Numerical Methods for Learning Physical Constraints and their Applications

Zhongrui Chen
Department of Computer Science
University of North Carolina at Chapel Hill

February 2022

Abstract

This thesis treats the theory and practice of *learning physical constraints*, and their use in interpolating and extrapolating motions. With the help of the evolving machine learning techniques, programs are capable of learning the physical properties and simulate future motions. To understand complex physical motions or make computationally expensive simulations much faster, there's a pressing need to develop a model that can simplify this task.

The contributions of this thesis begin with a theory that fits general physical motions into mass-spring models, followed by a neural particle simulator that leverages numerical methods that is end-to-end differentiable and trainable. Using this simulator, we can fit coordinates into this model to infer latent representation of collision radius, mass, spring constants in order to interpolate or extrapolate the motion. In the experiment section, we show that our method improves mean square error in chaotic motions by 100x compared to bilateral interpolation and the state-of-the-art and achieves similar performance in general physics motions.

Approved By:
Thesis Advisor: Praneeth Chakravarthula
Second Reader: Henry Fuchs

Contents

1 Introduction

1.1 Motivations

To help machines learn physical motions of objects, we first need to understand how human beings process visual information to ration physical motions of objects. For example, how can you tell that a moving object is a cloth, a ball, or a feather? How can you tell that you see a pendulum, a spring or a falling leaf merely from observing its movement?

We know that a cloth is not tangentially extensible and can not resist bending force in the normal direction. In this case, it's similar to a spring with a large spring constant. It's hard to either compress or pull the string, but it's easy to pull it in the normal direction.

We can do force analysis on the objects and find that most structural features can be represented by springs and therefore can be approximated by a deep neural network that embeds all the information about spring constants and resting lengths. All we need is to input our current state, and the deep learning black box will compute the required physical constants (i.e. acceleration and/or velocity) we need for estimating the next step.

On the other hand, if we know all physical priors, it's easy to simulate the future movements using a physical simulator – we can just step through it given all the constraints and properties of the objects.

However, physics priors are not always available. If we have a time series that sparsely samples 30 sets of coordinates per second of a physical motion, we can try to guess physics priors with some reasonable assumptions so that we can simulate the motion in any given granularity.

We can first start with some assumptions. For example, we can assume that there is a known number of particles in the system and the particles are either released from a steady state (see figure 1) or excited by an impulse (see figure 2). Then, we use a deep neural network to help with time integration and estimate physics parameters throughout the process (see figure 4).

1.2 Applications

With the mass-spring model theory and deep learning for time integration, we can apply them on sports analysis and beyond. In this work, we focus primarily on interpolating and extrapolating the coordinates of particles in simulations and videos.

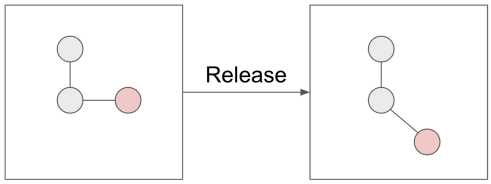

Figure 1: Three balls connected by strings released from a steady state.

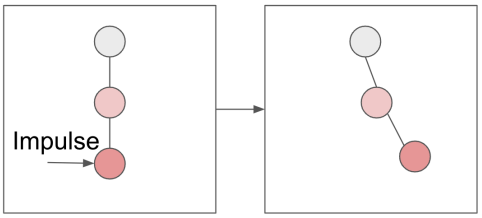

Figure 2: Three balls connected by strings excited by an impulse.

Imagine in pool games, the judge wants to see if one ball hits the other and it's a close call so eyes can't tell. Or, in modern badminton games and we want to know if the shuttlecock flew out of bounds. Even with the help of the recordings, we may not be able to tell scores and fouls apart at all times. With the help of our model, we can analyze whether two balls collide, reconstruct slow-motion trajectories and even predict several seconds into the future.

While a modern smartphone can easily record 240-fps videos, professional DSLRs are still required for capturing higher frame rates or better image qualities. What's more, recording higher frame rate videos poses a trade-off, sacrificing image qualities because of the higher shutter speeds. There are existing method that can interpolate intermediate frame, but they are mostly not physically-unaware. Existing methods estimate flows [15] or depth information [2] to get rid of artifacts in interpolated frames, but we can show that they fail in simple physical motions. In real world applications, physical correctness matters the most in sports videos, where inaccurate interpolation or prediction may result in unfair judging or visually unsatisfactory videos.

Therefore, it is of our great interest to generate physically-accurate high-quality slow-motion trajectories, which, combined with existing frame interpolation methods, can generate or notate high-quality slow-motion videos

from existing ones with lower frame rate. Then, without the help of professional devices, we can show our audience accurate trajectories and help sports analysis. Combined with the existing interpolation methods, it's possible to reconstruct intermediate frames that preserves physical property of the objects.

There has been a vast literature devoted into examining machine perceptions on general physics. For example, a human being can tell the motion of rigid bodies and springs because the latter is extensible. Can a machine learn and generalize such motions? With the development of Deep Neural Networks (DNN) and Generative Adversarial Networks (GAN), we can embed physics simulator into the network, so that the machine can capture the essential parts of the motion with least amount of data and labels. Interaction Networks[18, 4, 10, 21] model pairwise object interactions while energy-based methods[22, 8] approximate energy terms and use optimization methods to pick the best coordinates minimizing the constraint energy.

Most of the aforementioned network cannot train on some scene and test on others – motions from different scenes can be vastly different, for one can have collision and friction but the other doesn't. Therefore, it is reasonable to train our network on the given scene to learn enough physical priors of the motion, and then use the trained network to predict the intermediate time steps for this scene only.

1.2.1 Objectives

Learning to predict physical phenomena poses many challenges and is computationally demanding, since sparse observations from real-world can be stochastic and chaotic. There have been several early attempts [22, 8, 14] to build DNNs that can learn physics properties (i.e. joints, lengths, angles, etc.) from limited sparse observations. There are also discussions on learning physical properties from videos and reconstructing the videos from graphical simulations. Our method focuses on interpolation and extrapolation of the trajectory in videos and is easily pluggable into the existing frameworks.

Our goal is to approximate the governing equations and the underlying constraints by perceptrons with the knowledge of numerical methods. Then, with the help of object detection framework and image translation network, we can integrate into the existing frame interpolation methods and make them physics-aware.

Through experiments, we demonstrate that our framework is physics-

aware by observing simulations and videos. In section 3, we show that how our model compares to vanilla (simply averaging two coordinates) and existing frame interpolation methods in terms of object coordinates.

Our main contribution is a method of making physical predictions with variable time steps in simulations and real-world videos by predicting object motion from sparse observations. To test our ideas, we experiment on double pendulum simulation, collision simulation and billiard tables. We use object detection networks to extract object coordinates from real world videos. We carefully pick the bird's eye view videos of the games so that we don't have to deal with 3D modeling of our perspectives.

1.3 Related Works

In this section, we focus our discussions on recent learning-based methods to predict physical motions as well as extract information from temporally coherent frames to reconstruct intermediate frames. We also discuss neural physics simulators that takes in coordinate observations and predicts trajectories and motions. In addition, we discuss the existing method that combine both fields, learning physics from videos.

Recently, various attempts have been made to learn physical motions given coordinate information. Yang et al.[22] leverages MLP to estimate constraint energy and uses gradient descent to find the best coordinates to describe the next step of motion. Hamilton Neural Networks[8] makes an assumption that the given physical motion can be described by a set of governing equations and thus can be refactored into hamiltonian equations.

Frame interpolation. Depth Aware Interpolation[2] uses CNNs to estimate depth information to improve occlusion. It can gather contextual information from neighboring pixels and combine hierarchical features from depth maps to improve the image quality of intermediate frames. There are classical video frame interpolation methods estimating flow maps using optical flow algorithms[6, 11] and input frame warping[1, 3]. Asymetrical Bilateral Motion Estimation[15] is a state-of-the-art frame interpolation model that performs consistently better than previous methods in terms of motion. It predicts symmetrical bilateral motion fields to interpolate an anchor frame and use the fields to warp the input frames to create the interpolation. However, physically correct motions still pose a great challenge.

Learning physics from coordinates. Simulating constraint physical systems with the help of deep learning has been attempted in computational physics and computer graphics over the past years. With limited and sparse observance, state-of-the-art models can enforce shape, contact, collision, and boundary conditions

Yang et al. [22] introduces a novel model that takes a set of coordinates to learn underlying constraints of the physical system. It approximate the constraint energy it defines and use gradient descent to fix interpolated coordinates to follow the constraints. However, it doesn't consider temporal information. In our double pendulum experiment, if it's only exposed to the current bilinear interpolation and apply the constraint network to fix the position according to the constraint, it may result in a wrong coordinate even if it preserves the constraint properties.

There are also networks that uncover and enforce the Hamiltonian energy to predict the trajectory of a given set of governing equations, e.g. [8, 5, 16, 17, 12]. Most of these algorithms assume separability, which is rare in real world cases.

Learning physics model from videos. Yuan et al. [23] uses nearest neighbor approach to reconstruct predictions but it can only generate videos similar to those in the dataset. Finn et al. [7] proposes a method that can learn pixel motion from unlabeled video data. It predicts the change in Lagragian view and takes in sequence data and feed into a language model to make future predictions.

Wu et al. [19] proposes a generative model that reconstruct using a graphics engine from observed physics representation. There are also other promising works [13] that combine adversarial network with feedforward architecture to interpolate or extrapolate frames.

2 Methods

2.1 Overview

Assuming that we have n particles in the system, where $\mathbf{x}_i, \mathbf{v}_i$ denote the coordinate and velocity of the i-th particle for $1 \leq i \leq n$. All other notations follow Table 1.

Notation	
$\mathbf{x}_i$	The coordinate of the i-th particle.
$\mathbf{v}_i$	The velocity of the i-th particle.
k_{ij}^s	The spring constant of the spring connecting particle i and j.
k_{ij}^d	The damping constant of the spring connecting particle i and j.
$\mathbf{f}_{ij}$	Total force exerted from the spring ij to particle j.
$\mathbf{f}_{ij}^s$	Spring force exerted from the spring ij to particle j.
$\mathbf{f}_{ij}^d$	Damping force exerted from the spring ij to particle j.
$\mathbf{n}_{ij}$	The unit vector from particle i to j.
l_{ij}	The active length of spring ij.
s_{ij}	The resting length of spring ij.
m_i	The mass of particle i.
$\mathbf{g}$	Gravitational acceleration.

Table 1: Notations

In this section, we first assume that all pairs of the particles are either connected by a spring or not. If a pair of particles is connected by spring, then damping forces and spring forces will be exerted to both of the particles involved.

In the case of collision or explosion, our model will assume that there is a string with very large spring constant at the point of impact so that the objects will not run into each other. The result should be a approximation to collision and will be discussed in the later section.

Following the assumption that our system can be represented by the mass-spring-damper model, we show that according to Universal Approximation Theorem[9], we can approximate the function that estimates the acceleration and velocity for this step by a multilayer perceptron (MLP).

Therefore, we start this section with a theory that generalize particle motions using the mass-spring-damper model in section 2.2. Then, we show that we can use time integration to simulation the motions of all particles in the system in section 2.3. In section 2.3.1, we'll also note that parts of the numerical method is dependent on the physical priors that we don't know, so we can wrap the unknown variables in a deep learning black box. Lastly, in section 2.3.3, we'll show that we can use this model to interpolate and extrapolate coordinates given the time series.

2.2 Mass-spring-damper model

For springs between particle i and particle j, we have spring constant $k_{ij} = k_{ji}$, damping constant $k_{ij}^d = k_{ji}^d$ and force exerted from the spring to particle i, $\mathbf{f}_{ij}$, where

$$\mathbf{f}_{ij} = \mathbf{f}_{ij}^s + \mathbf{f}_{ij}^d \tag{1}$$

where $\mathbf{f}_{ij}^s$ is the spring force and $\mathbf{f}_{ij}^d$ being the damping force. Then,

$$\mathbf{f}_{ij}^s = k_{ij}(l_{ij} - s_{ij})\mathbf{n}_{ij} \tag{2}$$

where l_{ij} is the Euclidean distance between particle i and j, $\mathbf{n}_{ij}$ is the unit vector from particle i to j, and s_{ij} being the resting length of the spring ij. Since $l_{ij} = |\mathbf{x}_j - \mathbf{x}_i|$ and $\mathbf{n}_{ij} = \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}$,

$$\mathbf{f}_{ij}^s = k_{ij}(|\mathbf{x}_j - \mathbf{x}_i| - s_{ij})\frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|}. \tag{3}$$

The damping force is determined by both the coordinates and the velocities:

$$\mathbf{f}_{ij}^d = k_{ij}^d(\mathbf{v}_{ij} \cdot \mathbf{n}_{ij})\mathbf{n}_{ij} \tag{4}$$

where $\mathbf{v}_{ij}$ is the relative velocity of particle i with respect to particle j. Plug in $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$ and $\mathbf{n}_{ij}$, we have

$$\mathbf{f}_{ij}^d = k_{ij}^d \left((\mathbf{v}_j - \mathbf{v}_i) \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|} \right) \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|} \tag{5}$$

Now that we have all the spring forces, we can do force analysis on every particle.

$$\mathbf{f}_i = m_i\mathbf{g} + \sum_j \left(\mathbf{f}_{ij}^d + \mathbf{f}_{ij}^s \right) \tag{6}$$

where m_i is the mass of particle i and $\mathbf{g}$ being the gravitational acceleration.

2.3 Learning Time Integration Function

Now that we know the formula about forces with respect to coordinates and velocities, we can use numerical method to estimate coordinates at future discretized time step following section 2.3.1 and section 2.3.2. This is done under the assumption of knowing everything about the system (note that

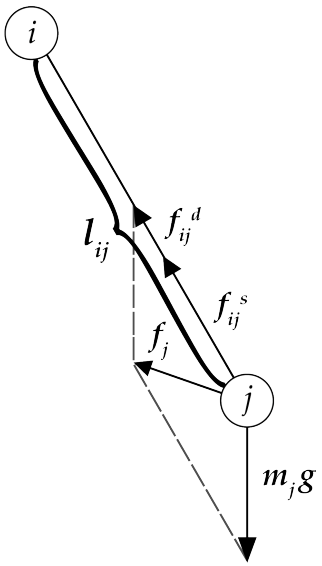

Figure 3: Force analysis on particle i, particle j, and spring ij

particle masses m_i and spring constants k_{ij} and s_{ij} are unknown to us) so that we know the exact formula. In our application where we only have sparse observation on the system, we can try to approximate acceleration with respect to coordinates and velocities using gradient descent.

According to Universal Approximation theorem[9], we can approximate any set of function using a multilayer feedforward network. Therefore, we can assume that we know how the function work, predict the interpolated coordinate using one half of a time step, and start from there and iterate once again. This should give us the coordinates of the next time step. Then, we can take the minimum square loss of the two sets of coordinates and use gradient descent to approximate the desired function with unknown constants. One other advantage of using a multilayer feedforward network to estimate the function is that physics models are not perfect and there are errors in numerical methods in integrations as well. This way, we can embed the fudge factor in the deep learning black box to make predictions more accurate and fit better with the sparse observations.

2.3.1 Euler method

Given the force analysis on each particle in figure 3, we can do time integration on all particles to estimate the state of the system after a small time step. First, we can get the acceleration of particle i from $\mathbf{f}_i$,

$$\mathbf{a}_i = \frac{\mathbf{f}_i}{m_i} \tag{7}$$

Following the acceleration, we can update the velocity of particle i

$$(\mathbf{v}_i)_{t+1} = (\mathbf{v}_i)_t + \Delta t(\mathbf{a}_i)_t. \tag{8}$$

where $(\mathbf{v}_i)_t$ and $(\mathbf{v}_i)_{t+1}$ defines the velocity of particle at discretized time step t and $t+1$, $(\mathbf{a}_i)_t$ defines the acceleration of particle i at time t.

Then we can predict the coordinates at next time step following

$$(\mathbf{x}_i)_{t+1} = (\mathbf{x}_i)_t + \Delta t(\mathbf{v}_i)_t. \tag{9}$$

Note that here, $\mathbf{v}_i$ is a variable dependent on $\mathbf{x}_i$ and $\mathbf{v}_i$, therefore we can rewrite the equation as

$$(\mathbf{x}_i)_{t+1} = (\mathbf{x}_i)_t + \Delta t \cdot net(\mathbf{x}(t), \mathbf{v}(t))_i \tag{10}$$

where $net(\mathbf{x}(t), \mathbf{v}(t))$ is a deep feedforward network that approximates the function $(\mathbf{v})_t$. Note that $\frac{d\mathbf{x}}{dt} = \mathbf{v}$, then we can rewrite the expression as $net(\mathbf{x}(t), t)$.

Thus, we can approximate coordinates coordinates at any time step following

$$(\mathbf{x}_i)_{t+1} = (\mathbf{x}_i)_t + \Delta t \cdot net(\mathbf{x}(t), t)_i \tag{11}$$

However, if we are interpolating the intermediate motion, using explicit Euler method would be equivlent to taking the average of two coordinates. Therefore, we combine learning scheme with numerical methods with higher convergence rate and with higher order terms estimation to predict the state at intermediate or prospective time steps.

2.3.2 Runge-Kutta method

Runge-Kutta method is a widely used numerical method for solving Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs).

By embedding explicit Runge-Kutta method into the network, we can estimate the coordinates at various time step with higher order of convergence. The Runge-Kutta method estimates four slopes k_1, k_2, k_3, k_4 and put greater weight for the slopes at the midpoint.

$$k_1 = net(\mathbf{x}(t), t) \tag{12}$$

$$k_2 = net(\mathbf{x}(t) + \frac{\Delta t}{2} k_1, t + 0.5) \tag{13}$$

$$k_3 = net(\mathbf{x}(t) + \frac{\Delta t}{2} k_2, t + 0.5) \tag{14}$$

$$k_4 = net(\mathbf{x}(t) + \Delta t k_3, t + 1) \tag{15}$$

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{16}$$

2.3.3 Neural Physics Network

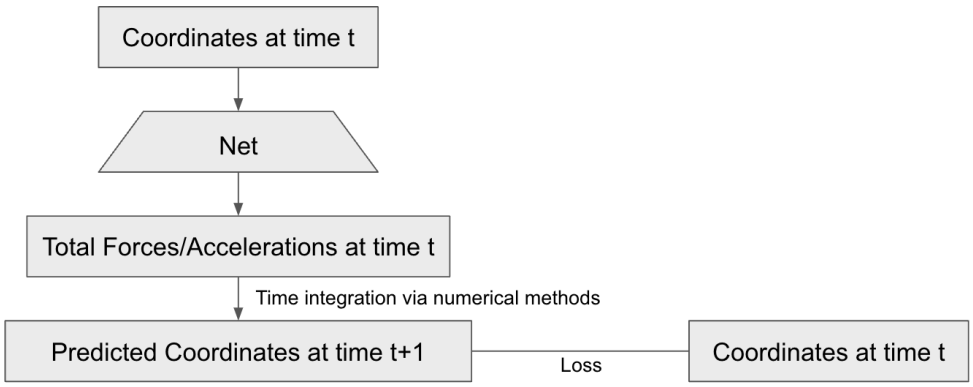

Figure 4: Network structure for neural physics simulator

For the physically-aware network, We approximate *net* function by neural network optimizers by training on the given coordinates to learn temporal and physical information from the observations. Then, we apply the learned network to predict interpolation or extrapolation. Figure 4 shows a full layout of the neural physics simulator network we are using.

Network Architecture We use a standard architecture to implement our Runge-Kutta net. For the interpolation network, we use a fully-connected network with $512 \times 512 \times 512$ elements. The output of *net* is the estimated velocity of the given coordinates $\mathbf{x}(t)$ at time step t. The numerical method then use the Runge-Kutta method to estimate the average velocity between time step t and time step $t + \Delta t$ and approximate the desired coordinates.

Interpolation Network For each interpolation task, we first train the network over a set of input coordinates, and then we'll be able to do the interpolation according to the approximated function. Here, we first estimate $net(\mathbf{x}(t), t)$ and predict the coordinates at time step t+0.5, $\tilde{\mathbf{x}}(t$+0.5) following

$$\tilde{\mathbf{x}}(t + 0.5) = \mathbf{x}(t) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{17}$$

where k_1, k_2, k_3, k_4 is described in section 2.3.2. Then we start from time step $t + 0.5$ and predict coordinates at time step $t + 1$ following

$$k_1 = net(\tilde{\mathbf{x}}(t + 0.5), t + 0.5) \tag{18}$$

$$\tilde{\mathbf{x}}(t + 1) = \tilde{\mathbf{x}}(t + 0.5) + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{19}$$

where k_2, k_3, k_4 is derived following section 2.3.2.

Then, the minimum squared loss is given by

$$\mathcal{L} = \frac{1}{2}(\tilde{\mathbf{x}}(t + 1) - \mathbf{x}(t + 1))^2 \tag{20}$$

Extrapolation Network For the extrapolation task, we test on the capability of generating future trajectories. Here, we use the same coordinate extraction method as above and feed the coordinates into the network to approximate $net(\mathbf{x}(t), t)$. For future coordinates approximation at time step $t + n + k$, suppose that we already extrapolated or have the ground truth of the coordinates at time step $t + n$

$$\mathbf{x}(t + n + k) = \mathbf{x}(t + n) + k \cdot net(\mathbf{x}(t), t). \tag{21}$$

3 Experiments

In this section, we try to validate our model in three scenes – double pendulum, collision of boxes and pool game. Then, we'll compare our model with Yang et. al[22]'s model *ConstraintNet*. We aim to show that learning physical constraints with temporal information is strictly better than without it.

In the double pendulum experiment, we connect three balls with two springs, pull them up so that three balls are horizontal and release them from this steady state (see figure 1 for the setup). We expect our model to be able to "learn" that there exist two strings between two pairs of balls. As a result, the model should be able to estimate the accelerations and velocity of all particles in this step so we can use numerical methods to integrate the coordinates to do the interpolations and extrapolations.

In the collision experiment, we set up the boundary and put 5×10 boxes and a ball moving horizontally towards the boxes. We sample all four corners of the boxes as well as the center of the ball to track the coordinates. As a result, the model should be able to learn the edges of the boxes as well as collision constraints between boxes and between boxes and boundaries.

In the pool game experiment, we use real-world video from YouTube to show that our model is capable of interpolating, extrapolating, and notating motions. We first use Detectron[20] to extract the coordinates of the balls in the game and then use our model to learn from the time series, eventually generate an interpolated and upsampled time series for notations in finer granularity.

3.1 Scene I: Double Pendulum

3.1.1 Dataset

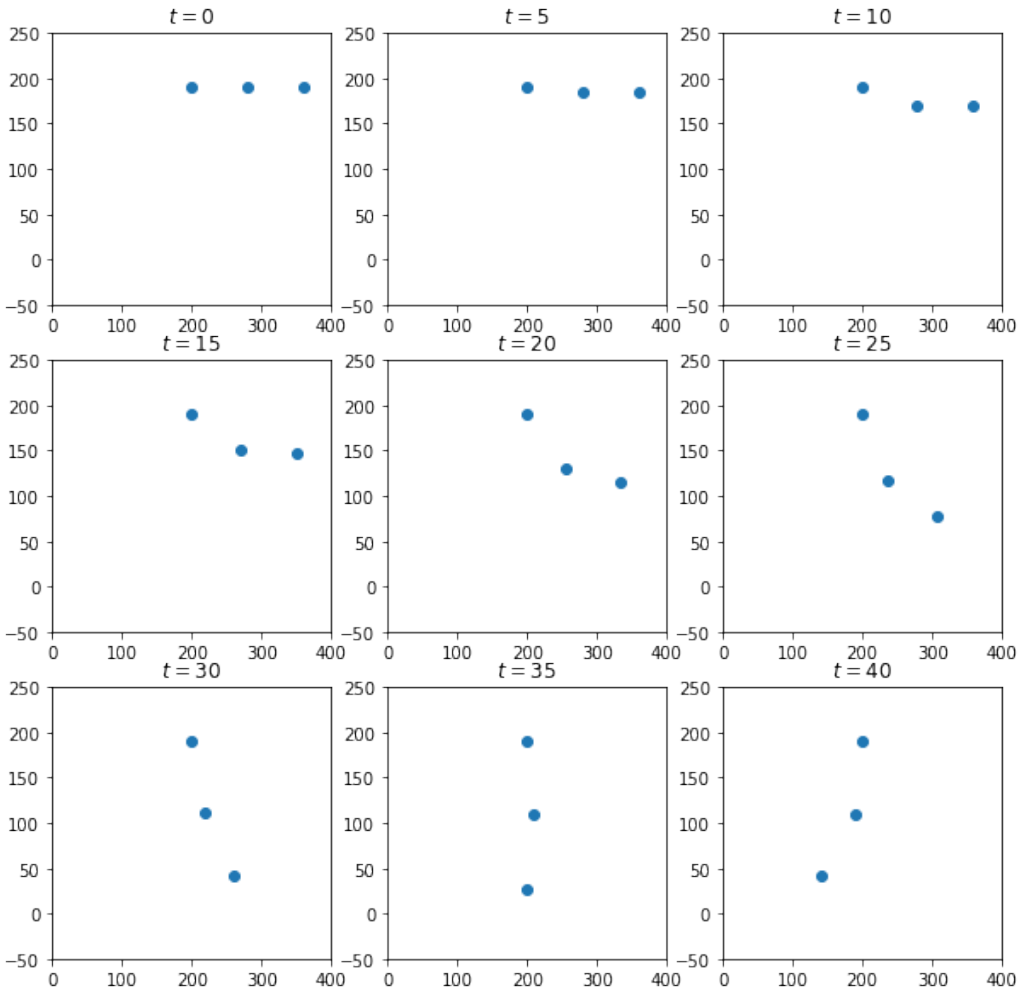

Figure 5: Time series data used for experiment 1

For the double pendulum scene, we first set up three balls connected by strings horizontally in 2D-space at $(200, 190)$, $(280, 190)$ and $(360, 190)$. Then, we release the balls from the steady state and they should form a double pendulum. We create this scene to validate our model because a single pendulum can be easily described by a linear equation thus does not pose significant challenge to the neural physics simulator. The motion of the double pendulum is chaotic. If we can interpolate the motion with a small error, we can say that our model is capable of physical-aware learning. Figure 5 visualizes the time series.

3.1.2 Training the Neural Physics Simulator

As stated in section 2.3.3, we are combining numerical methods with deep neural network to approximate the coordinates of the intermediate states.

To show that our model is capable of predicting physical motions, we first train our neural physics simulator with the double pendulum data set. We feed the model with coordinates of all particles at every other time step. Assuming that the model is capable of predicting the intermediate time step, we input the step at time t to get coordinates at time $t+1$ and then pass it back to the model again to get the predicted coordinates at time $t+2$. With the appropriate numerical methods, we hope that computing and the loss between the predicted coordinates and our labels and performing gradient descent to the network will give us a simulator that interpolates accurately.

3.1.3 Results

We test two models side-by-side with our model to show that our model can generalize chaotic physical motions better than traditional methods.

The first method we are comparing with is the vanilla interpolation, also called bilateral interpolation Given the coordinates at time t and $t+1$, $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$, respectively, the bilateral interpolation calculate the intermediate state following:

$$\mathbf{x}(t+0.5) = \frac{1}{2}(\mathbf{x}(t) + \mathbf{x}(t+1)). \tag{22}$$

Bilateral interpolation is used by intermediate frame synthesis models like ABME[15], so comparing our model with it shows how that performs as a plugin in use cases like video interpolation.

The second method we are comparing with is the neural projection network by Yang et al.[22]. The neural projection network estimates the potential energy of an object, therefore we know that how far off is the current state is from the object we are observing. For example, if we are observing a box and the four points we are sampling are not forming a box anymore, the potential energy will be large. Then it uses gradient descent to find the more accurate location for the four given points to form a box. After a set number of iterations, it should fix the shape of the object. For interpolation job, it first uses bilateral interpolation to get a relatively inaccurate result, and optimize potential energy from there, hoping to get a more accurate result.

By comparing to the two methods, we will show that our model is superior to them in several orders of magnitudes in terms of mean square error, and therefore generalizes chaotic motion like double pendulum better than the other two.

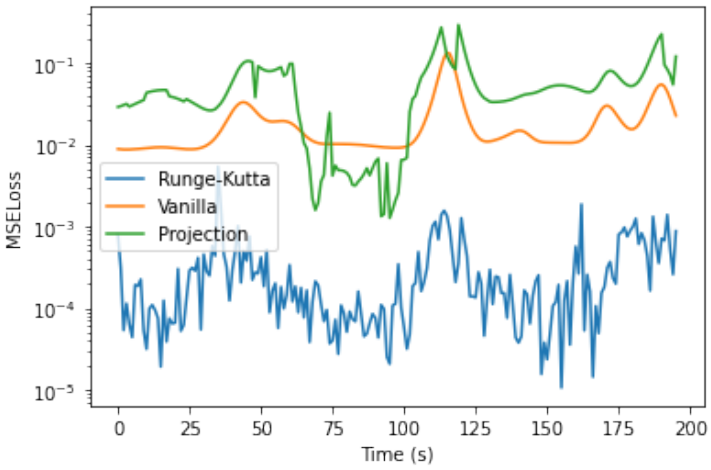

Figure 6: Mean Square Error of 3 models in the double pendulum scene.

Figure 6 shows us the MLP, together with Runge-Kutta numerical method achieve orders of magnitudes better in interpolation accuracy compared to the other two methods.

3.2 Scene II: Collision of Boxes

3.2.1 Dataset

Now that we have already proven our model is superior in predicting chaotic motions, we also want to show that it has comparable performance with other two methods in general cases. Therefore, we do a collision experiment, resembling the general scenario.

We set up a ball at $(0, 165)$, and 50 rectangular boxes to the right in a grid layout. Then, we set an impulse on the ball throwing it towards all the boxes. See figure 7 for a visualization of the physical motion.

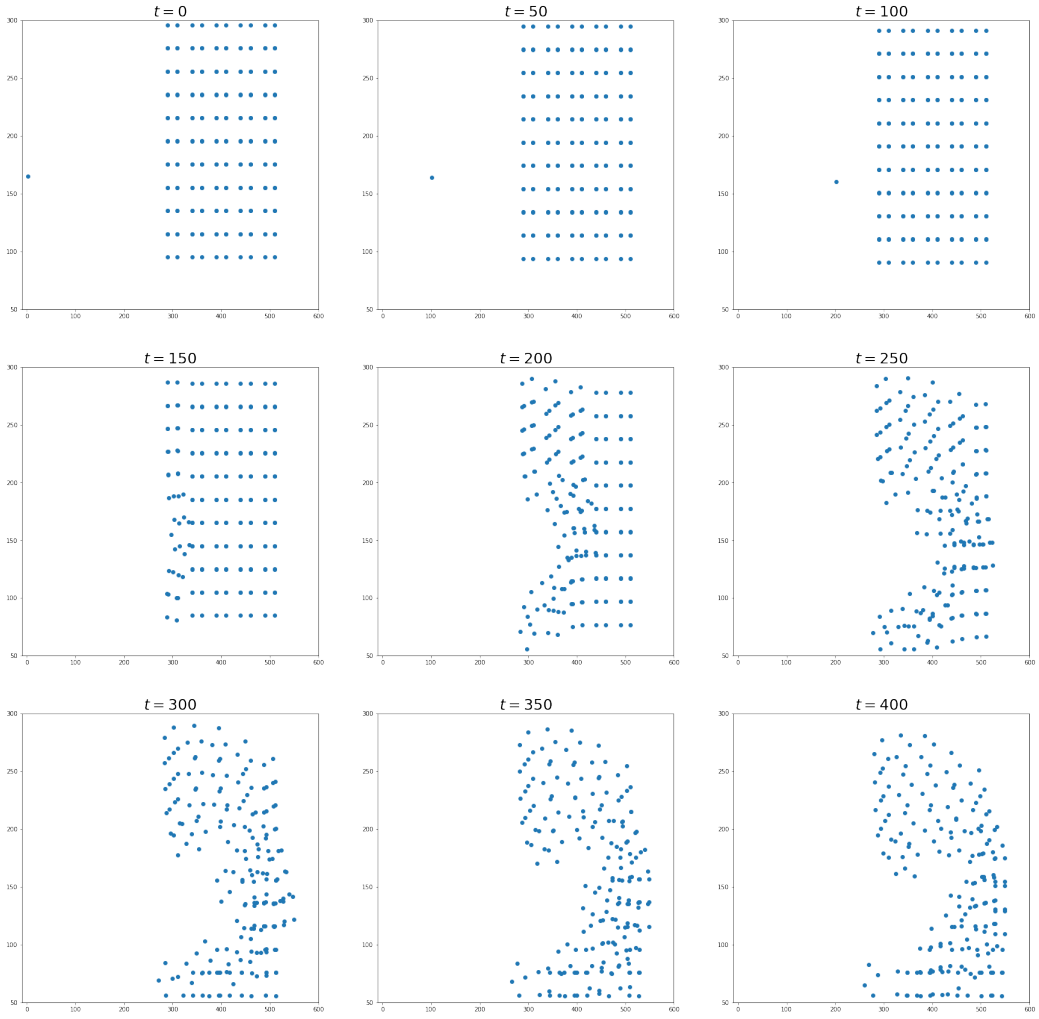

Figure 7: Time series data used for experiment 2

3.2.2 Results

From figure 8, we can see that our method has a comparable performance as the bilateral (vanilla) interpolation here. We don't have the results for neural projection network because their model need to have connectivity information and object information, which is unavailable to our model.

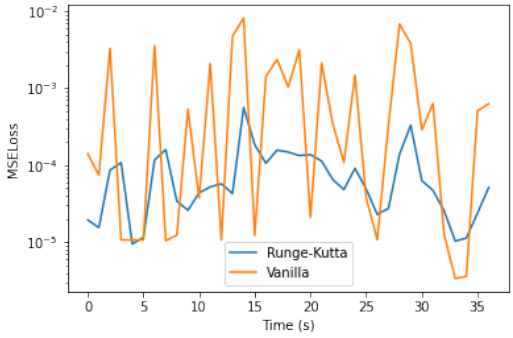

Figure 8: Mean Square Error for scene II

Now that we show that our method performs consistently better in chaotic motions and as good in general cases, we concluded that our model is physics-aware and can approximate coordinate interpolation.

4 Conclusions

We propose a novel physics-aware interpolation and extrapolation network that improves mean square error in chaotic motions by 100x and achieve similar performance in general cases. Our method consists of a MLP-based ODE solver and the numerical method to determine the intermediate or future coordinates.

4.1 Future Works

We theorize a physics model and leverage deep learning methods and numerical methods to help estimate intermediate motions, but there are several challenges we haven't addressed for it to work in video interpolations. Here, we assume that for video interpolations to work, we are using a object detection network to get the coordinate information, which is already known to

our method in this work, then calculate the intermediate coordinates. After we have the intermediate coordinates, we can feed the coordinates into a GAN to synthesize the intermediate frame. This is challenging because:

- Motion blurs make it hard to get an accurate estimate of coordinates.
- Label switching happens when similar objects collide.

In addition, we can also make our model more physically-aware by:

- Embed collision detection in the model and make collision radius learnable.
- Use more sophisticated numerical methods to further reduce the loss.
- Replace the bilateral interpolation in existing video interpolation models with our model to help improve physical-awareness in the model.

5 Acknowledgement

Many thanks towards Dr. Praneeth Chakravarthula, who was a caring and knowledgeable advisor to me during this project. I am also grateful to Dr. Henry Fuchs, who connected Praneeth with me in the first place and for being a second reader for the project.

References

[1] Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M. J., and Szeliski, R. A database and evaluation methodology for optical flow. *International journal of computer vision 92*, 1 (2011), 1–31.

[2] Bao, W., Lai, W.-S., Ma, C., Zhang, X., Gao, Z., and Yang, M.-H. Depth-aware video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), pp. 3703–3712.

[3] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. Performance of optical flow techniques. *International journal of computer vision 12*, 1 (1994), 43–77.

[4] Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. Interaction networks for learning about objects, relations and physics, 2016.

[5] Chen, Z., Zhang, J., Arjovsky, M., and Bottou, L. Symplectic recurrent neural networks. *arXiv preprint arXiv:1909.13334* (2019).

[6] Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., and Brox, T. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 2758–2766.

[7] Finn, C., Goodfellow, I., and Levine, S. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems* (2016), D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc.

[8] Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks, 2019.

[9] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[10] Hoshen, Y. Vain: Attentional multi-agent predictive modeling. *arXiv preprint arXiv:1706.06122* (2017).

[11] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., and Brox, T. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 2462–2470.

[12] Jin, P., Zhu, A., Karniadakis, G. E., and Tang, Y. Symplectic networks: intrinsic structure-preserving networks for identifying hamiltonian systems. *arXiv preprint arXiv:2001.03750* (2020).

[13] Lerer, A., Gross, S., and Fergus, R. Learning physical intuition of block towers by example. In *International conference on machine learning* (2016), PMLR, pp. 430–438.

[14] Nealen, A., Müller, M., Keiser, R., Boxerman, E., and Carlson, M. Physically based deformable models in computer graphics, 2005.

[15] Park, J., Lee, C., and Kim, C.-S. Asymmetric bilateral motion estimation for video frame interpolation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 14539–14548.

[16] Rezende, D. J., Racanière, S., Higgins, I., and Toth, P. Equivariant hamiltonian flows. *arXiv preprint arXiv:1909.13739* (2019).

[17] Toth, P., Rezende, D. J., Jaegle, A., Racanière, S., Botev, A., and Higgins, I. Hamiltonian generative networks. *arXiv preprint arXiv:1909.13789* (2019).

[18] Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems 30* (2017), 4539–4547.

[19] Wu, J., Lu, E., Kohli, P., Freeman, B., and Tenenbaum, J. Learning to see physics via visual de-animation. *Advances in Neural Information Processing Systems 30* (2017), 153–164.

[20] Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[21] Xu, Z., Liu, Z., Sun, C., Murphy, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Unsupervised discovery of parts, structure, and dynamics. *arXiv preprint arXiv:1903.05136* (2019).

[22] Yang, S., He, X., and Zhu, B. Learning physical constraints with neural projections, 2020.

[23] Yuen, J., and Torralba, A. A data-driven approach for event prediction. In *European Conference on Computer Vision* (2010), Springer, pp. 707–720.