Lecture 5: Data-Driven Recovery of Equations and Prediction

Haizhao Yang

Department of Mathematics
University of Maryland College Park

2022 Summer Mini Course
Tianyuan Mathematical Center in Central China

Problem Statement

Given observation:

$\boldsymbol{x}(t_1), \boldsymbol{x}(t_2), \ldots, \boldsymbol{x}(t_m)$ in $\mathbb{R}^n$ at time steps $t_1, \ldots, t_m$.

Goal:

- Identify the governing equation that generates data:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t))$$

  or in a discrete form

$$\boldsymbol{x}(t_{m+1}) = \boldsymbol{x}(t_m) + \Delta t * \boldsymbol{f}(\boldsymbol{x}(t_m))$$

- Predict future observation $\boldsymbol{x}(t_{m+1}), \ldots, \boldsymbol{x}(t_{m+s})$

SINDy

Sparse identification of nonlinear dynamics (Brunton et al.)

Data preparation

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \begin{array}{c} \text{state} \longrightarrow \\ \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{bmatrix} \end{array} \Big\downarrow \text{time}$$

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{bmatrix}.$$

SINDy

Dictionary generation

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | & | & | & \\ 1 & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \sin(2\mathbf{X}) & \cos(2\mathbf{X}) & \cdots \\ | & | & | & | & & | & | & | & | & \end{bmatrix}.$$

$$\mathbf{X}^{P_2} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_2^2(t_1) & x_2(t_1)x_3(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_2^2(t_2) & x_2(t_2)x_3(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \cdots & x_2^2(t_m) & x_2(t_m)x_3(t_m) & \cdots & x_n^2(t_m) \end{bmatrix}.$$

Goal: identify sparse coefficients $\Xi := [\xi_1, \xi_2, \ldots, \xi_n]$ such that

$$\boldsymbol{f}(\boldsymbol{X}) = \dot{\boldsymbol{X}} \approx \Theta(\boldsymbol{X})\Xi$$

SINDy

Modeling noisy data

$$\dot{\boldsymbol{X}} = \Theta(\boldsymbol{X})\Xi + \eta \boldsymbol{Z},$$

where

- $\boldsymbol{Z} \sim \mathcal{N}(0, \boldsymbol{I})$ is a matrix of i.i.d. normal Gaussian random variables
- η is the noise magnitude

Sparse modeling

LASSO algorithm:

$$\Xi^* = \operatorname*{argmin}_{\Xi} \|\Theta(\boldsymbol{X})\Xi - \dot{\boldsymbol{X}}\|_2^2 + \lambda\|\Xi\|_1.$$

Extension

Second order derivative in time

Central differencing scheme in time for

$$\ddot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t))$$

i.e.

$$\begin{aligned}\boldsymbol{x}(t+2\Delta t) &= 2\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t) + \Delta t^2 f(\boldsymbol{x}(t)) \\ &\approx 2\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t) + \Delta t^2 \boldsymbol{\Theta}(\boldsymbol{x}(t))\boldsymbol{\Xi}\end{aligned}$$

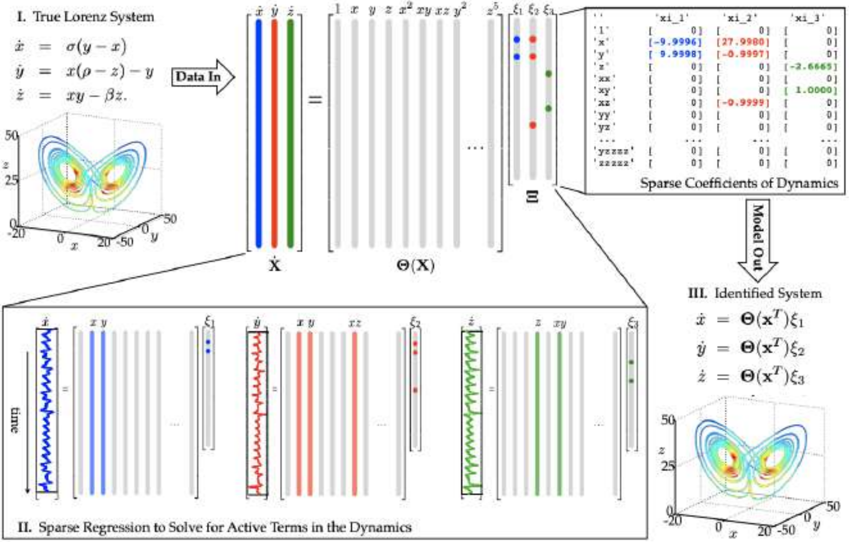
Extension

Discovering time-dependent PDEs

- Goal: identify the unknown f such that
$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \nabla \boldsymbol{x}(t), \nabla^2 \boldsymbol{x}(t))$$
or
$$\ddot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \nabla \boldsymbol{x}(t), \nabla^2 \boldsymbol{x}(t)).$$
- Data preparation:
$$\widehat{\boldsymbol{X}} = [\boldsymbol{X}, \nabla \boldsymbol{X}, \nabla^2 \boldsymbol{X}]$$
- LASSO algorithm:
$$\Xi^* = \underset{\Xi}{\operatorname{argmin}} \|\Theta(\widehat{\boldsymbol{X}})\Xi - \dot{\boldsymbol{X}}\|_2^2 + \lambda \|\Xi\|_1.$$

SINDy

Example and illustration:



SINDy

Prediction:

- $\boldsymbol{x}(t_1), \boldsymbol{x}(t_2), \ldots, \boldsymbol{x}(t_m)$ in $\mathbb{R}^n$ at time steps $t_1, \ldots, t_m$.
- Solve the LASSO problem:

$$\boldsymbol{\Xi}^* = \underset{\boldsymbol{\Xi}}{\operatorname{argmin}} \|\boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi} - \dot{\boldsymbol{X}}\|_2^2 + \lambda \|\boldsymbol{\Xi}\|_1,$$

  then

$$\boldsymbol{f}(\boldsymbol{X}) = \dot{\boldsymbol{X}} \approx \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi}$$

- Use the recurrent relation for prediction $\boldsymbol{x}(t_m) \to \boldsymbol{x}(t_{m+1})$:

$$\begin{aligned} \boldsymbol{x}(t_{m+1}) &= \boldsymbol{x}(t_m) + \Delta t * \boldsymbol{f}(\boldsymbol{x}(t_m)) \\ &\approx \boldsymbol{x}(t_m) + \Delta t * \boldsymbol{\Theta}(\boldsymbol{x}(t_m))\boldsymbol{\Xi} \end{aligned}$$

SINDy

Summary

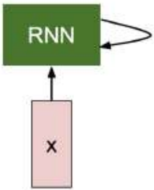
- Advantage: simple to implement and easy to solve
- Disadvantage: need prior knowledge to build dictionary and the dictionary is not powerful

Next: RNN as a model-free method

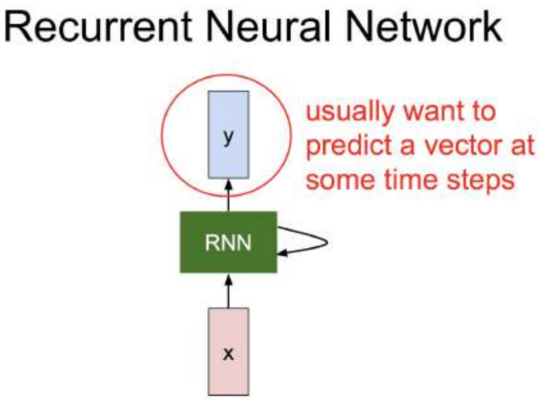
- Advantage: powerful representation
- Disadvantage: difficult to train

Review RNN

Recurrent Neural Network



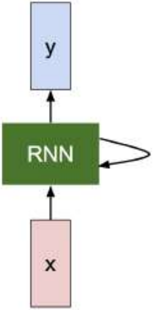
Review RNN


Recurrent Neural Network
y
usually want to
predict a vector at
some time steps
RNN
x

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\text{some function with parameters W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at some time step}})$$

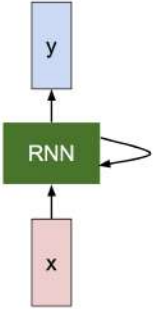


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

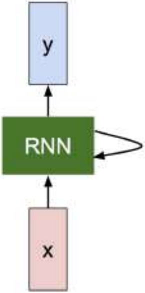
Notice: the same function and the same set of parameters are used at every time step.



Review RNN

(Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



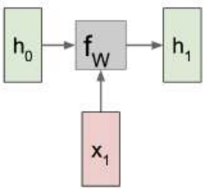
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

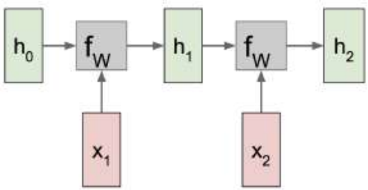
Review RNN

RNN: Computational Graph



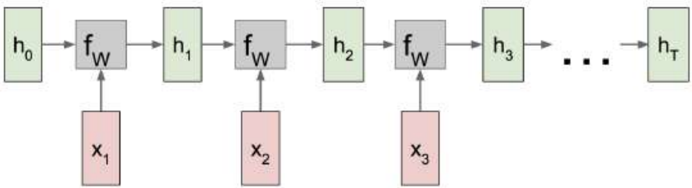
Review RNN

RNN: Computational Graph



Review RNN

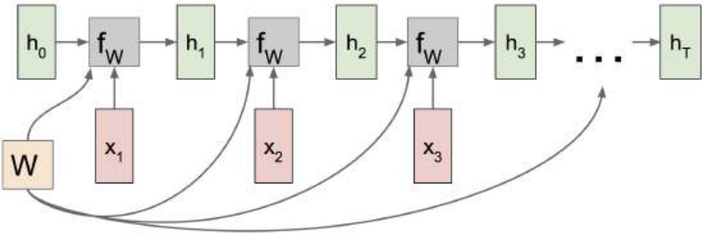
RNN: Computational Graph



Review RNN

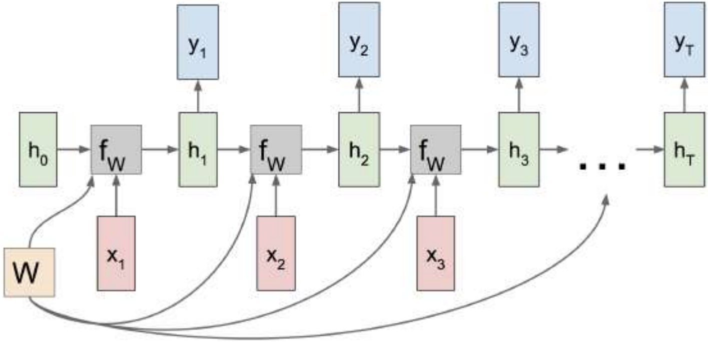
RNN: Computational Graph

Re-use the same weight matrix at every time-step

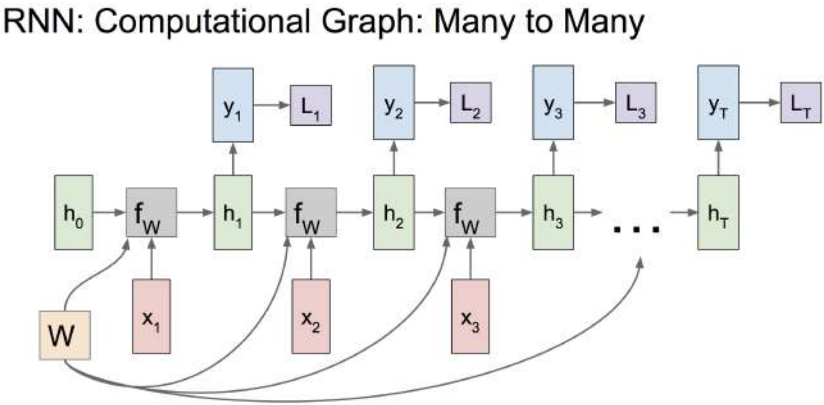


Review RNN

RNN: Computational Graph: Many to Many

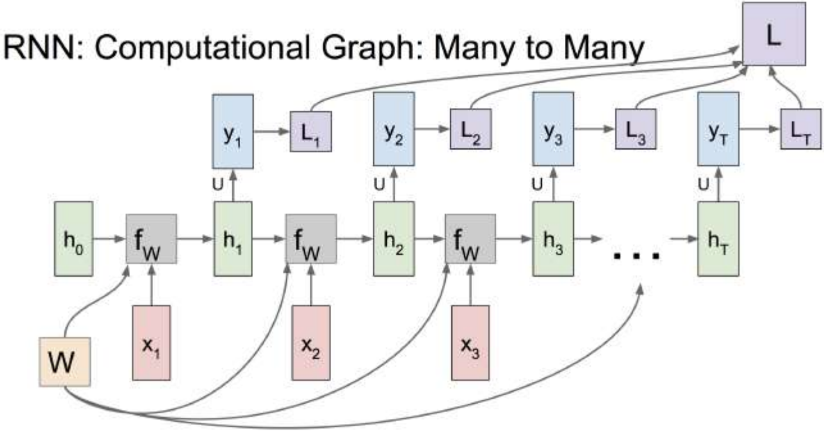


Review RNN



RNN: Computational Graph: Many to Many
y1
L1
y2
L2
y3
L3
yT
LT
h0
fW
h1
fW
h2
fW
h3
...
hT
W
x1
x2
x3

Review RNN



In dynamical system prediction, we want:

$$\big(y_1, y_2, \dots, y_T\big) \approx \big(x_2, x_3, \dots, x_{T+1}\big)$$

or i.e., x_{T+1} is predicted by y_T.

Long Short Term Memory (LSTM)

Vanilla RNN

$$\boxed{h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)}$$

LSTM

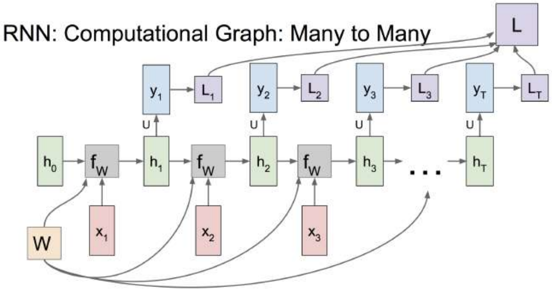
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

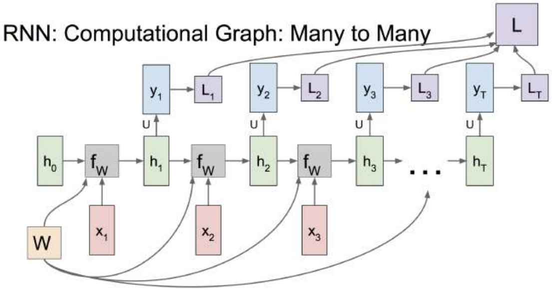
Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation

Prediction using RNN/LSTM



- Given: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m$ in $\mathbb{R}^n$
- Goal: Predict future observation $\boldsymbol{x}_{m+1}, \ldots, \boldsymbol{x}_{m+s}$
- Train an RNN or LSTM such that, given inputs $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m]$, it outputs $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{m-1}] \approx [\boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_m]$.
- Then, $\boldsymbol{x}_{m+1}$ is predicted by $\boldsymbol{y}_m$.
- Repeat this prediction for s times: $\boldsymbol{x}_{m+j}$ is predicted by $\boldsymbol{y}_{m+j-1}$.

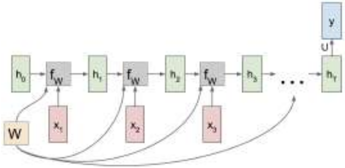
Prediction using RNN



Question: How to train an RNN?

- Requirement: $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{m-1}] \approx [\boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_m]$.
- Loss function: $\mathcal{L}(W, U) := \frac{1}{m-1} \sum_{i=1}^{m-1} (U f_W(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}) - \boldsymbol{x}_{i+1})^2$, where $\boldsymbol{h}_0$ is usually set as 0 and $\boldsymbol{h}_i = f_W(\boldsymbol{x}_i, \boldsymbol{h}_{i-1})$.
- Prediction after training: $\boldsymbol{x}_{i+1} = U f_W(\boldsymbol{x}_i, \boldsymbol{h}_{i-1})$ for $i = m, \ldots, m+s-1$.
- Issue: when m is large, gradient exploding or vanishing.

Prediction using RNN



Question: How to train an RNN efficiently?

- Idea: short-term fitting $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T] \to \boldsymbol{y} \approx \boldsymbol{x}_{T+1}$ with $T \ll m$.
- Predictor with memory T:

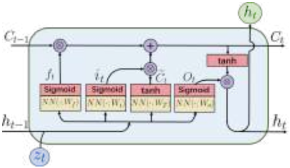
$$\mathcal{P}_T(\boldsymbol{z}_{i,T}; W, U) \approx \boldsymbol{x}_{i+T},$$

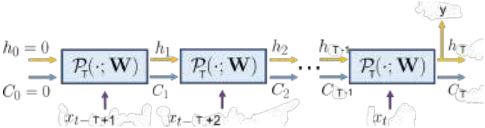
  where $\boldsymbol{z}_{i,T} = [\boldsymbol{x}_i, \ldots, \boldsymbol{x}_{i+T-1}]$ and $\boldsymbol{h}_0 = 0$.
- Loss function:

$$\mathcal{L}(W, U) := \frac{1}{m-T} \sum_{i=1}^{m-T} (\mathcal{P}_T(\boldsymbol{z}_{i,T}; W, U) - \boldsymbol{x}_{i+1})^2.$$

- Prediction after training: $\boldsymbol{x}_{i+1} = \mathcal{P}_T(\boldsymbol{z}_{i-T+1,T}; W, U)$ for $i = m, \ldots, m+s-1$.
- The new loss admits SGD since the sum is separable.

Prediction using LSTM





Question: How to train an LSTM efficiently?

- Idea: short-term fitting $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T] \to \boldsymbol{y} \approx \boldsymbol{x}_{T+1}$ with $T \ll m$.
- Predictor with memory T:

$$\mathcal{P}_T(\boldsymbol{z}_{i,T}; W, U) \approx \boldsymbol{x}_{i+T},$$

  where $\boldsymbol{z}_{i,T} = [\boldsymbol{x}_i, \ldots, \boldsymbol{x}_{i+T-1}]$ and $\boldsymbol{h}_0 = 0$.
- Loss function:

$$\mathcal{L}(W, U) := \frac{1}{m-T} \sum_{i=1}^{m-T} (\mathcal{P}_T(\boldsymbol{z}_{i,T}; W, U) - \boldsymbol{x}_{i+1})^2.$$

- Prediction after training: $\boldsymbol{x}_{i+1} = \mathcal{P}_T(\boldsymbol{z}_{i-T+1,T}; W, U)$ for $i = m, \ldots, m+s-1$.
- The loss admits SGD since the sum is separable.

Prediction using RNN/LSTM

Summary

- Data-driven
- Model-free
- Prediction without the recovery of governing dynamical system

Extension

- Central differencing scheme in time for
  $$\ddot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t))$$
  i.e.
  $$\boldsymbol{x}(t+2\Delta t) = 2\boldsymbol{x}(t+\Delta t) - \boldsymbol{x}(t) + \Delta t^2 f(\boldsymbol{x}(t))$$
- Also work for the prediction of the solution of time-dependent PDEs
- Can use deeper RNN/LSTM

Summary

SINDy

- Advantage: simple to implement and easy to solve
- Disadvantage: model-based; need prior knowledge to build dictionary and the dictionary is not powerful

RNN/LSTM

- Advantage: model-free method and powerful representation; cheap computation
- Disadvantage: difficult to train
- e.g., J. Harlim, S. W. Jiang, S. Liang, H. Yang. Machine Learning for Prediction with Missing Dynamics. Journal of Computational Physics, 2020

Question:

- Can we combine model free and model based methods?
- What's the benefit if we combine them?

PDE-Net (Long et al., JCP, 2019)

Assuming periodic boundary conditions in space

Finite difference for ∂_y is one matvec

$$\begin{pmatrix} \partial_y u(y_1) \\ \partial_y u(y_2) \\ \vdots \\ \partial_y u(y_m) \end{pmatrix} \approx \frac{1}{\Delta y} \begin{pmatrix} 1 & 0 & 0 & \dots & -1 \\ -1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} u(y_1) \\ u(y_2) \\ \vdots \\ u(y_m) \end{pmatrix}$$

Finite difference for ∂_y^2 is one matvec

$$\begin{pmatrix} \partial_y^2 u(y_1) \\ \partial_y^2 u(y_2) \\ \vdots \\ \partial_y^2 u(y_m) \end{pmatrix} \approx \frac{1}{\Delta y^2} \begin{pmatrix} -2 & 1 & 0 & \dots & 1 \\ 1 & -2 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & -2 \end{pmatrix} \begin{pmatrix} u(y_1) \\ u(y_2) \\ \vdots \\ u(y_m) \end{pmatrix}$$

Any linear operator on $u(y)$ is a matvec to $u(\boldsymbol{y})$

e.g., there exists $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ such that $\partial_y u(\boldsymbol{y}) + \partial_y^3 u(\boldsymbol{y}) = \boldsymbol{A} u(\boldsymbol{y})$.

PDE-Net (Long et al., JCP, 2019)

Assuming periodic boundary conditions in space

Any linear operator on $u(y)$ is a matvec to $u(\boldsymbol{y})$
e.g., there exists $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ such that $\partial_y u(\boldsymbol{y}) + \partial_y^3 u(\boldsymbol{y}) = \boldsymbol{A} u(\boldsymbol{y})$.

What about nonlinear operators?
e.g., $(\partial_y u(y))^2$?

PDE-Net (Long et al., JCP, 2019)

Deep neural network

Function composition in the parametrization:

$$y = h(y;\theta) := T \circ \phi(y) := T \circ h^{(L)} \circ h^{(L-1)} \circ \cdots \circ h^{(1)}(y)$$

where

- $h^{(i)}(y) = \sigma({W^{(i)}}^T y + b^{(i)})$;
- $T(y) = V^T y$;
- $\theta = (W^{(1)}, \cdots, W^{(L)}, b^{(1)}, \cdots, b^{(L)}, V)$.

Theorem

For any $g(y) \in C([0,1]^d)$ and any $\varepsilon > 0$, there exists an NN $h(y;\theta)$ such that $|g(y) - h(y;\theta)| \leq \varepsilon$.

What about nonlinear operators?

- $y^2 \approx h(y;\theta)$ for $y \in \mathbb{R}$
- $A * u(\boldsymbol{y}) \approx \partial_y u(\boldsymbol{y})$
- $(\partial_y u(\boldsymbol{y}))^2 \approx h(A * u(\boldsymbol{y});\theta)$ and treat $A * u(\boldsymbol{y})$ as a symbolic scalar in the NN $h(y;\theta)$

PDE-Net (Long et al., JCP, 2019)

What about nonlinear operators?

- $y^2 \approx h(y;\theta)$ for $y \in \mathbb{R}$
- $A * u(\boldsymbol{y}) \approx \partial_y u(\boldsymbol{y})$
- $(\partial_y u(\boldsymbol{y}))^2 \approx h(A * u(\boldsymbol{y});\theta)$ and treat $A * u(\boldsymbol{y})$ as a symbolic scalar in the NN $h(y;\theta)$

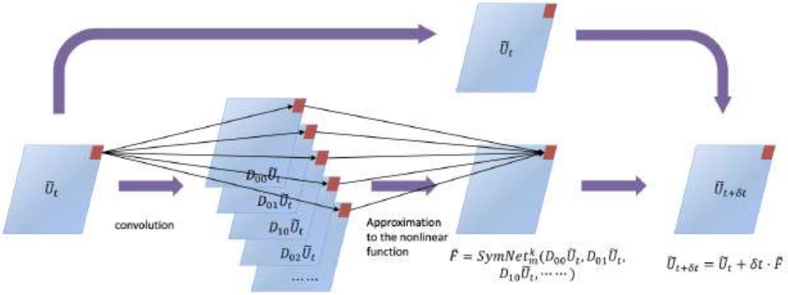
Symbolic NN

For any nonlinear operator $\mathcal{F}$ on $u(y)$, there exists an NN $h(y;\theta)$ such that

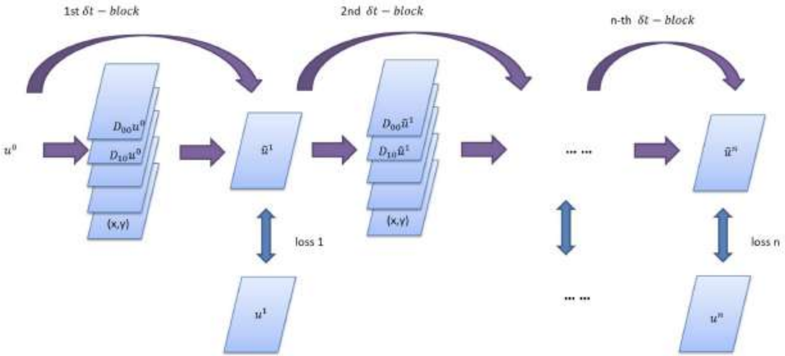
$$\mathcal{F}(u(\boldsymbol{y})) \approx h(u(\boldsymbol{y});\theta).$$

e.g., $(\partial_y u(\boldsymbol{y}))^3 + (\partial_y^2 u(\boldsymbol{y}))^2 \approx h(u(\boldsymbol{y});\theta)$ and treat $u(\boldsymbol{y})$ as a symbolic scalar in the NN $h(y;\theta)$

PDE-Net (Long et al., JCP, 2019)



PDE-Net (Long et al., JCP, 2019)



PDE-Net (Long et al., JCP, 2019)

Discussion

- Incorporate physics in the NN, e.g., convolution to implement derivatives
- Model based for linear operators and model-free for nonlinear operators
- Use symbolic NN (less parameters) instead of normal NN (more parameters)
- Can recover the governing equation since SymNet is explicit, e.g., $F(u_t, u_x) \approx SymNet(u_t, u_x)$.