Hamiltonian Neural Networks

Sam Greydanus
Google Brain
sgrey@google.com

Misko Dzumba
PetCube
mouse9911@gmail.com

Jason Yosinski
Uber AI Labs
yosinski@uber.com

Abstract

Even though neural networks enjoy widespread use, they still struggle to learn the basic laws of physics. How might we endow them with better inductive biases? In this paper, we draw inspiration from Hamiltonian mechanics to train models that learn and respect conservation laws in an unsupervised manner. We evaluate our models on problems where conservation of energy is important, including the chaotic three-body problem and pixel observations of a pendulum. Our model trains faster and generalizes better than a baseline network. An interesting side effect is that our model is perfectly reversible in time.

1 Introduction

Neural networks have a remarkable ability to learn and generalize from data. This has allowed them to excel at tasks such as image classification, drug discovery, reinforcement learning, and robotic dexterity. Even though these tasks are quite diverse, they all share the same underlying physical laws. It is of general interest, then, to endow these models with better physics priors.

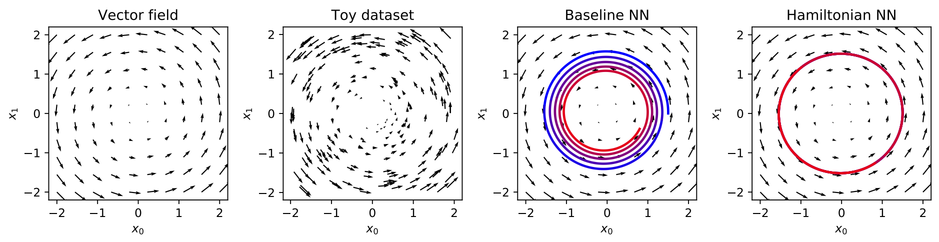


Figure 1: Learning the Hamiltonian of a mass-spring oscillator. *TODO: change diagram*

Untrained neural networks do not have physics priors; they learn approximate physics knowledge directly from data. This means they will never learn *exact* conservation laws. Consider the linear oscillator task shown in Figure 1. The oscillator obeys the law of conservation of energy, which means that for any point in time along a trajectory, the quantity $q^2 + p^2$ will be conserved. And yet, because the data is slightly noisy, the baseline neural network in Figure 1c will never learn this conservation law. How can we do better?

In this paper, we draw inspiration from Hamiltonian mechanics, which is a branch of physics used to describe the dynamics of a system in terms of its conserved quantities. Physicists usually write down an analytical expression for the Hamiltonian, but this can be difficult for complex, real-world systems. Here we take a different approach: we parameterize $\mathcal{H}_\theta$ with a neural network and learn it directly from real and synthetic data. Figure 1d shows the outcome of training a Hamiltonian Neural Network (HNN) on the linear oscillator dataset; it learns to exactly conserve a quantity that is close to $q^2 + p^2$.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2 Theory

Classical Hamiltonian Mechanics. William Hamilton introduced Hamiltonian mechanics in the 19th century as a mathematical reformulation of classical mechanics. Its original purpose was to express classical mechanics in a more unified and general manner. Over time, though, it has been applied to nearly every area of physics from statistical mechanics to quantum mechanics.

In Hamiltonian mechanics, we begin with a set of coordinates $(\mathbf{q}, \mathbf{p})$ that completely describe the state of a physical system. Then, we define a scalar function, $\mathcal{H}(\mathbf{q}, \mathbf{p})$ called the Hamiltonian such that

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}, \quad \frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}. \tag{1}$$

This equation tells us that moving the coordinates in the direction $\mathcal{S} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\right)$ gives us the time evolution of the system. It's important to note that $\mathcal{S}$ is a vector field over the inputs to $\mathcal{H}$. In fact, Differential geometry tells us that it is a special kind of vector field called the *symplectic gradient*. Whereas moving in the direction of the gradient of $\mathcal{H}$ changes the output as quickly as possible, moving in the direction of the symplectic gradient *keeps the output exactly constant*.

In general, the output of $\mathcal{H}$ can be any quantity that is conserved over time. However, a common choice for $\mathcal{H}$ is the total energy of the system. The coordinates $(\mathbf{q}, \mathbf{p})$ are generally position and momentum. In order to obtain the dynamics of a system from the Hamiltonian, we simply differentiate it with respect to $\mathbf{q}$ and $\mathbf{p}$ and then integrate the coordinates in the direction of the symplectic gradient.

$$(\mathbf{q}_1, \mathbf{p}_1) = (\mathbf{q}_0, \mathbf{p}_0) + \int_{t_0}^{t_1} \mathcal{S}(\mathbf{q}, \mathbf{p}) \; dt \tag{2}$$

Hamiltonian mechanics, like Newtonian mechanics, can predict the motion of a single pendulum or a bouncing ball. However, its true strengths become evident in systems with many more degrees of freedom. Celestial mechanics, which are chaotic for more than two bodies, are a good example. Correlated many-body quantum systems are best understood through the lens of the Hamiltonian as well. Some thermodynamics and fluid simulations use it to model interactions between thousands or even millions of particles.

Neural ODEs. Consider a learning problem where our aim is to model the dynamics of a system where conservation laws are at play. Though the data may be noisy, the underlying dynamics can be determined from Equation 2. One way to model the system would be to approximate $\mathcal{S}$ directly. Given a time-series dataset, this would amount to learning a mapping from $(\mathbf{q}_t, \mathbf{p}_t)$ to $(\frac{\Delta \mathbf{q}_t}{\Delta t_0}, \frac{\Delta \mathbf{p}_t}{\Delta t_0})$, which is a finite-differences approximation of $\mathcal{S}$.

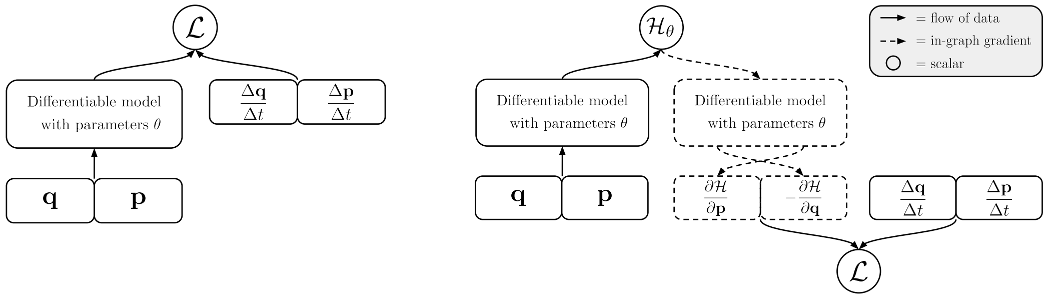
Figure 2(a) shows a schema of this model. Since it maps a set of coordinates to their time derivatives, we should think of it as a type of Neural ODE. Indeed, once we have trained this model we can solve Equation 2 numerically with an ODE solver as shown in Figure 1c.

Hamiltonian Neural Networks. The problem with the Neural ODE approach is that it ignores any conservation laws implicit in the data. In this paper, we propose learning a parametric function for $\mathcal{H}$ instead of $\mathcal{S}$. In doing so, we endow our model with the ability to learn *exactly* conserved quantities from data, in an unsupervised manner. We call these models Hamiltonian Neural Networks (HNNs) because they approximate the Hamiltonian function.

As with Neural ODEs, we train our models to predict the time derivatives of the inputs. Unlike Neural ODEs, we compute the time derivatives by computing the gradient of our model's output with respect to its inputs. Then, we directly optimize an L_2 loss over these gradients (Equation 3). This training procedure, outlined in Figure 2(b), allows HNNs to learn conserved quantities such as total energy from noisy, real-world data.

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

Apart from learning conservation laws, HNNs have several other interesting and potentially useful properties. For one, they are perfectly reversible. To see this, imagine flipping the signs of the partial derivative terms in Equation 1. Now, time "runs backwards" but the system's dynamics are exactly the same. A second observation is that we can add and remove energy from an HNN by integrating along the gradient of $\mathcal{H}$, giving us an interesting counterfactual tool (e.g. "What if the system had $1.5x$ its current energy?"). *TODO: perhaps remove this paragraph.*



$\mathcal{L}$
Differentiable model with parameters θ
$\frac{\Delta \mathbf{q}}{\Delta t}$
$\frac{\Delta \mathbf{p}}{\Delta t}$
$\mathbf{q}$
$\mathbf{p}$
$\mathcal{H}_\theta$
$\frac{\partial \mathcal{H}}{\partial \mathbf{p}}$
$-\frac{\partial \mathcal{H}}{\partial \mathbf{q}}$
= flow of data
= in-graph gradient
= scalar

(a) Baseline NN (b) Hamiltonian NN

Figure 2: HNN schema. The forward pass of an HNN is composed of a forward pass through a differentiable model as well as a backpropagation step through the model.

3 Learning a Hamiltonian from Data

In Related Work (Section 7) we review several cases where researchers optimized the gradients of neural networks. Even so, this technique is uncommon and not well-documented. Furthermore, previous results are substantially different in scope and implementation details. Based on these observations, our first step was to select three simple physics tasks and use them to evaluate both a) trainability and b) generalization in HNNs.

3.1 Methods

In all three tasks, we trained our models with gradient descent using the Adam optimizer with a learning rate of 10^{-3}. None of the datasets had over a thousand examples, so we set the batch size to be the size of the training set. On each dataset we trained a baseline model to approximate $\mathcal{S}$ and an HNN to approximate $\mathcal{H}$. All of these models were three-layer MLPs with 200 hidden units and `tanh` activations. We trained our models for 2000 gradient steps and evaluated them on a test set. See the Appendix for more details.

We evaluated all of our models on two metrics. The first of these was the test loss, which we used to measure how well each model fit the dataset. We called the second metric "energy MSE" and used it to measure how well our models generalized over time. We defined it as the mean square error between the total energy of a true trajectory and the total energy of a predicted trajectory. In order to generate a "predicted trajectory," we selected random initial coordinates from the test set and integrated our model according to Equation 2. In practice, this meant using fourth-order Runge-Kutta integration as in Chen et al. (2018). Since total energy is conserved in all of our toy tasks, the energy MSE metric was also a useful for measuring whether our models learned to conserve energy.

Task 1: Simulated linear oscillator. We already briefly discussed the first toy task in Section 1. The objective is to train an HNN to model the dynamics of a mass-spring system where both the spring and mass constants are set to one ($k = m = 1$). The Hamiltonian of this system is

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

We selected initial values of p and q from a normal distribution, generated 100 trajectories of 100 timesteps each, and added normal noise with $\sigma = 0.1$.

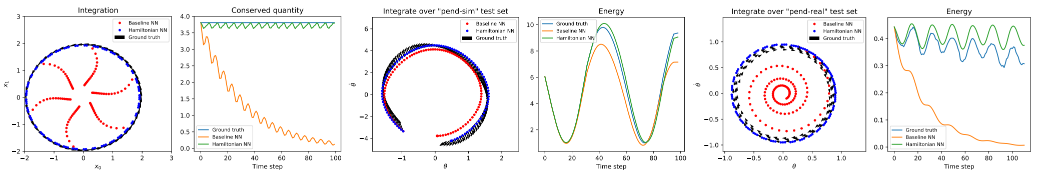
Task 2: Simulated pendulum. Our second toy task consisted of data from a simulated pendulum. Pendulums are not linear oscillators so they present a slightly more difficult problem. The Hamiltonian of a single pendulum with mass m, length l, and gravitational constant g is

$$\mathcal{H} = \frac{p^2}{2ml^2} + 2mgl \sin\frac{\theta}{2} \propto p^2 + \sin\frac{\theta}{2} \tag{5}$$

Instead of generating our own simulated pendulum dataset, we used one from a a related paper, Schmidt & Lipson (2009). Like our work, this paper attempted to infer conservation laws from raw data. We used their data with the hope of making clear comparisons.

Task 3: Real pendulum. Our third toy task was composed of position and momentum readings taken from a real pendulum. This data was significantly more noisy than the simulated datasets. Furthermore, the real pendulum had a small amount of friction, so it did not *strictly* obey a conservation law. We used this task to examine how robust HNNs are to noisy and biased data. This dataset was taken from the same supplementary materials as Task 2.

3.2 Results



(a) Task 1 (b) Task 2 (c) Task 3

Figure 3: On several toy tasks, the HNN learns to conserve the total energy of the system whereas the baseline model does not. TODO: In these plots the mass-to-spring constant ratio is not calibrated, thus the energy readings oscillate wildly. Fix this.

On the simple toy tasks, HNNs trained as quickly as the baseline networks and converged to test losses on the same order of magnitude. Table 1 shows a quick comparison of the three models and two metrics. We were surprised to see that the HNN converged to the lowest test loss on Task 1 and yet was substantially worse than the baseline on Task 2, even though the two tasks are quite similar. We think this difference could be because the dataset from Task 2 was collected at only a single energy level of the pendulum.

Table 1: *Toy tasks*

	Test loss		Energy MSE	
	Baseline	Hamiltonian	Baseline	Hamiltonian
Linear oscillator (sim)	1.1674	**0.7676**	4.3796	**4.7e-3**
Pendulum (sim)	**1.8e-3**	1.35e-2	0.8670	**0.120**
Pendulum (real)	**1.4e-3**	5.8e-3	7.83e-2	**2.3e-3**

Though the HNN and its baseline were closely matched on test loss, the HNN dramatically outperformed the baseline model on the energy MSE metric. Upon closer analysis, we found that over long time periods, the baseline model tended to slowly lose energy until the system reached the lowest-possible energy state. The HNN, meanwhile, learned a good proxy for total energy and conserved it indefinitely. Figure 4 gives a qualitative investigation of this behavior.

4 Modeling Larger Systems

TODO: get results on three-body problem and rename to the much sexier title: "Modeling Chaotic Systems"

Having found that HNNs train and generalize well on toy tasks, our next step was to train them on a more difficult task involving many pairs of (p, q) coordinates. A well-studied problem that fits this description is the two-body problem. In the two-body problem, two point particles interact with one another via an attractive force such as gravity.

When we set the masses and gravitational constant to one, we can write the Hamiltonian of this system according to Equation 6.

$$\mathcal{H} = \frac{|\mathbf{p_1} + \mathbf{p_2}|^2}{4} + |\mathbf{p_2} - \mathbf{p_1}|^2 + \frac{1}{|\mathbf{q_2} - \mathbf{q_1}|} \tag{6}$$

This system has eight degrees of freedom, given by the x and y position and momentum coordinates of the two bodies.

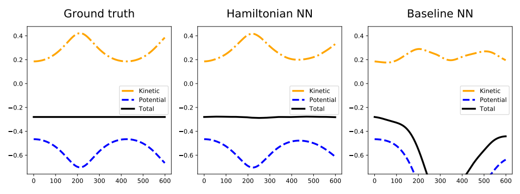
4.1 Methods

Our first step was to generate a dataset of 1000 near-circular two-body trajectories. We initialized all examples with center of mass zero, total momentum zero, and radius $|\mathbf{q_2} - \mathbf{q_1}|$ between 0.5 and 1.5. Even with these restrictions, we found that random initialization produced trajectories with kinetic and potential energies that spanned several orders of magnitude.

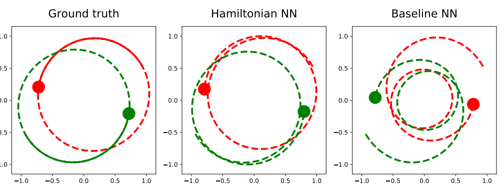
Upon closer examination, we found that this happened whenever the two bodies came in very close proximity of one another. In order to mitigate this issue, we solved for velocity vectors corresponding to perfectly circular orbitals and then scaled them by $1 + \mathcal{N}(0, \nu)$. Near-circular orbitals being the most numerically stable, we found that changing the parameter ν provided a convenient way to control the tradeoff between numerical stability and data diversity. We found that a setting of $\nu = 10^{-1}$ offered a good middle ground. After sampling an initial state, we solved for 50 evenly-spaced states in the range $t = [0, 20]$ using a fourth-order Runge-Kutta integrator. Several example trajectories can be found in the Appendix.

Our models and training procedure were identical to those described in Section 3 apart from a few minor changes. We switched to a batch size of 200 examples and trained our models for a total of $10,000$ gradient steps. Even the physics of this system are invariant to swapping the indices of the two bodies (relabeling body 1 as body 2 and vice versa), we did not build this symmetry into our model. The input to our model is simply a vector containing all 8 coordinates. There are more sophisticated approaches to training neural networks on permutation-invariant inputs, but we found them unnecessary for our purposes.

4.2 Results



(a) Energy vs. time



(b) Trajectories

Figure 4: On several toy tasks, the HNN learns to conserve the total energy of the system whereas the baseline model does not. TODO: Write a caption for these figures instead of reusing one from a previous figure.

Our results suggest that the HNN model scales well to this system. Figure 4(a) suggests that it learned to conserve a quantity nearly equal to the total energy of the system whereas the baseline model did not. Meanwhile, Figure 4(b) gives a qualitative comparison of the two trajectories. To our surprise, the HNN also trained more quickly than the baseline model and reached a far lower test loss (Table 2). The energy MSE metric was substantially lower than that of the baseline as well.

Improving the baseline. These findings made us worry that our baseline was not strong enough, so we trained an additional baseline model...TODO: actually try training a better baseline model.

We should also emphasize that even though the HNN was dramatically better than the baseline model on this dataset, there are others where this is not the case. For example, we also tried training our models on a dataset constructed using $\nu = 1$. Modeling orbits in this regime is a much more challenging task due to the numerical instability we discussed in Section 4.1. In fact, neither model was able to learn a reasonable solution. However, the HNN failed much more spectacularly than our baseline (see Appendix).

5 Learning a Hamiltonian from Pixels

One of the key advantages of neural networks is that they can learn abstract representations directly from high-dimensional data such as pixels or words. Having trained HNN models on position and momentum coordinates, we were eager to see whether we could train them on arbitrary coordinates, say, the latent vectors of an autoencoder.

With this in mind, we constructed a dataset of pixel observations of a pendulum and then used a fully differentiable model composed of an autoencoder and an HNN to learn its dynamics. This is, to our knowledge, the first instance of a Hamiltonian learned directly from pixel data.

5.1 Methods

In recent years, OpenAI Gym has been widely adopted by the machine learning community as a means for training and evaluating reinforcement learning agents. Some works have even trained neural networks to model the dynamics of Gym environments for use in model-based reinforcement learning. Seeing these efforts as related – and perhaps complimentary – to our work, we chose to use the `Pendulum-v0` environment in this experiment.

We used this dataset to build a dataset of 200 trajectories of 100 frames each. We only used trajectories where the maximum absolute displacement of the pendulum was less than $\frac{\pi}{6}$ radians. Starting from 400 x 400 x 3 RGB pixel observations, we cropped, desaturated, and downsampled them to 28 x 28 x 1 frames and concatenated each frame with its successor, so that the input to our model was a tensor of shape `batch` x 28 x 28 x 2. For more details on this transformation, refer to the Appendix.

In designing the autoencoder portion of the model, our main objective was simplicity and trainability. We chose to use fully-connected layers in lieu of convolutional layers because they are simpler. Furthermore, convolutional layers have been shown to struggle with extracting even simple position information from inputs Liu et al. (2018). Both the encoder and decoder were composed of four fully-connected layers with `relu` activations and residual connections. We used 200 hidden units on all layers except the latent vector $\mathbf{z}$, which we set to two units. We used the same HNN architecture and parameters as described in previous experiments.

Unless otherwise specified, we used the same training procedure as described in Section 4.1. As in the orbit experiment, we switched to a batch size of 200 examples and trained our model for $10,000$ gradient steps. We found that using a small amount of weight decay, 10^{-5} in this case, was beneficial. The most notable difference between this experiment and the others was the loss function. Our loss function was composed of three terms: the first being the HNN loss, the second being a classic autoencoder loss (L_2 loss over pixels), and the third being an auxiliary loss on the autoencoder's latent space.

Auxiliary loss. In spite of considerable effort (see Appendix), we were unable to avoid a third loss term. This term, which we will refer to as the "canonical coordinate loss," is defined in Equation 7. The purpose of this loss is to make the second half of $\mathbf{z}$, which we'll label $\mathbf{z_p}$, resemble the derivatives of the first half of $\mathbf{z}$, which we'll label $\mathbf{z_q}$. This loss encourages the latent vector $(\mathbf{z_q}, \mathbf{z_p})$ to have roughly same properties as canonical coordinates $(\mathbf{q}, \mathbf{p})$. These properties, measured by the Poisson bracket relations, are necessary for writing a Hamiltonian for a system.

$$\mathcal{L}_{CC} = \left\|\mathbf{z}_{\mathbf{p}}^{t} - (\mathbf{z}_{\mathbf{q}}^{t} - z_{\mathbf{q}}^{t+1})\right\|_2 \tag{7}$$

Although it would be better to avoid a canonical coordinate loss term, we found that it did not significantly degrade the autoencoder's performance. Furthermore, it is not domain-specific and can be applied to any autoencoder with an even-sized latent space.

5.2 Results

Our "PixelHNN" model trained and generalized well on the dataset. Unlike the baseline model, it learned to conserve a scalar quantity analogous to the total energy of the system. Figure 5 shows a

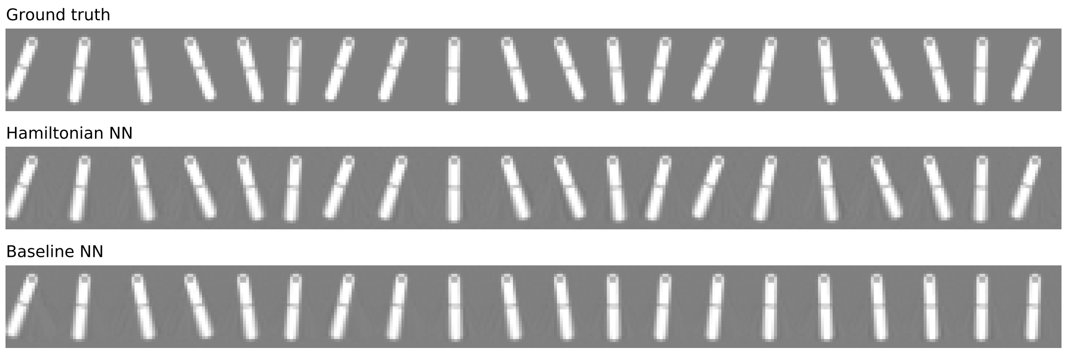


Figure 5: Predicting the dynamics of the pixel pendulum.

qualitative comparison of the dynamics predicted by the two models, starting from one of the test set trajectories. As in previous experiments, we computed these dynamics using Equation 2 and a fourth-order Runge-Kutta integrator. Unlike previous experiments, we performed this integration in the latent space of the autoencoder. Then, after integrating, we projected the trajectory into pixel space using the decoder.

The training dynamics of the PixelHNN were similar to those of its baseline. In fact, both ended with nearly the same test loss. Unlike the baseline model, though, the PixelHNN performed much better on the energy MSE metric[1] See Table 2 for details.

Table 2: *Harder tasks*

	Test loss		Energy MSE	
	Baseline	Hamiltonian	Baseline	Hamiltonian
Pixel Pendulum	**2.0e-4**	2.1e-4	3.9e-3	**4.8e-5**
Orbits	3.0e-5	**2.8e-6**	5.9e-2	**3.8e-5**

Scaling the latent dimension. In theory, we could train PixelHNN models with much larger latent dimensions. In fact, doing so is absolutely necessary for more complicated tasks. With this in mind, we tried training PixelHNN models with four-dimensional and six-dimensional latent spaces. Naive attempts to do this produced significantly worse results. While our model still learned to conserve a quality roughly equivalent to energy, integration in latent space soon led to strange artifacts.

6 Useful properties of HNNs

Hamiltonian mechanics is theory of remarkable breadth. Over the past two hundred years, many brilliant minds have contributed new theorems, techniques, and extensions. Simultaneously, the theory has expanded beyond classical mechanics into realms such as statistics, quantum mechanics, and fluid dynamics.

The main purpose of introducing HNNs was to endow neural networks with better physics priors. In this section, though, we will contemplate some of the other potential benefits of these models.

Adding and removing energy. So far, we have seen that integrating the symplectic gradient of the Hamiltonian can give us the time evolution of a system. We have not yet tried integrating the Riemann gradient $\mathcal{R} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \frac{\partial \mathcal{H}}{\partial \mathbf{q}}\right)$. Intuitively, this corresponds to adding or removing energy from the system.

It's interesting to alternate between integrating $\mathcal{R}$ and $\mathcal{S}$. Figure 6 shows how we can take advantage of this effect to "bump" the pendulum to a higher energy level. We could imagine using this technique to answer counterfactual questions such as "What would have happened if we had applied a torque?" See Supplementary Materials for a video of this effect.

[1]The energy MSE metric was not trivial to compute here. Since there is no analytical method for mapping from latent space trajectories to total energy, we had to *learn* an approximate mapping. However, that additional error is far smaller than the two orders of magnitude gap between the PixelHNN and its baseline. See Appendix.

Perfect reversibility. As neural networks have grown in size, the memory consumption of transient activations – the intermediate values needed to perform backpropagation – has become a notable bottleneck. Recent work by Gomez et al. (2017) proposes semi-reversible models that can construct one layer's activation from the activations of the next. In the same vein, Chen et al. (2018) uses clever application of the adjoint rule to train neural ODE with constant memory cost.

But while these models are more reversible than most, they are not perfectly reversible in the sense that their mappings are not perfectly bijective. Our approach, meanwhile is guaranteed to produce a perfectly reversible function. To see this, we can simply refer to a classic result from Hamiltonian mechanics called Liouville's Theorem: *The density of particles in phase space is constant*. In other words, $\mathcal{S}$ is a solenoidal, or mass-conserving, vector field. Every mapping defined by $\mathcal{S}$ is bijective.

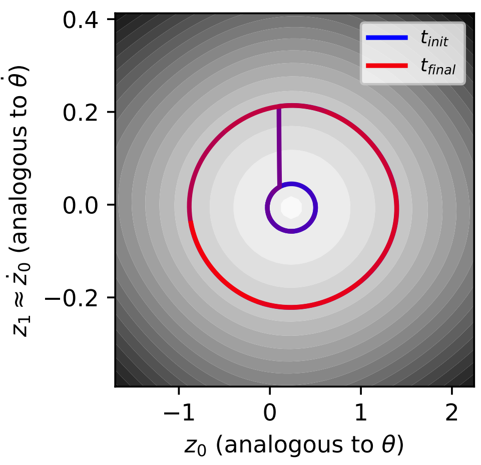


Figure 6: Visualizing integration in the latent space of the PixelHNN model. Here we alternately integrate $\mathcal{S}$ at a low energy (blue circle), then $\mathcal{R}$ (purple line), and then $\mathcal{S}$ at a higher energy level (blue circle).

7 Related work

For now this section will be a casual amalgamation of sources recommended to me. At the very end I'll remove the bullet points and write a comprehensive review.

- **Distilling natural laws from data.** The work of Schmidt & Lipson (2009) is highly complimentary to what we are doing in this paper. It's really well written and it has the same overall objective of learning natural laws in an unsupervised manner. Our work uses neural networks whereas their work uses an ad-hoc genetic algorithm to perform symbolic regression.
- **Neural ODEs.** Our model can be seen as a Neural ODE with restricted dynamics. Whereas Chen et al. (2018) learn arbitrary vector fields, HNNs specifically learn solenoidal vector fields. In some cases our model has the advantage and in others the Neural ODEs have the advantage.
- **Neural representation of potential energy surfaces.** This work by Behler & Parrinello (2007) was recommended to me by Dogus Cubik. I remember him telling me that it learns a mapping from coordinates to potential energy in a supervised manner. I need to actually read it. There's also a follow-up work where some other authors trained on full energy: TODO: ask him for the reference.
- **Learning molecular dynamics force fields.** This paper by Wang et al. (2018) was linked by Jason in our Slack chat. It appears that they learn the free energy $U(x)$ in a similar manner. I think it's complimentary to this paper but I need to read it more carefully.
- **Navier-stokes informed NNs.** In an effort to model fluid flows, Raissi et al. (2018) train a vanilla MLP in much the same way we do. They restrict their investigation to Navier-Stokes and don't learn the Hamiltonian. Like our work, it is an attempt to incorporate better physics priors into neural networks.
- **Visual Interaction Networks.** This super-cool paper by Watters et al. (2017) (DeepMind people) is one of my favorite attempts to incorporate physics priors into neural networks in a domain-agnostic manner. I read somewhere that, from a highly theoretical perspective, VINs are closely related to the Transformer.
- **Tenenbaum papers.** Josh Tenenbaum's lab has about a dozen papers about getting neural networks to learn physics. A good example might be Hamrick et al. (2018) and de Avila Belbute-Peres et al. (2018).

8 Discussion

We have introduced Hamiltonian Neural Networks, a family of parametric models that can be trained to approximate the Hamiltonian of a system directly from data. These models represent one practical attempt to endow neural networks with better physics priors. Our results suggest that these models train an generalize well in practice, scaling to systems with many pairs of canonical coordinates. When trained in conjunction with an autoencoder, they can even learn dynamics straight from pixels.

In this work, we focused on carefully-controlled experiments where the total energy of the system was exactly conserved. Most large-scale, real-world datasets do not have such explicitly-conserved quantities. In order to learn flexible, and general physics models for these systems, we will probably need to combine our approach with other techniques. For example, we might imagine using HNNs as a component of a larger and more general "world model."

The study of Hamiltonian mechanics is, in a way, a counterpoint to the study of deep neural networks. Whereas the former is an old, well-established, and well-understood theory, the latter is still in its infancy and has tendencies towards alchemy. Whereas the former describes the real world from first principles, the latter does so from large datasets. We believe that HNNs represent a unique opportunity to enjoy the best of both worlds.

References

Behler, J. and Parrinello, M. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical review letters*, 98(14):146401, 2007.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. pp. 6571–6583, 2018. URL `http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf`.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pp. 7178–7189, 2018.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pp. 2214–2224, 2017.

Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

Liu, R., Lehman, J., Molino, P., Such, F. P., Frank, E., Sergeev, A., and Yosinski, J. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pp. 9605–9616, 2018.

Raissi, M., Yazdani, A., and Karniadakis, G. E. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *science*, 324 (5923):81–85, 2009.

Wang, J., Olsson, S., Wehmeyer, C., Perez, A., Charron, N. E., de Fabritiis, G., Noe, F., and Clementi, C. Machine learning of coarse-grained molecular dynamics force fields. *ACS Central Science*, 2018.

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

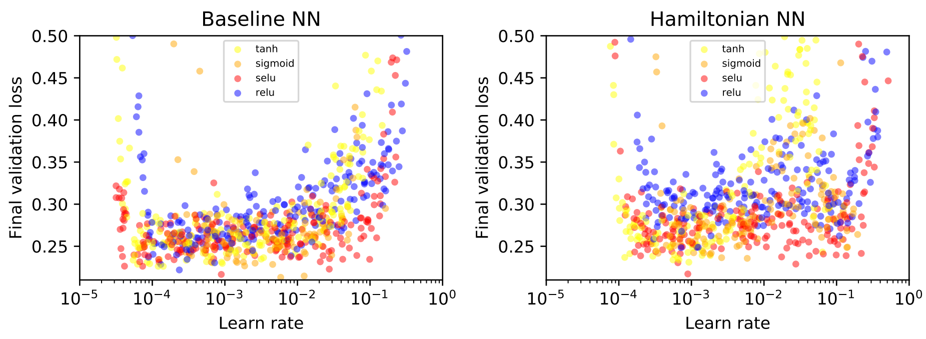


Figure 7: We trained a number of models with different learning rates and nonlinearities on the first toy task. We found that the `relu` activation degraded performance in HNN models but not baseline models. We suspect that the second-order gradient computed during HNN optimization suffers when some first-order gradients are set to zero by the `relu` activation. Based on this observation, we used `tanh` activations in all of our HNN models.

A Toy tasks

B Orbit experiment

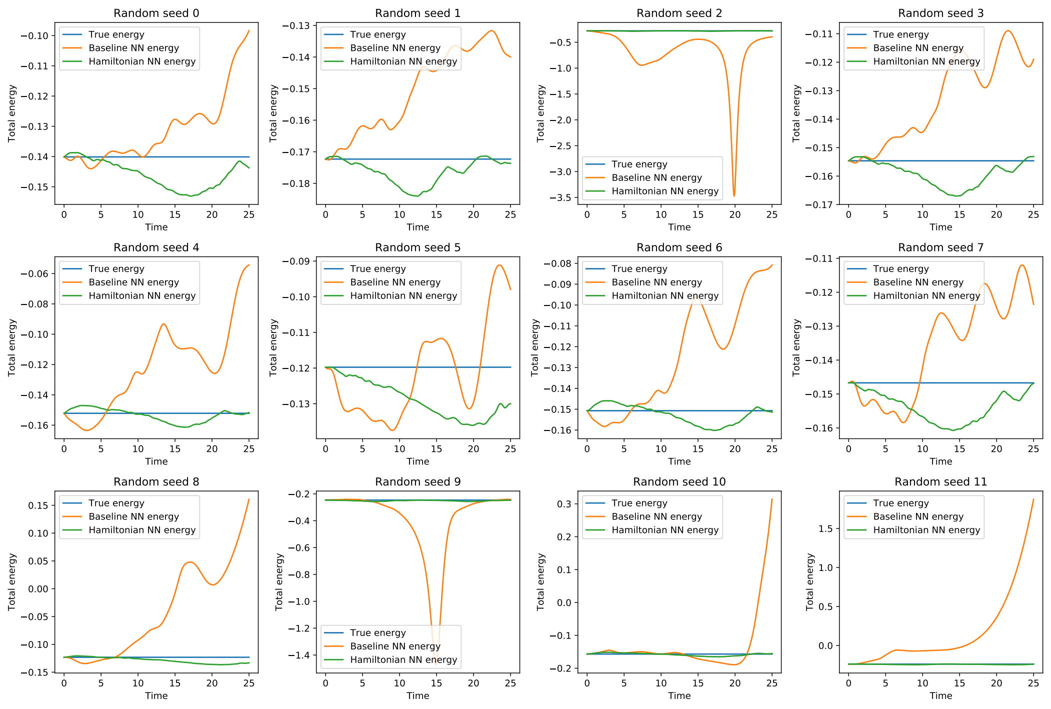
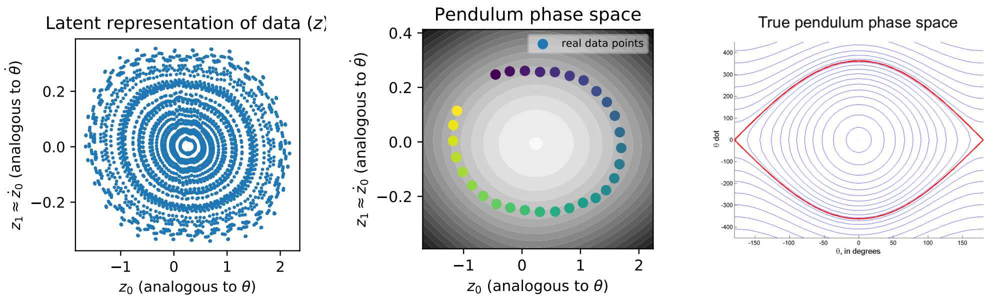


Figure 8: Some qualitative examples of integration results on the orbit task.

C PixelHNN experiment



(a) Latent space (b) Energy contours (c) True phase space

Figure 9: Latent space plots from the PixelHNN. It's interesting to note that the learned latent space bears a strong resemblance to the true phase space of a pendulum. In fact, the outer contour lines in Figure 9(b) become slightly diamond-shaped in a way that resembles the lines just inside the red boundary in Figure 9(c)

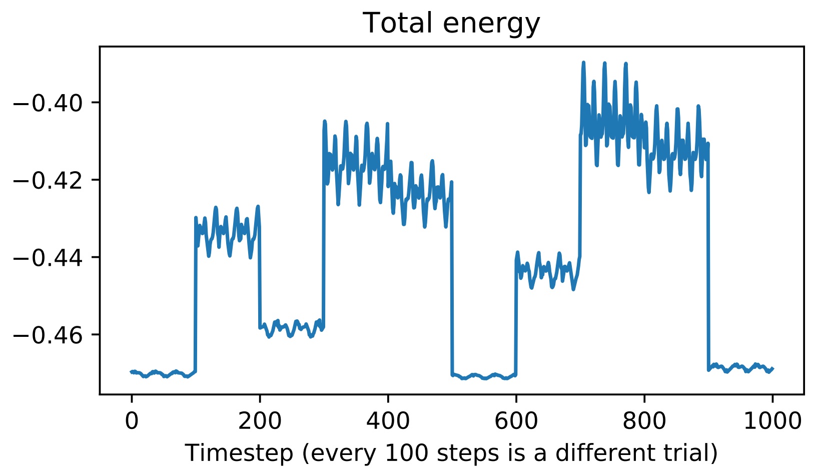


Figure 10: Here we plot the scalar output of a the trained PixelHNN network. This quantity is analogous to the total energy of the system; different trials show distinctly different energy levels. The within-trial fluctuations correspond to approximation error, which grows with larger energies.