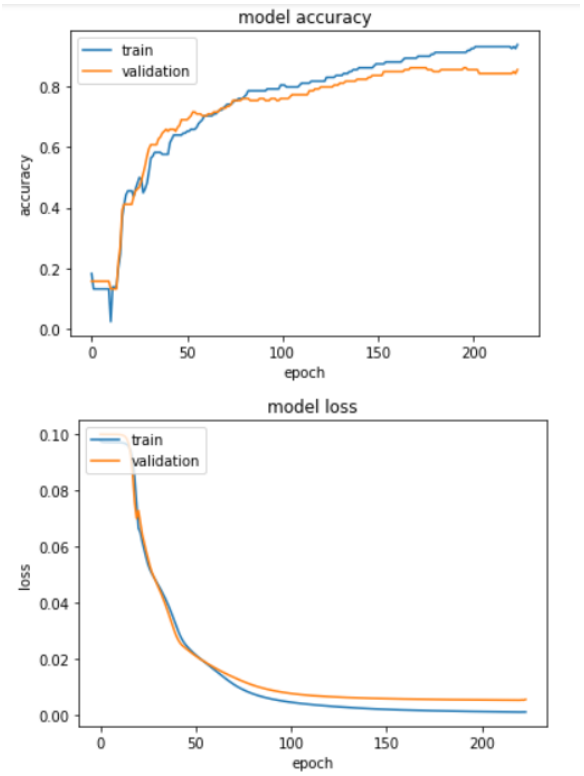
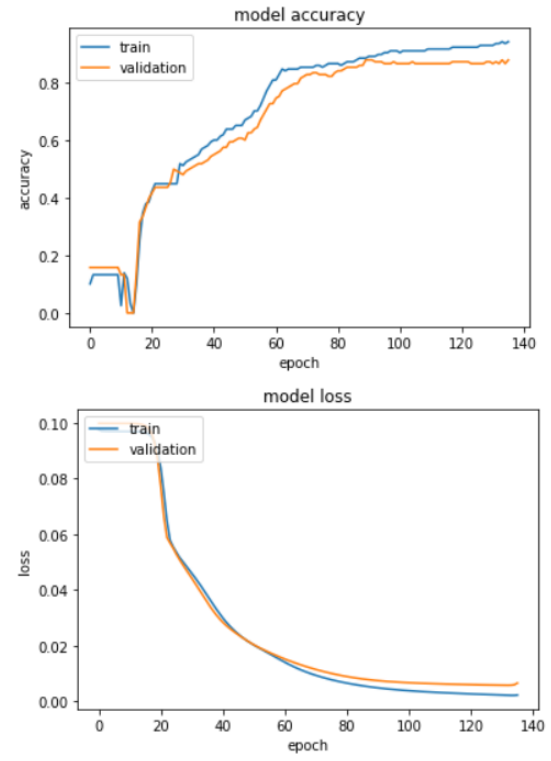
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):12, 8 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 12
- # of iterations per epoch: 7.
- Training stopped at 271 epochs.
- Training loss = 8.35e-04
- Training accuracy = 96%
- Validation loss = 0.0068
- Validation accuracy = 84%

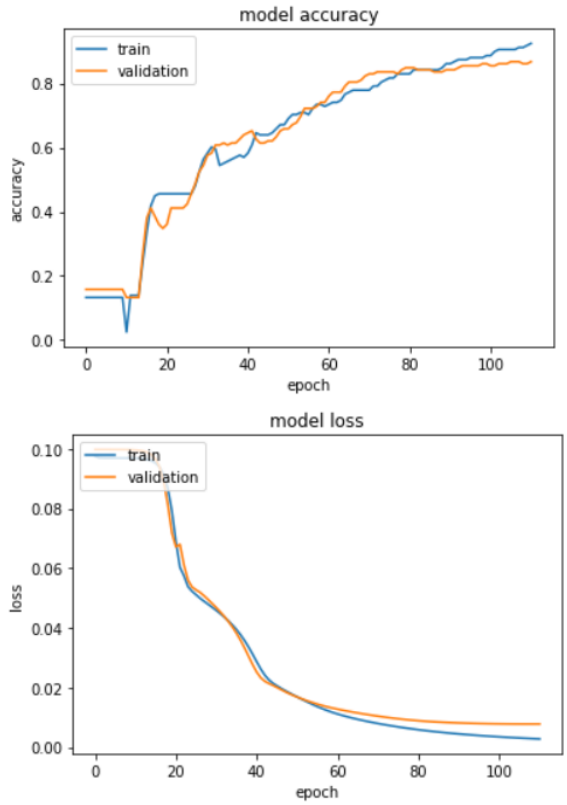


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):12, 8 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6).(for both input and output, so many-to-many)
- Batch_size = 15
- # of iterations per epoch: 6 .
- Training stopped at 240 epochs.
- Training loss = 0.0018
- Training accuracy = 93%
- Validation loss = 0.0051
- Validation accuracy = 89%
- Testing accuracy = 69%

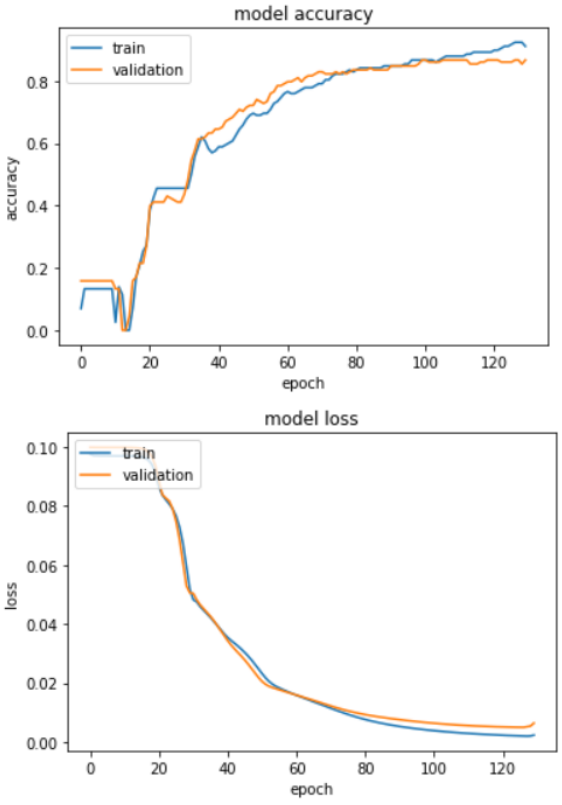


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):12, 8 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 333 epochs.
- Training loss = 4.55e-04
- Training accuracy = 96%
- Validation loss = 0.0052
- Validation accuracy = 84%
- Testing accuracy = 60%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):24, 8 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 224 epochs. • Training loss = 0.0012 • Training accuracy = 94% • Validation loss = 0.0056 • Validation accuracy = 85% • Testing accuracy = 74%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):24, 10 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 136 epochs. • Training loss = 0.0023 • Training accuracy = 94% • Validation loss = 0.0066 • Validation accuracy = 88% • Testing accuracy = 72%

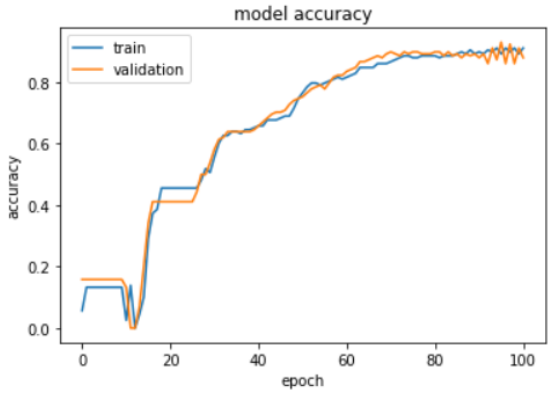
<table>
<tr>
<td> 
</td>
<td>

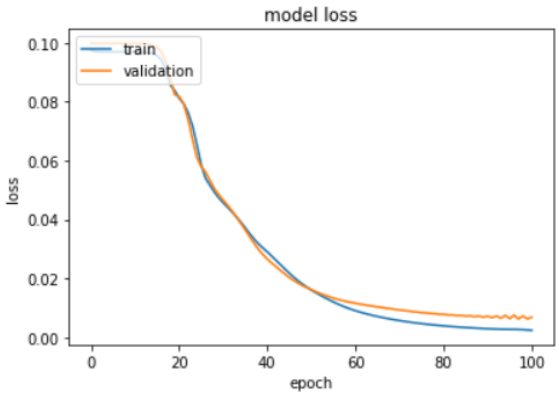
Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):24, 12 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 9
# of iterations per epoch: 9 .
Training stopped at 111 epochs.
Training loss = 0.0028
Training accuracy = 92%
Validation loss = 0.0078
Validation accuracy = 87%
Testing accuracy = 73%

</td>
</tr>
<tr>
<td> 
</td>
<td>

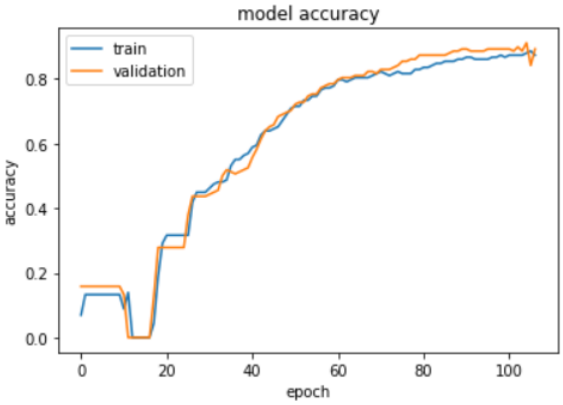
Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
# of cells (in order of layers):24, 16 and 6.
Learning_rate = 0.0009.
Data has been mean- normalized per feature.
UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
Data shape:(79, 2, 6). (for both input and output, so many-to-many)
Batch_size = 9
# of iterations per epoch: 9 .
Training stopped at 130 epochs.
Training loss = 0.0023
Training accuracy = 91%
Validation loss = 0.0064
Validation accuracy = 87%
Testing accuracy = 68%

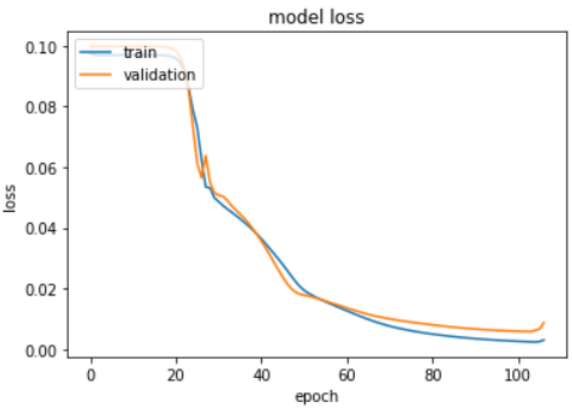
</td>
</tr>
</table>



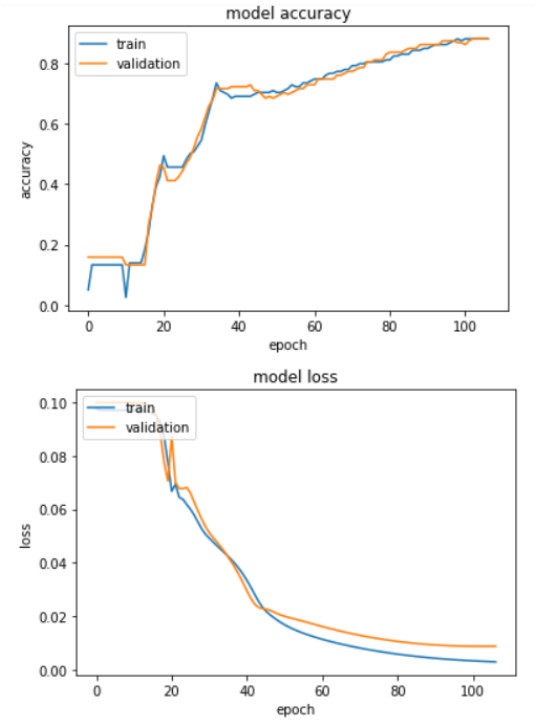
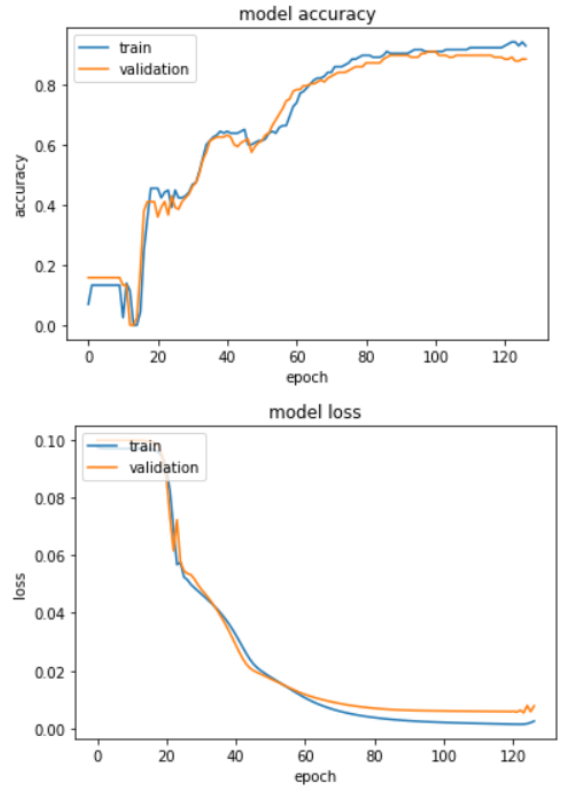


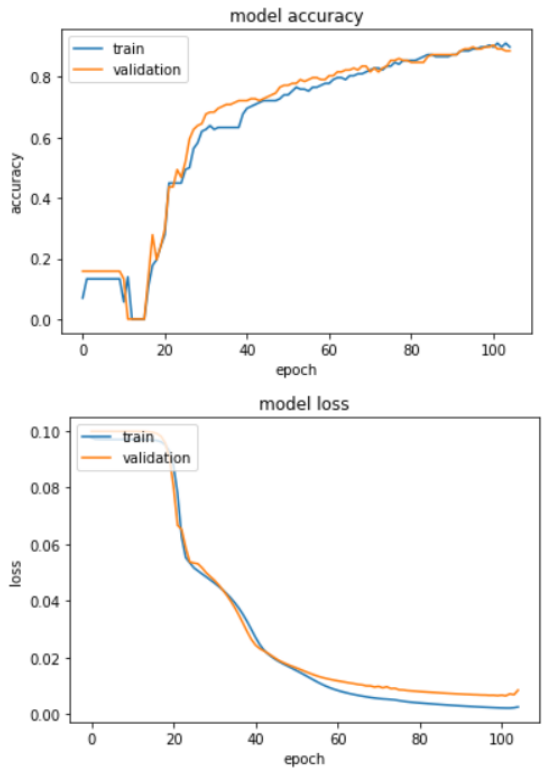
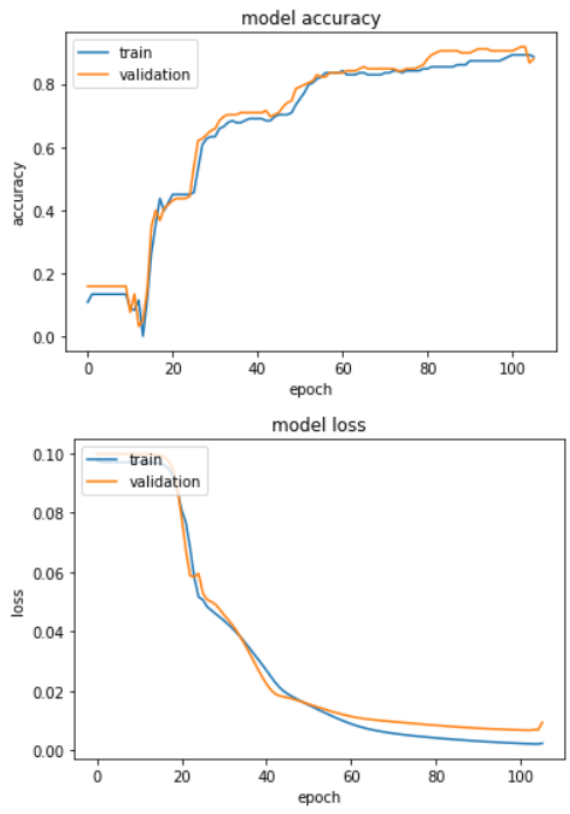
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 30, 16 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 101 epochs.
- Training loss = 0.0024
- Training accuracy = 91%
- Validation loss = 0.0064
- Validation accuracy = 88%
- Testing accuracy = 76%

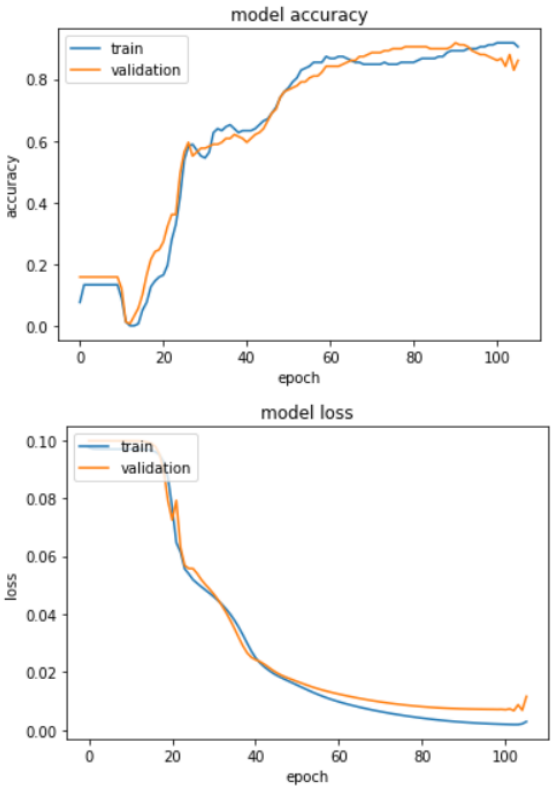




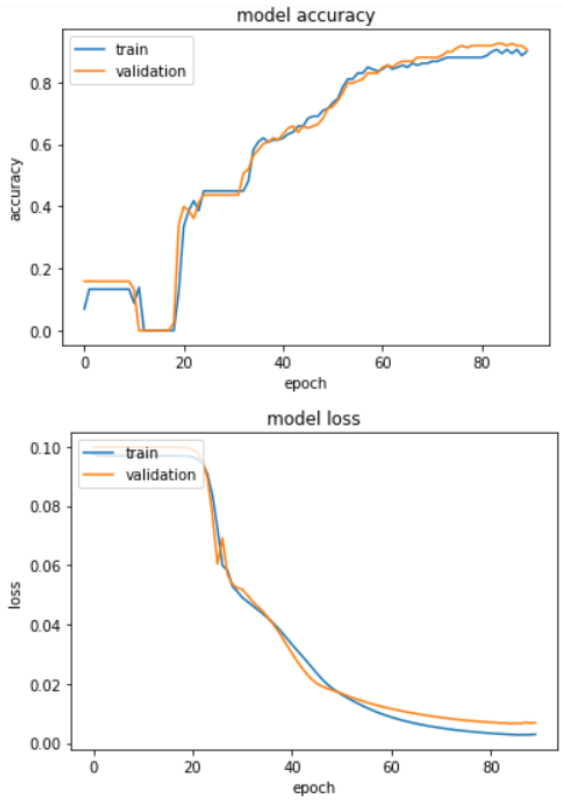
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 30, 22 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 107 epochs.
- Training loss = 0.003
- Training accuracy = 87%
- Validation loss = 0.0087
- Validation accuracy = 89%
- Testing accuracy = 81%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 30, 23 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 107 epochs. • Training loss = 0.0031 • Training accuracy = 88% • Validation loss = 0.009 • Validation accuracy = 88% • Testing accuracy = 79%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 36, 18 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 127 epochs. • Training loss = 0.0026 • Training accuracy = 93% • Validation loss = 0.0078 • Validation accuracy = 89% • Testing accuracy = 73%

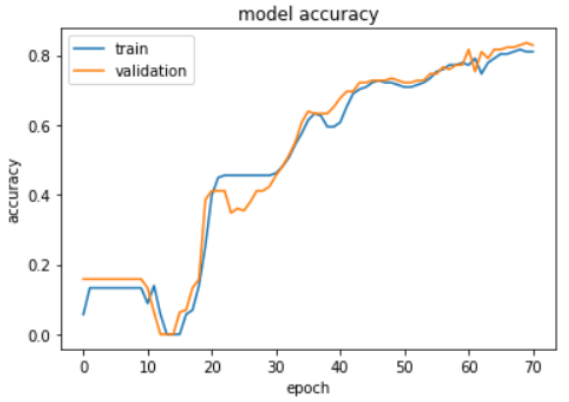
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 36, 21 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 105 epochs. • Training loss = 0.0025 • Training accuracy = 90% • Validation loss = 0.0085 • Validation accuracy = 89% • Testing accuracy = 76%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 36, 22 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 106 epochs. • Training loss = 0.0024 • Training accuracy = 89% • Validation loss = 0.0094 • Validation accuracy = 88% • Testing accuracy = 69%

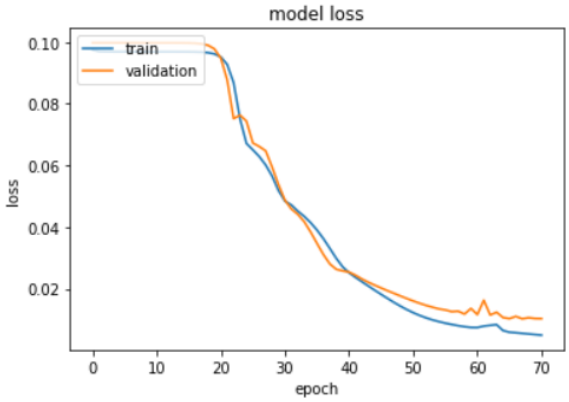


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 36, 24 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 106 epochs.
- Training loss = 0.0030
- Training accuracy = 91%
- Validation loss = 0.012
- Validation accuracy = 86%
- Testing accuracy = 73%

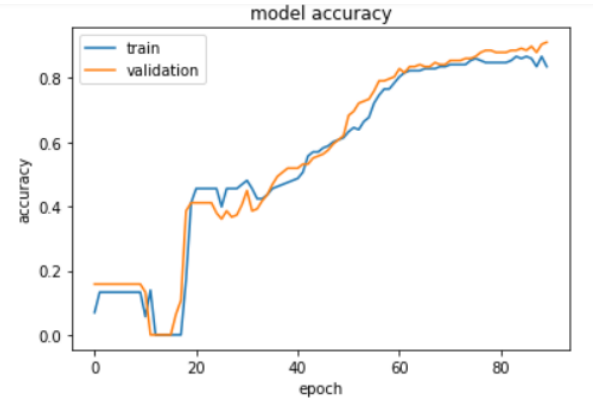


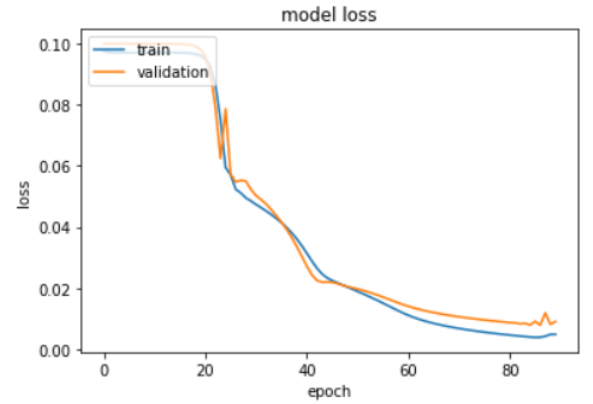
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 42, 26 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 90 epochs.
- Training loss = 0.0029
- Training accuracy = 90%
- Validation loss = 0.0069
- Validation accuracy = 91%
- Testing accuracy = 73%



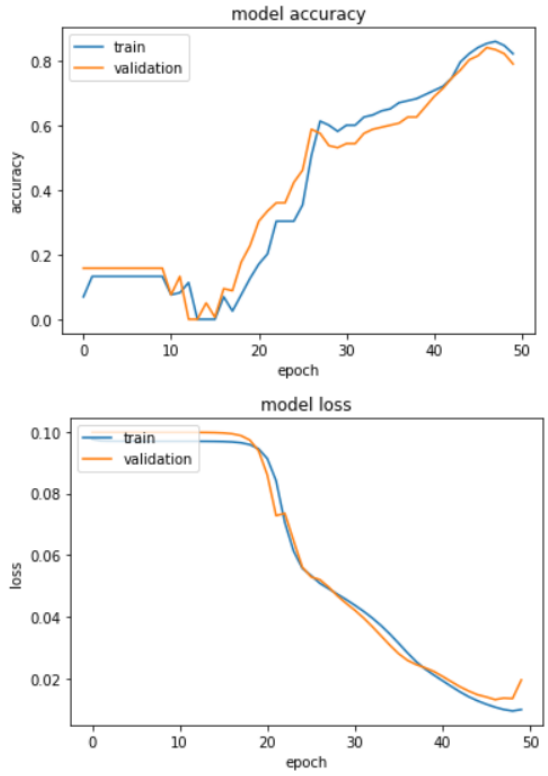


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 42, 28 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 71 epochs.
- Training loss = 0.005
- Training accuracy = 81%
- Validation loss = 0.0103
- Validation accuracy = 83%
- Testing accuracy = 77%

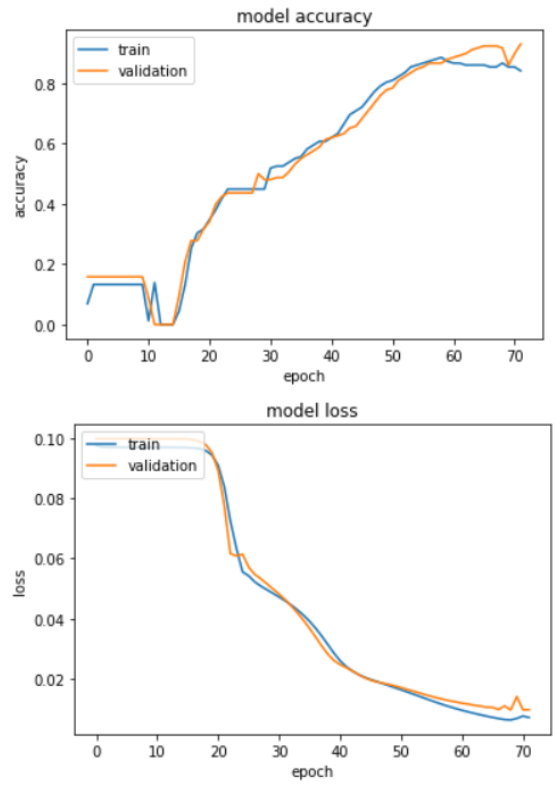




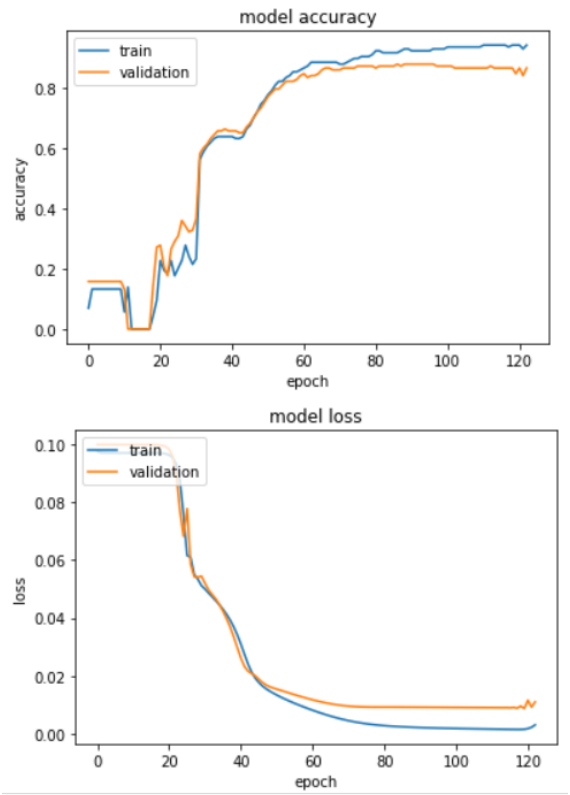
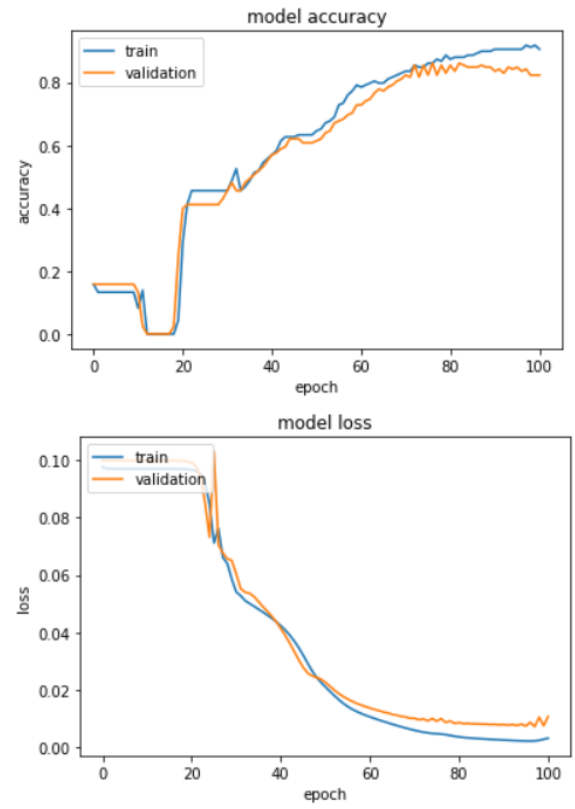
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 42, 30 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 90 epochs.
- Training loss = 0.0049
- Training accuracy = 84%
- Validation loss = 0.0091
- Validation accuracy = 91%
- Testing accuracy = 82%

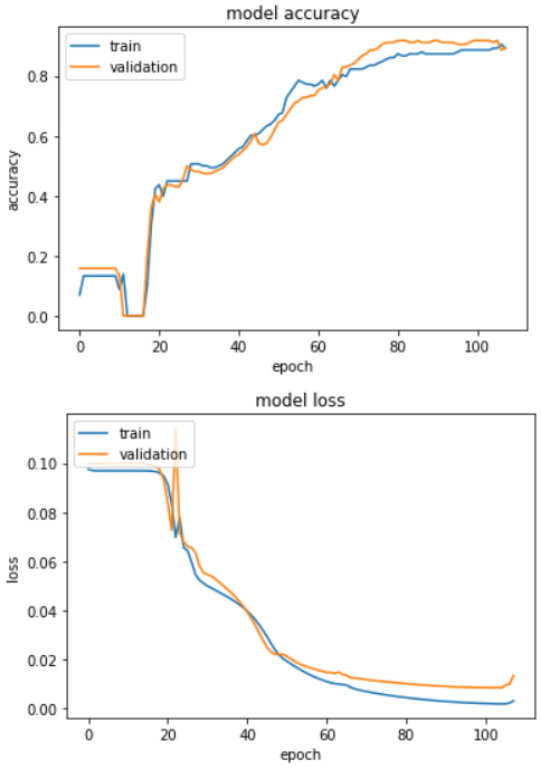


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 48, 26 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 50 epochs.
- Training loss = 0.0101
- Training accuracy = 82%
- Validation loss = 0.0197
- Validation accuracy = 79%
- Testing accuracy = 80%

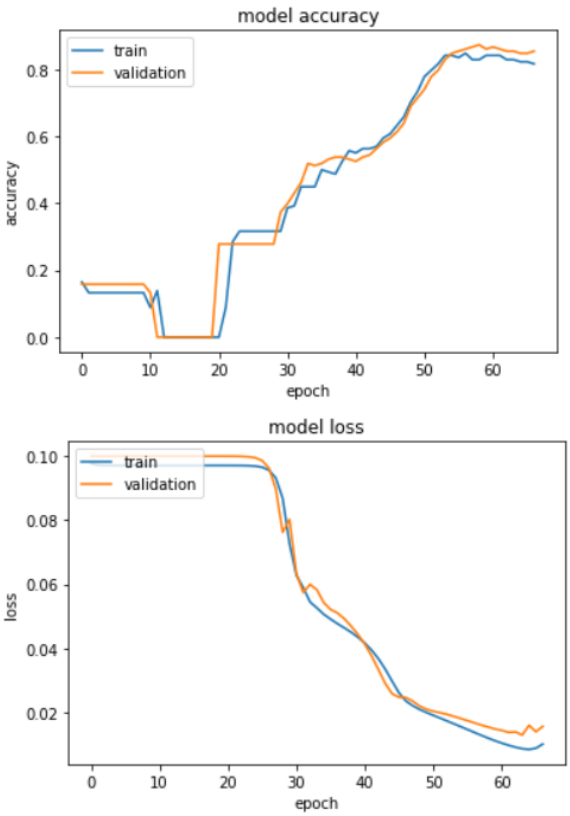


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 48, 24 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 72 epochs.
- Training loss = 0.0072
- Training accuracy = 84%
- Validation loss = 0.0097
- Validation accuracy = 93%
- Testing accuracy = 84%

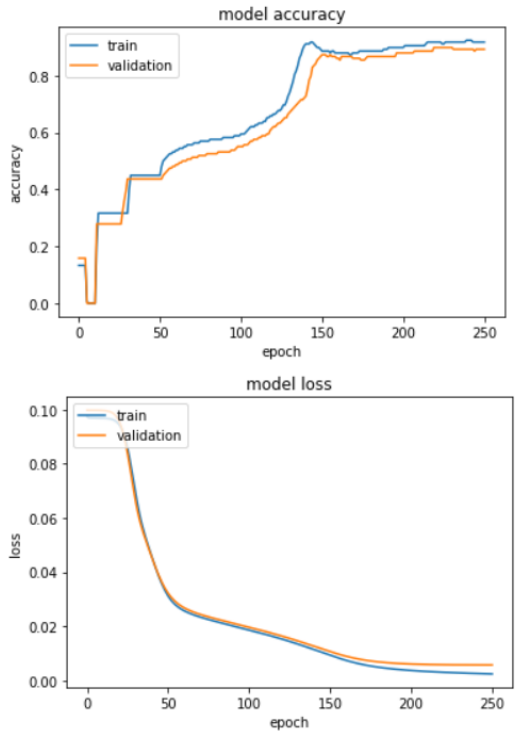
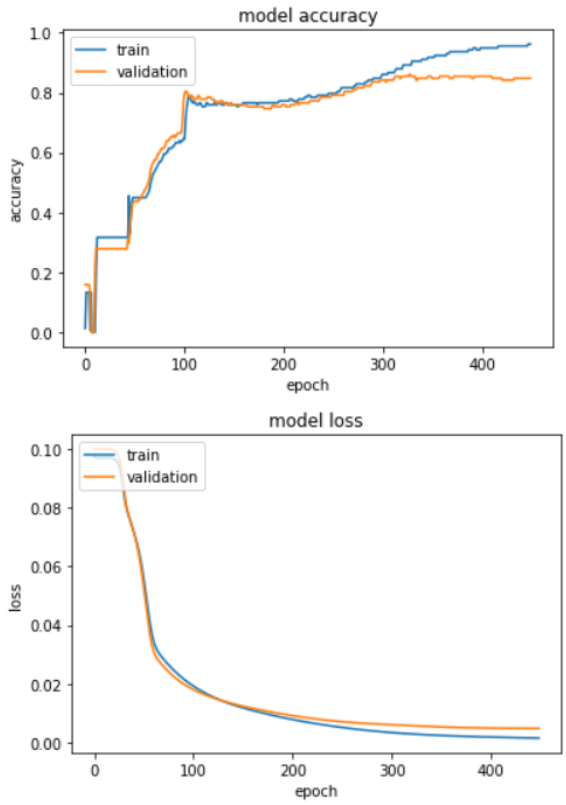
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 48, 22 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 123 epochs. • Training loss = 0.0031 • Training accuracy = 94% • Validation loss = 0.011 • Validation accuracy = 87% • Testing accuracy = 66%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 54, 30 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 9 • # of iterations per epoch: 9 . • Training stopped at 101 epochs. • Training loss = 0.0032 • Training accuracy = 91% • Validation loss = 0.0107 • Validation accuracy = 82% • Testing accuracy = 76%

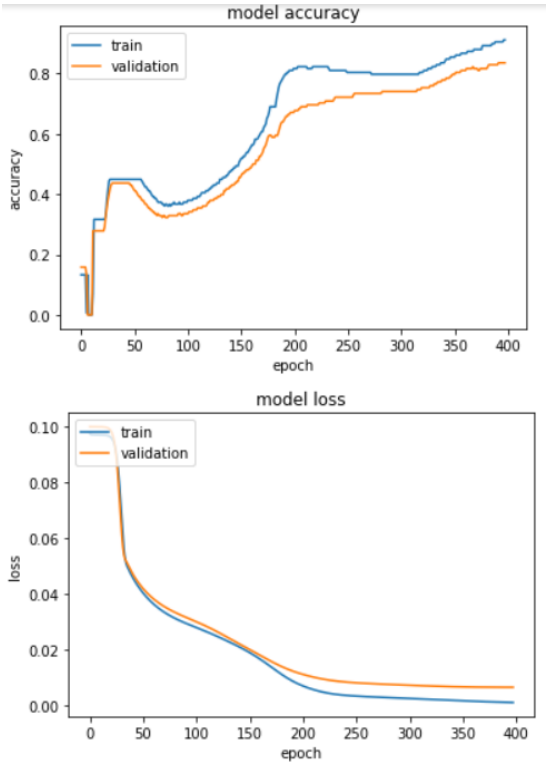
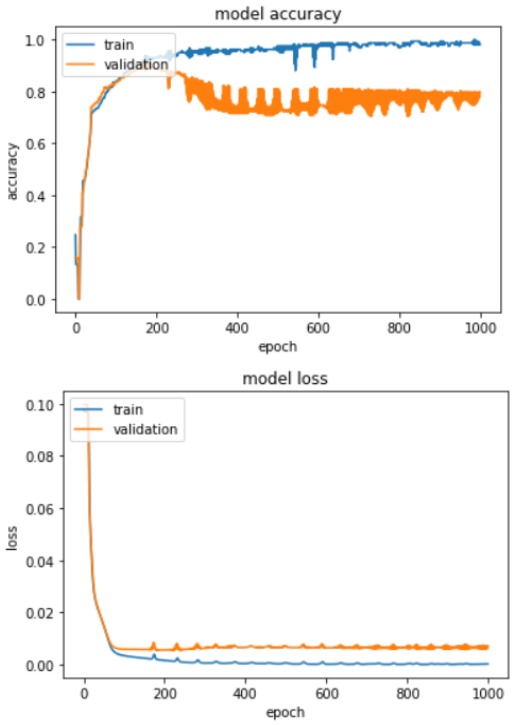


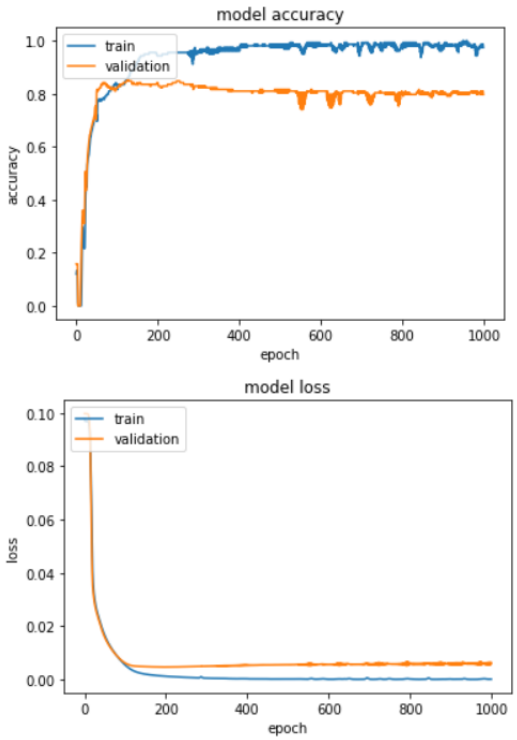
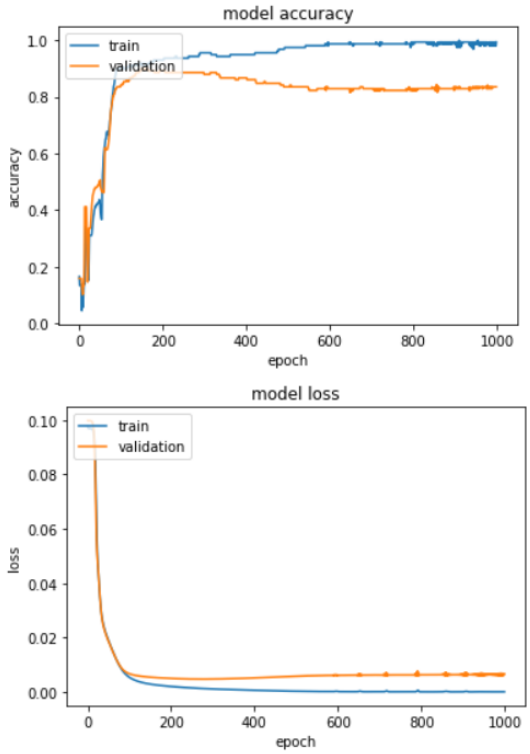
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 54, 42 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 108 epochs.
- Training loss = 0.0032
- Training accuracy = 89%
- Validation loss = 0.0135
- Validation accuracy = 89%
- Testing accuracy = 78%

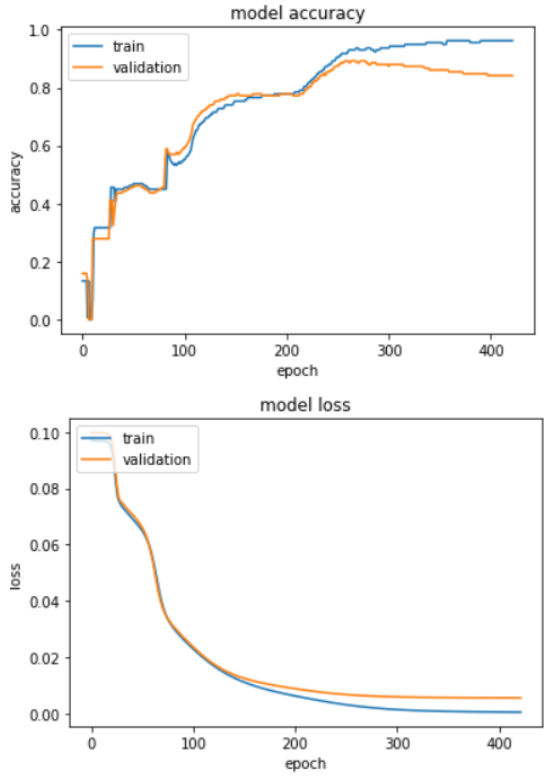
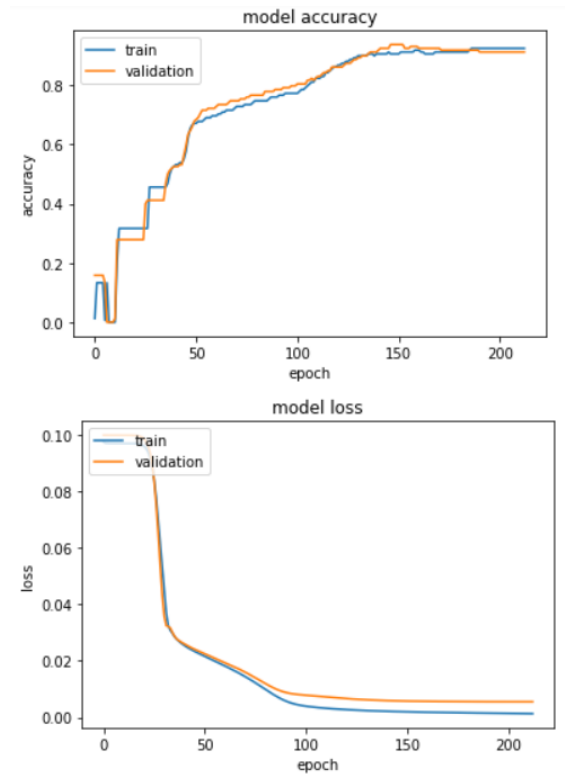


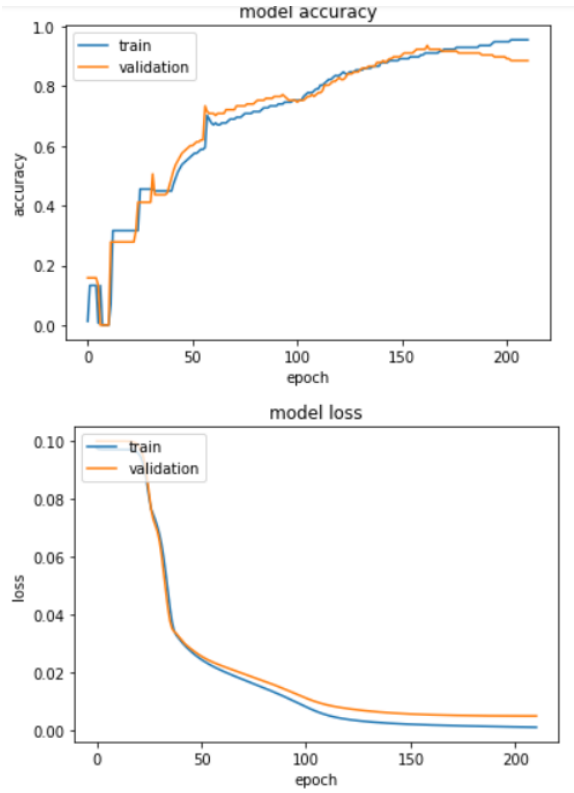
- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers): 54, 34 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(79, 2, 6). (for both input and output, so many-to-many)
- Batch_size = 9
- # of iterations per epoch: 9 .
- Training stopped at 67 epochs.
- Training loss = 0.0104
- Training accuracy = 82%
- Validation loss = 0.0159
- Validation accuracy = 85%
- Testing accuracy = 87%

One-to-One	
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 6, 6 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 21 • # of iterations per epoch: 8 . • Training stopped at 251 epochs. • Training loss = 0.0024 • Training accuracy = 92% • Validation loss = 0.0057 • Validation accuracy = 89% • Testing accuracy = 72%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 6, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 21 • # of iterations per epoch: 8 . • Training stopped at 449 epochs. • Training loss = 0.0016 • Training accuracy = 96% • Validation loss = 0.0048 • Validation accuracy = 85% • Testing accuracy = 70%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers): 7, 5 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(79, 2, 6). (for both input and output, so many-to-many) • Batch_size = 21 • # of iterations per epoch: 8 . • Training stopped at 398 epochs. • Training loss = 0.0011 • Training accuracy = 91% • Validation loss = 0.0065 • Validation accuracy = 84% • Testing accuracy = 70%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):12, 8 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(158, 1, 6). (for both input and output, so one-to-one) • Batch_size = 9 (of the 158 samples) • # of iterations per epoch: 18 . • Training stopped at 1000 epochs. (removed callbacks) • Training loss = 2.5e-04 • Training accuracy = 98% • Validation loss = 0.0072 • Validation accuracy = 80% • Testing accuracy = 63%

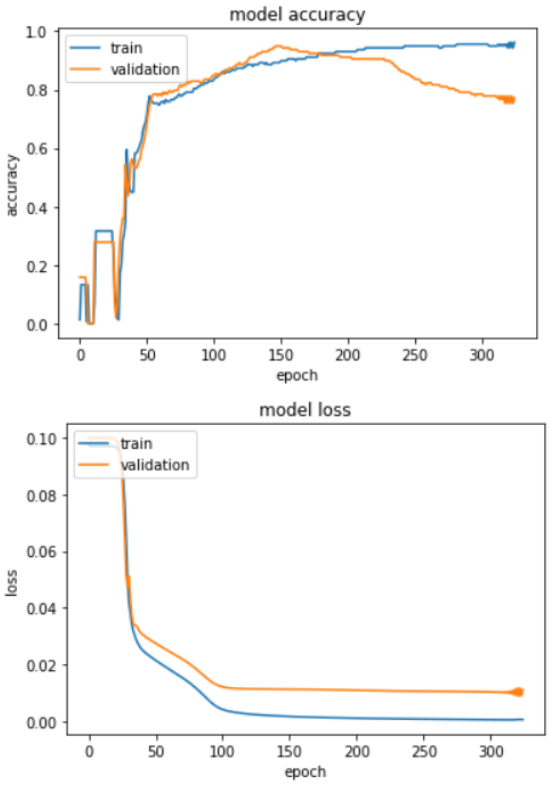
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):12, 8 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(158, 1, 6). (for both input and output, so one-to-one) • Batch_size = 12 • # of iterations per epoch: 14 . • Training stopped at 1000 epochs. (removed callbacks) • Training loss = 1.10e-04 • Training accuracy = 97% • Validation loss = 0.0063 • Validation accuracy = 80% • Testing accuracy = 62%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):12, 8 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(158, 1, 6). (for both input and output, so one-to-one) • Batch_size = 15 • # of iterations per epoch: 11 . • Training stopped at 1000 epochs. (removed callbacks) • Training loss = 7.24e-05 • Training accuracy = 99% • Validation loss = 0.0067 • Validation accuracy = 84% • Testing accuracy = 60%

	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):12, 8 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(158, 1, 6). (for both input and output, so one-to-one) • Batch_size = 21 • # of iterations per epoch: 8 . • Training stopped at 422 epochs. • Training loss = 4.74e-04 • Training accuracy = 96% • Validation loss = 0.0056 • Validation accuracy = 84% • Testing accuracy = 71%
	• Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer. • # of cells (in order of layers):24, 16 and 6. • Learning_rate = 0.0009. • Data has been mean- normalized per feature. • UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ. • Data shape:(158, 1, 6). (for both input and output, so one-to-one) • Batch_size = 21 • # of iterations per epoch: 8 . • Training stopped at 213 epochs. • Training loss = 0.0014 • Training accuracy = 92% • Validation loss = 0.0056 • Validation accuracy = 91% • Testing accuracy = 83%

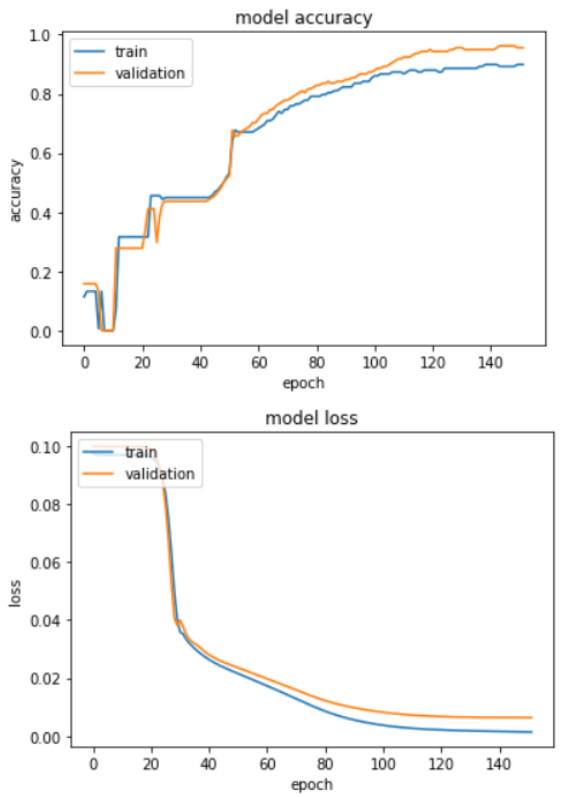


- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):24, 16 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(158, 1, 6). (for both input and output, so one-to-one)
- Batch_size = 21
- # of iterations per epoch: 8 .
- Training stopped at 211 epochs.
- Training loss = 9.62e-04
- Training accuracy = 96%
- Validation loss = 0.0049
- Validation accuracy = 89%
- Testing accuracy = 75%

[Note despite the same parameters, performance is different. This is due to the inbuilt random seed initializer in Keras.]



- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):30, 22 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(158, 1, 6). (for both input and output, so one-to-one)
- Batch_size = 21
- # of iterations per epoch: 8 .
- Training stopped at 325 epochs.
- Training loss = 7.65e-04
- Training accuracy = 96%
- Validation loss = 0.0113
- Validation accuracy = 77%
- Testing accuracy = 65%



- Order of layers: 2 LSTM layers and 1 TimeDistributed Dense layer.
- # of cells (in order of layers):45, 16 and 6.
- Learning_rate = 0.0009.
- Data has been mean- normalized per feature.
- UX, UY, UZ, PX, PY, PZ, UVX, UVY, UVZ, PVX, PVY and PVZ.
- Data shape:(158, 1, 6). (for both input and output, so one-to-one)
- Batch_size = 21
- # of iterations per epoch: 8 .
- Training stopped at 152 epochs.
- Training loss = 0.0014
- Training accuracy = 90%
- Validation loss = 0.0064
- Validation accuracy = 96%
- Testing accuracy = 83%

5.1.8 Weight Interpretation

The weights retain the NN's inclinations when processing the data and is what is transferred from one script to another. The weights define what the NN algorithm is based on how each neuron processes and 'learns' from the data given. The LSTM has four gates and each gate has its associated weight, along with overall LSTM weight and a RNN (recurrent) weight (or kernel) for each neuron. The weights are updated through back and forward propagation along each iteration in every epoch of the training process. Thus, the final updated weights are the last arrays at the final epoch once the NN has stopped training.

The following examples visually depict the LSTM calculations that are used in Keras when determining the total parameters. The total parameters are weights + biases.

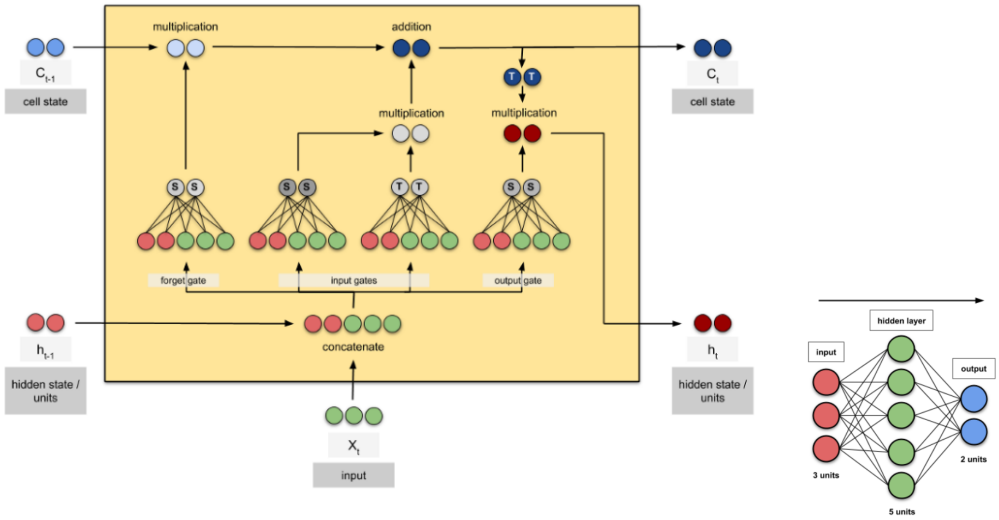


Figure 169: *Keras model of LSTM cell [44]. Note the hidden state and the memory state output dimensions equal to that of the output layer (2). While input is the number of features within the input shape. Keep in mind that there is a distinction between dimensionality and features. Dimensionality refers to the number of elements, like a 2-D or 3-D, in a matrix or tensor respectively. Unfortunately, in most documentation they are often used interchangeably.*

```
# define model
timesteps=40          # dimensionality of the input sequence
features=3            # dimensionality of each input representation
in the sequence
LSTMoutputDimension = 2 # dimensionality of the LSTM outputs (Hidden
& Cell states)


input = Input(shape=(timesteps, features))
output= LSTM(LSTMoutputDimension)(input)
model_LSTM = Model(inputs=input, outputs=output)

model_LSTM.summary()


Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 40, 3)]           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 2)                 48
=================================================================
Total params: 48
Trainable params: 48
Non-trainable params: 0
_________________________________________________________________
```

LSTM parameter number = 4 × ((x + h) × h + h)

LSTM parameter number = 4 × ((3 + 2) × 2 + 2)

LSTM parameter number = 4 × (12)

LSTM parameter number = 48

Figure 170: *LSTM parameter calculations [44]. Note the similarities between the table shown and the one in the actual script for this study. The model.summary() command is used to create this table. It calculates and displays the total number of parameters (weights + bias) per layer. Remember that the LSTM has two different weights, U and W, and has four gates (hence, the multiplied four).*

```
W = model_LSTM.layers[1].get_weights()[0]
U = model_LSTM.layers[1].get_weights()[1]
b = model_LSTM.layers[1].get_weights()[2]

print("W", W.size, ' calculated as 4*features*LSTMoutputDimension ',
4*features*LSTMoutputDimension)
print("U", U.size, ' calculated as
4*LSTMoutputDimension*LSTMoutputDimension ',
4*LSTMoutputDimension*LSTMoutputDimension)
print("b", b.size , ' calculated as 4*LSTMoutputDimension ',
4*LSTMoutputDimension)
print("Total Parameter Number: W+ U + b " , W.size+ U.size + b.size)
print("Total Parameter Number: 4 × ((x + h) × h +h) " , 4*
((features+LSTMoutputDimension)*LSTMoutputDimension+LSTMoutputDimensi
on))


W 24  calculated as 4*features*LSTMoutputDimension  24
U 16  calculated as 4*LSTMoutputDimension*LSTMoutputDimension  16
b 8  calculated as 4*LSTMoutputDimension  8
Total Parameter Number: W+ U + b  48
Total Parameter Number: 4 × ((x + h) × h +h)  48
```

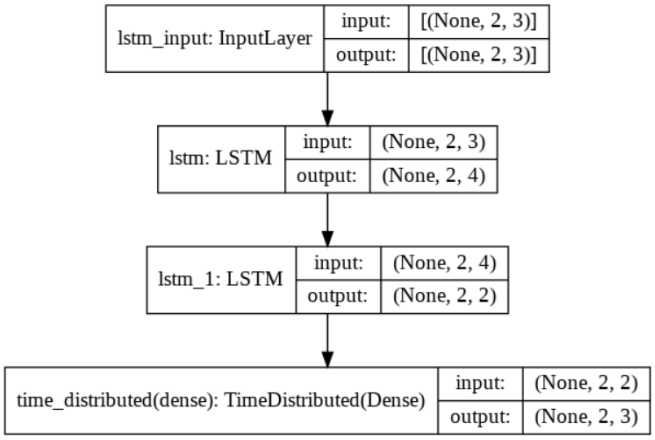
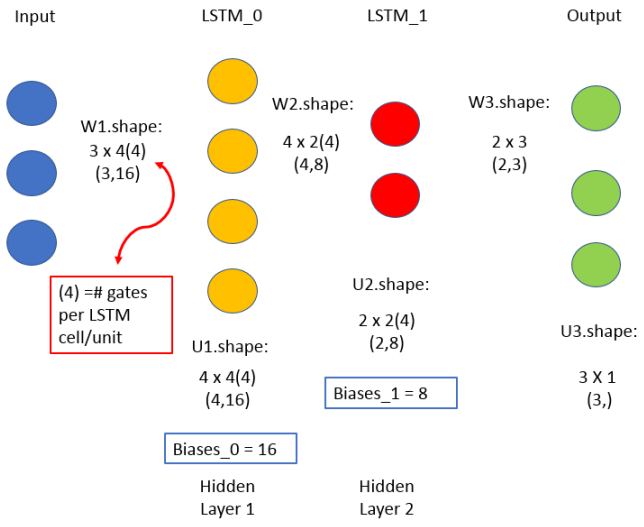
***Figure 171**: Getting the parameters in Keras [44]. Note that [0]=W, [1]=U and [2] = b; where 'W'= LSTM weights, 'U'= recurrent weights, and 'b' = biases. A similar process was done in this study's script. LSTM uses the same 'W', 'U' and 'b' for all time-steps [44]. Every back-prop will produce a new set of parameters and the last epoch is considered the final version. The final epoch's parameters is what is shown in the model.get_weights command.*

Thus, it stands to reason that the last epoch's weights or a 'learned representation' matrix created by an encoder are possible options to 'decode' the perturbations. Due to the time constraints, only the former was looked into.

Table 5.5. Parameters for three features.

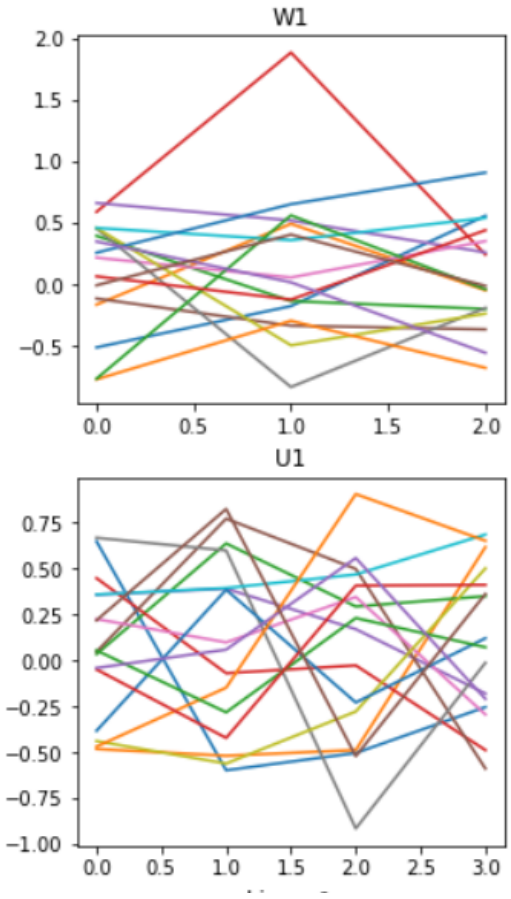
Visuals	Notes

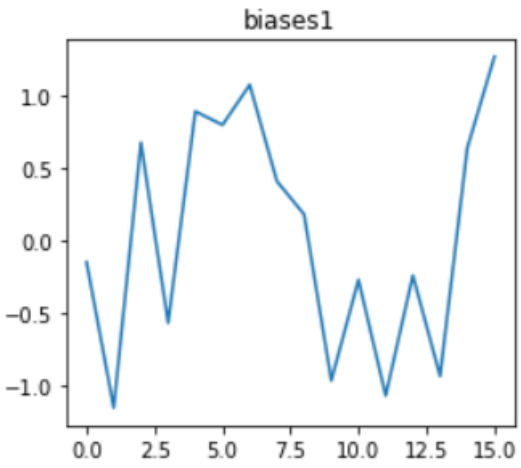
`model.summary()`
 	
`Model: "sequential"`
`_____`
`Layer (type) Output Shape Param #`
`=====`
`lstm (LSTM) (None, 2, 4) 128`
`_____`
`lstm_1 (LSTM) (None, 2, 2) 56`
`_____`
`time_distributed (TimeDistri (None, 2, 3) 9`
`=====`
`Total params: 193`
`Trainable params: 193`
`Non-trainable params: 0`
`_____`
	• Model summary for many-to-many and 3 features.
	• There are 4 total layers, input (sequential models don't show it), LSTM_0 (1st hidden layer), LSTM_1 (2nd hidden layer), and output (time-distributed dense).
	• Input shape: (None, 2, 3), where None refers to all the samples, which in this case is 79; 2 is the number of timesteps

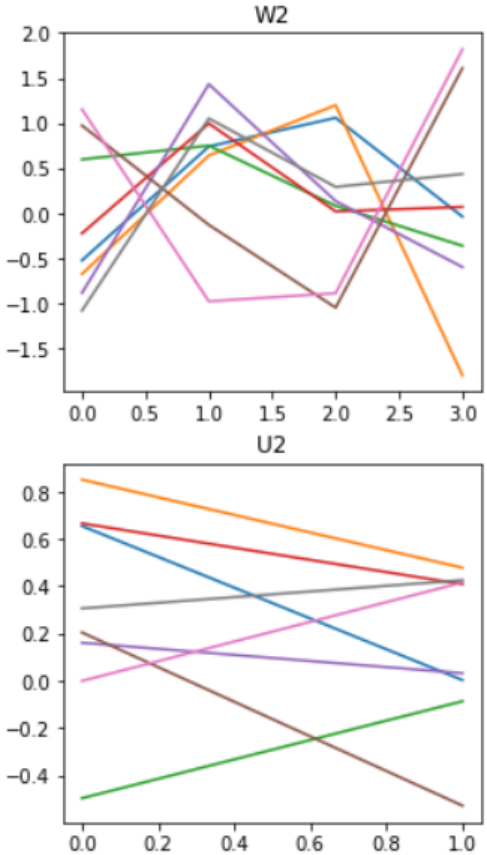
	per sample and 3 is the number of features per timestep. • LSTM_0: (None, 2, 4), b/c it has 4 units/cells and 2 time steps per sample. None = All= 79. • LSTM_1 : (None, 2, 2); b/c it has 2 units/cells and 2 time steps per sample. None = All = 79. • Output: (None, 2, 3); b/c it has 3 units/cells (it has to b/c the total number of features for the label or Y dataset is 3) and 2 time steps for every sample. None = All = 79.
	• Topology of the NN model that was summarized by the table presented in the previous row.
	• model.get_weights() results visualized. • Note that 'W' weights is a relationship or branches between the nodes/cells/units of each layer; while 'U' is the recurrent weight for each node within the layer itself. • Only the hidden layers have biases.

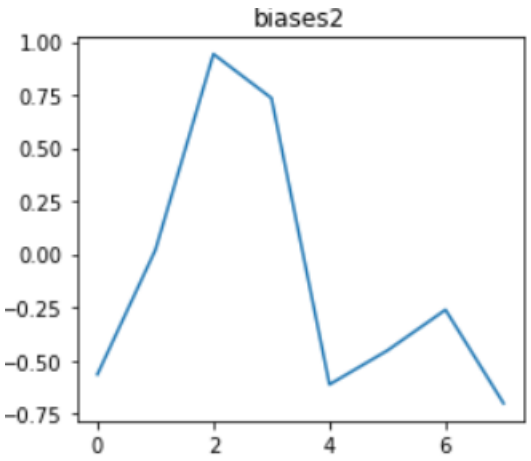
Thus, the final conclusion is that a generalized weight distribution is calculated through every epoch and the final (weight distribution) is per layer at the last epoch and not per timestep. It is a final generalization of all the time steps rather than a distribution presented for every timestep. This can be seen in the sections 'Create Model' and 'Weight Interpretation' in the .ipynb scripts. Weights are just the slopes (or inclinations of each node with respect to the inputs) of the learning process (as was discussed earlier in Chapter 3) and do not give information on what the solar perturbation matrix is. Thus, the Encoder's 'learned representation' matrix may be the solution to capture such information. Another plausible alternative are the biases. (Note: for all plots in Table 6, the X-axis is the indices and the Y-axis is the data (slope) values. These values can be viewed in the associated script (.ipynb) files in Appendix B.)

Table 5.6. Parameter line plots for 3 features.

Parameter Plots	**Notes**
	● 1st hidden layer's parameters ● 'W1' are the weights between the input and 1st hidden layer. ○ Shape: (3,16) ○ There are 16 different plots ○ Plots per column ○ Each column has 3 data points, b/c 3 refers to the number of units in the input layer. Which happens to be 3 (b/c there are 3 features). ○ X-axis is index : 0, 1, 2 ..etc. (since there are 3 data points per column, max index value is 2) ○ Y-axis is the data point value. (slope value). ● 'U1' is the recurrent weights within the 4 unit LSTM layer. ○ Shape: (4,16) ○ There are 16 different plots ○ Plots per column ○ Each column has 4 data points, b/c 4 refers to the number of units within that layer. Remember that a recurrent weight only loops between the

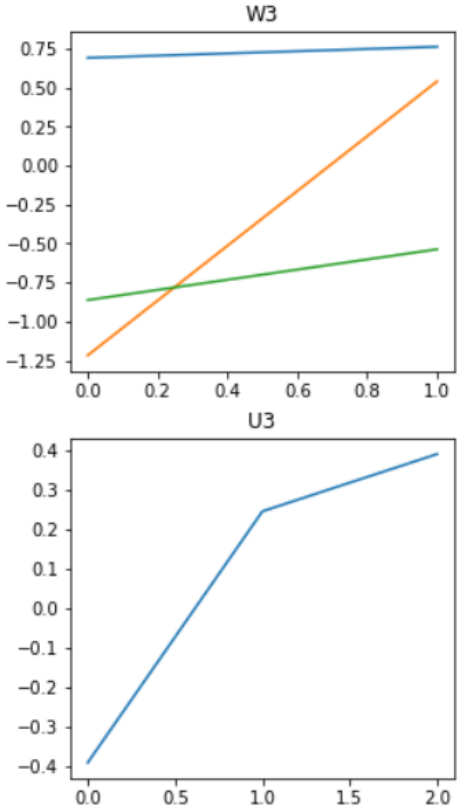
<table>
<tr>
<td> 
</td>
<td>nodes within a layer.

● ‘Biases1’ are the biases of this layer.
○ Shape: (16,)
○ Plots one whole array or vector column.
○ There are 16 points within that column.
○ The 16 points are due to the 4(4), where the 1st ‘4’ = # of units in this layer and the 2nd ‘4’ = # gates. Remember, that in a LSTM cell, each unit has 4 gates.</td>
</tr>
<tr>
<td> 
</td>
<td>● 2nd hidden layer’s parameters
● ‘W2’ are the weights between the 1st and 2nd hidden layers.
○ Shape: (4,8)
○ There are 8 different plots
○ Plots per column
○ Each column has 4 data points, b/c 4 refers to the number of units in the 1st hidden layer. And ‘8’ refers to 2(4), where ‘2’ = # units in the 2nd hidden layer and ‘4’ = # gates in each.
● ‘U2’ is the recurrent weights within the 2 unit LSTM layer.
○ Shape: (2,8)
○ There are 8 different plots
○ Plots per column
○ Each column has 2 data points, b/c 2 refers to the number of units within that layer. Remember that a recurrent weight only loops between the nodes within a layer.</td>
</tr>
</table>

<table>
<tr>
<td> 
</td>
<td>

‘Biases2’ are the biases of this layer.

Shape: (8,)
Plots one whole array or vector column.
There are 8 points within that column.
The 8 points are due to the 2(4), where ‘2’ = # of units in this layer and ‘4’ = # gates. Remember, that in a LSTM cell, each unit has 4 gates.

</td>
</tr>
<tr>
<td> 
</td>
<td>

Output layer’s parameters
‘W3’ are the weights between the 2nd hidden and output layers.

Shape: (2,3)
There are 3 different plots
Plots per column
Each column has 2 data points, b/c 2 refers to the number of units in the 1st hidden layer. And ‘8’ refers to 2(4), where ‘2’ = # units in the 2nd hidden layer and ‘4’ = # gates in each.

‘U3’ is the recurrent weights within the output dense-time-distributed layer.

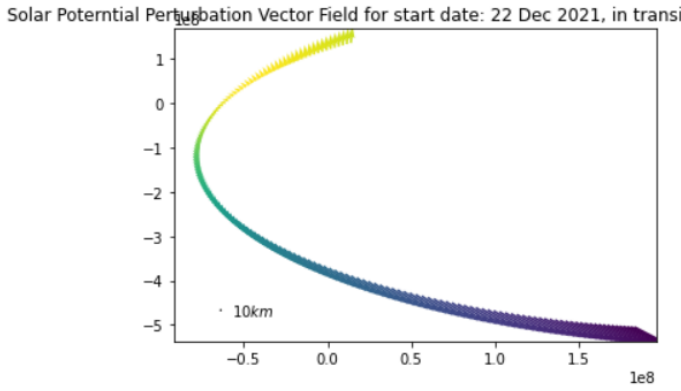
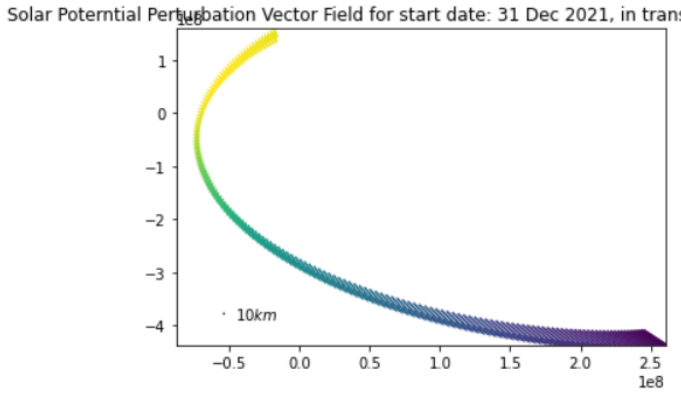
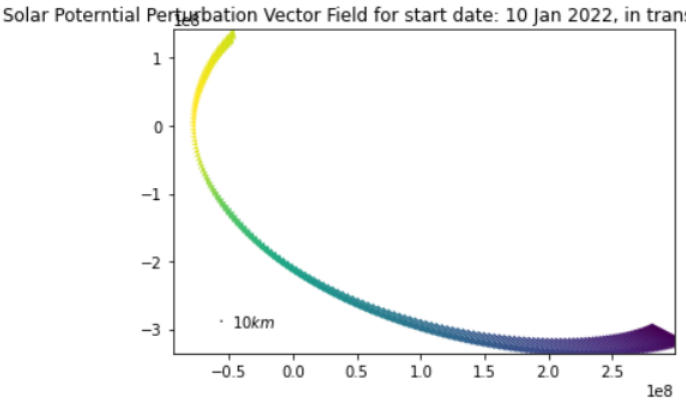
Shape: (3,)
There is 1 plot
Plots per column
Column has 3 data points, b/c 3 refers to the number of units within that layer. Remember that a recurrent weight only loops between the nodes within a layer.

The output layer has no biases (nor does the input layer).

</td>
</tr>
</table>

In the meantime, the vector plotting of the solar perturbation is done via quiver matplotlib tools.

Table 5.7. Perturbation vector plots.

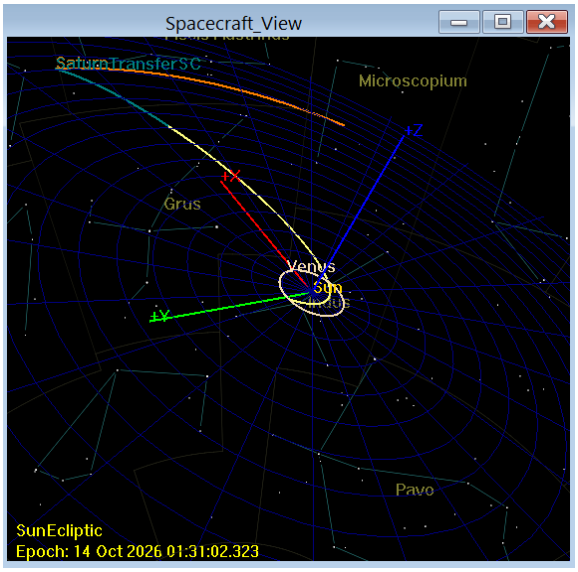
Vector Plots	Notes
	• Position only
	• Position only
	• Position only

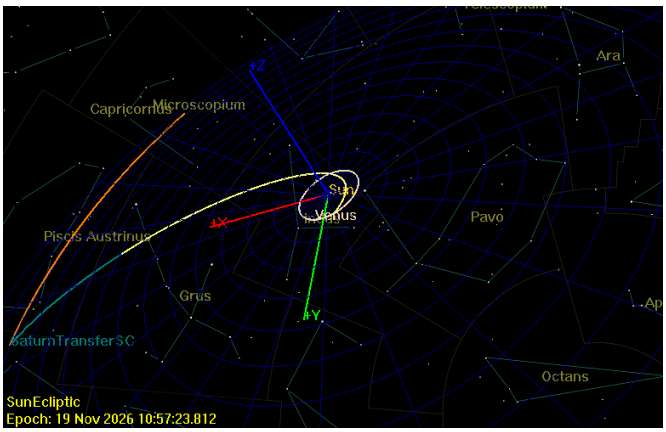
5.2 Burn Analysis at Two Year Mark

5.2.1 Results

This section will display the burn results at the two year mark of different GMAT runs, with respect to the three different initial state vectors, of the perturbed swing-by trajectory. The initial state vectors are defined as 'VMAG' and 'RMAG' in the table below. RMAG is the position magnitude of the 'Pos1' (in Matlab as discussed in Chapter 4) vector that is along Venus' periapsis and VMAG is the associated velocity.

Table 5.8. Two Year Burn Results

Visuals	Notes
 Solver Window - Target 'Target1_dv_toSaturnPeri' DefaultDC {SolveMode ... Control Variable | Current Value | Last Value | Difference TOI_Sun_dvtoSaturn.Element1 | 0.5498559189989783 | 0.5498559189989783 | 0 Constraints | Desired | Achieved | Difference (==) SaturnTransferSC.SaturnI | 2433165 | 2433164.345972237 | -0.6540277628228068 (==) SaturnTransferSC.SaturnI | 2689198 | 2689195.78091422 | -2.219085779972374 CONVERGED	• Start Date: 01 Jan 2022 00:000 • Initial VMAG = 47.78 km/s • Initial RMAG = 1.076e8 km • $\Delta v_{tot} = 0.55\ km/s$ • B-plane targeting • BdotR = 2,433,164 km • BdotT = 2,689,196 km • The B-vector lies in the positive region in the TR-plane. (4th quadrant)

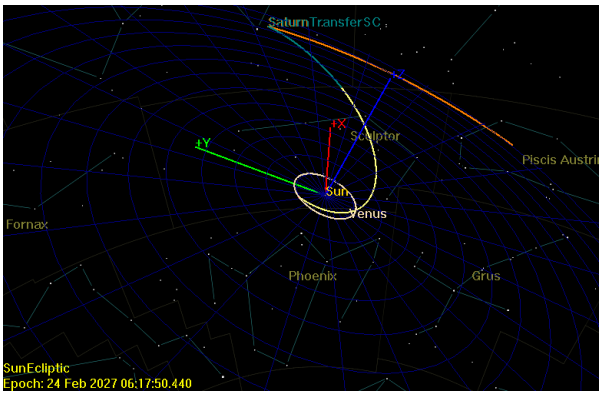


Solver Window - Target 'Target1_dv_toSaturnPeri' DefaultDC {SolveMode ...

Control Variable	Current Value	Last Value	Difference
TOI_Sun_dvtoSaturn.Element1	0.8035781091977892	0.8035781091977892	0
Constraints	**Desired**	**Achieved**	**Difference**
(==) SaturnTransferSC.SaturnI	1537345	1539692.348271167	2347.348271167371
(==) SaturnTransferSC.SaturnI	3699507	3699736.837136516	229.8371365163475

CONVERGED

- Start Date: 10 Jan 2022 00:00
- $\Delta v_{tot} = 0.8\ km/s$
- Initial VMAG = 47.82 km/s
- Initial RMAG = 1.075e8 km
- B-plane targeting
- BdotR = 1,539,692 km
- BdotT = 3,699,737 km
- The B-vector lies in the positive region in the TR-plane. (4th quadrant)
- Lies farther away from Saturn compared to the previous run.



Solver Window - Target 'Target1_dv_toSaturnPeri' DefaultDC {SolveMode ...

Control Variable	Current Value	Last Value	Difference
TOI_Sun_dvtoSaturn.Element1	1.321469936647287	1.321469936647287	2.220446049250313e-16
Constraints	**Desired**	**Achieved**	**Difference**
(==) SaturnTransferSC.SaturnI	-1370155	-1370071.158126924	83.84187307558022
(==) SaturnTransferSC.SaturnI	5272961	5273110.532289706	149.5322897061706

CONVERGED

- Start Date: 20 Jan 2022 00:00
- $\Delta v_{tot} = 1.32\ km/s$
- Initial VMAG = 47.86 km/s
- Initial RMAG = 1.0748e8 km
- B-plane targeting
- BdotR = -1,370,071 km
- BdotT = 5,273,111 km
- B-vector lies in the 1st quadrant in the TR-plane.
- Is the farthest from Saturn, in comparison to the previous runs.

Chapter 6 - Conclusions and Future Works

6.1.Conclusions and Future Works

6.1.1 Conclusions

- For the two year burn targeting, the further the spacecraft's initial condition was from the sun, the higher the burn required to achieve (some point) along Saturn's periapsis.
- Achieved testing, validation and training accuracy in the 90th percentile for the 3 features in many-to-many. Ideal cell and batch_size combinations were: 4, 2 and 3, and batch_size=12. The cell numbers are in order of the layers.
- Achieved training and validation accuracies in the 90th percentile but testing accuracy in the 80th percentile for the 6 features in many-to-many (and one-to-one). The ideal combinations for cells and batch_size were not found due to the fact that more data was needed to reach there. Appendix D sheds some light on cell number calculations that may lead to the optimal cell combinations.
- A successful NN model is one that does not overfit and acquires accuracies for all: training, validation and testing to be in the 90th percentile.
- Ideal NN model (for either 3 or 6 features and 158 data points) consists of 2 hidden (LSTM) layers and 1 TimeDistributed Dense layer.
- The testing accuracy is dependent on the size of the gap between the accuracy and training curves in the accuracy plot; the bigger the gap the less of a testing performance and vice versa.
- Callbacks are used to stop the training to continue once overfitting (gap between training and validation accuracy curves is huge) occurs steadily over 3 to 5 epochs. This is important because the operator should not want the NN model to over-familiarize with the training data.
- LSTM input gates' activations do change based on user input.
- Should have started the training process using the formula produced in Appendix D.
- Objective was to have loss at 10^{-10} but was only able to achieve, so far at 10^{-5}. However, upon closer inspection, loss may not be the same as precision. Thus, there is a predictive accuracy code that has been added to the script which gives the operator an idea of how well the algorithm predicts in comparison to the actual labels. When the training, accuracy and testing values are in the 90th percentile the predictive accuracy ranges between 60 % - 90 %.
- More often than not, needed to clear cache settings after every reload of a NN model to reduce chances of inaccuracies in different runs.
- More features require more data (time steps) and more cells per hidden layer. However, there is a formula that can be used to calculate the number of necessary neurons by the number of input and output cells. This calculation can be viewed in Appendix D.

Remember, due to the flexible and abstract nature of the NN, these calculations can aid as a starting basis only. There is a lot of trial and error in figuring out the right combination of neurons and batch sizes.

- Overfitting occurs when there is a large gap between training and validation curves in the accuracy plot. An early stopping callback is used to prevent the training from continuing once overfitting occurs. It is ideal to have the accuracy values for training, validation and testing to be as closely equal as possible.
- One-to-one and many-to-many model performances improved with the addition of a TimeDistributed(Dense) layer, turning return_sequences = True, keeping the input gates' activation functions = tanh, having three separate datasets for training, validation and testing, normalizing every feature appropriately, and adjusting batch size and cell numbers (once datashape has been finalized).
- The ideal number of iterations for 158 data points is between 6 and 9. (Batch size = 9, 12 and 15 for many-to-many but batch_size=21 for one-to-one.)
- The ideal number of hidden layers for 158 data points is 2. (The smaller the datasets the fewer hidden layers it will need. Usually 2 hidden layers is enough for most 'small' models.)
- Number of ideal iterations remains consistent with the total number of data points.
- LSTM doesn't need additional regularizers. (The fewer the layers and cells, the better)
- Selecting between one-to-one vs many-to-many for improvement in performance really depends on the data. But many-to-many has more flexibility.
- Back-end's random seed initializer, as well as GPU traffic and incomplete reloading of .ipynb file, impedes reproducibility of training results. However, may that be, there still is a similarity to the values running within the same conditions. If the results happen to be completely different, then either the .ipynb was not completely reloaded or needs to be reloaded again.
- Make sure to collect the total number of time steps for any data set to not be a prime number. That way the operator has flexibility in shaping the data and improving the performance of the model.
- Increasing batch_size for one-to-one models, along with all the aforementioned parameters in the previous bullet, helps decrease the noise/fluctuations in the training process.
- Having different data sets for training, validation and testing proved to have better performance results than using the splitting tool provided by sklearn. This splitting tool is meant to divide a dataset by a certain ratio: one for training and another for testing/validating.
- Data shaping and normalization techniques had a bigger impact than any other elements in improving model accuracy. (with Adam as optimizer and learning_rate = 0.0009)
- It is advisable to keep the learning_rate and choice of optimizer constant throughout the training process.

- Appendix D lists some calculations that one can use to find the optimal combinations of hidden layer cells and perhaps even for a selected batch_size.
- Adjusting the number of parameters (weights + biases) affects the overall model's performance in regards to over or under fitting. (Hint : biases = # of units x gates for LSTMs only and generally are found within the hidden layers.)
- If prefer one-to-one it is advisable to have a batch_size > 1. So, as to have a certain number of iterations done within that batch at every epoch. However, keep in mind that one-to-one has limited flexibility (in data shaping) when compared to many-to-many models.
- The type of data and what the model is expected to predict impacts all parameters involved.
- Increasing the number of features affects, in direct proportionality to, the number of cells per layer.
- There is an inverse correlation between the number of hidden cells with the number of epochs the NN model takes before overfitting occurs.
- Hidden and cell states' output shapes in LSTMs are equal to the number of output units. (Remember, that what is input or output changes with respect to which layer the magnifying glass is on.)
- Testing accuracy is directly related to the behavior of the validation accuracy.
- Can use GPU or TPU (especially when high cloud traffic prevents operators from using GPU) to train NN models. (Can select from 'Notebook Settings')
- Sometimes when the NN model outputs unusual results in the training process (after hitting the run button for the 'model.fit' cell command), it is best to do 'factory reset' or 'restart runtime', clear all cache, reload the .ipynb file and re-run the file. Re-doing it twice as a sanity check is advisable. If after all that, the training results are still funky, then the combination of cells and batch_size aren't satisfactory. It is best to change batch_size before changing the number of cells per hidden layer.
- An indicator that an .ipynb file has been successfully re-loaded is the 'Connect' on the upper left hand corner. (Make sure it does not have Re-connect before running another model.)
- Dimensions do not equal features. Dimensions is the shape of the data's tensor (for instance LSTM requires 3D data shaped tensors, so input_dim = 3), while features are the number of classes (in this case it was either 3 or 6).

6.1.2 Future Works

- Run a NN analysis for the velocity only using the position only (3 featured) model.
- Achieve the 90th percentile for testing, training and validation accuracy curves for the 6 features.

- Verify whether ensemble (bagging technique) or SDM can enhance performance of the NN model for 6 features.
- Investigate how training and validation loss should be interpreted differently than precision (tolerance). (Which was initially, mistakenly, perceived to be.)
- Functional model instead of sequential modeling.
- Investigate, further, perturbation readings thru weight and bias plots.
- Training with batch_shape instead of input_shape to turn on stateful and return_state. (requires functional modeling set up). It also aids in bringing in more trajectories to train within a single epoch. (Basically every batch_size will have batch_shape)
- Improve the overall accuracy of the model during the testing phase (not just in the training section of the learning process).
- Introduce more data for the six features analysis to improve accuracy scores and its consistency regardless of seed initializer.
- Model needs to be proven robust regardless of random seed initializers.
- Comparison between unit and mean normalization of data with respect to performance.
- Uploading more of the 3-set trajectories (training, validation and testing) to the NN model to improve accuracy. This can be achieved by replacing input_shape with batch_shape.
- Investigating whether bidirectional LSTM layers and encoder-decoder (or LSTM Autoencoders) will impact performance.
- NN burn optimization of flybys.
- Extending the focus of the NN and space to alternative areas and using other variables within orbital dynamics as features.
- Burning NN's weights to a SD card or another flash memory to a computer's microprocessor and its memory chips. (Nelson Wong's thesis, *"On Clustering Low-Cost SoC FPGA Devices for Deep Learning Inference Applications*", delves specifically into this topic. An introductory abstract of this can be viewed in Appendix E.)
- Infusing NN to an automated controller aboard a spacecraft.
- Testing spacecraft's NN controller using reinforcement or unsupervised learning.

REFERENCES

[1] Joseph H. M., "Lecture Notes for AST 5622 Astrophysical Dynamics," *Dept. of Astronomy and Physics, Saint Mary's University*, 2006.
URL:chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://gemelli.spacescience.org/~hahnjm/ast_608/2006spring/3body.pdf

[2] Wyatt, "Lecture 2 Planetary System Dynamics," *StuDocu* [online database], The Chancellor, Masters, and Scholars of the University of Cambridge, 2019, URL:
chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://people.ast.cam.ac.uk/~wyatt/lecture2_planetarysystemdynamics.pdf [cited 2018 - 2019].

[3] Mathis, M., "Inventions and Deceptions - Hill Sphere," *Malaga Bay* , URL:
https://malagabay.wordpress.com/2012/11/03/inventions-and-deceptions-hill-sphere/ [cited 3 November 2012]

[4] National Aeronautics and Space Administration, "NASA's Mars Exploration Program,"Porkchop" is the First Menu Item on a Trip to Mars," *NASA's Mars Exploration Program,*
URL:https://mars.nasa.gov/spotlight/porkchopAll.html#:~:text=Porkchop%20plot%2C%20that%20is.,Mars%20or%20any%20other%20planet. [cited 18 December 1987].

[5] Nagyfi, R., "The differences between Artificial and Biological Neural Networks," *towards data science* [online database]
URL:https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7 [cited 04 September 2018].

[6] Wertz, James R., and Larson, Wiley J., *Space Mission Analysis and Design*, 3rd ed., Space Technology Library, Microcosm Press, Kluwer Academic Publishers, 1999.

[7] Morgan, P., "Cassini Spacecraft-DSN Communications, Handling Anomalous Link Conditions, and Complete Loss-of-Spacecraft Signal," *Intech Open*, URL: 10.5772/intechopen.72075 [cited 10 December 2017]

[8] NASA , "Description and Flight Test Results of the NASA F-8 Digital Fly-by-Wire Control System," *A Collection of Papers from the NASA Symposium on Advanced Control Technology,* Flight Research Center, Edwards, California, July 1974.
URL:https://www.nasa.gov/centers/dryden/pdf/87858main_H-853.pdf

[9] Tomayk, E. J., "Computers Take Flight: A History of NASA's Pioneering Digital Fly-by-Wire Project," *NASA History Office* or *NASA Technical Report Server*, URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20050157919.pdf [cited 23 August 2013]

[10] Stamp, M., "Deep Thoughts on Deep Learning," Computer Science Department at San Jose State University., San Jose State University (SJSU), San Jose, CA, 2019. URL:
chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.cs.sjsu.edu/~stamp/RUA/ann.pdf

[11] Gupta, A., "Evolution of Convolutional Neural Network Architectures," *Medium* [online database],
URL:https://medium.com/the-pen-point/evolution-of-convolutional-neural-network-architectures-6b90d067e403 [cited 16 May 2020].

[12] Li, F., Johnson, J. and Yeung, S., "CS231n: Convolutional Neural Networks for Visual Recognition," Computer Science Department, Stanford University, URL: http://cs231n.stanford.edu/syllabus.html [cited 04 April 2017].

[13] Smet, S. D, "On the design of solar gravity driven planetocentric transfers using artificial neural networks," Ph.D. Dissertation, Ann and H.J Smead Department of Aerospace Engineering Sciences, University of Colorado, 2018.

[14] Parker, J. S., Scheeres, D. J., Smet, S. D, "Systematic Exploration of Solar Gravity Driven Orbital Transfers in the Martian System using Artificial Neural Networks," *2018 AAS/AIAA Astrodynamics Specialist Conference*, AAS 18-216, Snowbird, Utah, August 31, 2018.

[15] Howell, K. C. LaFarge, N. B. Miller, D, and Linares, R., "Guidance for Closed-Loop Transfers using Reinforcement Learning with Application to Libration Point Orbits," *AIAA Journal,* School of Aeronautics and Astronautics, 2020.

[16] Bassil, Y.., "Neural Network Model for Path-Planning of Robotic Rover Systems," Lebanese Association for Computational Sciences (LACSC), *International Journal of Science and Technology (IJST),* Vol.2, No.957, 2012.
URL:http://ejournalofsciences.org/archive/vol2no2/vol2no2_6.pdf

[17] Hunter, J., *Aerospace Engineering 142 Astrodynamics SJSU Course Reader*, SJSU, Aerospace Engineering Department at San Jose State Univ., 2020.

[18] Verma, S., "Input and Output Shape of LSTM in Keras," *Kaggle*,
URL: https://www.kaggle.com/shivajbd/input-and-output-shape-in-lstm-keras [cited 2018]

[19] Jules, K.,and Lin, P., "Artificial Neural Networks Applications: From Aircraft Design Optimization to Orbiting Spacecraft On-board Environment Monitoring," *NASA Technical Report Server,* NASA Glenn Research Center, Rept. NASA/TM-2002-211811, Cleveland, Ohio, August 2002. URL: https://ntrs.nasa.gov/citations/20020081338

[20] Nikis, M., "Astrodynamic Problems of the Space Station," *IAF Astrodynamics Committee, Acta Astronautica,* Vol 17, No. 5, pp. 491-494, 1988, Pergamon Press plc. Dec. 18 1987.
URL:https://www.sciencedirect.com/science/article/abs/pii/0094576588901452

[21] Brunton, S., "Data Visualization: Types of Data," URL:
https://www.youtube.com/watch?v=vvgh4sqWisA&list=PLMrJAkhIeNNQV7wi9r7Kut8liLFMWQOXn&index=18 [cited 5 June 2019]

[22] NASA, "GMAT Mathematical Specifications Draft," March 6, 2020.
URL:chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/http://gmat.sourceforge.net/docs/R2020a/GMATMathSpec.pdf [cited 30 October 2020]

[23] Curtis, H. D., *Orbital Mechanics for Engineering Students*, Elsevier, Embry-Riddle Aeronautical University, Daytona-Beach, Florida, 2010.
URL:https://documentcloud.adobe.com/gsuiteintegration/index.html?state=%7B%22ids%22%3A%5B%2214OPvewv08sLiPUdNmsHkGEobs9RZcXf7%22%5D%2C%22action%22%3A%22open%22%2C%22userId%22%3A%221148951022892886336977%22%2C%22resourceKeys%22%3A%7B%7D%7D

[24] Qian B., Su, J., Wen, Z., Jha, D.N, Li, Y., Guan, Y. Puthal, D., and James, P., "Orchestrating Development Lifecycle of Machine Learning Based IoT Applications: A Survey", *ResearchGate* [online journal database],
URL:chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.researchgate.net/profile/Zhenyu_Wen/publication/336642133_Orchestrating_the_Development_Lifecycle_of_Machine_Learning-Based_IoT_Applications_A_Taxonomy_and_Survey/links/5da9968292851c577eb824b6/Orchestrating-the-Development-Lifecycle-of-Machine-Learning-Based-IoT-Applications-A-Taxonomy-and-Survey.pdf?origin=publication_detail [cited Oct. 2019]

[25] Rashidi, H.H, Tram, Tran, N.K, Betts, E.V., Howell, L.P., and Green, R., "Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods", *ResearchGate* [online journal database], Vol. 6,doi: https://doi.org/10.1177/2374289519873088. URL:
chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.researchgate.net/publication/335604816_Artificial_Intelligence_and_Machine_Learning_in_Pathology_The_Present_Landscape_of_Supervised_Methods/fulltext/5d6fb21f4585151ee49b94aa/Artificial-Intelligence-and-Machine-Learning-in-Pathology-The-Present-Landscape-of-Supervised-Methods.pdf?origin=publication_detail [cited Sept. 2019]

[26] Hardij, R., "Predicting arrival times for tankers ships using recurrent neural networks", Master thesis for Data Science: Business and Governance Tilburg School of Humanities and Digital Sciences, Tilburg, the Netherlands, June 2018.

[27] Singh, A., "Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM)", KTH Royal Institute of Technology, School of Information and Technology, Stockholm, Sweden, 2017. URL:
chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/http://www.diva-portal.org/smash/get/diva2:1149130/FULLTEXT01.pdf

[28] Raval, S., "Build a Neural Network in 4 minutes", YouTube [online database],
URL:https://www.youtube.com/watch?v=h3l4qz76JhQ [retrieved 4 July 2020]

[29] Miller, S., “Mind: How to Build a Neural Network (Part One)”, SEGMENT, 2015. URL: https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/ [cited 10 August 2015]

[30] Nur, A.S., Radzi, N.H.M., Ibrahim, A.O., “Artificial Neural Network Weight Optimization: A Review" , *ResearchGate* [online journal database], *TELKOMNIKA [Indonesian Journal of Electrical Engineering],* Faculty of Computer Science and Information Technology, Alzaiem Alazhari University (AAU),Khartoum, Sudan and Soft Computing Research Group, Faculty of Computing, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor, Malaysia, Vol. 12, No. 9, September 2014, pp.6897~6902, doi:10.11591/telkomnika.v12i9.6264. URL: chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.researchgate.net/profile/Abdirashid_Nur3/publication/272566392_Artificial_Neural_Network_Weight_Optimization_A_Review/links/58c7b7dc458515478dcda561/Artificial-Neural-Network-Weight-Optimization-A-Review.pdf?origin=publication_detail

[31] Grover, P., “5 Regression Loss Functions All Machine Learners Should Know”, *Heartbeat,* [online database], URL: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0 [cited 5 June 2018]

[32] Itdxer, “What is batch size in neural network?”, *StackExchange,* URL:https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network [cited 5 April 2019]

[33] Dan, “Linear regression vs decision trees”, *ML Corner,* URL:https://mlcorner.com/linear-regression-vs-decision-trees/#:~:text=So%2C%20what%20is%20the%20difference,dataset%20and%20the%20output%20variable. [cited 24 July 2017]

[34] Stamp, M., “A Revealing Introduction to Hidden Markov Models”, Computer Science Department at San Jose State Univ., San Jose State University (SJSU), San Jose, CA, 2018. URL: chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf

[35] Jurafsky, D., and Martin, J.H, “Speech and Language Processing”, Stanford University, Mountain View, CA, Oct. 2019. URL: chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://web.stanford.edu/~jurafsky/slp3/A.pdf

[36] Ye, A., “11 Essential Neural Network Architectures, Visualized & Explained: Standard, Recurrent, Convolutional, & Autoencoder Networks”, *Medium* [online database], URL: https://medium.com/analytics-vidhya/11-essential-neural-network-architectures-visualized-explained-7fc7da3486d8 [cited 28 June 2020]

[37] Khuong, B., “The Basics of Recurrent Neural Networks (RNNs)”, *Medium* [online journal] , URL: https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f [cited 23 June 2019]

[38] Sanjeevi, M., “Chapter 10.1: DeepNLP — LSTM (Long Short Term Memory) Networks with Math”, *Medium* [online database], URL: https://medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235 [cited 21 January 2018]

[39] Krukrubo, L.A, “Scaling vs. Normalizing Data”, *Medium, TowardsAI* [online journal], URL: https://pub.towardsai.net/scaling-vs-normalizing-data-5c3514887a84 [cited 10 January 2021]

[40] Asaithambi, S., “Why, How and When to Scale your Features”, *Medium* [online journal], URL: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e [cited 03 December 2017]

[41] Usman, M., “Solving Sequence Problems with LSTM in Keras”, *StackAbuse* [online journal], URL: https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras [cited 09 September 2019]

[42] Brownlee, J., “How to use Data Scaling Improve Deep Learning Model Stability and Performance”, *Machine Learning Mastery,* URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ [cited 25 August 2020]

[43] Roy, B., “All About Feature Scaling”, *TowardsDataScience,* URL: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35 [cited 05 April 2020]

[44] Karakaya, M., “LSTM: Understanding the Number of Parameters”, *Medium* [online journal], URL: https://medium.com/deep-learning-with-keras/lstm-understanding-the-number-of-parameters-c4e087575756 [cited 06 Nov 2020]

APPENDICES

Appendix A - Matlab Scripts for the Lambert's Problem : CODE

Github repository link: https://github.com/tbejaoui1/AE_Master_Thesis_Matlab_Code.git

Appendix B - NN Model Training via Google Colab - Jupyter Notebook : CODE

Github repository link: https://github.com/tbejaoui1/AE_Master_Google_Colab.git

Appendix C - Datasets used for NN Model Training via Google Colab - Jupyter Notebook : SPREADSHEETS

Github repository link: https://github.com/tbejaoui1/AE_Masters_Datasets.git

Appendix D - Calculating Number of Neurons per Layer : LINK & FORMULAS

Andrzej Kasperski
University of Zielona Góra

You can also use the geometric pyramid rule (the Masters rule):

a) for one hidden layer the number of neurons in the hidden layer is equal to:

nbrHID = sqrt(nbrINP * nbrOUT)

nbrHID – the number of neurons in the hidden layer,

nbrINP – the number of neurons in the input layer,

nbrOUT – the number of neurons in the output layer.

b) for two hidden layers:

r = (nbrINP/nbrOUT)^(1/3)

nbrHID1 = nbrOUT*(r^2) – the number of neurons in the first hidden layer

nbrHID2 = nbrOUT*r – the number of neurons in the second hidden layer

c) for three hidden layers:

r = (nbrINP/nbrOUT)^(1/4)

nbrHID1 = nbrOUT*(r^3) – the number of neurons in the first hidden layer

nbrHID2 = nbrOUT*(r^2) – the number of neurons in the second hidden layer

https://www.researchgate.net/post/How-to-decide-the-number-of-hidden-layers-and-nodes-in-a-hidden-layer (make sure you are logged out to access all questions)

Example:
Since, this model has two hidden layers, we will use the 2nd equation, which is:

$r = \left(\frac{\#\ input\ cells}{\#\ output\ cells}\right)^{(1/3)}$;

Where:

- #input cells : Isn't specified when using Keras sequential modeling. But the rule of thumb is that the number of units in the input layer is equal to the number of features in the input training data. Thus, in this study, it was either 3 or 6.
- #output cells : It is equal to the number of features in the output dataset. In this study, it was either 3 or 6.

To determine number of cells in 1st hidden layer:
$\#cells\ in\ 1st\ hidden\ layer = (\#\ output\ cells)(r^2)$

To determine number of cells in 2nd hidden layer:
$\#cells\ in\ 2nd\ hidden\ layer = (\#output\ cells)(r)$

However, despite these formulations, there is a lot of controversy and most state that the right combination of neurons and batch_size is usually determined through trial and error.

Also, I have tweaked the above formulation where the #input cells did not just equal the features of the input dataset, but rather the input_shape itself. For example: if input_shape=(2,3), then #of input cells = 2 x 3 = 6. And when calculating the #cells in the 2nd hidden layer would have a (-1) in there. Thus, for example, for the many-to-many option for the 3 features, if using this theory:

- r = (6/3)^(⅓)= 1.26,
- #1st hidden layer = (3)[(1.26^(2))]=4.76 and
- # 2nd hidden layer = (3)(1.26)=3.76 - 1 = 2.76.

Thus, the number of cells would be 4, 2 and 3 for the two LSTM layers and output layer, respectively.

There's even another plausible theory that batch_size can be included in the calculations, where if the batch_size for the above example was 12, then #of input cells = 12 x 2 x 3 = 72. However, judgement is more in favor of just input_shape.

Appendix E - Nelson Wong’s Thesis : ABSTRACT

Tentative Title: "*On Clustering Low-Cost SoC FPGA Devices for Deep Learning Inference Applications*."
Nelson Wong’s description:
The thesis investigates the efficacy of linking multiple sub-$100 system-on-chip field programmable gate array devices to perform inferencing. This exploration involves Xilinx's XC7Z020 and XC7Z010, which contain block RAM (BRAM) and DSP slices scattered across their programmable logic fabric. The DSP slices are leveraged for their multiply-accumulate to efficiently perform vector-matrix multiplication, while block RAM slices cache network parameters to achieve sub-millisecond multi-layer inference (results pending).
I'm still far from the end of this thesis but the above should still hold true by the end. The SD card in the [SD card -> DDR memory -> BRAM cache] pipeline has been adjusted; attaching the cluster to network-attached storage and managing parameter loading over Ethernet made for a more flexible architecture. The XC7Z010 was especially targeted due to its popularity in previous-generation crypto currency miners; the Chinese online retail service AliExpress has been flooded with refurbished boards that use this chip and are very affordable (currently $16.50 each).