Semester Project Suggestion "Simulating the 3-body problem (planetary motion) with a neural network"

Overview. The Three-Body Problem (TBP) concerns three celestial bodies moving around each other, choreographed by nothing else than gravitational forces. The TBP is a is a classical problem in physics and mathematics and was first raised and studied by Newton for the triple sun, earth and moon (https://en.wikipedia.org/wiki/Three-body_problem). While it is easy enough to formulate the equations of motion (a set of 18 ODEs), it cannot be analytically solved: given the initial positions and velocities of the 3 bodies in 3D space at time 0, no analytical formula exists which could give the positions and velocities at future times *t*. The system is chaotic, which means that slight changes in the initial conditions quickly "explode" into large differences in the system state at later times. The chaoticity of gravitational three-body systems makes it expensive to predict future states with a given error tolerance (will the comet that passed us this year at 400M km distance hit earth in 120 years?). The needed numerical bit precision grows linearly with the prediction time, which makes the predictive simulation of gravitational stellar systems by dedicated arbitrary-precision solvers for ODEs expensive – often *prohibitively* expensive.

In this situation, recent work has attempted to replace numerical ODE solvers by trained neural networks (Breen et al 2019, paper included in the package for this semester project suggestion). The idea is to train a NN on a collection of high-precision analytically computed evolutions of a given three-body system, where each training timeseries is evolved from another initial condition. The NN is trained to predict the system state (positions and velocities of the three bodies) at times *t* after the starting time. The goal is that the trained system can be used to predict future system states from novel initial conditions not contained in the training dataset. Breen et al (2019) report savings in computation time when the neural network was used, compared to using the high-precision ODE solver, by a gain factor of up to 10e8. Figure 1 shows trajectories obtained from 4 initial conditions in the simulations of Breen et al.

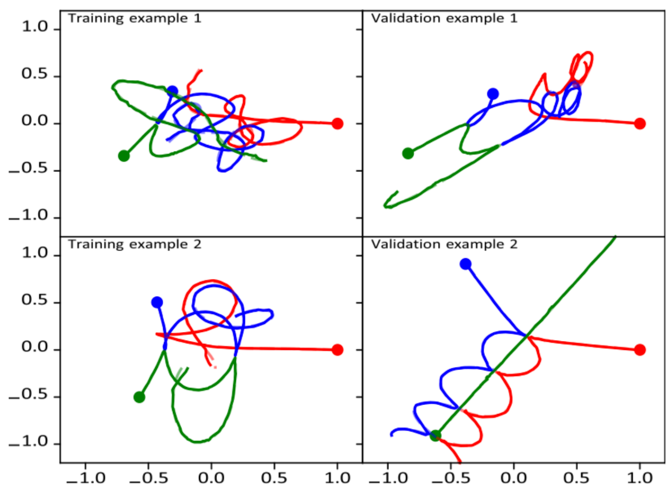

Figure 1. Four dances of three stars. Solid color lines: network predictions; transparent lines (mostly covered by the solid ones): ground truth from high-precision ODE solver. Dots mark initial positions of the three stars. Taken from Breen et al 2019.

Doing this on a professional level is way beyond what can be done in a semester project. The generation of training data alone kept a supercomputing cluster busy for 2 weeks for Breen et al, and the software environment that they built was complex and highly specialized. I suggest to do a basically similar thing as those pros did, but on a much trimmed-down scale. I furthermore suggest to compare two types of prediction methods, single-shot future prediction vs. iterated single-step prediction (Breen et al did only the former, which is more difficult to achieve but computationally cheaper at use-time).

This semester project topic should only be tackled by a group where at least one member knows what an ODE and an ODE solver is.

This kind of task – training a neural network from analytical high-precision simulation data in order to obtain a computationally cheaper simulation engine – has become increasingly adopted in the last few years in the natural sciences for all sorts of systems. The explosion of modeling powers brought about by deep learning, and also the hype about deep learning which has spilled into all science, can explain this.

Trimming down the task. A fully general attack at training neural models on three-body systems would be daunting. I suggest to follow Breen et al (and many other papers) to only treat the TBD in 2 dimensions: when all positions and velocity vectors of the 3 bodies lie in a plane at time $t = 0$ they will continue to stay in that plane forever. One still has a chaotic system but one dimension and 4 equations less.

Still, following in the footsteps of Breen et al would be too demanding. Much of their effort and ingenuity was spent on one aspect of their task which you can dismiss while retaining most of the challenge and fun. Namely, Breen et al wanted to train their network models on *correct* (up to a small tolerance) evolutions of the three-body system equations. To achieve this they had to prepare training data with ultra-precise arithmetics (100 digit precision numerics) and most sophisticated ODE solvers. You can cut cost and effort dramatically if you are satisfied with using a standard, floating-point precision ODE solver to generate training data. The timeseries that you will get – "the dancing of three stars" – will *numerically* deviate from the correct evolution soon after starting times, because that is bound to happen if you simulate chaotic dynamics with standard numerical resolution: the machine precision of floating-point numbers is actually a machine *im*precision, which becomes exponentially magnified as simulated time goes on. However, the dynamics that you obtain is still *qualitatively* the same as if you would run everything with perfect real-number precision. This leads to the following concrete task specification:

Data generation: Generate 12-dimensional timeseries (2D positions and velocities of 3 celestial bodies) by (i) fixing the masses of "your" three bodies, (ii) choosing initial conditions that do not lead to trajectories which are shooting too far out into space (you can limit that by making sure that the total system energy of your initial conditions is not too high), (iii) running a standard ODE solver on those initial conditions for some finite time $t \in [0, T]$. Collect as many and as long training and validation timeseries as you deem fit (it's cheap). Hints:

- You don't need to care for actual stellar physics. Your gravitational constant and the masses and distances need not be measured in realistic astronomical ranges. For simplicity you can set the gravitational constant to 1, masses and distances to small integer values, and measure time in an arbitrary unit.
- The sampling rate should be small enough to let the resulting trajectories look smooth if you plot them.
- System equations and Matlab code for this kind of data-generating simulation can be found at https://de.mathworks.com/matlabcentral/fileexchange/96932-three-body-problem (needs a MathWorks account to be accessed, I put a copy in the package)
- It will not be difficult to translate these short Matlab scripts to Python, and I am quite sure that you can spot similar code written in Phython on the web (I didn't search).
- Experiment a little with your masses and range of initial conditions until you have found a simulation regime that gives interesting trajectories whose spatial extension remains within reasonable bounds. The latter is helpful because it is not easy to feed neural networks with numerical timeseries that have a wide dynamic range.
- Be aware that these 3-body systems are, physically speaking, closed *energy preserving* systems. The initial conditions define a total energy of the system which remains identically the same all the time of the future evolution. The geometry of the "dance of the stars" … that is, the geometry of the *chaotic attractor* that you get, will be different for different total system energies. That is, if you want to create many training timeseries to train one model from, then all these training timeseries should come from initial conditions that have the same total system energy. – It will be easier to just create one *very* long training timeseries.
- Be aware that you are dealing with a 12-dimensional chaotic system: this is a *very* complicated system and you will need *very* large training datasets to get good results.
- If you aim very high (and have lots of compute power at your disposal) you can also train your model on training examples that have different total energy – your model can then generalize over different total energies. I have no intuition how quickly this becomes how difficult.

Objectives and network set-up. In order to predict trajectories from deterministic systems (chaotic or otherwise) there are two approaches. If you aim high, you could implement both and compare what you can get out of them.

- *Approach 1.* The first approach is to train an MLP to output the system state (positions and velocities) at future times $t \in (0, T]$, when the network is given the initial state at $t = 0$ plus the prediction time t. This is the approach taken by Breen et al. It is not easy to successfully train a feedforward network for this kind of prediction task – Breen et al implemented a deep MLP with 10 layers and a specially designed sort of layer. The shorter you choose the interval $(0, T]$ of considered futures, the easier to predict the futures $t \in (0, T]$.

- *Approach 2.* The second approach is to train a neural network (MLP or RNN) on a single-step prediction task. The NN lecture notes explain the principle in Section 4.1.2. You have to fix a time interval in your simulation time (for instance, 0.01 units) which becomes the discrete update step of the neural network. This should be a multiple of the time increment used in the ODE solver, such that you can temporally align timeseries generated by your NN with the ODE generated ones. The choice of how much simulated time should correspond to a network update step will have a strong impact on the accuracy of the resulting network-generated three-body dances.

  When your single-step predicting network is trained, you can use it like the MLP from Approach 1 to compute predictions of the three-body system state at future times t (constraint: t must be a multiple of the time interval that corresponds to a network update), by iterating the one-step prediction.

An interesting project objective which you might want to pursue is to compare the accuracies that you can achieve with these two approaches. Or rather, – since with both approaches you can achieve any accuracy you want if you invest large enough training data sets, large enough networks, and sufficient insight, skill and patience – an objective would be to explore tradeoffs between computational cost and accuracy in both approaches.

But don't put too much on your plate – comparing the two approaches is certainly a high-end semester project. If you implement only one of the two approaches and do that very well (that is, insightfully and technically correct – regardless of the achieved accuracy), and document it very well in the report, you can get a 9.0 grade.

Hints.

- Neural networks do not like input with a high dynamical range (that is, large *and* small values for the same input variable, where both the small and the large values carry much information). Your 3-body simulation will easily give position and velocity timeseries with a large dynamical range (for instance, one of the 3 bodies may sometimes be flung far off the common center of gravity, making for some extreme peaks in the amplitude of its position trace). You will likely get better results if you use a sigmoid-function transform of the raw input signals which dims down large amplitudes. Experimentation will be needed.
- The TBP literature often uses time-varying coordinate transformations to turn the original ODEs into a form that is more precisely solvable by ODE solvers. Go this way only if you know what you are doing, I don't expect this at all.
- If you feel a little overwhelmed and want to keep your project as safely doable as you can, then I recommend to opt for Approach 2 – it is the easier one to achieve reasonably nice-looking results.
- This task begs for nice graphics!

Reference

Breen, P. G., Foley, C. N., Boekholt, T. and Portugies Zwart, S. (2019), *Newton vs the machine: solving the chaotic three-body problem using deep neural networks*. https://arxiv.org/pdf/1910.07291.pdf