# 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

## 초록

세계의 정확한 모델은 기본 대칭에 대한 개념을 기반으로 구축됩니다. 물리학에서 이러한 대칭성은 에너지 및 운동량과 같은 보존 법칙에 해당합니다. 그러나 신경망 모델은 물리 과학에서 점점 더 많이 사용되고 있지만 이러한 대칭을 학습하는 데 어려움을 겪고 있습니다. 이 논문에서는 신경망을 사용하여 임의의 라그랑지안을 매개변수화할 수 있는 라그랑지안 신경망(LNN)을 제안합니다. 해밀턴을 학습하는 모델과 달리 LNN은 정식 좌표가 필요하지 않으므로 정식 모멘타를 알 수 없거나 계산하기 어려운 상황에서도 우수한 성능을 발휘합니다. 이전 접근 방식과 달리, 우리의 방법은 학습된 에너지의 함수 형태를 제한하지 않으며 다양한 작업에 대해 에너지를 절약하는 모델을 생성합니다. 이중 진자와 상대론적 입자에 대해 우리의 접근법을 테스트하여 기준 접근법에서 소실이 발생하는 경우 에너지 보존을 입증하고 해밀턴 접근법이 실패하는 경우 정식 좌표 없이 상대론을 모델링합니다. 마지막으로 라그랑지안 그래프 네트워크를 사용하여 이 모델을 그래프와 연속 시스템에 적용하는 방법을 보여주고, 1D 파동 방정식에서 이를 시연합니다.

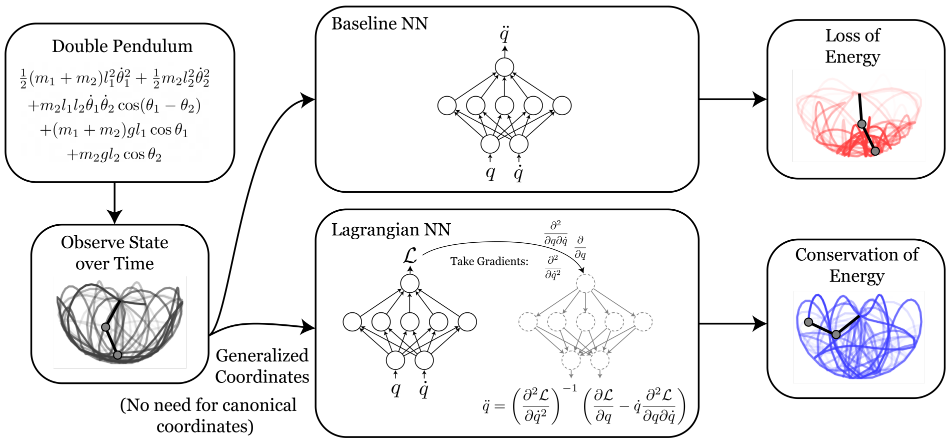

Double Pendulum
$\frac{1}{2}(m_1+m_2)l_1^2\dot\theta_1^2 + \frac{1}{2}m_2 l_2^2\dot\theta_2^2$
$+m_2 l_1 l_2 \dot\theta_1\dot\theta_2\cos(\theta_1-\theta_2)$
$+(m_1+m_2)g l_1\cos\theta_1$
$+m_2 g l_2\cos\theta_2$
Observe State over Time
Generalized Coordinates
(No need for canonical coordinates)
Baseline NN
$\ddot q$
$q$ $\dot q$
Lagrangian NN
$\mathcal{L}$
Take Gradients:
$\frac{\partial^2}{\partial q\partial\dot q}$ $\frac{\partial}{\partial q}$ $\frac{\partial^2}{\partial \dot q^2}$
$q$ $\dot q$
$\ddot q = \left(\frac{\partial^2\mathcal{L}}{\partial\dot q^2}\right)^{-1}\left(\frac{\partial\mathcal{L}}{\partial q} - \dot q\frac{\partial^2\mathcal{L}}{\partial q\partial\dot q}\right)$
Loss of Energy
Conservation of Energy

그림 1: 물리학자들은 라그랑지안을 사용하여 이중 진자(검은색)와 같은 물리 시스템의 동역학을 설명합니다. 신경망은 에너지를 보존할 수 없기 때문에 장기간에 걸쳐 이러한 동역학을 모델링하는 데 어려움을 겪습니다(빨간색). 이 백서에서는 신경망으로 임의의 라그랑지안을 학습하고 학습된 동역학에 강력한 물리적 사전을 유도하는 방법을 보여줍니다(파란색).

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

1 소개
신경망은 노이즈가 많은 고차원 데이터에서 패턴을 찾는 데 탁월합니다. 이미지 분류 (Krizhevsky 외., 2012), 언어 번역(Sutskever 외., 2014), 게임 플레이(Silver 외., 2017)와 같은 작업에서 뛰어난 성능을 발휘할 수 있습니다. 이러한 성공의 일부는 하드코딩된 번역 불변성 (예: 컨볼루션 필터), 영리한 아키텍처 선택(예: 셀프 어텐션 레이어), 잘 조정된 최적화 환경 (예: 배치 정규화)을 포함하는 "딥 프라이어"에 뿌리를 두고 있습니다. 그러나 이러한 모델은 어려운 작업에서 인간과 경쟁할 수 있는 능력에도 불구하고 물리적 세계의 역학에 대한 기본적인 직관이 많이 부족합니다. 인간은 위로 던진 공이 거의 같은 속도로 자신의 손으로 돌아올 것이라는 것을 빠르게 추론할 수 있는 반면, 신경망은 수천 개의 예제를 보고도 이러한 추상성을 파악하지 못할 수 있습니다(Bakhtin 외., 2019).
신경망 모델의 기본적인 문제는 기본적인 대칭과 보존 법칙을 학습하는 데 어려움을 겪는다는 것입니다. 이 문제에 대한 한 가지 해결책은 임의의 보존 법칙을 학습할 수 있는 신경망을 설계하는 것입니다. 이것이 바로 그레이다누스 외(2019)의 해밀턴 신경망과 토스 외(2019)의 해밀턴 생성 신경망의 핵심 동기입니다. 이 두 가지 모델은 모두 신경망으로 모델링된 미분 방정식 또는 "신경 ODE"로, Chen et al.(2018)에서 광범위하게 연구된 바 있습니다.
그러나 이러한 모델은 효과적인 보존 법칙을 학습할 수 있었지만, 해밀턴 형식주의에서는 시스템의 좌표가 "정식"이어야 합니다. 정식이 되기 위해서는 모델에 입력되는 좌표(q, p)가 푸아송 괄호 관계에 의해 주어진 엄격한 규칙 집합을 따라야 합니다:

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

where $\dot{q}$ indicates a time derivative and $\mathcal{L}$ the Lagrangian. 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.
유망한 대안은 시스템의 라그랑지안을 대신 학습하는 것입니다. 라그랑지안은 해밀토니안과 마찬가지로 총 에너지 보존을 강제하지만, 해밀토니안과 달리 임의의 좌표를 사용하여 이를 수행할 수 있습니다.
이 논문에서는 신경망을 사용하여 라그랑지안을 학습하는 방법을 보여줍니다. 이를 통해 해밀턴 신경망(HNN)이 모델링하지 못하는 복잡한 물리 시스템을 모델링할 수 있으며, 에너지 보존 측면에서 기준 신경망보다 뛰어난 성능을 보인다는 것을 증명합니다. 저희는 라그랑지안 학습과 밀접한 관련이 있는 방법인 Lutter 외(2019)의 "심층 라그랑지안 네트워크"(DeLaN)의 맥락에서 저희의 작업을 논의합니다. 우리의 작업과 달리 DeLaN은 연속 제어 애플리케이션을 위해 구축되었으며 강체 동역학만 모델링합니다. 우리의 모델은 라그랑지안의 함수 형태를 제한하지 않는다는 점에서 더 일반적입니다. 또한 섹션 5에서 라그랑지안 그래프 네트워크를 사용하여 연속 시스템과 그래프에 모델을 적용하는 방법을 보여줍니다.
표 1: 물리 역학을 위한 신경망 기반 모델의 개요. 라그랑지안 신경망은 여러 가지 바람직한 특성을 결합합니다. DeLaN은 학습된 운동 에너지를 명시적으로 제한하고, HNN은 정식 모멘타의 정의를 요구함으로써 암시적으로도 제한합니다.

| | 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:

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

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:

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

**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 $\nabla$ 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 (Gupta et al., 2019; Lutter et al., 2019). Other approaches 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), Toth et al. (2019), and Chen 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 $q$-dependent 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. Other Lagrangian-based approaches include Gupta et al. (2019); Qin (2019).

## 4 METHODS

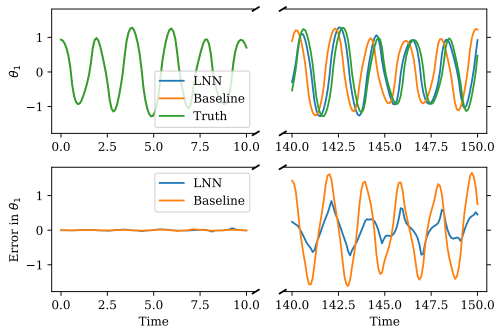
**Solving Euler-Lagrange with JAX.** Efficiently implementing equation 6 represents a formidable technical challenge. In order to train this forward model, we need to compute the inverse Hessian $(\nabla_{\dot{q}} \nabla_{\dot{q}}^{\top} \mathcal{L})^{-1}$ (we use the pseudoinverse to avoid potential 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 few lines of JAX (Bradbury et al., 2018) (see Appendix). We publish 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.

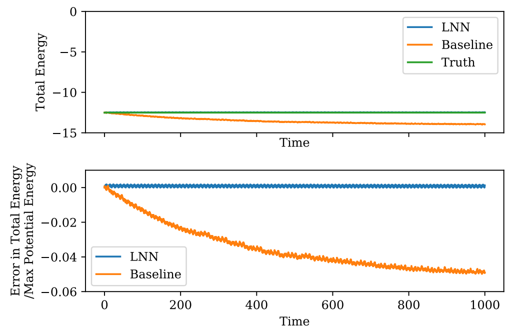
**Activation functions.** Since we take the Hessian of a LNN, the second-order derivative of the activation function is important. For example, the ReLU nonlinearity is a poor choice because its second-order derivative is zero. In order to find a better activation function, we performed a hyperparameter 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 $7.4 \times 10^{-2}$, 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



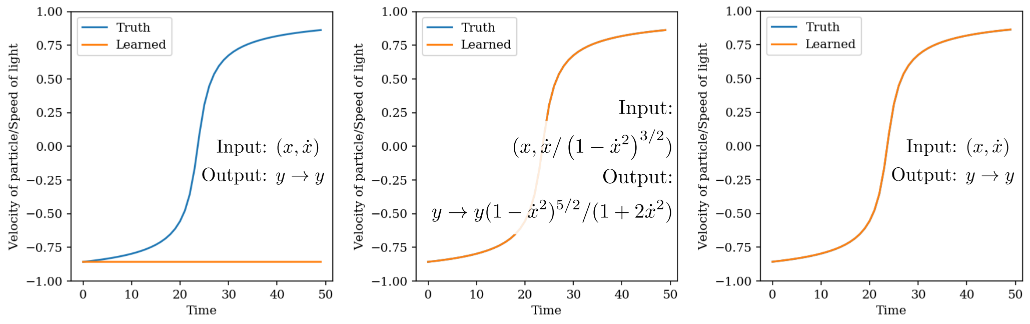
(b) Error in energy

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_nns`

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 quantities $\phi_i$, 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 $\{i, i-1, i+1\}$. Again, we can solve dynamics by plugging the Lagrangian into eq. (6), now with $\phi$ as the coordinates:

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

where, e.g., $\nabla_\phi \equiv \{\frac{\partial}{\partial\phi_1}, \frac{\partial}{\partial\phi_2}, \ldots, \frac{\partial}{\partial\phi_n}\}$. Note that the Hessian matrix is sparse with non-zero entries at "neighbor of neighbor" positions in the graph, which can make it much more efficient to calculate and invert. For example, in 1D it can be calculated with only 5 forward over backwards auto-differentiation passes and can be inverted in linear time.

We can think of this as a regular LNN where, instead of calculating the Lagrangian directly from a fixed set of coordinates, we are now accumulating a Lagrangian density over groups of coordinates. For a different connectivity, such as a graph network, or to approximate higher-order spatial derivatives for a continuous material, one would select $\mathcal{I}_i$ based on the adjacency matrix for the graph. This model is a type of Graph Neural Network (Scarselli et al., 2008). 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 $\phi$ 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: $\mathcal{L} = \int \left( \dot{\phi}^2 - \left( \frac{\partial \phi}{\partial x} \right)^2 \right) dx$. For the LNN to learn this, the MLP will need to learn to approximate a finite difference operator: $\mathcal{L}_i = \dot{\phi}_i^2 - \left( \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \right)^2$ for grid spacing $\Delta x$ (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 material.

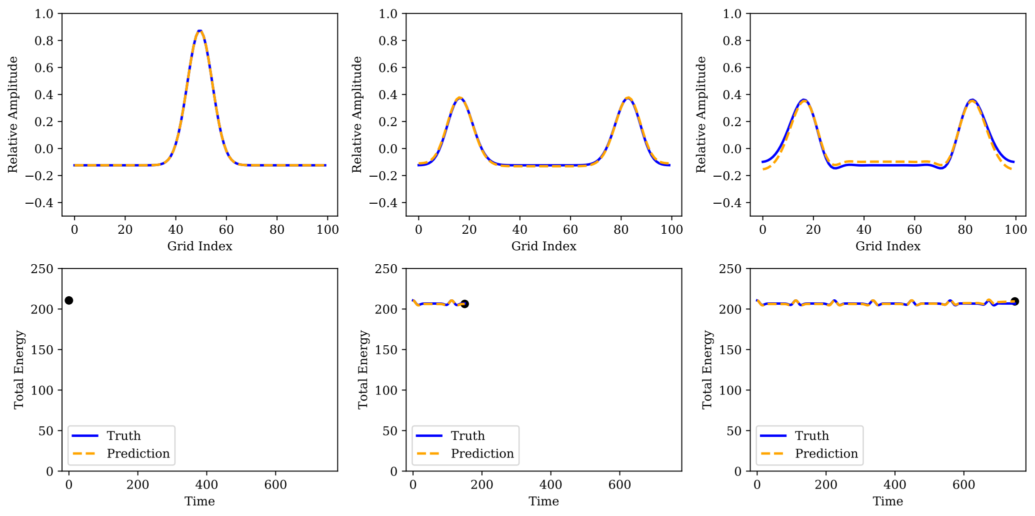


Figure 4: Results on the 1D wave equation task, using a Lagrangian Graph Network. Here, the LNN models the Lagrangian density over three adjacent gridpoints along the wave, and this density is summed. Here, the waves make approximately three and a half passes across the 100 grid points over the time plotted. A movie can be viewed by clicking on the plot or by going to `https://github.com/MilesCranmer/gifs/blob/master/wave_equation.gif`.

## 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 with 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`.

Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. *arXiv preprint arXiv:1909.13334*, 2019.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*, 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.

Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel J Kochenderfer. A general framework for structured learning of mechanical systems. *arXiv preprint arXiv:1902.08705*, 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.

Hong Qin. Machine learning and serving of discrete field theories. *arXiv preprint arXiv:1910.10147*, 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.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

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:

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

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

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

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 $\sigma$ 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:

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

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.