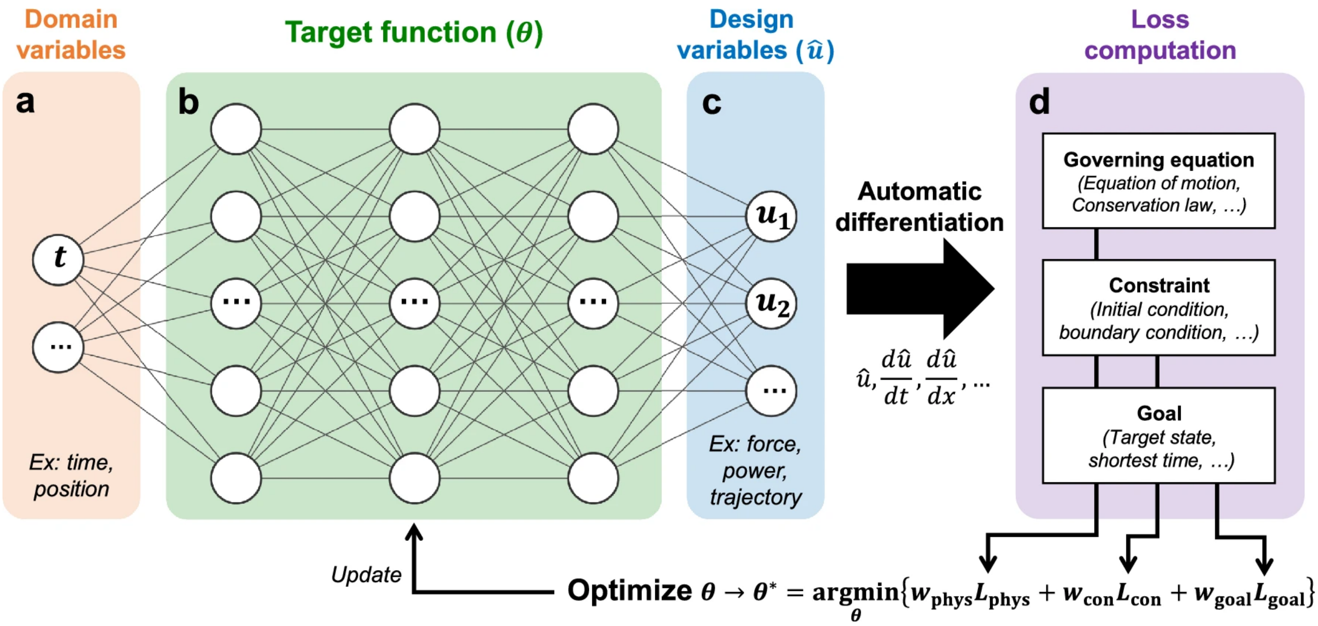
README

Reproduction Report: *Solving real-world optimization tasks using physics-informed neural computing*

Authors Group 9:

- Mathijs van Binnendijk (4583957)
- José Cunha (5216087)
- Lucas Van Mol (6059686)
- Nicolás Farjado Ramírez (5918340)

In this blog post we attempt to reproduce the paper *Solving real-world optimization tasks using physics-informed neural computing*[1]. The paper introduces a "goal loss" to the loss function of traditional Physics-Informed Neural Networks (PINNs) in order to optimize for certain tasks. The paper's code is available on GitHub and uses the DeepXDE library to implement the PINNs. We reproduce three of the examples produced in the paper by writing the network in pure PyTorch, and include a new example in order to further test the capabilities if the proposed architecture. The code for our reproduction is available online at https://github.com/TUDjose/pinn-reproducibility.



Pendulum

by Lucas Van Mol

In this example, a pendulum is attached to a low-torque actuator, and the PINN has been given the objective to invert the pendulum such that $cos(\theta) = -1$. The idea is that the network should learn to swing the pendulum in order to accumulate enough energy for inversion. The network has one input, time t, and two outputs angle θ and torque τ. It then acts like a function approximator, with the aim to learn the trajectory of the pendulum over time, constrained by physics laws, boundary conditions and objective angle:

$$f : t \mapsto \theta, \tau$$

There are three losses the network aims to minimize. The first is the physics loss, ensuring that forces are conserved:

$$\mathcal{F} = ml^2\ddot{\theta} - (\tau - mgl\sin\theta),$$

where τ is the torque, limited by $|\tau| = 1.5\ \mathrm{Nm}$. The L2 norm of $\mathcal{F}$ is averaged over the time domain to give the physics loss L_{phys}.

Boundary conditions are achieved through the constraint loss. In this example, we want $\theta, \dot{\theta}, \tau = 0$ at $t = 0$.

$$L_{con} = (\theta_{t=0})^2 + (\dot{\theta}_{t=0})^2 + (\tau_{t=0})^2$$

In a similar way, the goal loss is defined as:

$$L_{goal} = (\cos\theta_{t=t_f} - (-1))^2$$

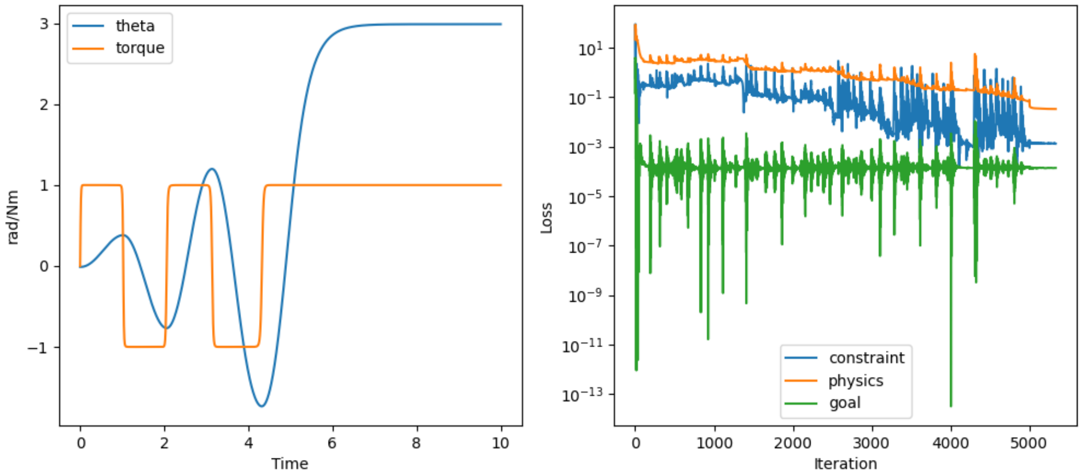
The final loss is then calculated as a weighted average of these three losses.

Own implementation

In our own implementation, we use the paper to implement the PINN in PyTorch, attempting to make minimal use of the author-provided code. We found that a few details, not explicitly mentioned by the paper, were crucial to getting the reimplementation to work. One such detail is point resampling. Initially, we used a grid-sampling technique, using `np.linspace()`, to sample points between t_{min} and t_{max}. However, changing this to be a random uniform sampling of points, resampled every 100 iterations, improved the network's ability to converge. We did need to make sure that the extremes t_{min} and t_{max} were always included in the sampling, such that the network could efficiently learn the constraints on the initial conditions and on the final objective state.

An example run of our implementation is shown below. We use the same hyperparameters as in the paper, and note that we do not get such a result every time - usually multiple runs are needed before getting a result with sufficiently low loss.

Learned function (left) and losses during training (right)



The figure above shows our implementation working for the pendulum example. On the left, we see that the network has learned to modulate the torque, causing the maximum pendulum angle to increase until inversion.

On the right, we see how the network optimizes:

- physics loss, defined by the gravitational and pendulum governing laws
- constraint loss, ensuring the pendulum starts at $\theta = 0$ with 0 velocity
- goal loss, ensuring the pendulum is at $\theta = \pi$ when $t = 10$.

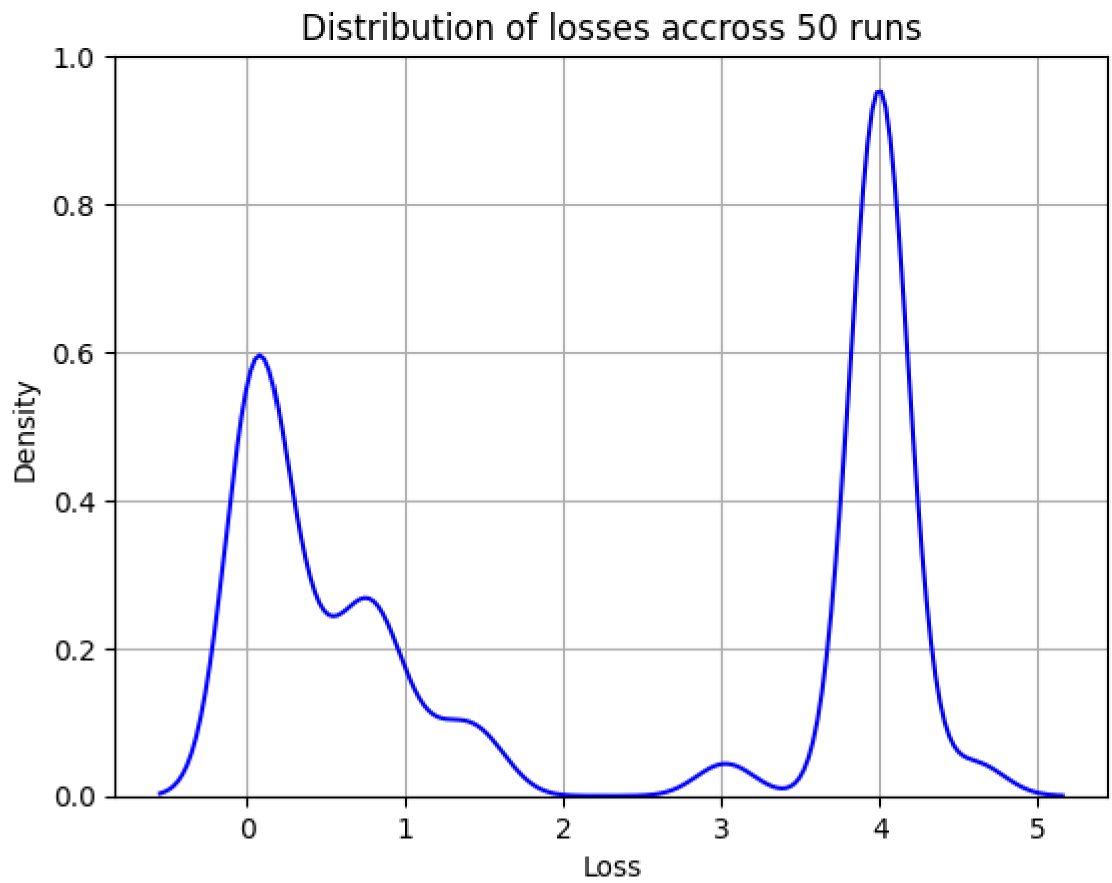
The time to train is within the same order of magnitude as the paper. However, this ignores the unfortunate fact that one has to do multiple runs in order to get a satisfying result - a problem also present in the author's code, but not mentioned in the paper.

Comparison to existing code

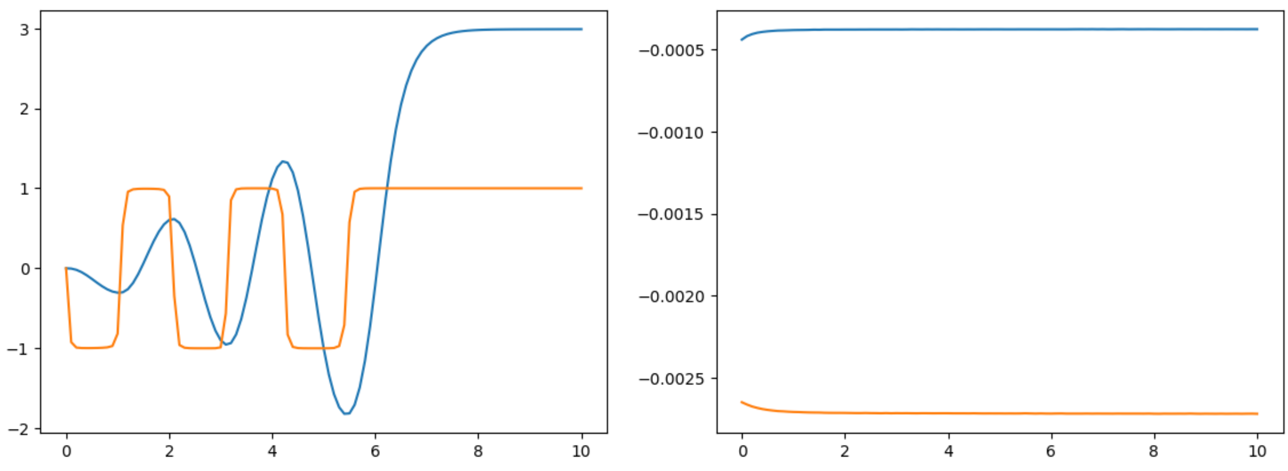
We found that the paper's code is flakier than the paper might suggest. In the author's code, under `data/pendulum/main.py`, the code sets a random seed as such:

```
# Set random seed
seed = 0
np.random.seed(seed)
tf.random.set_seed(seed)
dde.backend.tf.random.set_random_seed(seed)
```

and indeed we are able to reproduce the results with this given seed. However, removing the set seed does not always give good results. To test the reliability of network training, we ran the author's code for the pendulum example 50 times, and examined the weighted sum of losses:



We find that a significant amount of runs end up with a loss of around 4 - this was characteristic of runs that were not able to reach the objective state. Instead, the network got stuck in a local minimum, applying 0 torque and the pendulum staying still. Out of 50 runs, 10 had losses of <1e-1, and 3 had losses <1e-3. Examples runs are shown below.



The angle of the pendulum is shown in blue, and the torque is shown in orange. On the left is an example 'good run' of the author's code with a loss of 1e-4. On the right is a failed run, with a loss of ~4.

Spacecraft Swingby

by José Cunha

Paper implementation and existing code

In the paper, the problem of finding the swingby trajectory of a spacecraft that can reach the given destination using the least amount of thrust is presented. A spacecraft must fly between a series of astronomical bodies, leveraging the gravity felt due to each body, and using thrust to adjust its trajectory to reach a certain final point. This problem is solved using two loss functions, one for the physics (which has the goal loss embedded in it), and one for the boundary conditions (i.e. the constraint loss). The physics and constraint losses are defined from the functions for the thrust, and initial/final positions:

$$\mathcal{F} = \begin{cases} \frac{1}{T^2}\frac{d^2x}{dt_N^2} - \sum_{(x_0,y_0,GM_0)}\left[\frac{-GM_0(x-x_0)}{((x-x_0)^2+(y-y_0)^2)^{1.5}}\right], & x \text{ component} \\ \frac{1}{T^2}\frac{d^2x}{dt_N^2} - \sum_{(x_0,y_0,GM_0)}\left[\frac{-GM_0(y-y_0)}{((x-x_0)^2+(y-y_0)^2)^{1.5}}\right], & y \text{ component} \end{cases}$$

$$(x,y) = \begin{cases} (-1,-1) \text{ at } t_N = 0 \\ (1,1) \text{ at } t_N = 1 \end{cases}$$

The paper's implementation mentions the following configurations to train the network:

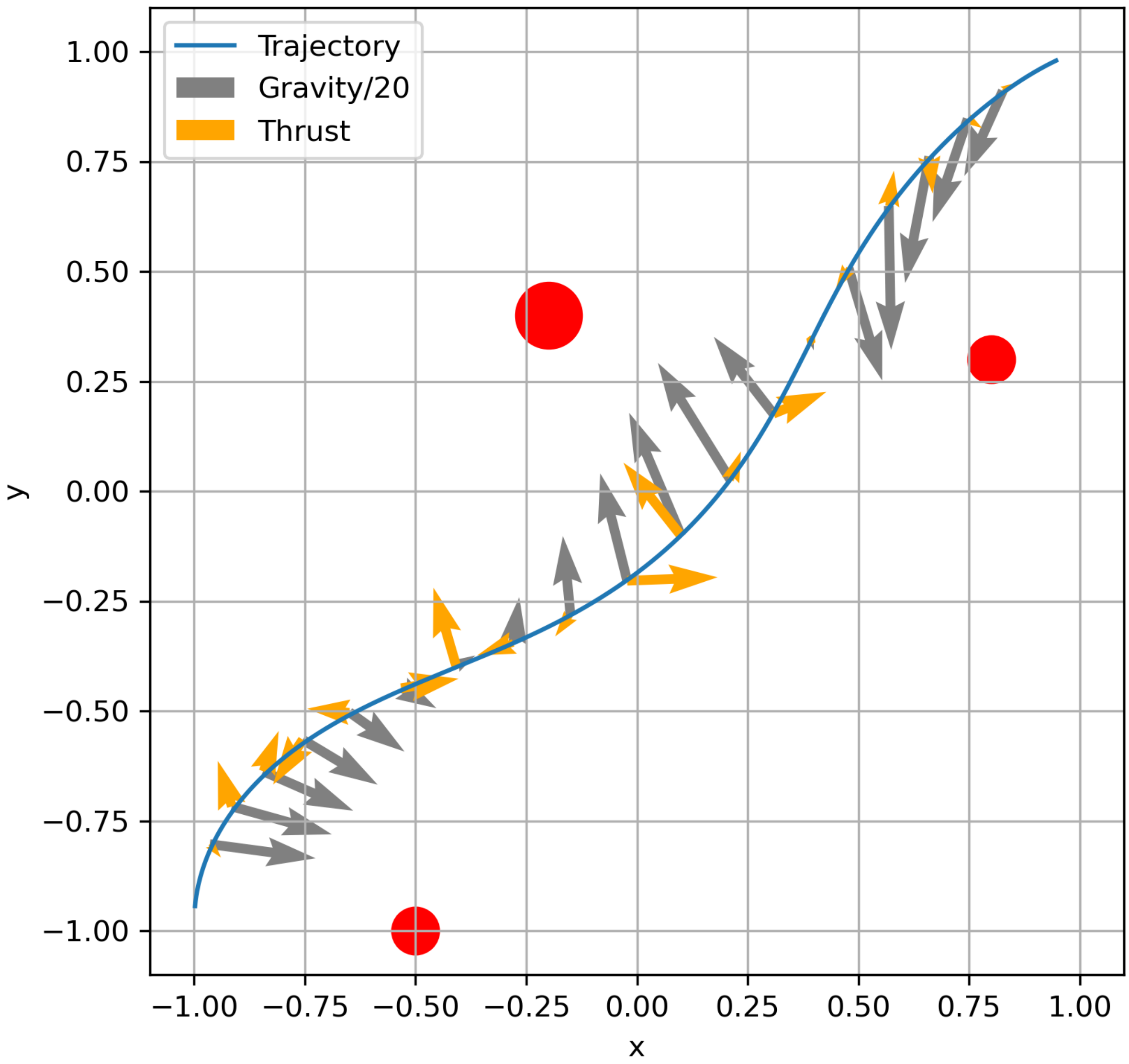
- learning rate = 0.001
- loss weights = $\{w_{phys}, w_{con}\} = \{1, 1\}$
- Adam optimizer epochs = 2000
- L-BFGS optimizer until convergence (1215 epochs needed)
- T is a trainable parameter that acts as a normalization factor

The paper however omits the method in which the variable T is trained, and this had to be inferred from the existing code written using the DeepXDE library. As shall be mentioned in the following section, the configuration used by the paper seems to be only valid when using the specific library, as the total loss tends to never decrease enough to obtain accurate results. Looking at the existing code, something that is not mentioned in the paper is the resampling of input values, which occurs every 100 epochs. This method proved useful in reducing the total loss when a plateau was found. Similarly, the trainable parameter T is only updated every 10 epochs, again omitted in the paper (along with any other mention of the training of T), though this larger training period did not seem to greatly affect the final results. Given this information, the paper is relatively incomplete for attempting a reproduction of the spacecraft swingby results, however, the provided code was useful in understanding the network architecture and the training steps taken.

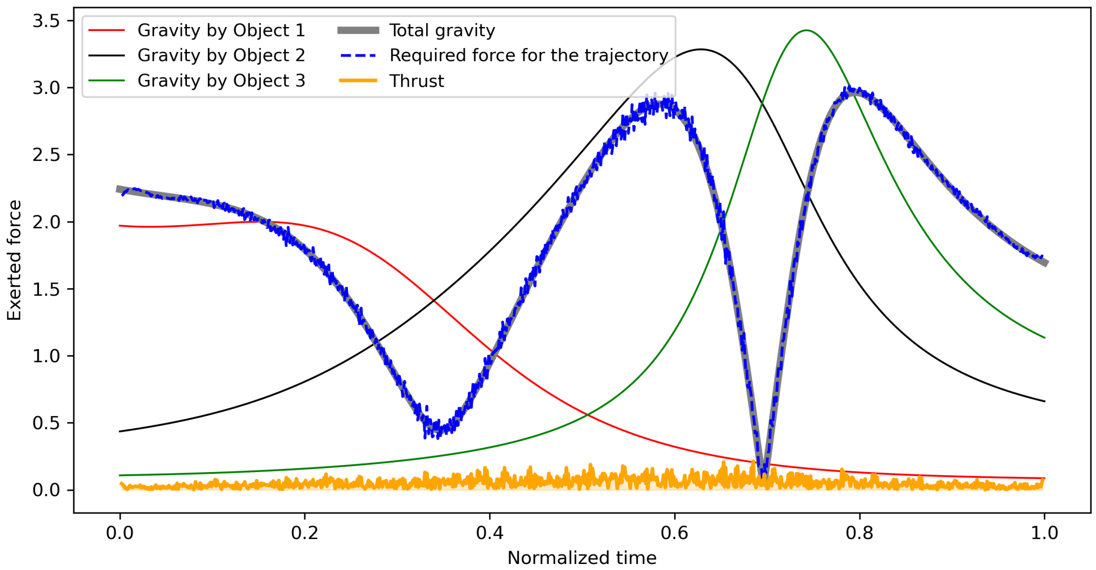
Own implementation

Using the paper's existing code as a starting point, we created our own implementation of the PINN using the pytorch library. The same network architecture was used (1 input, 3 hidden layers with 64 nodes, and 2 outputs), using the $tanh$ activation function for all layers and Glorot normal initialization of parameters. Unlike the paper, a random seed was not selected, somewhat demonstrating a better robustness of out method. The training parameters however differ greatly from the paper. The loss functions weights were updated to $\{w_{phys}, w_{con}\} = \{1, 10\}$, to ensure that the boundary conditions are met. Without this, the boundary conditions were never met. The learning rate was also increased to 0.004, in part to account for the larger weights, in order to reduce the loss. The same number of training epochs were used for the Adam optimizer (2000), though for the L-BFGS, many more were required (closer to 8000 to reach convergence). Also, for the latter half of the L-BFGS training, the learning rate was further increased to 0.05, in order to decrease the loss, while maintaining stability in the earlier parts of training. Input point resampling and the training of the variable T were also implemented using the existing code as reference, which helped in having more stability at higher learning rates.

With this, the spacecraft swingby PINN was trained, and the trajectory of the spacecraft, as well as the thrust exerted by the spacecraft and the gravity forces acting on it were plotted.

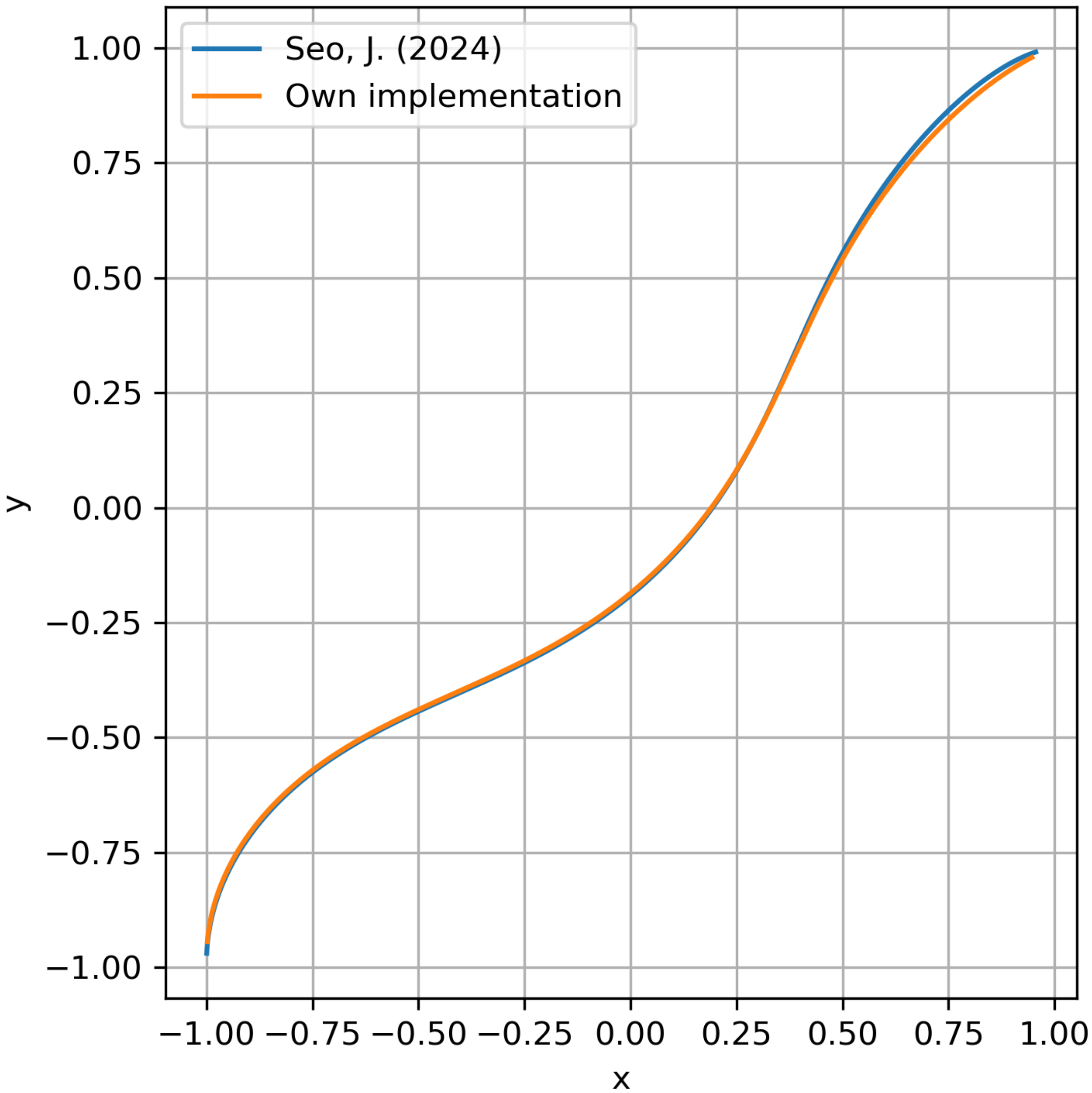


Trajectory
Gravity/20
Thrust
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x



Gravity by Object 1
Gravity by Object 2
Gravity by Object 3
Total gravity
Required force for the trajectory
Thrust
Exerted force
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Normalized time

Comparing these plots to the ones that ought to be reproduced from the paper (namely Figure 4. of the paper), it seems that the results are quite agreeable, where for most of the trajectory, the thrust required to swing the spacecraft between the bodies is very low. The actual trajectory is also practically the same, where especially close to the boundary conditions, the same accuracy is not met. However, it seems that the total loss achieved by our implementation is greater than that of the paper, since the required thrust is higher (not as minimal as the method would lead it to be) and thrust values are noisier. Therefore, it can be considered that the main task of the PINN (which is to satisfy the physics and boundary conditions) has been met, while the optimization task could still be improved.



Shortest Path

by Nicolás Fajardo Ramírez

Paper implementation and existing code

The other problem presented in the paper is to find the shortest between two points. This general problem was explored in two different applications: Fermat's principle of least time, and brachistochrone curves.

Fermat's Principle

Fermat's principle of least time states that the path a light ray follows to go from point a point A to a point B is the path that can be traversed in the least time. This principle explains the refraction of light, as its path gets bended upon changes of the refraction in the medium.

In order to find the shortest-time path of light, the following starting conditions and physics' loss expression are defined:

$$\mathcal{F}_{Fermat} = \left(\frac{1}{T}\frac{dx}{dt_N}\right)^2 + \left(\frac{1}{T}\frac{dy}{dt_N}\right)^2 - \left(\frac{c}{n}\right)^2$$

$$(x,y)_{Fermat} = \begin{cases}(0,0), & t_N = 0\\(1,1), & t_N = 1\end{cases}$$

Note that in the equation, c refers to the speed of light in a vaccum, and n is the refractive index of the medium.

Since the objective is to find the shortest path (the path of least time), the goal loss is defined as $ \mathcal{L}_{goal} = T , $sincethetime\$T$ that light takes to go from point A to point B is the value to be minimized.

Brachistochrone Curve

A brachistochrone curve, or shortest-time descent curve, is the path from point A to point B that ensures the shortest travel that can be achievable while under the effects of gravity g.

In order to find this shortest-descent curve, starting conditions and physics' loss are defined:

$$\mathcal{F}_{brach} = gy_0 - gy - \frac{1}{2}\left(\left(\frac{1}{T}\frac{dx}{dt_N}\right)^2 + \left(\frac{1}{T}\frac{dy}{dt_N}\right)^2\right)$$

$$(x,y)_{brach} = \begin{cases}(0,1), & t_N = 0\\(1,1), & t_N = 1\end{cases}$$

Similarly to the light refraction problem, as the objective is to minimize the time to traverse the path, the time T is selecte as the goal loss.

For both of the sub-problems, most of the same training parameters are the same: Adam optimizer steps set to 2000, learning rate set to 0.001, and loss weights are set to 1, 1, and 0.01 for the constarint, physics, and goal losses respectively (in order to give more weight ot the satisfaction of the physicall principle rather than the time's minimization). The only training parameter that was changed depending on the problem was the number of steps for the LBFGS optimizer, which was 1232 for the Fermat's principle problem, while being 2692 for the brachistochrone curve.

The paper states that an analytical and RL approaches can be taken to compare the PINN solution:

“images/shortest-path-initial.jpg” could not be found.

Figre a shows the paths achieved by the different solution approaches for the light's path, while figure b does so for the brachistochrone curve.

Own implementation

Using the knowledge gained from the previous problems, the shortest path problem was implemented using the same network architecture as
before, though with a sigmoid activation function for the output layer, as the output is bounded between 0 and 1. However, training
of these two similar problems was not as successful as the previous ones. The total loss was not able to decrease to the same extent as
the spacecraft swingby or inverted pendulum problem, and the results were not as accurate. An initial test keeping all the training parameters the same was performed, results for Fermat's problem are presented in the following image:

“images/fermat_initial.png” could not be found.

This image shows the shortest path encountered on the left side, while the loss progression on the right side. As mentioned before, the results weren't as accurate as the ones presented in the paper, the path achieved being only a straight line that does not even reach the boundary conditions ($(0, 0)$ and $(1, 1)$), and the loss, while displaying a quick reduction to a constant value, is still quite high, specially when compared to the losses of the previous problems.

The results for the brachistochrone using the same training conditions are presented in the following figure:

“images/brach_initial.png” could not be found.

Similarly to Fermat's problem, in this case the path achieved was a straight line that does not reach the boundary conditions ($(1, 0)$, $(0, 1)$) and the losses are quite high.

Several tests doing some alterations to the losses' weights were done, in order to look for a better solution. However, these did not successfully return the proper best path shown as the analytical solution in the paper.

For the case of Fermat's problem, the best solution found was upon assigning the constraint loss a weight of 5 while keeping the other ones the same:

“images/fermat_best.png” could not be found.

In contrast to the initial solution, this one displays the correct initial conditions in the path. However, it is still a straight line with high losses.

As for the brachistochrone curve problem, even though similar tests were made, no proper solution was found either. A solution found using weights of $[5.0, 1.0, 0.1]$ for constraint, physics, and goal losses is found in the next image:

“images/brach_best.png” could not be found.

similarly to the issues presented in fermat's problem, the resulting path is only a straight line that does not follow the initial conditions.

In sum, unfortunately, reproduction of the shortest path problems was not successful, and the results were not as accurate as the paper. Possibly further changes to the training parameters could help in achieving better results, or perhaps a more complex/different network architecture.

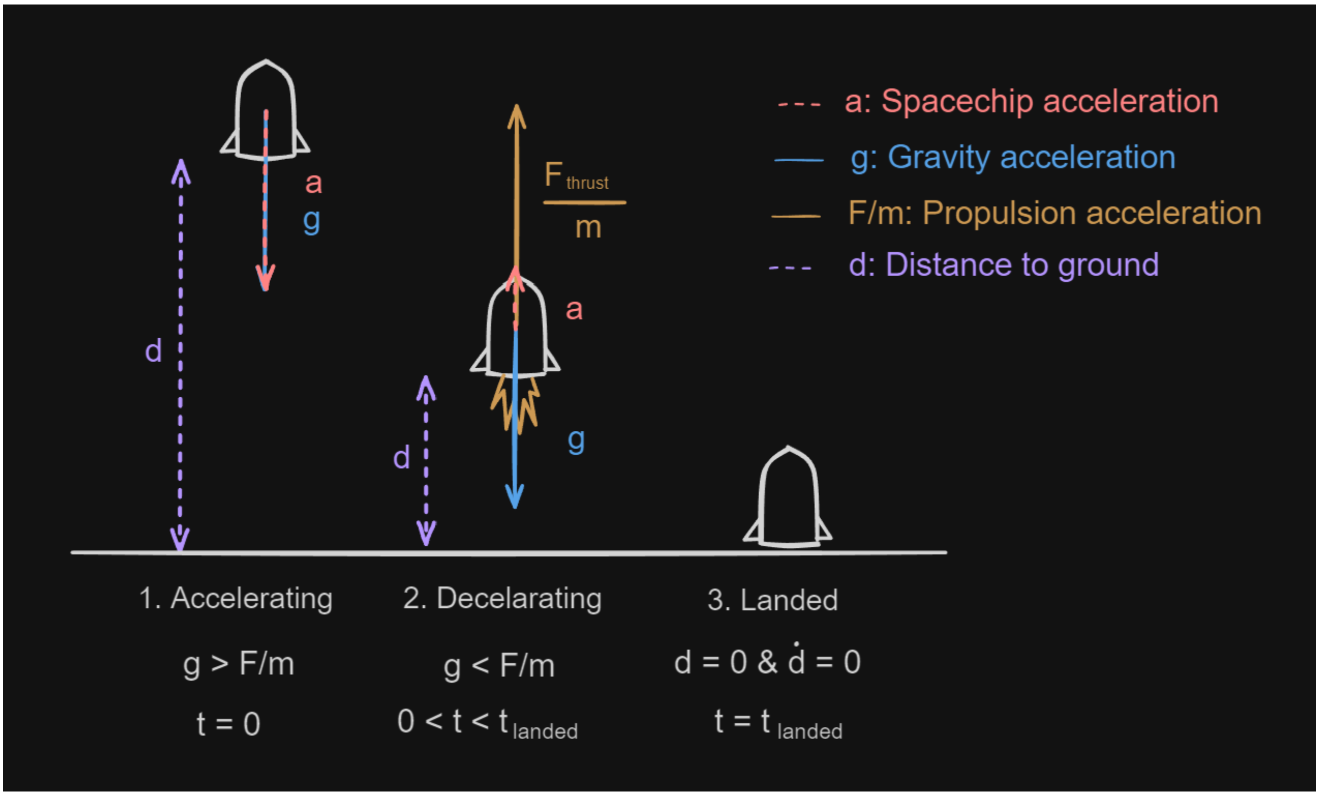
Reproduction on a new example

by Mathijs van Binnendijk

In order to see if PINN architecture proposed would extend to use cases apart from the ones used in the paper, a new example use case was derived. Requirements for the new use case were set to allow for a goal, boundary conditions, and a environment governed by physics equations. The next section will describe this new use case used as an experiment.

Method

The experiment performed was based around getting the PINN to predict the landing trajectory of a spacecraft in one dimension. A graphical description of the problem can be seen in the following figure.



The PINN takes the time t as an input and as an output it generates the propulsion force F_{thrust} and the distance to ground d for the given input time. Essentially it predicts the decent path of the landing and accompanying thrust required to follow this decent path.

Physics

The equation governing the acceleration of the spacecraft can be written down as:

$$\frac{F_{thrust}}{m} - g - \ddot{d} = 0$$

This means we can describe the residual of the governing physics equation by:

$$\mathcal{F}(t, d) = \frac{F_{thrust}(t)}{m} - g - \ddot{d}(t)$$

This can be used to evaluate the physics loss.

$$L_{phys}(d) = \frac{1}{N_\Omega} \sum_{j=1}^{N_\Omega} \|\mathcal{F}(t_j; d)\|_2^2 \text{ for } t_j \in \Omega.$$

- $L_{phys}(d)$: This is the physics loss, which quantifies the deviation of the neural network predictions from the governing physics equations.
- $\frac{1}{N_\Omega} \sum_{j=1}^{N_\Omega}$: This part calculates the average over a set of samples within the domain Ω. N_Ω represents the number of samples taken within the domain.
- $\|\mathcal{F}(t_j; d)\|_2^2$: This term measures the squared Euclidean norm of the residual function $\mathcal{F}(t_j; d)$ at each sample point t_j within the domain. Here, $\mathcal{F}(t_j; d)$ represents the residual of the governing physics equation at time t_j given by the distance d.

Conditions and constraints

The initial conditions are given by:

$$\{d, \dot{d}, \ddot{d}, F_{trust}\} = \{0, 0, -g, 0\} \text{ at } t = 0 \text{ s}$$

The maximum thrust is limited and cannot be negative meaning 0 \leq F_{thrust} \leq F_{max} . This is not enforced by implementing this boundary in the loss function but by limiting the output range of the network. The initial conditions lead to the following constraint loss function:

$$L_{const} = w_{21}d^2 + w_{22}\dot{d}^2 + w_{23}(\ddot{d} + g)^2 + w_{24}F_{thrust}^2 \text{ at } t = 0\text{s}$$

here all instances of w indicate weights.

Goal

From an initial state with height d, velocity v, and acceleration a we would like to end up at ground level ($d = 0$) with no velocity ($\dot{d} = 0$) at time t_{end}. This will be considered a successful landing. This leads to the following goal loss function:

$$L_{goal} = w_{31}d^2 + w_{32}\dot{d}^2 \text{ at } t = t_{end}$$

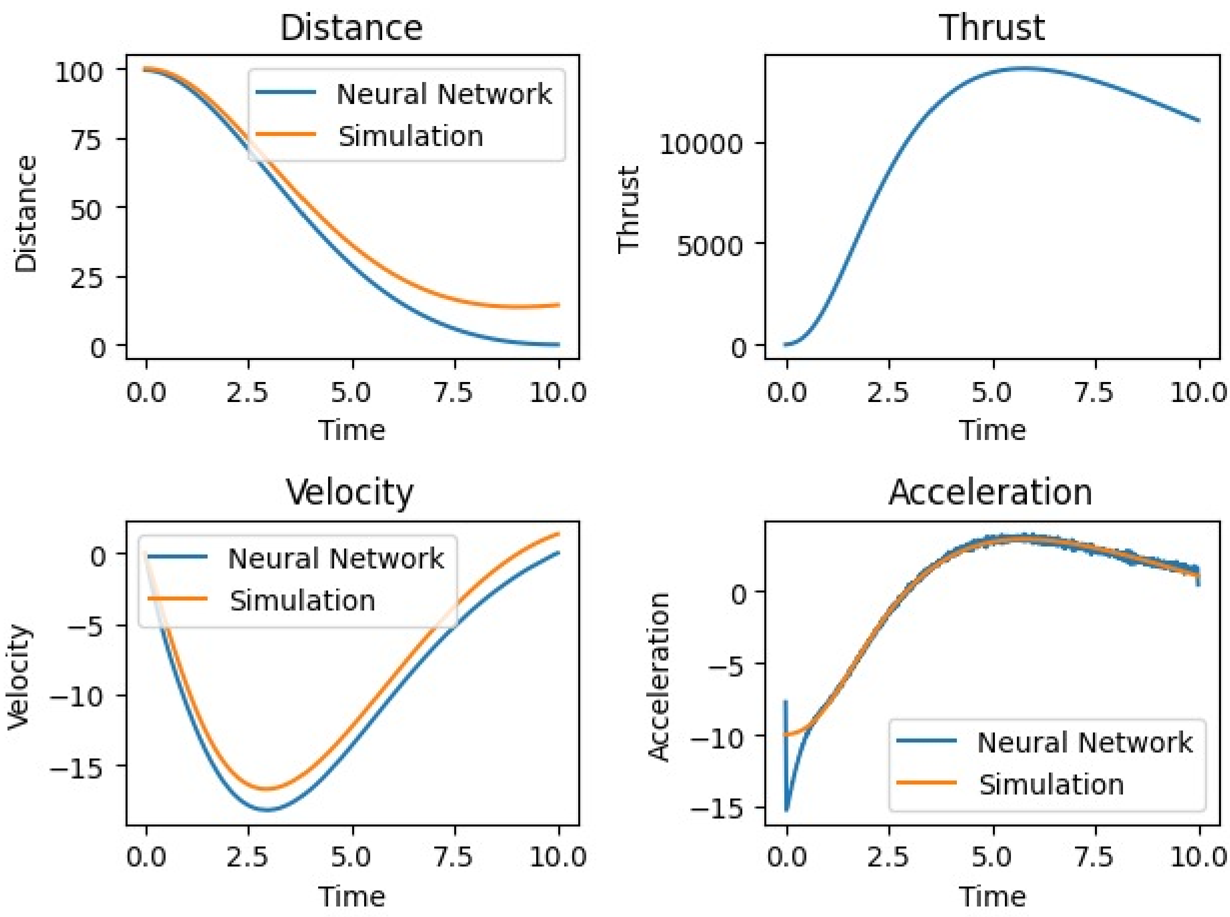
Combining all the losses

Combining the losses lead to the following optimisation problem:

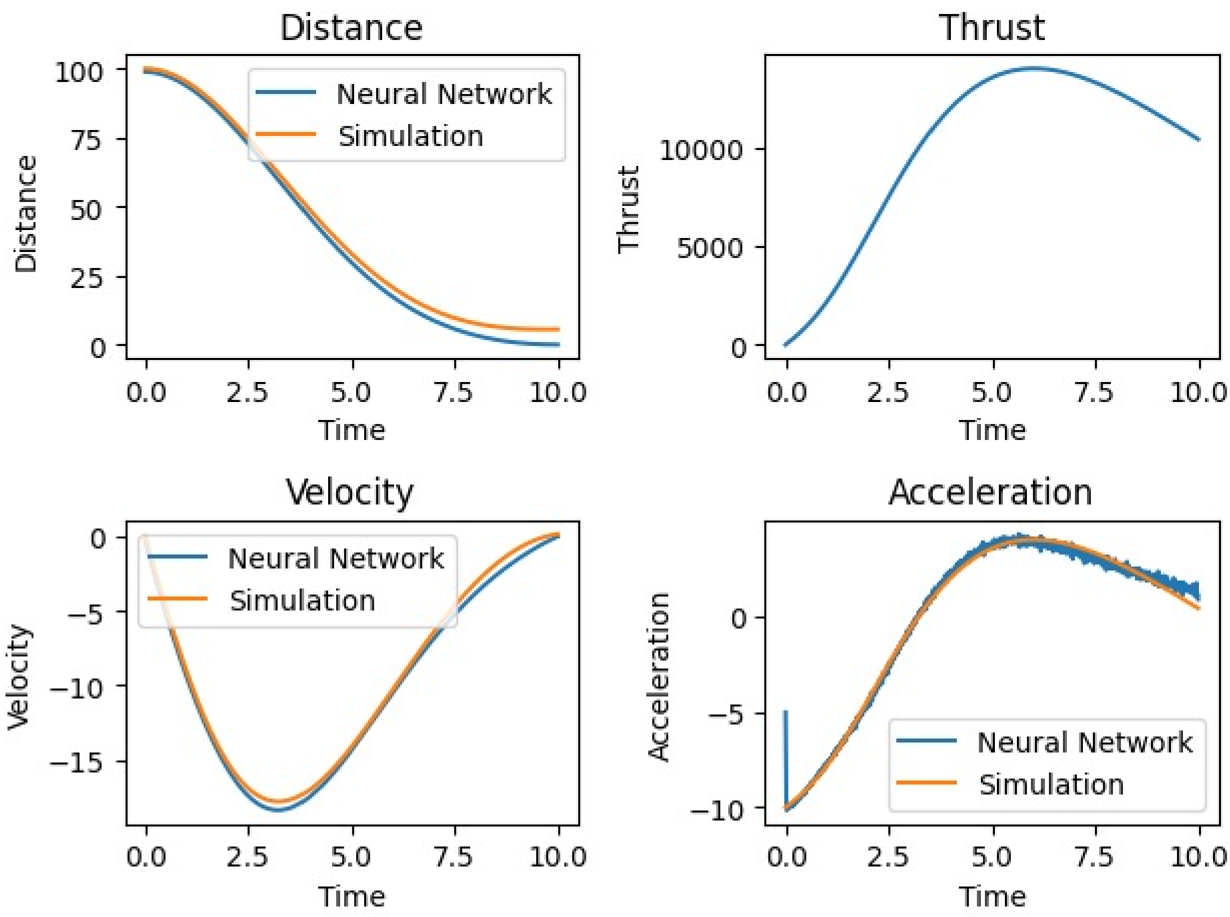
$$\text{argmin}\{w_{\text{phys}}L_{\text{phys}} + w_{\text{con}}L_{\text{con}} + w_{\text{goal}}L_{\text{goal}}\}$$

Results

The results of the PINN output are given in the figure below. In order to verify the results a simulation was carried out based on the thrust curve the PINN generated and the same initial conditions.



The results show that the simulated results roughly align with the PINN prediction. However, it is clear that there is still quite a big gap between the predicted distance to the ground and the simulated result. The acceleration plot gives a hint as to why this is potentially the case. Here we can see the predicted downwards acceleration near the beginning of the time domain does not match the simulation. This indicates the network is unable to accurately produce the distance decent at the start of the simulation. Ideally the physics loss function should prevent this initial acceleration error. This did not happen however, likely because the estimation error is averaged over all acceleration values. This means this narrow error peak near the beginning is averaged out and overall does not have a significant influence on the loss value. Increasing the weights for physics loss somewhat elevates this issue but results in worse performance for other evaluated elements such as initial conditions reproduction by the network. A strategy that did work in terms of improving this issue was to make the weights of the physics loss depend on time. Increasing the weights near the beginning of the simulation and decreasing the weights near the end. This yielded the following results where both the acceleration of the network output matched the simulation as well as the initial conditions being approximately correct. Though not perfect, this is also the best result achieved in terms of meeting the goal conditions.



A discussion on weights

Based on reproducing the results of the paper for a new experiment it became clear the technique of PINN usage for optimisation tasks is promising. However, it also shined a light on some of the limitations of this method. The main limitation seemed to be the amount of weights one is required to set in order to train the PINN successfully. The paper makes it seem like just three weights are required, one for physics, one for initial conditions, and finally one for the goal. During the experience of designing a new experiment it became clear that this is rather optimistic. For this experiment it was the case that one weight was needed for all the sub elements of the losses as well. For example, the constraint loss needed a weight for all the initial conditions, namely distance, velocity, acceleration and force. Not doing this runs the risk of only optimising conditions with overall higher residuals and leaving out he conditions with relatively little influence on the loss. Exacerbating this issue is the possibility of constant weights not being sufficient. For example, weights varying over time had to be used for the physics loss in order to punish the larger errors in estimating the acceleration near the beginning of the simulation. This exposes a different part of the weights issue, namely that multiplying an average of residuals over time with a constant weight is not a always a sufficient approach. Sometimes a variable weight over the domain might be required.

Conclusion

It is clear that deciding on weights might not be as simple as the problems the paper chooses to discuss would make it seem. However, with proper time put in tuning all different weights, the method proposed in the paper seems to be able to give good results when used on problems other than those discussed in the paper. Whether this promise translates to a larger set of problem types would need further experimentation and validation.

---

1. Seo, J. Solving real-world optimization tasks using physics-informed neural computing. Sci Rep 14, 202 (2024). https://doi.org/10.1038/s41598-023-49977-3↩