UNIVERSIDADE
BEIRA INTERIOR

LSTM Based Trajectory Prediction for Aerospace Vehicle Reentry

(Versão final após defesa)

João Pedro Monteiro Coelho

Dissertação para obtenção do Grau de Mestre em

Engenharia Aeronáutica

(Mestrado Integrado)

Orientador: Prof. Doutor Kouamana Bousson

Dezembro de 2025

Declaração de Integridade

Eu, **João Pedro Monteiro Coelho**, estudante com o número de aluno **a45807** do curso de **Mestrado Integrado em Engenharia Aeronáutica** da **Faculdade de Engenharia** da **Universidade da Beira Interior**, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridade da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer que, em particular, atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, assumindo assim na íntegra as responsabilidades da autoria.

Por ser verdade o que se antecede, abaixo apresento a minha assinatura.

Universidade da Beira Interior, Covilhã, 3 de dezembro de 2025

Assinado por: **João Pedro Monteiro Coelho**
Num. de Identificação: 30083875
Data: 2025.12.03 12:49:55+00'00'

CHAVE MÓVEL

(assinatura conforme Cartão de Cidadão, ou preferencialmente assinatura digital no documento original em formato eletrónico)

Dedicated to my family and friends, who have supported me through this journey.

Acknowledgements

With the end of my academic journey, I want to express my gratitude to those who helped me through these years.

First and foremost, I would like to give my deepest thanks to my supervisor, Dr. Kouamana Bousson, for the patience, guidance, encouragement and expertise in a matter fundamentally different from the basis that is Aeronautical Engineering such as Machine Learning. For the countless meetings, explanations and recommendations, his strong background in Machine Learning, Guidance, Navigation and Control clearly motivated me, allowing me to produce a document that I am truly proud of.

Next, I would like to thank my friends for each and every single day I have spent in Covilhã, in both good and bad moments. From those fast-paced weeks preparing for the exams, to some slow and delicious moments during class breaks, you have really stood out in my consideration from the very first year until now. I hope that this friendship lasts for a long time.

Finally, I would like to thank my family for all the support they gave me from the beginning. To my uncles and aunts, for the insight and spark they have always ignited in my curiosity. To my cousins, for their help and friendship. To my mother, for always making me believe that everything is possible, to my father, for always pushing me to do the best I could, to my brother, who always sparked joy in my life and, mostly, to my grandmothers and grandfather, my biggest fans in every moment.

Resumo Alargado

Dada a grande evolução da economia espacial e o seu impacto em setores estratégicos como a logística, navegação e defesa, o congestionamento de diversas órbitas começa a ser motivo de preocupação. Este fenómeno, aliado aos esforços de diversas potências internacionais para a crescente militarização do espaço, aumenta o risco crítico de quedas não controladas. Tentando completar pesquisas realizadas anteriormente, mais focadas a altitudes superiores a 80 kilómetros, esta dissertação pretende apurar um método de previsão para as secções mais inferiores da atmosfera terrestre, abordando também casos de órbita terrestre baixa (LEO). Utilizando conceitos como o modelo de turbulência de *Dryden*, bem como diversos modelos da atmosfera padrão, pretende-se aumentar a robustez e exatidão na previsão da posição do veículo no instante seguinte com base em três instantes anteriores. Utilizou-se uma rede neuronal *Long-Short Term Memory* (LSTM), estando uma parte dedicada à conceção e análise do modelo. Esta dissertação possui também uma componente teórica dedicada à apresentação de aspetos fundamentais de probabilidades, estatística e estimação - a base primária da inteligência artificial. Posteriormente, apresentam-se conceitos relacionados com a otimização dos parâmetros do modelo, desde a regularização à otimização, contendo medidas amplamente utilizadas como o *early stopping* e o aumento dos dados para teste, incluíndo também vários métodos comummente utilizados como *ADAM*, *NADAM*, *ADAMw* e *RMSProp*, cuja eficácia é colocada em estudo através de um método de otimização dos hiperparâmetros conhecido como o Estimador *Tree-Structured Parzen*. Este método foi também aplicado para analisar outros fatores como o ritmo de aprendizagem, o número de camadas escondidas, bem como a sua complexidade, assim como o tamanho do *batch* e as funções utilizadas para a métrica da perda, tendo por objetivo a melhor divergência de *Kullback-Leibler (Jensen-Shannon)* entre os dados de treino e teste. Em seguida, apresentam-se definições de *Gate Recurrent Units* bem como do próprio LSTM, que foi adotado para manter a evolução temporal em conta. Devido à inexistência de uma base de dados suficientemente grande para treinar este modelo, foi também criado um capítulo dedicado aos procedimentos, descrevendo o algoritmo utilizado para o mesmo. Para além das equações já explicadas anteriormente, recorreu-se a um modelo geocêntrico (com coordenadas esféricas) de três graus de liberdade. Utilizaram-se várias posições iniciais dos veículos aeroespaciais para gerar dados ao variar a anomalia real (θ), mantendo as outras coordenadas de Kepler constantes. Escolheram-se os satélites *UARS* e *KOSMOS 482* para testar este modelo, perfazendo uma média de cerca de 1500000 vetores de estado para treinar e testar. No que toca à arquitetura do mesmo, optou-se pela divisão em três modelos de entrada múltipla e saída simples devido à maior eficácia aquando do processo de otimização das posições x, y e z seguintes de forma individual. O passo de tempo adotado foi de $0,1$ segundos e o treino foi realizado em duas fases. Primeiramente, dividiu-se o total de dados disponíveis em 20% para teste, 64% para treino e 16% para validação. Após um máximo de 2000

épocas, os dados foram guardados e os dados de treino e validação foram juntos num só. Na segunda fase, escolheram-se os melhores valores do *Kullback-Leibler* entre duas hipóteses. A primeira envolvia treinar este novo conjunto de dados de raíz até atingir o número de épocas primeiramente registado. Já o segundo envolvia treinar por cima do modelo já guardado até atingir uma métrica de perda semelhante à obtida pelos dados de validação na primeira fase. De entre destas duas hipóteses, a que teve melhor desempenho foi sempre a segunda. Para testar a robustez do modelo, realizou-se um teste de *Kullback-Leibler* a cada um dos três componentes do modelo. Os resultados obtiveram uma média geométrica de 6,07 e 0,492 para os respetivos casos do *UARS* e do *KOSMOS 482*.

Palavras-chave

Reentrada Atmosférica, *Machine Learning*, Previsão de Trajetórias, *Long-Short Term Memory*, Validação Estatística

Abstract

Due to the rapid development of the machine learning field, its applications in trajectory prediction are now a matter of focus in academia and industry. In order to further broaden previous research, focusing more on a lower-atmosphere approach, this thesis aims to develop a model more focused on the latter steps of the reentry phase. The main objective is to create a deep learning model to predict the next position of the reentry aerospace vehicle based on a set of three previous positions. In order to test the robustness of this model, a Kullback-Leibler (Jensen-Shannon) divergence test was applied to each of the prediction models. The results were 6.07 and 0.492 for the *UARS* and *KOSMOS 482* satellites, respectively. The results show a good degree of correspondence between the training and test sets, implying that the learning process was well done. However, should a similar study be conducted, efforts should be taken to obtain more realistic data, whether through more complex simulation, or using real sensor input. This thesis made use of the Dryden turbulence model, as well as many atmospheric models such as the *International Civil Aviation Organization* and the *United States of America 1976*. Long-Short Term Memory was the chosen recurrent neural network to have the evolution through time into account, having a dedicated section to its creation and analysis. The optimization and regularization processes are also presented, making use of the Tree-Structured Parzen Estimation to tune hyperparameters present in the model such as loss functions and ratio, number of hidden layers and their inputs, batch size and optimization models. Due to the absence of a sufficiently broad data pool to train this model, a chapter dedicated to the simulation procedures was also written, describing the algorithm used for it. The satellites chosen to test this model were *UARS* and *KOSMOS 482*, choosing initial conditions based on their latest recorded orbit epochs, varying the true anomaly and therefore obtaining about 1500000 inputs divided between training and testing the model. For the LSTM architecture, a MISO approach was taken due to dimensionality, dividing the trained model into three separate position predictions (x, y and z) in a time step of 0.1 seconds.

Keywords

Atmosphere Reentry, Machine Learning, Trajectory Prediction, Long-Short Term Memory, Statistical Validation

Contents

List of Figures

List of Tables

List of Acronyms

AdaGrad	Adaptative Gradient
AdaM	Adaptative Momentum
AdaMw	Adaptative Momentum with decoupled weight decay
AI	Artificial Intelligence
ASAT	Anti-Satellite
BFGS	Broyden-Fletcher-Goldfarb-Shanno
CUDA	Compute Unified Device Architecture
CuDNNLSTM	CUDA Deep Neural Network LSTM
DOF	Degree of Freedom
e.g.	*exempli gratia*, or 'for example, '
ESA	European Space Agency
GEO	Geostationary Orbit
GRU	Gate Recurrent Units
GPU	Graphics Processing Unit
i.e.	*id est*, or 'that is,'
IADC	Inter-Agency Space Debris Coordination Committee
ICAO	International Civil Aviation Organization
ISA	International Standard Atmosphere
KL	Kullback-Leibler
L-BFGS	Limited Memory Broyden-Fletcher-Goldfarb-Shanno Algorithm
LEO	Low Earth Orbit
LS	Least Squares
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MIMO	Multi Input, Multi Output
MISO	Multi Input, Single Output
ML	Machine-Learning
MSE	Mean Squared Error
NADAM	Adaptative Momentum Algorithm with Nesterov Momentum
NASA	National Aeronautics and Space Administration
OST	Outer-Space Treaty
p.d.f	Probability-Density Function
PD	Payload Debris
PF	Payload Fragmentation Debris

PL	Payload
PM	Payload Mission Related Object
RB	Rocket Body
RD	Rocket Debris
ReLU	Rectified Linear Unit
RF	Rocket Fragmentation Debris
RM	Rocket Mission Related Object
RMS	Root Mean Square
RMSProp	Root-Mean Squared Propagation
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
R^2	Coefficient of Determination
SGD	Stochastic Gradient Descent
SSPP	Spoof Surface Plasmon Polariton
TAEM	Target Area Energy Management
TPE	Tree-Structured Parzen Estimator
UARS	Upper Atmosphere Research Satellite
UBI	Universidade da Beira Interior
UI	Unidentified
USA, US	United States of America
USSR	Union of Soviet Socialist Republics
WEF	World Economic Forum
w.r.t	With Reference To
ZOH	Zero-Order Hold Representation of the Inverse-Z Transform

List of Nomenclatures

Symbol	Description	Units
α	Angle of attack	[rad]
α_s	Sensitivity parameter	[–]
α_T	Exponential parameter to compute pressure above 86 kilometers	[K/m]
a_{sound}	Speed of sound	[m s^{-1}]
a	Semi-major axis	[m]
β	Conjugate-gradients parameter	[–]
$\mathbf{b}$	Bias parameter vector	[–]
$B_{TPE}(\cdot)$	Function to compute the bandwidth for TPE	[–]
C_b	Ballistic coefficient	[–]
C_D	Drag coefficient	[–]
C_L	Lift coefficient	[–]
$\text{Cov}(\cdot)$	Covariance	[–]
$\text{Cor}(\cdot)$	Linear correlation	[–]
$\mathbf{C}_t$	Memory cell matrix	[–]
$\widetilde{\mathbf{C}}_t$	Candidate memory cell matrix	[–]
d	Dimension	[–]
D	Drag	[N]
D_{KL}	Kullback–Leibler divergence	[–]
D_n	Kolmogorov–Smirnov test statistic	[–]
δ	Parameter to avoid division by zero	[–]
Δt	Time step	[s]
Δt_{csv}	Time step recorded in the *.csv* file	[s]
$\dot{Q}$	Heat-transfer rate	[W]
E	Lift-to-drag ratio	[–]
$E(\cdot)$	Expected value	[–]
$\mathbb{E}_{(\mathbf{x},\boldsymbol{y})\sim\hat{\boldsymbol{p}}_{\text{Data}}}$	Expectation w.r.t empirical distribution	[–]
e	Eccentricity	[–]
$\mathbf{e}_{\text{Set}}$	Error vector (*e.g.*, position error)	[m]
ε	Learning rate	[–]
η	Schedule multiplier	[–]
F_n	Empirical distribution function	[–]
$\mathbf{F}_t$	Forget gate matrix	[–]
$f(\cdot)$	Probability density function	[–]
$\hat{f}(\cdot)$	Estimate of a p.d.f.	[–]

Symbol	Description	Units
$\tilde{f}(\cdot)$	Pilot estimate of a p.d.f.	[–]
g	Gravitational acceleration	[m s^{-2}]
$g(\cdot)$	Geometric mean function	[–]
$\mathbf{g}$	Gradient vector	[–]
$\hat{\mathbf{g}}$	Gradient estimation vector	[–]
$\Gamma(\cdot)$	Function to compute the top quantile for TPE	[–]
h	Window width	[–]
h_{opt}	Optimal window width	[–]
h_{alt}	Altitude	[m]
$\mathbf{H}$	Hessian matrix	[–]
$\mathbf{H}_t$	Hidden state matrix	[–]
$\widetilde{\mathbf{H}}_t$	Candidate hidden state matrix	[–]
$\mathbf{h}^{(i)}$	Output vector of hidden layer i	[–]
i	Inclination	[rad]
$\mathbf{I}$	Identity matrix	[–]
$\mathbf{I}_t$	Input gate matrix	[–]
I_n	Indicator function	[–]
$J(\cdot)$	Objective function	[–]
K	Convective-heating correlation constant	[Wm^{-2}]
$K(\cdot)$	Kernel function	[–]
$K_e(\cdot)$	Epanechnikov kernel function	[–]
λ_i	Bandwidth factor	[–]
λ	Troposphere temperature variation rate	[K m^{-1}]
λ_{wd}	Weight decay parameter	[–]
λ_{20k}	Stratosphere temperature variation rate	[K m^{-1}]
L	Lift	[N]
L_{turb}	Turbulence length scale	[m]
$L(\cdot)$	Loss function	[–]
m	Mass	[kg]
m_{ML}	First momentum estimation parameter	[–]
$\mathbf{m}$	First moment vector	[–]
m_Q	Heat-flux density exponent	[–]
$M_1(\cdot)$	LS cross-validation score	[–]
$\mathbf{M_0}$	Approximate inverse of Hessian	[–]
μ	Mean	[–]
n_{load}	Load factor	[–]
n	Second momentum estimation parameter	[–]
$\mathbf{n}$	Second moment vector	[–]
N_{trials}	Number of trials used in TPE	[–]
n_Q	Heat-flux velocity exponent	[–]
$\mathcal{N}$	Gaussian distribution	[–]

Symbol	Description	Units
$\mathbf{O}_t$	Output gate matrix	[–]
$\mathcal{O}$	Set of hyperparameter combinations for TPE	[–]
p	Static pressure	[Pa]
p_0	Standard sea-level pressure	[Pa]
$p_{i\ k}$	Pressure at i kilometers	[Pa]
q	Dynamic pressure	[Pa]
R	Specific gas constant (air)	[J kg^{-1} K^{-1}]
$\mathbf{r}$	Gradient accumulation vector	[–]
r	Geocentric distance	[m]
r_0	Mean Earth radius	[m]
ρ	Mass density	[kg m^{-3}]
ρ_0	Standard sea-level density	[kg m^{-3}]
ρ_{ML}	Decay rate	[–]
$\boldsymbol{\rho}$	Decay rate vector	[–]
ρ_1, ρ_2	Exponential decay rates	[–]
$\mathbf{R}_t$	Reset gate matrix	[–]
σ_{sd}	Standard deviation	[–]
σ	RMS turbulence speed	[m s^{-1}]
σ_b	Bank angle	[rad]
$\boldsymbol{\sigma}^{(i)}$	Activation output vector i	[–]
$\boldsymbol{\Sigma}$	Co-variance matrix	[–]
S_{wet}	Wetted surface area	[m^2]
sup	Supremum	[–]
t	Time-step	[–]
T	Static temperature	[K]
T_0	Sea-level temperature	[K]
$T_{i\ k}$	Temperature at i kilometers	[K]
θ	Longitude (Chapter 3), True anomaly (Chapter 4)	[rad]
$\boldsymbol{\theta}$	Parameter (ML)	[–]
$\mathbf{v}$	Velocity vector	[–]
u, v, w	Velocity components	[m s^{-1}]
V	Velocity magnitude	[m s^{-1}]
V_c	Critical velocity	[m s^{-1}]
φ	Latitude	[rad]
ψ	Heading/course angle	[rad]
W_{20}	Turbulence severity parameter	[–]
$\mathbf{W}$	Weight matrix	[–]
$W_{TPE}(\cdot)$	Function to compute weights for TPE	[–]
$\mathbf{X}_{\text{clean}}$	Clean state vector	[m]

Symbol	Description	Units
$\mathbf{X}_{\text{state}}$	State vector	[m]
$\boldsymbol{X}$	Sample of random variables	[–]
X	Continuous/discrete variable	[–]
$\mathbf{x}$	Input vector	[–]
x, y, z	Cartesian coordinates	[m]
$\mathbf{y}$	Output vector	[–]
$\mathbf{Y}$	Measured state vector	[m]
$\mathbf{Z}_t$	Update gate matrix	[–]
$\mathcal{D}(\cdot)$	Dryden gust component	[m s^{-1}]
Φ	Dryden power spectral density	[m^3 s^{-2}]
γ	Flight-path angle	[rad]
γ_0	Ratio of specific heats	[–]
Ω_f	Spatial frequency	[rad m^{-1}]
Ω_{RAAN}	Right ascension of ascending node (RAAN)	[rad]
$\omega(\cdot)$	White noise output	[–]
ω	Argument of pericenter	[rad]

Chapter 1

Introduction

1.1 Contextualization

As cited by the *World Economic Forum* [43], space is expected to be worth $1.8 trillion by 2035, growing at an average of 9% *per annum*, playing a vital role in navigation, defense and connectivity. Without functional satellites, industries like logistics, digital communications or even defense would have a dramatic reduction in their efficiency. It can be seen, by observing Figure 1.1, that the impact of the space sector will spread to many other economies, making it a crucial asset in the near future.

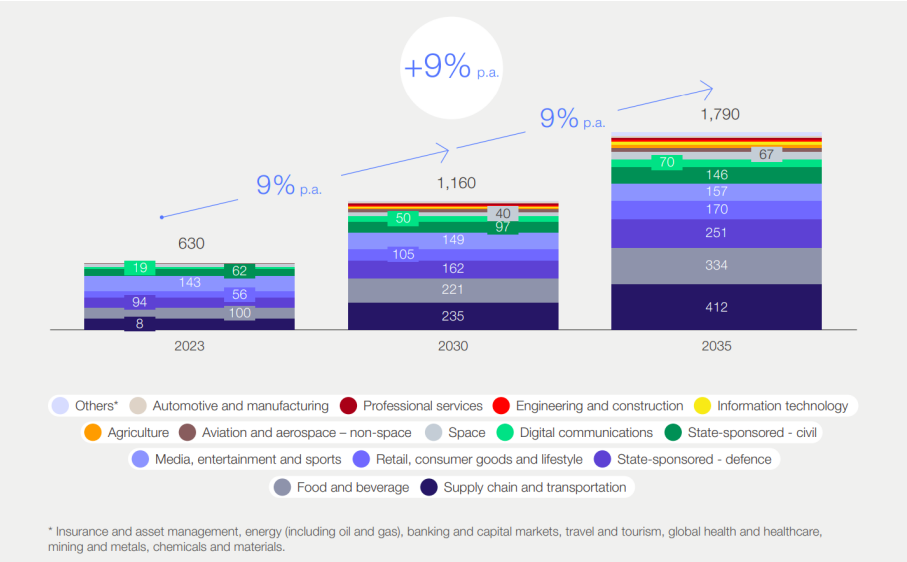


Figure 1.1: Expected growth in space economy (Credits: WEF[43])

This current trend is showing an enormous return on investment, attracting many new companies outside of government interest, and contributing to a clear democratization of this cluster.

1.2 Problem Statement

Recently, the evolution in space applications and the private sector investment in this area has led to a massive increase in satellite launches, as stated in Figure 1.2.

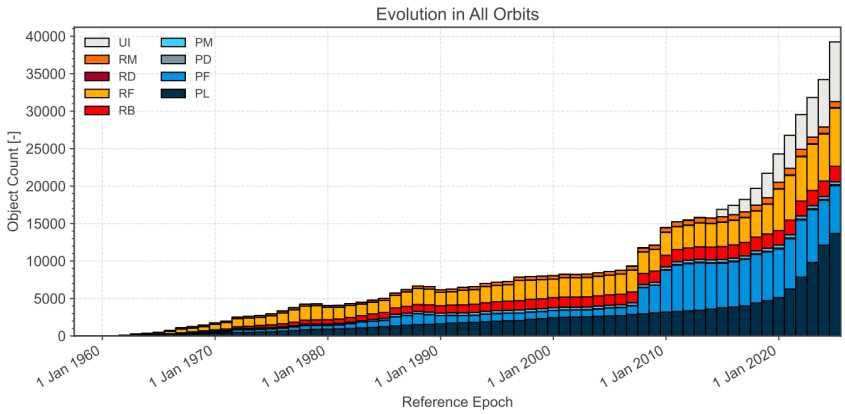


Figure 1.2: Evolution in orbital launches (Credits: ESA[32])

With this evolution, the overload in main orbiting altitudes has also become a motive of concern, as the eventual complexity in future space operations will undoubtedly increase the number of possible collisions, as estimated by IADC in Figure 1.3, generating a considerable amount of space debris. Even now, should future launches be canceled, this collision trend would not decay.

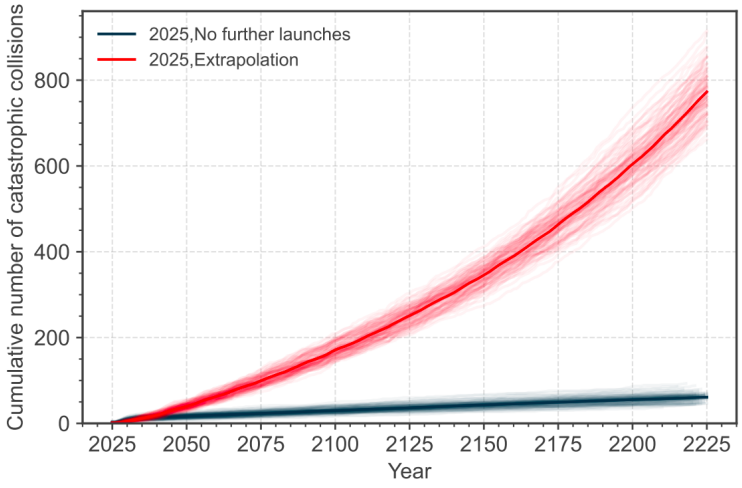


Figure 1.3: Number of cumulative collisions in LEO orbit (Credits: ESA, IADC [32])

The amount of reentries ocurred in the last thirty five years can be consulted in Figure 1.4. Even though the amount of controlled reentries is superior to the uncontrolled ones for the first time since last year, meaning that measures are being taken to prevent such scenarios, some unexpected events may lead to failures in controlling those vehicles.

Furthermore, the increasing trend in the militarization of the space sector should also be considered. Since the Cold War, the USA and USSR have been conducting anti-satellite weapon test activities, knowledge that has been kept and could be deployed again due to the current political world bi-polarization. In fact, China has also deployed some tests in this area since 2007, as well as India in this decade. According with the Center for Strate-

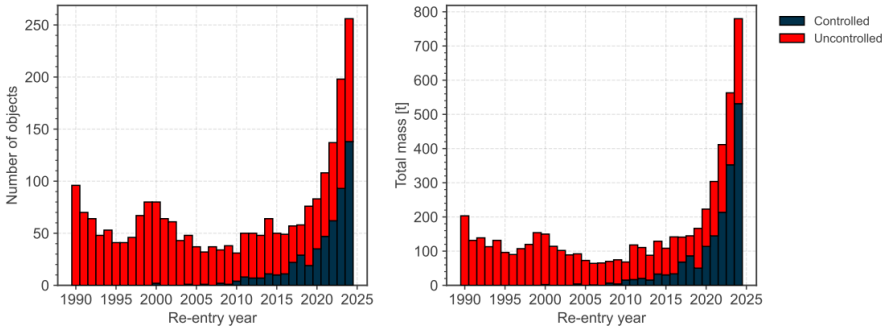


Figure 1.4: Evolution of controlled and uncontrolled reentries (Credits: ESA [32])

gic and International Studies [23], Chinese and Russian satellites in LEO keep showing concerning advanced maneuvering capabilities, increasing the ability to conduct successful rendezvous, proximity and docking operations, techniques that could be used by ASAT weapons. Moreover, the Russian government has increasingly stated that commercial assets with any kind of connection to the US military, like *Starlink*, should be considered as a legitimate threat. Another dubious attitude from this nation can be observed from their veto cast on the United States and Japan draft resolution to reaffirm the obligations of sovereign states under Article IV of the OST, calling on nations not to develop nuclear weapons specifically designed to be placed in orbit around the Earth. China has adopted a more discrete approach by abstaining from the previously mentioned voting process, but re-organizing their armed forces to include a space operations division, being recently observed practicing ”dogfighting” maneuvers with some of their satellites.

1.3 Current Approaches and Their Limitations

In terms of deep learning, many articles can be found, mainly directed to satellite in-orbit prediction, as an attempt reduce the degree of complexity and non-linearity brought by mathematical models. An example of this approach can be seen by reading the article produced by Senra [20], which focused on creating a robust model to predict the next state vector of a satellite based on fifty older states, and comparing its efficiency with an extended Kalman filter.

One of the first researches to directly approach the usage of an LSTM to predict the reentry of an uncontrolled object was made by Jung [11]. One of the main discoveries was the relation between the balancing of the ballistic coefficient distribution and the accuracy of the prediction.

Following their results, Salmaso [19] managed to broaden the model to include the ballistic coefficient as a legitimate input, alongside with the average solar index and the area-to-mass ratio of the vehicle. Their best-performing testing sets were the ones which had similar drag coefficient and orbit eccentricity.

These articles focus on using a substantial amount of previous data to predict the reentry position at the altitude of 80 kilometers. Usually, the main concern under this altitude is related with the increase in density, where the atmospheric phenomena becomes the dominant force for this bodies, thus enhancing the effects of turbulence on the vehicles and generating more uncertainty in solving this problem. Currently, the main research focus is equal or above this altitude, ignoring the remaining descent until touchdown.

1.4 Objectives

The main objective of this research is to develop a robust modeling framework to predict the reentry behavior of an uncontrolled vehicle (e.g. a satellite, comet, debris) in free flight. It will be more tuned into the noise parameters caused by turbulence, providing more accuracy in the results predicted. It will be more focused in the lower parts of the atmosphere, specifically below 86 kilometers, and will also make use of the novel applications provided by machine learning, being offered as a new alternative to mathematical models.

The prediction model will make use of a reduced number of previous states, while maintaining their current trend information. In order to train it, real data should be obtained and, if not possible, many previous simulations should be made using current mathematical models to better approximate to real-life case. The metric used to test the accuracy of the trained model will be more focused on comparing the error probability-density functions from both testing and training pools, in order to understand its behavior when submitted to novel cases.

1.5 Thesis Plan

This thesis will be divided in the following chapters:

- Theoretical Background, where the main concepts of reentry will be presented, as well as introducing the reader to some probabilities, statistics and estimation definitions. These concepts will be used to further introduce the reader to one of the main concepts - machine learning. Here, a broad introduction will also be made, with basic definitions for a simple regression model, going later to regularization topics and optimization. After that, more information will also be presented about hyperparametrization tuning, finishing with an approach to some recurrent neural networks.
- Models used in Simulation, should real data not be available, where the main models used for reentry are presented to the reader. These include the base three degree of

freedom model, as well as a constraint modeling. This document should also focus on presenting the an atmospheric model for a more precise simulation (,and other mainly used models should the initial conditions be in the upper atmosphere), as well as obtaining the Dryden turbulence model for noise robustness. Finally, the reader will be introduced to the main architecture of the RNN model used, as well as the process for hyperparameter tuning.

- Model Creation, explaining how the RNN model is created, its inputs, outputs and behavior. It will contain the procedures taken to train it, as well as the hyperparameters used for it. Some graphics related with training will also be presented, in order for the reader to understand the process. The final results for the error p.d.f. divergence will appear as well. In the case of real data not being available, it should also be explained primarly how the data used for training is generated - from the initial conditions of the studied cases, to the methods used to discretize the model, while also presenting a flowchart of the whole process.
- Conclusion and Further Research, where main conclusions, concerns and future study suggestions will be presented.

Chapter 2

Theoretical Background

2.1 Types of Reentry

There are two types of reentry: ballistic and lifting. The first case happens when the vehicle is only subjected to the air resistance, D. Here, the most important factor is the ballistic coefficient, C_b, as shown in equation **(2.1)**, where C_D is the drag coefficient, S_{wet} is the wet surface area and m is the mass. The lower C_b is, the lower is the heat transference.

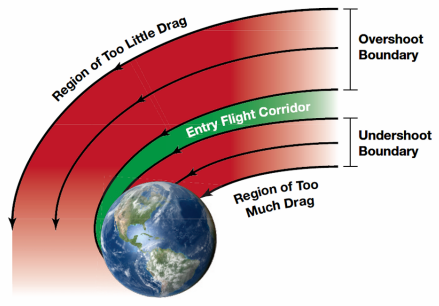
$$C_b = \frac{m}{C_D S_{wet}} \quad (2.1)$$

We can also obtain another form of the ballistic coefficient, B^*, in equation **(2.2)**.

$$B^* = \frac{\rho}{2C_b} = \frac{\rho C_D S_{wet}}{2m} \quad (2.2)$$

The second type of reentry should be treated more carefully, i.e. the vehicle must position itself in a way that the drag allows it to land, but not suffer major consequences, as stated in Figure 2.1b. The body should position itself in the entry flight corridor, avoiding unpredictable damage.

(a) Ballistic reentry (Credits: ESA [33])



(b) Lifting reentry (Credits: Hilton [25])

Figure 2.1: Reentry examples

Later on, the vehicle should orient itself in a certain angle of attack, which generally is $\alpha \in [-3; 3]^o$, in order to direct most of the heating to the appropriate shield. Lastly, we use

the bank angle, σ_b, to control the deceleration needed to prepare the vehicle for landing. In the case of having a crew aboard, the vehicle should also enter in the Target Area Energy Management (TAEM) phase.

2.2 Probabilities and Statistics Definitions

2.2.1 Main Concepts

Consider a probability density function, f, defined according with equation **(2.3)**, where P is a probability.

$$P(a < X < b) = \int_a^b f(x)dx \quad \forall a < b \quad (2.3)$$

The density estimation, as stated in Silverman[21], is the construction of an estimate of the density function from the observed data.

2.2.1.1 Expected Value

Let X be either a bounded continuous or a discrete random variable, whose probability-density function is f. The mean, or expected value, is defined according with equation **(2.4)**, in the case of a continuous variable, or **(2.5)**, if the variable is discrete.

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx \quad (2.4)$$

$$E(X) = \sum_{k=1}^{n} xf(x) \quad (2.5)$$

This definition works well with one variable functions but, since our case is far different, we can also present the multi-dimensional case, according with the Law of the Unconscious Statician. Let $\mathbf{X} \in \mathbb{R}^n$ be a sample of random variables according with equation **(2.6)** with the joint p.d.f $f(\mathbf{X})$, r be a real-valued function of n real variables and $Y = r(\mathbf{X})$.

$$\mathbf{X} = \begin{bmatrix} X_1 & \dots & X_n \end{bmatrix}^T \quad (2.6)$$

$E(Y)$ can be determined both for continuous **(2.7)** and discrete **(2.8)** cases.

$$E(Y) = \int_{\cdot} ..\mathbb{R}_n ... \int r(x_1, ...x_n) f(x_1, ..., x_n) dx_1, ..., dx_n \quad (2.7)$$

$$E(Y) = \sum_{\mathbf{X}} r(x_1, ...x_n) f(x_1, ..., x_n) dx_1, ..., dx_n \quad (2.8)$$

2.2.1.2 Variance

Let X be a random variable with finite mean $\mu = E(X)$. The variance of X can be defined according with equations**(2.9)** or **(2.10)**.

$$\text{Var}\,(X) = E\left[(X - \mu)^2\right] \quad (2.9)$$

$$\text{Var}\,(X) = E\left(X^2\right) - \mu^2 \quad (2.10)$$

2.2.1.3 Covariance

Let X and Y be random variables with finite means. Let $E(X) = \mu_X$ and $E(Y) = \mu_Y$. The covariance of X and Y is defined according with equation **(2.11)**.

$$\text{Cov}\,(X, Y) = E\left[(X - \mu_X)(Y - \mu_Y)\right] \quad (2.11)$$

2.2.1.4 Linear Correlation

Let X and Y be random variables with finite variances $\sigma^2_{sd_X}$ and $\sigma^2_{sd_Y}$, respectively. The correlation of X and Y is defined according with equation **(2.12)**.

$$\text{Cor}\,(X, Y) = \frac{\text{Cov}\,(X, Y)}{\sigma_{sd_X}\sigma_{sd_Y}} \quad (2.12)$$

2.2.1.5 Mean Vector and Covariance Matrix

For multivariate random variables, other topics apply. Let $\mathbf{X}^T = [X_1, ..., X_d] \in \mathbb{R}^d$ be a d-dimensional random vector. Let $\boldsymbol{\mu}$ denote the mean vector and $\boldsymbol{\Sigma}$ the covariance matrix. For $i, j \in \{1, ..., d\}$, the formula for an entry i in the mean vector is shown in equation **(2.13)**. Similarly, the entry in line i, column j of the covariance matrix is defined in equation **(2.14)**. In this last equation, the use of $(n-1)$ in the denominator is to get an unbiased estimate of the covariance matrix.

$$\boldsymbol{\mu}_i = \frac{\sum_{k=1}^{n} x_{k,i}}{n} \quad (2.13)$$

$$\boldsymbol{\Sigma}_{ij} = \frac{\sum_{k=1}^{n} \left(x_{k,i} - \overline{x_i}\right)\left(x_{k,j} - \overline{x_j}\right)}{n-1} \quad (2.14)$$

2.2.1.6 Univariate Normal Distribution

A random variable X has normal distribution with (finite) mean μ and (positive) variance σ_{sd}^2 if f has a continuous distribution with the p.d.f as described in equation **(2.15)**, for $x \in]-\infty; \infty[$.

$$f\left(x|\mu, \sigma_{sd}^2\right) = \frac{1}{\sigma_{sd}\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma_{sd}^2}} \quad (2.15)$$

2.2.1.7 Multivariate Normal Distribution

Consider the same random variable $\mathbf{X} \sim \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. If $\boldsymbol{\Sigma}$ is positive definite, then a definition of multivariate Gaussian distribution appears in equation **(2.16)**.

$$f\left(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{(2\pi)^{n/2}\left|\boldsymbol{\Sigma}\right|^{1/2}} e^{-0.5(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \quad (2.16)$$

2.2.2 Kernel Method

Assume a d-dimensional space, where $X_k \in \mathbb{R}^{d\times 1}$, for $k \in \{1, ..., n\}$, where n is the total number of samples. The multivariate Kernel density estimator, $\hat{f}(\mathbf{x})$, is defined in equation **(2.17)**, where h is the window width, and $K(\mathbf{x})$ is the Kernel function, that satisfies equation **(2.18)**.

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left[\frac{1}{h}\left(\mathbf{x} - \mathbf{X_i}\right)\right] \quad (2.17)$$

$$\int_{\mathbb{R}^d} K(\mathbf{x})\, d\mathbf{x} = 1 \quad (2.18)$$

For the scope of this document, where our elements may have up to three dimensions, we can consider two Kernel functions - the standard multivariate normal density function **(2.19)** and the Epanechnikov Kernel **(2.20)**, where c_d assumes the values presented in Table 2.1.

$$K\left(\mathbf{x}\right) = \left(2\pi\right)^{\left(-d/2\right)} \exp\left(-\frac{1}{2}\mathbf{x}^T\mathbf{x}\right) \quad (2.19)$$

$$K_e\left(\mathbf{x}\right) = \begin{cases} \frac{1}{2}c_d\left(d+2\right)\left(1-\mathbf{x}^T\mathbf{x}\right) & \textit{if } \mathbf{x}^T\mathbf{x} < 1 \\ \\ 0 & \text{otherwise} \end{cases} \quad (2.20)$$

d	$\mathbf{c_d}$
1	2
2	π
3	$4\pi/3$

Table 2.1: c_d Values according to the dimension

In order to obtain the optimal window width, h_{opt}, one needs to solve equation **(2.21)**, where $A(K)$ is depicted in equation **(2.22)**.

$$h_{opt} = A(K)n^{-\frac{1}{d+4}} \quad (2.21)$$

$$A(K) = \begin{cases} \left(\frac{4}{d+2}\right)^{\frac{1}{d+4}} & \text{Normal Kernel} \\ \\ \left(\frac{8}{c_d}\left(d+4\right)\left(2\sqrt{\pi}\right)^d\right)^{\frac{1}{d+4}} & \text{Epanechnikov Kernel} \end{cases} \quad (2.22)$$

Another method to obtain h_{opt} can be obtained by solving the least squares cross-validation equation **(2.23)**. However, it is important to keep in mind that this equation is usually computationally heavy for dimensions higher than three.

$$M_1(h) = n^{-2}h^{-d}\sum_i\sum_j K\left[h^{-1}\left(\mathbf{X_i} - \mathbf{X_j}\right)\right] + 2n^{-1}h^{-d}K(0) \quad (2.23)$$

2.2.3 Adaptative Kernel Estimate

In some cases, a strict window width is not enough to obtain the accuracy needed. It may be necessary to use a broader kernel in regions of lower density, as well as diminishing the window width in higher density ones.

Usually, it is required to define a pilot estimate, $\tilde{f}(\mathbf{t})$ that satisfies $\tilde{f}(\mathbf{X}_i) > 0$ for all i, as well as the local bandwidth factors, λ_i, according to equations **(2.24)** and **(2.25)**, where g is the geometric mean and $0 \leq \alpha_s \leq 1$ is the sensitivity parameter.

$$\lambda_i = \left[\frac{\tilde{f}\left(\mathbf{X_i}\right)}{g}\right]^{-\alpha_s} \quad (2.24)$$

$$\ln g = n^{-1}\sum \ln \tilde{f}\left(\mathbf{X_i}\right) \quad (2.25)$$

Next, one needs to define the adaptative, kernel estimate, $\hat{f}(\mathbf{t})$, according to equation **(2.26)**, where h is the static window width, as defined in equation **(2.23)**.

$$\hat{f}\left(\mathbf{t}\right) = n^{-1}\sum_{i=1}^{n} h^{-d}\lambda_i^{-d} K\left[\frac{\mathbf{t} - \mathbf{X_i}}{h\lambda_i}\right] \quad (2.26)$$

To finalize, it is important explaining which value of α should be selected. Usually, it is recommended to pick Breiman's choice, $\alpha = 1/d$. However, we can also pick Abraham-son's choice of $\alpha = 1/2$, given that the adaptative kernel bias is identically zero in both univariate and multivariate cases.

2.2.4 Nadaraya–Watson Estimator

For multivariate kernel density estimation, we can also use the Nadaraya–Watson estimator, defined according with equation **(2.27)**, noting that $K(\cdot)$ is at least of first order.

$$\hat{f}\left(\mathbf{x}, h\right) = \frac{\sum_{i=1}^{n}\frac{1}{h}K\left(\mathbf{x} - \mathbf{X_i}\right)y_i}{\sum_{i=1}^{n}\frac{1}{h}K\left(\mathbf{x} - \mathbf{X_i}\right)} \quad (2.27)$$

2.2.5 Kolmogorov-Smirnov Test

In this section, due to its complexity, both univariate and multivariate cases of the Kolmogorov-Smirnov test will be presented.

2.2.5.1 Univariate Case

For a cumulative distribution function, the definition is shown in equation **(2.28)**, with F_n defined in equation **(2.29)**, where $I_{[-\infty,x]}(X_i)$ is also defined in equation **(2.30)**.

$$D_n = \sup_x |F(x) - F_n(x)| \tag{2.28}$$

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{[-\infty,x]}(X_i) \tag{2.29}$$

$$I_{[-\infty,x]}(X_i) = \begin{cases} 1 & \text{if } X_i \leq x \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

Let S be subset of P. It is noted that an element $s \in P$ is the supreme of S if the condition **(2.31)** is fulfilled.

$$\begin{cases} x \leq s, & \forall x \in S \\ x \leq s', & \forall x \in S \rightarrow s \leq s' \end{cases} \tag{2.31}$$

2.2.5.2 Multivariate Case

For the multivariate case we can adapt the theory provided by Fasano-Franchesini [9]. It is firstly presented the two-dimensional case, which is later converted to the third. Here, for N_1 and N_2 distributions, we do the following process:

- For each point $i \in n$, divide the plane into quadrants;
- Sum the number of all the points from each distribution in each quadrant, getting $N_{i_{1,quadrant}}$ and $N_{i_{1,quadrant}}$;
- Obtain the normalized distance, given in equation **(2.32)**;

$$D_{i_{quadrant}} = \| \frac{N_{i_{1,quadrant}}}{n_1} - \frac{N_{i_{2,quadrant}}}{n_2} \| \tag{2.32}$$

- Obtain the maximum value for each point i, according to equation **(2.33)**;

$$D_i = max\left(D_{i_{quadrant}}\right) \quad (2.33)$$

- Obtain the Kolmogorov-Smirnov distance, which is the maximum of all D_i values.

For the three-dimensional case, the procedure is similar, but in this case we use the eigth octants present in the 3D space. The procedure repeats itself for the n-dimension, where the number of n-tants is given by equation **(2.34)**.

$$n_{n-tants} = 2^d \quad (2.34)$$

2.2.6 Kullback-Leibler

Due to the elevated complexity of the Kolmogorov-Smirnov, allied with the slow computational processing of itself, it was decided to adapt an alternative for verifying the robustness of the LSTM model. For that, we define the Kullback-Leibler[12] divergence as a measure of how much a model probability distribution Q is different from a true probability distribution P, as stated in equation **(2.35)**, where χ is the sample space, and x is a sample.

$$D_{KL}(P||Q) = \sum_{x\in\chi} P\left(x\right) \log\left[\frac{P\left(x\right)}{Q\left(x\right)}\right] \quad (2.35)$$

This divergence is always positive, as it can be physically interpreted as the expected excess surprisal, i.e. the quantification of the "surprise" result in a particular outcome. Unfortunately, this equation is not symmetric, *i.e.* $D_{KL}(P||Q) \neq D_{KL}(Q||P)$, requires a full continuity of the probability-density functions in order to be correctly applied, must always be positive, and does not provide a clear way of comparing two functions. In order to prevent some future implementation mistakes, the Jensen-Shannon variation of this divergence [13] is presented in equation **(2.36)** and should be the used formula each time the Kullback-Leibler divergence is referred in this document.

$$D_{KL}(P||Q) = \frac{1}{2}\sum_{x\in\chi}\left(P\left(x\right)\log\left[\frac{P\left(x\right)}{P\left(x\right)+Q\left(x\right)}\right] + Q\left(x\right)\log\left[\frac{Q\left(x\right)}{P\left(x\right)+Q\left(x\right)}\right]\right) \quad (2.36)$$

2.3 A Brief Introduction to Machine Learning

Due to the unpredictability of the reentry phase in current day systems, it is imperative that some machine learning elements are implemented to further comprehend the non-linear aspects of our atmosphere. The focus of this chapter is to give the reader a glimpse of the contents needed to critically analyze the creation and testing of the model.

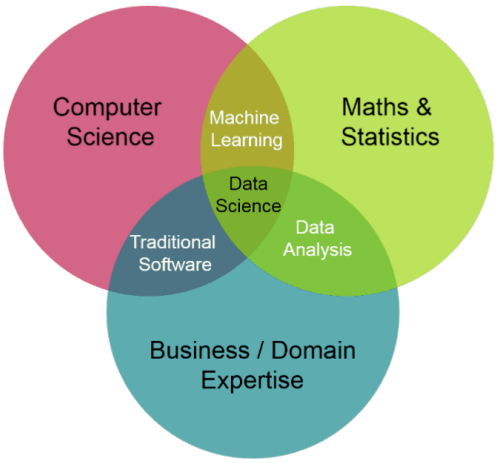


Figure 2.2: The representation of machine learning as a broad topic (Source: Zadeh [29])

In short, machine learning is a branch of computer science, that deals with co-relating aspects from various types of data, to further obtain a conclusion from it, with the help of statistics. As stated by Zhang[30], "Machine learning is the study of powerful techniques that can learn from experience", i.e. the ability for the computer to learn, and further predict a certain behavior based on previous observations.

The topic of machine learning can be divided in [37]:

- A Decision Process: The classification of certain patterns detected in the observed data;
- An Error Function: To address the accuracy of the novel model.
- A Model Optimization Process: The process of adapting those patterns in order to improve the accuracy.

Usually, a sample is required to train the model. The machine learning algorithms can be supervised or unsupervised, if there are samples used for testing or not. There is also the reinforcement category, which will not be covered since it is beyond the scope of this document.

There are various types of machine learning. The simplest form may be the linear regression. Consider an unknown function, defined in equation **(2.37)** where only the points (x_k, y_k) are known. Our objective would be to discover the m and b constants in such a way

that the error between the predicted function and the actual points would be the smallest possible.

$$\begin{cases} f : \mathbb{R} \to \mathbb{R} \\ f(x) = mx + b \end{cases} \tag{2.37}$$

To understand if our predictions are correct, we can use various parameters such as the mean absolute error **(2.38)**, the mean absolute percentage error **(2.39)**, the mean squared error **(2.40)** and its square root **(2.41)**, as well as the coefficient of determination **(2.42)**.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \|y_i - \hat{y}_i\| \tag{2.38}$$

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n} \|\frac{y_i - \hat{y}_i}{y_i}\| \tag{2.39}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.40}$$

$$\textit{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{2.41}$$

$$R^2 = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} \left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2} \tag{2.42}$$

This is the basis for a simple regression model. This concepts can later be expanded to multivariate datasets, or even to other function types. With this example, the definitions of over-fitting and under-fitting can be explained. Depending on the size of the testing sample (compared with the training sample), our data can become over-fitted, *e.g.* the error observed in the training set is low, but further testing shows a substantially larger one, or under-fitted, *e.g.* the error is exaggerated in both testing and training samples. A more visual example can be seen in Figure 2.3.

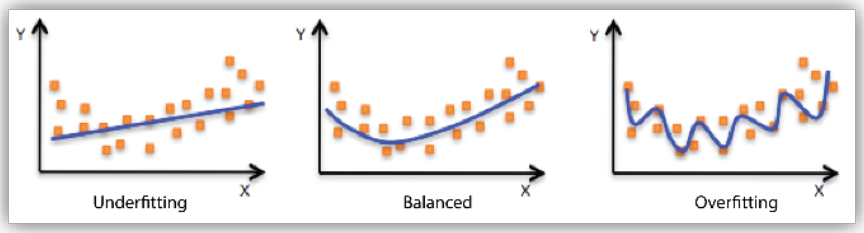


Figure 2.3: Overfitting example on the training data used for linear regression (Credits: Amazon [31])

2.3.1 Regularization

A good machine learning algorithm should perform well in both training data and new inputs. This reduction of the error in testing is known as regularization. A model that avoids overfitting is an example of a well-regularized algorithm. In sum, a good model should include the generating process and many other possible ones, while also matching the true data-generating process. There are many ways in which this can be performed.

2.3.1.1 Weight Decay

Many regularization approaches are based on limiting the capacity of the machine learning models, by adding a penalty norm to the objective function. One example is the L^1 or L^2 parameter regulation, commonly known as weight decay.

The term will be responsible for reducing some parameters that do not contribute to the optimization function to almost zero magnitude, focusing only on those that have contributed for it.

2.3.1.2 Dataset Augmentation

The first attempt an engineer should take in order to improve regularization is considering the possibility of increasing the existing training pool. In some cases, especially where the amount of data available is insufficient, the engineer could try to create fake data. In some classification problems, this approach is usually taken to increase the robustness of the algorithm.

An interesting perspective would be to introduce noisy datasets to the previously trained model, or even between inputs, from which hidden units could also be included.

In many Bayesian approaches to machine learning, the model weights are usually considered as uncertain, and presented as a probability distribution that reflects so. The addition of noise usually confirms this uncertainty, enhancing stability of the function to be learned. As it will be adopted later on this research, noise does not necessarily need to be just inserted for augmentation purposes. In fact, a engineer should pay attention to the possibility of this technique to push the model into regions where it is insensible to small variations, finding points where it may not necessarily be the minima, and consequently harming the training process as a whole.

Noise could also be inserted at the output. This technique is recommended when the previously known output has known errors. However, the purposeful inclusion of mistakes may cause more harm than good if the set is carefully reviewed. In some ways, it can hold the training process, leading to never-ending conversion processes.

2.3.1.3 Early Stopping

When training large models, one can notice the steady decay of the training set over time, while the validation sets forms a U-shaped curves, tending to rise again. With that being said, it is possible to return the training to the epoch which had the best parameters for the validation set.

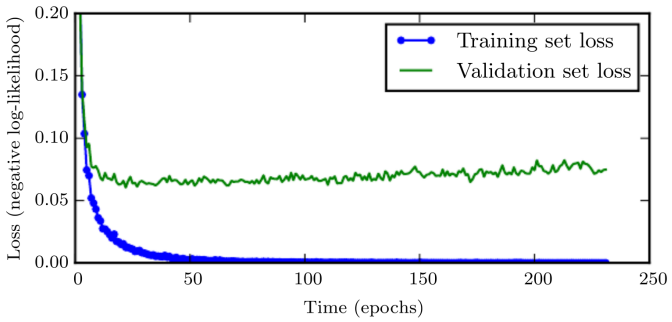


Figure 2.4: Comparison between the training and validation loss curves (Credits: Goodfellow [10])

This process is widely used and known as early stopping. This is an unobtrusive form of regularization, i.e. does not interfere directly with the sets used, and can be combined with many other commonly used techniques.

The biggest used issue provided by this technique may be related with the necessity of the inclusion of the validation set, taking away elements that could be used for training. In order to prevent this issue, one could adopt any of these strategies:

- Register the epoch in which the best hyperparameter of the validation set was recorded, re-initialize the model, incorporate the validation set back into the training pool and train until that epoch is reached;
- Halt the training in which the best hyperparameter of the validation set was recorded, incorporate the validation set back into the training pool and resume training until the loss function goes below the previously recorded one in the validation set;

Both methods have advantages and disadvantages. The first one provides us with a cleaner training set, but more than duplicates the training time. Also, since the training set is bigger, we do not have a firm assumption that the epoch registered is the *de facto* best before the overfitting phenomena has happened, since the dataset will require more parameter updates. At first thought, the second method takes far less time than the previous one, since the training is made on top of the first round, however, there is no guarantee that the new training set may reach the recorded value, keeping the process going on indefinitely.

On a conclusion note, early stopping can be similar to L^2 regularization. In fact, this

method has the advantage of automatically determine the correct amount of regularization, while weight decay requires many prior experiments to do so.

2.3.2 Optimization

In order to train a model correctly, one needs to find its accuracy between the training set and real test sets. It is imperative that the machine learning algorithm improves and adapts as tests are done. This ability to evolve is a form of optimization. Usually, the main concern is that the performance of the learning phase is made as fast as possible. The cost function that defines this spectrum is elicited in equation **(2.43)**, where L is a loss function, $f(\mathbf{x}, \boldsymbol{\theta})$ is the predicted output when the input is $\mathbf{x}$ and $\hat{p}_{Data}$ is the empirical distribution.

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{Data}} L\left(f(\mathbf{x}, \boldsymbol{\theta}), y\right) \quad (2.43)$$

The simpliest form the cost function, as stated in Goodfellow[10], can be written according with equation **(2.44)**.

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right), y^{(i)}\right) \quad (2.44)$$

Before advancing, and adapting a common expression, it is important to notice that simply solving this equation "can bring more harm than good". This empirical risk minimization process, due to its simpleness, can result in an over-fitting problem (as stated in Figure 2.3). When training, one should also adopt a validation set alongside the training and testing sets. Its main function should be to accompany the training evolution, allowing the engineer to understand the degree of over-fit based on the discrepancy between this and the training set.

2.3.2.1 Gradient-Descent Algorithms

The most effective modern optimization models are based on the gradient-descent theorem. By quickly defining the gradient property to minimize the objective function, J, as stated in equation **(2.45)**, we can write the stochastic gradient descent algorithm.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{Data}} \nabla_{\boldsymbol{\theta}} \log p_{Model}(\mathbf{x}, y, \boldsymbol{\theta}) \quad (2.45)$$

Usually, the algorithm is based in the following steps:

1. Define a learning rate, ε_k, and a initial parameter, $\boldsymbol{\theta}$;
2. Sample a batch of n examples from the training set with corresponding targets;
3. Compute the gradient estimate, as defined in equation **(2.46)**;
4. Update the $\boldsymbol{\theta}$, according with equation **(2.47)**;
5. Repeat the last three steps if the stopping criterion is not met.

$$\hat{\mathbf{g}} = \frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right), y^{(i)}\right) \quad (2.46)$$

$$\boldsymbol{\theta}_{New} = \boldsymbol{\theta}_{Old} - \varepsilon\hat{\mathbf{g}} \quad (2.47)$$

It is important to point a qualitative description of the learning rate. The general rule is that, the higher this value is, the faster the minimum is reached. The main problem is that the risk of that minimum being local is not low. If a lower value is selected, other issues appear, like the saddle points, for example, where the gradient becomes null. The engineer should select a reasonable value to avoid both cases.

Another method was introduced in order to reduce the time needed for optimization, adding momentum to the original computation. Here, similarly to realistic kinematics phenomena, we add a momentum, substituting the $-\varepsilon\hat{\mathbf{g}}$ term in equation **(2.47)** with $\mathbf{v}$ and replacing equation **(2.46)** completely with equation **(2.48)**, where $m_{\text{ML}} \in [0, 1]$ determines how quickly the contributions of previous gradients decay.

$$\mathbf{v}_{New} = m_{\text{ML}}\mathbf{v}_{Old} - \varepsilon\nabla_{\boldsymbol{\theta}}\left(\frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right), y^{(i)}\right)\right) \quad (2.48)$$

To sum up, this version has the following steps:

1. Define a learning rate, ε_k, a initial parameter, $\boldsymbol{\theta}$, a initial velocity, $\mathbf{v}$ and a momentum parameter, m_{ML};
2. Sample a batch of n examples from the training set with corresponding targets;
3. Compute the gradient estimate, as defined in equation **(2.48)**;

4. Update the $\boldsymbol{\theta}$, by adding $\mathbf{v}$ to it;
5. Repeat the last three steps if the stopping criterion is not met.

2.3.2.2 AdaGrad Algorithm

This algorithm has an adaptative learning rate, i.e. uses a separate learning rate for each parameter and automatically adapts these learning rates as the training process goes on. The algorithm goes as follows:

- Initialize the method with a global learning rate, ε, parameter, $\boldsymbol{\theta}$, another parameter to avoid a possible division by zero, $\delta = 10^{-7}$ and a gradient accumulation variable, $\mathbf{r} = \mathbf{0}$;
- Compute the gradient, as stated in equation **(2.49)**;
- Accumulate the squared gradient, as explained in equation **(2.50)** (where $\odot$ means element-wise multiplication);
- Update the parameter, according with equation **(2.51)** (with division and square root applied element-wise);
- Repeat the last three steps if the convergence criteria is not reached.

$$\mathbf{g} = \frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right), y^{(i)}\right) \quad (2.49)$$

$$\mathbf{r_{New}} = \mathbf{r_{Old}} + \mathbf{g} \odot \mathbf{g} \quad (2.50)$$

$$\boldsymbol{\theta}_{New} = \boldsymbol{\theta}_{Old} - \frac{\varepsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g} \quad (2.51)$$

It is important to note that this method, even though it works well with convex functions, it is not recommendable to train deep neural network models, as the accumulation of the beginning square gradients may result in a premature (and excessive) learning rate decay.

2.3.2.3 RMSProp

The RMSProp is a modification of the AdaGrad algorithm to perform better in non-convex shaped functions. In sum, it uses an exponential algorithm to force the system to discard

the very first gradients, allowing a faster convergence. This comes with the introduction of a new parameter - the decay rate, ρ_{ML}. With that, the algorithm goes as follows:

- Define a global learning rate, ε, a decay rate, ρ_{ML}, the initial parameter, the accumulation variables $\mathbf{r} = 0$, $\boldsymbol{\theta}$ and a small constant $\delta \approx 10^{-6}$ for the same reason as used in AdaGrad;
- Repeat the gradient computation as stated in equation **(2.49)**;
- Accumulate the squared gradient, as explained in the new equation **(2.52)**;
- Repeat the same $\boldsymbol{\theta}$ update formula as shown in equation **(2.51)**;
- Repeat the last three steps if the convergence criteria is not reached.

$$\mathbf{r}_{New} = \rho_{ML}\mathbf{r}_{Old} + (1 - \rho_{ML})\,\mathbf{g} \odot \mathbf{g} \quad (2.52)$$

Along with ADAM, this method is widely used in deep-learning.

2.3.2.4 ADAM

The Adaptative Moments algorithm, uses a combination of both the momentum and RMSProp methods. It is commonly used in deep-learning problems as well due to its robustness. However, it is important to note that this algorithm is still not perfect, as it may diverge due to poor variance control, in rare cases.

The method goes as follows:

- Define a step size $\varepsilon \approx 0.001$, exponential decay rates $\rho_1 \approx 0.9,\ \rho_2 \approx 0.999$, a small constant $\delta \approx 10^{-8}$, the initial parameters $\boldsymbol{\theta}$, the first and second moment variables, $\mathbf{m} = \mathbf{0}$ and $\mathbf{n} = \mathbf{0}$, and a time step $t = 0$;
- Compute the gradient as stated in equation **(2.49)**, and update the time $t_{New} = t_{Old} + 1$;
- Update the first and second moment estimates, as stated in equation **(2.53)**;
- Correct the bias in both moments, as shown in equation **(2.54)**; Compute and apply the update, as explained in equation **(2.55)**;

$$\begin{cases} \mathbf{m}_{New} = \rho_1 \mathbf{m}_{Old} + (1-\rho_1)\,\mathbf{g} \\ \mathbf{n}_{New} = \rho_2 \mathbf{n}_{Old} + (1-\rho_2)\,\mathbf{g} \odot \mathbf{g} \end{cases} \tag{2.53}$$

$$\begin{cases} \hat{\mathbf{m}} = \dfrac{\mathbf{m}_{New}}{1-\rho_1^t} \\ \hat{\mathbf{n}} = \dfrac{\mathbf{n}_{New}}{1-\rho_2^t} \end{cases} \tag{2.54}$$

$$\boldsymbol{\theta}_{New} = \boldsymbol{\theta}_{Old} - \varepsilon \frac{\hat{\mathbf{m}}}{\sqrt{\hat{\mathbf{n}}} + \delta} \tag{2.55}$$

2.3.2.5 NADAM

As an attempt to improve the rate of convergence present in the original ADAM, Doznat[7] attempted to implement Nesterov's accelerated gradient into it. The algorithm is defined below:

- Define ε_0, ..., ε_t; $\rho_{ML_0}, ..., \rho_{ML_t}$; n and δ.
- Initialize momentum vectors, $\mathbf{m}_0 = \mathbf{0}$, $\mathbf{n}_0 = \mathbf{0}$;
- Complete equations **(2.56)** to **(2.61)**;
- Repeat the last step if the conversion criteria is not met.

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_{t-1}} f_t(\boldsymbol{\theta}_{t-1}) \tag{2.56}$$

$$\mathbf{m}_t = \rho_{ML_t}\mathbf{m}_{t-1} + (1-\rho_{ML_t})\mathbf{g}_t \tag{2.57}$$

$$\mathbf{n}_t = n\mathbf{n}_{t-1} + (1-n)\mathbf{g}_t^2 \tag{2.58}$$

$$\hat{\mathbf{m}} = \frac{\rho_{ML_{t-1}}\mathbf{m}_t}{1-\Pi_{i=1}^{t+1}\rho_{ML_i}} + \frac{(1-\rho_{ML_t})\mathbf{g}_t}{1-\Pi_{i=1}^{t+1}\rho_{ML_i}} \tag{2.59}$$

$$\hat{\mathbf{n}} = \frac{n\mathbf{n}_t}{1-n^t} \tag{2.60}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \frac{\varepsilon_t}{\sqrt{\hat{\mathbf{n}}_t + \delta}}\hat{\mathbf{m}}_t \tag{2.61}$$

2.3.2.6 ADAMw

The usage of adaptive momentum gradients (such as ADAM) and gradient (like RMSProp), although being commonly chosen for many model training, are also usually shown as having a poor generalization. The main reasons may be due to nonidentical regularization and weight decay, or even a poor performance the prior one.

For those reasons, a weight decay (λ_{wd}) addition was proposed as an attempt to refine the already existent ADAM method, as described below[14]:

- Define the first variables as $\varepsilon = 0.001$, $\lambda_{\text{wd}} = 0.001$, $\rho_1 = 0.9$, $\rho_2 = 0.999$ and $\delta = 10^{-8}$;
- Initialize the time-step, the parameter vector, the first and second moment vectors and the schedule multiplier: $t_0 = 0$, $\boldsymbol{\theta}_{t_0}$, $\mathbf{m}_{t_0} = \mathbf{0}$, $\mathbf{v}_{t_0} = \mathbf{0}$, η_t_0.
- Define a stopping criterion;
- Solve equations **(2.62)** to **(2.70)** until the stopping criterion is met;
- Obtain the optimized parameter, $\boldsymbol{\theta}_n$;

$$
\begin{aligned}
t_{k+1} &= t_k + 1 && (2.62)\\
\nabla f_{t_k}\left(\boldsymbol{\theta}_{t_{k-1}}\right) &= \text{SelectBatch}\left(\boldsymbol{\theta}_{t_{k-1}}\right) && (2.63)\\
\mathbf{g}_{t_k} &= \nabla f_{t_k}\left(\boldsymbol{\theta}_{t_{k-1}}\right) + \lambda_{\text{wd}}\boldsymbol{\theta}_{t_{k-1}} && (2.64)\\
\mathbf{m}_{t_k} &= \rho_1 \mathbf{m}_{t_{k-1}} + (1-\rho_1)\,\mathbf{g}_{t_k} && (2.65)\\
\mathbf{n}_{t_k} &= \rho_2 \mathbf{n}_{t_{k-1}} + (1-\rho_2)\,\mathbf{g}^2_{t_k} && (2.66)\\
\hat{\mathbf{m}}_{t_k} &= \frac{\mathbf{m}_{t_k}}{1-\rho_1^{t_k}} && (2.67)\\
\hat{\mathbf{n}}_{t_k} &= \frac{\mathbf{n}_{t_k}}{1-\rho_2^{t_k}} && (2.68)\\
\eta_{t_k} &= \text{SetScheduleMultiplier}(t_k) && (2.69)\\
\boldsymbol{\theta}_{t_k} &= \boldsymbol{\theta}_{t_{k-1}} - \eta_{t_k}\left(\frac{\varepsilon\hat{\mathbf{m}}_{t_k}}{\sqrt{\hat{\mathbf{n}}_{t_k}}+\delta} + \varepsilon\boldsymbol{\theta}_{t_{k-1}}\right) && (2.70)
\end{aligned}
$$

2.3.2.7 Newton

This method is an example of a second-order algorithm, i.e., that uses second-order derivatives to improve the optimization process. This method is based on the second-order Taylor's expansion to approximate the cost function, $J(\boldsymbol{\theta})$ near some point $\boldsymbol{\theta}_0$, according to equation **(2.71)**, where **H** is the positive definite hessian matrix of J with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_0$.

$$
J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta}-\boldsymbol{\theta}_t)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}_t)^T \mathbf{H}(\boldsymbol{\theta}-\boldsymbol{\theta}_0) \quad (2.71)
$$

The method is based on the following algorithm:

- Start with an initial parameter, $\boldsymbol{\theta}_0$;

- Compute the gradient and the Hessian, according to equations **(2.49)** and **(2.72)**;
- Compute and apply the update, shown in equation **(2.73)**;
- Repeat the last two steps if the convergence criteria is not met.

$$\mathbf{H} = \frac{1}{n}\nabla^2_{\boldsymbol{\theta}} \sum_i^n L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}\right), \mathbf{y}^{(i)}\right) \quad (2.72)$$

$$\boldsymbol{\theta}_{New} = \boldsymbol{\theta}_{Old} - \mathbf{H}^{-1}\mathbf{g} \quad (2.73)$$

It is important to notice that, in the rare cases that the eigenvalues of the Hessian matrix are not positive in its entirety (i.e. the Hessian matrix is not positive definite), for example, near a saddle point (a common case in deep-learning), we can regularize it, by substituting equation **(2.73)** with **(2.74)**, in a new algorithm variation called Levenberg-Marquardt.

$$\boldsymbol{\theta}_{New} = \boldsymbol{\theta}_{Old} - \left[\mathbf{H}\left(f\left(\boldsymbol{\theta}_{Old}\right)\right) + \varepsilon\mathbf{I}\right]^{-1}\mathbf{g} \quad (2.74)$$

The ε value should be adjusted according with the distance to zero, i.e., the more negative the eigenvalues are, the higher the ε term needs to be. This may lead in a computationally heavy algorithm, becoming incompatible with large neural networks for the current processing capabilities. Even more, this method also requires a matrix inversion every step, which is also another argument for this motive of concern.

2.3.2.8 Conjugate Gradients

This method avoids the last issue pointed, by substituting the inversion process by iteratively descending conjugate directions. The algorithm is defined below:

- Define the initial parameters and gradients, $\boldsymbol{\theta}_0$, $\boldsymbol{\rho}_0 = \mathbf{0}$, $\boldsymbol{g}_0 = \mathbf{0}$ and $t = 1$;
- Compute the gradient $\mathbf{g}_t$ according with equation **(2.49)**;
- Compute β_t with Fletcher-Reeves' or Polak-Ribière's equation **(2.75)**, remembering that, for the case of a nonlinear conjugate gradient, β_t should be set back to zero if t is a multiple of previously defined set of iterations;
- Compute the search direction, explained in equation **(2.76)**;
- Perform line search, as stated in equation **(2.77)**, where $argmin$ are the input points where the function output is minimized (keeping in mind that, for the case of a truly quadratic cost function, ε^* should be analytically solved);

- Apply the update, as stated in equation **(2.78)** , and increase the t parameter by one;

- Repeat the last six steps, if the convergence criteria is not yet met.

$$\beta_t = \begin{cases} \dfrac{\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right)^T \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right)}{\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)^T \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)} & \text{(Fletcher-Reeves)} \\ \dfrac{\left(\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right) - \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)\right)^T \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_t\right)}{\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)^T \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}_{t-1}\right)} & \text{(Polak-Ribière)} \end{cases} \tag{2.75}$$

$$\boldsymbol{\rho}_t = -\mathbf{g}_t + \beta_t \boldsymbol{\rho}_{t-1} \tag{2.76}$$

$$\varepsilon^* = \operatorname{argmin}_{\varepsilon} \sum_{i=1}^{n} L\left(f\left(\mathbf{x}^{(i)}, \boldsymbol{\theta}_t + \varepsilon \boldsymbol{\rho}_t\right), \mathbf{y}^{(i)}\right) \tag{2.77}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \varepsilon^* \boldsymbol{\rho}_t \tag{2.78}$$

2.3.2.9 BFGS

The Broyden–Fletcher–Goldfarb–Shanno algorithm tries to reduce the computational burden that the Newton method is, just like the previous shown method. However, instead of exclusively relying in conjugate gradients to accurately find a close point to the minimum along the search line, it computes an approximation of the inverse Hessian matrix, $\mathbf{M}_t$, that becomes closer to the true matrix. The algorithm is defined below:

- Define the initial values $\boldsymbol{\theta}_0$, $\boldsymbol{\rho}_0 = \mathbf{0}$, $\mathbf{g}_0 = \mathbf{0}$, $t = 1$ and $\mathbf{M}_0 = \mathbf{I}$;

- Obtain the search direction, according with equation **(2.79)**;

- Perform line search, as stated in equation **(2.77)**;

- Apply the update, as stated in equation **(2.78)**, and increase the t parameter by one;

- Compute the gradient according with equation **(2.49)**;

- Obtain the difference vectors, as shown in equation **(2.80)**

- Compute the new Hessian inverse approximate, as stated in equation **(2.81)**;

- Repeat the last six steps if the convergence criteria is not reached yet.

$$\boldsymbol{\rho}_t = \mathbf{M}_t \mathbf{g}_t \tag{2.79}$$

$$\begin{cases} \mathbf{m}_t = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t \\ \mathbf{n}_t = \mathbf{g}_{t+1} - \mathbf{g}_t \end{cases} \tag{2.80}$$

$$\mathbf{M}_{t+1} = \left(\mathbf{I} - \frac{\mathbf{m}_t \mathbf{n}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \right) \mathbf{M}_t \left(\frac{\mathbf{n}_t \mathbf{m}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \right) + \frac{\mathbf{m}_t \mathbf{m}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \tag{2.81}$$

This method, even though it is faster than the original Newton one, still relies in intense memory requirements, as it needs to store $\mathbf{M}_t$. One assumption to decrease this requirement is by substituting equation **(2.81)** with **(2.82)**, removing the need to store $\mathbf{M}_t$ as the iteration goes on. This variation is called Limited Memory BFGS (or L-BFGS).

$$\mathbf{M}_{t+1} = \left(\mathbf{I} - \frac{\mathbf{m}_t \mathbf{n}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \right) \mathbf{I} \left(\frac{\mathbf{n}_t \mathbf{m}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \right) + \frac{\mathbf{n}_t \mathbf{m}_t^T}{\mathbf{n}_t^T \mathbf{m}_t} \tag{2.82}$$

2.3.2.10 Notes on Hyperparametrization Tuning

In order to enhance the accuracy of any trained model, one needs to apply the most adequate hyperparameters for a certain learning algorithm. In most cases, they include the learning rate, activation functions, loss functions, or even the batch size.

Currently, various methods exist to accurately choose these parameters:

- Manual search;
- Grid search;
- Random search;
- Bayesian search.

When doing a manual search, as the name implies, the engineer usually relies on intuition and repetitive trial-and-error to obtain the most desired model, i.e. a model that performs well in a wide variety of cases, with good regularization and good accuracy. This implies the need for one to understand the impact of such hyperparameters in the final model. Goodfellow [10] presents us with some examples, where the learning rate should be highlighted, as it controls the effective capacity of a model if a way far more complex than the rest of the other hyperparameters.

- Number of hidden units - If increased, the representational capacity of the model is positively impacted, but may lead to a slower training;

- Learning Rate - When tuned optimally, enhanced the capacity of the model;
- Dropout rate - When decreased, i.e. when the discard process happens after a larger number of epochs, may lead to a smaller error;
- Convolution kernel width - When increased, the parameters present on the model also increase, but may lead to memory shortage.

The rest of the methods follow an automatic route, without the need for human intervention. They are excellent when the user has no previous knowledge of earlier studied models in the same subject, or in the begin of studying machine learning, however, they are usually slow, and require intense computational load.

When the number of hyperparameters is less than three, it is recommended to use grid search. The user sets a small finite set of values, which are then interpreted by the algorithm to select the best version. We can check its operation for a small set of two hyperparameters (with 3 values each) in Figure 2.5. The major drawback of this method is related with the amount of computational waste when running it.

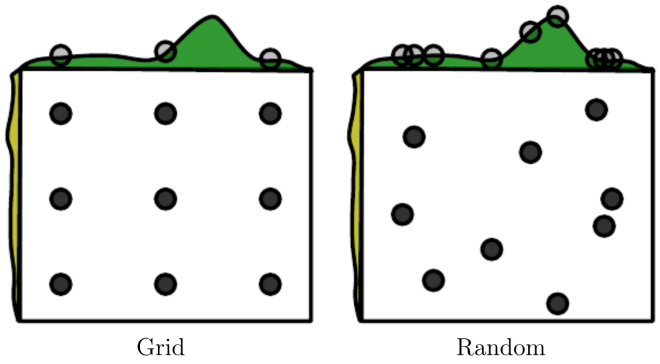


Figure 2.5: Comparison between searches (Credits: Goodfellow [10])

The other method relies on the use of distributions to enhance the search. Basically, with the same ease of programming offered by the prior algorithm, we can apply whether a Bernoulli or multinoulli for binary or discrete, or a uniform distribution on a log scale for continuous values. This approach allows us to reduce the computational burden that the prior method required to run on experimental trials that were, in the perspective of the user, unnecessary, contributing to a faster convergence.

The last algorithm covered in this thesis is the Bayesian search. Let's firstly define the Bayes' rule, as stated in equation **(2.83)**.

$$P\left(x|y\right) = \frac{P\left(x\right)P\left(y|x\right)}{P\left(y\right)} \approx \frac{P\left(x\right)P\left(y|x\right)}{\sum_x P\left(y|x\right)P\left(x\right)} \quad (2.83)$$

This property can be applied in the search through the sets of hyperparameters defined. It

defines this search as a probabilistic model, with the objective of minimizing an objective function, exploiting the Bayes's rule to make decisions about where in the hyperparameter space to next evaluate the function. This method, however, should be taken with relative carefulness, as it may sometimes perform better than most users, os sometimes taking unexplained decisions that may lead to results far worse than other searches.

2.3.3 Recurrent Neural Networks

Neural networks are deep learning models. Their goal is to approximate some function f to present the accurate value of $\mathbf{y}$, by learning the parameters of $\boldsymbol{\theta}$ that most accurately describe it, with $\mathbf{x}$ as an input, as shown in equation **(2.84)**.

$$f^* (\mathbf{x}, \boldsymbol{\theta}) \approx \mathbf{y} \quad (2.84)$$

Since machine learning is commonly used to predict non-linear functions, it usually decomposes them in simpler ones, like shown in equation **(2.85)** (with the parameter $\boldsymbol{\theta}$ omitted). From now on, it is important for the reader to perceive this as layers. The first layer, $f^{(1)}$, is usually defined as the input layer, since the information starts flowing from there and the last, $f^{(n)}$, is the output layer. The rest are called hidden layers, since it is not meant for the engineer to study their output. A general overview of the network can be seen in Figure 2.6. Some authors like Goodfellow[10] also classify the input as an hidden layer, but this view will not be adopted in this document.

$$f(\mathbf{x}) = f^{(n)} \left(f^{(n-1)} \left(\dots \left(f^{(1)} (\mathbf{x}) \right) \dots \right) \right) \quad (2.85)$$

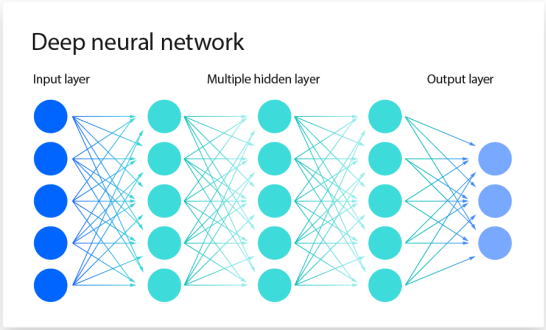


Figure 2.6: Example of a large neural network (Credits: IBM [36])

The major difference between this and other models shown until now is the use of gradient-

based optimizers to try to drive the cost function to the lowest value, with global convergence guarantees, with the weights (another definition of the parameters) of each function initialized to a small random value.

The architecture of a layer $i \in \{1, ..., m\}$ is defined in equation **(2.86)**, considering $\mathbf{h}^{(i)} \in \mathbb{R}^{n \times h}$ as the output of the hidden layer, $\boldsymbol{\sigma}^{(i)}$ as the activation function, $\mathbf{X}^{(i)} \in \mathbb{R}^{n \times d}$ as the d input(s) with size n, joined into one matrix, $\mathbf{W}^{(i)} \in \mathbb{R}^{d \times h}$ as the weight parameter and $\mathbf{b}^{(i)} \in \mathbb{R}^{1 \times h}$ as the bias parameter.

$$\begin{cases} \mathbf{h}^{(i)} = \boldsymbol{\sigma}^{(i)} \left(\mathbf{X}^{(i)} \mathbf{W}^{(i)} + \mathbf{b}^{(i)} \right) \\ \mathbf{X}^{(i+1)} = \mathbf{h}^{(i)} \end{cases} \tag{2.86}$$

2.3.3.1 Gate Recurrent Units

Gated Recurrent Units are capable of storing vital information obtained at some point in time during training, bringing it in back when needed to increase the accuracy of the model.

These models also have the capacity of independently decide if the information obtained is no longer needed, using forget gates, distinguishing them from the original RNN's.

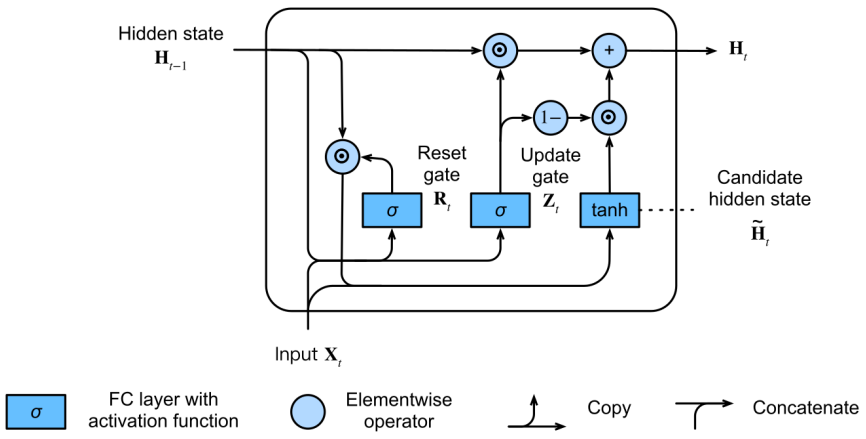


Figure 2.7: Overview of a GRU model (Credits: Zhang [30])

While observing Figure 2.7, we can check the elements that constitute each unit in time-step $t \in 1, ..., m$:

- The input, $\mathbf{X}_t \in \mathbb{R}^{n \times d}$;
- The hidden state of the previous time step, $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$
- The reset gate, $\mathbf{R}_t \in \mathbb{R}^{n \times h}$;

- The update gate, $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$;
- The candidate hidden state, $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$;

We can obtain the equations **(2.87)** to **(2.89)** that show its computation, for n examples, d inputs, h hidden units, $\mathbf{W}_{xr}, \mathbf{W}_{xz}, \mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hr}, \mathbf{W}_{hz}, \mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ weight parameters, and $\mathbf{b}_r, \mathbf{b}_z, \mathbf{b}_h \in \mathbb{R}^{1 \times h}$ biases.

$$\mathbf{R}_t = \sigma\left(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r\right) \tag{2.87}$$

$$\mathbf{Z}_t = \sigma\left(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z\right) \tag{2.88}$$

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h\right) \tag{2.89}$$

The reset and update gates work on somewhat opposite ways. One could say that the former gate works to compute how much of the old information is wanted to be forgotten, and the other checks how much is to be updated for future iterations, by the use of a sigmoid function. However, they hold a deeper meaning, in a way that the latter has the ability to capture long-term dependencies, while the other can remember short-term relations. The candidate hidden state is to the amount of information that may (or may not) be passed to the next hidden state from this iteration.

2.3.3.2 The LSTM Network

The Long-Short Term Memory Networks were implemented as a way of preserving temporal information from earlier datasets, unlike Gate Recurrent Units. Different from conventional neural-networks, they can preserve information that may lead in a more accurate prediction in time-based models. For our case, this was the used model, according to the nature of our root issue.

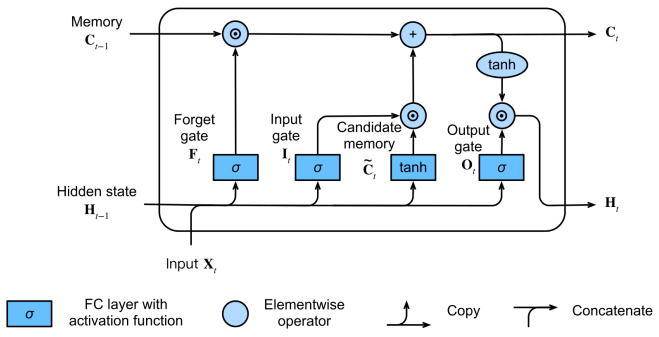


Figure 2.8: Overview of a LSTM model (Credits: Zhang [30])

Upon further observation of Figure 2.8, it can be seen that this model has a few elements:

- Forget Gate, $\mathbf{F}_t$: The gate that decides how much of the past information is used;
- Input Gate, $\mathbf{X}_t$: The gate that decides how much of the received information for the current time-stamp is used;
- Output Gate, $\mathbf{O}_t$: The gate the decides how much of the received information is passed to the next hidden state;
- Candidate Memory, $\widetilde{\mathbf{C}}_t$: The gate that decides how much of the recieved information is passed to the Memory;
- Hidden State, $\mathbf{H}_t$, contains the short-term information from the earlier cells;
- Memory, $\mathbf{C}_t$, contains the long-term information from the earlier cells.

The gate functions are presented in equations **(2.90)** to **(2.95)**, where $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ is the input, $\mathbf{H}_t \in \mathbb{R}^{n\times h}$ is the hidden state, $\mathbf{I}_t \in \mathbb{R}^{n\times h}$ is the input gate, $\mathbf{F}_t \in \mathbb{R}^{n\times h}$ is the forget gate, $\mathbf{O}_t \in \mathbb{R}^{n\times h}$ is the output gate, $\widetilde{\mathbf{C}}_t \in \mathbb{R}^{n\times h}$ is the candidate memory, $\mathbf{C}_t \in \mathbb{R}^{n\times h}$ is the memory cell, σ is an activation function, $\odot$ is a point-wise multiplication, $\mathbf{W}$ are various weight parameters to be found by the model ($\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo}, \mathbf{W}_{xc} \in \mathbb{R}^{d\times h}$, $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho}, \mathbf{W}_{hc} \in \mathbb{R}^{h\times h}$) and $\mathbf{b} \in \mathbb{R}^{1\times h}$ are bias parameters that follow the same procedure.

$$\mathbf{I}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i\right) \tag{2.90}$$
$$\mathbf{F}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right) \tag{2.91}$$
$$\mathbf{O}_t = \sigma\left(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o\right) \tag{2.92}$$
$$\widetilde{\mathbf{C}}_t = \tanh\left(\mathbf{X}_t\mathbf{W}_{xc} + \mathbf{H}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c\right) \tag{2.93}$$
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \widetilde{\mathbf{C}}_t \tag{2.94}$$
$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh\left(\mathbf{C}_t\right) \tag{2.95}$$

2.3.3.3 Notes on Activation Functions

It is also important to expand on the meaning of activation function. This can be defined as the decider if the neuron is activated, i.e, if the information is passed to the next iteration. Most used examples are *Rectified Linear Unit*, *sigmoid* and *tanh*, which are present in equations **(2.96)**, **(2.97)** and **(2.98)**.

$$ReLU\,(x) = \max\,(x, 0) \tag{2.96}$$

$$sigmoid\,(x) = \frac{1}{1 + \exp(-x)} \tag{2.97}$$

$$\tanh\,(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \tag{2.98}$$

The Rectified Linear Units are really useful for object recognition due to its binary behavior. The philosophy behind this method explains that every model is easy to optimize if their behavior is closer to linear. Recurrent networks, where sequential inputs are present, may use this function to easily propagate information through several time steps. For LSTM networks, there is a specific kind of linear activation.

The logistic sigmoid and hyperbolic tangent activation functions are closely related. For the case of the first one, their use is now largely discouraged in feedforward networks (and, consequently, in our case). Apart from that, even if a user would be encouraged to try and use it, the second one is a better choice due to its similarity to the identity function.

Unfortunately, as present in the *TensorFlow* [1] documentation, this study involves the usage of *CuDNNLSTM* layers. This technology was developed by *NVIDIA* to leverage the GPU usage in Machine Learning training, providing a considerable acceleration in this procedure. The downside with this technique is the only availability on using the *tanh* as an activation function. As a trade-off, following the same philosophy as Senra [20], it was decided to prioritize the choice of this framework.

2.3.3.4 Notes on Dimensionality

One may notice that, as the number of dimensions to study increase, so does the complexity of the problem. This phenomena is known as the curse of dimensionality. A visual example can be seen in Figure 2.9.

In this case, we can see that the space of optimization comprises only 10 possible solutions. If we increase to a two dimensional space, the number of solutions increase to $10 \cdot 10 = 100$ possible outcomes. The last image presents the 1000 space of possibilities for the three dimensional case.

We can also analyze a perfect example for this thesis. If we increase the number of outputs, then the number of weights to optimize will increase exponentially. This increase in complexity will lead to a slower process, as well as a decrease in result precision due to the nature of the problem. With this into account, an engineer should try to simplify the model as best as possible to ensure its speed and accuracy.

Figure 2.9: The curse of dimensionality visualized (Credits: Goodfellow [10])

Chapter 3

Simulation Models

3.1 Three-DOF Dynamic Model

This model was based on Wang[27]. The trajectory of a hypersonic vehicle is considered as a three-DOF reentry motion model of a rotating sphere, where the sideslip angle is zero. The position parameters, including the geocentric distance r, longitude θ, and latitude φ, are defined in the geocentric spherical fixed coordinate system. The velocity parameters include the velocity V, track angle γ , and course angle ψ. For these formulas, the values, σ_b, L and D represent, the bank angle, the Lift and the Drag, respectively. The dynamic three-DOF reentry motion equations expressed by the above-listed parameters are as presented in **(3.1)**, **(3.2)**, **(3.3)**, **(3.4)**, **(3.5)**, **(3.6)**:

$$\frac{dr}{dt} = V \sin\gamma \quad (3.1)$$

$$\frac{d\theta}{dt} = \frac{V \cos\gamma \sin\psi}{r \cos\varphi} \quad (3.2)$$

$$\frac{d\varphi}{dt} = \frac{V \cos\gamma \cos\psi}{r} \quad (3.3)$$

$$\frac{dV}{dt} = -\frac{D}{m} - g \sin\gamma \quad (3.4)$$

$$\frac{d\gamma}{dt} = \frac{1}{V}\left[\frac{L \cos\sigma_b}{m} + \left(\frac{V^2}{r} - g\right) \cdot \cos\gamma\right] \quad (3.5)$$

$$\frac{d\psi}{dt} = \frac{1}{V}\left(\frac{L \cdot \sin\sigma_b}{m \cdot \cos\gamma} + \frac{V^2}{r} \cdot \cos\gamma \cdot \sin\psi \cdot \tan\varphi\right) \quad (3.6)$$

3.2 Constraint Modeling

3.2.1 Dynamic Pressure

Considering the influence of the dynamic pressure on the requirement for lateral stability of the control system, the constraint of the dynamic pressure in the reentry process, which is explained in equation **(3.7)** needs to be met.

$$q = \frac{1}{2} \cdot \rho V^2 \quad (3.7)$$

3.2.2 Heat Flow

Considering the stagnation point is an area where a vehicle is heated more severely, the heat flow of the stagnation is generally taken as a constraint. It is modeled according with equation **(3.8)**.

$$\dot{Q} = K\left(\frac{\rho}{\rho_0}\right)^{n_Q}\left(\frac{V}{V_c}\right)^{m_Q} \quad (3.8)$$

3.2.3 Overload

The overload constraint needs to be considered in the reentry process for the purpose of structural safety. S_{wet} is the reference area, C_L is the lift coefficient of the body and E is the ratio between lift and drag. The overload constraint is defined according to equation **(3.11)**, where the Lift(L) and Drag(D) are modeled in equations **(3.9)** and **(3.10)**, respectively.

$$L = \frac{1}{2}\rho S_{wet} V^2 C_L \quad (3.9)$$

$$D = \frac{L}{E} \quad (3.10)$$

$$n_{\text{load}} = \sqrt{L^2 + D^2} \quad (3.11)$$

3.3 Atmospheric Model

In order to have more accuracy in the atmosphere modeling, the *ICAO* Standard Atmosphere [8] equations are in use, modeling the temperature, pressure and density with more realism until 86 kilometers. Should initial conditions be above this altitude, the USA 1976 Standard Atmosphere [16] model is adopted since the previous model does not cover it. The gravitational acceleration uses a standard exponential model, adopted in the *ISA* document [38].

3.3.1 Gravitational Acceleration, g

Considering that the radius of the Earth(r_0) is 6371 kilometers and that the gravitational acceleration at the surface of the Earth is $g_0 = 9.80665m/s^2$, the equation for the gravity acceleration can be consulted in **(3.12)**.

$$g(h_{alt}) = g_0 \cdot \left(\frac{r_0}{r_0 + h_{alt}}\right)^2 \quad (3.12)$$

3.3.2 Temperature, T

Considering that the atmosphere is a dynamic environment, i.e. its behavior changes depending in which layer the body is located, the temperature equation is no exception. For this equation, $T_0 = T(h_{alt} = 0) = 288.15K$ and $T_{11k} = T(h_{alt} = 11000) = 216.65K$ are the standard temperatures from sea-level up to the tropopause, respectively. Considering that the temperature variation constant is $\lambda = -6.5 \cdot 10^{-3}$ K/m in the troposphere layer, i.e. up to 11 kilometers, and its value changes to $\lambda_{20k} = 1 \cdot 10^{-3}$ K/m, the equations are elucidated in **(3.13)**. Following the same procedures, other constants are applied: $T_{32k} = 228.65K$ and $\lambda_{32k} = 2.8 \cdot 10^{-3}$ for the mid-stratosphere, $T_{47k} = 270.65K$ for the upper-stratosphere, $T_{51k} = 270.65K$ and $\lambda_{51k} = -2.8 \cdot 10^{-3}$ for the lower mesosphere, $T_{71k} = 214.65K$, $\lambda_{71k} = -2.0 \cdot 10^{-3}$K/m on the upper mesosphere, and on the termosphere $T_{86k} = 186.87K$, $T_{91k} = 263.19K$, $T_{110k} = 240K$, $\lambda_{110k} = 1.2 \cdot 10^{-2}$K/m, $T_{120k} = 360K$, $\lambda_{120k} = 1.2 \cdot 10^{-2}$K/m.

$$T(h_{alt}) = \begin{cases} T_0 + \lambda\, h_{alt} & h_{alt} \in [0, 11]\ \text{km} \\ T_{11k} & h_{alt} \in]11, 20]\ \text{km} \\ T_{20k} + \lambda_{20k}\,(h_{alt} - 20000) & h_{alt} \in]20, 32]\ \text{km} \\ T_{32k} + \lambda_{32k}\,(h_{alt} - 32000) & h_{alt} \in]32, 47]\ \text{km} \\ T_{47k} & h_{alt} \in]47, 51]\ \text{km} \\ T_{51k} + \lambda_{51k}\,(h_{alt} - 51000) & h_{alt} \in]51, 71]\ \text{km} \\ T_{71k} + \lambda_{71k}\,(h_{alt} - 71000) & h_{alt} \in]71, 86]\ \text{km} \\ T_{86k} & h_{alt} \in]86, 91]\ \text{km} \\ T_{91k} - 76.32\left[1 - \left(\dfrac{h_{alt} - 91000}{-19942.9}\right)^2\right]^{1/2} & h_{alt} \in]91, 110]\ \text{km} \\ T_{110k} + \lambda_{110k}\,(h_{alt} - 110000) & h_{alt} \in]110, 120]\ \text{km} \\ 1000 - (1000 - T_{120k}) \exp\left[\lambda_{120k} \dfrac{(120000 - h_{\text{alt}})(r_0 + 120000)}{r_0 + h_{alt}}\right] & h_{alt} \in]120, 200]\ \text{km} \end{cases} \tag{3.13}$$

3.3.3 Density, ρ

Considering that the density at the surface of the Earth is $\rho_0 = 1.225 kg/m^3$ and that the perfect gas constant is $R = 287.05 J/kg/K$, the equation can be viewed in **(3.14)**. For simplification purposes, the air was considered as a perfect gas over 86 kilometers.

$$\rho(h_{alt}) = \begin{cases} \rho_0 \left(\dfrac{T}{T_0}\right)^{-\frac{g_0}{\lambda R}-1} & h_{alt} \in [0; 11]\text{km} \\ \rho_{11k} \exp\left(\dfrac{-g_0}{R \cdot T_{11k}}\,(h_{alt} - 11000)\right) & h_{alt} \in]11; 20]\text{km} \\ \dfrac{p}{RT} & h_{alt} \in]20; 200]\text{km} \end{cases} \tag{3.14}$$

3.3.4 Pressure, p

Considering that $p_0 = 101325 Pa$, $p_{11k} = 22632 Pa$ and $p_{20k} = 5474.9 Pa$ are the pressures for the surface of the Earth, at the beginning of the tropopause and at the beginning of the stratosphere, respectively, the equation for this physical value is shown in **(3.15)**.

Following the same procedure, other constants are applied: in the mid–stratosphere $p_{32k} = 868.02 Pa$, in the upper-stratosphere $p_{47k} = 110.91 Pa$, in the lower mesosphere $p_{51k} =$

$66.94Pa$ and, above it, $p_{71k} = 3.96Pa$ and $p_{86k} = 0.3734Pa$. The value of the exponential correlation is given by equation **(3.16)**.

$$p(h_{alt}) = \begin{cases} p_0 \left(1 + \dfrac{\lambda\, h_{alt}}{T_0}\right)^{-\frac{g_0}{\lambda R}} & h_{alt} \in [0, 11] \text{ km} \\ p_{11k} e^{\left[-\dfrac{g_0\,(h_{alt} - 11000)}{R\,T_{11k}}\right]} & h_{alt} \in]11, 20] \text{ km} \\ p_{20k} \left(\dfrac{T}{T_{20k}}\right)^{-\frac{g_0}{\lambda_{20k} R}} & h_{alt} \in]20, 32] \text{ km} \\ p_{32k} \left(\dfrac{T}{T_{32k}}\right)^{-\frac{g_0}{\lambda_{32k} R}} & h_{alt} \in]32, 47] \text{ km} \\ p_{47k} e^{\left[-\dfrac{g_0\,(h_{alt} - 47000)}{R\,T_{47k}}\right]} & h_{alt} \in]47, 51] \text{ km} \\ p_{51k} \left(\dfrac{T}{T_{51k}}\right)^{-\frac{g_0}{\lambda_{51k} R}} & h_{alt} \in]51, 71] \text{ km} \\ p_{71k} \left(\dfrac{T}{T_{71k}}\right)^{-\frac{g_0}{\lambda_{71k} R}} & h_{alt} \in]71, 86] \text{ km} \\ p_{86k} \left(\dfrac{T}{T_{86k}}\right)^{-\frac{g_0}{\alpha_T R}} & h_{alt} \in]86, 200] \text{ km} \end{cases} \tag{3.15}$$

$$\alpha_T = \frac{T_{200} - T_{86}}{200000 - 86000} = 9.53 \cdot 10^{-3} \tag{3.16}$$

3.3.5 Speed of Sound, a_{sound}

In order to obtain the Mach speed, the only element unknown in the formula **(3.17)** is $\gamma_0 = 1.4$. Note that the speed in Mach is obtained by the division between the speed, in m/s and a_{sound}.

$$a_{\text{sound}} = \sqrt{\gamma_0 \cdot R \cdot T} \tag{3.17}$$

3.4 Noise Model

The turbulence model proposed by Dryden[41] allows us to model possible disturbances in our reentry model, by further improving the robustness of it. It defines the behavior of possible gusts in spectral densities as shown in equations **(3.18)**, **(3.19)** and **(3.20)**,

where σ_i and L_{turb_i} are the root medium square (RMS) velocity and length scale of turbulence, for $i \in \{u, v, w\}$, and $\Omega_f = \omega/V$ is the spatial frequency, given by the division between the time frequency and the velocity module.

$$\Phi_{u_g}\left(\Omega_f\right) = \sigma_u^2 \frac{2L_{turb_u}}{\pi} \frac{1}{1 + \left(L_{turb_u}\Omega\right)^2} \tag{3.18}$$

$$\Phi_{v_g}\left(\Omega_f\right) = \sigma_v^2 \frac{2L_{turb_v}}{\pi} \frac{1 + 12\left(L_{turb_v}\Omega\right)^2}{\left(1 + 4\left(L_{turb_v}\Omega\right)^2\right)^2} \tag{3.19}$$

$$\Phi_{w_g}\left(\Omega_f\right) = \sigma_w^2 \frac{2L_{turb_w}}{\pi} \frac{1 + 12\left(L_{turb_w}\Omega\right)^2}{\left(1 + 4\left(L_{turb_w}\Omega\right)^2\right)^2} \tag{3.20}$$

Adopting the methodology provided by Madden [15] in equations **(3.21)**, **(3.22)** and **(3.23)**, where $\omega_i\left(t\right)$ are white noise for $i \in \{u, v, w\}$, we can obtain a time-based function, by using a zero-order hold to discretize the turbulence filters. A more detailed description can be consulted in Annex A.

$$u_{\text{Noise}} = \omega_u \cdot \sigma_u \cdot \sqrt{\frac{\pi}{\Delta t}} \frac{\sqrt{\frac{2V}{\pi L_{turb_u}}}}{s + \frac{V}{L_{turb_u}}} \tag{3.21}$$

$$v_{\text{Noise}} = \omega_v \cdot \sigma_v \cdot \sqrt{\frac{\pi}{\Delta t}} \cdot \frac{\sqrt{\frac{3V}{\pi L_{turb_v}}}}{s + \frac{V}{L_{turb_v}}} \cdot \frac{s + \frac{V}{\sqrt{3}L_{turb_v}}}{s + \frac{V}{L_{turb_v}}} \tag{3.22}$$

$$w_{\text{Noise}} = \omega_w \cdot \sigma_w \cdot \sqrt{\frac{\pi}{\Delta t}} \cdot \frac{\sqrt{\frac{3V}{\pi L_{turb_w}}}}{s + \frac{V}{L_{turb_w}}} \cdot \frac{s + \frac{V}{\sqrt{3}L_{turb_w}}}{s + \frac{V}{L_{turb_w}}} \tag{3.23}$$

3.4.1 RMS Velocity and Length Scale of Turbulence

According to Dryden's model, we can also model both σ_i and L_{turb_i}, according to altitude and probability. The equations for L_{turb_i} are defined in imperial units in **(3.24)**, **(3.25)** and **(3.26)**. The same principle applies for the σ_i equations, in **(3.27)**, **(3.28)** and **(3.29)**, where the turbulence severity term, W_{20}, is defined in Table 3.1.

$$L_{turb_u} = \begin{cases} \dfrac{h_{alt}}{(0.177 + 0.000823h)^{1.2}} & h_{alt} < 1000\text{ft} \\ 1000 & h_{alt} = 1000\text{ft} \\ 250 + 0.75h_{alt} & 1000 < h_{alt} < 2000\text{ft} \\ 1750 & h_{alt} \geq 2000\text{ft} \end{cases} \tag{3.24}$$

$$L_{turb_v} = \begin{cases} \dfrac{0.5h_{alt}}{(0.177 + 0.000823h_{alt})^{1.2}} & h_{alt} < 1000\text{ft} \\ 500 & h_{alt} = 1000\text{ft} \\ 125 + 0.375h_{alt} & 1000 < h_{alt} < 2000\text{ft} \\ 875 & h_{alt} \geq 2000\text{ft} \end{cases} \tag{3.25}$$

$$L_{turb_w} = \begin{cases} 0.5h_{alt} & h_{alt} < 1000\text{ft} \\ 500 & h_{alt} = 1000\text{ft} \\ 125 + 0.375h_{alt} & 1000 < h_{alt} < 2000\text{ft} \\ 875 & h_{alt} \geq 2000\text{ft} \end{cases} \tag{3.26}$$

$$\sigma_u = \begin{cases} \dfrac{\sigma_w}{(0.177 + 0.000823h_{alt})^{0.4}} & h_{alt} < 1000\text{ft} \\ 0.1W_{20} & h_{alt} \geq 1000\text{ft} \end{cases} \tag{3.27}$$

$$\sigma_v = \begin{cases} \dfrac{\sigma_w}{(0.177 + 0.000823h_{alt})^{0.4}} & h_{alt} < 1000\text{ft} \\ 0.1W_{20} & h_{alt} \geq 1000\text{ft} \end{cases} \tag{3.28}$$

$$\sigma_w = 0.1W_{20} \tag{3.29}$$

Turbulence Severity	W_{20}[kts]	Probability[%]
Light	15	99.89999
Moderate	30	10^{-1}
Severe	45	10^{-5}

Table 3.1: Turbulence intensities and their probability

3.5 LSTM Model

The state vector dealt by the model can be defined in equation **(3.30)**.

$$\mathbf{X}_{clean}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} \quad (3.30)$$

The noise input is denoted according with equation **(3.31)**.

$$\begin{cases} \mathbf{X}(t) = \mathbf{X}_{\text{clean}}(t) \\ \mathbf{Y}(t) = \mathbf{X}_{measured}(t) = \mathbf{X}_{clean}(t) + \mathbf{X}_{noise}(t) = \mathbf{X}_{clean}(t) + \boldsymbol{\mathcal{D}}\left(\boldsymbol{\omega}(t)\right) \end{cases} \quad (3.31)$$

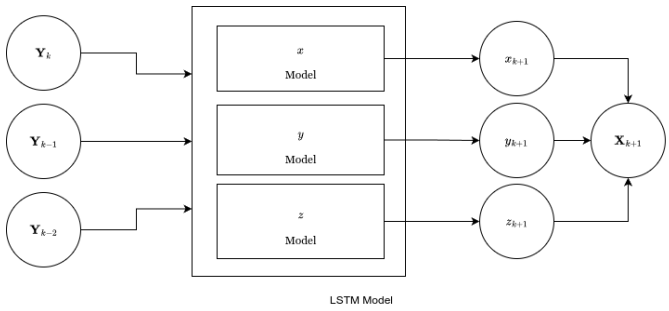


Figure 3.1: Flowchart of the LSTM model behavior

The input of the LSTM model is, then, defined according with equation **(3.32)**, and the output is combined from each model to produce the one shown in equation **(3.33)**.

$$\Big[[\mathbf{Y}_k] \quad [\mathbf{Y}_{k-1}] \quad [\mathbf{Y}_{k-2}]\Big] = \left[\begin{bmatrix} x_{measured_k} \\ y_{measured_k} \\ z_{measured_k} \end{bmatrix} \quad \begin{bmatrix} x_{measured_{k-1}} \\ y_{measured_{k-1}} \\ z_{measured_{k-1}} \end{bmatrix} \quad \begin{bmatrix} x_{measured_{k-2}} \\ y_{measured_{k-2}} \\ z_{measured_{k-2}} \end{bmatrix} \right] \quad (3.32)$$

$$\mathbf{X}_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ z_{k+1} \end{bmatrix} \quad (3.33)$$

To define the LSTM model, firstly it was proposed a MIMO model, but that assumption was quickly discarded, since a MISO approach would allow for a reduction in needed input, while also increasing the accuracy (Senra[20]). Furthermore, as seen in Chapter

2.3.3.4, if the output parameters are decreased, the training process will occur in a space with lower dimensionality, optimizing each parameter with more efficiency, speed and precision. For that, the main model is comprised on three MISO models, as shown in Figure 3.1.

3.5.1 Hyperparameter Tuning

The hyparparameters studied were the ones below:

- Hidden Layers: {1; 2; 3};
- Hidden Layer Inputs: {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096};
- Optimizer: {SGD, RMSProp, ADAM, NADAM, ADAMw};
- Loss Functions: {MAE, MSE, Huber};
- Loss Ratio: [1e-6, 1e-3];
- Batch Size: {32, 64, 128}.

The method to use is the Tree-Structured Parzen Estimator [28]. Its algorithm is briefly described below, defining a number of trials, N_{trials}, a number of parameters to analyze, N_{param}, a function to compute the top quantile, i.e., the set of promising trials, $\Gamma_{TPE}(\cdot)$, a function to compute the weigths, $W_{TPE}(\cdot)$, a Kernel function, $K(\cdot)$ and a function to compute the bandwidth, $B_{TPE}(\cdot)$.

- Select a random set of variables to analyze, $\boldsymbol{\theta}_k$, for $k \in \{1, ..., N_{trials}\}$;
- Obtain the objective function, $J(\boldsymbol{\theta}_k)$;
- Add to the set of hyperparameter combinations to study, $\mathcal{O}$;
- Divide the set in combinations worth, or not, studying, using $\Gamma_{TPE}(\cdot)$;
- Assign weights to past trials via $W_{TPE}(\boldsymbol{\theta}_k)$, in order for the kernel density estimation to take the importance of each observation into account;
- Obtain the bandwidth $B_{TPE}(\cdot)$, to control the smoothness of the kernel function around this trial;
- Apply the Bayesian theory to construct two probability models for promising and non-promising candidates, $P(\boldsymbol{\theta}|\mathcal{O}^{(0)})$ and $P(\boldsymbol{\theta}|\mathcal{O}^{(1)})$;
- Sample, and obtain the most likely best step, $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}^* \in \mathit{argmax}_{\boldsymbol{\theta} \in \mathcal{S}} r(\boldsymbol{\theta}|\mathcal{O})$;
- Repeat the last seven steps, until N_{trials} is reached.

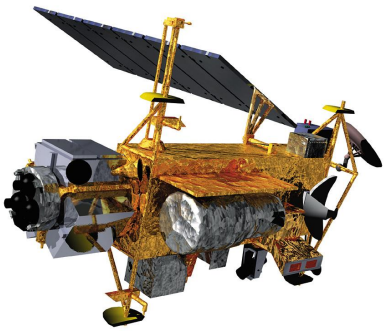
Chapter 4

Model Creation

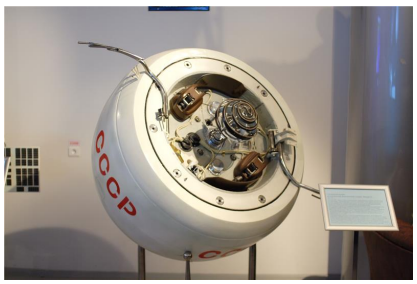
4.1 Data Generation

Currently, not many provided datasets have reentry data available, being, consequentially, necessary to obtain the data pool through simulation. In order to train and test the future LSTM model, a C program is to be designed.

Two cases are to be analyzed. The first, inspired in the UARS, involves the most large sized space debris estimation made by NASA [22], while the second is inspired in recent events from Kosmos 482. A general image of each vehicle is presented in Figure 4.1 and their general characteristics are described on Table 4.1.



(a) Body 1 - UARS (Credits: NASA [22])



(b) Body 2 - KOSMOS 482 (Venera-4 model) (Credits: ESA [34])

Figure 4.1: Example vehicles

Mass [kg]	160
C_D [-]	0.5
S_{wet} [m^2]	2.5

(a) Body 1 - UARS debris (SSPP Structure)

Mass [kg]	495
C_D [-]	2.2
S_{wet} [m^2]	4.3

(b) Body 2 - KOSMOS 482

Table 4.1: Characteristics of the vehicles

The initial condition orbits are defined according to Table 4.2, with fixed semi-major axis (a), eccentricity (e), inclination (i), longitude of ascending node (Ω) and argument of periapsis (ω). Multiple values of θ are adopted to generate sufficient data for the training. Since the altitude present in this initial conditions is above 86 kilometers, the *USA1976* model should also be taken into account.

The solution is generated through the discretization of the continuous state-space model

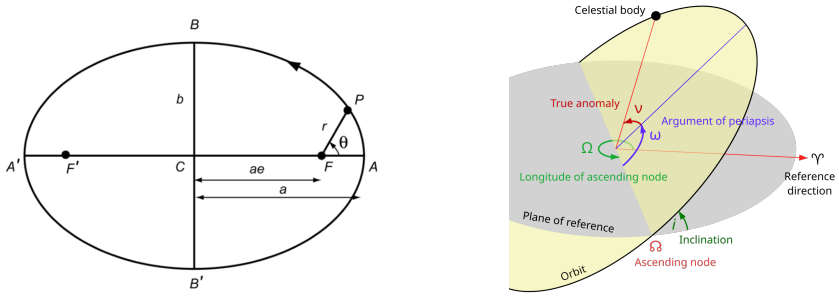


Figure 4.2: General specifications about orbits (Credits: Wakker [26], Wikipedia [42])

a [m]	$6.511 \cdot 10^6$
e [-]	$3.64 \cdot 10^{-4}$
θ [º]	$\{0, 1, ..., 359\}$
i [º]	56.93
Ω [º]	262.2
ω [º]	318.14

(a) Body 1 - UARS (Source: Heavens-Above [35])

a [m]	$6.521 \cdot 10^6$
e [-]	$3.54 \cdot 10^{-4}$
θ [º]	$\{0, 1, ..., 359\}$
i [º]	51.95
Ω [º]	241.9
ω [º]	103.7

(b) Body 2 - KOSMOS 482 (Source: SatCat [39])

Table 4.2: Orbit characteristics

presented in equations **(3.1)**, **(3.2)**, **(3.3)**, **(3.4)**, **(3.5)** and **(3.6)**, while applying Euler's Modified Semi-Implicit Method, shown in equation **(4.1)** [5]. The time-step adopted for the simulation process is of $\Delta t = 0.001$ seconds. It is important to note that, due to the reference frame adopted by the equations cited above, the computation is to be made entirely in geocentric spherical fixed coordinate system, with the exception of the initial condition conversion from Keplerian orbital parameters as well as the recording procedure, which is performed in geocentric Cartesian fixed coordinates. It is also important to refer the use of body referential system for the case of the Dryden turbulence model.

$$X_{k+1} = X_k + \frac{\Delta t}{2}\left(f\left(X_k\right) + f\left(X_k + \frac{\Delta t}{2}\cdot f\left(X_k\right) + \frac{\Delta t}{2}\cdot f\left(X_k + \frac{\Delta t}{2}\cdot f\left(X_k\right)\right)\right)\right) \tag{4.1}$$

After simulation, the values are saved in SI units inside a *.csv* format file. Its header can be verified in Table 4.3. It should be noted that the time-step recorded is to be different, $\Delta t_{\text{csv}} = 0.1s$.

time	X	Y	Z
0.0	0	0	0
...	...	...	...

Table 4.3: *.csv* File format

The evolution in altitude can be consulted in Annex B. After the simulation is done, noise needs to be introduced, following equations **(3.21)**, **(3.22)** and **(3.23)**, with a the fol-

lowing noise input $\omega(t) \sim \mathcal{N}(0, 100)$. Since the values outputted are velocities, a simple integration process should be made to obtain the position differences. The noisy values are stored in a different file, which will serve as the input for the model.

The resume of the process can be consulted in Figure 4.3.

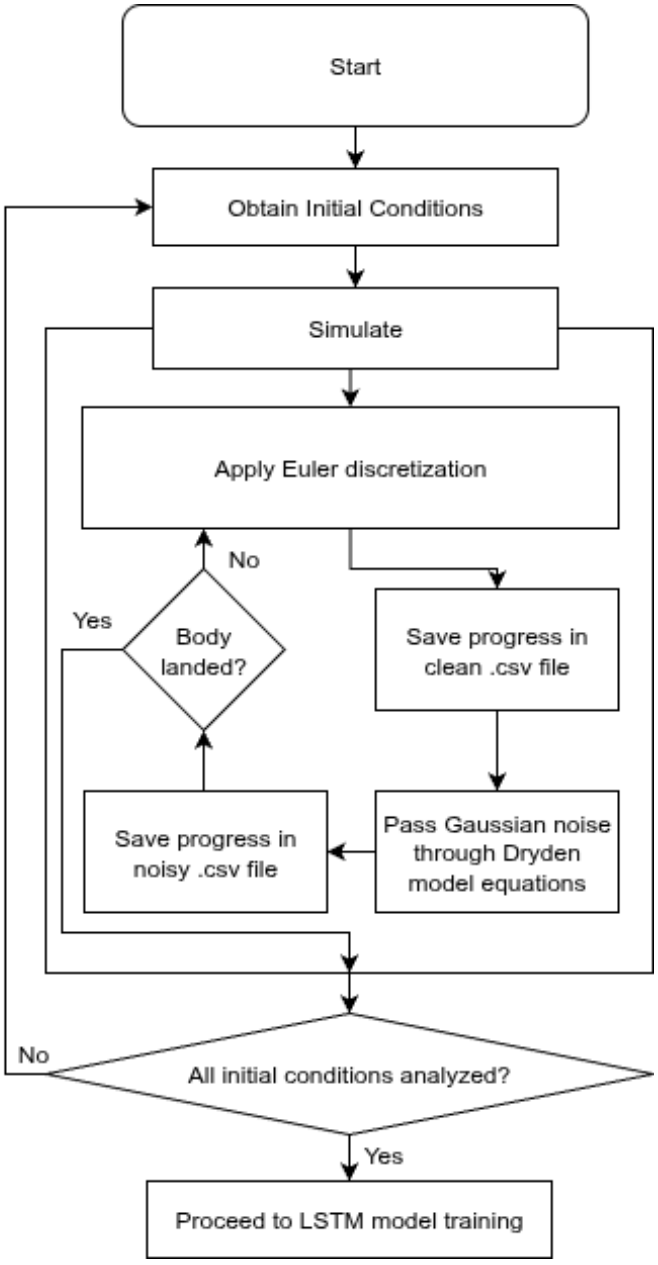


Figure 4.3: Flowchart of the simulation program

4.2 LSTM Training and Testing

From the simulation, a total of 1462899 and 1642932 state vectors were produced for the respective UARS and KOSMOS 482 vehicles.

In order to create a robust model, the Kullback-Leibler diversion between the error in both training and testing sets should be one of the main concerns. We firstly define the error functional, $\mathbf{e}_{Set}(\cdot) \in \mathbb{R}^3$, as the absolute difference between the real ($\mathbf{X}$) and predicted value ($\hat{\mathbf{X}}$), as stated in equation **(4.2)**.

$$\mathbf{e}_{Set}\left(\mathbf{X}_k\right) = \|\mathbf{X} - \hat{\mathbf{X}}\| \quad (4.2)$$

Since we are using a MISO model system, we need to treat each error per dimension, as shown in equation **(4.3)**.

$$\mathbf{e}_{Set}\left(\mathbf{X}_k\right) = \begin{bmatrix} e_{x_{Set}}\left(\mathbf{X}_k\right) \\ e_{y_{Set}}\left(\mathbf{X}_k\right) \\ e_{z_{Set}}\left(\mathbf{X}_k\right) \end{bmatrix} = \begin{bmatrix} \|x - \hat{x}\| \\ \|y - \hat{y}\| \\ \|z - \hat{z}\| \end{bmatrix} \quad (4.3)$$

For each dimension, the values of the error in each set are to be recorded, to create a probability-density function of them using a Gaussian kernel density estimation, as stated by equation **(2.19)**.

Finally, the sets need to be submitted to the Kullback-Leibler test, as stated in equation **(2.36)**.

It is important to define how the training, test and validation sets were divided. Following the recommendations provided by Goodfellow [10], the final decision is presented in Table 4.4.

Set [%]	Phase 1	Phase 2
Training	64	80
Validation	16	-
Test	20	20

Table 4.4: Training-validation-test set division

Before the first phase took place, some adjustments need to be made. Firstly, it was decided that all the input should be noisy data, as stated in the previous section, while all output would be clean, following the considerations made in Chapter 2.3.1.2. Early stopping is also considered, being activated if the loss function does not improve after 50 epochs. It is also important to comment on the decision to discard weight decay regularization methods. As stated by Goodfellow [10], the early stop effect can sometimes be compared with these effects, while also not directly impacting the training speed and efficiency.

Proceeding to the optimization, the hyperparameter tuning was made using *Optuna* [2] framework, adopting the tree-structured Parzen[28] estimator approach due to the large number of parameters to tune. After fifty attempts, the following results were reached for

all the three models:

Layers	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
Input	9	256	2	1	1 (Dense)
Optimizer Algorithm	ADAM				
Loss Function	MAE				
Batch Size	32				

Table 4.5: Final LSTM architecture

After the main architecture is defined, it is imperative that the loss ratio is accurately studied. Again, a TPE approach is made, but this time just for one hyperparameter. Using the Kullback-Leibler divergence between the training and validation set, defining 2000 epochs as the maximum, enabling early stopping and restoring the best validation loss, fifteen trials are then performed. The resultant loss ratios can be observed for each model, in Table 4.6.

The first phase is conducted through up to 2000 epochs. The previously mentioned optimization and regularization parameters were applied. During the training, the evolution of the MAE loss function, MAPE, MSE and RMSE values can be consulted in Figures 4.4 to 4.27, which also include the second phase contents.

It is important to comment that, during phase 2, it was verified that the second method performed better than the first on both cases. For this reason, their error probability-density functions can be consulted in Figures 4.28 to 4.39.

The Kullback-Leibler tests are registered on Table 4.7, obtaining a geometrical norm of 6.07 and 0.492 for the respective cases of UARS and KOSMOS 482.

The discrepancy between these two cases should be a matter of further commentary. One could argue that the main cause for this difference is related with the nature of the division process of the data set into training and testing pools, which should probably be the most concrete cause. In order to address this subject, another training was made. The results expressed by these two saw a decrease in the KL test in the UARS case by approximately 38% to 3.74, while the KOSMOS 482 case experienced a relatively massive increase to 4.74. The detailed values for each model can be consulted in the Appendix C. More comments on this matter can be found in the next chapter.

Loss Ratio	Model x	Model y	Model z
UARS Debris (SSPP Structure)	$1.607 \cdot 10^{-5}$	$1.006 \cdot 10^{-5}$	$1.005 \cdot 10^{-5}$
KOSMOS 482	$9.27 \cdot 10^{-5}$	$9.27 \cdot 10^{-5}$	$1.006 \cdot 10^{-5}$

Table 4.6: Loss ratio for each model

Training-Test	Model x	Model y	Model z
UARS Debris (SSPP Structure)	$7.44 \cdot 10^{-3}$	4.78	3.75
KOSMOS 482	$1.76 \cdot 10^{-3}$	$1.27 \cdot 10^{-1}$	$4.76 \cdot 10^{-1}$

Table 4.7: Kullback-Leibler test values

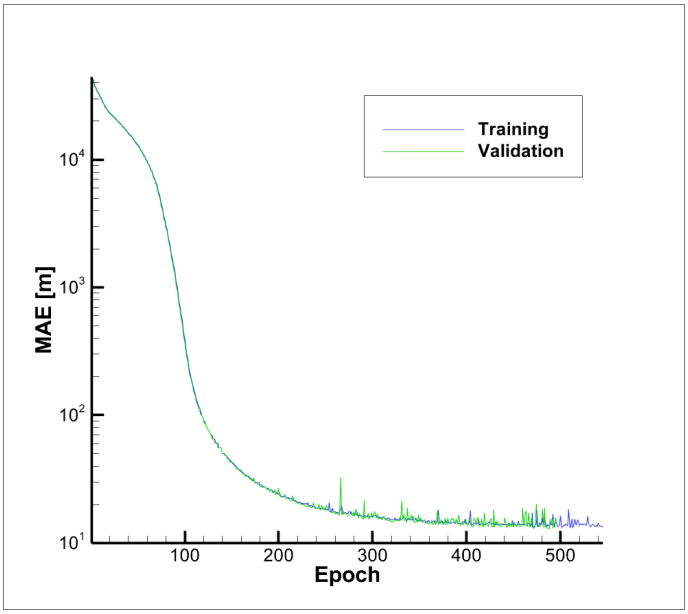


Figure 4.4: Mean absolute error x – Body 1 (UARS)

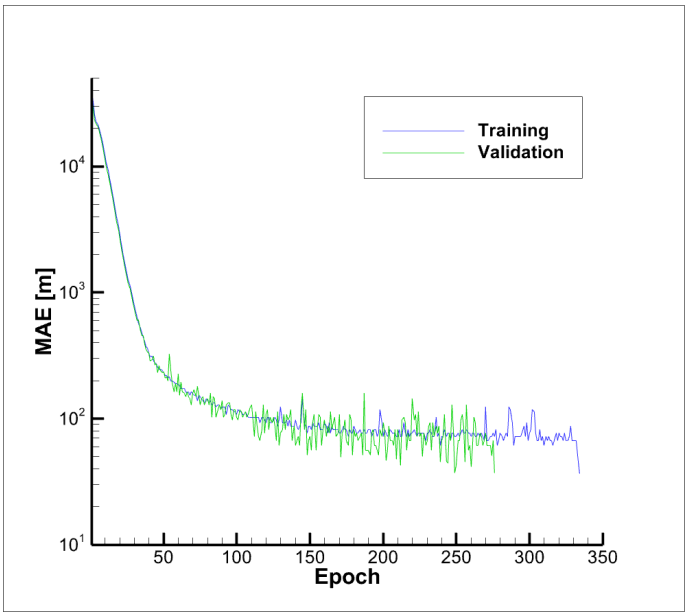


Figure 4.5: Mean absolute error x – Body 2 (KOSMOS 482)

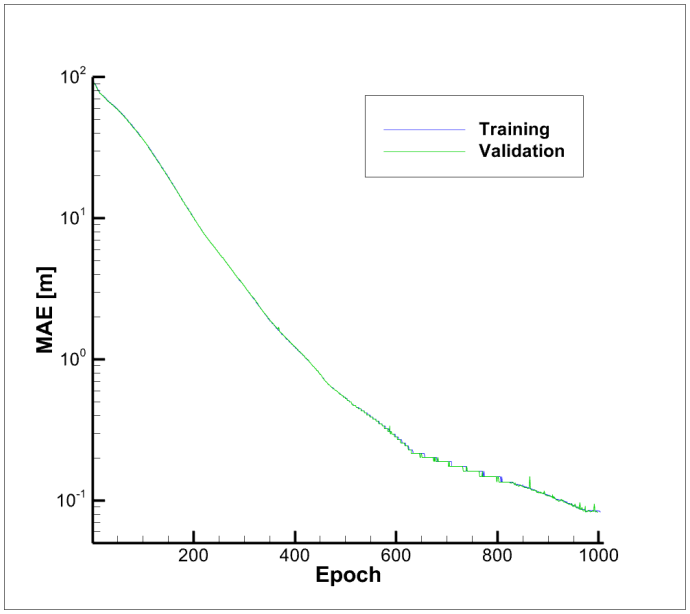


Figure 4.6: Mean absolute error y – Body 1 (UARS)

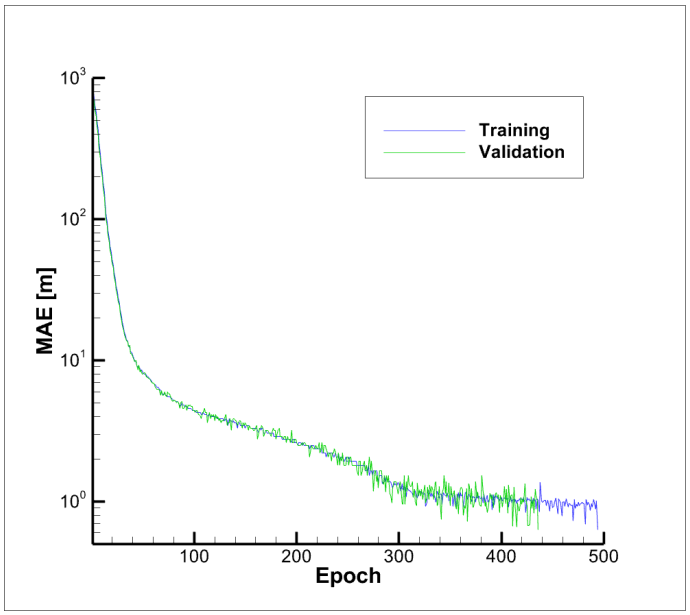


Figure 4.7: Mean absolute error y – Body 2 (KOSMOS 482)

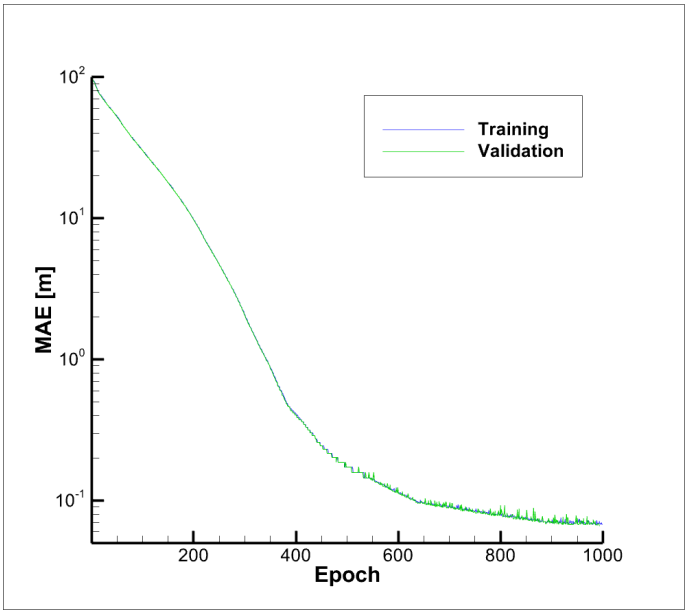


Figure 4.8: Mean absolute error z – Body 1 (UARS)

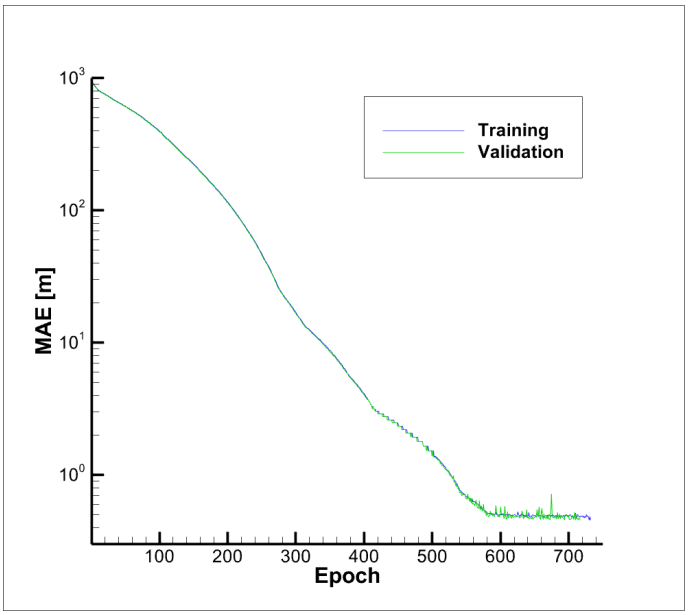


Figure 4.9: Mean absolute error z – Body 2 (KOSMOS 482)

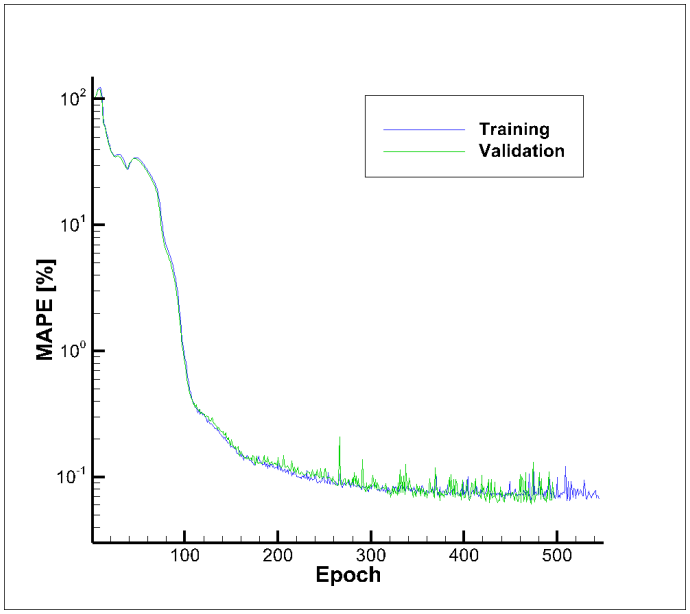


Figure 4.10: Mean absolute percentage error x – Body 1 (UARS)

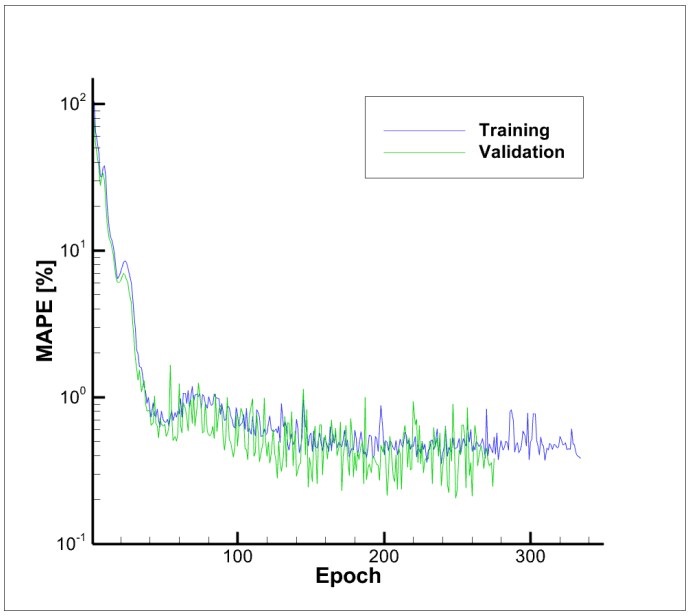


Figure 4.11: Mean absolute percentage error x – Body 2 (KOSMOS 482)

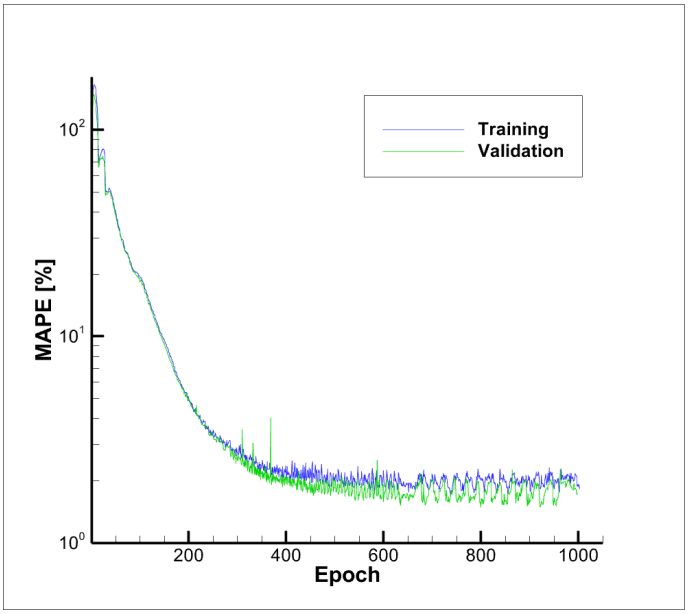


Figure 4.12: Mean absolute percentage error y – Body 1 (UARS)

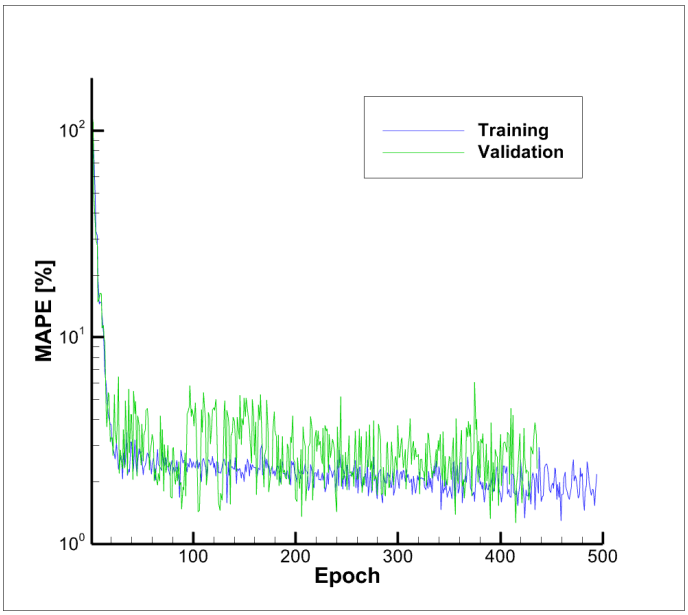


Figure 4.13: Mean absolute percentage error y – Body 2 (KOSMOS 482)

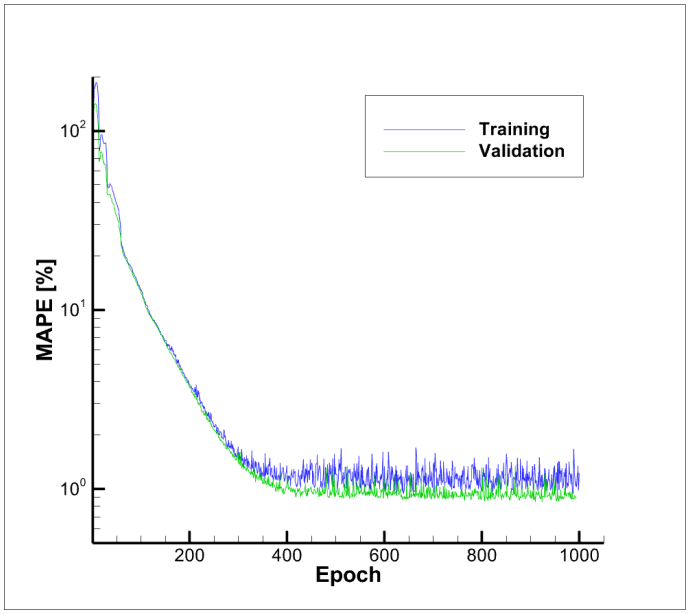


Figure 4.14: Mean absolute percentage error z – Body 1 (UARS)

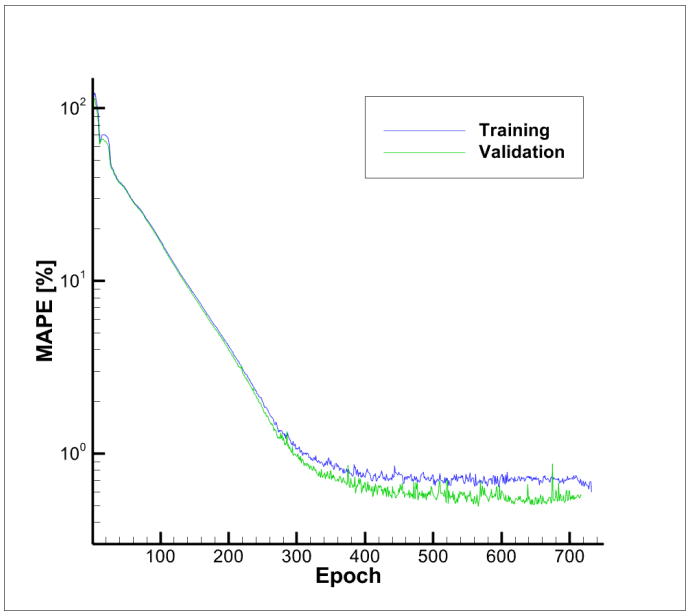


Figure 4.15: Mean absolute percentage error z – Body 2 (KOSMOS 482)

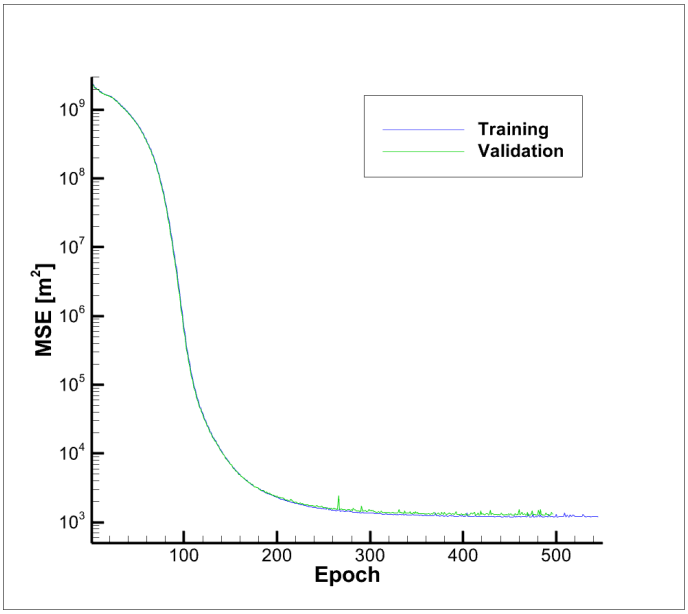


Figure 4.16: Mean squared error x – Body 1 (UARS)

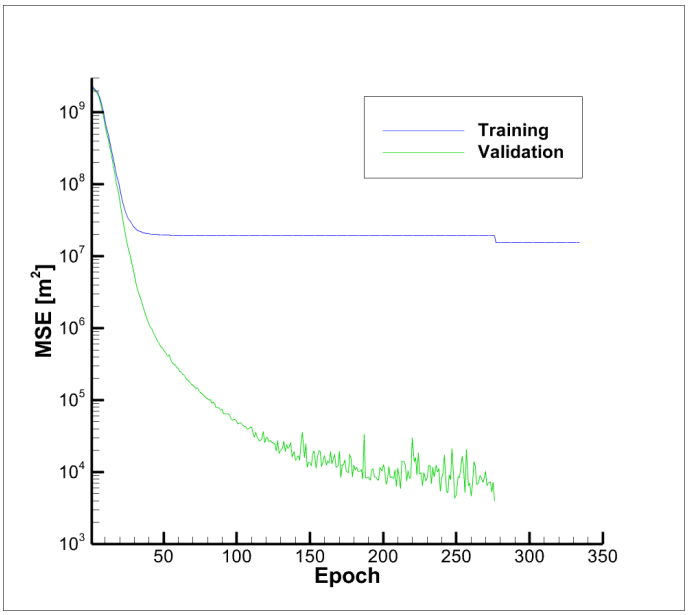


Figure 4.17: Mean squared error x – Body 2 (KOSMOS 482)

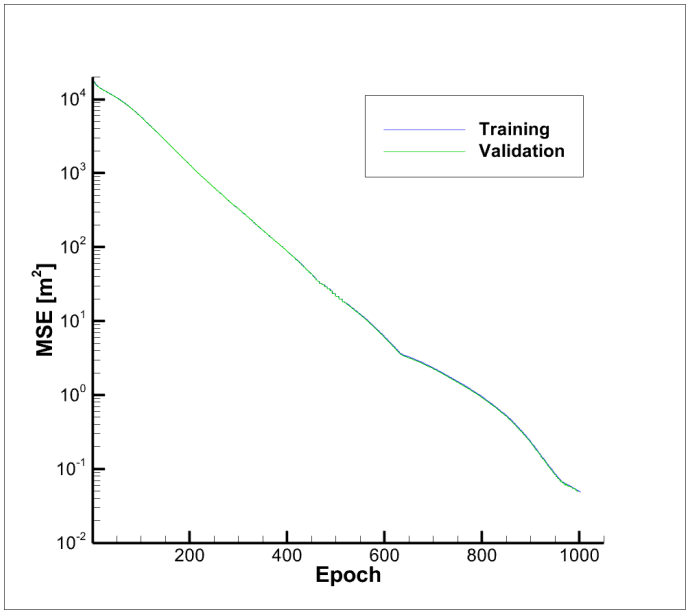


Figure 4.18: Mean squared error y – Body 1 (UARS)

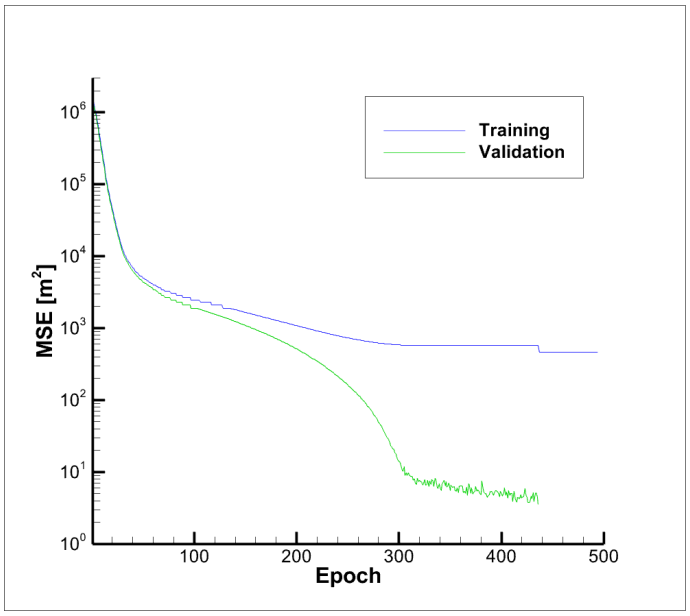


Figure 4.19: Mean squared error y – Body 2 (KOSMOS 482)

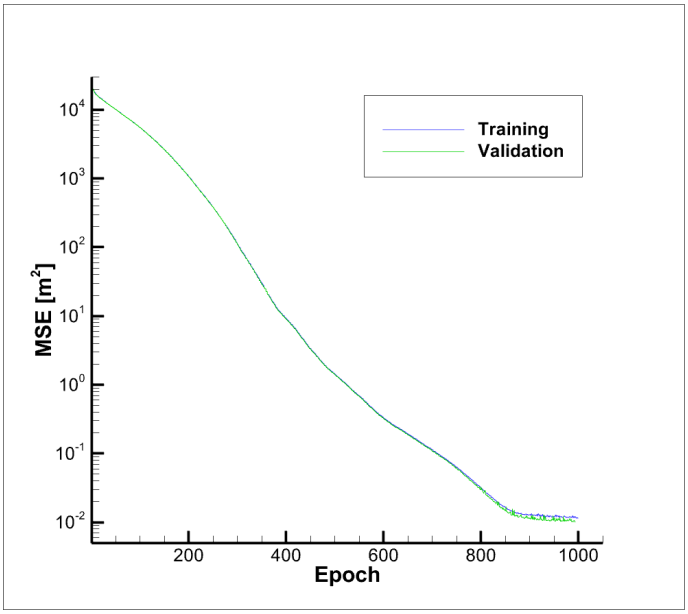


Figure 4.20: Mean squared error z – Body 1 (UARS)

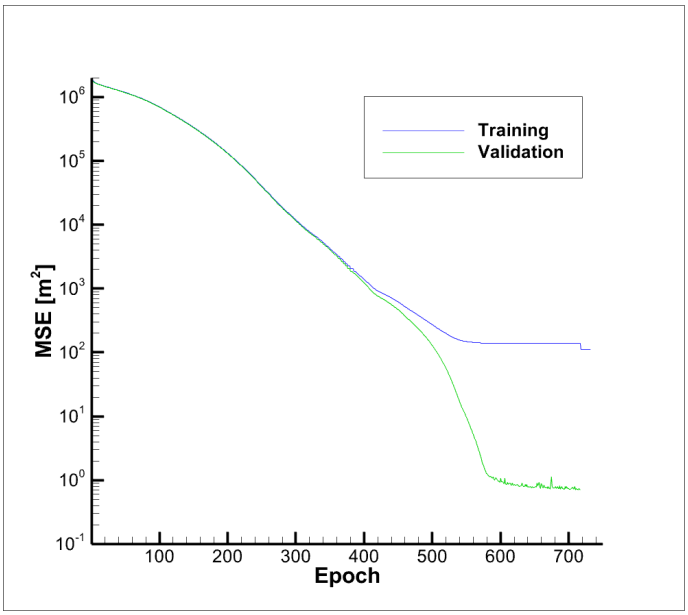


Figure 4.21: Mean squared error z – Body 2 (KOSMOS 482)

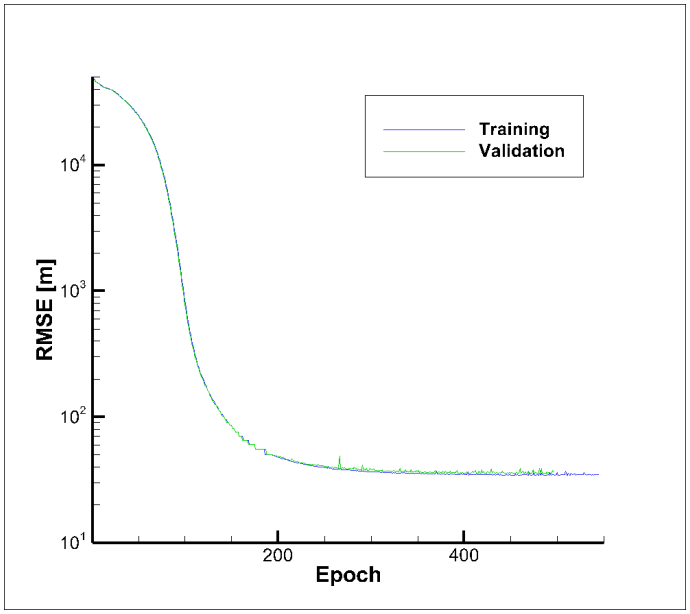


Figure 4.22: Root mean squared error x – Body 1 (UARS)

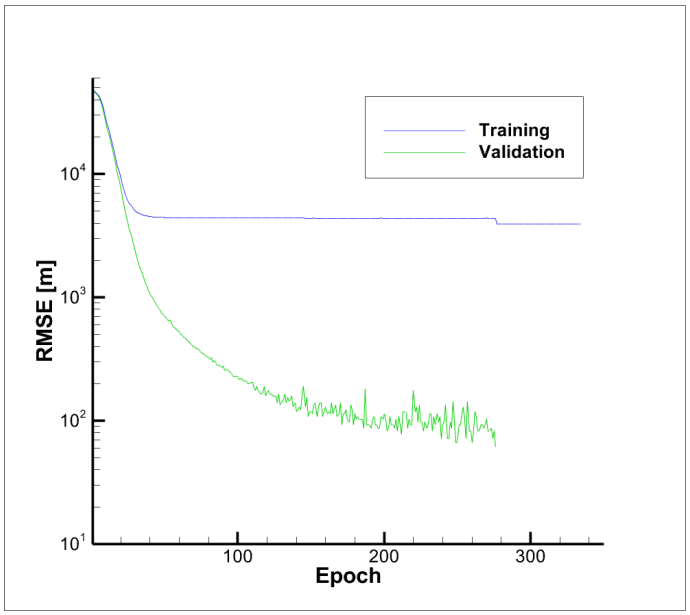


Figure 4.23: Root mean squared error x – Body 2 (KOSMOS 482)

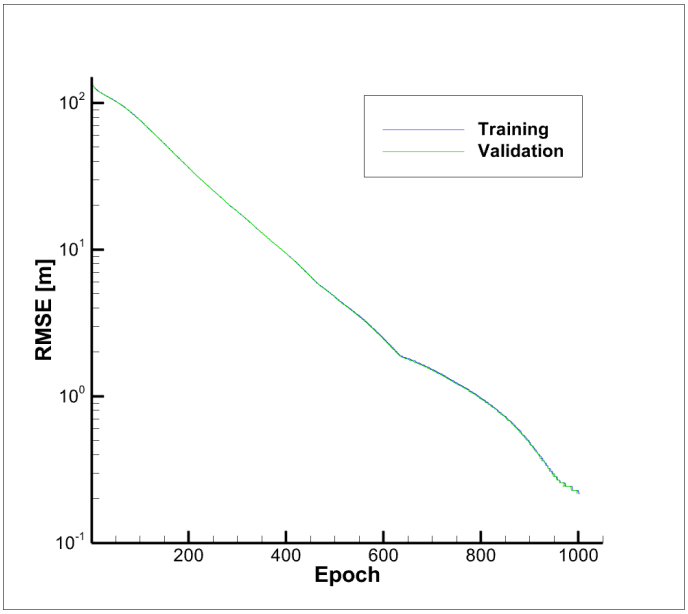


Figure 4.24: Root mean squared error y – Body 1 (UARS)

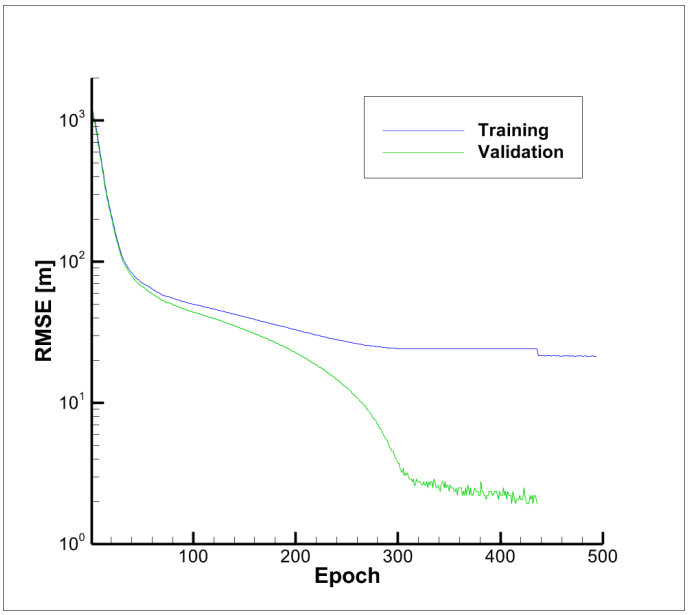


Figure 4.25: Root mean squared error y – Body 2 (KOSMOS 482)

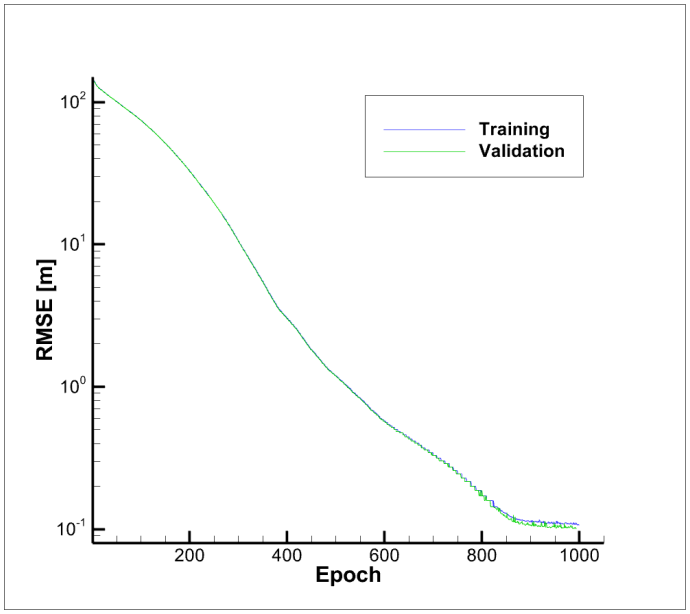


Figure 4.26: Root mean squared error z – Body 1 (UARS)

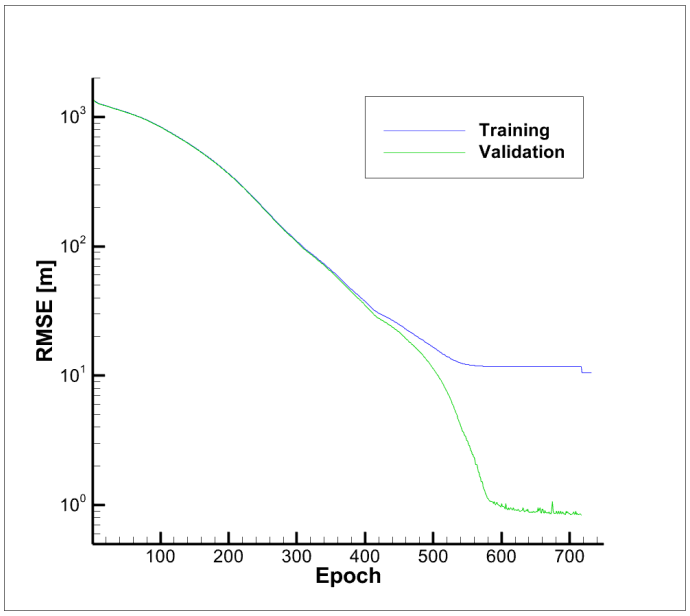


Figure 4.27: Root mean squared error z – Body 2 (KOSMOS 482)

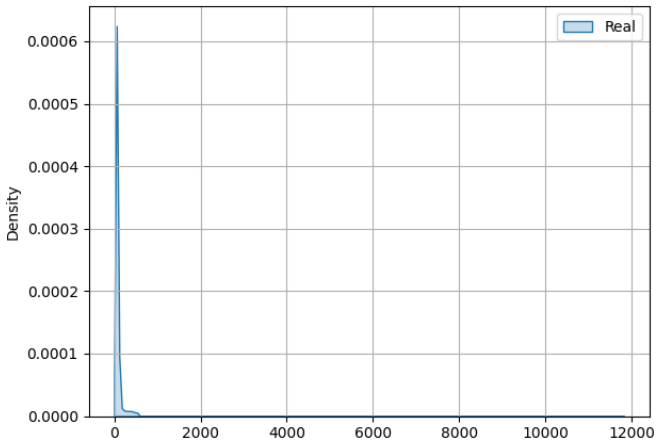


Figure 4.28: Error function (x) – Training (UARS)

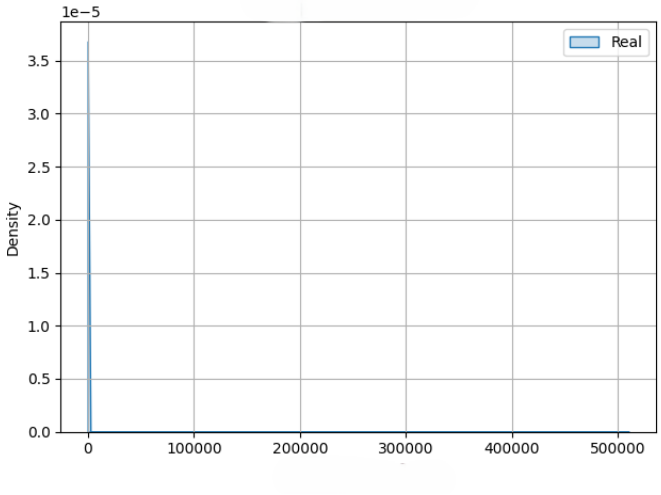


Figure 4.29: Error function (x) – Test (UARS)

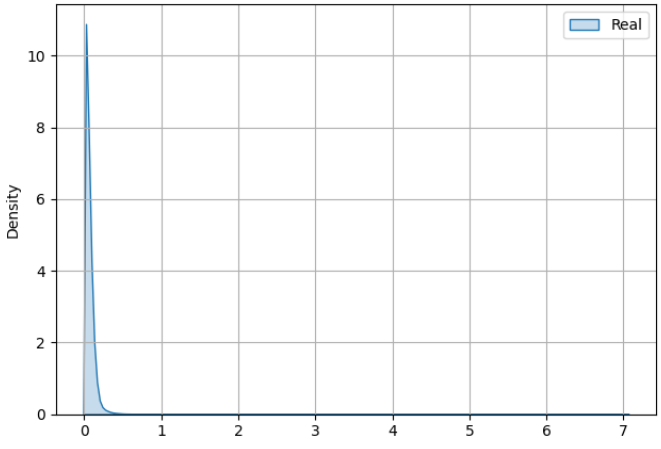


Figure 4.30: Error function (y) – Training (UARS)

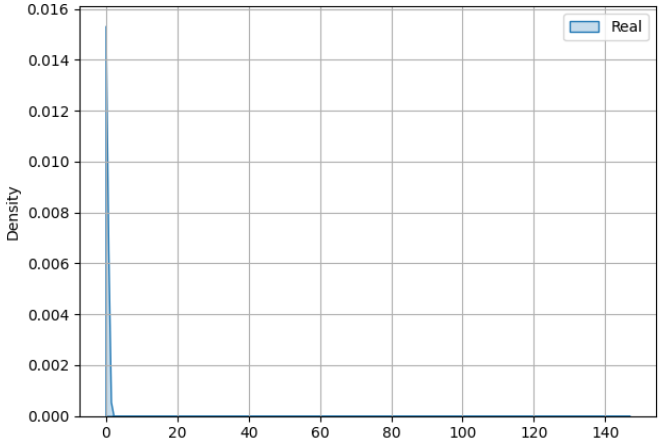


Figure 4.31: Error function (y) – Test (UARS)

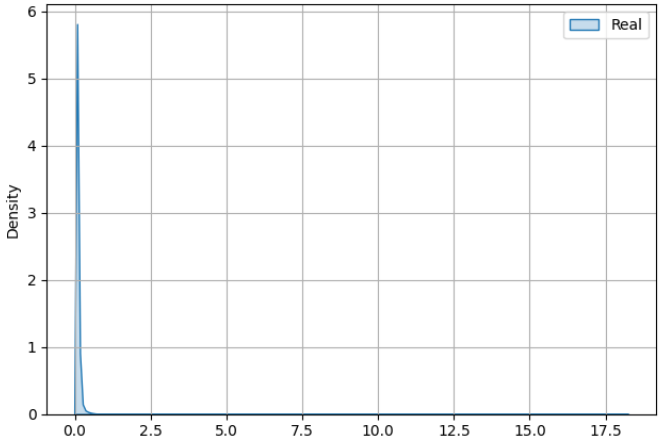


Figure 4.32: Error function (z) – Training (UARS)

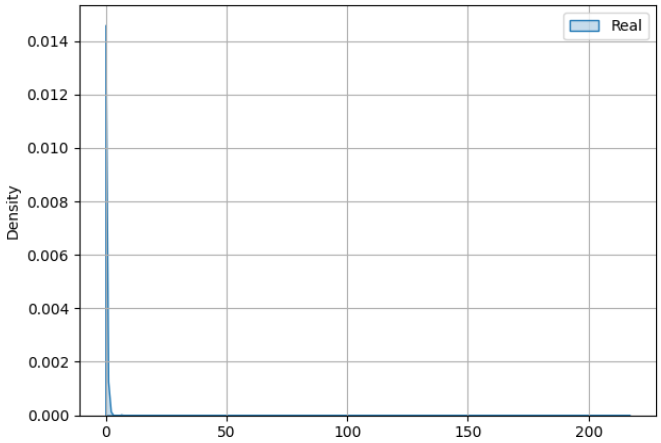


Figure 4.33: Error function (z) – Test (UARS)

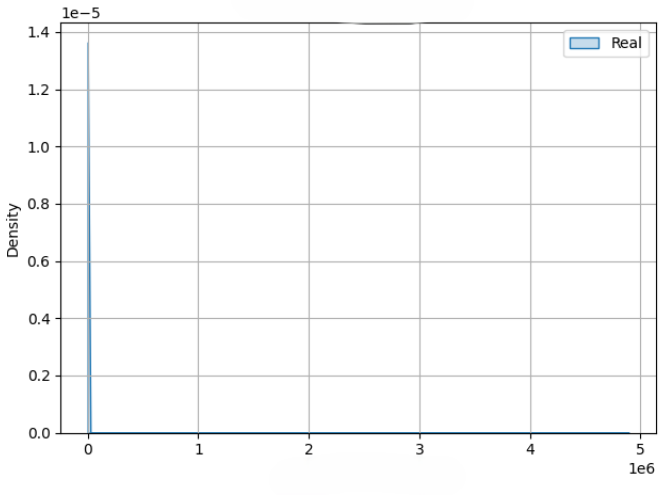


Figure 4.34: Error function (x) – Training (KOSMOS 482)

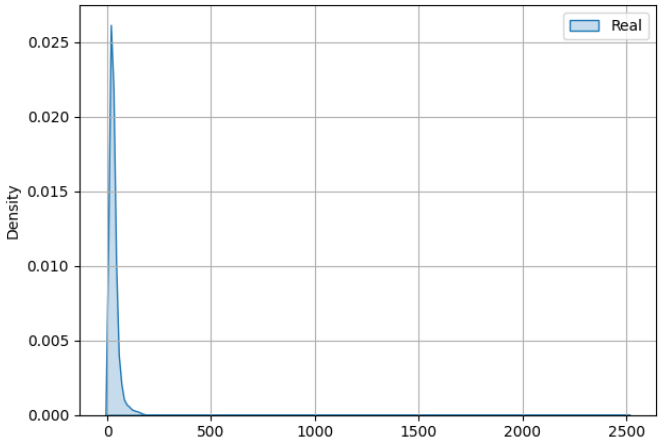


Figure 4.35: Error function (x) – Test (KOSMOS 482)

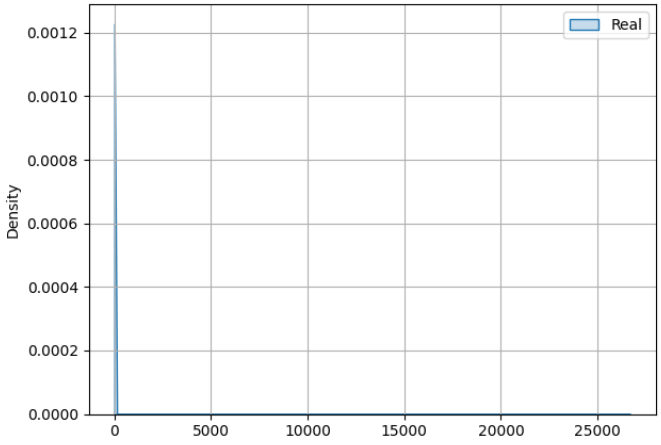


Figure 4.36: Error function (y) – Training (KOSMOS 482)

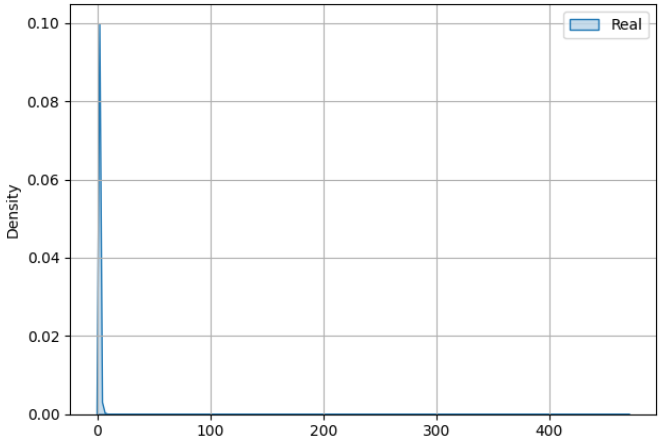


Figure 4.37: Error function (y) – Test (KOSMOS 482)

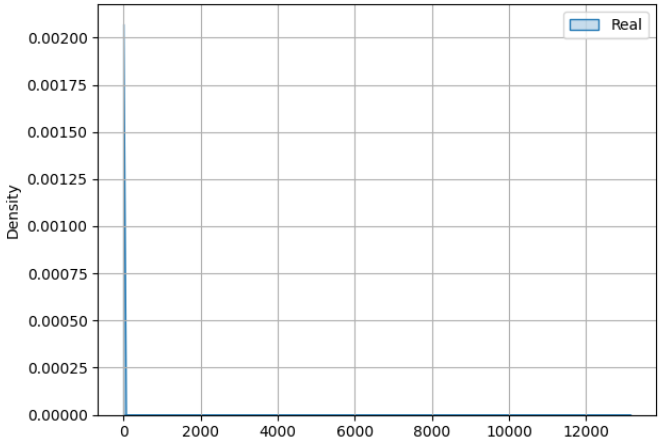


Figure 4.38: Error function (z) – Training (KOSMOS 482)

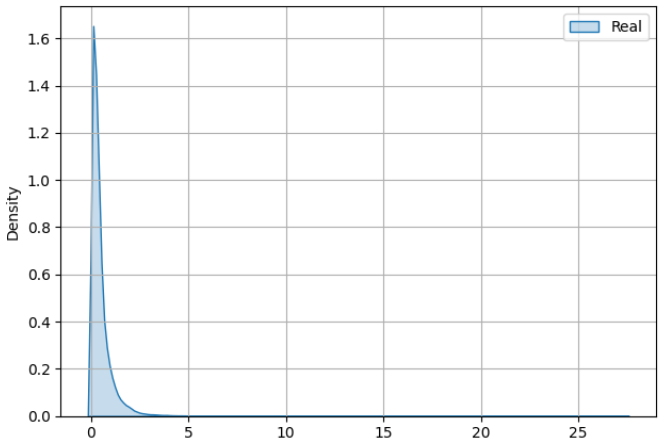


Figure 4.39: Error function (z) – Test (KOSMOS 482)

Chapter 5

Conclusions and Further Research

The objective of this thesis was to develop a framework to predict the reentry behavior of an uncontrolled object based on a small number of earlier states, offering robustness to noise parameters such as turbulence while also implementing novel tools from the area of machine learning to do so.

An LSTM network was developed to estimate the reentry trajectory of an aerospace vehicle, predicting the next state based on the current and two older states with a time-step of 0.1 seconds each.

Due to the sporadic existence of reentry data, the training pool had to be simulated. It was decided to adopt an atmospheric model based on *ICAO*, adopting the *USA1976* model from the termosphere onwards. Constraints were also applied, as well as a noise model. The Dryden turbulence model was the chosen one, were a Gaussian noise $\omega \sim \mathcal{N}(0, 100)$ was declared as the input.

It was decided to implement three MISO LSTM models due to the curse of dimensionality, as stated in Chapter 2.3.3.4, and the hyperparameter tuning had the number of hidden layers and their inputs, the optimizer and loss functions, the loss ratio and the batch size into account. The optimization algorithm chosen was the Tree-Structured Parzen Estimator with 50 trials, to select the combination with the best KL test result. To refine the loss ratio once more, another TPE was adopted, with fifteen trials for each model. The final model consisted of three hidden layers, with 256, 2 and 1 input respectively, an ADAM optimizer algorithm, a mean absolute error loss function and a batch size of 32. The loss ratio was in the order of 10^{-5}, varying per model.

The training set contained 80% of the entire data pool, and 20% of it was saved for testing. The training process was divided in two steps. Firstly, 20% of the training pool was defined for validation purposes, totaling 64% for training, 16% for validation and 20% for testing. In the second phase, the validation and training sets were joined into one, and two methods were applied, choosing the one with better KL results. The first consisted in discarding the whole training process and repeat it until the epoch number recorded in the first phase was reached. The second one implied the recording of the validation loss result, and continue training with the joined training set until that value was reached. In every model, the second method was the best-performing one. The training process was of up to 2000 epochs and included regularization parameters such as early stopping in the loss function of the validation set, in the case of not registering any results for 50

consecutive epochs.

For the case of KOSMOS 482, the training process was complete in 276 epochs in the first phase and 58 epochs in the second one for x model, 436 and 58 for the y model, finishing with 717 and 15 for z. For the UARS satellite, it took 496 epochs for the first phase and 51 for the second in the x model, 998 epochs and 5 for the y model, concluding with 994 and 5 epochs for the z model.

Focusing on the mean absolute error evolution through epochs, from Figure 4.4 to Figure 4.9, we can notice a rapid convergence in the x model cases, followed by some variance in the final epochs, as primarly noticed in the KOSMOS 482, as shown in Figure 4.5. In the y case, we have an approximately steadier, slower rate of convergence in the UARS satellite, whereas, in KOSMOS 482 case, the first epochs create a rapid advancement in the convergence process, but decelerates in the last epochs and is accompanied by a high variance. Another variance phenomena can be noticed in the z model, in the KOSMOS 482 case. In general terms, the models get a high level of accuracy in the prediction process, up to the order of tens of meters, indicating a good training process, where the y and z models have a fair advantage, possibly due to the lower magnitude of the data.

In the case of the mean absolute percentage error from Figure 4.10 to Figure 4.15, the noise in the latter epochs is widely superior in the KOSMOS 482 case, especially in the y model, as noticed in Figure 4.13. The z model is the only exception to this, where the UARS has a more unstable convergence process, as it can be seen in Figure 4.14. These variance phenomena can be explained by the attempt of the model to learn the noise behavior. Other possibilities should not be discarded, such as the decision by both *Optuna* or *Tensorflow* solver to force the SGD as the optimizer or maybe some overfitting effects. Here, the data proves the aforementioned difference in data magnitude of x compared with y and z, since the first has a lower percentage error compared with the latter ones, revealing that the training process had more relative accuracy.

Commenting on the mean squared errors and its roots, from Figure 4.16 to Figure 4.27, we can notice that there exist may exist more outliers, or a few strong ones, in the KOSMOS 482 satellite models, as it can be seen in Figures 4.23, 4.25 and 4.27. The highest value recorded is, by far, on the model from the first referenced figure. Another phenomena can be noticed, where the training values are largely superior to the validation ones. This can be explained by the law of the large numbers and its counterpart. Consider that, in a population, four samples are obtained, where the first two are large, A1 and A2, and the latter ones small, B1 and B2. If we would compare their averages, between A1 and A2, and repeat the same process for B1 and B2, we would notice that the value of the latter case would be bigger, as their separate means are farther from the global population average. On the other hand, the difference in the bigger sample means would be smaller, as they are closer to the global average.

In conclusion, the Kullback-Leibler divergence results show a good correspondence be-

tween the training errors and the testing ones, implying that the learning process was well made. Analyzing the p.d.f. distribution, a somewhat prolonged tail can be seen in almost every case. This is explained due to the noisy nature of the inputs, providing reliability in a set of harsh environments. By comparing the two vehicles, we can check that the first case (KOSMOS 482) was clearly the best-performing one. Moreover, the error probability-density functions show that the test results have actually an higher accuracy that the training ones, by verifying that their expected values are nearer zero as well as their variance being inferior than the training one. In the UARS case, even though that the learning process has shown better general training and validation values at first, this trend was compromised when comparing the final error p.d.f.'s. This may be due to different vehicle characteristics, or starting orbit. For this reason, it is important to refer that every case is unique, an thus deserves more attention to detail in developing an LSTM framework, stating that what may have worked for one case, may not be entirely fit for the other, and vice-versa.

In the future, should a similar study be conducted, an adoption for more realistic data should be taken into account. Variables such as pressure, temperature, atmospheric atomic composition, and position are to be the number one priority to have in consideration. If these data are not available, efforts should be taken to conduct a more accurate modeling of the upper atmosphere, having in consideration specific characteristics for the specified case *e.g.* the constitution of chemical components in the upper atmosphere and the current maximum temperature, in order to conduct a more robust simulation. The impact of attitude in the model accuracy should also be a motive of study, where the wet area could be chosen an a method to simulate this parameter. An attempt to concatenate this and earlier studies stated in Chapter 1 should also be taken into account, in order to produce a single accurate reentry model starting from, at least, 200 kilometers. The variance in the Kullback–Leibler divergence results, as stated in the Chapter 4, merits dedicated investigation, wherein the deployment of a Monte Carlo method is suggested as a primary avenue for addressing this issue. Another method of dividing the training and testing sets should also be tested, comprising on near 50% of the total dataset on both cases, in order to counter the effect of the law of the large numbers. This assumption comes from the Pareto principle, which states that roughly 80% of the deviance comes from 20% of the data in question, resulting in the phenomena of divergence between the validation and training metrics as mostly noticed in Figures 4.17, 4.19 and 4.21.

Bibliography

[1] Abadi, M. , Agarwal, A. , Barham, P. , Brevdo, E. , Chen, Z. , Citro, C. , Corrado, G. S. , Davis, A. , Dean, J. , Devin, M. , Ghemawat, S. , Goodfellow, I. , Harp, A. , Irving, G. , Isard, M. , Jia, Y. , Jozefowicz, R. , Kaiser, L. , Kudlur, M. , Levenberg, J. , Mané, D. , Monga, R. , Moore, S. , Murray, D. , Olah, C. , Schuster, M. , Shlens, J. , Steiner, B. , Sutskever, I. , Talwar, K. , Tucker, P. , Vanhoucke, V. , Vasudevan, V. , Viégas, F. , Vinyals, O. , Warden, P. , Wattenberg, M. , Wicke, M. , Yu, Y. , Zheng, X. , "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org. 33

[2] Akiba, T. , Sano, S. , Yanase, T. , Ohta, T. , Koyama, M. , "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 48

[3] Anderson Jr., J. D. , *Introduction to Flight*. New York, NY: McGraw-Hill Education, 8th ed., 2016.

[4] Blatt, T. , "Anti-satellite weapons and the emerging space arms race," *Harvard International Review*, vol. 41, no. 3, pp. pp. 29–34, 2020.

[5] Cromer, A. H. , "Stable solutions using the euler approximation," *American Journal of Physics*, vol. 49, no. 5, pp. 455–459, 1981. 46

[6] DeGroot, M. H. Schervish, M. J. , *Probability and Statistics*. Boston: Pearson, 4th ed., 2012.

[7] Dozat, T. , "Incorporating nesterov momentum into adam," in *International Conference on Learning Representations (ICLR) Workshop*, 2016. 23

[8] Dunlop, S. , *ICAO standard atmosphere*. Oxford University Press, 2008. 37

[9] Fasano, G. Franceschini, A. , "A multidimensional version of the kolmogorov-smirnov test," *Monthly Notices of the Royal Astronomical Society*, vol. 225, pp. 155–170, 1987. 13

[10] Goodfellow, I. , Bengio, Y. , Courville, A. , *Deep Learning*. MIT Press, 2016. xvii, 18, 19, 27, 28, 29, 34, 48

[11] Jung, O. , Seong, J. , Jung, Y. , Bang, H. , "Recurrent neural network model to predict re-entry trajectories of uncontrolled space objects," *Advances in Space Research*, vol. 68, no. 6, pp. 2515–2529, 2021. 3

[12] Kullback, S. Leibler, R. A. , "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, March 1951. 14

[13] Lin, J. , "Divergence measures based on the shannon entropy," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 145–151, 1991. 14

[14] Loshchilov, I. Hutter, F. , “Decoupled weight decay regularization,” 2019. 24

[15] Madden, M. M. , “Verifying implementation of the dryden turbulence model and milf-8785 gust gradient,” in *2018 Modeling and Simulation Technologies Conference*, (Atlanta, GA), June 2018. 40, 75

[16] Minzner, R. A. , “The 1976 standard atmosphere above 86-km altitude: Recommendations of task group ii to coesa,” NASA Special Publication SP-398, National Aeronautics and Space Administration (NASA), Washington, D.C., 1976. Prepared by NASA Goddard Space Flight Center. 37

[17] Montenbruck, O. Gill, E. , *Satellite Orbits: Models, Methods and Applications*. Berlin, Heidelberg: Springer, 2000.

[18] Moreira, A. , “Optimal and robust control of atmospheric reentry trajectories,” Master’s Thesis on Aeronautical Engineering, Department of Aerospace Sciences, Faculty of Engineering, University of Beira Interior, 2022.

[19] Salmaso, F. , Trisolini, M. , Colombo, C. , “A machine learning and feature engineering approach for the prediction of the uncontrolled re-entry of space objects,” *Aerospace*, vol. 10, no. 3, 2023. 3

[20] Senra, F. , “Feedback neural network based orbital trajectory prediction,” Master’s Thesis on Aeronautical Engineering, Department of Aerospace Sciences, Faculty of Engineering, University of Beira Interior, 2023. 3, 33, 42

[21] Silverman, B. W. , *Density Estimation for Statistics and Data Analysis*, vol. 26 of *Monographs on Statistics and Applied Probability*. London: Chapman and Hall, 1986. 8

[22] Stansbery, E. Johnson, N. L. , “Nasa upper atmosphere research satellite (uars) reentry prediction and analysis,” in *International Symposium on Sustainable Space Development and Utilization for Humankind: Orbital Space Debris – Challenges & Opportunities*, (Shinagawa, Tokyo, Japan), NASA Johnson Space Center / Orbital Debris Program Office, Mar. 2012. Presentation; NASA Technical Report JSC-CN-25915; NTRS Document ID 20120003110. 45

[23] Swope, C. , Bingen, K. A. , Young, M. , LaFave, K. , “Space threat assessment 2025,” technical report, Center for Strategic and International Studies, Aerospace Security Project, Washington, DC, 04 2025. 3

[24] Tong, Y. L. , *The Multivariate Normal Distribution*. Springer Series in Statistics, New York: Springer, 1990.

[25] Vinh, N. X. , Busemann, A. , Culp, R. D. , *Hypersonic and Planetary Entry Flight Mechanics. University of Michigan Press, Ann Arbor. 1980. 357 pp. Illustrated.* The University of Michigan Press, 1 ed., 1980. 7

[26] Wakker, K. F. , *Fundamentals of Astrodynamics*. Institutional Repository, Delft University of Technology, 1 ed., 2015. xvii, 46

[27] Wang, J. , Wu, Y. , Liu, M. , Yang, M. , Liang, H. , "A real-time trajectory optimization method for hypersonic vehicles based on a deep neural network," *Aerospace*, vol. 9, no. 4, 2022. 35

[28] Watanabe, S. , "Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance," *arXiv preprint arXiv:2304.11127*, 2023. 43, 48

[29] Zadeh, A. Y. Shahbazy, M. , "A review into data science and its approaches in mechanical engineering," *CoRR*, vol. abs/2012.15358, 2020. xvii, 15

[30] Zhang, A. , Lipton, Z. C. , Li, M. , Smola, A. J. , *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`. xvii, 15, 30, 31

[31] Amazon Web Services, "Model fit: Underfitting vs. overfitting." `https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html`. Visited on 2025-09-13. xvii, 16

[32] European Space Agency, Space Debris Office, "Esa's annual space environment report," Technical Report GEN-DB-LOG-00288-OPS-SD, Issue 9.0, European Space Agency, ESOC, Darmstadt, Germany, 2025. ESA Unclassified — Releasable to the Public. xvii, 2, 3

[33] European Space Agency, "Esa's re-entry predictions: Cluster re-entry." `https://reentry.esoc.esa.int/home/blog/cluster-reentry`, 2025. Visited on 2025-09-30. 7

[34] European Space Agency, "Re-entry prediction soviet-era venera venus lander (cosmos-482 descent craft)," May 2025. Blog post, updated 13 May 2025. 45

[35] Heavens-Above (DLR/GSOC), "Uars - orbit (21701)." `https://www.heavens-above.com/orbit.aspx?satid=21701`, 2025. Visited on 2025-09-30. 46

[36] IBM, "What is a neural network?," October 2021. Visited on 2025-07-25. xvii, 29

[37] IBM, "What is machine learning (ml)?." `https://www.ibm.com/think/topics/machine-learning`, 2021. Visited on 2025-08-26. 15

[38] International Standard Organization, "Iso 2533:1975," *International Standard Organization*, 1975. 37

[39] SatCat, "Kosmos 482 - orbit (6073)." `https://www.satcat.com/sats/6073`, 2025. Visited on 2025-09-30. 46

[40] The MathWorks, Inc., "Dryden wind turbulence model," 2025.

[41] United States Department of Defense, “Flying qualities of piloted aircraft (mil std 1797a),” tech. rep., U.S. DoD, 1990. 39

[42] Wikipedia contributors, “Orbital elements.” https://en.wikipedia.org/wiki/Orbital_elements, 2023. Visited on 2023-05-01. xvii, 46

[43] World Economic Forum, “Space: The $1.8 trillion opportunity for global economic growth,” tech. rep., World Economic Forum, Geneva, Switzerland, April 2024. Insight Report, in knowledge partnership with McKinsey & Company. xvii, 1

Appendix A

Dryden Inverse Z-Transform

In order to compute the inverse Z-transform of the Dryden equations **(3.21)**, **(3.22)** and **(3.23)**, a zero-order hold (ZOH) representation was needed, as stated by Madden [15]. This process is presented in equation **(A.1)**. Here, x represents the input of each controller, which is the white noise, while y is the output.

$$\frac{as+b}{cs+d} \longrightarrow y_k = C_1 y_{k-1} + C_2 x_k + C_3 x_{k-1} \qquad \begin{cases} C_1 = \exp\left(-\frac{d\Delta t}{c}\right) \\ C_2 = \dfrac{a}{c} \\ C_3 = \dfrac{b}{d}\left[1 - \exp\left(-\frac{d\Delta t}{c}\right)\right] - \dfrac{a}{c} \end{cases} \tag{A.1}$$

For the u case, we can represent the constants according with equation **(A.2)**, with the converted variables. Please note that the representation of x_k is absent due to a being a null value.

$$\begin{cases} a = 0 \\ b = \sigma_u \sqrt{\dfrac{\pi}{\Delta t}} \sqrt{\dfrac{2V}{\pi L_u}} \\ c = 1 \\ d = \dfrac{V}{L_u} \\ x_{k-1} = \omega\left(t_{k-1}\right) \\ y_{k-1} = u_{Noise}\left(t_{k-1}\right) \end{cases} \tag{A.2}$$

A visual representation of the controller is shown in Figure A.1.

Moving on to the case of v, since there are two fractions, we need to use two sequential controllers. Their inputs and constants are present in equation **(A.3)**. Here, we can quickly get to the assumption that the input of the second controller is the input of the first one. The diagram can be consulted in Figure A.2.

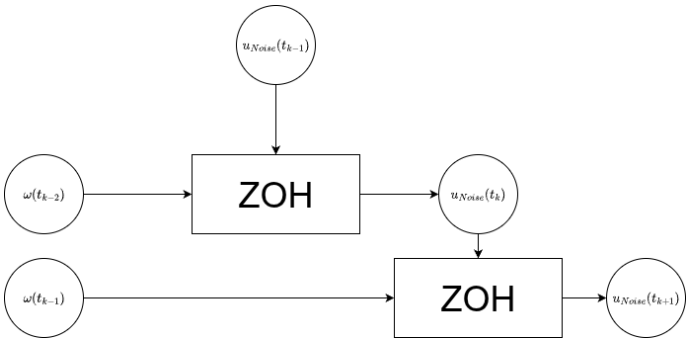


Figure A.1: Dryden noise generation process (u_{Noise})

$$\begin{cases} a_1 = 0 \\ b_1 = \sigma_v \sqrt{\dfrac{\pi}{\Delta t}} \sqrt{\dfrac{3V}{\pi L_v}} \\ c_1 = 1 \\ d_1 = \dfrac{V}{L_v} \\ x_k^{(1)} = \omega\left(t_k\right) \\ x_{k-1}^{(1)} = \omega\left(t_{k-1}\right) \end{cases} \qquad \begin{cases} a_2 = 1 \\ b_2 = \dfrac{V}{\sqrt{3} L_v} \\ c_2 = 1 \\ d_2 = \dfrac{V}{L_v} \\ x_k^{(2)} = y_k^{(1)} \\ x_{k-1}^{(2)} = y_{k-1}^{(1)} \\ y_{k-1}^{(2)} = v_{Noise}\left(t_{k-1}\right) \end{cases} \tag{A.3}$$

A visual representation of the controller is shown in Figure A.2.

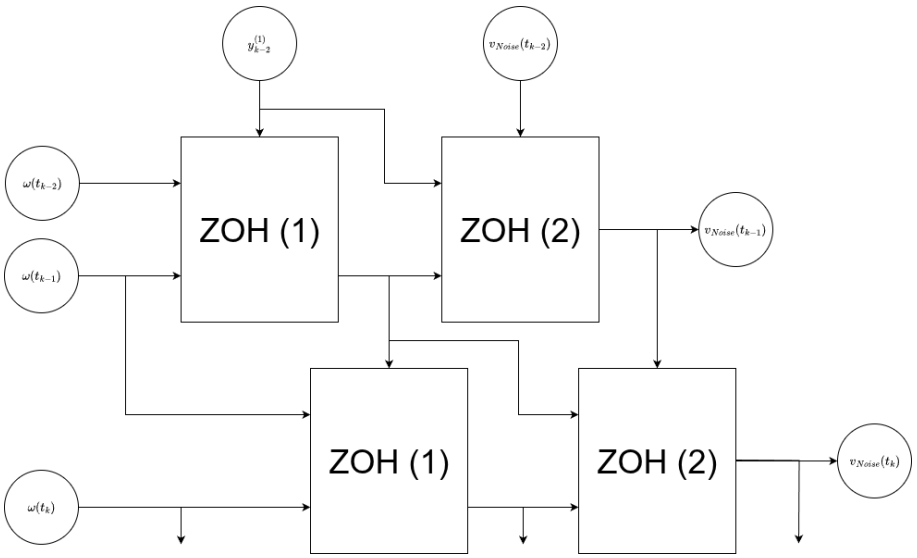


Figure A.2: Dryden noise generation process (v_{Noise})

The same approach is made for the w_{Noise} as well, following identical equations as **(A.3)** and diagram in Figure A.2.

Appendix B

Simulation Altitude Results

In this section, the plot for the reentry simulation altitude can be seen in Figure B.1 and Figure B.2. It is important to note that the results for the UARS case are more concentrated due to differences in the orbital characteristics.

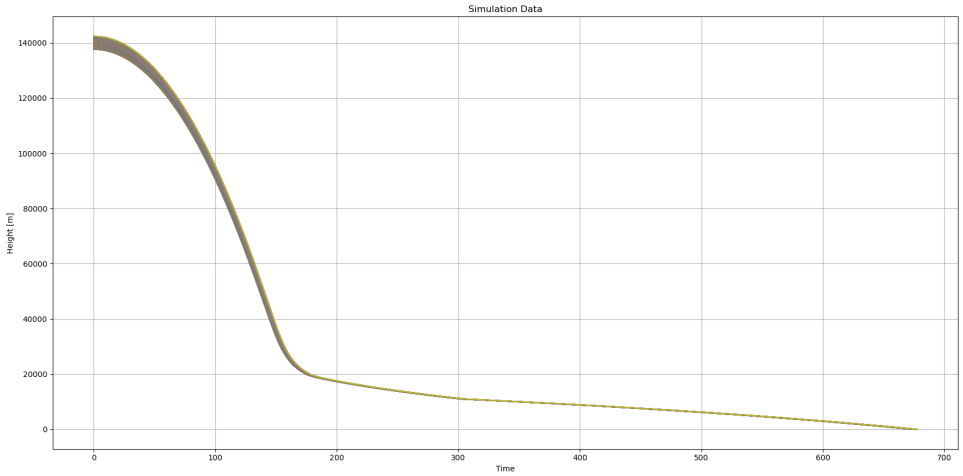


Figure B.1: Evolution of altitude for the UARS body

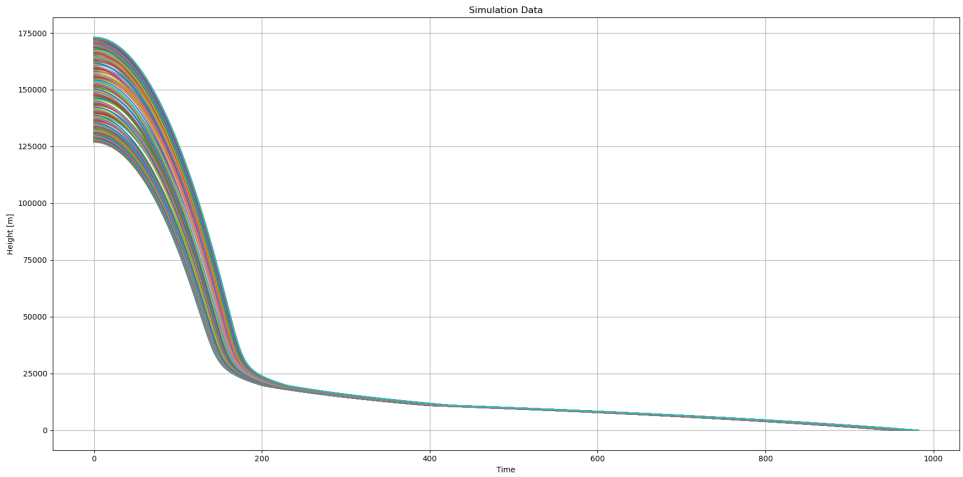


Figure B.2: Evolution of altitude for the KOSMOS 482 body

Appendix C

Second Training

Training-Test	Model x	Model y	Model z
UARS Debris (SSPP Structure)	$4.49 \cdot 10^{-3}$	3.35	1.67
KOSMOS 482	$1.64 \cdot 10^{-2}$	1.77	4.40

Table C.1: Kullback-Leibler test values for second training

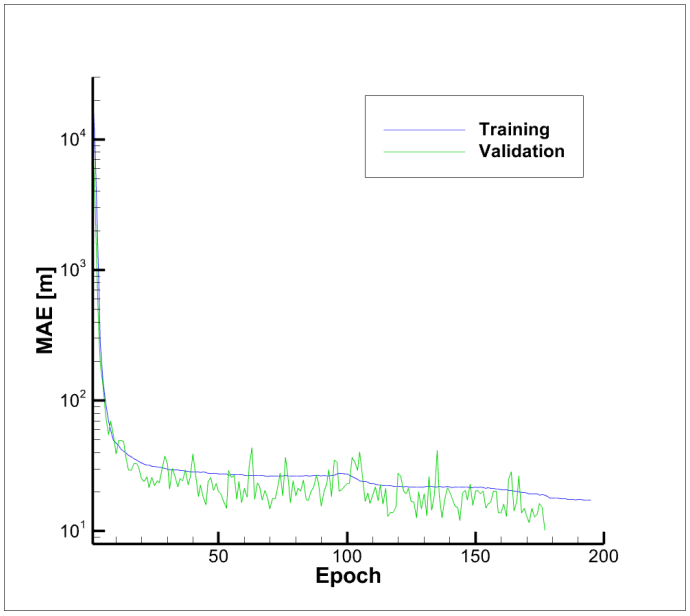


Figure C.1: Mean absolute error x – Body 1 (UARS)

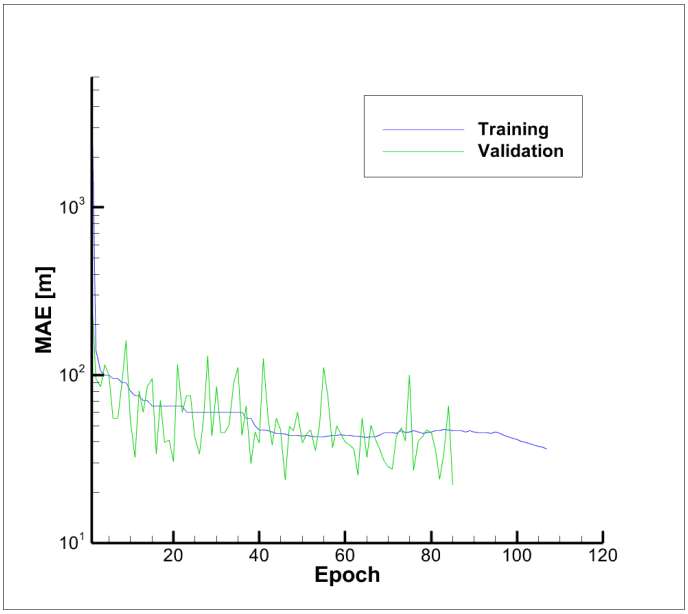


Figure C.2: Mean absolute error x – Body 2 (KOSMOS 482)

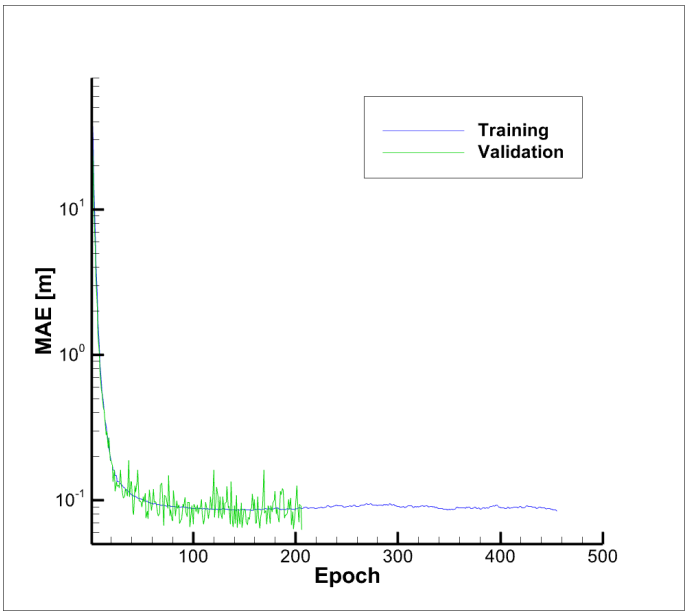


Figure C.3: Mean absolute error y – Body 1 (UARS)

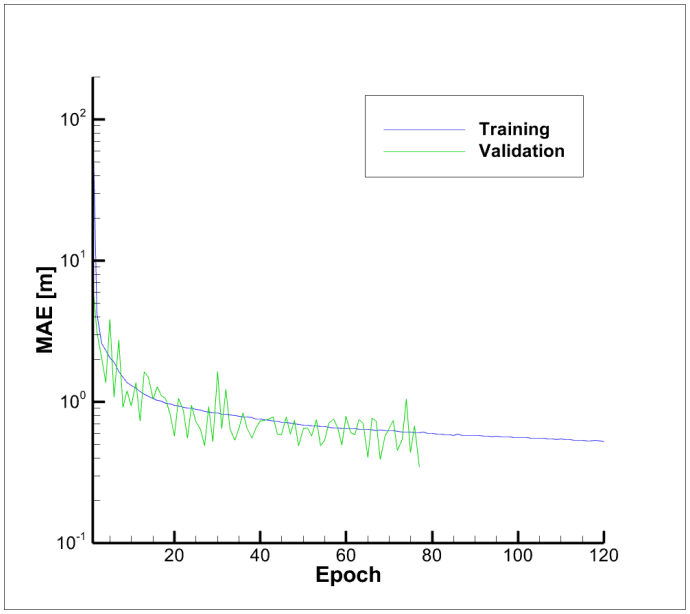


Figure C.4: Mean absolute error y – Body 2 (KOSMOS 482)

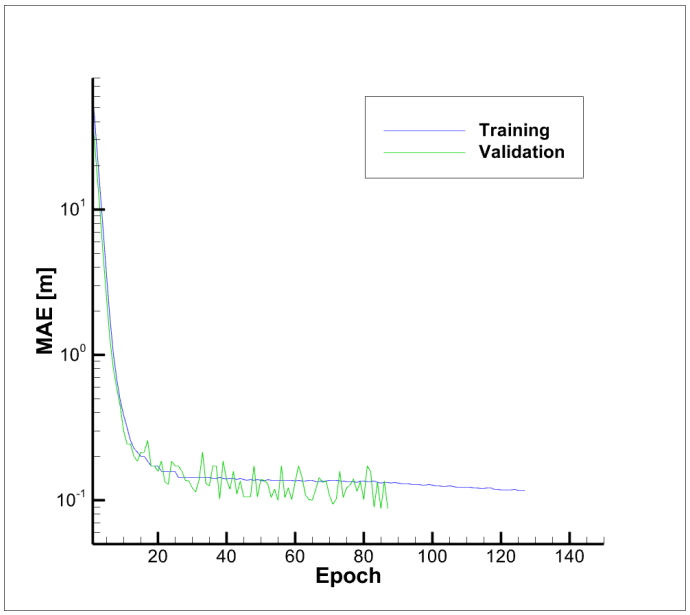


Figure C.5: Mean absolute error z – Body 1 (UARS)

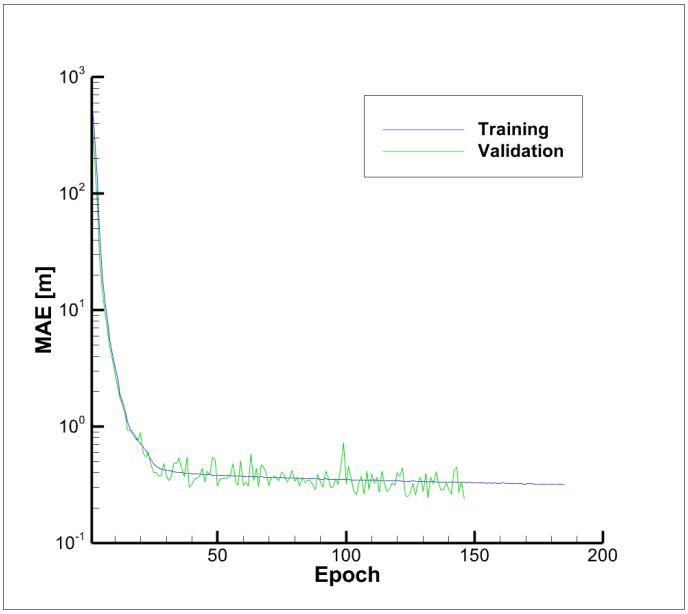


Figure C.6: Mean absolute error z – Body 2 (KOSMOS 482)

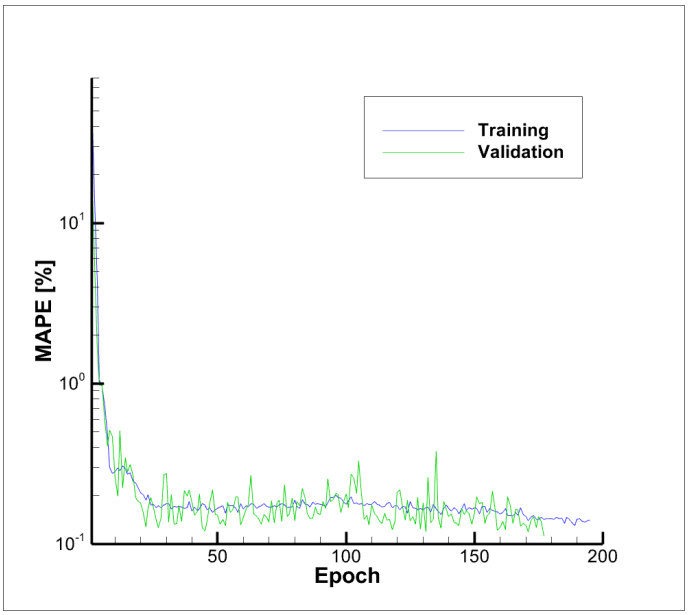


Figure C.7: Mean absolute percentage error x – Body 1 (UARS)

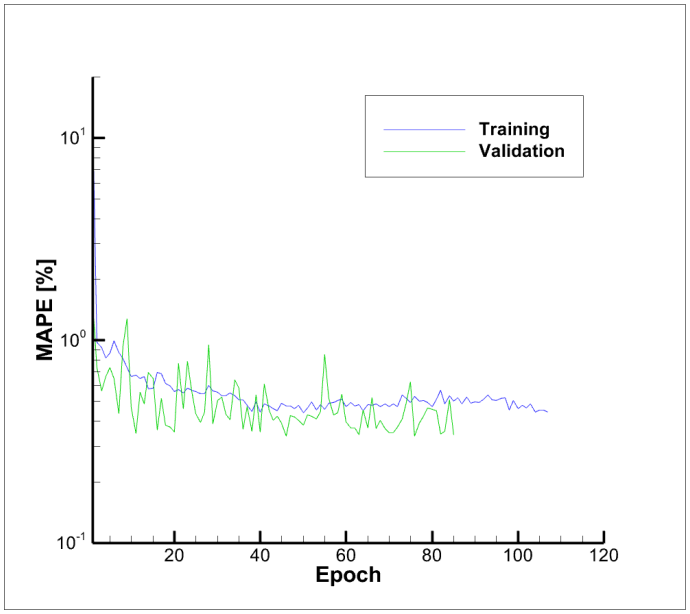


Figure C.8: Mean absolute percentage error x – Body 2 (KOSMOS 482)

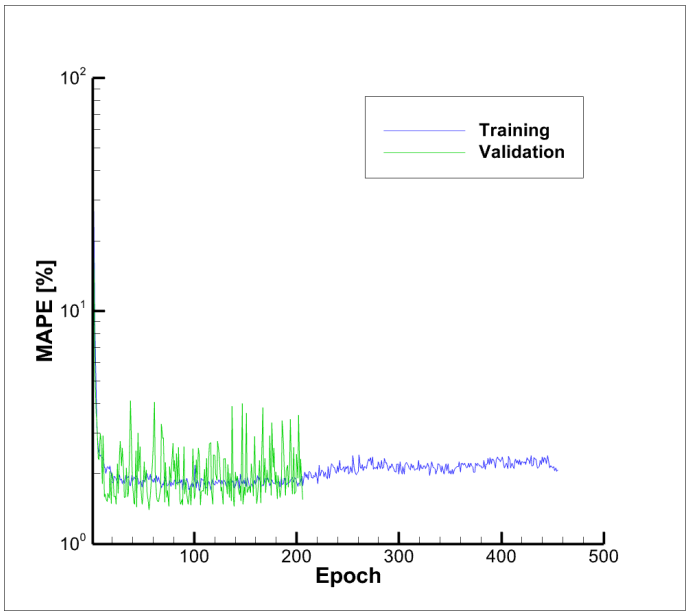


Figure C.9: Mean absolute percentage error y – Body 1 (UARS)

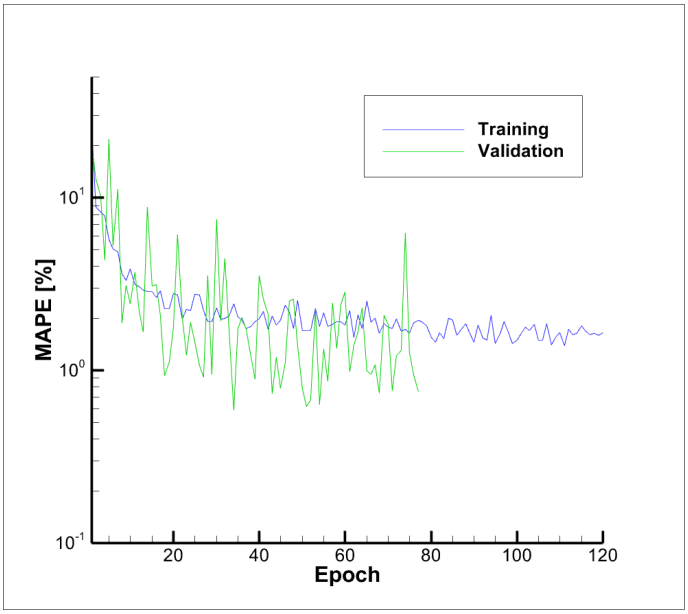


Figure C.10: Mean absolute percentage error y – Body 2 (KOSMOS 482)

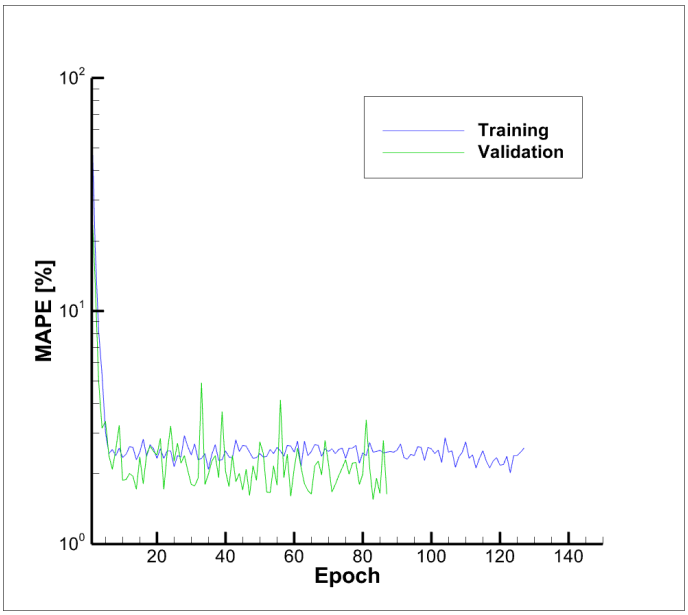


Figure C.11: Mean absolute percentage error z – Body 1 (UARS)

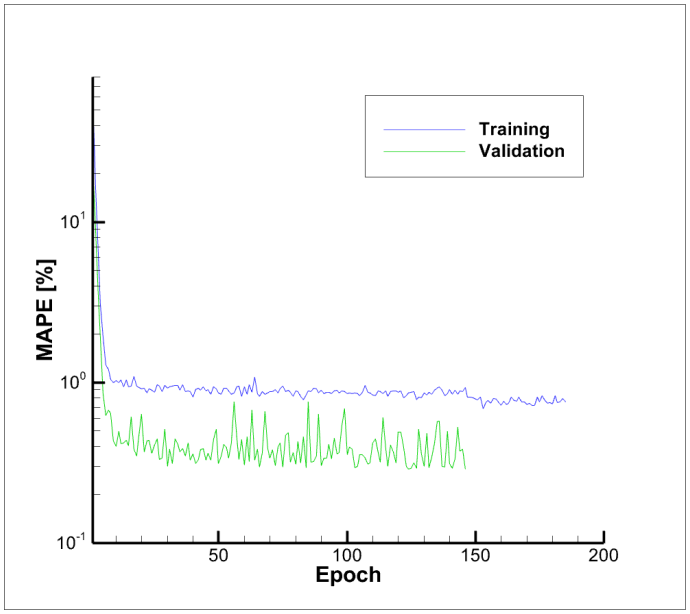


Figure C.12: Mean absolute percentage error z – Body 2 (KOSMOS 482)

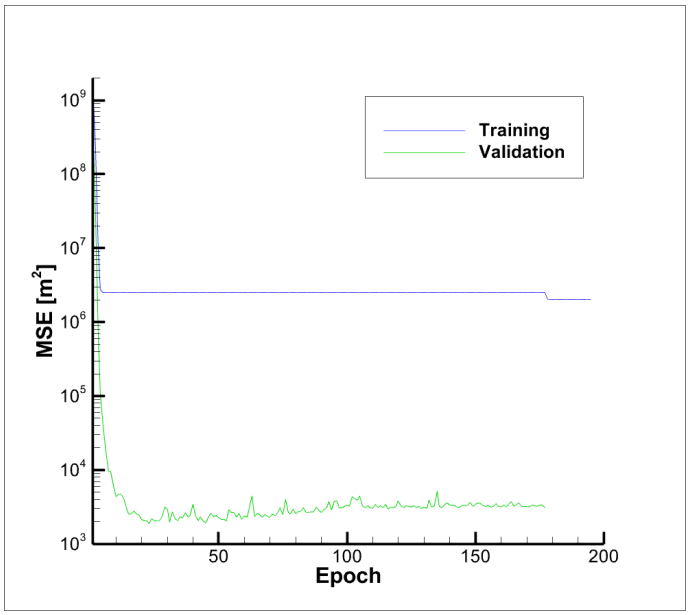


Figure C.13: Mean squared error x – Body 1 (UARS)

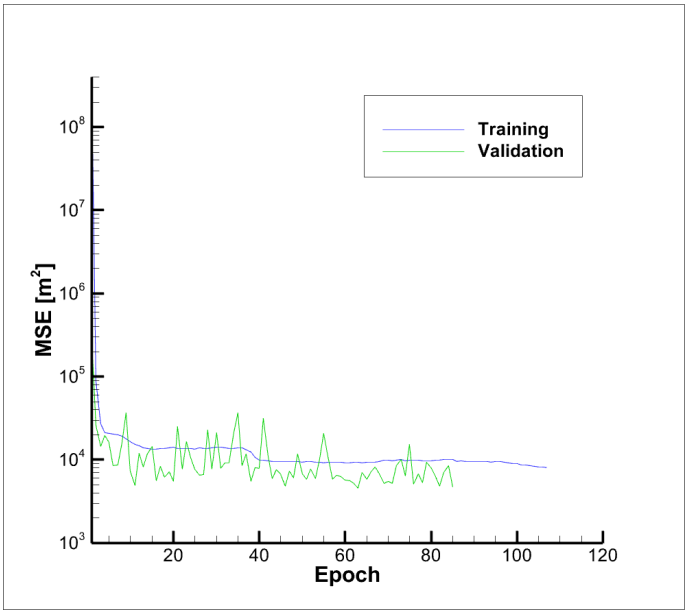


Figure C.14: Mean squared error x – Body 2 (KOSMOS 482)

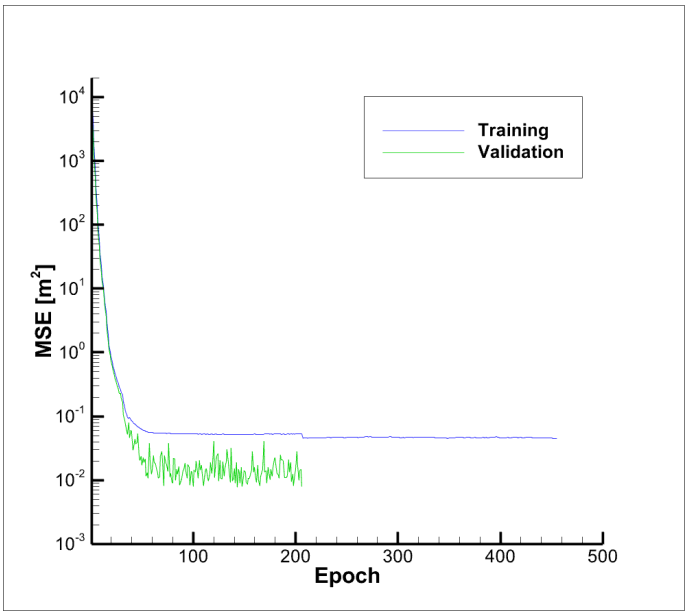


Figure C.15: Mean squared error y – Body 1 (UARS)

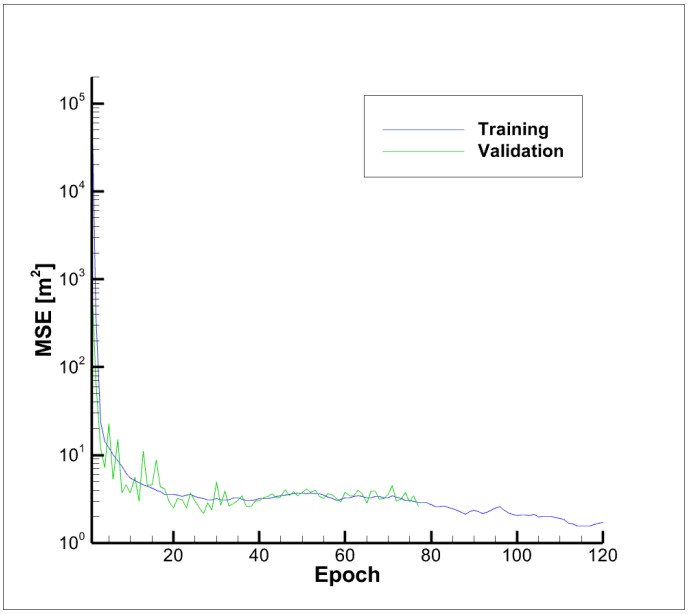


Figure C.16: Mean squared error y – Body 2 (KOSMOS 482)

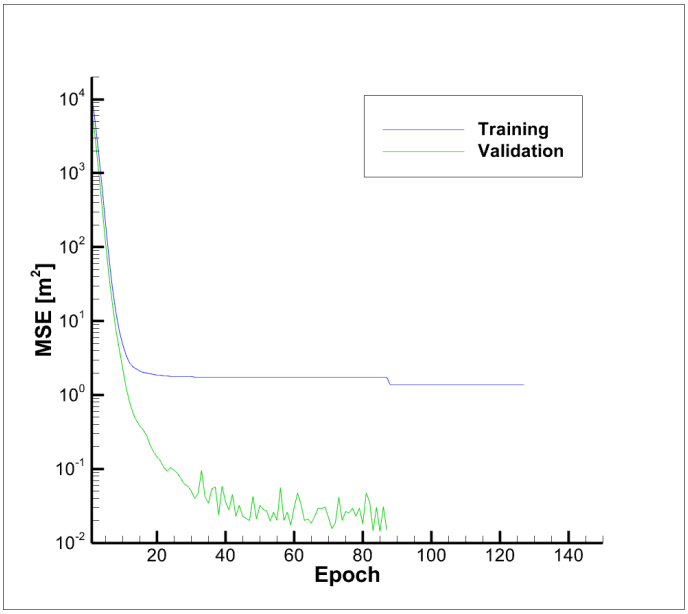


Figure C.17: Mean squared error z – Body 1 (UARS)

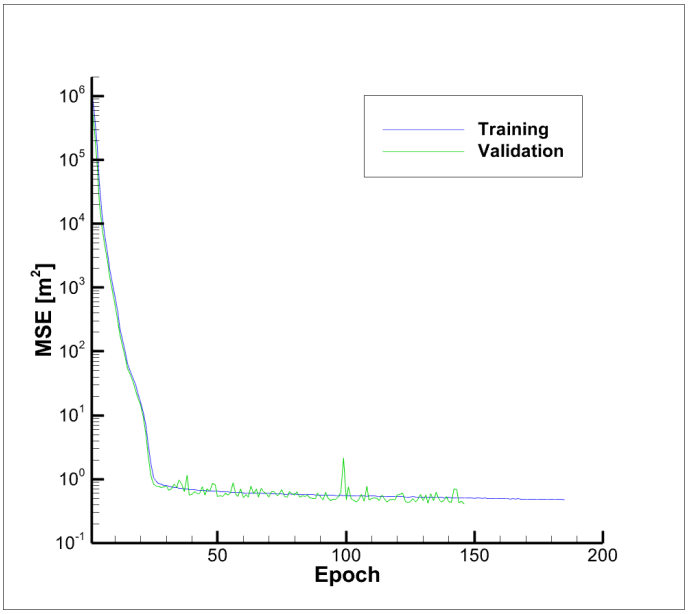


Figure C.18: Mean squared error z – Body 2 (KOSMOS 482)

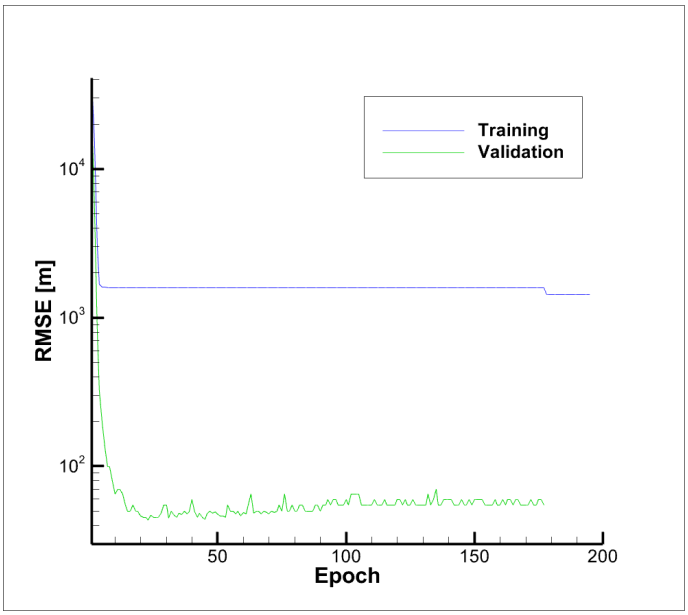


Figure C.19: Root mean squared error x – Body 1 (UARS)

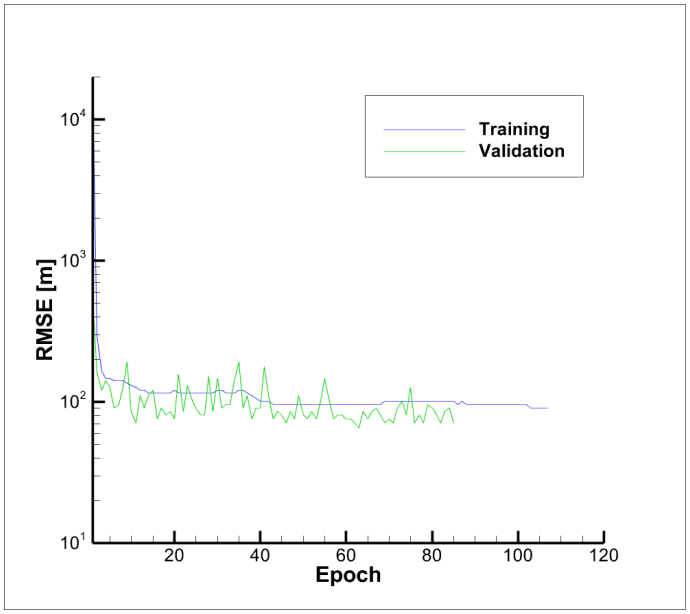


Figure C.20: Root mean squared error x – Body 2 (KOSMOS 482)

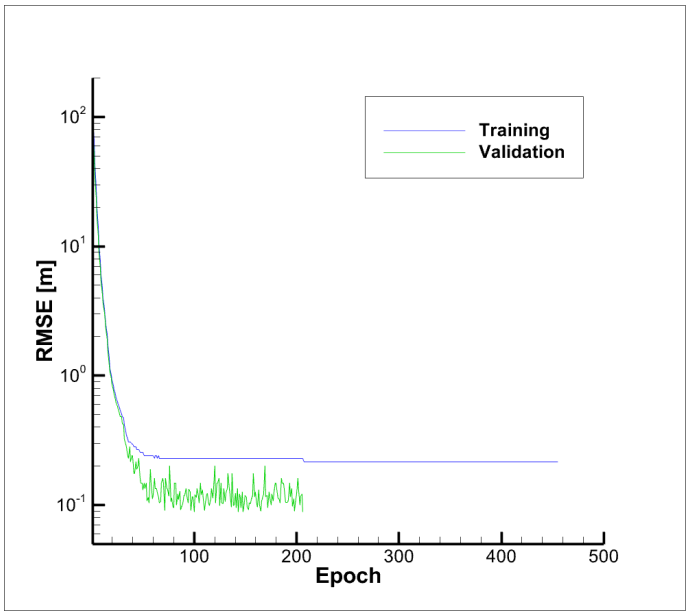


Figure C.21: Root mean squared error y – Body 1 (UARS)

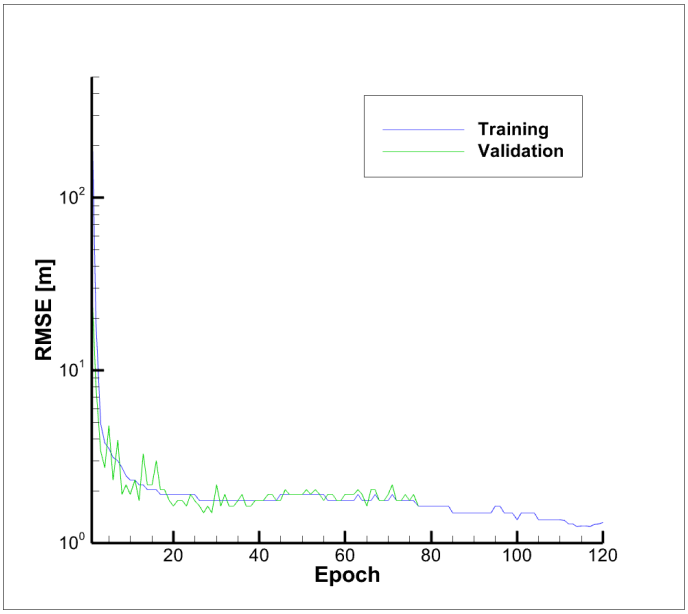


Figure C.22: Root mean squared error y – Body 2 (KOSMOS 482)

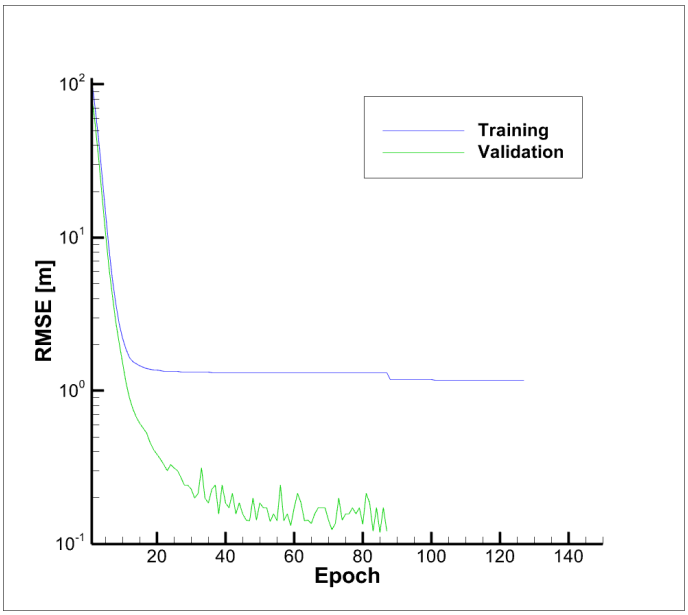


Figure C.23: Root mean squared error z – Body 1 (UARS)

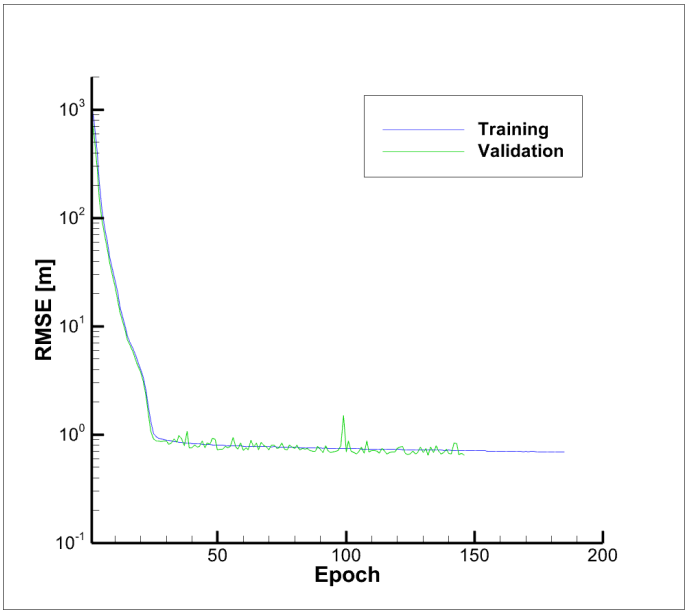


Figure C.24: Root mean squared error z – Body 2 (KOSMOS 482)

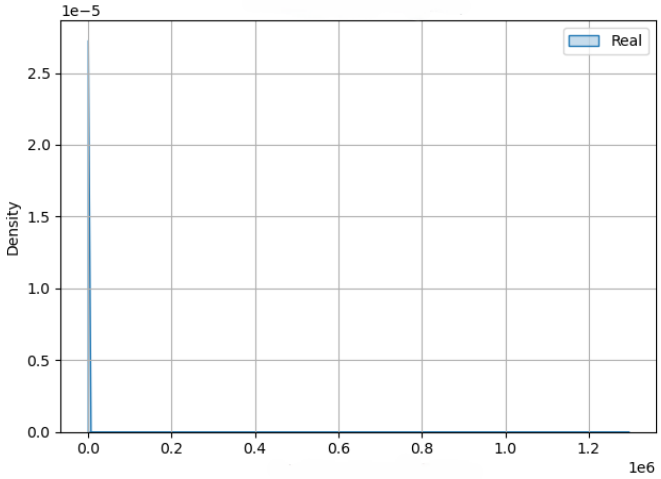


Figure C.25: Error function (x) – Training (UARS)

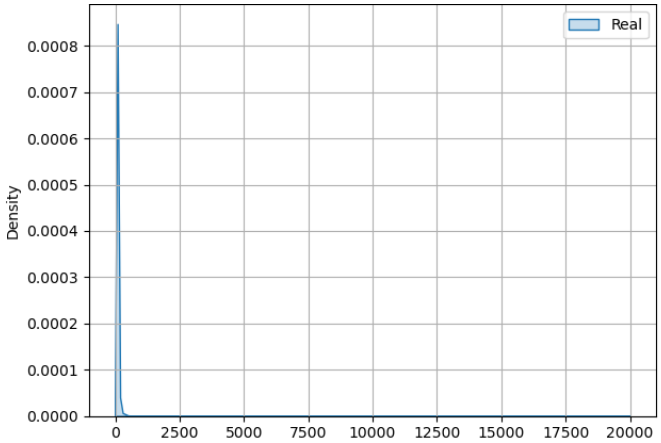


Figure C.26: Error function (x) – Test (UARS)

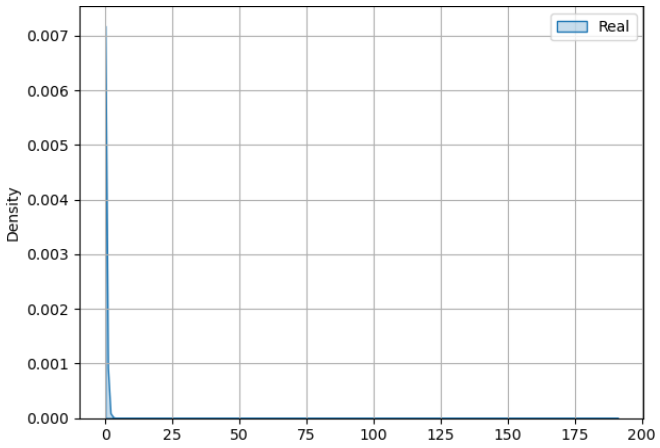


Figure C.27: Error function (y) – Training (UARS)

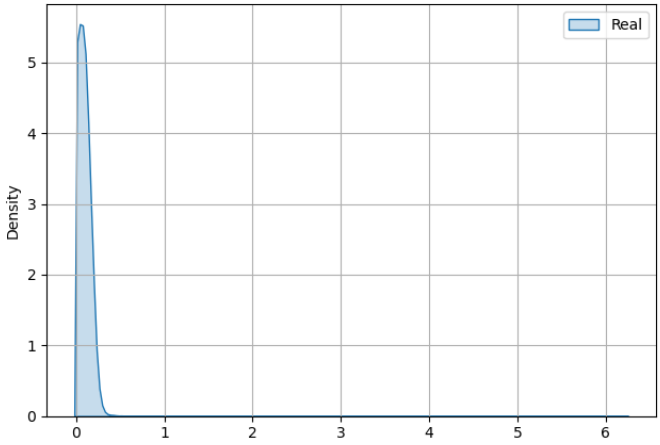


Figure C.28: Error function (y) – Test (UARS)

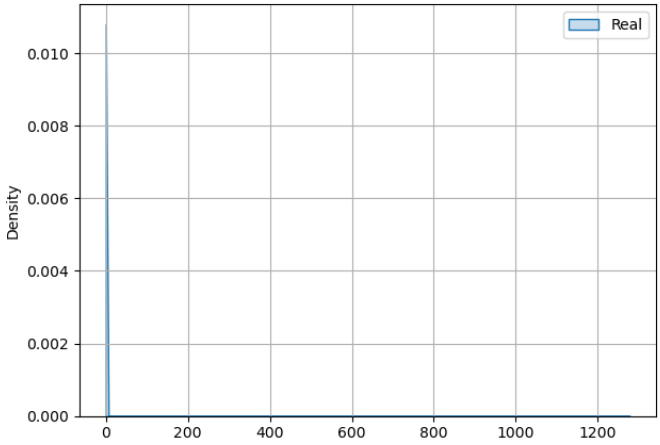


Figure C.29: Error function (z) – Training (UARS)

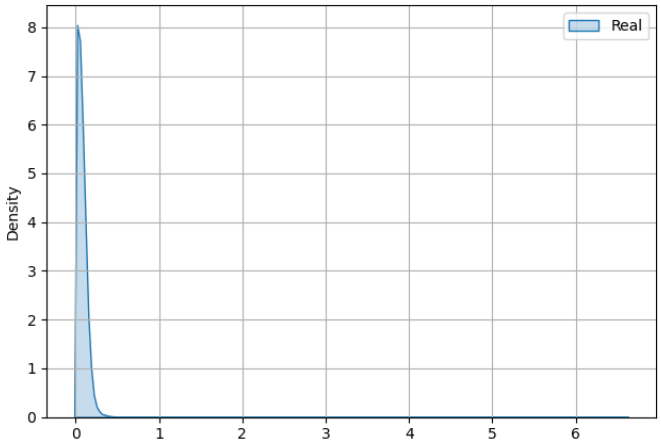


Figure C.30: Error function (z) – Test (UARS)

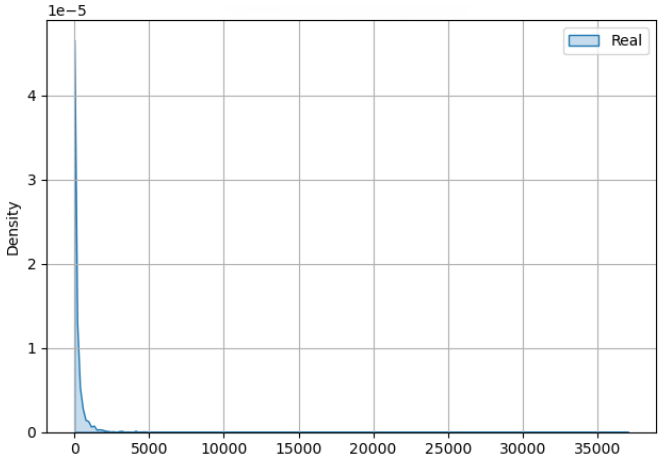


Figure C.31: Error function (x) – Training (KOSMOS 482)

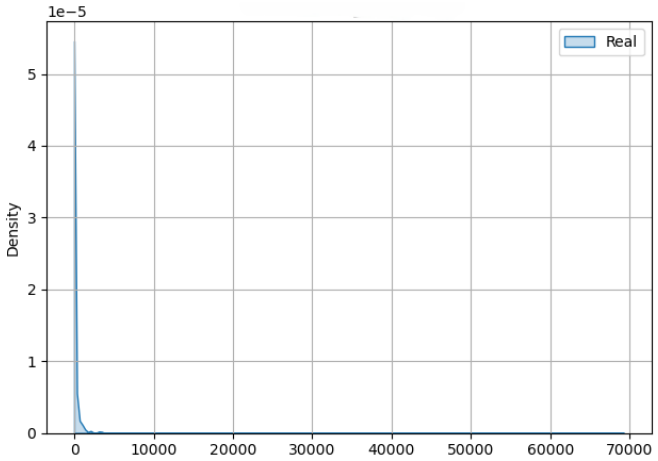


Figure C.32: Error function (x) – Test (KOSMOS 482)

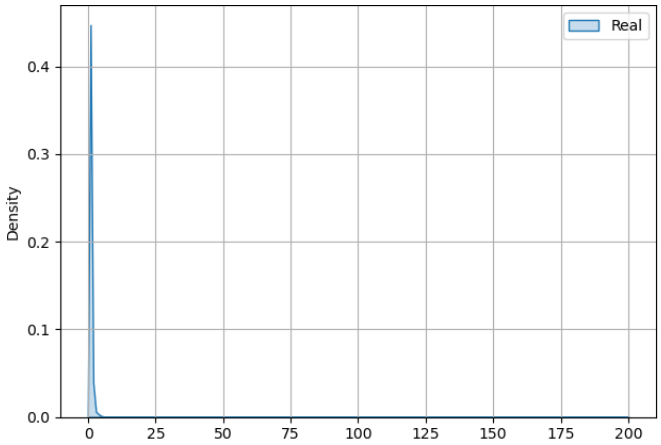


Figure C.33: Error function (y) – Training (KOSMOS 482)

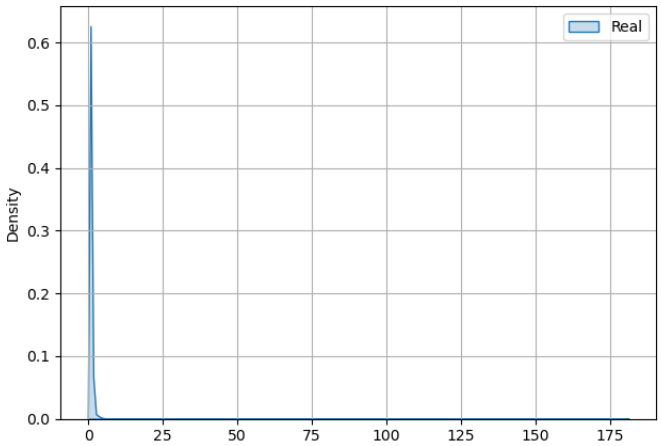


Figure C.34: Error function (y) – Test (KOSMOS 482)

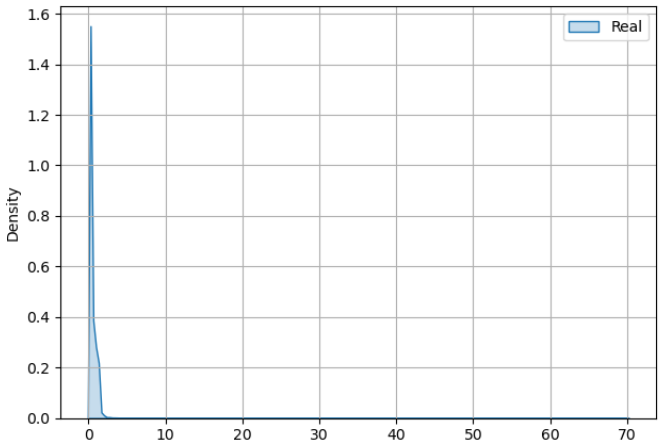


Figure C.35: Error function (z) – Training (KOSMOS 482)

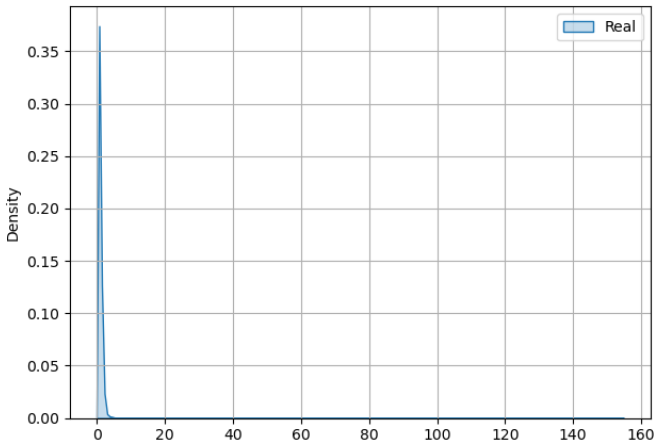


Figure C.36: Error function (z) – Test (KOSMOS 482)

Appendix D

Article (ISATECH'25)

The following pages show the article accepted to be presented at the International Symposium On Aircraft Technology, MRO & Operations 2025, which will take place in Almaty, Kazakhstan, from 3rd to 5th of December, 2025.

LSTM Based Atmospheric Reentry Trajectory Prediction

João Coelho, Kouamana Bousson

Abstract:

The rapid growth of the space economy has led to increased satellite launches, causing orbital congestion and a rise in uncontrolled reentries. This study focuses on predicting the reentry behavior of aerospace vehicles using a machine learning (ML) model, addressing the research gap below 80 km. The ML model employs three Long-Short Term Memory (LSTM) networks for positional state prediction in a cartesian frame from three prior states. Hyperparameters are tuned using Optuna's Tree-Structured Parzen Estimator to minimize Kullback-Leibler divergence (KL) between train and test error distributions. Training uses Tensorflow, with 80% of data for training (20% of that initially for validation) and 20% for testing. The KL results for UARS and KOSMOS 482 cases are 6.08 and 0.493, respectively, which validates successfully the proposed method.

Keywords: Atmospheric Reentry, Machine Learning, Trajectory Prediction, LSTM, Statistical Validation.

Nomenclature

LSTM	:	Long-Short Term Memory
ML	:	Machine Learning
KL	:	Kullback-Leibler (Jensen-Shannon) Divergence
NASA	:	National Aeronautics and Space Administration
UARS	:	Upper-Atmosphere Research Satellite
MAE	:	Mean Absolute Error

1. Introduction

Following previous studies (Salmaso, 2023; Jung et al., 2021), a robust modelling framework is developed to predict the reentry trajectory of space objects entering the Earth atmosphere or spacecraft vehicles. This is specifically the case of

J. Coelho[1], K. Bousson[1](✉)
[1]University of Beira Interior, Faculty of Engineering, Department of Aerospace Sciences, Covilhã, Portugal
e-mail: joao.pedro.monteiro.coelho@ubi.pt; bousson@ubi.pt

T.H. Karakoc et al. (eds.), ISATECH'25 Proceedings of International Symposium on Aviation Technology, MRO and Operations 2025
DOI 10.1007/…….

obsolete satellites that are disposed by means of burning up via atmospheric reentry, and inhabited spacecraft returning to Earth. It will be more tuned into the noise parameters caused by turbulence, providing more accuracy in the results predicted. To monitor the reentry process, various methods have been used, based on passive as well as active trajectory control procedures. The robustness of these methods may be significantly enhanced by resorting to the prediction of the reentry trajectory. While Machine Learning methods have been used for space trajectory predictions, they focused on classical neural network concepts (Jung et al., 2021; Stansbery and Johnson, 2012).

The present paper proposes a method for atmospheric reentry prediction based on the Long-Short Term Memory (LSTM) framework (Goodfellow et al., 2016) that inherently embodies the representativeness of a full dynamical system with the added values stemming from the fact that it bears the strength of keeping track of long-term memory of previous event states. The described research work focuses in the lower parts of the atmosphere, specifically below 80 km, and a new alternative to mathematical models by exploiting the full strength of LSTM techniques. The metric used to test the accuracy of the trained model is based on comparing the error probability density functions from both testing and training data pools.

2. Simulation Procedures

Due to the reduced availability of real sensor data, a simulation needs to be done first to obtain the data to train. It was decided to create a C program to do so. Two cases are to be analysed. The first, inspired in the UARS SSPP structure, involves the largest sized space debris estimation made by NASA (Stansbery and Johnson, 2012), while the second is inspired in recent events from Kosmos 482 (European Space Agency, 2025). Their general characteristics are described in Table 1.

Table 1: Characteristics of vehicles

Vehicle	**UARS**	**KOSMOS 482**
Mass [kg]	160	495
C_D [-]	0.5	2.2
S_{Wet} [m^2]	2.5	4.3
a[m]	$6.511 \cdot 10^6$	$6.521 \cdot 10^6$
e [-]	$3.64 \cdot 10^{-4}$	$3.54 \cdot 10^{-4}$
θ[^{o}]	{0, 1, …, 359}	{0, 1, …, 359}
i[^{o}]	56.93	51.95
Ω[^{o}]	262.2	241.9
ω[^{o}]	318.14	103.7

The initial orbit characteristics were defined based on last available data (Peat, 2011; Celestrack, 2025), being shown in Table 1, with fixed semi-major axis (a), eccentricity (e), inclination (i), longitude of ascending node (Ω) and argument of periapsis (ω). Multiple values of the true anomaly, θ, were adopted to generate sufficient data for the training.

The trajectory of the aerospace vehicle is considered as a three degree of freedom reentry motion model of a rotating sphere, where the sideslip is assumed to be equal to zero. The position parameters, including the geocentric distance r, longitude θ, latitude φ, are defined in the geocentric spherical fixed coordinate system. The velocity parameters include the speed V, track angle γ and course angle ψ. The International Civil Aviation Organization (ICAO) Standard Atmosphere equations were used (ICAO, 2008).

Finally, the turbulence model proposed by Dryden (Madden, 2018) is used to model possible disturbances and noises in our reentry model, further improving the robustness of it. The equations are represented in the body reference frame, from where the Laplace transform was used. The white noise used for the input was $\omega \sim \mathcal{N}(0,100)$. To compute the inverse z-transform of the equations, a zero-order hold representation was needed (Madden, 2018).

Using the initial conditions provided in Table 1, the solution is generated through discretization of the continuous state-space model of the trajectory of the aerospace vehicle, with the application of the Euler's Modified Semi-Implicit Method (Cromer, 1981) with the time-step of 0.001s. All operations were conducted in the geocentric cartesian reference frame.

3. Model Composition and Training

The three-dimensional state vectors are defined in Eq. 1, where $\boldsymbol{D}$ corresponds to the Dryden model output.

$$\begin{cases} \boldsymbol{X}(t) = \boldsymbol{X}_{Clean}(t) \\ \boldsymbol{Y}(t) = \boldsymbol{X}_{Measured}(t) = \boldsymbol{X}_{Clean}(t) + \boldsymbol{X}_{Noise}(t) = \boldsymbol{X}_{Clean}(t) + \boldsymbol{D}(\omega(t)) \end{cases} \tag{1}$$

The LSTM based model can be viewed in Fig. 1.

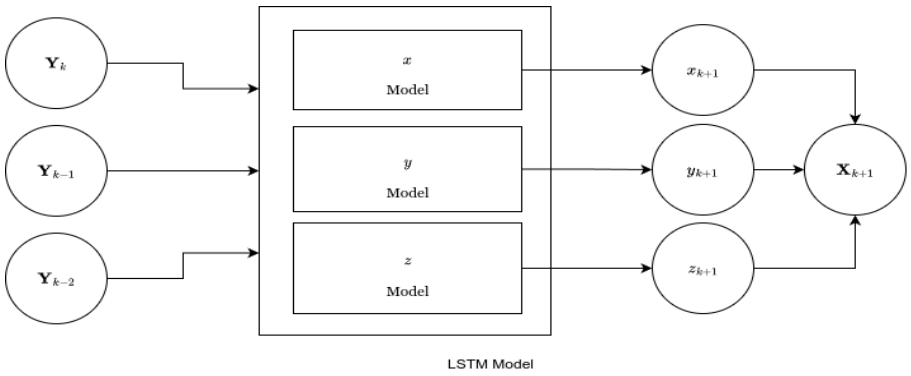


Fig.1. Flowchart of the LSTM model behavior

All training was made using Python *Tensorflow*. The hyperparameter tuning was also made using the *Optuna* framework (Akiba et al., 2019), adopting the tree-structured Parzen estimator approach (Watanabe, 2023) in two phases. The primary one is focused on the six parameters. Those parameters are described below:

- Hidden Layers: {1, 2, 3};

- Hidden Layer Inputs: {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096};
- Optimizer: {SGD, RMSProp, ADAM, NADAM, ADAMw};
- Loss Functions: {MAE, MSE, Huber};
- Loss Ratio: [1e-6, 1e-3];
- Batch Size: {32, 64, 128}.

The objective of this search was to obtain the combination of hyperparameters that shown the best Kullback-Leibler divergence value (Lin, 1991) between the error probability-density functions in both training and testing sets, which were generated using a Gaussian kernel density estimation procedure. Since the based model is comprised of three multiple input single output (MISO) LSTM models, a geometric mean approach was made at the end. The final loss ratio for each of the models can be consulted in Table 2.

Loss Ratio | Epochs	Model x	Model y	Model z
UARS	$1.607 \cdot 10^{-5} \mid 496+51$	$1.006 \cdot 10^{-5} \mid 998+5$	$1.005 \cdot 10^{-5} \mid 994+5$
KOSMOS 482	$9.27 \cdot 10^{-5} \mid 276+58$	$9.27 \cdot 10^{-5} \mid 436+58$	$1.006 \cdot 10^{-5} \mid 717+15$

Table 2: Loss ratio and epochs for each model

With the architecture defined, the actual training process takes place. It is made with a maximum number of 2000 epochs, achieving the end once there is no improvement in the loss function during an interval of 50 epochs. The division between training, validation and test sets is of 64%, 16% and 20% respectively. Once this first phase is completed, the training and validation sets are joined, and two independent possibilities happen. Either the last epoch number is recorded, and the training process is made from scratch until there, or the value registered by the loss function in the validation set is recorded and the training is made over the weights obtained in the first phase until that value is reached by the training set. The option used depended on the best registered Kullback-Leibler divergence value.

4. Conclusion

The number of epochs spent by the training process can be observed in Table 2. It was also noted that the best option for the training approach during the second phase was always the second one. The Kullback-Leibler divergence values for UARS and KOSMOS 482 cases are 6.08 and 0.493, respectively.

The divergence results show a good correspondence between the training errors and the testing ones, implying that the training process was well done. The discrepancy between the values for both cases encouraged a second train, where the UARS got a decrease of approximately 38% to 3.74, while the KOSMOS got an exponential increase up to 4.74. For this, one could argue that the main reason for this difference is related to the random nature of the division between the training and testing sets. Nevertheless, should a similar study be conducted, an adoption for more realistic data should be considered and, if still not available, a more precise modelling

should be done, having in consideration variables such as the chemical composition of the atmosphere, solar index, and many more.

References

Akiba T, Sano S, Yanase T, Ohta T, Koyama M (2019) Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

CELESTRACK (2025) COSMOS 482 Descent Craft. Data post, updated 10 May 2025. https://celestrak.org/NORAD/elements/graph-altitude.php?CATNR=6073

Cromer AH (1981) Stable solutions using the Euler approximation. American Journal of Physics 49(5):455-459. https://doi.org/10.1119/1.12589

Dunlop S (2008) ICAO standard atmosphere. Oxford University Press. https://doi.org/10.1093/acref/9780199541447.013.2720

European Space Agency (2025) Re-entry prediction Soviet-era Venera Venus lander (Cosmos-482 descent craft). Blog post, updated 13 May 2025. https://blogs.esa.int/rocketscience/2025/05/07/reentry-prediction-soviet-era-venera-venus-lander-cosmos-482-descent-craft

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press.

Jung O, Seong J, Jung Y, Bang H (2021) Recurrent neural network model to predict re-entry trajectories of uncontrolled space objects. Advances in Space Research 68(6):2515-2529. https://doi.org/10.1016/j.asr.2021.04.041

Lin J (1991) Divergence measures based on the Shannon entropy. IEEE Transactions on Information Theory. Vol. 37, No. 1, pp.145-151. https://doi.org/10.1109/18.61115

Madden MM (2018) Verifying Implementation of the Dryden Turbulence Model and MIL-F-8785 Gust Gradient. Modelling and Simulation Technologies Conference, Atlanta, GA. https://doi.org/10.2514/6.2018-3580

Peat (2025) UARS – Orbit. Blog post, updated 24 September 2011. https://www.heavens-above.com/orbit.aspx?satid=21701

Salmaso F, Trisolini M, Colombo C (2023) A Machine Learning and Feature Engineering Approach for the Prediction of the Uncontrolled Re-Entry of Space Objects. Aerospace 10(3):297. https://doi.org/10.3390/aerospace10030297

Stansbery E, Johnson NL (2012) NASA Upper Atmosphere Research Satellite (UARS) Re-entry Prediction and Analysis. International Symposium on Sustainable Space Development and Utilization for Humankind: Orbital Space Debris – Challenges & Opportunities, Shinagawa, Tokyo, Japan.

Watanabe S (2023) Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. arXiv:2304.11127. https://doi.org/10.48550/arXiv.2304.11127