Dicing up Newton: Exploring Deep Learning Solutions to the Chaotic Three-Body Problem with Nonlinear Data Transformations

Arthur Campello (arthurcc@stanford.edu)

November 17, 2020

1 Introduction

Many important problems in physics, chemistry, biology, and economics involve "N-body" dynamics, where large number of identical units mutually interact to present larger-scale behaviors. Effectively simulating these dynamics can elucidate complex mysteries ranging from income inequality in ride sharing [1] to the possibility of fusion reactors to solve the energy crisis [2]. Unfortunately, these systems invariably exhibit nonlinear and chaotic behaviors that obscure analytical solutions to problems. Fully simulating N-body dynamics allows researchers to better explore their properties, but becomes unfeasible for small time steps, large time scales, many particles, and various iterations.

Thankfully, recent deep learning techniques have allowed simulated input/output N-body data to serve as training data for N-body dynamics models with near instant results. This is made possible by the nonlinear nature of most neural-network architectures, which allows them to "tease out" nonlinear dynamics. These could eventually escalate to algorithms that take in rules governing particle interactions and estimate the dynamics of N-body systems of these particles without the need for problem-specific training.

A common toy model for chaotic N-body systems is the deceptively challenging "Three-body problem" proposed by Sir Isaac Newton. This is a problem where three objects with masses m_1, m_2, and m_3 at respective positions $\vec{r}_1$, $\vec{r}_2$, and $\vec{r}_3$ are mutually gravitationally attracted according to the following formula:

$$\vec{F}_{ij} = \frac{g m_i m_j}{(\vec{r}_i - \vec{r}_j)^2}, \tag{1}$$

where i and j are labels for different masses and g is an arbitrary constant. Without loss of generality, one can bound the masses by translating their positions to the non-accelerating center-of-mass frame and fixing them to a two-dimensional plane. Figure 1 shows an example trajectory set of three masses. The mutual coupling of the masses ensures the behavior is chaotic.

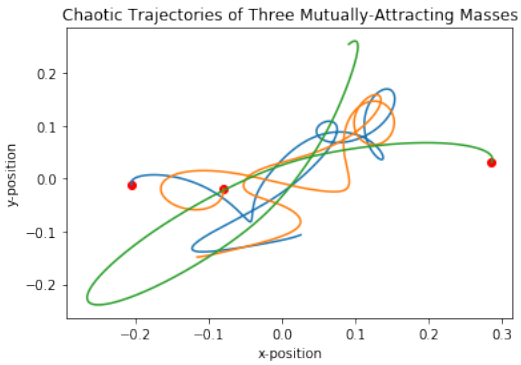

Figure 1: An example trajectory of three gravitationally-attracted bodies showing chaos

Astoundingly, the first academic paper to explore the performance of neural networks in tackling this problem emerged only this year [3]. While intriguing, the paper explores only a single neural network type with a very limited parameter search protocol and fails to make performance comparisons between deep learning approaches from which one can draw insights.

This report takes a first step towards comparative evaluations of models for solving the Three body problem. Guided by the nonlinear nature of the problem, it will specifically focus on testing

whether nonlinear transformations of data can offset the need for nonlinear functions embedded in neural networks and streamline learning. The nonlinear transformation examined involves "dicing up" scalar values into vectors of binary values using the following quasi-bijective function $f : [-1, 1] \to \{0, 1\}^n$ that transforms scalars x into vectors $\vec{v}$ with elements of values 0 or 1:

$$v_k = \text{nint}\left(2^{(k-2)} \bmod \left(x + 1, 2^{(2-k)}\right)\right), \quad (2)$$

$$x = \sum_{k=1}^{n} \frac{1}{2^{(k-1)}}(v_k - 1), \quad (3)$$

where $\text{nint}(x)$ is the "nint" function that rounds x to the nearest integer. The schematic in Figure 2 more intuitively represents this transformation and makes clear its nonlinear nature.

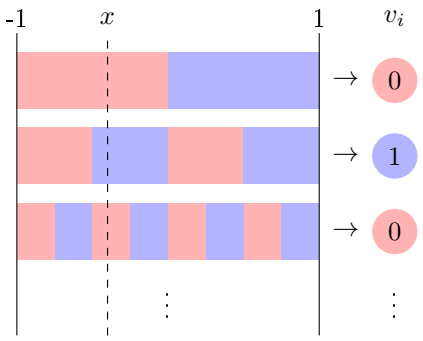

Figure 2: Schematic showing transformation of x to v performed equation 2

While this transformation increases the number of variables needed to represent the state of the system, it introduces potentially useful nonlinearities into the model and broadens the list of activation functions one can use between layers.

The possible inputs and outputs explored across models are either "scalar" or "binary" according to the following definition: "scalar" data comprises of length 12 vectors where the first, second, and third groups of four values correspond to each of the three masses and, within each group, the first two values correspond to x and y positions and the second two to x and y velocities; "binary" data mimics this structure with each scalar value replaced by n consecutive binary values according to equation 2.

The report centers on a detailed an extensive comparison between learning models of different complexities that take in either scalar values or binary-transformed ones as inputs and/or outputs. This will test whether one may reject the hypothesis that this particular transformation plays a role in enhancing this model and will point to a conclusion on whether such approaches may generally be useful for understanding N-body systems.

This comparison is followed by an investigation into whether binary transformations impact the performance of Residual Neural Networks (ResNets) on this problem. Given that virtually no precedent exists for whether skip connections serve well in N-body settings, either outcome can help inform whether to consider ResNets for similar applications.

The results of these deep learning experiments likely generalize to other chaotic N-body problems and have the potential to accelerate the development of deep learning solutions to difficult questions in the most challenging corners of various fields.

2 Related Work

The piece of literature most relevant to this work is an article published less than one year before this writing titled "Newton and the Machine" [3]. This work claims that neural networks serve well to solve the three body problem and demonstrates an effective model, but does not perform comparative experiments like this report does. By obscuring the iterative process of arriving at the authors' model, the work limits itself to a single successful case study rather than a preliminary guide for the development of future N-body deep learning models. The lack of advancement in this area of deep learning arises from the fact that simulated scenarios are often too specific for the development of trained models. This means the simulation work required to train models defeats the purpose of using the model.

3 Dataset

All data for this project arises from finite difference simulations of the three-body dynamics described in the introduction. Specifically, mass positions are initialized randomly within the range $(-1, 1)$ such that their center of mass is at the

origin. Velocities of these masses are also initialized randomly so that the center of mass remains stationary. The introduction explains how these restrictions preserve generality. These initialized position and momenta quantities are placed in a vector of length 12 (two quantities for three masses in two dimensions) and this is the input. The corresponding output is obtained by performing a finite-difference simulation according to equation 1 with $g = 1$, $dt = 0.001$ along 500 time steps. "data_generation_with_FDS.py" performs these calculations and logs initial and final positions and velocities of the three masses (in the $x - y$ plane) as rows of a CSV file.

The simulated nature of the data means that the dataset may be of any desired length, making iterative regularization unnecessary as any overfitting can be overcome by training with more data. Furthermore, all data is statistically identical, meaning no data mismatch is possible.

The data obtained is then augmented using the interchangeability the effective "labels" of the masses. Since three masses correspond to six labeling permutations, the data may be augmented six-fold. This augmentation is performed by functions in "data_augmentation_and_transformation.py"

For all models trained, 20000 exampled were obtained by simulations which were then augmented to 120000 examples. 100000 of these were used as the training set, 10000 as the dev set, and 10000 as the test set.

4 Methods

Across all models explored in this project, the mean squared error (MSE) loss function

$$\mathcal{L}(y, \hat{y}) = \sum_i (\hat{y}_i - y_i)^2 \tag{4}$$

was used along with the Adam optimization method [4] with minibatches of size 20 over 100 epochs. Regularization considerations were limited due to the large amount of data and the comparative focus of this project.

4.1 Neural Network Comparisons

The first comparison approach involved only linear and logistic regressions. Specifically, the following four regression models – where X and Y denote inputs and outputs and s and b subscripts denote scalar and binary – were evaluated on the dataset:

$$\hat{Y}_s(X_s) = WX_s + \vec{b}, \tag{M1}$$

$$\hat{Y}_s(X_b) = W(X_b - 1/2) + \vec{b}, \tag{M2}$$

$$\hat{Y}_b(X_s) = \sigma\left(WX_s + \vec{b}\right), \tag{M3}$$

$$\hat{Y}_b(X_b) = \sigma\left[W(X_b - 1/2) + \vec{b}\right]. \tag{M4}$$

The subtraction of $1/2$ from input binary data serves to "balance" the $0, 1$ data so that it averages zero. The data types used in these models dictates the following shapes for trained parameters W and $\vec{b}$ as functions of n: $\vec{b} \in \mathbb{R}^{12\times 1}$ for Y_s models and $\mathbb{R}^{12n\times 1}$ for Y_b models; $W \in \mathbb{R}^{12\times 12}$, $\mathbb{R}^{12n\times 12n}$, $\mathbb{R}^{12\times 12n}$, $\mathbb{R}^{12n\times 12}$, respectively to the models in the order presented. Results for these are reported for $n = 10$; above $n = 8$ this parameter has virtually no effect on the performance.

Following this, similar performance comparisons were made using deeper neural networks of five layers. The first of these models has scalar inputs and outputs and is defined by the equation set

$$A_1 = \text{ReLu}(W_1 X_s + \vec{b}_1),$$

$$A_i = \text{ReLu}(W_i A_{i-1} + \vec{b}_i), \quad i = 2, 3, 4$$

$$\hat{Y}_s(X_s) = W_5 A_4 + \vec{b}_5. \tag{M5}$$

with the optimized parameters being of size $W_i \in \mathbb{R}^{12\times 12}$ and $\vec{b}_i \in \mathbb{R}^{12\times 1}$ for $i = 1, 2, 3, 4, 5$. The second model has binary inputs and outputs and is defined by

$$A_1 = \tanh[W_1(X_b - 1/2) + \vec{b}_1],$$

$$A_i = \tanh(W_i A_{i-1} + \vec{b}_i), \quad i = 2, 3, 4$$

$$\hat{Y}_b(X_b) = \sigma(W_5 A_4 + \vec{b}_5). \tag{M6}$$

Now the parameters are of size $W_1 \in \mathbb{R}^{12\times 12n}$, $W_{2,3,4} \in \mathbb{R}^{12\times 12}$, $W_5 \in \mathbb{R}^{12n\times 12}$, $\vec{b}_{1,2,3,4} \in \mathbb{R}^{12\times 1}$, and $\vec{b}_5 \in \mathbb{R}^{12n\times 1}$. Again $n = 10$ was used for the final analysis. With foresight based on results from the regression comparisons, a final deep model was examined, defined by the equation set:

$$A_1 = \tanh[W_1(X_b - 1/2) + \vec{b}_1],$$

$$A_2 = \tanh(W_2 A_1 + \vec{b}_2)$$

$$A_i = \text{ReLu}(W_i A_{i-1} + \vec{b}_i), \quad i = 3, 4$$

$$\hat{Y}_s(X_b) = W_5 A_4 + \vec{b}_5. \qquad \text{(M7)}$$

The parameter sizes for this final model are $W_1 \in \mathbb{R}^{12\times 12n}$, $W_{2,3,4,5} \in \mathbb{R}^{12\times 12}$, and $\vec{b}_{1,2,3,4,5} \in \mathbb{R}^{12\times 1}$. As before, only $n = 10$ results are reported.

For all iterations of the regression and deep neural network models, checks are performed for overfitting (despite the large dataset size) and some are reported. The test MSE and its standard deviation values for all models are reported and used as performance metrics; lowest MSE is the primary metric and lower standard deviations break ties if needed.

4.2 ResNet Effect Comparisons

The effects of adding skip connections across nodes (i.e. transforming the neural network into a ResNet) are shown for models M5-M7. To guarantee effective comparative evaluations, the same skip connection structure was added to all models, meaning the "skipped" layers remained unchanged across models. Specifically, for M5, A_2 is replaced as

$$A_2 \to \text{ReLu}(W_2 A_2 + \vec{b}_2 + W_1 X_S + \vec{b}_1)$$

and, for M6 and M7, as

$$A_2 \to \tanh(W_2 A_2 + \vec{b}_2 + W_1 X_S + \vec{b}_1).$$

These new models are called M5R-M7R (each corresponding corresponding to the model it modifies). This stage of the evaluations involves a comparison between each of the three models with and without a skipped connection using the same metric as before. During the investigation process, several other skip connection architectures were explored but nearly all presented the same effects as reported in the next section.

5 Results and Analysis

5.1 Neural Network Comparisons

I first report the results of the first four models from the previous section: M1-M4. Figure 3 shows the convergence of the training and dev error rates for the M1 over epochs. The proximity of the two shows that the model is not overfitting the data and this was similarly verified for the other three regression models.

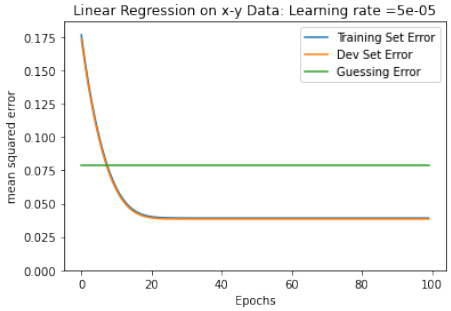

Figure 3: Training and Dev MSE for model M1 over 100 epochs, with 'guessing error' shown

Table 1 below shows the test set MSE and error standard deviation across the four regression models and the number of parameters in each (with $n = 10$).

Model	Test Set Error	Parameters
M1: $\hat{Y}_s(X_s)$	0.037748 $\pm$0.0336	156
M2: $\hat{Y}_s(X_b)$	0.036856 $\pm$0.0323	1452
M3: $\hat{Y}_b(X_s)$	0.037750 $\pm$0.0336	1560
M4: $\hat{Y}_b(X_b)$	0.036872 $\pm$0.0326	14520

Table 1: Test MSE values for regression models with model parameter numbers

Although models M2 and M4 perform the best among the four models, both have large parameter numbers and only perform marginally better than the standard for the group. This result is especially surprising given that model M4 has nearly 100 times more parameters that M1 and has shown not to suffer from overfitting. Another interesting comparison is that between M2 and M3 as M2 performs better despite having a comparable number of parameters (but fewer) and both models make use of both datatypes. This indicates that this family of regression models may perform most efficiently when using binary data as inputs and scalar values as outputs.

As with the regression model, a training convergence plot is shown in Figure 4 for M5, the first model in the deep neural network class. This plot and similar analyses for models M6-M7, M5R-M7R again rule out overfitting concerns.

Figure 4: Training and Dev MSE for model M5 over 100 epochs, with 'guessing error' shown

Table 2 below shows the test error rate across these five-layer neural networks and the number of parameters in each (again with $n = 10$).

Model	Test Set Error	Parameters
M5: $\hat{Y}_s(X_s)$	0.025434 $\pm$0.0245	780
M6: $\hat{Y}_s(X_b)$	0.031150 $\pm$0.0301	2364
M7: $\hat{Y}_b(X_b)$	0.042159 $\pm$0.0384	4080

Table 2: Test MSE values for five-layer NN models with model parameter numbers

A striking result from the table above is that M7, with purely binary inputs and outputs, performs worse than M4, its much simpler regression counterpart, and M1, the 156-parameter linear regression. Another notable detail is that M6, with binary inputs and scalar outputs, outperforms M2, M3, M4, and M7 despite comparable parameter numbers. Among all the models, M5 clearly performed the best despite a low number of parameters.

5.2 ResNet Effect Comparisons

Model	Test Set Error	Parameters
M5R: $\hat{Y}_s(X_s)$	0.025721 $\pm$0.0246	780
M6R: $\hat{Y}_s(X_b)$	0.031557 $\pm$0.0303	2364
M7R: $\hat{Y}_b(X_b)$	0.042343 $\pm$0.0387	4080

Table 3: Test MSE values for five-layer ResNet models with model parameter numbers

Table 3 above shows that adding a skip connection to the model only weakens its performance. for this evaluation, several other skip connections were attempted but none yielded a lower test mean squared error.

Figure 5 below compares all models – except ResNets to avoid redundancy – evaluated in this report along the axes of MSE and log(parameters)

conclusion

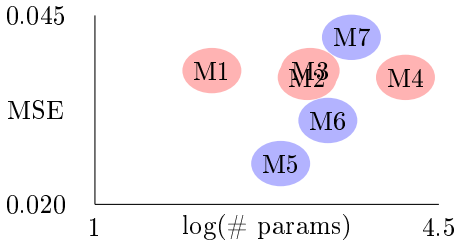

Figure 5: Plot of test MSE versus log(parameters) of all models outlined in report except ResNets; regressions in red and deep neural networks in blue

6 Conclusion

From the results and analysis in the previous section, it is clear that not all nonlinear transformations are "created equal" in the context of model building to encompass chaos. Specifically, nonlinear transformations of data appear not to offset the need for nonlinear functions in a deep learning model, even when more parameters are involved. Though the results of the MSE values alone are mixed, the models involving scalar data become clear winners when accounting for the number of parameters involved. Additionally, it appears that the data transformations have no impact on the effects of adding skip connections to the models.

While these results do not conclusively confirm or reject any hypotheses, they shed interesting light on the nature of deep learning for N-body problems. They indicate that performance does not strongly correlated with the number of parameters, is not hypersensitive to network structure, and responds differently to the type of input and output data even if the information it carries is the same. New insights like these can prove useful as deep learning for advanced modeling becomes more generalized and more widely found at the forefront of scientific challenges.

7 Contributions

I, Arthur Campello, am the sole author of this work and performed all the related work including formulating the problem, writing and running the code, and drafting this written report, all on my own without external assistance.

References

[1] Bokányi, E. and Hannák, A. Understanding Inequalities in Ride-Hailing Services Through Simulations. *Nature Scientific Reports* 10, 6500 (2020)

[2] Beck, A. *N*-body Plasma Simulation. *Doctoral Dissertation: Observatoire de Paris, 75* 2008.

[3] Breen, Philip G. and Foley, Christopher N and Boekholt, Tjarda and Zwart, Simon Portegies. Newton versus the machine: solving the chaotic three-body problem using deep neural networks. *Monthly Notices of the Royal Astronomical Society* 494(2):2465–2470, 2020.

[4] Kingma, D. and Ba, J. (2015) Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference on Learning Representations*, ICLR 2015.

A Code Structure

Section 2 of the report details the roles of the files named "data_generation_with_FDS.py" and named "data_augmentation_and_transformation.py". All other ".py" files correspond to specific models and are identified by prefixes "regression," "deep_nn", and "deep_resn" corresponding regression models (M1-M4), deep neural network models (M5-M7), and residual neural network models (M5R-M7R). Included Notebook files serve to run the models corresponding to their file names.

B Note on Recursive Neural Networks (RNNs)

Because the system in focus evolves in time, it would seem natural to consider using an RNN to model it's behavior. RNNs prove very useful in most time series forecasting applications because data from more than one time step before the estimated future value often proves relevant to the model. Because of the chaotic nature of the three body problem, however, the next state of a sequence depends almost entirely on the value before it. Thus, sequence data over time poses very little additional "modeling potential" compared to input/output sets. Earlier iterations of this project involved analysis of RNNs but this proved redundant with the analysis presented.