Editors' Suggestion | Featured in Physics

Metalearning Generalizable Dynamics from Trajectories

Qiaofeng Li,[1,2,3] Tianyi Wang,[2] Vwani Roychowdhury,[2,*] and M. K. Jawed[1,†]
[1]Department of Mechanical and Aerospace Engineering, University of California, Los Angeles, Los Angeles, California 90095, USA
[2]Department of Electrical and Computer Engineering, University of California, Los Angeles, Los Angeles, California 90095, USA
[3]Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA

(Received 3 January 2023; revised 24 May 2023; accepted 21 June 2023; published 10 August 2023)

We present the interpretable meta neural ordinary differential equation (iMODE) method to rapidly learn generalizable (i.e., not parameter-specific) dynamics from trajectories of multiple dynamical systems that vary in their physical parameters. The iMODE method learns metaknowledge, the functional variations of the force field of dynamical system instances without knowing the physical parameters, by adopting a bilevel optimization framework: an outer level capturing the common force field form among studied dynamical system instances and an inner level adapting to individual system instances. *A priori* physical knowledge can be conveniently embedded in the neural network architecture as inductive bias, such as conservative force field and Euclidean symmetry. With the learned metaknowledge, iMODE can model an unseen system within seconds, and inversely reveal knowledge on the physical parameters of a system, or as a neural gauge to "measure" the physical parameters of an unseen system with observed trajectories. iMODE can be generally applied to a dynamical system of an arbitrary type or number of physical parameters and is validated on bistable, double pendulum, Van der Pol, Slinky, and reaction-diffusion systems.

DOI: 10.1103/PhysRevLett.131.067301

Building predictive models of dynamical systems is a central challenge across diverse disciplines of science and engineering. Traditionally, this has been achieved by first manually deriving the governing equations with carefully chosen state variables and then fitting the undetermined physical parameters using observed data, e.g., [1–3]. In order to avoid the painstaking formulation of analytical equations, researchers have recently leveraged advances in machine learning and the data-fitting power of neural networks (NNs) to make the modeling process both automatic and more expressive [4]. This is achieved by either adopting the conventional physics-based approach as a starting point and then replacing various components with data-driven modules [5,6], or directly learning discrete dynamics using autoregressive models from high-dimensional observations [7–9]. These works, while promising, need to fit dedicated models separately for different system instances with different parameters, which limits a model's applicability to one specific instance.

In this Letter, our goal is to learn metaknowledge, the form of dynamics that is unrestricted to specific physical parameters or initial and boundary conditions, on dynamical systems to reveal physical insights [10–12] and to significantly improve the generalization ability of data-driven models. Specifically, we learn the shared dynamics form from the trajectories generated by a series of dynamical system instances in spite of their diversified behaviors in data, without knowing the system parameters. This separates our work from Refs. [13,14] and neural operators [15–18], in which true parameters should be provided. This goal aligns with that of multitask metalearning [19], which aims to leverage the similarities between different tasks to enable better generalization and efficient adaptation to unseen tasks.

We propose an efficient and interpretable method to model a family of dynamical systems using their observed trajectories, by combining gradient-based metalearning (GBML) [20–24] with neural ordinary differential equations (NODEs) [6,25,26]. In recognizing that the systems have shared dynamics form and varying physical parameters, we separate the model parameters into two parts: the shared parameters that capture the shared form of dynamics, i.e., the metaknowledge, and the adaptation parameters that account for variations across system instances. The method generalizes well on unseen systems from the same family, and the adaptation parameters show good interpretability. The intrinsic dimension of the varying system parameters can be estimated by analyzing the adaptation parameters. Given ground truth of the system parameters, simple correspondence can be established between the adaptation parameters and actual physical parameters through diffeomorphism, which can be utilized as a "neural gauge" to measure properties of new systems through observed trajectories. We name our method interpretable meta neural ODE (iMODE).

In a general autonomous second-order system, the state of the system $\mathbf{y}$ contains the position (generalized

coordinates) $\mathbf{x}$ and the velocity $\dot{\mathbf{x}}$. The dynamics of the second-order system is expressed equivalently as a first-order system

$$\dot{\mathbf{y}} = \begin{bmatrix} \dot{\mathbf{x}} \\ \ddot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}} \\ \mathbf{M}^{-1}\mathbf{F}_{\boldsymbol{\phi}}(\mathbf{y}) \end{bmatrix}, \quad \text{where } \mathbf{y} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} \tag{1}$$

where $\mathbf{F}_{\boldsymbol{\phi}}$ is the force vector containing all the internal and external forces, and $\mathbf{M}$ is the mass matrix. With a set of physical parameters $\boldsymbol{\phi}$, the force function $\mathbf{F}(\cdot)$ dictates the dynamics of the system, which determines a unique trajectory $\mathbf{y}(t)$ given an initial condition $\mathbf{y}(t_0)$. In the remainder of the Letter, without loss of generality, mass is normalized to an identity matrix, i.e., $\mathbf{M} = \mathbf{I}$. It should be noted that, with this general formulation, the proposed method is applicable to systems other than second order (e.g., first-order diffusion-reaction systems; see Supplemental Material (SM) [27] S6) and can be easily extended to nonautonomous systems (the forcing term should be provided).

Trajectories are collected from multiple system instances into a dataset $\mathcal{D}$. Consider N_s instances that share the dynamics form $\mathbf{F}_{\boldsymbol{\phi}}(\cdot)$, but have distinct physical parameters, $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_{N_s}\}$, respectively. From each system instance, N_{tr} trajectories are observed, each containing observations across T time steps. In summary, $\mathcal{D} = \{\{\mathbf{y}_{i,j}(t_k)\}_{k=0}^{T} | i = 1, \ldots, N_s, j = 1, \ldots, N_{tr}\}$. The data-driven model is trained on $\mathcal{D}$, knowing which trajectories are from the same system instance (i.e., given both the index i and j of trajectories), but is not given the knowledge of $\{\boldsymbol{\phi}_i\}_{i=1}^{N_s}$. Take the pendulum system as an example. An instance is a pendulum with a specific arm length (since the inertia is normalized), therefore $\boldsymbol{\phi}$ includes only the arm length. A trajectory contains the location and speed of the pendulum during a time period.

In our framework, a neural network $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{y}; \boldsymbol{\eta})$ [Fig. 1(a); see SM [27] S7 for detailed description] replaces $\mathbf{F}_{\boldsymbol{\phi}}(\mathbf{y})$ in Eq. (1) to approximate the observed trajectories, where $\boldsymbol{\eta}$ is adapted to each system instance such that with a certain $\boldsymbol{\eta}_i$, $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{y}; \boldsymbol{\eta}_i)$ approximates the force function of the ith system instance $\mathbf{F}_{\boldsymbol{\phi}_i}(\mathbf{y})$. After training, $\boldsymbol{\eta}$ becomes a proxy for the physical parameters $\boldsymbol{\phi}$. $\boldsymbol{\theta}$ is the model parameters that capture the functional form of dynamics shared across system instances. The predicted trajectory starting from an initial condition $\mathbf{y}_0$ is given by integration (the fifth-order Dormand-Prince-Shampine solver is used throughout this Letter to compute integrals)

$$\hat{\mathbf{y}}(t, \mathbf{y}_0, \boldsymbol{\theta}, \boldsymbol{\eta}) = \mathbf{y}_0 + \int_{t_0}^{t} \mathbf{f}_{\boldsymbol{\theta}}(\hat{\mathbf{y}}(\tau); \boldsymbol{\eta}) d\tau \tag{2}$$

For brevity, we denote the trajectory $\mathbf{y}_{i,j}(t)$ as $\mathbf{y}_{i,j}$, the corresponding prediction $\hat{\mathbf{y}}(t, \mathbf{y}_{i,j}(t_0), \boldsymbol{\theta}, \boldsymbol{\eta})$ as $\hat{\mathbf{y}}_{i,j}(\boldsymbol{\theta}, \boldsymbol{\eta})$,

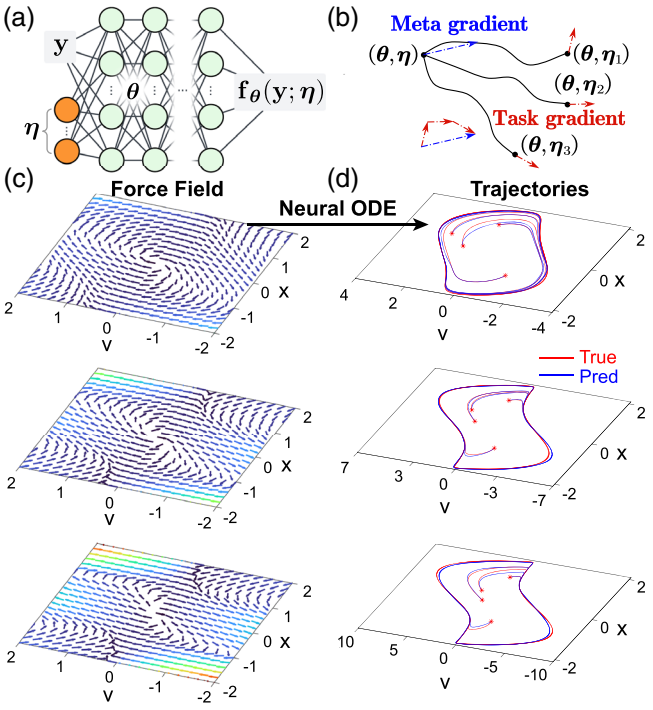

FIG. 1. (a) In iMODE, a neural module $\mathbf{F}_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$ takes the concatenation of system state $\mathbf{y}$ and the adaptation parameters $\boldsymbol{\eta}$ and generates the estimated force as output. (b) The bilevel iteration process in the iMODE method. The NN weights $\boldsymbol{\theta}$ are shared across system instances while $\boldsymbol{\eta}$ is adapted for each instance. The metagradient with respect to $\boldsymbol{\theta}$ aggregates the gradients evaluated with instance-adapted $\boldsymbol{\eta}$. (c) Examples of estimated force field $\mathbf{f}_{\boldsymbol{\theta}}(\cdot; \boldsymbol{\eta})$ for Van der Pol system instances that differ in their ϵ parameter (in ascending order from top to bottom). The estimation quality is further evaluated through the trajectories generated by the fields as shown in (d). (d) The estimated force field can be used to predict system trajectories for unseen initial conditions through integration [Eq. (2)]. The signature limit cycles of Van der Pol systems are faithfully reproduced.

and use $\|\mathbf{y}_{i,j} - \hat{\mathbf{y}}_{i,j}(\boldsymbol{\theta}, \boldsymbol{\eta})\|^2$ to denote $\sum_{k=0}^{T} (\mathbf{y}_{i,j}(t_k) - \hat{\mathbf{y}}(t_k, \mathbf{y}_{i,j}(t_0), \boldsymbol{\theta}, \boldsymbol{\eta}))^2$, the squared difference between $\mathbf{y}_{i,j}$ and $\hat{\mathbf{y}}_{i,j}(\boldsymbol{\theta}, \boldsymbol{\eta})$ across all time steps.

The goal of the modeling is formulated as a bilevel optimization [Fig. 1(b)],

$$\text{outer: } \min_{\boldsymbol{\theta}} \tilde{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{(m)}), \quad \text{where} \tag{3}$$

$$\mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\zeta}) = \frac{1}{N_{tr} T} \sum_{j=1}^{N_{tr}} \|\mathbf{y}_{i,j} - \hat{\mathbf{y}}_{i,j}(\boldsymbol{\theta}, \boldsymbol{\zeta})\|^2, \tag{4}$$

$$\text{inner: } \boldsymbol{\eta}_i^{(l+1)} = \boldsymbol{\eta}_i^{(l)} - \alpha \nabla_{\boldsymbol{\eta}} \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{(l)}), \quad \boldsymbol{\eta}_i^{(0)} = \boldsymbol{\eta} \tag{5}$$

where the inner level involves an m-step gradient descent adapting $\boldsymbol{\eta}$ for each instance, while the outer level finds the

optimal initialization for $\boldsymbol{\theta}$. α is the inner-level stepsize and $\boldsymbol{\eta}_i^{(m)}$ is the adaptation parameters for the ith system instance after m steps of adaptation. For short, we denote such ith adaptation result as $\boldsymbol{\eta}_i$. Note that $\boldsymbol{\eta}_i$ depends on both $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ as shown in Eq. (5). To avoid higher-order derivatives, we simplify such dependency following the first-order model agnostic metalearning (first-order MAML) [20] and use the outer-level step as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\beta}{N_s}\sum_i \nabla_{\boldsymbol{\theta}}\mathcal{L}_i(\boldsymbol{\theta},\boldsymbol{\eta}_i), \quad \left(\text{assuming that } \frac{\partial \boldsymbol{\eta}_i}{\partial \boldsymbol{\theta}} = \mathbf{0}\right) \tag{6}$$

where β is the outer-level stepsize. At both the inner level and outer level, the gradient calculation for functions involving integrals is enabled by NODEs [6,25,26].

As shown in Fig. 1(c), $\mathbf{f}_{\boldsymbol{\theta}}(\cdot;\boldsymbol{\eta})$ specifies a force field that morphs as $\boldsymbol{\eta}$ changes. Note that m is normally quite small (e.g., 5), so given trajectories of a previously unseen system, $\boldsymbol{\eta}$ can be efficiently updated with few gradient steps, adapting the NN to specify a force field explaining behaviors of the new system, which is one order-of-magnitude faster compared to training from scratch [Fig. 3(a)]. Trajectories with arbitrary initial conditions can be inferred based on the force field [Fig. 1(d)]. Generally speaking, there is no restriction to the dimension of $\boldsymbol{\phi}$, i.e., the number of physical parameters of the system. However, as it increases, more system instances (larger training dataset) and hence longer training time of iMODE is expected.

First we validate the modeling capability of the iMODE algorithm on three cases: oscillating pendulum, bistable oscillator, and Van der Pol system (see SM [27] S1 for detailed description). The oscillating pendulum has one physical parameter, i.e., the arm length (rotational inertia normalized). Figure 2(a) shows that the predicted trajectories using task-adapted NNs match the ground truth of each system. Figure 2(b) shows that the learned $\boldsymbol{\eta}$ correlates well with the effective stiffness of the pendulum, i.e., $1/L$. Effectively $\boldsymbol{\eta}$ acts as a proxy of the true arm length and can be used to infer such parameters of unseen systems.

The bistable system has a potential energy function controlled by two parameters, k_1 and k_3. Its potential energy has two local minima, or potential wells. When the initial conditions vary, the bistable system can oscillate intrawell or interwell. Figure 2(c) shows that the task adapted trajectories ($m = 5$) match the ground truth well. Figure 2(d) shows that the identified $\boldsymbol{\eta} \in \mathbb{R}^2$ has two principal axes, along which k_1 and k_3 increases. As mentioned, $\boldsymbol{\eta}$ is effectively a proxy for k_1 and k_3. Later we will show that the mapping from $\boldsymbol{\eta}$ to $\boldsymbol{\phi} = [k_1, k_3]$ can be constructed as a diffeomorphism with NODEs.

The Van der Pol system has three physical parameters $\boldsymbol{\phi} = [\epsilon, \delta, \omega]$. It exhibits limit cycles due to the negative damping for small oscillation amplitudes. Figure 2(e) shows that the evolution of limit cycles due to the change

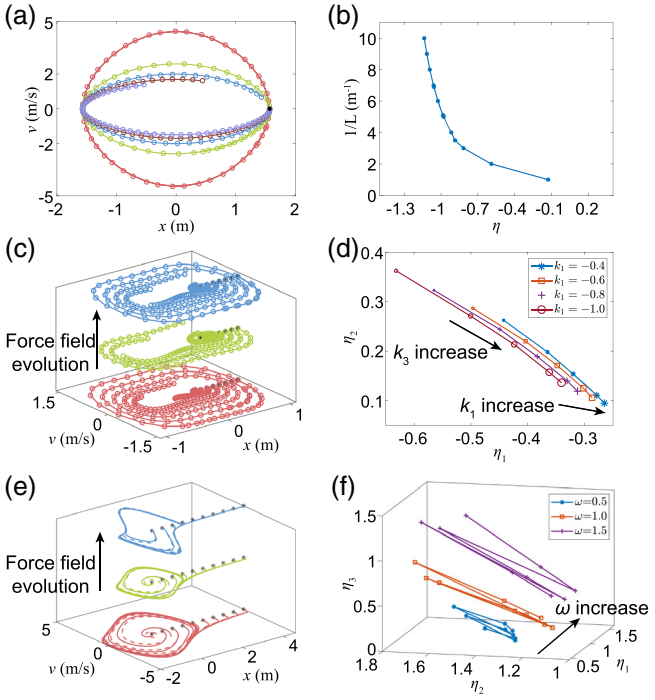

FIG. 2. (a) The metalearning results for the pendulum. The iMODE trajectory prediction (circles) with different arm lengths (different colors) match those of the ground truth (solid lines). (b) The learned $\boldsymbol{\eta}$ is in good correlation with the effective stiffness of different pendulums ($1/L$). (c) The predicted trajectories (circles) match those of ground truth (solid lines) with different initial conditions (black stars) and different system parameters (different colors) for the bistable system. (d) Two principal axes can be identified from the latent space of the learned $\boldsymbol{\eta}$, each regarding the variation of one physical parameter. (e) Similar to (c) but for the Van der Pol system. (f) The principal axis with respect to the variation of ω for the Van der Pol system.

of physical parameters is well predicted. Three principal axes can be found for the identified $\boldsymbol{\eta}$. The one for ω is shown in Fig. 2(f) (see SM [27] S1 for the other two). Again, the mapping from $\boldsymbol{\eta}$ to $\boldsymbol{\phi} = [\epsilon, \delta, \omega]$ can be constructed as a diffeomorphism.

The fast adaptation of iMODE is demonstrated with the bistable systems in Fig. 3(a). The iMODE is able to adjust the adaptation parameters in five steps to learn the dynamics of unseen system instances. Training the same network from scratch (random initialization) on the same test dataset requires much more epochs to achieve a comparable accuracy. When evaluated on trajectories with unseen initial conditions (of distinct conserved energies), the performance of iMODE-adapted models outperforms that of the model trained from scratch by several orders of magnitude, showing superior generalization ability with limited data (see SM [27] S3 for a more disparate comparison when data is scarce).

Second, we demonstrate the combination of the iMODE algorithm with certain physics priors for efficient modeling of more complicated systems. Since iMODE does not

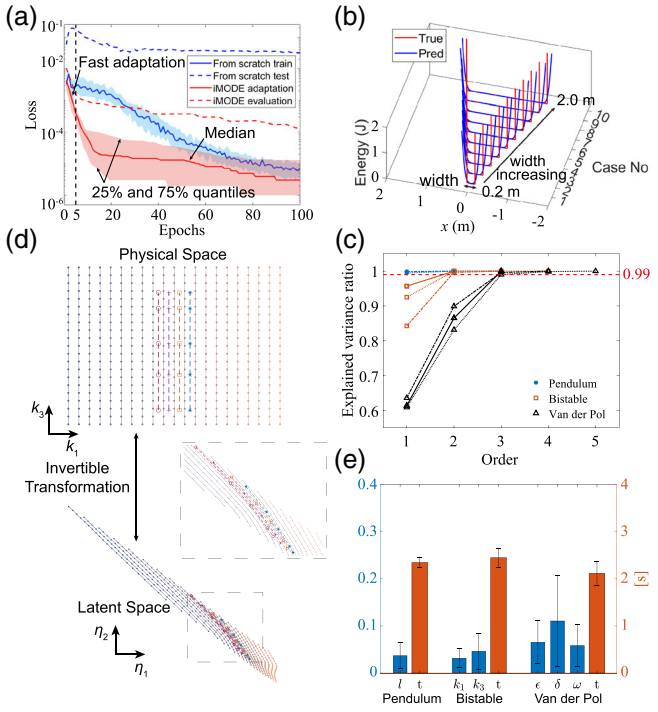

FIG. 3. (a) Comparison of iMODE test adaptation vs training from scratch on (50) unseen bistable system instances with randomly chosen physical parameters. iMODE demonstrates fast adaptation and good generalization within the first five adaptation steps. (b) The true and learned potential energy functions for the wall bouncing system. The width of the potential well increases as the adaptation parameter increases. (c) The number of top PCA components that preserve a significant portion ($> 99\%$) of the variance gives a good estimation of the dimension of true physical parameters. (d) The diffeomorphism constructed by NODE for the bistable system. It shows how a grid in the physical space is continuously deformed into the latent space of adaptation parameters. (e) The mean error and computation time of the neural gauge for 100 systems with randomly generated unseen parameters.

assume specific architecture of $\mathbf{f}_{\boldsymbol{\theta}}$, a wide range of neural network architectures can be adopted to embed appropriate inductive biases. For example, in bistable and the following wall bouncing and Slinky systems, the assumption of conservative force is introduced, where the system dynamics is determined by a potential energy function. Accordingly we take a specific form for the neural force estimator $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x};\boldsymbol{\eta}) = \partial E_{\boldsymbol{\theta}}(\mathbf{x};\boldsymbol{\eta})/\partial \mathbf{x}$. That is, the NN first outputs an energy field and then induces the force field from the energy field (using autodifferentiation [29]). In this way, iMODE enables the fast adaptation of not only the force field but also the potential energy field for the parametric systems. The learned potential energy functions are shown in Fig. 3(b). The wall bouncing system has a potential energy well that is not a linear function of the well's (half-)width w or the particle position x (see SM [27] S2). However, iMODE is still able to approximate the discontinuous energy function. $\boldsymbol{\eta}$ correlates well with the true width w, i.e., we can control the width of the potential energy well by tuning $\boldsymbol{\eta}$ (see SM [27] S2).

The intrinsic dimension $d_{\boldsymbol{\phi}}$ of the physical parameters $\boldsymbol{\phi}$ can be estimated by applying principal component analysis (PCA) to the collection of the $\boldsymbol{\eta}$ vectors, each adapted to one of the system instances. Using an "elbow" method on the cumulative explained variance ratio curve of the PCA result, the number of the principal components that explain the most of the variance has a good correspondence with d_{ϕ}, as long as $d_{\boldsymbol{\eta}} \geq d_{\boldsymbol{\phi}}$, where $d_{\boldsymbol{\eta}}$ is the dimension chosen for $\boldsymbol{\eta}$. The PCA results on the pendulum, bistable system, and Van der Pol system are shown in Fig. 3(c) (see SM [27] S4 for the results of other systems). Taking the Van der Pol system as an example, $d_{\boldsymbol{\eta}}$ is respectively 3, 4, or 5 for the three curves with triangle markers. In all three cases, the first three principal components explain more than 99% of the variance, and the elbow appears at 3, which corresponds well with the fact that $d_{\boldsymbol{\phi}} = 3$ for the Van der Pol system.

Neural gauge.—Without labels for the physical parameters, iMODE develops a latent space of adaptation parameters accounting for the variations in dynamics among system instances. Given the physical parameter labels of the system instances in the training data, a mapping between the space of the physical parameters and the latent space can be established so that the corresponding physical parameters can be estimated given any point in the latent space. iMODE therefore can be exploited as a neural gauge to identify the physical parameters of unseen system instances, and the establishment of such mappings can be seen as a calibration process. We propose to construct such mappings as diffeomorphism, which can be learned with a neural ODE $d\mathbf{z}(t)/dt = \mathbf{g}_{\boldsymbol{\xi}}(\mathbf{z})$, such that starting from a given point in the latent space, $\mathbf{z}(0) = \boldsymbol{\eta}_i$, the state $\mathbf{z}$ at $t = 1$ gives the corresponding physical parameters, $\mathbf{z}(1) = \boldsymbol{\phi}_i$, $i = 1, \ldots, N_s$. For simplicity, the dimension of the latent space and that of the physical parameter space are assumed to match ($d_{\boldsymbol{\eta}} = d_{\boldsymbol{\phi}}$); see SM [27] S5 for more general treatment. $\mathbf{g}_{\boldsymbol{\xi}}$ is a NN whose weights are optimized by $\boldsymbol{\xi} = \arg\min_{\boldsymbol{\xi}} \sum_i \|\mathbf{z}_i(1) - \boldsymbol{\phi}_i\|_2^2$.

Figure 3(d) shows the learned diffeomorphism for the bistable system. The diffeomorphism establishes a bijection between the physical space and the latent space so that a grid in the physical parameter space can be continuously transformed into the adaptation parameter space (see SM [27] S9). The visualization highlights the advantages of diffeomorphism mapping: (1) the transformation is smooth so that the local geometric structure is preserved; (2) invertible transformation allows a better interpretation of the latent space compared to degenerating ones.

After constructing the diffeomorphism, we test the physical parameter identification performance on 100 randomly selected unseen instances (with random physical parameters). The identification error and time cost are shown in Fig. 3(e) for pendulum, bistable, and Van der Pol systems.

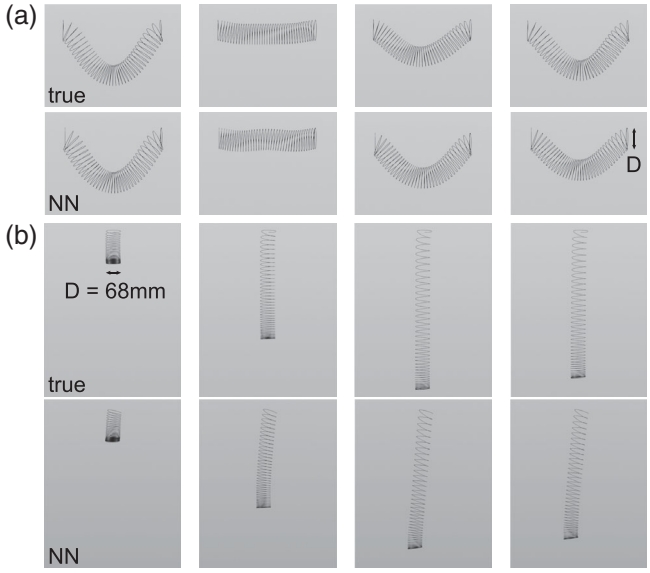

FIG. 4. (a) The testing performance of the iMODE model on an unseen Slinky (of an unseen Young's modulus) with the same boundary condition as the training dataset. The top row is ground truth and the bottom row is the iMODE model prediction at 0.28, 0.47, 0.65, 0.83 s (left to right). The mean squared error of the 3D Slinky reconstruction over the entire trajectory is 8.0×10^{-4} m^2. (b) The testing performance of the iMODE model on unseen initial and boundary conditions. The top row is ground truth and the bottom row is the iMODE model prediction at 0.15, 0.32, 0.48, 0.65 s (left to right). The mean squared error of the 3D Slinky reconstruction over the entire trajectory is 12.6×10^{-4} m^2.

The end-to-end identification starting from data feeding normally takes around 2 s (see SM [27] S8 for details).

Complex systems.—We further demonstrate that iMODE applies to complex systems with two examples: a 40-cycle Slinky (Fig. 4) and a reaction-diffusion system described by the Kolmogorov-Petrovsky-Piskunov (KPP) equation. In the Slinky case, we embed Euclidean invariance for the energy field and induce equivariance for the force field. iMODE is able to learn from four Slinky cases (of Young's modulus 50, 60, 70, and 80 GPa, dropping under gravity from a horizontal initial configuration with both ends fixed) and then quickly generalize (with 2 adaptation steps) to an unseen Slinky (of Young's modulus 56 GPa) under unseen initial and boundary conditions. In the KPP equation case, iMODE is able to learn the reaction term with different reaction strength coefficients in five adaptation steps under Neumann boundary conditions and directly generalize to unseen Dirichlet boundary conditions. Refer to SM [27] S6 for details.

We have presented the iMODE method, i.e., interpretable meta NODE. As a major difference from existing NN-based methods, iMODE learns metaknowledge on a family of dynamical systems, specifically the functional variation of the derivative (force) field. It constructs a parametrized functional form of the derivative field with a shared NN across system instances and latent adaptation parameters adapted for different instances. The NN and adaptation parameters are learned from the difference between the ground truth and the solution calculated by an appropriate ODE solver. We have validated with various examples the generalizability, interpretability, and fast adaptation ability of the iMODE method. iMODE could open new possibilities for numerous potential applications. Two examples are autonomous modeling of dynamical systems where the underlying physics is difficult to express as an explicit function of controllable experiment parameters: (1) the force-deformation constitutive relation of cells as a function of ion concentrations in the culture medium, and (2) agile maneuvering of dynamical systems where the timely knowledge on the interaction between the dynamical system and its external environment is required, such as robotic control in a rapidly changing and partially understood environment.

The following research grants were gratefully acknowledged: NSF (CMMI-2053971) for Q. L., T. W., V. R., and M. K. J.; and NSF (CAREER-2047663, CMMI-2101751, and OAC-2209782) for M. K. J.

*vwani@ee.ucla.edu
†khalidjm@seas.ucla.edu

[1] J. Sprakel, S. B. Lindström, T. E. Kodger, and D. A. Weitz, Stress Enhancement in the Delayed Yielding of Colloidal Gels, Phys. Rev. Lett. **106**, 248303 (2011).
[2] M. K. Jawed, P. Dieleman, B. Audoly, and P. M. Reis, Untangling the Mechanics and Topology in the Frictional Response of Long Overhand Elastic Knots, Phys. Rev. Lett. **115**, 118302 (2015).
[3] R. Alert, A. Martínez-Calvo, and S. S. Datta, Cellular Sensing Governs the Stability of Chemotactic Fronts, Phys. Rev. Lett. **128**, 148101 (2022).
[4] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Physics-informed machine learning, Nat. Rev. Phys. **3**, 422 (2021).
[5] M. Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. **19**, 932 (2018).
[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, Neural ordinary differential equations, Adv Neural Inf Process Syst. **31** (2018).
[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. U.S.A. **113**, 3932 (2016).
[8] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, Data-driven discovery of coordinates and governing equations, Proc. Natl. Acad. Sci. U.S.A. **116**, 22445 (2019).
[9] B. Chen, K. Huang, S. Raghupathi, I. Chandratreya, Q. Du, and H. Lipson, Automated discovery of fundamental variables hidden in experimental data, Nat. Comput. Sci. **2**, 433 (2022).

[10] R. Iten, T. Metger, H. Wilming, L. del Rio, and R. Renner, Discovering Physical Concepts with Neural Networks, Phys. Rev. Lett. **124**, 010508 (2020).
[11] Z. Liu and M. Tegmark, Machine Learning Conservation Laws from Trajectories, Phys. Rev. Lett. **126**, 180604 (2021).
[12] Z. Liu and M. Tegmark, Machine Learning Hidden Symmetries, Phys. Rev. Lett. **128**, 180201 (2022).
[13] K. Lee and E. J. Parish, Parameterized neural ordinary differential equations: Applications to computational physics problems, Proc. R. Soc. A **477**, 20210162 (2021).
[14] S. Desai, M. Mattheakis, H. Joy, P. Protopapas, and S. J. Roberts, One-shot transfer learning of physics-informed neural networks, arXiv:2110.11286.
[15] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv:2003.03485.
[16] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv:2010.08895.
[17] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. **3**, 218 (2021).
[18] S. Wang, H. Wang, and P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. **7**, eabi8605 (2021).
[19] H. Wang, H. Zhao, and B. Li, Bridging multi-task learning and metalearning: Towards efficient training and effective adaptation, in *Proceedings of the 38th International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2021), pp. 10991–11002.
[20] C. Finn, P. Abbeel, and S. Levine, Model-agnostic metalearning for fast adaptation of deep networks, in *Proceedings of the International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2017), pp. 1126–1135.
[21] A. Nichol, J. Achiam, and J. Schulman, On first-order metalearning algorithms, arXiv:1803.02999.
[22] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine, Online metalearning, in *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research Vol. 97, edited by K. Chaudhuri and R. Salakhutdinov (Proceedings of Machine Learning Research, 2019), pp. 1920–1930.
[23] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, Metalearning with implicit gradients, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2019), Vol. 32.
[24] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, Rapid learning or feature reuse? Towards understanding the effectiveness of MAML, arXiv:1909.09157.
[25] R. T. Q. Chen, B. Amos, and M. Nickel, Learning neural event functions for ordinary differential equations, *International Conference on Learning Representations* (2021).
[26] Q. Li, T. Wang, V. Roychowdhury, and M. Jawed, Rapidly encoding generalizable dynamics in a Euclidean symmetric neural network, Extreme Mech. Lett. **58**, 101925 (2023).
[27] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevLett.131.067301 for details on iMODE training, wall bouncing system, double pendulum system, dimension determination of latent space, demonstration on complex systems, and movies on diffeomorphism, which includes Ref. [28].
[28] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, Densely connected convolutional networks, in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 2261–2269.
[29] A. Paszke *et al.*, PyTorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035.

Meta-learning generalizable dynamics from trajectories – Supplemental Information –

Qiaofeng Li, Tianyi Wang, Vwani Roychowdhury, and M.K. Jawed

S1 Testing performance of the iMODE method

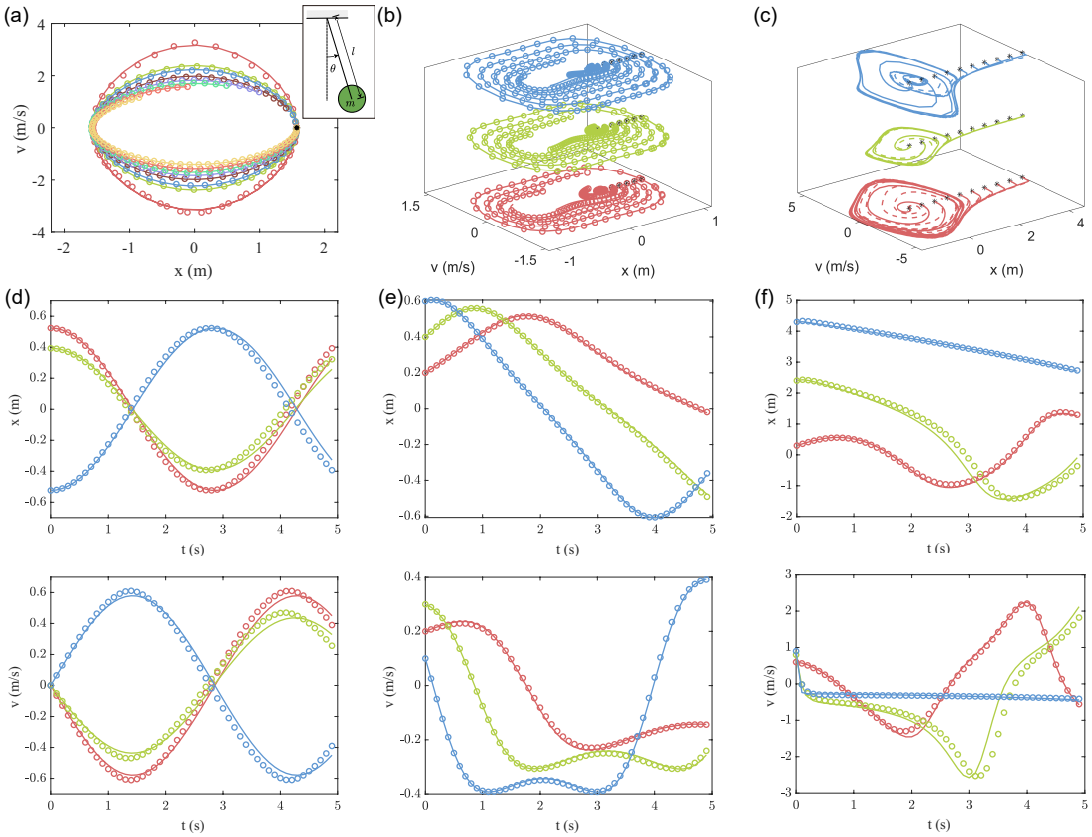

Figure S1: (color online). The testing performance of (a) the pendulum system, (b) the bistable system, (c) the Van der Pol system with unseen physical parameters. The solid lines are ground truth. In (a) and (b) the circles, in (c) the dashed lines, are predictions of corresponding iMODE models. Different colors represent different parametric systems. (d), (e), and (f) show the testing performance of the pendulum, bistable, and Van der Pol systems with unseen initial conditions. The first row in (d), (e), and (f) plots system coordinate v.s. time with 3 unseen initial conditions; the second row plots the corresponding system velocity v.s. time. Solid lines and circles represent ground truth and iMODE predictions.

1.1 Oscillating pendulum

$$ml^2\ddot{\theta} + mgl\sin(\theta) = 0 \quad \text{s.t.} \quad \theta(0) = \theta_0,\ \dot{\theta}(0) = \dot{\theta}_0 \tag{1}$$

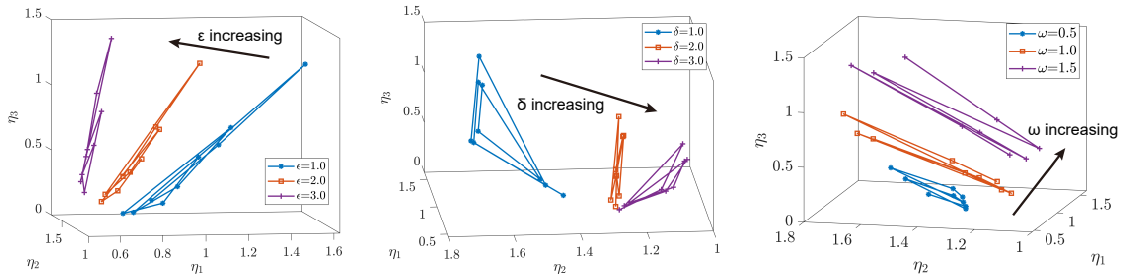

Figure S2: (color online). The three variation directions in the latent space for the physical parameters of the Van der Pol system ϵ, δ, and ω.

The training dataset contains 5 system instances with $l = [1, 3, 5, 7, 9]$ m. A long trajectory of 10 s is generated for each instance with the initial position and velocity $\pi/2$ and 0 s^{-1} and time marching stepsize 10 ms. During training, a batch of 20 trajectories of 1 s are pulled out randomly in each epoch. So essentially the iMODE training is seeing 1 s trajectories with differential initial conditions.

The learnt iMODE model is tested on 8 unseen system instances with $l = [2, 3.5, 4, 5.1, 6, 6.9, 8, 10]$ m. The task adaptation is similarly done as the training, i.e., seeing a batch of 20 randomly pulled-out trajectories of 1 s for each testing system instance. The task adaptation only takes 5 steps. Then the learnt model for each instance is used to calculate a trajectory of 5 s given an initial condition, and compared with ground truth. The results are shown in Fig. S1(a). The solid lines (the ground truth) match well with the circles (prediction).

1.2 Bistable oscillator

$$\ddot{x} + k_1 x + k_3 x^3 = 0 \quad s.t. \quad x(0) = x_0,\ \dot{x}(0) = \dot{x}_0 \tag{2}$$

The training dataset contains 20 system instances, a mesh of $k_1 = [-0.4, -0.6, -0.8, -1.0]$ and $k_3 = [2.0, 2.9, 3.7, 4.6, 5.0]$. Trajectories of multiple initial conditions with stepsize 10 ms and time span 10 s are generated for each instance. During training, a batch of 100 randomly pulled-out trajectories of 1 s is used for each epoch. During testing, task adaptation takes 5 steps on previously unseen systems $[k_1, k_3] = [-0.5, 3.1], [-0.7, 4.2], [-0.5, 4.7]$. The learnt models calculate trajectories of 5 s given an initial condition. The results are shown in Fig. S1(b).

1.3 Van der Pol system

$$\ddot{x} - \epsilon\dot{x}(1 - \delta x^2) + \omega^2 x = 0 \quad s.t. \quad x(0) = x_0,\ \dot{x}(0) = \dot{x}_0 \tag{3}$$

The training dataset contains 27 system instances, a mesh of $\epsilon = [1.0, 2.0, 3.0]$, $\delta = [1.0, 2.0, 3.0]$, and $\omega = [0.5, 1.0, 1.5]$. Trajectories of multiple initial conditions with stepsize 10 ms and time span 10 s are generated for each instance. During training, a batch of 100 randomly pulled-out trajectories of 1 s is used for each epoch. During testing, task adaptation takes 5 steps on previously unseen systems instances $[\epsilon, \delta, \omega] = [1.2, 1.2, 2.1], [1.2, 1.8, 1.4], [2.6, 1.5, 2.5]$. The learnt models calculate trajectories of 5 s given an initial condition. The results are shown in Fig. S1(c).

The 3 variation directions in the latent space $\boldsymbol{\eta} \in \mathbb{R}^3$ for ϵ, δ, and ω are shown in Fig. S2.

1.4 New initial conditions

The comparison between the actual and predicted trajectories for previously encountered system instances and new initial conditions are given in Fig. S1(d), (e), and (f). The first and second rows show position and velocity trajectories of 3 unseen initial conditions. The physical parameters for the pendulum, bistable, and Van der Pol systems are $l = [7]$, $[k_1, k_3] = [-0.6, 5.0]$, and $[\epsilon, \delta, \omega] = [1.0, 2.0, 1.5]$.

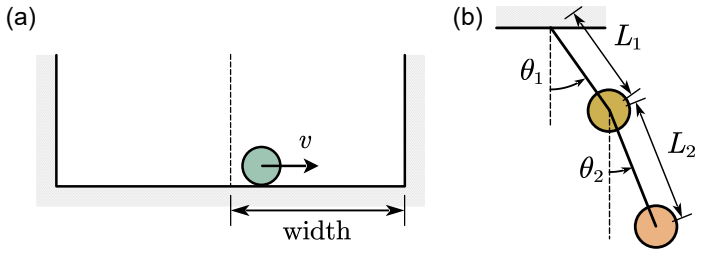

Figure S3: (color online). (a) The wall bouncing system. (b) The double pendulum system.

S2 Other systems

2.1 Wall bouncing system

The governing equation for the wall bouncing system (Fig. S3(a)) is

$$\begin{aligned} \ddot{x}+F(x)=0 \quad s.t. \quad x(0)=x_0,\ \dot{x}(0)=\dot{x}_0 \\ F(x)=\begin{cases} -k(x-w), & x \geq w \\ 0 \qquad\quad , & |x|<w \\ -k(x+w), & x \leq -w \end{cases} \end{aligned} \tag{4}$$

x and v are the particle position and velocity, $k = 1000$ N/m is a large constant to approximate a stiff wall, w is the (half-)width of the potential well, as shown in Fig. S3(a). The system has a potential energy well with the following form

$$E(x)=\begin{cases} 0, & |x|<w \\ \infty, & |x| \geq w \end{cases} \tag{5}$$

The training dataset contains 10 system instances, with the width increasing from 0.1 m to 1.0 m by 0.1 m. Trajectories of multiple initial conditions (initial position 0 m, and initial velocities ranging from 0.1 m/s to 1.0 m/s) with stepsize 10 ms and time span 10 s are generated for each instance. During training, a batch of 100 randomly pulled-out trajectories of 1 s is used for each epoch.

In this case the intermediate output of the NN is the energy and the force is derived by taking the derivative of the output with respect to the input, i.e.

$$F=\frac{\partial E}{\partial x}=\frac{\partial(\mathrm{NN}_{\boldsymbol{\theta}}(x;\eta)+\mathrm{NN}_{\boldsymbol{\theta}}(-x;\eta))}{\partial x} \tag{6}$$

The second equality takes advantage of the assumption that the energy is symmetric with respect to x. The learning results show $\eta \in \mathbb{R}$ to be in perfect correlation with the width w of the potential well (Fig. S4). In other words, we can control the width of the constructed potential well of the NN, which is another way to interpret the physical meaning of the adaptation parameter η.

2.2 Double pendulum

The double pendulum, as shown in Fig. S3(b), has two masses $m_1 = m_2 = 1$ kg and arm lengths L_1 and L_2. The governing equations are

$$\begin{aligned} \dot{\theta}_1 &= \omega_1 \\ \dot{\theta}_2 &= \omega_2 \\ \dot{\omega}_1 &= \frac{-g\left(2m_1+m_2\right)\sin\theta_1 - m_2 g\sin\left(\theta_1-2\theta_2\right) - 2\sin\left(\theta_1-\theta_2\right)m_2\left(\omega_2^2 L_2+\omega_1^2 L_1\cos\left(\theta_1-\theta_2\right)\right)}{L_1\left(2m_1+m_2-m_2\cos\left(2\theta_1-2\theta_2\right)\right)} \\ \dot{\omega}_2 &= \frac{2\sin\left(\theta_1-\theta_2\right)\left(\omega_1^2 L_1\left(m_1+m_2\right)+g\left(m_1+m_2\right)\cos\theta_1+\omega_2^2 L_2 m_2\cos\left(\theta_1-\theta_2\right)\right)}{L_2\left(2m_1+m_2-m_2\cos\left(2\theta_1-2\theta_2\right)\right)} \end{aligned} \tag{7}$$

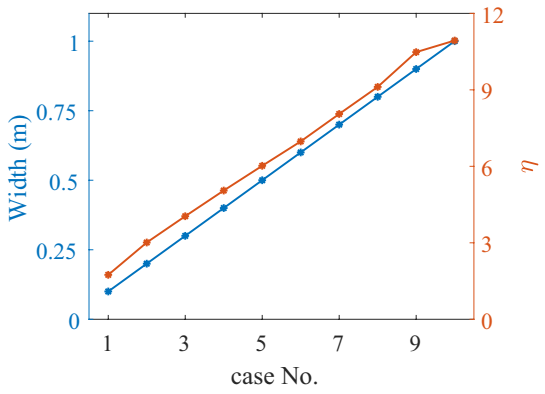

Figure S4: (color online). The true widths of the wall bouncing system and the learnt adaptation parameters η are in perfect correlation (99.87%).

The training dataset contains 16 system instances, a mesh of $L_1 = [0.5, 0.6, 0.7, 0.8]$ m and $L_2 = [0.5, 0.6, 0.7, 0.8]$ m. Trajectories of initial locations [π/4,π/4] and initial velocities [0,0] with stepsize 10 ms and time span 10 s are generated for each system. During training, a batch of 100 randomly pulled-out trajectories of 1 s is used for each epoch. Task adaptation takes 5 steps. The learnt latent space of adaptation parameters is shown in Fig. S5. It is clear that two directions exist corresponding to the variation of physical parameters L_1 and L_2. This again underlines the interpretability of $\boldsymbol{\eta} \in \mathbb{R}^2$.

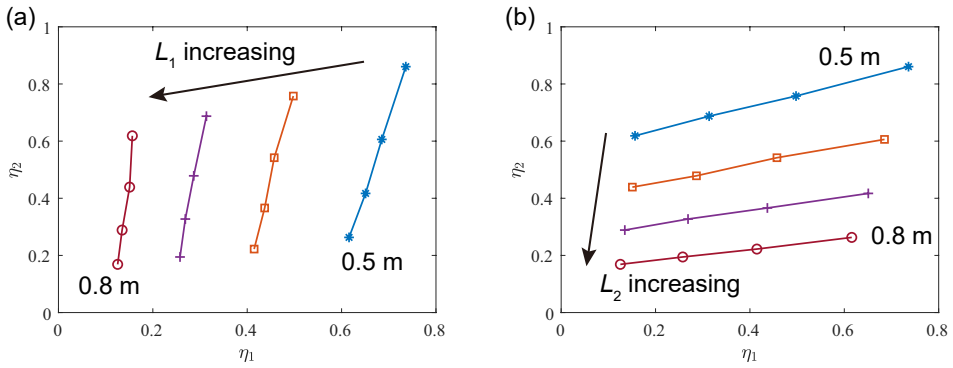

Figure S5: (color online). The learnt latent space of adaptation parameters for the double pendulum system. (a) Each marked line shows systems with the same L_1. (b) Each marked line shows systems with the same L_2. There are clearly two directions in the latent space (indicated by the arrows) corresponding to the change of physical parameters L_1 and L_2.

S3 Comparison between iMODE and training from scratch

We compare the performance of iMODE adaptation to the "training from scratch" (TFS) approach. The iMODE adaptation starts with a weight initialization trained from a training dataset. It updates the adaptation parameter $\boldsymbol{\eta} \in \mathbb{R}^2$ on a testing dataset, which is not included in the training dataset. The TFS approach uses the same NN architecture and hyperparameters as in the iMODE. The TFS NN is randomly initialized and all the weights are updated on the same testing dataset. After training the TFS NN and adapting the iMODE NN using the same testing dataset, the two NNs are evaluated on an unseen evaluation dataset.

Note that while comparing iMODE and TFS we have used the terms "testing dataset" (used to train the two models) and "evaluation dataset" (used to evaluate the trained models). This has been done to remind the reader that there is a training dataset unseen by the TFS NN. In other words, the TFS NN and iMODE NN are both trained/adapted on the same testing dataset, and as the

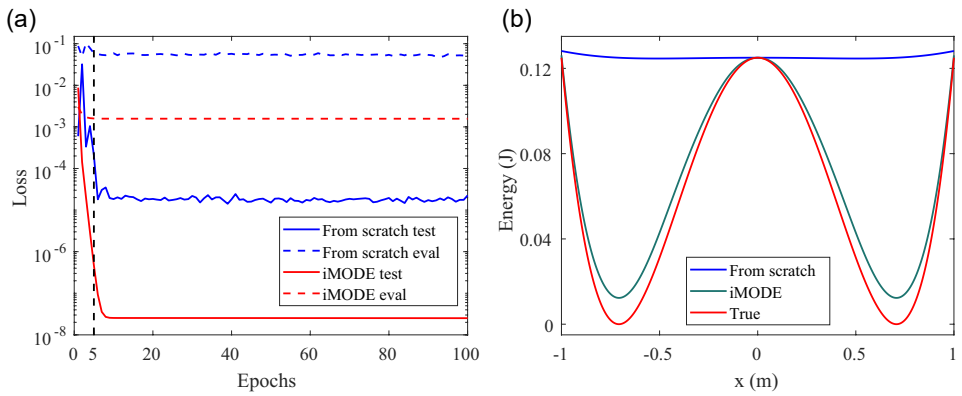

Figure S6: (color online). (a) The training/adaptation and evaluation performance of the TFS and iMODE NNs, given a single trajectory of the bistable system with physical parameters $k_1 = -1.0$ and $k_3 = 2.0$, and initial condition $x_0 = 0.7$ m and $\dot{x}_0 = 0$ m/s. (b) The iMODE learns the correct double-well potential energy function while the TFS approach learns nothing due to data scarcity.

training/adaptation proceeds, tested on the same evaluation dataset (trajectory prediction from given initial conditions). The only difference between the TFS NN and iMODE NN is that the weights of the former are randomly initialized while those of the latter are optimized based on a training dataset obtained previously.

As shown in Fig. 3(a), the iMODE significantly outperforms the TFS approach in terms of adaptation speed (v.s. training speed in the TFS approach) and evaluation accuracy. This means that the iMODE approach can learn the dynamics of an unseen system more rapidly and predict future events more accurately than a TFS NN. This observation is pronounced in the following case: we feed these two NNs a single trajectory of the bistable system with physical parameters $k_1 = -1.0$ and $k_3 = 2.0$, and initial condition $x_0 = 0.7$ m and $\dot{x}_0 = 0$ m/s. After training/adaptation, we evaluate the TFS and iMODE NNs on trajectories of the same system but with different initial conditions (of distinct conserved energies). The training/adaptation and evaluation curves are shown in Fig. S6(a). The iMODE outperforms the TFS approach in both training/adaptation and evaluation accuracy. The learnt energy functions of both approaches are compared in Fig. S6(b). The energy function of the TFS NN is totally incorrect due to the data scarcity. Under this specific initial condition, the bistable system is only oscillating intra-well. So the information contained in the trajectory is insufficient to depict the entire potential energy surface. Meanwhile the iMODE NN learns an accurate double-potential-well function from the same data because appropriate prior knowledge on the energy functions of bistable systems (i.e. double-well) is already embedded in its weight initialization.

We emphasize that in terms of the amount of data seen, this comparison seems unfair to the TFS NN. This comparison is to demonstrate that iMODE is able to mine and incorporate the inherent knowledge from an existing dataset (the training dataset), and transfer it onto a new dataset (the testing dataset) rapidly and effectively, especially in the scenario of scarce data. The TFS approach, without the ability to leverage existing dataset when no labels on physical parameters are available, has to rely only on the new dataset (the testing dataset) and cannot perform comparably with scarce data.

S4 Dimension determination of physical parameters with PCA

The workflow of using PCA to determine the optimal dimension d of the adaptation parameters $\boldsymbol{\eta}$ for a parametric system is: (1) Make a rough guess $\tilde{d}$ on the dimension, then run the iMODE algorithm on the trajectories of N_s systems; (2) Form a matrix with the results $M = [\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_{N_s}]$; (3) Perform PCA on M. A significant portion of variance (e.g. 99%) will be preserved in the first $\hat{d}$ dimensions, an estimation for d; (4) repeat the process with different initial guesses $\tilde{d}$. The optimal dimension is more credible when different $\tilde{d}$ results in the same $\hat{d}$.

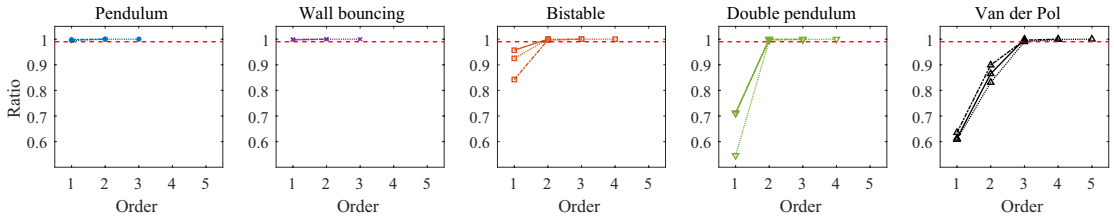

Figure S7: (color online). PCA can be used to determine the optimal dimensions of the adaptation parameters $\boldsymbol{\eta}$ in the studied systems. These optimal dimensions prove to equal the true dimension $d_{\boldsymbol{\phi}}$ of the physical parameters $\boldsymbol{\phi}$ of the systems. The red dashed line indicates 0.99.

The PCA determination results of all systems are shown in Fig. S7. The red dashed lines mark the 99% variance preservation. Dotted lines in each case mean that the initial guess $\tilde{d}$ is the dimension of real physical parameters $d_{\boldsymbol{\phi}}$ plus 2. For example, in the Van der Pol system, the dotted line means that the initial guess $\tilde{d} = 5$. After the PCA, if we preserve 4 or 3 principal components, the variance energy is still preserved by more than 99%. If we further reduce the number of preserved principal components to 2 or 1, we see a sudden drop (to below the 99% threshold), which indicates the optimal dimension to be 3. The $\tilde{d}$ for dashed-dotted and solid lines are the dimension of real physical parameters $d_{\boldsymbol{\phi}}$ plus 1 and 0 respectively. With different $\tilde{d}$, we can repeatedly confirm the optimal dimension of $\boldsymbol{\eta}$, to be 3 in the case of Van der Pol system (which is the true dimension of physical parameters). For other systems, the workflow is the same.

S5 *Neural Gauge* diffeomorphism

As suggested by the PCA analysis in Section S4, the adaptation parameters $\{\boldsymbol{\eta}_i\}_{i=1}^{N_s}$ that are adapted to the system instances belonging to a family of dynamical systems occupy a $d_{\boldsymbol{\phi}}$-dimensional manifold, even if the latent space they reside in is $d_{\boldsymbol{\eta}}$-dimensional and $d_{\boldsymbol{\eta}} \geq d_{\boldsymbol{\phi}}$. Therefore, a diffeomorphism can be established mapping $\boldsymbol{\eta}$ in the latent space to their corresponding physical parameters $(\{\boldsymbol{\phi}_i\}_{i=1}^{N_s})$ even if their dimensions do not match, assuming $d_{\boldsymbol{\eta}} \geq d_{\boldsymbol{\phi}}$. Practically, the neural ODE modelling such diffeomorphism can be defined as $\mathrm{d}\mathbf{z}(t)/\mathrm{d}t = \mathbf{g}_{\boldsymbol{\xi}}(\mathbf{z})$, such that for $i = 1, \ldots, N_s$, starting from a given point in the latent space, $\mathbf{z}(0) = \boldsymbol{\eta}_i$, the state $\mathbf{z}$ at $t = 1$, $\mathbf{z}(1) = \begin{bmatrix}\boldsymbol{\phi}_i^{\mathrm{T}} & 0 & \ldots & 0\end{bmatrix}^{\mathrm{T}}$ is the concatenation of corresponding physical parameters and $d_{\boldsymbol{\eta}} - d_{\boldsymbol{\phi}}$ padding zeros.

S6 Complex cases

6.1 Slinky: the Euclidean symmetric neural network

The NN used in the Slinky case follows the Euclidean symmetric neural network (ESNN) (see Ref. [26] of main manuscript) architecture. The Slinky is decomposed into 40 consecutive triplets, i.e., the 2D representation of 3 adjacent cycles. We denote the coordinates of the ith triplet as $\boldsymbol{\xi}_i = \left[\mathbf{x}_{i-1}^{\mathrm{T}}, \mathbf{x}_i^{\mathrm{T}}, \mathbf{x}_{i+1}^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^9$. $\mathbf{x}_i \in \mathbb{R}^3$ is the coordinates of the ith bar, including the x and y coordinates of the bar center and the inclination angle of the bar. The potential energy associated with the middle bar of a triplet is only a function of the coordinates of the 3 bars of the same triplet (and of the adaptation parameters), i.e.

$$E_i = E_i(\boldsymbol{\xi}_i; \boldsymbol{\eta}) \tag{8}$$

In the following we will omit the subscript i for brevity. The induced force $\mathbf{F}$ from E is

$$\mathbf{F} = \frac{\partial E}{\partial \mathbf{x}_i} = \frac{\partial E(\mathbf{z}; \boldsymbol{\eta})}{\partial \mathbf{x}_i} = \frac{\partial E(\boldsymbol{\xi}; \boldsymbol{\eta})}{\partial \mathbf{x}_i} \tag{9}$$

where $\mathbf{z} \in \mathbb{R}^6$ is the relative coordinates between the bars of the ith triplet. We enforce Euclidean invariance, i.e., translational, rotational, and chiral invariance, on E with respect to $\boldsymbol{\xi}$, by taking $E(\mathbf{z})$

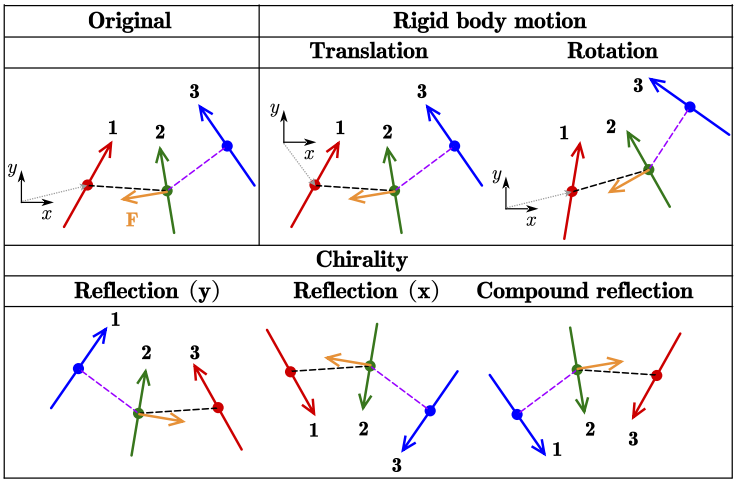

Figure S8: (color online). The Euclidean invariance on energy and induced equivariance on force of the NN used in the Slinky system case. All the Euclidean transformed configurations have the same energy as the original configuration. The elastic forces on the middle bars are Euclidean-transformed accordingly.

the following form, i.e., the ESNN

$$E(\mathbf{z};\boldsymbol{\eta}) = \mathrm{NN}_{\boldsymbol{\theta}}(\mathbf{z};\boldsymbol{\eta}) + \mathrm{NN}_{\boldsymbol{\theta}}\left(R_x(\mathbf{z});\boldsymbol{\eta}\right) + \mathrm{NN}_{\boldsymbol{\theta}}\left(R_y(\mathbf{z});\boldsymbol{\eta}\right) + \mathrm{NN}_{\boldsymbol{\theta}}\left(R_x\left(R_y(\mathbf{z})\right);\boldsymbol{\eta}\right) \tag{10}$$

where $R_x(\cdot)$ and $R_y(\cdot)$ denote reflection with respect to the x and y axes. Note that

$$R_x\left(R_x(\cdot)\right) = I(\cdot),\ R_y\left(R_y(\cdot)\right) = I(\cdot),\ \text{and}\ R_x\left(R_y(\cdot)\right) = R_y\left(R_x(\cdot)\right) \tag{11}$$

where $I(\cdot)$ is the identity operation. It is easy to prove the chiral invariance of E, i.e.,

$$E(\mathbf{z};\boldsymbol{\eta}) = E\left(R_x(\mathbf{z});\boldsymbol{\eta}\right) = E\left(R_y(\mathbf{z});\boldsymbol{\eta}\right) = E\left(R_x\left(R_y(\mathbf{z})\right);\boldsymbol{\eta}\right) \tag{12}$$

Then from Eq. (9), $\mathbf{F}$ is equivariant to rigid body and chiral transformations on $\boldsymbol{\xi}$, as shown in Fig. S8, including translation, rotation, and reflection, regardless of $\boldsymbol{\eta}$. The ESNN will be applied on each triplet in the Slinky to calculate the elastic force acting on each bar. The assembled force vector is used to update the system state inside the NODE framework. The difference between the true and predicted trajectories is used to update the ESNN weights $\boldsymbol{\theta}$. After training and performing trajectory predictions in 2D, a geometric method can be used to reconstruct the 3D Slinky configurations. See Ref. [26] of main manuscript for more implementation details.

The training dataset contains 4 Slinkies of different Young's modulus (50, 60, 70, and 80 GPa). The Slinkies are clamped at both ends and freely drop under gravity from a horizontal initial configuration. Two inner steps are taken to update $\eta \in \mathbb{R}$ for each Slinky. Note when η is updated, the NN always preserves energy invariance and force equivariance with respect to the coordinates of the Slinky. After training the NN, we perform task adaptation (2 steps) on a unseen Slinky of Young's modulus 56 GPa and observe a good fitting result (Fig. 4(a)). The resulting NN is then directly applied to computation under an unseen boundary condition and Slinky orientation (the bottom end is free and the Slinky drops under gravity from a vertical initial configuration) without any modification (Fig. 4(b)). We can achieve this because the model-agnostic nature of the iMODE method allows us to embed the Euclidean symmetries into the NN.

6.2 Kolmogorov-Petrovsky-Piskunov (KPP) equation

To solve the KPP equation, we discretize the spatial domain [0,1] into 20 segments. So the the partial differential equation system (here x denotes the spatial coordinate)

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ru(1-u) \tag{13}$$

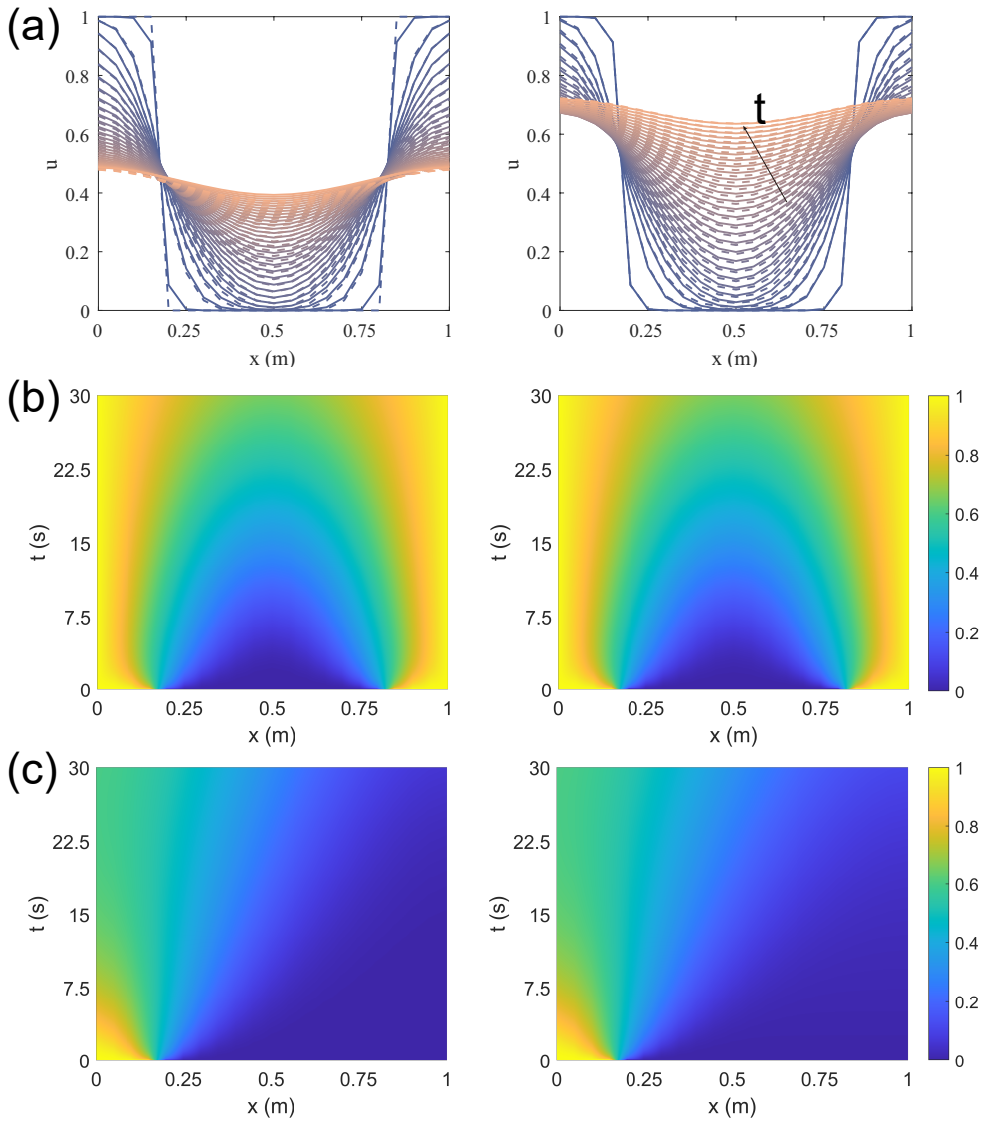

Figure S9: (color online). (a) The training results of the iMODE algorithm on the KPP system for $r = 0.01$ (left) and $r = 0.05$ (right). Solid lines are ground truth. Dashed lines are iMODE predictions. The arrow indicates time marching of u. (b) The iMODE testing results on an unseen system $r = 0.034$ with an unseen boundary condition. The ground truth (left) match the iMODE prediction (right) well. (c) The iMODE testing results ($r = 0.034$) on an unseen initial condition from the training dataset. The ground truth (left) match the iMODE prediction (right) well.

is represented by an ordinary differential equation system containing 21 variables. The diffusion term is approximated by 2nd-order central difference and the diffusivity is assumed known. The meta-learning is performed to learn the reaction term with different reaction strength coefficients r (without knowing the mathematical form). The training dataset contains 5 systems with $r = 0.01,\ 0.02,\ 0.03,\ 0.04,\ 0.05$. The Neumann boundary condition $u'(0) = u'(1) = 0$ ($'$ denotes derivative with respect to x) is used across the training dataset. The iMODE task adaptation takes 5 iterations. The training results for $r = 0.01,\ 0.05$ are shown in Fig. S9(a). The iMODE NN is then adapted on the data from a unseen system instance with $r = 0.034$. The resulted NN is directly applied to computation with unseen initial and boundary conditions (Dirichlet type $u(0) = u(1) = 1$). The results of the latter are shown in Fig. S9(b) and a good agreement is observed. This again validates the capability of the iMODE algorithm to fast adapt on unseen complex parametric systems and accurately predict on initial and boundary conditions different from those in the training dataset.

Another testing result for the KPP system is shown in Fig. S9(c). The testing has the same type of boundary condition ($u'(0) = u'(1) = 0$) as the training dataset but an unseen initial condition. The prediction (right) matches the ground truth (left) well.

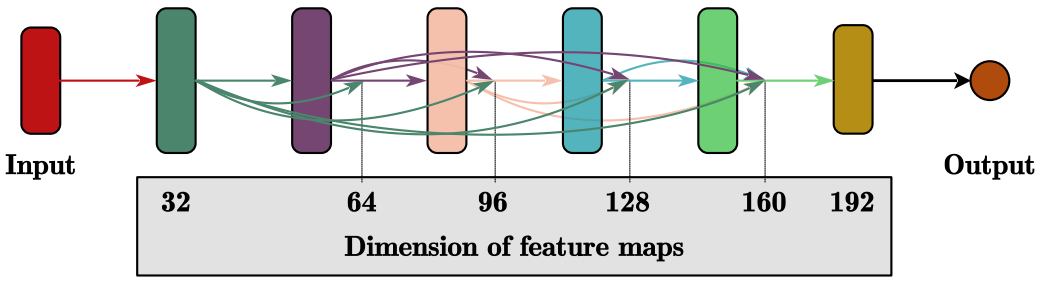

Figure S10: (color online). The DenseNet-like structure.

S7 Neural network architecture

Throughout this letter, we use a DenseNet-like architecture (see Ref. [29] of main manuscript) for our neural networks (NNs), where shortcut pathways are created for a layer from all its previous layers. It takes in the input and first increases the feature dimension to 32 by a fully-connected (FC) layer. Then the feature is passed through FC layers with Softplus activation. A new feature with an increased dimension is formed by concatenating the previous feature with the FC layer output, i.e.,

$$\begin{aligned}
&\mathbf{f}_i = \begin{bmatrix} \mathrm{FC}(\mathbf{f}_{i-1}) \in \mathbb{R}^{32} \\ \mathbf{f}_{i-1} \end{bmatrix}, \quad i = (1, 2, \ldots, 5) \\
&\mathbf{f}_0 = \mathrm{FC}(\mathrm{Input}) \in \mathbb{R}^{32} \\
&\mathrm{Output} = \mathrm{FC}(\mathbf{f}_5)
\end{aligned} \tag{14}$$

where $\mathbf{f}_i$ is the feature map for the ith layer. After passing through 5 densely connected layers, the feature dimension is increased to 192. This feature is then passed through a FC layer with no activation to produce the final output.

For pendulum, bistable, wall bouncing, and Slinky systems, the NN input is the vector concatenating the system position $\mathbf{x}$ and the adaptation parameter $\boldsymbol{\eta}$, i.e. $[\mathbf{x}^{\mathrm{T}}, \boldsymbol{\eta}^{\mathrm{T}}]^{\mathrm{T}}$. The output is a scalar, i.e. the energy of the system. The force vector is calculated by back-propagating the NN output with respect to $\mathbf{x}$. For Van der Pol system, the input is the vector concatenating the system state $\mathbf{y}$ and $\boldsymbol{\eta}$, i.e. $[\mathbf{y}^{\mathrm{T}}, \boldsymbol{\eta}^{\mathrm{T}}]^{\mathrm{T}}$. The output is the force vector. For KPP system, the input is $[u, \eta]$. The output is the reaction forcing term.

S8 Numerical cases runtimes

	training epochs	training time (s)	adaptation steps	adaptation time (s)
Pendulum	200	1031.67	5	1.21
Bistable	200	8317.95	5	1.96
Van der Pol	200	5616.90	5	1.04
Wall bouncing	100	3314.49	5	5.49
Double pendulum	200	3290.17	5	1.09
Slinky	300	79528.42	2	74.16
KPP	500	5448.72	5	0.91

All the numerical cases are run on a NVIDIA RTX 2080 Ti GPU.

S9 Supplementary movie

Movie S1. The diffeomorphism for the bistable system. The data points are transformed from the physical space (subtracted mean) to the latent space of adaptation parameters (subtracted mean). The right subplot is the enlarged view of the left plot.

Movie S2. The diffeomorphism for the Van der Pol system. The data points are transformed from the physical space (subtracted mean) to the latent space of adaptation parameters (subtracted mean). The right subplot is the enlarged view of the left plot.