北京国际数学研究中心
BEIJING INTERNATIONAL CENTER FOR MATHEMATICAL RESEARCH

北京大学人工智能研究院
Academy for Artificial Intelligence, Peking University

BIBDR 北京大数据研究院

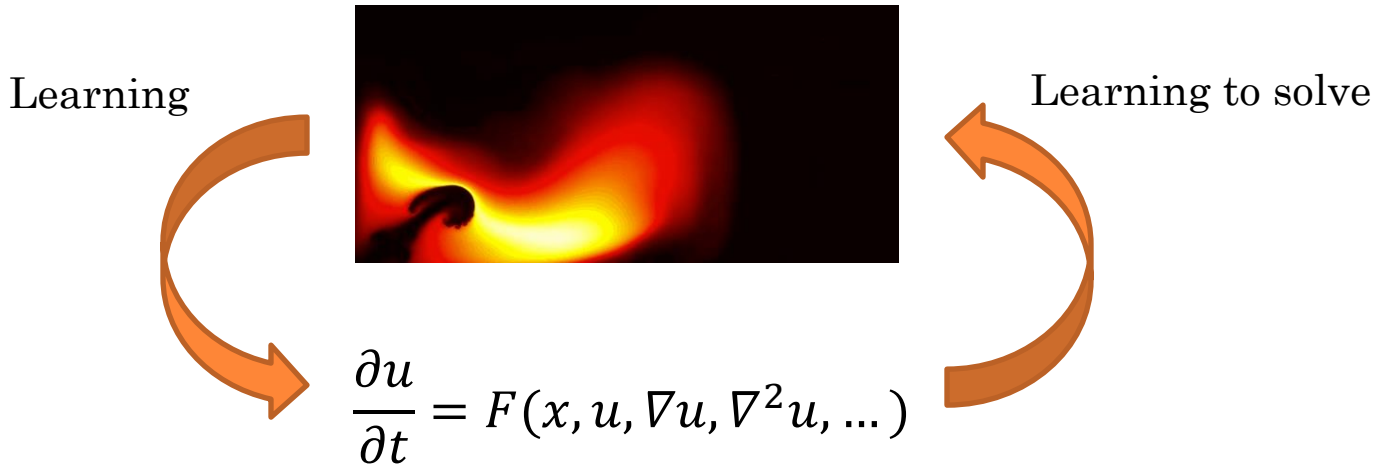
Learning and Learning to Solve PDEs

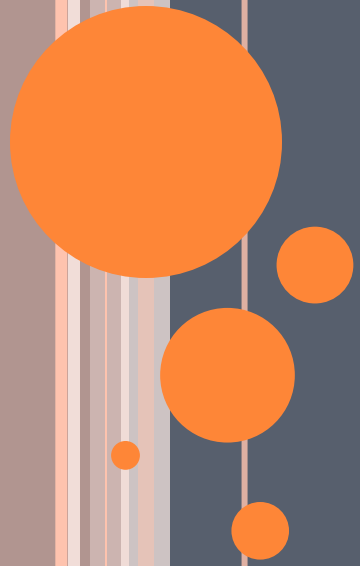
Bin Dong (董彬)

- Beijing International Center for Mathematical Research, Peking University
- Center for Theory of Artificial Intelligence, Institute for Artificial Intelligence, Peking University
- Laboratory for Biomedical Image Analysis & Laboratory of Deep Learning Research, Beijing Institute of Big Data Research

Outline

- Overview and Motivations
- Deep Neural Networks (DNNs) and PDEs
  - Learning PDEs: PDE-Net for Inverse Problems
  - Learning to Solve PDEs: A Reinforcement Learning Framework
  - Learning to Solve Parameterized PDEs: A Meta-Learning Approach





Overview

Motivations and Intuitions

Deep Learning from Mathematics Perspective

- Deep learning has been a great success.
- However, it is also in lack of
  - Theoretical guidance
  - Interpretability and robustness
- Our perspective: control

$$\tilde{f}_{L,N}(\boldsymbol{x};\boldsymbol{\Theta}) : \mathbb{R}^n \mapsto \mathbb{R},$$

can be recursively defined as: $\boldsymbol{\Theta}^\ell = (\boldsymbol{\Theta}^{\ell-1}, \theta^\ell)$, $\tilde{f}_{\boldsymbol{\Theta}^\ell} = (\theta^\ell \circ \sigma \circ \tilde{f}_{\boldsymbol{\Theta}^{\ell-1}})$, $\theta^\ell : \mathbb{R}^{N_\ell} \to \mathbb{R}^{N_{\ell+1}}$ with $\theta^\ell(\boldsymbol{x}) = \boldsymbol{W}^\ell \boldsymbol{x} + \boldsymbol{b}^\ell$, and $\tilde{f}_{L,N} := \tilde{f}_{\boldsymbol{\Theta}^L}$.

Composite structures can be viewed as dynamics.

Deep Learning from Control Perspective

- Control Perspective: Supervised Learning (SL)、Reinforcement Learning (RL)、Meta Learning (Meta)

Training loss: $J(\boldsymbol{u}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\boldsymbol{P}}\ \ell(g(\boldsymbol{X}_1), y) + \int_0^1 R\big(s, \boldsymbol{X}_s, \boldsymbol{u}(s)\big)\, ds$

Dynamics:

$$d\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t, \boldsymbol{u}(t, \boldsymbol{X}_t, \boldsymbol{\theta}_t))dt + \boldsymbol{\sigma}(\boldsymbol{X}_t, t)d\boldsymbol{W}_t, \quad \boldsymbol{X}_0 = \boldsymbol{x}, \quad t \in (0,1]$$

$$\boldsymbol{u}(t, \boldsymbol{X}_t, \boldsymbol{\theta}_t) = \begin{cases} \boldsymbol{u}(t) & \textit{Typical SL-control} \\ \boldsymbol{u}(\boldsymbol{X}_t, \boldsymbol{\theta}) & \textit{Typical RL-control} \\ \boldsymbol{u}(\boldsymbol{\theta}_t) & \textit{Typical Meta-control} \end{cases}$$

Other variants:

- RL-control: $\boldsymbol{u}(\boldsymbol{X}_t, \boldsymbol{\theta}_t)$
- Meta-control: $\boldsymbol{u}(\boldsymbol{\theta}_t, \boldsymbol{Z}_t)$ with $\boldsymbol{Z_t} = \boldsymbol{SDE}(\boldsymbol{Z_t}, \boldsymbol{X_t})$

Solver:

- *BP/PMP*
- *DP*

Common: ——

Uncommon: ·····

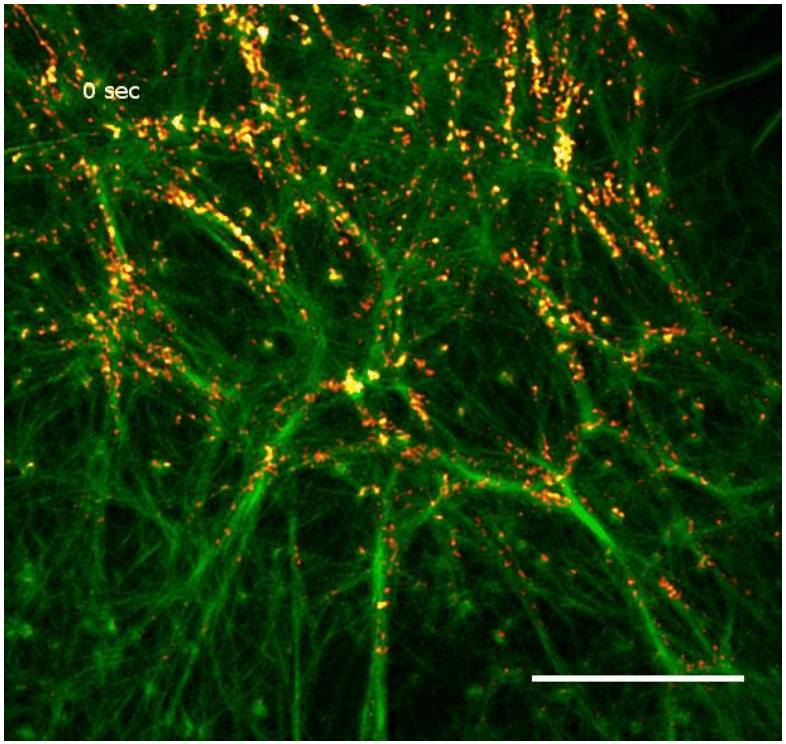
- E, CMS, 5(1):1–11, 2017.
- Li et al., JMLR, 18(1), 2017.
- Haber, Ruthotto, IP, 34(1), 2017.

PDE-Net: Learning PDEs from Data

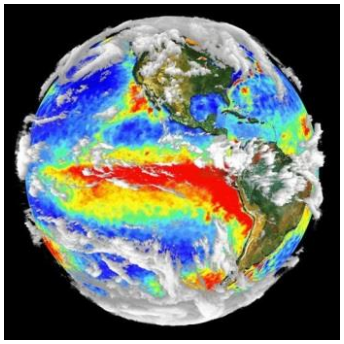
- Zichao Long, Yiping Lu, Xianzhong Ma and Bin Dong, *PDE-Net: Learning PDEs from Data*, ICML 2018. (arXiv:1710.09668)
- Zichao Long, Yiping Lu and Bin Dong, *PDE-Net 2.0: Learning PDEs from Data with A Numeric-Symbolic Hybrid Deep Network*, Journal of Computational Physics, 399, 108925, 2019 (arXiv:1812.04426).

PDE-Net: Learning PDEs from Data

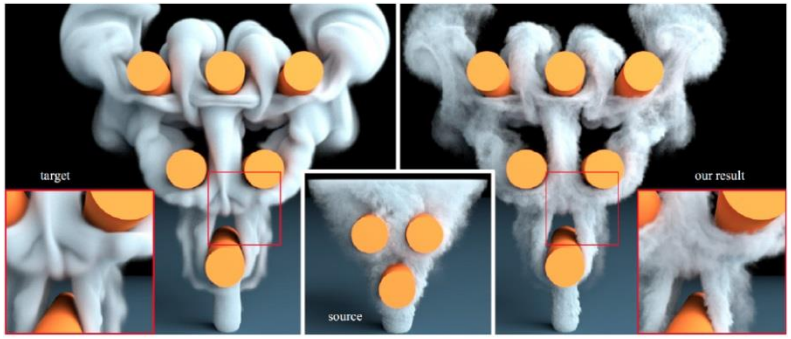
- Can we learn principles (e.g. PDEs) from data?



Biology



Meteorology



Computer Graphics

PDE-Net: Learning PDEs from Data

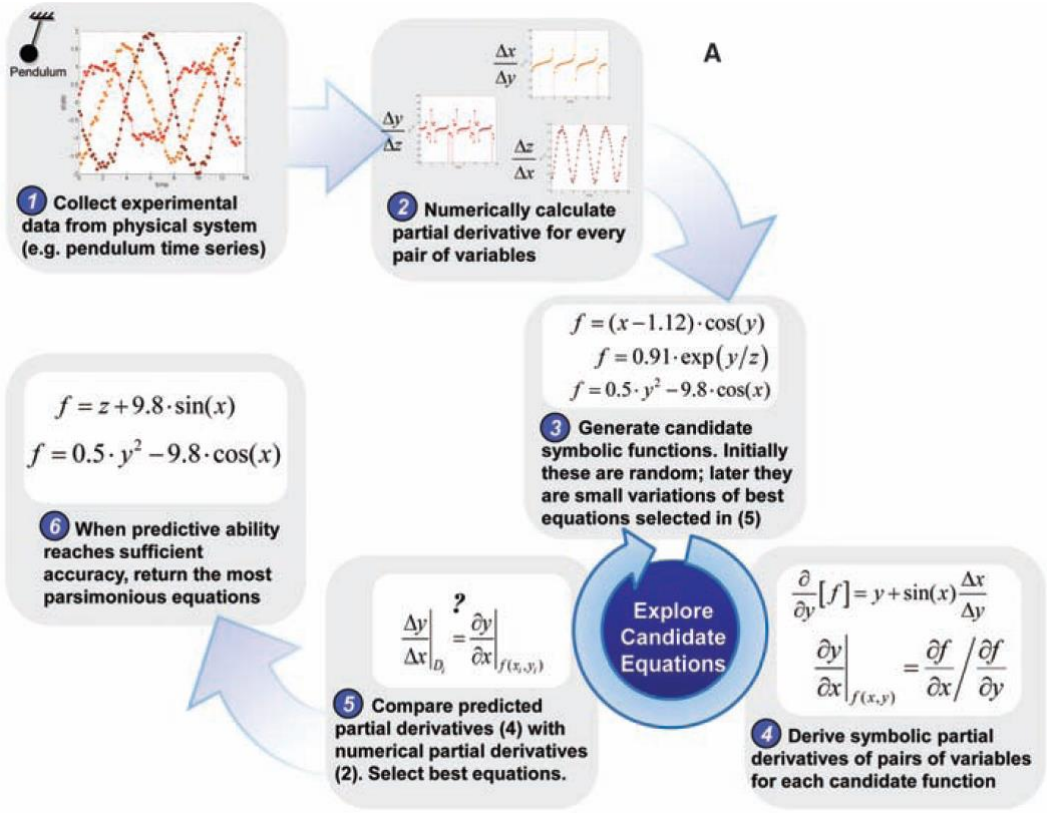
- Earlier work

REPORT

Distilling Free-Form Natural Laws from Experimental Data

Michael Schmidt[1], Hod Lipson[2,3,*]
+ See all authors and affiliations

Science 03 Apr 2009:
Vol. 324, Issue 5923, pp. 81-85
DOI: 10.1126/science.1165893



Other earlier attempts:

- Bongard & Lipson, PNAS, 2007
- Lin, Zhang & Tang, MSR-TR-2008-189.
- Liu, Lin, Zhang & Su. ECCV 2010.

PDE-Net: Learning PDEs from Data

- Earlier work
  - Dictionary based sparse regression
    - Construct dictionary

$$\Theta(U) = [1 \quad U \quad U^2 \quad \cdots U_x \quad UU_x \quad \cdots U_x^2]$$

    - Fit variable ξ

$$U_t = \Theta(U)\xi$$

    - Sparse regression

$$\min_{\xi} ||\Theta\xi - U_t||_2^2 + \lambda||\xi||_0$$

- S. Brunton, J. L. Proctor and J. N. Kutz Proceedings of the National Academy of Sciences, 2016
- Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Science Advances, 3(4), 2017.
- Hayden Schaeffer. Proc. R. Soc. A, volume 473, The Royal Society, 2017.

PDE-Net: Learning PDEs from Data

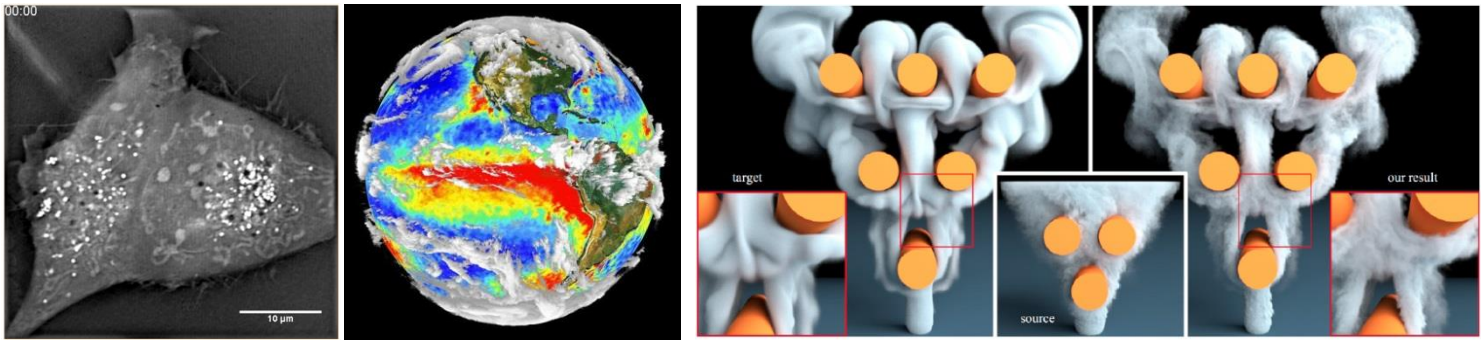
- Room for improvements
  - Can we go beyond sparse coding framework (linear dictionary)?

    —— Bigger model class with less prior knowledge
  - Can we learn discrete forms of differential operators and does it help?

    ——More accurate estimation of the PDE and prediction

PDE-Net: Learning PDEs from Data

- Can we learn principles (e.g. PDEs) from data?



S. Sato et al., Siggraph 2018

- Initial attempt:
  - Combining deep learning and numerical PDEs
- Objectives:
  - Predictive and expressive power (deep learning)
  - Transparency: to reveal hidden physics (numerical PDEs)

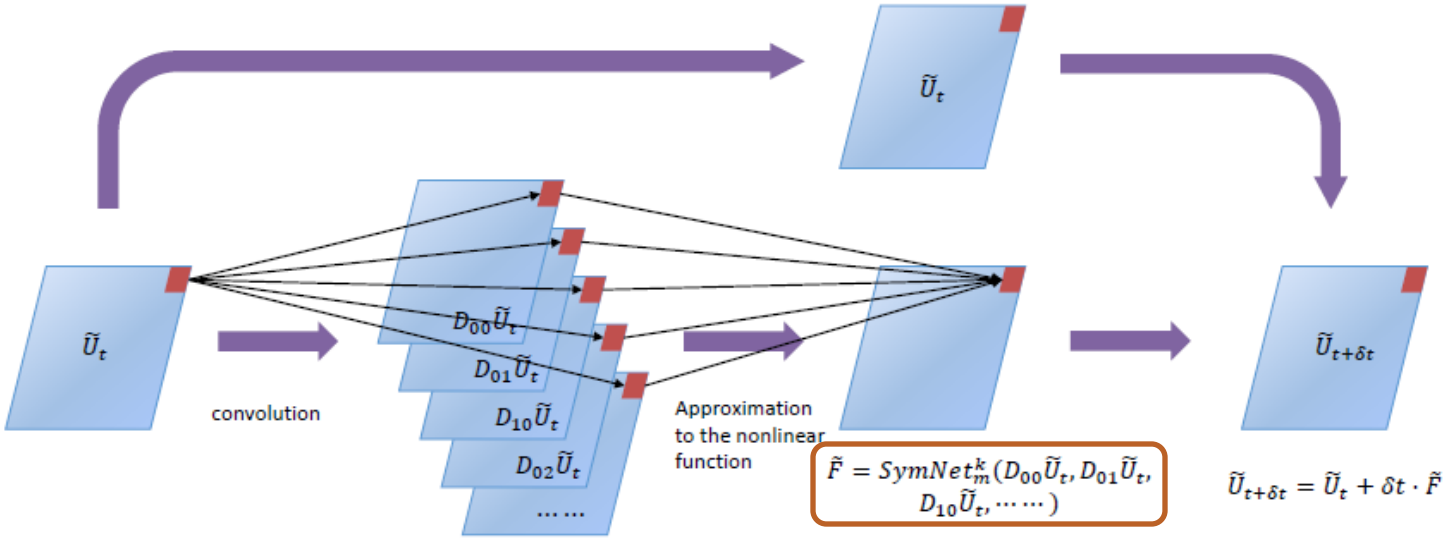
PDE-Net: Learning PDEs from Data

- PDE-Net 2.0

Assuming: $\dfrac{\partial u}{\partial t} = F\left(u, \nabla u, \nabla^2 u, \dots\right), \qquad u \in \mathbb{R}^d$

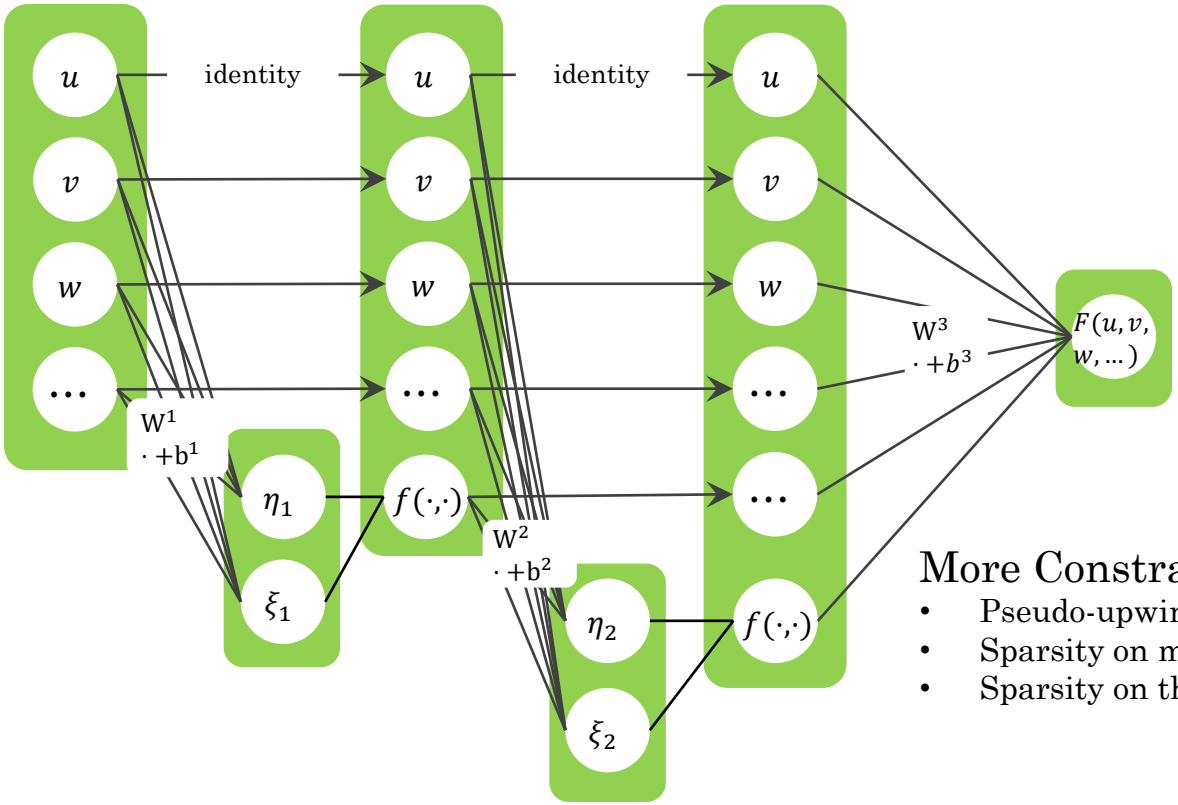
Prior knowledge on F:

- Addition and multiplication of the terms;
- Maximum order.



PDE-Net: Learning PDEs from Data

- PDE-Net 2.0



More Constraints:

- Pseudo-upwind
- Sparsity on moment matrices
- Sparsity on the symbolic network

Similar to $EQL/EQL^{\div}$: Sahoo, Lampert, and Martius, ICML 2018.

PDE-Net: Learning PDEs from Data

- Constraints on kernels (granting transparency)
  - Moment matrix

$$M(q) = (m_{i,j})_{N\times N}, \text{ where } m_{i,j} = \frac{1}{(i-1)!(j-1)!}\sum_{k\in\mathbb{Z}^2} k_1^{i-1}k_2^{j-1}q[k_1,k_2]$$

  - We can approximate any differential operator at any prescribed order by constraining $M(q)$
  - For example: approximation of $\frac{\partial f}{\partial x}$ with a 3×3 kernel

$$\begin{pmatrix} 0 & 0 & \star \\ 1 & \star & \star \\ \star & \star & \star \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & \star \\ 0 & \star & \star \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

1st order learnable | 2st order learnable | 1st order frozen

- **J.F. Cai, B. Dong, S. Osher and Z. Shen, Journal of the American Mathematical Society, 2012.**
- **B. Dong, Q. Jiang and Z. Shen, Multiscale Modeling & Simulation, 2017**

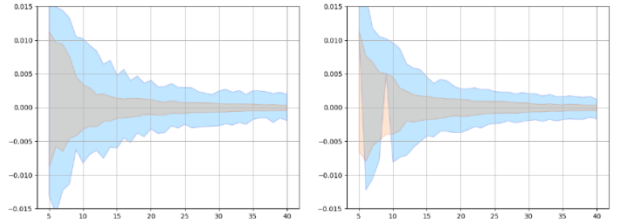
PDE-Net: Learning PDEs from Data

- Example: Burger's equation

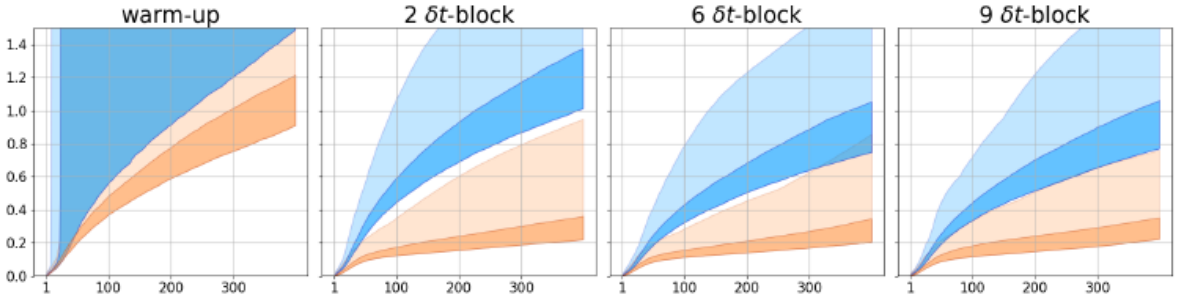
$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\, \boldsymbol{u} = \nu \nabla^2 \boldsymbol{u}$$
$$\nu = 0.05$$

Correct PDE	$u_t = -uu_x - vu_y + 0.05(u_{xx} + u_{yy})$ $v_t = -uv_x - vv_y + 0.05(v_{xx} + v_{yy})$
Frozen-PDE-Net 2.0	$u_t = -0.906uu_x - 0.901vu_y + 0.033u_{xx} + 0.037u_{yy}$ $v_t = -0.907vv_y - 0.902uv_x + 0.039v_{xx} + 0.032v_{yy}$
PDE-Net 2.0	$u_t = -0.979uu_x - 0.973u_y v + 0.052u_{xx} + 0.051u_{yy}$ $v_t = -0.973uv_x - 0.977vv_y + 0.053v_{xx} + 0.051v_{yy}$

Model recovery



Remainer weights of $\boldsymbol{u}, \boldsymbol{v}$



Prediction

PDE-Net: Learning PDEs from Data

- Example: Burger's + reaction

$$
\begin{aligned}
u_t &= -uu_x - vu_y + \nu\Delta u + \lambda(A)u - \omega(A)v \\
v_t &= -uv_x - vv_y + \nu\Delta v + \omega(A)u + \lambda(A)v \\
A^2 &= u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2
\end{aligned}
$$

Correct PDE	$u_t = -uu_x - vu_y + 0.1\Delta u + (v-u)(u^2+v^2) + u$ $v_t = -uv_x - vv_y + 0.1\Delta v - (v+u)(u^2+v^2) + v$
Frozen-PDE-Net 2.0	$u_t = -0.86uu_x - 0.90vu_y + 0.09u_{xx} + 0.09u_{yy}$ $+1.02u^2v - 1.02u^3 - 1.01uv^2 + 1.01u + 0.99v^3$ $v_t = -0.87uv_x - 0.85vv_y + 0.09v_{xx} + 0.09v_{yy}$ $+1.04u^2v - 1.02uv^2 - 1.01v^3 + 0.99v - 0.99u^3$
PDE-Net 2.0	$u_t = -0.98vu_y - 0.93uu_x + 0.10u_{xx} + 0.10u_{yy}$ $-1.05uv^2 + 0.99v^3 - 0.98u^3 + 0.98u + 0.97u^2v$ $v_t = -0.99uv_x - 0.96vv_y + 0.10v_{yy} + 0.10v_{xx}$ $-1.04u^2v - 1.02v^2 - 1.02uv^2 + 1.01v - 1.00u^3$

Model Recovery

Learning to Solve Conservation Laws via Reinforcement Learning

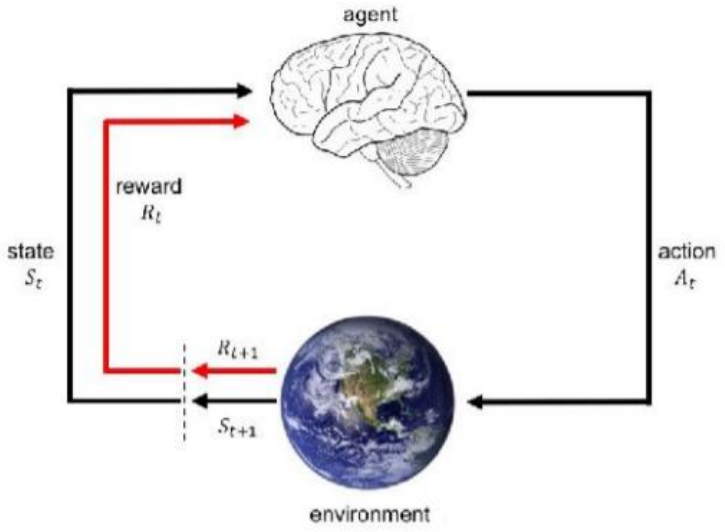
- Yufei Wang, Ziju Shen, Zichao Long and Bin Dong, *Learning to Discretize: Solving 1D Scalar Conservation Laws via Deep Reinforcement Learning*, accepted by CiCP 2020 (arXiv: 1905.11079).

Neural Networks (NNs) and Numerical PDEs – A Highly incomplete List

- NNs as a new ansatz:
  - C. Beck, W. E, A. Jentzen. JNS, 1–57, 2017.
  - W. E and B. Yu, CMS, 6(1), 1-12, 2018.
  - J. Han, A. Jentzen, W. E. PNAS, 115(34):8505–8510, 2018.
  - M. Raissi, P. Perdikaris, G. E. Karniadakis. JCP, 378:686–707, 2019.
  - G.-J. Both, S. Choudhury, P. Sens, R. Kusters. arXiv:1904.09406, 2019.
  - C. Michoski, M. Milosavljevic, T. Oliver, D. Hatch. arXiv:1905.04351, 2019.
  - Y. Zhang, G. Bao, X. Ye, H. Zhou, arXiv:1907.08272.
  - D. Pfau, J.S. Spencer, A.G. Matthews, W. M. Foulkes, arXiv:1909.02487.
  - W. Cai and Z. Xu, arXiv:1910.11710
  - Z. Liu, W. Cai, Z.Q.J. Xu, arXiv:2007.11207.
- NNs integrated with classical solvers
  - D. Ray, J. S Hesthaven. JCP, 367:166–191, 2018.
  - J. Magiera, D. Ray, J. S Hesthaven, C. Rohde., JCP, 409, p.109345, 2020.
  - N. Discacciati, J. S. Hesthaven, D. Ray. JCP, p. 109304, 2020.
  - Y. Feng, T. Liu, K. Wang, JSC, 83(21), 2020.
- NNs to approximate complex solution mappings
  - Y. Khoo, J. Lu, L. Ying. arXiv:1707.03351, 2017.
  - Y. Khoo, L. Ying. SISC, 41(5): A3182-A3201, 2019.
  - Y. Fan, L. Lin, L. Ying, L. Zepeda-Núnez. MMS, 17(4):1189-213, 2019.
  - Y. Li, J. Lu, A. Mao. JCP, 409, p.109338, 2020.

A Brief Intro to Reinforcement Learning

- Reinforcement learning (RL)



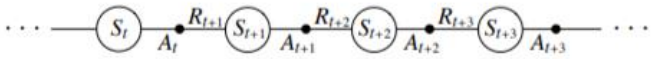
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

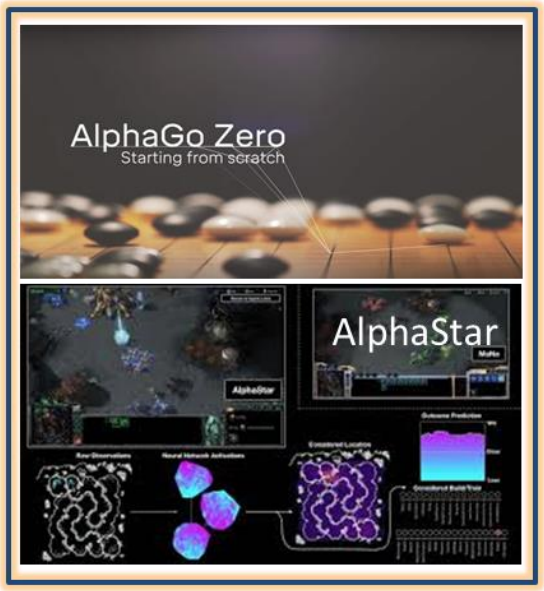
produces action at step t: $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$



- RL is to learn to make **sequential decisions** by interacting with the environments (learn from rewards).
- Can be modeled as a finite Markov Decision Process (MDP):



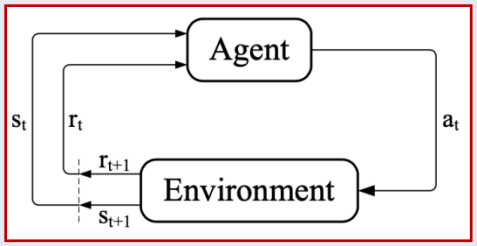
A Brief Intro to Reinforcement Learning

- Markov Decision Process

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.



A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

From David Silver's tutorial on RL

A Brief Intro to Reinforcement Learning

- Goal: $\max_{\pi} \mathbb{E}^{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)} \sum_{t \geq 0} \gamma^t R^{s_t}_{a_t}$
- Algorithms
  - Q-Learning (DQN, DDQN)
  - Policy Gradient (PG, PPO, DDPG, etc.)

Deep Q-Learning

The function to approximate is a Q-function that satisfies the Bellman equation:

Q(s,a,Θ) ≈ Q*(s,a)

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Forward Pass

Sample a (s,a) pair — Predicted Q-value with Θ_i

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s, a\right]$$

Sample a future state s' — Predict Q-value with Θ_{i-1}

Backward Pass

Gradient update (with respect to Q-function parameters Θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)) \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Slide concept: Serena Yeung, "Deep Reinforcement Learning", Stanford University CS231n, 2017.

— **Pros:** can use off-policy data
— **Cons:** cannot handle continuous/stochastic actions

Policy Gradients

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic} \\
G_\theta^{-1} \nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}$$

Each leads a stochastic gradient ascent algorithm

Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$

From David Silver's tutorial on RL

— **Pros:** works on continuous/stochastic actions
— **Cons:** high variance, can only use on-policy data

Learning to Discretize

- 1D scalar conservation laws (CLs):

$$u_t(x,t) + f_x(u(x,t)) = 0, \quad a \le x \le b, \ t \in [0,T], \quad u(x,0) = u_0(x).$$

- Discrete setting: $u_j^n = u(x_j, t_n)$, U_j^n is the approximated solution, $f_j^n = f(u_j^n)$, and $\hat{f}_j^n$ is the approximated flux on the grids

$$x_j = a + j\Delta x, \ t_n = n\Delta t \quad \text{with } j = 0,1,...,J = \frac{b-a}{\Delta x}, \ n = 0,1,...,N = \frac{T}{\Delta t}.$$

- WENO schemes

Algorithm 1: A Conservation Law Solving Procedure

Input: initial values $u_0^0, u_1^0, ..., u_J^0$, flux $f(u)$, Δx, Δt, evolve time N, left shift r and right shift s.
Output: $\{U_j^n | \ j = 0,...,J, \ n = 1,...,N\}$
$U_j^0 = u_j^0, \ j = 0,...,J$
for $n = 1$ **to** N **do**
  for $j = 0$ **to** J **do**
    Compute the numerical flux $\hat{f}_{j-\frac{1}{2}}^n = \pi^f(U_{j-r-1}^{n-1}, U_{j-r+1}^{n-1}, ..., U_{j+s-1}^{n-1})$ and $\hat{f}_{j+\frac{1}{2}}^n = \pi^f(U_{j-r}^{n-1}, U_{j-r+1}^{n-1}, ..., U_{j+s}^{n-1})$, e.g., using the WENO scheme
    Compute $\frac{du_j(t)}{dt} = -\frac{1}{\Delta x}(\hat{f}_{j+\frac{1}{2}}^n - \hat{f}_{j-\frac{1}{2}}^n)$
    Compute $U_j^n = \pi^t(U_j^{n-1}, \frac{du_j(t)}{dt})$, e.g., using the Euler scheme $U_j^n = U_j^{n-1} + \Delta t \frac{du_j(t)}{dt}$
Return $\{U_j^n | \ j = 0,...,J, \ n = 1,...,N\}$

$$\hat{f}_{j+\frac{1}{2}} = \sum_{r=-2}^{1} w_r \hat{f}_{j+1/2}^r, \quad \sum_{r=-2}^{1} w_r = 1.$$

Roe speed:

$$\bar{a}_{j+\frac{1}{2}} = \frac{f_{j+\frac{1}{2}} - f_{j-\frac{1}{2}}}{u_{j+\frac{1}{2}} - u_{j-\frac{1}{2}}}$$

Learning to Discretize

- RL-WENO

Algorithm 1: A Conservation Law Solving Procedure

Input: initial values $u_0^0, u_1^0, ..., u_J^0$, flux $f(u)$, Δx, Δt, evolve time N, left shift r and right shift s.
Output: $\{U_j^n | \ j = 0, ..., J, \ n = 1, ..., N\}$
$U_j^0 = u_j^0, \ j = 0, ..., J$
for $n = 1$ **to** N **do**
  for $j = 0$ **to** J **do**
    Compute the numerical flux $\hat{f}^n_{j-\frac{1}{2}} = \pi^f(U^{n-1}_{j-r-1}, U^{n-1}_{j-r+1}, ..., U^{n-1}_{j+s-1})$ and $\hat{f}^n_{j+\frac{1}{2}} = \pi^f(U^{n-1}_{j-r}, U^{n-1}_{j-r+1}, ..., U^{n-1}_{j+s})$, e.g., using the WENO scheme
    Compute $\frac{du_j(t)}{dt} = -\frac{1}{\Delta x}(\hat{f}^n_{j+\frac{1}{2}} - \hat{f}^n_{j-\frac{1}{2}})$
    Compute $U_j^n = \pi^t(U_j^{n-1}, \frac{du_j(t)}{dt})$, e.g., using the Euler scheme $U_j^n = U_j^{n-1} + \Delta t \frac{du_j(t)}{dt}$
Return $\{U_j^n | \ j = 0, ..., J, \ n = 1, ..., N\}$

State: $s_j = \left\{ f\left(U^{n-1}_{j-r-1}\right), ..., f\left(U^{n-1}_{j+s}\right), \bar{a}_{j\pm\frac{1}{2}} \right\}$

State transition

Action: $\pi^{RL}(s_j) = \left(w^{-2}_{j-\frac{1}{2}}, w^{-1}_{j-\frac{1}{2}}, w^{0}_{j-\frac{1}{2}}, w^{1}_{j-\frac{1}{2}}, w^{-2}_{j+\frac{1}{2}}, w^{-1}_{j+\frac{1}{2}}, w^{0}_{j+\frac{1}{2}}, w^{1}_{j+\frac{1}{2}}\right)$

$$\hat{f}_{j-\frac{1}{2}} = \sum_{i=-2}^{1} w^i_{j-\frac{1}{2}} \hat{f}^i_{j-\frac{1}{2}}, \quad \hat{f}_{j+\frac{1}{2}} = \sum_{i=-2}^{1} w^i_{j+\frac{1}{2}} \hat{f}^i_{j+\frac{1}{2}}$$

Learning to Discretize

- RL-WENO

Algorithm 2: General RL Running Procedure

Input: initial values $u_0^0, ..., u_J^0$, flux $f(u)$, Δx, Δt, evolve time N, left shift r, right shift s and RL policy π^{RL}
Output: $\{U_j^n | \ j = 0, ..., J, \ n = 1, ..., N\}$
$U_j^0 = u_j^0, \ j = 0, ..., J$
for *Many iterations* **do**
  Construct initial states $s_j^0 = g_s(U_{j-r-1}^0, ..., U_{j+s}^0)$ for $j = 0, ..., J$
  for $n = 1$ **to** N **do**
    for $j = 0$ **to** J **do**
      Compute the action $a_j^n = \pi^{RL}(s_j^n)$ that determines how $\hat{f}_{j+\frac{1}{2}}^n$ and $\hat{f}_{j-\frac{1}{2}}^n$ is computed
      Compute $\frac{du_j(t)}{dt} = -\frac{1}{\Delta x}(\hat{f}_{j+\frac{1}{2}}^n - \hat{f}_{j-\frac{1}{2}}^n)$
      Compute $U_j^n = \pi^t(U_j^{n-1}, \frac{du_j(t)}{dt})$, e.g., the Euler scheme $U_j^n = U_j^{n-1} + \Delta t \frac{du_j(t)}{dt}$
      Compute the reward $r_j^n = g_r(U_{j-r-1}^n - u_{j-r-1}^n, \cdots, U_{j+s}^n - u_{j+s}^n)$.
    Construct the next states $s_j^{n+1} = g_s(u_{j-r-1}^n, ..., u_{j+s}^n)$ for $j = 0, ..., J$
    Use any RL algorithm to train the RL policy π^{RL} with the transitions $\{(s_j^n, a_j^n, r_j^n, s_j^{n+1})\}_{j=0}^J$.
Return the well-trained RL policy π^{RL}.

$g_r(\cdot) = \|\cdot\|_\infty$

Trained by: $f = \frac{1}{2}u^2$ & WENO-5 & Euler & $(\Delta x, \Delta t) = (0.04, 0.002)$ & $T = 0.8$

$$\frac{dJ(\pi_\theta)}{d\theta} = E_{s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi_\theta(\cdot|s_t)}[\nabla_\theta \log \pi_\theta(s_t, a_t) Q^{\pi_\theta}(s_t, a_t)]$$

Learning to Discretize

- Experimental settings
  - Burger's equation:

$$u(x,t)_t + (\frac{1}{2}u(x,t)^2)_x = \eta u(x,t)_{xx} + F(x,t), \quad u(x,0) = u_0(x), \ x \in D, \ t \in [0,T]$$

  - Three scenarios:
    - Inviscid

$$\eta = 0, \ F(x,t) = 0$$
$$u_0(x) = a + b \cdot \sin(c\pi x) + d \cdot \cos(e\pi x)$$
$$|a| + |b| + |d| = 4, \ |a| \le 1.2, \ |b| \le 3 - |a|, \ c \in \{4,6,8\}.$$

    - Forcing

$$\eta \in \{0.01, 0.02, 0.04\}, \ u_0(x) = 0$$
$$F(x,t) = \sum_{i=1}^{N} A_i \sin(\omega_i t + 2\pi l_i x / L + \psi_i)$$
$$N = 20, \ A_i \in [-0.5, 0.5], \ \omega_i \in [-0.4, 0.4], \ \phi_i \in [0, 2\pi], \ l_i \in \{3,4,5,6\}$$

    - Viscous: similar as inviscid except $\eta = \{0.01, 0.02, 0.04\}$.

Learning to Discretize

- Experimental settings
  - Compared methods:
    - 5th order WENO (X.-D. Liu, S. Osher, T. Chan, JCP 1994)
    - L3D (Y. Bar-Sinai et al., PNAS, 2019)
    - PINN (M. Raissi, P. Perdikaris, G. Karniadakis, JCP, 378:686–707, 2019)
  - General assessments:

Train \ Test	inviscid w/o forcing random initial value	viscous w/ forcing fixed initial value	viscous w/o forcing random initial value
inviscid w/o forcing random initial value	RL-WENO ✓(Table 1, 2) L3D ? PINNs ✓(Table 5)	RL-WENO ✓(Table 6) L3D ?	RL-WENO ✓(Table 7) L3D ?
viscous w/ forcing fixed initial value	RL-WENO ✓(Table 9) L3D ✗	RL-WENO ✓(Table 8) L3D ✓(Table 8)	RL-WENO ✓(Table 9) L3D ✗
viscous w/o forcing random initial value	RL-WENO ✓(Table 10) L3D ?	RL-WENO ✓(Table 11) L3D ?	RL-WENO ✓(Table 10) L3D ? PINNs ✓(Table 5)

  - RL-WENO is trained on “Inviscid” unless specified
  - Computation time:

$(\Delta x, \Delta t)$	RL-WENO	WENO	L3D	PINN
(0.02,0.002)	2.23	4.14	0.59	2148.34
(0.04,0.004)	0.87	1.09	0.51	2024.14
(0.05,0.005)	0.6	0.71	0.5	2058.66

Learning to Discretize

- RL-WENO v.s. WENO-5:
  - Overall accuracy:
    - Comparable with WENO-5

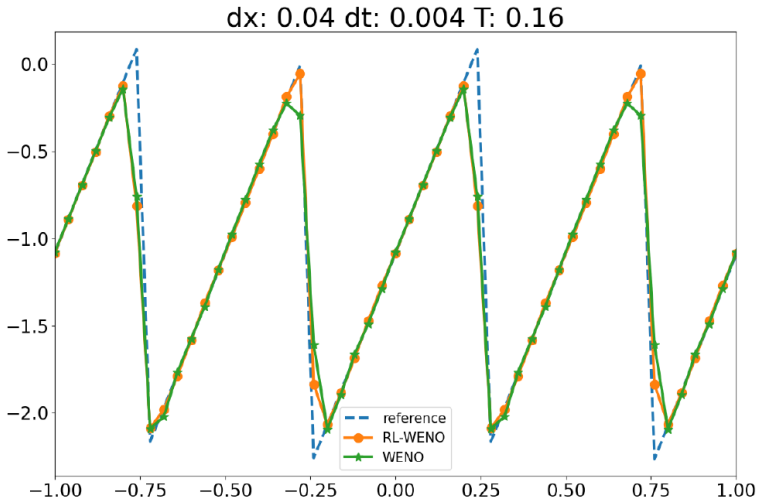
Training setting

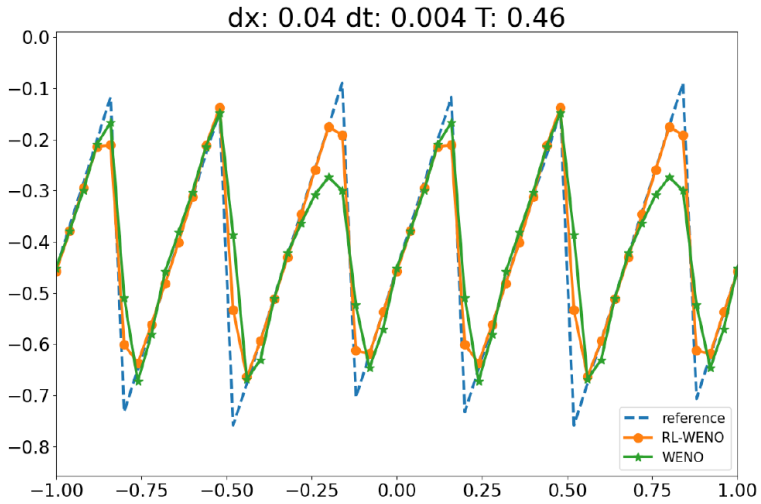
Δx / Δt	0.02		0.04		0.05	
	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO
0.002	10.81 (3.96)	11.13 (3.83)	18.79 (12.44)	19.45 (9.32)	28.61 (33.37)	35.95 (27.7)
0.003	10.83 (3.96)	11.14 (3.82)	18.82 (12.48)	19.47 (9.31)	28.62 (33.34)	35.96 (27.67)
0.004	10.83 (3.96)	11.14 (3.84)	18.89 (12.69)	19.48 (9.33)	28.61 (33.27)	35.93 (27.63)
0.005	10.85 (3.96)	11.15 (3.84)	18.96 (12.84)	19.52 (9.35)	28.48 (33.04)	35.93 (27.61)
0.006	10.89 (3.93)	11.16 (3.83)	18.95 (12.79)	19.51 (9.3)	28.58 (33.08)	35.89 (27.5)

$$f(u) = \frac{1}{2}u^2$$

Learning to Discretize

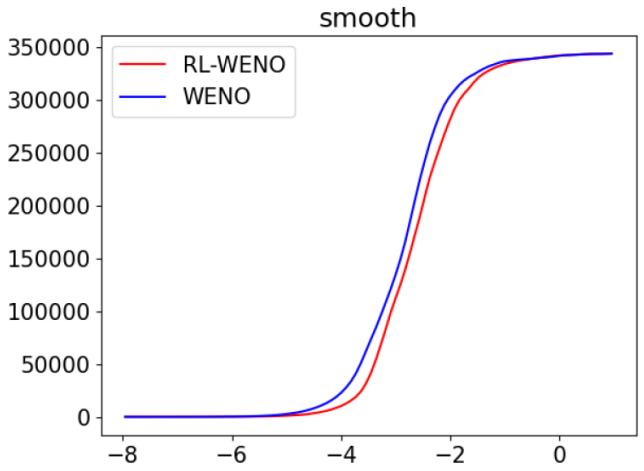
- RL-WENO v.s. WENO-5:
  - Near singularities (solution curves)



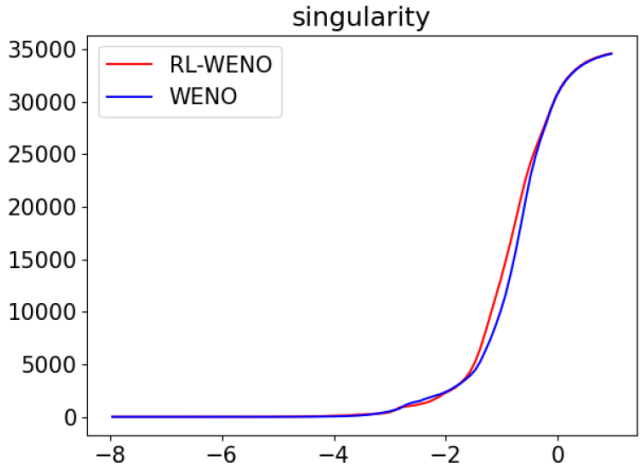


Learning to Discretize

- RL-WENO v.s. WENO-5:
  - Near singularities (accumulated errors)



(a) Smooth regions, $f(u) = \frac{1}{2}u^2$



(b) Near singularities, $f(u) = \frac{1}{2}u^2$

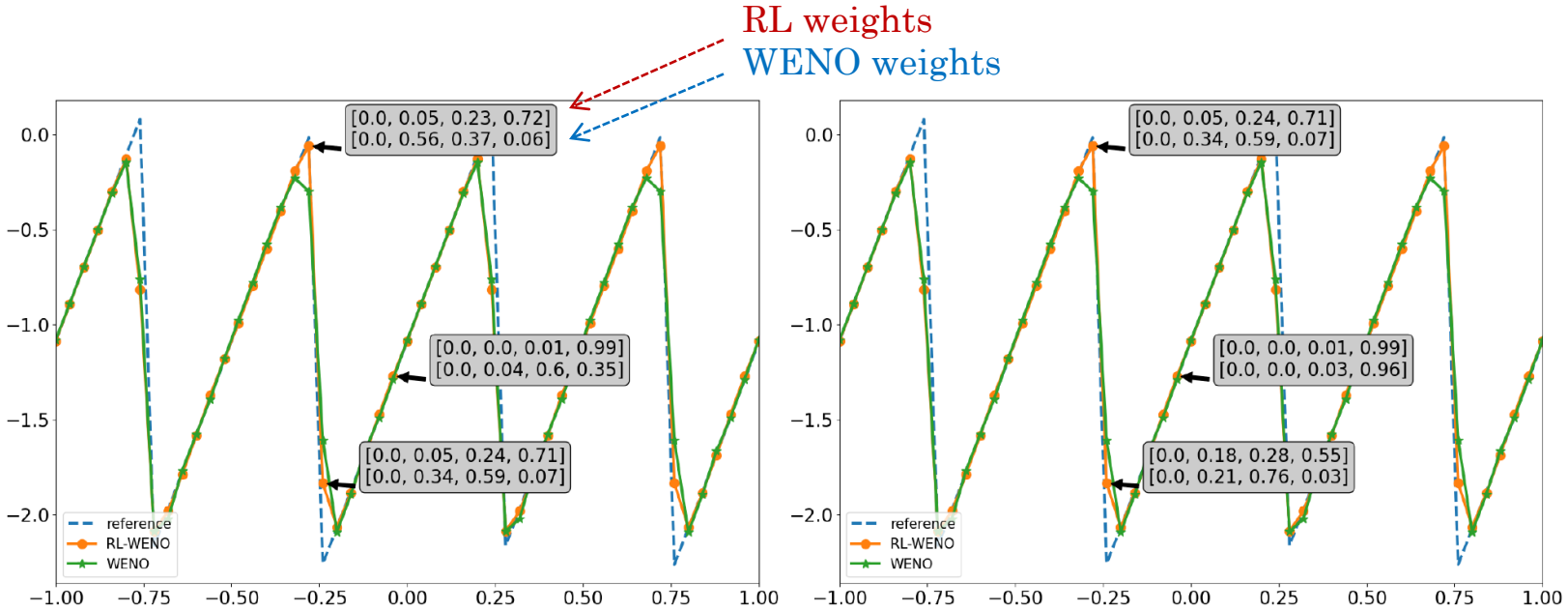
$$\pi^{RL}(s_j) = \left(w^{-2}_{j-\frac{1}{2}}, w^{-1}_{j-\frac{1}{2}}, w^{0}_{j-\frac{1}{2}}, w^{1}_{j-\frac{1}{2}}, w^{-2}_{j+\frac{1}{2}}, w^{-1}_{j+\frac{1}{2}}, w^{0}_{j+\frac{1}{2}}, w^{1}_{j+\frac{1}{2}}\right)$$

Action

$$\hat{f}_{j-\frac{1}{2}} = \sum_{i=-2}^{1} w^{i}_{j-\frac{1}{2}} \hat{f}^{i}_{j-\frac{1}{2}}, \quad \hat{f}_{j+\frac{1}{2}} = \sum_{i=-2}^{1} w^{i}_{j+\frac{1}{2}} \hat{f}^{i}_{j+\frac{1}{2}}$$

Learning to Discretize

- RL-WENO v.s. WENO-5:
  - Strategies of weights ($w^{r}_{j\pm\frac{1}{2}}$) selection



$\hat{f}_{j-\frac{1}{2}}$ $\hat{f}_{j+\frac{1}{2}}$

Learning to Discretize

- Generalization of RL-WENO:
  - To other flux

Δt \ Δx	0.02		0.04		0.05	
	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO
0.002	10.05 (2.74)	10.25 (2.65)	16.89 (3.59)	17.09 (3.56)	17.07 (4.2)	17.4 (4.37)
0.003	-	10.26 (2.65)	16.9 (3.59)	17.1 (3.56)	17.09 (4.2)	17.42 (4.37)
0.004	-	-	16.9 (3.6)	17.1 (3.56)	17.1 (4.2)	17.43 (4.37)
0.005	-	-	-	17.12 (3.56)	17.11 (4.22)	17.43 (4.38)

$$f(u) = \frac{1}{16}u^4$$

  - Train on “Inviscid” and test on “Forcing”

Δx \ η	0.01		0.02		0.04	
	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO
0.02	4.75 (0.73)	4.74 (0.7)	2.38 (0.46)	2.59 (0.45)	1.07 (0.2)	1.02 (0.27)
0.04	10.78 (1.4)	10.3 (1.35)	6.95 (1.15)	6.6 (0.95)	3.84 (0.7)	3.68 (0.62)
0.05	13.97 (1.93)	13.55 (2.07)	9.76 (1.44)	9.33 (1.4)	5.67 (0.89)	5.42 (0.81)

Learning to Discretize

- Generalization of RL-WENO:
  - Train on "Inviscid" and test on "Viscous"

η / Δx	0.01		0.02		0.04	
	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO
0.02	1.85 (0.58)	1.94 (0.55)	0.85 (0.38)	0.87 (0.37)	0.45 (0.22)	0.41 (0.21)
0.04	4.91 (1.64)	4.93 (0.73)	2.34 (0.82)	2.33 (0.51)	1.14 (0.58)	0.96 (0.36)
0.05	8.31 (5.49)	7.66 (1.71)	3.59 (1.05)	3.64 (0.77)	1.84 (0.88)	1.52 (0.5)

  - Train on "Forcing", test on "Inviscid" and "Viscous"

η / Δx	0		0.01		0.02		0.04	
	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO	RL-WENO	WENO
0.02	13.06 (3.22)	11.13 (3.83)	1.9 (0.58)	1.94 (0.55)	0.92 (0.37)	0.87 (0.37)	0.49 (0.22)	0.41 (0.21)
0.04	18.7 (7.65)	19.48 (9.33)	4.76 (1.19)	4.93 (0.73)	2.3 (0.78)	2.33 (0.51)	1.13 (0.62)	0.96 (0.36)
0.05	28.57 (22.3)	35.88 (27.48)	7.75 (2.76)	7.66 (1.71)	3.51 (1.24)	3.64 (0.77)	1.72 (0.95)	1.52 (0.5)

Learning to Discretize

- RL-WENO v.s. L3D:
  - Train on "Forcing" and test on "Forcing"

η	Δx	RL-WENO	WENO	L3D
0.01	0.02	4.64 (0.7)	4.74 (0.7)	3.09 (0.9)
	0.04	10.36 (1.45)	10.3 (1.35)	nan (nan)
	0.05	13.33 (2.05)	13.55 (2.07)	nan (nan)
0.02	0.02	2.41 (0.43)	2.59 (0.45)	6.89 (0.74)
	0.04	6.53 (1.13)	6.6 (0.95)	4.08 (1.12)
	0.05	9.19 (1.54)	9.33 (1.4)	7.58 (1.11)
0.04	0.02	1.09 (0.21)	1.02 (0.27)	11.01 (1.3)
	0.04	3.42 (0.65)	3.68 (0.62)	11.22 (1.22)
	0.05	4.97 (0.91)	5.42 (0.81)	9.37 (1.21)

Learning to Discretize

- RL-WENO v.s. PINN:
  - RL-WENO trained on "Viscous"

η	Δx	RL-WENO	WENO	PINNs
0	0.02	10.81 (3.96)	11.13 (3.83)	40.06(18.82)
	0.04	18.89 (12.69)	19.48 (9.33)	51.61(23.68)
	0.05	28.48 (33.04)	35.93 (27.61)	51.18(18.31)
0.01	0.02	1.64 (0.57)	1.94 (0.55)	34.09(26.92)
	0.04	4.29 (0.93)	4.93 (0.73)	48.28(29.71)
	0.05	7.6 (3.94)	7.66 (1.71)	47.95(19.86)
0.02	0.02	0.74 (0.35)	0.87 (0.37)	31.13(34.35)
	0.04	1.91 (0.58)	2.33 (0.51)	47.41(35.84)
	0.05	2.97 (1.07)	3.64 (0.77)	43.05(20.1))
0.04	0.02	0.42 (0.21)	0.41 (0.21)	34.98(37.83)
	0.04	0.87 (0.38)	0.96 (0.36)	52.27(44.74)
	0.05	1.38 (0.61)	1.52 (0.5)	47.0(24.47)

Learning to Discretize

- Potential of the proposed RL framework
  - Most numerical solvers of conservation law can be interpreted naturally as a sequential decision making process;
  - The policy π^f (i.e. agent) considers long-term accuracy (non-greedy).
  - RL can gracefully handle non-smooth norms of the reward and discrete action space.
  - Learning the policy π^f in RL framework making the method meta-learning-like, i.e. it can learn the principles of discretization mimicking human experts.

Learning to Solve Prameterized PDEs: A Meta-Learning Approach

- Yuyan Chen, Bin Dong and Jinchao Xu, *Meta-MgNet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations*, arXiv:2010.14088, 2020.

Parameterized Partial Differential Equations (PDEs)

- General parameterized PDEs:

$$\mathcal{L}(u, \boldsymbol{x}, t; \boldsymbol{\eta}) = 0, \qquad x \in \Omega \subset \mathbb{R}^d, t \geq 0$$

  - Boundary condition: $\mathcal{B}(u, \boldsymbol{x}_{BC}, t; \boldsymbol{\eta}) = 0$
  - Initial condition: $u_0 = u_{IC}(\boldsymbol{x}; \boldsymbol{\eta})$
  - Others
    - State variable: $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t; \boldsymbol{\eta}) \in \mathbb{R}^m$
    - Parameter vector: $\boldsymbol{\eta} \in \boldsymbol{D} \subset \mathbb{R}^p$
- Linear steady parameterized PDEs:

$$\begin{cases} \mathcal{A}_\eta \underset{\sim}{u} = \underset{\sim}{f}, & \text{in } \Omega, \\ \underset{\sim}{u} = \underset{\sim}{u}_b, & \text{on } \partial\Omega. \end{cases}$$

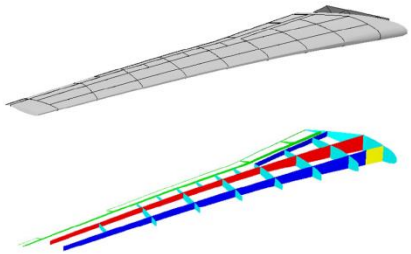
e.g. 2D anisotropic diffusion equation with $\boldsymbol{\eta} = (\varepsilon, \theta)$

$$\begin{cases} -\nabla \cdot (C \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

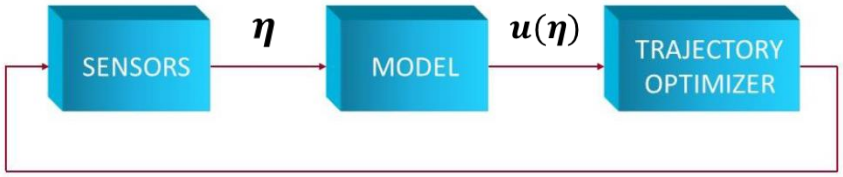
$$C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

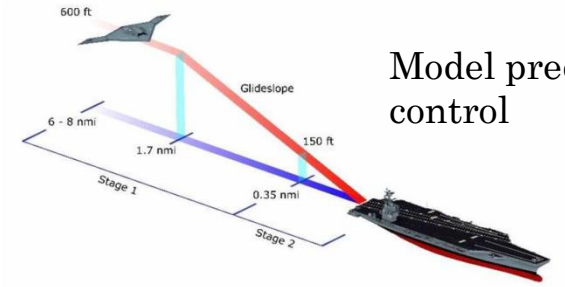
Applications That Require Efficient Solvers For Parameterized PDEs

- Typical parameters of interest
  - Shape parameters
  - Material (properties) parameters
  - Operation parameters (e.g. flight conditions, cruise conditions, etc.)
  - Initial and boundary conditions



- Scenarios require solving $\boldsymbol{u}(\boldsymbol{\eta})$ for multiple $\boldsymbol{\eta}$
  - Inverse problems
  - Uncertainty quantification
  - Design optimization
  - Optimal control
  - Model predictive control





Model predictive control

Multigrid and the Multigrid Network (MgNet)

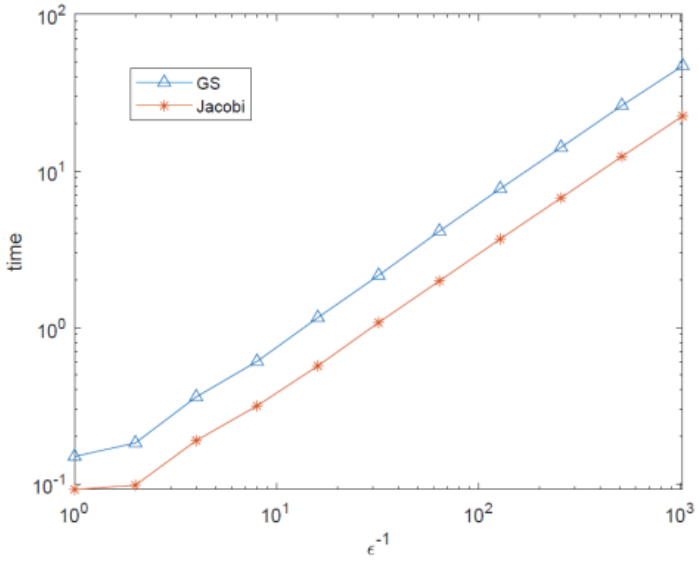
Multigrid method viewed as CNN – a control perspective

Direct Application of Existing Numerical Solvers

- We focus on the linear problem: $\boldsymbol{A_\eta u = f}$
- Multigrid method (MG) has linear complexity
- However, CPU time for MG can go up significantly when $\boldsymbol{\eta}$ is within certain range
- For example

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases}$$

$$C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$



Direct Application of Existing Numerical Solvers

- We focus on the linear problem: $\boldsymbol{A_\eta u = f}$
- Multigrid method (MG) has linear complexity
- However, CPU time for MG can go up significantly when $\boldsymbol{\eta}$ is within certain range
- This can be improved by manually adjusting crucial components in MG:
  - Smoother: damped coefficient of damped Jacobi smoother
  - Prolongations
  - Restrictions
- Can machine learning help?

MG v.s. CNN (He & Xu, 2019)

- MG is an iterative scheme

$$\boldsymbol{u}_{t+1} = \boldsymbol{u}_t + \mathrm{MG}(\boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}_t), t = 0,1, \dots, T$$

- Here, MG is given by

Algorithm 1 $\mathbf{u} = \mathrm{Mg}(\mathbf{f}; J, \nu_1, \cdots, \nu_J)$

Hyper-parameters: number of grids J, times of smooth in each grid: $\nu_1, \cdots, \nu_J$
Input: right hand side $\mathbf{f}$
Output: approximate solution $\mathbf{u}$
Initialization:

$$\mathbf{f}^1 \leftarrow \mathbf{f}, \quad \mathbf{u}^{1,0} \leftarrow \mathbf{0}, \quad \mathbf{r}^{1,0} \leftarrow \mathbf{f}.$$

Smoothing and restriction from fine to coarse level (nested)
for $\ell = 1 : J$ **do**
  Smoothing
  if $\ell = J$ **then**

$$\mathbf{u}^{\ell,1} \leftarrow (\mathbf{A}^\ell)^{-1}\mathbf{r}^{\ell,0}$$

  else
    for $i = 1 : \nu_\ell$ **do**

$$\mathbf{u}^{\ell,i} \leftarrow \mathbf{u}^{\ell,i-1} + \mathbf{B}^\ell \mathbf{r}^{\ell,i-1}$$
$$\mathbf{r}^{\ell,i} \leftarrow \mathbf{f}^\ell - \mathbf{A}^\ell \mathbf{u}^{\ell,i}.$$

    end for
  end if
  Form restricted residual

$$\mathbf{f}^{\ell+1} \leftarrow \mathbf{R}_\ell^{\ell+1}\mathbf{r}^{\ell,\nu_\ell}, \quad \mathbf{u}^{\ell+1,0} \leftarrow \mathbf{0}, \quad \mathbf{r}^{\ell+1,0} \leftarrow \mathbf{f}^{\ell+1}.$$

end for
Prolongation and restriction from coarse to fine level
for $\ell = J - 1 : 1$ **do**
  Coarse grid correction

$$\mathbf{u}^{\ell,\nu_\ell} \leftarrow \mathbf{u}^{\ell,\nu_\ell} + \mathbf{P}_{\ell+1}^{\ell}\mathbf{u}^{\ell+1,\nu_\ell}.$$

end for
return $\mathbf{u} = \mathbf{u}^{1,\nu_1}$.

To make this an CNN, the operators $\boldsymbol{B}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{P}$, need to be expressed as convolutions.

MG v.s. CNN (He & Xu, 2019)

- Converting MG to an CNN
  - System $\boldsymbol{A}$: if the basis functions $\{\phi_{k,j,i}\}$ are generated by translates of a set of functions $\{\varphi_k\}$, then

Theorem 1. *If a discretized scheme in FDM or FEM satisfies Assumption 2.1, the discretized PDE $\mathcal{K}v = f$ can be written in the form $\mathsf{K} \star \mathsf{v} = \mathsf{f}$ with K and f given as follows*

$$\mathsf{K} = (\mathsf{K}_{l,k,j,i}) = K(\varphi_k(x - ih, y - jh), \varphi_l(x, y)) \quad \text{and} \quad \mathsf{f} = (\mathsf{f}_{l,j,i}) = f(\phi_{l,j,i}).$$

$\mathcal{K}$	difference scheme	kernel K
∂_x	$\frac{1}{h}(\mathsf{v}_{i,j} - \mathsf{v}_{i-1,j})$	$\frac{1}{h}\begin{pmatrix}-1 & 1 & 0\end{pmatrix}$
	$\frac{1}{h}(\mathsf{v}_{i+1,j} - \mathsf{v}_{i,j})$	$\frac{1}{h}\begin{pmatrix}0 & -1 & 1\end{pmatrix}$
∂_y	$\frac{1}{h}(\mathsf{v}_{i,j} - \mathsf{v}_{i,j-1})$	$\frac{1}{h}\begin{pmatrix}-1\\1\\0\end{pmatrix}$
	$\frac{1}{h}(\mathsf{v}_{i,j+1} - \mathsf{v}_{i,j})$	$\frac{1}{h}\begin{pmatrix}0\\-1\\1\end{pmatrix}$
∂_{xx}	$\frac{1}{h^2}(\mathsf{v}_{i-1,j} + \mathsf{v}_{i+1,j} - 2\mathsf{v}_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}1 & -2 & 1\end{pmatrix}$
∂_{xy}	$\frac{1}{4h^2}(\mathsf{v}_{i-1,j-1} + \mathsf{v}_{i+1,j+1} - \mathsf{v}_{i+1,j-1} - \mathsf{v}_{i-1,j+1})$	$\frac{1}{4h^2}\begin{pmatrix}1 & 0 & -1\\0 & 0 & 0\\-1 & 0 & 1\end{pmatrix}$
∂_{yy}	$\frac{1}{h^2}(\mathsf{v}_{i,j-1} + \mathsf{v}_{i,j+1} - 2\mathsf{v}_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}1\\-2\\1\end{pmatrix}$
Δ	$\frac{1}{h^2}(\mathsf{v}_{i-1,j} + \mathsf{v}_{i+1,j} + \mathsf{v}_{i,j-1} + \mathsf{v}_{i,j+1} - 4\mathsf{v}_{i,j})$	$\frac{1}{h^2}\begin{pmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{pmatrix}$

[Chen, Dong and Xu, preprint 2020]

MG v.s. CNN (He & Xu, 2019)

- Converting MG to an CNN
  - System ***A***: if the basis functions $\{\phi_{k,j,i}\}$ are generated by translates of a set of functions $\{\varphi_k\}$, then

Theorem 1. *If a discretized scheme in FDM or FEM satisfies Assumption 2.1, the discretized PDE* $\mathcal{K}v = f$ *can be written in the form* $\mathsf{K} \star \mathsf{v} = \mathsf{f}$ *with* K *and* f *given as follows*

$$\mathsf{K} = (\mathsf{K}_{l,k,j,i}) = K(\varphi_k(x - ih, y - jh), \varphi_l(x, y)) \quad \textit{and} \quad \mathsf{f} = (\mathsf{f}_{l,j,i}) = f(\phi_{l,j,i}).$$

[Chen, Dong and Xu, preprint 2020]

  - Operator ***R***: convolution with a stride≥ 2
  - Operator ***P***: deconvolution (upsampling + convolution)
  - Operator ***B***: replaced by convolution or a small CNN
- This leads to the multigrid network (MgNet)

PDE-MgNet: for Solving PDEs

- PDE-MgNet

$$u_{t+1} = u_t + \text{PDE-MgNet}(f - A \star u_t).$$

Algorithm 2 $u = \text{PDE-MgNet}(f; J, \nu_1, \cdots, \nu_J)$

Hyper-parameters: number of grids J, times of smooth in each grid: $\nu_1, \cdots, \nu_J$
Input: right-hand sidef
Output: approximate solution u
Initialization

$$f^1 \leftarrow f, \quad u^{1,0} \leftarrow 0, \quad r^{1,0} \leftarrow f.$$

Smoothing and restriction from fine to coarse level
for $\ell = 1 : J$ **do**
  Smoothing:
  if $\ell = J$ **then**
    Convert $r^{\ell,0}$ into vector form $\mathbf{r}^{\ell,0}$ and A^ℓ into matrix form $\mathbf{A}^\ell$.

$$\mathbf{u}^{\ell,1} \leftarrow (\mathbf{A}^\ell)^{-1}\mathbf{r}^{\ell,0}.$$

    Convert $\mathbf{u}^{\ell,1}$ into tensor form $u^{\ell,1}$.
  else
    for $i = 1 : \nu_\ell$ **do**

$$u^{\ell,i} \leftarrow u^{\ell,i-1} + B^{\ell,i-1} \star r^{\ell,i-1},$$
$$r^{\ell,i} \leftarrow f^\ell - A^\ell \star u^{\ell,i}.$$

    end for
  end if
  Form restricted residual

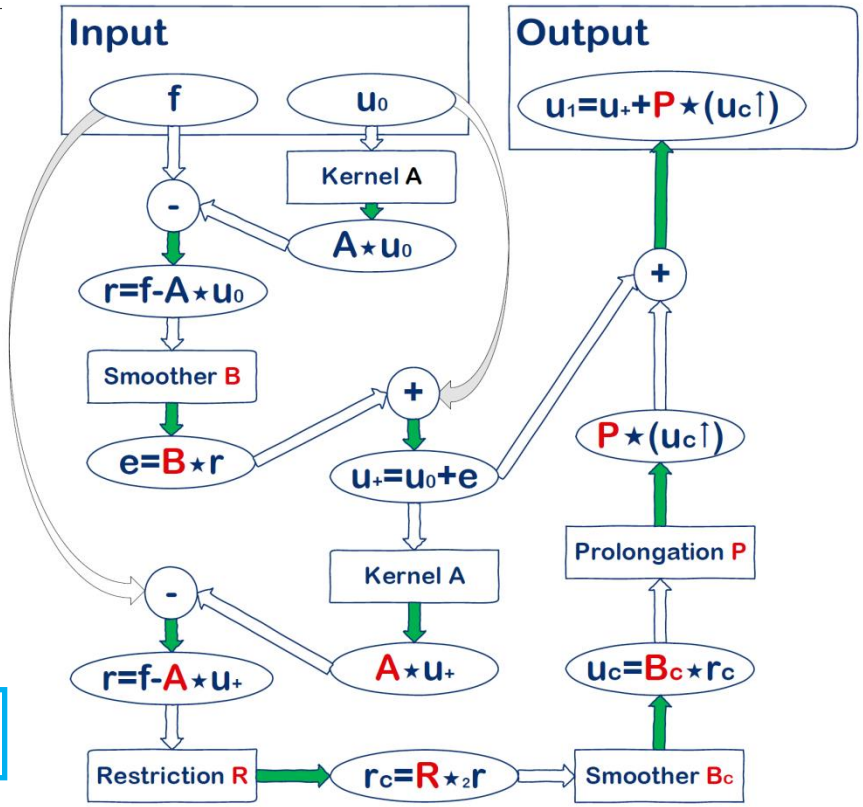
$$f^{\ell+1} \leftarrow R_\ell^{\ell+1} \star_2 r^{\ell,\nu_\ell}, \quad u^{\ell+1,0} \leftarrow 0, \quad f^{\ell+1,0} \leftarrow f^{\ell+1}.$$

end for
Prolongation from coarse to fine level
for $\ell = J - 1 : 1$ **do**
  Coarse grid correction

$$u^{\ell,\nu_\ell} \leftarrow u^{\ell,\nu_\ell} + P_{\ell+1}^\ell \star^2 u^{\ell+1,\nu_\ell}.$$

end for
return $u = u^{1,\nu_1}$.

$$Loss \approx \frac{1}{M_{\text{train}}} \sum_{f \in \mathfrak{X}_F} \frac{||f - A \star u_T||^2}{||f||^2}.$$



PDE-MgNet: for Solving PDEs

- PDE-MgNet

$$u_{t+1} = u_t + \text{PDE-MgNet}(f - A \star u_t).$$

- We can apply PDE-MgNet to solve parametric PDE: $\boldsymbol{A_\eta u = f}$
- Two supervised learning strategies:
  - For a given set of $\boldsymbol{\eta}$, train PDE-MgNet
  - For every given $\boldsymbol{\eta}$, train PDE-MgNet (we call this PDE-MgNet-$\boldsymbol{\eta}$)
- Drawback: poor generalization!

A Meta-Learning Approach

Solving parameterized PDEs as multi-task learning

- Yuyan Chen, Bin Dong and Jinchao Xu, *Meta-MgNet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations*, arXiv:2010.14088, 2020.

Motivation

- Single task: learning a solver for a given $\boldsymbol{\eta}$
- Multi-task: learning a solver for a set of $\boldsymbol{\eta}$
- Key difference from supervised learning strategy:
  - Leveraging **common structures** hidden in the tasks!
- Effective approach: Meta-Learning
  - Finding a good initialization for all tasks

(Finn, Abbeel and Levine, 2017; Nichol, Achiam and Schulman, 2018)

  - Designing a hypernetwork to infer suitable parameters for each task

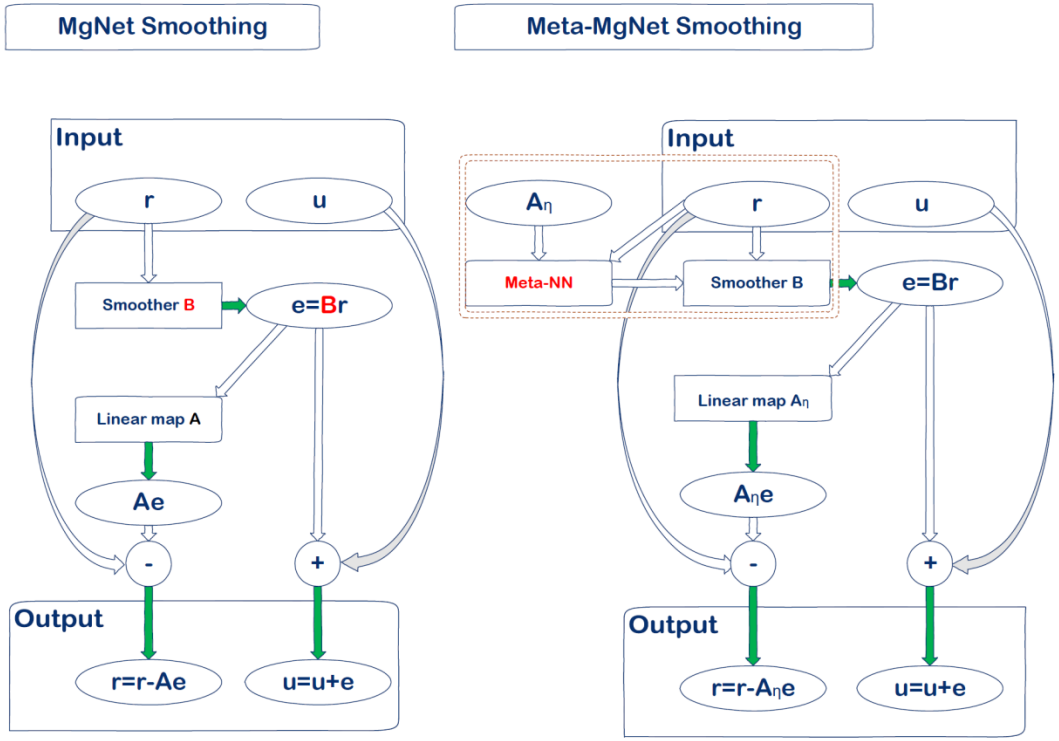
(Ha, Dai and Le, 2016; Lorraine and Duvenaud, 2018; Brock, Lim, Ritchie and Weston, 2017;Zhang, Liu, Yu and Dong, 2020)

Meta-MgNet

- Architecture:

$$u_{t+1} = u_t + \text{Meta-MgNet}(f - A_\eta \star u_t, A_\eta),$$

- All components of Meta-MgNet is the same as PDE-MgNet, except for the smoother:



Meta-NN

Input:

- Kernel of the operator $\boldsymbol{A_\eta}$
- Residual $\boldsymbol{r}$

Output: a set of vectors that spans a subspace for subspace correction

META-MGNET

- Learning subspace correction by Meta-NN $\mathcal{G}$:

Algorithm 4 $\mathcal{B} = B_{sc}(\mathsf{r}, \mathsf{A}_\eta; \mathcal{G})$

Hyper-parameters: Meta-NN $\mathcal{G}$
Inputs: r, A_η
Output: $\mathcal{B}$
1. Calculate subspace:

$$\mathsf{G} \leftarrow \mathcal{G}(\mathsf{r}, \mathsf{A}_\eta),$$

where G is a tensor with shape $L \times K \times J \times I$.
2. Reshape the tensor G to $L \times KJI$ matrix, and write its transpose as $\mathbf{G}$, which is a $KJI \times L$ matrix.
3. Do subspace correction with the subspace $\mathbb{G} = \text{range}(\mathbf{G})$:

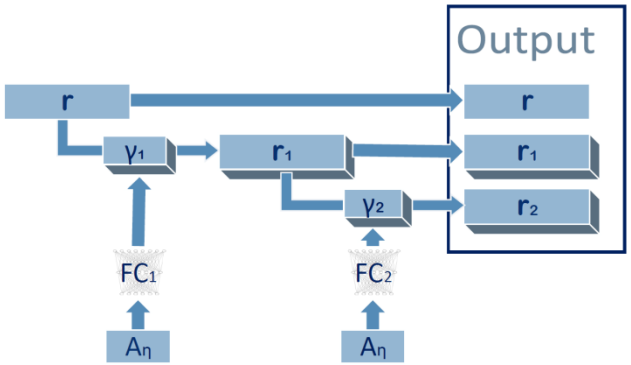
$$\mathbf{S} \leftarrow \mathbf{A}_\eta \mathbf{G}.$$

$$\mathbf{e} = \mathbf{G}(\mathbf{S}^\top \mathbf{S})^{-1} \mathbf{S}^\top \mathbf{r}.$$

4. Define the effect of $\mathcal{B}$ as

$$\mathcal{B}(\mathsf{r}, \mathbf{A}_\eta; \mathcal{G}) := \text{Reshape}(\mathbf{e}).$$

where Reshape$(\cdot)$ means to reshape $\mathbf{e}$ to the same shape as tensor r.
return $\mathcal{B}$.



Meta-NN $\mathcal{G}_\theta(r, A_\eta)$

- Motivation: Krylov subspace

$$\mathbb{G}_K = \{\mathbf{f}_0(\mathbf{A})\mathbf{r}, \mathbf{f}_1(\mathbf{A})\mathbf{r}, ..., \mathbf{f}_k(\mathbf{A})\mathbf{r}\}, \text{ and } \mathbf{f}_i(\mathbf{A}) = \mathbf{A}^i.$$

- Meta-NN: $\mathcal{G}_{\boldsymbol{\theta}}(r, A_\eta) = \mathcal{N}_{FC_{\boldsymbol{\theta}}(A_\eta)}(r)$,
  - $\mathcal{N}_{\boldsymbol{\gamma}}$ is a 3-layer dense-net block (Huang et al. 2017) with parameter $\boldsymbol{\gamma}$
  - $FC_{\boldsymbol{\theta}}$ is a 2-layer fully connected neural network with parameter $\boldsymbol{\theta}$
- Convergence guarantee √

EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Training:
  - $\theta = 0,\ \lg\frac{1}{\varepsilon} \sim \mathcal{U}[0,5]$
  - randomly generate 100 right-hand-side function $f \sim \mathcal{N}(0,1)$.
- Testing:
  - $\theta = 0,\ \varepsilon = 10^{-l}, l = 0,1,\dots,5$ (in-distribution generalization)
  - randomly generate 10 right-hand-side function $f \sim \mathcal{N}(0,1)$
  - select stopping criteria

$$\frac{||\mathbf{f} - \mathbf{A}_\eta \mathbf{u}_t||_2}{||\mathbf{f}||_2} < 10^{-6}.$$

- Report mean±std of number of iterations and computation time for each experiment with each compared algorithm

EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Results: $\theta = 0$

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG(Krylov)	MG(GS)	MG(line-GS)	MG(Jacobi)
$\epsilon = 1$	4.0 ± 0.00	-	7.0 ± 0.00	4.0 ± 0.00	10.0 ± 0.00	-	15.0 ± 0.00
$\epsilon = 10^{-1}$	7.5 ± 0.50	19.2 ± 0.40	21.2 ± 0.60	7.9 ± 0.30	33.7 ± 0.48	-	90.2 ± 0.98
$\epsilon = 10^{-2}$	35.1 ± 1.04	178.9 ± 2.74	149.7 ± 3.44	52.5 ± 0.81	253.6 ± 4.19	553.6 ± 27.56	752.8 ± 12.23
$\epsilon = 10^{-3}$	171.6 ± 6.34	$1.2e3 \pm 12.85$	910.9 ± 15.64	345.9 ± 3.88	$1.9e3 \pm 25.56$	62.3 ± 1.76	$5.6e3 \pm 119.42$
$\epsilon = 10^{-4}$	375.2 ± 5.88	-	$3.1e3 \pm 35.70$	$2.2e3 \pm 27.94$	-	11.0 ± 0.00	-
$\epsilon = 10^{-5}$	797.8 ± 12.76	-	$9.9e3 \pm 40.81$	$7.6e3 \pm 81.96$	-	11.0 ± 0.00	-
wall time							
$\epsilon = 1$	0.03 ± 0.00	-	$\mathbf{0.02 \pm 0.00}$	$\mathbf{0.02 \pm 0.00}$	0.14 ± 0.01	-	0.04 ± 0.00
$\epsilon = 10^{-1}$	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00	$\mathbf{0.04 \pm 0.00}$	0.48 ± 0.02	-	0.23 ± 0.00
$\epsilon = 10^{-2}$	$\mathbf{0.22 \pm 0.01}$	0.44 ± 0.01	0.37 ± 0.01	0.25 ± 0.00	3.47 ± 0.15	12.6 ± 0.86	1.85 ± 0.03
$\epsilon = 10^{-3}$	$\mathbf{1.06 \pm 0.04}$	3.04 ± 0.03	2.28 ± 0.05	1.64 ± 0.02	27.33 ± 0.68	1.40 ± 0.04	13.95 ± 0.29
$\epsilon = 10^{-4}$	2.31 ± 0.03	-	7.69 ± 0.13	10.56 ± 0.14	-	$\mathbf{0.27 \pm 0.01}$	-
$\epsilon = 10^{-5}$	4.91 ± 0.08	-	24.49 ± 0.14	35.67 ± 0.40	-	$\mathbf{0.27 \pm 0.02}$	-

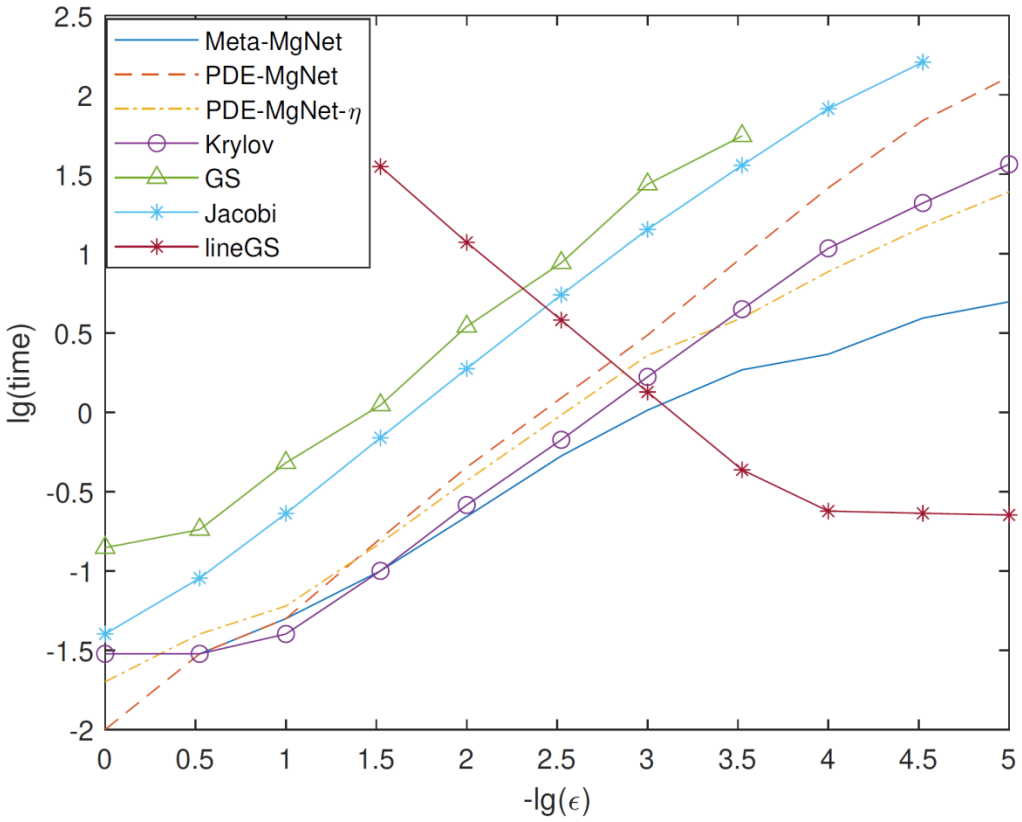
- Best in-distribution (out-of-distribution as well) generalization
- Even better than PDE-MgNet-$\boldsymbol{\eta}$, indicating the benefit of exploiting common structure through multi-task learning perspective

Experiments

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Results:



EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Training:
  - $\theta = 0, \lg\frac{1}{\varepsilon} \sim \mathcal{U}[2,3]$
  - randomly generate 100 right-hand-side function $f \sim \mathcal{N}(0,1)$.
- Testing:
  - $\theta = 0, \varepsilon = 1, 10^{-1}, 10^{-4}, 10^{-5}$(out-of-distribution transfer for ε)
  - randomly generate 10 right-hand-side function $f \sim \mathcal{N}(0,1)$
  - select stopping criteria $\dfrac{||\mathbf{f} - \mathbf{A}_\eta \mathbf{u}_t||_2}{||\mathbf{f}||_2} < 10^{-6}.$
- Report mean±std of number of iterations and computation time for each experiment with each compared algorithm

EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Results: $\theta = 0$

#iterations	Meta-MgNet	PDE-MgNet	PDE-MgNet-η	MG(Krylov)	MG(GS)	MG(line-GS)	MG(Jacobi)
$\epsilon = 1$	7.0 ± 0.00	-	7.0 ± 0.00	4.0 ± 0.00	10.0 ± 0.00	-	15.0 ± 0.00
$\epsilon = 10^{-1}$	10.0 ± 0.00	23.0 ± 0.00	21.2 ± 0.60	7.9 ± 0.30	33.7 ± 0.48	-	90.2 ± 0.98
$\epsilon = 10^{-4}$	340.7 ± 3.52	$5.8e3 \pm 121.90$	$3.1e3 \pm 35.70$	$2.2e3 \pm 27.94$	-	11.0 ± 0.00	-
$\epsilon = 10^{-5}$	817.2 ± 97.97	-	$9.9e3 \pm 40.81$	$7.6e3 \pm 81.96$	-	11.0 ± 0.00	-
wall time							
$\epsilon = 1$	0.05 ± 0.00	-	$\mathbf{0.02 \pm 0.00}$	$\mathbf{0.02 \pm 0.00}$	0.14 ± 0.01	-	0.04 ± 0.00
$\epsilon = 10^{-1}$	0.07 ± 0.00	0.06 ± 0.00	0.06 ± 0.00	$\mathbf{0.04 \pm 0.00}$	0.48 ± 0.02	-	0.23 ± 0.00
$\epsilon = 10^{-4}$	2.08 ± 0.02	14.38 ± 0.32	7.69 ± 0.13	10.56 ± 0.14	-	$\mathbf{0.27 \pm 0.01}$	-
$\epsilon = 10^{-5}$	4.99 ± 0.59	-	24.49 ± 0.14	35.67 ± 0.40	-	$\mathbf{0.27 \pm 0.02}$	-

- Best out-of-distribution transfer
- Even better than PDE-MgNet-$\boldsymbol{\eta}$, indicating the benefit of exploiting common structure through multi-task learning perspective

EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Training:
  - $\theta = \mathcal{U}[\frac{1}{8}\pi, \frac{3}{8}\pi], \lg\frac{1}{\varepsilon} \sim \mathcal{U}[0,5]$
  - randomly generate 100 right-hand-side function $f \sim \mathcal{N}(0,1)$.
- Testing:
  - $\theta = 0.05\pi, 0.12\pi, 0.4\pi, 0.5\pi, \varepsilon = 10^{-l}, l = 0,1,\ldots,5$ (out-of-distribution transfer for θ)
  - randomly generate 10 right-hand-side function $f \sim \mathcal{N}(0,1)$
  - select stopping criteria $\dfrac{||\mathbf{f} - \mathbf{A}_\eta \mathbf{u}_t||_2}{||\mathbf{f}||_2} < 10^{-6}.$
- Report mean±std of number of iterations and computation time for each experiment with each compared algorithm

EXPERIMENTS

- 2D anisotropic diffusion equation

$$\begin{cases} -\nabla \cdot (C\nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \qquad C = C(\epsilon, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

- Results:

#iterations	Meta-MgNet,$\theta = 0.05\pi$	PDE-MgNet,$\theta = 0.05\pi$	Meta-MgNet,$\theta = 0.12\pi$	PDE-MgNet,$\theta = 0.12\pi$
$\epsilon = 1$	3.0 ± 0.00	-	3.0 ± 0.00	-
$\epsilon = 10^{-1}$	10.6 ± 0.49	-	10.1 ± 0.30	132.6 ± 3.10
$\epsilon = 10^{-2}$	71.5 ± 1.57	-	72.0 ± 1.61	566.3 ± 11.36
$\epsilon = 10^{-3}$	322.4 ± 7.03	-	233.2 ± 7.49	2060.8 ± 55.09
$\epsilon = 10^{-4}$	526.7 ± 14.64	-	306.0 ± 7.78	2850.3 ± 150.40
$\epsilon = 10^{-5}$	557.4 ± 14.72	-	314.0 ± 4.86	2852.5 ± 33.31
wall time				
$\epsilon = 1$	0.03 ± 0.00	-	0.03 ± 0.00	-
$\epsilon = 10^{-1}$	0.07 ± 0.00	-	0.07 ± 0.00	0.35 ± 0.01
$\epsilon = 10^{-2}$	0.46 ± 0.01	-	0.46 ± 0.01	1.49 ± 0.04
$\epsilon = 10^{-3}$	2.05 ± 0.04	-	1.48 ± 0.05	5.42 ± 0.18
$\epsilon = 10^{-4}$	3.34 ± 0.09	-	1.95 ± 0.05	7.49 ± 0.45
$\epsilon = 10^{-5}$	3.54 ± 0.09	-	1.99 ± 0.03	7.49 ± 0.15
#iterations	Meta-MgNet,$\theta = 0.4\pi$	PDE-MgNet,$\theta = 0.4\pi$	Meta-MgNet,$\theta = 0.5\pi$	PDE-MgNet,$\theta = 0.5\pi$
$\epsilon = 1$	3.0 ± 0.00	-	3.0 ± 0.0	-
$\epsilon = 10^{-1}$	9.0 ± 0.00	51.7 ± 1.27	8.9 ± 0.30	49.3 ± 1.42
$\epsilon = 10^{-2}$	65.2 ± 1.54	434.5 ± 7.75	53.5 ± 1.12	428.8 ± 12.83
$\epsilon = 10^{-3}$	240.3 ± 3.41	1698.2 ± 50.92	262.9 ± 5.96	2786.2 ± 16.81
$\epsilon = 10^{-4}$	327.5 ± 5.33	2423.0 ± 67.84	526.4 ± 25.51	-
$\epsilon = 10^{-5}$	339.5 ± 6.92	2505.2 ± 85.99	908.7 ± 27.43	-
wall time				
$\epsilon = 1$	0.03 ± 0.00	-	0.03 ± 0.00	-
$\epsilon = 10^{-1}$	0.06 ± 0.00	0.14 ± 0.00	0.06 ± 0.00	0.13 ± 0.01
$\epsilon = 10^{-2}$	0.42 ± 0.01	1.15 ± 0.03	0.35 ± 0.01	1.13 ± 0.03
$\epsilon = 10^{-3}$	1.53 ± 0.02	4.48 ± 0.12	1.67 ± 0.04	7.34 ± 0.11
$\epsilon = 10^{-4}$	2.09 ± 0.03	6.38 ± 0.22	3.35 ± 0.16	-
$\epsilon = 10^{-5}$	2.15 ± 0.04	6.61 ± 0.31	5.76 ± 0.18	-

Conclusions and Future Directions

- Deep learning ≈ optimal control
- Hypernetwork structure leads to good generalization
- Future work:
  - PDE-Net
    - Real dynamical data
  - Meta-learning for multigrid method
    - Learning prolongations and restrictions
    - Learning on irregular grid with graph neural networks
    - Learning other iterative numerical solving with meta-learning
  - RL approach for conservation laws
    - 2D or 3D cases
    - More aggressive policy design
    - Adaptive or moving mesh

Thanks for Your Attention!

My Webpage:

http://bicmr.pku.edu.cn/~dongbin