Dynamic System and Optimal Control Perspective of Deep Learning(Part II)

Tijin Yan

2020.12.27

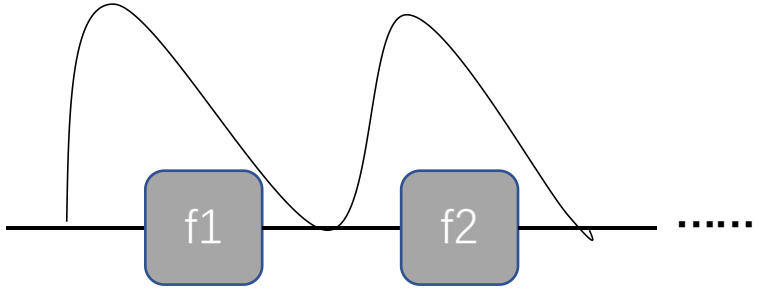
Outline

- **DNN & ODE**
- **Stability Analysis with Mean Field Theory**
- **DNN & Physical Systems**

DNN & ODE

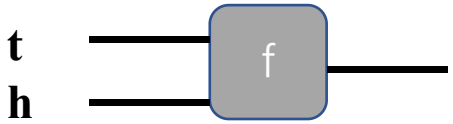
Neural ODE

$$h_{t+1} = h_t + f(h_t, \theta_t)$$



Hypothesis: replace layers with δt

$$\frac{\partial h(t)}{dt} = \mathrm{f}(\mathrm{h}(\mathrm{t}),\ \mathrm{t},\ \theta)$$



Fewer Parameters;
Variety of Numerical Integration Methods

Difficulties: How to train? ⟶ **Adjoint Sensitivity Method**

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{h}(t)}$$

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

def aug_dynamics$([\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta)$: ▷ Define dynamics on augmented state

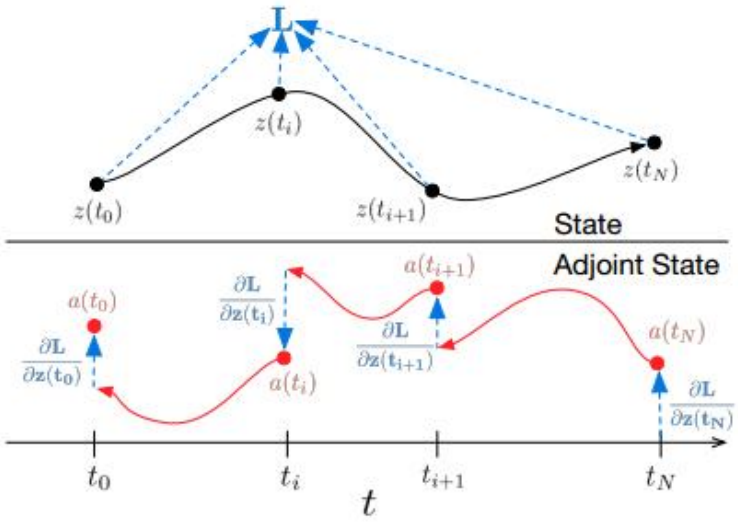
  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \mathrm{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

Chen, Tian Qi, et al. "Neural Ordinary Differential Equations."NeurIPS2018 (best paper)

Neural ODE



	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\mathcal{O}(1)$	$\mathcal{O}(\tilde{L})$

L: Layers of ResNet; \hat{L}: NFE

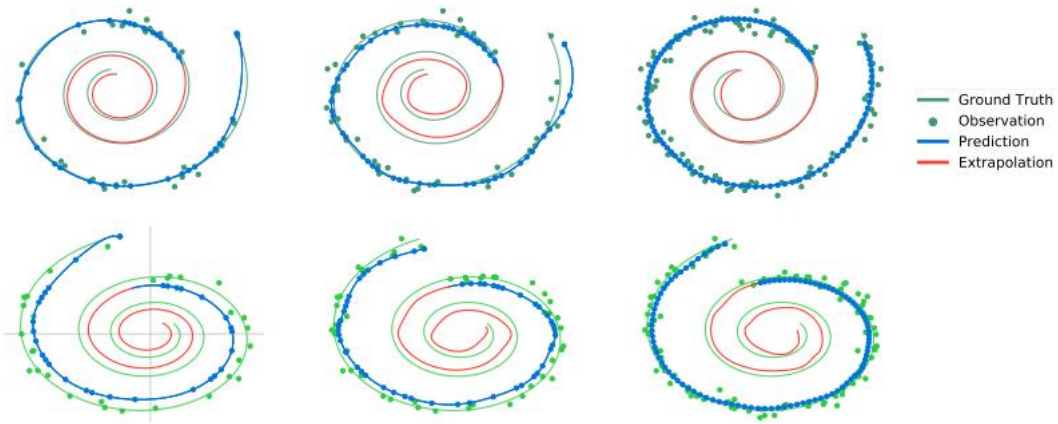
Memory Efficient; Longer Training Time

$$\log p(x) = \log p(z) - \log \left| \det \frac{\partial f}{\partial z} \right|$$

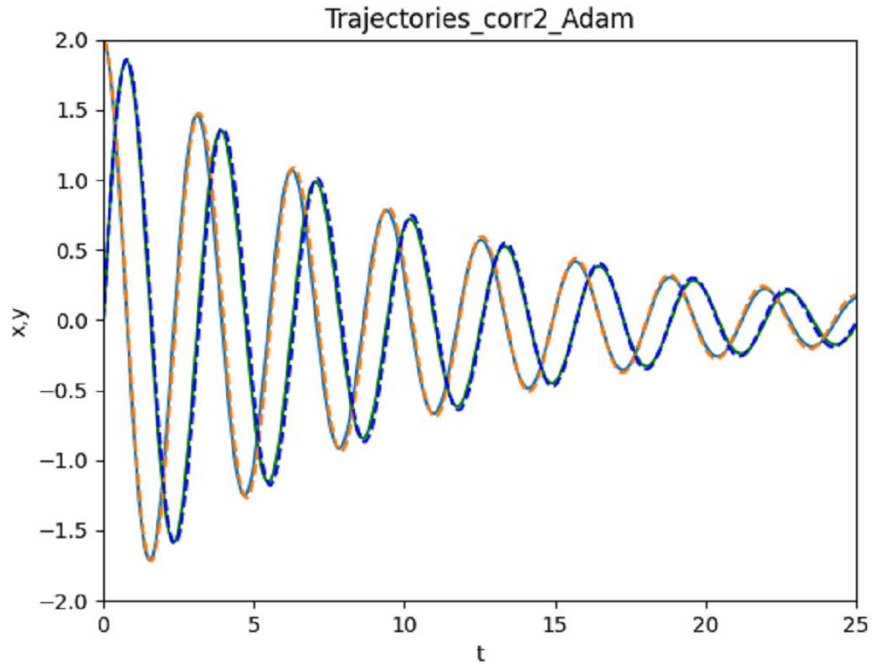
Theorem 1 (Instantaneous Change of Variables). *Let $\mathbf{z}(t)$ be a finite continuous random variable with probability $p(\mathbf{z}(t))$ dependent on time. Let $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$ be a differential equation describing a continuous-in-time transformation of $\mathbf{z}(t)$. Assuming that f is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then the change in log probability also follows a differential equation,*

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

Chen, Tian Qi, et al. "Neural Ordinary Differential Equations." NeurIPS2018 (best paper)



(a) 30 time points (b) 50 time points (c) 100 time points



- **Speed up Training**
- **Improve Performance**
- **Neural Architecture Design**

$$\underbrace{\begin{bmatrix} \mathbf{z}_0 \\ \log p(\mathbf{x}) - \log p_{z_0}(\mathbf{z}_0) \end{bmatrix}}_{\text{solutions}} = \underbrace{\int_{t_1}^{t_0} \begin{bmatrix} f(\mathbf{z}(t), t; \theta) \\ -\operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt}_{\text{dynamics}}, \qquad \underbrace{\begin{bmatrix} \mathbf{z}(t_1) \\ \log p(\mathbf{x}) - \log p(\mathbf{z}(t_1)) \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}}_{\text{initial values}}$$

$$\operatorname{Tr}(A) = E_{p(\boldsymbol{\epsilon})}[\boldsymbol{\epsilon}^T A \boldsymbol{\epsilon}].$$

$$\begin{aligned} \log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt \\ &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon}\right] dt \\ &= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\int_{t_0}^{t_1} \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\epsilon} dt\right] \end{aligned}$$

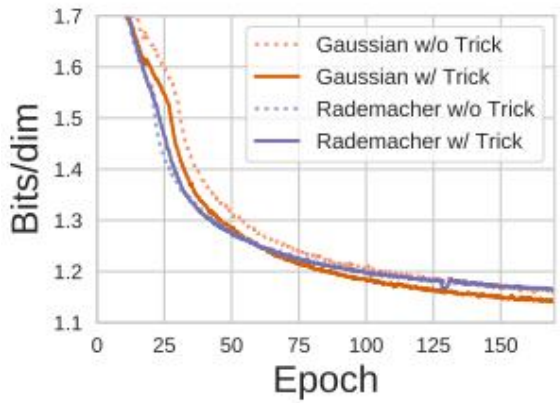
Algorithm 1 Unbiased stochastic log-density estimation using the FFJORD model

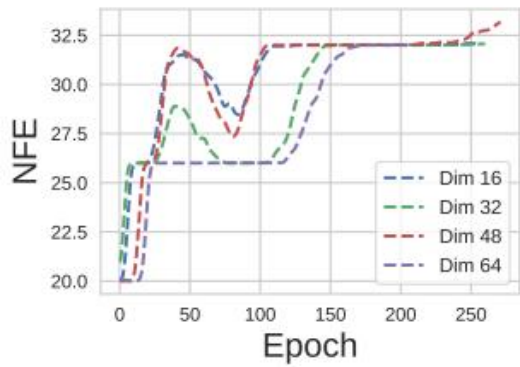
Require: dynamics f_θ, start time t_0, stop time t_1, minibatch of samples $\mathbf{x}$.
- $\boldsymbol{\epsilon} \leftarrow$ sample_unit_variance($\mathbf{x}$.shape) ▷ Sample $\boldsymbol{\epsilon}$ outside of the integral
- **function** $f_{aug}([\mathbf{z}_t, \log p_t], t)$: ▷ Augment f with log-density dynamics.
  - $f_t \leftarrow f_\theta(\mathbf{z}(t), t)$ ▷ Evaluate dynamics
  - $g \leftarrow \boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}}\big|_{\mathbf{z}(t)}$ ▷ Compute vector-Jacobian product with automatic differentiation
  - $\widetilde{\operatorname{Tr}} = \text{matrix\_multiply}(g, \boldsymbol{\epsilon})$ ▷ Unbiased estimate of $\operatorname{Tr}(\frac{\partial f}{\partial \mathbf{z}})$ with $\boldsymbol{\epsilon}^T \frac{\partial f}{\partial \mathbf{z}} \boldsymbol{\epsilon}$
  - **return** $[f_t, -\widetilde{\operatorname{Tr}}]$ ▷ Concatenate dynamics of state and log-density
- **end function**
- $[\mathbf{z}, \Delta_{logp}] \leftarrow \text{odeint}(f_{aug}, [\mathbf{x}, \vec{0}], t_0, t_1)$ ▷ Solve the ODE, ie. $\int_{t_0}^{t_1} f_{aug}([\mathbf{z}(t), \log p(\mathbf{z}(t))], t)\, dt$
- $\log \hat{p}(\mathbf{x}) \leftarrow \log p_{\mathbf{z}_0}(\mathbf{z})$ - Δ_{logp} ▷ Add change in log-density
- **return** $\log \hat{p}(\mathbf{x})$

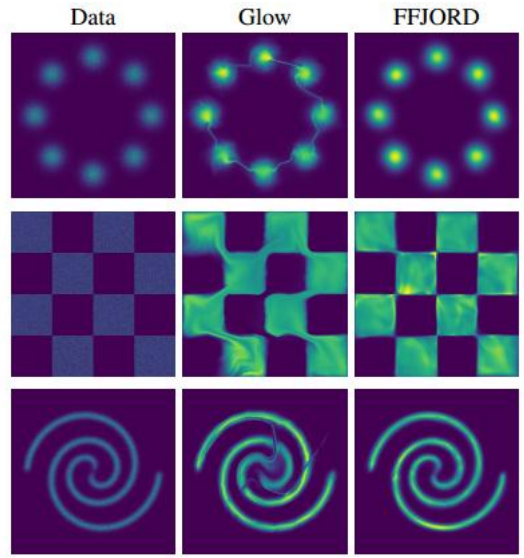
Theorem 1. The variance of the estimator for Tr(A) grows $\|A\|_F^2$ asymptotic to

$$\underbrace{\operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}}\right)}_{D \times D} = \underbrace{\operatorname{Tr}\left(\frac{\partial g}{\partial h}\frac{\partial h}{\partial \mathbf{z}}\right)}_{D \times D} = \underbrace{\operatorname{Tr}\left(\frac{\partial h}{\partial \mathbf{z}}\frac{\partial g}{\partial h}\right)}_{H \times H} = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T \frac{\partial h}{\partial \mathbf{z}}\frac{\partial g}{\partial h}\boldsymbol{\epsilon}\right]$$

Ffjord: Free-form continuous dynamics for scalable reversible generative models.







Ffjord: Free-form continuous dynamics for scalable reversible generative models.

$$\begin{cases} \mathbf{z}_{k+1} = \mathbf{z}_k + \epsilon\psi(\mathbf{x}, s_k, \mathbf{z}_k) + \epsilon^{p+1} g_\omega(\epsilon, \mathbf{x}, s_k, \mathbf{z}_k) \\ \mathbf{z}_0 = h_x(\mathbf{x}) \\ \hat{\mathbf{y}}_k = h_y(\mathbf{z}_k) \end{cases} \qquad k = 0, 1, \ldots, K-1$$

Residual fitting We first start by defining the *residual* of the solver (2)

$$\mathcal{R}(s_k, \mathbf{z}(s_k), \mathbf{z}(s_{k+1})) = \frac{1}{\epsilon^{p+1}} \left[\mathbf{z}(s_{k+1}) - \mathbf{z}(s_k) - \epsilon\psi(\mathbf{x}, s_k, \mathbf{z}(s_k))\right]$$

The discrepancy between the residual term and g

$$\ell = \frac{1}{K} \sum_{k=0}^{K-1} \|\mathcal{R}(s_k, \mathbf{z}(s_k), \mathbf{z}(s_{k+1})) - g_\theta(\epsilon, \mathbf{x}, s_k, \mathbf{z}(s_k))\|_2$$

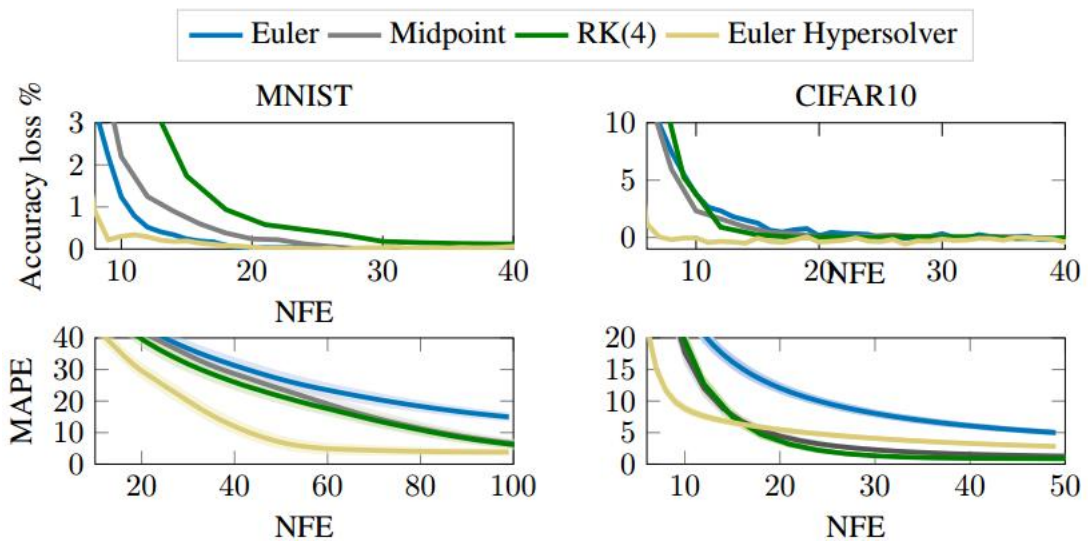
Theorem 1 (Hypersolver's Local Truncation Error). *If g_ω is a $\mathcal{O}(\delta)$ approximator of $\mathcal{R}$, i.e.*

$$\forall k \in \mathbb{N}_{\leq K} \quad \|\mathcal{R}(s_k, \mathbf{z}(s_k), \mathbf{z}(s_{k+1}) - g_\theta(\epsilon, \mathbf{x}, s_k, \mathbf{z}(s_k))\|_2 \leq \mathcal{O}(\delta),$$

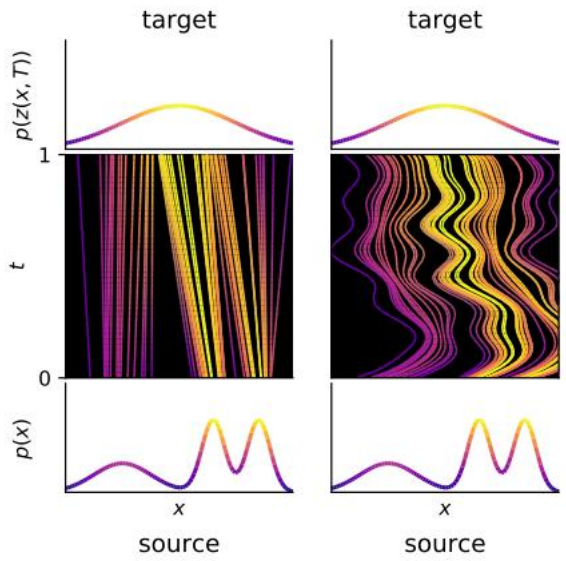
then, the local truncation error e_k of the hypersolver is $\mathcal{O}(\delta\epsilon^{p+1})$.

Trajectory fitting The second type of hypersolvers training aims at containing the global truncation error by minimizing the difference between the exact and approximated solutions in the whole depth domain $\mathcal{S}$, i.e.

$$L = \sum_{k=1}^{K} \|\mathbf{z}(s_k) - \mathbf{z}_k\|_2$$



Hypersolvers: *Toward Fast Continuous-Depth Models*



(a) Optimal transport map (b) generic flow

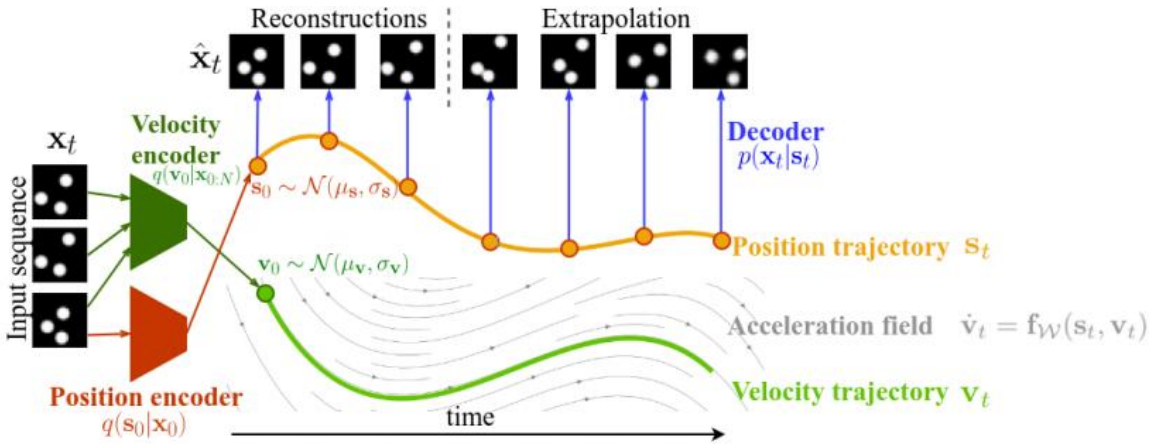
How to Train Your Neural ODE

$$\begin{cases} \dot{\mathbf{s}}_t &= \mathbf{v}_t \\ \dot{\mathbf{v}}_t &= \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t), \end{cases} \qquad \begin{bmatrix} \mathbf{s}_T \\ \mathbf{v}_T \end{bmatrix} = \begin{bmatrix} \mathbf{s}_0 \\ \mathbf{v}_0 \end{bmatrix} + \int_0^T \underbrace{\begin{bmatrix} \mathbf{v}_t \\ \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t) \end{bmatrix}}_{\tilde{\mathbf{f}}_{\mathcal{W}}(\mathbf{z}_t)} dt,$$

$$\frac{\partial \log q(\mathbf{z}_t|\mathcal{W})}{\partial t} = -\mathrm{Tr}\left(\frac{d\tilde{\mathbf{f}}_{\mathcal{W}}(\mathbf{z}_t)}{d\mathbf{z}_t}\right) dt = -\mathrm{Tr}\begin{pmatrix} \frac{\partial \mathbf{v}_t}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{v}_t}{\partial \mathbf{v}_t} \\ \frac{\partial \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{v}_t} \end{pmatrix} = -\mathrm{Tr}\left(\frac{\partial \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{v}_t}\right), \tag{5}$$

which results in the log densities over time,

$$\log q(\mathbf{z}_T|\mathcal{W}) = \log q(\mathbf{z}_0|\mathcal{W}) - \int_0^T \mathrm{Tr}\left(\frac{\partial \mathbf{f}_{\mathcal{W}}(\mathbf{s}_t, \mathbf{v}_t)}{\partial \mathbf{v}_t}\right) dt. \tag{6}$$



$$\begin{aligned} \log p(X) &\geq \underbrace{-\mathrm{KL}[q(\mathcal{W}, Z|X)||p(\mathcal{W}, Z)] + \mathbb{E}_{q(\mathcal{W},Z|X)}[\log p(X|\mathcal{W}, Z)]}_{\text{ELBO}} \\ &= -\mathbb{E}_{q(\mathcal{W},Z|X)}\left[\log \frac{q(\mathcal{W})q(Z|\mathcal{W}, X)}{p(\mathcal{W})p(Z)}\right] + \mathbb{E}_{q(\mathcal{W},Z|X)}[\log p(X|\mathcal{W}, Z)] \\ &= -\mathrm{KL}[q(\mathcal{W})||p(\mathcal{W})] + \mathbb{E}_{q(\mathcal{W},Z|X)}\left[-\log \frac{q(Z|\mathcal{W}, X)}{p(Z)} + \log p(X|\mathcal{W}, Z)\right] \\ &= \underbrace{-\mathrm{KL}[q(\mathcal{W})||p(\mathcal{W})]}_{\text{ODE regularization}} + \underbrace{\mathbb{E}_{q_{\text{enc}}(\mathbf{z}_0|X)}\left[-\log \frac{q_{\text{enc}}(\mathbf{z}_0|X)}{p(\mathbf{z}_0)} + \log p(\mathbf{x}_0|\mathbf{z}_0)\right]}_{\text{VAE loss}} \\ &\quad + \underbrace{\sum_{i=1}^{N} \mathbb{E}_{q_{\text{ode}}(\mathcal{W},\mathbf{z}_i|X,\mathbf{z}_0)}\left[-\log \frac{q_{\text{ode}}(\mathbf{z}_i|\mathcal{W}, X)}{p(\mathbf{z}_i)} + \log p(\mathbf{x}_i|\mathbf{z}_i)\right]}_{\text{dynamic loss}} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{ODE}^2\text{VAE}} = &-\beta\,\mathrm{KL}[q(\mathcal{W})||p(\mathcal{W})] + \mathbb{E}_{q(\mathcal{W},Z|X)}\left[-\log \frac{q(Z|\mathcal{W}, X)}{p(Z)} + \log p(X|\mathcal{W}, Z)\right] \\ &- \gamma\mathbb{E}_{q(\mathcal{W})}\left[\mathrm{KL}[q_{\text{ode}}(Z|X)||q_{\text{enc}}(Z|\mathcal{W}, X)]\right]. \end{aligned}$$

ODE2VAE: Deep generative second order ODEs with Bayesian neural networks.

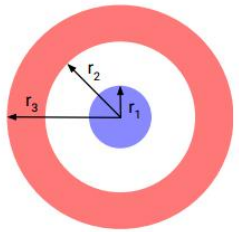
Theorem 1. Neural Ordinary Differential Equations (ODEs) **learn representations that preserve the topology of the input space** and prove that this implies the existence of functions Neural ODEs cannot represent.

For all t ∈ [0, T], $\phi_t : R^d \to R^d$ is a **homeomorphism**.

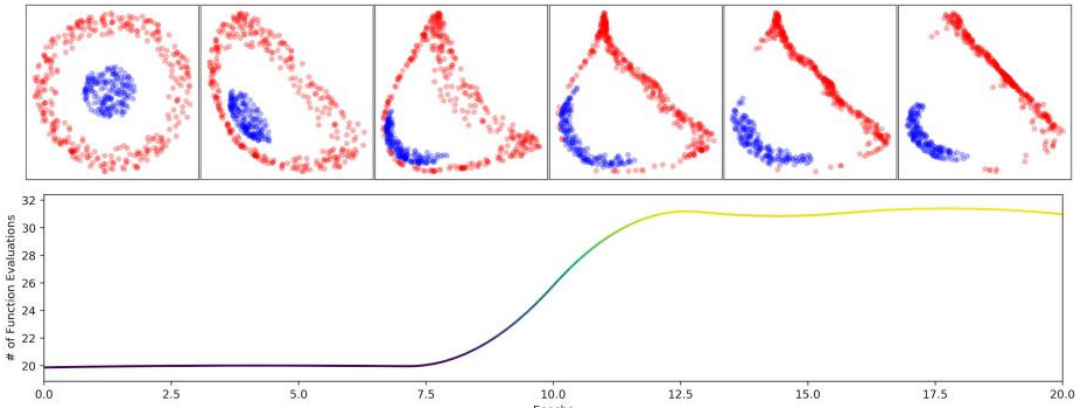
(a) ϕ_t is continuous

(b) ϕ_t is a bijection

(c) ϕ_t^{-1} is continuous

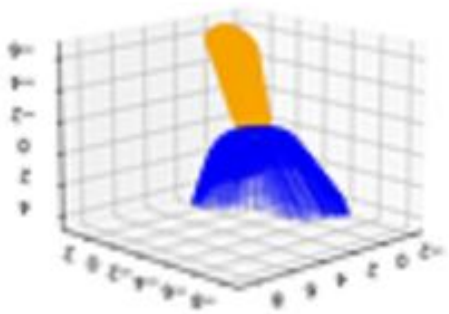


$$\begin{cases} g(\mathbf{x}) = -1 & \text{if } \|\mathbf{x}\| \leq r_1 \\ g(\mathbf{x}) = 1 & \text{if } r_2 \leq \|\mathbf{x}\| \leq r_3, \end{cases}$$



Augment the space on which we learn and solve the ODE from R_d to R_{d+p}, allowing the ODE flow to lift points into the additional dimensions to avoid trajectories intersecting each other

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix} = \mathbf{f}\left(\begin{bmatrix}\mathbf{h}(t)\\ \mathbf{a}(t)\end{bmatrix}, t\right), \qquad \begin{bmatrix}\mathbf{h}(0)\\ \mathbf{a}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{x}\\ \mathbf{0}\end{bmatrix}$$



Augmented Neural ODEs

Contributions: **(1) Augmented Strategies**

General form of neural ode

Neural Ordinary Differential Equation

$$\begin{cases} \dot{\mathbf{z}} = f_{\theta(s)}(s, \mathbf{x}, \mathbf{z}(s)) \\ \mathbf{z}(0) = h_x(\mathbf{x}) \\ \hat{\mathbf{y}}(s) = h_y(\mathbf{z}(s)) \end{cases} \quad s \in \mathcal{S} \quad (1)$$

Input	$\mathbf{x}$	$\mathbb{R}^{n_x}$
Output	$\hat{\mathbf{y}}$	$\mathbb{R}^{n_y}$
(Hidden) State	$\mathbf{z}$	$\mathbb{R}^{n_z}$
Parameters	$\theta(s)$	$\mathbb{R}^{n_\theta}$
Neural Vector Field	$f_{\theta(s)}$	$\mathbb{R}^{n_z}$
Input Network	h_x	$\mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$
Output Network	h_y	$\mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$

Augmented Neural ODEs:
(1) 0-Augmentation[ANODE]
(2) Input-layer Augmentation

$$\mathbf{z}(0) = h_x(\mathbf{x}) \qquad h_x := [\mathbf{x}, \tilde{h}(\mathbf{x})]$$

Introduce a loss function **distributed on the whole depth domain**

NODE: $$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

DNODE: $$\ell := L(\mathbf{z}(S)) + \int_{\mathcal{S}} l(\tau, \mathbf{z}(\tau))\mathrm{d}\tau$$

The problem can be converted to:

$$\min_\theta \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \ell_k$$
$$\text{subject to } \dot{\mathbf{z}} = f_{\theta(s)}(s, \mathbf{x}_k, \mathbf{z}(s)) \quad s \in \mathcal{S} \quad , \quad \forall k \in \mathcal{K}$$
$$\mathbf{z}(0) = h_x(\mathbf{x}_k), \quad \hat{\mathbf{y}}(s) = h_y(\mathbf{z}(s))$$

Theorem 1 (Generalized Adjoint Method). *Consider the loss function (2). Then,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta} = \int_{\mathcal{S}} \mathbf{a}^\top(\tau) \frac{\partial f_\theta}{\partial \theta} \mathrm{d}\tau \quad \textit{where } \mathbf{a}(s) \textit{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top \frac{\partial f_\theta}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Dissecting Neural ODEs

Contributions: **(2) Depth–Variance**

$$\dot{z} = f_\theta(s, z(s))$$

In fact, while each residual block is characterized by its own parameters vector , the authors consider model where the depth variable ***s*** enters in the dynamics *per se* rather than in the map $s \to \theta(s)$

Theorem 2 (Infinite–Dimensional Gradients). *Consider the loss function (2) and let $\theta(s) \in \mathbb{L}_2$. Then, sensitivity of ℓ with respect to $\theta(s)$ (i.e. directional derivative in functional space) is*

$$\frac{\delta \ell}{\delta \theta(s)} = \mathbf{a}^\top(s) \frac{\partial f_{\theta(s)}}{\partial \theta(s)} \quad \textit{where } \mathbf{a}(s) \textit{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s) \frac{\partial f_{\theta(s)}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Spectral based method:

$$\theta(s) = \sum_{j=1}^{m} \alpha_j \odot \psi_j(s)$$

orthogonal basis

$$\frac{\mathrm{d}\ell}{\mathrm{d}\alpha} = \int_{\mathcal{S}} \mathbf{a}^\top(\tau) \frac{\partial f_{\theta(s)}}{\partial \theta(s)} \psi(\tau) \mathrm{d}\tau, \quad \psi = (\psi_1, \ldots, \psi_m)$$

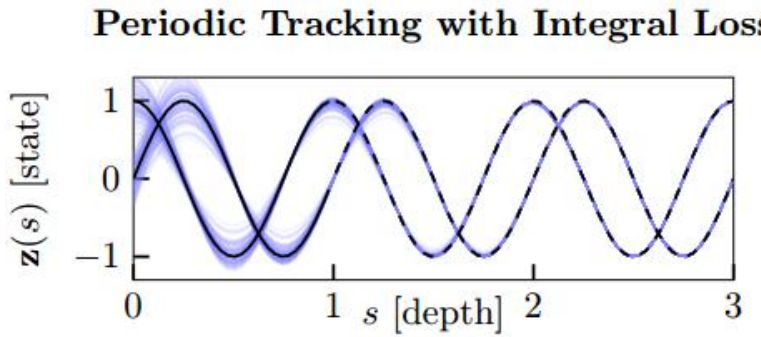
Stacked Neural ODEs:

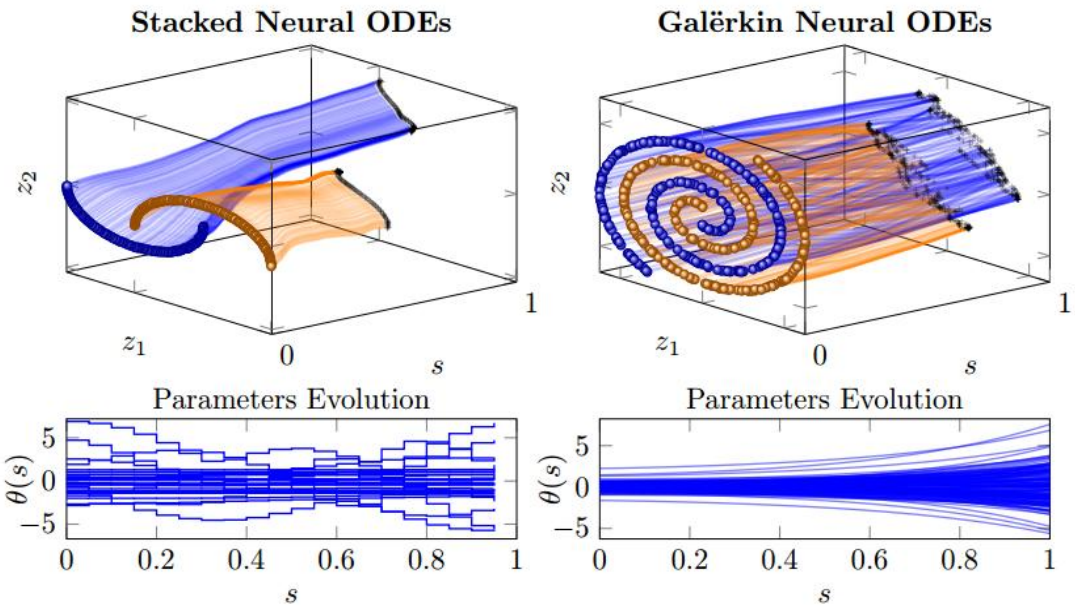
$$\mathbf{z}(S) = h_x(\mathbf{x}) + \sum_{i=0}^{p-1} \int_{s_1}^{s_{i+1}} f_{\theta_i}(\tau, \mathbf{x}, \mathbf{z}(\tau)) \mathrm{d}\tau$$

Corollary 2 (Stacked Gradients). *Under the assumptions of Theorem 2, if $\theta(s) = \theta_i \; \forall s \in [s_i, s_{i+1}]$,*

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta_i} = -\int_{s_{i+1}}^{s_i} \mathbf{a}^\top(\tau) \frac{\partial f_{\theta_i}}{\partial \theta_i} \mathrm{d}\tau \quad \textit{where } \mathbf{a}(s) \textit{ satisfies } \begin{cases} \dot{\mathbf{a}}^\top(s) = -\mathbf{a}^\top(s) \frac{\partial f_{\theta_i}}{\partial \mathbf{z}} - \frac{\partial l}{\partial \mathbf{z}} & s \in [s_i, s_{i+1}] \\ \mathbf{a}^\top(S) = \frac{\partial L}{\partial \mathbf{z}(S)} \end{cases}$$

Dissecting Neural ODEs





Contributions: **(3) Data control and adaptive-depth**

Proposition 2. *For all $\epsilon > 0$, $x \in \mathbb{R}$ there exists a parameter $\theta > 0$ such that*

$$|\varphi(x) - z(1)| < \epsilon, \tag{8}$$

where $z(1)$ is the solution of the Neural ODE

$$\begin{cases} \dot{z}(s) = -\theta(z(s) + x) \\ z(0) = x \end{cases}, \quad s \in [0, 1] \ . \tag{9}$$

General data-controlled Neural ODE

$$\dot{\mathbf{z}}(s) = f_{\theta(s)}(s, \mathbf{x}, \mathbf{z}(s))$$
$$\mathbf{z}(0) = h_x(\mathbf{x})$$

Adaptive Depth for Neural ODE

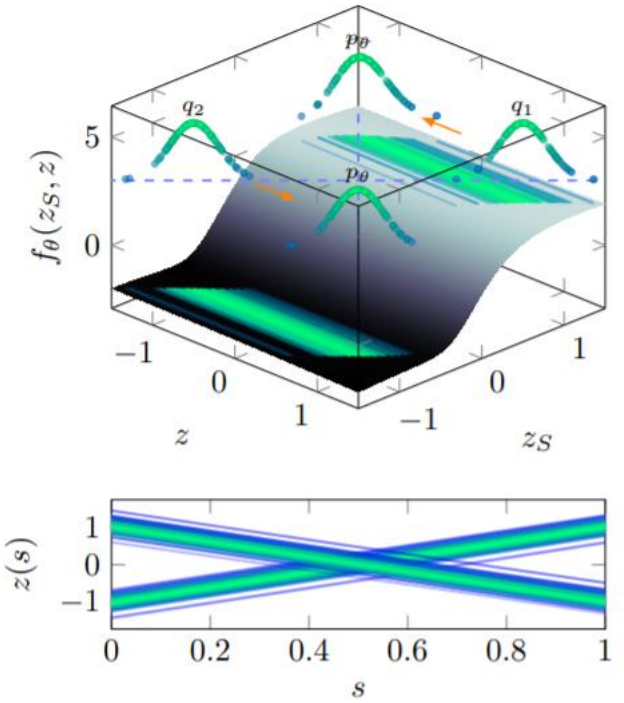
$$\hat{\mathbf{y}} = \mathbf{z} + \int_0^{g_\omega(\mathbf{x})} f_{\theta(s)}(\tau, \mathbf{x}, \mathbf{z}(\tau)) d\tau$$

Data-control in Normalizing Flows

$$\mathbf{z}(0) = \mathbf{z}_S + \int_S^0 f_\theta(s)(s, \mathbf{z}(s))\mathrm{d}s$$

$$\dot{z}(s) = f_\theta(z_S, z), \quad z_S \sim q_1 \text{ or } z_S \sim q_2$$

Conditional Continuous Normaling Flows



Data-control allows Neural ODEs to learn a family of vector fields, conditioning on input data information
Depth-adaptation sidesteps known expressivity limitations of continuous-depth models.

- **Neural ODE & CNF**
  - **Speed up training**
    - **FFJORD: trace estimator**
    - **How to train your Neural ODE (regulator)**
    - **Hypersolvers: improve numerical integrator**
  - **Improve Performance**
    - **Augmentation: 0-augmentation; IL-augmentation**
    - **Depth variance: spectral based; stack NODE;**
    - **Data Control; Adaptive Depth**
  - **Architecture**
    - **High order ODE**

Stability Analysis with Mean Field Theory

Interpretation of deep learning as a parameter estimation problem of nonlinear dynamical systems.

1. Given a network architecture and parameters obtained by some optimization process, is the forward propagation problem well-posed?
2. Is the learning problem well-posed? In other words, given sufficient training are there parameters such that the DNN generalizes well or can generalization be improved by adding appropriate regularization?

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j) \quad \text{for} \quad j = 0, \ldots, N-1.$$

$$\begin{aligned} \min \quad & \frac{1}{s} S(\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top), \mathbf{C}) + \alpha R(\mathbf{W}, \mu, \mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}) \\ \text{s.t.} \quad & \mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j), \quad j = 0, 1, \ldots, N-1, \end{aligned}$$

S: convex **R**: penalty

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}^\top(t)\mathbf{y}(t) + b(t)), \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0,$$

The ode is stable if K(t) change sufficiently slow and

$$\max_{i=1,2,\ldots,n} Re(\lambda_i(\mathbf{J}(t))) \leq 0, \quad \forall t \in [0, T]$$

$$\begin{aligned} \mathbf{J}(t) &= \left(\nabla_{\mathbf{y}} \left(\sigma(\mathbf{K}(t)^\top\mathbf{y} + b(t))\right)\right)^\top \\ &= \operatorname{diag}\left(\sigma'(\mathbf{K}(t)^\top\mathbf{y} + b(t))\right)\mathbf{K}(t)^\top. \end{aligned}$$

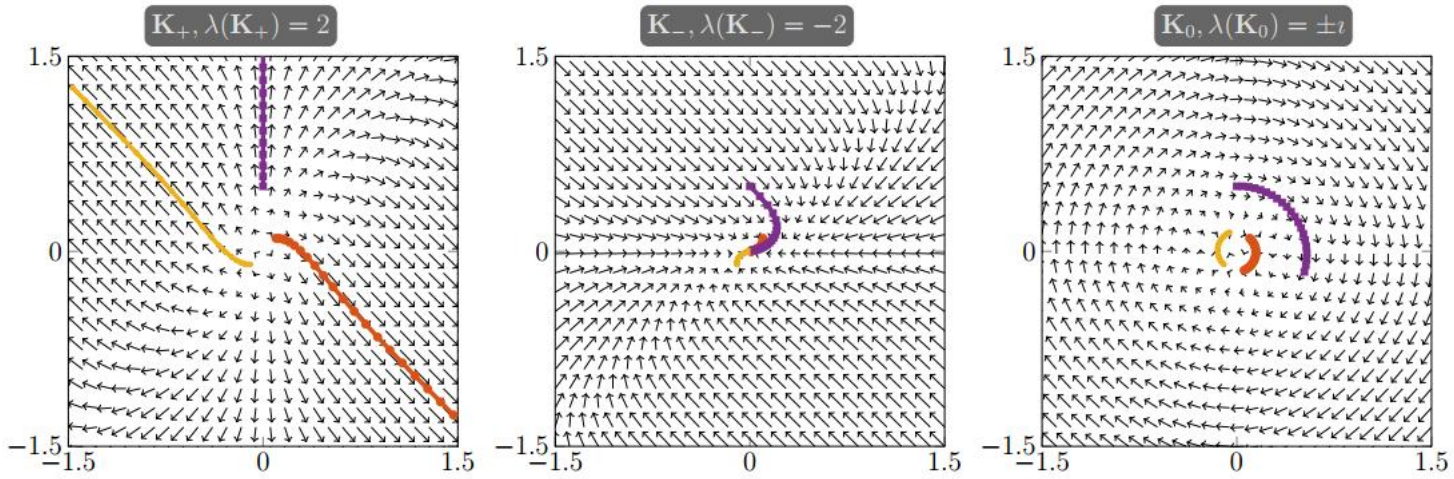
$$\max_{i=1,2,\ldots,n} Re(\lambda_i(\mathbf{K}(t))) \leq 0, \quad \forall t \in [0, T].$$

Lemma 1 (Stability of Forward Euler Method) *The forward propagation in Eq. 2.1 is stable if*

$$\max_{i=1,2,\ldots,n} |1 + h\lambda_i(\mathbf{J}_j)| \leq 1, \quad \forall j = 0, 1, \ldots, N-1. \tag{3.9}$$

λ(J(t)) is the ith eigenvalue of the Jacobian of the right hand side

Stable Architectures for Deep Neural Networks



(1) K<0 is not a sufficient condition. The gradients of the objective function in will **vanish** since small changes in the kernels will have no noticeable impact on the values of the outputs.

In summary, our discussion of ResNets illustrates that stable forward propagation and well-posed learning problems can be obtained for deep networks when

$$Re\left(\lambda_i(\mathbf{K}(t))\right) \approx 0, \quad \forall i = 1, 2, \ldots, n, \ t \in [0, T]. \tag{3.11}$$

Stable Architectures for Deep Neural Networks

Antisymmetric weight matrices:

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma\left(\frac{1}{2}\mathbf{Y}_j\left(\mathbf{K}_j - \mathbf{K}_j^\top - \gamma\mathbf{I}\right) + b_j\right), \quad j = 0, \ldots, N-1,$$

$$\dot{\mathbf{y}}(t) = \sigma\left(\frac{1}{2}(\mathbf{K}(t) - \mathbf{K}(t)^\top - \gamma\mathbf{I})\mathbf{y}(t) + b(t)\right), \quad \forall t \in [0, T].$$

Hamilton inspired neural network:

$$\dot{\mathbf{y}}(t) = -\nabla_{\mathbf{z}} H(\mathbf{y}, \mathbf{z}, t) \quad \text{and} \quad \dot{\mathbf{z}}(t) = \nabla_{\mathbf{y}} H(\mathbf{y}, \mathbf{z}, t), \quad \forall t \in [0, T].$$

$$H(\mathbf{y}, \mathbf{z}) = \frac{1}{2}\mathbf{z}^\top\mathbf{z} + f(\mathbf{y}),$$

$$\ddot{\mathbf{y}}(t) = \nabla_{\mathbf{y}} f(\mathbf{y}(t)), \quad \forall t \in [0, T].$$

Stable for all weight matrices with non-positive real eigenvalues

$$\mathbf{K}(\mathbf{C}) = -\mathbf{C}^\top\mathbf{C}, \quad \text{for some} \quad \mathbf{C} \in \mathbb{R}^{n\times n}.$$

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}(t)\mathbf{z}(t) + b(t)) \quad \text{and} \quad \dot{\mathbf{z}}(t) = -\sigma(\mathbf{K}(t)^\top\mathbf{y}(t) + b(t)).$$

$$\frac{\partial}{\partial t}\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(t) = \sigma\left(\begin{pmatrix}0 & \mathbf{K}(t)\\ -\mathbf{K}(t)^\top & 0\end{pmatrix}\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(t) + b(t)\right), \quad \begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(0) = \begin{pmatrix}\mathbf{y}_0\\ 0\end{pmatrix}$$

Symplectic forward propagation(discrete version):

$$\mathbf{y}_{j+1} = \begin{cases} 2\mathbf{y}_j + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 0 \\ 2\mathbf{y}_j - \mathbf{y}_{j-1} + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 1, 2, \ldots, N-1 \end{cases}$$

$$\mathbf{z}_{j+\frac{1}{2}} = \mathbf{z}_{j-\frac{1}{2}} - h\sigma(\mathbf{K}_j^\top\mathbf{y}_j + b_j) \quad \text{and} \quad \mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_j\mathbf{z}_{j+\frac{1}{2}} + b_j).$$

Stable Architectures for Deep Neural Networks

$$\mathbf{z}_{j+\frac{1}{2}} = \mathbf{z}_{j-\frac{1}{2}} - h\sigma(\mathbf{K}_j^\top \mathbf{y}_j + b_j) \quad \text{and} \quad \mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_j \mathbf{z}_{j+\frac{1}{2}} + b_j).$$

$$\begin{aligned}
\frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} &= h\mathrm{diag}\left(\sigma'(\mathbf{K}_k\mathbf{y}_k + b_k)\right)(\mathbf{y}_k \otimes \mathbf{I}) =: \mathbf{C}_1 \\
\frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} &= -h\mathrm{diag}\left(\sigma'(-\mathbf{K}_k^\top \mathbf{z}_{k+\frac{1}{2}} + b_k)\right)(\mathbf{I} \otimes \mathbf{Z}_{k+\frac{1}{2}} + \mathbf{K}_k^\top \otimes \mathbf{I}) \\
&=: \mathbf{C}_2,
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k} &= \frac{\partial \mathbf{z}_{j-\frac{1}{2}}}{\partial \mathbf{K}_k} + h\mathrm{diag}\left(\sigma'(\mathbf{K}_j\mathbf{y}_j + b_j)\right)\mathbf{K}_j \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} \\
\frac{\partial \mathbf{y}_{j+1}}{\partial \mathbf{K}_k} &= \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} - h\mathrm{diag}\left(\sigma'(-\mathbf{K}_j^\top \mathbf{z}_{j+\frac{1}{2}} + b_j)\right)\mathbf{K}_j^\top \frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k}.
\end{aligned}$$

$$\begin{aligned}
\mathbf{B}_j &= -h\mathrm{diag}\left(\sigma'(\mathbf{K}_j\mathbf{y}_j + b_j)\right)\mathbf{K}_j, \\
\mathbf{A}_j &= h\mathrm{diag}\left(\sigma'(-\mathbf{K}_j^\top \mathbf{z}_{j+\frac{1}{2}} + b_j)\right)\mathbf{K}_j^\top.
\end{aligned}$$

Parameter update with adjoint method:

$$\begin{pmatrix}
\mathbf{I} & & & & & & \\
 & \mathbf{I} & & & & & \\
-\mathbf{I} & \mathbf{B}_{k+1} & \mathbf{I} & & & & \\
 & -\mathbf{I} & \mathbf{A}_{k+1} & \mathbf{I} & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & -\mathbf{I} & \mathbf{A}_N & \mathbf{I} & \\
 & & & & -\mathbf{I} & \mathbf{B}_N & \mathbf{I}
\end{pmatrix}
\begin{pmatrix}
\frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} \\
\frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} \\
\vdots \\
\\
\frac{\partial \mathbf{z}_{N+\frac{1}{2}}}{\partial \mathbf{K}_k} \\
\frac{\partial \mathbf{y}_{N+1}}{\partial \mathbf{K}_k}
\end{pmatrix}
=
\begin{pmatrix}
\mathbf{C}_1 \\
\mathbf{C}_2 \\
0 \\
\vdots \\
\\
0
\end{pmatrix},$$

Stable Architectures for Deep Neural Networks

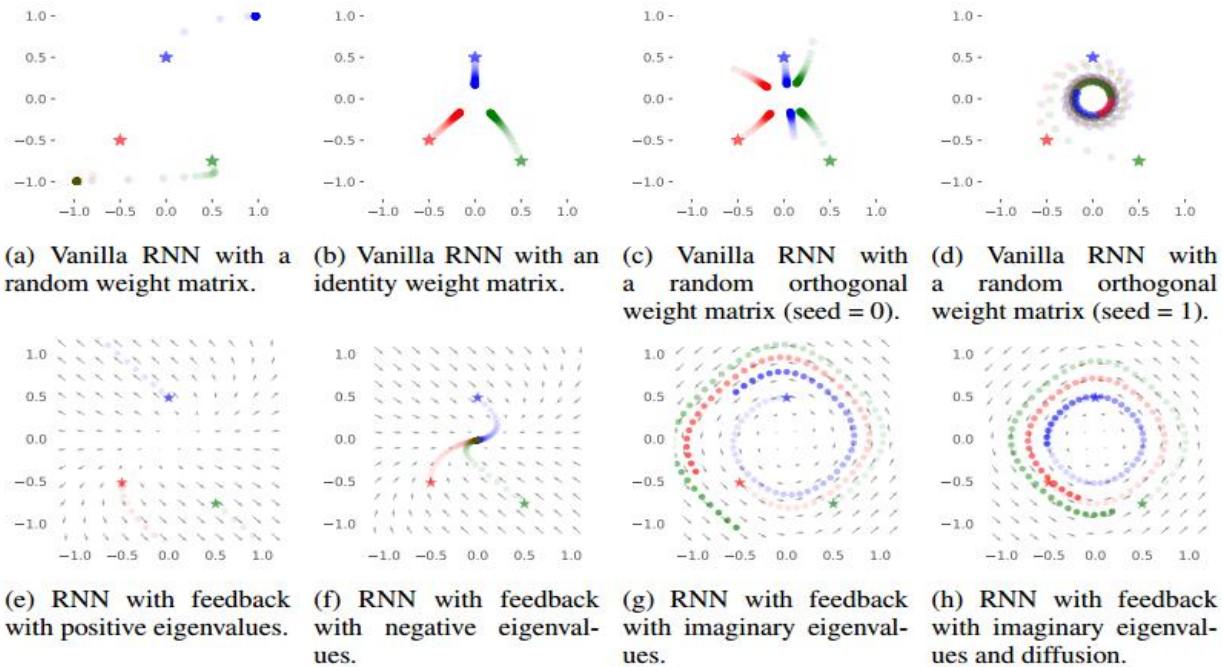
$$\boldsymbol{h}'(t) = \tanh\left((\boldsymbol{W}_h - \boldsymbol{W}_h^T)\boldsymbol{h}(t) + \boldsymbol{V}_h\boldsymbol{x}(t) + \boldsymbol{b}_h\right),$$

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \epsilon \tanh\left((\boldsymbol{W}_h - \boldsymbol{W}_h^T - \gamma\boldsymbol{I})\boldsymbol{h}_{t-1} + \boldsymbol{V}_h\boldsymbol{x}_t + \boldsymbol{b}_h\right),$$

Incorporate input gate

$$\boldsymbol{z}_t = \sigma\left((\boldsymbol{W}_h - \boldsymbol{W}_h^T - \gamma\boldsymbol{I})\boldsymbol{h}_{t-1} + \boldsymbol{V}_z\boldsymbol{x}_t + \boldsymbol{b}_z\right),$$

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \epsilon\boldsymbol{z}_t \circ \tanh\left((\boldsymbol{W}_h - \boldsymbol{W}_h^T - \gamma\boldsymbol{I})\boldsymbol{h}_{t-1} + \boldsymbol{V}_h\boldsymbol{x}_t + \boldsymbol{b}_h\right),$$



(a) Vanilla RNN with a random weight matrix.

(b) Vanilla RNN with an identity weight matrix.

(c) Vanilla RNN with a random orthogonal weight matrix (seed = 0).

(d) Vanilla RNN with a random orthogonal weight matrix (seed = 1).

(e) RNN with feedback with positive eigenvalues.

(f) RNN with feedback with negative eigenvalues.

(g) RNN with feedback with imaginary eigenvalues.

(h) RNN with feedback with imaginary eigenvalues and diffusion.

ANTISYMMETRICRNN: A DYNAMICAL SYSTEM VIEW ON RECURRENT NEURAL NETWORKS

$$\dot{\mathbf{Y}}(t) = \sigma(\mathbf{K}(t)\mathbf{Z}(t) + \mathbf{b}(t)),$$
$$\dot{\mathbf{Z}}(t) = -\sigma(\mathbf{K}(t)^T\mathbf{Y}(t) + \mathbf{b}(t)),$$

$$\dot{\mathbf{Y}}(t) = \mathbf{K}_1^T(t)\sigma(\mathbf{K}_1(t)\mathbf{Z}(t) + \mathbf{b}_1(t)),$$
$$\dot{\mathbf{Z}}(t) = -\mathbf{K}_2^T(t)\sigma(\mathbf{K}_2(t)\mathbf{Y}(t) + \mathbf{b}_2(t)).$$

$$\begin{pmatrix}\dot{\mathbf{Y}}\\ \dot{\mathbf{Z}}\end{pmatrix} = \begin{pmatrix}\mathbf{K}_1^T & 0\\ 0 & -\mathbf{K}_2^T\end{pmatrix}\sigma\left(\begin{pmatrix}0 & \mathbf{K}_1\\ \mathbf{K}_2 & 0\end{pmatrix}\begin{pmatrix}\mathbf{Y}\\ \mathbf{Z}\end{pmatrix} + \begin{pmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{pmatrix}\right).$$

$$\mathbf{J} = \nabla_{\binom{\mathbf{Y}}{\mathbf{z}}}\begin{pmatrix}\mathbf{K}_1^T & 0\\ 0 & -\mathbf{K}_2^T\end{pmatrix}\sigma\left(\begin{pmatrix}0 & \mathbf{K}_1\\ \mathbf{K}_2 & 0\end{pmatrix}\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}\right)$$
$$= \begin{pmatrix}\mathbf{K}_1^T & 0\\ 0 & -\mathbf{K}_2^T\end{pmatrix}\mathrm{diag}(\sigma')\begin{pmatrix}0 & \mathbf{K}_1\\ \mathbf{K}_2 & 0\end{pmatrix},$$

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\mathbf{K}_{j1}^T\sigma(\mathbf{K}_{j1}\mathbf{Z}_j + \mathbf{b}_{j1}),$$
$$\mathbf{Z}_{j+1} = \mathbf{Z}_j - h\mathbf{K}_{j2}^T\sigma(\mathbf{K}_{j2}\mathbf{Y}_{j+1} + \mathbf{b}_{j2}).$$

Midpoint Network

$$\frac{\mathbf{Y}_{j+1} - \mathbf{Y}_{j-1}}{2h} = \mathcal{F}(\mathbf{Y}_j). \tag{11}$$

This gives the following forward propagation

$$\mathbf{Y}_{j+1} = \mathbf{Y}_{j-1} + 2h\mathcal{F}(\mathbf{Y}_j), \quad \text{for } j = 1, \ldots, N-1, \tag{12}$$

where $\mathbf{Y}_1$ is obtained by one forward Euler step. To guarantee stability for a single layer we can use the function $\mathcal{F}$ to contain an anti-symmetric linear operator, that is,

$$\mathcal{F}(\mathbf{Y}) = \sigma((\mathbf{K} - \mathbf{K}^T)\mathbf{Y} + \mathbf{b}). \tag{13}$$

The Jacobian of this forward propagation is

$$\mathbf{J} = \mathrm{diag}(\sigma')(\mathbf{K} - \mathbf{K}^T), \tag{14}$$

which has only imaginary eigenvalues. This yields the single layer midpoint network

$$\mathbf{Y}_{j+1} = \begin{cases} 2h\sigma((\mathbf{K}_j - \mathbf{K}_j^T)\mathbf{Y}_j + \mathbf{b}_j), & j = 0, \\ \mathbf{Y}_{j-1} + 2h\sigma((\mathbf{K}_j - \mathbf{K}_j^T)\mathbf{Y}_j + \mathbf{b}_j), & \mathrm{j} > 0. \end{cases} \tag{15}$$

$$\ddot{\mathbf{Y}}(t) = -\mathbf{K}(t)^T\sigma(\mathbf{K}(t)\mathbf{Y}(t) + b(t)), \quad \mathbf{Y}(0) = \mathbf{Y}_0. \tag{16}$$

$$\mathbf{Y}_{j+1} = \begin{cases} 2\mathbf{Y}_j - h^2\mathbf{K}_j^T\sigma(\mathbf{K}_j\mathbf{Y}_j + \mathbf{b}_j), & j = 0, \\ 2\mathbf{Y}_j - \mathbf{Y}_{j-1} - h^2\mathbf{K}_j^T\sigma(\mathbf{K}_j\mathbf{Y}_j + \mathbf{b}_j), & j > 0. \end{cases}$$

正则项没有

Reversible Architectures for Arbitrarily Deep Residual Neural Networks

$$\mathbf{h}_t = \mathcal{W}_{\text{FC}}(\mathbf{x}_t; \boldsymbol{\theta}_t) := \mathbf{W}_t \mathbf{x}_t + \mathbf{b}_t, \text{ where}$$
$$\mathbf{W}_t^{(i,j)} \sim \mathcal{N}(0, \sigma_w^2/N_t) \quad \mathbf{b}_t^{(i)} \sim \mathcal{N}(0, \sigma_b^2) \ .$$

According to Central Limit Theorem,

$$q_t := \frac{1}{N_t} \sum_{i=0}^{N_t} (\mathbf{h}_t^{(i)})^2 \ .$$

$$q_{t+1} = \sigma_w^2 \mathbb{E}_{\mathbf{h}_t^{(i)} \sim \mathcal{N}(0, q_t)} \left[\phi^2(\mathbf{h}_t^{(i)}) \right] + \sigma_b^2$$

$$q_0 = \frac{1}{N_0} \boldsymbol{x}_0 \cdot \boldsymbol{x}_0.$$

[24] showed that for any bounded ϕ and finite value of σ_w and σ_b, there exists a fixed point, $q^* := \lim_{t \to \infty} q_t$, regardless of the initial state q_0.

Similarly, for a pair of input $(\boldsymbol{x}^{(\alpha)}, \boldsymbol{x}^{(\beta)})$ we can derive the following recurrence relation

$$q_{t+1}^{(\alpha,\beta)} = \sigma_w^2 \mathbb{E}_{(\mathbf{h}_t^{(\alpha)}, \mathbf{h}_t^{(\beta)})^\intercal \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_t)} [\phi(\mathbf{h}_t^{(\alpha)}) \phi(\mathbf{h}_t^{(\beta)})] + \sigma_b^2 \quad (4)$$

$$\text{where} \quad \boldsymbol{\Sigma}_t = \begin{pmatrix} q_t^{\alpha} & q_t^{(\alpha,\beta)} \\ q_t^{(\alpha,\beta)} & q_t^{\beta} \end{pmatrix} \quad (5)$$

$$q_0^{(\alpha,\beta)} = \frac{1}{N_0} \boldsymbol{x}_0^{(\alpha)} \cdot \boldsymbol{x}_0^{(\beta)}.$$

ditions for q^* to exist, we also have the fixed points for the covariance and correlation, respectively denoted $q^*_{(\alpha,\beta)}$ and $c^* = q^*_{(\alpha,\beta)} / \sqrt{q^*_\alpha q^*_\beta} = q^*_{(\alpha,\beta)} / q^*$. The element of the

$$\Sigma^*_{(\alpha,\beta)} = q^* \left(\delta_{(\alpha,\beta)} + (1 - \delta_{(\alpha,\beta)}) c^* \right) \ ,$$

$$|q_t - q^*| \sim e^{-t/\xi_{q^*}} \qquad \xi_{q^*}^{-1} = -\log \chi_{q^*}$$
$$|c_t - c^*| \sim e^{-t/\xi_{c^*}} \qquad \xi_{c^*}^{-1} = -\log \chi_{c^*}$$

is introduced. Poole et al. (2016) note that the $c^* = 1$ fixed point is stable if $\chi_1 < 1$ and is unstable otherwise. Thus, $\chi_1 = 1$ represents a critical line separating an ordered phase (in which $c^* = 1$ and all inputs end up asymptotically correlated) and a chaotic phase (in which $c^* < 1$ and all inputs end up asymptotically decorrelated). For the case of $\phi = \tanh$, the phase diagram in fig. 1 (a) is observed.

Deep Learning Theory Review: An Optimal Control and Dynamic Systems Perspective

DNN & Physical Systems

Preliminaries:
(1) 最小作用量原理
(2) 欧拉-拉格朗日方程
(3) 勒让德变换
(4) 哈密顿方程
(5) PMP最大原理

Euler-Lagrange Equations

$$\frac{d}{dt}\left[\nabla_v L(\mathbf{x}^*(t), \dot{\mathbf{x}}^*(t))\right] = \nabla_x L(\mathbf{x}^*(t), \dot{\mathbf{x}}^*(t)).$$

⟶ L = T-V

Generalized Momentum

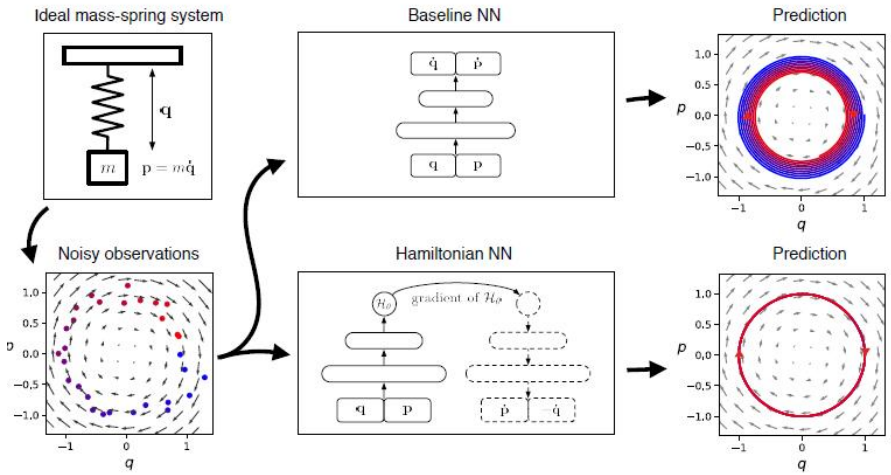
$$\mathbf{p}(t) := \nabla_v L(\mathbf{x}(t), \dot{\mathbf{x}}(t)) \qquad (0 \le t \le T).$$

Hamilton's Equations

$$H(x,p) = p \cdot \mathbf{v}(x,p) - L(x, \mathbf{v}(x,p)),$$

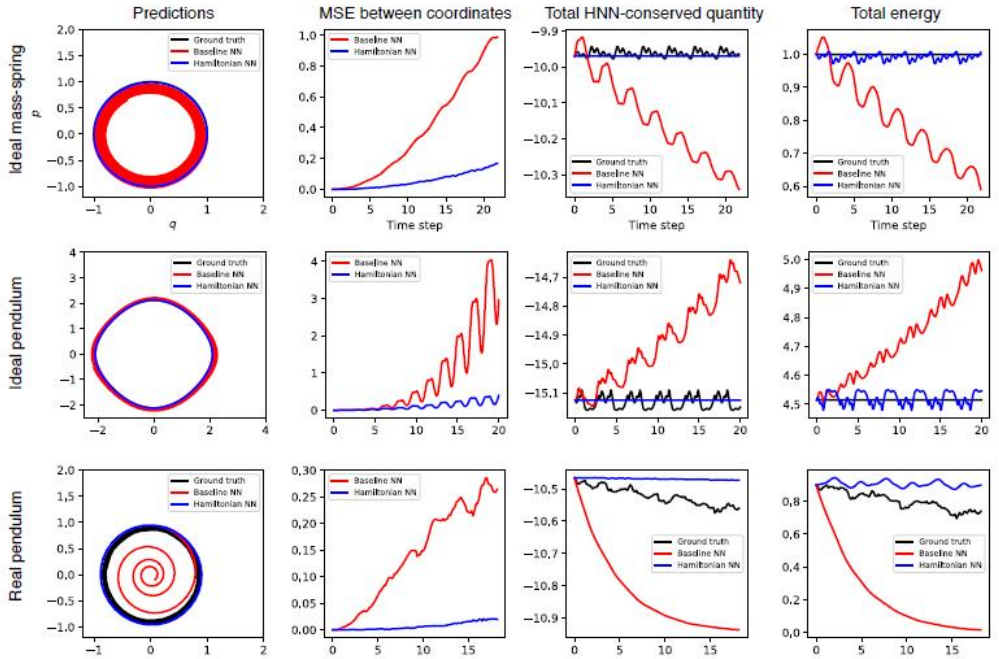
$$\begin{cases} \dot{\mathbf{x}}(t) = \nabla_p H(\mathbf{x}(t), \mathbf{p}(t)) \\ \dot{\mathbf{p}}(t) = -\nabla_x H(\mathbf{x}(t), \mathbf{p}(t)) \end{cases}$$

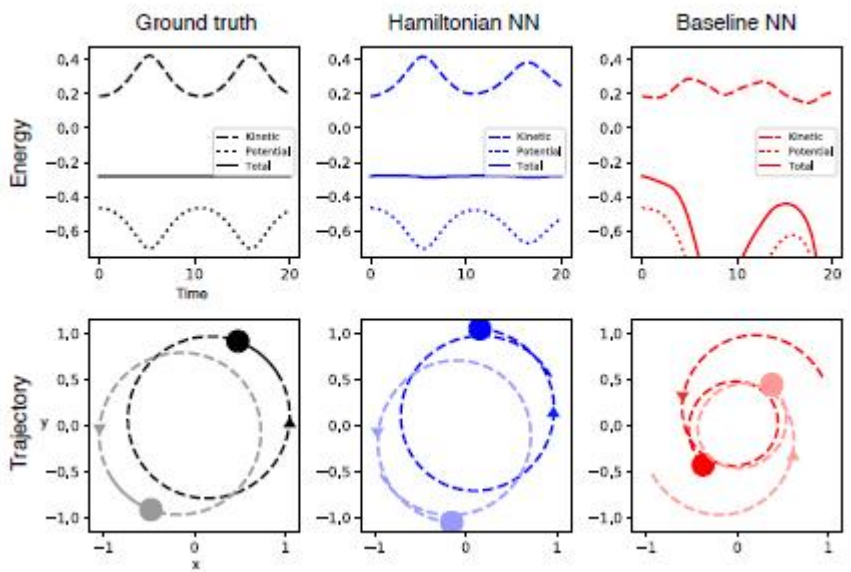
⟶ H = T+V



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

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



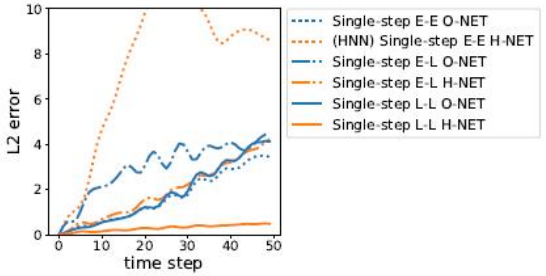


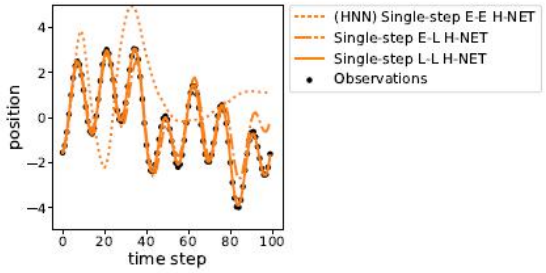
Hamiltonian Neural Networks

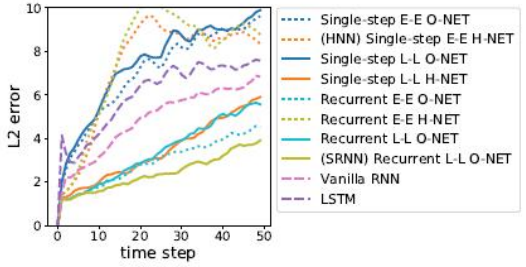
$$\dot{p} = -\frac{\partial H}{\partial q}, \qquad \dot{q} = +\frac{\partial H}{\partial p}, \qquad H(p,q) = K(p) + V(q).$$

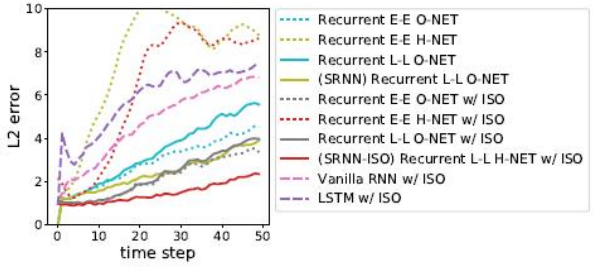
- (1) The time derivative model can be an O-NET or H-NET.
- (2) The training integrator can be any explicit integrators.
- (3) The training integrator can be consist of a single step or multiple steps.
- (4) The testing integrator can also be chosen freely and can use a different time-step size as it does not involve back-propagration.

in the framework that we have adopted so far, the initial states p_0 and q_0 are treated as the actual initial states from which the system begins to evolve despite the added noise in observation. **L-BFGS-B algorithm**

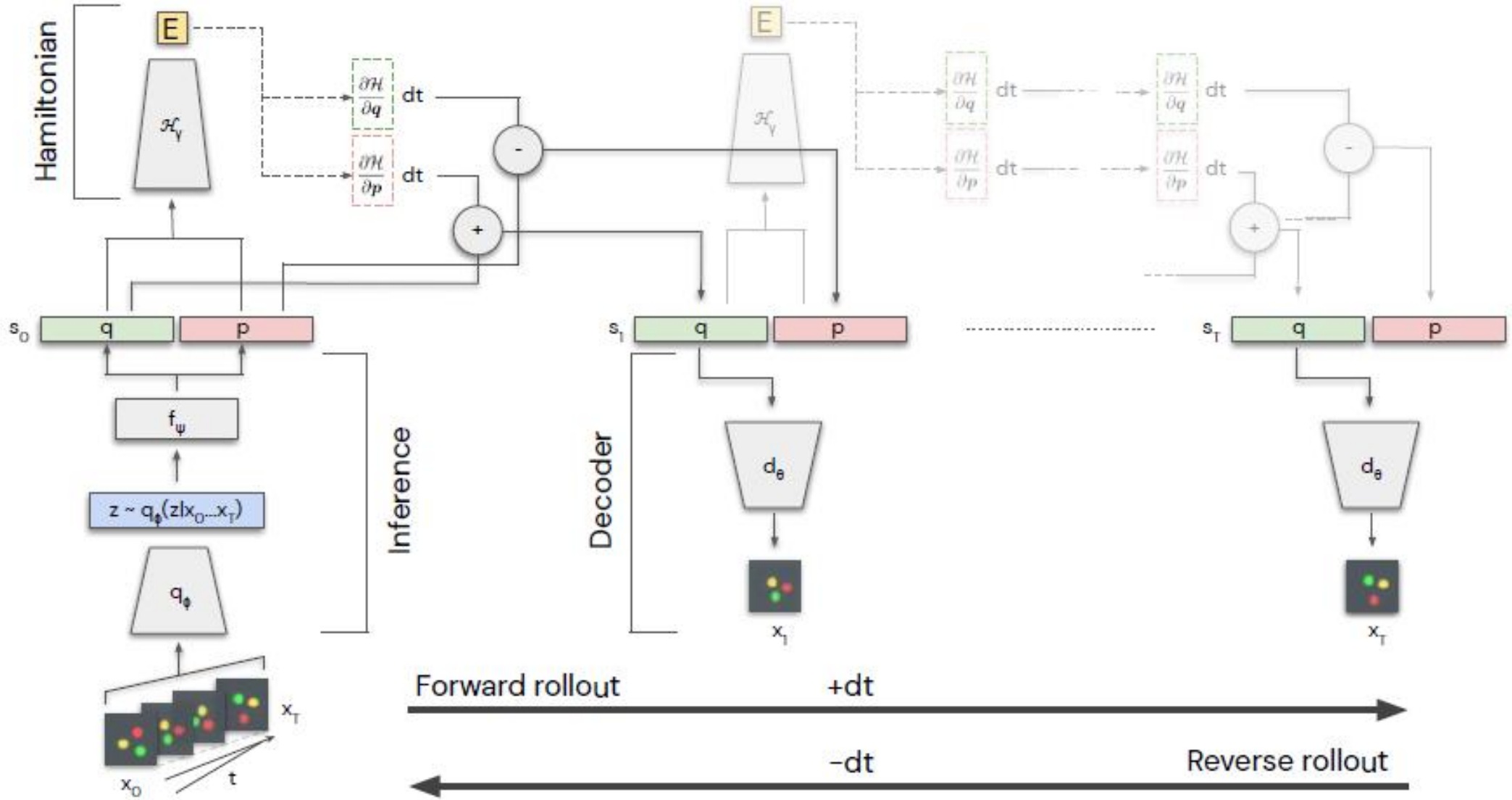




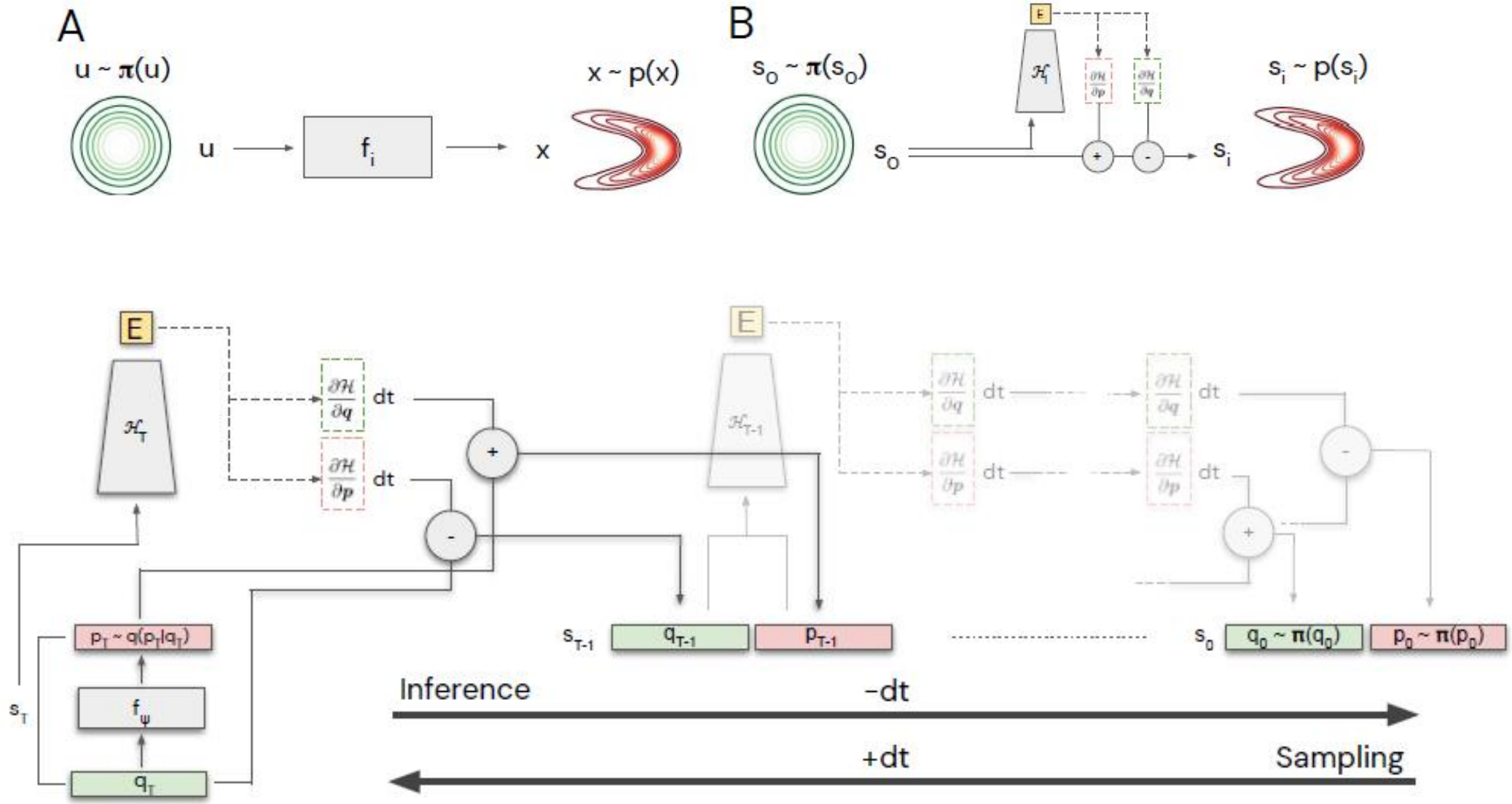




Symplectic recurrent neural network



Hamiltonian Generative Networks



Hamiltonian Generative Networks

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}(\mathbf{q})\mathbf{p} + V(\mathbf{q}),$$

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \qquad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}},$$

With external control u

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \begin{bmatrix}\frac{\partial H}{\partial \mathbf{p}}\\ -\frac{\partial H}{\partial \mathbf{q}}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{g}(\mathbf{q})\end{bmatrix}\mathbf{u},$$

Control via energy forcing:

$$\mathbf{u}(\mathbf{q},\mathbf{p}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\mathbf{p})$$

$\beta(q)$ **Potential energy shaping**

$\mathbf{v}(\mathbf{p})$ **Damping injection**

$\beta(q)$ **shape the potential energy such that the desired Hamiltonian $H_d(q; p)$ has a minimum at** $(q^*, 0)$

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \begin{bmatrix}\frac{\partial H}{\partial \mathbf{p}}\\ -\frac{\partial H}{\partial \mathbf{q}}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{g}(\mathbf{q})\end{bmatrix}\boldsymbol{\beta}(\mathbf{q}) = \begin{bmatrix}\frac{\partial H_d}{\partial \mathbf{p}}\\ -\frac{\partial H_d}{\partial \mathbf{q}}\end{bmatrix},$$

$$H_d(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}(\mathbf{q})\mathbf{p} + V_d(\mathbf{q}).$$

$$\boldsymbol{\beta}(\mathbf{q}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\left(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\right).$$

Assume

$$\mathbf{v}(\mathbf{p}) = -\mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}(\mathbf{K}_d\mathbf{p}).$$

$$V_d(\mathbf{q}) = \frac{1}{2}(\mathbf{q}-\mathbf{q}^\star)^T\mathbf{K}_p(\mathbf{q}-\mathbf{q}^\star),$$

$$\mathbf{u} = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\left(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q}-\mathbf{q}^\star) - \mathbf{K}_d\mathbf{p}\right).$$

PD Controller with an additional energy compensation

SYMPLECTIC ODE-NET: LEARNING HAMILTONIAN DYNAMICS WITH CONTROL

Training Neural ODE with constant forcing: $\mathbf{x} = (\mathbf{q}, \mathbf{p})$

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_\theta(\mathbf{x}, \mathbf{u}) \\ \mathbf{0} \end{bmatrix} = \tilde{\mathbf{f}}_\theta(\mathbf{x}, \mathbf{u}).$$

The idea here is to use different constant external forcing to get the system responses and use those responses to train the model. With a trained model, we can apply a time-varying $\bar{\mathbf{u}}$ to the dynamics $\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, \bar{\mathbf{u}})$ and generate estimated trajectories. When we synthesize the controller, $\bar{\mathbf{u}}$ remains constant in each integration step. As long as our model interpolates well among different values of constant $\bar{\mathbf{u}}$, we could get good estimated trajectories with a time-varying $\bar{\mathbf{u}}$. The problem is then how to design the network architecture of $\tilde{\mathbf{f}}_\theta$, or equivalently $\mathbf{f}_\theta$ such that we can learn the dynamics in an efficient way.

Learning from Generalized Coordinates and Momentum

$$\mathbf{f}_\theta(\mathbf{q}, \mathbf{p}, \mathbf{u}) = \begin{bmatrix} \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ -\frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{g}_{\theta_3}(\mathbf{q}) \end{bmatrix} \mathbf{u} \qquad H_{\theta_1,\theta_2}(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \mathbf{M}_{\theta_1}^{-1}(\mathbf{q})\mathbf{p} + V_{\theta_2}(\mathbf{q})$$

SYMPLECTIC ODE-NET: LEARNING HAMILTONIAN DYNAMICS WITH CONTROL

Pendulum

$$(\mathbf{x}_1(\mathbf{q}), \mathbf{x}_2(\mathbf{q}), \mathbf{x}_3(\dot{\mathbf{q}}), \mathbf{u})_{t_0,\ldots,t_n} = (\cos \mathbf{q}, \sin \mathbf{q}, \dot{\mathbf{q}}, \mathbf{u})_{t_0,\ldots,t_n}$$

$$\dot{\mathbf{x}}_1 = -\sin \mathbf{q} \circ \dot{\mathbf{q}} = -\mathbf{x}_2 \circ \dot{\mathbf{q}}$$
$$\dot{\mathbf{x}}_2 = \cos \mathbf{q} \circ \dot{\mathbf{q}} = \mathbf{x}_1 \circ \dot{\mathbf{q}}$$
$$\dot{\mathbf{x}}_3 = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{p}) = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2))\mathbf{p} + \mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2)\dot{\mathbf{p}}$$

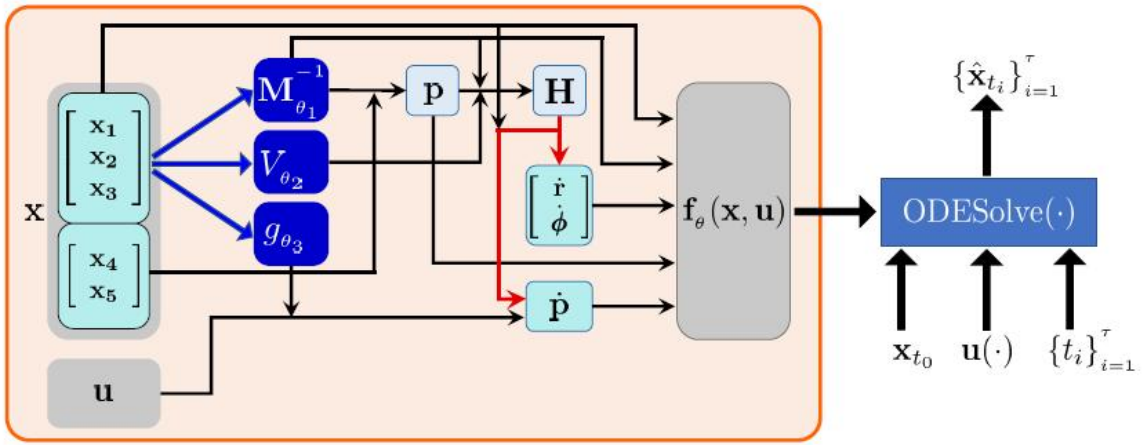
$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}}$$
$$\dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u} = -\frac{\partial \mathbf{x}_1}{\partial \mathbf{q}}\frac{\partial H}{\partial \mathbf{x}_1} - \frac{\partial \mathbf{x}_2}{\partial \mathbf{q}}\frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u}$$
$$= \sin \mathbf{q} \circ \frac{\partial H}{\partial \mathbf{x}_1} - \cos \mathbf{q} \circ \frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u} = \mathbf{x}_2 \circ \frac{\partial H}{\partial \mathbf{x}_1} - \mathbf{x}_1 \circ \frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u}$$

$$\mathbf{f}_\theta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{u}) = \begin{bmatrix} -\mathbf{x}_2 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ \mathbf{x}_1 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}_{\theta_1}^{-1}(\mathbf{x}_1,\mathbf{x}_2))\mathbf{p} + \mathbf{M}_{\theta_1}^{-1}(\mathbf{x}_1,\mathbf{x}_2)\left(\mathbf{x}_2 \circ \frac{\partial H_{\theta_1\theta_2}}{\partial \mathbf{x}_1} - \mathbf{x}_1 \circ \frac{\partial H_{\theta_1\theta_2}}{\partial \mathbf{x}_2} + \mathbf{g}_{\theta_3}(\mathbf{x}_1,\mathbf{x}_2)\mathbf{u}\right) \end{bmatrix}$$

$$H_{\theta_1,\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \mathbf{M}_{\theta_1}^{-1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{p} + V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2)$$
$$\mathbf{p} = \mathbf{M}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{x}_3$$

SYMPLECTIC ODE-NET: LEARNING HAMILTONIAN DYNAMICS WITH CONTROL

Hybrid Systems $(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5, \mathrm{u})_{t_0,\ldots,t_n} = (\mathrm{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathrm{r}}, \dot{\boldsymbol{\phi}}, \mathrm{u})_{t_0,\ldots,t_n}.$



$$\mathrm{p} = \mathrm{M}_{\theta_1}(\mathrm{x}_1, \mathrm{x}_2, , \mathrm{x}_3) \begin{bmatrix} \mathrm{x}_4 \\ \mathrm{x}_5 \end{bmatrix}$$

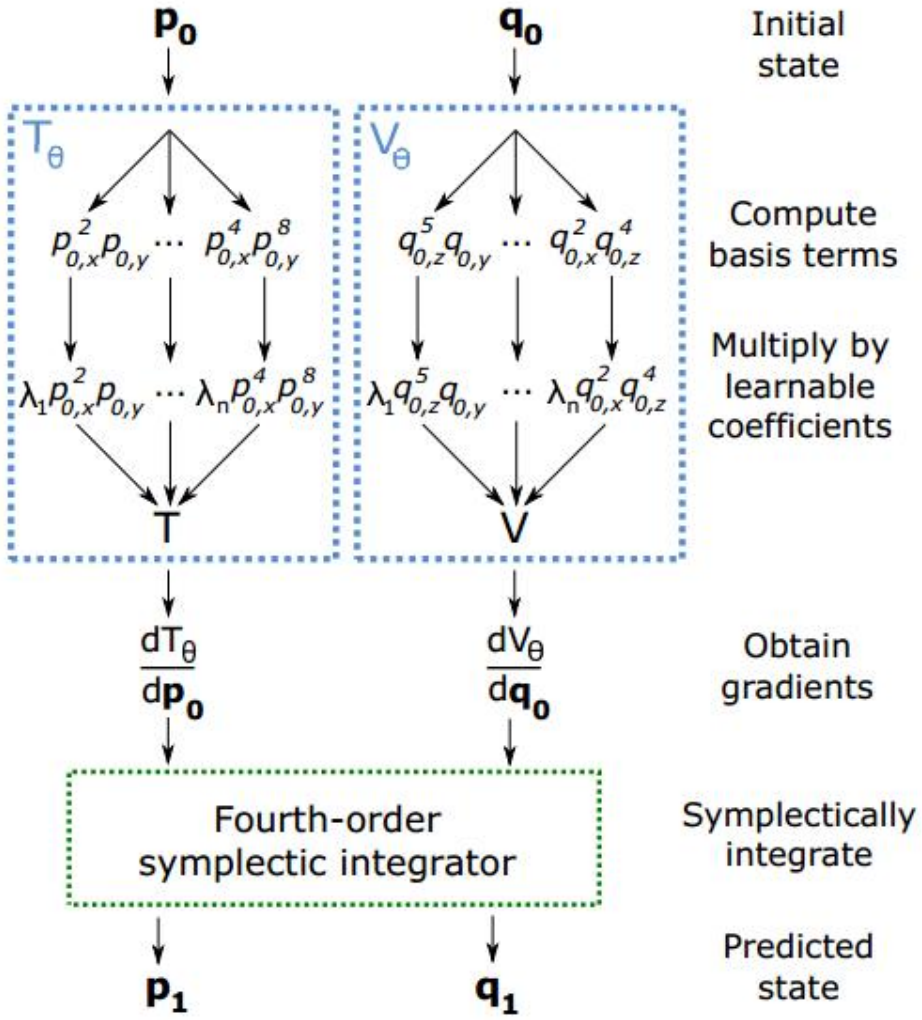
$$H_{\theta_1,\theta_2}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{p}) = \frac{1}{2}\mathrm{p}^T \mathrm{M}_{\theta_1}^{-1}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)\mathrm{p} + V_{\theta_2}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$$

$$\dot{\mathrm{q}} = \begin{bmatrix} \dot{\mathrm{r}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} = \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathrm{p}}$$

$$\dot{\mathrm{p}} = \begin{bmatrix} -\frac{\partial H_{\theta_1,\theta_2}}{\partial \mathrm{x}_1} \\ \mathrm{x}_3 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathrm{x}_2} - \mathrm{x}_2 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathrm{x}_3} \end{bmatrix} + \mathrm{g}_{\theta_3}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)\mathrm{u}$$

$$\begin{bmatrix} \dot{\mathrm{x}}_1 \\ \dot{\mathrm{x}}_2 \\ \dot{\mathrm{x}}_3 \\ \dot{\mathrm{x}}_4 \\ \dot{\mathrm{x}}_5 \end{bmatrix} = \begin{bmatrix} \dot{\mathrm{r}} \\ -\mathrm{x}_3\dot{\boldsymbol{\phi}} \\ \mathrm{x}_2\dot{\boldsymbol{\phi}} \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathrm{M}_{\theta_1}^{-1}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3))\mathrm{p} + \mathrm{M}_{\theta_1}^{-1}(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)\dot{\mathrm{p}} \end{bmatrix} = \mathrm{f}_\theta(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \mathrm{x}_5, \mathrm{u})$$

SYMPLECTIC ODE-NET: LEARNING HAMILTONIAN DYNAMICS WITH CONTROL



$$\mathcal{H} = T(\mathbf{p}) + V(\mathbf{q}) \qquad \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}, \frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}$$

Each network performs a sparse regression within a user-specified function search space, which may be broadly defined to include multi-variate polynomial terms of specified degree, trigonometric terms, and so on.

Pros: No need for derivatives of p and q

Sparse Symplectically Integrated Neural Networks

Difficulties: Nonseparability typically causes chaos and instability.
Enable accurate and robust predictions of long-term Hamiltonian dynamics based on short-term observation data.

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\partial \mathcal{H}}{\partial p}, \quad \frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial q},$$

$$\overline{\mathcal{H}}(q,p,x,y) := \mathcal{H}_A + \mathcal{H}_B + \omega \mathcal{H}_C$$

$$\mathcal{H}_A = \mathcal{H}(q,y), \quad \mathcal{H}_B = \mathcal{H}(x,p), \quad \mathcal{H}_C = \frac{1}{2}\left(\|q-x\|_2^2 + \|p-y\|_2^2\right)$$

$$\begin{cases} \frac{\mathrm{d}q}{\mathrm{d}t} = \frac{\partial \overline{\mathcal{H}}}{\partial p} = \frac{\partial \mathcal{H}(x,p)}{\partial p} + \omega(p-y), \\ \frac{\mathrm{d}p}{\mathrm{d}t} = -\frac{\partial \overline{\mathcal{H}}}{\partial q} = -\frac{\partial \mathcal{H}(q,y)}{\partial q} - \omega(q-x), \\ \frac{\mathrm{d}x}{\mathrm{d}t} = \frac{\partial \overline{\mathcal{H}}}{\partial y} = \frac{\partial \mathcal{H}(q,y)}{\partial y} - \omega(p-y), \\ \frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{\partial \overline{\mathcal{H}}}{\partial x} = -\frac{\partial \mathcal{H}(x,p)}{\partial x} + \omega(q-x), \end{cases}$$

$$(q,p,x,y)|_{t=t_0} = (q_0,p_0,x_0,y_0)$$
$$(q,p,x,y)|_{t=t_n} = (q_n,p_n,x_n,y_n)$$

Algorithm 1 Integrate (4) by using the second-order symplectic integrator

Input: $q_0, p_0, t_0, t, \mathrm{d}t$; ϕ_1^δ, ϕ_2^δ, and ϕ_3^δ in (5);
Output: $(\hat{q}, \hat{p}, \hat{x}, \hat{y}) = (q_n, p_n, x_n, y_n)$
$(q_0, p_0, x_0, y_0) = (q_0, p_0, q_0, p_0)$
$n = \mathrm{floor}[(t - t_0)/\mathrm{d}t]$
for $i = 1 \to n$ **do**
  $(q_i, p_i, x_i, y_i) = \phi_1^{\mathrm{d}t/2} \circ \phi_2^{\mathrm{d}t/2} \circ \phi_3^{\mathrm{d}t} \circ \phi_2^{\mathrm{d}t/2} \circ \phi_1^{\mathrm{d}t/2} \circ (q_{i-1}, p_{i-1}, x_{i-1}, y_{i-1})$;
end

$$\begin{bmatrix} q \\ p - \delta[\partial \mathcal{H}_\theta(q,y)/\partial q] \\ x + \delta[\partial \mathcal{H}_\theta(q,y)/\partial p] \\ y \end{bmatrix}, \quad \begin{bmatrix} q + \delta[\partial \mathcal{H}_\theta(x,p)/\partial p] \\ p \\ x \\ y - \delta[\partial \mathcal{H}_\theta(x,p)/\partial q] \end{bmatrix}, \text{ and } \frac{1}{2}\left[\begin{pmatrix} q+x \\ p+y \end{pmatrix} + R^\delta \begin{pmatrix} q-x \\ p-y \end{pmatrix} \right]$$

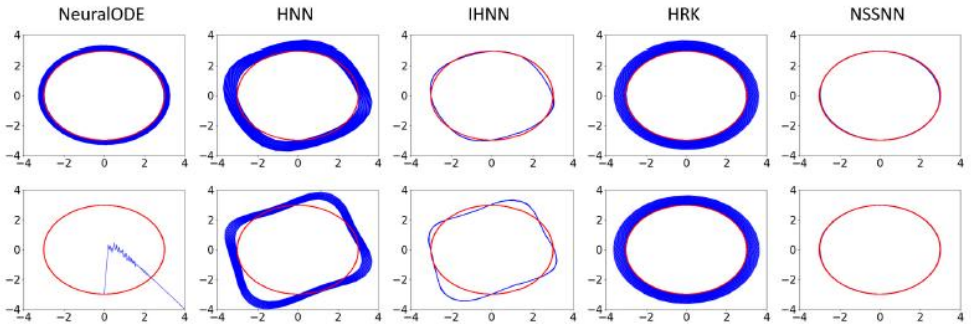
$$R^\delta := \begin{bmatrix} \cos(2\omega\delta) I & \sin(2\omega\delta) I \\ -\sin(2\omega\delta) I & \cos(2\omega\delta) I \end{bmatrix}$$

$$\mathcal{L}_{NSSNN} = \frac{1}{N_b} \sum_{j=1}^{N_b} \|q^{(j)} - \hat{q}^{(j)}\|_1 + \|p^{(j)} - \hat{p}^{(j)}\|_1 + \|q^{(j)} - \hat{x}^{(j)}\|_1 + \|p^{(j)} - \hat{y}^{(j)}\|_1,$$

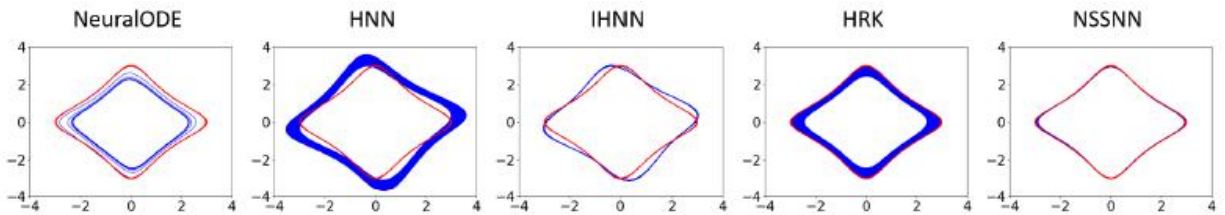
No need for derivatives of p and q

NONSEPARABLE SYMPLECTIC NEURAL NETWORKS

Methods	NSSNN	HNN	NeuralODE	TaylorNet	SSINN	SRNN	SympNet
Solve nonseparable systems	✓	✓	✓				✓
Solve separable systems	✓	✓	✓	✓	✓	✓	✓
Preserve symplectic structure	✓	Partially		✓	Partially	✓	✓
Utilize continuous dynamics	✓		✓	✓	✓	✓	✓
No need for derivatives in dataset	✓		✓	✓	✓	✓	✓
Long-term predictability	✓	Partially		✓	✓	✓	✓
Extend to N-body system	✓	✓	✓	✓		✓	



$$\mathcal{H} = 0.5(q^2 + p^2)$$



$$\mathcal{H} = 0.5(q^2 + 1)(p^2 + 1)$$

NONSEPARABLE SYMPLECTIC NEURAL NETWORKS

$$L = T - V \quad T = \tfrac{1}{2}\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}} \quad dV/d\mathbf{q} = \mathbf{g}(\mathbf{q})$$

Lower triangular matrix: L

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}_i} - \frac{\partial L}{\partial \mathbf{q}_i} = \boldsymbol{\tau}_i$$

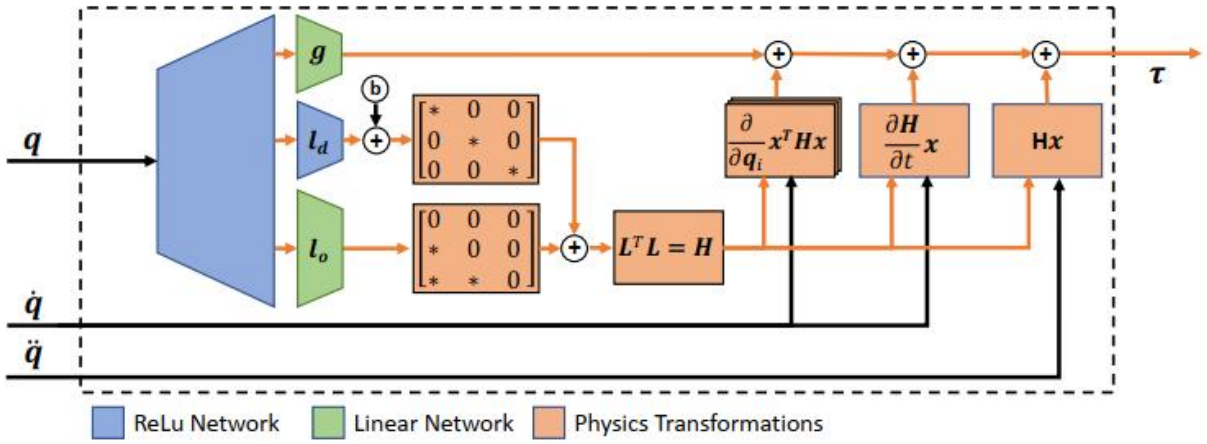
$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \underbrace{\dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T}_{:=\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}$$

$$\hat{\mathbf{H}}(\mathbf{q}) = \hat{\mathbf{L}}(\mathbf{q};\theta)\hat{\mathbf{L}}(\mathbf{q};\theta)^T \qquad \hat{\mathbf{g}}(\mathbf{q}) = \hat{\mathbf{g}}(\mathbf{q};\psi)$$

$$(\theta^*, \psi^*) = \arg\min_{\theta,\psi} \ \ell\left(\hat{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \theta, \psi),\ \boldsymbol{\tau}\right)$$

with $$\hat{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \theta, \psi) = \hat{\mathbf{L}}\hat{\mathbf{L}}^T\ddot{\mathbf{q}} + \frac{d}{dt}\left(\hat{\mathbf{L}}\hat{\mathbf{L}}^T\right)\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T\hat{\mathbf{L}}\hat{\mathbf{L}}^T\dot{\mathbf{q}}\right)\right)^T + \hat{\mathbf{g}}$$

$$\text{s.t. } 0 < \mathbf{x}^T\ \hat{\mathbf{L}}\hat{\mathbf{L}}^T\mathbf{x} \quad \forall\, \mathbf{x} \in \mathbb{R}_0^n$$



$$\hat{f}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}; \theta, \psi) = \left(\hat{\mathbf{L}}\hat{\mathbf{L}}^T\right)^{-1}\left(\boldsymbol{\tau} - \frac{d}{dt}\left(\hat{\mathbf{L}}\hat{\mathbf{L}}^T\right)\dot{\mathbf{q}} + \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T\hat{\mathbf{L}}\hat{\mathbf{L}}^T\dot{\mathbf{q}}\right)\right)^T - \hat{\mathbf{g}}\right)$$

$$(\theta^*, \psi^*) = \arg\min_{\theta,\psi} \ \ell\left(\hat{f}^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}; \theta, \psi),\ \boldsymbol{\tau}\right) + \lambda\ \Omega(\theta, \psi)$$

Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning

$$\frac{d}{dt}\mathbf{H}(\mathbf{q}) = \frac{d}{dt}\left(\mathbf{L}\mathbf{L}^T\right) = \mathbf{L}\frac{d\mathbf{L}^T}{dt} + \frac{d\mathbf{L}}{dt}\mathbf{L}^T$$

derivative of the vectorized form l.

$$\frac{d}{dt}\mathbf{l} = \frac{\partial \mathbf{l}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial t} + \sum_{i=1}^{N}\frac{\partial \mathbf{l}}{\partial \mathbf{W}_i}\frac{\partial \mathbf{W}_i}{\partial t} + \sum_{i=1}^{N}\frac{\partial \mathbf{l}}{\partial \mathbf{b}_i}\frac{\partial \mathbf{b}_i}{\partial t}$$

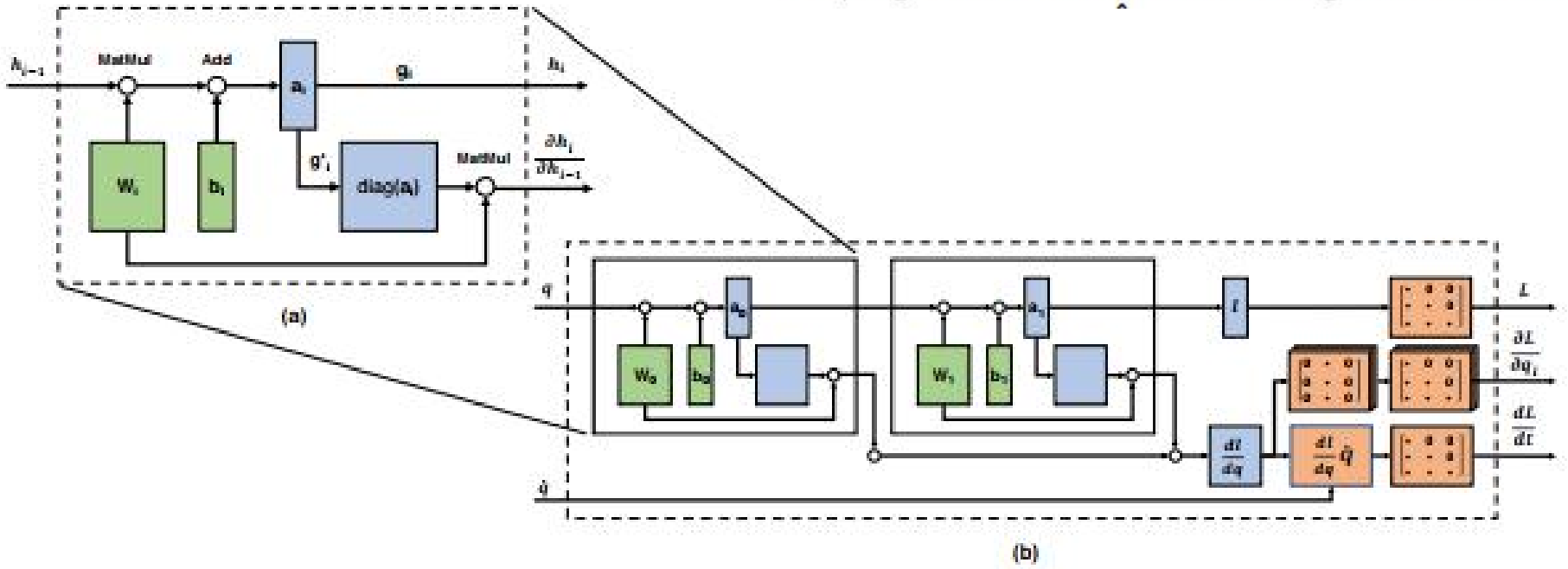
$$\frac{d}{dt}\mathbf{l} = \frac{\partial \mathbf{l}}{\partial \mathbf{q}}\dot{\mathbf{q}}.$$

$$\frac{\partial \mathbf{l}}{\partial \mathbf{q}} = \frac{\partial \mathbf{l}}{\partial \mathbf{h}_{N-1}}\frac{\partial \mathbf{h}_{N-1}}{\partial \mathbf{h}_{N-2}} \cdots \frac{\partial \mathbf{h}_1}{\partial \mathbf{q}}$$

$$\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \mathrm{diag}\left(g'(\mathbf{W}_i^T\mathbf{h}_{i-1} + \mathbf{b}_i)\right)\mathbf{W}_i$$

g' is the derivative of the non-linearity

$\partial \mathbf{L}/\partial \mathbf{q}_i$ can be constructed using the columns of previously derived $\partial \mathbf{l}/\partial \mathbf{q}$.



Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning

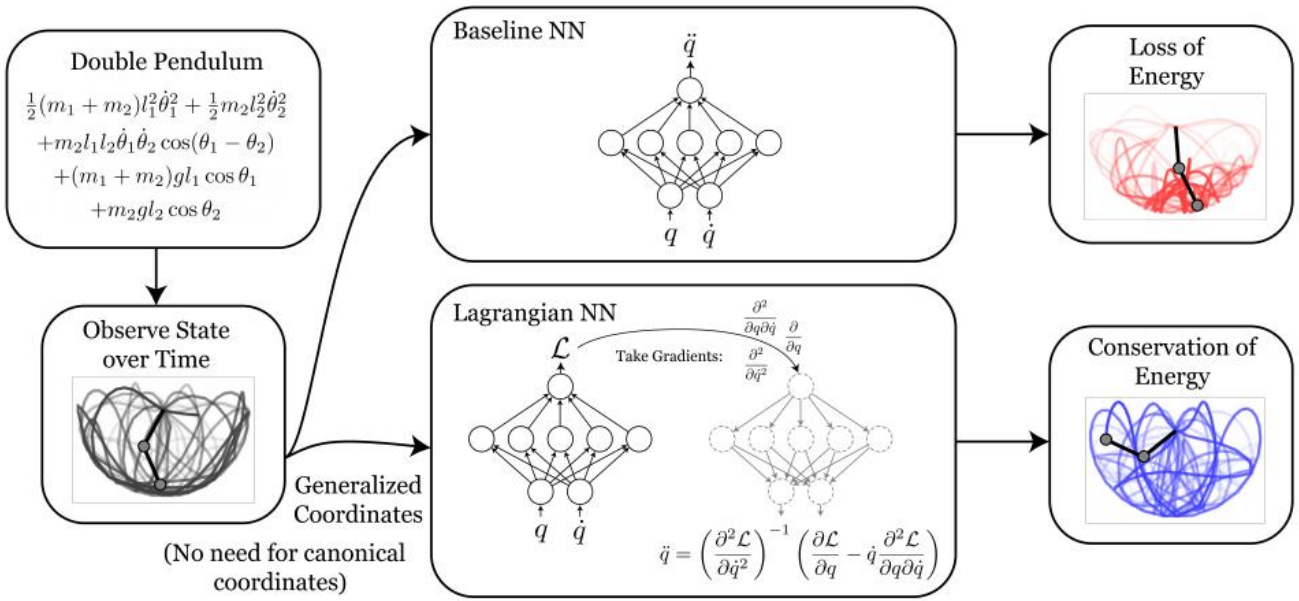
Cons of DeLAN: **restrict function form**

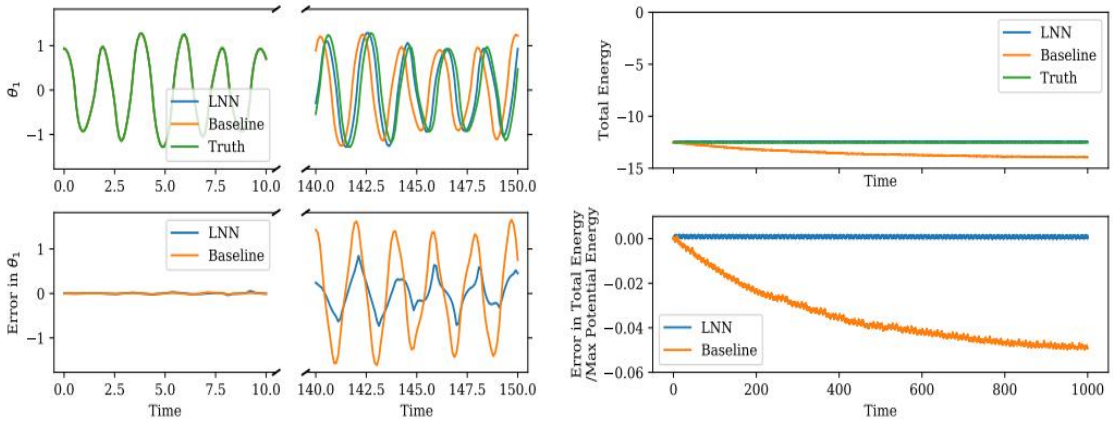
Chain rule: $\frac{d}{dt}\nabla_{\dot{q}}\mathcal{L} = \nabla_q \mathcal{L}$

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

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

	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					✓





(a) Error in angle (b) Error in energy

LAGRANGIAN NEURAL NETWORKS

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{Q}^{nc}$$

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{q}),$$

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\Big(-\frac{1}{2}\frac{\mathrm{d}\mathbf{M}(\mathbf{q})}{\mathrm{d}t}\dot{\mathbf{q}} - \frac{\mathrm{d}V(\mathbf{q})}{\mathrm{d}\mathbf{q}} + \mathbf{g}(\mathbf{q})\mathbf{u}\Big).$$

$$\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$$

Control via energy shaping

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{g}(\mathbf{q})\boldsymbol{\beta}(\mathbf{q}) \iff \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L_d}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L_d}{\partial \mathbf{q}} = 0,$$

$$L_d(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V_d(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V_d(\mathbf{q}).$$

$\beta(q)$ shapes the potential energy V of the original system into a desired potential energy V_d. The potential energy V_d is designed to have a global minimum at q^*.

$$\boldsymbol{\beta}(\mathbf{q}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\Big).$$

Assume:

$$\mathbf{v}(\dot{\mathbf{q}}) = -\mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}(\mathbf{K}_d\dot{\mathbf{q}}).$$

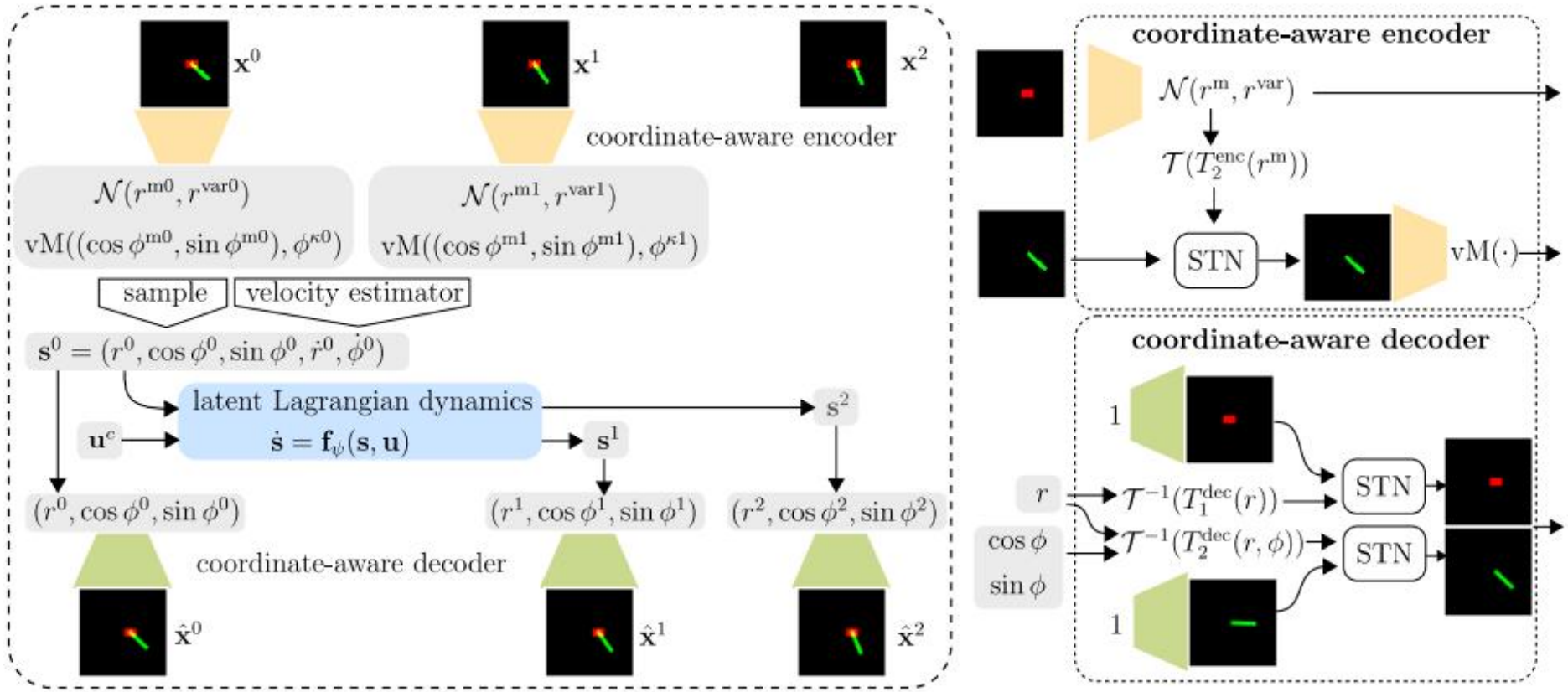
$$V_d(\mathbf{q}) = \frac{1}{2}(\mathbf{q} - \mathbf{q}^\star)^T \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star),$$

$$\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\left(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star) - \mathbf{K}_d\dot{\mathbf{q}}\right)$$

proportional-derivative (PD) controller with energy compensation.

Unsupervised Learning of Lagrangian Dynamics from Images for Prediction and

Training Neural ODE with constant control



Unsupervised Learning of Lagrangian Dynamics from Images for Prediction and

DNN & physical system

- **Hamiltonian**
  - **Hamilton Neural Network: learn Hamilton function**
  - **Symplectic recurrent neural network**
  - **SSINN: learn with sparse regression**
  - **NSSNN: train with nonseparable Hamiltonian function**
  - **Symplectic ode-net: add data control and connect with NODE**
  - **HGN&HNF: apply energy based model to generative models.**
- **Lagrangian**
  - **DeLAN: Design NN architecture based on Lagrangian function**
  - **LNN: free form of Lagrangian function**
  - Application: Learning from Images for Prediction and

Control/applied mathematics + DNN = ???

Looking forward to cooperation!

HAPPY NEW YEAR!