Project 2

Newton vs the machine: solving the chaotic three-body problem using deep neural networks

Number of pages (excluding title page & references): 7

Hampus Hansen

CID: `hhansen`

Erik Karlsson Öhman

CID: `erikohm`

1 Problem description

The first formulation of the three body problem was made by Newton in 1687, where he attempted to find the long term stability of the gravitational system consisting of Earth, the Sun and the Moon [1]. Since then, physicists and mathematicians have tried to find the explicit analytical expression of such a system, but without success due to the chaotic nature of the three body systems. Some of the greatest names in the sciences have wrestled with this problem, and in 1889 Henri Poincaré was awarded the King Oscar prize by Oscar II for his progress in the problem. After Lars Phragmen's persistent correspondance Poincaré noted a serious error in the paper published in Acta Mathematica. The founder of the journal requested the already mailed journals to be returned, and Poincaré personally paid for the second corrected printing, a price that far exceeded the sum he recieved for the King Oscar prize [2]. The three body problem is clearly an extremely challenging problem...

Furthermore, the numerical solution is very computationally demanding, with some instances requiring infinite computational time [3]. The problem studied in this project is to solve the dynamics of a three-body system with equal mass under mutual influence of classical gravitational forces. The particles are assumed to start from rest. The coordinate system is chosen such that the dynamics of the particles are fully characterized by two coordinates, (x, y) [6]. The first particle, p_1, always starts in the point $(1, 0)$ and the second particle, p_2 starts somewhere in the second quantile. Given that the centre of mass-framework is utilized, it is possible to infer the third particle's position from the other particle's postitions, $x_3 = -(x_2 + x_1)$ and $y_3 = -(y_2 + y_1)$. The provided data consists of 9000 trajectories, each for a separate initialization, and is generated by `Brutus` [4]. The computational time for generating each data instance ranged from a few minutes to hours.

To more efficiently predict the trajectories an artificial neural network (ANN) can be implemented, as shown in the article *Newton vs the machine: solving the chaotic three-body problem using deep neural networks* by Breen et al. [3], in which they show that the ANN can decrease the computational time by a factor of 10^8. This project aims to recreate the results from [3] by constructing an ANN and training it on data generated by Brutus. The predictive power of this ANN will be compared to the pre-trained model used in [3]. The computational time of a prediction given a new initialization will be compared to the typical computational time required by `Brutus`.

2 Method description

An artificial neural network is specified by its architecture, activation rule and learning algorithm. The ANN used in this project is a feed forward ANN. The architecture specifies the dimension of the input layer, the number of hidden layers, the number of neurons in each hidden layer and the output dimension. The input array is three dimensional, specifying the time interval and the initialization, i.e. the start position of the second particle, p_2, from which all other particle starting positions can be inferred. The hidden layers consists of ten dense layers each with 128 neurons. The output layer is four dimensional, consisting of the position for p_1 and p_2 throughout the time inverval specified in the input. The model was constructed with the functions `keras.models.Sequential()`, `keras.layers.InputLayer()` and `keras.layers.Dense()` from the TensorFlow library.

The activation rule determines how the neurons feed the signal forward, dictated by the activation function in each layer. The activation functions used in the hidden layers were Rectified Linear Units (ReLU) defined by

$$f_{\text{ReLU}}(z) = \max(0, z), \tag{1}$$

where z is the activation and consists of weights, w, and some bias, b, such that

$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{i,j}^l a_i^{l-1} + b_j^l, \tag{2}$$

for some neuron j in layer l with input signals a_i^{l-1}. Setting the activation function to zero for negative valued activations introduces a non-linearity, which is essencial for fitting non-linear systems. This also ensures that the neurons with a very small activation is made completly quiet, which increases effeciency and counters noisy networks. The drawback of this is that some neurons will not be trained properly since the the gradient is zero for all negative activations, resulting in a worse fit. This could be solved by using the Exponential Linear Units (ELU), or the leaky ReLU activation function, but this was considered not necessary for the purpose of this project. The activation function of the output layer is

linear since this is typically the prefered activation function for the output for a regression-type problem.

The learning algorithm is the process that changes the weights during training. Back propagation allows the computation of gradients of the cost function. The cost function to minimize is the Mean Absolute Error (MAE). The choice is motivated by the goal to minimize the distance between the data and the predicted trajectory. The method used for minimizing is a stochastic gradient decent (SGD) method implemented with `keras.optimizers.Adam` with learning rate 0.001 and moments 0.5, motivated by the project description [6]. These hyperparameters generated satisfactory results, and were thus not tweaked further. The computation of the gradients is made possible by back propagation. The stochasticity in the SGD method is introduced by randomly shuffling the data and splitting it into mini-batches of size 5000 with the function `batch()` from the TensorFlow library. For the extra task the batch size is 1000 [6]. The time interval consists of 1000 data points, and this divisibility with the batch size ensures that no trajectory is split into separate mini batches. For both the main and extra task 90% of the data is used for training and 10% for validation.

During the extra task a further constraint is imposed on the problem, namely the conservation of energy. This is done by altering the mean absolute error loss function to also include a term corresponding to the energy error. Whether the network, now subject to the new constraint, effectively maintains energy conservation is also analyzed. This investigation is then followed by a comparison with the pre-trained network and the data generated by `Brutus`.

In practice the new loss function will look like `tf.keras.losses.MeanAbsoluteError()(y, y_pred) + 0.001*energy_loss`, where the `energy_loss` variable is given by

$$\Delta E = |E_k^{\text{Predicted}}(t) + E_p^{\text{Predicted}}(t) - E_p^{\text{Input}}(0)| \tag{3}$$

$$E_{loss} = \overline{\Delta E} \tag{4}$$

i.e. the mean of the difference between the total energy of the system and the initial energy of the system. As the system starts in rest the initial energy is simply the potential energy at $t = 0$. Furthermore the potential energy for three bodies is given by

$$E_p = -G\left(\frac{m_1 m_2}{|\mathbf{r}_1 - \mathbf{r}_2|} + \frac{m_1 m_3}{|\mathbf{r}_1 - \mathbf{r}_3|} + \frac{m_2 m_3}{|\mathbf{r}_2 - \mathbf{r}_3|}\right) \tag{5}$$

where $\mathbf{r}_i = (x_i, y_i)$ is the position of a body. In this particular case we set $G = 1$ and $m_1 = m_2 = m_3 = 1$. The kinetic energy of the bodies is simply given by

$$E_k = \frac{m_1 \mathbf{v}_1^2}{2} + \frac{m_2 \mathbf{v}_2^2}{2} + \frac{m_3 \mathbf{v}_3^2}{2}. \tag{6}$$

The dataset generated by `Brutus` includes v_1 and v_2. By requiring conservation of momentum we can deduce that $\mathbf{v}_3 = \frac{-(m_1\mathbf{v}_1 + m_2\mathbf{v}_2)}{m_3} = -(\mathbf{v}_1 + \mathbf{v}_2)$. The ANNs however does not directly predict any velocities, instead they are estimated by differentiating the predicted positions with regards to time $\mathbf{v}_i^{t+1} = \frac{\mathbf{r}_i^{t+1} - \mathbf{r}_i^t}{\Delta t}$.

The factor of 0.001 is a hyperparameter of our loss function and it essentially determines to what extent the neural network prioritizes minimizing the mean absolute error versus the minimzing of the energy loss. Due to the extensive training time of the network, the optimization of this hyperparameter has not been explored. The neural network employed in the extra task is apart from the loss function identical to the one outlined in the main task.

3 Results and discussion

The results and the following discussion are presented below.

3.1 Results on training and validation

The model prediction of two random examples when the initialization is from the training data and the validation data is presented in Figure 1. A first observation is that the model predicts the general tendencies of system relatively well. At the end of the time interval it starts to deviate from the `Brutus` computed trajectory. A second observation is that the performance of the model is similar when initialized with data from the training and validation set. This indicates that the model generalizes well to input data outside of the training set.

However, some initializations, from both the training and validation set, did not result in a successful prediction of the system's dynamics. These initializations are likely difficult to predict if the system is highly chaotic early on in the time interval, and these are probably the same trajectories that took the longest time for `Brutus` to calculate.

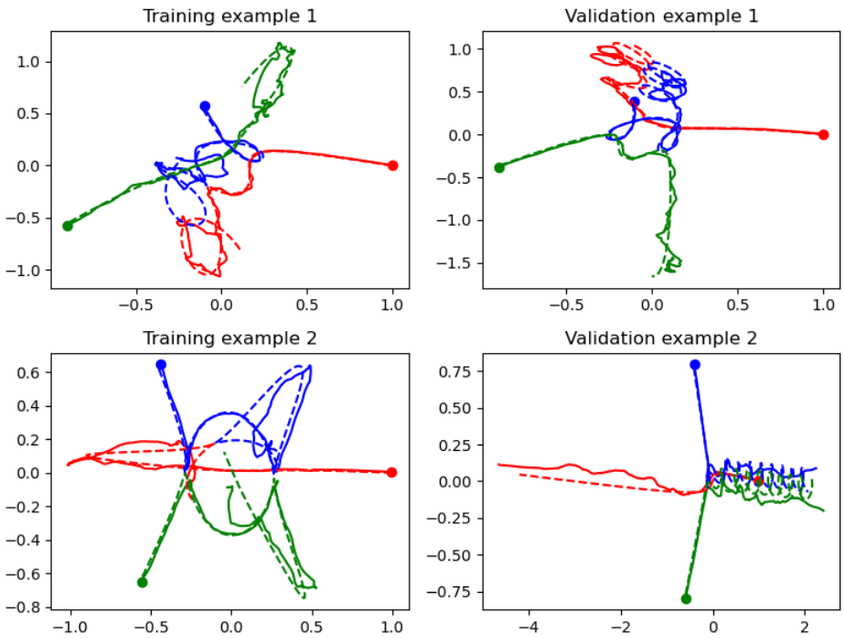

Figure 1: Two simulated trajectories initialized with data from the training set and two simulated trajectories initialized with data from the validation set. Solid lines are trajectories predicted by our neural network, dashed lines are trajectories generated by `Brutus`.

The pre-trained neural network outperforms our neural network, as seen in Figure 2. This is expected since the pre-trained neural network has optimized hyperparameters and is trained over 10000 epochs. Despite this, there is still a clear, but small, deviation from the `Brutus` calculated trajectory at the end of the time interval. The average prediction times of a trajectory were found to be $89\,\text{ms} \pm 1.25\,\text{ms}$ for the pre-trained neural network and $94.1\,\text{ms} \pm 8.85\,\text{ms}$ for our neural network.

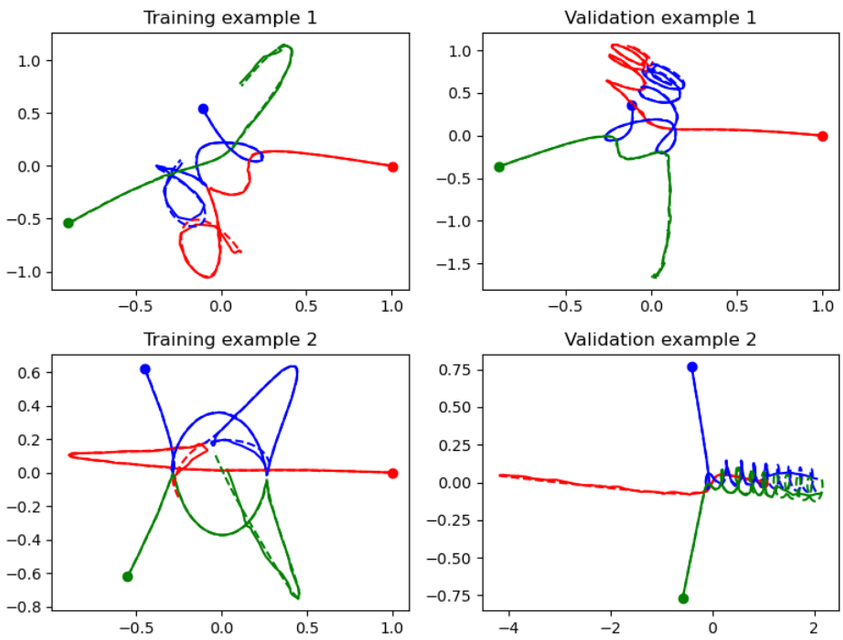

Figure 2: Two simulated trajectories initialized with data from the training set and two simulated trajectories initialized with data from the validation set. Solid lines are trajectories predicted by the pre-trained neural network, dashed lines are trajectories generated by `Brutus`.

From Figure 3 it is apparent that the mean absolute error decreases with epochs for both training and validation data. If the model were to be significantly overfitted the validation error would eventually increase whilst the training error would remain small. This does not occur, the training and validation scores are 0.0695 and 0.0650 respectively after 100 epochs, and as such the model is likely not overfitted. This conclusion is also supported by the fact that the trajectories intialized by validation data does not seem to perform notably worse than the trajectories intialized by training data in Figure 1.

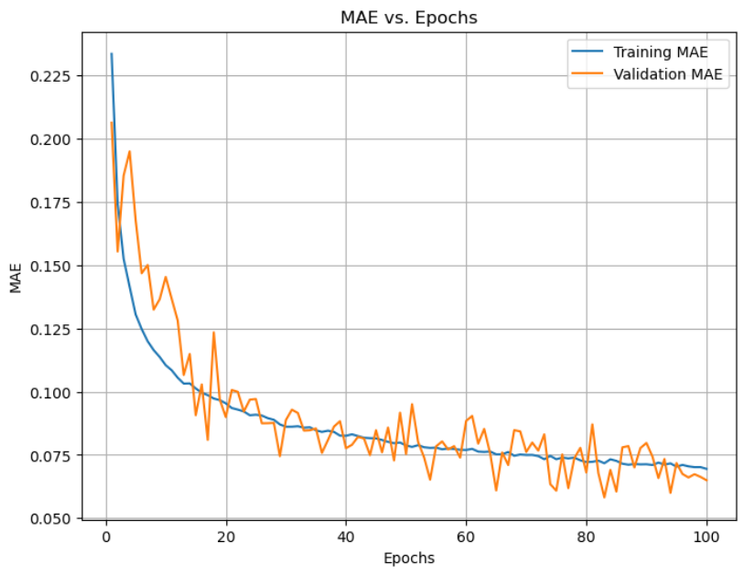

Figure 3: Mean absolute error as a function of training epochs. The blue line represents the training loss and the orange line the validation loss.

The chaotic nature of the system is visualized in Figure 4. As time passes, the slight shift in initial position results in greater spread of the particle's position, as expected. The final position is widely spread out, which is indicative of the chaotic nature of the three-body systems. The spread of the last and second to last time differs dramatically between the two neural networks. This is anticipated since the predictions at the end of the time interval

starts to differ significantly from each other, compare Figure 1 and 2.

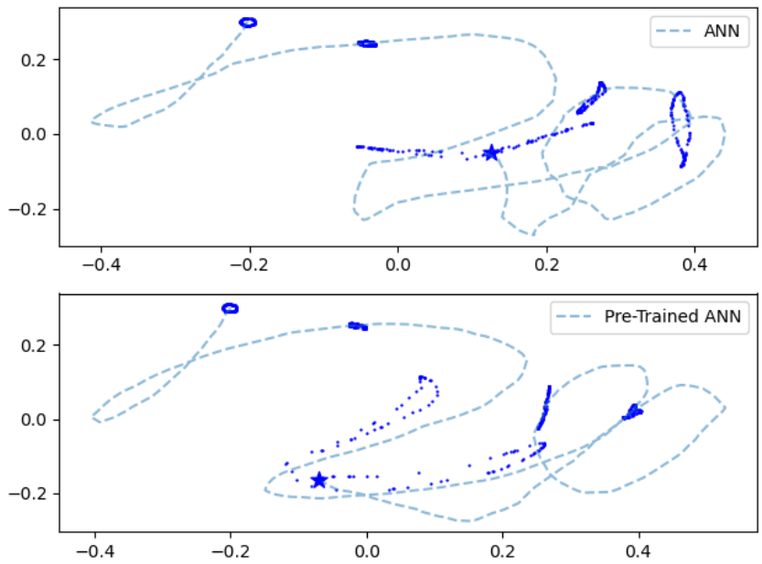

Figure 4: Visualization of sensitivity depending on initial position. The sequences of dark blue dots are generated by shifting the inital position around a circumference with radius 0.01 and centred at $(-0.2, 0.3)$ at times $t = \{0.0,\ 0.95,\ 1.9,\ 2.95,\ 3.8\}$. The dashed line represents the trajectory of p_2 with initial position at the circle's centre for our ANN (pre-trained ANN) in the upper (lower) figure. The stars represents the final position for the dashed trajectories.

4 Extra Task

By calculating the relative energy error for the ANN with custom loss function, the pre-trained network and `Brutus` a new metric of performance is obtained. The relative energy error is given by the following expression

$$E_{\text{relative error}} = \left| \frac{E_k^{\text{Predicted}}(t) + E_p^{\text{Predicted}}(t) - E_p^{\text{Input}}(0)}{E_p^{\text{Input}}(0)} \right|. \tag{7}$$

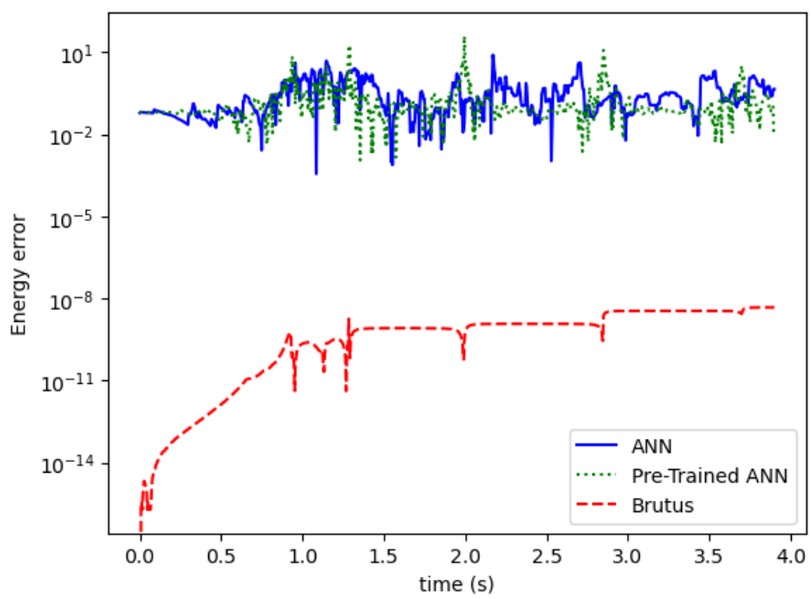

Figure 5: Relative energy error for a trajectory intialized with validation data. Blue solid lines denote the ANN with custom energy preserving loss function, dotted green denote the pre-trained ANN and the dashed red lines correspond to data generated by `Brutus`.

Apparent from the figure is that the error of both ANNs is roughly between 10^{-2} and 10^{1} most of the time. For the data generated by `Brutus` the error is initially below 10^{-14} and gradually increases to 10^{-8} at the end of the trajectory. This monotonous, gradual increase in errors with small magnitudes suggests that the energy discrepancy may stem from limitations in numerical precision rather than underlying issues with the `Brutus` model. Overall the figure is quite similar to the results presented by Breen et al. [3], the most notable difference being that the error for the ANN in the research paper is generally lower, albeit with a higher variance. This discrepancy is expected since Breen et al. uses a second neural network to predict velocities of the particles instead of differentiating the positions with regard to time. If velocities were predicted in a more elaborate way, for instance by using the analytical expression for acceleration in the three body problem

$$\mathbf{a}_i = -G(m_i m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} + m_i m_k \frac{\mathbf{r}_i - \mathbf{r}_k}{|\mathbf{r}_i - \mathbf{r}_k|^3}) \tag{8}$$

$$\mathbf{v}_i^t = \mathbf{v}_i^{t-1} + \mathbf{a}_i^{t-1}\Delta t \tag{9}$$

a result more akin to that in the research paper may have been obtained. A similar argument could be made as to why the neural network with the energy conservation constraint is not an apparent improvement to the neural network that only employs mean average error as loss function. If the velocities were to be calculated in another way, or perhaps predicted by another neural network, the network with custom loss function could possibly perform better. However it is clear from viewing Figure 6 that the network still gives somewhat satisfactory results.

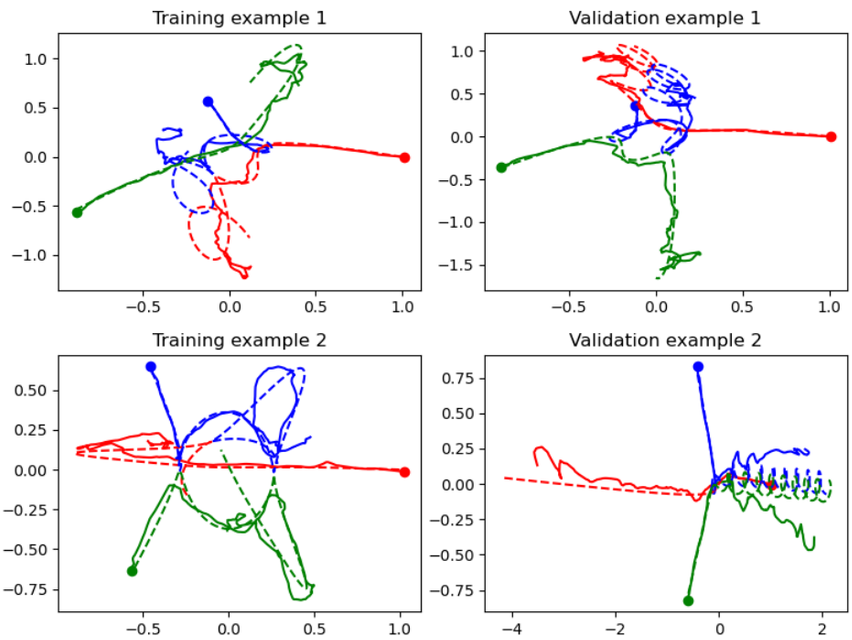

Figure 6: Two simulated trajectories initialized with data from the training set and two simulated trajectories initialized with data from the validation set. Solid lines are trajectories predicted by the neural network with custom loss function, dashed lines are trajectories generated by `Brutus`.

Apparent from Figure 6 is also that the NN with custom loss seem to perform about as well on training examples as validation examples, in similarity to the NN in the main task. This observation is further verified by analyzing Figure 7.

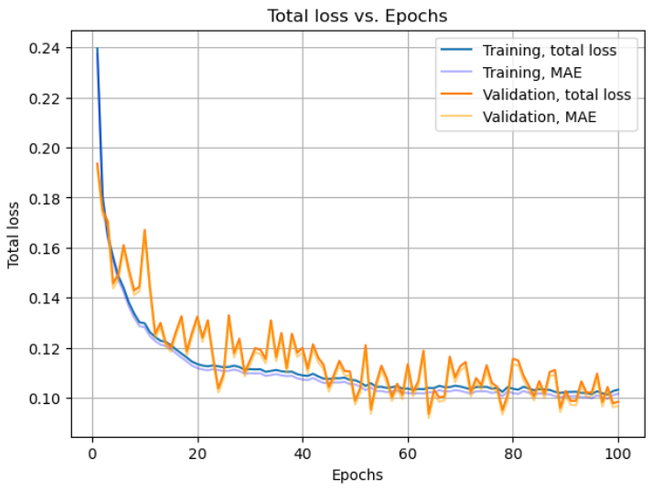

Figure 7: Total loss as a function of training epochs. The blue line represents the training loss and the orange line the validation loss. Faded lines represent the mean average error.

By employing the same arguments as in the discussion of Figure 1 and Figure 3 it can be concluded that the network is likely not suffering of any overfitting. The training and validation scores after 100 epochs for the NN with custom loss were 0.1033 and 0.0985 respectively. Conclusively the average prediction time for the NN with custom loss was $50.8\,\text{ms} \pm 4.67\,\text{ms}$.

5 Conclusion

In this project a comparison between our own artificial neural network and a pre-trained neural network generated by Breen et al. [3] has been made. These two neural networks have both been trained on data provided by `Brutus`, and this data has also been used to compare the predictive capabilites of the neural networks. A modification to the loss function of our network was made, motivated by knowledge of the physical problem. This however did not improve the performance of the network. The results presented in *Newton vs the machine: solving the chaotic three-body problem using deep neural networks* by Breen et al. have been reproduced, albeit to less success likely since our neural network were trained over substantially fewer epochs and its hyperparameters were not optimized. Optimizing the hyperparameters and increasing the number of epochs would have improved the predictive capabilities of our neural network. Exploring different architectures and activation functions could also have improved our neural networks.

It is highly valuable to predict the dynamics of a three-body system with an artificial neural network, despite its inherent inaccuracies. When comparing the prediction time of the neural networks (in the timescale of 100 ms) to the brute force numerical computation of `Brutus` (ranging from a few minutes to hours [3]), it is apparent that neural networks are useful methods when approaching the intricate numerical and analytical complexities inherent to the three-body problem.

6 References

References

[1] Newton I., *Philosophiae Naturalis Principia Mathematica*, London, England: Joseph Streater 1687

[2] Barrow-Green J., *Poincare and the Three-Body Problem*. Isis, Vol. 89, No. 2, pp. 345-346, Jun. 1998. [Online]. [Available from: https://www.jstor.org/stable/237789?x=-422.36411798441475&y=-62.92894984489732&w=1182.3641179844149&h=1197.9289498448975&index=0], (2023-10-23)

[3] Breen S.G, Foley C.N, Boekholt T., Portegies Zwart S. *Newton vs the machine: solving the chaotic three-body problem using deep neural networks*, [Internet], 2019. [Available from: https://arxiv.org/abs/1910.07291], (2023-10-23)

[4] Boekholt, T., Portegies Zwart, S. *On the reliability of N-body simulations.* Comput. Astrophys. Vol. 2, No. 2, Mar. 2015. [Online] [Available from: https://doi.org/10.1186/s40668-014-0005-3 (2023-10-24)

[5] Forrsén C. *Learning from data*, [Internet], 2023 [Available from: https://cforssen.gitlab.io/tif285-book/content/Intro/welcome.html], (2023-10-23)

[6] Forssén C. *Project 2: Newton vs the machine: solving the chaotic three-body problem using deep neural networks*, [Internet, Jupyter Notebook], 2023. [Available from: https://github.com/physics-chalmers/tif285/blob/master/doc/pub/Projects/Project2/TIF285_Project2.ipynb], (2023-10-23)