Data Seminar

04/02/2021

Outline

1. DeepXDE: A Deep Learning Library of Solving Differential Equations. Lu L, Meng X, Mao Z, et al. SIAM Review 2021.
2. Solving high-dimensional partial differential equations using deep learning. Han J, Jentzen A, Weinan E. PNAS 2018.
3. Forecasting Sequential Data Using Consistent Koopman Autoencoders. Azencot O, Erichson N B, Lin V, et al. ICML 2020.

Outline

1. **DeepXDE: A Deep Learning Library of Solving Differential Equations. Lu L, Meng X, Mao Z, et al. SIAM Review 2021.**
2. Solving high-dimensional partial differential equations using deep learning. Han J, Jentzen A, Weinan E. PNAS 2018.
3. Forecasting Sequential Data Using Consistent Koopman Autoencoders. Azencot O, Erichson N B, Lin V, et al. ICML 2020.

Motivation & Idea

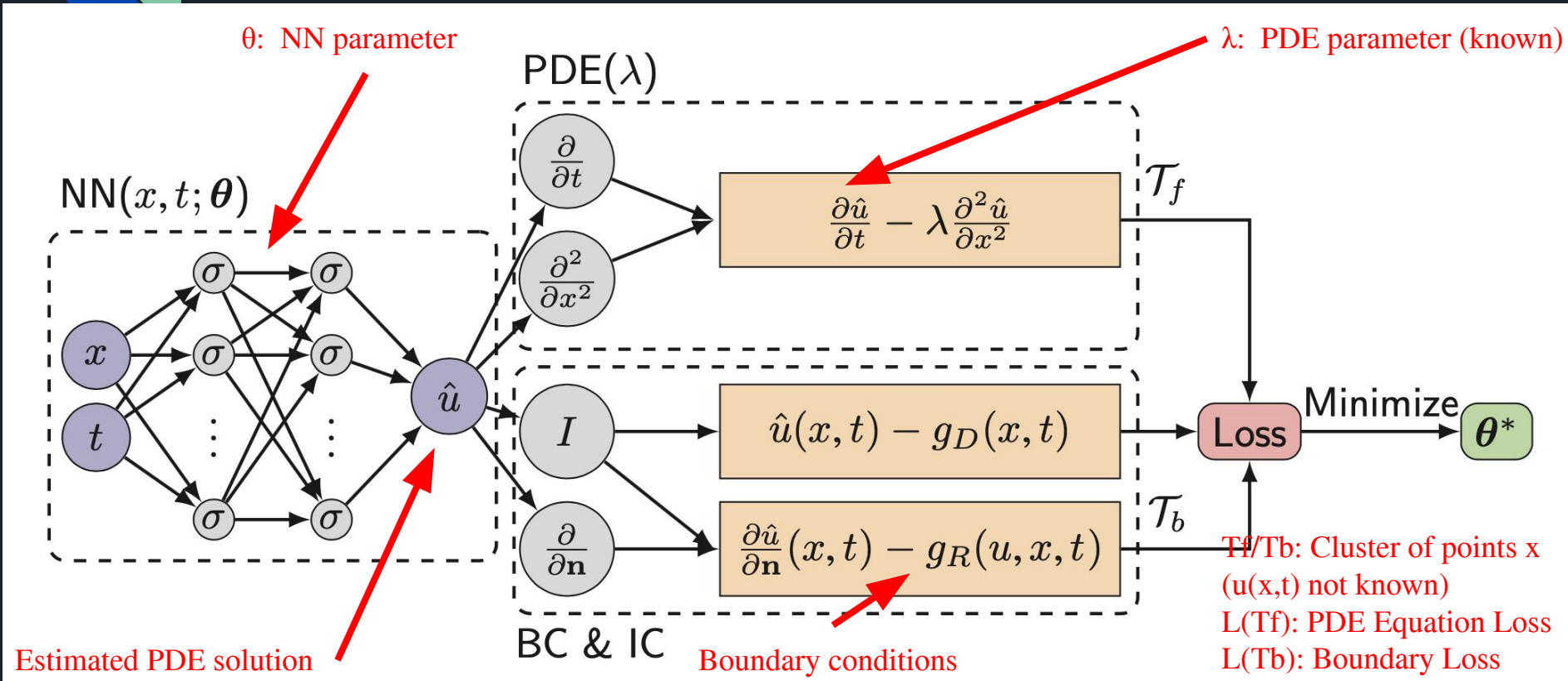
Motivation:

- Traditional mesh-based methods to solve PDEs like FDM and FEM: Suffer from the curse of dimensionality.
- Deep learning method: Mesh free, taking advantage of automatic differentiation.

Ideas:

- Physics-Informed Neural Network (PINN) to embed a PDE into the loss of the neural network.
- Residual-based adaptive refinement (RAR) to improve the training efficiency.

Framework


θ: NN parameter
λ: PDE parameter (known)
PDE(λ)
NN(x, t; θ)
$\frac{\partial}{\partial t}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial \hat{u}}{\partial t} - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}$
$\mathcal{T}_f$
x
t
σ
$\hat{u}$
I
$\hat{u}(x,t) - g_D(x,t)$
Loss
Minimize
θ^*
$\frac{\partial}{\partial \mathbf{n}}$
$\frac{\partial \hat{u}}{\partial \mathbf{n}}(x,t) - g_R(u,x,t)$
$\mathcal{T}_b$
Tf/Tb: Cluster of points x
(u(x,t) not known)
L(Tf): PDE Equation Loss
L(Tb): Boundary Loss
BC & IC
Estimated PDE solution
Boundary conditions

Loss function

The loss is composed by two parts: PDE equation loss and boundary loss:

- Lf(θ, Tf) is the PDE equation loss
  - Tf are the points x that we will calculate the ||f(x,λ)||^2, but u(x,t) for x in Tf is not known.
- Lf(θ, Tb) is the boundary loss
  - Tb are the points x that we will calculate the boundary loss, where we know B(u,x)=0.
- Wf and wb are loss weights.

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = w_f\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) + w_b\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b), \tag{2.2}$$

where

$$\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|}\sum_{\mathbf{x}\in\mathcal{T}_f}\left\| f\left(\mathbf{x}; \frac{\partial\hat{u}}{\partial x_1},\ldots,\frac{\partial\hat{u}}{\partial x_d}; \frac{\partial^2\hat{u}}{\partial x_1\partial x_1},\ldots,\frac{\partial^2\hat{u}}{\partial x_1\partial x_d};\ldots;\boldsymbol{\lambda}\right)\right\|_2^2,$$

$$\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|}\sum_{\mathbf{x}\in\mathcal{T}_b}\|\mathcal{B}(\hat{u},\mathbf{x})\|_2^2,$$

Solving Inverse Problems: Infer λ

This time, some PDE parameters λ are unknown, but we know some I(u,x) = 0 for some points x.

Extra Loss (data loss):

We add another loss term Li(θ,λ,Ti) where Ti are the points we will calculate data loss, where we know some extra information that I(u,x)=0.

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_b) + w_i \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i),$$

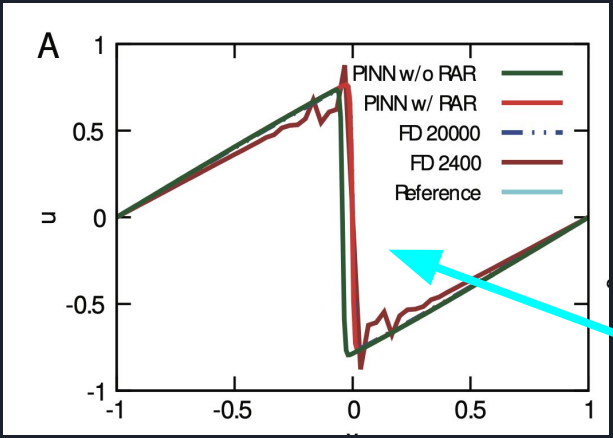
where

$$\mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \|\mathcal{I}(\hat{u}, \mathbf{x})\|_2^2.$$

Residual-Based Adaptive Refinement (RAR)

However, some PDE has steep gradients (discontinuity).

We add more residual points in the locations where PDE equation loss is large until the mean PDE loss is smaller than a pre-set threshold ε0.



$$\mathcal{E}_r = \frac{1}{V}\int_{\Omega}\left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right)\right\| d\mathbf{x} \tag{2.3}$$

is smaller than a threshold $\mathcal{E}_0$, where V is the volume of Ω.

Steep gradients (sudden change of u)

V: space of x:
Eg. 0<x1<1, 0<x2<3, then V=3

Experiment 1: PINN vs SEM

In experiment 1, the author wants to show that PINN solved u(x,t) is close to the SEM solution (Here SEM solution is seen as ground-truth.)

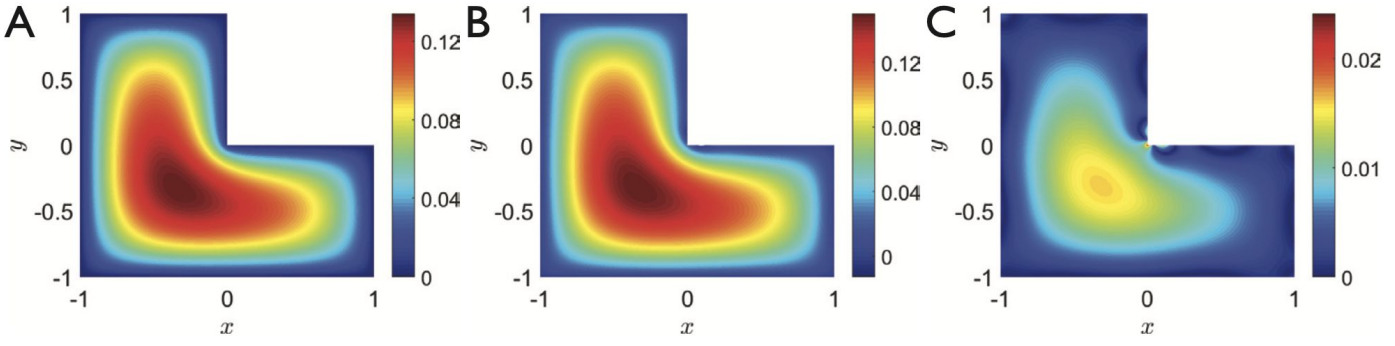
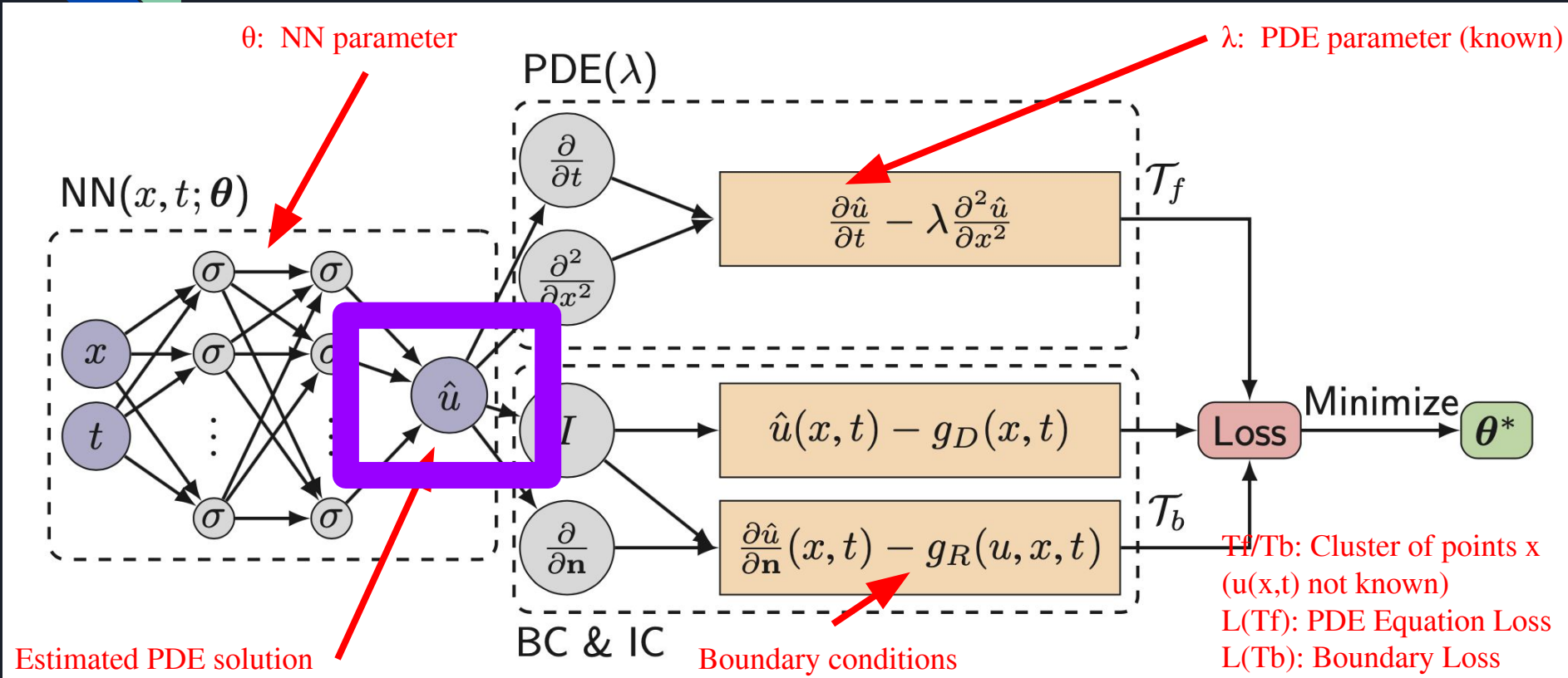


Fig. 7 *Example in subsection* 4.1. *Comparison of the PINN solution with the solution obtained by using the spectral element method (SEM).* (A) *The SEM solution* u_{SEM}, (B) *the PINN solution* u_{NN}, (C) *the absolute error* $|u_{SEM} - u_{NN}|$.

Experiment 1: PINN vs SEM



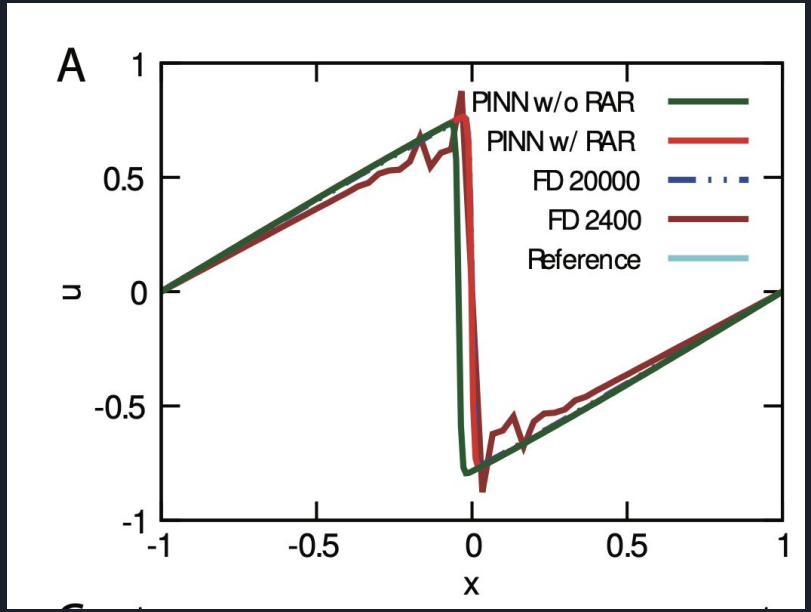
Experiment 2: PINN With RAR

In experiment 2, the author wants to show that PINN w/ RAR fits the ground-truth better than PINN w/o RAR.

- Reference is ground truth.
- FD (Finite difference method) is the baseline.
- FD 20000/2400 means 20000/2400 mesh.

Here, PINN w/ RAR & FD 20000 performs good

(They are superposing the ground-truth reference)



Experiment 3: Solving Inverse Problem

In experiment 3, the author wants to shows that PINN can also used to solve the inverse problem (Infer PDE parameters λ)

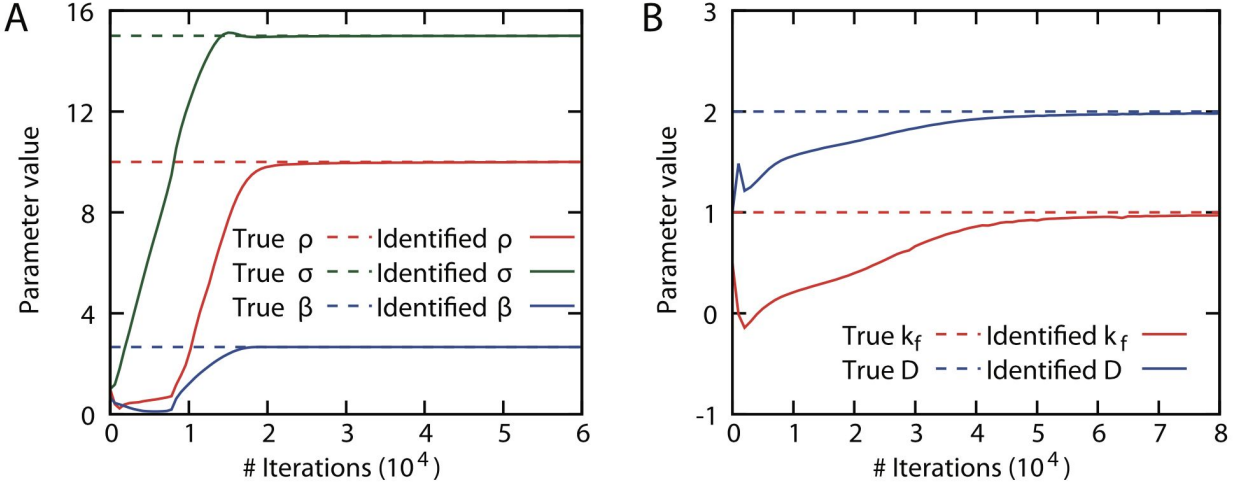
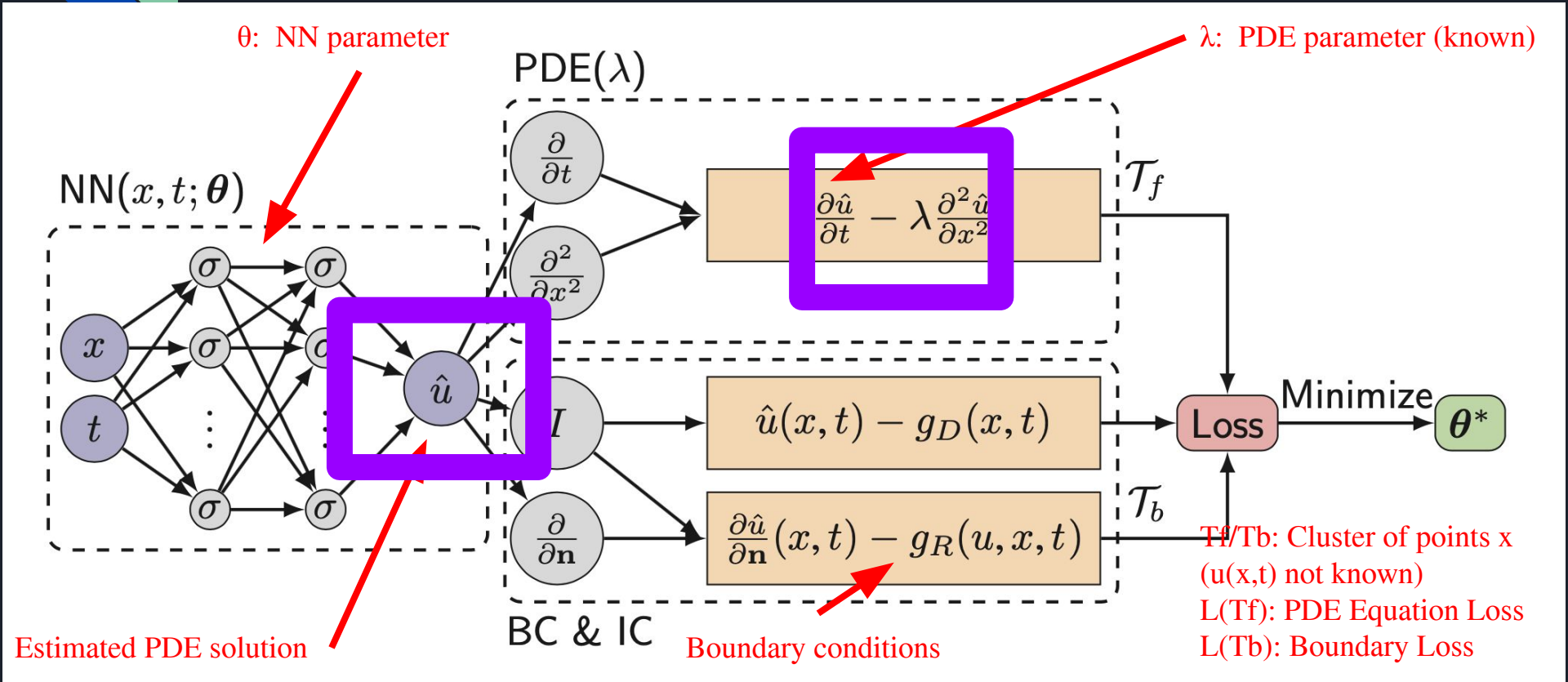


Fig. 9 *Examples in subsections* 4.3 *and* 4.4. *The identified values of* (A) *the Lorenz system and* (B) *diffusion-reaction system converge to the true values during the training process. The parameter values are scaled for plotting.*

Experiment 3: Solving Inverse Problem



Summary

Physics-Informed Neural Network (PINN) to embed a PDE into the loss of the neural network.

- When PDE parameters λ are known, PINN can learn NN parameter θ that approximates $u(x,t)$.
- When PDE parameters λ unknown but we have data information $I(u,x)=0$, PINN can learn both NN parameter θ and infer the PDE parameters λ.

Residual-based adaptive refinement (RAR) to improve the training efficiency.

Outline

1. DeepXDE: A Deep Learning Library of Solving Differential Equations. Lu L, Meng X, Mao Z, et al. SIAM Review 2021.
2. **Solving high-dimensional partial differential equations using deep learning. Han J, Jentzen A, Weinan E. PNAS 2018.**
3. Forecasting Sequential Data Using Consistent Koopman Autoencoders. Azencot O, Erichson N B, Lin V, et al. ICML 2020.

Motivation & Idea

Motivation:

- Traditional methods to solve PDEs suffer from the curse of dimensionality.
- Deep learning methods can handle high dimensional PDEs.

Idea:

- Use neural networks to approximate the gradient of the unknown solution.

Mathematical setup

This paper focus on semilinear parabolic PDEs with the following form:

$$\frac{\partial u}{\partial t}(t,x)+\frac{1}{2}\operatorname{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\operatorname{Hess}_x u)(t,x)\right)+\nabla u(t,x)\cdot\mu(t,x)$$
$$+f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right)=0 \qquad \textbf{[1]}$$

Where t is time, x is in d dimensions. μ and σ are functions.

Specifically, the author focus only on the solution at t=0, x=ε. Besides, Xt in the PDE can be stochastic process with Brownian motion:

$$X_t=\xi+\int_0^t \mu(s,X_s)\,ds+\int_0^t \sigma(s,X_s)\,dW_s.$$

Mathematical setup

Besides, we have two important equations:

$$X_{t_{n+1}} - X_{t_n} \approx \mu(t_n, X_{t_n})\,\Delta t_n + \sigma(t_n, X_{t_n})\,\Delta W_n, \qquad \textbf{[4]}$$

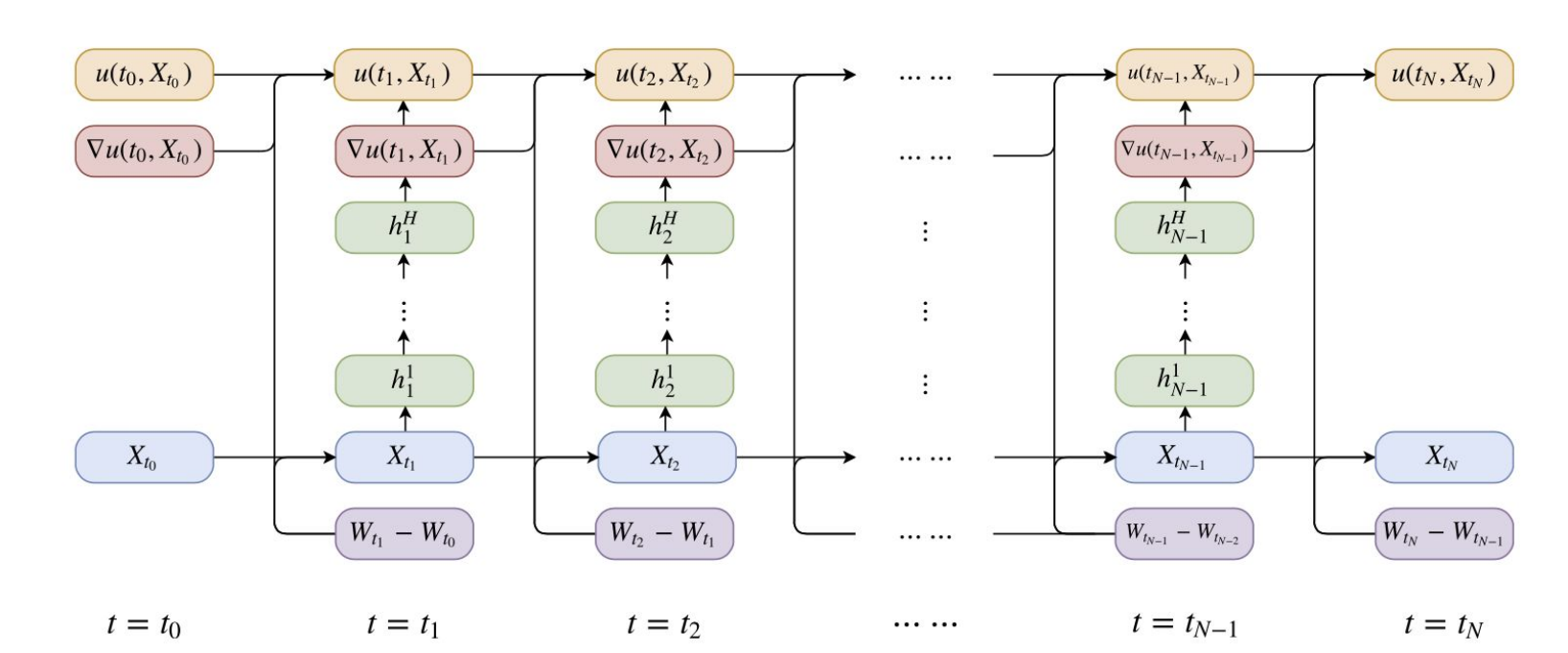
and

$$\begin{aligned} & u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\ \approx & -f\left(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^{\mathrm{T}}(t_n, X_{t_n})\nabla u(t_n, X_{t_n})\right)\Delta t_n \qquad \textbf{[5]} \\ & + [\nabla u(t_n, X_{t_n})]^{\mathrm{T}}\,\sigma(t_n, X_{t_n})\,\Delta W_n, \end{aligned}$$

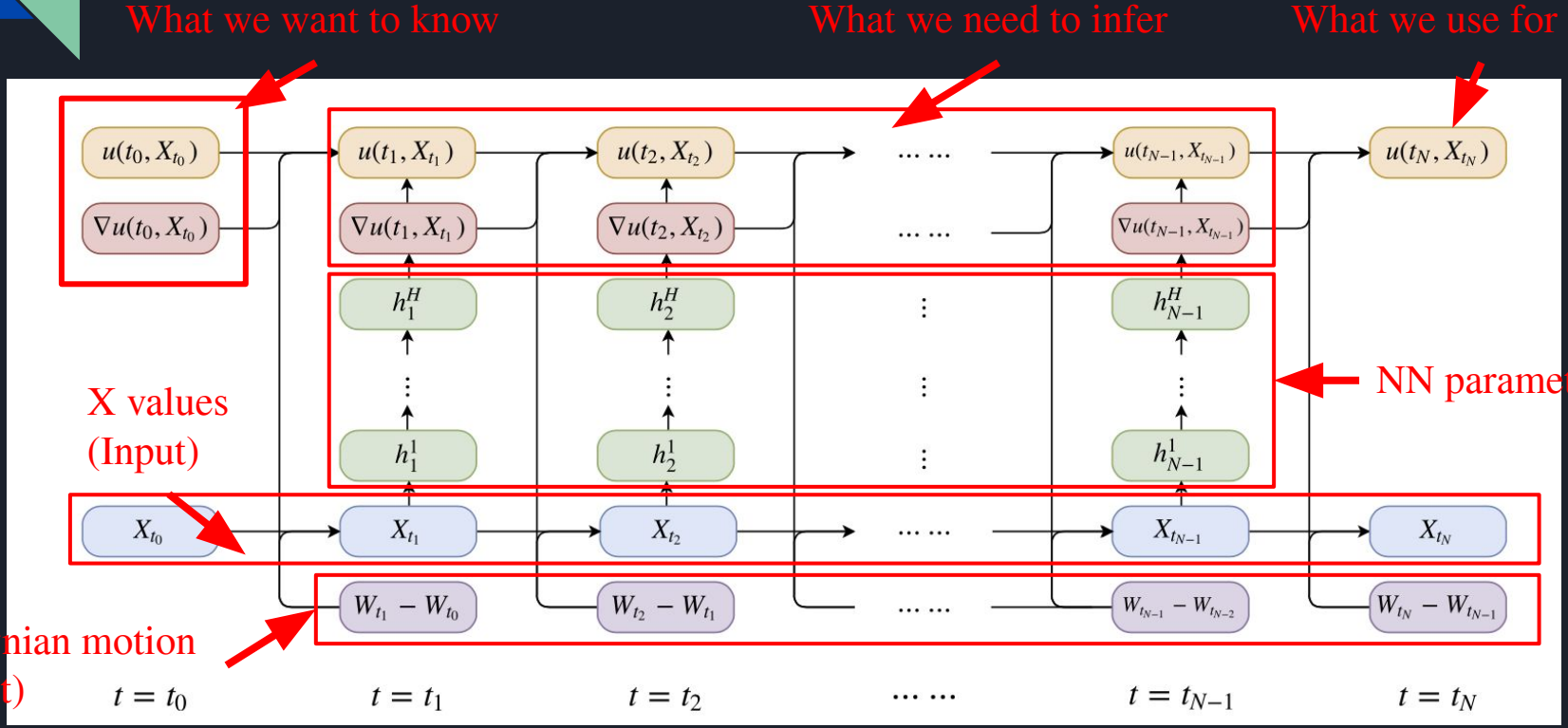
This means given del u(tn, Xtn), Xtn (input) and Wtn (input), the whole system will be known.

Therefore, the author wants to approximate del u(tn, Xtn) using neural networks.

Framework

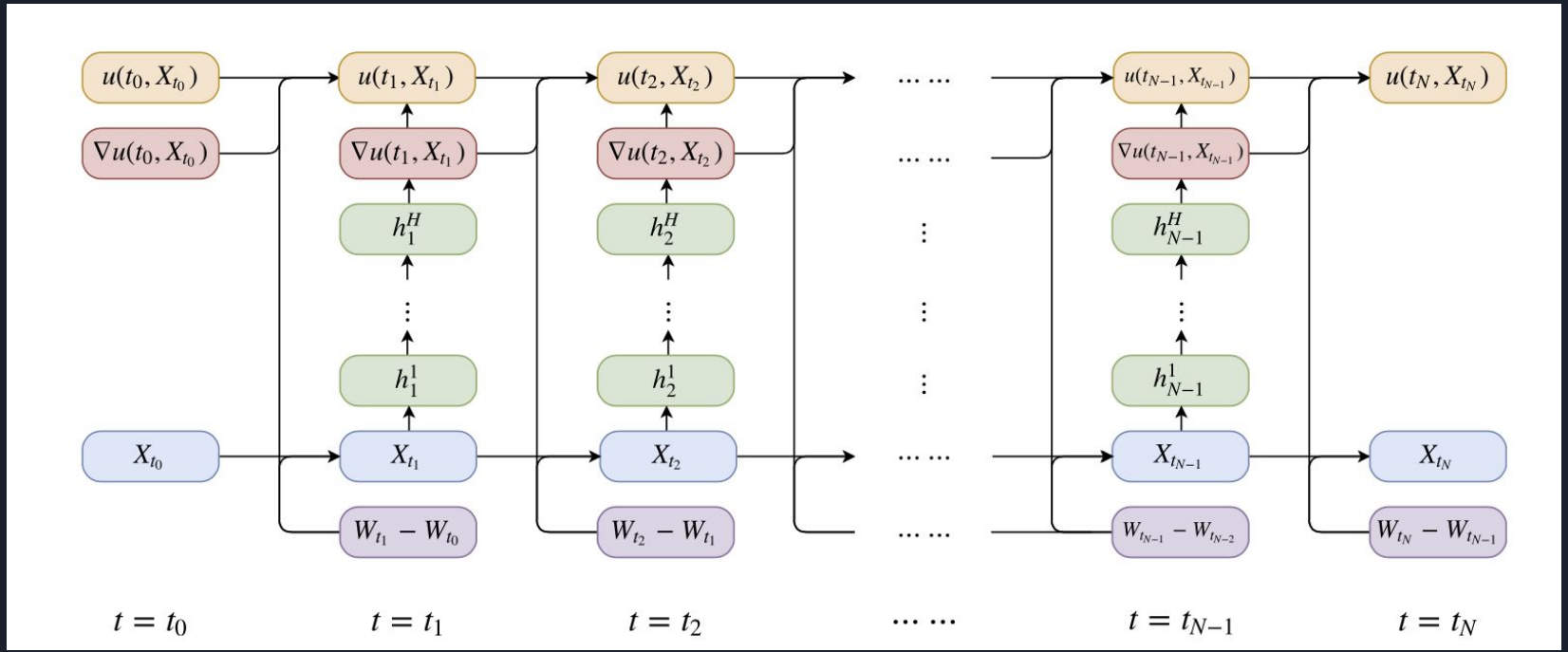

$u(t_0, X_{t_0})$
$\nabla u(t_0, X_{t_0})$
X_{t_0}
$u(t_1, X_{t_1})$
$\nabla u(t_1, X_{t_1})$
h_1^H
h_1^1
X_{t_1}
$W_{t_1} - W_{t_0}$
$u(t_2, X_{t_2})$
$\nabla u(t_2, X_{t_2})$
h_2^H
h_2^1
X_{t_2}
$W_{t_2} - W_{t_1}$
$u(t_{N-1}, X_{t_{N-1}})$
$\nabla u(t_{N-1}, X_{t_{N-1}})$
h_{N-1}^H
h_{N-1}^1
$X_{t_{N-1}}$
$W_{t_{N-1}} - W_{t_{N-2}}$
$u(t_N, X_{t_N})$
X_{t_N}
$W_{t_N} - W_{t_{N-1}}$
$t = t_0$
$t = t_1$
$t = t_2$
$t = t_{N-1}$
$t = t_N$

Framework


What we want to know
What we need to infer
What we use for loss
$u(t_0, X_{t_0})$
$\nabla u(t_0, X_{t_0})$
$u(t_1, X_{t_1})$
$\nabla u(t_1, X_{t_1})$
$u(t_2, X_{t_2})$
$\nabla u(t_2, X_{t_2})$
$u(t_{N-1}, X_{t_{N-1}})$
$\nabla u(t_{N-1}, X_{t_{N-1}})$
$u(t_N, X_{t_N})$
h_1^H
h_2^H
h_{N-1}^H
h_1^1
h_2^1
h_{N-1}^1
NN parameters
X values
(Input)
X_{t_0}
X_{t_1}
X_{t_2}
$X_{t_{N-1}}$
X_{t_N}
$W_{t_1} - W_{t_0}$
$W_{t_2} - W_{t_1}$
$W_{t_{N-1}} - W_{t_{N-2}}$
$W_{t_N} - W_{t_{N-1}}$
Brownian motion
(Input)
$t = t_0$
$t = t_1$
$t = t_2$
$t = t_{N-1}$
$t = t_N$

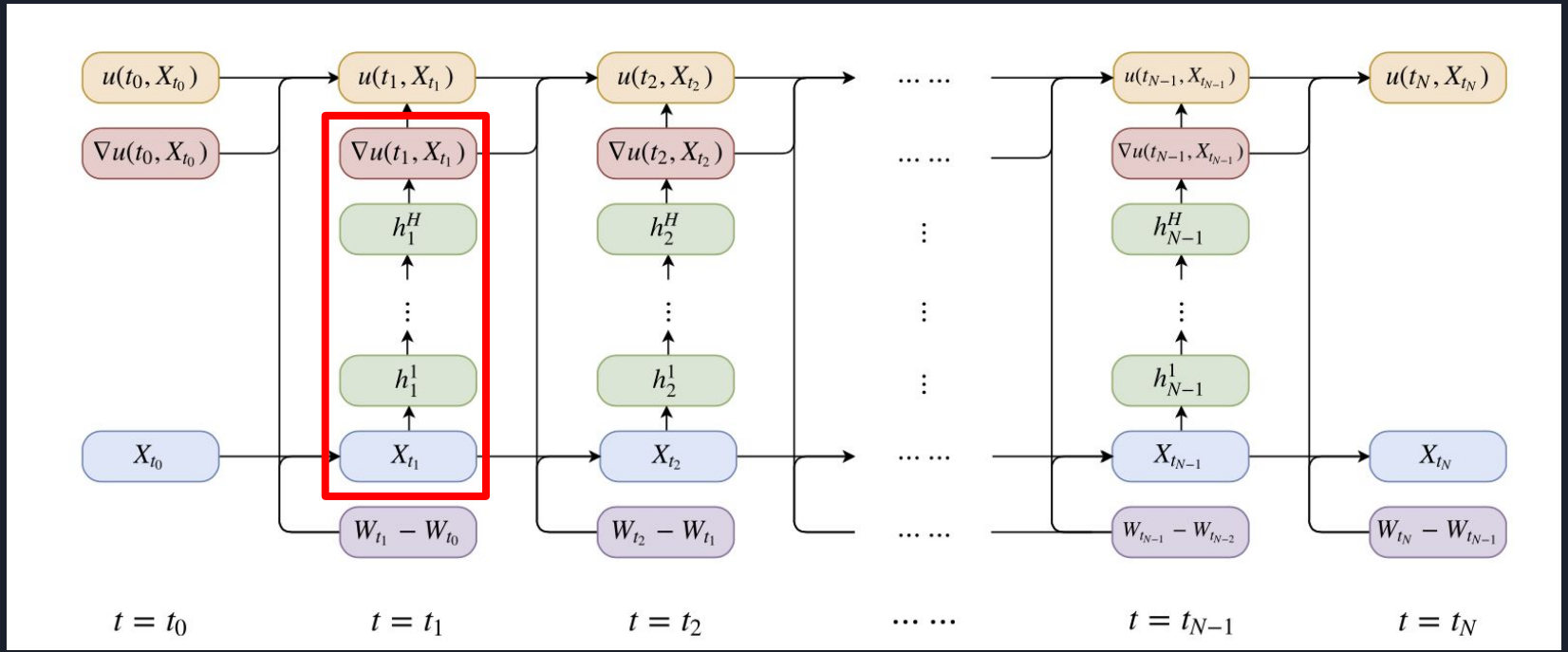
Framework

There exist 3 kinds of connections in the network:



Framework

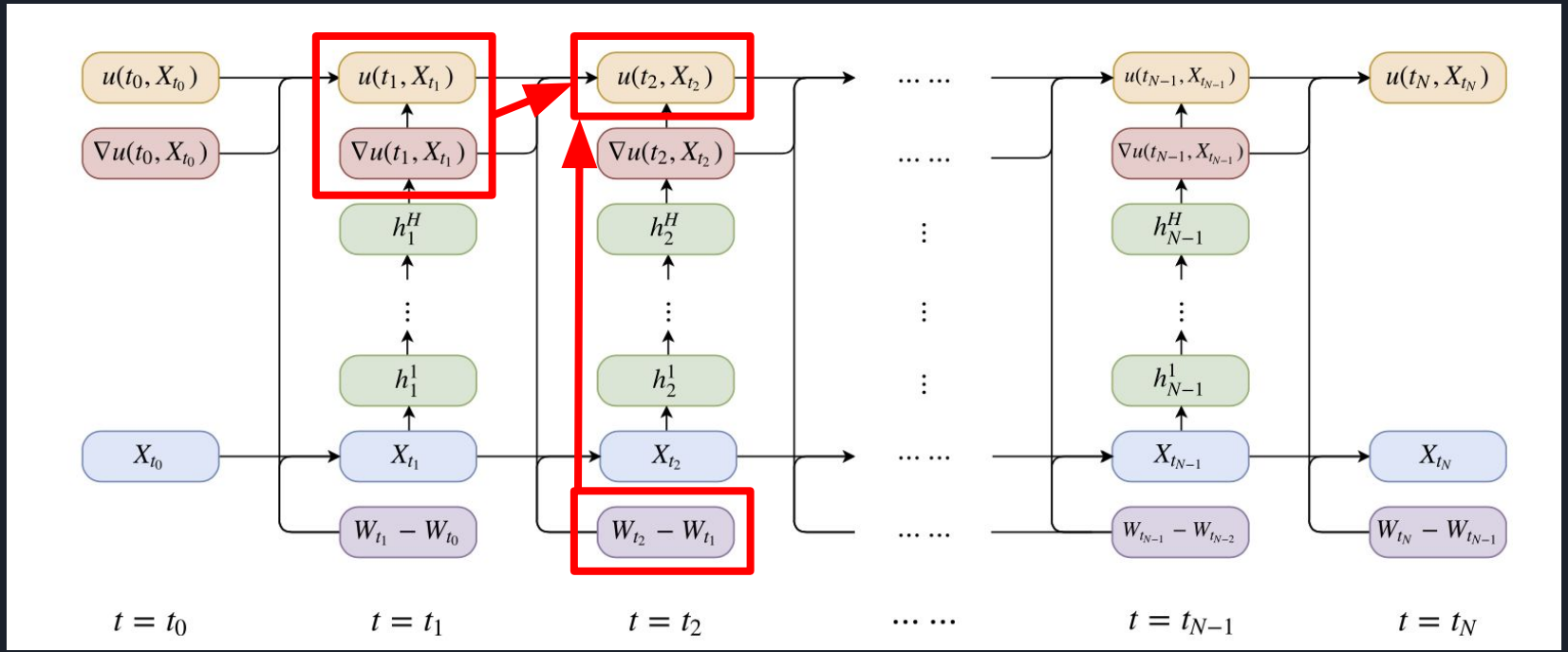
(1) Xtn -> del u(tn,Xtn): Where the NN parameters θn need to be learned.



Framework

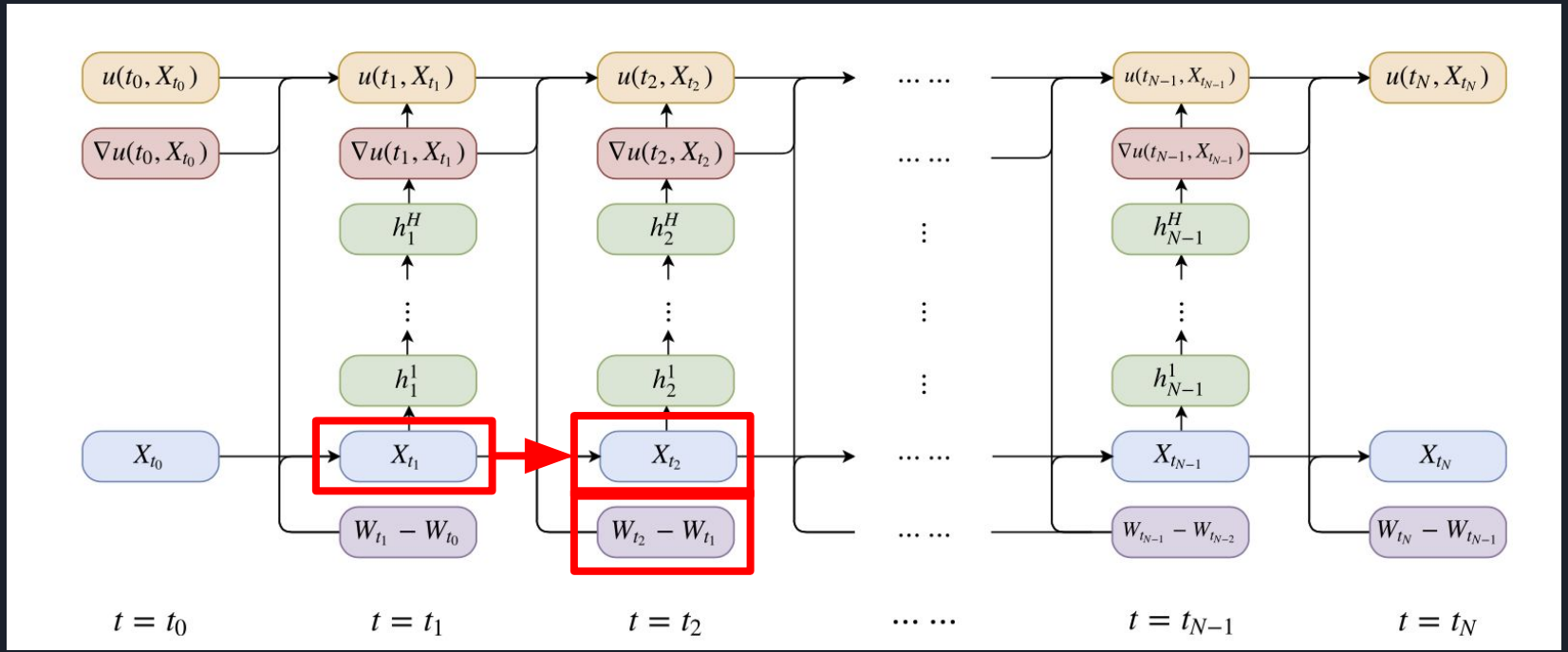
$$
\begin{aligned}
& u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\
\approx & -f\left(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n})\right)\Delta t_n \qquad \textbf{[5]} \\
& + [\nabla u(t_n, X_{t_n})]^{\mathrm{T}}\,\sigma(t_n, X_{t_n})\,\Delta W_n,
\end{aligned}
$$

(2) u(tn,Xtn), del u(tn,Xtn), Wtn+1-Wtn -> u(tn+1,Xtn+1)



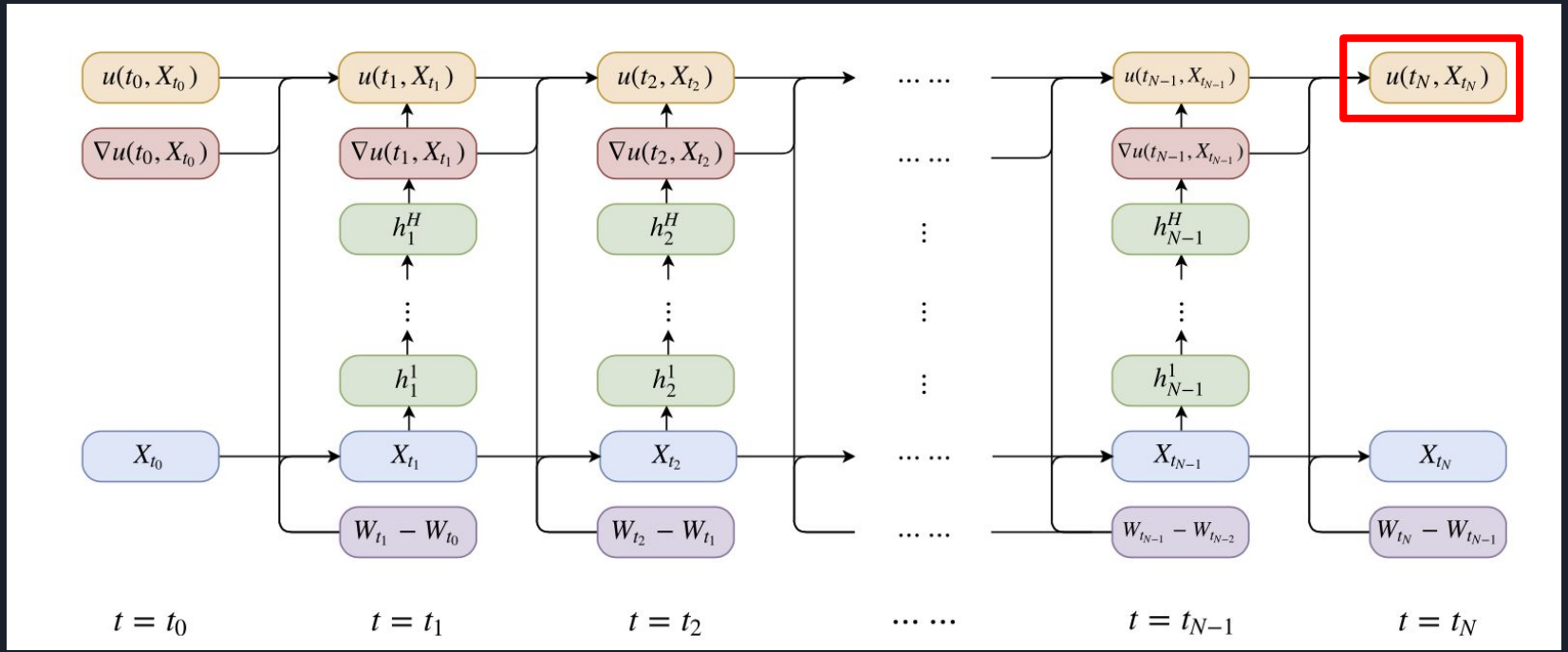
Framework

(3) Xtn, Wtn+1-Wtn -> Xtn+1 (Xt short cut)



Loss

(3) Loss = || u(tN, XtN) - estimated_u(tN, XtN) ||



Outline

1. DeepXDE: A Deep Learning Library of Solving Differential Equations. Lu L, Meng X, Mao Z, et al. SIAM Review 2021.
2. Solving high-dimensional partial differential equations using deep learning. Han J, Jentzen A, Weinan E. PNAS 2018.
3. Forecasting Sequential Data Using Consistent Koopman Autoencoders. Azencot O, Erichson N B, Lin V, et al. ICML 2020.

Motivation & Idea

Motivation:

- Using RNN to forecast sequential data fault to capture the physical structures.
- Physics-based methods like Koopman autoencoder can help to capture such structures.

Idea:

- Use the Koopman autoencoder to analysis (and predict) the sequential data.

Mathematical Background

Given the sequential data: $Z_{k+1} = \varphi(Z_k)$, the Koopman operator satisfies $K_\varphi f(z) = f(\varphi(z))$.

Similarly, we have $Z_{k-1} = \psi(Z_k)$ and $u_\psi f(z) = f(\psi(z))$.

Besides, we have $C = XK_\varphi X^{-1}$ and $D = Xu_\psi X^{-1}$ where X is a finite transformation matrix.

Therefore, our work is to find C, D, and X.

To model X, the author use Xd to approximate X, Xe to approximate X^{-1}

Mathematical Background

Here, the author assume we are given the scalar observations of the system: f_{k+1}(z) = f_k(φ(z)). Therefore, we have f_{k+L}=f_{k}*φ^L.

Besides, we will also have

- f_{k+L}=Xd*C^L*Xe(f_k)
- f_{k-L}=Xd*D^L*Xe(f_k)

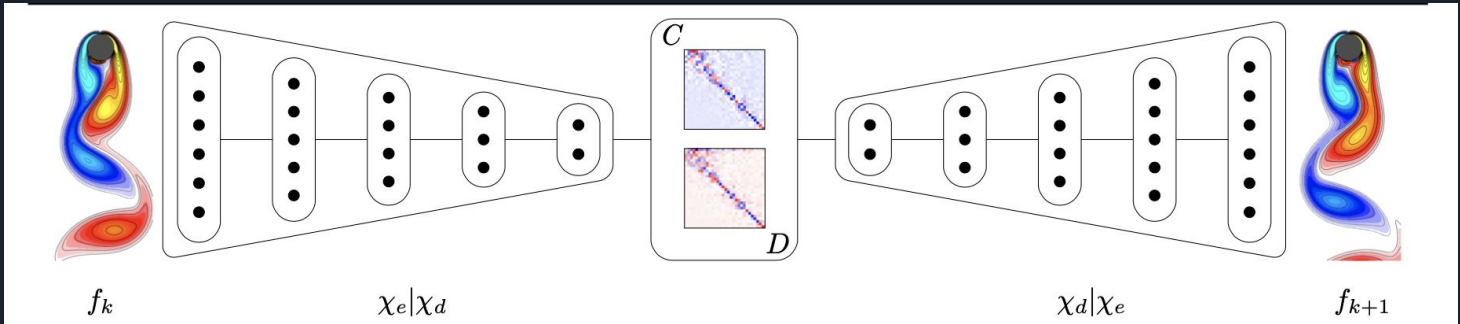


Figure 3. Our network takes observations f_k, and it learns a latent representation of them via an autoencoder architecture χ_e and χ_d. Then, the latent variables are propagated forward and backward in time using the linear layers C and D, respectively. We emphasize that all the above connections are bi-directional and so information can flow freely from left to right or right to left.

Loss 1: Identity Loss

For simplification, here we show the situation L=1.

- f_{k+1}=Xd*C^1*Xe(f_k)
- f_{k-1}=Xd*D^1*Xe(f_k)

The reconstructed version of f should be close to f

$$\tilde{f} = \chi_d \circ \chi_e(f) \,,$$

$$\mathcal{E}_{\text{id}} = \frac{1}{2n} \sum_{k=1}^{n} \|f_k - \tilde{f}_k\|_2^2 \,,$$

Where k is the time in range [1,2,...,n].

Loss 2 & 3: Forward and Backward Loss

- f_{k+1}=Xd*C^1*Xe(f_k)
- f_{k-1}=Xd*D^1*Xe(f_k)

The forecast f (forward and backward) should be close to ground truth.

$$\mathcal{E}_{\text{fwd}} = \frac{1}{2\lambda_s n} \sum_{l=1}^{\lambda_s} \sum_{k=1}^{n} \| f_{k+l} - \hat{f}_{k+l} \|_2^2 ,$$

$$\mathcal{E}_{\text{bwd}} = \frac{1}{2\lambda_s n} \sum_{l=1}^{\lambda_s} \sum_{k=1}^{n} \| f_{k-l} - \check{f}_{k-l} \|_2^2 ,$$

Here, we want to ensure our forecast to be well in λs steps forward and backward.

Loss 4: Forward and Backward Loss

- f_{k+1}=Xd*C^1*Xe(f_k)
- f_{k-1}=Xd*D^1*Xe(f_k)

C * D should be close to identity matrix.

$$\mathcal{E}_{\text{con}} = \sum_{k=1}^{\kappa} \frac{1}{2k} \|D_{k*}C_{*k} - I_k\|_F^2 + \frac{1}{2k} \|C_{k*}D_{*k} - I_k\|_F^2 ,$$

Here, the Dk* are the upper k rows of D and C*k are the leftmost k columns of C. Ik is k-diminsion identity matrix. K is the dimension of C and D.