ICS435 Final Project
Physics-Informed Neural Networks

Bret Witt

11 May 2021

Introduction

1. Machine Learning for Planetary Mobility and Autonomy

   (a) The Hawaii Space Grant Consortium (HSGC) had funded my research proposal for a neural network based system identification method to learn the underlying system dynamics of rover's on planetary or lunar exploration missions. The primary objective is to pave way for adaptive autonomous exploration that is able to identify potentially hazardous terrain, most notably low cohesion sand-traps and permanently shaded or low visibility areas. By learning from previous rover state (via state estimation methods such as Kalman Filters) and control inputs, a sufficiently expressive neural network should be able to predict the state of the rover in the next timestep. In order to do this, a plugin to simulate terramechanical effects is currently being developed in Gazebo (an Open-Source robotics simulation) while infrastructure for path planning has not been discussed in depth, although one will eventually likely be made. The primary purpose of this project is to gain familiarity with the concepts beneath PINNs and a familiarity with PyTorch since this is the first time I'm using it. To make sure that my implementation is correct, I've decided to follow Raissi's example to the tee to make it easier to validate my results.

2. Physics-Informed Neural Networks

   (a) As a potential improvement over regular neural networks in system identification, the current hypothesis is that Physics-Informed Neural Networks (PINNs) [1] may perform better overall for our purposes. One key feature that we are focused on is that PINNs require low amounts of measured data. This is particularly attractive to us since it may mean we can deal with lower sensor sampling rates and sparser sensor configurations, this is especially good in the context of the private space industry which will likely prefer scalable options which reducing data processing overhead will do.

   (b) Unlike traditional feed-forward neural networks which typically exclusively take labeled data to compute the loss and its gradient during backprop, PINNs can take randomly sampled points with the occasional labeled data in order to compute the loss function. The loss function is calculated based on how much they conform to physics-based constraints.

Part I: The Non-Linear Schrodinger Equation

1. The Non-Linear Schrodinger Equation (NLSE) is the complex-valued function

$$f = ih_t + 0.5h_{xx} + |h|^2 h = 0 \tag{1}$$

   where $|h|$ is the complex norm

2. The NLSE also has the following boundary conditions when $0 < t < \pi/2$ and $-5 < x < 5$

$$h(x, 0) = 2sech(x) \tag{2}$$

$$h(5, t) = h(-5, t) \tag{3}$$

$$h_x(5, t) = h_x(-5, t) \tag{4}$$

PINN Setup

1. PINNs are structured with a function approximating prior that tries to learn the solution to a partial differential equation. In the case of learning the NLSE, we have a neural network that fits $h(x, t)$ Most of the work in PINNs is done at the loss function which in our case calculates the L2 norm between the approximated $h(x, t)$ and the specified constraint equations, for instance equations 2,3, and 4.

2. Neural network prior architecture

   (a) The neural network is identical to Raissi, it is a deep neural network using 4 hidden layers with 100 neurons and tanh activation functions

   (b) The neural network takes two real values, x and t both standing for position and time. It is not entirely clear what the units are from the paper or from any other research I've done online.

   (c) The neural network prior outputs two real values, $u(x, t)$ and $v(x, t)$ both representing the real and complex part of $h(x, t)$ respectively. The actual nature of $h(x, t)$ relates to the probability of a particle being at a position x at time t. I tried to do this as complex numbers, however, the gradient calculations were clipping off the complex part so for the sake of time I had to revert to two real valued outputs and calculate the norms and complex parts of loss functions.

3. Training data is mostly comprised of $\{x, t\}$ pairs generated using a Latin Hypercube Sampling (LHS) strategy on equation 3 and 4 while the pair $\{h, t\}$ is generated from LHS sampling of equation 2.

4. The loss function is calculated as such

$$J = J_B + J_f + J_0 \tag{5}$$

$$J_B = \frac{1}{N_B} \sum_{i=0}^{N_B} |h_i(-5, t) - h_i(5, t)|^2 + |h_i(-5, t) - h_i(5, t)|^2 \tag{6}$$

where N_B is the size of the boundary training set with pairs $\{x_B, t\}$ and $x_B \in \{-5, 5\}$

$$J_0 = \frac{1}{N_0} \sum_{i=0}^{N_0} |h_i(0, t) - h_i|^2 \tag{7}$$

where N_0 is the size of the initial training set with pairs $\{h, t\}$

$$J_f = \frac{1}{N_f} \sum_{i=0}^{N_f} |f(x, t)|^2 \tag{8}$$

where N_f is the size of the collocated training set with tr aining pairs $\{x, t\}$ This works since the PINN should be fitting a PDE of the form $f = h + D[h] = 0$ and minimizing this quantity minimizes the error of $h(t, x)$

5. All norms are calculated using $|a + bi| = \sqrt{a^2 + b^2}$, as a result I had to break up each loss function into a complex and real part. All were identical except for J_f The complex part of J_f is represented as

$$f^c = \hat{h}^c_t - 0.5\hat{h}^c_{xx} - ((\hat{h}^r)^2 + (\hat{h}^c)^2)h^r \tag{9}$$

6. The loss function is minimized with the help of the autograd functionality in PyTorch

Training

1. The successful model was trained on 20,000 epochs on an Adams optimizer with a learning rate of about 0.00025. Batch size is not applicable since data was not loaded sequentially, all training data was used in a single loss step. Experimentation was done with Stochastic Gradient Descent (SGD) as well, however, the training curve was hard to control. Learning rate was chosen based on the smoothest curve with minimal increase in loss. Raissi's paper validated their model against a Runge-Kutta integrated solution of $h(x, t)$, however that is not the case here.

2. CUDA optimization was necessary to train it in a timely manner, 20,000 epochs would've originally taken several hours but using the GPU cut it down to under half an hour

Results

1. As shown below, the results matched Raissi's remarkably well. This is of course not a surprise since almost everything about our PINNs were identical, however learning the concepts behind PINNs and PyTorch were very important to me so I'm satisfied with this result.

2. You can view the Jupyter Notebook on my GitHub profile

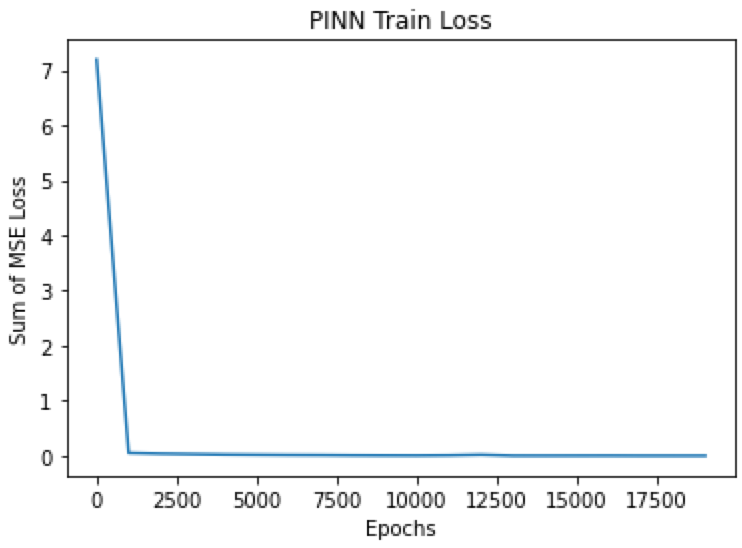

Figure 1: PINN Train Loss

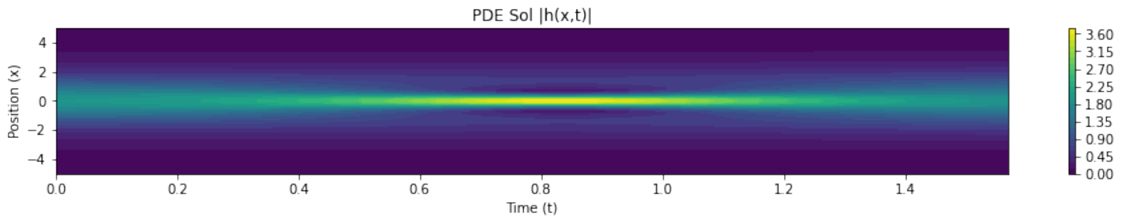

Figure 2: My Results

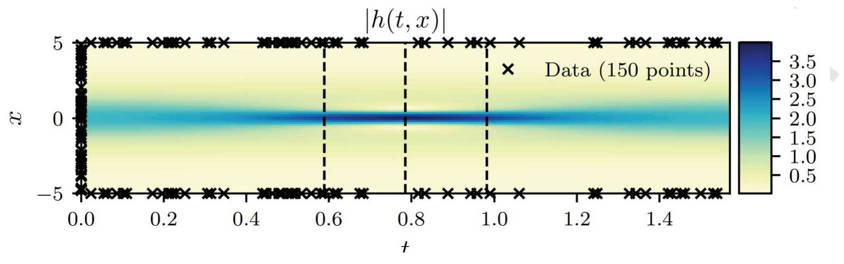

Figure 3: Raissi's Results

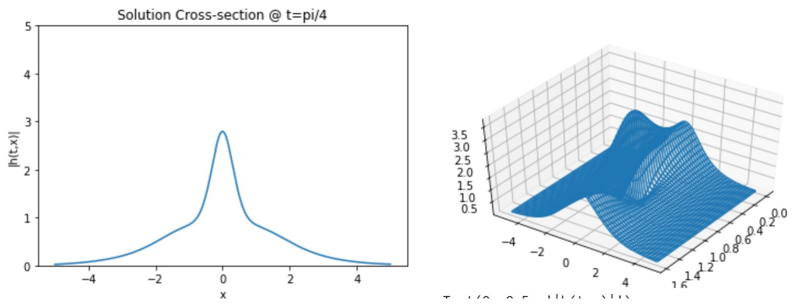

Figure 4: Cross section and 3D View of solution

Part II: Two Body Systems

Intro

1. In order to apply the knowledge of PINNs to a new problem, I decided to try to learn two-body systems that follow the inverse square law. This is especially helpful as it is falls into the class of system identification problems, so problems identified here might be pretty similar to problems identified in my project. This part of the PINN has proven to be extremely difficult and incomplete, but I intend to keep working on it through the summer

PINN Setup

1. The PINN will try to learn the trajectories r_1, r_2 of two particles stepped by timestep Δt
2. The neural network prior takes in $u^t = [r_1^t, r_2^t, \dot{r}_1^t, \dot{r}_2^t]$ then predicts $u^{t+1} = [r_1^{t+1}, r_2^{t+1}, \dot{r}_1^{t+1}, \dot{r}_2^{t+1}]$
3. The neural network uses 4 hidden layers with 100 neuron heights each and tanh activation functions. The PINN also uses an Adams optimizer with learning rate 0.0003. Learning rate was optimized by looking at the loss graph and fine tuning it for the smoothest curve. Before getting more positive results I don't intend on messing with the hyperparameters too much.
4. The loss function uses the following physics-based constraints
   (a) Newton's Third Law and Law of Gravitation

$$f_{21} = m_1\ddot{r_1} + \frac{Gm_1m_2}{|r_2 - r_1|^2}\hat{r}_{21} = 0 \tag{10}$$

$$f_{12} = m_2\ddot{r_2} + \frac{Gm_1m_2}{|r_1 - r_2|^2}\hat{r}_{12} = 0 \tag{11}$$

   (b) Conservation of Energy

$$\epsilon_1 = \frac{1}{2}m_1\dot{r_1}^2 - \frac{Gm_1m_2}{|r_1 - r_2|} \tag{12}$$

$$\epsilon_2 = \frac{1}{2}m_2\dot{r_2}^2 - \frac{Gm_1m_2}{|r_2 - r_1|} \tag{13}$$

$$\epsilon_{tot} = \epsilon_1 + \epsilon_2 \tag{14}$$

$$\dot{\epsilon}_{tot} = 0 \tag{15}$$

(c) Conservation of Angular Momentum

$$L = r \times mv \tag{16}$$

$$\dot{L} = 0 \tag{17}$$

(d) Discreteness is introduced by approximating acceleration over the time-step we want before being used to calculate the constraint in equation 10. This seemed to work, however we might remove this in favor of the method introduced by Raissi in the discrete-time inference section of the paper which uses two neural network priors

$$a_i = \frac{\dot{r}_i^{t+1} - \dot{r}_i^{t}}{\Delta t} \tag{18}$$

5. The loss function uses the MSE of all these constraints

Training Data

1. For now the training data is relatively simple, using LHS strategies, training pairs of particles positions and their velocities are generated. Positions are bounded around (-3,-3,-3) to (3,3,3) and velocities are bounded along (-5,-5,-5) and (5,5,5) All units are in meters and seconds.
2. I have also experimented with adding more imbalanced data such as two particles extremely close to each other (we'll need a softening constant later), however discarded it for simplicity sakes with an eye on bringing it back later.
3. Unlike the example, I took the liberty to split the data into a training and validation set. The ratio is 1/3 for validation and 2/3 for training. The main motivation is of course to ensure that the PINN is generalizing since we are training a general solution to the two body problem.

Training

1. The training loss seems to be heavily dependent on the Δt seen in equation 18, although that's not of any particular surprise, for our purposes we set it to around 0.1 with an average validation loss of 0.0025 over 10,000 epochs and a sample size of 20,000 training pairs. The validation loss never indicated overfitting despite the high epochs.
2. Masses for both particles are set to 1 and for purposes of debugging and avoiding normalization G = 1
3. The result as of now isn't particularly impressive, it does exhibit some behavior you'd see in a two-body system when lucky, however most of the time they avoid each other rather than going straight for each other. There is also other strange behavior where they will mirror each other or follow each-others paths but not bee-line straight for the other particle as you'd expect. This could mean there's a bug, but that is a determination for later

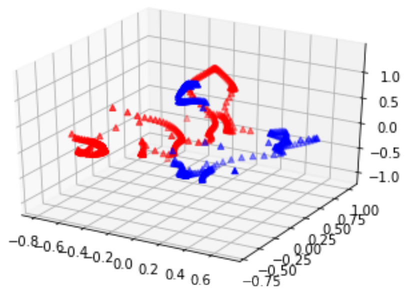

Figure 5: Trajectory of two bodies, according to a really bad PINN

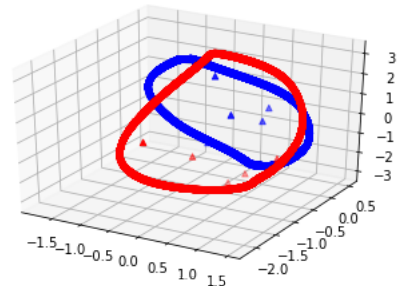

Figure 6: A more believable result

References

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". en. In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125 (visited on 05/16/2022).