Lagrangian Neural Networks

Miles Cranmer
Princeton University
mcranmer@princeton.edu

Sam Greydanus
Oregon State University
greydanus.17@gmail.com

Stephan Hoyer
Google Research
shoyer@google.com

Peter Battaglia
DeepMind
peterbattaglia@google.com

David Spergel[*]
Flatiron Institute
davidspergel@flatironinstitute.org

Shirley Ho[†]
Flatiron Institute
shirleyho@flatironinstitute.org

Abstract

Accurate models of the world are built upon notions of its underlying symmetries. In physics, these symmetries correspond to conservation laws, such as for energy and momentum. Yet even though neural network models see increasing use in the physical sciences, they struggle to learn these symmetries. In this paper, we propose Lagrangian Neural Networks (LNNs), which can parameterize arbitrary Lagrangians using neural networks. In contrast to models that learn Hamiltonians, LNNs do not require canonical coordinates, and thus perform well in situations where canonical momenta are unknown or difficult to compute. Unlike previous approaches, our method does not restrict the functional form of learned energies and will produce energy-conserving models for a variety of tasks. We test our approach on a double pendulum and a relativistic particle, demonstrating energy conservation where a baseline approach incurs dissipation and modeling relativity without canonical coordinates where a Hamiltonian approach fails. Finally, we show how this model can be applied to graphs and continuous systems using a Lagrangian Graph Network, and demonstrate it on the 1D wave equation.

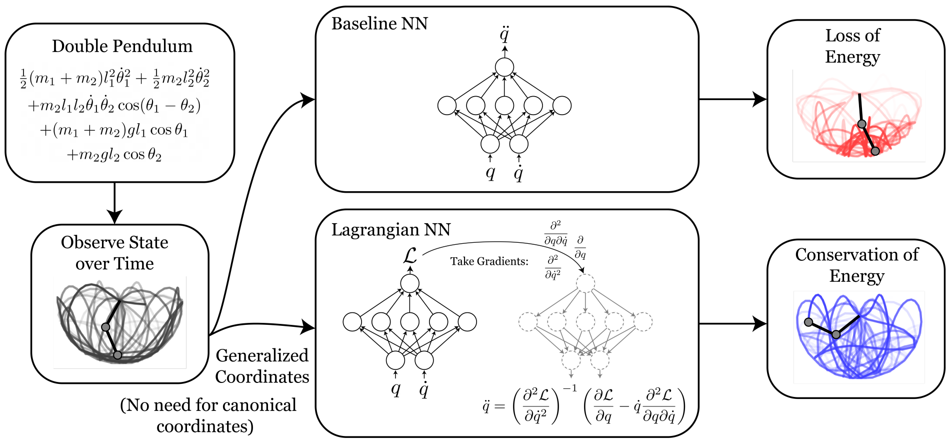

Double Pendulum
$\frac{1}{2}(m_1+m_2)l_1^2\dot\theta_1^2+\frac{1}{2}m_2l_2^2\dot\theta_2^2$
$+m_2l_1l_2\dot\theta_1\dot\theta_2\cos(\theta_1-\theta_2)$
$+(m_1+m_2)gl_1\cos\theta_1$
$+m_2gl_2\cos\theta_2$
Baseline NN
$\ddot q$
q $\dot q$
Loss of Energy
Observe State over Time
Generalized Coordinates
(No need for canonical coordinates)
Lagrangian NN
$\mathcal{L}$
Take Gradients:
$\frac{\partial^2}{\partial q\partial\dot q}$ $\frac{\partial}{\partial q}$ $\frac{\partial^2}{\partial\dot q^2}$
q $\dot q$
$\ddot q=\left(\frac{\partial^2\mathcal{L}}{\partial\dot q^2}\right)^{-1}\left(\frac{\partial\mathcal{L}}{\partial q}-\dot q\frac{\partial^2\mathcal{L}}{\partial q\partial\dot q}\right)$
Conservation of Energy

Figure 1: Physicists use Lagrangians to describe the dynamics of physical systems like the double pendulum (black). Neural networks struggle to model these dynamics over long time periods due to their inability to conserve energy (red). In this paper, we show how to learn arbitrary Lagrangians with neural networks, inducing a strong physical prior on the learned dynamics (blue).

[*]Also affiliated with Princeton University
[†]Also affiliated with New York University, Princeton University, and Carnegie Melon University

1 Introduction

Neural networks excel at finding patterns in noisy and high-dimensional data. They can perform well on tasks such as image classification (Krizhevsky et al., 2012), language translation (Sutskever et al., 2014), and game playing (Silver et al., 2017). Part of their success is rooted in "deep priors" which include hard-coded translation invariances (e.g., convolutional filters), clever architectural choices (e.g., self-attention layers), and well-conditioned optimization landscapes (e.g., batch normalization). Yet, in spite of their ability to compete with humans on challenging tasks, these models lack many basic intuitions about the dynamics of the physical world. Whereas a human can quickly deduce that a ball thrown upward will return to their hand at roughly the same velocity, a neural network may never grasp this abstraction, even after seeing thousands of examples (Bakhtin et al., 2019).

The basic problem with neural network models is that they struggle to learn basic symmetries and conservation laws. One solution to this problem is to design neural networks that can learn arbitrary conservation laws. This was the core motivation behind Hamiltonian Neural Networks by Greydanus et al. (2019) and Hamiltonian Generative Networks by Toth et al. (2019). These are both types of differential equations modeled by neural networks, or "Neural ODEs," which were studied extensively in Chen et al. (2018).

Yet while these models were able to learn effective conservation laws, the Hamiltonian formalism requires that the coordinates of the system be "canonical." In order to be canonical, the input coordinates (q, p) to the model should obey a strict set of rules given by the Poisson bracket relations:

$$p_i \equiv \frac{\partial \mathcal{L}}{\partial \dot{q}_i} \iff \{q_i, q_j\} = 0 \quad \{p_i, p_j\} = 0 \quad \{q_i, p_j\} = \delta_{ij}, \tag{1}$$
$$\text{where } \{f, g\} = \sum_i \left(\frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i} \right),$$

where $\dot{q}$ indicates a time derivative. The problem is that many datasets have dynamical coordinates that do not satisfy this constraint: p_i is not simply $\dot{q}_i \times$ mass in many cases. A promising alternative is to learn the Lagrangian of the system instead. Like Hamiltonians, Lagrangians enforce conservation of total energy, but unlike Hamiltonians, they can do so using arbitrary coordinates.

In this paper, we show how to learn Lagrangians using neural networks. We demonstrate that they can model complex physical systems which Hamiltonian Neural Networks (HNNs) fail to model while outperforming a baseline neural network in energy conservation. We discuss our work in the context of "Deep Lagrangian Networks," (DeLaNs) described in Lutter et al. (2019), a closely related method for learning Lagrangians. Unlike our work, DeLaNs focus on continuous control applications and only model rigid body dynamics. Our model is more general in that it does not restrict the functional form of the Lagrangian. We also show how our model can be applied to continuous systems and graphs using a Lagrangian Graph Network in section 5.

Table 1: An overview of neural network based models for physical dynamics. Lagrangian Neural Networks combine many desirable properties. DeLaNs explicitly restrict the learned kinetic energy, and HNNs implicitly do as well by requiring a definition of the canonical momenta.

	Neural net	Neural ODE	HNN	DeLaN	LNN (ours)
Can model dynamical systems	✓	✓	✓	✓	✓
Learns differential equations		✓	✓	✓	✓
Learns exact conservation laws			✓	✓	✓
Learns from arbitrary coords.	✓	✓		✓	✓
Learns arbitrary Lagrangians					✓

2 Theory

Lagrangians. The Lagrangian formalism models a classical physics system with coordinates $x_t = (q, \dot{q})$ that begin in one state x_0 and end up in another state x_1. There are many paths that these coordinates might take as they pass from x_0 to x_1, and we associate each of these paths with a scalar value called "the action." Lagrangian mechanics tells us that if we let the action be the functional:

$$S \;=\; \int_{t_0}^{t_1} T(q_t, \dot{q}_t) - V(q_t, \dot{q}_t) \;\; dt, \tag{2}$$

where T is kinetic energy and V is potential energy, then there is only one path that the physical system will take, and that path is the stationary (e.g., minimum) value of S. In order to enforce the requirement that S be stationary, i.e., $\delta S = 0$, we define the Lagrangian to be $\mathcal{L} \equiv T - V$, and derive a constraint equation called the *Euler-Lagrange equation* which describes the path of the system:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} = \frac{\partial \mathcal{L}}{q_j}. \tag{3}$$

Euler-Lagrange with a parametric Lagrangian. Physicists traditionally write down an analytical expression for $\mathcal{L}$ and then symbolically expand the Euler-Lagrange equation into a system of differential equations. However, since $\mathcal{L}$ is now a black box, we must resign ourselves to the fact that there can be no analytical expansion of the Euler-Lagrange equation. Fortunately, we can still derive a numerical expression for the dynamics of the system. We begin by rewriting the Euler-Lagrange equation in vectorized form as

$$\frac{d}{dt}\nabla_{\dot{q}}\mathcal{L} = \nabla_q \mathcal{L} \tag{4}$$

where $(\nabla_{\dot{q}})_i \equiv \frac{\partial}{\partial \dot{q}_i}$. Then we can use the chain rule to expand the time derivative $\frac{d}{dt}$ through the gradient of the Lagrangian, giving us two new terms, one with a $\ddot{q}$ and another with a $\dot{q}$:

$$(\nabla_{\dot{q}}\nabla_{\dot{q}}^{\top}\mathcal{L})\ddot{q} + (\nabla_q \nabla_{\dot{q}}^{\top}\mathcal{L})\dot{q} = \nabla_q \mathcal{L}. \tag{5}$$

Here, these products of ∇ operators are matrices, e.g., $(\nabla_q \nabla_{\dot{q}}^{\top}\mathcal{L})_{ij} = \frac{\partial^2 \mathcal{L}}{\partial q_j \partial \dot{q}_i}$. Now we can use a matrix inverse to solve for $\ddot{q}$:

$$\boxed{\ddot{q} = (\nabla_{\dot{q}}\nabla_{\dot{q}}^{\top}\mathcal{L})^{-1}[\nabla_q \mathcal{L} - (\nabla_q \nabla_{\dot{q}}^{\top}\mathcal{L})\dot{q}].} \tag{6}$$

For a given set of coordinates $x_t = (q_t, \dot{q}_t)$, we now have a method for calculating $\ddot{q}_t$ from a black box Lagrangian, which we can integrate to find the dynamics of the system. In the same manner as Greydanus et al. (2019), we can also write a loss function in terms of the discrepancy between $\ddot{x}_t^{\mathcal{L}}$ and $\ddot{x}_t^{\text{true}}$.

3 Related Work

Physics priors. A variety of previous works have sought to endow neural networks with physics-motivated inductive biases. These include work for domain-specific problems in molecular dynamics (Rupp et al., 2012), quantum mechanics (Schütt et al., 2017), or robotics (Lutter et al., 2019). Other are more general, such as Interaction Networks (Battaglia et al., 2016), which models physical interactions as message passing on a graph.

Learning invariant quantities. Recent work by Greydanus et al. (2019) and Toth et al. (2019), built on previous approaches of endowing neural networks with physical priors by demonstrating how to learn invariant quantities by approximating a Hamiltonian with a neural network. In this paper, we follow the same approach as Greydanus et al. (2019), but with the objective of learning a Lagrangian rather than a Hamiltonian so not to restrict the learned kinetic energy.

DeLaN. A closely related previous work is "Deep Lagrangian Networks", or DeLaN (Lutter et al., 2019), in which the authors show how to learn specific types of Lagrangian systems. They assume that the kinetic energy is an inner product of the velocity: $T = \dot{q}^T M \dot{q}$, where M is a positive definite matrix. This approach works well for rigid body dynamics, which includes many systems encountered in robotics. However, many systems do not have this kinetic energy, including, for example, a charged particle in a magnetic field, and a fast-moving object in special relativity.

4 Methods

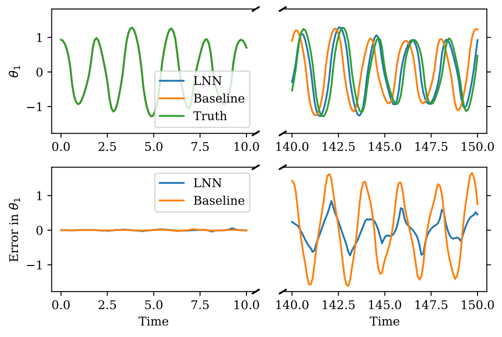
Solving Euler-Lagrange with JAX. Efficiently implementing equation 6 represents a formidable technical challenge. In order to train this forward model, we need to compute the inverse Hessian $(\nabla_{\dot{q}}\nabla_{\dot{q}}^{\top}\mathcal{L})^{-1}$ (we use the pseudoinverse to avoid potential singular matrices) of a neural network and then perform backpropagation. The matrix inverse scales as $\mathcal{O}(d^3)$ with the number of coordinates d. Perhaps surprisingly, though, we can implement this operation in a single line of JAX (Bradbury et al., 2018) (see Appendix). We publish all of our code on GitHub[1].

Training details. For both baseline, HNN, and LNN models, we used a four-layer neural network model with 500 hidden units, a decaying learning rate starting at 10^{-3}, and a batch size of 32. We initialize our model using a novel initialization scheme described in appendix C, which was optimized for LNNs.

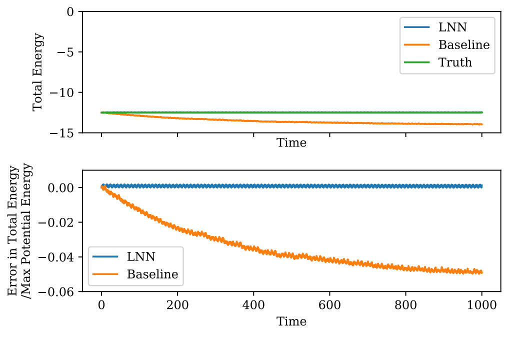
Activation functions. Since we take the Hessian of a LNN, the second-order derivative of the activation function is important. For example, the ReLU nonlinearity is a poor choice because its second-order derivative is zero. In order to find a better activation function, we performed a hyperparameter search over ReLU^2 (squared), ReLU^3, tanh, sigmoid, and softplus activations on the double pendulum problem. We found that softplus performed best and thus used it for all experiments.

5 Experiments

Double pendulum. The first task we considered was a dataset of simulated double pendulum trajectories. We set the masses and lengths to 1 and learned the instantaneous accelerations over 600,000 random initial conditions. We found that the LNN and the baseline converged to similar final losses of 7.3 and 7.4×10^{-2}, respectively. The more significant difference was in energy conservation; the LNN almost exactly conserved the true energy over time, whereas the baseline did not. Averaging over 40 random initial conditions with 100 time steps each, the mean energy discrepancy between the true total energy and predicted was 8% and 0.4% of the max potential energy for the baseline and LNN models respectively.



(a) Error in angle



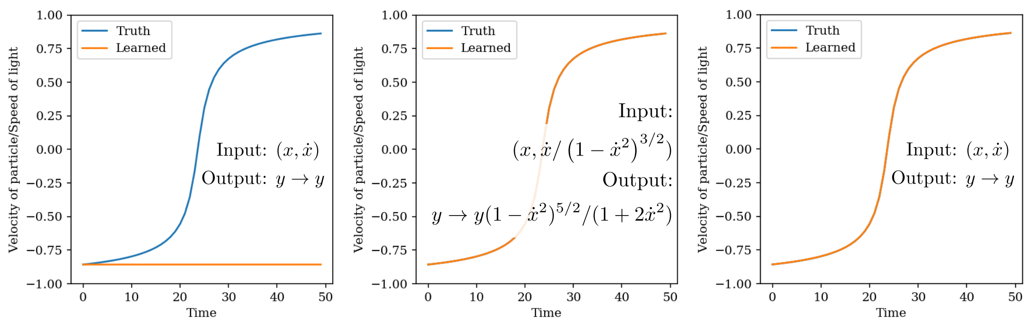
(b) Error in energy

Figure 2: Results on the double pendulum task. In (a), we see that the LNN and baseline model perform similarly in modelling the dynamics of the pendulum over short time periods. However, if we plot out the energy over a very long time period in (b) we can see that the LNN model conserves the total energy of the system significantly better than the baseline.

Relativistic particle in a uniform potential. The second task was a relativistic particle in a uniform potential. For a particle with a mass of 1 in a potential g and with $c = 1$, special relativity gives the Lagrangian $\mathcal{L} = ((1 - \dot{q}^2)^{-1/2} - 1) + gq$. The canonical momenta of this system are $\dot{q}(1 - \dot{q}^2)^{-3/2}$, which means that a Hamiltonian Neural Network will fail if given simple observables like $\dot{q}$ and q. The DeLaN model will also struggle since it assumes that T is second order in $\dot{q}$. To verify these

[1]`https://github.com/MilesCranmer/lagrangian_nn_public`

predictions, we trained HNN and LNN models on systems with random initial conditions and values of g. Figure 3 shows that the HNN fails without canonical coordinates whereas the LNN can work without this extra *a priori* knowledge, and learns the system as accurately as an HNN trained on canonical coordinates.



(a) HNN, arbitrary coords. (b) HNN, canonical coords. (c) LNN, arbitrary coords.

Figure 3: Results on the relativistic particle task. In (a) an HNN model fails to learn dynamics from non-canonical coordinates. In (b) the HNN succeeds when given canonical coordinates. Finally in (c) the LNN learns accurate dynamics even from non-canonical coordinates.

Wave equation with Lagrangian Graph Networks. Equation (6) is applicable to many different types of systems, including those with disconnected coordinates such as graph-like and grid-like systems. To model this, we now learn a Lagrangian density which is summed to form the full Lagrangian. This is similar to the Hamiltonian Graph Network of Sanchez-Gonzalez et al. (2019), or more specifically, the Flattened Hamiltonian Graph Network of Cranmer et al. (2020). For simplicity, we focus on 1D grids with locally-dependent dynamics (i.e., nodes are connected to their two adjacent nodes).

A continuous material can be described in terms of a quantity ϕ_i, such as the displacement of a guitar string, at each gridpoint x_i. One way to model this type of system is to treat the quantities on adjacent gridpoints as independent coordinates, and sum the regular LNN equation over all connected groups of coordinates. For n gridpoints, the total Lagrangian is then:

$$\mathcal{L} = \sum_{i=1:n} \mathcal{L}_i, \text{ for } \mathcal{L}_i = \mathcal{L}_{\text{density}}\left(\{\phi_j, \dot{\phi}_j\}_{j\in\mathcal{I}_i}\right) \tag{7}$$

where $\mathcal{I}_i = \{i, \ldots\}$ is the set of indices connected to i. For a 1D grid where only the adjacent gridpoints affect the dynamics of the center gridpoint, this is $\{i, i-1, i+1\}$. With the adjacency matrix set, we can then write down the forward model for $\ddot{\phi}_i$:

$$\boxed{\ddot{\phi} = \left(\nabla_{\dot{\phi}}\nabla_{\dot{\phi}}^{\top}\mathcal{L}\right)^{-1}\left(\nabla_{\phi}\mathcal{L} - \left(\nabla_{\phi}\nabla_{\dot{\phi}}^{\top}\mathcal{L}\right)\dot{\phi}\right)} \tag{8}$$

where, e.g., $\nabla_\phi \equiv \{\frac{\partial}{\partial\phi_1}, \frac{\partial}{\partial\phi_2}, \ldots \frac{\partial}{\partial\phi_n}\}$. Note that since the Hessian in eq. (8) is sparse, and one can invert a tridiagonal matrix in linear time, one can generally not worry about computational scaling of this inverse for large graphs.

We can think of this as a regular LNN, where the coordinates are all quantities at the current and adjacent grid points. For a different connectivity, such as a graph network, or for higher-order spatial derivatives, one would select $\mathcal{I}_i$ based on the adjacency matrix for the graph. This model is a type of Graph Neural Network. The Lagrangian density itself, $\mathcal{L}_{\text{density}}$, we model as an MLP. Since we write our models in JAX, we can easily vectorize this forward model over the grid.

We consider the 1D wave equation, with ϕ representing the wave displacement: $\phi(x,t)$. For wave speed $c = 1$, the equation describing its dynamics can be written as: $\ddot{\phi} = \frac{\partial^2\phi}{\partial x^2}$. The Lagrangian for this differential equation is: $\mathcal{L} = \int\left(\dot{\phi}^2 - \phi_x^2\right)dx$. For the LNN to learn this, the MLP will need to learn to approximate a finite difference operator: $\mathcal{L}_i = \frac{1}{2}\dot{\phi}_i^2 - \frac{1}{2}\left(\frac{\phi_{i+1}-\phi_{i-1}}{2\Delta x}\right)^2$ for grid spacing Δx (or any translation + scaling of this equation). We simulate the wave equation in a box with periodic boundary conditions, and learn the dynamics with this Lagrangian Graph Network, shown in fig. 4. The Lagrangian Graph Network models the wave equation accurately and almost exactly conserves energy integrated across the continuous material.

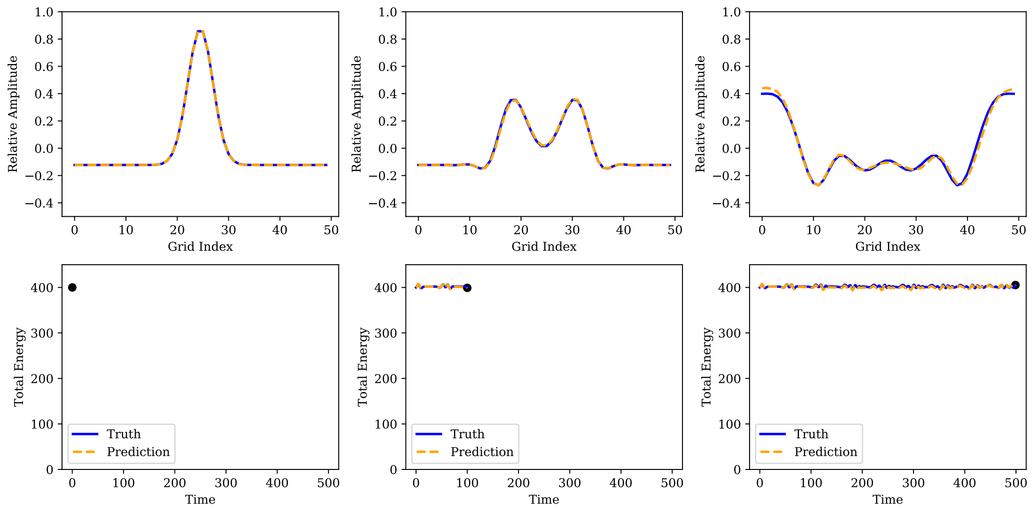


Figure 4: Results on the 1D wave equation task, using a Lagrangian Graph Network. Here, the LNN models the Lagrangian density over three adjacent gridpoints along the wave, and this density is summed. Here, the waves make approximately four and a half passes across the 50 grid points over the time plotted.

6 Conclusion

We have introduced a new class of neural networks, Lagrangian Neural Networks, which can learn arbitrary Lagrangians. In contrast to models that learn Hamiltonians, these models do not require canonical coordinates and thus perform well in situations where canonical momenta are unknown or difficult to compute. To evaluate our model, we showed that it could effectively conserve total energy on a complex physical system: the double pendulum. We showed that our model could learn non-trivial canonical momenta on a task where Hamiltonian learning struggles. Finally, we demonstrated a graph version of the model and learned the 1D wave equation.

Acknowledgments Miles Cranmer would like to thank Jeremy Goodman for several discussions and validation of the theory behind the Lagrangian approach, and Adam Burrows and Oliver Philcox for very helpful comments on a draft of the paper.

References

Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. In *Advances in Neural Information Processing Systems*, pp. 5083–5094, 2019.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. *Advances in neural information processing systems*, pp. 6571–6583, 2018. URL `http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf`.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering Physical Equations with Graph Networks. *forthcoming*, 2020.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*, pp. 15353–15363, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian Networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*, 2019.

Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters*, 108(5):058301, 2012.

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

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

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian Generative Networks. *International Conference on Machine Learning*, 2019.

Appendix

A Solving Euler-Lagrange with JAX

Here we will present a simple JAX implementation of Equation 6. Assume that `lagrangian` is a differentiable function that takes three vectors as input and outputs a scalar. Meanwhile, `q_t` is a vector of the velocities ($\dot{q}$) of `q` (q) and `q_tt` contains the accelerations ($\ddot{q}$). The vector `m` represents non-dynamical parameters.

```
q_tt = (
 jax.numpy.linalg.pinv(jax.hessian(lagrangian, 1)(q, q_t, m)) @ (
    jax.grad(lagrangian, 0)(q, q_t, m)
  - jax.jacobian(jax.jacobian(lagrangian, 1), 0)(q, q_t, m) @ q_t
 )
)
```

When this is called in a loss function with the LNN parameters as input: `loss(params, ...)`, one can write `jax.grad(loss, 0)(params, ...)`, to get the gradient.

B Example of Lagrangian Forward Model

To demonstrate how this may work if one has learned the exact function for $\mathcal{L}$, let us study an example of a ball falling in a gravity g along the direction q_1:

$$\mathcal{L} = \frac{1}{2} m \left(\dot{q}_1^2 + \dot{q}_2^2\right) - mgq_1, \tag{9}$$

where g is the local scalar gravitational field, and m is the mass of the ball. To obtain the dynamics with our forward model, we calculate the required derivatives which gives us:

$$\nabla_{\dot{q}} \nabla_{\dot{q}}^{\top} \mathcal{L} = \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix}, \tag{10}$$

$$\nabla_{q} \nabla_{\dot{q}}^{\top} \mathcal{L} = 0, \text{ and} \tag{11}$$

$$\mathcal{L}_q = \begin{pmatrix} -mg \\ 0 \end{pmatrix}. \tag{12}$$

Thus, we find

$$\begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{pmatrix} = (\nabla_{\dot{q}} \nabla_{\dot{q}}^{\top} \mathcal{L})^{-1} \left[\nabla_q \mathcal{L} - (\nabla_q \nabla_{\dot{q}}^{\top} \mathcal{L})\dot{q}\right] \tag{13}$$

$$= \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix}^{-1} \left[\begin{pmatrix} -mg \\ 0 \end{pmatrix}\right] \tag{14}$$

$$= \begin{pmatrix} -g \\ 0 \end{pmatrix} \tag{15}$$

which simply says that the particle accelerates downwards along q_1, and moves at constant velocity along q_2.

C Initialization

Regular optimization techniques such as Kaiming (He et al., 2015) and Xavier (Glorot & Bengio, 2010) initialization are optimized so that in a regular neural network, the gradients of the output with respect to each parameter will have a mean of zero and standard deviation of one. Since the unusual optimization objective of a Lagrangian Neural Network is very nonlinear with respect to its parameters, we found that classical initialization schemes were insufficient.

To find a better initialization scheme, we conducted an empirical optimization of the KL-divergence of the gradient of each parameter with respect to a univariate Gaussian. We repeated this over a variety of neural network depths and widths and fit an empirical formula to our results.

To do this, we ran 2500 optimization steps with different initialization variances on each layer for an MLP of fixed depth and width. Biases were always initialized to zero. We recorded the optimized σ values over $\sim$200 random hyperparameter settings with the number of hidden nodes between 50 and 300 and the number of hidden layers between one and three. Then, we used *eureqa* (Schmidt & Lipson, 2009) to find fit an equation using symbolic regression that predicted the optimal initialization variance as a function of the hyperparameters:

$$\sigma = \frac{1}{\sqrt{n}} \begin{cases} 2.2 & \text{First layer} \\ 0.58i & \text{Hidden layer } i \in \{1, \dots\} \\ n, & \text{Output layer,} \end{cases} \tag{16}$$

This model was optimized for 2 input coordinates and 2 input coordinate velocities which were sampled from univariate Gaussians.

During training, the hidden weight matrices had dimensions $n \times n$ and each weight matrix was sampled from $\mathcal{N}(0, \sigma^2)$. A 100-node 4-layer model would have weight matrices with shapes $\{(4, 100), (100, 100), (100, 100), (100, 1)\}$ and each one had initializations sampled from $\{\mathcal{N}(0, 0.22), \mathcal{N}(0, 0.058), \mathcal{N}(0, 0.116), \mathcal{N}(0, 10)\}$, respectively.