
Contents

1	Introduction	9
2	Background	13
2.1	Classical Mechanics	13
2.1.1	Harmonic Oscillator	14
2.1.2	Analytical solution of Ordinary Differential Equations	15
2.1.3	Numerical solution of Ordinary Differential Equations	18
2.1.4	Solving the harmonic oscillator	20
2.1.5	Langrange and Hamiltonian Equations	21
2.1.5.1	Lagrange Equations	22
2.1.5.2	Hamiltonian Equations	25
2.2	Neural Network Architectures	26
2.2.1	Deep forward Network - Multilayer Perceptron	26
2.2.2	Activation functions and Loss functions	28
2.2.3	Recurrent Neural Networks	29
2.2.4	Optimizers	31
2.2.5	Backpropagation of Neural Models	31
3	Datasets	33
3.1	Oscillator	33
3.2	N-body problem	34
3.2.1	Symplectic numerical methods to solve N-body problem	35
3.2.2	Twobody problem	36
3.2.3	Three body problem	39
3.3	N Pendulum	41
4	Methods	44
4.1	New paradigm - NeuralODE	44
4.2	Recurrent time steppers	45

4.3	Hamiltonian Neural Network	45
4.4	Graph Neural Network	46
4.4.1	Convolutional Graph Neural Network	48
4.4.2	Gated Attention Network	49
4.5	Graph Hamiltonian Neural Network	49
4.6	Improved Graph Hamiltonian Neural Network	51
4.7	Gated Recurrent Graph Hamiltonian Neural Network - GRUGHNN	51
4.8	Training for Physics Informed Models based on ODE	53
5	Experiments	55
5.1	Experimentation of the Models on the Datasets	55
5.1.1	Harmonic Oscillator	56
5.1.2	Twobody-problem	59
5.1.3	Three-body problem	63
5.2	Experimentation of Physics informed datasets on three body Dataset	67
5.2.1	Threebody dataset	68
5.2.2	Figure 8 solutions	69
5.3	Experimentation of Physics informed models on N body problem and N pendelum	72
5.3.1	N-body pendulum	72
5.3.2	N-body problem	74
6	Conclusions	79

List of Figures

1.1	PINN model for Laplace Equation	10
1.2	HNN model with integration solver step	12
2.1	Multilayer perceptron[6]	27
2.2	ReLU activation	28
2.3	Tanh activation	29
2.4	Recurrent neural unit(many to many)[16]	30
2.5	Gated recurrent unit[33]	31
3.1	Pendelum with 3 degrees of freedom in pybullet	43
4.1	Architecture of time-stepper model	46
4.2	Graph representation	47
4.3	Architecture of Graph Neural Hamiltonian Network	50
4.4	Architecture of improved GHNN	52
4.5	Architecture of GRUGHNN	53
5.1	Losses and energy accuracy on various neural models(Oscillator)	58
5.2	Evaluation of the sample on the models, Trajectory and Energy(Oscillator)	59
5.3	Losses and energy accuracy on various neural models(Oscillator)	61
5.4	Evaluation of the sample on the models, Trajectory and Energy(Twobody)	62
5.5	Evaluation of the sample on the models, phase space (Twobody)	63
5.6	Losses and energy accuracy on various neural models(Threebody)	65
5.7	Evaluation of the sample on the models, Trajectory and Energy(Threebody)	66
5.8	Evaluation of the sample on the models, phase space (Threebody)	67
5.9	Losses at various pinn models (threebody dataset)	69
5.10	Trajectories , phase spaces and hamiltonian at various pinn models	69
5.11	Losses at various models (figure dataset)	71
5.12	Trajectories , phase spaces and hamiltonian at various models(figure dataset)	71

5.13 Losses till 500 epochs(Pendulum)	73
5.14 Losses till 1000 epochs on GRUGHNN(Pendulum)	74
5.15 trajectories of pendelums after 1000 epochs	75
5.16 Losses till 1000 epochs on GRUGHNN (Nbody)	76
5.17 Trajectories of nbody after 1000 epochs	77
5.18 Hamiltonian of nbody cases	78



List of Tables

5.1	Figure 8 Dataset samples[38]	70
5.2	Evaluation of 3dof sample on various models and training procedures	73
5.3	Evaluation of 4dof sample on various models and training procedures	73
5.4	Evaluation of 3dof sample on GRUGNN and training procedures	73
5.5	Evaluation of 4dof sample on GRUGNN models and training procedures,	74
5.6	Evaluation of 3dof sample on GRUGNN and training procedures, (Nbody)	76
5.7	Evaluation of 4dof sample on GRUGNN and training procedures (Nbody)	76

+

Abstract

The introduction of the adjoint method for NeuralODEs[9] has created new ways for physics-based trajectory predictions. In the subsequent chapters of this thesis, we look at deep learning models and methods targeted at predicting physical systems' behavior. We use the guiding light of Hamiltonian equations to dive into systems such as the harmonic oscillator and the three-body problem. We are intrigued with exploring graph neural networks' ability to be used in physics informed models: in the N-body problem and in the N-pendulum. This work tests the ability of graph neural networks to generalize the behavior from single realizations of elements at the node level of the graph. In this way, we hope to exemplify how such models can capture highly dynamic physical processes.

1 Introduction

The intersection of physics and deep learning has never been more pronounced than it is today. With advancements in hardware and computational capabilities, we are now positioned to predict physical phenomena with unprecedented precision using deep learning techniques, particularly through physics-informed neural networks.

Physics-Informed Neural Networks (PINN) are neural networks that encode model equations, like Partial Differential Equations (PDE), as a component of the neural network itself[11]. For example let us take Laplace equation which is defined as

$$\Delta u(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega, \mathbf{x} \in R^2 \quad (1.1)$$

$$u = S \quad \mathbf{x} \in \partial\Omega \text{ Dirichlet boundary condition,} \quad (1.2)$$

$$\frac{\partial u}{\partial \mathbf{n}} = f(\mathbf{x}) \quad \mathbf{x} \in \partial\Omega \text{ Neumann boundary condition.} \quad (1.3)$$

The Operator Δ is called as Laplace Operator and it defines $\Delta = \sum_i \frac{\partial^2}{\partial^2 x_i}$.

The Ω is domain and $\partial\Omega$ is boundary of the domain. To build a PINN model we will use an neural network which will learn the solution u and from that output we will calculate all needed derivatives, residuals and other parts for the optimization.

In figure 1.1 we can observe the architecture of such physics informed model. The Loss functions that we need to use to properly train our network are

$$\text{Loss}_s(\Theta) = \frac{1}{N} \sum_i^N (u_{\Theta}(\mathbf{x}_i) - u_i)^2, \quad (1.4)$$

$$\text{Loss}_r(\Theta) = \frac{1}{R} \sum_i^R \left(\frac{\partial u_{\Theta}(\mathbf{x}_i)}{\partial x_i} - f(\mathbf{x}_i) \right)^2, \quad (1.5)$$

$$\text{Loss}_l(\Theta) = \Delta u_{\Theta}, \quad (1.6)$$

$$\text{Loss}(\Theta) = \omega_s \text{Loss}_s + \omega_r \text{Loss}_r + \omega_l \text{Loss}_l. \quad (1.7)$$

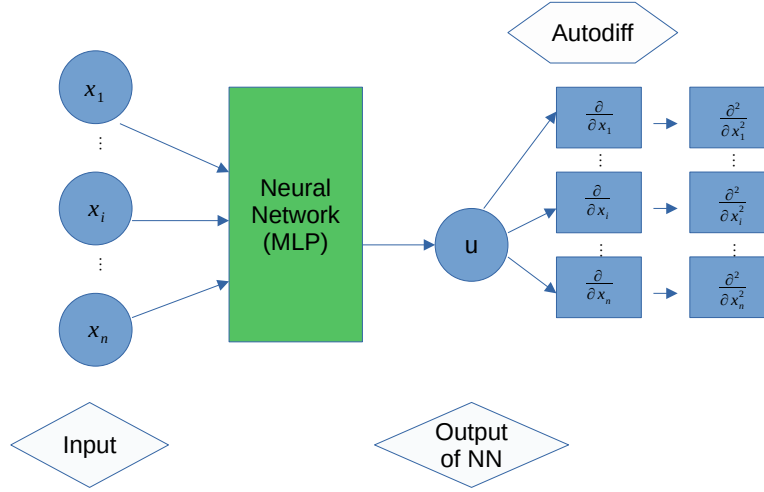


Figure 1.1: PINN model for Laplace Equation

The laplace equation is often used to calculate or describe physical phenomena which are bounded to domain, like electrical fields. The domain Ω needs to be discretized. With given discretization we divide R as number of \mathbf{x}_i which are on the boundary of the domain and N the Number of all \mathbf{x}_i in Ω . The $\omega_s, \omega_r, \omega_l$, in this case are coefficients $0 \leq \omega \leq 1$ and they are there to improve optimization capabilities which is defined as

$$\Theta^* = \arg \min \text{Loss}(\Theta). \quad (1.8)$$

The Θ are learnable parameters which can be updated with Θ^* trough optimization algorithm.

Such model wouldn't be possible without automatic differentiation[36] which is offered from machine learning library `pytorch`[37]. In this example we need a large amount of data, and the optimization of the model could be very slow, especially if we don't have

a computation capable hardware for it. Obtaining the data can be done through precise measurement or using some numerical solvers like FEM[26] or Isogeometric Analysis[35]. Keep in mind that there are Poisson Equation, Wave Equation which are time-dependent t .

Those PDEs are mostly applied in electrotechnical (Electro-magnetical fields) and civil Engineering (static in construction). In this thesis we won't work on the PDEs but in similar vein, we will try to train a model to predict the dynamics of some physical phenomena and non-/holonomic hamiltonian systems.

The dynamics for robots called Joint Space Dynamics are defined as second order ordinary differential equation[22] .

$$\mathbf{M}\ddot{\mathbf{x}}(t) + \mathbf{D}(\dot{\mathbf{x}}(t)) + \mathbf{C}(\mathbf{x}(t)) = \tau, \quad (1.9)$$

but there are other forms, as Lagrange Equations of Motion if our system is holonomic[3]. Lagrange function are neat idea to find dynamics of the system in the joint space and it is based on derivatives of Kinetic T and Potential U Energy

$$\mathcal{L} = T - V, \quad (1.10)$$

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = 0. \quad (1.11)$$

The Lagrange Equations are used in Deep Langrangian Networks[29]. In similar manner, there are Hamiltonian Equations of Motion. They work for holonomic and non-holonomic hamiltonian systems and they are capable to build first order ordinary equation

$$\dot{x} = f(x, t) \quad (1.12)$$

or in Hamiltonian Form where $\mathcal{H} = T + V$

$$\dot{\mathbf{z}} = \mathbf{J} \frac{\partial \mathcal{H}}{\partial \mathbf{z}}(\mathbf{z}) \quad (1.13)$$

where $\mathbf{z} = [\mathbf{q}, \mathbf{p}]^T$ and $\mathbf{J} = \begin{bmatrix} 0 & \mathbf{I}_n \\ -\mathbf{I}_n & 0 \end{bmatrix}$.

One of the neural models which use hamiltonian equations is introduced in the [19] and it is called Hamiltonian Neural Network. Its architecture you can observe in figure 1.2. We will make it more interesting introducing the Graph Neural Network instead Conventional Multilayer Perceptron and try to use it as a base of our architecture. We hypothesize that implementing a Graph Neural Network could improve the architecture's performance.

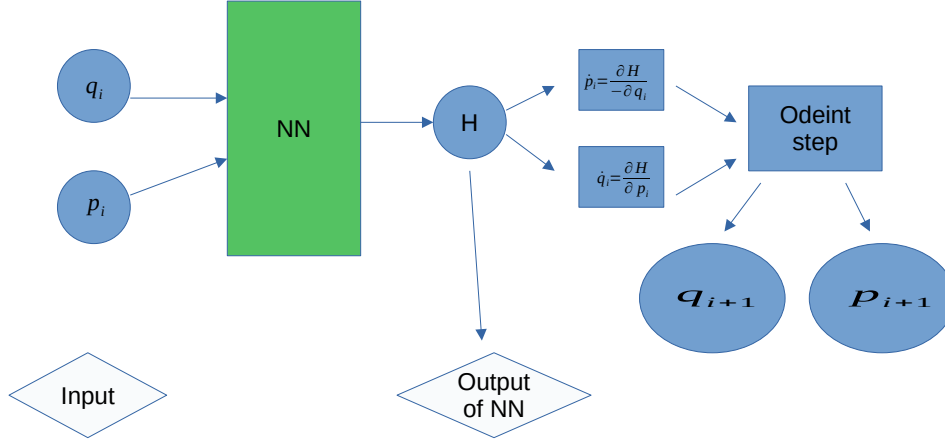


Figure 1.2: HNN model with integration solver step

In this thesis, we will also discuss Hamiltonian equations and how to derive them. During our research, we encountered the issue that there are no high-quality datasets available for training physics-based neural networks. Given this, we will explain how to create datasets based on Hamiltonian Equations of Motion.

Next we will revisit state of the art neural architectures and test NeuralODE as new Neural Paradigm, and how they learn dynamics from given data.

In the experiments we tested the capabilities of Graph Neural Networks based PINN Models, how do they adapt to diverse training data and their possibility to reduce or increase a degree of freedom in the training. For example how similar is the movement between 3 and 4 bodied pendulum on the same trained set of neural parameters. We hope that we will give valuable insight in those topics.

2 Background

In this chapter, we revisit some of the important concepts from Classical Mechanics and further develop our understanding of common neural network architectures. This is a critical foundation which will be required to understand how Physics-Informed Neural Networks work, as discussed in greater detail in the Methods chapter.

2.1 Classical Mechanics

Classical Physics presents the basics for understanding the laws presiding over the behavior of objects with mass in our physical reality. This aspect of the science of Classical Mechanics describes how mass moves in space: motion is the result of forces applied to matter. Moving on, Sir Isaac Newton did develop laws that describe and predict the behavior of such objects, which today are called the Newton's Laws of Motion.[31]

- Newton's 1st law: Unless acted on by an outside force the natural motion of an object is constant velocity
- Newton's 2nd law: The effect of an applied force \mathbf{F} upon an object of mass m is to induce an acceleration \mathbf{a} such that

$$\mathbf{F} = m\mathbf{a}$$

If mass is constant we can introduce momentum \mathbf{p} .

$$\mathbf{F} = \frac{d\mathbf{p}}{dt}$$

The derivative of momentum with respect to time is force. With this equation we can obtain momentum as

$$\mathbf{p} = m\mathbf{v}$$

where \mathbf{v} is velocity.

- Newton's 3rd law: If an object applies a force \mathbf{F} on a second object, then the second object applies an equal and opposite force $-\mathbf{F}$ on the first object.
In the physics it is most famous law and it is often referenced as *actio = reactio* from latin.

One of the simplest systems that adheres to these laws is the harmonic oscillator.

2.1.1 Harmonic Oscillator

Harmonic Oscillator is system in classical physics which describes the situation where body is displaced from equilibrium position and it experience the restoring force \mathbf{F} proportional to its displacement.[7][18] This system is described with Ordinary Differential Equation of second order :

$$m\ddot{\mathbf{x}}(t) = -k\mathbf{x}(t), \quad \mathbf{x}(0) = \mathbf{x}_0 \quad (2.1)$$

We will consider this system as one dimensional $\mathbf{x} = x$ along straight line which produces simple harmonic motion [7]. It is built with the spring stiffness k and mass m connected to it. To show the simple harmonic motion let us solve the equation.

First we will build the the linear system of equations to get the first order ordinary differential equation in form:

$$\dot{\mathbf{z}} = \mathbf{A}\mathbf{z} + \mathbf{b}. \quad (2.2)$$

We are beginning with parameter z :

$$z_1 = x \quad (2.3)$$

$$z_2 = \dot{x} \quad (2.4)$$

$$\dot{z}_1 = \dot{x} = z_2 \quad (2.5)$$

$$\dot{z}_2 = \ddot{x} = -\frac{k}{m}z_1 \quad (2.6)$$

With this we got:

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad (2.7)$$

We got differential ordinary equation which is homogeneous ($\mathbf{b} = 0$). To solve this Problem we need to refresh the knowledge of solving the ODEs

2.1.2 Analytical solution of Ordinary Differential Equations

A Differential Ordinary Equation of the first order is defined as:

$$\dot{\mathbf{z}} = \mathbf{A}\mathbf{z} + \mathbf{b}u, \quad \mathbf{z}(0) = \mathbf{z}_0 \quad (2.8)$$

Before we show analytical solution of mentioned linear system, let us solve first the simplest case[1]:

$$\dot{z}(t) = az(t) + bu, \quad z(0) = z_0 \quad (2.9)$$

Every solution of ODE has homogeneous z_{hom} and particular solution z_{par}

$$z(t) = z_{hom} + z_{par}. \quad (2.10)$$

When we calculate de homogeneous solution we set $b = 0$ and we precede with simple integration.

$$\int \frac{dz}{z} = \int a dt \quad (2.11)$$

$$z_{hom} = c \exp(at) \quad (2.12)$$

To find particular solution we will create it with variation of the constants method.
We define:

$$z_{part} = c(t) \exp(at) \quad (2.13)$$

and we will put it in the differential equation

$$\dot{z}_{part}(t) = az_{part}(t) + bu(t), \quad (2.14)$$

$$a \exp(at)c(t) + \exp(at)\dot{c}(t) = a \exp(at)c(t) + bu(t), \quad (2.15)$$

$$\dot{c}(t) = \exp(-at)bu(t). \quad (2.16)$$

After integration of \dot{c} we get:

$$z_{part} = c(0) \exp(at) + \int_0^t \exp(at - \tau)bu(\tau)d\tau \quad (2.17)$$

With inclusion of the Initial condition $z(0) = z_0$ we get final form of the solution

$$z = \underbrace{z_0 \exp(at)}_{\text{homogeneous}} + \underbrace{\int_0^t \exp(a(t - \tau))bu(\tau)d\tau}_{\text{particular}}. \quad (2.18)$$

Let so go back and solve the equation 2.8.
 Obtaining the solution goes similar to simple case but it is vectorized.
 The solution is made by homogeneous and particular solution as above.

$$\dot{\mathbf{z}} = \mathbf{z}_{hom} + \mathbf{z}_{part} \quad (2.19)$$

Lets assume from the example what we have done previously that our homogeneous solution is defined as:

$$\mathbf{z}_{hom} = \exp(\mathbf{A}t)\mathbf{c} \quad (2.20)$$

This is easy to prove trough Taylor series of the $\exp(\mathbf{A}t)$.

$$\exp(\mathbf{A}t) = \sum_{i=0}^{\infty} \frac{(\mathbf{A}t)^i}{i!} = \mathbf{I} + \mathbf{A}t + \mathbf{A}^2 \frac{t^2}{2!} + \mathbf{A}^3 \frac{t^3}{3!} + \mathcal{O}(t^4) \quad (2.21)$$

$$\frac{d \exp(\mathbf{A}t)}{dt} = \mathbf{A} + \mathbf{A}^2 t + \mathbf{A}^3 \frac{t^2}{2!} + \mathcal{O}(t^3) = \quad (2.22)$$

$$\mathbf{A} \left(\mathbf{I} + \mathbf{A}t + \mathbf{A}^2 \frac{t^2}{2!} \right) + \mathcal{O}(t^3) \quad (2.23)$$

$$\mathbf{A} \exp(\mathbf{A}t) \quad (2.24)$$

Lets put it in the equation

$$\dot{\mathbf{z}}_{hom} = \frac{d \exp(\mathbf{A}t)}{dt} \mathbf{c} = \quad (2.25)$$

$$\mathbf{A} \exp(\mathbf{A}t) \mathbf{c} = \quad (2.26)$$

$$\mathbf{A} \mathbf{z}_{hom} \quad (2.27)$$

and the homogeneous solution

$$\boxed{\mathbf{z}_{hom} = \exp(\mathbf{A}t)\mathbf{c}} \quad (2.28)$$

fulfills the equation.

With the variation of the constant method we obtain the particular solution:

$$\exp(\mathbf{A}t)\mathbf{c}(0) + \int_0^t \exp(\mathbf{A}(t - \tau))\mathbf{b}u(\tau)d\tau \quad (2.29)$$

With inclusion of the initial condition $z(0) = z_0$ we get final form of the solution

$$\mathbf{z} = \underbrace{\exp(\mathbf{A}t)\mathbf{x}_0}_{\text{homogeneous}} + \underbrace{\int_0^t \exp(\mathbf{A}(t-\tau))\mathbf{b}u(\tau)d\tau}_{\text{particular}}. \quad (2.30)$$

In our work the most important part is homogeneous solution and to obtain that we should use standard practice to solving it.

- Find eigenvalues and eigenvectors: Every matrix with full rank is diagonalizable

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1} \quad (2.31)$$

$$\mathbf{D} = \text{diag}(\{\lambda_1, \dots, \lambda_i, \dots, \lambda_n\}) \text{ Eigenvalues} \quad (2.32)$$

$$\mathbf{V} = [\mathbf{v}_1 | \dots | \mathbf{v}_i | \dots | \mathbf{v}_n] \text{ Eigenvectors} \quad (2.33)$$

It is easily obtained through characteristic polynomial $p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}) = 0$

- To find general solution we need to transform the $\dot{\mathbf{z}} = \mathbf{A}\mathbf{z}$

$$\dot{\mathbf{z}} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}\mathbf{z} \quad (2.34)$$

$$\underbrace{\mathbf{V}^{-1}\dot{\mathbf{z}}}_{\Theta} = \mathbf{D}\underbrace{\mathbf{V}^{-1}\mathbf{z}}_{\Theta} \quad (2.35)$$

$$\dot{\Theta} = \mathbf{D}\Theta \quad (2.36)$$

This system is now trivial to solve.

- Set it in general solution $\Theta_i = c_i \exp(\lambda_i t)$ In case of repeated eigenvalue we should use $\Theta_i = c_i \frac{t^r}{r!} \exp(\lambda_i t)$ where r is number of repetitions.
- back-substitute \mathbf{z}

$$\mathbf{z} = \mathbf{V}\Theta = \sum_{i=1}^n \mathbf{v}_i \Theta_i \quad (2.37)$$

$$\mathbf{z} = \sum_{i=1}^n \mathbf{v}_i \underbrace{c_i \exp(\lambda_i t)}_{\Theta} \quad (2.38)$$

- apply initial conditions

$$\mathbf{z}(0) = \sum_{i=1}^n \mathbf{v}_i c_i = \mathbf{V}\mathbf{c} = \mathbf{z}_0 \quad (2.39)$$

$$\mathbf{c} = \mathbf{V}^{-1}\mathbf{z}_0 \quad (2.40)$$

In this procedure we obtained the homogeneous solution. The particular solution is in some cases trivial but in some not solvable analytically due to $\mathbf{b}u(t)$. It is possible that it is nonlinear function. In those cases we use numerical methods to obtain the solutions.

2.1.3 Numerical solution of Ordinary Differential Equations

For every ODE there is solution, sometimes there is no analytical solution but we can obtain numerical one through numerical integrators. Still numerical solutions are only approximations of the analytical solution. Due to their stability we differ two categories of one-step methods and that are explicit and implicit methods.

- Explicit Methods are trivial to calculate because the next step is dependent only on the previous step:

$$x(t_{i+1}) = x(t_i) + h\Phi(x(t_i)), t_i) \quad (2.41)$$

- Implicit Methods are challenging because the next step is dependent on itself.

$$x(t_{i+1}) = x(t_i) + h\Phi(x(t_{i+1})), t_{i+1}) \quad (2.42)$$

To obtain the step it is needed to know method function Φ and solving a next step through some method like for example Einstein-Raphson method.

To assure the stability and correctness of the methods we need to set some properties. The properties that the every one-step method should have, are Consistency and Convergence[24].

The Consistency is proven over local discretization error. Let say that for $(x, y) \in G$ and $\mu = \mu(\eta)$ is a solution for the ODE $\mu' = f(\eta, \mu)$, $\mu(x) = y$, then the discretisation error is

$$le(x, y; h) = \mu(x + h) - \mu(x) - h\Phi(x, y; h) \quad (2.43)$$

and discretisation error per step

$$\Delta(x, y; h) = \frac{le(x, y; h)}{h}. \quad (2.44)$$

A one-step method is consistent if

$$\lim_{h \rightarrow 0} \Delta(x, y; h) = 0. \quad (2.45)$$

For the method to be convergent need to be consistent. The general definition of convergency is

$$\lim_{h \rightarrow 0} \Phi(x, y; h) = f(x, y) \quad (2.46)$$

To prove this we should look for the consistency and convergency rank.

- Consistency rank p

$$||\Delta(x, y; h)|| \leq Kh^p \quad (2.47)$$

- Convergency rank p

$$E(h) = \max_{j=0,1,\dots,N} ||y_j - y(x_j)|| \text{ global discretisation error} \quad (2.48)$$

$$\lim_{h \rightarrow 0} E(h) = 0 \text{ Convergence} \quad (2.49)$$

$$E(h) \leq Kh^p \quad (2.50)$$

This properties are highly important to know because we will use for the experiments mostly the Runge-Kutta methods which are based on those properties.

Most important for us are

- Euler method

$$x_{i+1} = x_i + hf(x_i, t_i) \quad (2.51)$$

- Runge-Kutta 4 Method(RK4)

$$K_1 = f(x_i, t_i) \quad (2.52)$$

$$K_2 = f(x_i + \frac{h}{2}K_1, t_i + \frac{h}{2}) \quad (2.53)$$

$$K_3 = f(x_i + \frac{h}{2}K_2, t_i + \frac{h}{2}) \quad (2.54)$$

$$K_4 = f(x_i + hK_3, t_i + h) \quad (2.55)$$

$$x_{i+1} = x_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \quad (2.56)$$

- Dormand- Prince method is most accurate method for obtaining he solutions of ODEs[5]. It is widely used in experiments.
This butcher shema builds the DOPRI5 method

$$\begin{array}{cccccc}
 0 & & & & & \\
 \frac{1}{3} & \frac{1}{3} & & & & \\
 \frac{3}{10} & \frac{4}{45} & \frac{9}{40} & & & \\
 \frac{4}{5} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{32}{9} & & \\
 \frac{8}{9} & \frac{9017}{3168} & -\frac{355}{33} & \frac{64448}{46732} & -\frac{212}{49} & \\
 1 & \frac{35}{384} & 0 & \frac{5247}{500} & \frac{176}{125} & -\frac{5103}{18656} \\
 1 & \frac{35}{384} & 0 & \frac{113}{500} & \frac{192}{125} & -\frac{2187}{6784} \\
 \hline
 & \frac{35}{384} & 0 & \frac{113}{500} & \frac{192}{125} & -\frac{2187}{6784}
 \end{array} \quad (2.57)$$

2.1.4 Solving the harmonic oscilator

In one of the previous subsections we obtained the first order ODE for harmonic oscilator and we described how it should be solved.

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \mathbf{z}_0 = \begin{bmatrix} x_0 \\ v_0 \end{bmatrix} \quad (2.58)$$

Through the system Matrix $A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix}$ we got complex eigenvalues, which are $\lambda_1 = i\sqrt{\frac{k}{m}}$ and $\lambda_2 = -i\sqrt{\frac{k}{m}}$.

General solution of this problem is

$$z_1 = x = c_1 \exp(i\sqrt{\frac{k}{m}}t) + c_2 \exp(-i\sqrt{\frac{k}{m}}t) \quad (2.59)$$

With applying of the euler formula:

$$\exp(iat) = \cos(at) + i \sin(at) \quad (2.60)$$

We get simplified solution

$$z_1 = x = (c_1 + c_2) \cos\left(\sqrt{\frac{k}{m}}t\right) + (c_1 - c_2)i \sin\left(\sqrt{\frac{k}{m}}t\right) \quad (2.61)$$

$$z_2 = \dot{x} = -(c_1 + c_2)\sqrt{\frac{k}{m}} \sin\left(\sqrt{\frac{k}{m}}t\right) + (c_1 - c_2)\sqrt{\frac{k}{m}}i \cos\left(\sqrt{\frac{k}{m}}t\right). \quad (2.62)$$

Now we can apply initial conditions

$$x(0) = x_0 = c_1 + c_2, \quad (2.63)$$

$$\dot{x}(0) = v_0 = i(c_1 - c_2). \quad (2.64)$$

The velocity in our cases is an real number which we can assume that constants c_1 and c_2 are equal. This simplifies our equations and we can substitute $x_0 = c_1 + c_2$

$$z_1 = x = x_0 \cos \left(\sqrt{\frac{k}{m}} t \right), \quad (2.65)$$

$$z_2 = \dot{x} = -x_0 \sqrt{\frac{k}{m}} \sin \left(\sqrt{\frac{k}{m}} t \right) \quad (2.66)$$

The is harmonic at we can se that clearly because it is periodic and we can read an angle velocity of periodic movement $\omega = \sqrt{\frac{k}{m}}$,

$$z_1 = x = x_0 \cos (\omega t) \quad (2.67)$$

$$z_2 = \dot{x} = -x_0 \omega \sin (\omega t) \quad (2.68)$$

2.1.5 Langrange and Hamiltonian Equations

In robotics, research and model creation of the robot models are not only one-bodied system. Mostly they are are rigid-body systems with own joint space, like a manipulator which has N degrees of freedom. Every of the joints and links have own properties like mass, friction, etc.

The kinematics of the model and irreversibility is possible to calculate with mathematic methods. Kinematic is just establishing the function which with correct input like angles of joints q give as an output in form of target position of the endeffector in cartesian coordinates. This is possible to establish with Denavit–Hartenberg convention[2]. On the other hand determining dynamics is not trivial task. The dynamics of the system describe the movement and possible trajectories which are determined trough control and initial conditions of the system. This means that it is fully describable trough the ODE mentioned in section2.1.2

One of the most popular methods for solving and establishing the dynamics of a model is through the use of Lagrange equations.

2.1.5.1 Lagrange Equations

Lagrange Equations are derived from D'Alembert Principle[23] which is an alternative to the Newton's second law[31]:

$$\mathbf{F} = m\mathbf{a} \quad (2.69)$$

In the case of the rigid body:

$$\mathbf{F}_i = \sum_i m_i \mathbf{a}_i \quad (2.70)$$

To derive Lagrangian Equations which describes dynamics we need first to define constraints. Let us define homonomic constraints. For N bodies $B = \{\mathbf{r}_1, \dots, \mathbf{r}_i, \dots, \mathbf{r}_N\}$; $\mathbf{r}_i = \mathbf{r}_i(q_1, \dots, q_i, \dots, q_n, t)$

$$G_k(\mathbf{r}_1, \dots, \mathbf{r}_i, \dots, \mathbf{r}_n, t) = 0 \text{ for } k = 1, \dots, M \quad (2.71)$$

Where the q_i are generalized coordinates and the parameters are also constrained as $3 \cdot M \cdot N = n$.

In case of two dimensional N-Pendulum some of that constraints would look like:

$$G_i := ||\mathbf{r}_i - \mathbf{r}_{i+1}|| - d_i = 0 \quad (2.72)$$

This says that the distance between two neighboring joints are constant. holonomic constraint defines the relation between bodies i on some configuration space \mathcal{A} . The holonomic constraints are intergrable. Lets go back to the second newton law and lets apply holonomic constraint. With holonomic constraint we can spilt \mathbf{F} on constraint force \mathbf{F}^c and applied force \mathbf{F}^a .

$$\mathbf{F} = \mathbf{F}^c + \mathbf{F}^a \quad (2.73)$$

Now we will apply virtual Work principle, which comes from orthogonality of virtual displacement and constraint forces [39]

$$\sum_i \mathbf{F}_i^c \cdot \underbrace{\delta \mathbf{r}_i}_{\text{virtual displacement}} = 0 \quad (2.74)$$

to the constrained Newton's second law we get following formula which is D'Alembert principle[17]:

$$\mathbf{F}_i^c = \mathbf{F}_i - \mathbf{F}_i^a \quad (2.75)$$

$$\sum_i \mathbf{F}_i^c \cdot \delta \mathbf{r}_i = \sum_i (\mathbf{F}_i - \mathbf{F}_i^a) \cdot \delta \mathbf{r}_i \quad (2.76)$$

$$\sum_i (\mathbf{F}_i - m_i \ddot{\mathbf{r}}_i) \cdot \delta \mathbf{r}_i = 0 \text{ D'Alembert Principle} \quad (2.77)$$

If we need virtual displacement for the generalized coordinates:

$$\delta \mathbf{r}_i = \sum_j^n \frac{\partial \mathbf{r}_i}{\partial q_j} \cdot \delta q_j \quad (2.78)$$

Now we can derive Langrange Equations applying last equation to D’Alambert principle:

$$\begin{aligned} \sum_i^N \mathbf{F}_i \cdot \delta \mathbf{r}_i &= \\ \sum_i^N \mathbf{F}_i \sum_j^n \frac{\partial \mathbf{r}_i}{\partial q_j} \cdot \delta q_j &= \\ \sum_j^n \delta q_j \underbrace{\sum_i^N \mathbf{F}_i \frac{\partial \mathbf{r}_i}{\partial q_j}}_{\mathbf{Q}_j} &= \boxed{\sum_j^n \delta q_j \cdot \mathbf{Q}_j} \\ \sum_i^N m_i \ddot{\mathbf{r}}_i \cdot \delta \mathbf{r}_i &= \\ \sum_i^N m_i \ddot{\mathbf{r}}_i \sum_j^n \frac{\partial \mathbf{r}_i}{\partial q_j} \cdot \delta q_j &= \sum_j^n \delta q_j \sum_i^N m_i \ddot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial q_j} \\ \text{with substitution of: } \frac{\partial \dot{\mathbf{r}}_i}{\partial \dot{q}_j} &= \frac{\partial}{\partial \dot{q}_j} \left(\frac{d\mathbf{r}_i}{dt} + \sum_i^N \frac{\partial \mathbf{r}_i}{\partial q_j} \dot{q}_j \right) = \frac{\partial \mathbf{r}_i}{\partial q_j} \\ \sum_i^N m_i \ddot{\mathbf{r}}_i \sum_j^n \frac{\partial \mathbf{r}_i}{\partial q_j} \cdot \delta q_j &= \sum_j^n \delta q_j \sum_i^N m_i \ddot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} \\ \text{with: } \sum_i^N \frac{d}{dt} m_i \dot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} &= \sum_i^N m_i \ddot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} + m_i \dot{\mathbf{r}}_i \frac{d}{dt} \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} \\ \sum_j^n \delta q_j \sum_i^N m_i \ddot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} &= \sum_j^n \delta q_j \sum_i^N \frac{d}{dt} \left(m_i \dot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial \dot{q}_j} \right) - m_i \dot{\mathbf{r}}_i \frac{\partial \mathbf{r}_i}{\partial q_j} \end{aligned}$$

We get this equation

$$\sum_i^N m_i \ddot{\mathbf{r}}_i \cdot \delta \mathbf{r}_i = \sum_j^n \delta q_j \left[\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \left(\sum_i^N \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 \right) - \frac{\partial}{\partial \dot{q}_j} \left(\sum_i^N \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 \right) \right] \quad (2.79)$$

With those two made equations we can finally put it together in D'Alembert Principle formula:

$$\sum_j^n \delta q_j \left[\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \left(\sum_i^N \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 \right) - \frac{\partial}{\partial \dot{q}_j} \left(\sum_i^N \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 \right) - \mathbf{Q}_j \right] = 0 \quad (2.80)$$

This Equation is almost complete, we need to define the energies and their specifications

- Kinetic Energy - Energy possessed by bodies due to their motion.
The Formula for kinetic energy is

$$T(q, \dot{q}) = \sum_i^N \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 \quad (2.81)$$

- Potential Energy - Energy held by body because of its relative position to other bodies or other target.
There are many types of potential energy:
Pendulum - $U(q = h) = mgh$
Oscillator - $U(q) = \frac{kq}{2}$

In the Formula \mathbf{Q} is generalized force and it is created from the gradient of the potential energy and it is conservative :

$$\mathbf{Q}_j = - \sum_i^N \nabla_{\mathbf{r}_i} U(q) \frac{\partial \mathbf{r}_i}{\partial q_j} = - \frac{\partial U}{\partial q_j} \quad (2.82)$$

Potential Energy is only dependent on position of the bodies $\frac{\partial U}{\partial \dot{q}} = 0$.

The Lagrange Equations for holonomic constraints and conservative force is defined as:

$$\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} (T - U) - \frac{\partial}{\partial q_j} (T - U) = 0 \quad (2.83)$$

$$\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \mathcal{L} - \frac{\partial}{\partial q_j} \mathcal{L} = 0 \quad (2.84)$$

\mathcal{L} is called Lagrangian. In robotics this equation is not so interesting without control. For controlling the langrangian system we set a formula as

$$\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \mathcal{L} - \frac{\partial}{\partial q_j} \mathcal{L} = \tau \quad (2.85)$$

Where τ is torque/momentum or controlling input.

2.1.5.2 Hamiltonian Equations

To derive Hamiltonian Equations we need a Lagrangian \mathcal{L} , because we need:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \dot{\mathbf{p}} \quad (2.86)$$

$$\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} = \mathbf{p} \quad (2.87)$$

Through Legendre Transformation we can obtain Hamiltonian \mathcal{H}

$$\mathcal{H}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{p}, t) = \sum_i^N \dot{\mathbf{q}}_i \cdot \mathbf{p}_i - \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) \quad (2.88)$$

With total derivation[34] we get

$$d\mathcal{H} = \sum_i^N \left[\dot{\mathbf{q}} d\mathbf{p}_i + \mathbf{p}_i d\dot{\mathbf{q}}_i - \frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} d\mathbf{q}_i - \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}_i} d\dot{\mathbf{q}}_i \right] - \frac{\partial \mathcal{L}}{\partial t} dt \quad (2.89)$$

$$= \sum_i^N [\dot{\mathbf{q}} d\mathbf{p}_i + \mathbf{p}_i d\dot{\mathbf{q}}_i - \dot{\mathbf{p}}_i d\mathbf{q}_i - \mathbf{p}_i d\dot{\mathbf{q}}_i] - \frac{\partial \mathcal{L}}{\partial t} dt \quad (2.90)$$

$$= \sum_i^N [\dot{\mathbf{q}} d\mathbf{p}_i - \dot{\mathbf{p}}_i d\mathbf{q}_i] - \frac{\partial \mathcal{L}}{\partial t} dt \quad (2.91)$$

$$(2.92)$$

On the other hand we want hamiltonian which is dependent on canoical moment and generalized coordinates $\mathcal{H} = \mathcal{H}(\mathbf{q}, \mathbf{p}, t)$ In that case the total derivation of Hamiltonian \mathcal{H} is

$$d\mathcal{H} = \sum_i^N \left[\frac{\partial \mathcal{H}}{\partial \mathbf{q}_i} d\mathbf{q}_i + \frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} d\mathbf{p}_i \right] + \frac{\partial \mathcal{H}}{\partial t} dt \quad (2.93)$$

With comparing both equivalent derivations we can define the equations of motion:

$$\dot{\mathbf{p}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \quad (2.94)$$

$$\dot{\mathbf{q}} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} \quad (2.95)$$

$$\frac{\partial \mathcal{H}}{\partial t} = -\frac{\partial \mathcal{L}}{\partial t} \quad (2.96)$$

Hamiltonian equations are very useful because in the case $\frac{\partial \mathcal{H}}{\partial t} = 0$ our Total Energy $\mathcal{H} = E = T + U$ is conserved and often we can derive time independent motion. This can be simplified as

$$\dot{\mathbf{x}} = \mathbf{J} \frac{\partial \mathcal{H}}{\partial \mathbf{x}}(\mathbf{x}) \quad (2.97)$$

where $\mathcal{H} = T + U$ and $\mathbf{J} = \begin{bmatrix} 0 & \mathbf{I}_n \\ -\mathbf{I}_n & 0 \end{bmatrix}$

In applied robotics we use Port-Hamiltonian Equation[44]

$$[\mathbf{J} - \mathbf{R}] \frac{\partial \mathcal{H}}{\partial \mathbf{x}}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u. \quad (2.98)$$

If we set \mathbf{R} and u as zero we get a trivial hamitonian equation. $\mathbf{R} = \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{r} \end{bmatrix}$ is in this case dissipation matrix or coriolis matrix and $\mathbf{g} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}$ is control vector.

2.2 Neural Network Architectures

At the beginning of our research we experimented with feedforward networks and recurrent networks.

2.2.1 Deep forward Network - Multilayer Perceptron

Feedforward networks like Multilayer perceptron approximates a certain function $\mathbf{f}(\mathbf{x})$ which is often interpreted as a mapping $\mathbf{f}(\mathbf{x}|\Theta)$.

The main goal of feedforward networks is to fit the model so that we get the best approximation. Fitting the model means measuring how the model adapts the data similar to data on which it is trained on.

Multilayer perceptron is a fully connected feedforward neural network. The nodes of the MLP are often called neurons and they build a layer. A minimal MLP has three consecutive layers, input, hidden and output layer, which form a complete bipartite graph.

Mathematical expression of one neuron:

$$y = \sigma\left(\sum_i w_i \cdot x_i + b\right) \quad (2.99)$$

Mathematical expression of the feedforward layer network

$$\mathbf{h}^{(k+1)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}\right) \quad (2.100)$$

and we can see graphical representation in Figure 2.1. In this equation σ is called activation function, \mathbf{W} as weights and \mathbf{b} as bias which are our parameters $\Theta = \{\mathbf{W}, \mathbf{b}\}$.

The activation function in the network introduces non-linearity to improve approximation

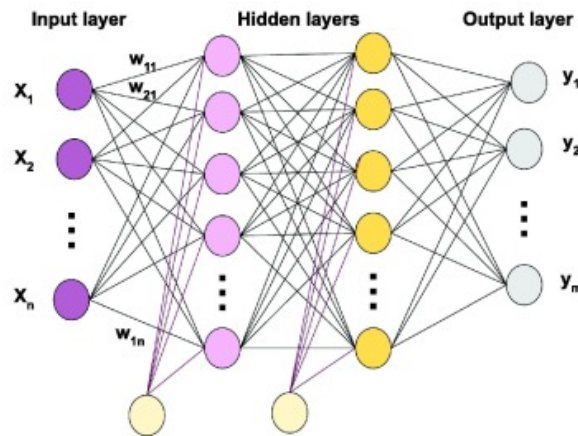


Figure 2.1: Multilayer perceptron[6]

capability.

2.2.2 Activation functions and Loss functions

In every neural architecture, especially MLP, there is a activation function σ . They are differentiable functions which are used to introduce nonlinearity into the model. They must be differentiable because of the backpropagation. Most used activation functions are:

- ReLU

$$\text{ReLU}(x) = \max(x, 0) \quad (2.101)$$

This function passes only positive values.

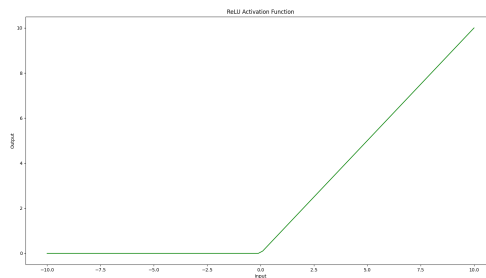


Figure 2.2: ReLU activation

- tanH - It gives an output $y \in [-1, 1]$

Output layer has no activation function. it is connected to the loss function. They are used for optimization comparing model output with the target. The most used loss functions are:

- Mean Absolute Error(MAE)

$$\text{MAE}(y, \hat{y}) = \frac{1}{N} \sum_i^N |y_i - \hat{y}_i| \quad (2.102)$$

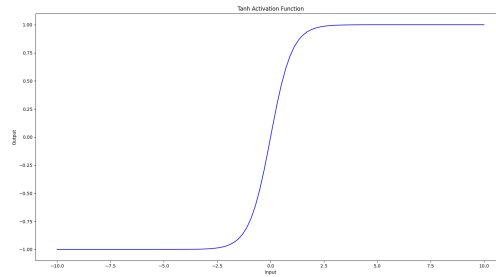


Figure 2.3: Tanh activation

- Mean Squared Error(MSE)

$$\text{MSE}(y, \hat{y}) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2 \quad (2.103)$$

- HuberLoss

$$\text{HuberLoss}(y, \hat{y}; \delta) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |(y - \hat{y})| < \delta \\ \delta((y - \hat{y}) - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \quad (2.104)$$

2.2.3 Recurrent Neural Networks

Recurrent Neural Networks are very interesting concept of Neural Network architecture. It is mostly used for speech recognition and for the time series data[30].

As the name says it feed forward network which propagate recurrently. It means that propagation is dependent on repeating the model to fit the size of the data, like data for time-series which can have various time length. There is three types which we can use[12]:

- Many to many

This type is used for speech recognition we use a dataset which has sentences and it outputs some values which could be understood from other model. For example translation or video captioning

- One to many
Used for the time series data from one input we can create recurrence that represent a time point with new output. From one input we get more outputs. time-series can be for example music generation.
- Many to one
From many outputs we get only one output. This is very good model for spam detection

Let say that we have a sequence $S = \{\mathbf{x}^0, \mathbf{x}^1, \dots, \mathbf{x}, \dots, \mathbf{x}^n\}$. the first element \mathbf{h}^0 will be initialized as 0. The architecture of the RNN-cell[41] described in mathematical formulation

$$\begin{aligned}\mathbf{h}^{(i)} &= \sigma(\mathbf{W}_{hh} \cdot \mathbf{h}^{(i-1)} + \mathbf{W}_{hx} \mathbf{x}^{(i-1)} + \mathbf{b}_h) \\ \mathbf{y}^{(i)} &= \sigma(\mathbf{W}_{yh} \cdot \mathbf{h}^{(i)} + \mathbf{b}_y)\end{aligned}$$

and graphical appearance is shown in 2.4

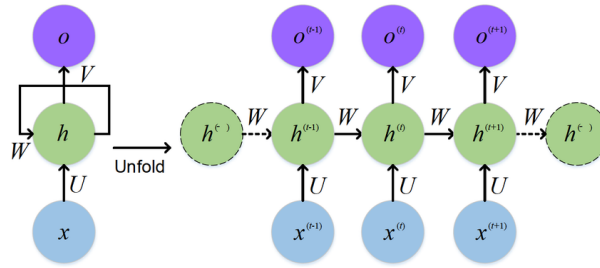


Figure 2.4: Recurrent neural unit(many to many)[16]

There is similar and better model then RNN and that is Gated Recurrent Unit called GRU. We use it in same way as RNN but it his own unique architecture[14]:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \quad (2.105)$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \quad (2.106)$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \quad (2.107)$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \quad (2.108)$$

and we can see it in figure 2.5 The GRU unit is specific because it has reset gate r_i and update gate u_i . Such architecture give better performance then at simple RNN because it has memory capabilities.

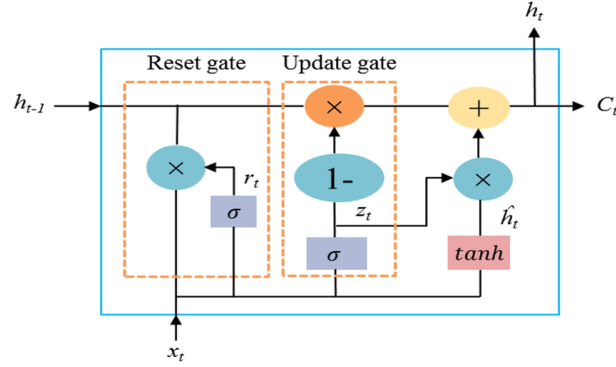


Figure 2.5: Gated recurrent unit[33]

2.2.4 Optimizers

To do backpropagation we need an optimizer to update our learnable parameters. The easiest optimizer is stochastic gradient descent.

$$\mathbf{W}^{(i+1)} = \mathbf{W}^{(i)} - \mu \frac{\partial \text{Loss}(z)}{\partial \mathbf{W}^{(i)}} \quad (2.109)$$

There are more better one and for our use case we will use AdamW[28]. We use this optimizer because it shows good performance in both regression and classification models.

2.2.5 Backpropagation of Neural Models

After the forward pass and calculated loss, we need to do just backward pass and optimization step.

In backward pass we are computing the gradients of our learnable parameters to use it for the optimization of the model.

For example we will use stochastic gradient descent(SGD)

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \gamma \cdot \frac{\partial \text{Loss}(z)}{\partial \mathbf{W}_i} \quad (2.110)$$

$$\mathbf{b}_{i+1} = \mathbf{b}_i - \gamma \cdot \frac{\partial \text{Loss}(z)}{\partial \mathbf{b}_i} \quad (2.111)$$

where γ is a learning rate and one layered MLP without bias \mathbf{b} .

$$\mathbf{z}_1 = \mathbf{W}_0 \mathbf{y}_0 \quad (2.112)$$

$$\mathbf{y}_1 = \sigma(\mathbf{z}_1) \quad (2.113)$$

$$\text{Loss}(y_1, \hat{y}) = \text{MSE}(y_1, \hat{y}) \quad (2.114)$$

The obtaining the gradients of the weights is called backpropagation because for its calculation we need to apply chain rule for derivation.

$$\frac{\partial \text{Loss}(z)}{\partial \mathbf{W}_0} = \frac{\partial \text{Loss}(z)}{\partial \mathbf{y}_1} \cdot \frac{\partial \mathbf{y}_1}{\partial \mathbf{z}_1} \cdot \frac{\partial \mathbf{z}_1}{\partial \mathbf{W}_0} \quad (2.115)$$

Recurrent neural networks often fall as a victim of gradient vanishing or gradient exploding. In this manner the models doesn't learn because the derivatives calculated through backpropagation are too small or too big. It is important that gradients slowly decrease through learning that whole model can fit the data.

3 Datasets

Now we consider the study case of the datasets:

- N body problem
- Two body problem
- Threebody problem
- N pendulum

First we will mathematically describe the system and later we will explain how to prepare and create trajectories corresponding to their starting point/initial value.

3.1 Oscilator

As we already shown the oscillator solution, we could show solution of the oscillator trough hamiltonian equations. For this we will use Kinetic Energy $T = \frac{1}{2}mv^2$ and Potential energy $U = \frac{1}{2}kx^2$. Using the momentum $p = mv$ we can rewrite our Kinetic energy as $T = \frac{1}{2m}p^2$.

Using Hamiltonian Energy definition we obtain following equation:

$$\mathcal{H} = \frac{1}{2m}p^2 + \frac{1}{2}kx^2 \quad (3.1)$$

This case is trivial. We could get \dot{x} and \dot{p} trough hamiltonian equations $\dot{q} = \frac{\partial \mathcal{H}}{\partial p}$ and $\dot{p} = -\frac{\partial \mathcal{H}}{\partial q}$, or we could compare Hamiltonian with ellipse formula

$$1 = \frac{x^2}{a^2} + \frac{y^2}{b^2}, \quad (3.2)$$

and set the components in parametric equation

$$(x, y) = (a \cos(\phi), b \sin(\phi)) \rightarrow (p, q) = \left(\sqrt{2Hm} \cos(\phi), \sqrt{\frac{2H}{k}} \sin(\phi) \right) \quad (3.3)$$

of the ellipse.

This equation represents kinematics of the oscillator. To find periodicity let say that $\phi = \omega t + \Phi$ and we know that the $p = mx$, with this equation we can calculate ω .

$$p = m\dot{x} = \sqrt{2Hm} \cos(\omega t + \Phi), \quad (3.4)$$

$$\dot{x} = \sqrt{\frac{2H}{m}} \cos(\omega t + \Phi), \quad (3.5)$$

$$x = \int \sqrt{\frac{2H}{m}} \cos(\omega t + \Phi) dt = \frac{1}{\omega} \sqrt{\frac{2H}{m}} \sin(\omega t + \Phi) = \sqrt{\frac{2H}{k}} \sin(\omega t + \Phi), \quad (3.6)$$

$$\omega = \sqrt{\frac{k}{m}} \quad (3.7)$$

The oscillator has harmonic, periodic movement which means that after some time T the movement will be repeated $x(t_0) = x(t_N)$. With that property we can calculate the period over circle circumference:

$$T = \frac{2\pi}{\omega} \quad (3.8)$$

With this we can create a dataset within the program code. For coding we used framework `pytorch` to calculate this dataset numerically.

For the obtain the solution of ODE we used package `torchdiffeq`[8] to solve numerical equation with `dopri5` method, which is most accurate method for solving ODEs.

To create unique trajectories for the dataset we just need to specify energy region.

3.2 N-body problem

N-body problem is most complex problem in physics. It describes a movement of bodies in the free space. There are no conditions and there are multitude of solutions. This problem is even a topic in quantum theory.

We will just observe and compute real or deterministic trajectories of this problem. In Newtonian formulation the N-body problem is

$$m_i \ddot{\mathbf{x}}_i = - \sum_{i \neq j, j=1}^n G \frac{m_i m_j (\mathbf{x}_i - \mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|^3} \quad (3.9)$$

and its Hamiltonian:

$$\mathcal{H} = \sum_i^n \frac{1}{2m_i} \|\mathbf{p}_i\|^2 - \sum_{1 < i, j < n, i \neq j} \frac{G m_i m_j}{\|\mathbf{x}_i - \mathbf{x}_j\|} \quad (3.10)$$

for $i \in [1, n]$. N body problem is very complex problem because we need to achieve a conservative property of the Hamiltonian in the system.

The biggest issue in this dataset are the crashes between the bodies. This can be resolved with simple trick adding the ϵ to fix maximal potential energy

$$- \sum_{1 < i, j < n, i \neq j} \frac{G m_i m_j}{\|\mathbf{x}_i - \mathbf{x}_j\| + \epsilon}. \quad (3.11)$$

We secured that the value in the dominator never reaches 0. This is important because we use numerical solvers to create trajectories. If we have energy which achieves very big difference in values between two time steps, our solution will be inaccurate. For calculation of such trajectories we are forced not to calculate over the Hamiltonian equations, but to calculate the trajectories using acceleration, velocity and position of the bodies through some symplectic numerical methods. The code base for computation of the dataset we borrowed from [32]. In the code we introduced changes like solver, we used leapfrog and beeman method for calculating the trajectories.

To create the dataset we carefully chose those constalations which are in fixed domain and the hamiltonian looks conservative. This conservative property is observable through mean and standard deviation of the Hamiltonian Energy

3.2.1 Symplectic numerical methods to solve N-body problem

Symplectic numerical methods are designed for numerical solution of Hamilton Equations. It possesses conserved quality to approximate the Total Energy the Hamiltonian.

They have two forms. One in canonical coordinates (q, p) and in langrangian coordinates (x, v, a) . Most used ones are in langrangian form and we will use it to create trajectory for N-body

problem.

From the Newtonian formula for N-body problem we can get acceleration function which will be used in numerical solvers. In first step we used velocity verlet

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_i + \mathbf{a}_i \frac{h^2}{2} \quad (3.12)$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + (\mathbf{a}_i + \mathbf{a}_{i+1}) \frac{h}{2} \quad (3.13)$$

and for the next steps with timestep h we used more stable beeman method

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{6}(4\mathbf{a}_i - \mathbf{a}_{i-1}) \quad (3.14)$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{h}{6}(2\mathbf{a}_{i+1} + 5\mathbf{a}_i - \mathbf{a}_{i-1}). \quad (3.15)$$

This needed to be done because the beeman method needs acceleration of the next and previous step to be calculated. We used langragian coordinates because it is most easiest way to create trajectories. In canonical coordinates we need more formulas like for hamiltonian energy and its derivation. In this way we can very fast calculate the trajectories of nbody problem. The dataset is fully adjustable. We can adjust gravity constant and masses how we like. As proof of concept we used $G = 1$ and $m_i = 1$.

3.2.2 Twobody problem

Twobody problem is one of the most easiest dataset which could be created from N-body problem. It has one trivial periodic solution which is binary star movement. Two stars trough their movement maintain constant distance r and their center of mass \mathbf{p}_m is static at the origin.

The newtonian form of twobody Problem is written as

$$m_1 \ddot{\mathbf{q}}_1 = -G \frac{m_1 m_2 (\mathbf{q}_1 - \mathbf{q}_2)}{\|\mathbf{q}_1 - \mathbf{q}_2\|^3} \quad (3.16)$$

$$m_2 \ddot{\mathbf{q}}_2 = -G \frac{m_1 m_2 (\mathbf{q}_2 - \mathbf{q}_1)}{\|\mathbf{q}_2 - \mathbf{q}_1\|^3} \quad (3.17)$$

From those equations we can get a statement about third newton law:

$$m_1 \ddot{\mathbf{q}}_1 - m_2 \ddot{\mathbf{q}}_2 = 0 = \mathbf{F}_{ij} - \mathbf{F}_{ji} \quad (3.18)$$

and its hamiltonian

$$\mathcal{H} = \frac{\|\mathbf{p}_1\|}{2m_1} + \frac{\|\mathbf{p}_2\|}{2m_2} - G \frac{m_1 m_2}{\|\mathbf{q}_1 - \mathbf{q}_2\|}. \quad (3.19)$$

We said that we have static center mass point. center mass point is obtained using following formula

$$\mathbf{q}_m = \frac{1}{m_1 + m_2} (m_1 \mathbf{q}_1 + m_2 \mathbf{q}_2). \quad (3.20)$$

If we set the center mass point in origin the equation simplifies and we get one about velocity

$$m_1 \mathbf{q}_1 + m_2 \mathbf{q}_2 = 0, \quad (3.21)$$

$$m_1 \dot{\mathbf{q}}_1 + m_2 \dot{\mathbf{q}}_2 = \mathbf{p}_1 + \mathbf{p}_2 = 0. \quad (3.22)$$

From now on we will write $\mathbf{p} = \mathbf{p}_1 = -\mathbf{p}_2$. Lets make some formulas about distance

$$\mathbf{q}_2 = \frac{m_1}{m_2} \mathbf{q}_1, \quad (3.23)$$

$$\|\mathbf{q}_2\| = \mathbf{q}_m=0 \frac{m_1}{m_2} \|\mathbf{q}_1\|, \quad (3.24)$$

$$r_2 = \frac{m_1}{m_2} r_1. \quad (3.25)$$

With this we got interesting property

$$\frac{r_1}{r_2} = \frac{m_2}{m_1}. \quad (3.26)$$

This can be used to calculate constant distance between the bodies

$$r = r_1 \left(1 + \frac{m_1}{m_2} \right). \quad (3.27)$$

With those equations we can simplify hamiltonian

$$\mathcal{H} = \|\mathbf{p}\|^2 \left(\frac{1}{m_1} + \frac{1}{m_2} \right) - G \frac{m_1 m_2}{r}. \quad (3.28)$$

Still to make a dataset with twobody trajectories we need a period T . Let assume that the bodies moves on a circular path

$$\mathbf{q}_1 = r_1 \begin{bmatrix} \cos(\Phi) \\ \sin(\Phi) \end{bmatrix}, \quad (3.29)$$

$$\mathbf{q}_2 = r_2 \begin{bmatrix} \cos(\Phi + \pi) \\ \sin(\Phi + \pi) \end{bmatrix}. \quad (3.30)$$

The momentum is tangential in dependency of orientation of position

$$\mathbf{p}_1 = \|\mathbf{p}\| \begin{bmatrix} \sin(\Phi) \\ -\cos(\Phi) \end{bmatrix} \quad (3.31)$$

$$\mathbf{p}_2 = \|\mathbf{p}\| \begin{bmatrix} \sin(\Phi + \pi) \\ -\cos(\Phi + \pi) \end{bmatrix} \quad (3.32)$$

with $\Phi \in [0, 2\pi]$. Let say that $\Phi = \omega t$ and we want to find a period of movement. With the equations

$$\dot{\mathbf{q}}_1 = r_1 \omega \begin{bmatrix} \sin(\omega t) \\ -\cos(\omega t) \end{bmatrix} \quad (3.33)$$

$$= \frac{\mathbf{p}_1}{m_1} = \frac{\|\mathbf{p}\|}{m_1} \begin{bmatrix} \sin(\omega t) \\ -\cos(\omega t) \end{bmatrix} \quad (3.34)$$

$$\|\mathbf{p}\| = r_1 m_1 \omega, \quad (3.35)$$

we calculated the value of the momentum. This momentum we can get from physics and newton third law: if some body moves around some point in radial path, we can calculate a centripetal Force and this Force should be equal to Gravitational Force

$$F_c = \frac{\|\mathbf{p}\|^2}{m_1 r_1} = \frac{G m_1 m_2}{r^2} = F_g. \quad (3.36)$$

Substituting $\|\mathbf{p}\|$ we get ω

$$\omega = \frac{1}{r} \sqrt{G \frac{m_2}{r_1}} \quad (3.37)$$

and T

$$T = \frac{2\pi}{\omega}. \quad (3.38)$$

In twobody problem to create some diverse dataset we vary the distance between the bodies keeping the masses intact. We need to be careful because with grater distance r the period T will be longer.

The mathematical formulation of the solution which is used for code is taken from hamiltonian equations

$$\dot{\mathbf{x}} = J \frac{\partial H}{\partial \mathbf{x}}(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{m_1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{m_1} \\ 0 & 0 & 0 & 0 & -\frac{1}{m_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{m_2} \\ -\mu_{1,2} & 0 & \mu_{1,2} & 0 & 0 & 0 \\ 0 & -\mu_{1,2} & 0 & \mu_{1,2} & 0 & 0 \end{bmatrix} \begin{bmatrix} q_{1x} \\ q_{1y} \\ q_{2x} \\ q_{2y} \\ p_x \\ p_y \end{bmatrix} \quad (3.39)$$

where $\mu_{ij} = \frac{Gm_i m_j}{r^3}$. We used Dopri5 to create the trajectories. To diversify our dataset we suggest to define the distance region and pick randomly the distance between the bodies. The masses should be the same.

3.2.3 Three body problem

Three body problem is from formulation similar to two body problem, but finding the periodic solutions is very complex task. Fortunately the M. Šuvlakov and V. Dmitrašinović have found many of the initial values to create periodic three body trajectory in 2D space and made data public over the website for everyone to use.[43][38]. It is important to mention that they made it to mathematical problem which means that the parameters like the gravitational constant G and all masses m are set to 1. Even today there are scientists which are documenting initial values for more periodic solutions [20]. Having those conditions we need just Hamiltonian equations which are easy to obtain.

Obtaining the formula for straight-forward calculation the trajectories of the threebody problem we need to define a function

$$\Xi(\mathbf{q}_i, \mathbf{q}_j) = \Xi_{i,j} = \frac{Gm_i m_j}{\|\mathbf{q}_i - \mathbf{q}_j\|^3}. \quad (3.40)$$

In following consider $\mathbf{m}_i^{-1} = \text{diag}(\frac{1}{m_i}, \frac{1}{m_i})$ and coordinates are two dimensional.

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{m}_1^{-1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{m}_2^{-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{m}_3^{-1} \\ -(\Xi_{1,2} + \Xi_{1,3}) & \Xi_{1,2} & \Xi_{1,3} & 0 & 0 & 0 \\ \Xi_{1,2} & -(\Xi_{1,2} + \Xi_{2,3}) & \Xi_{2,3} & 0 & 0 & 0 \\ \Xi_{1,3} & \Xi_{2,3} & -(\Xi_{1,3} + \Xi_{2,3}) & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad (3.41)$$

To diversify our dataset we found that the three-body problem has two interesting properties due its defined potential energy. Their potential energy is defined over gravitational force between the bodies. In another words we can translate the trajectories together in space and rotate it without changing the hamiltonian value. Let us prove this over Hamiltonian Energy

$$\mathcal{H} = \frac{\|\mathbf{p}_1\|^2}{2m_1} + \frac{\|\mathbf{p}_2\|^2}{2m_2} + \frac{\|\mathbf{p}_3\|^2}{2m_3} - G \frac{m_1 m_2}{\|\mathbf{q}_1 - \mathbf{q}_2\|} - G \frac{m_2 m_3}{\|\mathbf{q}_2 - \mathbf{q}_3\|} - G \frac{m_1 m_3}{\|\mathbf{q}_1 - \mathbf{q}_3\|}. \quad (3.42)$$

- Translation:

Lets define $\mathbf{q}_i = \lim_{\mathbf{q}_t \rightarrow 0} (\mathbf{q}_i + \mathbf{q}_t)$ where \mathbf{q}_t is translation vector. When we substitute \mathbf{q}_i . The hamiltoinian acts as follows

$$\begin{aligned}
\mathcal{H} &= \frac{\|\mathbf{p}_1\|^2}{2m_1} + \frac{\|\mathbf{p}_2\|^2}{2m_2} + \frac{\|\mathbf{p}_3\|^2}{2m_3} \\
&\quad - \lim_{\mathbf{q}_t \rightarrow 0} G \frac{m_1 m_2}{\|\mathbf{q}_1 + \mathbf{q}_t - (\mathbf{q}_2 + \mathbf{q}_t)\|} - \lim_{\mathbf{q}_t \rightarrow 0} G \frac{m_2 m_3}{\|\mathbf{q}_2 + \mathbf{q}_t - (\mathbf{q}_3 + \mathbf{q}_t)\|} \\
&\quad - \lim_{\mathbf{q}_t \rightarrow 0} G \frac{m_1 m_3}{\|\mathbf{q}_1 + \mathbf{q}_t - (\mathbf{q}_3 + \mathbf{q}_t)\|} \\
&= \frac{\|\mathbf{p}_1\|}{2m_1} + \frac{\|\mathbf{p}_2\|}{2m_2} + \frac{\|\mathbf{p}_3\|}{2m_3} \\
&\quad - G \frac{m_1 m_2}{\|\mathbf{q}_1 - \mathbf{q}_2 + (\mathbf{q}_t - \mathbf{q}_t)\|} - G \frac{m_2 m_3}{\|\mathbf{q}_2 - \mathbf{q}_3 + (\mathbf{q}_t - \mathbf{q}_t)\|} \\
&\quad - G \frac{m_1 m_3}{\|\mathbf{q}_1 - \mathbf{q}_3 + (\mathbf{q}_t - \mathbf{q}_t)\|} \\
&= \frac{\|\mathbf{p}_1\|}{2m_1} + \frac{\|\mathbf{p}_2\|}{2m_2} + \frac{\|\mathbf{p}_3\|}{2m_3} - G \frac{m_1 m_2}{\|\mathbf{q}_1 - \mathbf{q}_2\|} - G \frac{m_2 m_3}{\|\mathbf{q}_2 - \mathbf{q}_3\|} - G \frac{m_1 m_3}{\|\mathbf{q}_1 - \mathbf{q}_3\|}
\end{aligned}$$

- Rotation: Lets define $\mathbf{q}_i = r_i \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix}$ and $\mathbf{p}_i = p_i \begin{bmatrix} -\sin(\beta + \alpha) \\ \cos(\beta + \alpha) \end{bmatrix}$ where $\beta \in (0, 2\pi]$ but it is defined and $\alpha \in (0, 2\pi]$ which is free parameter. Let substitute this in Hamiltonian

$$\begin{aligned}
\mathcal{H} &= \frac{\left\| p_1 \begin{bmatrix} -\sin(\beta + \alpha) \\ \cos(\beta + \alpha) \end{bmatrix} \right\|^2}{2m_1} + \frac{\left\| p_2 \begin{bmatrix} -\sin(\beta + \alpha) \\ \cos(\beta + \alpha) \end{bmatrix} \right\|^2}{2m_2} + \frac{\left\| p_3 \begin{bmatrix} -\sin(\beta + \alpha) \\ \cos(\beta + \alpha) \end{bmatrix} \right\|^2}{2m_3} \\
&\quad - G \frac{m_1 m_2}{\left\| r_1 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} - r_2 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} \right\|} \\
&\quad - G \frac{m_2 m_3}{\left\| r_2 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} - r_3 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} \right\|} \\
&\quad - G \frac{m_1 m_3}{\left\| r_1 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} - r_3 \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} \right\|}
\end{aligned}$$

From trigonometry we know that $\cos(\phi)^2 + \sin(\phi)^2 = 1$.

From that we get $\left\| \begin{bmatrix} \cos(\beta + \alpha) \\ \sin(\beta + \alpha) \end{bmatrix} \right\| = \left\| \begin{bmatrix} \cos(\beta) \\ \sin(\beta) \end{bmatrix} \right\| = 1.$

Applying this theorem we get

$$\mathcal{H} = \frac{p_1^2}{2m_1} + \frac{p_2^2}{2m_2} + \frac{p_3^2}{2m_3} - G \frac{m_1 m_2}{|r_1 - r_2|} - G \frac{m_2 m_3}{|r_2 - r_3|} - G \frac{m_1 m_3}{|r_1 - r_3|} \quad (3.43)$$

which is equivalent to the previous definition.

For the dataset creation we wanted that our trajectory mass middle point is fixed. We needed only rotation around it. For such rotation we used formula for such transformation

$$\hat{\mathbf{q}} = \begin{bmatrix} 1 & 0 & 0 & x_m \\ 0 & 1 & 0 & y_m \\ 0 & 0 & 1 & z_m \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_m \\ 0 & 1 & 0 & -y_m \\ 0 & 0 & 1 & -z_m \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix}, \quad (3.44)$$

which for two dimensional cases can be rewritten as

$$\hat{\mathbf{q}} = \mathbf{R}\mathbf{q} - \mathbf{R}\mathbf{q}_m + \mathbf{q}_m, \quad (3.45)$$

where $\mathbf{R} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}.$

3.3 N Pendulum

The N Pendulum is straight forward model. To make it in canocnical coordinates which are angles of the joints, still we need to define Kinematic of the model.

$$x_i = \sum_i^n l_i \sin(\Theta_i) \quad (3.46)$$

$$y_i = - \sum_i^n l_i \cos(\Theta_i) \quad (3.47)$$

Their temporal derivations are:

$$\dot{x}_i = \sum_i^n \dot{\Theta}_i l_i \cos(\Theta_i) \quad (3.48)$$

$$\dot{y}_i = \sum_i^n \dot{\Theta}_i l_i \sin(\Theta_i) \quad (3.49)$$

From this kinematic model we just calculated the Hamiltonian and proceed with autodifferentiaion from pytorch framework to calculate the values of hamiltonian equations

$$\mathcal{H} = \frac{1}{2} \sum_i^n m_i (\dot{x}_i^2 + \dot{y}_i^2) + \sum_i^n m_i g_i y_i. \quad (3.50)$$

We set a canonical coordinates (Θ, p_Θ) and we can calculate $p_\Theta = \frac{\partial \mathcal{H}}{\partial \dot{\Theta}}$. Making the Hamiltonan only dependent on Θ and p_Θ we can get hamiltonian equations:

$$\dot{\Theta}_i = \frac{\partial \mathcal{H}}{\partial p_{\Theta_i}} \quad (3.51)$$

$$p\dot{\Theta}_i = -\frac{\partial \mathcal{H}}{\partial \Theta} \quad (3.52)$$

This was done using auto-differentiation and we managed to create dataset models for (1,2,3,4) degrees of freedom, which offers very accurate hamiltonian values in the dataset. The dataset need fixed inital values like $\Theta \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ and starting $p_{\Theta_i} = 0$.

We can create a dataset with observation. With interest to creating of 3D models as 'urdf files, we created the framework which trough python coding crates accurate urdf model, in our case N Pendelum. This was used to create pendulum with N number of joints. This model was created for and observed in `pybullet`[10]. From the use case we suggest if you are working with pendelums where $N < 10$ please use the pytorch numerical technique, otherwise use `pybullet` observation. `Pybullet` can extract, almost in real-time, the movement and velocities of the pendelum. Only we need to calculate if Hamiltonian energy is stable enough for our experiments.

In the case that we didn't got any velocity for our dataset it is possible to obtain it over trajectory trough "euler trick"

$$\mathbf{v}_i = \mathbf{f}(\mathbf{x}_i) = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{h}. \quad (3.53)$$

In Figure 3.1 you can see the Pendelum in `Pybullet` simulation software

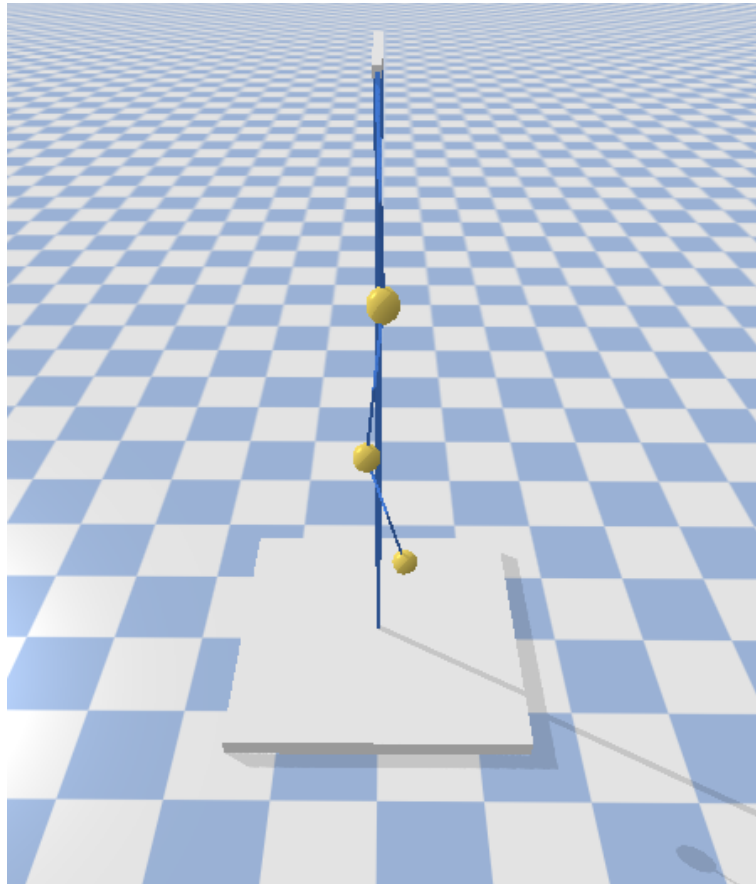


Figure 3.1: Pendulum with 3 degrees of freedom in pybullet

4 Methods

After previous chapters we should achieve basic understanding of neural network models and physics. Now we study state of the art methods which should capture hamiltonian model on studied datasets.

4.1 New paradigm - NeuralODE

Neural ODE is very interesting and important concept for solving the Ordinary Differential Equations and training the neural networks with trajectory data.

The default situation is described as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t | \Theta) \quad (4.1)$$

Where $\mathbf{f}(\mathbf{x}, t | \Theta)$ represents the multilayer perception which creates vector field. This method has its history from residual networks where their feed-forward network looks like euler step

$$\mathbf{h}^{(i+1)} = \mathbf{h}^{(i)} + \mathbf{f}(\mathbf{h}^{(i)} | \Theta) \quad (4.2)$$

In the paper [9] it is introduced a adjoint method which solves problem with vanishing gradients.

This is achieved trough continuous backpropagation which is stated in the paper. We can compare it

	residual network	adjoint method
a_t or $a(t)$:	$\frac{\partial L}{\partial z}$	$\frac{\partial L}{\partial z(t)}$
forward-pass:	$z_{i+1} = z_t + hf(z_t)$	$z(t+1) = z(t) + \int_t^{t+1} f(t)dt$
backward-pass:	$a_t = a_{t+1} + ha_{t+1} \frac{\partial f(z_t)}{\partial z_t}$	$a(t) = a(t+1) + \int_{t+1}^t a(t) \frac{\partial f(z(t))}{\partial z(z)} dt$
gradients:	$\frac{\partial L}{\partial \theta} = ha_{t+h} \frac{\partial f(z(t), \Theta)}{\partial \Theta}$	$\frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t) \frac{f(z(t), \Theta)}{\partial \Theta}$

For PINN models the adjoint method over framework `torchdiffEq` doesn't work properly because of the computed gradients which PINNs produce. In case of PINN models we suggest to use normal rollout with ode solver like RK4.

4.2 Recurrent time steppers

In chapter 2 we introduced recurrent models which are RNN and GRU. Those models are very good at learning the time series data. To make it more physics informed we applied an euler method which it makes to time stepper models:

$$[\mathbf{v}_i, \mathbf{h}_{i+1}] = \text{MLP}(\text{RNN}(\mathbf{x}_i, \mathbf{h}_i) \text{ or } \text{GRU}(\mathbf{x}_i, \mathbf{h}_i)) \quad (4.3)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i \cdot dt \quad (4.4)$$

We describe it this procedure as a rollout. This model has one big static parameter dependency which is fixed dt . We suggest to not to change this parameter after training. In Chapter Experimentation we will show the performance of RNN and GRU time stepper models. In the figure 4.1 we can see the architecture of the model.

4.3 Hamiltonian Neural Network

This model comes from the paper [19] it is one of the physics informed models because it uses differentiation technique and applies the Hamiltonian equations. In this case the $\mathbf{x} = [\mathbf{q}, \mathbf{p}]$ are canonical or natural coordinates(Cartesian space). The forward pass looks like:

$$H_{\Theta} = \mathbf{f}(\mathbf{x}|\Theta) \quad (4.5)$$

$$\frac{d\mathbf{x}}{dt}(\mathbf{x}) = \left[\frac{\partial H_{\Theta}}{\partial \mathbf{p}}(\mathbf{x}), -\frac{\partial H_{\Theta}}{\partial \mathbf{q}}(\mathbf{x}) \right]^T \quad (4.6)$$

In the paper they train it only over the produced vectorfield. For our experiments we made rollout part for the network to create trajectory. To make trajectory we use ODE solver

$$\mathbf{x} = \text{odeint} \left(\frac{d\mathbf{x}}{dt}, \mathbf{x}_0, t \right) \quad (4.7)$$

The odeint operator in this equation is a solver for ODEs like euler or RK4 method. We can observe it in the figure from chapter Introduction 1.2.

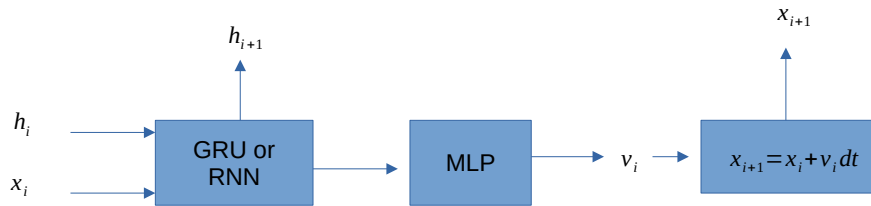


Figure 4.1: Architecture of time-stepper model

4.4 Graph Neural Network

Graph neural Networks are very new neural architecture which is available today. It is used broadly in social sciences and in chemistry like generating new chemical compounds. The main part of GNN is a graph \mathbf{G} . It is defined as a collection of vertices and edges $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$ where $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_i, \dots, \mathbf{v}_n\}$ has n vertices and $\mathbf{E} = \{\mathbf{e}_1, \dots, \mathbf{e}_i, \dots, \mathbf{e}_m\}$ has m edges.

The edge is connection between two nodes it can be directed (only one direction between the nodes) or undirected (both directions between the nodes). This is very important because the graph has many properties which we can use building it as Graph Neural Model.

This connectivity can be represented with adjacency matrix \mathbf{A} . Adjacency matrix is denoted

as $\mathbf{A} \in \{0, 1\}^{n \times n}$

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } (\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \quad (4.8)$$

Most important property of graphs is creating the normalized Laplacian matrix. This is foundation of the idea about graph neural networks and it is defined as:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{\frac{1}{2}} \quad (4.9)$$

where \mathbf{D} is degree matrix. Degree of a node is the number of nodes that are adjacent to the node in question. This matrix has only diagonal elements.

In the figure 4.2 you can see a representation of the graph together with adjacency matrix. We recognize 3 main training types of GNNs and that are graph-, edge and node-oriented

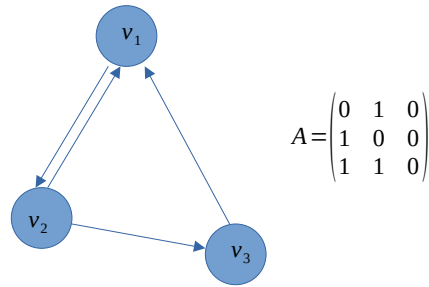


Figure 4.2: Graph representation

learning.

We will do only node oriented learning because we care that our data from the nodes comes dynamically right. For this we implemented two Graph Neural Layer to our work.

4.4.1 Convolutional Graph Neural Network

First idea about graph neural network was Spectral Graph Network (SGN) [4] in signal processing tasks which is based on spectrality of the modified Laplacian in form

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{\frac{1}{2}} \mathbf{W} \mathbf{D}^{\frac{1}{2}} \quad (4.10)$$

where we exchange adjacency matrix with learnable weight matrix. Because of cost of computation and hard backpropagation, the ChebNet was introduced[13]. It was dependent of fourier transformation and had high computational complexity. First Kipf and Welling used both ideas and simplified Laplacian to create broadly used architecture called convolutional graph network [21].

The propagation of the layer mathematically looks like

$$\mathbf{h}^{(i+1)} = \sigma(\tilde{\mathbf{D}}^{\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{\frac{1}{2}} \mathbf{W} \mathbf{h}^{(i)}) \quad (4.11)$$

where we modify adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ as $\tilde{\mathbf{D}}_{ij} = \sum_{j=1}^N \tilde{\mathbf{A}}_{ij}$. This mathematical expression of propagation of the graph layer is equivalent to message passing on the and aggregation on the graph from and between the neighbouring nodes. In this case with \sum operator.

There are other types of aggregations like mean, sum and sub. The default message passing and aggregation formula is defined as

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right) \quad (4.12)$$

where γ and ϕ represents MLPs and \bigoplus the aggregation function. While the neural graph network doesn't depend on the graph structure itself, it relies on message passing and aggregation between the nodes. With this understanding, we can develop more diverse designs of graph neural networks. We should not forget, that those message passing and aggregation can be parallelized and calculated on GPUs improving performance. In our use cases we will use Framework DGL[46].

4.4.2 Gated Attention Network

In paper [45] is introduced new type of architecture which incorporates weighting factors/attentional coefficients. This weighting factors α shows importance of the outgoing node to the ingoing node in question. The forward pass of one layer is defined as:

$$\mathbf{z}_i^{(l)} = \mathbf{W}_f^{(l)} \mathbf{h}_i^{(l)}, \quad (4.13)$$

$$e_{ij}^{(l)} = \text{LeakyReLU}(\mathbf{W}_a^{(l)T} (\mathbf{z}_i^{(l)} \parallel \mathbf{z}_j^{(l)})), \quad (4.14)$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \quad (4.15)$$

$$\mathbf{h}_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} \mathbf{z}_j^{(l)} \right) \quad (4.16)$$

This architecture is computationally efficient and is applicable to inductive problems, thus the creation of weighting factors doesn't depend on global graph structure. We will have always one weighting factor per edge no matter which graph we use. For better training and preventing overfitting, we can set a dropout at MLP which creates weighting factors [42], it is even suggested for the small graphs.

4.5 Graph Hamiltonian Neural Network

This architecture is inspired on implementation of HNN [19] and paper [40]. In the paper they implemented own Graph Neural Network with complicated architecture called GN, which has many applications in deep learning. They tried this architecture incorporate with odeint solver to create HOGN model. In our case we will use conventional GNN model to create the same. In our cases our graph will built without loop nodes: Nodes with edge connected itself. With many trials we achieve same valued features at the nodes using sum or mean aggregation at fully connected graph, With discarding self loops we don't have this problem anymore.

It is important that the output of our Graph network is a scalar value because it represents \mathcal{H} . Without this condition, auto differentiation won't work. There is the formula for

GHNN.

$$H_{\Theta} = \mathbf{f}(\mathbf{x}|\Theta) = \text{GNN layer}(\mathbf{x}) \quad (4.17)$$

$$\frac{d\mathbf{x}}{dt} = \left[\frac{\partial H_{\Theta}}{\partial \mathbf{p}}(\mathbf{x}), -\frac{\partial H_{\Theta}}{\partial \mathbf{q}}(\mathbf{x}) \right]^T \quad (4.18)$$

The Graph Neural Network can be made with more layers and we can add activation functions. In the experiments we used GAT layer.

This model gives trajectory through using of one ode solvers. Its architecture you can observe in Figure 4.3.

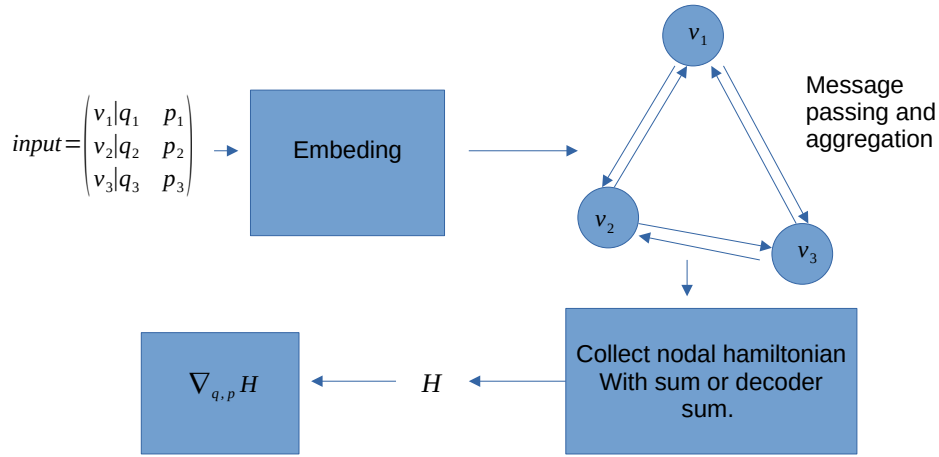


Figure 4.3: Architecture of Graph Neural Hamiltonian Network

4.6 Improved Graph Hamiltonian Neural Network

As GHNN mentioned before didn't showed very good performance we were forced to find some new architecture. We found new layer which we could use and it is inspired by paper [47]. The adapting adjacency matrix is very interesting concept but in the scope of coding, the DGL don't allow fitting the adjacency matrix.

In the paper adaptive adjacency matrix was calculated as

$$\mathbf{A}_{\text{adp}} = \text{Softmax}(\text{ReLU}(\mathbf{E}_1 \mathbf{E}_2^T)), \quad (4.19)$$

This procedure looks very alike attentional coefficient calculation. We made our architecture for one layer as

$$\mathbf{y}_{gcn} = \text{GCN}(\mathbf{x}), \quad (4.20)$$

$$\mathbf{y}_{gat} = \text{GAT}(\mathbf{x}), \quad (4.21)$$

$$\mathbf{y}_{mlp} = \text{MLP}(\mathbf{y}_{gcn} | \mathbf{y}_{gat}), \quad (4.22)$$

$$\mathbf{y} = \mathbf{y}_{gcn} + \mathbf{y}_{gat} + \mathbf{y}_{mlp} \quad (4.23)$$

$$H_{\Theta} = \sum \mathbf{y} \text{ sum of the all features} \quad (4.24)$$

$$\frac{d\mathbf{x}}{dt} = \left[\frac{\partial H_{\Theta}}{\partial \mathbf{p}}(\mathbf{x}), -\frac{\partial H_{\Theta}}{\partial \mathbf{q}}(\mathbf{x}) \right]^T \quad (4.25)$$

We used the multilayer perceptron as some sort of decoder which uses concatenated input of outputs from GAT and GCN layer. After that we sum everything together.

We will use it in some of our experiments. The architecture can be observed in Figure 4.4

This architecture shows better performance then GHNN, but it is relatively slow.

4.7 Gated Recurrent Graph Hamiltonian Neural Network - GRUGHNN

In previous section we introduced the GHNN, which is base an Hamiltonian layer and ode-solver. This architecture is not different but improved with a GRU unit at the beginning of the input. this model is equivalent to the GRU stepper but we added a GHNN in it. It

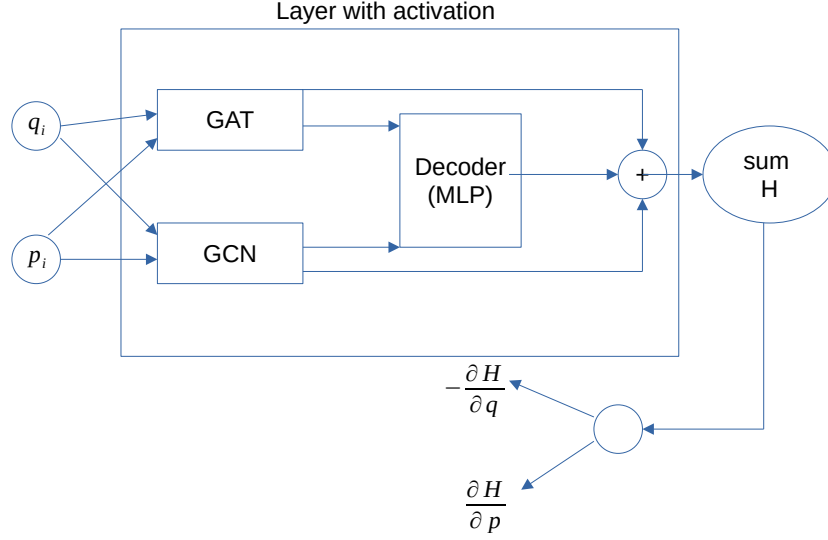


Figure 4.4: Architecture of improved GHNN

is Graph Neural Network with a memory feature. If we compare this model to spatio-temporal models you can see similarities. Only two models have similar architecture to this. We will mention Gated Graph Sequence Neural Network[25] and GNN-GRU[15], but they don't use auto differentiation. Without autodifferentiation we can not call our architectures as "hamiltonian".

$$[\mathbf{a}_i, \mathbf{h}_{i+1}] = \text{GRU}(\mathbf{x}_i, \mathbf{h}_i) \quad (4.26)$$

$$H_i = \text{GNN}(\mathbf{a}_i) \quad (4.27)$$

$$\mathbf{v}_i = \left[\frac{\partial H_i}{\partial \mathbf{p}}(\mathbf{x}), -\frac{\partial H_i}{\partial \mathbf{q}}(\mathbf{x}) \right]^T \quad (4.28)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i \cdot dt \quad (4.29)$$

Architecture of our network you can observe in Figure 4.5

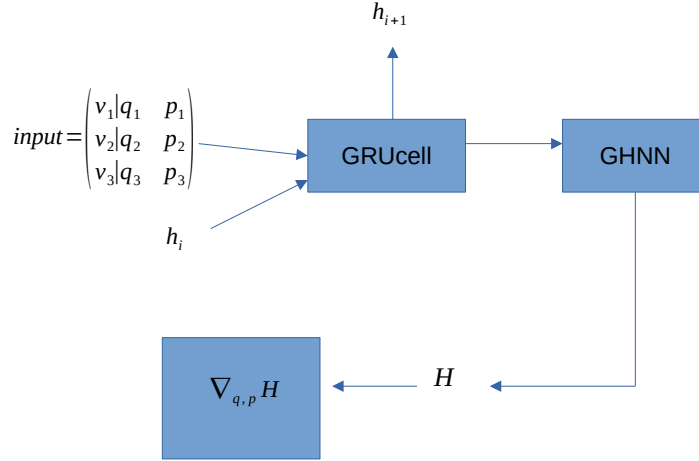


Figure 4.5: Architecture of GRUGHNN

4.8 Training for Physics Informed Models based on ODE

As we introduced the models we can observe that they are intended to be used in some ODE solver. That means that the data and our datasets are time series oriented. The specifics of time-series data are they have three dimensions which are time t , sample space s and feature space f . Let us define it as $\mathcal{V}^{t,s,f}$.

This sample is very specific, and in case of mini-batching we can't just pick some values and make a batch, the data follow the timeflow. We need, to create the batches, specify the

length of the sequence and how many sequences are in the sample. We call it `timebatch size` and `batch size`.

Such `time batches` we call snapshots $\mathcal{V}^{t,f}$, because it represents recorded confined sequence for one specific initial value. For example we have a sequence with 128 time points and with `timebatch size` = 32 we can make 96 snapshots with sequence size 32. Even if the data repeats itself it is important that we need to use many initial values as possible.

If we have too many samples or too big sequence we can implement `stride` technique. We specify which sequence will we recorded for our dataset. For example with `stride`=4 we specify that we want every forth initial value and its sequence recorded. Resulting in smaller dataset.

In case of graph batched data we need to be careful. First our sample space will change in $\mathcal{V}^{s \times n, t, f}$ where n means nodes. First we need to transform our made snapshots to graph structure, which means to place the features at their corresponding nodes and we get $\mathcal{V}^{n, t, f}$. The batching is very straight forward, it could be called "stacking the graphs". We make one big graph using s times graphed independent snapshots. We call it independent because there are no edge connections between other graphs and propagating between nodes of the graphs is done in parallel.

In case of Hamiltonian Neural Networks and extracting a hamiltonian values of the independent graphs we suggest to unbatch it first and create a normal batch of hamiltonian values. It makes calculation of hamiltonian easier. If you calculate from the batched form, you need to be careful because the hamiltonian can be incorrect because, we need to sum the features in every independent graph separately.

5 Experiments

Now we perform comparison experiments to compare the state of the art models, common models and PINN(Graph Neural Network) models. Our goal is implicitly train our dataset to proposed networks in fully supervised manner. In cases of PINN models we supervise trajectory, vectorfield and hamiltonian energy made by the model itself. Our goal is to show if the models are capable implicitly train the data.

We decided to do 3 types of experiments.

In first experiment we will test the training capabilities of common and state of the art models on our made datasets(oscillator, twobody problem, threebody problem).

In second experiment we will test our PINN models on threebody problem scenarios. In third experiment we will train the PINN models on N-Pendulum and N-body dataset and look at capability of changing the graph structure: adding or discarding a node.

5.1 Experimentation of the Models on the Datasets

In this section we experiment on following models

- Multilayer perceptron
 - 4 layers with size: oscillator 128,twobody 256, threebody 512
 - activations: tanh, relu, relu, identity
- NeuralODE
 - 3 layers with size oscillator 128, twobody 256, threebody 512
 - activations: tanh, relu, identity
- GRU - size: oscillator 256,

-
- RNN - size: oscillator 256,
 - GRU time stepper - size: oscillator 256, twobody 256, threebody 512
 - RNN time stepper - size: oscillator 256, twobody 256, threebody 512

on various datasets made with hamiltonan equation. We used optimizer AdamW and loss function

$$Loss = \text{Huber}(\mathbf{y}, \hat{\mathbf{y}}) \quad (5.1)$$

For the Hamiltonian accuracy we used MSE Loss. We need to disclaim that the hamiltonian is not part of the model. It predefined function by the real model, we just use it to see if the trajectory prediction shows possible conservation of energy inside the model as a black box problem.

5.1.1 Harmonic Oscillator

In previous chapter we disscused about creation of the Oscillator Dataset. In this Experiment we chose $k = 0.8$ and $m = 1.0$ and made 30 trajectories for training data and 3 trajectories for test data with 128 time points in one period.

The trajectories are different because we set our Hamiltonian/Total Energy in region $H = [5, 15]$.

We predict very well generalization because we are working with following type of equation

$$\dot{\mathbf{x}} = \mathbf{Ax}. \quad (5.2)$$

The matrix \mathbf{A} for every trajectory is fully constant and it should be trivial case.

We train the models with "rollout" technique. It is training step where we take beginning of our relevant trajectory and we let the model to predict rest of trajectory.

We managed to make batch-wised training making the snapshots of the trajectory with length of 32 time points. We took 30 train trajectories and 3 test trajectories.

With such training we got following results in figures 5.1,5.2:

- Multilayer perceptron

After observation we can conclude that the baseline MLP did learned hamiltonian system but not accurate enough. In this case which is learned for 1000 epochs we see that we have maybe overtrained the network. From the energy accuracy we can see that the metric for energy accuracy which is made over the trajectory is little bit to high in all of the epochs.

- NeuralODE

In this case we trained properly the hamiltonian system. Energy doesn't show conservative property but the difference between true hamiltonian isn't too high. In comparison with MLP case the energy accuracy here is more accurate than MLP in every epoch.

- GRU

In this case we see classic example of overtraining, After 500 epoch the losses and energy accuracy started to rise and a evaluation sample shows similar result to MLP case.

- RNN In this case RNN shows overall good performance and it could be compare to NeuralODE case. The trajectory doesn't show periodic movement, but overall the hamiltonian shows conservative property

- GRU Time Stepper

GRU stepper is very interesting hamiltonian accuracy shows better performance than NeuralODE. in our Evaluation sample we got result where movement isn't periodic and overshoots the end point. The Hamiltonian of evaluation sample rises. We have assumption that we see here numerical approximation error due to model architecture.

- RNN Time Stepper

This model shows interesting behaviour in which evaluation plot looks similar to GRU Stepper but the energy hold conservative property. The energy accuracy is in the range with NeuralODE.

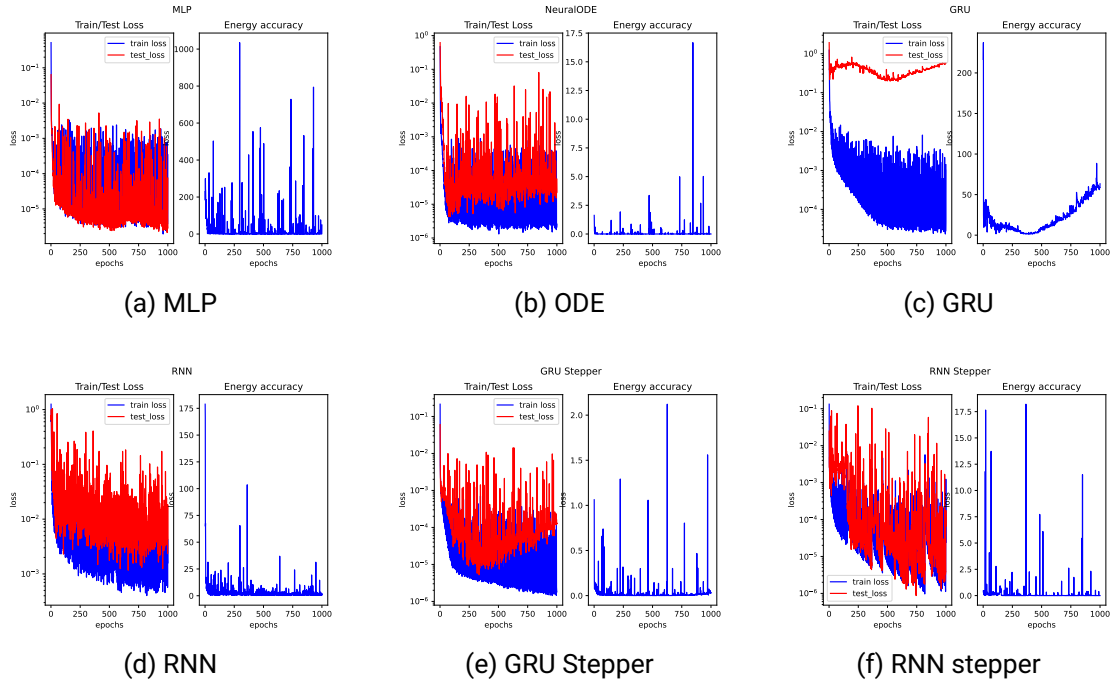


Figure 5.1: Losses and energy accuracy on various neural models(Oscillator)

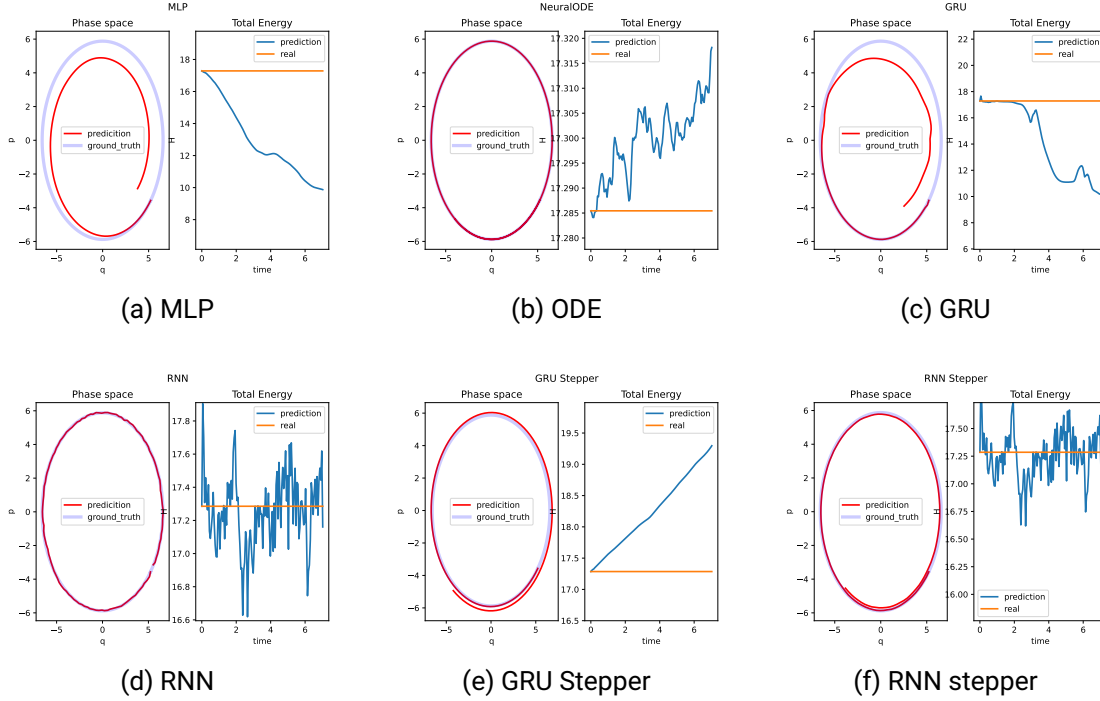


Figure 5.2: Evaluation of the sample on the models, Trajectory and Energy(Oscillator)

5.1.2 Twobody-problem

In previous chapter we discussed about creation of the Twobody Dataset. In this Experiment we chose masses $m = \{1.0, 3.0\}$ and $G = 1$, we made 30 trajectories for training data and 3 trajectories for test data with 128 time points in one period. The train data is batched and we have done batched learning

$$\dot{\mathbf{x}} = \mathbf{A}(\|\mathbf{q}_1(t=0)\|)\mathbf{x}, \mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \mathbf{p} \end{bmatrix} \quad (5.3)$$

The matrix \mathbf{A} is constant only for dynamics of one trajectory. We train the models with "rollout" technique, exactly like explained in the oscillator case and we got following results 5.3, 5.4, 5.5.

- Multilayer perceptron
From the loss figure we see that the model has learned a hamiltonian system. Even

the energy accuracy looks very stable. On the other hand the evaluation plot is little bit inaccurate.

- NeuralODE
The trajectory is accurate and energy at evaluation sample looks that hold conservative property. Over all it is most accurate model of all.
- GRU
similar to MLP the trajectory is little bit inaccurate but the energy shows conservative property.
- RNN
It looks similar to GRU but the trajectory looks like it oscillates around ground truth data
- GRU Time Stepper
The trajectory and energy on the evaluation sample looks inaccurate but it follows the flow of trajectory. '
- RNN Time Stepper
It has same characteristics as GRU Stepper but it has slightly worse performance,

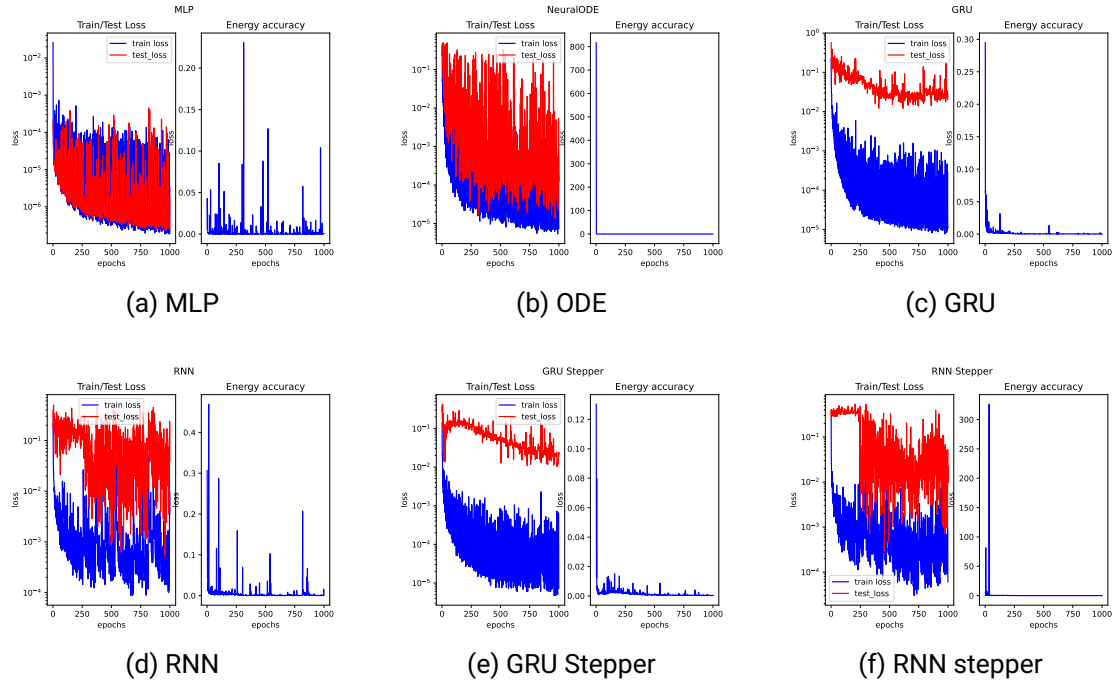


Figure 5.3: Losses and energy accuracy on various neural models(Oscillator)

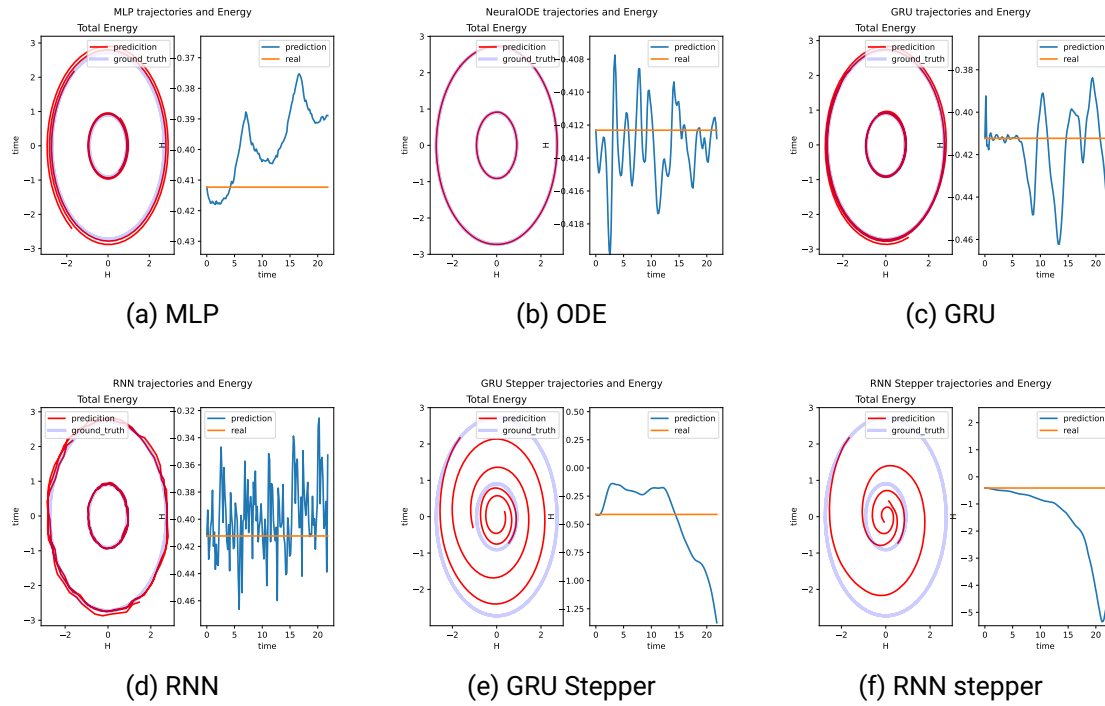


Figure 5.4: Evaluation of the sample on the models, Trajectory and Energy(Twobody)

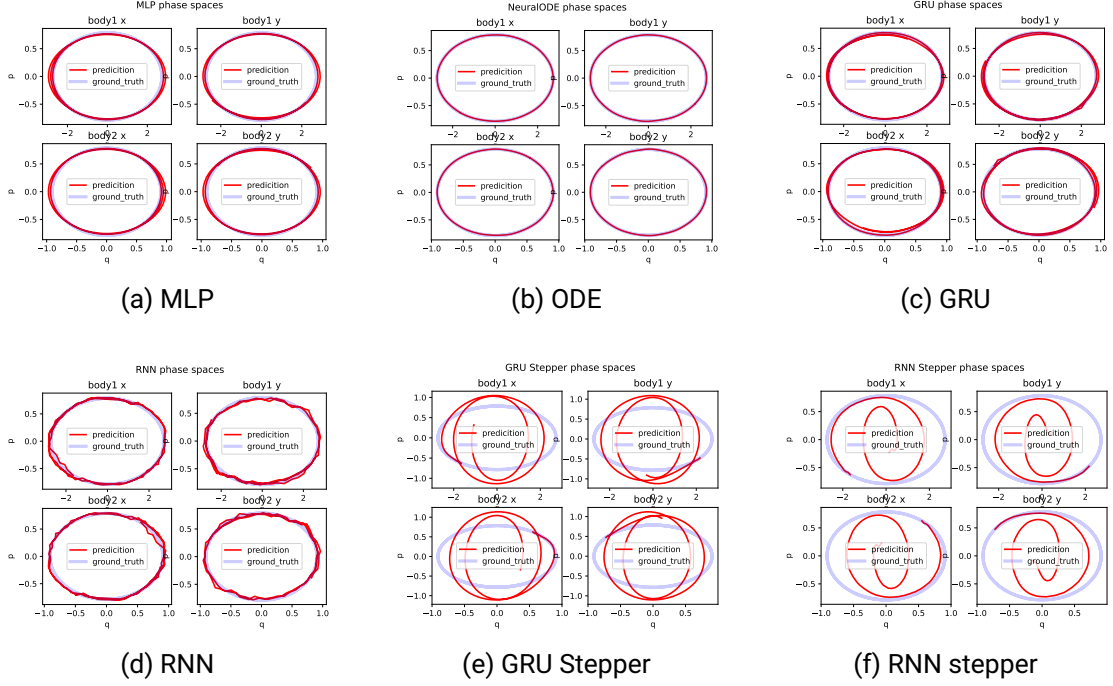


Figure 5.5: Evaluation of the sample on the models, phase space (Twobody)

5.1.3 Three-body problem

In previous chapter we discussed about creation of the Three-body Dataset. In this Experiment we chose masses $m = \{1.0, 1.0, 1.0\}$ and $G = 1$, we made 30 trajectories for training data and 3 trajectories for test data with 128 time points in one period.

The trajectories are different because we set $\alpha \in [0, 1.0\text{rad}]$ as a rotation around the origin. In this case we have same Total Energy for all trajectories

$$\dot{\mathbf{x}} = \mathbf{A}(\mathbf{q})\mathbf{x}, \mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \mathbf{p} \end{bmatrix} \quad (5.4)$$

The matrix \mathbf{A} is variational and depends strictly on position of the body in every timestep. We used batched technique to learn the dataset and we got following results in Figures 5.6, 5.7, 5.8

-
- Multilayer perceptron
It trained well, Energy accuracy has gone minimal after 150 epochs of training and shows very good trajectory.
 - NeuralODE
Interestingly enough the trajectory on evaluation sample looks accurate but total energy doesn't look accurate. Energy accuracy metric is after 200 epoch minimal which is good sign. ' '
 - GRU
The model looks overtrained if we look at loss figure. Furthermore trajectory and Energy is inaccurate, but it follows the figure 8 shape.
 - RNN
This model has failed.
 - GRU Time Stepper
It shows very good performance, similar to Neural ODE.
 - RNN Time Stepper
It looks better than RNN but it looks like it has issues to learn for data.

From all Datasets we can conclude that RNN and RNN Stepper are bad models for the physics problems. On the other hand RNN model shows little bit of good performance but it isn't right model for this task. Through experimentations we see that NeuralODE and GRU Stepper show better performance than the baseline MLP.

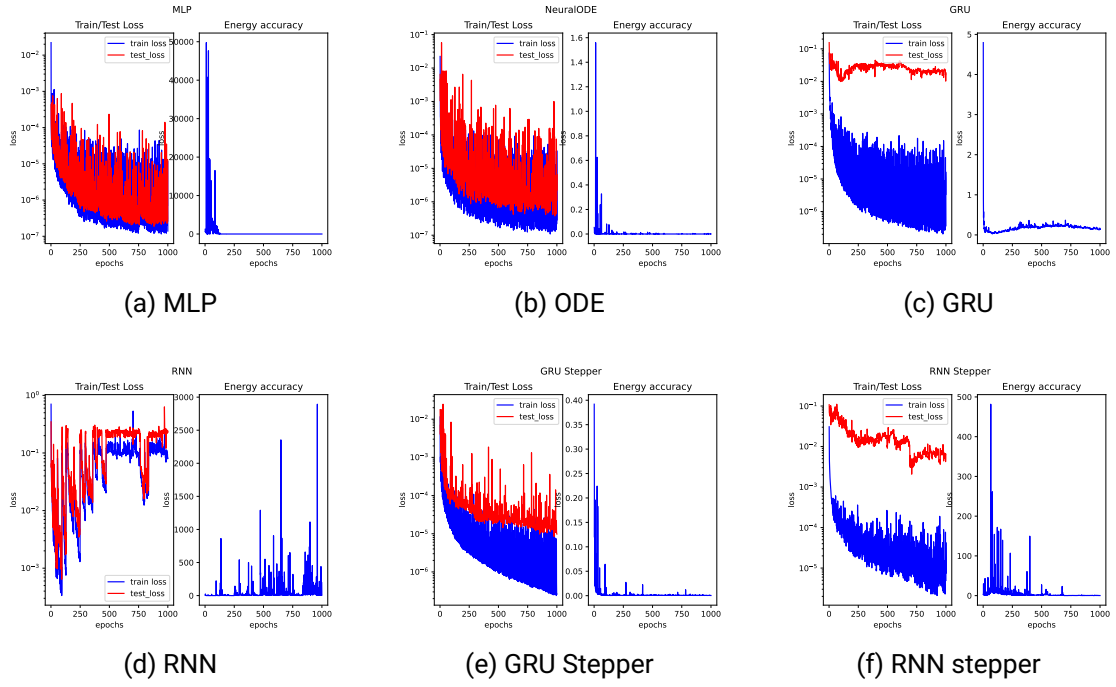


Figure 5.6: Losses and energy accuracy on various neural models(Threebody)

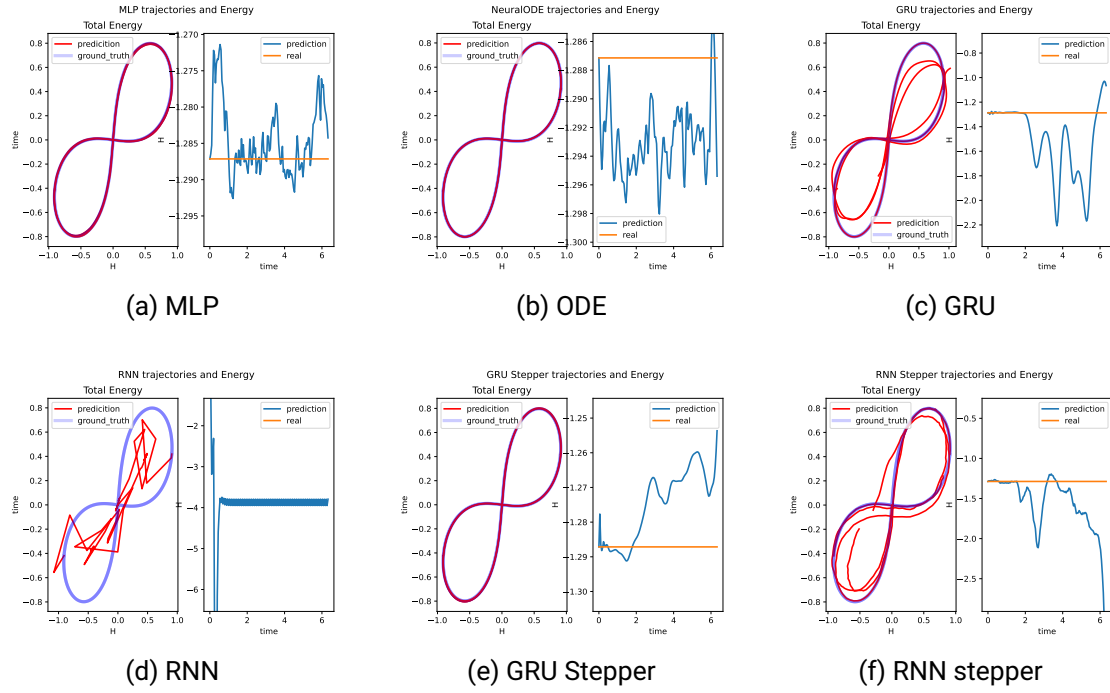


Figure 5.7: Evaluation of the sample on the models, Trajectory and Energy(Threebody)

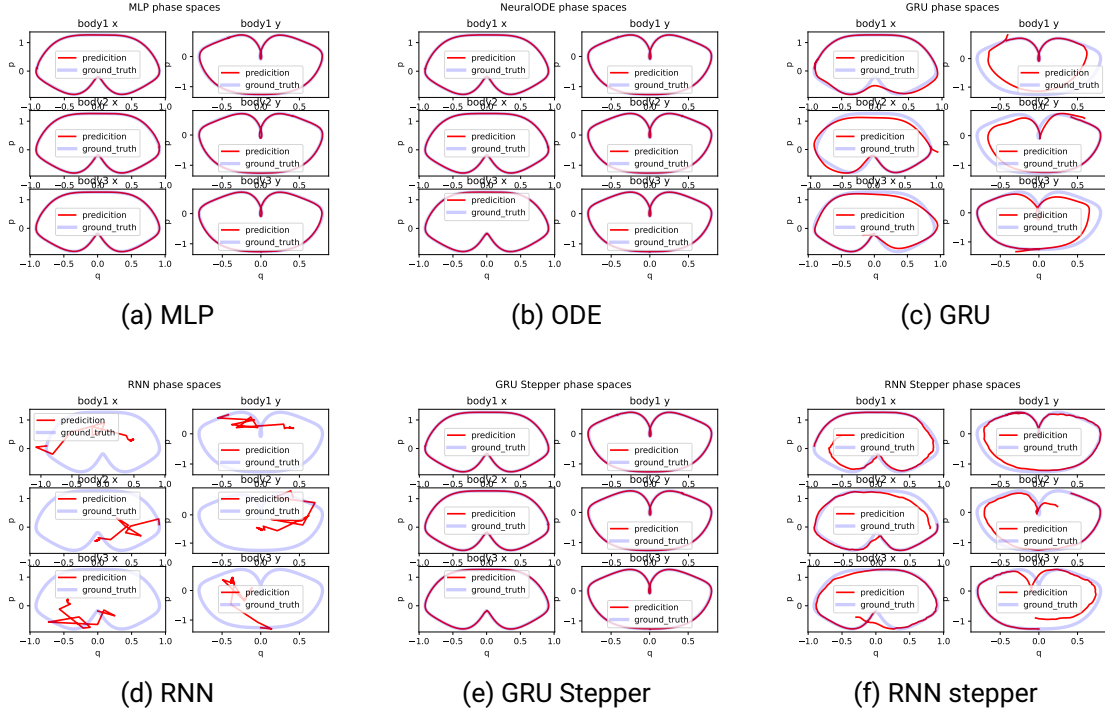


Figure 5.8: Evaluation of the sample on the models, phase space (Threebody)

5.2 Experimentation of Physics informed datasets on three body Dataset

After the showcase of our datasets and architectures, we will now do some training of the physics informed neural networks. We choose two datasets, one will three body problem identical to the experiment before and one dataset with mixed solutions of the three body problems. We chosed carefully similar trajectories for better training performance. the models are:

- improved Graph Hamiltonian Neural Network
- Hamiltonian Neural Network
- GRU- Graph Hamiltonian Neural Network(GAT layer)

This models are called physics informed not only because they are made through differentiation but all of the models gives Total energy from the model directly. With this information we can make physics-informed loss. The physics informed-loss that we chose is

$$Loss = \omega_r MSE(\dot{\mathbf{q}}, \frac{d}{dt}\hat{\mathbf{q}}) + \omega_v MSE(\dot{\mathbf{p}}, \frac{d}{dt}\hat{\mathbf{p}}) + \omega_h MSE(H(\mathbf{q}, \mathbf{p}), \hat{H}) \quad (5.5)$$

with $[\omega_r, \omega_v, \omega_h] = 1.0, 1.0, 0.05$.

Hyperparameters of our models:

- HNN
2 layers with tanH activation after the first layer,
hidden size: 256
- improved GHNN
2 Combined(GAT + GCN) layers with tanH activation with dropout at attention
 $p = 0.65$
hidden size: 256
output size before constructing H: 16
- GRUGHNN
GRUcell + 2 GAT layers with tanH activation with dropout at attention $p = 0.65$
hidden size: 256
output size before constructing H: 16

We trained our models for 200 epochs.

5.2.1 Threebody dataset

For this experiment we choosed 10 trajectories in domain $\alpha = \pi/8$. α is maximal angle which creates rotational region for creation of the dataset. From those 10 trajectories we made batches for training data and test data. The time sequence is 32 timepoints and 32 samples pro batch. The is $dt=0.05$. After 100 epochs we get following results which can be observed in Figures 5.10 5.9.

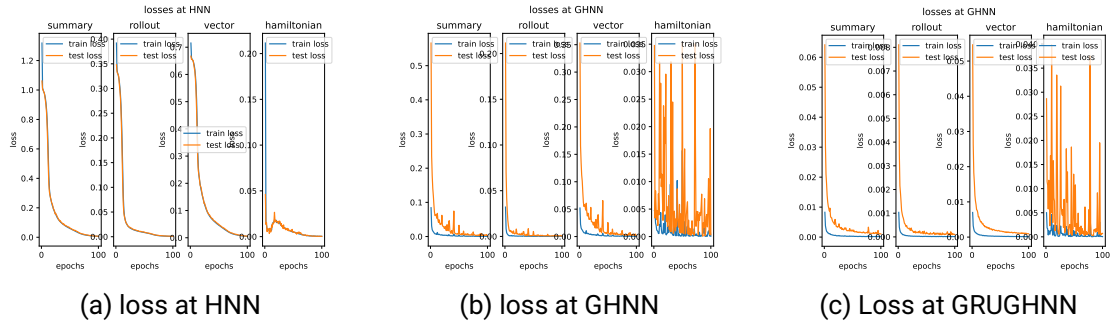


Figure 5.9: Losses at various pinn models (threebody dataset)

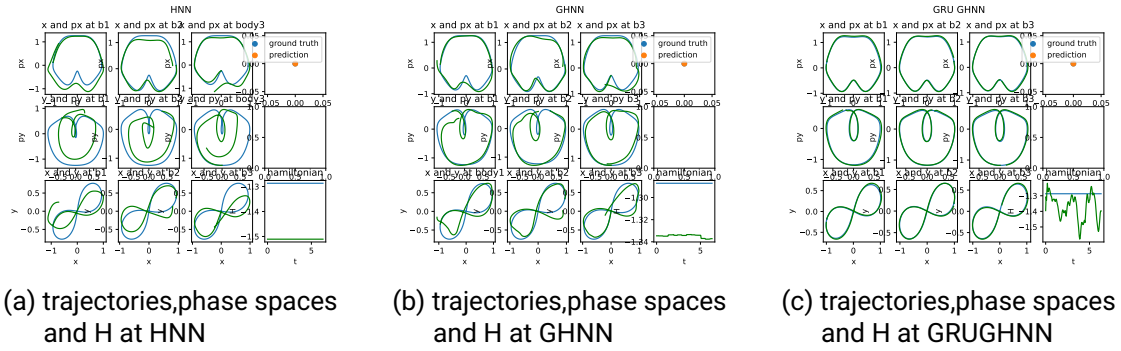


Figure 5.10: Trajectories , phase spaces and hamiltonian at various PINN models

We can observe that hamiltonian at HNN and GHNN model is overall constant. Best performance shows the GRUGHNN where we have almost perfect trajectory with oscillating hamiltonian.

5.2.2 Figure 8 solutions

In this Dataset we use 4 Types of Figure 8 solutions which differs from eachother in period and energy. Those samples are in the table 5.1 We use v.7.a, v.7.b, v.7.d solutions for training and testing and figure 8 for evaluation and visualization. We took 5 trajectories from every solution and it is fully supervised with loss mentioned before. Because of different periods, which results with different number of time points, we implemented

stride and took every forth snapshot from the trajectories. For batching our time sequences have 32 timepoints and there are 32 samples pro batch.
In figures 5.11, 5.16, we can see their loss progression, trajectories and phase spaces.

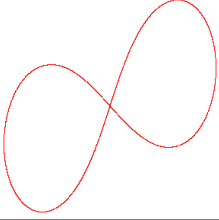
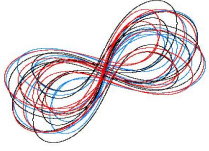
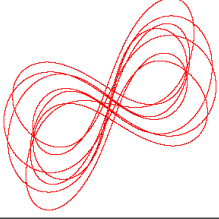
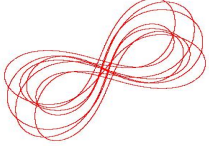
	<p>Specifications</p> <ul style="list-style-type: none"> • Name : v1 a/ Figure 8 • H: -1.287146 • T: 6.324449s
	<p>Specifications</p> <ul style="list-style-type: none"> • Name : v.7.a • H: -1.570451 • T: 32.849071s
	<p>Specifications</p> <ul style="list-style-type: none"> • Name : v.7.c • H: -1.504302 • T: 35.043086s
	<p>Specifications</p> <ul style="list-style-type: none"> • Name : v.7.d • H: -1.080763 • T: 57.545253s

Table 5.1: Figure 8 Dataset samples[38]

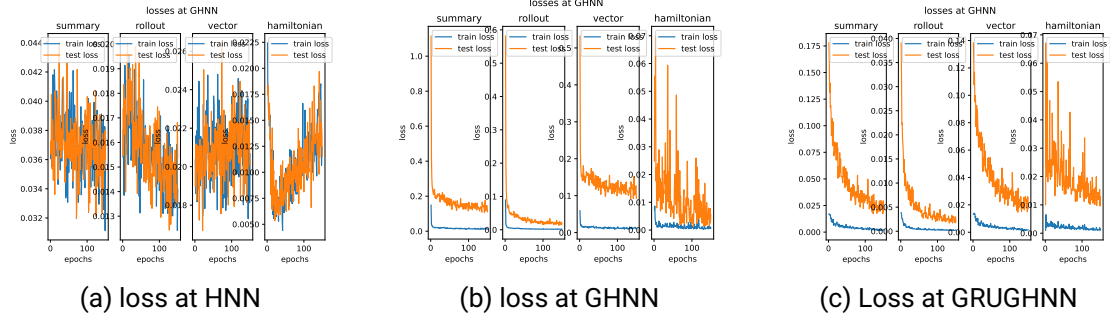


Figure 5.11: Losses at various models (figure dataset)

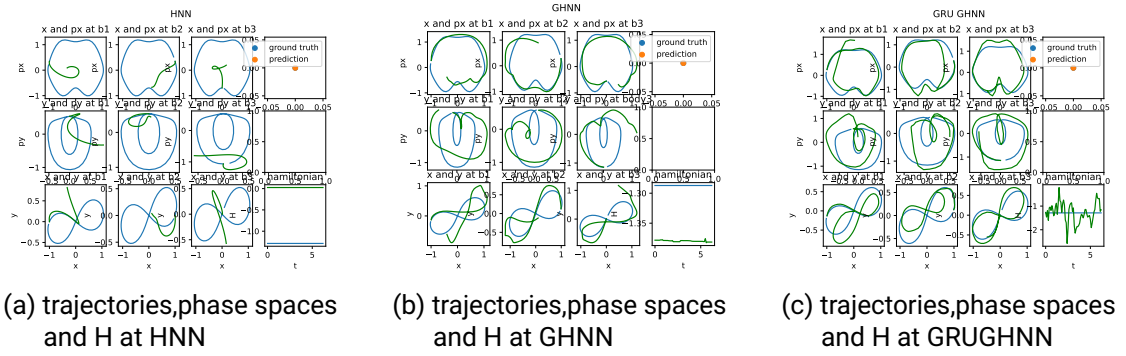


Figure 5.12: Trajectories , phase spaces and hamiltonian at various models (figure dataset)

We can clearly see that the proposed models are better than baseline HNN. They show better performance and better data fitting. Interestingly the hamiltonian on HNN and improved GHNN is almost constant and the trajectories have movement. Even though the trajectories of GRUGHNN are not accurate and hamiltonian tries to be conservative, it tries to follow the ground truth trajectory better than other models.

5.3 Experimentation of Physics informed models on N body problem and N pendelum

In this experiment we took only physics informed graph models GHNN and GRUGHNN. Here we have two samples of datasets for n-pendelum and n body problem. The goal of this experiment is to see if the model is capable to generalize or make correct trajectory adding or discarding one more degree of freedom. For graph neural network that shouldn't be a problem because the size of the network depends not at size of the graph but on the size of the feature space. HNN model in this case is not usable.

We will use simmlar hyperparameters for our models for both cases:

- GHNN
2 GAT layers with tanH activation with dropout at attention $p = 0.65$
hidden size: 128
output size before constructing H: 16
- GRUGHNN
GRUcell + 2 GAT layers with tanH activation with dropout at attention $p = 0.65$
hidden size: 128
output size before constructing H: 16

5.3.1 N-body pendulum

In this case we took 20 trajectories with 128 timepoints. The dataset is created with randomised inital values: $p_{\Theta} = 0$ and $\Theta = [-\pi, \pi]$ but we discarded those which trajectory is $|\Theta| > 2\pi$. We batched training data with batch size of 32 with 32 points in trajectory, vectorfield and hamiltonian energy. We supervised within the loss

$$Loss = \omega_r(Hub(\mathbf{q}, \hat{\mathbf{q}}) + Hub(\mathbf{p}, \hat{\mathbf{p}})) + \omega_v(Hub(\dot{\mathbf{q}}, \frac{d}{dt}\hat{\mathbf{q}}) + Hub(\dot{\mathbf{p}}, \frac{d}{dt}\hat{\mathbf{p}})) + \omega_h Hub(H(\mathbf{q}, \mathbf{p}), \hat{H}) \quad (5.6)$$

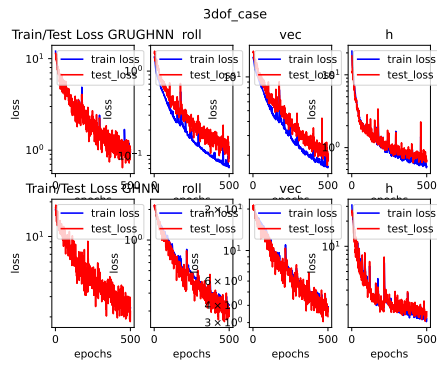
with $\omega = 1.00, 0.5, 0.1$.

Table 5.2: Evaluation of 3dof sample on various models and training procedures

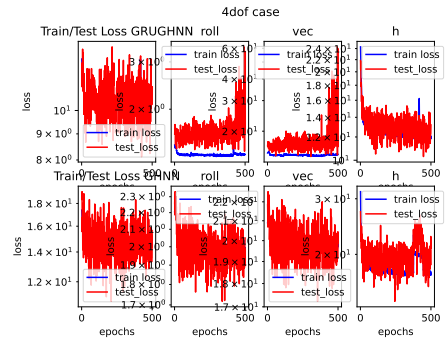
dof3	roll	vec	h
HGNN 3dof	8.58	22.91	0.57
HGNN 4dof	150.74	501.93	10.93
GRUHGNN 3dof	2.57	21,7211	6,7363
GRUHGNN 4dof	5.6	29.85	5.96

Table 5.3: Evaluation of 4dof sample on various models and training procedures

dof4	roll	vec	h
HGNN 3dof	12.35	31.85	2.12
HGNN 4dof	10.45	32.28	22.9
GRUHGNN 3dof	5.61	24.72	10.71
GRUHGNN 4dof	7.00	33.1747	12.5609



(a) Losses at 3dof training



(b) Losses at 4dof training

Figure 5.13: Losses till 500 epochs(Pendulum)

Table 5.4: Evaluation of 3dof sample on GRUGNN and training procedures

dof3	roll	vec	h
GRUHGNN 3dof	3.49	23,17	7.7
GRUHGNN 4dof	6.71	26.6007	45.09

Table 5.5: Evaluation of 4dof sample on GRUGNN models and training procedures,

dof4	roll	vec	h
GRUGNN 3dof	59.33	24.95	20.82
GRUGNN 4dof	8.47	26.8	50.2

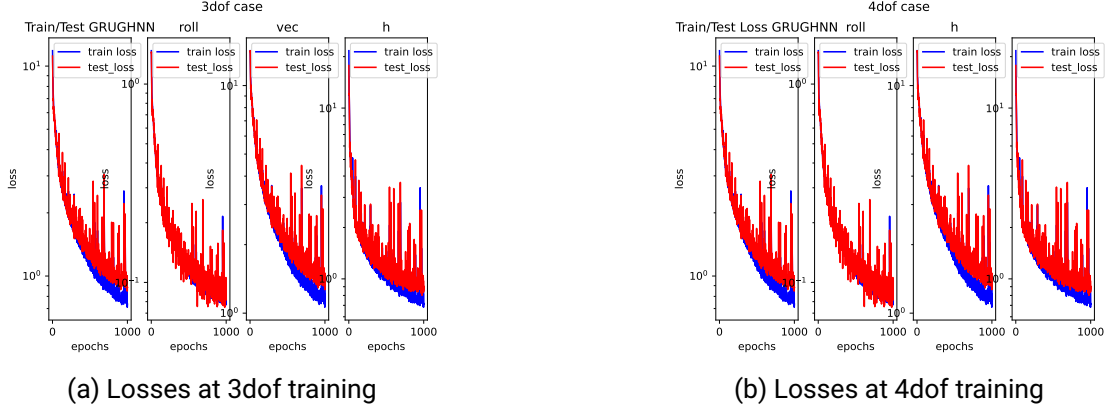


Figure 5.14: Losses till 1000 epochs on GRUGHNN(Pendulum)

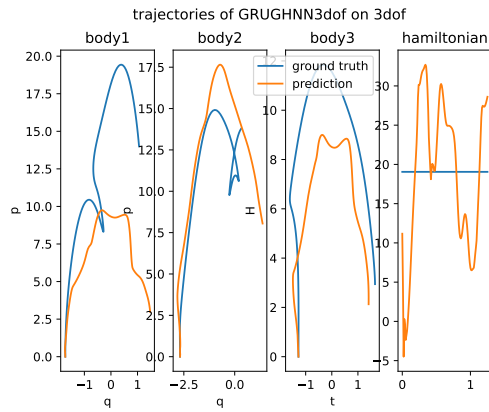
In the tables 5.4 and 5.5 we compared the HuberLoss values of evaluation trajectory of 3dof pendulum and 4dof pendulum on the PINN models, which are train on 3dof and 4dof dataset under 500 epochs. For better comparison we evaluated 1000 epochs only on GRUGHNN. Chosed trajectories for visualization iare fully unknown to the models.

From the values in the tables5.4 and 5.5 we can read that GHNN dosen't like losing degree of freedom. We see very big difference in those values. In other cases there is difference but not exaggerated.

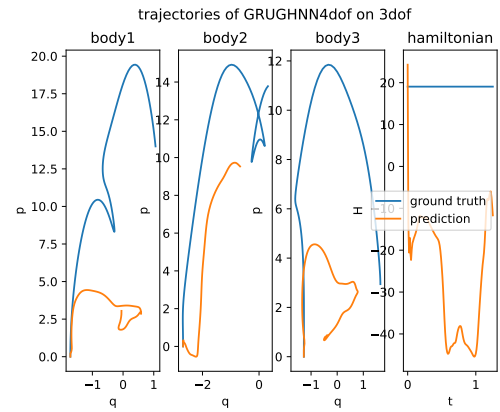
Interestingly Huberloss for the vector field at GRUGHNN is realtivly stable in comparison with GHNN. The trajectories are observable in figure 5.15

5.3.2 N-body problem

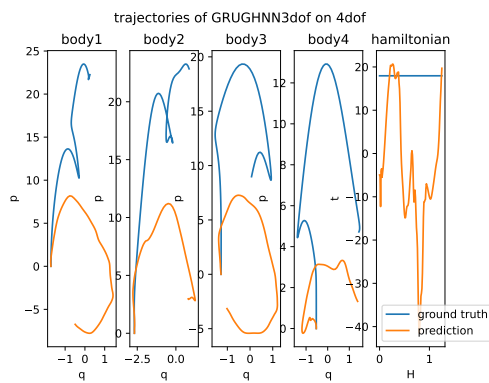
In the similar manner as N-pendulum we made N-body Problem. First we needed to set up domain of movement and we chosed 20 samples of trajectories of 128 timesteps with $dt = 0.01$. In the dataset we have position velocity and energy and it is suprivised same as N-pendulum case and we reused only GRUGHNN model because it showed greatest



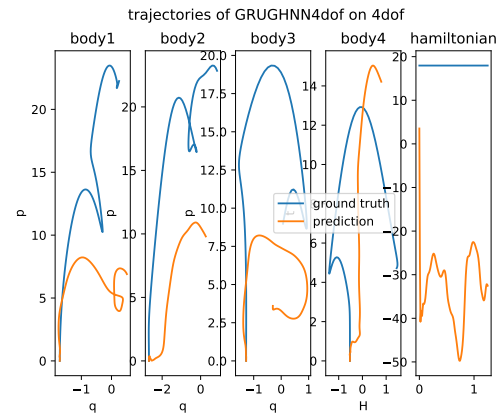
(a) GRUGHNN trained on 3dof sample and made 3dof trajectory



(b) GRUGHNN trained on 4dof sample and made 3dof trajectory



(c) GRUGHNN trained on 3dof sample and made 4dof trajectory



(d) GRUGHNN trained on 4dof sample and made 4dof trajectory

Figure 5.15: trajectories of pendelums after 1000 epochs

performance in last experiments. Training was done trough 1000 epochs. This is just proof of concept that adding and discarding the graph nodes is possible.

Table 5.6: Evaluation of 3dof sample on GRUGNN and training procedures, (Nbody)

dof3	roll	vec	h
GRUHGNN 3dof	0.88	9.19	6.03
GRUHGNN 4dof	0.87	6.97	5.81

Table 5.7: Evaluation of 4dof sample on GRUGNN and training procedures (Nbody)

dof4	roll	vec	h
GRUHGNN 3dof	1.25	7.39	9.37
GRUHGNN 4dof	1.08	10.11	9.37

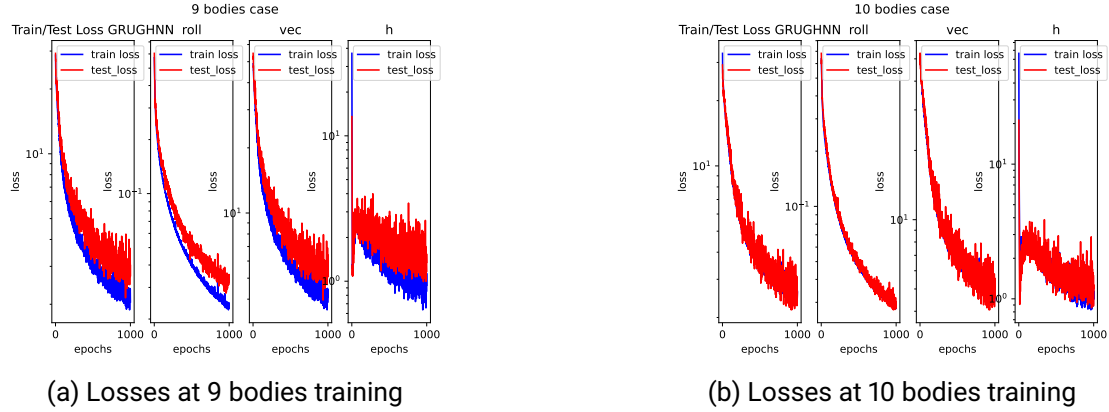
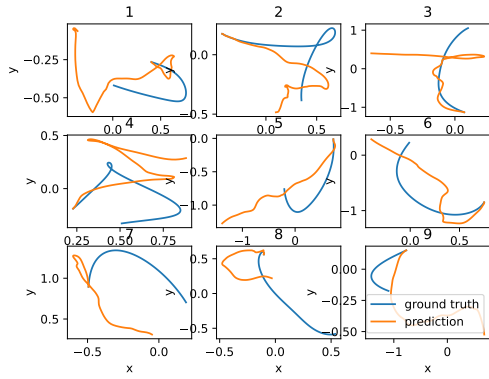
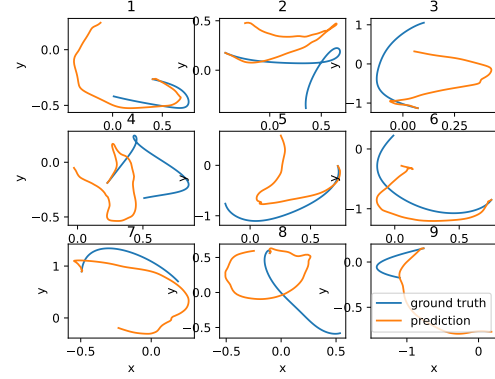


Figure 5.16: Losses till 1000 epochs on GRUGNN (Nbody)

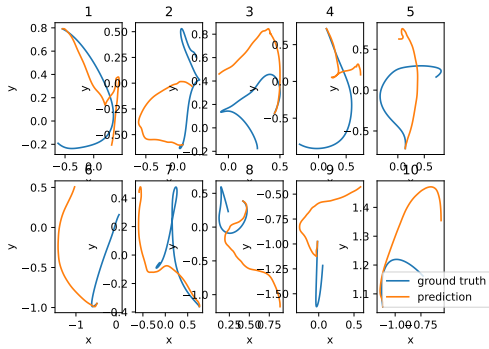
The results of the adding and discarding a body are really suprising good enough. The HuberLoss in the tables 5.6 and 5.7 shows relatively simmlar values Still the model tolerates more losing the body then gain it. This can be observed in ?? It is fully understandable looking at Nbody problem. Adding a body we could heavy disrupt the system. Mostly interesting result is hamiltonian figure 5.18. It shows that model trained on 9dof sample better accepts 10dof sample. Following the logic it is possible because we used GAT layers which at training of 4dof samples had few more weighting factors to create and fit. With those experiments we got an good insight in the capabilities in those PINN models.



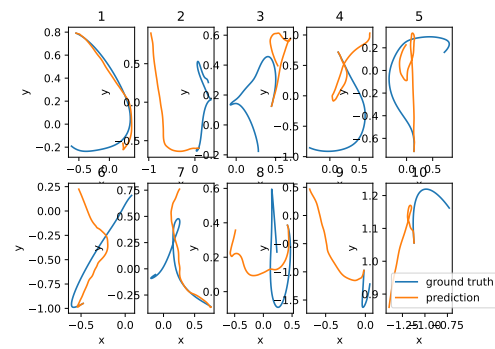
(a) GRUGHNN trained on 9dof sample and made 9dof trajectory



(b) GRUGHNN trained on 10dof sample and made 9dof trajectory



(c) GRUGHNN trained on 9dof sample and made 10dof trajectory



(d) GRUGHNN trained on 10dof sample and made 10dof trajectory

Figure 5.17: Trajectories of nbody after 1000 epochs

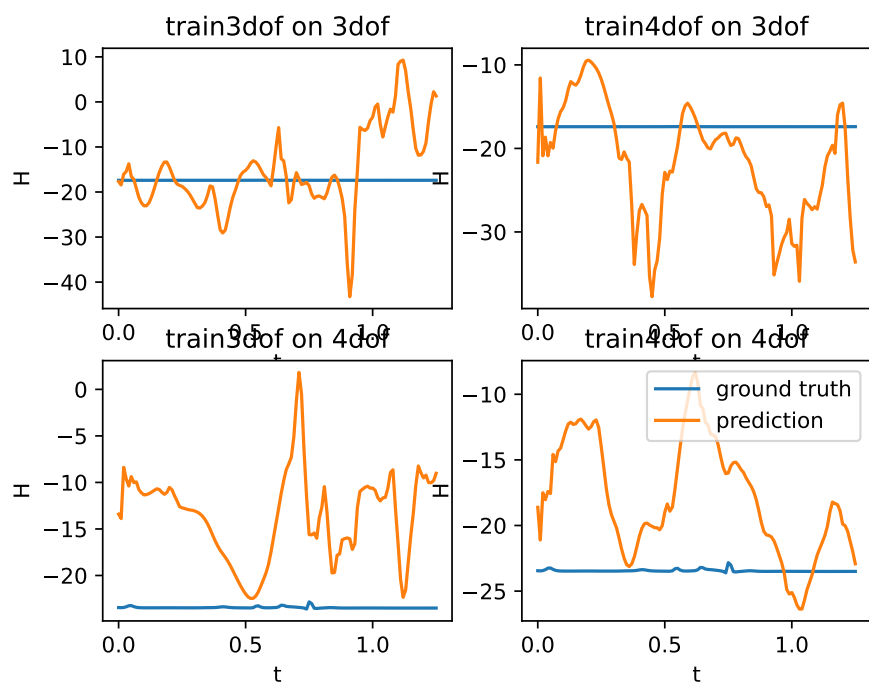


Figure 5.18: Hamiltonian of nbody cases

6 Conclusions

Physics-informed neural networks showed great capability in learning hamiltonian systems directly from data, though the graph structure of GNNs presented a really interesting alternative to classic multilayer perceptrons. The main challenges for those models remain being computationally demanding and requiring huge, high-quality datasets. Most of the experiments we showed in this thesis would never have been possible with a normal-end PC, hence underlining how access to high-performance computing resources played a an important role, for instance, using the Lichtenberg Cluster and high-end Laptop.

In this study, we successfully generated datasets from the hamiltonian equations and subsequently demonstrated that most of state of the art models can learn from the time-series data. This provides further evidence of the data-driven models can encapsulate complicated physical systems.

Through the experiments we demonstrated the performance of physics-informed models and tested our own models on our datasets. Although the results weren't 100 % accurate enough due to hardware issues we managed to obtain promising outcomes.

In addition, we envision future studies on the Kolmogorov-Arnold Networks, since this is a newly proposed architecture that blends graph structures with spline functions that have a learned shape as a parameter to model natural data. Preliminary studies suggest KAN can learn physics-based systems and partial differential equations, which would represent a potentially strong alternative to traditional MLPs and GNNs[27]. In essence, KAN may prove to be that key leap forward that the domain of physical machine learning is pushing for.

Bibliography

- [1] Jürgen Adamy. *Systemdynamik und Regelungstechnik II*. Shaker Verlag, 2010, pp. 135–144.
- [2] Ravi Balasubramanian. “The Denavit Hartenberg Convention”. In: (Jan. 2011).
- [3] A D Bermúdez Manjarres. “Operational classical mechanics: holonomic systems”. In: *Journal of Physics A: Mathematical and Theoretical* 55.40 (Sept. 2022), p. 405201. ISSN: 1751-8121. DOI: 10.1088/1751-8121/ac8f75. URL: <http://dx.doi.org/10.1088/1751-8121/ac8f75>.
- [4] Joan Bruna et al. *Spectral Networks and Locally Connected Networks on Graphs*. 2014. arXiv: 1312.6203 [cs.LG]. URL: <https://arxiv.org/abs/1312.6203>.
- [5] M. Calvo, J.I. Montijano, and L. Randez. “A fifth-order interpolant for the Dormand and Prince Runge-Kutta method”. In: *Journal of Computational and Applied Mathematics* 29.1 (1990), pp. 91–100. ISSN: 0377-0427. DOI: [https://doi.org/10.1016/0377-0427\(90\)90198-9](https://doi.org/10.1016/0377-0427(90)90198-9). URL: <https://www.sciencedirect.com/science/article/pii/0377042790901989>.
- [6] Kit Yan Chan et al. “Deep neural networks in the cloud: Review, applications, challenges and research directions”. In: *Neurocomputing* 545 (2023), p. 126327. ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2023.126327>. URL: <https://www.sciencedirect.com/science/article/pii/S0925231223004502>.
- [7] Suresh Chandra, Mohit Kumar Sharma, and Monika Sharma. “Harmonic Oscillator”. In: *Fundamentals of Mechanics*. Foundation Books, 2014, pp. 150–174.
- [8] Ricky T. Q. Chen. *torchdiffeq*. 2018. URL: <https://github.com/rtqichen/torchdiffeq>.
- [9] Ricky T. Q. Chen et al. *Neural Ordinary Differential Equations*. 2019. arXiv: 1806.07366 [cs.LG]. URL: <https://arxiv.org/abs/1806.07366>.

-
-
- [10] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2016–2019.
- [11] Salvatore Cuomo et al. “Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What’s Next”. In: *Journal of Scientific Computing* 92.3 (July 2022), p. 88. ISSN: 1573-7691. DOI: 10.1007/s10915-022-01939-z. URL: <https://doi.org/10.1007/s10915-022-01939-z>.
- [12] Susmita Das et al. “Recurrent Neural Networks (RNNs): Architectures, Training Tricks, and Introduction to Influential Research”. In: *Machine Learning for Brain Disorders*. Ed. by Olivier Colliot. New York, NY: Springer US, 2023, pp. 117–138. ISBN: 978-1-0716-3195-9. DOI: 10.1007/978-1-0716-3195-9_4. URL: https://doi.org/10.1007/978-1-0716-3195-9_4.
- [13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG]. URL: <https://arxiv.org/abs/1606.09375>.
- [14] Rahul Dey and Fathi M. Salem. “Gate-variants of Gated Recurrent Unit (GRU) neural networks”. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2017), pp. 1597–1600. URL: <https://api.semanticscholar.org/CorpusID:8492900>.
- [15] F. Donnangelo et al. “Predicting loss in optical transport segments: A GNN-GRU approach for a nationwide optical network”. In: *2023 23rd International Conference on Transparent Optical Networks (ICTON)*. 2023, pp. 1–4. DOI: 10.1109/ICTON59386.2023.10207281.
- [16] Weijiang Feng et al. “Audio visual speech recognition with multimodal recurrent neural networks”. In: May 2017, pp. 681–688. DOI: 10.1109/IJCNN.2017.7965918.
- [17] Mattias Flygare. “Holonomic versus nonholonomic constraints”. In: 2012. URL: <https://api.semanticscholar.org/CorpusID:118677177>.
- [18] Steven L. Garrett. *The Simple Harmonic Oscillator*. Cham: Springer International Publishing, 2020, pp. 59–131. ISBN: 978-3-030-44787-8. DOI: 10.1007/978-3-030-44787-8_2. URL: https://doi.org/10.1007/978-3-030-44787-8_2.
- [19] Sam Greydanus, Misko Dzamba, and Jason Yosinski. *Hamiltonian Neural Networks*. 2019. arXiv: 1906.01563 [cs.NE]. URL: <https://arxiv.org/abs/1906.01563>.

-
- [20] Ana Hudomal. “New periodic solutions to the three-body problem and gravitational waves”. In: *Master of Science Thesis at the Faculty of Physics, Belgrade University* (2015).
- [21] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG]. URL: <https://arxiv.org/abs/1609.02907>.
- [22] Visak Kumar et al. *Joint Space Control via Deep Reinforcement Learning*. 2021. arXiv: 2011.06332 [cs.R0]. URL: <https://arxiv.org/abs/2011.06332>.
- [23] Cornelius Lanczos. *CHAPTER IV. D’ALEMBERT’S PRINCIPLE. The Variational Principles of Mechanics*. Toronto: University of Toronto Press, 1949, pp. 88–110. URL: <https://doi.org/10.3138/9781487583057-007>.
- [24] Jens Lang. *Vorlesungskript Numerik Gewöhnlicher Differentialgleichungen*. 2021, pp. 11–15.
- [25] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG]. URL: <https://arxiv.org/abs/1511.05493>.
- [26] Wing Kam Liu, Shaofan Li, and Harold Park. *Eighty Years of the Finite Element Method: Birth, Evolution, and Future*. 2021. arXiv: 2107.04960 [math.NA]. URL: <https://arxiv.org/abs/2107.04960>.
- [27] Ziming Liu et al. *KAN: Kolmogorov-Arnold Networks*. 2024. arXiv: 2404.19756 [cs.LG]. URL: <https://arxiv.org/abs/2404.19756>.
- [28] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG]. URL: <https://arxiv.org/abs/1711.05101>.
- [29] Michael Lutter, Kim Listmann, and Jan Peters. *Deep Lagrangian Networks for end-to-end learning of energy-based control for under-actuated systems*. 2019. arXiv: 1907.04489 [cs.R0]. URL: <https://arxiv.org/abs/1907.04489>.
- [30] Yao Ma and Jiliang Tang. “Foundations of Deep Learning”. In: *Deep Learning on Graphs*. Cambridge University Press, 2021, pp. 43–72.
- [31] Joseph D. Romano Matthew J. Benacquista. *Classical Mechanics*. 2018. ISBN: 978-3-319-68779-7.
- [32] Philip Mocz. *Vectorized N-body code (Python)*. 2020. URL: <https://github.com/pmocz/nbody-python>.
- [33] Saeed Mohsen. “Recognition of human activity using GRU deep learning algorithm”. In: *Multimedia Tools and Applications* 82 (May 2023). DOI: 10.1007/s11042-023-15571-y.

-
-
- [34] B. W. Montague. “Basic Hamiltonian mechanics”. In: *CERN Accelerator School: Course on Advanced Accelerator Physics (CAS)*. 1993, pp. 1–14.
- [35] Vinh Phu Nguyen, Stéphane Bordas, and Timon Rabczuk. “Isogeometric analysis: An overview and computer implementation aspects”. In: *Mathematics and Computers in Simulation* 117 (May 2012). doi: 10.1016/j.matcom.2015.05.008.
- [36] Adam Paszke et al. “Automatic Differentiation in PyTorch”. In: *NIPS 2017 Workshop on Autodiff*. Long Beach, California, USA, 2017. URL: <https://openreview.net/forum?id=BJJsrmfCZ>.
- [37] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [38] Institute of Physics Belgrade. *THREE-BODY GALLERY*. URL: <http://three-body.ipb.ac.rs/>.
- [39] Subhankar Ray and J. Shamanna. *Virtual Displacement in Lagrangian Dynamics*. 2004. arXiv: physics/0410123 [physics.ed-ph]. URL: <https://arxiv.org/abs/physics/0410123>.
- [40] Alvaro Sanchez-Gonzalez et al. *Hamiltonian Graph Networks with ODE Integrators*. 2019. arXiv: 1909.12790 [cs.LG]. URL: <https://arxiv.org/abs/1909.12790>.
- [41] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG]. URL: <https://arxiv.org/abs/1912.05911>.
- [42] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [43] Milovan Šuvakov and V. Dmitrašinović. “Three Classes of Newtonian Three-Body Planar Periodic Orbits”. In: *Physical Review Letters* 110.11 (Mar. 2013). ISSN: 1079-7114. doi: 10.1103/physrevlett.110.114301. URL: <http://dx.doi.org/10.1103/PhysRevLett.110.114301>.
- [44] Arjan van der Schaft. “Port-Hamiltonian systems: an introductory survey”. English. In: *International Congress of Mathematicians, 2006*. European Mathematical Society Publishing House (EMS Ph), 2006.

-
- [45] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML]. URL: <https://arxiv.org/abs/1710.10903>.
- [46] Minjie Wang et al. “Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks”. In: (Sept. 2019). arXiv: 1909.01315 [cs.LG].
- [47] Zonghan Wu et al. *Graph WaveNet for Deep Spatial-Temporal Graph Modeling*. 2019. arXiv: 1906.00121 [cs.LG]. URL: <https://arxiv.org/abs/1906.00121>.