Project 2 - Learning from Data

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

Mathias Arvidsson CID: matharv, Carl Strandby CID: carstran

Program: Teknisk Fysik.

Course: Learning from Data, [Tif285], Chalmers, Fall 2022.

Contents

1 Introduction and Nature of the problem.

An artificial neural network (ANN) is a tool for solving computationally demanding problems which has become popular in recent years. One feature contributing to its popularity is the ANN's ability to optimize a computational system to a certain kind of process, without specifying the rules of said process. The experiment presented in this report is an example of this, where an ANN learns to predict trajectories of three bodies under the influence of each others gravity pull without knowing anything about Newton's laws. This prediction is trained on data and then compared to a numerical solution calculated by Brutus integrator developed by [**Tjoecholt Portegeis Zwart**], and a pre-trained ANN supplied by Breen et al[**Breen_2020**].

The main features that define a given ANN is it's architecture, activation rule and learning algorithm [**forssen_2022**]. A simple and commonly used architecture consists of an input layer specifying the input to the network, followed by a sequence of fully connected layers and lastly an output layer specifying the output. In each layer is a number of nodes, where each node is connected to all nodes in the previous layer. The node takes it's input, commonly tensors, along all of it's connections, transforms them, usually with a tensor product, and then performs an activation function on them. This activation rule determines the nodes response to it's input before passing the information to the next layer. Furthermore, each connection between nodes is associated with a weight. During a so called training period the network is allowed to adjust these weights according to a learning algorithm. A common type of learning algorithm compares the difference between the networks output and a target output and uses this to adjust the weights of the network.

In this experiment an ANN is utilized to solve the famous three-body-problem. The three bodies, starting at rest are under the influence of each-others gravitational pull, which results in complex interactions making the three body problem a classically chaotic system, that is, a system in which a small difference in input variables can have a big impact on the outcome. This also means that numerical solutions are computationally difficult, since small approximations made by a computer in order to store variables in it's memory with a given number of decimals can result in large deviations. A recently developed numerical method for solving any N-body problem is the Brutus numerical integrator developed by T.Boekholt et al. In the paper Newton vs the Machine, Brutus is used to generate solutions to starting configurations for the three-body problem. Such solutions can take between a few minutes, up to many hours to compute [**Breen_2020**]. Generated numerical solutions are then used as training data for an ANN, and their solutions are compared. The purpose of this experiment is to create and train an ANN to replicate results of the ANN developed by Breen et al by using a similar architecture and training on data from Brutus [**Breen_2020**].

2 Description of methods used

The ANN was created in python using the open-source machine learning package ”TensorFlow”, developed by the Google Deep-mind team, a framework with descriptions and functions for implementation of machine learning systems. For the purpose of this task, the architecture was designed to be the same as in Breen et al. First an input layer with 3 nodes, followed by 10 fully connected layers, with 128 nodes in each, and an lastly an output layer with 4 nodes. This amounts to about 150,000 trainable parameters, or weights. The activation function used in the fully connected layers was of the type ReLu (Rectified Linear Unit), which implements the function

$$a(x) = \text{argmax}(0, x). \tag{1}$$

The last layer used the activation function Lu (Linear unit), simply defined as $a(x) = x$. The learning algorithm was based on a Mean Absolute Error, abbreviated MAE, with the tensorflow optimizer Adam [**Adam_TF**] set with a learning rate of 0.001 and moments 0.5.

The training data was based on numerically calculated three-body trajectories calculated with Brutus by T.Boekholt et al. The complete dataset contained 9,000 trajectories, which was split into batches of 100 trajectories each, and split into 90% test-data and 10% training-data. This training data contained input data to the ANN which consisted to a time series from zero to 3.9 seconds and an initial position for one of the particles, labeled the second particle. The initial position of the first particle was always set to $(1, 0)$ in the target data and thus the network would learn over time to assume this to be the position of this particle. The coordinate-plane used to model the trajectories were constructed as a center-of-mass-frame, meaning that the positions for three particles of equal mass fulfills the two conditions $x_1 + x_2 + x_3 = 0$, and $y_1 + y_2 + y_3 = 0$. Therefore the position of the third particle can always be inferred from the first two. Target data for the ANN to train models against contained the particle trajectories for the first and the second particles over the time series from zero to 3.9 seconds. Some trajectories was deemed "faulty" and were thus removed from the dataset. These were trajectories in which the particles "stuck" together in the center, identified by the last coordinates of the time series being zero for both the first and second particle.

A total of three models was trained using the ANN. First a smaller model utilising a separate smaller architecture. The results of this model is not included in this paper, since it's only function was to verify the functionality of the ANN. Secondly, a model labeled Big_model was trained using the previously described architecture. This model was trained for 420 epochs, each epoch corresponding to one passing of the training data. The third model was trained for the optional task, with the same design as the big model, but with an altered cost function, thus this model was labeled Big_model_With_Custom_Loss. The alteration was to include an extra term in addition to the MAE, which included a weighted difference between the initial potential energy and the total (kinetic and potential) energy at each timestep, thus punishing trajectories which did not conserve the energy of the three particle system.

The predictions of the two trained models was compared to a pre-trained ANN-model, labeled Pre_Trained_Model, developed by Breen et al. This model was trained using 10,000 epochs with a batch size of 5000 instances [**Breen_2020**] corresponding to five trajectories.

3 Results

The following two sections describes the results of the experiment. Section 3.1 focuses on the model Big_Model and Section 3.2 focuses on the model Big_model_With_Custom_Loss

3.1 Basic

The complexity of the Big model was the same as the model from Breen with 149636 parameters [**Breen_2020**]. Figure 1 shows the Mean Absolute Error for the 420 training epochs. It can be seen that the MAE starts at about 0.19 and over time drops to roughly 0.05. At the end of the training the slope is seen to be slightly negative.

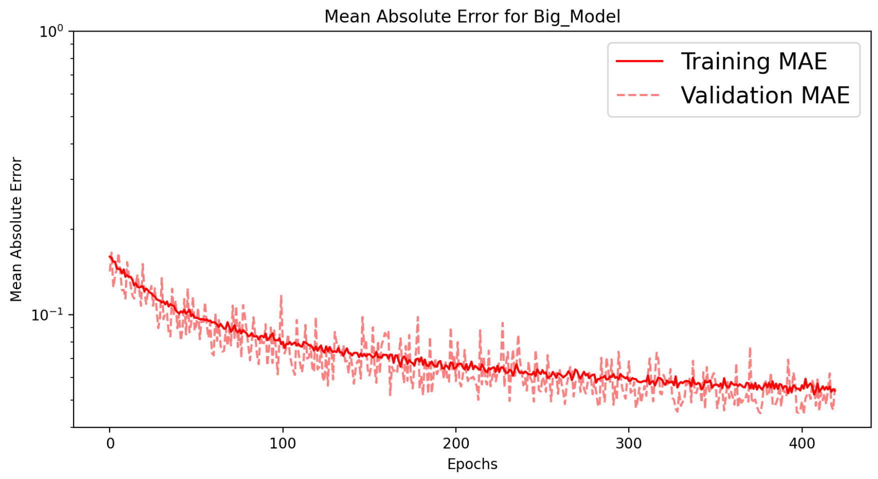


Figure 1: Mean Absolute Error for each of the 420 epochs for the trained model. MAE is shown for both the training data (solid red line), and the validation data(dashed semi-transparent line). The y-axis is in logarithmic scale. The slope of the curve at the end of the training can be seen to be slightly negative.

Figure 2 shows two different predicted trajectories, the left column trajectories predicted with Big_Model and the right column with Pre_trained_model. The shapes of the trajectories seem to be similar, however the lines representing the particle trajectories of the Big_Model seem to be rougher. These can be compared to the "True Trajectories", numerically calculated with Brutus, which are seen in figures 3a and 3b. The computational time of performing a prediction was between 0.03 - 0.05 seconds with a standard at home computer. In comparison, the time needed by Brutus to compute a prediction numerically ranged from minutes to hours[**Breen_2020**].

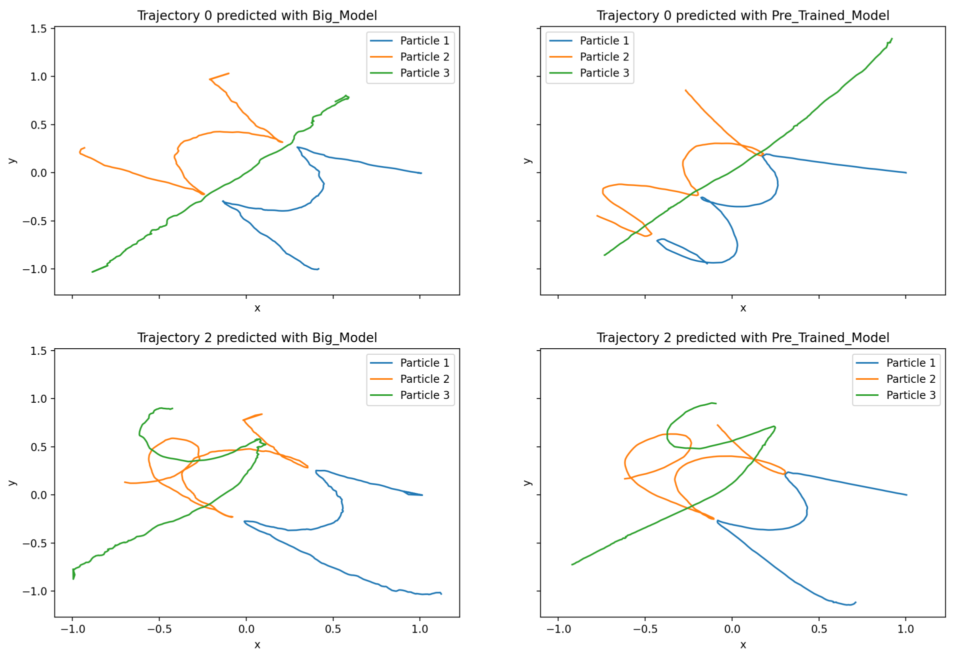
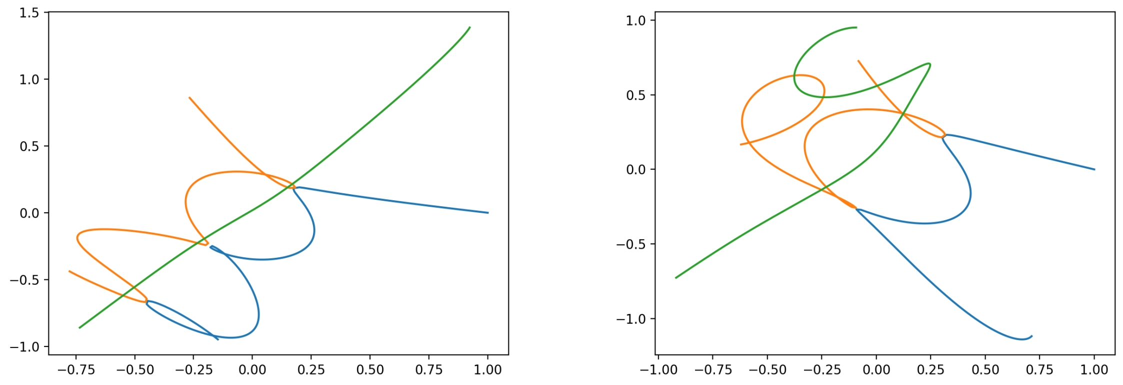


Figure 2: Two different predicted trajectories from our trained model (left column) and the Pre_Trained_Model(right column).



(a) Trajectory 0, numerically calculated with Brutus. Corresponds to the true trajectory for the top row of figure 2.

(b) Trajectory 2, numerically calculated with Brutus. Corresponds to the true trajectory for the bottom row of figure 2.

Figure 3: The two true trajectories, numerically calculated with Brutus.

Lastly, the chaotic nature of the three body problem was explored. This was done by adding a small weighted random offset to the initial position of the second particle which resulted in a notable difference. Three resulting trajectories of which is shown in figure 4. It is important to note that these figures where chosen because the difference in trajectories is notable, yet they display some of the characteristics of the unaltered trajectory.

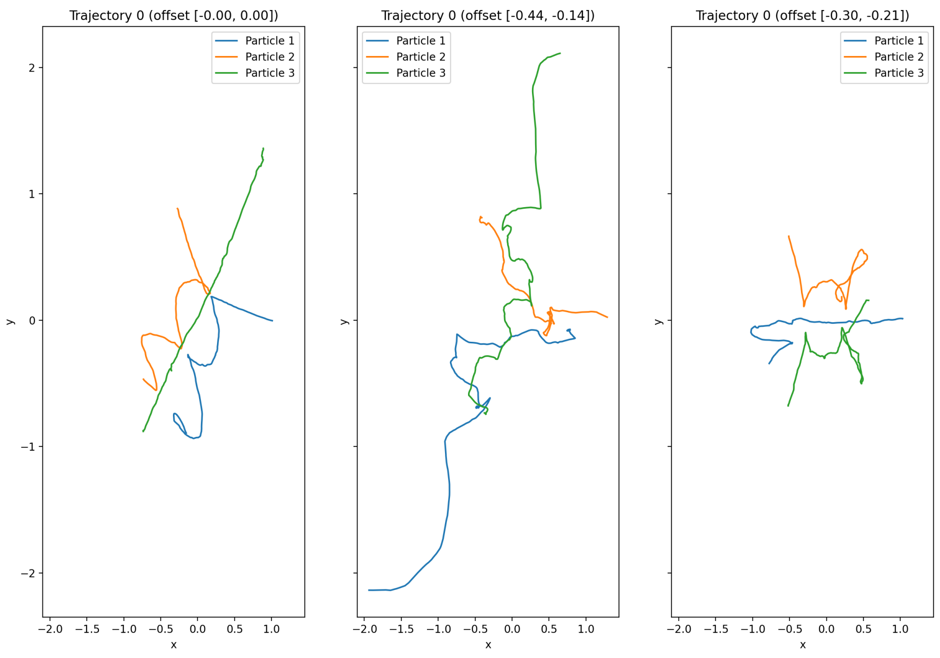


Figure 4: Chaotic behaviour of 3 body problem. The initial position of the second particle was offset by adding a small weighted random number. The figure to the left shows a trajectory with no offset whereas the middle and the right shows particle trajectories with an offset. An important note is that these figures where chosen because the difference in trajectories is notable, yet they display some of the characteristics of the unaltered trajectory.

3.2 Optional

The model with the custom loss function, punishing predictions that did not conserve the initial energy was trained for 420 epochs. Figure 5 shows a comparison with between predictions from Big_model_With_Custom_Loss, shown in the left column, with ones from Big_Model and Pre_Trained_Model in the middle and right column respectively.

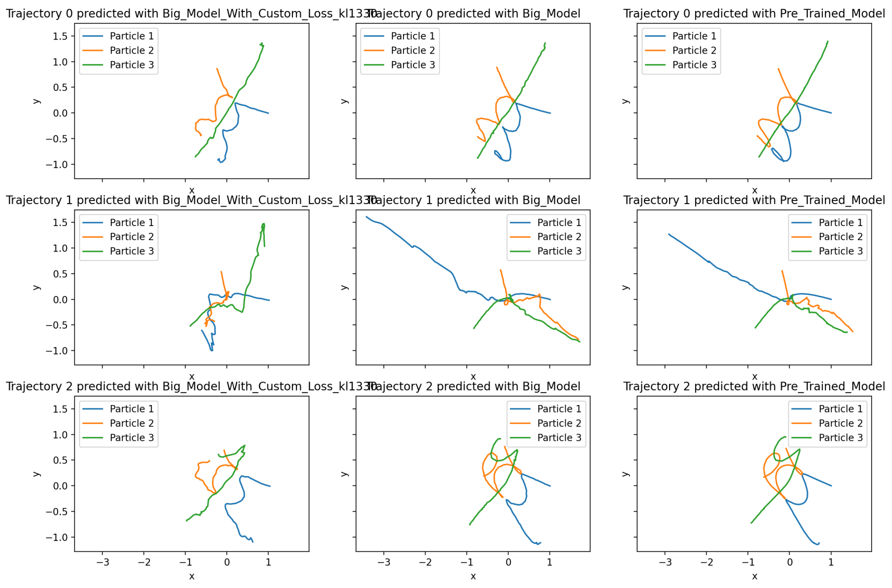


Figure 5: Three predicted trajectories using three different ANN models. The left most column shows the predictions from the model using the custom cost function, while the middle column shows the predictions of Big_model, which utilized the MAE cost function, and the right-most column shows the predictions of the pre-trained model.

Figure 6 shows a comparison of energy conservation of the three models. It can be seen that the Model utilising the custom loss function produced a more varying energy loss, compared the predictions of Brutus and Big_Model. Energy loss for Brutus is taken to be simply the difference between the kinetic energy provided in the data and the kinetic energy of it's predicted trajectory, since data from Brutus doesn't contain predicted initial positions from which an initial potential energy can be calculated. For the Big_Model and Big_model_With_Custom_Loss the energy loss was defined as before. Furthermore, figure 6 also shows a few singularities, this was expected for points in which the particles positions get close to each other, and an inherent feature of the supplied trajectories.

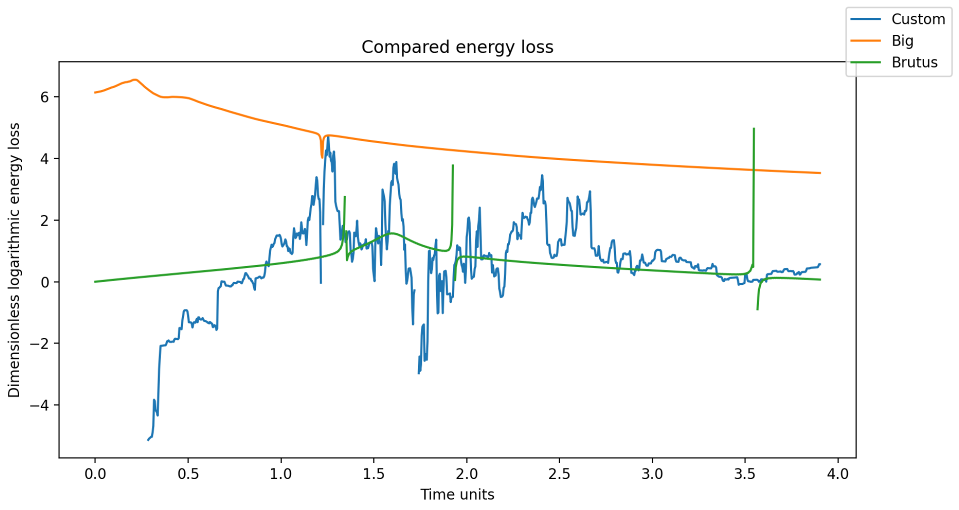


Figure 6: Computed energy loss for three different models over one full trajectory (approximately 3.9 time units). The scale on the y-axis is that of the natural logarithm. Importantly, the energy loss calculated with Brutus is simply the difference between Brutus calculated kinetic energy and kinetic energy calculated from positions from Brutus.

4 Discussion

Overall, the resulting predictions of Big_Model were relatively similar, but rougher than those calculated numerically with Brutus, or those predicted by the Pre_Trained_Model from Breen et al. However, the length of training of the two models was significantly different with Big_Model being trained on 420 epochs and Pre_Trained_Model on 10,000. It would therefore be interesting to increase the training time to explore if predictions would become increasingly similar to those of Pre_Trained_Model.

One interesting limitation inherent in the approach of solving computationally difficult problems with an ANN is that in order to train a model, one first requires training data, and this will often require rigorous experiments and numerical solutions to be made beforehand. There is also a risk that errors or undetected biases present during gathering of training data is present in predictions made with the artificial neural network. If one has adequate training data, the results of this experiment seem to indicate that the computation time for solving the three body problem with ANN compared to Brutus significantly favours the ANN-approach. The accuracy of the ANN predictions will however still remain dependent on the accuracy of the numerical solutions. This interplay between numerical solutions and ANN-predictions could perhaps be an interesting area of future research, where perhaps an ANN can be used to identify areas of interest, and guide the numerical solver ”in the right direction”.

Another common problem in machine learning is over-fitting, where the system is trained more and more, gaining accuracy for the training data, but sacrificing accuracy of predictions on validation data for accuracy in predictions for the training data. One scenario where over fitting can occur is when the training data is always processed in the same order. The system then learns not only from the data but from the sequence of data, making it unstable for different datasets. This can be avoided by shuffeling data between epochs. Another is during long periods of training where an ANN prediction could expect to follow an exact trajectory and even a slight deviation could be interpreted by the ANN-model as being in completely new territory. One measure made during this experiment to avoid this was to save the model repeatedly during training, and only when the loss function on the validation data was reduced. Figure 1 showing the Mean Absolute Error from the training of Big_Model, does however indicate that the predictions of Big_Model could have improved further with more training, given the negative slope of the validation curve.

In conclusion, given the aim of reproducing data from Breen et al and exploring the approach of using an artificial neural network to solve computationally demanding problems, the experiment can be said to have succeeded with indicating the utility of solving computationally demanding problems with an ANN. Results were however rough and less accurate than those produced by Breen et al, and it is difficult to guess if this difference would entirely go away from a longer training.