Continuous Network Models for Sequential Predictions

Michael W. Mahoney

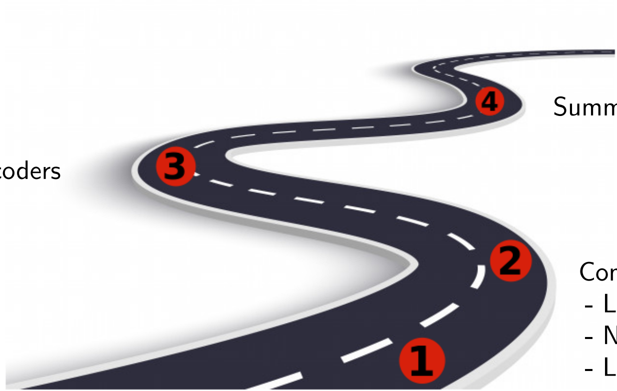
ICSI, LBNL, and Dept of Statistics, UC Berkeley

http://www.stat.berkeley.edu/~mmahoney/

January 2022

(Joint work with N. Benjamin Erichson (University of Pittsburgh); and also with Liam Hodgkinson (ICSI and UCB), Alejandro Queiruga (LBNL → Google), Soon Hoe Lim (KTH Royal Institute), and Omri Azencot (UCLA → Ben-Gurion).

Outline



Summary

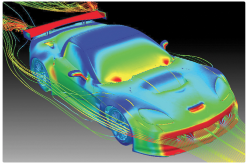
Dynamic Autoencoders

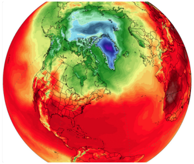
Continuous-time RNNs
- Lipschitz RNNs
- Noisy RNNs
- LEMs

Introduction

Introduction

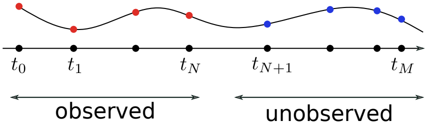
- Many interesting problems exhibit multi-scale and non-linear temporal structures, for instance, turbulent fluid flows, climate and vision.



- Neural networks provide a flexible and powerful framework for sequential data modeling.

- The aim is to map a sequence $\mathbf{x}_0, \ldots, \mathbf{x}_N \in \mathbb{R}^m$ to a target sequence $\mathbf{y}_0, \ldots, \mathbf{y}_K \in \mathbb{R}^d$.



But, Neural Networks are Brittle and Prone to Fail

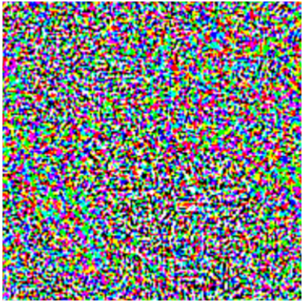
For Example, Adversarial Examples

clean example

"king penguin"
62.8% confidence

+

adversarial perturbation



=

adversarial example

"panda"
89.7% confidence

- Szegedy et al. "Intriguing properties of neural networks." ICLR (2014).
- Goodfellow et al. "Explaining and harnessing adversarial examples." ICLR (2015).

Leveraging Ideas from Dynamical Systems

- Viewing DNNs from a dynamical systems and control theoretic point of view provides us with powerful tools to study the stability and long-term behavior of neural networks.

ML applied to scientific applications (SciML)

Machine Learning

$$\boxed{y = F(x) + \epsilon}$$

Dynamical Systems

$$\boxed{\dot{y} = f(t, y, x)}$$

What can we learn from dynamical systems and control theory?

Network Architecture Design	$\longleftrightarrow$	Numerical Methods
Training	$\longleftrightarrow$	Stability and Control

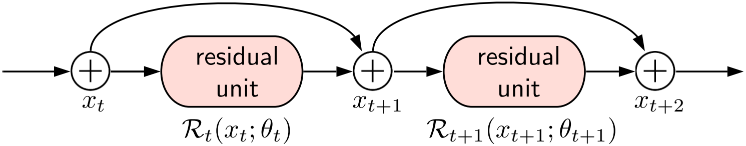
- **Hope:** The dynamical system perspective can help us to better understand black-box ML methods as well as to help us to design more robust models.

Connection between Deep Learning and Differential Equations

- The essential building blocks of ResNets are so-called residual units.

$$x_{t+1} = x_t + \mathcal{R}_t(x_t; \theta_t). \tag{1}$$

- The function $\mathcal{R}_t : \mathbb{R}^n \to \mathbb{R}^n$ denotes the t-th residual module (a non-linear map), parameterized by θ_t, which takes a signal $x_t \in \mathbb{R}^n$ as input and returns $x_{t+1} \in \mathbb{R}^n$.

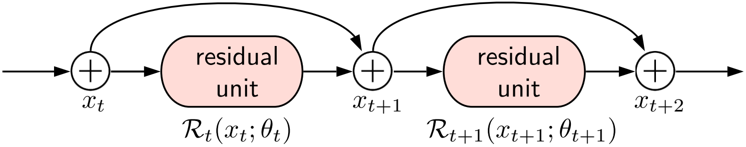


Connection between Deep Learning and Differential Equations

- The essential building blocks of ResNets are so-called residual units.

$$x_{t+1} = x_t + \mathcal{R}_t(x_t; \theta_t). \tag{1}$$

- The function $\mathcal{R}_t : \mathbb{R}^n \to \mathbb{R}^n$ denotes the t-th residual module (a non-linear map), parameterized by θ_t, which takes a signal $x_t \in \mathbb{R}^n$ as input and returns $x_{t+1} \in \mathbb{R}^n$.



- Now, consider an ODE that takes the form:

$$\frac{dx(t)}{dt} = \mathcal{R}(x(t); \theta_t). \tag{2}$$

- The form of forward Euler draws a connection between ODEs and residual units

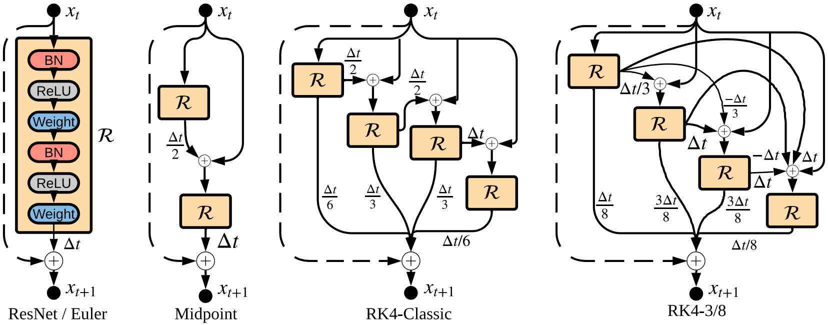
$$x_{t+1} = x_t + \Delta t \ \mathcal{R}(x_t; \theta_t)). \tag{3}$$

Connection between Architecture Design and Numerical Methods

- Algebraically, a residual unit *also* closely resembles many time-stepping schemes

$$x_{t+1} \approx x_t + \Delta t \cdot \texttt{scheme}\left[\mathcal{R},\, x_t,\, \Delta t\right],$$

where $\texttt{scheme}$ represents one step of a numerical integration scheme.



- This connection between numerical methods and neural networks has provided inspiration for the design of many new network architectures: PolyNet, FractalNet, RevNet, etc.

Numerical Integration and ML Work in Opposite Directions

- When a numerical integrator is used in scientific computing, we start with a differential equation, and we formulate a discrete model which can produce discrete data.
- Learning a model using a data-driven approach, however, works in the opposite direction, i.e., we fit a discrete model to a collection of discrete data points.

Numerical Integration

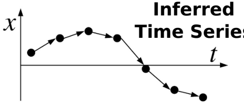
$$\frac{dx}{dt} = f$$

Ground Truth Dynamics ↓

$$x_{t+1} = x_t + \Delta t f(x_t)$$

Approximation yield data ↓

Inferred Time Series



Learning the Dynamics

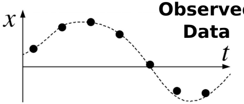
$$\mathcal{R}(x_t; \theta)$$

↑ Now, what is R?

$$\min \|x_{t+1} - (x_t + \Delta t \ \mathcal{R}(x_t; \theta))\|$$

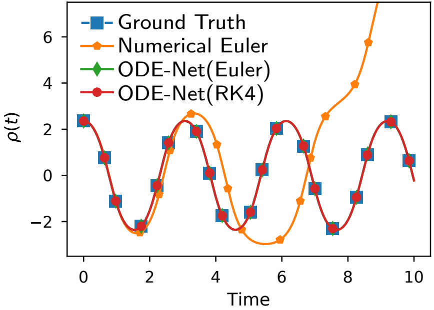
↑ Use data to learn a modell

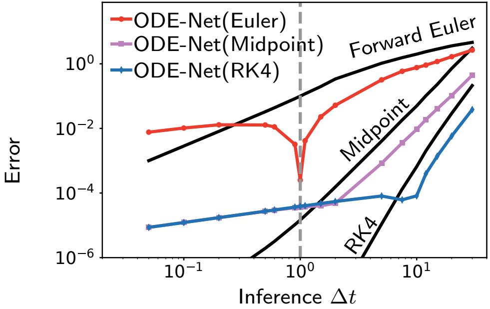
Observed Data



Example: Nonlinear Pendulum[1]

- It is ‘misleading’ to interpret an ODE-Net(Euler) model as a forward Euler discretization of the differential equation, despite having a similar algebraic form.
- That is, the analogy of a ResNet to forward Euler is superficial and at best incomplete.
- In contrast, ODE-Net(Midpoint) and ODE-Net(RK4) can meaningfully be interpreted as a discrete approximation to a continuous dynamical system.





[1] Continuous-in-Depth Neural Networks (arXiv:2008.02389)

Stability of Linear Neural Networks

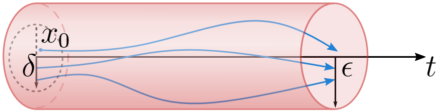
- We might be interested to study whether latent trajectories of a neural networks under small perturbations of the initial condition (input) x_0, are stable.

- This is easy to check for simple linear neural networks.

- A linear neural network $\mathbf{A} : \mathbb{R}^k \to \mathbb{R}^k$ that maps $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ given by

$$\mathbf{x}_{t+1} = \mathbf{A}(\mathbf{x}_t) \quad t = 0, 1, 2, \ldots$$

  is stable if all trajectories that starting arbitrarily close to the origin (in a ball of radius δ) remain arbitrarily close (in a ball of radius ϵ).

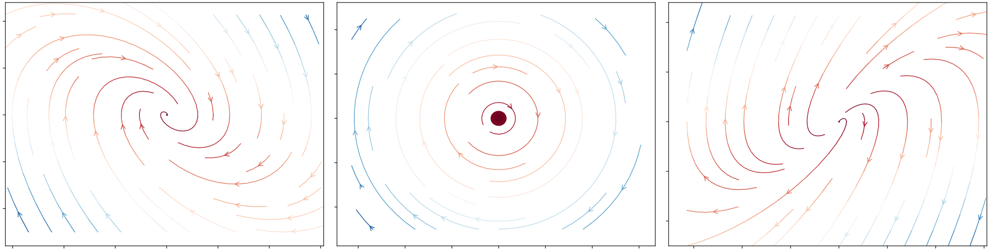


Vanishing and Exploding Gradients Problem

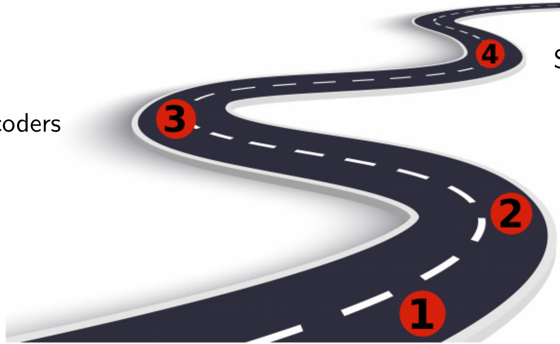
- Training gradient-based sequential models is difficult.
- Consider the following linear DT dynamical system $h_{t+1} = Ah_t$, then it follows

$$x_{t+s} = A^s x_t = (A_s \circ \cdots \circ A_1)(x_t) \tag{4}$$

- Depending on the eigenvalues of A, there are three situations that can occur



Outline



Summary

Dynamic Autoencoders

Continuous-time RNNs

- Lipschitz RNNs

- Noisy RNNs
- LEMs

Introduction

Basic Recurrent Unit

$$\overset{\text{state}}{h_{t+1}} = \overset{\text{nonlinearity}}{\sigma}(\underset{\text{hidden-to-hidden matrix}}{W} h_t + \underset{\text{input-to-hidden matrix}}{U} \overset{\text{input}}{x_t} + \underset{\text{bias}}{b})$$

- Recurrent units are networks with *feedback connections*.
- RNNs can use their hidden state (memory) to process variable length sequences of inputs.
- **Challenge:** RNNs are known to have stability issues and are difficult to train, most notably due to the vanishing and exploding gradients problem.

Lipschitz Recurrent Neural Network (ICLR 2021)[2]

- We can view RNNs as dynamical systems whose temporal evolutions are governed by an abstract system of differential equations with an external input:

$$\begin{cases} \dot{h}(t) = \sigma(Wh + Ux + b), & (5) \\ y = Dh. & (6) \end{cases}$$

[2] Lipschitz Recurrent Neural Networks (ICLR 2021).

Lipschitz Recurrent Neural Network (ICLR 2021)[2]

- We can view RNNs as dynamical systems whose temporal evolutions are governed by an abstract system of differential equations with an external input:

$$\begin{cases} \dot{h}(t) = \sigma(Wh + Ux + b), & (5) \\ y = Dh. & (6) \end{cases}$$

- We propose a continuous-time recurrent unit that describes the hidden state's evolution with two parts: a well-understood **linear component** plus a **Lipschitz nonlinearity**.

$$\dot{h} = Ah + \sigma(Wh + Ux + b)$$

We assume that the nonlinearity σ is an M-Lipschitz function.

[2] Lipschitz Recurrent Neural Networks (ICLR 2021).

Stability Analysis of Lipschitz Recurrent Units

- Assuming that σ is an M-Lipschitz function, we prove that our model is *globally exponentially stable* under some mild conditions on A and W.

Theorem 1

Let h^* be an equilibrium point of a DE of the form $\dot{h} = Ah + \sigma(Wh + Ux + b)$ for some $x \in \mathbb{R}^p$. The point h^* is globally exponentially stable if

- the eigenvalues of $A^{\text{sym}} := \frac{1}{2}(A + A^T)$ are strictly negative,
- W is non-singular,
- and $\sigma_{\min}(A^{\text{sym}}) > M\sigma_{\max}(W)$.

- The proof relies on the classical Kalman-Yakubovich-Popov lemma and circle criterion.
- Intuitively, global exponential stability is guaranteed if the matrix A has eigenvalues with real parts sufficiently negative to counteract expanding trajectories in the nonlinearity.

Symmetric-Skew Hidden-to-Hidden Matrices

- We propose the following symmetric-skew decomposition for constructing hidden matrices:

$$S_{\beta,\gamma} := (1-\beta)\cdot(M+M^T)+\beta\cdot(M-M^T)-\gamma I. \tag{7}$$

- Using this construction, we can easily bound the spectrum via the parameters β and γ.

Proposition 1

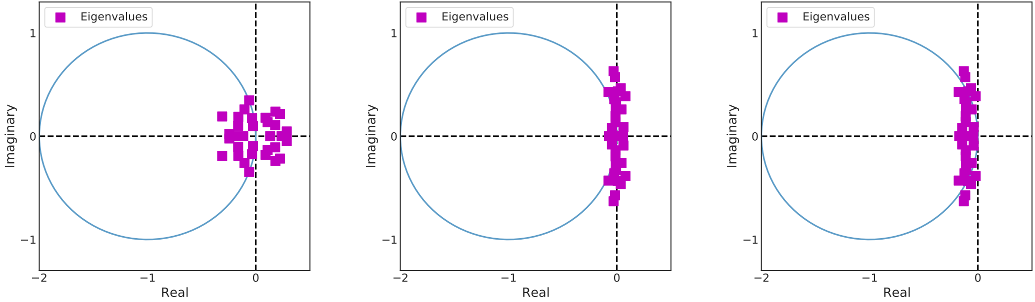
Let $S_{\beta,\gamma}$ satisfy (7), and let $M^{\text{sym}} = \frac{1}{2}(M+M^T)$. The real parts $\Re\lambda_i(S_{\beta,\gamma})$ of the eigenvalues of $S_{\beta,\gamma}$ lie in the interval

$$[(1-\beta)\lambda_{\min}(M^{\text{sym}})-\gamma, (1-\beta)\lambda_{\max}(M^{\text{sym}})-\gamma].$$

Illustration of the Symmetric-Skew Decomposition

- Our symmetric-skew scheme allows us to construct hidden-to-hidden matrices that exhibit dynamics with moderate decay and growth behavior.

- This helps to mitigate the problem of exploding and vanishing gradients.



(a) $\beta = 0.5,\ \gamma = 0$ (b) $\beta = 0.75,\ \gamma = 0.00$ (c) $\beta = 0.75,\ \gamma = 0.1$

Putting it All Together

- Our Lipschitz recurrent neural network takes the functional form:

$$\begin{cases} \dot{h} &= A_{\beta_A,\gamma_A} h + \tanh(W_{\beta_W,\gamma_W} h + Ux + b) \;, & \text{(8a)} \\ y &= Dh \;, & \text{(8b)} \end{cases}$$

- The hidden-to-hidden matrices $A_{\beta,\gamma} \in \mathbb{R}^{N\times N}$ and $W_{\beta,\gamma} \in \mathbb{R}^{N\times N}$ are of the form

$$\begin{cases} A_{\beta_A,\gamma_A} &= (1-\beta_A)(M_A + M_A^T) + \beta_A(M_A - M_A^T) - \gamma_A I & \text{(9a)} \\ W_{\beta_W,\gamma_W} &= (1-\beta_W)(M_W + M_W^T) + \beta_W(M_W - M_W^T) - \gamma_W I, & \text{(9b)} \end{cases}$$

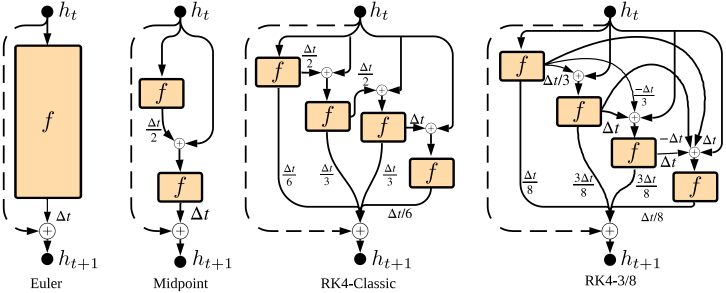
where $\beta_A, \beta_W \in [0,1]$, $\gamma_A, \gamma_W > 0$ are parameters and $M_A, M_W \in \mathbb{R}^{N\times N}$ are matrices.

Training Continuous-time Recurrent Units

- Letting $f(h,t) = Ah + \tanh(Wh + Ux(t) + b)$ so that $\dot{h}(t) = f(h,t)$, the approximate solutions for h_{t+1} given h_t are given by

$$h_{t+1} \approx h_t + \Delta t \cdot \texttt{scheme}\left[f,\, h_t,\, \Delta t\right] \;,$$

where $\texttt{scheme}$ represents one step of a numerical integration scheme.

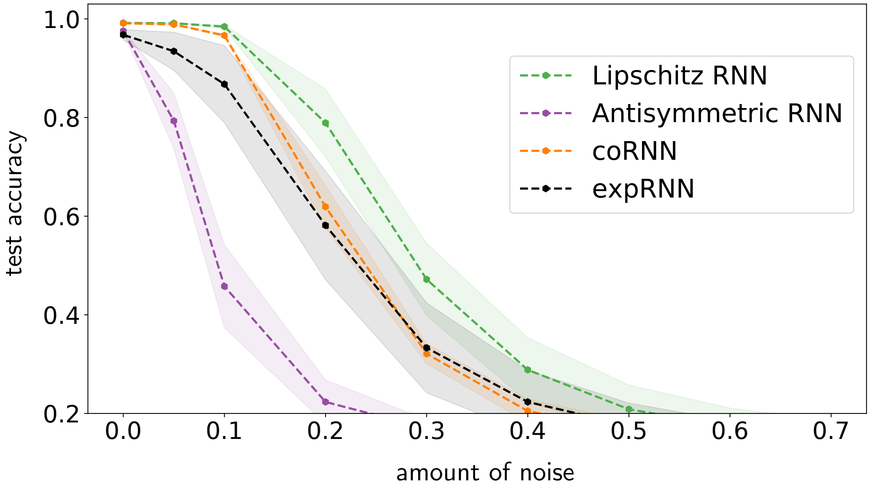


Empirical Evaluation

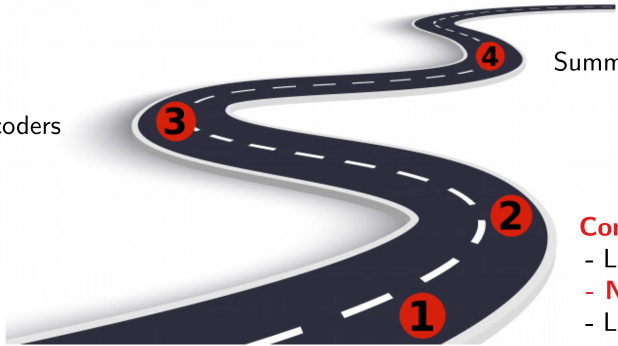
Table 1: Evaluation accuracy on ordered and permuted pixel-by-pixel MNIST.

Name	ordered	permuted	N	# params
LSTM baseline by (Arjovsky et al., 2016)	97.3%	92.7%	128	≈68K
MomentumLSTM (Nguyen et al., 2020)	99.1%	94.7%	256	≈270K
Unitary RNN (Arjovsky et al., 2016)	95.1%	91.4%	512	≈9K
Full Capacity Unitary RNN (Wisdom et al., 2016)	96.9%	94.1%	512	≈270K
Soft orth. RNN (Vorontsov et al., 2017)	94.1%	91.4%	128	≈18K
Kronecker RNN (Jose et al., 2018)	96.4%	94.5%	512	≈11K
Antisymmteric RNN (Chang et al., 2019)	98.0%	95.8%	128	≈10K
Incremental RNN (Kag et al., 2020)	98.1%	95.6%	128	≈4K/8K
Exponential RNN (Lezcano-Casado & Martinez-Rubio, 2019)	98.4%	96.2%	360	≈69K
Sequential NAIS-Net (Ciccone et al., 2018)	94.3%	90.8%	128	≈18K
Lipschitz RNN using Euler (ours)	99.0%	94.2%	64	≈9K
Lipschitz RNN using RK2 (ours)	99.1%	94.2%	64	≈9K
Lipschitz RNN using Euler (ours)	**99.4%**	**96.3%**	128	≈34K
Lipschitz RNN using RK2 (ours)	99.3%	96.2%	128	≈34K

Robustness with Respect to Input Perturbations



Outline



Summary

Dynamic Autoencoders

Continuous-time RNNs

- Lipschitz RNNs
- **Noisy RNNs**
- LEMs

Introduction

From Good Old RNNs to SDEs: An Illustration

- Two steps:
  (1) Adding leaky integrator ("damping" term):
  $$h_{t+1} = \alpha h_t + \beta f(h_t, x_t) \tag{10}$$
  (2) Injecting noise (various motivations and benefits):
  $$h_{t+1} = \alpha h_t + \beta f(h_t, x_t) + \theta \xi_t, \quad \alpha, \beta, \theta > 0, \tag{11}$$
  where the ξ_t are i.i.d. random vectors (e.g., zero mean Gaussian)

From Good Old RNNs to SDEs: An Illustration

- Two steps:
  (1) Adding leaky integrator ("damping" term):
  $$h_{t+1} = \alpha h_t + \beta f(h_t, x_t) \tag{10}$$
  (2) Injecting noise (various motivations and benefits):
  $$h_{t+1} = \alpha h_t + \beta f(h_t, x_t) + \theta \xi_t, \quad \alpha, \beta, \theta > 0, \tag{11}$$
  where the ξ_t are i.i.d. random vectors (e.g., zero mean Gaussian)

- Setting $\alpha = 1 - \gamma \Delta t$, $\beta = \Delta t$, $\theta = \sqrt{\Delta t}\sigma$ and $\xi_t =$ i.i.d. standard Gaussian, we get Euler-Mayurama approximation of the stochastic differential equation:
  $$dh_t = -\gamma h_t dt + f(h_t, x_t)dt + \sigma dB_t, \quad t \in [0, T], \tag{12}$$
  where $(B_t)_{t \geq 0}$ is a Brownian motion.

Noisy Recurrent Neural Networks (NeurIPS 2021)

Let x be an input signal, we consider the following (Itô) SDE model

$$\mathrm{d}\, h_t = f(h_t, x_t)\, \mathrm{d}\, t + \sigma(h_t, x_t)\, \mathrm{d}\, B_t, \qquad y_t = V h_t, \tag{13}$$

where $(B_t)_{t \geq 0}$ is an r-dimensional Brownian motion.

- The functions f and σ are referred to as the *drift* and *diffusion* coefficients, respectively.

Noisy Recurrent Neural Networks (NeurIPS 2021)

Let x be an input signal, we consider the following (Itô) SDE model

$$\mathrm{d}\, h_t = f(h_t, x_t)\,\mathrm{d}\, t + \sigma(h_t, x_t)\,\mathrm{d}\, B_t, \qquad y_t = V h_t, \tag{13}$$

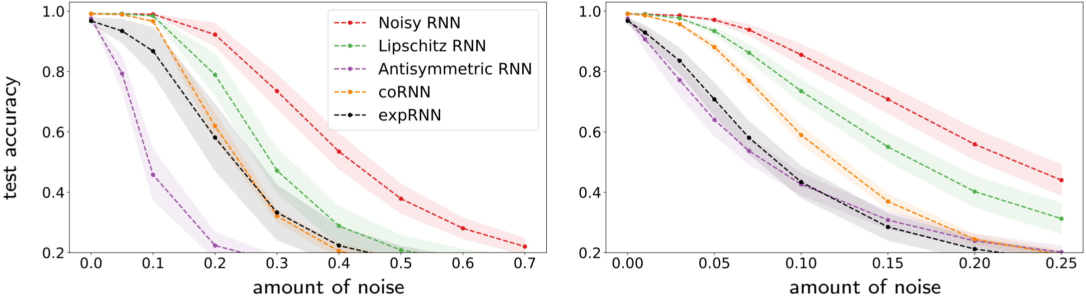
where $(B_t)_{t\geq 0}$ is an r-dimensional Brownian motion.

- The functions f and σ are referred to as the *drift* and *diffusion* coefficients, respectively.
  - $$f(h,x) = Ah + a(Wh + Ux + b), \tag{14}$$
    where $a : \mathbb{R} \to \mathbb{R}$ is a Lipschitz continuous scalar activation function.
  - $$\sigma(h,x) = \epsilon(\sigma_1 I + \sigma_2 \mathrm{diag}(f(h,x))), \tag{15}$$
    where the noise level $\epsilon > 0$ is small, and $\sigma_1 \geq 0$ and $\sigma_2 \geq 0$ are tunable parameters
- Noise injection can be viewed as a stochastic learning strategy used to improve robustness of the learning model against data perturbations.

Robustness with Respect to Input Perturbations

Table 1: Robustness w.r.t. white noise (σ) and S&P (α) perturbations on the ordered MNIST task.

Name	clean	$\sigma = 0.1$	$\sigma = 0.2$	$\sigma = 0.3$	$\alpha = 0.03$	$\alpha = 0.05$	$\alpha = 0.1$
Antisymmetric RNN	97.5%	45.7%	22.3%	17.0%	77.1%	63.9%	42.6%
CoRNN	99.1%	96.6%	61.9%	32.1%	95.6%	88.1%	58.9%
Exponential RNN	96.7%	86.7%	58.1%	33.3%	83.6%	70.7%	43.4%
Lipschitz RNN	**99.2%**	98.4%	78.9%	47.1%	97.6%	93.4%	73.5%
Noisy RNN (mult./add. noise: 0.02/0.05)	99.1%	**98.9%**	**92.2%**	**73.5%**	**98.5%**	**97.1%**	**85.5%**

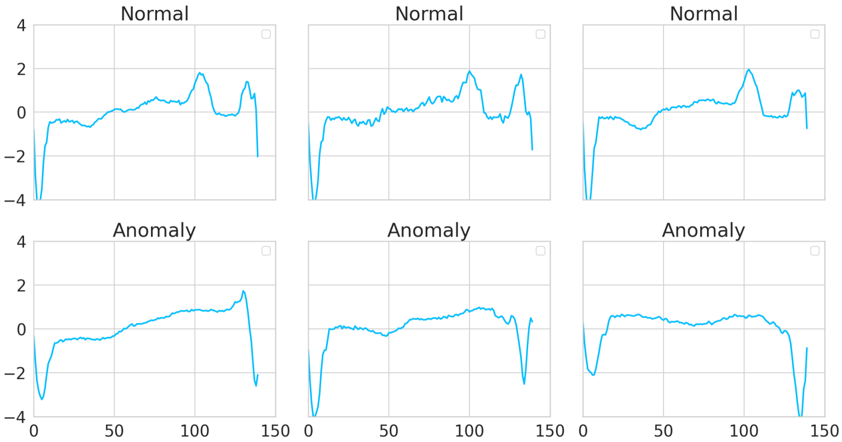


(a) White noise perturbations.

(b) Salt and pepper perturbations.

Electrocardiogram (ECG) Classification

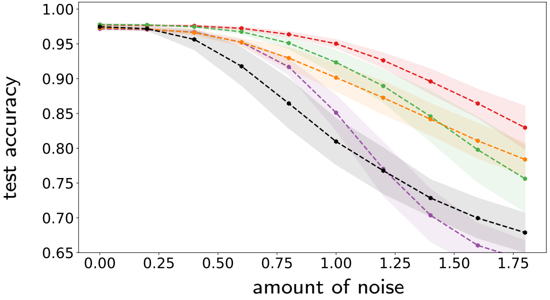
- The Electrocardiogram (ECG) classification task aims to discriminate between normal and abnormal heart beats of a patient that has severe congestive heart failure.
- We use 500 sequences of length 140 for training, and $4,000$ sequences for testing.



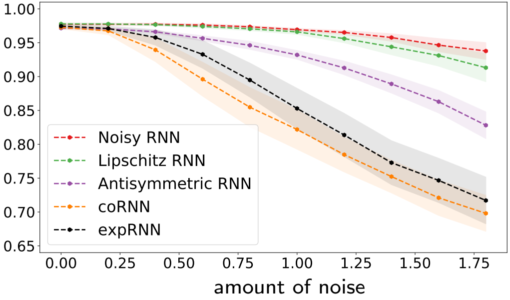
Results for Electrocardiogram (ECG) Classification

Table 2: Robustness w.r.t. white (σ) and multiplicative (σ_M) noise perturbations on the ECG task.

Name	clean	$\sigma = 0.4$	$\sigma = 0.8$	$\sigma = 1.2$	$\sigma_M = 0.4$	$\sigma_M = 0.8$	$\sigma_M = 1.2$
Antisymmetric RNN	97.1%	96.6%	91.6%	77.0%	96.6%	94.6%	91.2%
CoRNN	97.5%	96.8%	92.9%	87.2%	93.9%	85.4%	78.4%
Exponential RNN	97.4%	95.6%	86.4%	76.7%	95.7%	89.4%	81.3%
Lipschitz RNN	**97.7%**	97.4%	95.1%	88.9%	97.6%	97.0%	95.6%
Noisy RNN (mult./add. noise: 0.03/0.06)	**97.7%**	**97.5%**	**96.3%**	**92.6%**	**97.7%**	**97.3%**	**96.5%**



(a) Additive white noise perturbations.



(b) Multiplicative white noise perturbations.

Regularization Induced by Noise Injection

Theorem 2: Implicit regularization induced by noise injection

Under mild assumption on f and σ,

$$\mathbb{E}\ell(h_M^\delta) = \ell(\bar{h}_M^\delta) + \frac{\epsilon^2}{2}[\hat{Q}(\bar{h}^\delta) + \hat{R}(\bar{h}^\delta)] + \mathcal{O}(\epsilon^3), \tag{16}$$

as $\epsilon \to 0$, where the terms $\hat{Q}$ and $\hat{R}$ are given by

$$\hat{Q}(\bar{h}^\delta) = \nabla l(\bar{h}_M^\delta)^T \sum_{k=1}^{M} \delta_{k-1} \hat{\Phi}_{M-1,k} \sum_{m=1}^{M-1} \delta_{m-1} \operatorname{vec} v_m, \tag{17}$$

$$\hat{R}(\bar{h}^\delta) = \sum_{m=1}^{M} \delta_{m-1} \mathit{tr}(\sigma_{m-1}^T \hat{\Phi}_{M-1,m}^T H_{\bar{h}^\delta} l\ \hat{\Phi}_{M-1,m} \sigma_{m-1}), \tag{18}$$

with $\operatorname{vec} v_m$ a vector with the pth component $(p = 1, \ldots, d_h)$:

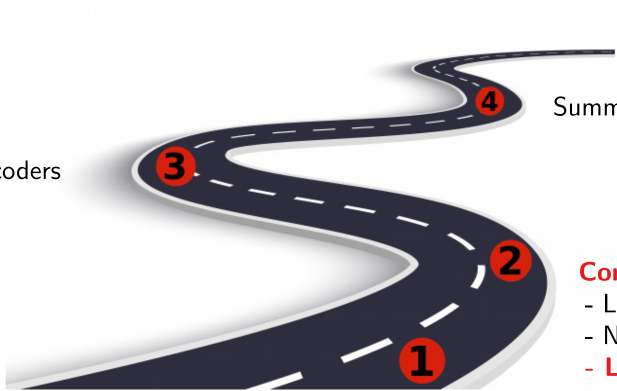
$$[v_m]^p = \mathit{tr}(\sigma_{m-1}^T \hat{\Phi}_{M-2,m}^T H_{\bar{h}^\delta} [f_M]^p \hat{\Phi}_{M-2,m} \sigma_{m-1}). \tag{19}$$

Moreover,

$$|\hat{Q}(\bar{h}^\delta)| \leq C_Q \Delta^2, \quad |\hat{R}(\bar{h}^\delta)| \leq C_R \Delta, \tag{20}$$

for $C_Q, C_R > 0$ independent of Δ.

Outline



Summary

Dynamic Autoencoders

Continuous-time RNNs
- Lipschitz RNNs
- Noisy RNNs
- **LEMs**

Introduction

From Multi-Resolution to Long Expressive Memory Units

- A simple example of a system of *two-scale ODEs* is given by

$$\frac{d\boldsymbol{y}}{dt} = \tau_y \left(\sigma \left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{x} + \mathbf{b}_y \right) - \boldsymbol{y} \right), \quad \frac{d\boldsymbol{z}}{dt} = \tau_z \left(\sigma \left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{x} + \mathbf{b}_z \right) - \boldsymbol{z} \right). \tag{21}$$

- Here τ_y and τ_z are the two time scales,
- $\boldsymbol{y}(t) \in \mathbb{R}^{d_y}$, and $\boldsymbol{z}(t) \in \mathbb{R}^{d_z}$ are the vectors of *slow* and *fast* variables.

From Multi-Resolution to Long Expressive Memory Units

- A simple example of a system of *two-scale ODEs* is given by

$$\frac{d\boldsymbol{y}}{dt} = \tau_y \left(\sigma\left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{x} + \mathbf{b}_y\right) - \boldsymbol{y}\right), \quad \frac{d\boldsymbol{z}}{dt} = \tau_z \left(\sigma\left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{x} + \mathbf{b}_z\right) - \boldsymbol{z}\right). \tag{21}$$

- Here τ_y and τ_z are the two time scales,
- $\boldsymbol{y}(t) \in \mathbb{R}^{d_y}$, and $\boldsymbol{z}(t) \in \mathbb{R}^{d_z}$ are the vectors of *slow* and *fast* variables.

- Two scales (one fast and one slow) are not enough for for complicate problems, in practice.

From Multi-Resolution to Long Expressive Memory Units

- A simple example of a system of *two-scale ODEs* is given by

$$\frac{d\boldsymbol{y}}{dt} = \tau_y \left(\sigma\left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{x} + \mathbf{b}_y\right) - \boldsymbol{y}\right), \quad \frac{d\boldsymbol{z}}{dt} = \tau_z \left(\sigma\left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{x} + \mathbf{b}_z\right) - \boldsymbol{z}\right). \tag{21}$$

- Here τ_y and τ_z are the two time scales,
- $\boldsymbol{y}(t) \in \mathbb{R}^{d_y}$, and $\boldsymbol{z}(t) \in \mathbb{R}^{d_z}$ are the vectors of *slow* and *fast* variables.

- Two scales (one fast and one slow) are not enough for for complicate problems, in practice.

- We can generalize this idea to a *multiscale* version, provided by the following set of ODEs,

$$\begin{aligned} \frac{d\boldsymbol{y}}{dt} &= \hat{\sigma}\left(\mathbf{W}_2 \boldsymbol{y} + \mathbf{V}_2 \mathbf{u} + \mathbf{b}_2\right) \odot \left(\sigma\left(\mathbf{W}_y \boldsymbol{z} + \mathbf{V}_y \mathbf{u} + \mathbf{b}_y\right) - \boldsymbol{y}\right), \\ \frac{d\boldsymbol{z}}{dt} &= \hat{\sigma}\left(\mathbf{W}_1 \boldsymbol{y} + \mathbf{V}_1 \mathbf{u} + \mathbf{b}_1\right) \odot \left(\sigma\left(\mathbf{W}_z \boldsymbol{y} + \mathbf{V}_z \mathbf{u} + \mathbf{b}_z\right) - \boldsymbol{z}\right). \end{aligned} \tag{22}$$

Training Long Expressive Memory Units

- We discretize the system of ODEs with using an implicit-explicit time-stepping scheme

$$
\begin{aligned}
\mathbf{\Delta t}_n &= \Delta t \hat{\sigma}(\mathbf{W}_1 \boldsymbol{y}_{n-1} + \mathbf{V}_1 \mathbf{u}_n + \mathbf{b}_1), \\
\overline{\mathbf{\Delta t}}_n &= \Delta t \hat{\sigma}(\mathbf{W}_2 \boldsymbol{y}_{n-1} + \mathbf{V}_2 \mathbf{u}_n + \mathbf{b}_2), \\
\boldsymbol{z}_n &= (1 - \mathbf{\Delta t}_n) \odot \boldsymbol{z}_{n-1} + \mathbf{\Delta t}_n \odot \sigma(\mathbf{W}_z \boldsymbol{y}_{n-1} + \mathbf{V}_z \mathbf{u}_n + \mathbf{b}_z), \\
\boldsymbol{y}_n &= (1 - \overline{\mathbf{\Delta t}}_n) \odot \boldsymbol{y}_{n-1} + \overline{\mathbf{\Delta t}}_n \odot \sigma(\mathbf{W}_y \boldsymbol{z}_n + \mathbf{V}_y \mathbf{u}_n + \mathbf{b}_y).
\end{aligned}
\tag{23}
$$

- The particular model mitigates the exploding and vanishing gradients problem.
- We can show that this model is a universal approximation of general dynamical systems,
- and also a universal approximation of multiscale dynamical systems.

Results

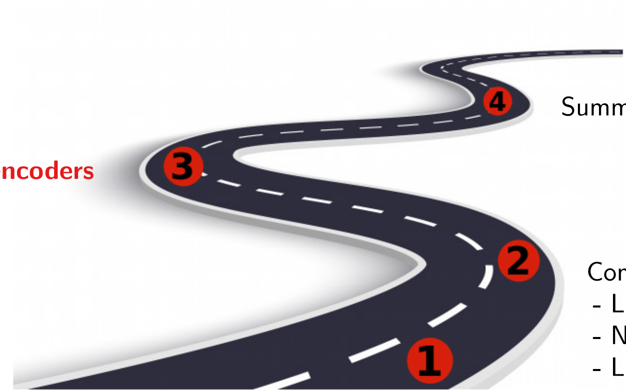
Table 3: Test accuracies on Google12 (benchmark for speech recognition).

Model	test accuracy	# units	# params
tanh-RNN	73.4%	128	27k
LSTM	94.9%	128	107k
GRU	95.2%	128	80k
expRNN	92.3%	128	19k
coRNN	94.7%	128	44k
LEM	**95.7%**	128	107k

Table 4: Test L^2 error on heart-rate prediction.

Model	test L^2 error	# units	# params
LSTM	9.93	128	67k
expRNN	1.63	256	34k
coRNN	1.61	128	34k
UnICORNN (3 layers)	1.31	128	34k
LEM	**0.85**	128	67k

Outline



Summary

Dynamic Autoencoders

Continuous-time RNNs
- Lipschitz RNNs
- Noisy RNNs
- LEMs

Introduction

A Naive Deep Learning Approach for Sequence Modeling

- Train a neural network $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}^m$ that learns the following map

$$\hat{\mathbf{x}}_{t+1} = \mathcal{F}(\mathbf{x}_t), \quad t = 0, 1, 2, \ldots, T.$$

- During inference time, we can obtain predictions by composing the model k-times

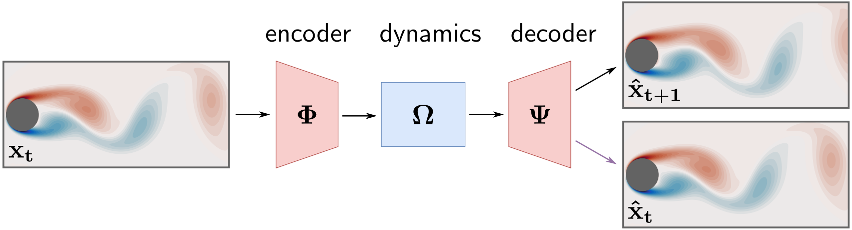
$$\hat{\mathbf{x}}_{t+k} = \mathcal{F} \circ \mathcal{F} \circ \mathcal{F} \circ \ \ldots \ \circ \mathcal{F}(\mathbf{x}_t).$$

- This approach typically fails to provide accurate predictions over long time horizons.

- Further, $\mathcal{F}$ is difficult to analyze and ignores any prior knowledge.

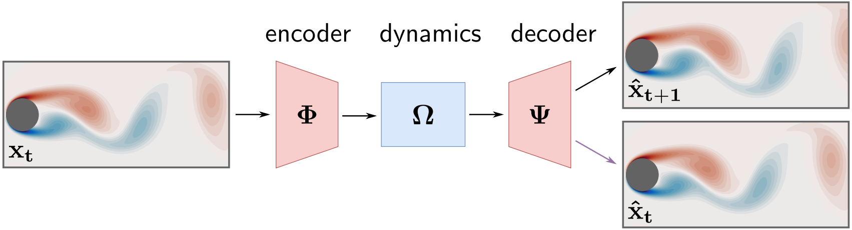
Dynamic Autoencoders

- The general idea is to design a model that consists of three components.
  - Non-linear encoder $\mathbf{\Psi}$: embeds inputs in a low-dimensional latent space.
  - A linear forward map $\mathbf{\Omega}$: evolves latent variables in time.
  - Non-linear decoder $\mathbf{\Phi}$: lifts latent variables back in high-dimensional space.



Dynamic Autoencoders

- The general idea is to design a model that consists of three components.
  - Non-linear encoder $\boldsymbol{\Psi}$: embeds inputs in a low-dimensional latent space.
  - A linear forward map $\boldsymbol{\Omega}$: evolves latent variables in time.
  - Non-linear decoder $\boldsymbol{\Phi}$: lifts latent variables back in high-dimensional space.



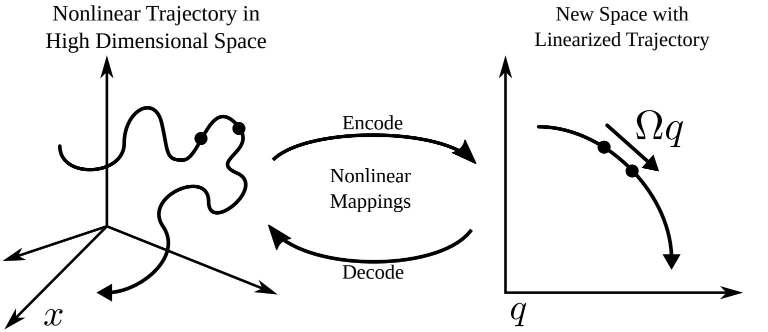
- We train the DAE by balancing between the forward prediction loss and reconstruction loss

$$\min \frac{1}{T}\sum_{t=0}^{T-1} \underbrace{\|\mathbf{x}_{t+k} - \boldsymbol{\Phi}\circ\boldsymbol{\Omega}_k\circ\cdots\circ\boldsymbol{\Omega}_1\circ\boldsymbol{\Psi}(\mathbf{x}_t)\|_2^2}_{\text{prediction loss}} + \lambda\underbrace{\|\mathbf{x}_t - \boldsymbol{\Phi}\circ\boldsymbol{\Psi}(\mathbf{x}_t)\|_2^2}_{\text{reconstruction loss}}.$$

Dynamic Autoencoders Learn Coordinate Transformations

- The key observation is that we can learn a coordinate transformation so that the latent variables $\mathbf{q}_t = \mathbf{\Psi}(\mathbf{x}_k)$ can be evolved in time by a linear map

$$\hat{\mathbf{q}}_{t+k} = \mathbf{\Omega}^k \mathbf{q}_t.$$



Motivated by Applied Koopmanism

- Koopman analysis provides a framework to study nonlinear dynamical system that is based on a coordinate transformation which embeds a nonlinear system in a space where the temporal evolution can be described by a linear operator.

$$\underbrace{\mathbf{x}_{n+1} = \mathcal{A}(\mathbf{x}_n)}_{\text{nonlinear}} \quad \to \quad \underbrace{z_n := g(\mathbf{x}_n)}_{\text{coordinate transform}} \quad \to \quad \underbrace{z_{n+1} = \mathcal{K}\, z_n}_{\text{linear}}$$

- $\mathcal{K}$ is a linear operator that evolves the observables $g(\mathbf{x}_n)$ in time.

Motivated by Applied Koopmanism

- Koopman analysis provides a framework to study nonlinear dynamical system that is based on a coordinate transformation which embeds a nonlinear system in a space where the temporal evolution can be described by a linear operator.

$$\underbrace{\mathbf{x}_{n+1} = \mathcal{A}(\mathbf{x}_n)}_{\text{nonlinear}} \quad \rightarrow \quad \underbrace{z_n := g(\mathbf{x}_n)}_{\text{coordinate transform}} \quad \rightarrow \quad \underbrace{z_{n+1} = \mathcal{K}\, z_n}_{\text{linear}}$$

- $\mathcal{K}$ is a linear operator that evolves the observables $g(\mathbf{x}_n)$ in time.

$$\begin{tikzcd}
\boxed{\mathbf{x}_n} \arrow[r, "\mathcal{A}"] \arrow[d] & \boxed{\mathbf{x}_{n+1}} \arrow[r, "\mathcal{A}"] \arrow[d] & \boxed{\mathbf{x}_{n+2}} \arrow[d] \\
\boxed{z_n} \arrow[r, "\mathcal{K}"] & \boxed{z_{n+1}} \arrow[r, "\mathcal{K}"] & \boxed{z_{n+2}}
\end{tikzcd}$$

Motivated by Applied Koopmanism

- Koopman analysis provides a framework to study nonlinear dynamical system that is based on a coordinate transformation which embeds a nonlinear system in a space where the temporal evolution can be described by a linear operator.

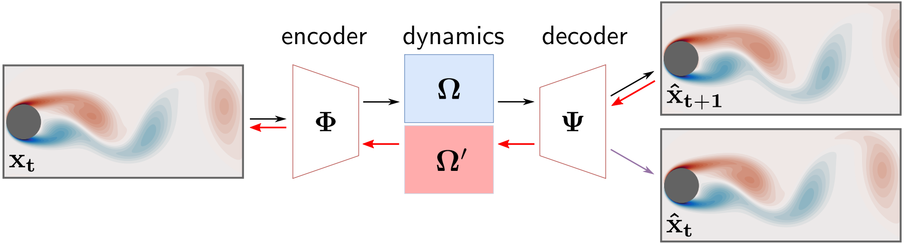
$$\underbrace{\mathbf{x}_{n+1} = \mathcal{A}(\mathbf{x}_n)}_{\text{nonlinear}} \quad \rightarrow \quad \underbrace{z_n := g(\mathbf{x}_n)}_{\text{coordinate transform}} \quad \rightarrow \quad \underbrace{z_{n+1} = \mathcal{K}\, z_n}_{\text{linear}}$$

- $\mathcal{K}$ is a linear operator that evolves the observables $g(\mathbf{x}_n)$ in time.

$$\begin{tikzcd}
\boxed{\mathbf{x}_n} \arrow[r, "\mathcal{A}"] \arrow[d] & \boxed{\mathbf{x}_{n+1}} \arrow[r, "\mathcal{A}"] \arrow[d] & \boxed{\mathbf{x}_{n+2}} \arrow[d] \\
\boxed{z_n} \arrow[r, "\mathcal{K}"] & \boxed{z_{n+1}} \arrow[r, "\mathcal{K}"] & \boxed{z_{n+2}} \\
\boxed{\mathbf{q}_n} \arrow[r, "\mathbf{\Omega}"] \arrow[u, dashed] & \boxed{\mathbf{q}_{n+1}} \arrow[r, "\mathbf{\Omega}"] \arrow[u, dashed] & \boxed{\mathbf{q}_{n+2}} \arrow[u, dashed]
\end{tikzcd}$$

Consistent Dynamic Autoencoders (ICML 2020) [3]

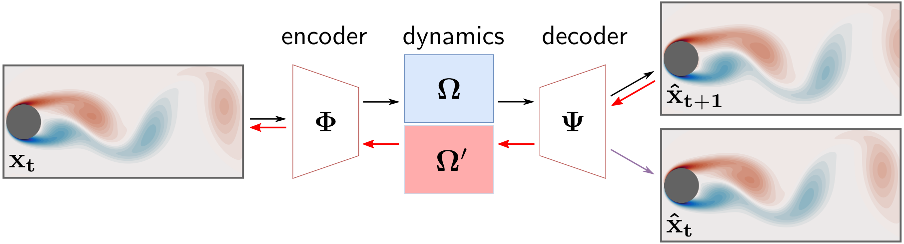
- We can learn a model that learns both a forward map $\boldsymbol{\Omega}$ and backward map $\boldsymbol{\Omega}'$.



[3] Forecasting Sequential Data Using Consistent Koopman Autoencoders (ICML, 2020).

Consistent Dynamic Autoencoders (ICML 2020) [3]

- We can learn a model that learns both a forward map $\boldsymbol{\Omega}$ and backward map $\boldsymbol{\Omega}'$.



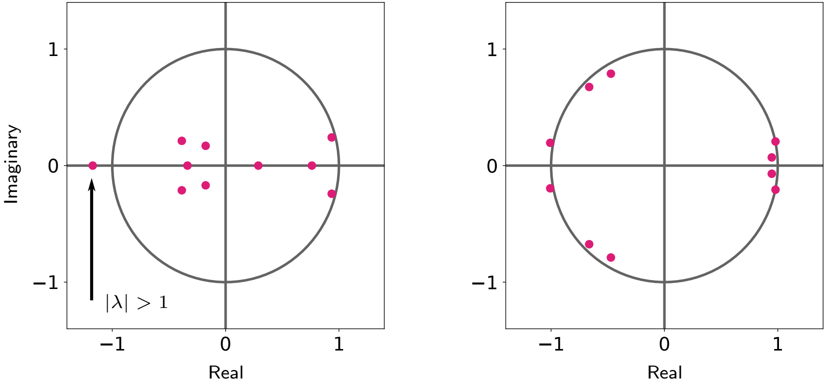
- Then we can promote consistency by considering the following loss term

$$\rho_c = \sum_{k=1}^{\kappa} \frac{1}{2k} \|\boldsymbol{\Omega}_{k*}\boldsymbol{\Omega}'_{*k} - \mathbf{I}_k\|_F^2 + \frac{1}{2k} \|\boldsymbol{\Omega}'_{*k}\boldsymbol{\Omega}_{k*} - \mathbf{I}_k\|_F^2 \ , \tag{24}$$

where $\boldsymbol{\Omega}'_{k*}$ and $\boldsymbol{\Omega}_{*k}$ are the upper k rows of $\boldsymbol{\Omega}'$ and leftmost k columns of the matrix $\boldsymbol{\Omega}$.

[3] Forecasting Sequential Data Using Consistent Koopman Autoencoders (ICML, 2020).

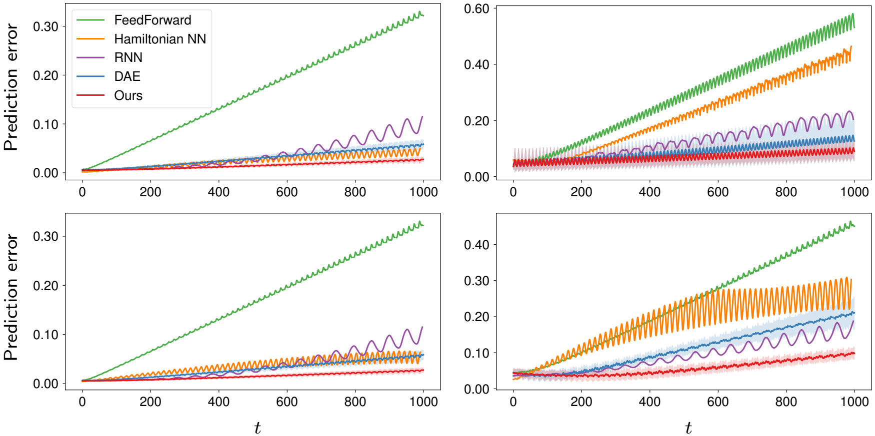
Consistency Stabilizes Weights



(a) Standard DAE (b) Consistent DAE

High-dimensional Nonlinear Pendulum with no Friction

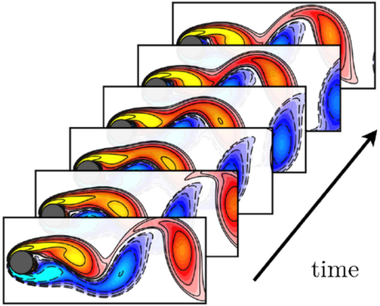
Prediction errors, over a time horizon of 1000 steps, for clean and noisy observations from a pendulum with initial conditions $\theta_0 = 0.8$ (top row) and $\theta_0 = 2.4$ (bottom row).

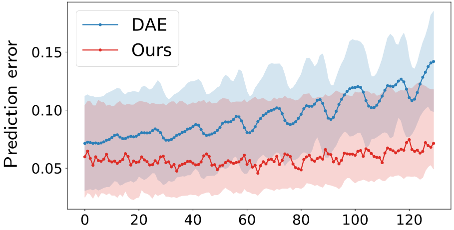


(a) Clean observations

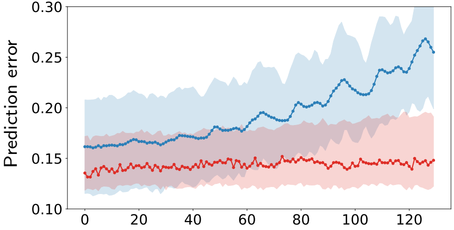
(b) Noisy observations

High-dimensional Fluid Flow Past a Cylinder





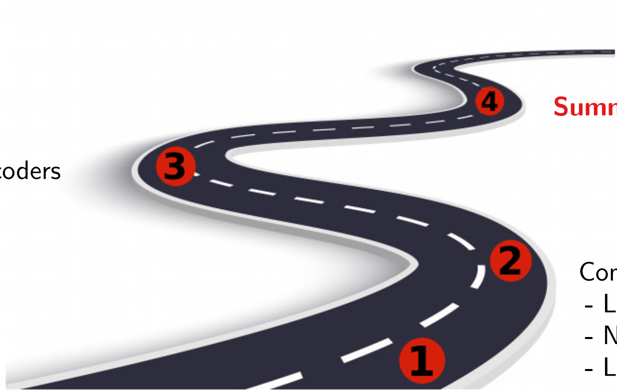
(a) Clean observations



(b) Noisy observations

Outline

Dynamic Autoencoders



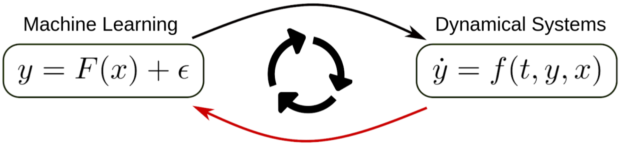
Summary

Continuous-time RNNs
- Lipschitz RNNs
- Noisy RNNs
- LEMs

Introduction

Summary

- A richer understanding between DS and DL enables us to design more robust models.
- Noise injection can be viewed as a stochastic learning strategy used to improve robustness of the learning model against data perturbations
- Our empirical results show that CT RNNs achieve superior robustness to input perturbations, while maintaining state-of-the-art generalization performance



The dynamical systems perspective can help us to better understand black-box ML methods as well as to help us to design more robust models.