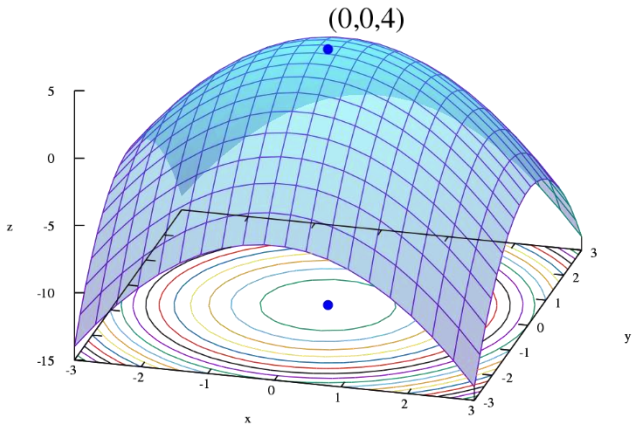
TIF285: Learning From Data

PROJECT 2



BROU BONI Joël

SANOGO Ibrahim Bechir

Introduction

The aim of this project is to replicate the results of a paper conducted by Philip G. Breen, Christopher N. Foley, Tjarda Boekholt and Simon Portegies Zwart.

The authors studied in this paper the complex problem of **solving the equations of motion for three bodies under their own gravitational force**. It is complicated to solve this problem because of the unpredictable and potentially infinite computational cost, due to the system's chaotic nature. They demonstrated to what extent a trained **ANN** (**A**rtificial **N**eural **N**etwork) can replace existing numerical solvers (such as Brutus), enabling fast and scalable simulations of many-body systems to shed light on outstanding phenomena such as the one we are studying. Over a bounded time interval, an ANN provides accurate solutions at fixed computational cost and up to 100 million times faster than a state-of-the-art solver (**Brutus**), described in the paper.

In this project, we will mainly compare the results of the dataset(**data_project2.npz**) with a pretrained ANN(**Breen_NN_project2.h5**) and our own-built ANN called model.

A complete description of the points mentioned above will be given in the next part:

“What is the structure of the dataset and the two ANN?”

“What kind of data are we actually studying?”

The project is divided into 2 parts: the basic part where we must represent and interpret *Figure 3,4 and 5* of the paper.

Then, there is the extra part where we must reproduce *Figure 6*.

Basic Part

Part A

“What is the structure of the dataset? What kind of data are we actually studying?

The dataset was built from the data_project2.npz file. It is a multidimensional array of shape (9000,1000,9). The first axis labels the 9000 data samples, the second axis represents the 1000 timesteps (from t=0 to t=3.90). The nine columns of the last axis correspond to: [t, $x1$, $y1$, $x2$, $y2$, $vx1$, $vy1$, $vx2$, $vy2$]. So our dataset is basically composed with 9000 possible behavior of particles according to their starting points studied for 1000 time steps during which we identifies some characteristics of the particles such as their positions on the X and Y axis and the velocities.

x1, x2: position of particle 1,2 in the x axis at time t
y1, y2: position of particle 1,2 in the y axis at time t

vx1, vx2: velocity of particle 1,2 in the x axis at time t
vy1, vy2: velocity of particle 1,2 in the y axis at time t

By symmetry, particle 3 is built given the positions of particles 1 and 2.

x3= -x1-x2 : position of particle 3 in the x axis at time t

y3= -y1-y2 : position of particle 3 in the y axis at time t

The original data is composed of 2561-time steps, reaching 10 time units, but as discussed in the paper the training of the ANN works better for the shorter time interval that we use: t ∈ [0, 3.90].

Entrée [2]:
```
# Load and unpack a compressed npy array
load_data = np.load('data_project2.npz')
data = load_data['arr_0']
print(data.shape)
suppr = np.where(data[:,999,1] == 0)
suppr = np.asarray(suppr).T
data_clean = np.delete(data,suppr,axis=0)
print(data_clean.shape)
data=data_clean
```

```
(9000, 1000, 9)
(7623, 1000, 9)
```

Before going on the explanation of the pretrained and own ANN, we cleaned the dataset.

Some samples were filled with nothing but 0 so we deleted them.

It was necessary because the quality of our own-built model increased greatly after this operation.

"What is the structure of the dataset and the two ANN?"

The pretrained ANN (well described in the paper) and our own-built model shares the same structure: feed-forward ANN consisting of **10 hidden layers of 128 interconnected nodes.**

Both models have a total of **149,636 trainable parameters**:

Entrée [142]:
```
trainable_count = np.sum([K.count_params(w) for w in model.trainable_weights])
non_trainable_count = np.sum([K.count_params(w) for w in model.non_trainable_weights])
print('Total params: {:,}'.format(trainable_count + non_trainable_count))
print('Trainable params: {:,}'.format(trainable_count))
```

```
Total params: 149,636.0
Trainable params: 149,636
```

Training (90% of the dataset) was performed using the adaptive moment estimation optimization algorithm (ADAM) and Mean Absolute Error (MAE) for both our loss function and metric. Our model has 100 epochs, each epoch was separated into batches of 5000, and setting the rectified linear unit (ReLU) as our activation function. By entering a time t and the initial location (x2(t=0), y2(t=0) of particle 2 into the input layer the two ANN returns the locations of the particles 1 (x1(t), y1(t)) and 2 (x2(t), y2(t)). It is always assumed that particle 1 is initially positioned at (1, 0).

By summarizing all these elements, we can reproduce *Figure 3* with our own-built model.

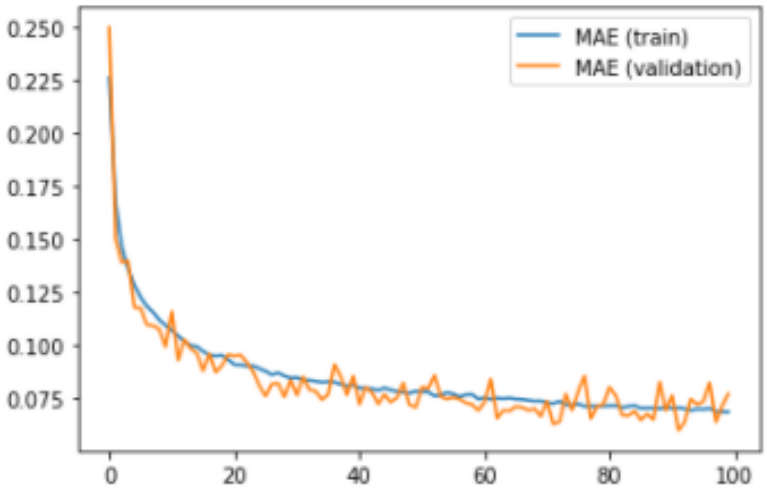


Figure 1 Mean Absolute Error (MAE) for each Epochs epoch

Contrary to the case described in the paper, our ANN works with one-time interval. Indeed, t ∈ [0, 3.90] is supposed to give the best results. Blue line represents the loss on the training set and the orange one is the loss on the validation set. T ≤ 3.9 corresponds to 7623 labels/timepoints (the entire dataset). The results illustrate a typical occurrence in ANN training, there is an initial phase of rapid learning epochs (until 20-25), followed by a stage of much slower learning in which relative prediction gains are smaller with each epoch.

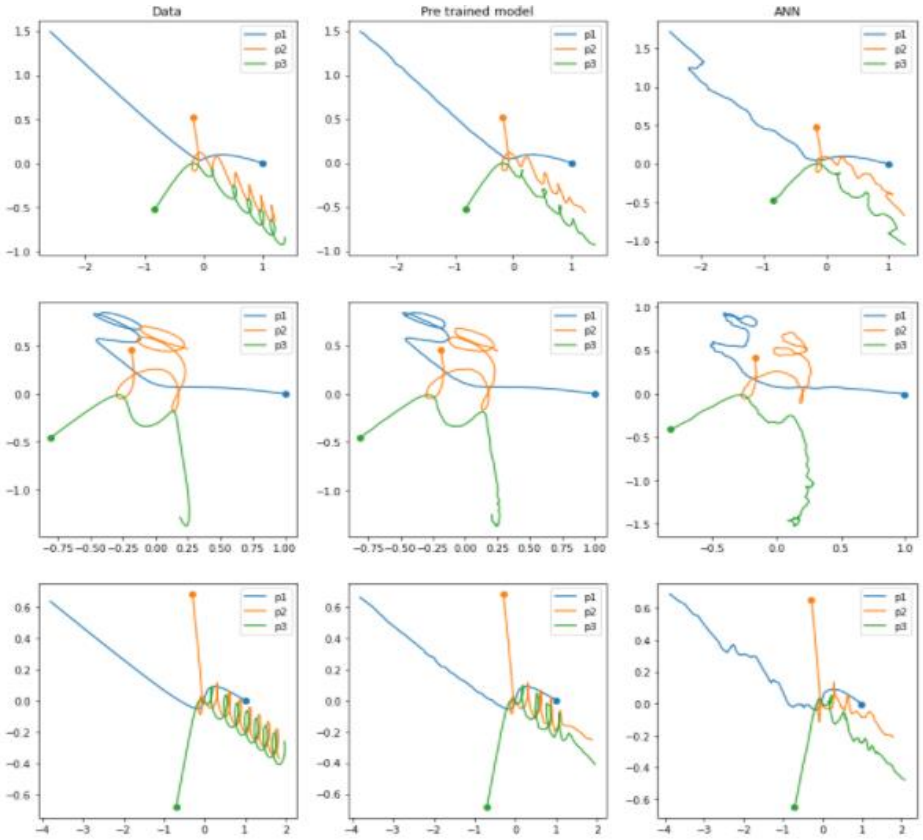
Part B

After representing Figure 3: Mean Absolute Error (MAE) vs epoch, our next step in this project was to validate the pre-trained ANN and our own ANN by showing some trajectories and compare it to the real trajectories. To realize this task, we trained our own ANN for 4 hours and 4 minutes. Then, we just add to predict the output given an input from the training set and another from the test set. As said his name, the pre-trained was already trained so there was no need to feed it with data before prediction.

Presented are two examples from the training set (left) and one from the validation set (right). All examples were randomly chosen from their datasets. The bullets indicate the initial conditions.

The curves represent the orbits of the three particles (blue, orange, and green) the latter obtained from symmetry). The solution from the pre-trained network (on the middle) is hardly distinguishable from the converged solutions (dashes, acquired using Brutus). As we can see our own-built ANN is not as good as the pretrained model. This can be explained by the fact that we had to delete 1377 samples from the dataset. It is because these trajectories intersect at the center of mass at some timestep, the recording has then been stopped for these moments where collision happens. The pretrained model has therefore been trained with more dataset than ours. Finally, it is interesting to check the computational cost of making a prediction with the ANN It is about 0.95 ms for the pre-trained model and our own-built model. This is much shorter than the numerical integration performed by Brutus which takes minutes to hours for finding a trajectory. The time saving is therefore enormous, especially when you see the performance of the models compared to the results obtained with Brutus.

Figure 2 Validation of the trajectories for the pre-trained ANN and our own-built ANN and the dataset



Part C

Finally, we can explore the chaotic aspect of this motion by creating a thousand different trajectories from a slightly disturbed initial condition.

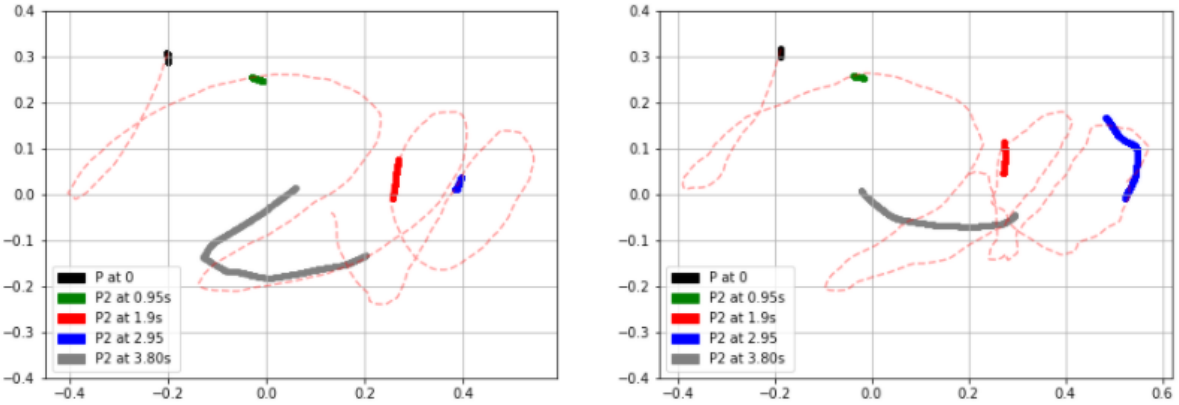


Figure 3 Visualization of the sensitive dependence on initial position

Presented are trajectories from 1000 random initializations in which particle x2 is initially situated on the circumference of a ring of radius 0.01 centered at (-0.2, 0.3). For clarity, these locations were benchmarked against the trajectory of x2 initially located at the center of the ring.The trajectories where randomly generated and the locations of the particles at each of these five timepoints, t ∈ {0.0, 0.95, 1.9, 2.95, 3.8}, are computed using either the pre-trained ANN (left) or our own-built (right). We noticed while plotting the curve that one of the times given in the paper does not seem to correspond to what

they get figure 5. Indeed, for T = 2.95s we get the blue dots while at T = 2.85s we get the yellow dots like the arc on the right of figure 5. To avoid any confusion, we have decided to represent both.

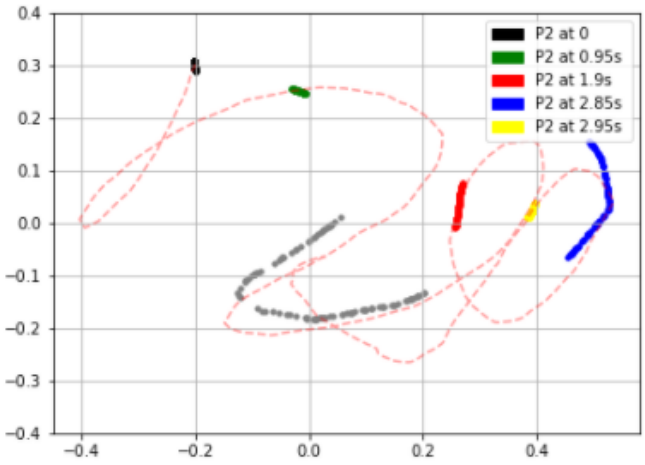


Figure 4 Visualization of the sensitive dependence on initial position (Pre-Trained)

This plot highlights the ability of the pre-trained model to accurately emulate the divergence between nearby trajectories, and closely match the Brutus results with a lower computation time. This plot also show that the pre-trained model is better than ours. Indeed, the behaviors of the particles for the pretrained ANN looks like the Brutus ones that we can see on the paper. The behavior of our model is the same than Brutus and the pre-trained ones for most of the timesteps except for the last one where the point reparation is not really the same.

Extra Part

The last part of this project was to train our ANN such as it could respect the laws of physics (e.g the conservation of energy. To fulfill this purpose, we had to create functions that compute the potential energy and the kinetic energy at each time step.

We used the following formulas to compute the two energies:

$$velocity = \frac{Distance\ made\ by\ the\ particle\ between\ 2\ time\ steps}{Difference\ between\ 2\ time\ steps} = \frac{\Delta d}{\Delta t} = \frac{\Delta(\sqrt{x^2 + y^2})}{\Delta t}$$

$$Potential\ Energy = \frac{-G*m*M}{R} = -1/R$$

R is the distance between two points

$$Kinetic\ Energy = \frac{1}{2} * m * v^2$$

After making these TensorFlow functions, we used them to check if the conservation of energy was respected for our model. And we realized that it was not really the case for our model. Indeed, the energy of our system in not really equal to 0 every time because of the big spikes of the potential energy because of big spikes that can appear has you can see down below. But the two other models are respecting it better.

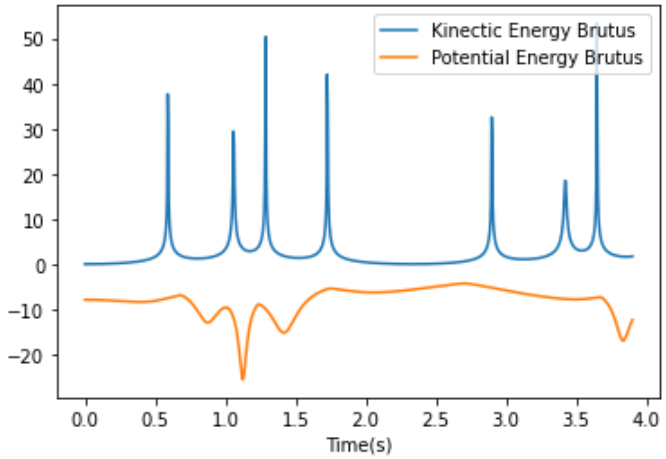


Figure 5 Evolution of the Potential and Kinetic Energy Brutus

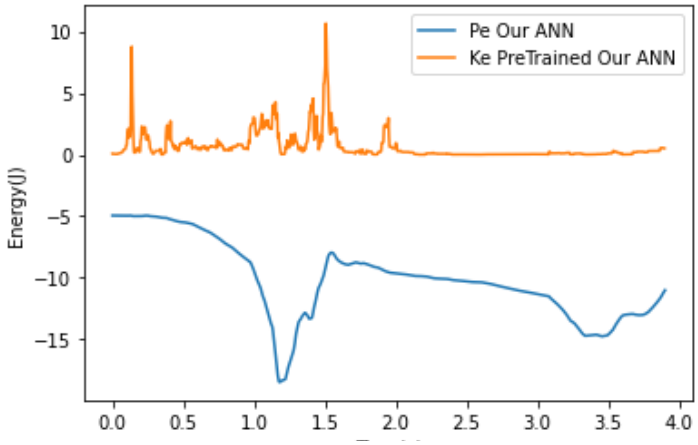


Figure 6 Evolution of Potential and Kinetic Energy ANNA

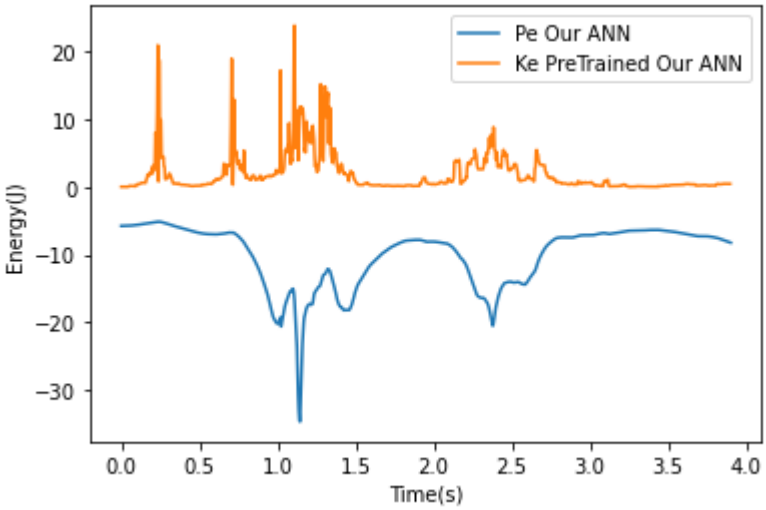


Figure 7 Evolution of Potential and Kinetic Energy Pretrained

To improve our model, it is then necessary to make the custom loss function to force our system to respect this physic law. We tried to make this function and manage to make it work but as we write this report(Wednesday) and considering how long it takes to train the model we didn't had the time to present it in this report.

Conclusion :

This project has shown us the efficiency and usefulness of a deep learning algorithm.Indeed, we have been able to see that with a good deep learning algorithm, we are able to represent complex systems much faster than some calculation programs. However, this is possible only if we have a large amount of data to train the model and a sufficiently powerful computer to make this training as fast as possible.

Bibliography

Paper that inspired this project: https://arxiv.org/abs/1910.07291

Tensorflow Documentation : https://www.tensorflow.org/api_docs/python/tf

Lecture notes : https://chalmers.instructure.com/courses/10188