UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/SIOC 228 Machine Learning for Physical Applications
Lecture 11: Physics Informed Machine Learning II

Yuanyuan Shi

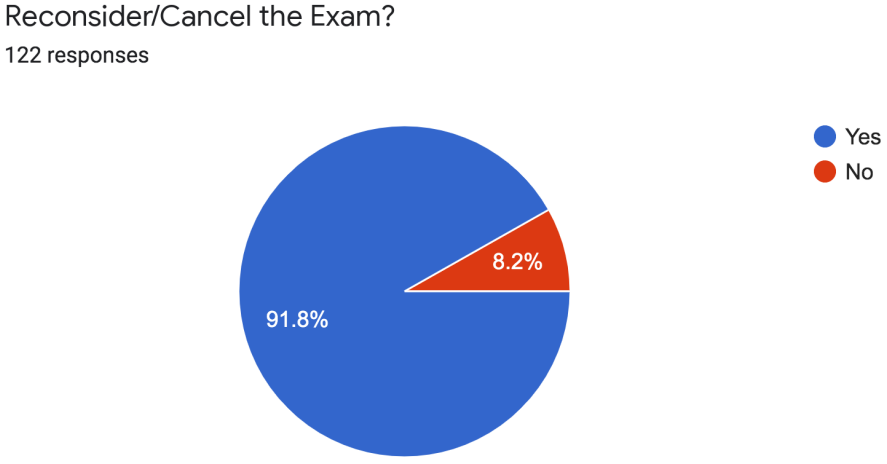
Assistant Professor, ECE

University of California, San Diego

Reference: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equation: https://www.sciencedirect.com/science/article/pii/S0021999118307125
A First Course in the Numerical Analysis of Differential Equations, https://www.uaar.edu.pk/fs/books/28.pdf

Logistics

UC San Diego



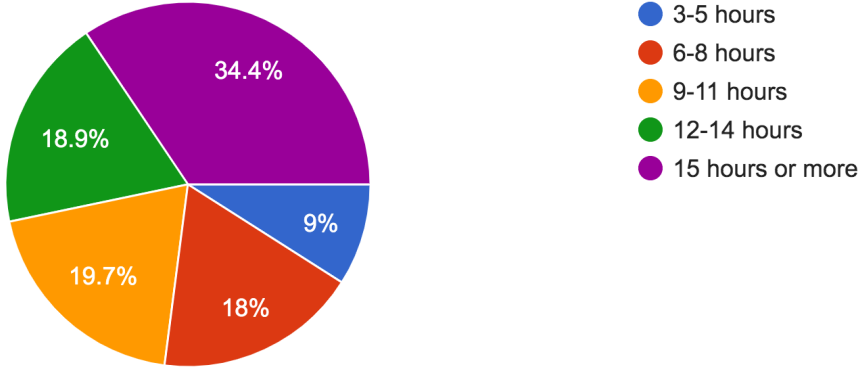
We will cancel the exam on May 17th, and have regular lecture for that day. Final grade will only depend on 40% Homework, 40% Project and 5% Participation → scale them to 100% as the final grades

Logistics

UC San Diego

How many hours did you spend on HW1 Written part?

122 responses

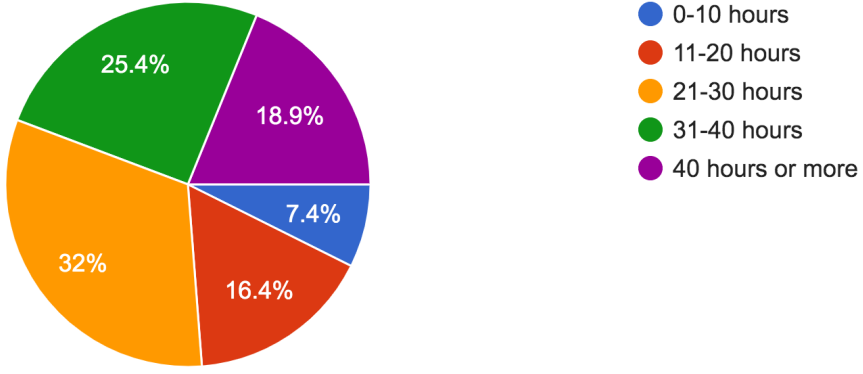


Logistics

UC San Diego

How many hours did you spend on HW1 Coding part?

122 responses

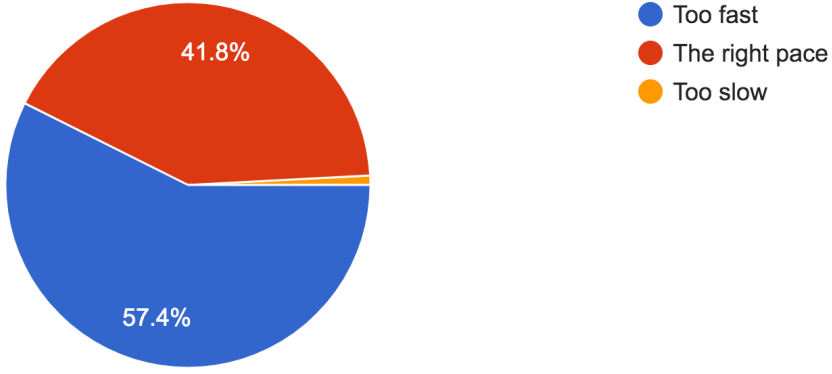


Logistics

UC San Diego

The speed with which we cover the material covered in lecture is

122 responses



Logistics

UC San Diego

- We'll try to improve based on the feedbacks.
- **Please respect to the rules and final deadlines, for fairness of the entire class.**
- We really appreciate all the helpful suggestions. Keep in mind we have a **good diversity** in this class. Everyone brings in different ideas and experiences, everyone potentially also has different learning objectives and goals. Please be kind and patient, and try to understand each other.

Review: Pipeline of Operator Learning

UC San Diego

- Heat equation: $u_t = u_{xx}$ (The change in time is equal to 2nd-order difference in space.)
- Numerical solver to generate training data under different initial conditions $u(0, x) \longrightarrow u(T, x)$

Supervised learning

Get a dataset {(a , u)} (existing solvers or experiments)

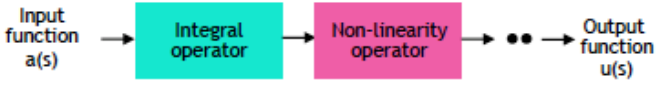
- Initialize NN, F: a->u
- For N epoch, For batches of data pairs (a, u)
- Compute the prediction F(a)
- Minimize ||F(a)-u||

Review: Operator learning

UC San Diego

- Operator learning

$$F:\ a(s) \rightarrow u(s)$$
$$(R^n \rightarrow R^m) \rightarrow (R^n \rightarrow R^m)$$

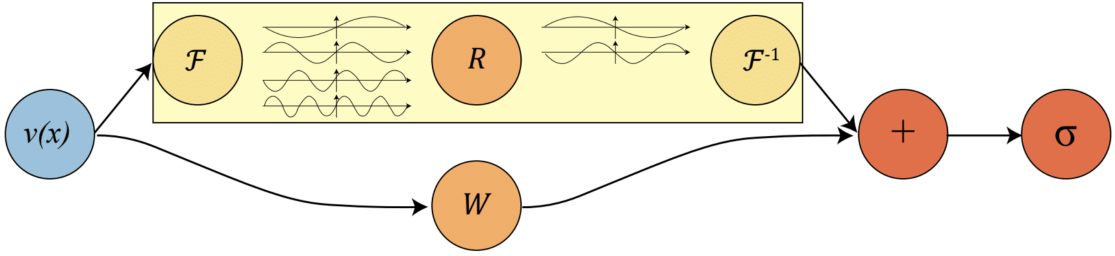


$$v_0(s) = P(x(s), s)$$
$$v_{l+1}(s) = \sigma\left(W_l v_l(s) + \int_D \kappa_l(s,z) v_l(z)\, dz + b_l(s)\right), \quad l = 0, \ldots, L-1$$
$$y(s) = Q(v_L(s))$$
$$P : \mathbb{R}^{m+n} \rightarrow \mathbb{R}^{d_0}, \quad W_l \in \mathbb{R}^{d_{l+1} \times d_l}, \quad \kappa_l : \mathbb{R}^{2n} \rightarrow \mathbb{R}^{d_{l+1} \times d_l}, \quad b_l : \mathbb{R}^n \rightarrow \mathbb{R}^{d_{l+1}}, \quad Q : \mathbb{R}^L \rightarrow \mathbb{R}^r$$

Review: Operator learning

UC San Diego

- **Operator learning**
  - Graph-based Neural Operator
  - Deep Operator Network (DeepONet)
  - Fourier Neural Operator



https://arxiv.org/abs/2006.09535

Supervised learning

Get a dataset {(a , u)} (existing solvers or experiments)

- Initialize NN, F: a->u
- For N epoch, For batches of data pairs (a, u)
- Compute the prediction F(a)
- Minimize ||F(a)-u||

What if there is no existing solvers or data available?

Outline

UC San Diego

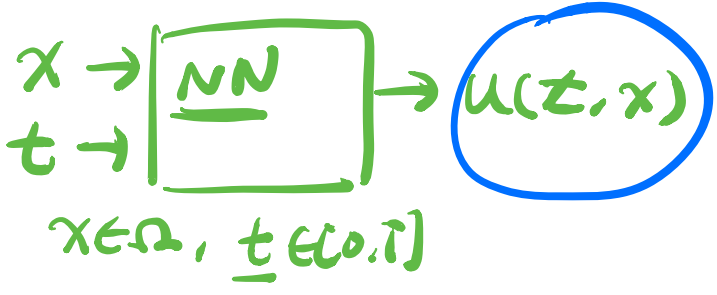
- Continuous Time Physics Informed Neural Network (PINN)
- Discrete Time PINN
- Extensions
  - Data driven parameter discovery via PINN
  - PINN + Neural Operator (PINO)
  - Applications: ML-Accelerated PDE Observers

- Suppose we want to solve a partial differential equations of the general form

$$u_t = -\mathcal{N}[u]$$

where $x \in \Omega$, $t \in [0, T]$. $u(t, x)$ denotes the solution. $\mathcal{N}[u]$ is a nonlinear differential operator.

Example: 1D heat equation $u_t = u_{xx}$ $\mathcal{N}[u] = -u_{xx}$

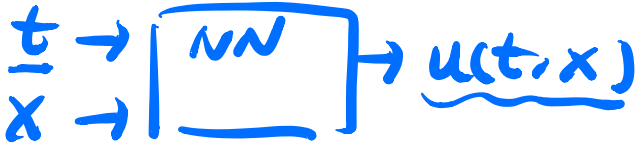


Any requirement for the solution u(t, x)?

- We define $f(t, x)$ to be given by

$$f := u_t + \mathcal{N}[u]$$

- We appromate the PDE solution $u(t, x)$ by a deep neural network.
- Thus, we will have $f(t, x)$ a physics informed neural network (PINN). This neural network can be defined by computing the involved gradients via backpropagation.



PINN (Physics laws on NN)

$f(t,x) = u_t - u_{xx} \equiv 0$ $\forall x \in \Omega$, $t \in [0,T]$

heat equation

Example: Burgers' Equation

UC San Diego

- As a more complete example, we consider the 1D Burgers' Equation.

X

$$u_t + uu_x = \frac{0.01}{\pi} u_{xx}, x \in [-1,1], t \in [0,1]$$

$$u(0,x) = -\sin(\pi x)$$

$$u(t,-1) = u(t,1) = 0$$

✓

- A fundamental partial differential equation occurring in various areas of applied mathematics, such as fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. https://en.wikipedia.org/wiki/Burgers%27_equation

Example: Burgers' Equation

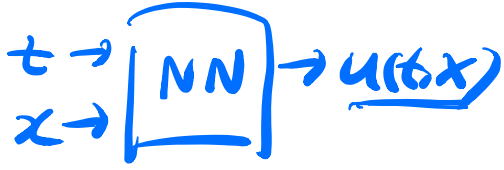
UC San Diego

- As a more complete example, we consider the 1D Burgers' Equation.

$$u_t + uu_x = \frac{0.01}{\pi} u_{xx}, x \in [-1,1], t \in [0,1]$$

$$u(0,x) = -\sin(\pi x)$$

$$u(t,-1) = u(t,1) = 0$$



- Define Solution $u(t,x)$ as a neural network
- Define a PINN $f(t,x)$

$$f(t,x) := u_t + u \cdot u_x - \frac{0.01}{2} u_{xx}$$

Example: PINN for Burgers' Equation

UC San Diego

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the physics informed neural network $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)
    u_x = tf.gradients(u, x)
    u_xx = tf.gradients(u_x, x)
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

Example: PINN for Burgers' Equation

UC San Diego

- The shared parameters between the neural network $u(t, x)$ and the physics informed neural network $f(t, x)$ can be learned by minimizing the mean squared error loss

Question:

- **What should $f(t, x)$ be for physical system $u_t + uu_x = \frac{0.01}{\pi} u_{xx}$?**

$f(t,x)=0$, $\forall x \in [-1,1]$, $t \in [0,1]$

- **Any other constraints?**

$u(0,x) = -\sin(\pi x)$ ← initial

$u(t,-1) = u(t,1)$ ← boundary condition

$\{t^i, x^i, u^i\}$

Example: PINN for Burgers' Equation

UC San Diego

- The shared parameters between the neural network $u(t, x)$ and the physics informed neural network $f(t, x)$ can be learned by minimizing the mean squared error loss

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u^i(t^i, x^i) - u^i|^2$$

$$MSE = MSE_f + MSE_u$$

physics law

initial & boundary conditions

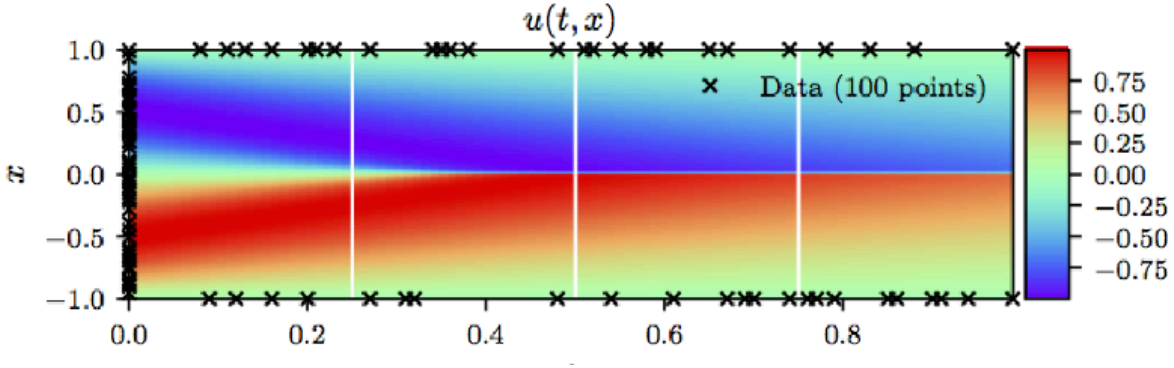
$\min MSE$
weights, biases

$x \rightarrow$
$t \rightarrow$ $\tilde{u}(W, b)$ $\rightarrow u(t, x)$

Example: PINN for Burgers' Equation

UC San Diego

- The shared parameters between the neural network $u(t, x)$ and the physics informed neural network $f(t, x)$ can be learned by minimizing the mean squared error loss



+ 10,000 (x, t) points for enforcing the physics law $f(t, x) = 0$

PINNs v.s. Classic PDE Solvers

UC San Diego

PINNs is like a PDE solver

- Give a PDE instance: with certain initial and boundary condition → Solution for that instance
- Compared to classic numerical solvers (e.g. FDM, FEM)
  - ✓ Simple and flexible
  - ✓ Complex geometry, high dim, etc
  - X optimization can be tricky
  - X no guarantee for convergence

less accurate than classic solvers

PINNs (solve) v.s. Operator Learning (learn)

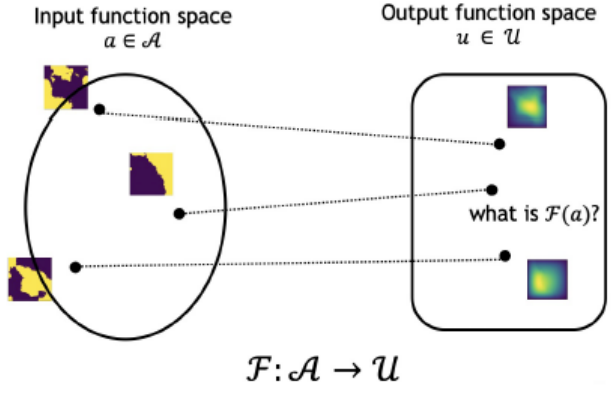
UC San Diego

PINNs is like a PDE solver

- Give a PDE instance: with certain initial and boundary condition →
- Compared to classic numerical solvers (e.g. FDM, FEM)
  - ✓ Simple and flexible
  - ✓ Complex geometry, high dim, etc
  - X .
  - X .

Operator Learning

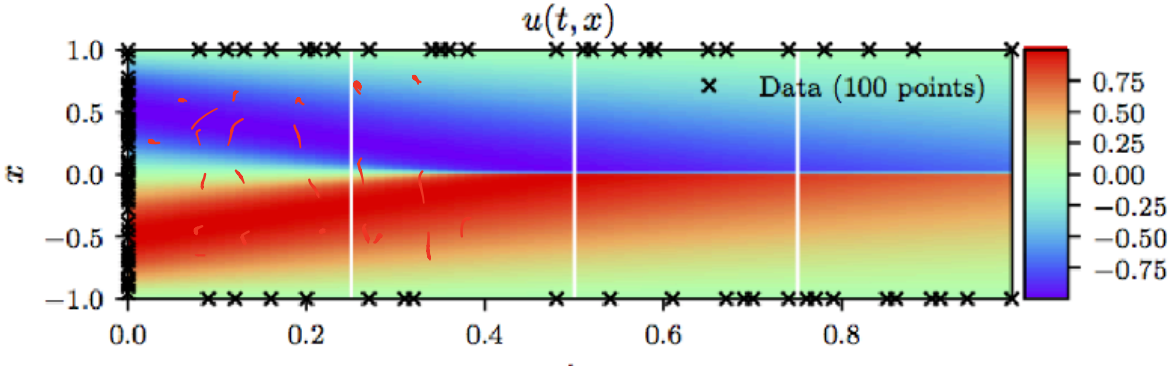
learn the solution operator $\mathcal{F}$
interpolate u in the function space.



but requires existing solvers or experiment data that contains solutions under different initial/boundary conditions or PDE parameters

What's Next…

UC San Diego



+ 10,000 (x,t) points for enforcing the physics law $f(t,x)=0$

- **A large number of points** in order to enforce physics constraints in the **entire spatio-temporal domain**
- May introduce a severe bottleneck in higher dimensional problems, as the total number of points needed to globally enforce a physics informed constrain will increase exponentially.
- **What can we do to improve this?**

Outline

UC San Diego

- Continuous Time Physics Informed Neural Network (PINN)
- **Discrete Time PINN**
- Extensions
  - Data driven parameter discovery via PINN
  - PINN + Neural Operator (PINO)
  - Applications: ML-Accelerated PDE Observers

Numerical Solutions of PDEs

UC San Diego

- The discrete time PINN models are built upon the **numerical methods (i.e., Runge-Kutta Method)**
- Numerical methods to approximate the solution of the following ODEs

$$\frac{dy}{dt} = f(t, y), y(t^0) = y^0$$

and you want to solve for $y(t), t \geq t^0$

- We know the value of y at a single point t=to
- Given any t≥to, y, we can tell the slope from the differential equation

Can we predict the value of y at t^0+h, t^0+2h, t^0+3h, ...

0.1s
0.01s

$\Rightarrow$ $y(t)$ by making the following Approximation

$$f(t,y) \approx f(t^0, y^0), \text{ for } t \in [t^0, t^0+h]$$

Sufficient small h

$$y(t) = y^0 + \int_{t^0}^{t} f(z, y(z))\, dz$$

$$= y^0 + f(t^0, y^0)\,(t-t^0), \quad t \in [t^0, t^0+h]$$

$t^1 = t^0+h, \quad y^1 = y^0 + hf(t^0, y^0)$

$t^2 = t^0+2h, \quad y^2 = y^1 + hf(t^1, y^1)$

$\vdots$

Continue to produce $t^3, t^4, \ldots$

$$y^{n+1} = y^n + hf(t^n, y^n), \quad n=0,1,\ldots$$

Euler Method

Explicit/Implicit Runge-Kutta Method

$$y^{n+1} = y^n + h\sum_{j=1}^{q} b_j f(t^n + c_j h, y(t^n + c_j h))$$

$t^n + c_1 h, t^n + c_2 h \cdots t^n + c_q h$: RK nodes

$b_1, b_2, \cdots, b_q$ RK weights

Runge-Kutta expresses $y(t^n + c_j h)$

by updating y^n with a linear combination

of $f(t^n + c_i h, y(t^n + c_i h)), \forall i = 1, \cdots, q$

$$y(t^n + c_j h) = y^n + h \sum_{i=1}^{q} a_{j,i} f(t^n + c_i h, y^{n+c_i})$$

denotes as y^{n+c_j}

$$\sum_{i=1}^{q} a_{j,i} = c_j$$

$A = (a_{j,i})_{j,i=1,2,\cdots,q}$ $\Leftarrow$ RK matrix

Explicit RK: A is lower triangular matrix

Implicit RK: A can be arbitrary matrix

See Reference "A first course in the Numerical Analysis of Differential Equations" for more info.

Runge-Kutta Methods

UC San Diego

- The family of Runge–Kutta methods is a generalization of the RK4 method, given by

$$y^{n+1} = y^n + h\sum_{j=1}^{q} b_j f(t^n + c_j h, y(t^n + c_j h))$$

we express the value of y^{n+1} as combination of values of y at nodes $t^n + c_1 h, t^n + c_2 h, \dots, t^n + c_q h$.

- But we don't know the value of $y(t^n + c_j h)$, need to have an approximation!

$$y(t^n + c_j h) = y^n + h\sum_{i=1}^{q} a_{j,i} f(t^n + c_i h, y(t^n + c_i h))$$

$A = (a_{j,i})_{i,j=1,2,\dots,q}$ RK matrix; $b = \begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}$ RK weights

$c = \begin{bmatrix} c_1 \\ \vdots \\ c_q \end{bmatrix}$ RK nodes $\quad \sum_{i=1}^{q} a_{j,i} = c_j$

(Note: the above equations may have different but equivalent definitions in other texts)

Runge-Kutta Methods

UC San Diego

4-th order Runge-Kutta Method

$$y^{n+1} = y^n + h\left(\frac{1}{6}f(t^{n+c_1}, y^{n+c_1}) + \frac{1}{3}f(t^{n+c_2}, y^{n+c_2}) + \frac{1}{3}f(t^{n+c_3}, y^{n+c_3}) + \frac{1}{6}f(t^{n+c_4}, y^{n+c_4})\right)$$

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

c	A
	$b^\top$

- $t^{n+c_1} = t^n + c_1 h = t^n$ $\quad y^{n+c_1} = y^n$
- $t^{n+c_2} = t^n + c_2 h = t^n + \frac{1}{2}h$, $\quad y^{n+c_2} = y^n + \frac{h}{2}f(t^{n+c_1}, y^{n+c_1})$
- $t^{n+c_3} = t^n + c_3 h = t^n + \frac{1}{2}h$ $\quad y^{n+c_3} = y^n + \frac{h}{2}f(t^{n+c_2}, y^{n+c_2})$
- $t^{n+c_4} = t^n + h$ $\quad y^{n+c_4} = y^n + hf(t^{n+c_3}, y^{n+c_3})$

(Note: the above equations may have different but equivalent definitions in other texts)

Discrete Time PINNs

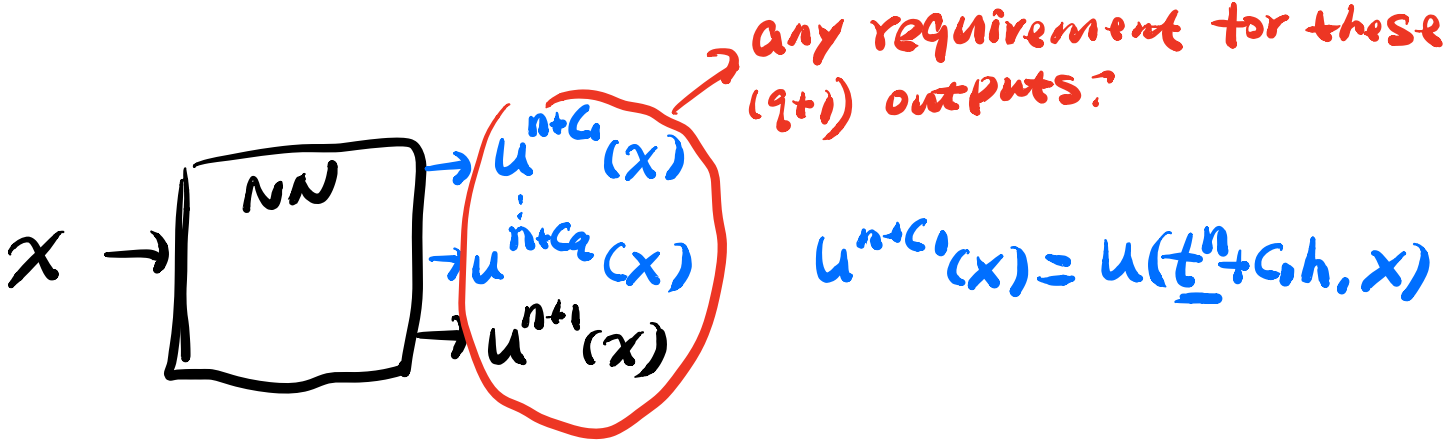
UC San Diego

- Suppose we want to solve a partial differential equations of the general form

$$u_t = -\mathcal{N}[u]$$

where $x \in \Omega$, $t \in [0, T]$. $\mathcal{N}[u]$ is a nonlinear differential operator.

- Denote $u^n(x) = u(t^n, x), u^{n+1}(x) = u(t^{n+1}, x)$. We know: $u^n(x) \rightarrow$ want to predict $u^{n+1}(x)$



$$u^{n+c_j} = u^n + h\sum_{i=1}^{q} a_{j,i}\,(-\mathcal{N}[u^{n+c_i}]),\quad j=1,2,\cdots,q$$

$$u^{n+1} = u^n + h\sum_{i=1}^{q} b_i\,(-\mathcal{N}[u^{n+c_i}])$$

Rearrangement:

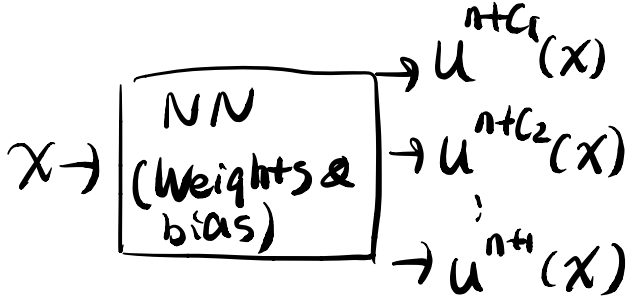
$$u^n = \underbrace{u^{n+c_j} + h\sum_{i=1}^{q} a_{j,i}\,\mathcal{N}[u^{n+c_i}]}_{\text{PINN}} \quad \swarrow\ u_j^n,\ j=1,\cdots,q$$

$$u^n = \underbrace{u^{n+1} + h\sum_{i=1}^{q} b_i\,\mathcal{N}[u^{n+c_i}]}_{\text{PINN}} \quad \swarrow\ u_{q+1}^n$$

UC San Diego

At step t^n, we have $u(t^n, x)$, want to predict $u(t^{n+1}, x)$

Neural Network



- multi-output NN with $(q+1)$ outputs

Physics Informed NN with $(q+1)$ outputs

$$u_1^n(x) = u^{n+c_1}(x) + h\sum_{i=1}^{q} a_{1,i}\, N[u^{n+c_i}(x)]$$

$$\vdots$$

$$u_{q+1}^n(x) = u^{n+1}(x) + h\sum_{i=1}^{q} b_i\, N[u^{n+c_i}(x)]$$

- use $u^n(x)$ as label

Example: Allen–Cahn Equation

UC San Diego

- Consider the Allen–Cahn equation along with periodic boundary conditions

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, x \in [-1,1], t \in [0,1]$$
$$u(0,x) = \mathrm{x}^2\cos(\pi x)$$
$$u(t,-1) = u(t,1)$$
$$u_x(t,-1) = u_x(t,1)$$

- For the Allen–Cahn equation, the nonlinear operator is given by

$$\mathcal{N}[u] = -0.0001u_{xx} + 5u^3 - 5u$$

- Define a deep neural network, takes x as input and output $(q+1)$ items

$$[u^{n+c_1}(x), u^{n+c_2}(x), \cdots, u^{n+1}(x)]$$

- Define a PINN also with $(q+1)$ outputs

$$u_i^n(x) = u^{n+c_i}(x) + h\sum_{j=1}^{q} a_{i,j} N[u^{n+c_j}(x)], \quad i=1,\cdots,q+1$$

$$N[u^{n+c_i}(x)] = -0.0001 u_{xx}^{n+c_i}(x) + 5(u^{n+c_i}(x))^3 - 5u^{n+c_i}(x)$$

- Use $u^n(x)$ as supervised label for the PINN outputs $u_1^n(x), u_2^n(x) \cdots u_{q+1}^n(x)$

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u_j^n(x_i) - u^n(x_i)|^2$$

Here $\{x_i, u^n(x_i)\}$ corresponding to data at time t^n

Example: Allen–Cahn Equation

UC San Diego

- Similarly, we add a loss for the boundary and initial conditions

$$SSE_b = \sum_{i=1}^{q} |u^{n+c_i}(-1) - u^{n+c_i}(1)|^2 + |u^{n+1}(-1) - u^{n+1}(1)|^2$$
$$+ \sum_{i=1}^{q} |u_x^{n+c_i}(-1) - u_x^{n+c_i}(1)|^2 + |u_x^{n+1}(-1) - u_x^{n+1}(1)|^2.$$

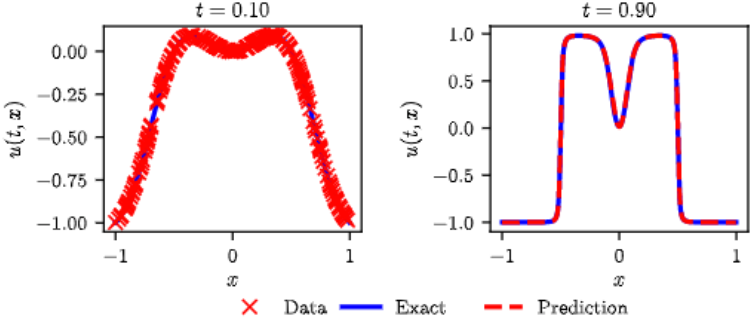
- Therefore, we can train the shared parameters of multi-output neural network $u(x)$ and multi-output PINN by minimizing the sum of squared loss (or MSE loss):

$$SSE = SSE_n + SSE_b$$

Example: Allen–Cahn equation

UC San Diego

- For classic Runge–Kutta methods, the time step Δt (or h) are usually confined to be small in order to ensure numerical stability and obtain accurate solution
- For PINNs, we can choose an arbitrarily large number of stages (q). This enables us to take very large time steps while retaining stability and high predictive accuracy.
- For example, choose $t^n = 0.1, t^{n+1} = 0.9$ with q=100 Runge–Kutta stages (NN has in total 101 output quantities), allowing us to resolve the entire spatio-temporal solution **in a single step**.



- 4 hidden layers and 200 neurons per layer
- 200 training data points for $\mathrm{u}(0.1, \mathrm{x})$ randomly sampled from $\mathrm{x} \in [-1,1]$