2AMM10 Assignment 2 : Gravitational Mechanics Prediction and Simulation with Machine Learning

1 Introduction

The exponential increase in compute power has led to computer simulations playing a prominent part in virtually all science and engineering disciplines. Numerical simulations are an attractive alternative to real-life experiments when exploring a large range of input parameters, or when conducting real experiments is expensive or impossible. However, there are also limitations to this paradigm. The engineering of a faithful simulation program can require many years of engineering efforts, and this requires complete knowledge of the actual physical laws that govern the evolution of the system in the first place. Finally, performing accurate simulations can still be expensive when the simulation engine is computationally intensive.

Over the last years, there has been an increased interest of the machine learning community in building data-driven simulation engines powered by deep learning to alleviate one or more of the above limitations. Generally speaking, it could make sense to use a so-called *neural simulator* if conducting real-life experiments and performing traditional numerical simulations is impossible or expensive. However, deep learning models are typically data hungry. This can be a problem, precisely because it only makes sense to use a neural simulator when acquiring the data in another way (by real-life experiment or traditional numerical simulation) is difficult. Therefore, it is crucial to make neural simulators as data efficient as possible. One way to achieve this is to incorporate the symmetries of the problem in the model architecture.

In this assignment, you will focus on building a deep learning simulator to approximate gravitational dynamics. To guide you along the way, the assignment consists of three tasks. Each of these three tasks is explained in more details in Section 2. It is recommended to work through these tasks in order as the solution for each task might provide a good starting point for the following task, and you should be able to re-use parts of your code across tasks. Try to keep the potential use cases of neural simulators as well as their pitfalls as explained in the two paragraphs above in mind when coming up with your solution.

2 Assignment Description

It was already mentioned that we will study gravitational dynamics in this assignment. More precisely, we study how a system of $n \in \{4, \dots, 9\}$ objects $p_1, p_2, \dots, p_n$ moving in 2D space evolves over time. At time t, the system is described by the mass $m_i \in \mathbb{R}$, the position $\mathbf{x}_i^t \in \mathbb{R}^2$ and velocity $\mathbf{v}_i^t \in \mathbb{R}^2$ of each object p_i. After consulting world-renowned astrophysicist M. Vlenkovski, we learned that at any time t the evolution of the system is governed solely by the forces of all pairs of objects acting upon each other. Furthermore, for any two objects, the force between them depends only on their relative locations and the masses of the objects. For all tasks, functions for loading and reading the data and a short explanation on the data structure are provided in the skeleton Jupyter notebook accompanying this assignment.

2.1 Task 1: Point Prediction

In task 1, we have datapoints of objects moving through space, observed at two distinct points in time: a datapoint contains all p_i in the system, and the associated initial positions and velocities $\mathbf{x}_i^0$ and $\mathbf{v}_i^0$ and terminal positions $\mathbf{x}_i^T$, where $T = 5$ time units. The goal of the assignment is to investigate if a machine learning model can be used to predict the terminal positions $\mathbf{x}_i^T$ based on the information available at $t = 0$. If this model performs well, it could be used to 'shortcut' iterative physics-based solvers that require many small steps to predict the terminal positions of the planets, offering a cheaper alternative compared to prediction with a physics-based model.

Here, the training and test data consist of the initial positions and velocities $\mathbf{x}_i^0$ and $\mathbf{v}_i^0$ of all objects p_i, which serve as model input, and the terminal positions $\mathbf{x}_i^T$, which the model should predict.

Hints:

- What architecture would be suitable for this kind of data? What motivations do you have for your choice of architecture (symmetries, pragmatic, intuition)?
- Assume you're flying in a spaceship to observe planetary objects orbiting each other. What would you observe if the spaceship would have been upside down instead? What would you see if the spaceship were rotated?

2.2 Task 2: Simulation

Even if the model of task 1 would do a perfect job at predicting the terminal positions of the observed objects in the test set, it is difficult to know whether we can trust this model. One reason for this is the black-box input-output formulation of the machine learning task. To alleviate this issue, in the second task you are going to build a model that can produce full *simulations* of the system, rather than only point predictions. As in task one, we assume that at inference time we have access to the initial positions and velocities $\mathbf{x}_i^0$ and $\mathbf{v}_i^0$ and that all objects p_i are observed. Now, the goal is to predict a *sequence* $\mathbf{x}_i^{1:T}$ of positions up to the terminal state at time $T = 5$.

For this task, the training data consists of trajectories $\mathbf{x}_i^{0:T}$ and $\mathbf{v}_i^{0:T}$ of objects p_i. The test set consist of datapoints containing initial positions $\mathbf{x}_i^0$ and velocities $\mathbf{v}_i^0$, paired with the remainder of the trajectories $\mathbf{x}_i^{1:T}$ of the objects p_i for evaluating the performance of your method. You can assume that the temporal resolution of the provided data is orders of magnitude coarser than the internal stepsize of a high-accuracy physics based solver. Consequently, a model that works well for this task can still simulate very fast, while adding some interpretability to the predicted outcomes.

Hints:

- We only care about the performance of your model with respect to the predicted positions for its evaluation. However, depending on your approach, it might be useful to also have your model predict velocities in order to infer positions later on.
- As in task 1, it is important that your model architecture is aligned with the properties (symmetries, structure) of the data, in order to make your model as data efficient as possible. Keep this in mind when coming up with a solution for this task.

2.3 Task 3: Simulation with Uncertainty Quantification

In task 2, our goal was to build a model that can generate a full trajectory from an initial system configuration. In this setting, it is still difficult to know if we can trust the model, as it is forced to predict a single trajectory for any given set of initial conditions and cannot quantify its uncertainty. However, in the scientific context, it is crucial to have uncertainty estimates for any simulation model in order to put any value on the simulation outcome.

The goal of task 3 is to build a model that can still simulate whole trajectories, but at the same time express uncertainty by modeling a *distribution* over possible trajectories.

This means that we want to tune the model parameters such that the model's distribution over trajectories expresses its belief over the possible system evolutions. As each simulation starts from a specified starting point, the distribution should be conditioned on the initial state of the system. Note that in this formulation, we take a Bayesian perspective on probability, as there is only one true trajectory for each initial state.

The data you should use for this task is identical to the data of task 2.

Hints:

- What methods do you know for modeling (conditional) probability distributions over high-dimensional data?
- As in tasks 1 and 2, it is important that your model architecture is aligned with the structure of the data as much as possible. Try to develop a solution that achieves this. However, you can still get many points even if your solution does not fully align with the structure of the data.
- This is not a simple task, and many possible solution directions are viable. Take this into account when planning the assignment.

3 Deliverables

The submission of this assignment is done in ANS. There are two deliverables:

- Implementation of the solution
- Description and explanation of your approach

For the implementation, a skeleton Jupyter Notebook with functions that can load the provided data for each task are provided. The notebooks also contain short bits of self-explanatory code on how the data is formatted and how to load the data in python. Please submit your code in the provided skeleton Jupyter notebook. You need to upload the Jupyter Notebook code (.ipynb file extension) and a PDF version (.pdf file extension) with the execution output. The maximum upload size is 25MB. So, you may need to clean some image output from the code Notebook before uploading. If you are using Google colab, you can generate the PDF by going to "File → Print → Save as PDF". Please generate the PDF after running all cells of your solution (with possibly images removed if necessary for file size constraints).

The description of your approach is submitted as a digital group test in ANS. Rather than an unstructured report, the group test is formatted in a number of questions that you need to answer to describe your solution and understanding of the assignment. Please see ANS for the details.