TIF285 Learning from data, Project 2: Newton vs the machine: solving the chaotic three-body problem using deep neural networks

Tomas Lundberg and Jonas H. Fritz*
Chalmers University of Technology
(Dated: December 8, 2021)

The three body problem has been one of the most elusive, yet most studied problems in classical mechanics. Despite its simplicity, no analytical solutions exist and its behavior is chaotic. Direct numerical calculations based on solving the equations of motion exist, however, due to the chaotic nature of the system, they are inherently limited and exact solution require an infinite amount of computation time. We reproduce a new approach taken by [1], who use an artificial neural network (ANN) to predict the three body trajectories from initial conditions. The network is trained using trajectories obtained via the Brutus algorithm[2].

Contents

I. Introduction

Newton's equations of motions are among the most well known equations in physics and the two body problem is central to any undergraduate course on classical mechanics. Surprisingly to students, adding another body to the problem makes it unsolvable analytically. In fact, the three body problem is often introduced in chaos theory as a simple system that can exhibit chaos. For specific special cases (e.g. when one of the masses is negligible), approximate solutions can be found. In general though, one most resort to numerical calculations, which have become more feasible with the improvement of computational power over the past decades. Nevertheless, due to the chaotic nature of the problem, an exact solution would require infinite computation time and calculated trajectories are unreliable. We reproduce results from [1], who take a novel approach in solving the three body problem. We use an artificial neural network (ANN) trained on precalculated trajectories using Brutus, to solve the three body problem from any initial conditions.

II. Background

The great interest in artificial neural networks arises from the universal approximation theorem, which states that a sufficiently large ANN can approximate any function to arbitrary accuracy. While we won't be concerned with the proof of this theorem, we will consider its applications. In this section we introduce how a single neuron works and build a full (dense) neural network from there. A major challenge in constructing neural networks is finding the correct parameters, called weights. This process is called training the neural network and will also be explained.

A. Single Neuron

Let us first consider a single neuron as illustrated in Fig. 1. It has inputs $\{x_1 \dots x_p\}$ and a bias w_0. The inputs are weighted with their respective weights $\{w_1 \dots w_p\}$. The activation is then calculated as

$$z = w_0 + \sum_{i=1}^{p} w_i x_i \,. \tag{1}$$

The output of the neuron is then given by a non-linear function of its activation $f(z)$. In this project we use the rectifier function

$$f(z) = z\theta(z) \tag{2}$$

* lutomas@student.chalmers.se, jonasher@student.chalmers.se

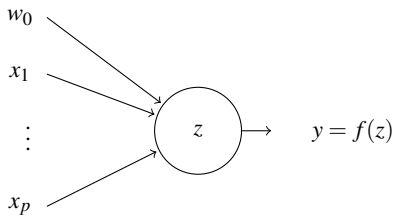

FIG. 1: Single neuron with inputs $\{x_1 \dots x_p\}$ and a bias w_0. These give the neurons activation z which results in an output $y = f(z)$.

with the heaviside-function $\theta(z)$, which is 0 below zero and 1 above. A neuron which uses this activation function is called rectified linear unit (ReLU).

B. Neural Networks

A dense, or fully connected, neural network as used in this project consists of multiple single neurons arranged in layers. The first layer is the input layer. It is connected to the first hidden layer, which is connected to the second hidden layer and so on, until the last hidden layer, which is connected to the output layer. A schematical illustartion of a one layer deep dense neural network is shown in 2. In principle the width of each hidden layer is arbitrary and independent from the other layers. In practice one usually chooses the same width w for each hidden layer. The width of in- and output layer must correspond to the shape of in- and output data respectively. The hidden layers consist of ReLUs as introduced in the previous section. The output $y_i^l = f(z_i^l)$ of the i-th node in the l-th layer is passed on as input the neurons in the $l+1$-th layer. The activation of the neuron is in turn given by the output from the previous neurons and their associated weights

$$z_i^l = w_{i,0}^l + \sum_{j=1}^{w} w_{i,j}^l y_j^{l-1} \,. \tag{3}$$

Finally, the output layer uses a linear activation function.

C. Training

The universal approximation theorem states that a neural network with arbitrary accuracy exists, yet it does not make a statement how its weights are found. To do so, we train the network on a given set of inputs and outputs, called the training set, which we know to be correct. We then introduce the concept of a cost function, which measures how far away our network is from approximating the function it is supposed to resemble. The task is then to minimize this cost function. There are both

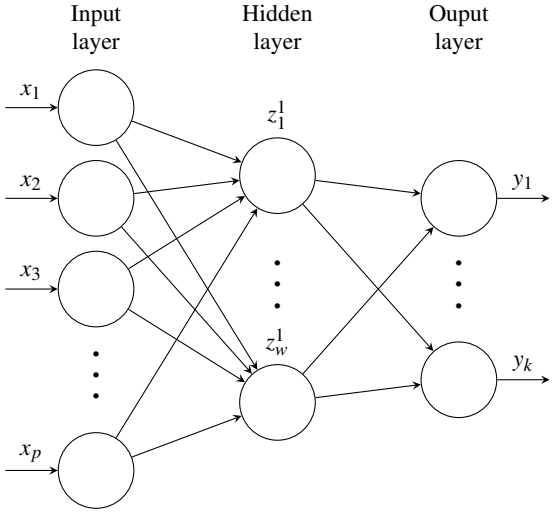

FIG. 2: Scheme of a dense artificial neural network (ANN). It consists of an input layer of width p, a hidden layer w and an output layer of width k. The input signal is given by $\{x_1 \dots x_p\}$ which cause the activation z_i^l in the hidden layer according to Eq. (3). The output of each neuron to the neurons in the next layer is given by a non-linear activation function $f(z)$. Finally, the hidden layer passes its output to the output layer.

different choices of cost function and minimization procedures. The mean absolute error (MAE) is a very common choice of cost function and for minimization various gradient descent methods are widely used. The MAE is given by

$$\mathrm{MAE}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \left|\mathbf{y}_i(\mathbf{x}_i|\mathbf{w}) - \mathbf{t}_i\right| \,, \tag{4}$$

where $\mathbf{y}_i(\mathbf{x}_i|\mathbf{w})$ is the output of the network for a given input $\mathbf{x}_i$ and its current weights $\mathbf{w}$. The output that the network is supposed to give, the target, is given by $\mathbf{t}_i$, and each index i corresponds to a sample of the training data set and we sum over all N samples of this set. It is important to emphasize that optimizing the weights crucially depends on the size and quality of the data set used and one is often constrained by the available training data.

Gradient descent methods work by calculating the cost-gradient at the current point in parameter space and adjusting the weights in the direction of steepest descent.

III. Method

Training artificial neural networks is a tedious and iterative endeavour. Therefore, we will go through our methodology and discuss some of the pitfalls we encountered and how we resolved them.

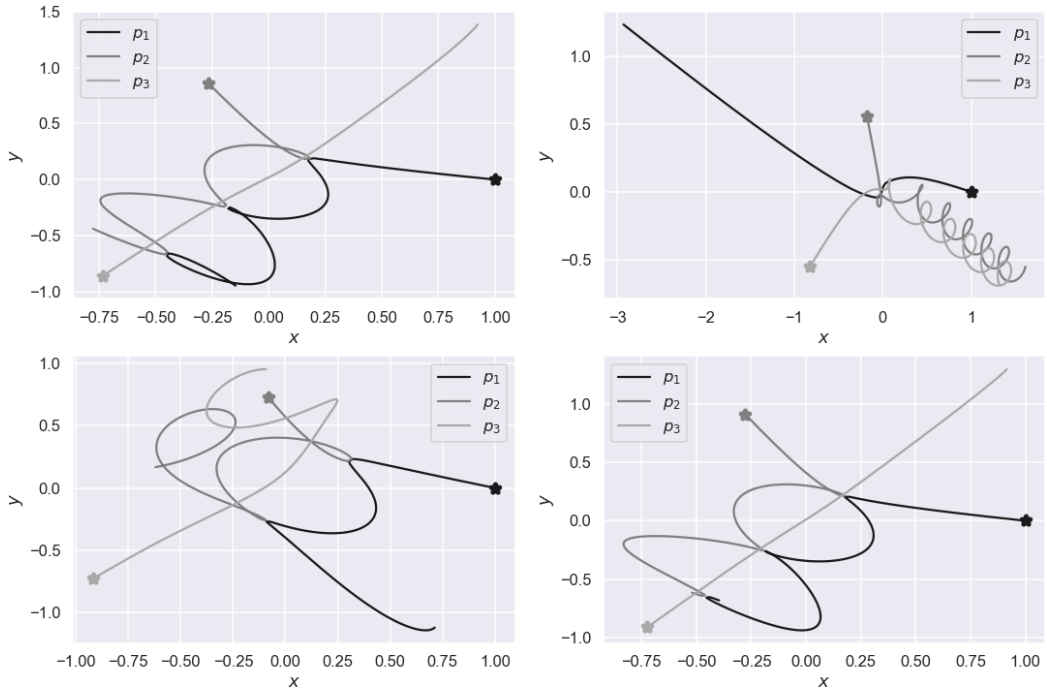

FIG. 3: Trajectories used for training the network. Each subfigure corresponds to a separate sample and the different colors correspond to the different particles. The initial positions are indicated by the star and by convention the particles are at rest at $t = 0$. The coordinate system is always chosen such that the first particle always starts at $(1,0)$ and the second particle in the $x \leq 0$ half-plane. The trajectories are generated by Brutus, a state of the art numerical calculator for integrating the equations of motion.

A. The data

The three body problem can be confined to two dimensions (x,y) by using the reference frame of the center of mass. Furthermore, without loss of generality, the initial position of particle one, p_1, can be set to $p_1(t=0) = (1,0)$ and the second particle, p_2, set to somewhere in the unit semi circle with $x \leq 0$. In the center of mass frame the position of the third particle is then given by $p_3(t) = -p_2(t) - p_2(t)$. By convention, particles are at rest at $t = 0$. Four example plots of trajectories can be seen in Fig. 3. This data was supplied by the authors of [1]. It consists of 9000 sample trajectories from the state-of-the-art numerical computer called Brutus. Each sample consists of a 1000×9 matrix with the first column corresponding to time t_i ranging from 0 to 3.9 time units. That is, each time step is $\Delta t = t_{i+1} - t_i = 3.9/1000$ time units. The next 4 columns are the x and y positions of p_1 and p_2 at time t_i. In the last four columns correspond to the velocities at a given time t_i for particles 1 and 2. Due to conservation of momentum, the velocity of the third particle is given by $v_{p_3} = -v_{p_1} - v_{p_2}$. The velocity data was not used in training, rather it was used to calculate the deviation from energy conservation for the Brutus trajectories, Fig. 8.

1377 of the 9000 samples had last rows which consisted of only zeros. By including these in the training data, the network gets biased to predict the particles to be at $p_1 = p_2 = p_3 = (0,0)$ at $t_i = 0$. For this reason we chose to remove these samples which reduced the data set to a total of 7623 trajectories.

B. Constructing the network

The goal of the ANN is to predict the position of p_1, p_2 and p_3 at time t. Since we are in the CM-frame, the ANN is only required to output the positions of two of the particles. Therefore, in line with [1], we defined the output to be four numbers; (x_1, y_1, x_2, y_2) at time index t_i. Due to the definition of having $p_1(t=0) = (1,0)$ all information for this output is found in the input $(t_i, x_2(t=0), y_2(t=0))$. To summarize, the ANN predicts (x_1, y_1, x_2, y_2) at time index t_i supplied with $(t_i, x_2(t=0), y_2(t=0))$. This defined our input and output layers to be of shapes 3 and 4 respectively.

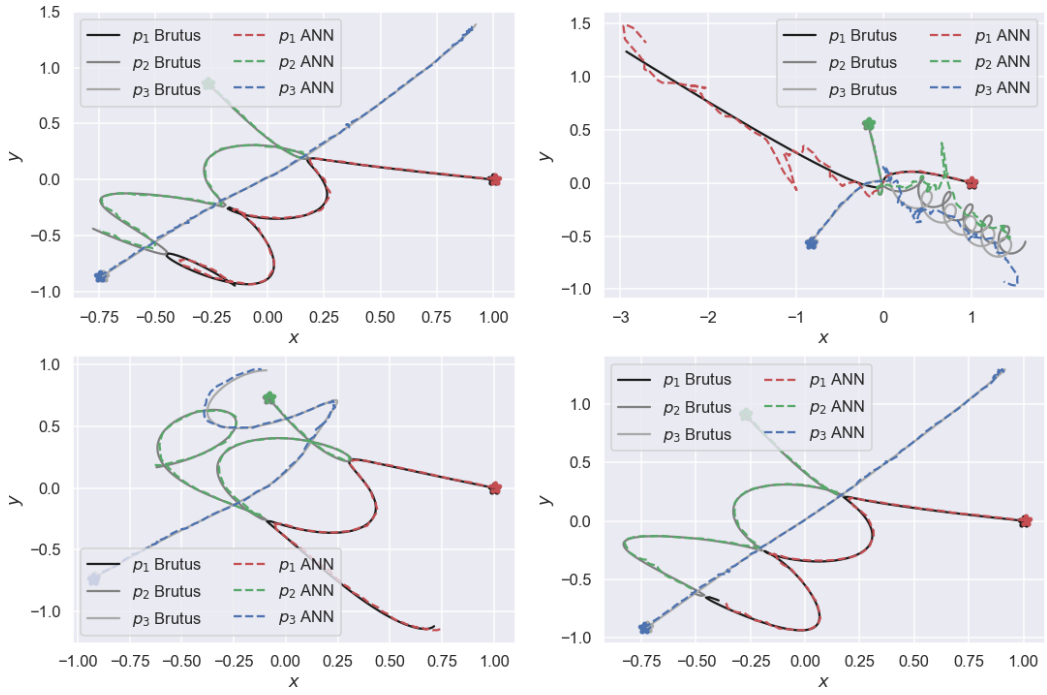

FIG. 4: Four arbitrarily chosen representative trajectories predicted by our neural network (dashed lines) overlayed with the actual trajectories computed by Brutus (full lines). The four subfigures each show a trajectory and each color corresponds to one particle. The stars indicate the starting position. Except for the top right figure, there is a very good agreement between the ANN predicted trajectories and the directly calculated ones. In the top right, the correct trajectory shows many close pass events, which the network has trouble predicting.

Moreover, we used the same network structure as in [1], that is, 10 hidden layers with 128 nodes each. In total this sums to 149,636 trainable parameters. All hidden layers had neurons with ReLU activation functions and Glorot normal weight initializers[3]. Finally, we used an ADAM optimizer, an extension of stochastic gradient descent, with a learning rate of 0.001 and $\beta_1 = \beta_2 = 0.5$ [4].

C. Training the network

Due to limited CPU resources we trained the network only 100 epochs compared to the 1000 epochs of [1]. This did however yield comparable results, as we will see further on, which can be understood upon inspection of Figure 3 in [1] where a clear display of diminishing returns is visible for epochs 100-1000.

The data set consisting of 7633 samples was made into an input and output data set of size $7627 \cdot 1000 \times 3$ and $7627 \cdot 1000 \times 4$ respectively. From this, a training and validation set was made with a random split of 90%-10%. We used a batch size of 5000 where the training data was shuffled during each epoch. The loss function that was minimized in training was the mean absolute error (MAE).

D. Energy conserving cost function

A fundamental principle in classic mechanics is the conservation of energy. The network we specified in the previous section minimizes only the MAE. With our prior knowledge that energy conservation must hold we can customize a new loss function that takes this into consideration.

In order to uphold energy conservation the network can only consider one trajectory at a time. To this end we respecified the input shape of the network to be 1000×3. The first column was $t_{i=0,1,\ldots,999}$ and column two and three held the initial position of particle two. The output now corresponded to the full 1000-step trajectories of particle one and two. Furthermore, the batch size was chosen to be one. Other than that, all settings were kept the same as before.

The new custom loss function consisted of an MAE term and an energy conserving term weighted by a parameter α. The energy conserving term was calculated

by first computing the total energy composed of the potential energy

$$E_{pot}(t_i) = -\sum_{j=1}^{3} \frac{1}{\sqrt{p_j^2(t_i)}}, \tag{5}$$

and kinetic energy

$$E_{kin}(t_i) = \sum_{j=}^{3} \frac{v_{p_j}^2(t_i)}{2}, \tag{6}$$

with $m = G = 1$. We then divided the total energy by the initial energy, i.e the potential energy at $t = 0$, and subtracted 1 to obtain the relative energy error of the trajectory. The term used in the custom cost function was the sum of this relative energy error. The velocities needed for the kinetic energy terms was computed for a trajectory p_j as

$$v_{p_j}(t_i) = \frac{p_j(t_i) - p_j(t_{i+1})}{\Delta t}. \tag{7}$$

An α that was too high lead to trajectories that were just stationary. An α that was too low did not influence the trajectories at all. Weighing this together lead us to use $\alpha = 0.01$.

IV. Results and discussion

In this section we first present the results from our ANN with MAE as the loss function. We show example trajectories, discuss computational cost and explore the chaotic behaviour of the three body problem. Lastly we look at our energy conserving network.

A. Evaluating the network

In Fig. 5 we can see how the MAE cost function is minimized over the course of the 100 training epochs. Most of the learning takes place in the first few epochs and then decreases more and more slowly. We note that the training and validation MAE is decreasing together which is a sign that we are not over training the network.

If we compare the learning curve with the corresponding one seen in Figure 3 of [1] we see that our results match very well. The MAE in Fig. 5 is still decreasing at epoch 100 so letting it train for a longer time would probably have lead to more learning. However, when inspecting Figure 3 of [1] it seems as if we could have best gained a factor of 2 over 900 more epochs.

In Fig. 4 we see four representative predicted trajectories from the ANN on top of the same trajectories computed by Brutus. For three of them (top left, bottom left and bottom right), the predicted trajectories match

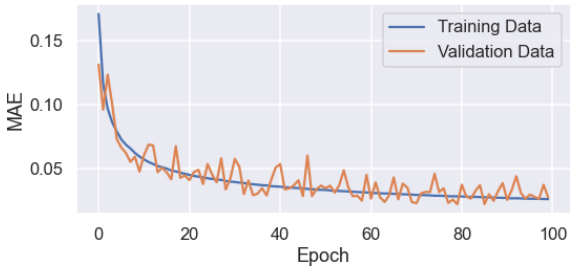

FIG. 5: Mean absolute error over the number of training epochs. Eavluated using both the training data (blue) and validation data (orange). There is a sharp decrease of loss in the beginning, where the network is still completely untrained. As training goes on the changes to the weights decrease in each step so there are diminishing returns in training run time. While the MAE of training data strictly decreases, the MAE of the validation data fluctuates, which highlights the importance of validation data.

almost perfectly with the Brutus ones. At the end of these trajectories, we can start noting a slight deviation. Furthermore, the predicted trajectories are somewhat less smooth.

However, for one of the trajectories in Fig. 4 (top right), the predicted trajectories deviate significantly. Upon inspection of the Brutus trajectories we see that there are many close encounters between particle two and three in this case. During these close encounters the chaotic behaviour of the system becomes especially important which explains the difficulty of predicting this example for the ANN. Using the ANN from the paper on the same example leads to better, but still far from perfect, trajectories. Perhaps in these difficult examples additional training could be most beneficial.

Brutus, or a similar numerical solver, will always have better accuracy in calculating trajectories for the three body problem. It is the gain in computation time which is the reason why we want to use an ANN. Calculating a trajectory with Brutus on a powerful computer takes on the order of minutes to hours. Predicting a similar trajectory with our ANN on a standard PC takes roughly 10 ms. That is, 5 to 6 orders of magnitude faster. On standardized machines the gain in computational cost is 8 orders of magnitude[1].

B. Chaotic behaviour

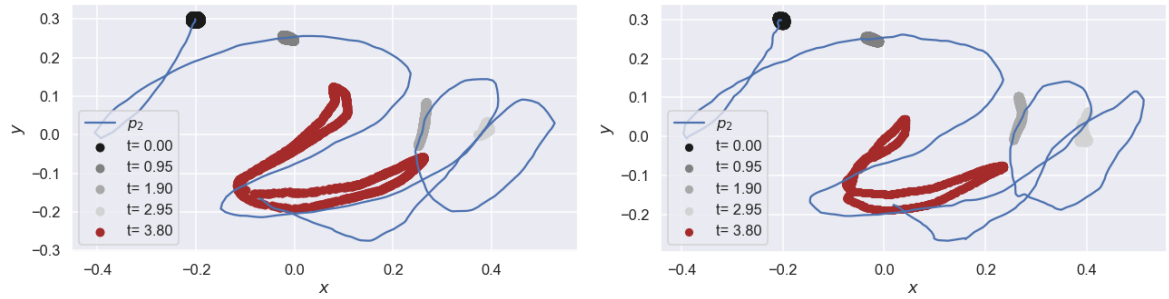

FIG. 6: Trajectories of an ensemble of the second particle starting on the circle centered at (-0.2,0.3) with radius 0.01 (black). Only the trajectory of the particle starting in the center is shown in blue. At regular time steps, the position of the other particles from the ensemble is shown. The left shows the calculations made by Brutus and the right by our neural network. Due to the chaotic nature of the system, slightly different initial positions lead to drastically different outcomes (red curve). This shows that our network correctly reproduces the chaotic nature of the system, despite not explicitly being trained on trajectories with very similar initial positions.

To explore the chaotic behaviour of the three body problem we make 1000 predictions with slightly perturbed initial conditions. We let p_2 start somewhere on the circle centered at (-0.2,0.3) with radius 0.01 and compared the positions of the different trajectories at four arbitrarily chosen times; $t \in \{0.95, 1.9, 2.95, 3.8\}$. We did this both with the ANN from the paper (left) and with our own ANN (right). The results are shown in Fig. 6.

We can first note that the unperturbed trajectory is smoother for the paper ANN. This is probably due to the fact that their network was trained for a longer time. Other than that the only noticeable difference is the slightly different positions at the last time, $t = 3.9$.

Common for both networks is that for $t \in \{0.95, 1.9, 2.95\}$ the positions are spread over a similar area as for $t = 0$. It is for $t = 3.8$ that the chaotic behaviour becomes prominent. The minor difference in initial positions leads to a wide range of different positions after 3.8 time units.

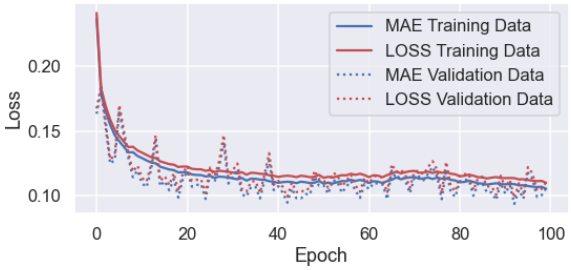

FIG. 7: MAE (blue) and custom loss function with energy conservation term (red) over training epoch for both training data (full line) and validation data (dotted line). Both run in parallel to each other, indicating that the energy term only enters as a constant offset, which isn't minimized during training. This indicates that the energy term needs to be multiplied by a larger numerical factor to have an effect on training.

C. Energy conserving cost function

The result of the training of the network with the implemented energy conserving loss function can be seen in Fig. 7. Both the MAE and the total loss is plotted for the training and validation data. The energy conserving term is the difference between the loss and MAE.

We can first note that the energy conserving term looks constant throughout the training. Therefore, it is probably the case that no energy conserving behaviour was learnt.

We can also note that in comparison with the results shown in Fig. 5 the MAE is almost one order of magnitude larger. Furthermore, the learning seem to plateau at around epoch 50; a behaviour that was not seen in Fig. 5. In fact, this happened even with $\alpha = 0$. This leads us to believe that it was the different input shape that caused the significant decrease in accuracy since this was the only difference in the structure of the two networks. This is probably also the reason why no energy conserving behaviour was learnt. Notwithstanding the fact that the input shape probably disturbs the learning, it might be the case that α is set to small since the order of the energy conserving term in the loss function is smaller than the fluctuations of the validation data, However, being able to reproduce the same results in *Fig.* 5 with $\alpha = 0$ should be solved first before tuning α.

In Fig. 8 the relative energy error, as defined in Section III D, is shown for a representative trajectory for Brutus, the paper ANN and our Energy conserving ANN. We see first that the error for Brutus is many orders of magnitude smaller than for the ANN's.

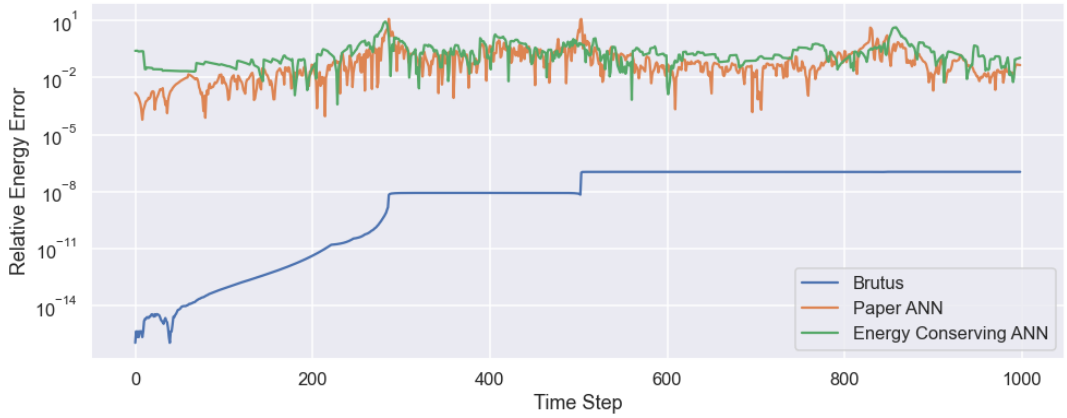

FIG. 8: The sum of kinetic and potential energy is calculated along the trajectory and related to the the initial energy given purely by potential energy. This is down for Brutus (blue), our ANN with an energy conservation term in its cost function (green) and the ANN used in[1]. Brutus still shows the best energy conservation by at least three orders of magnitude. The inclusion of the energy term in the cost function did not show significant improvement.

Since the Brutus trajectories are computed with equations that follow energy conservation this is expected.

Moreover, following the discussion in the previous paragraphs, it is to no surprise that our energy conserving ANN displays the same or larger error than for the paper ANN.

V. Conclusion

The three body problem is impossible to solve analytically and difficult to solve numerically. Precise calculations require a high amount of computation time. With the help of artificial neural networks the computation time can be significantly decreased, while still showing good agreement with results obtained through explicit calculations. We follow [1] in training a 10 layer, 128 neuron wide, dense neural network on $\sim$ 7500 trajectories pre-calculated by the Brutus algorithm [2]. We find that the network performs very well on most trajectories, but struggles with trajectories that feature many close encounters between the masses. The network is also able to reproduce the chaotic nature of the three body problem. However, a key feature in physical systems, energy conservation, could not be reproduced.

To include the fundamental principle of energy conservation in to our system we constructed a custom loss function which consisted of one MAE term and one term corresponding to the relative error in energy conservation. To this end, we redefined the input shape of the network in order to make sure that the network considered only one trajectory at a time. However, this structure of the network led to less accurate results, even with the energy conserving term set to 0. Therefore, we did not see any improvements in energy conservation. In order to improve this we would in a continued study try and implement the energy conservation with the same network structure as used in the original ANN.

The main advantage of using ANNs to predict three body trajectories comes from the great improvement in computational cost. A conventional numerical approach can offer more accuracy but at a cost which is orders of magnitude larger. We find that even on a regular PC the computation time is $10^5 - 10^6$ times smaller compared to Brutus.

In this report we have trained an ANN on a simplified version of the three body problem. We have considered only stationary initial conditions and particles with identical masses. For further study it would be of interest to see how well it behaves in a more general version of the problem with an additional degrees of freedom in the third spatial dimension. Since the same restrictions to solvability apply to the quantum mechanical version of the problem, namely that the solution of the two body problem is straight forward and the three body problem can only be approximated, it would be interesting to apply ANNs to the Helium atom to find energy eigenstates.

[1] P. G. Breen, C. N. Foley, T. Boekholt, and S. P. Zwart, Monthly Notices of the Royal Astronomical Society **494**, 2465 (2020).
[2] T. Boekholt and S. P. Zwart, Computational Astrophysics and Cosmology **2** (2015), 10.1186/s40668-014-0005-3.
[3] X. Glorot and Y. Bengio (JMLR Workshop and Conference Proceedings, Chia Laguna Resort, Sardinia, Italy, 2010) pp. 249–256.
[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," (2014), arXiv:1412.6980 [cs.LG].