

Physics-inspired Deep Learning

KTAI 콜로кви움
2024년 10월 28일

Noseong Park



Tenured Associate Professor

School of Computing

Korea Advanced Institute of Science & Technology

noseong@kaist.ac.kr



BigDyL
[big dyl]

Big Data Analytics & Learning Laboratory
Korea Advanced Institute of Science & Technology

Contents

- Science for Deep Learning
 - Why deep learning based on differential equations?
 - Physics-inspired deep learning for graphs
 - Physics-inspired deep learning for spatiotemporal forecasting
- Deep Learning for Science
 - What are Partial Differential Equations
 - Deep Learning for solving PDEs
- Conclusion

Science for Deep Learning

The background is a dark blue gradient. It features numerous faint, white-outlined rectangles of various sizes, some of which contain illegible text, suggesting overlapping documents or code windows. Scattered throughout the background are strings of binary code (0s and 1s) in a light blue or white color, some appearing to float or move across the frame.

What are differential equations?

- Let $\mathbf{h}(t)$ be a state vector.
 - For describing rockets, $\mathbf{h}(t) = [x, y, velocity, fuel, oxygen]$.
 - For describing COVID-19, $\mathbf{h}(t) = [susceptible, infected, recovered]$.
- $\frac{d\mathbf{h}(t)}{dt}$ is a differential equation describing how $\mathbf{h}(t)$ changes over time.
 - People are good at the deductive process of defining $\frac{d\mathbf{h}(t)}{dt}$ or $\frac{\partial \mathbf{h}(x,y,z,t)}{\partial t}$.
 - For instance, one person wrote $\vec{F} = m\vec{a} = \frac{d}{dt}(m\vec{v}) = \frac{d^2}{dt^2}(m\vec{u})$.
- We are interested in solving the following initial value problem (IVP) to know the state in the future.

$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T \frac{d\mathbf{h}(t)}{dt} dt$$

What are differential equations? – contd.

- IVPs are sometime analytically solved.

$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T \mathbf{M}\mathbf{h}(t)dt = e^{\mathbf{M}T}\mathbf{h}(0)$$

- However, they exist notoriously difficult cases.
 - The Navier-Stokes equation is one of the Millenium problems.
 - We then rely on a solver to approximate the solution.

$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t))dt$$

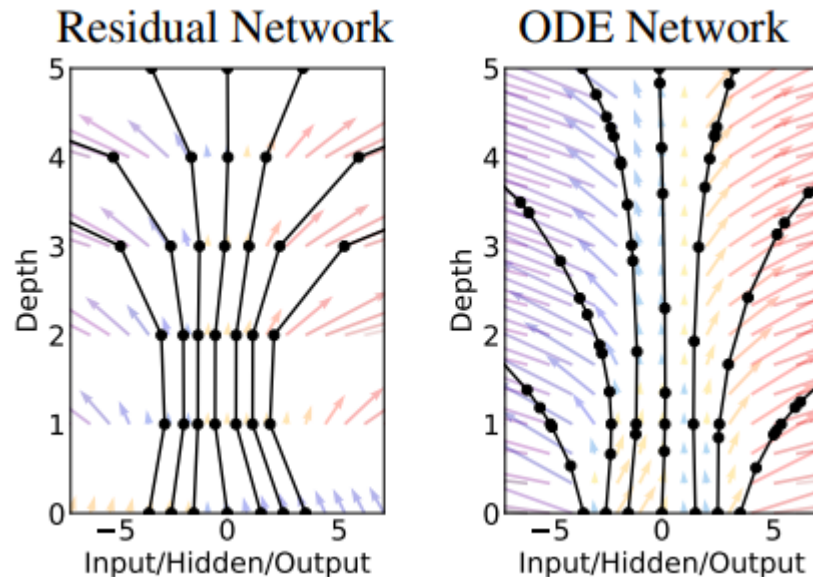
<Euler discretization>



$$\begin{aligned}\mathbf{h}(t+h) &= \mathbf{h}(0) + hf(\mathbf{h}(t)) \\ \mathbf{h}(t+2h) &= \mathbf{h}(t+h) + hf(\mathbf{h}(t+h)) \\ &\vdots\end{aligned}$$

Deep learning based on diff. eqs.

- Many successful deep neural networks are inspired/part of differential equations:
 - Neural ODEs are a continuous generalization of ResNet.



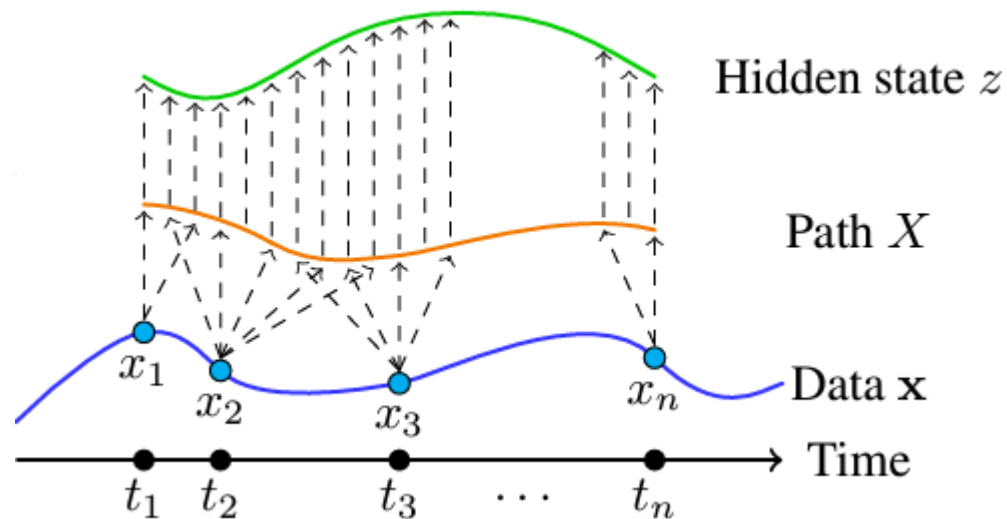
$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t); \theta) dt$$

<Euler discretization>

$$\mathbf{h}(t + h) = \mathbf{h}(0) + hf(\mathbf{h}(t); \theta)$$

Deep learning based on diff. eqs. – contd.

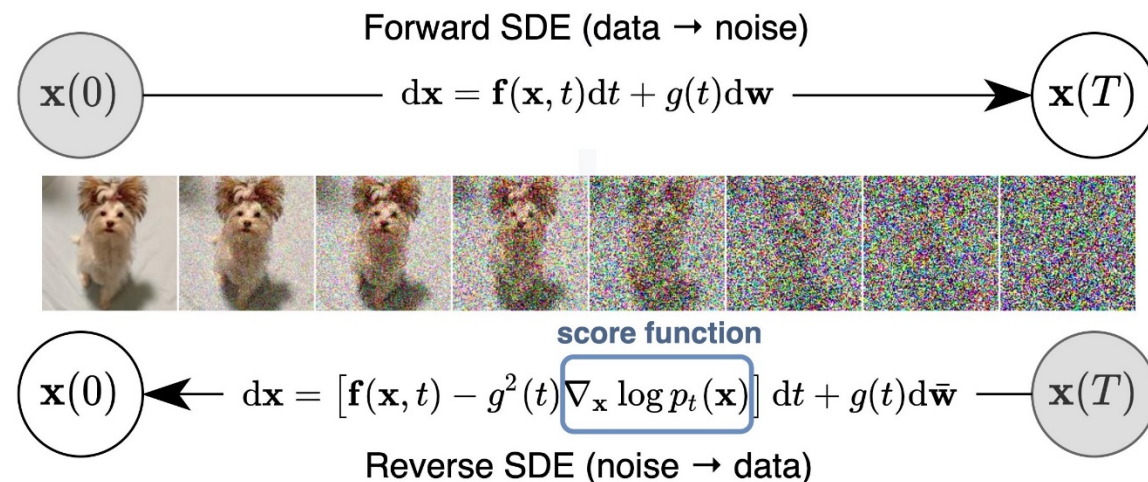
- Many successful deep neural networks are inspired/part of differential equations:
 - Neural ODEs are a continuous generalization of ResNet;
 - Neural CDEs are continuous recurrent neural networks and generalized state-space models.



$$\begin{aligned} \mathbf{h}(T) &= \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t); \theta) dX \\ &= \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t); \theta) \frac{dX}{dt} dt \end{aligned}$$

Deep learning based on diff. eqs. – contd.

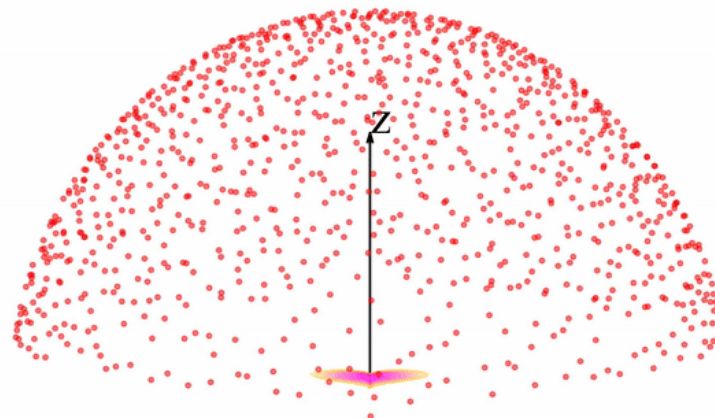
- Many successful deep neural networks are inspired/part of differential equations:
 - Neural ODEs are a continuous generalization of ResNet;
 - Neural CDEs are continuous recurrent neural networks and generalized state-space models;
 - Diffusion models are based on diffusion equations.



Deep learning based on diff. eqs. – contd.

- Many successful deep neural networks are inspired/part of differential equations:
 - Neural ODEs are a continuous generalization of ResNet;
 - Neural CDEs are continuous recurrent neural networks and generalized state-space models;
 - Diffusion models are based on diffusion equations;
 - Poisson process models are based on the Maxwell's equations.

$$d\mathbf{x} = -\mathbf{E}(\mathbf{x})dt$$



Deep learning based on diff. eqs. – contd.

- Many successful deep neural networks are inspired/part of differential equations:
 - Neural ODEs are a continuous generalization of ResNet;
 - Neural CDEs are continuous recurrent neural networks and generalized state-space models;
 - Diffusion models are based on diffusion equations;
 - Poisson process models are based on the Maxwell's equations;
 - Mamba is expected to be a successor to Transformers.

$$\begin{array}{llll}
 h'(t) = Ah(t) + Bx(t) & (1a) & h_t = \bar{A}h_{t-1} + \bar{B}x_t & (2a) & \bar{K} = (C\bar{B}, C\bar{A}\bar{B}, \dots, C\bar{A}^k\bar{B}, \dots) & (3a) \\
 y(t) = Ch(t) & (1b) & y_t = Ch_t & (2b) & y = x * \bar{K} & (3b)
 \end{array}$$

Discretization. The first stage transforms the “continuous parameters” (Δ, A, B) to “discrete parameters” (\bar{A}, \bar{B}) through fixed formulas $\bar{A} = f_A(\Delta, A)$ and $\bar{B} = f_B(\Delta, A, B)$, where the pair (f_A, f_B) is called a discretization rule. Various rules can be used such as the zero-order hold (ZOH) defined in equation (4).

$$\bar{A} = \exp(\Delta A) \quad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B \quad (4)$$

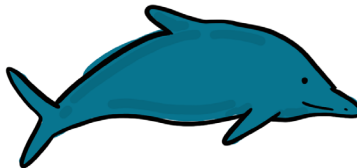
Nanos gigantum humeris insidentes

- Even in the era of large models, physical knowledge, written in the form of diff. eqs., provide us intuitive inductive biases toward designing effective neural networks.

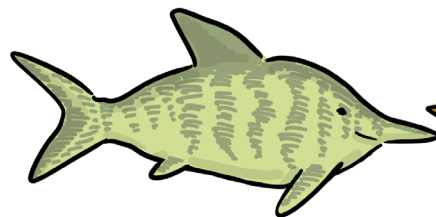
Shad (fish)



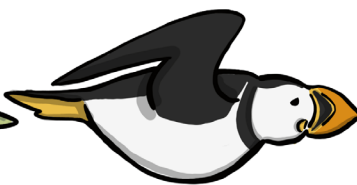
Dolphin (mammal)



Ichthyosaur (reptile)



Puffin (bird)

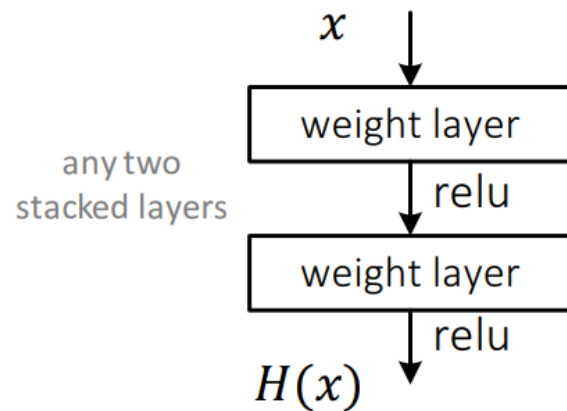


Neural Ordinary Differential Equations

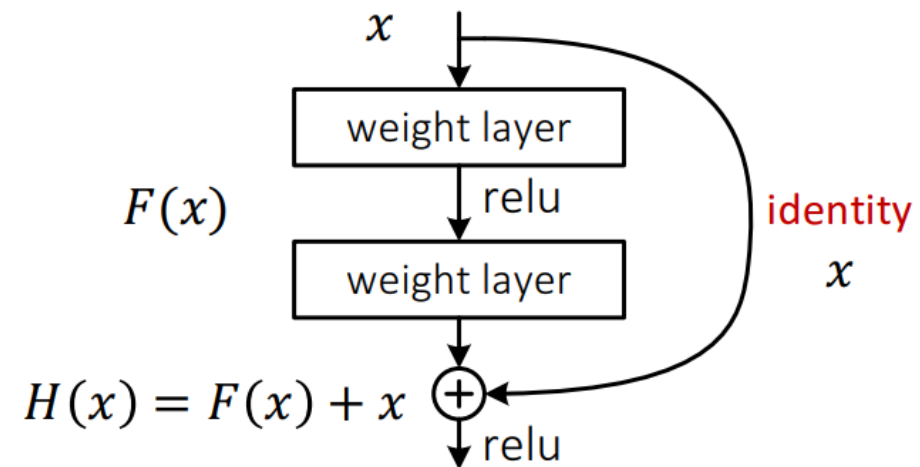
<Introduction>

Plain vs. residual connections

- Plain net



- Residual net



Why are ResNets successful?

$$x_{l+1} = x_l + F(x_l)$$



$$x_{l+2} = x_{l+1} + F(x_{l+1})$$

$$x_{l+2} = x_l + F(x_l) + F(x_{l+1})$$

⋮

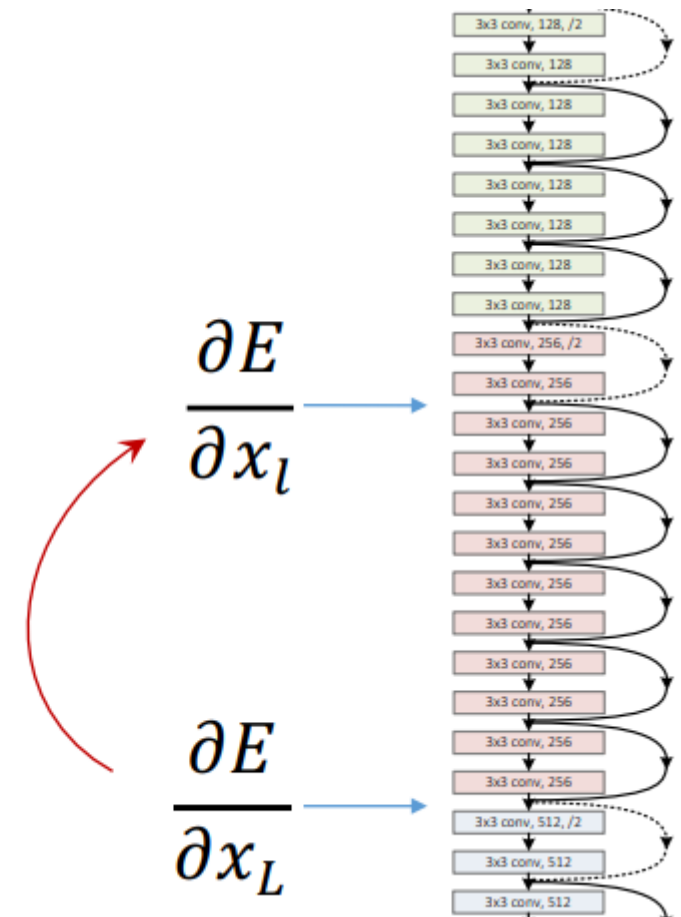
$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

Why are ResNets successful? – contd.

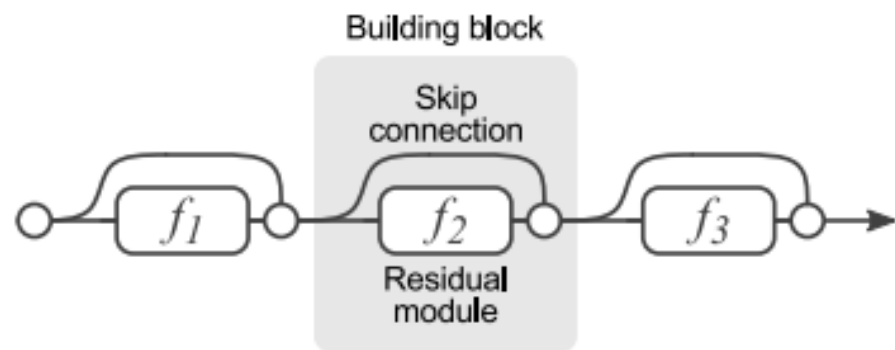
$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$



$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L} \frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L} \left(1 + \frac{\partial}{\partial x_l} \sum_{i=1}^{L-1} F(x_i) \right)$$

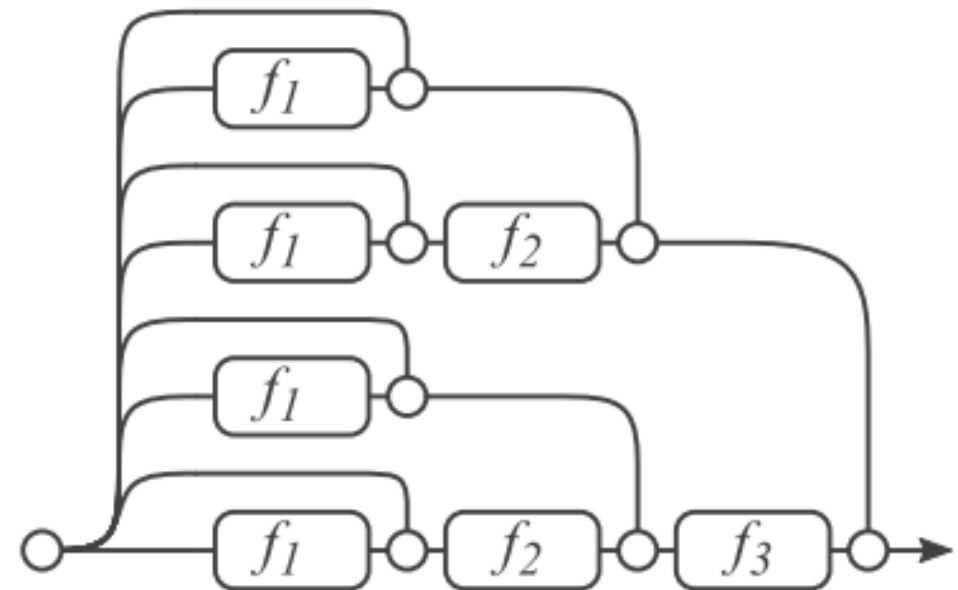


Why are ResNets successful? – contd.



(a) Conventional 3-block residual network

=



(b) Unraveled view of (a)

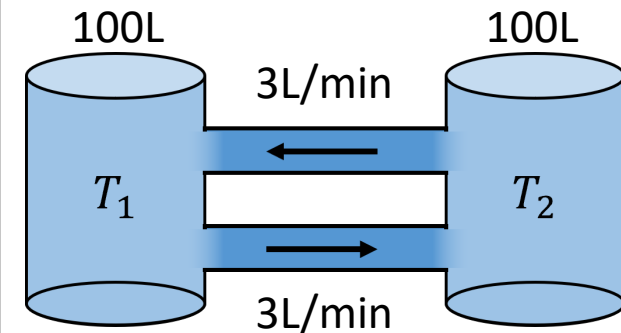
An example of ODEs

- T_1 has 100 liters of water, and T_2 has 100 liters of fertilizer.
- $\mathbf{z}(t) = (z_1(t), z_2(t))$ means the amount of fertilizer at time t .

$$z_1' = \text{inflow per minute} - \text{outflow per minute} = -0.03 z_1 + 0.03 z_2$$

$$z_2' = \text{inflow per minute} - \text{outflow per minute} = 0.03 z_1 - 0.03 z_2$$

$$\therefore \mathbf{z}' = \mathbf{A}\mathbf{z} \text{ or } \mathbf{z}' - \mathbf{A}\mathbf{z} = 0, \text{ where } \mathbf{A} = \begin{bmatrix} -0.03 & 0.03 \\ 0.03 & -0.03 \end{bmatrix}$$



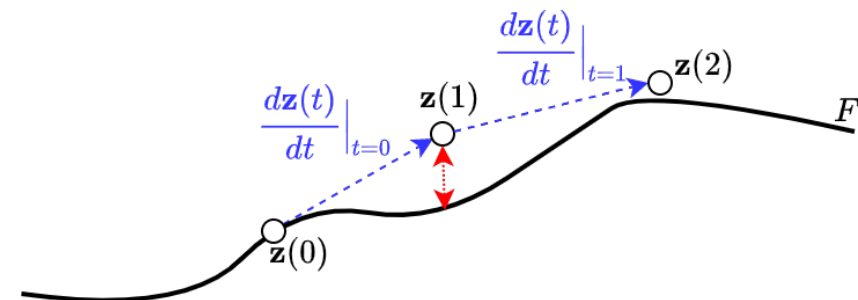
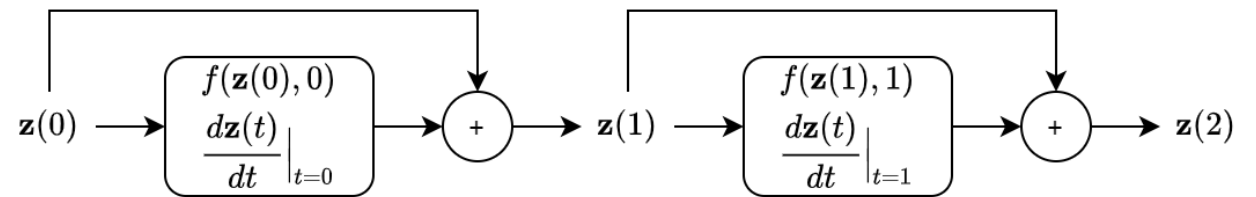
- When we have an initial value of $\mathbf{z}(0) = (0, 100)$, what is $\mathbf{z}(2)$? This kind of problem is called initial value problem (IVP) or forward problem.
- Given data, what is \mathbf{A} ? This kind of problem is called backward problem.

Euler method vs. residual connection

- Among various ODE solvers, the (explicit) Euler method is the simplest method.
- The (explicit) Euler method and the residual connection look similar to each other.

Problem to solve: $\mathbf{z}(2) = \mathbf{z}(0) + \int_0^2 f(\mathbf{z}(t), t) dt$, where $f(\mathbf{z}(t), t) = \frac{d\mathbf{z}(t)}{dt}$

Solve with the Euler method: $\mathbf{z}(h) = \mathbf{z}(0) + h \times f(\mathbf{z}(0), 0)$
 $\mathbf{z}(2h) = \mathbf{z}(h) + h \times f(\mathbf{z}(h), h)$
 \vdots



Runge–Kutta (RK) method

Now pick a step-size $h > 0$ and define

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4),$$

$$t_{n+1} = t_n + h$$

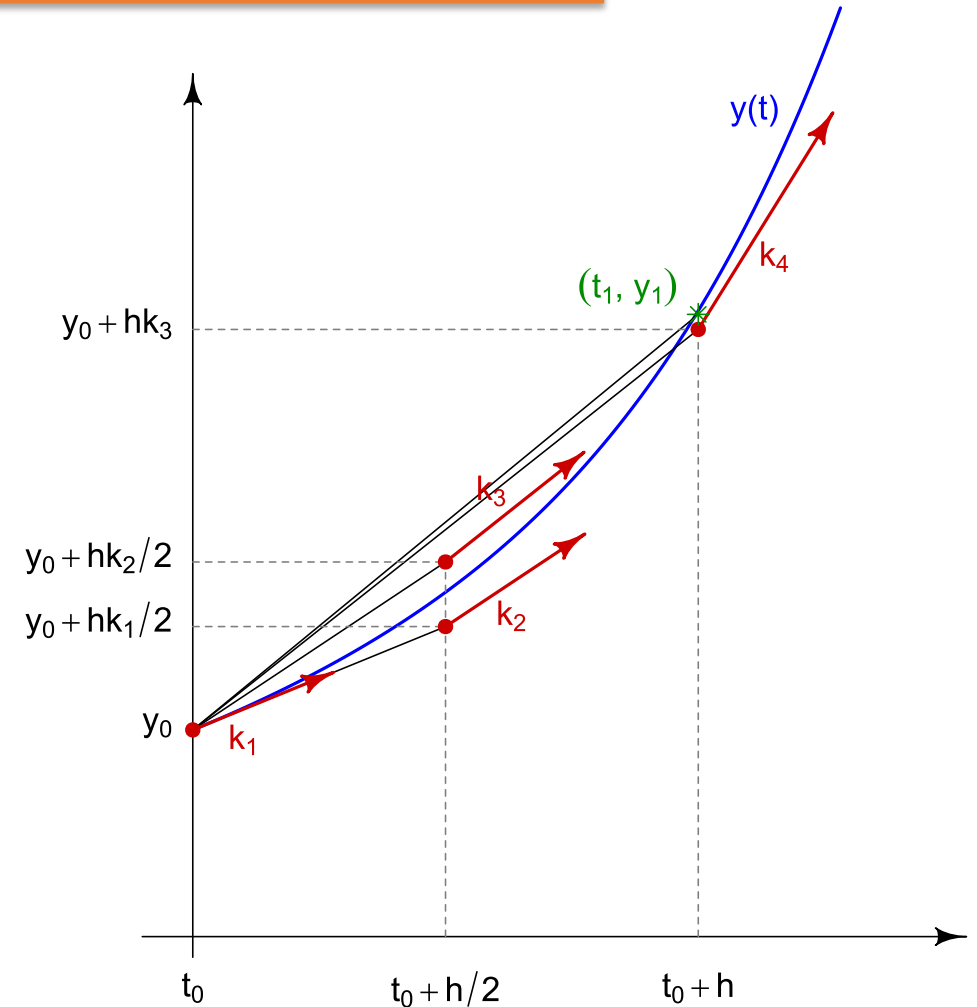
for $n = 0, 1, 2, 3, \dots$, using^[3]

$$k_1 = f(t_n, y_n),$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right),$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right),$$

$$k_4 = f(t_n + h, y_n + hk_3).$$

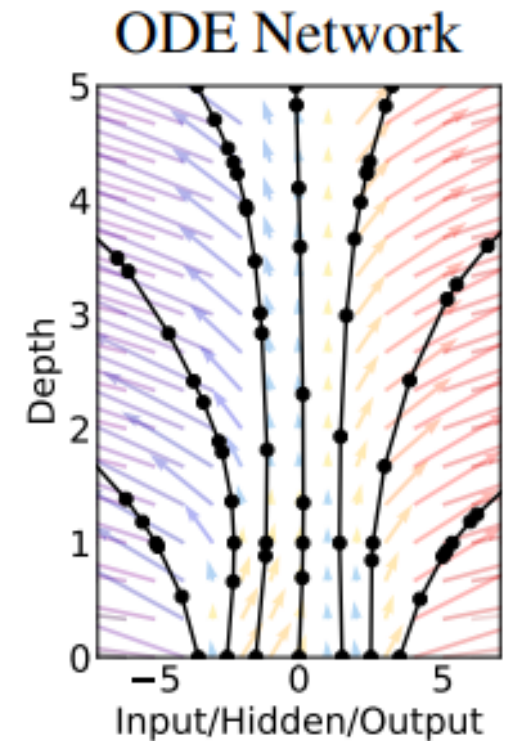
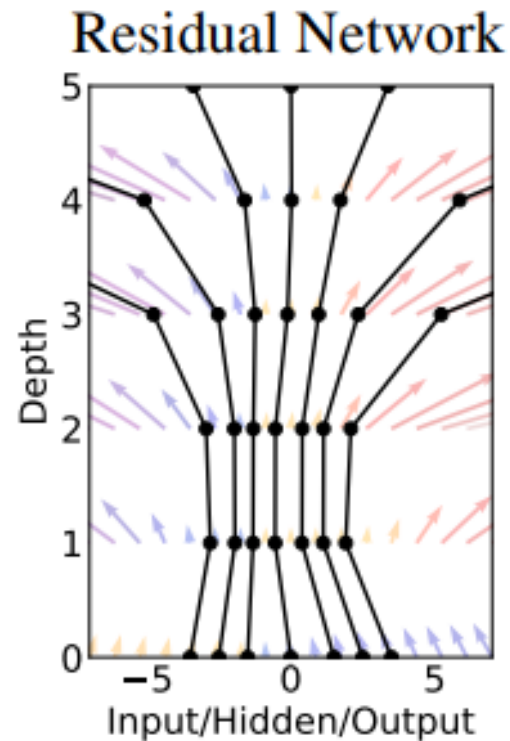


Dormand–Prince (DOPRI) method

- After comparing the RK4 and RK5 results,
 - Use a large step-size h if the difference is small.
 - Use a small step-size h if the difference is large.
- In other words, the (adaptive) step-size is inversely proportional to the estimated difference.
- We omit its detailed mathematical definition.

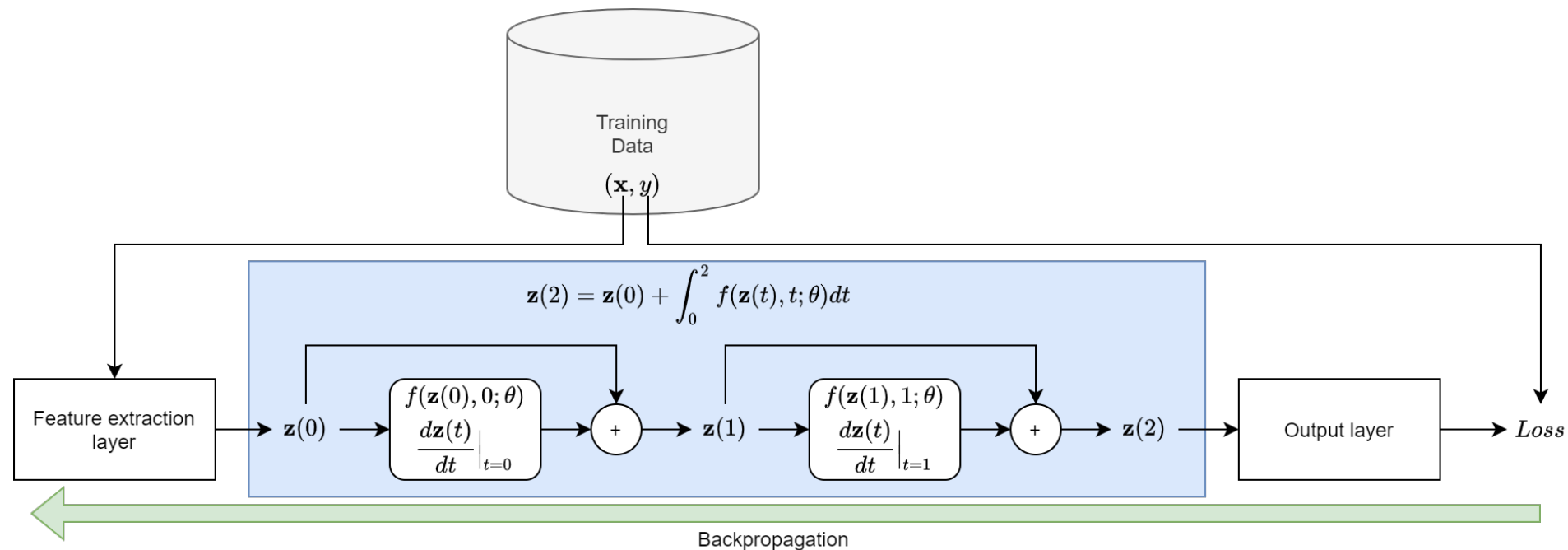
Continuous ResNet

- Solve $y = z(T)$, given the initial condition $z(0) = x$, with a black-box solver.
 - $O(1)$ space complexity
 - $O\left(\frac{T}{h}\right)$ time complexity
- Parametrize $\frac{dz(t)}{dt} = f(z(t), \theta(t))$.



NODE-based image classifier

- A typical construction of NODEs is as follows: FE \rightarrow NODE \rightarrow Output.
- The NODE layer is analogous to (continuous) residual layers.
- We can use the standard backpropagation algorithm to train.



How to train

Ultimately want to optimize some loss.

$$L(\underline{z(T)}) = L \left(\underline{z(t_0) + \int_{t_0}^T f(z(t), t, \theta) dt} \right) = L(\text{ODESolve}(z(t_0), t_0, T, \theta))$$

Last hidden vector

How to calculate the last hidden vector

$$\frac{\partial L}{\partial \theta} = ?$$

Adjoint sensitivity method

- We can use the standard backpropagation to train NODEs.
 - However, the depth by DOPRI frequently becomes large.
- We can calculate the gradients with a reverse-mode integral.
- Which one is better?
 - In our experience, case by case.

Adjoint sensitivity method – contd.

- How to convert the left-hand side to the right-hand side:
 - Instead of the step-size h , use an integral with $\lim_{h \rightarrow 0}$.

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

$$\frac{\partial z_{t+h}}{\partial z_t} = (1 + h \frac{\partial f(z_t)}{\partial z_t})$$

$$a_t = \boxed{\frac{\partial z_{t+h}}{\partial z_t} \cdot \frac{\partial L}{\partial z_{t+h}}} = (1 + h \frac{\partial f(z_t)}{\partial z_t}) a_{t+h}$$

chain rule

$$\text{Backward: } a_t = \frac{\partial z_{t+h}}{\partial z_t} \cdot \frac{\partial L}{\partial z_{t+h}} = (1 + h \frac{\partial f(z_t)}{\partial z_t}) a_{t+h}$$

Adjoint method. Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t)) dt$

$$\text{Backward: } \underbrace{a(t)}_{\text{Adjoint State}} = a(t+1) + \underbrace{\int_{t+1}^t a(t) \frac{\partial f(z(t))}{\partial z(t)} dt}_{\text{Adjoint DiffEq}}$$

$$\text{Gradients: } \frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t) \frac{\partial f(z(t), \theta)}{\partial \theta} dt$$

Adjoint sensitivity method – contd.

- How to convert the left-hand side to the right-hand side:
 - Instead of the step-size h , use an integral with $\lim_{h \rightarrow 0}$.

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + ha_{t+h} \frac{\partial f(z_t)}{\partial z_t}$

$$\frac{\partial L}{\partial \theta} = \underbrace{\frac{\partial L}{\partial z_{t+h}} \cdot \frac{\partial z_{t+h}}{\partial \theta}}_{\text{chain rule}} = a_{t+h} \frac{\partial (z_t + hf(z_t))}{\partial \theta}$$

Gradients: $\frac{\partial L}{\partial \theta} = ha_{t+h} \frac{\partial f(z(t), \theta)}{\partial \theta}$

Adjoint method. Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t)) dt$

Backward: $\underbrace{a(t)}_{\text{Adjoint State}} = a(t+1) + \underbrace{\int_{t+1}^t a(t) \frac{\partial f(z(t))}{\partial z(t)} dt}_{\text{Adjoint DiffEq}}$

Gradients: $\frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t) \frac{\partial f(z(t), \theta)}{\partial \theta} dt$

Adjoint sensitivity method – contd.

<https://ilya.schurov.com/post/adjoint-method/>

- How to convert the left-hand side to the right-hand side:
 - Instead of the step-size h , use an integral with $\lim_{h \rightarrow 0}$.

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + ha_{t+h} \frac{\partial f(z_t)}{\partial z_t}$

Gradients: $\frac{\partial L}{\partial \theta} = ha_{t+h} \frac{\partial f(z(t), \theta)}{\partial \theta}$

Adjoint method. Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t)) dt$

Backward: $\underbrace{a(t)}_{\text{Adjoint State}} = a(t+1) + \underbrace{\int_{t+1}^t a(t) \frac{\partial f(z(t))}{\partial z(t)} dt}_{\text{Adjoint DiffEq}}$

Gradients: $\frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t) \frac{\partial f(z(t), \theta)}{\partial \theta} dt$

Graph Convolutional Networks

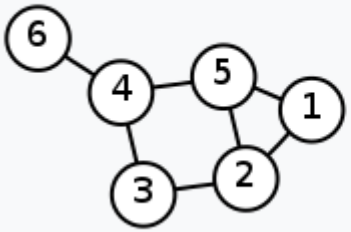
<Introduction>

Multiplication by the Laplacian

- Consider a graph G with Laplacian L and a graph signal \mathbf{x} on G .
- Signal $\mathbf{y} = L\mathbf{x}$ results from multiplying \mathbf{x} with the Laplacian.
- Component y_i of \mathbf{y} is as follows:

$$y_i = \sum_{j \in \mathcal{N}_i} (\mathbf{x}_i - \mathbf{x}_j).$$

- y_i measures the difference between \mathbf{x} at a node and its neighborhood, i.e., difference operator.

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Heat diffusion over graph

- Via $\mathbf{x}_t = -\mathbf{L}\mathbf{x}$, we say the signal diffuses through the graph, i.e., heat equation.
 - \mathbf{x}_t (or $\frac{d\mathbf{x}}{dt}$) stands for the time-derivative of \mathbf{x} .
 - The Euler method updates the temperature after $\mathbf{x}(t+h) = \mathbf{x}(t) - h\mathbf{L}\mathbf{x}(t)$.
 - It means a gradient flow which minimizes the Dirichlet energy.
- Temperature at each location is averaged with its neighbors' temperatures.
 - $-\mathbf{L}\mathbf{x}(t)$ will be negative (resp. positive) if my temperature is higher (resp. smaller) than those of neighbors.

Graph convolutional networks [Kipf & Welling, 2017]

- Let us use the normalized Laplacian $\tilde{\mathbf{L}}$ and $h = 1$:

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \tilde{\mathbf{L}}\mathbf{x}(t) = (\mathbf{I} - \tilde{\mathbf{L}})\mathbf{x}(t).$$

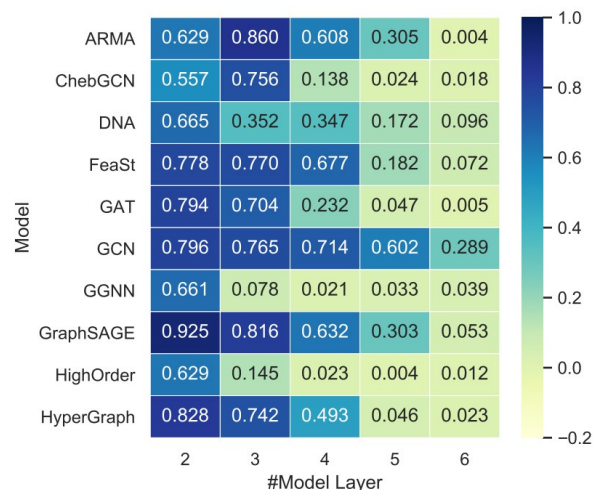
- Therefore, the GCN by Kipf and Welling uses the following diffusion process augmented with a trainable parameter (or diffusivity) \mathbf{W} :

$$\mathbf{x}(t+1) = \sigma((\mathbf{I} - \tilde{\mathbf{L}})\mathbf{x}(t)\mathbf{W}), \text{ where } \sigma \text{ is a non-linear activation.}$$

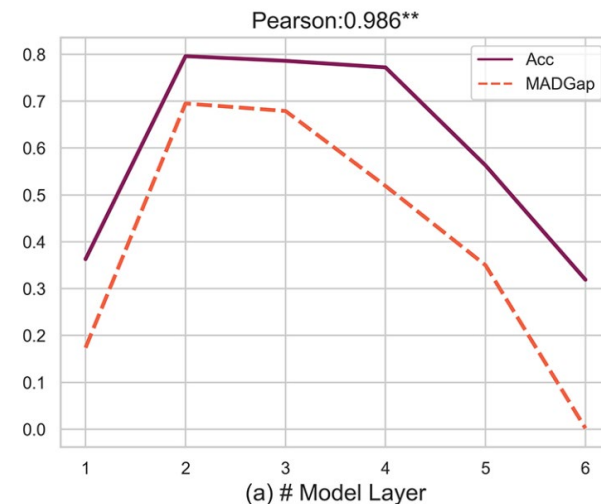
- At the same time, one can consider that this is a first-order graph filtering approach.
 - Given $\mathbf{H} = \sum_{\ell=0}^L h_{\ell} \mathbf{S}^{\ell}$, $\ell = 1$ and $\mathbf{S} = \mathbf{I} - \tilde{\mathbf{L}} = \tilde{\mathbf{A}}$.

Oversmoothing problem

- Many papers proposed similar approaches based on the diffusion equation-based interpretation of GCNs, e.g., GRAND [Chamberlain et al., 2021].
- One major drawback of these approaches is *oversmoothing*.
 - All nodes' last hidden vectors become similar to each other when the number of GCN layers is large [Chen et al., 2020a].



<Mean Avg. Distance (MAD)>



<Test accuracy of GCN on CORA>

Oversmoothing problem – contd.

- The history of GCNs is basically the history of battling with the oversmoothing problem.
 - GCNII [Chen et al., 2020b] tries to overcome the problem by i) initial residual connection, and ii) identity mapping.

$$\mathbf{x}(t+1) = \sigma\left((\alpha(\mathbf{I} - \tilde{\mathbf{L}})\mathbf{x}(t) + (1 - \alpha)\mathbf{x}(0))(\beta\mathbf{I} + (1 - \beta)\mathbf{W})\right)$$

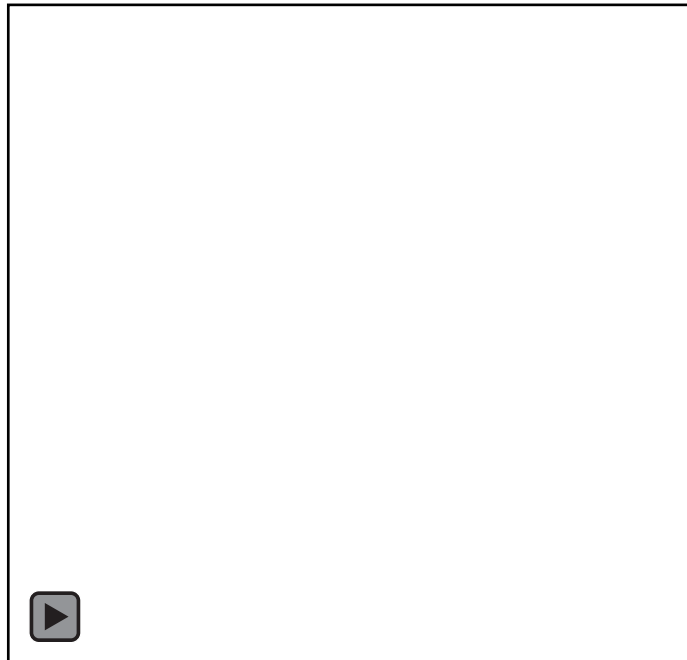
- A series of graph sparsification methods have been proposed in [Rong et al., 2019; Hasanzadeh et al., 2020; Zheng et al., 2020].
- Our answer to this problem is to use reaction-diffusion equations.
 - ACMP [Wang et al., 2023] is also based on the Allen-Cahn equation.
 - However, we consider more diverse reaction-diffusion equations.

Overcoming the oversmoothing problem

<Choi et al., GREAD: Graph Neural Reaction-Diffusion Networks, ICML, 2023>

Reaction-diffusion equations

- The reaction-diffusion system is frequently used in chemistry to represent substances reacting and diffusing over the spatial domain.
 - Multiple substances are spreading over the space while transforming into each other and at the end, a Turing pattern is formed.



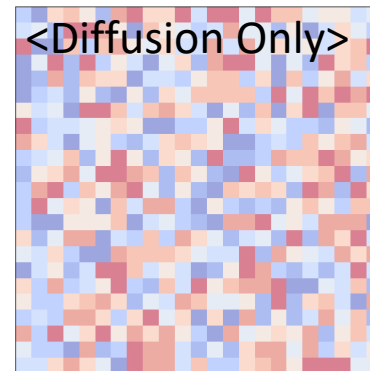
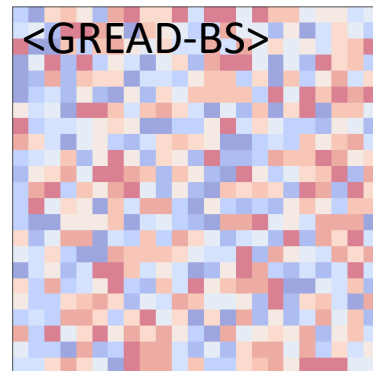
Reaction-diffusion equations – contd.

- In computer vision, it is known that an alternating sequence of the blurring (e.g., $x_t = -\tilde{L}x$) and sharpening (e.g., $x_t = \tilde{L}x$) operations also creates Turing patterns.



Reaction-diffusion equations – contd.

- The following visualization (in our ICML paper) also delivers the intuition of GREAD's successful node classification.
 - Assume a 2D grid network with 1D node signal (red is high signal, blue is low signal).



Graph neural reaction-diffusion networks

- Given a graph signal $\mathbf{X} \in \mathbb{R}^{N \times D}$, GREAD consists of the following three parts:
 - Initial embedding layer: $\mathbf{H}(0) = \mathbf{e}(\mathbf{X})$,
 - **Reaction-diffusion** layer: $\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T -\alpha \tilde{\mathbf{L}} \mathbf{H}(t) + \beta \mathbf{r}(\mathbf{H}(t)) dt$,
 - Output layer: $\hat{\mathbf{y}} = \mathbf{o}(\mathbf{H}(T))$.
- We also proposed to learn a normalized adjacency matrix $\bar{\mathbf{A}}$ as in [Li et al., 2018], and its Laplacian counterpart is $\bar{\mathbf{L}} = (\mathbf{I} - \bar{\mathbf{A}})$.
 - We use the self-attention method to learn a graph from data.

Blurring-sharpening equations

- Given a hidden signal $\mathbf{H}(t) \in \mathbb{R}^{N \times D}$ at time (or layer) t , we apply the blurring operation:

$$\mathbf{B}(t) = \mathbf{H}(t) - \bar{\mathbf{L}}\mathbf{H}(t) = \mathbf{H}(t) + (\bar{\mathbf{A}} - \mathbf{I})\mathbf{H}(t) = \bar{\mathbf{A}}\mathbf{H}(t).$$

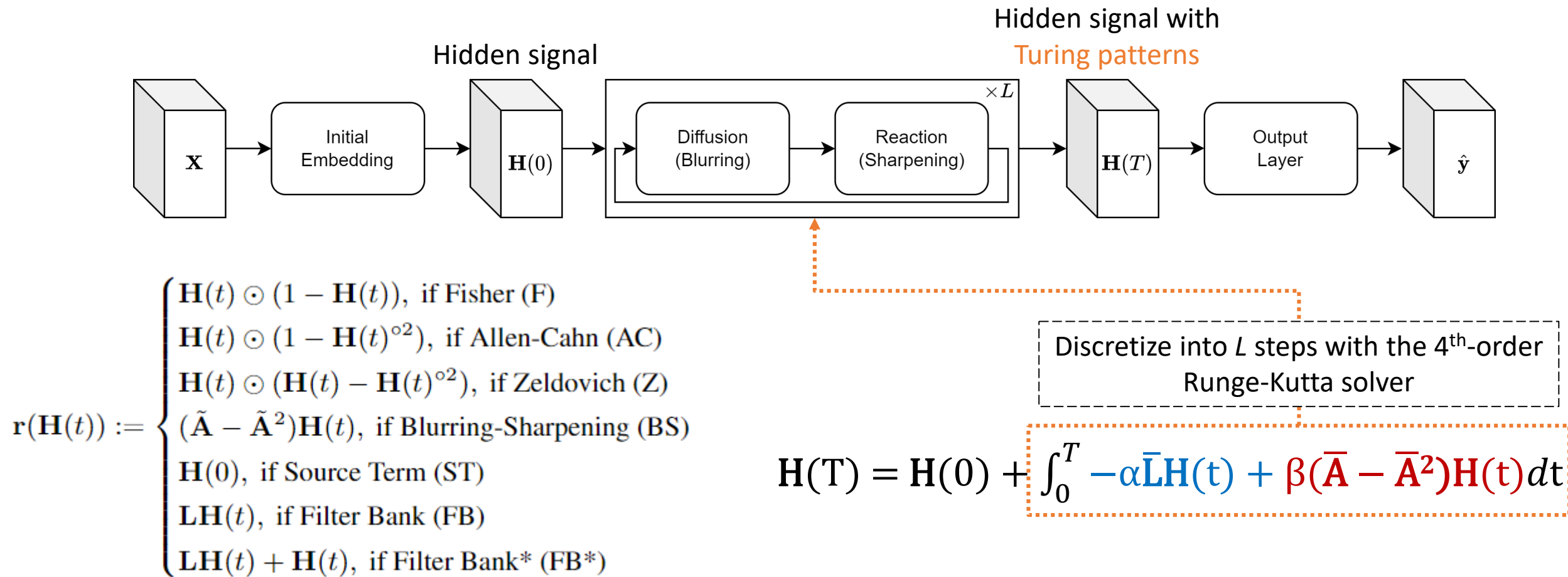
- We then apply the following sharpening operation:

$$\mathbf{H}(t+1) = \mathbf{B}(t) + \bar{\mathbf{L}}\mathbf{B}(t) = \mathbf{H}(t) - \bar{\mathbf{L}}\mathbf{H}(t) + (\bar{\mathbf{A}} - \bar{\mathbf{A}}^2)\mathbf{H}(t).$$

- Therefore, our main proposed model, GREAD-BS, is as follows:

$$\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T -\alpha \bar{\mathbf{L}}\mathbf{H}(t) + \beta (\bar{\mathbf{A}} - \bar{\mathbf{A}}^2)\mathbf{H}(t) dt.$$

Architecture of GREAD-BS



Experimental environments

- We consider 9 homophily and heterophily node classification datasets.
 - Neighboring nodes in a graph tend to have the same class label as its homophily rate increases (or as its heterophily rate decreases).
- We also consider 28 baselines.
- The source codes/datasets and their reproducibility information is at <https://github.com/jeongwhanchoi/gread>.

Table 3. Benchmark dataset properties and statistics

Dataset	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed
Classes	5	5	5	5	5	5	6	7	3
Features	1,703	1,703	1,703	932	2,089	235	1,433	3,703	500
Nodes	183	251	183	7,600	5,201	2,277	2,708	3,327	19,717
Edges	279	466	277	26,752	198,353	31,371	5,278	4,552	44,324
Hom. ratio	0.11	0.21	0.30	0.22	0.22	0.23	0.81	0.74	0.80

Experimental results

Table 2. The average ranking/accuracy and the Olympic ranking of some selected high-performing models on 9 real-world datasets. ‘*’ (resp. ‘†’) indicates that an improvement over GloGNN (resp. ACM-GCN) is statistically significant ($p < 0.05$) under the Wilcoxon signed-rank test.

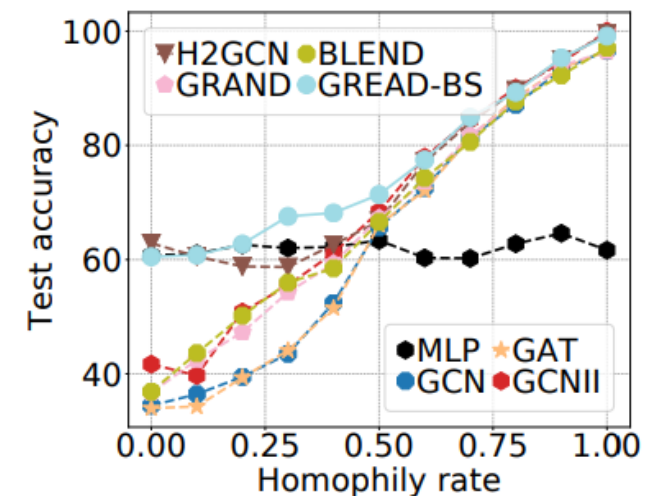
Method	Average		Olympic Ranking		
	Ranking	Accuracy	Gold	Silver	Bronze
GREAD-BS	1.56	76.64 ^{*†}	5	4	0
GREAD-FB*	6.72	74.51 [†]	0	0	3
GREAD-F	7.50	74.13	1	1	1
GloGNN	8.17	74.99	0	0	1
GREAD-AC	8.50	73.71	1	0	0
ACM-GCN	8.67	74.92	0	1	0
GGCN	9.50	75.05	0	1	0
Sheaf	10.33	75.06	1	1	0

Table 4. Results on real-world datasets: mean \pm std. dev. accuracy for 10 different data splits. We show the best three methods in red (first), blue (second), and purple (third). Other missing 16 baselines are in Appendix B.

Dataset	Texas	Wisconsin	Cornell	Film	Squirrel	Chameleon	Cora	Citeseer	PubMed	Avg.
Geom-GCN	66.76 \pm 2.72	64.51 \pm 3.66	60.54 \pm 3.67	31.59 \pm 1.15	38.15 \pm 0.92	60.00 \pm 2.81	85.35 \pm 1.57	78.02\pm1.15	89.95 \pm 0.47	63.87
H2GCN	84.86 \pm 7.23	87.65 \pm 4.98	82.70 \pm 5.28	35.70 \pm 1.00	36.48 \pm 1.86	60.11 \pm 2.15	87.87 \pm 1.20	77.11 \pm 1.57	89.49 \pm 0.38	71.33
GGCN	84.86 \pm 4.55	86.86 \pm 3.29	85.68 \pm 6.63	37.54 \pm 1.56	55.17 \pm 1.58	71.14\pm1.84	87.95 \pm 1.05	77.14 \pm 1.45	89.15 \pm 0.37	75.05
LINKX	74.60 \pm 8.37	75.49 \pm 5.72	77.84 \pm 5.81	36.10 \pm 1.55	61.81\pm1.80	68.42 \pm 1.38	84.64 \pm 1.13	73.19 \pm 0.99	87.86 \pm 0.77	71.11
GloGNN	84.32 \pm 4.15	87.06 \pm 3.53	83.51 \pm 4.26	37.35 \pm 1.30	57.54\pm1.39	69.78 \pm 2.42	88.31 \pm 1.13	77.41 \pm 1.65	89.62 \pm 0.35	74.99
ACM-GCN	87.84 \pm 4.40	88.43\pm3.22	85.14 \pm 6.07	36.28 \pm 1.09	54.40 \pm 1.88	66.93 \pm 1.85	87.91 \pm 0.95	77.32 \pm 1.70	90.00 \pm 0.52	74.92
GCNII	77.57 \pm 3.83	80.39 \pm 3.40	77.86 \pm 3.79	37.44 \pm 1.30	38.47 \pm 1.58	63.86 \pm 3.04	88.37\pm1.25	77.33 \pm 1.48	90.15\pm0.43	70.16
CGNN	71.35 \pm 4.05	74.31 \pm 7.26	66.22 \pm 7.69	35.95 \pm 0.86	29.24 \pm 1.09	46.89 \pm 1.66	87.10 \pm 1.35	76.91 \pm 1.81	87.70 \pm 0.49	63.96
GRAND	75.68 \pm 7.25	79.41 \pm 3.64	82.16 \pm 7.09	35.62 \pm 1.01	40.05 \pm 1.50	54.67 \pm 2.54	87.36 \pm 0.96	76.46 \pm 1.77	89.02 \pm 0.51	68.94
BLEND	83.24 \pm 4.65	84.12 \pm 3.56	85.95 \pm 6.82	35.63 \pm 1.01	43.06 \pm 1.39	60.11 \pm 2.09	88.09 \pm 1.22	76.63 \pm 1.60	89.24 \pm 0.42	71.79
Sheaf	85.05 \pm 5.51	89.41\pm4.74	84.86 \pm 4.71	37.81\pm1.15	56.34 \pm 1.32	68.04 \pm 1.58	86.90 \pm 1.13	76.70 \pm 1.57	89.49 \pm 0.40	75.06
GRAFF	88.38\pm4.53	87.45 \pm 2.94	83.24 \pm 6.49	36.09 \pm 0.81	54.52 \pm 1.37	71.08\pm1.75	87.61 \pm 0.97	76.92 \pm 1.70	88.95 \pm 0.52	74.92
GREAD-BS	88.92\pm3.72	89.41\pm3.30	86.49\pm7.15	37.90\pm1.17	59.22\pm1.44	71.38\pm1.31	88.57\pm0.66	77.60\pm1.81	90.23\pm0.55	76.64
GREAD-F	89.73\pm4.49	86.47 \pm 4.84	86.49\pm5.13	36.72 \pm 0.66	46.16 \pm 1.44	65.20 \pm 1.40	88.39 \pm 0.91	77.40 \pm 1.54	90.09 \pm 0.31	74.13
GREAD-AC	85.95 \pm 2.65	86.08 \pm 3.56	87.03\pm4.95	37.21 \pm 1.10	45.10 \pm 2.11	65.09 \pm 1.08	88.29 \pm 0.67	77.38 \pm 1.53	90.10 \pm 0.36	73.71
GREAD-Z	87.30 \pm 5.68	86.29 \pm 4.32	85.68 \pm 5.41	37.01 \pm 1.11	46.25 \pm 1.72	62.70 \pm 2.30	88.31 \pm 1.10	77.39 \pm 1.90	90.11 \pm 0.27	73.45
GREAD-ST	81.08 \pm 5.67	86.67 \pm 3.01	86.22\pm5.98	37.66 \pm 0.90	45.83 \pm 1.40	63.03 \pm 1.32	88.47\pm1.19	77.25 \pm 1.47	90.13\pm0.36	72.93
GREAD-FB	86.76 \pm 5.05	87.65 \pm 3.17	86.22\pm5.85	37.40 \pm 0.55	50.83 \pm 2.27	66.05 \pm 1.21	88.03 \pm 0.78	77.28 \pm 1.73	90.07 \pm 0.45	74.48
GREAD-FB*	87.03 \pm 3.97	88.04\pm1.63	85.95 \pm 5.64	37.70\pm0.51	50.57 \pm 1.52	65.83 \pm 1.10	88.01 \pm 0.80	77.42\pm1.93	90.08 \pm 0.46	74.51

Experimental results – contd.

- We also have other experimental results.
 - Learning a normalized adjacency matrix $\bar{\mathbf{A}}$ is better.
 - α and β should be vectors in $-\alpha\bar{\mathbf{L}}\mathbf{H}(t) + \beta(\bar{\mathbf{A}} - \bar{\mathbf{A}}^2)\mathbf{H}(t)$.
 - An optimal T varies in $\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T -\alpha\bar{\mathbf{L}}\mathbf{H}(t) + \beta(\bar{\mathbf{A}} - \bar{\mathbf{A}}^2)\mathbf{H}(t)dt$.
 - An optimal ODE step-size h varies when solving $\int_0^T -\alpha\bar{\mathbf{L}}\mathbf{H}(t) + \beta(\bar{\mathbf{A}} - \bar{\mathbf{A}}^2)\mathbf{H}(t)dt$ with RK4.
 - GREAD-BS works well irrespective of the homophily rate.

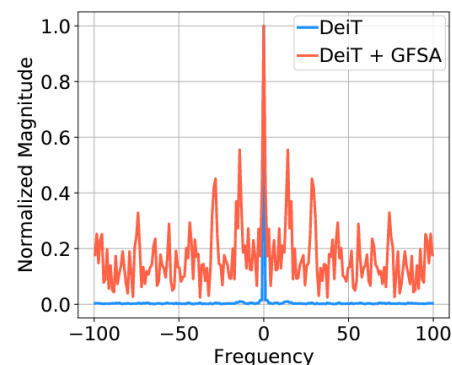


Transformers with graph filters

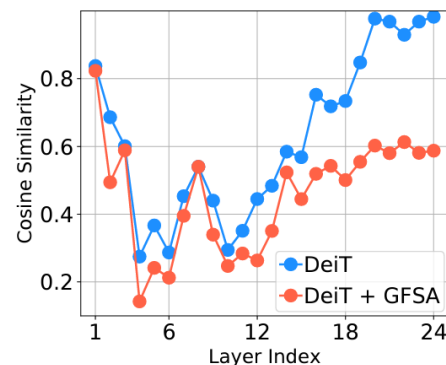
- The self-attention layer consists of a GCN and a residual connection.
 - The normalized adjacency matrix (or the attention map) is generated.

$$\text{SA}(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{X}\mathbf{W}_{\text{key}}(\mathbf{X}\mathbf{W}_{\text{qry}})^{\top}}{\sqrt{d}}\right)\mathbf{X}\mathbf{W}_{\text{val}} = \bar{\mathbf{A}}\mathbf{X}\mathbf{W}_{\text{val}}$$

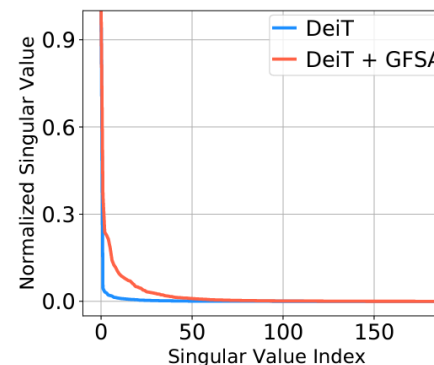
- Redesigning the self-attention layer with advanced graph filters leads to non-trivial enhancements in various domains.



(a) Filter response

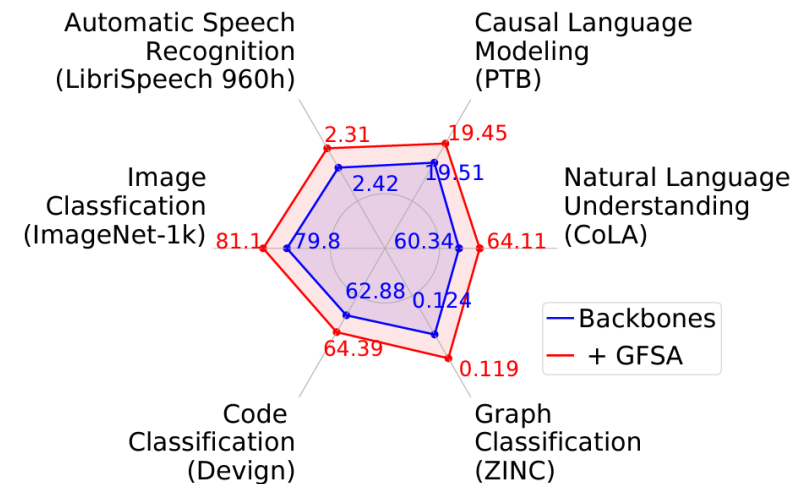


(b) Cosine similarity



(c) Singular value

<DeiT on ImageNet-1K>

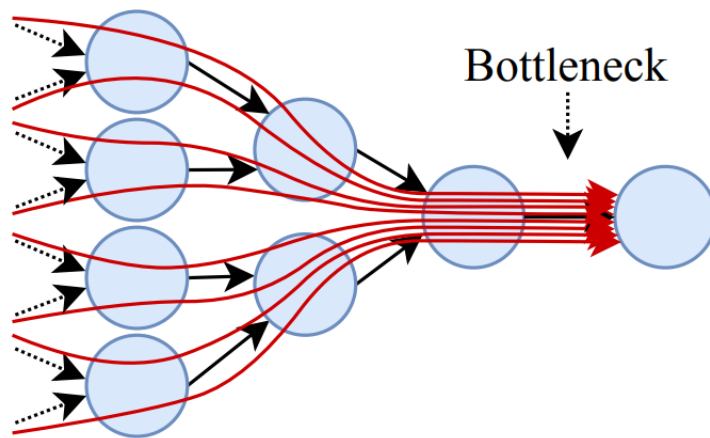


Overcoming the oversquashing problem

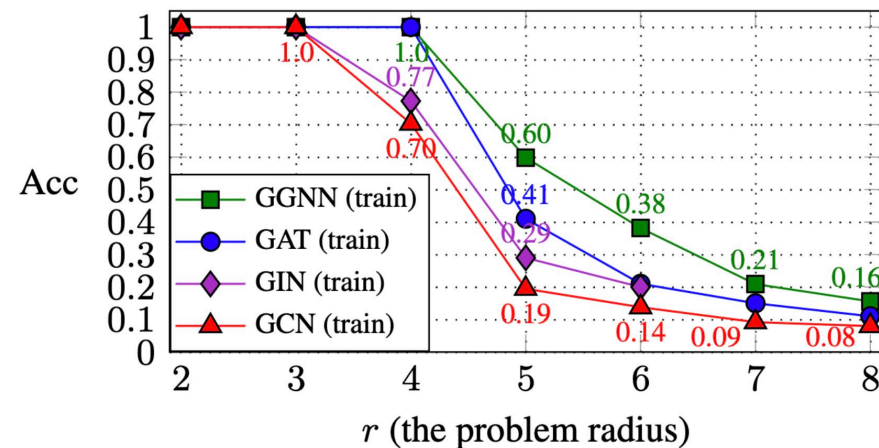
<Choi et al., PANDA: Expanded Width-Aware Message Passing Beyond Rewiring,
ICML 2024>

Oversquashing problem

- The oversquashing problem was introduced in [Alon & Yahav, 2021].
 - information from a node's exponentially-growing receptive field is compressed into a fixed-size vector



<Oversquashing example>

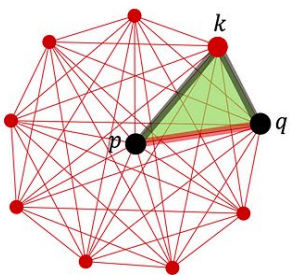


<Accuracy across problem radius (tree depth)>

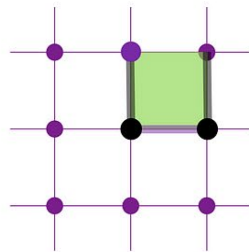
Long-distance dependency + Fast volume growth = Oversquashing

Oversquashing problem – contd.

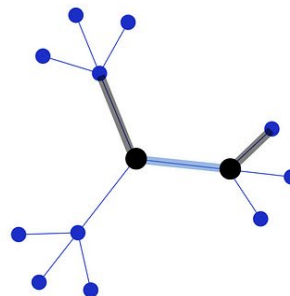
- Following the study by Alon & Yahav, it became popular to find indicators for oversquashing and propose rewiring methods.
- The Ricci curvature was used in [Topping et al., 2021].
 - In differential geometry, a natural object that allows us to distinguish different geometries is the Ricci curvature.
 - The oversquashing problem is caused by strongly negatively-curved edges.



Clique (>0)

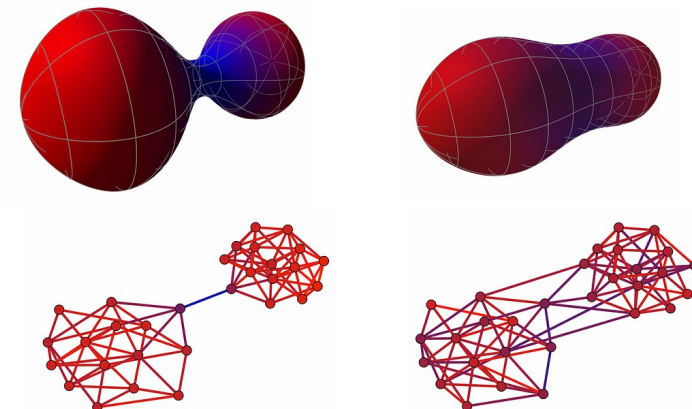


Grid ($=0$)



Tree (<0)

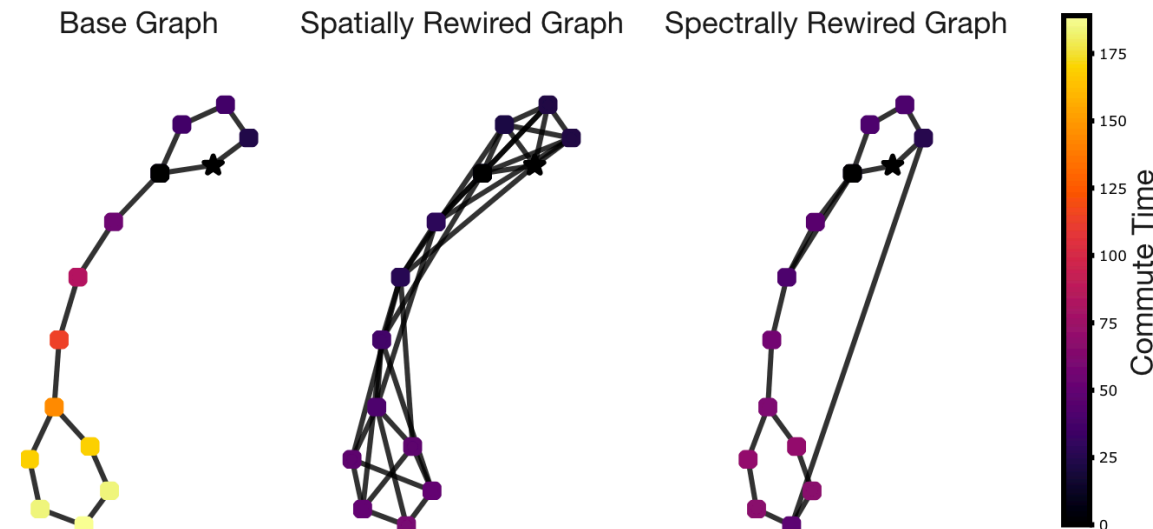
<Ricci curvature on graphs>



<An analogous process of curvature-based graph rewiring>

Oversquashing problem – contd.

- Large commute-time distances contribute to oversquashing.
 - Spectral rewiring: increase the Cheeger constant of the graph (“clusterdness”), which leads to lower commute time [Karhadkar et al., 2022, Arnaiz-Rodríguez et al., 2022].
 - Spatial rewiring: inserting edges reduces the total effective resistance of the graph (=commute-time distance up to scale) [Topping et al., 2021, Deac et al., 2022].



<Effect of different rewiring methods on the graph connectivity in [Di Giovanni et al., 2023]>

Oversquashing problem – contd.

- Let us consider an MPNN (GNN) of the following form.

$$\mathbf{h}_v^{(\ell+1)} = \phi^{(\ell)} \left(\mathbf{h}_v^{(\ell)}, \psi^{(\ell)}(\{\mathbf{h}_u^{(\ell)} : u \in \mathcal{N}(v)\}) \right)$$

- Small Jacobian norms indicate poor information propagations [Di Giovanni et al., 2023].

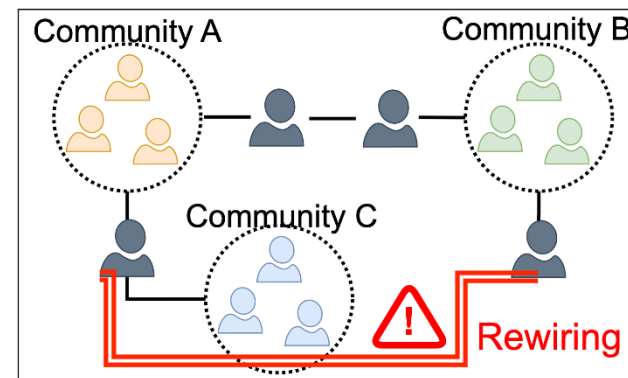
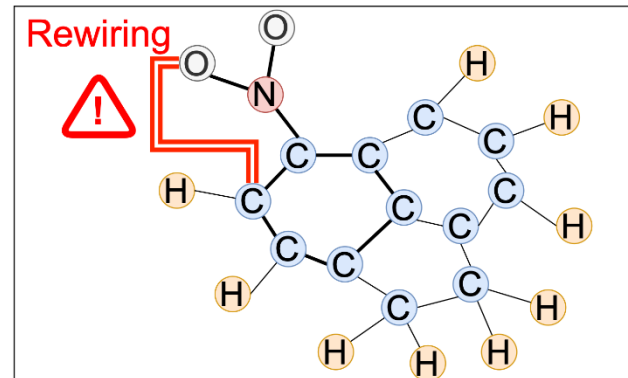
- L = depth (number of layers)
- p = width (hidden dimension)
- z = Lipschitz constant
- w = maximum element of weight matrices

Theorem (Sensitivity bound): For any $u, v \in V$

$$\left\| \frac{\partial \mathbf{h}_v^{(\ell)}}{\partial \mathbf{h}_u^{(0)}} \right\|_1 \leq \underbrace{(zwp)^L}_{\text{model}} \underbrace{(\mathbf{I} + \mathbf{A})^L_{ij}}_{\text{topology}}$$

Limitations of rewiring methods

- Existing rewiring methods only focus on rewiring that changes the graph topology to address oversquashing.
- The rewiring methods can inadvertently introduce inaccuracies within domain-specific contexts.



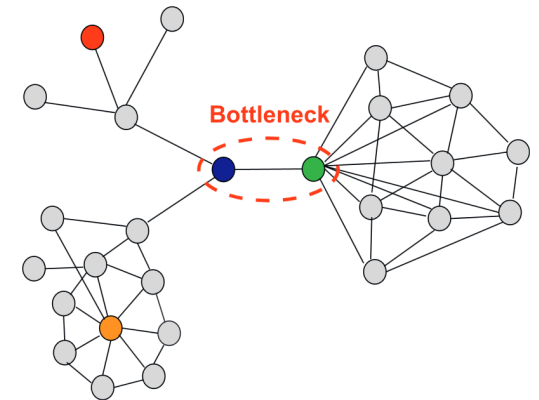
<Potential pitfalls of rewiring in domain-specific graphs>

Motivations

- Di Giovanni et al. (2023) provide a theoretical justification that increasing the width of the model (i.e., the hidden size) can also improve its sensitivity.

Theorem (Sensitivity bound): For any $u, v \in V$

$$\left\| \frac{\partial \mathbf{h}_v^{(\ell)}}{\partial \mathbf{h}_u^{(0)}} \right\|_1 \leq \underbrace{(zwp)^L}_{\text{model}} \underbrace{(\mathbf{I} + \mathbf{A})^L}_{\text{topology}}_{ij}$$



<Bottleneck nodes [Yu et al, 2007]>

- We aim to design a new message passing paradigm that mitigates oversquashing by selectively expanding the widths of bottleneck nodes.

Proposed method

- We can define bottleneck nodes as high centrality nodes in terms of betweenness centrality and so on [Yu et al., 2007, Topping et al., 2022].
- Increasing the hidden widths of the bottleneck nodes enables capturing more information.

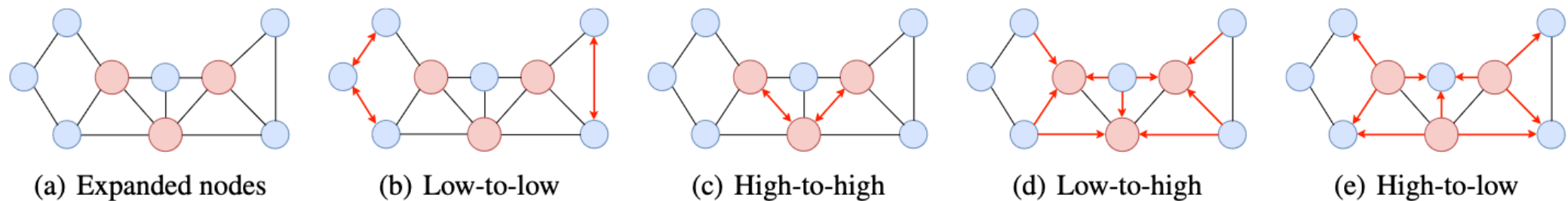
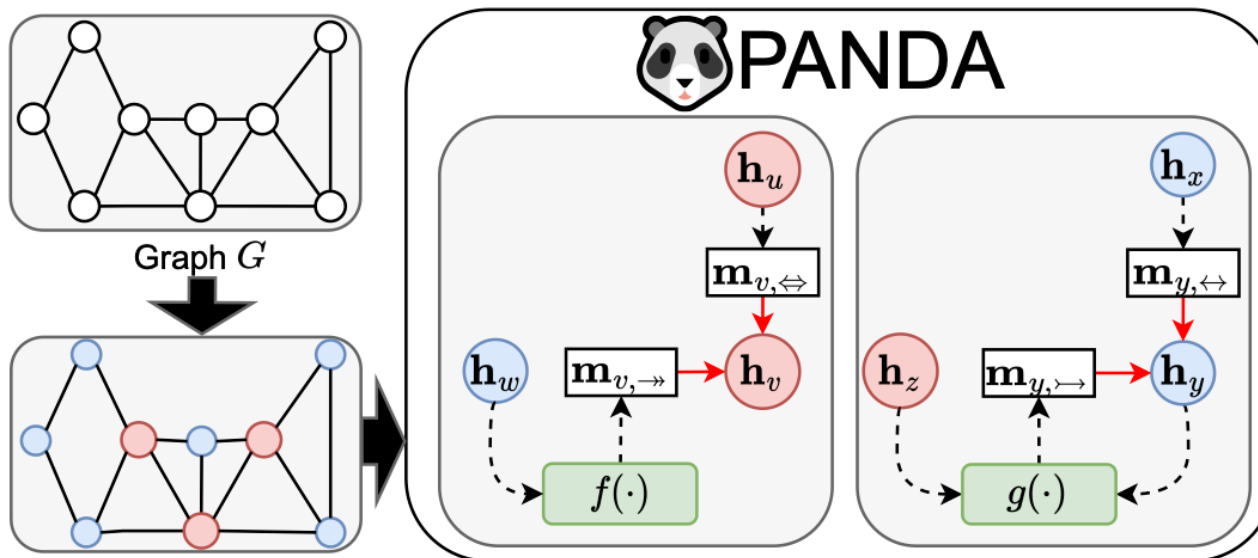


Figure 3. Examples of PANDA's message passing mechanism. The size of the node indicates the size of hidden dimension.

Proposed method – contd.

- Our PANDA message passing works in the following way.
 - First, we selectively expand widths according to centrality measures.
 - Then, our PANDA message passing enables signal propagation among nodes with different widths (low and high-width nodes).



$f(\cdot)$: A linear transformation that expands the width of low-dim nodes.

$g(\cdot)$: A dimension selector that selects dimensions to be propagated from high-dim nodes to low-dim nodes.

Experimental Results

Table 1. Results of PANDA and baselines for GCN and GIN. We show the best three in red (first), blue (second), and purple (third).

Method	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN (None)	68.255 \pm 1.098	49.770 \pm 0.817	72.150 \pm 2.442	27.667 \pm 1.164	70.982 \pm 0.737	33.784 \pm 0.488
+ Last Layer FA	68.485 \pm 0.945	48.980 \pm 0.945	70.050 \pm 2.027	26.467 \pm 1.204	71.018 \pm 0.963	33.320 \pm 0.435
+ Every Layer FA	48.490 \pm 1.044	48.170 \pm 0.801	70.450 \pm 1.960	18.333 \pm 1.038	60.036 \pm 0.925	51.798 \pm 0.419
+ DIGL	49.980 \pm 0.680	49.910 \pm 0.841	71.350 \pm 2.391	27.517 \pm 1.053	70.607 \pm 0.731	15.530 \pm 0.294
+ SDRF	68.620 \pm 0.851	49.400 \pm 0.904	71.050 \pm 1.872	28.367 \pm 1.174	70.920 \pm 0.792	33.448 \pm 0.472
+ FoSR	70.330 \pm 0.727	49.660 \pm 0.864	80.000 \pm 1.574	25.067 \pm 0.994	73.420 \pm 0.811	33.836 \pm 0.584
+ BORF	Time-out	50.100 \pm 0.900	75.800 \pm 1.900	24.700 \pm 1.000	71.000 \pm 0.800	Time-out
+ GTR	68.990 \pm 0.610	49.920 \pm 0.990	79.100 \pm 1.860	27.520 \pm 0.990	72.590 \pm 2.480	33.050 \pm 0.400
+ CT-Layer	51.580 \pm 1.019	50.320 \pm 0.944	75.899 \pm 3.024	17.383 \pm 1.030	60.357 \pm 1.060	52.146 \pm 0.415
+ PANDA	80.690 \pm 0.721	63.760 \pm 1.012	85.750 \pm 1.396	31.550 \pm 1.230	76.000 \pm 0.774	68.400 \pm 0.452
GIN (None)	86.785 \pm 1.056	70.180 \pm 0.992	77.700 \pm 0.360	33.800 \pm 0.115	70.804 \pm 0.827	72.992 \pm 0.384
+ Last Layer FA	90.220 \pm 0.475	70.910 \pm 0.788	83.450 \pm 1.742	47.400 \pm 1.387	72.304 \pm 0.666	75.056 \pm 0.406
+ Every Layer FA	50.360 \pm 0.684	49.160 \pm 0.870	72.550 \pm 3.016	28.383 \pm 1.052	70.375 \pm 0.910	32.984 \pm 0.390
+ DIGL	76.035 \pm 0.774	64.390 \pm 0.907	79.700 \pm 2.150	35.717 \pm 1.198	70.759 \pm 0.774	54.504 \pm 0.410
+ SDRF	86.440 \pm 0.590	69.720 \pm 1.152	78.400 \pm 2.803	35.817 \pm 1.094	69.813 \pm 0.792	72.958 \pm 0.419
+ FoSR	87.350 \pm 0.598	71.210 \pm 0.919	78.400 \pm 2.803	29.200 \pm 1.367	75.107 \pm 0.817	73.278 \pm 0.416
+ BORF	Time-out	71.300 \pm 1.500	80.800 \pm 2.500	35.500 \pm 1.200	74.200 \pm 0.800	Time-out
+ GTR	86.980 \pm 0.660	71.280 \pm 0.860	77.600 \pm 2.840	30.570 \pm 1.420	73.130 \pm 0.690	72.930 \pm 0.420
+ CT-Layer	54.589 \pm 1.757	50.000 \pm 0.974	56.850 \pm 4.253	16.583 \pm 0.907	61.107 \pm 1.184	52.304 \pm 0.605
+ PANDA	91.055 \pm 0.402	72.560 \pm 0.917	88.750 \pm 1.570	46.200 \pm 1.410	75.759 \pm 0.856	75.200 \pm 0.481

Experimental Results – contd.

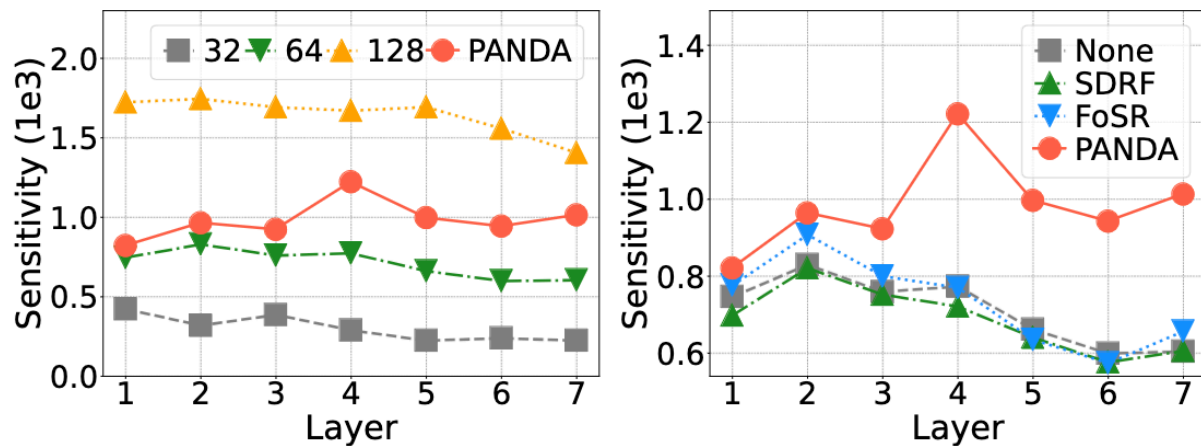
Method	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
R-GCN	49.850 \pm 0.653	50.012 \pm 0.917	69.250 \pm 2.085	28.600 \pm 1.186	69.518 \pm 0.725	33.602 \pm 1.047
PANDA-GCN	80.690 \pm 0.721	63.760 \pm 1.012	85.750 \pm 1.396	31.550 \pm 1.230	76.000 \pm 0.774	68.400 \pm 0.452

<PANDA-GCN vs. R-GCN>

$C(\mathcal{G})$	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
Degree	80.690 \pm 0.721	62.100 \pm 1.043	85.200 \pm 1.568	31.117 \pm 1.258	75.375 \pm 0.800	68.162 \pm 0.471
Betweenness	80.000 \pm 0.659	59.630 \pm 1.152	85.750 \pm 1.396	29.600 \pm 1.208	74.589 \pm 0.791	67.844 \pm 0.547
Closeness	79.700 \pm 0.664	61.160 \pm 0.992	84.700 \pm 1.554	29.967 \pm 1.231	76.000 \pm 0.774	68.400 \pm 0.452
PageRank	80.340 \pm 0.826	63.760 \pm 1.012	85.450 \pm 1.569	31.550 \pm 1.230	74.098 \pm 0.851	67.540 \pm 0.500
Load	79.500 \pm 0.732	59.840 \pm 1.153	85.700 \pm 1.549	28.167 \pm 1.090	74.188 \pm 0.814	67.802 \pm 0.506

<Performance comparison by various centrality measures for PANDA-GCN>

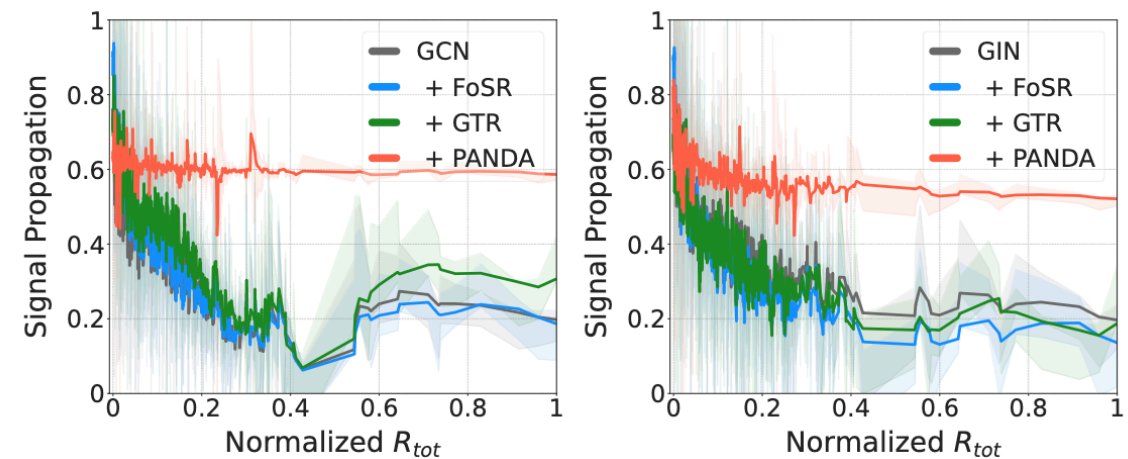
Experimental Results – contd.



(a) Sensitivity w.r.t. p

(b) Sensitivity w.r.t. methods

Empirical sensitivity across layers for GCN on MUTAG. Compared to other methods, PANDA shows higher sensitivity that is maintained even in deeper layer.



(a) GCN on REDDIT-BINARY

(b) GIN on REDDIT-BINARY

The amount of signal propagated across the graph w.r.t. the normalized total effective resistance. PANDA maintains continuous information flows even under high bottleneck conditions.

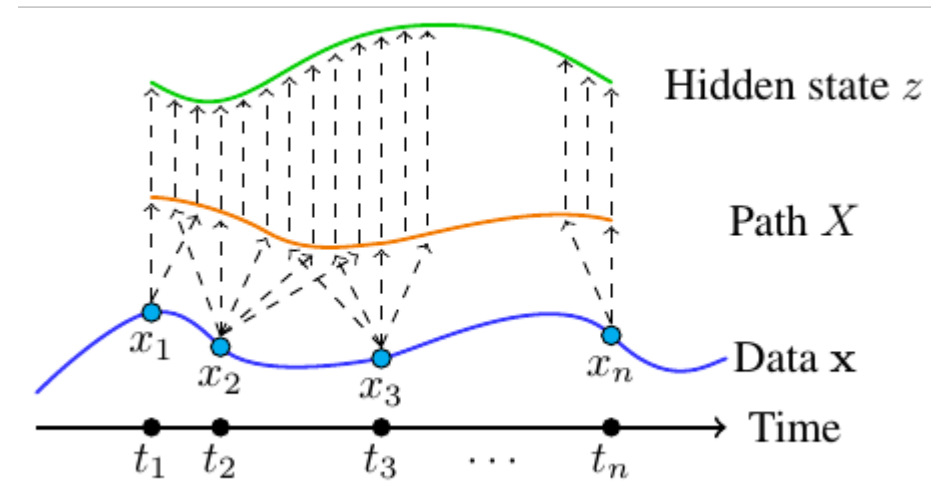
Neural Controlled Differential Equations

<Introduction>

Continuous-time recurrent neural networks

- NCDEs can be understood as continuous-time RNNs [Kidger et al., 2020].
 - NCDEs continuously model the hidden state z of RNNs.
 - The hidden state changes over time in response to X .

$$z_t = z_{t_0} + \int_{t_0}^t f_{\theta}(z_s) dX_s = z_{t_0} + \int_{t_0}^t f_{\theta}(z_s) \frac{dX}{ds}(s) ds$$



Generalized selective state-space models

- Deep SSMs are one of the candidates for the post-Transformer architecture.
 - For the past couple of years, there have been notable contributions.
 - For some tasks, they outperform Transformers.
- CDEs (in conjunction with the rough path theory) provide theoretical foundations for deep SSMs [Ciron et al., 2024].
 - Many SSM variants are special cases of CDEs.
 - Their uniform closure can be characterized by the rough path theory.
- The CDE-based generalization learns path-to-path operators in a balanced manner since it does not solely rely on the last neural network layer for non-linear computation.

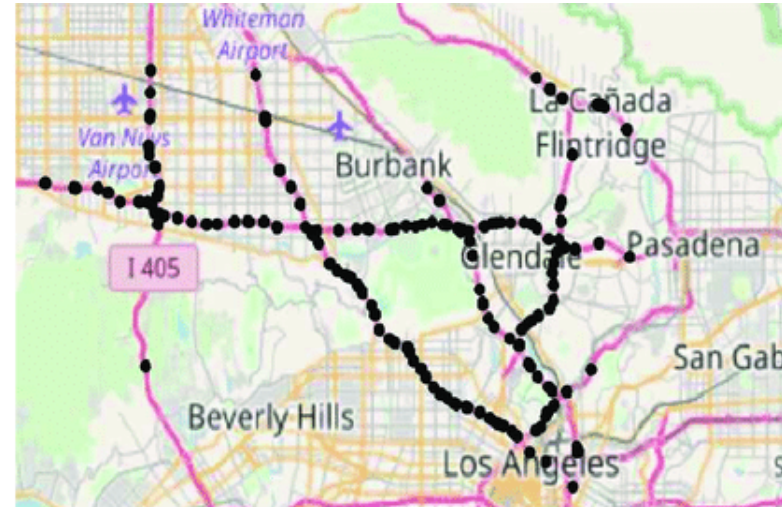
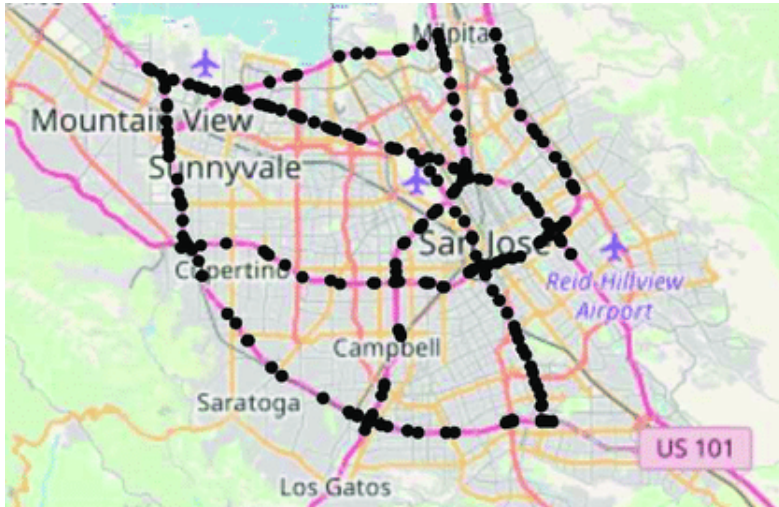
Combining the temporal and spatial processing for traffic forecasting

<Choi et al., Graph Neural Controlled Differential Equations for Traffic Forecasting,
AAAI, 2022>

Motivations

61

- Traffic forecasting has a high impact on our daily lives.
 - Many traffic sensors are deployed but often malfunctioning.
 - Irregular spatiotemporal observations are ubiquitous.
- We need a novel framework to process them and forecast future traffic conditions, e.g., traffic volume, speed, etc.



Proposed method

- We resort to the neural CDE technology for its robustness to irregularity.
 - Each vertex means a traffic sensor.
 - There is a graph of traffic sensors.

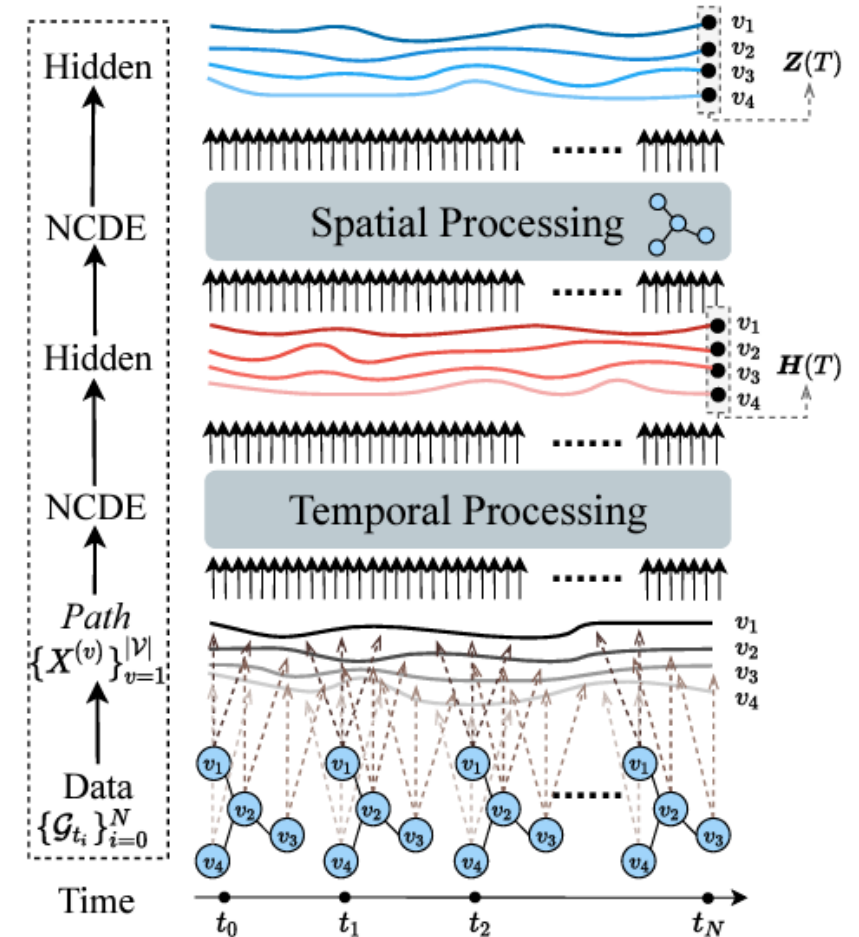


Figure 1: The overall workflow in our proposed STG-NCDE

Proposed method

- $H(T)$ means the set of hidden states of nodes.
- We process each node separately.

$$\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T f(\mathbf{H}(t); \theta_f) \frac{d\mathbf{X}(t)}{dt} dt$$

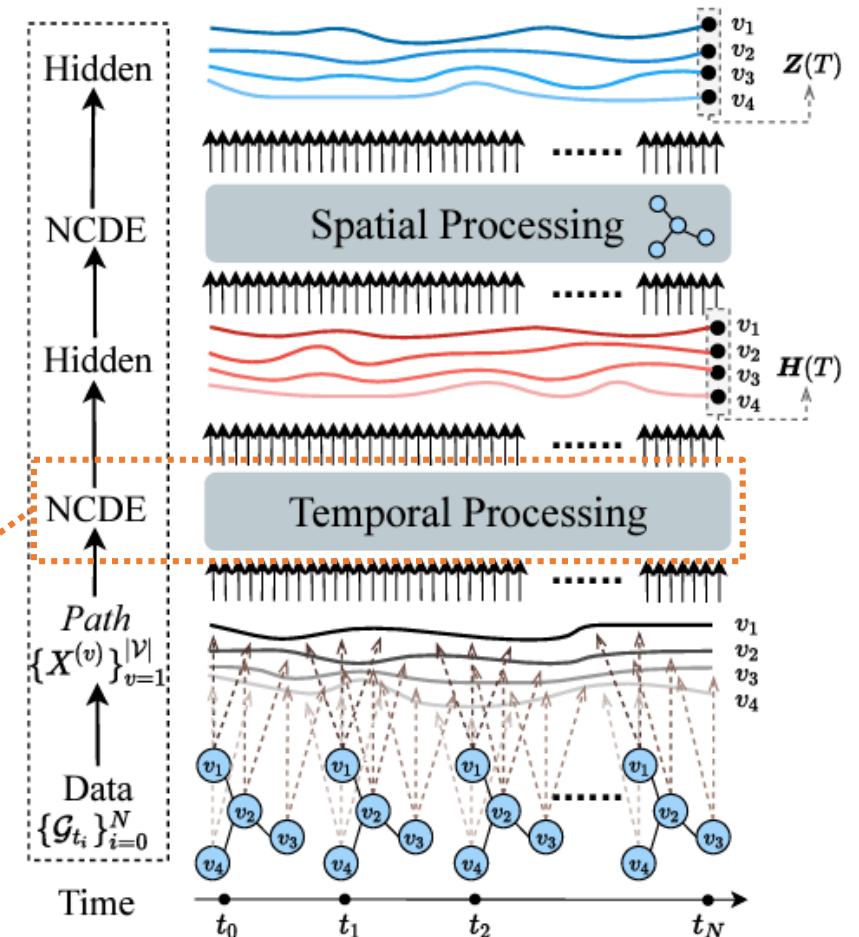


Figure 1: The overall workflow in our proposed STG-NCDE

Proposed method

- Given an adjacency matrix, we merge the hidden state derivatives of nodes, by using GCN [Kipf & Welling, 2017].

$$\mathbf{Z}(T) = \mathbf{Z}(0) + \int_0^T \underbrace{g(\mathbf{Z}(t); \theta_g) \frac{d\mathbf{H}(t)}{dt}}_{\text{This is a GCN.}} dt$$

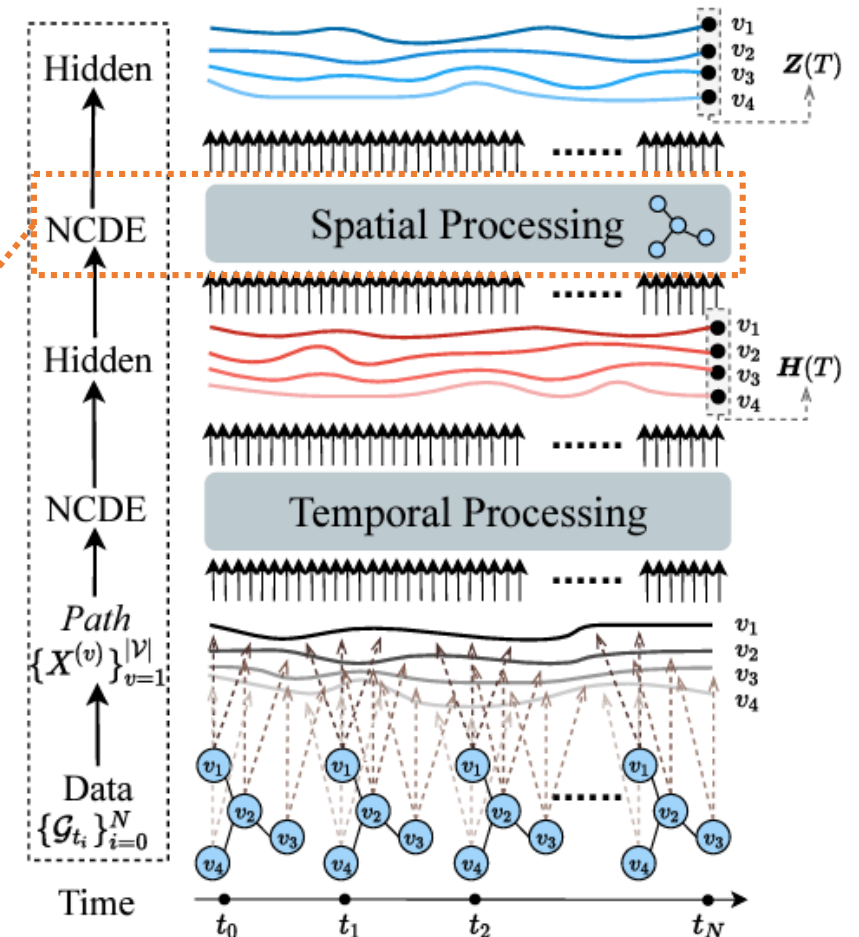


Figure 1: The overall workflow in our proposed STG-NCDE

Proposed method

- Two ODEs can be merged into a single ODE.

$$\mathbf{Z}(T) = \mathbf{Z}(0) + \int_0^T g(\mathbf{Z}(t); \boldsymbol{\theta}_g) f(\mathbf{H}(t); \boldsymbol{\theta}_f) \frac{d\mathbf{X}(t)}{dt} dt$$

<Merge them>

$$\mathbf{Z}(T) = \mathbf{Z}(0) + \int_0^T g(\mathbf{Z}(t); \boldsymbol{\theta}_g) \frac{d\mathbf{H}(t)}{dt} dt$$

$$\mathbf{H}(T) = \mathbf{H}(0) + \int_0^T f(\mathbf{H}(t); \boldsymbol{\theta}_f) \frac{d\mathbf{X}(t)}{dt} dt$$

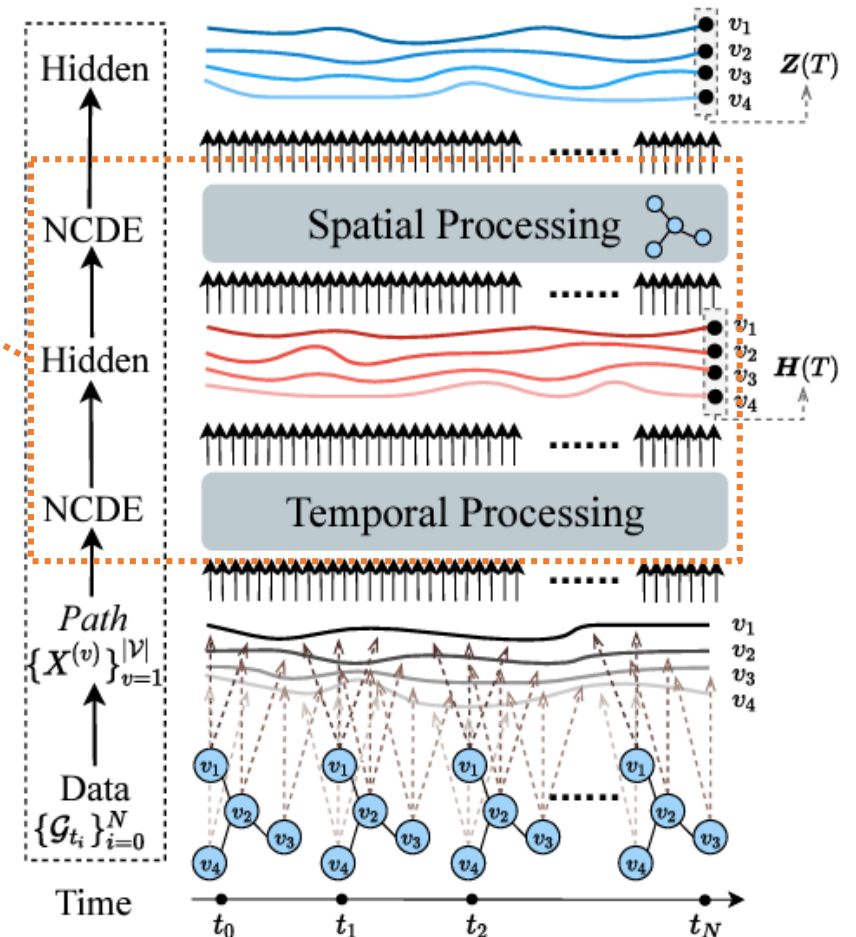
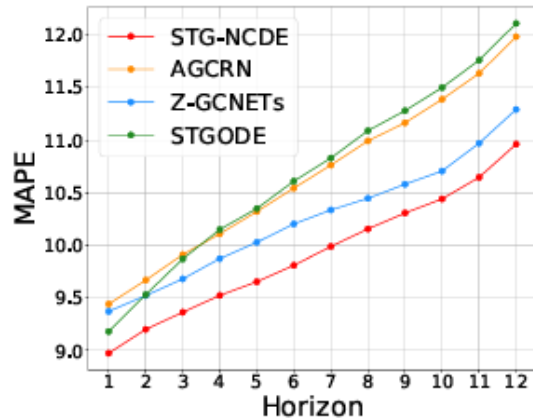


Figure 1: The overall workflow in our proposed STG-NCDE

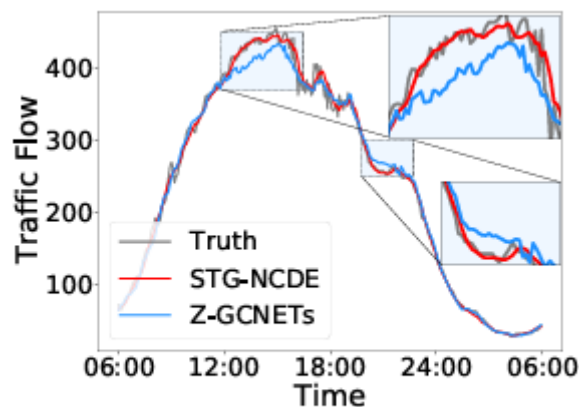
Experimental environments & results

Dataset	$ \mathcal{V} $	Time Steps	Time Range	Type
PeMSD3	358	26,208	09/2018 - 11/2018	Volume
PeMSD4	307	16,992	01/2018 - 02/2018	Volume
PeMSD7	883	28,224	05/2017 - 08/2017	Volume
PeMSD8	170	17,856	07/2016 - 08/2016	Volume
PeMSD7(M)	228	12,672	05/2012 - 06/2012	Velocity
PeMSD7(L)	1,026	12,672	05/2012 - 06/2012	Velocity

Table 1: The summary of the datasets used in our work. We predict either traffic volume (i.e., # of vehicles) or velocity.



(d) MAPE on PeMSD8



(b) Node 261 in PeMSD4

Model	MAE	RMSE	MAPE
STGCN	14.88 (117.0%)	24.22 (113.6%)	12.30 (121.8%)
DCRNN	14.90 (117.1%)	24.04 (112.7%)	12.75 (126.1%)
GraphWaveNet	15.94 (125.3%)	26.22 (122.9%)	12.96 (128.2%)
ASTGCN(r)	14.86 (116.9%)	23.95 (112.3%)	12.25 (121.3%)
STSGCN	14.45 (113.5%)	23.58 (110.5%)	11.42 (113.0%)
AGCRN	13.32 (104.7%)	22.29 (104.5%)	10.37 (102.7%)
STFGNN	13.92 (109.5%)	22.57 (105.8%)	11.30 (111.9%)
STGODE	13.56 (106.6%)	22.37 (104.8%)	10.77 (106.6%)
Z-GCNETs	13.22 (104.0%)	21.92 (102.7%)	10.44 (103.4%)
STG-NCDE	12.72 (100.0%)	21.33 (100.0%)	10.10 (100.0%)

Table 2: The average error of some selected highly performing models across all the six datasets. Inside the parentheses, we show their performance relative to our method.

Table 6: Forecasting error on irregular PeMSD8. More results in other datasets are in Appendix.

Model	Missing rate	MAE	RMSE	MAPE
STG-NCDE		15.68	24.96	10.05%
Only Temporal	10%	21.18	33.02	13.26%
Only Spatial		16.85	26.63	11.12%
STG-NCDE		16.21	25.64	10.43%
Only Temporal	30%	21.46	33.37	13.57%
Only Spatial		18.46	29.03	12.16%
STG-NCDE		16.68	26.17	10.67%
Only Temporal	50%	22.68	35.14	14.11%
Only Spatial		17.98	28.12	11.87%

Deep Learning for Science

The background is a dark blue gradient. It features numerous faint, white-outlined rectangles of various sizes, some of which contain illegible text, suggesting overlapping documents or data pages. Scattered throughout the background are strings of binary code (0s and 1s) in a light blue or white color, some appearing to float or be part of a larger digital pattern.

What are partial differential equations?

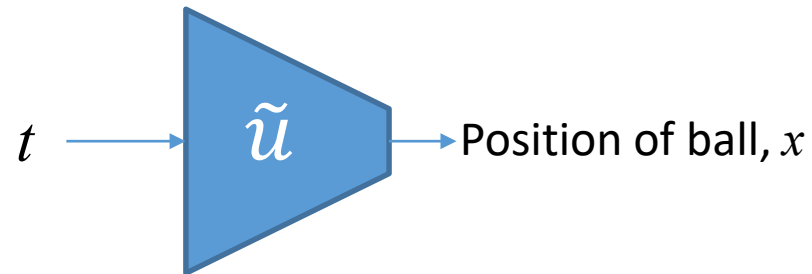
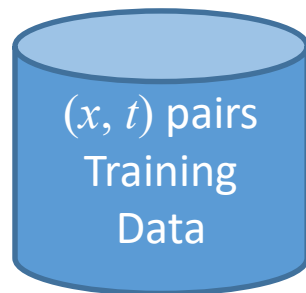
- The second law of motion $\vec{F} = m\vec{a} = \frac{d}{dt}(m\vec{v}) = \frac{d^2}{dt^2}(m\vec{u})$, where $\vec{u} = (x, y, z, t)$ is a coordinate of an object at time t can be extended to other fields, e.g., fluid dynamics where ρ replaces m .
- The Black-Scholes equation is a Nobel Prize-awarded model for the dynamics of the European option market.
 - The spatiotemporal coordinate (x, y, z, t) can be replaced with the coordinate (s, t) , where s is the underlying asset price.
- Likewise, PDEs are the essential language describing the natural/social/financial dynamics.

Physics-informed Neural Networks

<Introduction>

An example of PINNs

- Suppose a regression task to predict the position of a falling ball given time t .



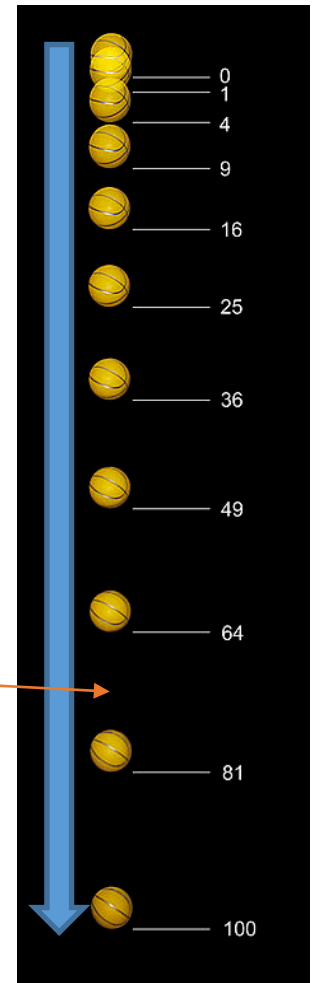
- There is one known governing equation that \tilde{u} should follow:

$$u_{tt} - g = 0, \text{ where } g = 9.80665 m/s^2.$$

- We can use the following loss with no training data:

$$(\tilde{u}(0; \theta) - 0) + (\text{tf.grad}(\text{tf.grad}(\tilde{u}(t; \theta), t), t) - 9.80665).$$

Query about time t ?



Training PINNs

- PINNs parameterize both the solution u and the governing equation f .

- $\tilde{u}(x, t; \theta)$: neural network approximation of the solution $u(x, t)$
- $\tilde{f}(x, t; \theta)$: neural network approximation of the governing equation f
- The neural network \tilde{f} shares the same network weights with \tilde{u} .

- In the case of the inviscid Burgers' equation, for instance,

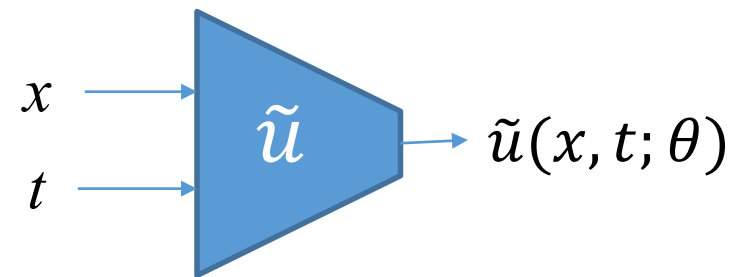
$$\tilde{f}(x, t, \tilde{u}; \theta) = \text{tf.grad}(\tilde{u}(x, t; \theta), t) + \tilde{u}(x, t; \theta) \text{tf.grad}(\tilde{u}(x, t; \theta), x).$$

- PINNs train θ with the following loss:

- $L \stackrel{\text{def}}{=} w_u L_u + w_f L_f$

- $L_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(x_u^i, t_u^i) - \tilde{u}(x_u^i, t_u^i; \theta)|^2$

- $L_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |\tilde{f}(x_f^i, t_f^i, \tilde{u}; \theta)|^2$

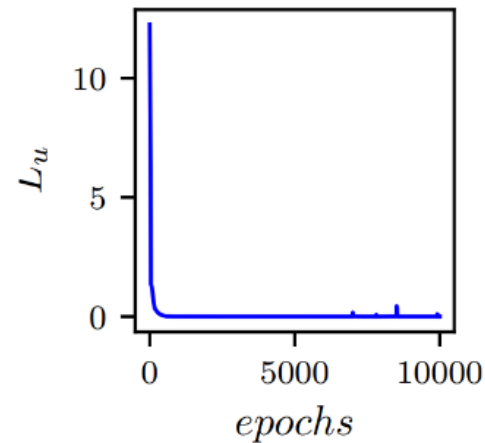


Stabilizing the PINN training

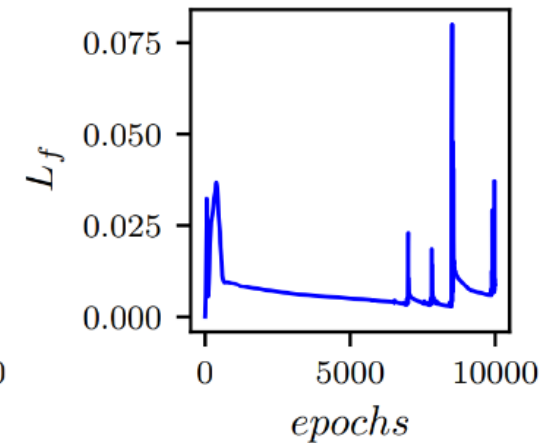
<Kim et al., DPM: A Novel Training Method for Physics-Informed Neural Networks
in Extrapolation, AAAI, 2021>

Motivations

- L_u converges fast.
- L_f fluctuates and does not decrease below a certain value.
 - Recall that governing equations include frequently highly non-linear operators.
- Learning governing equations correctly is a key of PINNs, but what we observed shows its difficulty.



(a) L_u curve

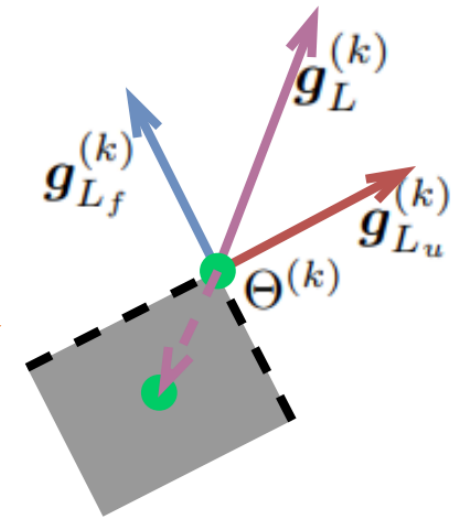


(b) L_f curve

Proposed method

- We dynamically modify the gradient to ensure a decrease of L_f if $L_f > \epsilon$.
- The gradient $g^{(k)}$ at iteration k is defined as follows:

$$g^{(k)} = \begin{cases} g_{L_u}^{(k)}, & \text{if } L_f \leq \epsilon \\ g_L^{(k)}, & \text{if } L_f > \epsilon \wedge g_{L_u}^{(k)} \cdot g_{L_f}^{(k)} \geq 0 \\ v^* + g_L^{(k)}, & \text{otherwise} \end{cases}$$



(c) Updating Θ

- The optimal gradient manipulation is analytically solved as follows:

$$v^* = \frac{-g_L^{(k)} \cdot g_{L_f}^{(k)} + \delta}{\|g_{L_f}^{(k)}\|_2^2}.$$

Solving Parameterized PDEs with Meta Learning

<Cho et al., Hypernetwork-based Meta-Learning for Low-Rank Physics-Informed
Neural Networks, NeurIPS, 2023>

Motivations

- We need to solve many similar problems in real-world applications.

$$u_t + \beta u_x - \nu u_{xx} - \rho u(1 - u) = 0, \quad x \in \Omega, \quad t \in [0, T]$$

- In the above convection-diffusion-reaction equation, some parameter settings are trivial to solve with PINNs whereas others are not.
- We need an adaptive-rank framework to solve parameterized PDEs.
 - Use simple (resp. complicated) DNNs for easy (resp. difficult) cases.

$$\mathbf{h}^{l+1} = \text{LR-FC}^l(\mathbf{h}^l) \quad \Leftrightarrow \quad \mathbf{h}^{l+1} = U_r^l(\Sigma_r^l(V_r^{l\top} \mathbf{h}^l)) + \mathbf{b}^l$$

Proposed Method

$$\begin{aligned}
 \mathbf{h}^1 &= \sigma(W^0 \mathbf{h}^0 + \mathbf{b}^0), \\
 \mathbf{h}^{l+1} &= \sigma(U^l(\Sigma^l(\boldsymbol{\mu})(V^{l\top} \mathbf{h}^l)) + \mathbf{b}^l), l = 1, \dots, L, \\
 u_{\Theta}((\mathbf{x}, t); \boldsymbol{\mu}) &= \sigma(W^{L+1} \mathbf{h}^{L+1} + \mathbf{b}^{L+1}), \text{ where } \Sigma^l(\boldsymbol{\mu}) = \text{diag}(\mathbf{s}^l(\boldsymbol{\mu})).
 \end{aligned}$$

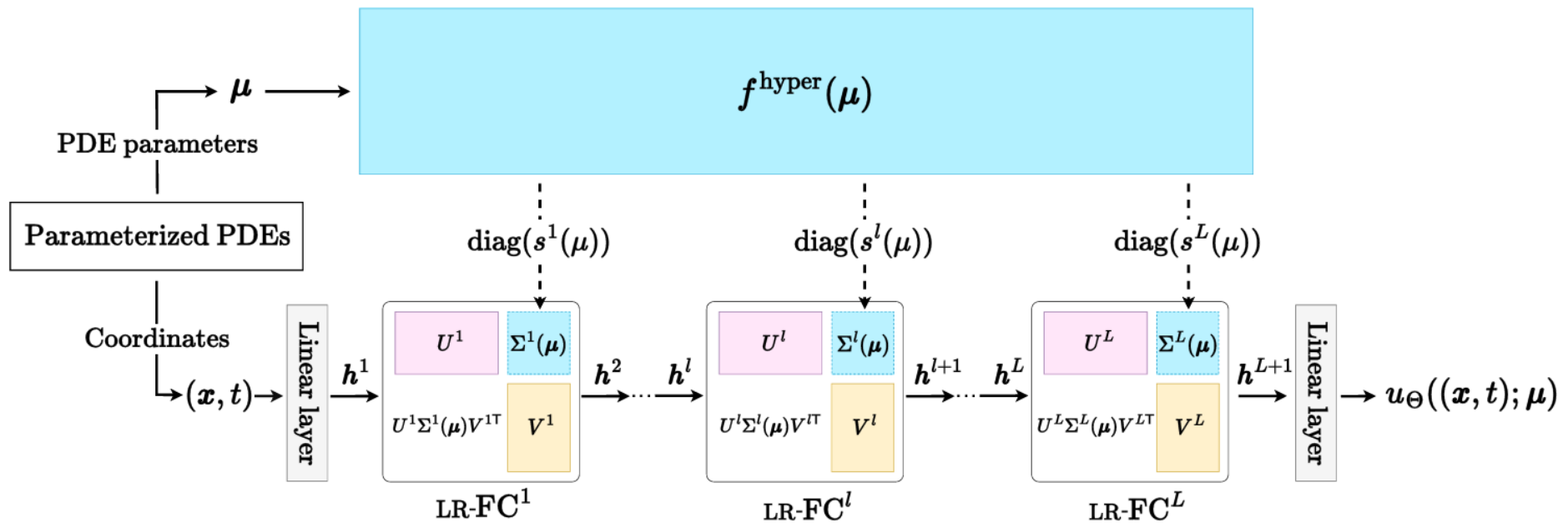
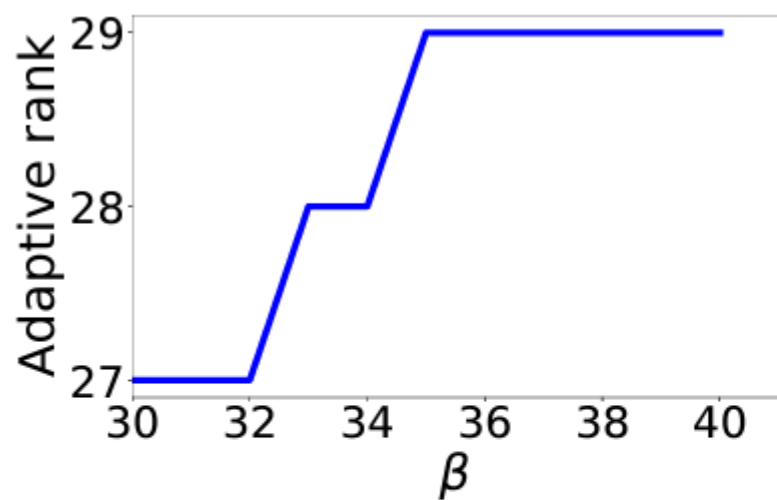
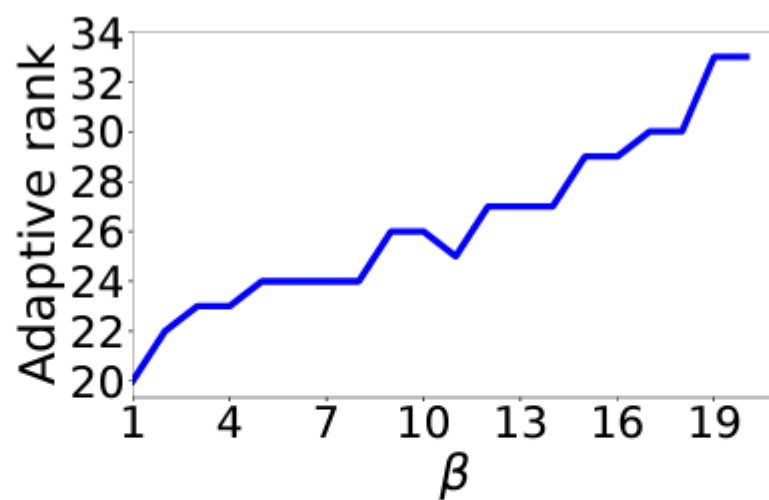


Figure 1: The architecture of Hyper-LR-PINN consisting of i) the hypernetwork generating model parameters (i.e., diagonal elements) of LR-PINN and ii) LR-PINN approximating solutions.

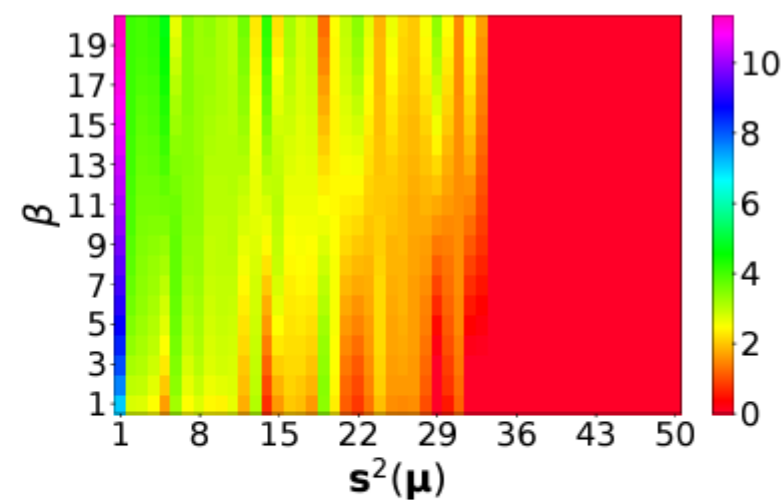
Adaptive rank



(a) $\beta \in [30, 40]$



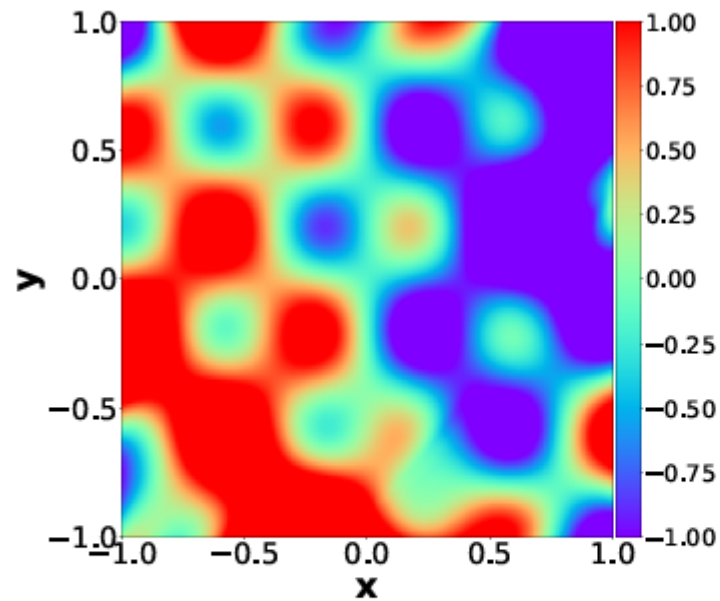
(b) $\beta \in [1, 20]$



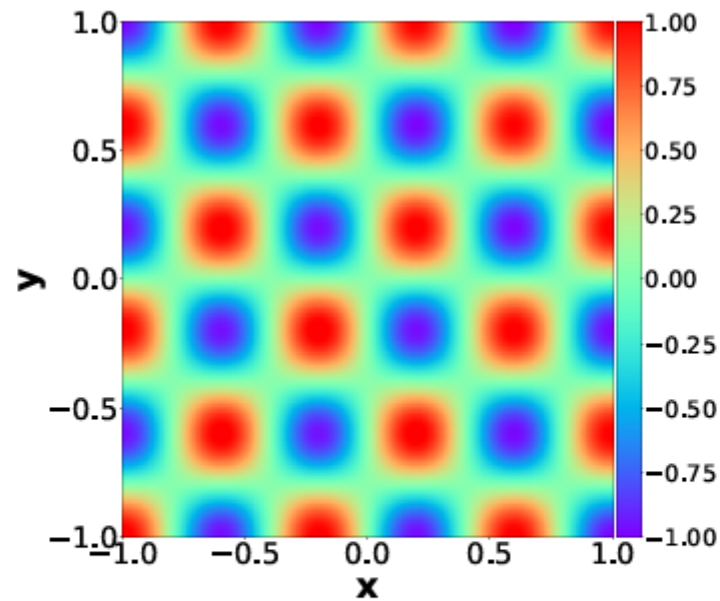
(c) $\beta \in [1, 20]$

Figure 4: Adaptive rank on convection equation (the left and the middle panels). The magnitude of the learned diagonal elements s^2 of the second hidden layer for varying $\beta \in [1, 20]$ (the right panel).

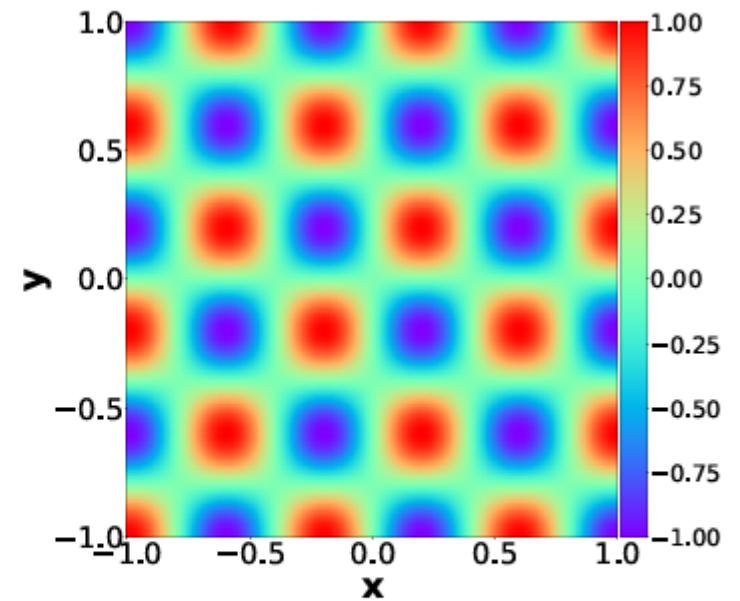
Visualization of solutions



(a) PINN (Abs.err.=0.7403)



(b) Ours (Abs.err.=0.0285)



(c) Exact solution

Figure 7: [2D-Helmholtz equation] Solution snapshots for $a = 2.5$

Conclusion

- Deep learning based on differential equations provide us novel ways to design and analyze neural networks.
 - GCNs can be enhanced by adopting more complicated hidden dynamics inspired by physical phenomena.
 - NCDEs are generalized forms of various recurrent models and can be extended to the spatiotemporal processing.
 - The current Transformer paradigm, which scales well up to very large models, costs a lot. We need an alternative paradigm.
- Deep learning can also be used for solving PDEs.
 - However, we do not know governing equations in many cases.