UCLA Samueli Computer Science

ScAi

Learning Continuous System Dynamics from Irregularly-Sampled Partial Observations

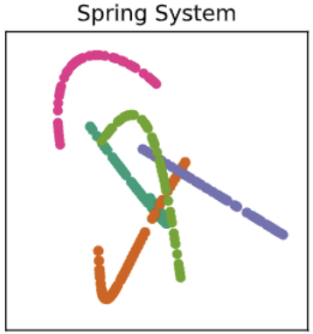
Zijie Huang[1], Yizhou Sun[1], Wei Wang[1]

[1]University of California, Los Angeles

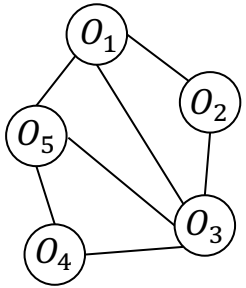
Motivation: Learning multi-agent system dynamics

- Many real-world systems, such as moving planets, particles connected by springs, can be considered as multi-agent dynamic systems, where objects **interact** with each other and **co-evolve** along with the time.

- Given **observed trajectories** and the **interaction graph among agents**, can we reason how objects interact in a complex system, i.e. the dynamics of the system, and make predictions in the future?



Observed trajectories



Interaction graph

Motivation: Learning multi-agent system dynamics

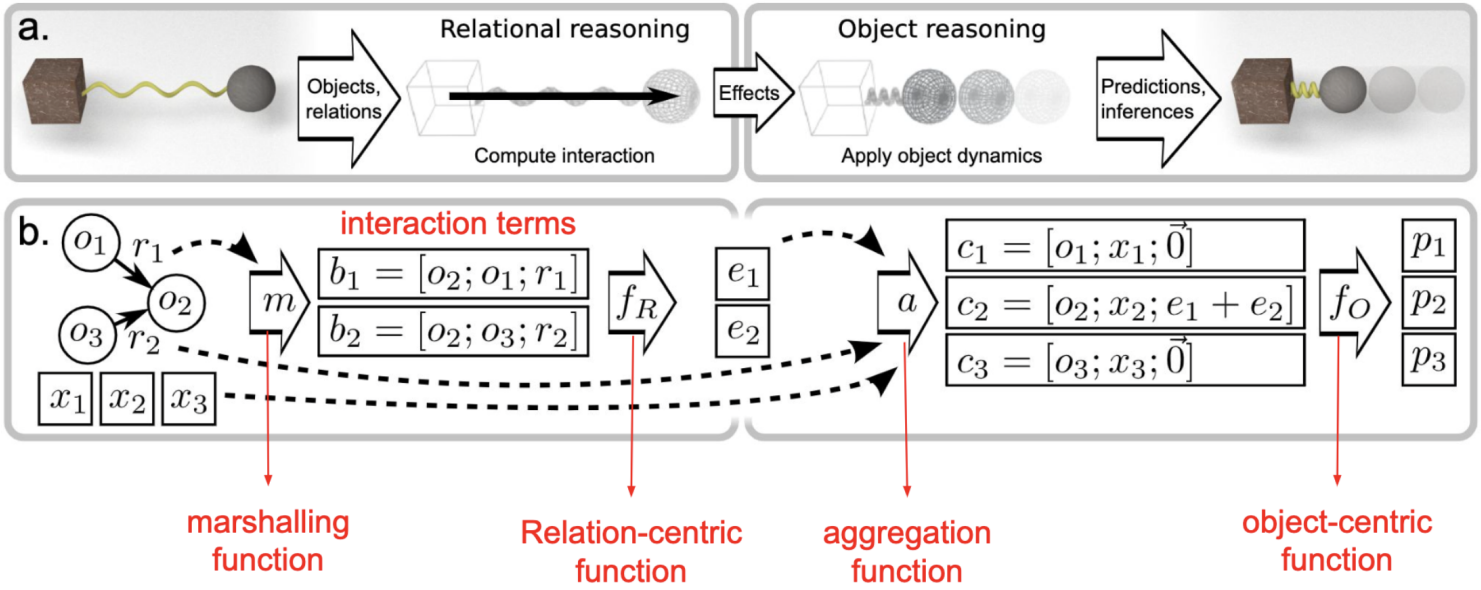
- Existing neural physical simulators employ graph neural networks (GNN) to approximate pair-wise object interactions, which reveals how system changes from timestamp t, to timestamp t+1.

- They usually decompose the system into distinct objects and relations and try to learn the object function and relation function from data.

$$e_{(i,j)}^{t+1} = f_R(o_i^t, o_j^t), \qquad o_i^{t+1} = f_O(\sum_{j \in \mathcal{N}_i} e_{(i,j)}^{t+1}, o_i^t)$$

Motivation: Learning multi-agent system dynamics

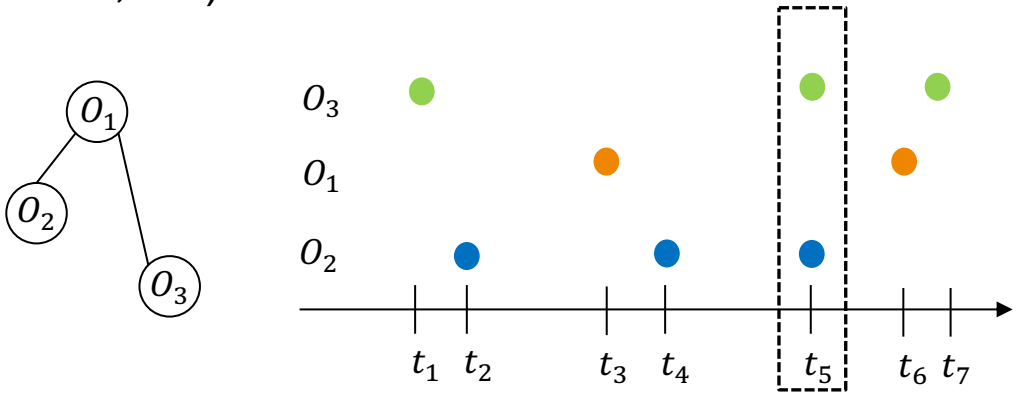
- Example: Interaction Network (IN) [1]



[1] Battaglia, Peter and Pascanu, Razvan, etc. Interaction Networks for Learning about Objects, Relations and Physics. (NIPS 2016)

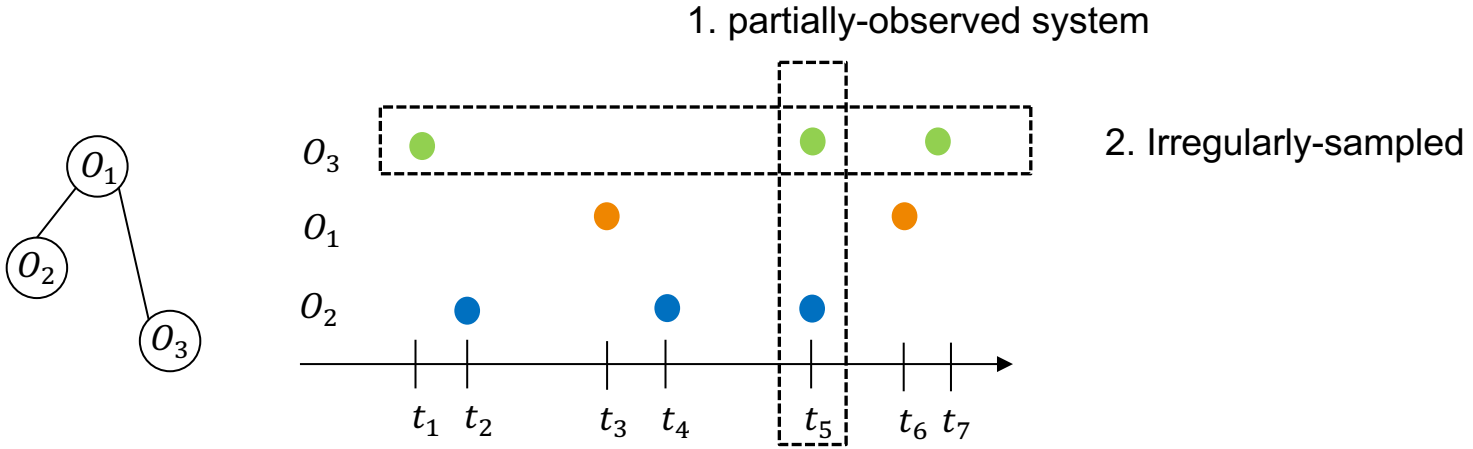
Motivation: Learning multi-agent system dynamics

- **Problem1**: They only work for **fully** observable systems, where individual trajectories of each object can be accessed at every sampling time. In reality, many applications have to deal with **partial** observable states, meaning that the observations for different agents are not temporally aligned. (e.g. caused by sensor broken, etc)



Motivation: Learning multi-agent system dynamics

- **Problem2**: The visibility of a specific object might change over time, meaning that observations can happen at non-uniform intervals for each agent, i.e.**irregularly-sampled** observations.



Multivariate Timeseries Modeling with missing data

Trajectory prediction problem in dynamic system can be formulated as multivariate timeseries prediction. Exsiting works have designed several variants of RNN to handle missing data issue [1].

$\boldsymbol{X}$: Input time series (2 variables);

$\boldsymbol{s}$: Timestamps for $\boldsymbol{X}$;

$$\boldsymbol{X} = \begin{bmatrix} 47 & 49 & NA & 40 & NA & 43 & 55 \\ NA & 15 & 14 & NA & NA & NA & 15 \end{bmatrix}$$

$$\boldsymbol{s} = [\,0 \quad 0.1 \quad 0.6 \quad 1.6 \quad 2.2 \quad 2.5 \quad 3.1]$$

$\boldsymbol{M}$: Masking for $\boldsymbol{X}$;

$\boldsymbol{\Delta}$: Time interval for $\boldsymbol{X}$.

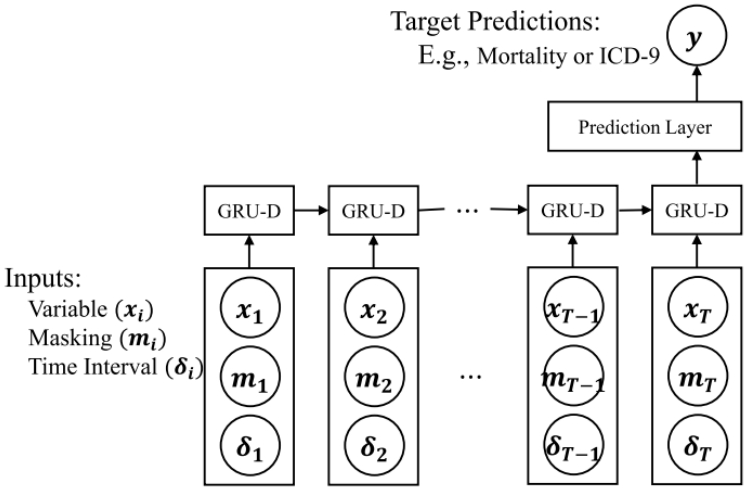
$$\boldsymbol{M} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\Delta} = \begin{bmatrix} 0.0 & 0.1 & 0.5 & 1.5 & 0.6 & 0.9 & 0.6 \\ 0.0 & 0.1 & 0.5 & 1.0 & 1.6 & 1.9 & 2.5 \end{bmatrix}$$

[1] Zhengping Che and Sanjay Purushotham, etc. Recurrent Neural Networks for Multivariate Time Series with Missing Values. (Nature 2018)

Multivariate Timeseries Modeling with missing data

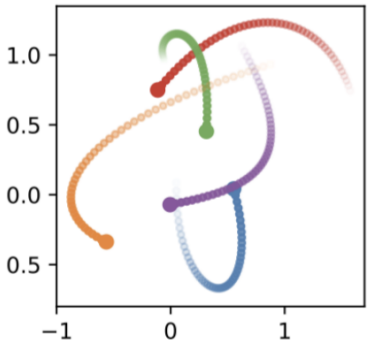
Trajectory prediction problem in dynamic system can be formulated as multivariate timeseries prediction. Exsiting works have designed several variants of RNN to handle missing data issue [1].



[1] Zhengping Che and Sanjay Purushotham, etc. Recurrent Neural Networks for Multivariate Time Series with Missing Values. (Nature 2018)

Multivariate Timeseries Modeling with missing data

However, it differs from multi-agent dynamic systems in that the model does not assume a **continuous interaction among each variable**, i.e. the underlying graph structure is not considered. Such interaction plays a very important role in multi-agent dynamic systems which drives the system to move forward.



Balls connected by springs

$$\mathbf{F}_{ij} = -C(\mathbf{x}_i - \mathbf{x}_j)$$

[1] Zhengping Che and Sanjay Purushotham, etc. Recurrent Neural Networks for Multivariate Time Series with Missing Values. (Nature 2018)

Our solution : LG-ODE

- Learn continuous multi-agent system dynamics from **irregularly-sampled partial** observations considering **graph structure**.

- Learn the dynamic nature of a system using **neural ordinary differential equations**, where we parameterize ODE function with a GNN to capture continuous interaction among agents. Infer the **initial states** for each agent **simultaneously** using variational autoencoder.

- Applications on motion capture, spring system, and charged particle datasets.

Background: ODE for dynamic system

- In continuous multi-agent dynamic system, the state evolution is governed by a series of first-order ordinary differential equations that drive the system states forward in infinitesimal steps over time.

$$\dot{\boldsymbol{z}}_i^t := \frac{d\boldsymbol{z}_i^t}{dt} = g_i(\boldsymbol{z}_1^t, \boldsymbol{z}_2^t \cdots \boldsymbol{z}_N^t)$$

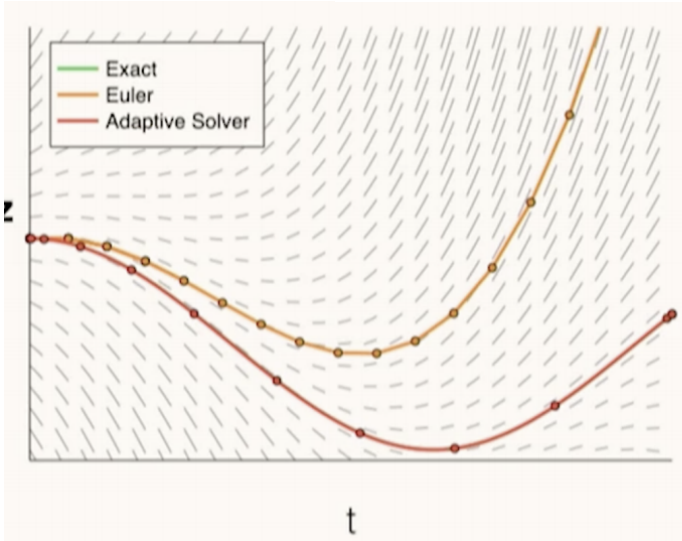
- Examples of real world systems with nonlinear interaction dynamics

Applications	Vertex	State at vertex i	Dynamics
Ecological (1949; 2006)	Species	Abundance	$\dot{x}_i = B_i + x_i(1 - \frac{x_i}{K_i})(\frac{x_i}{C_i} - 1) + \sum_j A_{ij} \frac{x_i x_j}{D_i + E_i x_i + H_j x_j}$
Regulatory (2006; 2008)	Gene	Expression level	$\dot{x}_i = -Bx_i^f + \sum_j A_{ij} R \frac{x_j^h}{x_j^h + 1}$
Epidemic (2001; 2004; 2005)	Person	Infection rate	$\dot{x}_i = -Bx_i + \sum_j A_{ij} R(1 - x_i)x_j$

Background: ODE for dynamic system

Given the latent initial states $z_0^0, z_1^0 \cdots z_N^0 \in \mathbb{R}^d$ for each object, solution for the initial-value problem (IVP) can be evaluated at any desired times using a numerical ODE solver such as Runge-Kutta.

$$z_i^T = z_i^0 + \int_{t=0}^{T} g_i(z_1^t, z_2^t \cdots z_N^t)dt$$



- Vector-valued $\mathbf{z}$ changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$

Background: NeuralODE

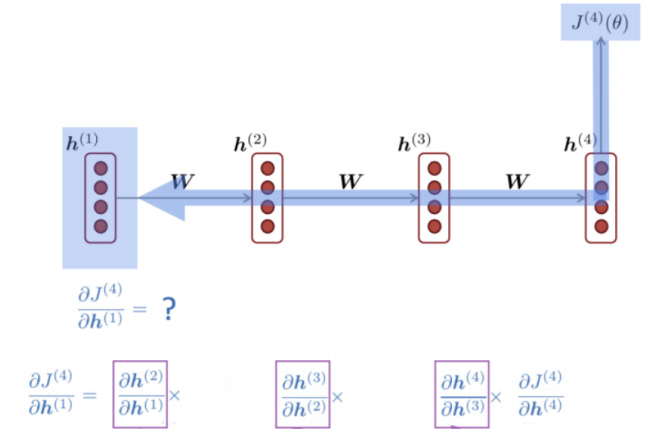
- Can we use neural network as the ODE function and automatically learn its parameters from data?

  – Yes! Using NeuralODE [1].

[1] Ricky T. Q. Chen and Yulia Rubanova, etc. Neural Ordinary Differential Equations. (NIPS 2018 best paper)

Background: NeuralODE

- Main challenge for using NN as ODE function: How to do back propagation during training?
  - Naïve Approach: Know the solver and backprop though the solver. – Memory-intensive



[1] Ricky T. Q. Chen and Yulia Rubanova, etc. Neural Ordinary Differential Equations. (NIPS 2018 best paper)

Background: NeuralODE

- Main challenge for using NN as ODE function: How to do back propagation during training?
  - Adjoint Method: Conduct back propagation by solving another ODE backwards in time. – Constant Memory Cost
  - This enables us to learn the ODE function efficiently from data, like other neural network architectures!

[1] Ricky T. Q. Chen and Yulia Rubanova, etc. Neural Ordinary Differential Equations. (NIPS 2018 best paper)

Background: ODE for dynamic system

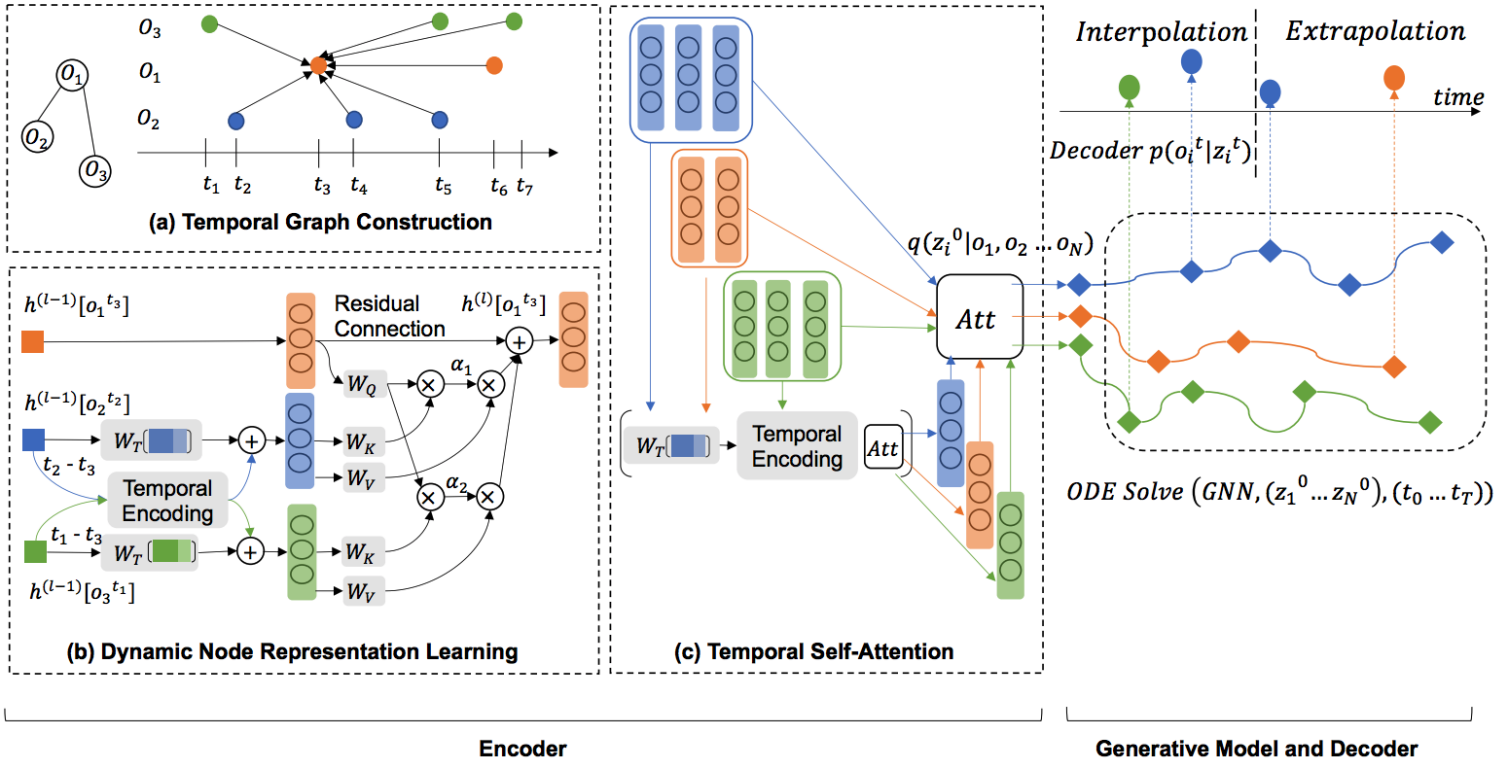
- To capture the continuous interaction among agents, we parameterize the ODE function using a GNN.

$$\dot{\boldsymbol{z}}_i^t := \frac{d\boldsymbol{z}_i^t}{dt} = g_i(\boldsymbol{z}_1^t, \boldsymbol{z}_2^t \cdots \boldsymbol{z}_N^t) = f_O(\sum_{j \in \mathcal{N}_i} f_R([\boldsymbol{z}_i^t, \boldsymbol{z}_j^t]))$$

- Given the ODE function, the latent initial states $\boldsymbol{z}_i^0$ for each object determine the whole trajectories.

- How to infer the initial states for multi-agent system in an unsupervised way from irregularly-sampled partial observations?

Overall Framework



VAE Loss

$$ELBO(\theta, \phi) = \mathbb{E}_{\boldsymbol{Z}^0 \sim q_\phi(\boldsymbol{Z}^0|o_1,\cdots o_N)}[\log p_\theta(o_1, \dots, o_N)] - \mathrm{KL}[q_\phi(\boldsymbol{Z}^0|o_1, \cdots, o_N) \| p(\boldsymbol{Z}^0)]$$

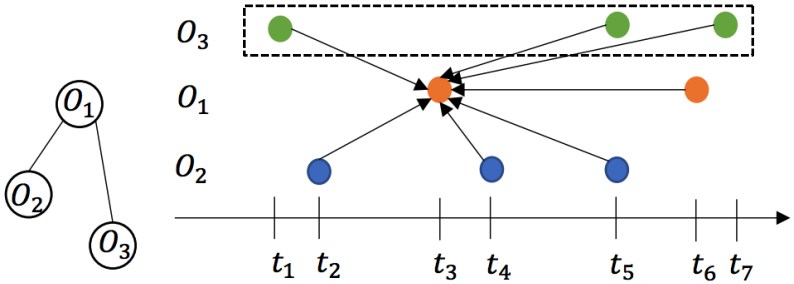
Dealing with irregularly-sampled partial observations

- Now the dynamics are modeled via the latent variable z, which is continuous.
- x now are used for
  - 1.) supervision signals to adjust the latent trajectories, thus can be irregular not temporarily aligned.
  - 2.) Infering initial states inference, and propse a nover encoder to handle the irregularity partial observations.

Encoder for inferring initial states

- In multi-agent dynamic system, agents are **highly-coupled and related**. Encoding temporal pattern to infer initial state for each agent independently fails to consider structural information.

- Our solution -- two-step encoding framework to infer initial states for all agent **simultaneously**: We incorporate structural information by first aggregating information from neighbors' observations, then employ a temporal self-attention mechanism to encode observation sequence for each object.

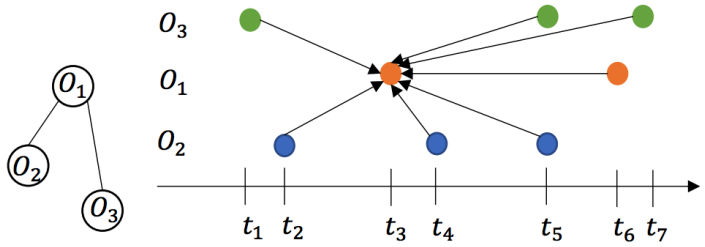


$$\boldsymbol{h}_i^t = f_{\text{update}}(o_i, \{o_j | \text{if } j \in \mathcal{N}_i\})$$

$$\boldsymbol{u}_i = f_{\text{aggre}}(\boldsymbol{h}_i^{t_1}, \boldsymbol{h}_i^{t_2} \cdots \boldsymbol{h}_i^{t_{T_i}})$$

Encoder for inferring initial states

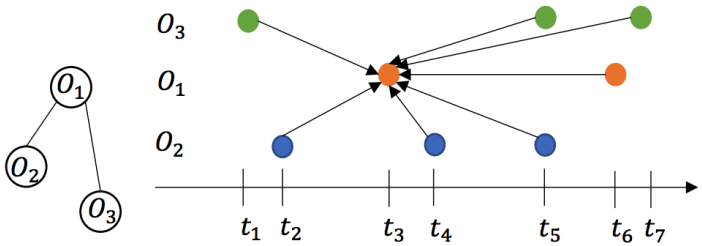
- **Step 1: Dynamic Node Representation Learning**. We preserve temporal edges and nodes across times to form a temporal graph, where every node is an observation.



$$\boldsymbol{h}_t^l = \boldsymbol{h}_t^{l-1} + \sigma\Big(\sum_{s \in \mathcal{N}_t} \big(\textbf{Attention}(\boldsymbol{h}_s^{l-1}, \boldsymbol{h}_t^{l-1}) \cdot \textbf{Message}(\boldsymbol{h}_s^{l-1})\big)$$

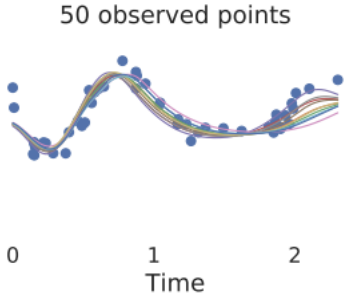
Encoder for inferring initial states

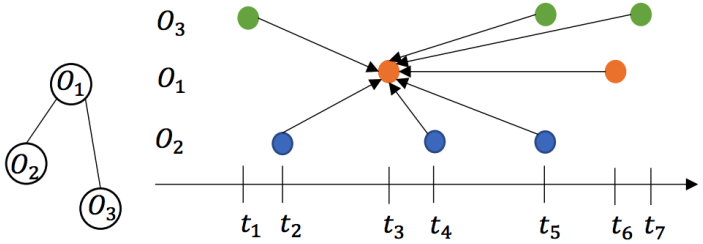
- **Why do not construct several graph snapshots, i.e. without cross-time edges?**
  - When system is partially observed, each snapshot may only contain a small portion of objects. Thus abundant structural information within snapshots is ignored.
  - Also, structural information across different snapshots explicitly incoporates historical influence from the past.

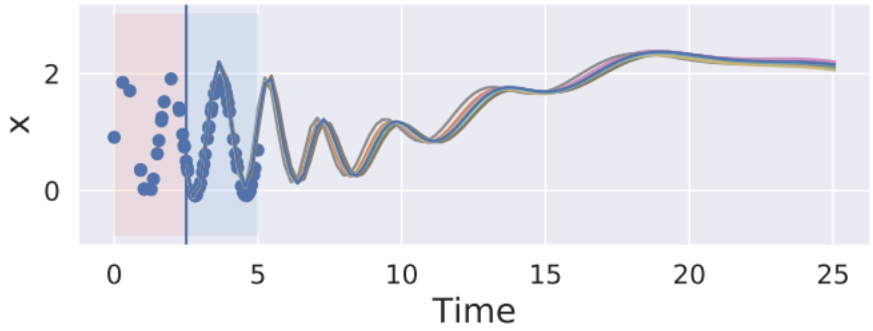


Encoder for inferring initial states

- **Cross-time edges construction to preserve autoregressive property.**
  - Task1- Interpolation: only preserve e_ij when t(i)< t(j)
  - Task2 – Extrapolation: only preserve e_ij when t(i)> t(j)





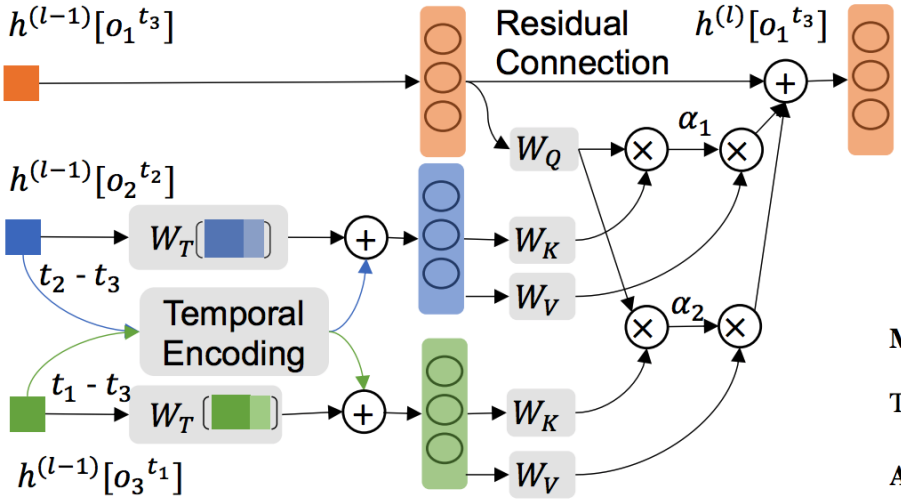


Encoder for inferring initial states

- **Cross-time edges construction with sliding window.**
  - Suppose on average every object has K observations, and there are E relations among objects. The constructed temporal graph has $O(EK^2 + (K - 1)KN)$ edges, which grows rapidly with the increase of average observation number K. We therefore set a slicing time window that filters out edges when the relative temporal gap is larger than a preset threshold.

Encoder for inferring initial states

- **Step 1: Dynamic Node Representation Learning**.



Transformer-based attention mechanism with **learnable** temporal encoding.

$$\boldsymbol{h}_t^l = \boldsymbol{h}_t^{l-1} + \sigma\Big(\sum_{s \in \mathcal{N}_t} (\textbf{Attention}(\boldsymbol{h}_s^{l-1}, \boldsymbol{h}_t^{l-1}) \cdot \textbf{Message}(\boldsymbol{h}_s^{l-1}))$$

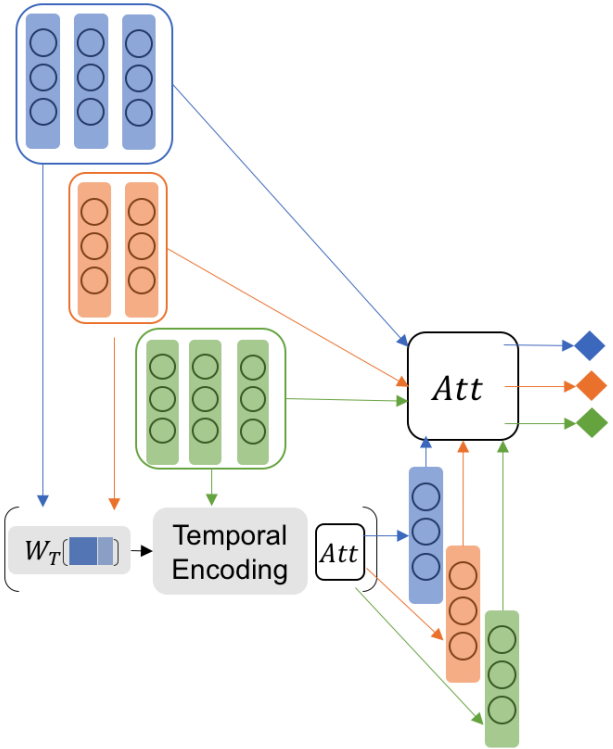
$$\textbf{Message}(\boldsymbol{h}_s^{l-1}, \Delta t(s,t)) = \boldsymbol{W}_v \hat{\boldsymbol{h}}_s^{l-1}, \quad \hat{\boldsymbol{h}}_s^{l-1} = \sigma(\boldsymbol{W}_t[\boldsymbol{h}_s^{l-1} || \Delta t_{st}]) + \text{TE}(\Delta t_{st})$$

$$\text{TE}(\Delta t)_{2i+1} = cos(\Delta t / 10000^{2i/d}), \quad \text{TE}(\Delta t)_{2i} = sin(\Delta t / 10000^{2i/d})$$

$$\textbf{Attention}(\boldsymbol{h}_s^{l-1}, \boldsymbol{h}_t^{l-1}, \Delta t(s,t)) = (\boldsymbol{W}_k \hat{\boldsymbol{h}}_s^{l-1})^T (\boldsymbol{W}_q \boldsymbol{h}_t^{l-1}) \cdot \frac{1}{\sqrt{d}}$$

Encoder for inferring initial states

- **Step 2: Temporal Self-Attention**.



Introduce a global sequence vector to calculate a weighted sum of observations as the sequence representation.

Add positioal encoding similar as in Step1 to take into account temporal information within the sequence.

$$\boldsymbol{a}_i = \tanh((\frac{1}{N}\sum_t \hat{\boldsymbol{h}}_i^t)\boldsymbol{W}_a), \quad \boldsymbol{u}_i = \frac{1}{N}\sum_t \sigma(\boldsymbol{a}_i^T \hat{\boldsymbol{h}}_i^t)\hat{\boldsymbol{h}}_i^t$$

Generative Model and Decoder

$$q_\phi(\boldsymbol{z}_i^0|o_1, o_2 \cdots o_N) = \mathcal{N}\left(\boldsymbol{\mu}_{z_i^0}, \boldsymbol{\sigma}_{z_i^0}\right), \quad \text{where } \boldsymbol{\mu}_{z_i^0},\ \boldsymbol{\sigma}_{z_i^0} = f(\boldsymbol{u}_i)$$

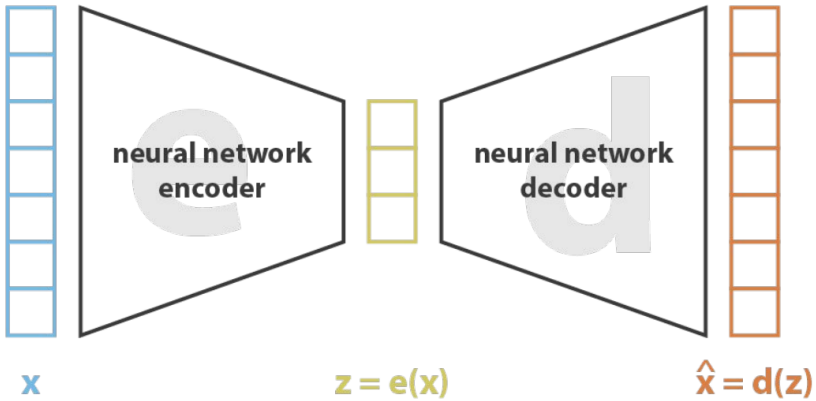
$$\begin{gathered}
\boldsymbol{z}_i^0 \sim p(\boldsymbol{z}_i^0) \approx q_\phi(\boldsymbol{z}_i^0|o_1, o_2 \cdots o_N) \\
\boldsymbol{z}_i^0, \boldsymbol{z}_i^1 \cdots \boldsymbol{z}_i^T = \text{ODESolve}(g_i, [\boldsymbol{z}_1^0, \boldsymbol{z}_2^0 \cdots \boldsymbol{z}_N^0], (t_0, t_1 \cdots t_T)) \\
\boldsymbol{o}_i^t \sim p(\boldsymbol{o}_i^t|\boldsymbol{z}_i^t)
\end{gathered}$$

Using Simple MLP as decoder function.

VAE v.s. Autoencoder

- Why using VAE, instead of Autoencoder to infer initial states?

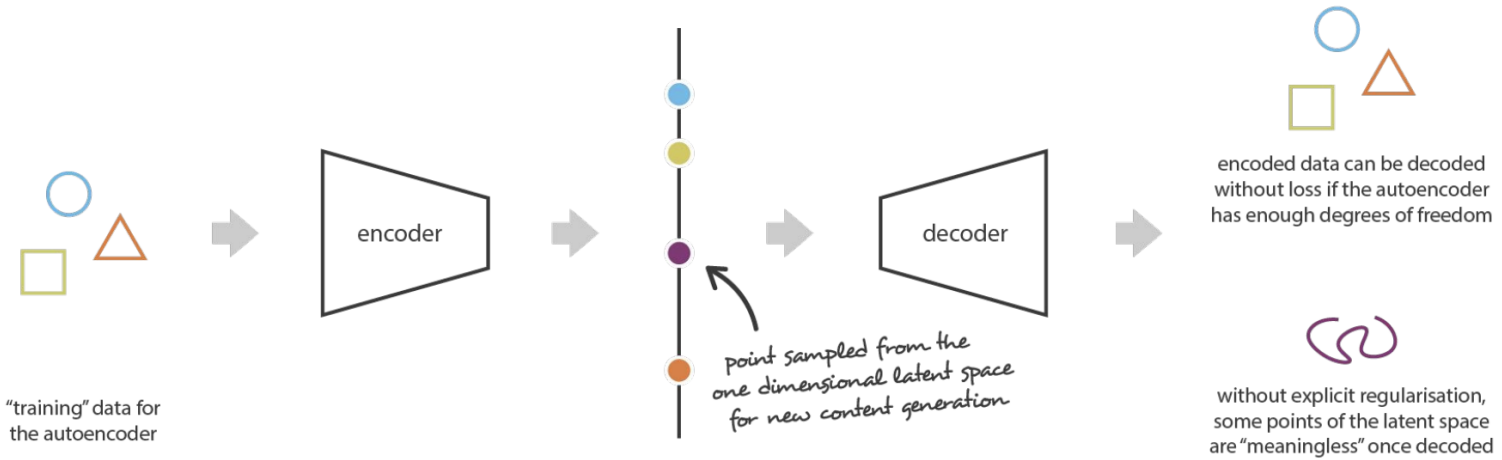
Autoencoder



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

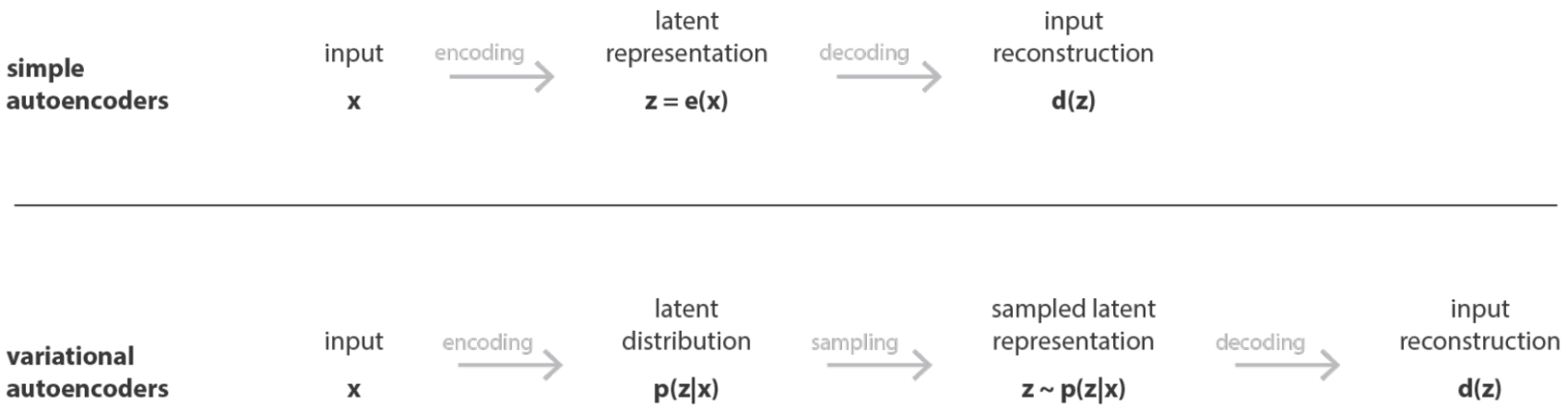
VAE v.s. Autoencoder

Can we generate new data from an autoencoder?



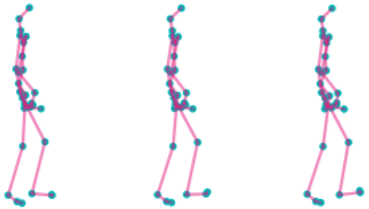
VAE v.s. Autoencoder

Variational Autoencoder



Evaluation

- Datasets:

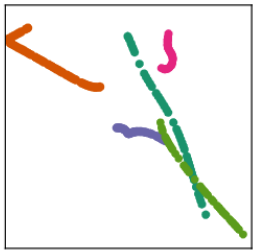


Motion Capture

$$\mathbf{F}_{ij} = -C(\mathbf{x}_i - \mathbf{x}_j)$$

Spring System

$$\mathbf{F}_{ij} = C \cdot \text{sign}(q_i \cdot q_j) \frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||^3}$$



Charged Particles

- Statistics:

	#Train Samples	#Test Samples	# Max Training Timestamps	# Testing Timestamps
Motion	120	27	50	40
Spring & Charged Particles	20k	5k	50	40

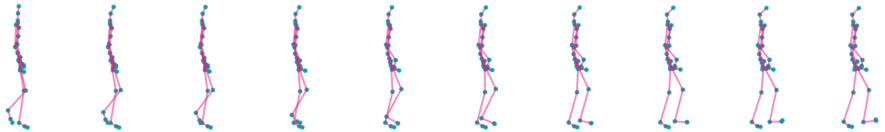
Evaluation

- Tasks:
  - **Interpolation**: In this task, we condition on a subset of observations (40%, 60%, 80%) from time (t_0, t_N) and aim to reconstruct the full trajectories in the same time range.
  - **Extrapolation**: In this task we split the time into two parts: (t_0, t_{N1}) and (t_{N1} , t_N). We condition on the first half of observations and reconstruct the second half. For training, we condition on observations from (t_1, t_2) and reconstruct the trajectories in (t_2, t_3). For testing, we condition on the observations from (t_1, t_3) but tries to reconstruct future trajectories within (t_3, t_4).

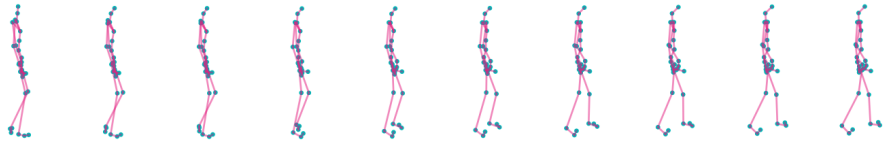
Evaluation - Interpolation Results

Table 1: Mean Squared Error(MSE) $\times 10^{-2}$ on Interpolation task.

	Springs			Charged			Motion		
Observed ratio	40%	60%	80%	40%	60%	80%	40%	60%	80%
Latent-ODE	0.5454	0.5036	0.4290	1.1799	1.1198	0.8332	0.7709	0.4826	0.3603
Weight-Decay	1.1634	1.1377	1.6217	2.8419	2.2547	1.5390	1.9007	2.0023	1.6894
Edge-GNN	1.3370	1.2786	0.8188	1.5795	1.5618	1.1420	2.7670	2.6582	1.8485
NRI + RNN	0.5225	0.4049	0.3548	1.3913	1.1659	1.0344	0.5845	0.5395	0.5204
LG-ODE	**0.3350**	**0.3170**	**0.2641**	**0.9234**	**0.8277**	**0.8046**	0.4515	**0.2870**	**0.3414**
LG-ODE-first	1.3017	1.1918	1.0796	2.5105	2.6714	2.3208	1.4904	1.3702	1.2107
LG-ODE-mean	0.3896	0.3901	0.3268	1.1246	1.0050	0.9133	0.6415	0.5834	0.5549
LG-ODE-no att	0.5145	0.4198	0.4510	0.9372	0.9503	0.9752	0.6991	0.6998	0.7452
LG-ODE-no PE	0.4431	0.4278	0.3879	1.0450	1.0350	0.9621	0.4677	0.4808	0.4799
LG-ODE-fixed PE	0.4285	0.4445	0.4083	0.9838	0.9775	0.9524	**0.4215**	0.4371	0.4313



(a) Groundtruth.



(b) Predictions with 0.6 observation ratio.

Evaluation – Extrapolation Results

Table 2: Mean Squared Error(MSE) $\times 10^{-2}$ on Extrapolation task.

Extrapolation	Springs			Charged			Motion		
Observed ratio	40%	60%	80%	40%	60%	80%	40%	60%	80%
Latent-ODE	6.6923	4.2478	4.3192	13.5852	12.7874	20.5501	2.4186	2.9061	2.6590
Weight-Decay	6.1559	5.7416	5.3712	9.4764	9.1008	9.0886	16.8031	13.6696	13.6796
Edge-GNN	6.0417	4.9220	3.2281	9.2124	9.1410	8.8341	13.2991	13.9676	9.8669
NRI + RNN	2.6638	2.4003	2.5550	7.1776	6.9882	6.6736	3.5380	3.0119	2.6006
LG-ODE	**1.7839**	1.8084	**1.7139**	6.5320	**6.4338**	**6.2448**	**1.2843**	**1.2435**	1.2010
LG-ODE-first	6.5742	6.3243	5.7788	9.3782	9.2107	8.4765	3.8864	3.2849	3.0001
LG-ODE-mean	2.2499	2.1165	2.2516	9.1355	8.7820	8.4422	1.3169	1.3008	1.2534
LG-ODE-no att	2.3847	2.1216	1.9634	7.2958	7.3609	6.7026	3.4510	3.2178	3.9917
LG-ODE-no PE	1.7943	1.8172	1.7332	6.9961	6.7208	6.5852	1.5054	1.2997	1.2029
LG-ODE-fixed PE	1.7905	**1.7634**	1.7545	**6.4520**	6.4706	6.3543	1.4624	1.2517	**1.1992**

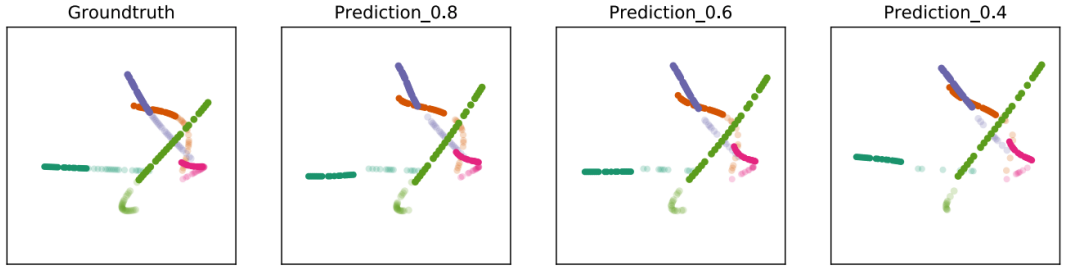


Figure 4: Visualization of extrapolation results for spring system. Semi-transparent paths denote observations from first-half of time, from which the latent initial states are estimated. Solid paths denote model predictions.

Future Work

- Consider the co-evolution of graph struction and node attributes
- Consider high-order ODE functions to capture more complexed system dynamics.

NeuralODE - Another view: continuous depth neural network

Resnet as Euler integrators

- Many people have noticed that there's a close connection between residual network and Euler integrators.
  - Hidden units look like: $z_{l+1} = F_l(z_l) = z_l + f_l(z_l)$
  - Final output is the composition: $z_L = F_{L-1} \circ F_{L-2} \cdots \circ F_0(z_0)$

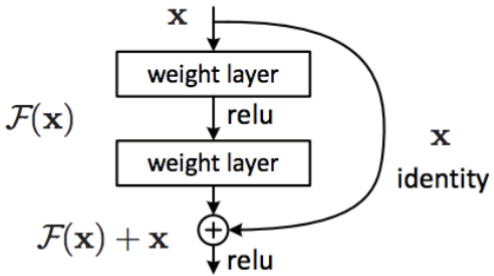
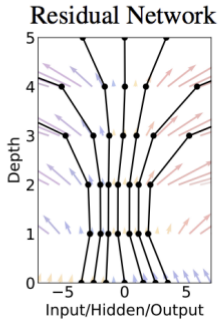


Figure 2. Residual learning: a building block.



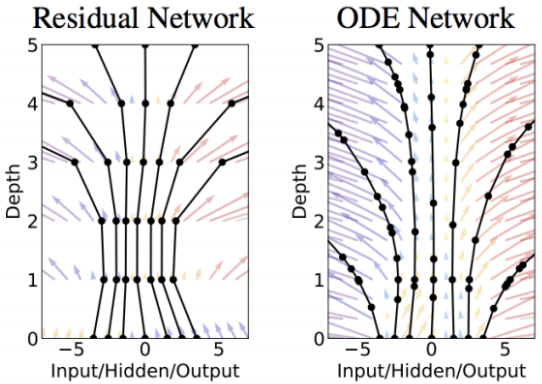
More About NeuralODE

Resnet as Euler integrators

- What happens as we add more layers and take smaller steps?
- In the limit, we parameterize the continuous dynamics of hidden units using an ordinary differential equation (ODE) specified by a neural network:

$$z_{t+1} = z_t + f(z_t, \theta)$$

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \theta(t))$$



More About NeuralODE

- **Pros**: Memory Efficiency; Adaptive Computation, Parameter Efficiency
- **Cons**: Running time is relatively slow, largely due to the ODESolver part. Recent works are studying how to accelerate the training process of NeuralODE.
- **Existing Works**:
    - Continuous Graph Neural Network that are robust to over-smoothing and hence allow us to build deeper networks, which in turn are able to capture the long-range dependencies between nodes.[1]

[1] Louis-Pascal A. C. Xhonneux and Meng Qu and Jian Tang. Continuous Graph Neural Network. (ICML2020)

More About NeuralODE

- **Existing Works**:
  - Learning to Encode Position for Transformer with Continuous Dynamical Model[1]. Learnable positional encoding that differs in different layers (by using different initial states).

[1] Xuanqing Liu and Hsiang-Fu Yu,etc. Learning to Encode Position for Transformer with Continuous Dynamical Model. (ICML2020)

UCLA Samueli Computer Science

ScAi

Thanks for Listening!