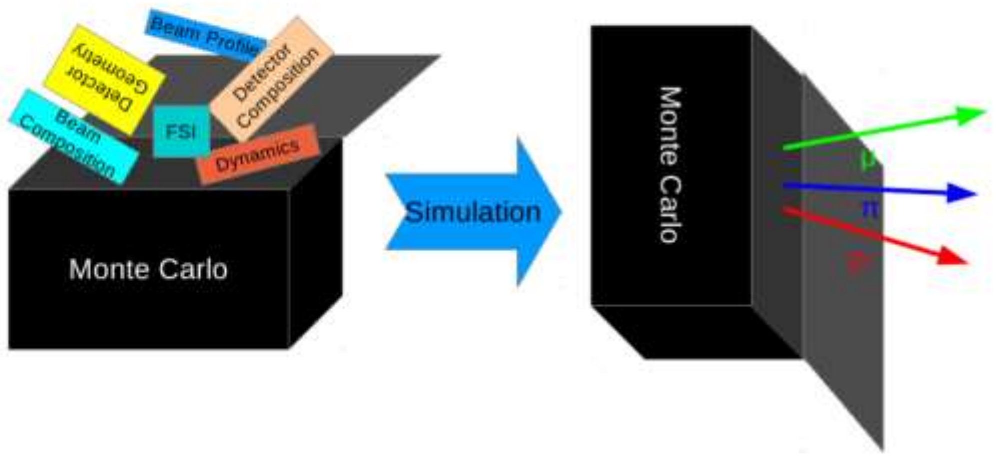
Physics-Informed Machine Learning

03 January 2024

Melik Kaan Şelale

Scientific Research

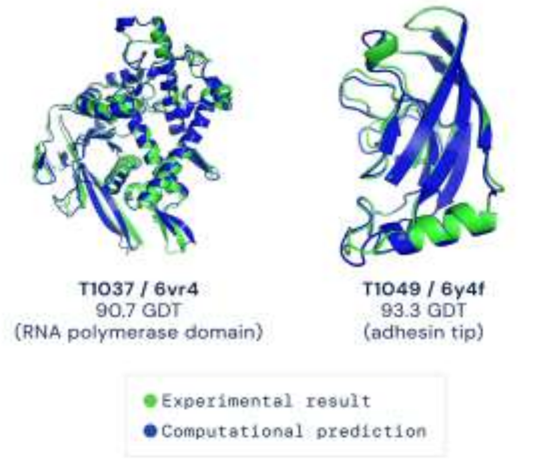
- Traditional way: Theoretical derivation + experimental verification
- Changes after development of Computational Methods

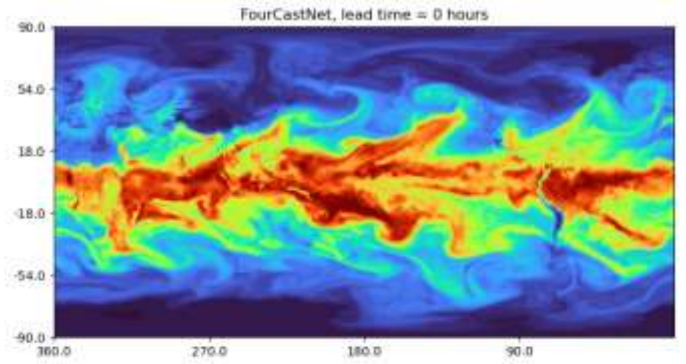


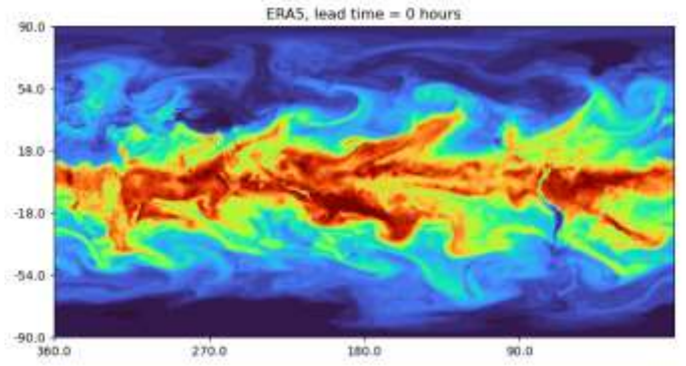
- Learning from observational data

Deep Neural Networks in Modeling Physical Systems

- Alphafold2:
  - Solved(?) the 50 year-old-challange of protein folding problem.
  - Is the solved problem the process of protein folding, or the final prediction?
  - As a further reading about this question: https://www.pnas.org/doi/10.1073/pnas.2214423119
- FourCastNet:
  - Ultra-large learning-based weather forecasting system
- Deep Potential:
  - Neural models for learning large-scale molecular potential satisfying symmetry



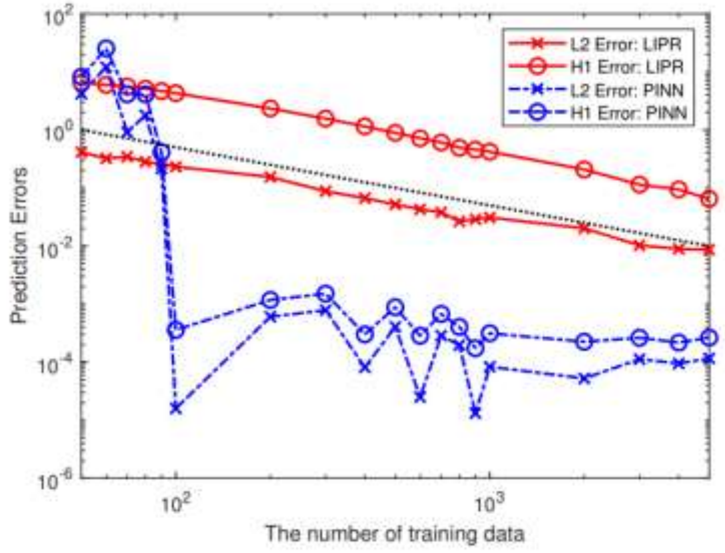




Limitations of Machine Learning Based Models

- Generalization errors

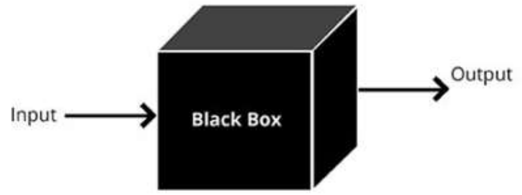
The L2 and H1 convergence for the 1D Poisson equation whose exact solution is $u^*(x) = (1 - x^2)\sin(6\pi x)$



Source: https://arxiv.org/pdf/2004.01806.pdf

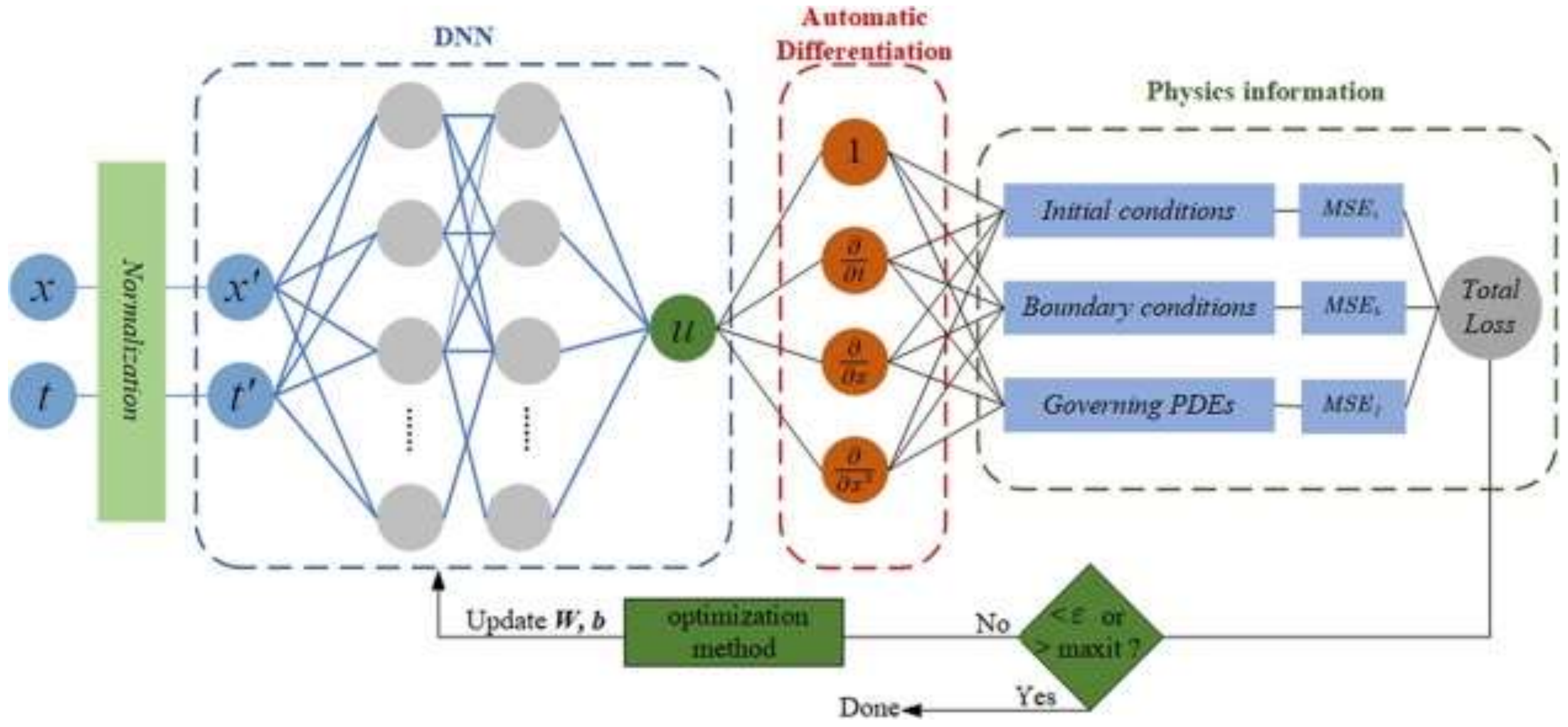
- Lack of robustness

- Lack of Interpretability



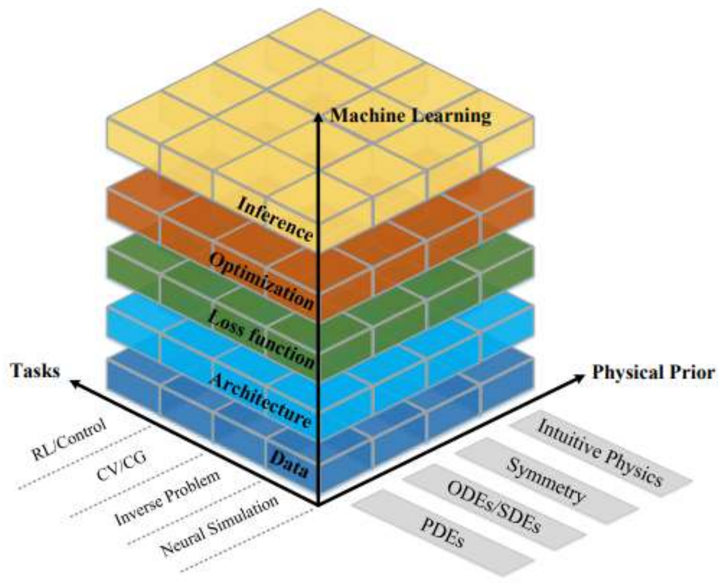
Solution: Integration of Physical Laws Into Machine Learning Models

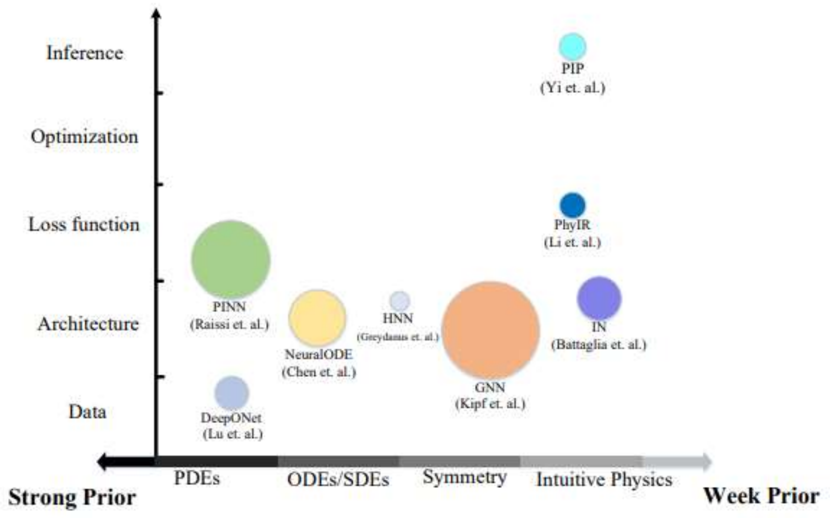
- Physics-Informed Machine Learning (PIML): A paradigm that seeks to construct models that make use of both empirical data and prior physical knowledge to enhance performance on tasks that involve a physical mechanism.

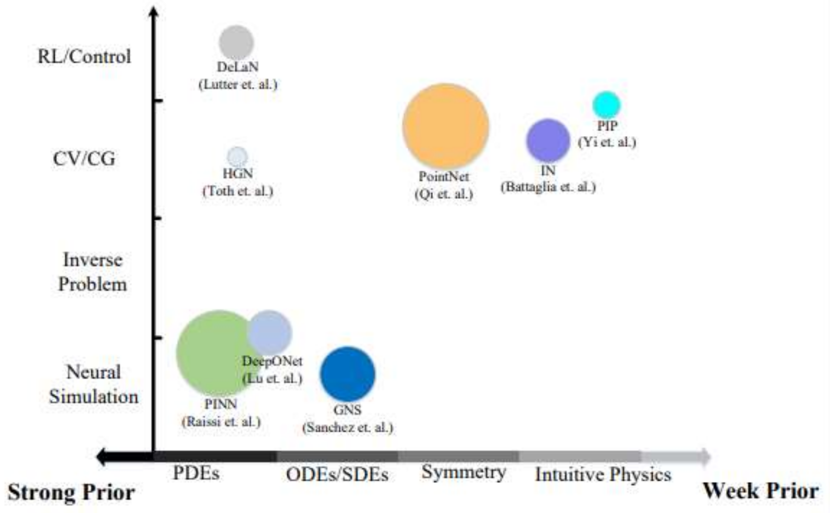


Representation of Physics Prior

- Physics prior knowledge that describes the behavior of real world can be giving in various ways.







- **Differential Equations**
  - Differential equations (PDEs, ODEs, SDEs) precisely represent scientific phenomena with strongest prior and, making them effective for modeling various physical systems.

$$\frac{\hbar^2}{2m}\nabla^2\Psi + V\Psi = \frac{i\hbar\partial}{\partial t}\Psi$$

$$(\mathbf{i}\hbar\boldsymbol{\gamma}^{\mu}\boldsymbol{\nabla}_{\mu} - \mathbf{mc})\boldsymbol{\psi} = \mathbf{0}$$

$$\nabla\cdot\mathbf{E} = \frac{\rho}{\varepsilon_0}$$
$$\nabla\cdot\mathbf{B} = 0$$
$$\nabla\times\mathbf{E} = -\frac{\partial\mathbf{B}}{\partial t}$$
$$\nabla\times\mathbf{B} = \mu_0\mathbf{j} + \frac{1}{c^2}\frac{\partial\mathbf{E}}{\partial t}$$

- These equations can be used as *governing equations*

Partial differential equation: $\mathcal{F}(u;\theta)(\boldsymbol{x}) = 0,$
Initial Conditions: $\mathcal{I}(u;\theta)(x,t_0) = 0,$
Boundary Conditions: $\mathcal{B}(u;\theta)(x,t) = 0.$

u: state variables of the physical system
θ: parameters of physical system
x: spatial coordinates

- **Symmetry Constraints**
  - Collection of transformations that can be applied to objects where the abstract set of symmetries is capable of transforming diverse objects.

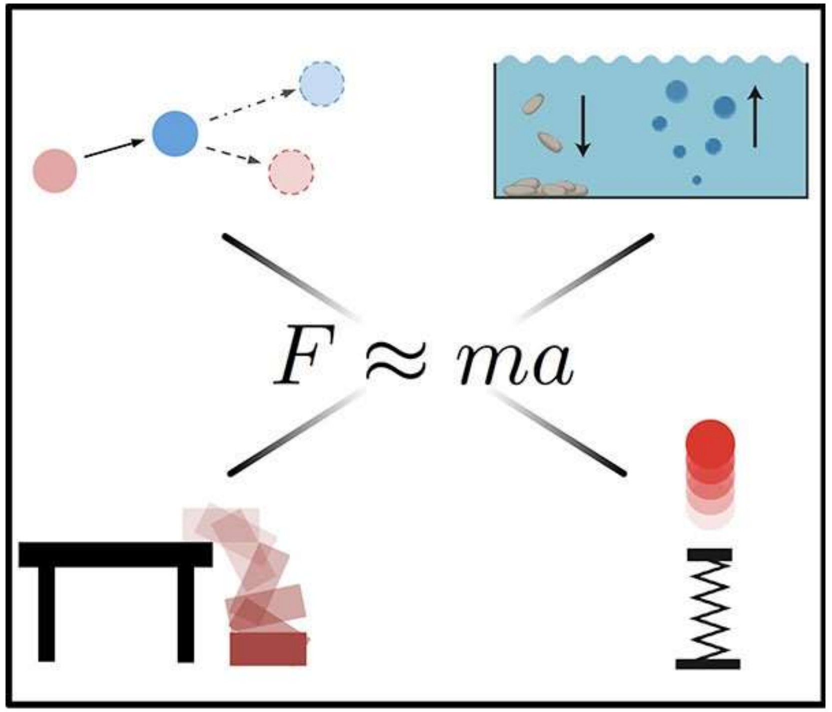
  - Rotation, translation, permutation, scale, topological invariance

$$\varphi(x) = \varphi(s(x))$$

  - Advantages:
    - Better generalization
    - Reduced data redundancy
    - Increased interpretability
    - Handling complex data structures

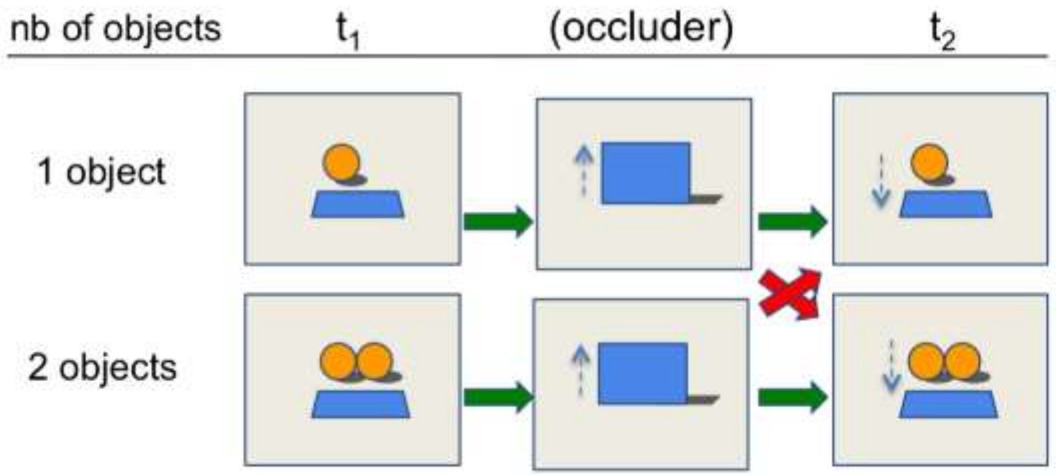
- **Intuitive Physics**
  - Common-sense knowledge about physical world.

    - Object permanence
    - Gravity
    - Newton's Second Law
    - Conservation Laws



In general intuitive physics can be incorporated as constraints or regularizers to enhance machine learning.

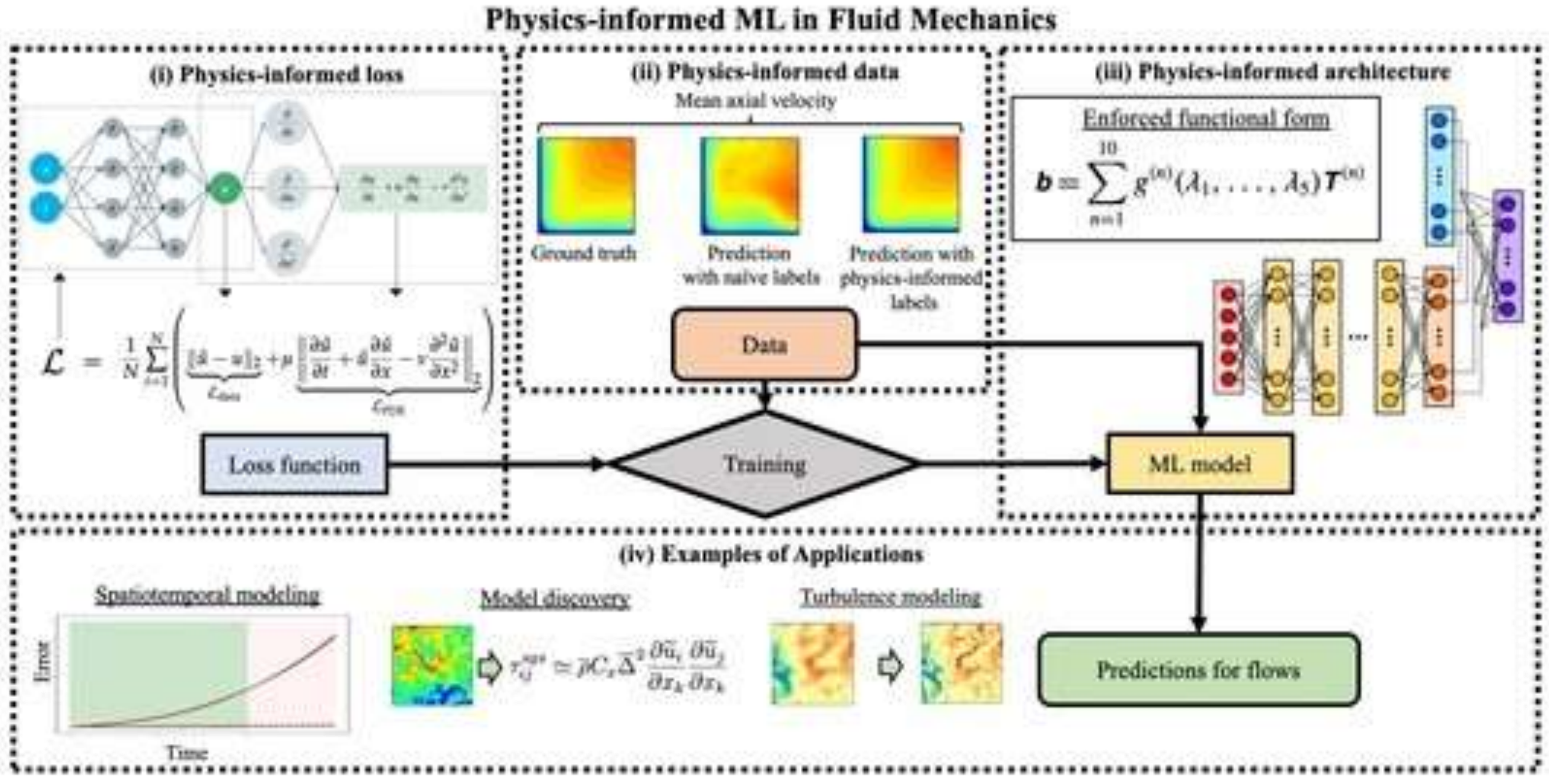
Physical simulations generate training data for machine learning models, enhancing their comprehension of physical phenomena.



Integrating Physics into ML

- Physics-informed data
- Embed physical prior into the model design
- Loss functions and optimization methods
- Pre-trained ML models, design different inference algorithms to enforce physical prior

- Result: Improved performance!


Physics-informed ML in Fluid Mechanics
(i) Physics-informed loss
(ii) Physics-informed data
Mean axial velocity
Ground truth
Prediction with naive labels
Prediction with physics-informed labels
(iii) Physics-informed architecture
Enforced functional form
$b = \sum_{n=1}^{10} g^{(n)}(\lambda_1, \ldots, \lambda_5) T^{(n)}$
Data
Loss function
Training
ML model
(iv) Examples of Applications
Spatiotemporal modeling
Error
Time
Model discovery
Turbulence modeling
Predictions for flows

Tasks of PIML

- Necessity of real-world physical processes
- Scientific problems and ML problems

- PIML => Two main categories:
- *Neural Simulation*
- *Inverse Problems*

Neural Simulation

- Neural simulation focuses on predicting or forecasting the states of physical systems using physical knowledge and data.
- Consist of two parts:
  - Neural Solver
  - Neural Operator
- Challenges of Traditional ODEs/PDEs Solvers (Numerical Methods)
  - Curse of dimensionality
  - Difficulty in incorporating data from experiments and problems where governing equations are partially unknown

Neural Solver

- Problem: Use neural networks to represent and solve the state of the physcial system if physical laws are completely known.
- Solving a single PDEs/ODEs using neural networks
- Advantages:
  - Flexibility to integrate the data and knowledge
  - High-dimensions

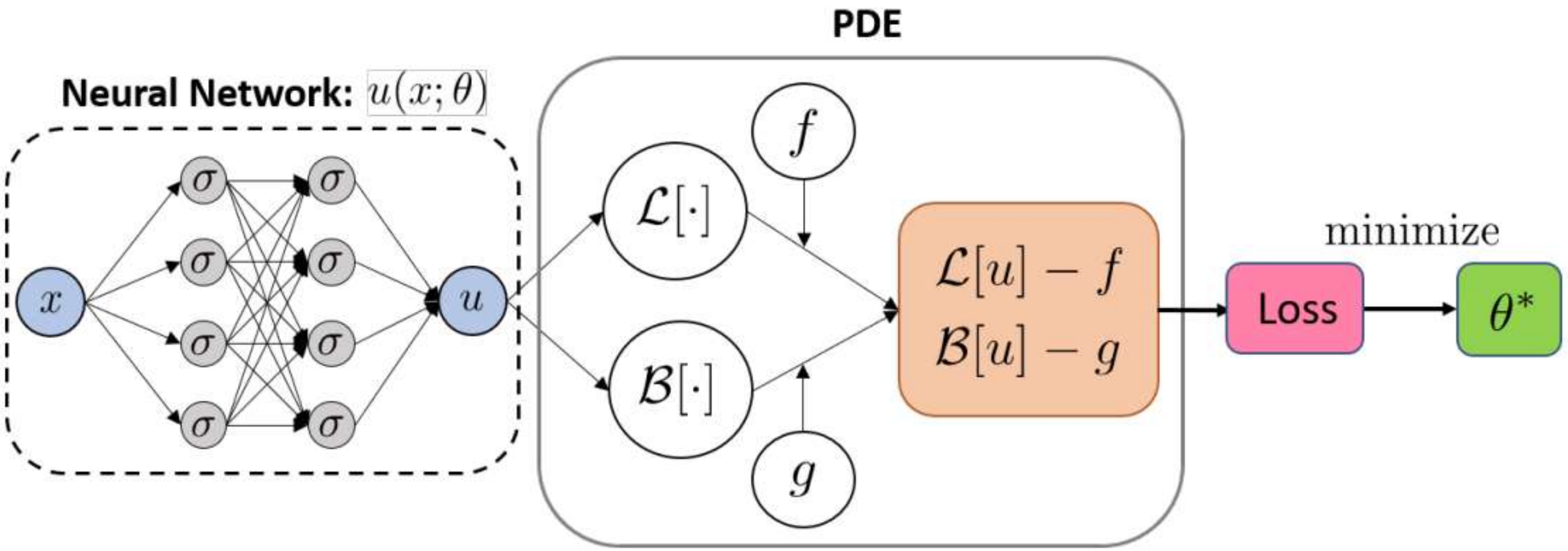
- The most representative approach: Physics Informed Neural Networks

Physics Informed Neural Networks (PINNs)

- Flexible neural network method that incorporate PDE constraints into learning paradigm
- Loss function:

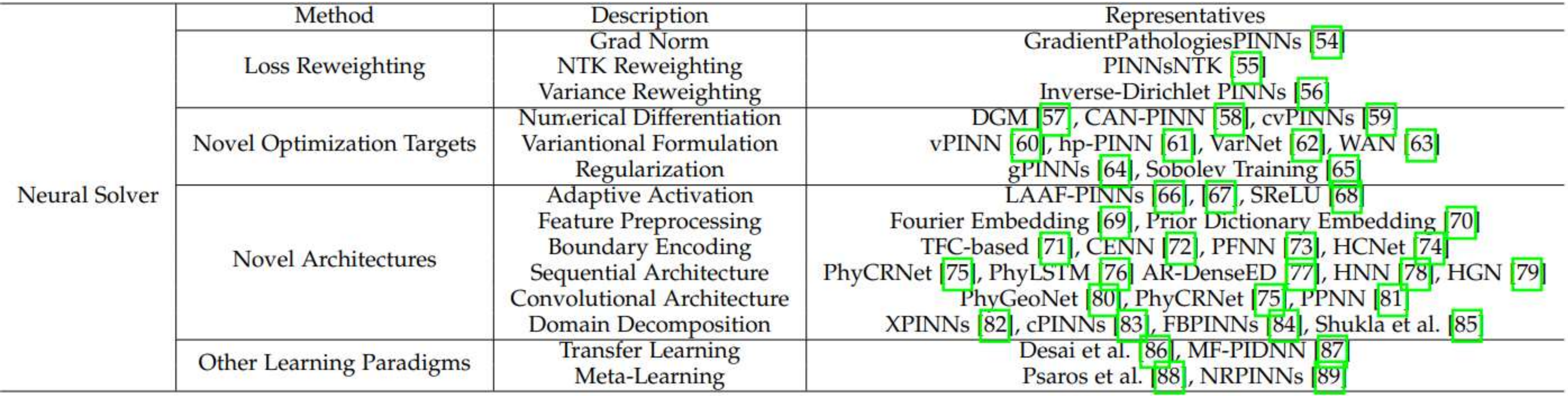
$$\begin{aligned}\mathcal{L} = &\frac{\lambda_r}{|\Omega|}\int_{\Omega}\|\mathcal{F}(u_w;\theta)(\boldsymbol{x})\|^2\mathrm{d}\boldsymbol{x} + \frac{\lambda_i}{|\Omega_0|}\int_{\Omega_0}\|\mathcal{I}(u_w;\theta)(\boldsymbol{x})\|^2\mathrm{d}\boldsymbol{x} \\ &+\frac{\lambda_b}{|\partial\Omega|}\int_{\partial\Omega}\|\mathcal{B}(u_w;\theta)(\boldsymbol{x})\|^2\mathrm{d}\boldsymbol{x} + \frac{\lambda_d}{N}\sum_{i=1}^{N}\|u_w(\boldsymbol{x}_i) - u(\boldsymbol{x}_i)\|^2,\end{aligned}$$

Schematic of PINNs



PINN Variants and Some limitations

- Convergence of different loss terms
- Complexity of physical systems and MLP
- Structure of loss functions for PDEs and initial/boundary conditions.



	Method	Description	Representatives
Neural Solver	Loss Reweighting	Grad Norm	GradientPathologiesPINNs [54]
		NTK Reweighting	PINNsNTK [55]
		Variance Reweighting	Inverse-Dirichlet PINNs [56]
	Novel Optimization Targets	Numerical Differentiation	DGM [57], CAN-PINN [58], cvPINNs [59]
		Variantional Formulation	vPINN [60], hp-PINN [61], VarNet [62], WAN [63]
		Regularization	gPINNs [64], Sobolev Training [65]
	Novel Architectures	Adaptive Activation	LAAF-PINNs [66], [67], SReLU [68]
		Feature Preprocessing	Fourier Embedding [69], Prior Dictionary Embedding [70]
		Boundary Encoding	TFC-based [71], CENN [72], PFNN [73], HCNet [74]
		Sequential Architecture	PhyCRNet [75], PhyLSTM [76] AR-DenseED [77], HNN [78], HGN [79]
		Convolutional Architecture	PhyGeoNet [80], PhyCRNet [75], PPNN [81]
		Domain Decomposition	XPINNs [82], cPINNs [83], FBPINNs [84], Shukla et al. [85]
	Other Learning Paradigms	Transfer Learning	Desai et al. [86], MF-PIDNN [87]
		Meta-Learning	Psaros et al. [88], NRPINNs [89]

Loss Re-Weighting

$$\mathcal{L} = \frac{\lambda_r}{|\Omega|}\int_{\Omega}\|\mathcal{F}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x} + \frac{\lambda_i}{|\Omega_0|}\int_{\Omega_0}\|\mathcal{I}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x}$$
$$+\frac{\lambda_b}{|\partial\Omega|}\int_{\partial\Omega}\|\mathcal{B}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x} + \frac{\lambda_d}{N}\sum_{i=1}^{N}\|u_w(\boldsymbol{x}_i) - u(\boldsymbol{x}_i)\|^2,$$

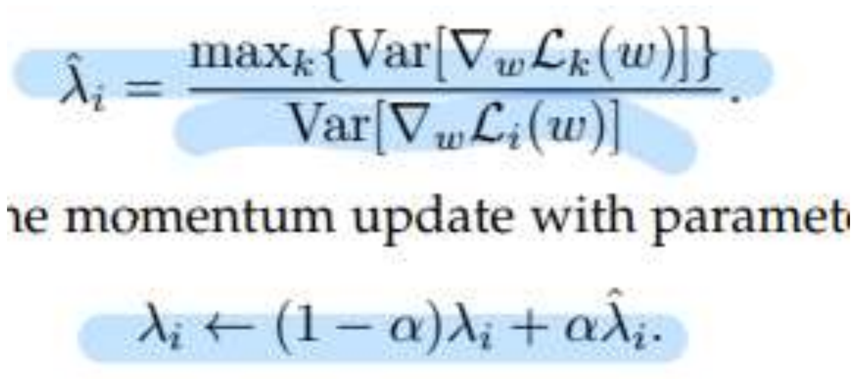
- **The problem**: The problem arises when directly optimizing a loss function like that, as it may lead to incorrect results due to differences in scale and convergence speed among various losses.
- One solution is re-weight different losses to balance the training process an accelerate the convergence speed.
    - One famous work https://arxiv.org/abs/2001.04536 introduces a method for large differences between gradients of loss terms.

$$\hat{\lambda}_i = \frac{\max\{\nabla_w \mathcal{L}_r(w_n)\}}{\overline{|\nabla_w \mathcal{L}_i(w_n)|}}.$$

, the learning rate λ_i is updated by

$$\lambda_i \leftarrow (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i,$$

- Another study https://arxiv.org/abs/2107.00940 uses gradient variance to balance the training of PINNs. Also known as Inverse-Dirichlet Weighting



$$\hat{\lambda}_i = \frac{\max_k\{\mathrm{Var}[\nabla_w \mathcal{L}_k(w)]\}}{\mathrm{Var}[\nabla_w \mathcal{L}_i(w)]}.$$

ıe momentum update with paramet

$$\lambda_i \leftarrow (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i.$$

- Other studies:
  - Using theory of Neural Tangentin Kernel. https://arxiv.org/abs/2007.14527
  - Using some characteristic quantities. https://arxiv.org/abs/2112.05489v2 , https://arxiv.org/abs/2002.06269

Data Re-Sampling

- Another solution for handling imbalance learning process problem
- Sample quasi-random points or low-discrepancy sequence of points from the geometric domain.
  - Further reading: https://arxiv.org/pdf/2202.06416.pdf
  - Sobol sequence, Latin hypercube sampling, Halton sequence, Hammersley sampling, and Faure sampling
- Focusing points from areas with higher error
  - Further reading: https://arxiv.org/pdf/2104.12325.pdf

$$q_j^{(i)} = \frac{\left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}_j)\right\|_2}{\sum_{j=1}^{N} \left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}_j)\right\|_2}, \quad \forall j \in \{1, \cdots, N\},$$

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_f\left[J(\boldsymbol{\theta})\right]$$

$$\boldsymbol{\theta}^* \approx \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{j=1}^{N} \frac{f(\boldsymbol{x}_j)}{q(\boldsymbol{x}_j)} J(\boldsymbol{\theta}; \boldsymbol{x}_j), \quad \boldsymbol{x}_j \sim q(\boldsymbol{x}).$$

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \frac{\eta^{(i)}}{m} \sum_{j \in M} \frac{1}{N q_j^{(i)}} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(i)}; \boldsymbol{x}_j).$$

- Using generative model sample from loss distribution
  - Deep adaptive sampling (DAS) for solving PDEs
  - Residuals => Distributions – Krnet to generate
  - More samples on large residuals
  - Further reading: https://arxiv.org/pdf/2112.14038.pdf

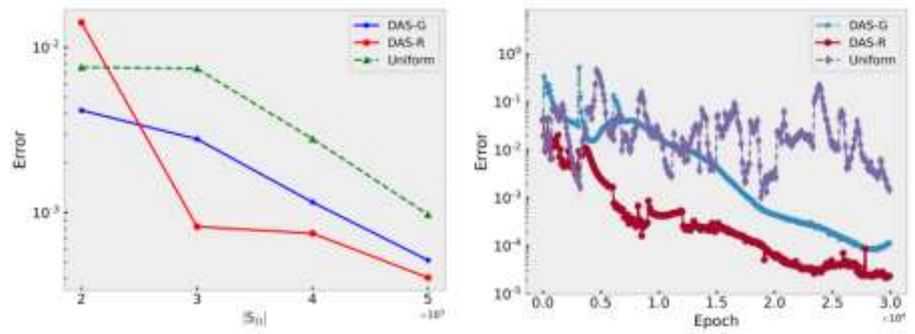


Figure 1: Approximation errors for the two-dimensional peak test problem. Left: The error w.r.t sample size $|S_\Omega|$; Right: The error w.r.t epoch for $|S_\Omega| = 5 \times 10^3$.

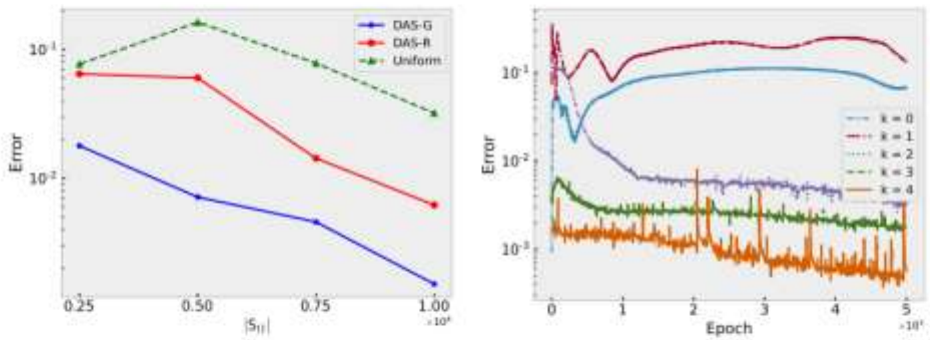
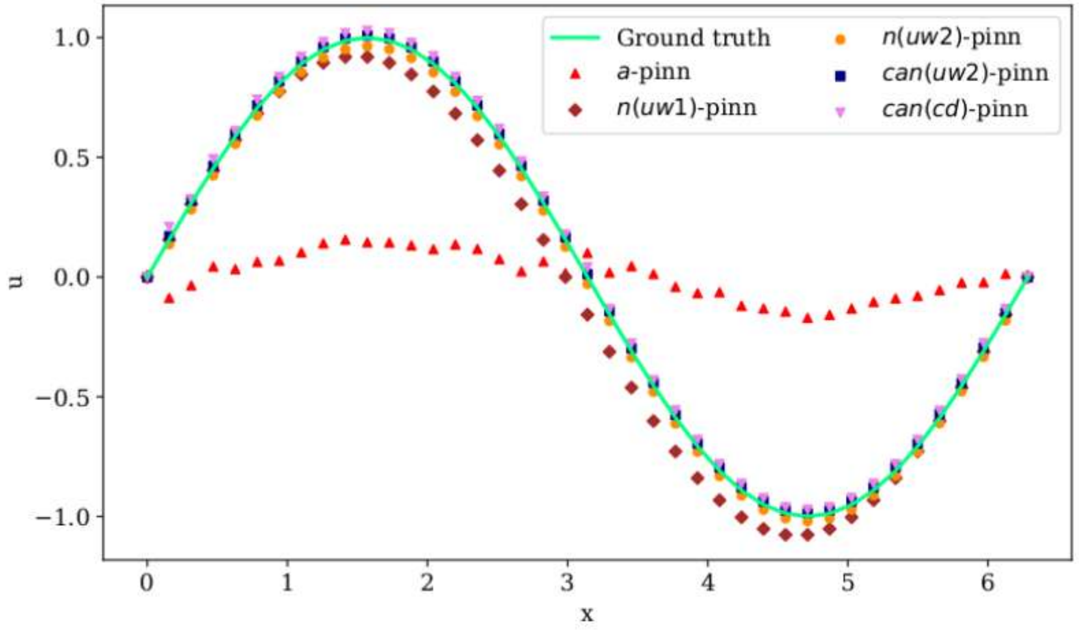


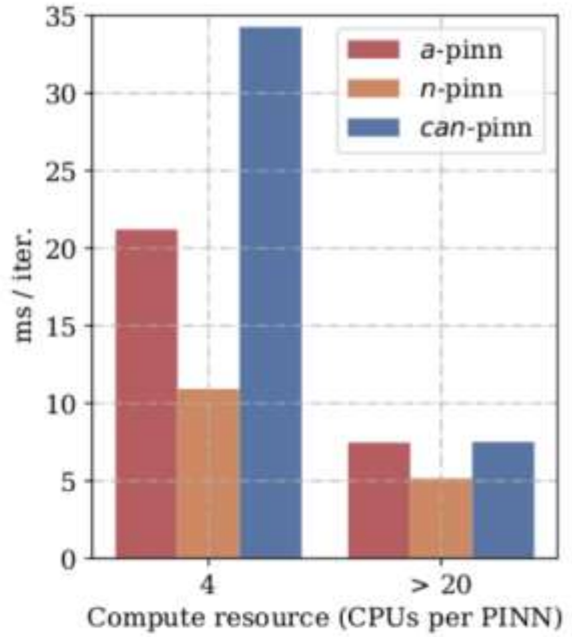
Figure 5: Approximation errors for the two-dimensional test problem with two peaks. Left: The error w.r.t sample size $|S_\Omega|$; Right: The errors of DAS-G at each adaptivity iteration steps. $|S_\Omega| = 10^4$.

- The proposed approach for Physics-Informed Neural Networks (PINNs) incorporates importance sampling to improve convergence. It suggests sampling collocation points based on a distribution proportional to the loss function, enhancing gradient information and potentially accelerating convergence.

Novel Optimization Objectives

- **Incorporating Numerical Differentiation:**
  - Vanilla PINNs uses automatic differentation => Backpropogation
  - DGM https://arxiv.org/pdf/1708.07469.pdf shows the high cost of higher-order derivatives => Monte Carlo
  - CAN-PINN https://arxiv.org/ftp/arxiv/papers/2110/2110.15832.pdf combines automatic differentiation and numerical differentation





- **Regularization terms**
  - Existed: L2-L1 regularization
  - Novel: Gradient-enhanced PINNs https://arxiv.org/pdf/2111.02801.pdf
    - Higher order derivatives => Regularization terms

$$\mathcal{D}_i^k \mathcal{F}(u)(\boldsymbol{x}) = \frac{\partial^k}{\partial x_i^k} \mathcal{F}(u)(\boldsymbol{x}) = 0.$$

$$\mathcal{L}_{\text{reg}} = \sum_{k,i \in K,I} \sum_{\boldsymbol{x}_j \in \mathcal{D}_r} \lambda_{k,i} \| \mathcal{D}_i^k \mathcal{F}(u)(\boldsymbol{x}_j) \|^2.$$

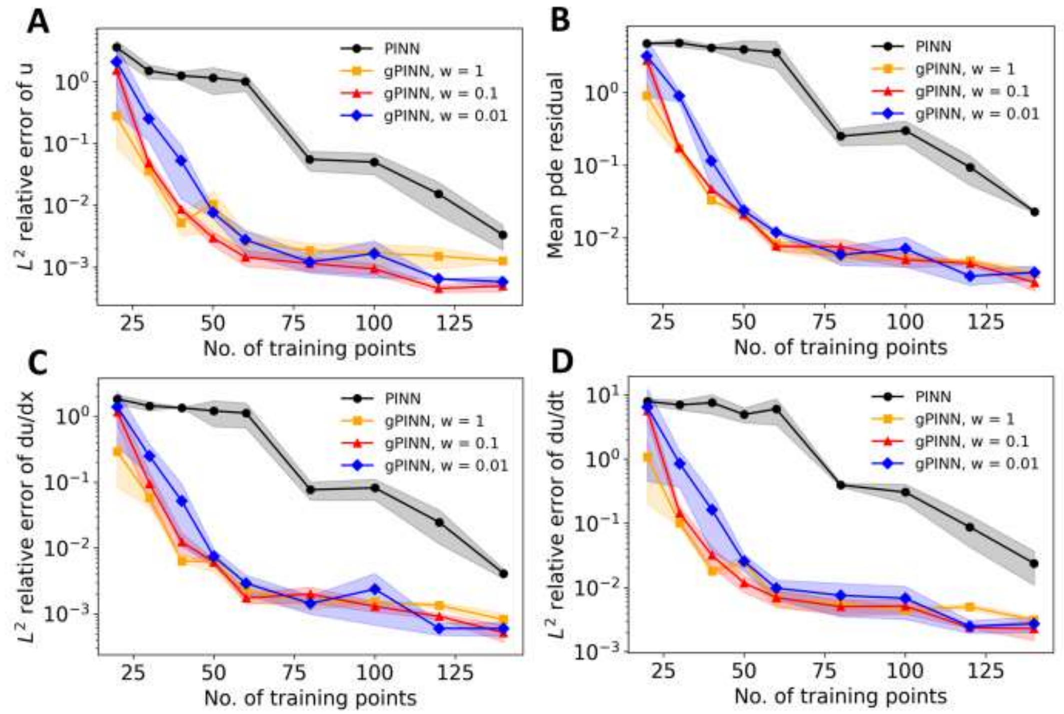
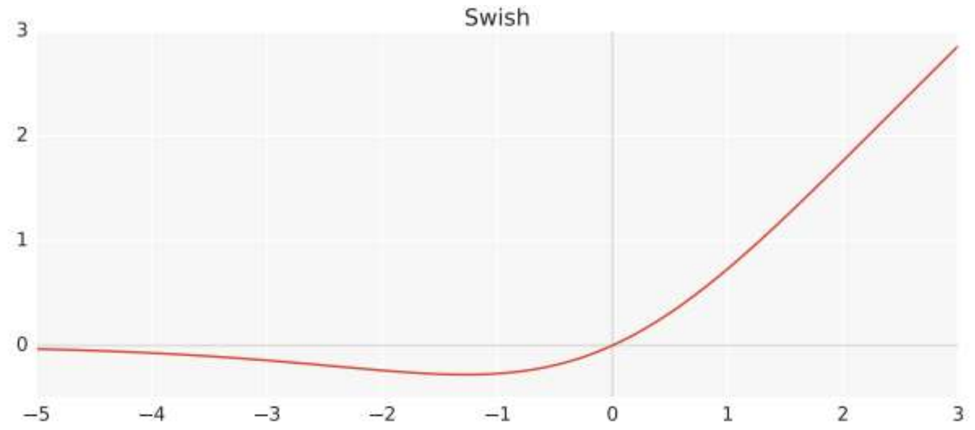
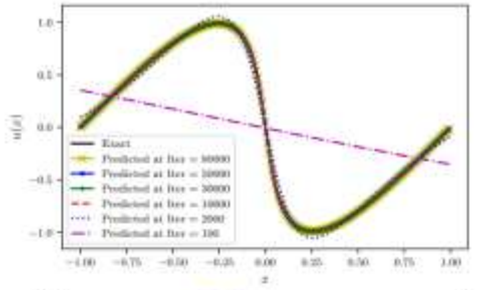


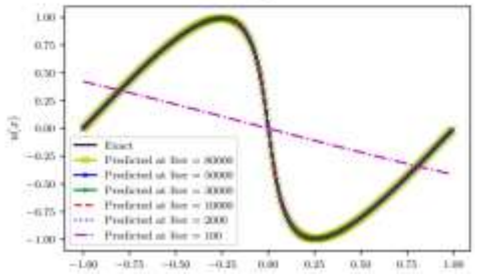
Figure 3: **Example in Section 3.2.2: Comparison between PINN and gPINN.** (**A**) L^2 relative error of u for PINN and gPINN with w =1, 0.1, and 0.01. (**B**) Mean absolute value of the PDE residual. (**C**) L^2 relative error of $\frac{du}{dx}$. (**D**) L^2 relative error of $\frac{du}{dt}$.

Novel Neural Architectures

- Novel Architectures > Specific problems
- CNN, RNN, LSTM
- Vanilla PINNs > MLPs > success on general PDEs > failure on complex PDEs
- **Activation Functions**
  - Traditionals: ReLU, Sigmoid, tanh
  - Necessity of smooth activation functions in PINNs. => Higher order derivatives
  - Swish activation function
  - **Adaptive activation functions** https://arxiv.org/pdf/1906.01170.pdf
    - **Dynamic changes in the topology of the loss function during optimization**
    - **Scaleable hyperparameter**
    - **Outperform traditional fixed activation functions. Convergence rate, solution accuracy**
    - **Prior information from governing equations**
    - **Gradient vanishing problem**



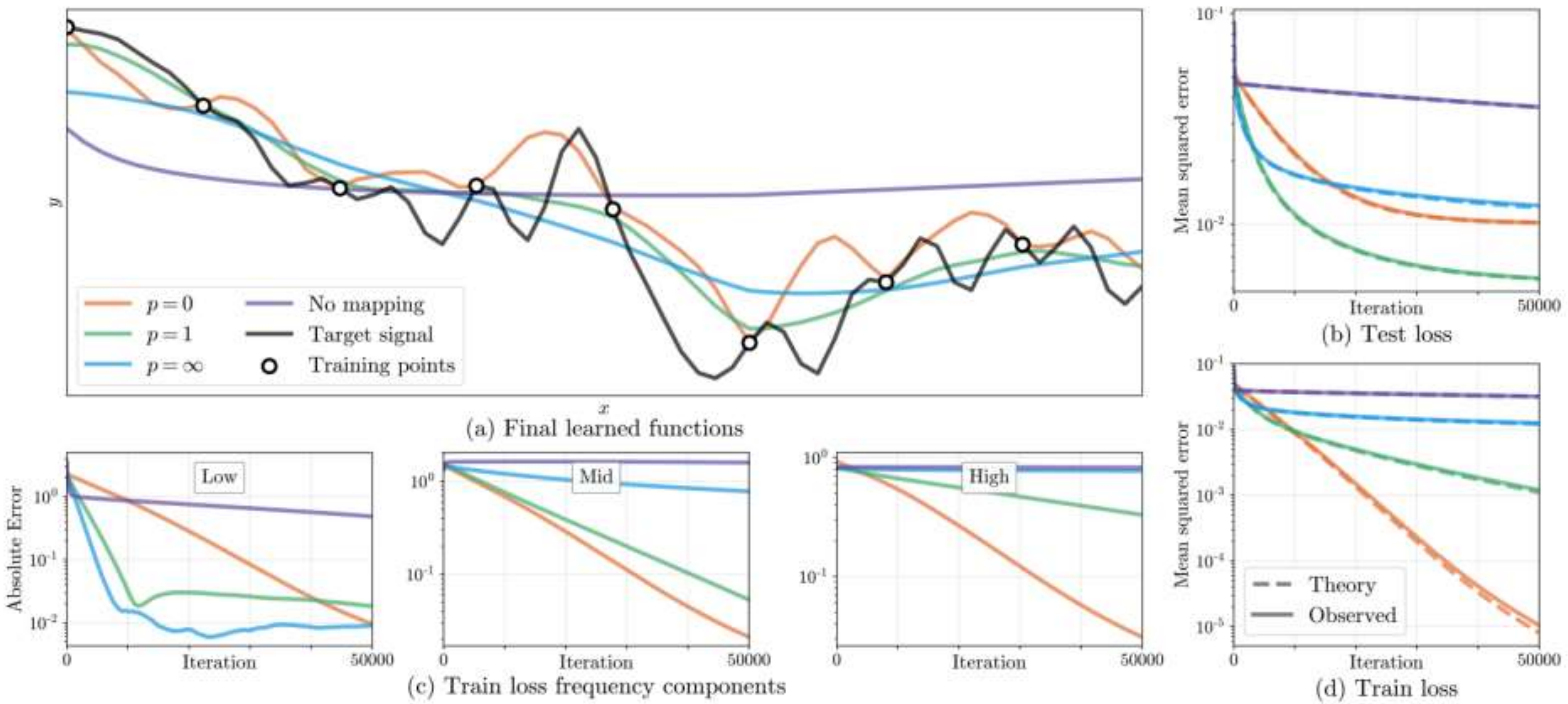




- **Feature Preprocessing**
  - Feature preprocessing is a basic tool before we feed data into NNs
  - Accelerate
  - Scaling, magntiude
  - **Fourier feature embedding** https://arxiv.org/pdf/2006.10739.pdf

$$\gamma(\boldsymbol{x}) = (\sin(2\pi \boldsymbol{b}_1^T \cdot \boldsymbol{x}), \cos(2\pi \boldsymbol{b}_1^T \cdot \boldsymbol{x}), \ldots, \sin(2\pi \boldsymbol{b}_m^T \cdot \boldsymbol{x}), \cos(2\pi \boldsymbol{b}_m^T \cdot \boldsymbol{x})).$$

b: scale parameters



- **Multiple NNs and Boundary Encoding**
  - Capaticy problems of a single MLP
  - Using multiple MLPs without sharing parameters to separatley output each component
  - **1- Multiple NNs to reduce the order of PDEs** https://arxiv.org/pdf/2002.10558.pdf
    - PINN outperforms traditional PINN in accuracy and trainability, particularly in capturing steady velocity and pressure fields. The study also examines the robustness of the mixed-variable scheme through exploration of different hyperparameters.
  - **2- Multiple NNs to encode boundary conditions with hard constraints**
    - Using NNs by fitting f and g on boundaries seperately
    - Most PDEs do not have an analytical general solution

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0,$$

$$u = f(x - ct) + g(x + ct).$$

- **Sequential Neural Architecture**
  - Famous architectures for sequential data recognition: RNN, LSTM, GRU, Transformer
  - Physical systems are time dependent and can be modeled as sequential data
  - These neural architectures can be combine to train PINNs instead of train the model without discretization of time with vanilla PINNs

$$\frac{\partial u}{\partial t} + F\left(u, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}, \ldots; \theta\right) = 0.$$

$$\mathcal{L}_{\text{reg}} = \left\| \frac{u_{i+1} - u_i}{\Delta t} - F\left(u_i, \frac{\partial u_i}{\partial x_1}, \cdots \frac{\partial u}{\partial x_d}, \ldots, \theta\right) \right\|^2 .$$

where 'u_i' is the output of the neural networks at time t_i

  - For instance using LSTM to represent solution of u
  - https://arxiv.org/pdf/2002.10253.pdf

- **Convolutional Architectures**
  - Widely used in image processing and computer vision
  - In the realm of numerical computing, specific convolutional kernels can be interpreted as a numerical approximation to differential operators.
  - Mostly jointly used with recurren architectures like LSTM
  - Popular architectures like vanilla CNN may not perfrom well in complex physical systems. Therefore new methods need to be developed
  - As an example: **https://arxiv.org/pdf/2004.13145.pdf**

Neural Operator

- To approximate a latent operator, which is a mapping between parameters and state variables using neural networks.
- Basically, the neural operator aims to solve a class of differential equations that map given parameters to its solutions.
- A neural solver solves specific instances of equations, while a neural operator learns a general surrogate model for the entire equation family.

$$\min_{w \in W} \|G_w(\theta)(\boldsymbol{x}) - \tilde{G}(\theta)(\boldsymbol{x})\|,$$

G: Latent Operator
$\tilde{G}$: Ground Truth

Category & Formulation	Rrepresentative	Description
Direct Methods $G_w(\theta)(\boldsymbol{x}) = b_0 + \sum_{k=1}^{p} b_k(\theta)t_k(\boldsymbol{x})$	DeepONet [167]	Parameterize b_k and t_k with neural networks, which are trained with supervised data.
	Physics-informed DeepONet [168]	Train DeepONet with a combination of data and physics-informed losses.
	Improved Architectures for DeepONet [169], [170]	Including modified network structures (see Eq. (95)), input transformation ($\boldsymbol{x} \mapsto (\boldsymbol{x}, \sin(\boldsymbol{x}), \cos(\boldsymbol{x}), \dots)$), POD-DeepONet (see Eq. (97)), and output transformation (see Eq. (98) and Eq. (99)).
	Multiple-input DeepONet [171]	A variant of DeepONet taking multiple various parameters as input, i.e., $\tilde{G}\colon \Theta_1 \times \Theta_2 \times \cdots \times \Theta_n \to Y$.
	Pre-trained DeepONet for Multi-physics [172], [173]	Model a multi-physics system with several pre-trained DeepONets serving as building blocks.
	Other Variants	Including Bayesian DeepONet [174], multi-fidelity DeepONet [175], and MultiAuto-DeepONet [176].
Green's Function Learning $G_w(\theta)(\boldsymbol{x}) = \int_\Omega \mathcal{G}(\boldsymbol{x}, \boldsymbol{y})\theta(\boldsymbol{y})\mathrm{d}\boldsymbol{y} + u_{\text{homo}}(\boldsymbol{x})$, where θ is a function $\theta = v(\boldsymbol{x})$	Methods for Linear Operators [177], [178]	Parameterize $\mathcal{G}$ and u_{homo} with neural networks, which are trained with supervised data (and possibly physics-informed losses).
	Methods for Nonlinear Operators [179]	Discretize the PDEs and use trainable mappings to linearize the target operator, where Green's function formula is subsequently applied to construct the approximation.
Grid-based Operator Learning $G_w(\theta) = \{u(\boldsymbol{x}_i)\}_{i=1}^{N}$, where $\{u(\boldsymbol{x}_i)\}_{i=1}^{N}$ and $\theta = \{v(\boldsymbol{x}_i)\}_{i=1}^{N}$ are discretizations of input and output functions in some **grids**	Convolutional Neural Network [80], [180]	A convolutional neural network is utilized to approximate such an image-to-image mapping, where the loss function is based on supervised data (and possibly physics-informed losses).
	Fourier Neural Operator [181]	Several Fourier convolutional kernels are incorporated into the network structure, to better learn the features in the frequency domain.
	Neural Operator with Attention Mechanism [182], [183], [184]	The attention mechanism is introduced to the design of the network structure, to improve the abstraction ability of the model.
Graph-based Operator Learning $G_w(\theta) = \{u(\boldsymbol{x}_i)\}_{i=1}^{N}$, where $\{u(\boldsymbol{x}_i)\}_{i=1}^{N}$ and $\theta = \{v(\boldsymbol{x}_i)\}_{i=1}^{N}$ are discretizations of input and output functions in some **graphs**	Graph Kernel Network [185]	A graph kernel network is employed to learn such a graph-based mapping.
	Multipole Graph Neural Operator [186]	The graph kernel is decomposed into several multi-level sub-kernels, to capture multi-level neighboring interactions.
	Graph Neural Opeartor with Autogressive Methods [187]	Extend graph neural operators to time-dependent PDEs.

TABLE 3: A brief summary of the methods in the neural operator.

Direct Methods

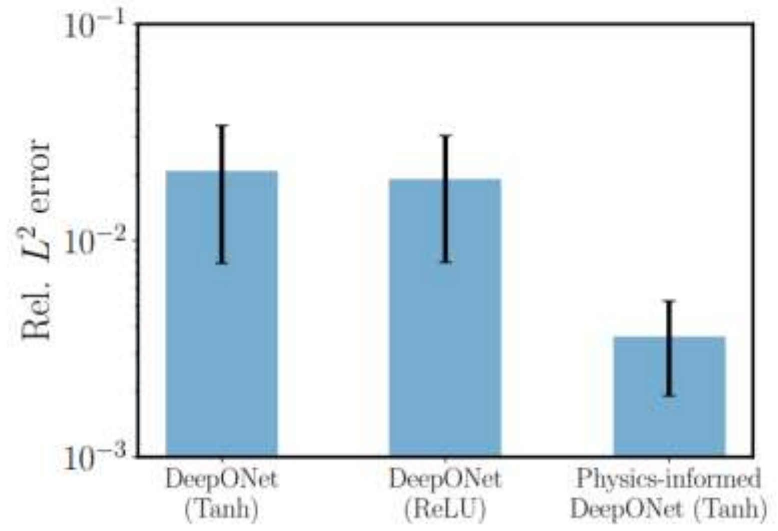
- The direct methods parameterize the mapping by a neural network based on the Universal Approximation Theorem
- This theorem states that a NN with a single hidden layer can approximate accurately any nonlinear continuous functional and (nonlinear) operator
- **DeepONets**
  - Architecture: $$G_w(\theta)(\boldsymbol{x}) = b_0 + \sum_{k=1}^{p} b_k(\theta) t_k(\boldsymbol{x}),$$
  - Loss Function $$\mathcal{L} = \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \left\| G_w(\theta_i)(\boldsymbol{x}_j) - \tilde{G}(\theta_i)(\boldsymbol{x}_j) \right\|^2.$$
  - Difference between neural solvers like PINNS:
    - DeepONet is able to learn latent operators from data, not just the solution to a certain instance of PDEs

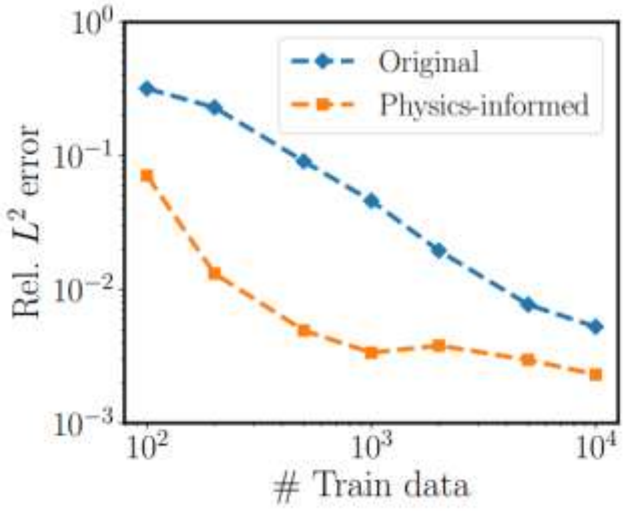
https://www.researchgate.net/publication/350158010_Learning_nonlinear_operators_via_DeepONet_based_on_the_universal_approximation_theorem_of_operators

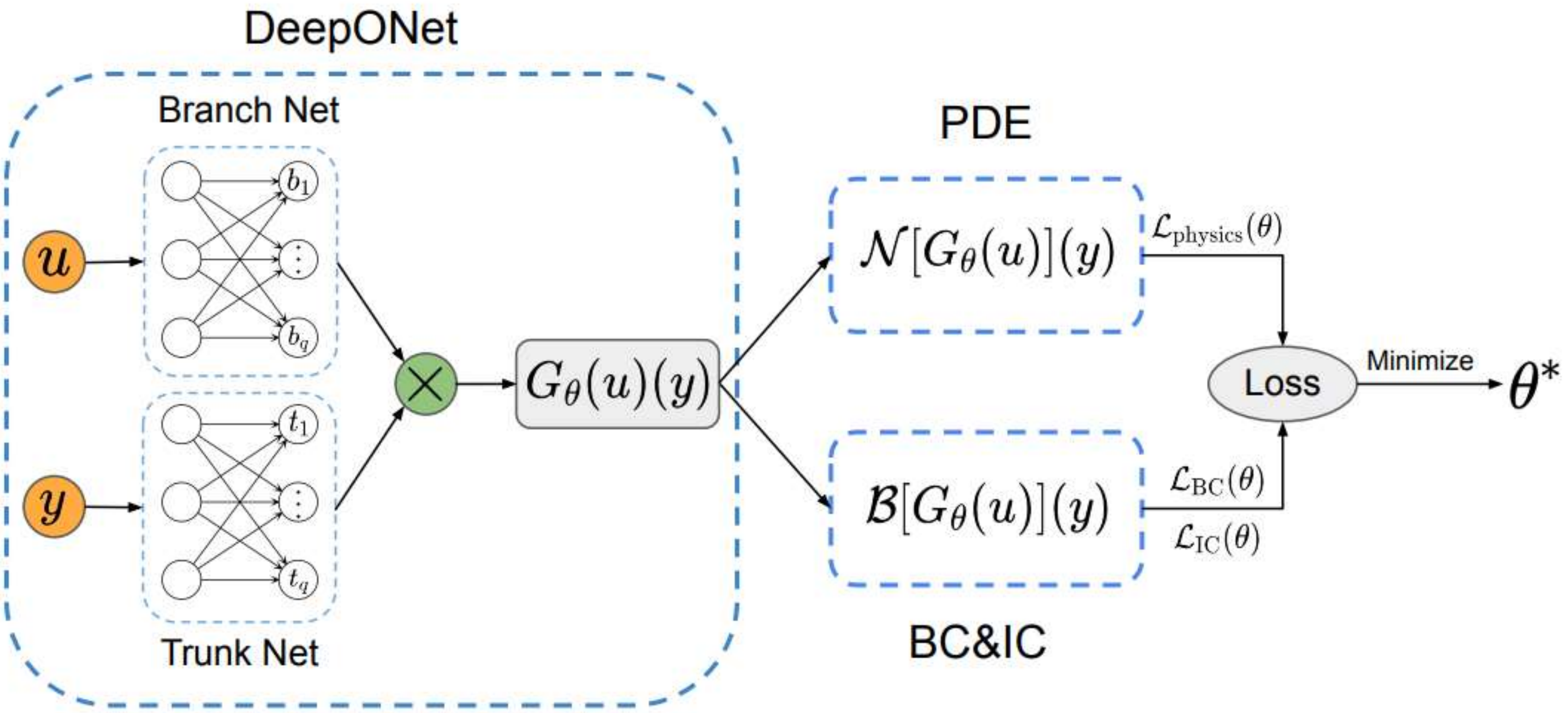
- Limitations of DeepONet:
  - DeepONets may not satisfy the underlying physical principles due to their reliance on large datasets.
  - The need for extensive paired input-output observations, which can be expensive to obtain from complex PDEs.
- **Physics-informed DeepONets** https://arxiv.org/pdf/2103.10974.pdf
  - Physics-informed DeepONets introduce a regularization mechanism during training to ensure physical consistency.
  - Physics-informed DeepONet directly combines the approaches of PINNs and DeepONet
    - DeepONet’s large data demand problem
    - PINNs’ poor apporximation of the solution to complex PDEs

$$\mathcal{L} = \mathcal{L}_{\text{operator}} + \mathcal{L}_{\text{physics}},$$

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_1}\sum_{i=1}^{N_1}\left(\frac{\lambda_r}{N_r}\sum_{j=1}^{N_r}\|\mathcal{F}(u_w;\theta_i)(\boldsymbol{x}_j)\|^2 + \frac{\lambda_i}{N_i}\sum_{j=1}^{N_i}\|\mathcal{I}(u_w;\theta_i)(\boldsymbol{x}_j)\|^2 + \frac{\lambda_b}{N_b}\sum_{j=1}^{N_b}\|\mathcal{B}(u_w;\theta_i)(\boldsymbol{x}_j)\|^2\right),$$






DeepONet
Branch Net
Trunk Net
u
y
b_1
b_q
t_1
t_q
$G_\theta(u)(y)$
PDE
$\mathcal{N}[G_\theta(u)](y)$
$\mathcal{L}_{\text{physics}}(\theta)$
BC&IC
$\mathcal{B}[G_\theta(u)](y)$
$\mathcal{L}_{\text{BC}}(\theta)$
$\mathcal{L}_{\text{IC}}(\theta)$
Loss
Minimize
θ^*

Green’s Function Learning

Matrix: $I = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$

Functions: $\delta(\mathrm{x} - x_0)$

- **Green’s Function:** $\mathcal{L}u(\mathbf{x}) = f(\mathbf{x})$ $\quad u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{x}_0) f(\mathbf{x}_0) d\mathbf{x}_0.$

$\mathcal{L}u(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_i)$ + homogeneous boundary conditions

$\mathcal{L}_x G(\mathbf{x}, \mathbf{x}_0) = \delta(\mathbf{x} - \mathbf{x}_0)$

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{x}_0)\delta(\mathbf{x}_0 - \mathbf{x}_i) d\mathbf{x}_0 = G(\mathbf{x}, \mathbf{x}_i).$$

$\mathcal{L}_x G(\mathbf{x}, \mathbf{x}_0) = 0, \quad \text{when } \mathbf{x} \neq \mathbf{x}_0,$

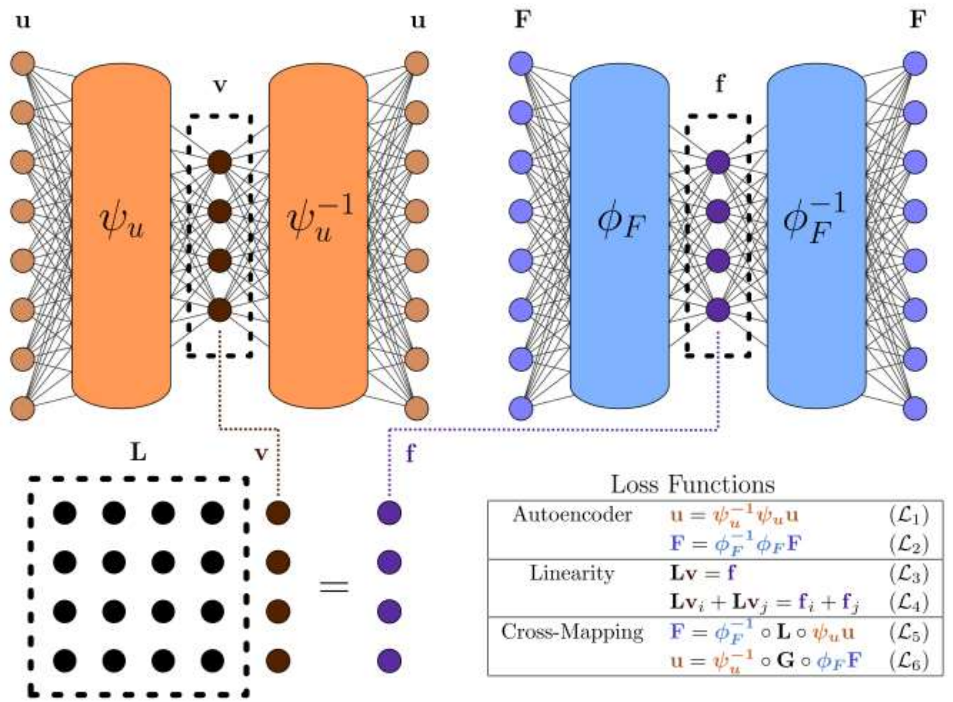
- Green's function learning aims to approximate the challenging analytical expression of Green's function (G(x, y)) using neural networks.
- Focusing on mapping the forcing term (f) to the solution (u) based on the structure of the Green's function

- **Green’s Function Learning for Linear Operators**
  - This equation can be directly used or linear opreators

$$u(\boldsymbol{x}) = \int_{\Omega} \mathcal{G}(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} + u_{\text{homo}}(\boldsymbol{x}),$$

  - Advantages of Green’s function learning:
    - Contains more priors about the physical system
    - Mathematically easier to approximate Green’s function than latent opreator
    - Structure of Green’s function can be employed felxibly, since many physical properties can be encoded into the network architecture to improve accuracy
  - Limitations of Green’s function learning :
    - Limited to special class of PDEs
    - Input dimension of Green’s function is twice the spatial dimension

- **Green’s Function Learning for Non-linear Operators**
  - Problem occurs when using non-linear operator and a linear boundary condition
  - **DeepGreen** Approach
    - DeepGreen employs deep autoencoders to learn invertible coordinate transformations that linearize the nonlinear BVPs.



Loss Functions

Autoencoder	$\mathbf{u} = \psi_u^{-1}\psi_u \mathbf{u}$	$(\mathcal{L}_1)$
	$\mathbf{F} = \phi_F^{-1}\phi_F \mathbf{F}$	$(\mathcal{L}_2)$
Linearity	$\mathbf{Lv} = \mathbf{f}$	$(\mathcal{L}_3)$
	$\mathbf{Lv}_i + \mathbf{Lv}_j = \mathbf{f}_i + \mathbf{f}_j$	$(\mathcal{L}_4)$
Cross-Mapping	$\mathbf{F} = \phi_F^{-1} \circ \mathbf{L} \circ \psi_u \mathbf{u}$	$(\mathcal{L}_5)$
	$\mathbf{u} = \psi_u^{-1} \circ \mathbf{G} \circ \phi_F \mathbf{F}$	$(\mathcal{L}_6)$

Figure 2: DeepGreen architecture. Two autoencoders learn invertible coordinate transformations that linearize a nonlinear boundary value problem. The latent space is constrained to exhibit properties of a linear system, including linear superposition, which enables discovery of a Green’s function for nonlinear boundary value problems.

$$\begin{aligned} \mathcal{F}_L(u) &= f, \boldsymbol{x} \in \Omega, \\ \mathcal{B}_L(u) &= g, \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

Grid-based Operator Learning

- Latent operator can also be formalize as grid-based mapping

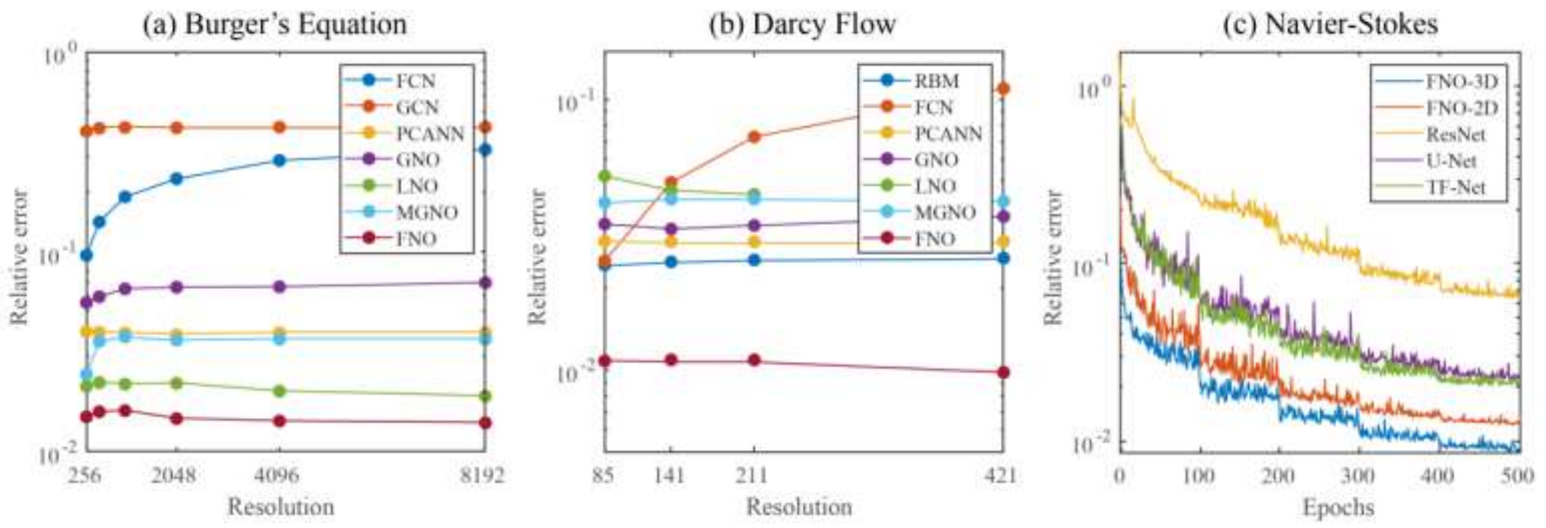
  (Grid-based operator) $\tilde{G}: \theta \mapsto \{u(\boldsymbol{x}_i)\}_{i=1}^N,$ where u is the solution of PDEs under parameters θ and x

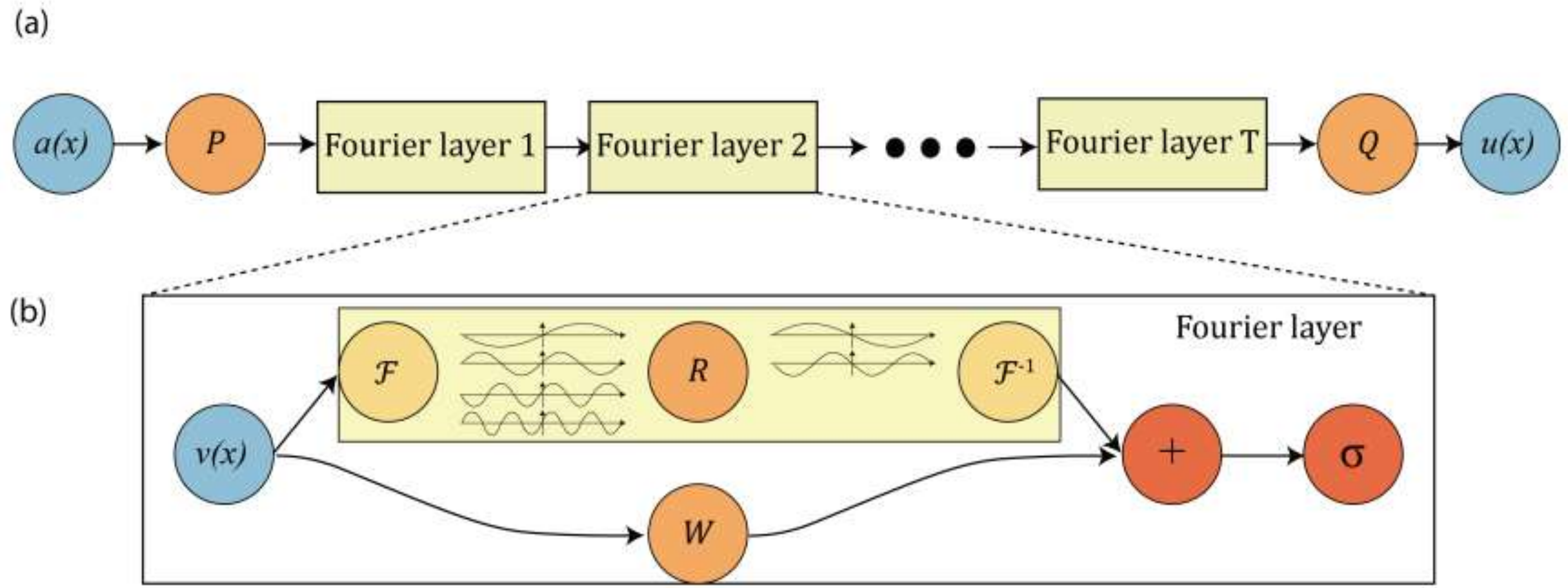
- If θ is a function and shares the same grid with u

$$\tilde{G}: \theta = \{v(\boldsymbol{x}_i)\}_{i=1}^N \mapsto \{u(\boldsymbol{x}_i)\}_{i=1}^N,$$

- If the points are uniformly distributed(regular grid) notations can be replaced with tensors. Such operators are also called image-to-image mappings

- **Concolutional Neural Networks**
  - Through convolutional operations, convolutional neural networks skillfully capture the structure within the given "images" during training, enabling them to acquire a concise representation of the essential physical principles
  - Problem: Curse of dimensionality.
- **Fourier Neural Operators**
  - FNO is another architecture for learning image-toimage mappings which considers the features via the Fourier transformation.
  - Using the Fourier transformation, FNOs can effectively abstract the features, which makes their performance better than other architectures





(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

Figure 2: **top:** The architecture of the neural operators; **bottom:** Fourier layer.