Machine Learning of Conservation laws via Hamiltonian Neural Networks in Ideal Spring System

Shaoxuan Chen
UMass Amherst
710 N Pleasant St, Amherst, MA 01003
shaoxuanchen@umass.edu

Abstract

The topic of identification of conservation laws and of the potential integrability of a Hamiltonian dynamical system has been central to classical and quantum systems. In this project the idea of identifying and learning the conservation law of spring system through the Hamiltonian Neural Networks was explored. And the network was evaluated by the data on the meshgrid to check whether it can learn a paraboloid and the order of accuracy it can reach.

1. Introduction

The traditional neural networks like the multilayer perceptron(MLP) does not have physics priors. They learn the approximate physics knowledge directly from data which prevent them from learning the exact physical laws. [2]. For example, the traditional neural network might just learn an approximation of the conservation law and the forward simulation of the system might drifts over time to higher or lower energy states. The Hamiltonian Neural Networks(HNNs) solves this problem by using the Hamiltonian equation which relates the state of the system to the conserved energy. In this project, we did some exploration on using the HNNs to learn the conservation law of the ideal spring system based on ground truth gradients and the numerical calculated gradients.

2. Theory

Hamiltonian Mechanics. The HNNs was designed based on the equation called the Hamiltonian, which is a system can be used to describe the conserved quantity and the change of the system with respect to time. Sam *et al.* [2] proposed the HNNs that was inspired from the Hamiltonian mechanics to train models that learn and respect exact conservation laws in an unsupervised manner. If we focus on the spring system. We can first define the coordinates of the system **(q,p)**. which **q**=$(q_1, q_2, ..., q_N)$ represents the positions of the spring, **p**=$(p_1, p_2, ..., p_N)$ represents their momentum at the corresponding position.[2]. We define **S** be the time derivatives of the coordinates of the system. Chen *et al.* [1], as shown in Equation (1), proposed that using the Neural ODEs to calculated the trajectory of the system over time by integrating the differential equations of the system from an initial state.

$$(q_1, p_1) = (q_0, p_0) + \int_{t_0}^{t_1} S(q, p)\, dt \tag{1}$$

Then we can define a scalar function, $\mathcal{H}(q, p)$, called the Hamiltonian, which can also be treated as the total conserved energy of the system. As shown in Equation (2), the Hamiltonian satisfies the following conditions:

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}} \tag{2}$$

Hamiltonian Neural Networks. We can see from Equation (2) that moving the coordinates of **(q, p)** in the direction $\mathbf{S}_H = (\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}})$ will give us the time evolution of the system. [2]. If we define the neural network to be $\mathcal{H}_\theta$. Then combined with the loss function defined for the network as shown in Equation (3), we will have the basic framework for the HNNs.

$$\mathcal{L}_{HNN} = \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} - \frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} + \frac{\partial \mathbf{p}}{\partial t} \right\|_2 \tag{3}$$

As shown in Figure 1, if we were given some trajectories of the ideal mass-spring system, compared with the traditional Neural Networks which takes the input **(q, p)** and the ground truth gradients **(q*, p*)** and try to predict the gradient of the system $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$. The HNNs take the same input but instead predict the conserved energy $\mathcal{H}_\theta$. Then by taking the gradient of $\mathcal{H}_\theta$ with respect to the coordinates **(q, p)** we will have the predicted gradient of the system $(\hat{\mathbf{q}}, \hat{\mathbf{p}})$. Then we can do optimization based on Equation (3) and thus we have the basic framework of HNNs. The HNN outperforms the traditional neural networks such as the HNN conserves a quantity that closely resembles total energy $\mathcal{H}(q, p)$ and diverges more slowly or not at all.[2]

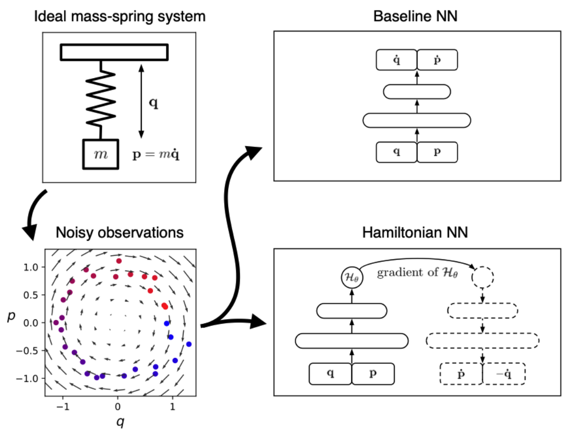


Figure 1. Framework of HNNs Applied in Ideal Spring System

3. Methods and the Setting of Neural Networks

In this project the idea of modeling the dynamics, identifying and learning the conservation law of the frictionless mass-spring system through the Hamiltonian Neural Networks was explored. Firstly the true Hamiltonian systems of spring physics models will be defined at very beginning to be used to sample the trajectory data **(q, p)**. Most of the data will be self-generated based on the physics models.

After hyper-parameter search of the learning rate, batch size, the number of layers and the corresponding neurons per layer, in all three tasks, we trained our models with a learning rate of 1e-3 and used the Adam optimizer[4]. Also different from the work by Sam *et al.* [2] that used the full size of data, we used the batch size data and set the batch size to be 64. We confirmed that the *tanh* activation function has the best performance for this task. All of our HNN models have six fully-connected layers and 400 hidden units per layer. They were given a vector input (q, p) and output the vector $(\frac{\partial \mathbf{q}}{\partial t}, \frac{\partial \mathbf{p}}{\partial t}) = (\hat{\mathbf{q}}, \hat{\mathbf{p}})$ using the derivative of a scalar quantity as shown in Equation (2). We used analytic time derivatives as the targets and trained the networks for 10000 gradient steps and evaluated them based on the relative error of the conserved energy and/or the analytic time derivatives.

The networks will also be evaluated by the data on the meshgrid to check whether it can learn a paraboloid and the order of accuracy it can reach. To be more specific, the final well-trained neural network will be used to predict the conservation energy $\hat{\mathcal{H}}(q, p)$ as well as the corresponding gradient, $(\hat{\mathbf{q}}, \hat{\mathbf{p}})$, of the data taken from the meshgrid. The results will be compared with the ground truth and evaluated by the relative error.

4. Experiments and Results

For now all of the experiments focus only on the frictionless mass-spring system. The system's Hamiltonian is given in Equation (4).

$$\mathcal{H} = \frac{1}{2}kq^2 + \frac{p^2}{2m} \tag{4}$$

where k is the spring constant and m is the mass constant. For simplicity, we set k = m = 2. Then we sampled initial coordinates with total energies uniformly. By using Equation (1) we can have the trajectory of the system over time based on some initial state. For example, if we have five initial states and choose the time interval Δt to be equally 0.1 second and the maximum radius of the system to be 100. By integrating out over time from 0 to 2. The final trajectories was shown in Figure 2.

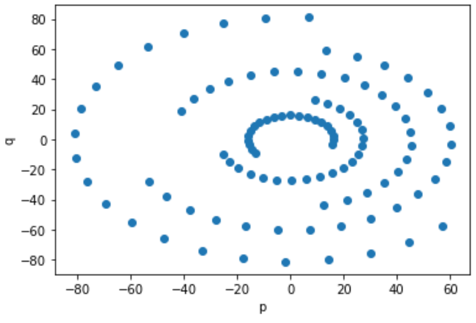


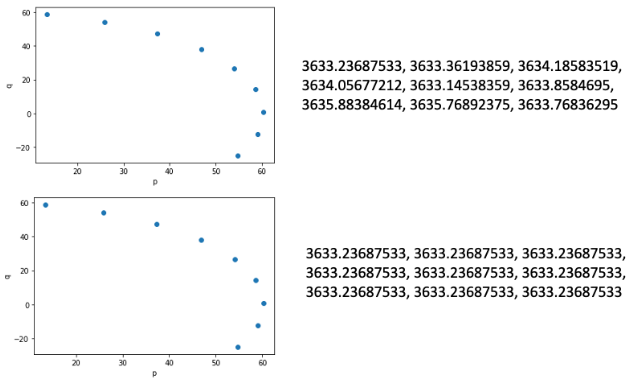
Figure 2. five sample trajectories of the spring system over time t ∈ [0,2] with $\Delta t = 0.1$

One thing needs to mention is that since our ultimate goal in this report is whether the network can successfully learn the paraboloid since the ground truth Hamiltonian function of the ideal spring system is Equation (4). So I will mainly show the experiment results on meshgrid data other than the training and testing accuracy archived by the network during optimization.

4.1. Evaluation on Integrator

The first improvement compared to the Sam *et al.* [2]'s work is that, instead of using Explicit Runge-Kutta method of order 5-'RK45', we used the Explicit Runge-Kutta method of order 8-'DOP853' [3]. It is a 7-th order interpolation polynomial accurate to 7-th order used for the dense output. And it can be applied in the complex domain. If we just have one initial state and use the integrator to integrate out the trajectory and calculate the corresponding conserved energy by Equation (4) with k = m = 2. The result of comparison is shown in Figure 3.

We can see that the trajectories integrated out by RK45 and DOP853 are almost the same. However, since the data points are all along one trajectory, theoretically their conserved energy should just be a constant because we assume the system is ideal without friction. The DOP853 integrator did a perfect job in this as the conserved energy $\mathcal{H}$ are all equal to the constant 3633.23687533. But the conserved



```
3633.23687533, 3633.36193859, 3634.18583519,
3634.05677212, 3633.14538359, 3633.8584695,
3635.88384614, 3635.76892375, 3633.76836295
```

```
3633.23687533, 3633.23687533, 3633.23687533,
3633.23687533, 3633.23687533, 3633.23687533,
3633.23687533, 3633.23687533, 3633.23687533
```

Figure 3. Sampled Trajectories (left) and Corresponding Conserved Energy (right) from Integrator 'RK45' (up), and 'DOP853' (bottom)

energy calculated based on RK45 is not a constant which means that the RK45 integrator is not numerically stable compared to DOP853. Thus, we choose the DOP853 integrator in the following experiments to satisfy our goal for the ideal setting.

4.2. Learning from Raw Data with Analytic Time Derivatives

4.2.1 Evaluation on Conserved Energy

We first explored whether the HNNs can learn well from the raw data generated by the ideal spring system. Sam *et al*. [2] uniformly sample the initial state data distributed between [-1, 1]. Here the initial position with larger radius, 100, was tried, which means the initial data was distributed between [-100, 100]. We constructed training and testing sets of 100 trajectories in total. Each trajectory had 50 observations; each observation was a concatenation of **(q, p)**. We used the L_2 loss, Equation (3), to do optimization of the HNN network.

After the training, for each pair of **(q, p)**, the network will output a predicted value of conservation energy $\hat{\mathcal{H}}$. Compared with the true conservation energy $\mathcal{H}$ calculated by Equation (4), we used the relative error between them to check the performance of the network. The results are visualized in Figure 4.

As shown in Figure 4, the x axis represents the errors along the trajectory. The y axis represents each samples. We can see that overall the errors are low. But for some parts in yellow the error is high. And it has a periodic pattern along the trajectory, which matches the physics pattern of the spring system.

4.2.2 Evaluation on Meshgrid Data

Then we also tried to sample data p, q from meshgrid [-100, 100] to explore whether the HNN model can learn a

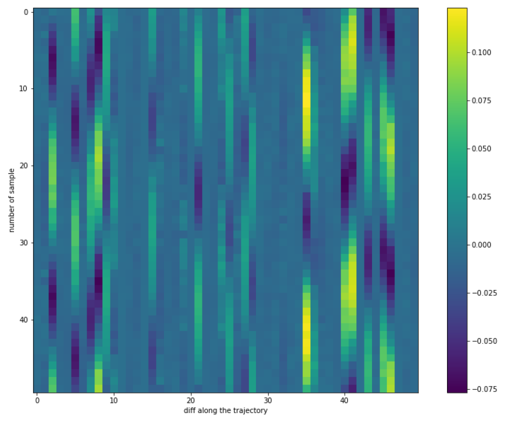


Figure 4. Relative error of spring system. 50 samples of trajectories, each trajectory has 50 observations.

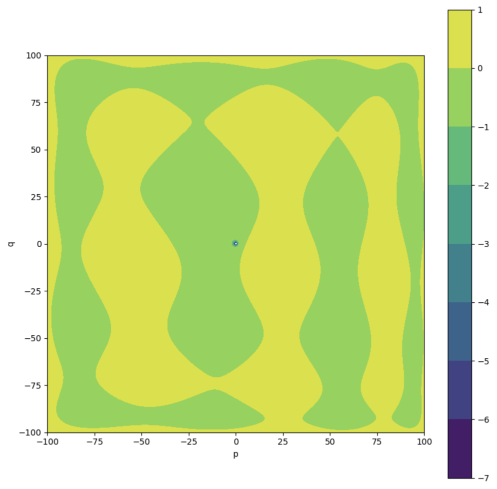


Figure 5. Relative error of spring system in meshgrid data, 2-D

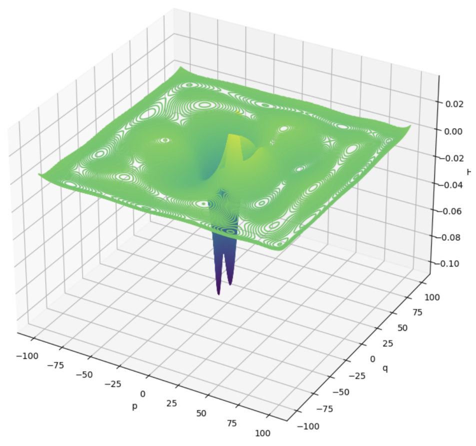


Figure 6. Relative error of spring system in meshgrid data, 3-D

paraboloid in Equation (4) with k = m = 2. The results in

Figure 5 and Figure 6 shows the relative error in 2-D and 3-D, respectively. We can see that the error is very large when p and q is close to the center (0,0), which suggests that we might need to avoid sampling near the center in the later experiments to avoid large error.

For now, the problem is that after the radius used to sample the initial data was increased, the relative error will increase a lot. We need to minimize the error such that no matter how large value of the radius we used, the absolute value of relative error between $\hat{\mathcal{H}}$ and $\mathcal{H}$ should be as small as possible, e.g. smaller than 1e-3.

4.3. Learning from Normalized Data with Analytic Time Derivatives

In this experiment we normalized the data into a ball within [-1,1] before feed the data in to the network. And we tried the learning rate scheduling technique after the first 10000 steps. Basically we reschedule the learning rate to be 1e-7 to fine tune the network for 2000 more steps. The training and testing data are shown in Figure 7.

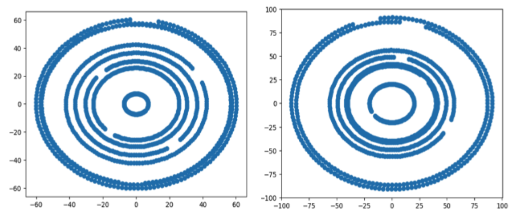


Figure 7. Training and Testing Data for Experiment 4.3 and 4.4.

Then we sample data p, q from meshgrid [-100, 100] to explore whether the HNN model can learn a paraboloid in Equation (4) with k = m = 2. We calculated the relative error of the analytic time derivatives, i.e., the ground truth gradient **(q*, p*)**, with the predicted gradient of the system $(\hat{\mathbf{q}}, \hat{\mathbf{p}})$. The results of different order of accuracy achieved by the network is shown in Figure 8. The white part in the figure means that the relative error of data in that region is larger than the tolerance. For example, the left figure in Figure 8 shows that if we set the tolerance range for the relative error to be [-1e-3, 1e-3], our network can achieve the predict accuracy on most part of the meshgrid data except than the data on the boundary and in the center. And if we set the tolerance range for the relative error to be [-5e-4, 5e-4], our network still has a decent performance. The majority of the data on the meshgrid can still be well-predicted by the network.

4.4. Learning from Normalized Data with Numerical Time Derivatives

In this experiment, compared to the work done by Sam *et al.* [2], we further improve the network to be capable of learning from the numerical time derivatives rather than the

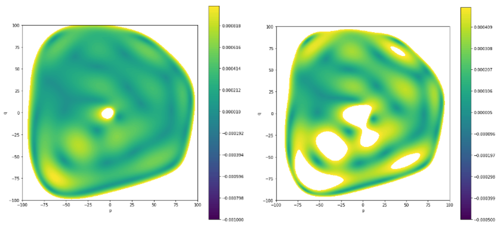


Figure 8. Left: Relative Error within [-1e-3, 1e-3]; Right: Relative Error within [-5e-4, 5e-4]

analytic time derivatives, which means that we do not need the ground truth Hamiltonian of the system. By simply observing the trajectories and use numerical scheme to calculate the time derivatives is enough to train the network. We also normalized the data into a ball within [-1,1] before feed the data in to the network. Instead of being fed with **(q*, p*)**, the network are fed by the numerical calculated gradients, $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$, based on the second order approximation of the first derivative shown in Equation (5).

$$f(x') = \frac{f(x + \Delta t) - f(x - \Delta t)}{2\Delta t} \tag{5}$$

We used the same data, and kept the same setting for training and evaluating the network as in 4.3. The final results of different order of accuracy achieved by the network is shown in Figure 9. It is reasonable that the network fed by the numerical calculated gradients will have lower performance than being fed by ground truth gradients. To be more specific, compared with the results in Figure 8, there is one more small white region in the bottom left of the figure for range [-1e-3, 1e-3]. And there are also relative larger white region in the right figure for range [-5e-4, 5e-4]. But overall, the network have good performance as most part of the meshgrid data are well predicted.

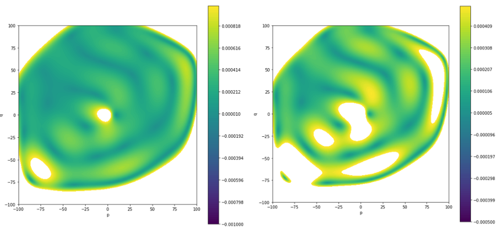


Figure 9. Left: Relative Error within [-1e-3, 1e-3]; Right: Relative Error within [-5e-4, 5e-4]

5. Discussion and Future Directions

In this project we used the the Hamiltonian Neural Networks to learn the conservation law of the ideal mass-spring system in different settings. We found that the DOP 853 integrator is better for integrating the trajectories. And we reached our temporary goal that the HNN can be well trained such that the majority of the meshgrid data's time derivatives, **(q*, p*)**, can be predicted within the accuracy of [-1e-3, 1e-3] with respect to the relative error. Also one improvement we made is that our network does not need to be learn from the ground truth Hamiltonian function, instead the trajectories of the system is enough for training the network.

As for the future work, one thing we can explore is whether the HNN can still performs well if we have larger radius of the trajectories. Since we only set the radius to be 100 for now. Later we want to increase the radius to be even larger, e.g. 10000. We will also extend the work to more complex physics system like ideal pendulum, real pendulum and two-body problem later.

References

[1] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018. 1

[2] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *NeurIPS*, page 15379–15389, 2019. 1, 2, 3, 4

[3] Ernst Hairer, Syvert Norsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 1993. 2

[4] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014. 2