Rethinking Physics Informed Neural Networks

Presented by **Amir Gholami**,

In collaboration with:

A. Krishnapriyan, S. Subramanian, S. Zhe, M. Kirby, M. Mahoney, K. Keutzer

University of California Berkeley, University of Utah

Babuška Forum Seminar, UT Austin, Oct 2021






Outline

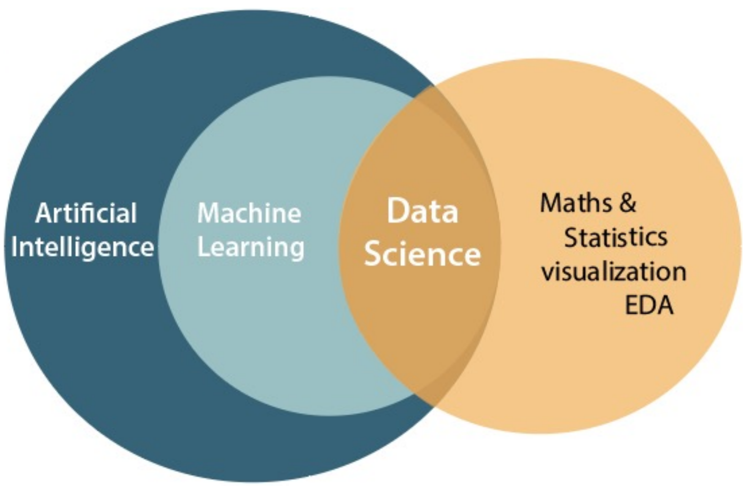
- **Introduction**
- Physics Informed Neural Networks
- Challenges associated with PINNs
- Conclusions and Future Work

Third Pillar: Computational Science

AI: Techniques that enable machines to have human-level of intelligence

ML: Learn patterns in data and perform predictions

Data Science: Methods to draw insights from data
(through math, stats, visualization, etc.)



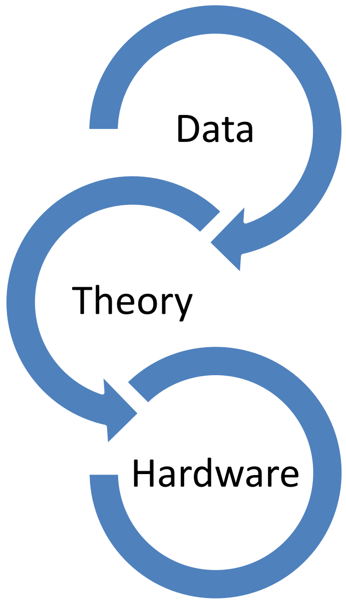


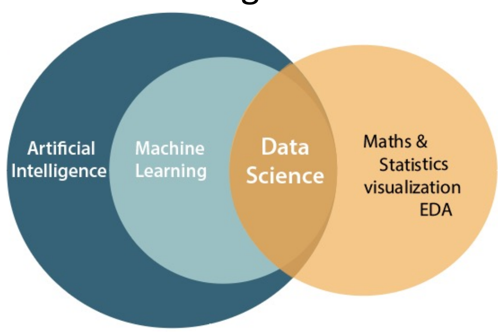
Illustration from H. Sri Kovela

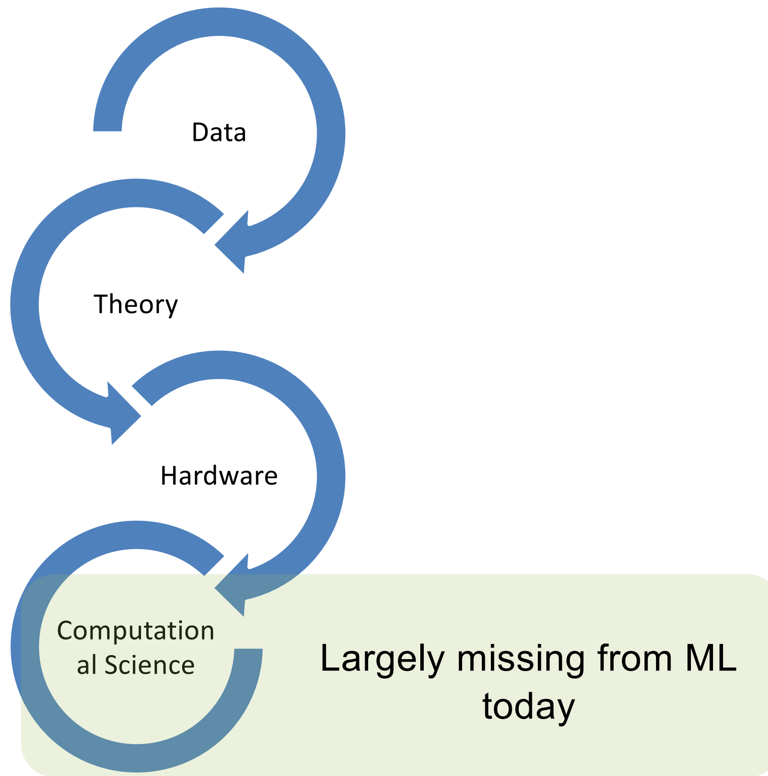
Third Pillar: Computational Science

Computational Science is an important tool that we can use to incorporate physical invariances into learning, but until recently it was missing from mainstream ML.

"**Computational Science** can **analyze past events** and **look into the future**. It can explore the effects of thousands of scenarios for or in lieu of actual experiment and be used to study events beyond the reach of expanding the boundaries of experimental science"
–Tinsley Oden, 2013

To make further progress in ML it is crucial that we incorporate computational science into learning.

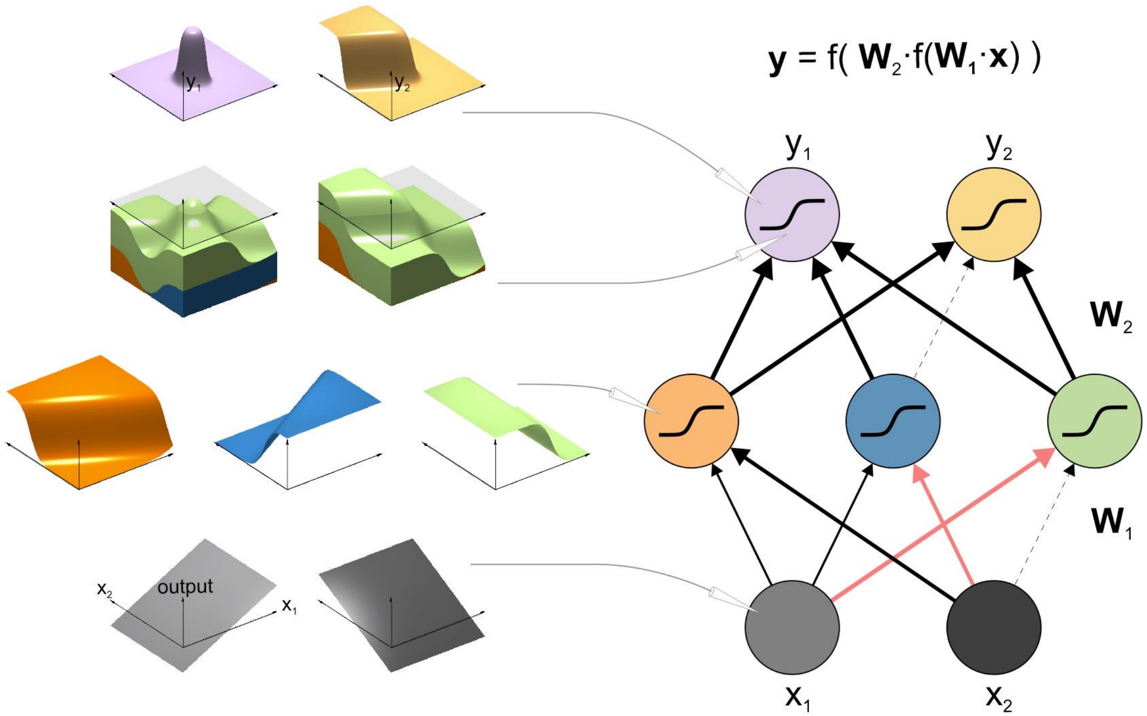


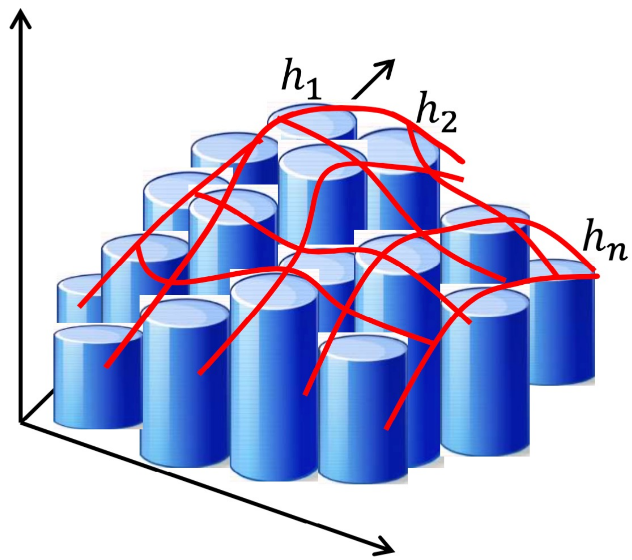


Dr. J. Tinsley Oden's Commemorative Speech: "THE THIRD PILLAR: The Computational Revolution of Science and Engineering", Honda Prize, 2013.

MLPs are Universal Function Approximators

Now let's take a step back and see what are Neural Networks?

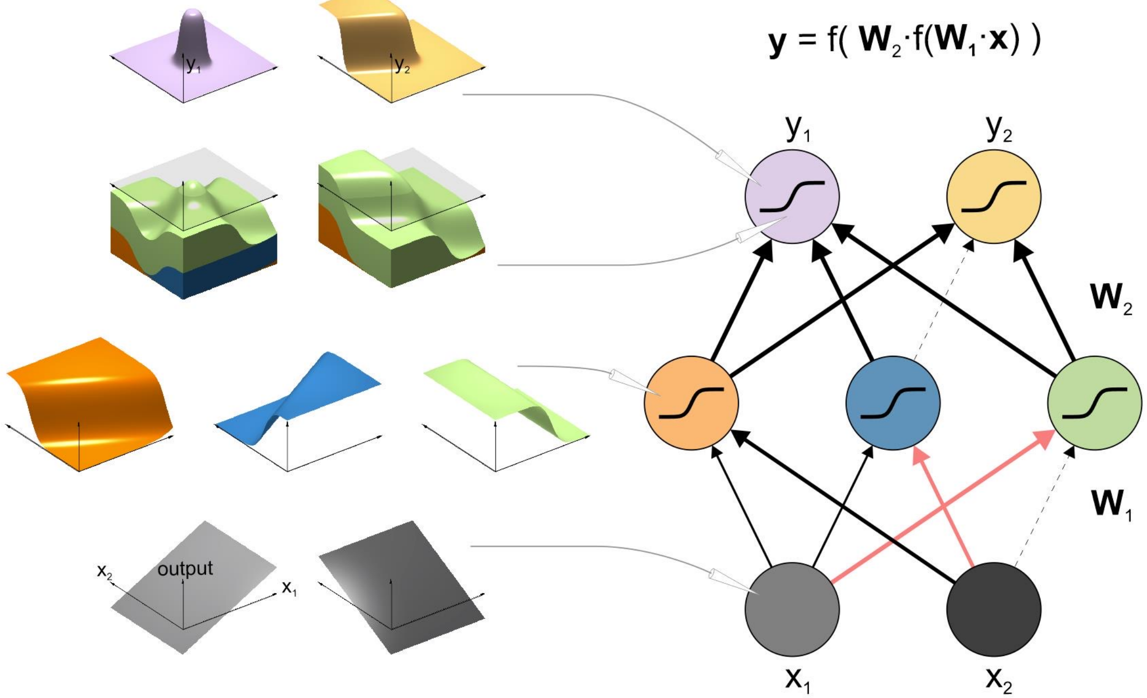




G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems, 2(4):303–314, 1989.*
K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989.
Kriegeskorte N, Golan T. Neural network models and deep learning-a primer for biologists. arXiv preprint arXiv. 1902.

MLPs are Universal Function Approximators

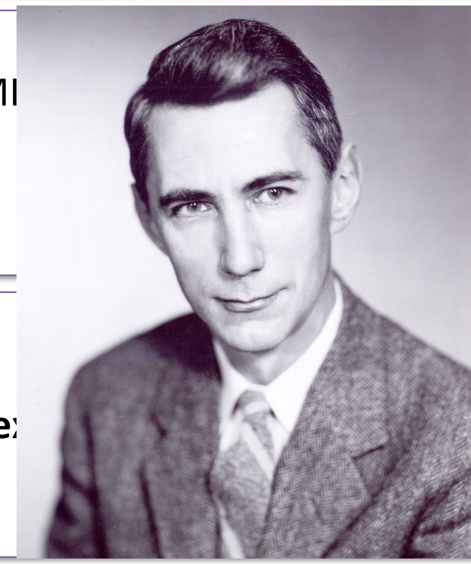
Now let's take a step back and see what are Neural Networks?



Ml ons

e

ex of



Theorem: There exists a Boolean function of n>2 variables that requires at least $2^n/n$ Boolean gates, regardless of depth!

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems, 2(4):303–314, 1989.*
K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989.
Kriegeskorte N, Golan T. Neural network models and deep learning-a primer for biologists. arXiv preprint arXiv. 1902.

Universal Function *Approximation*

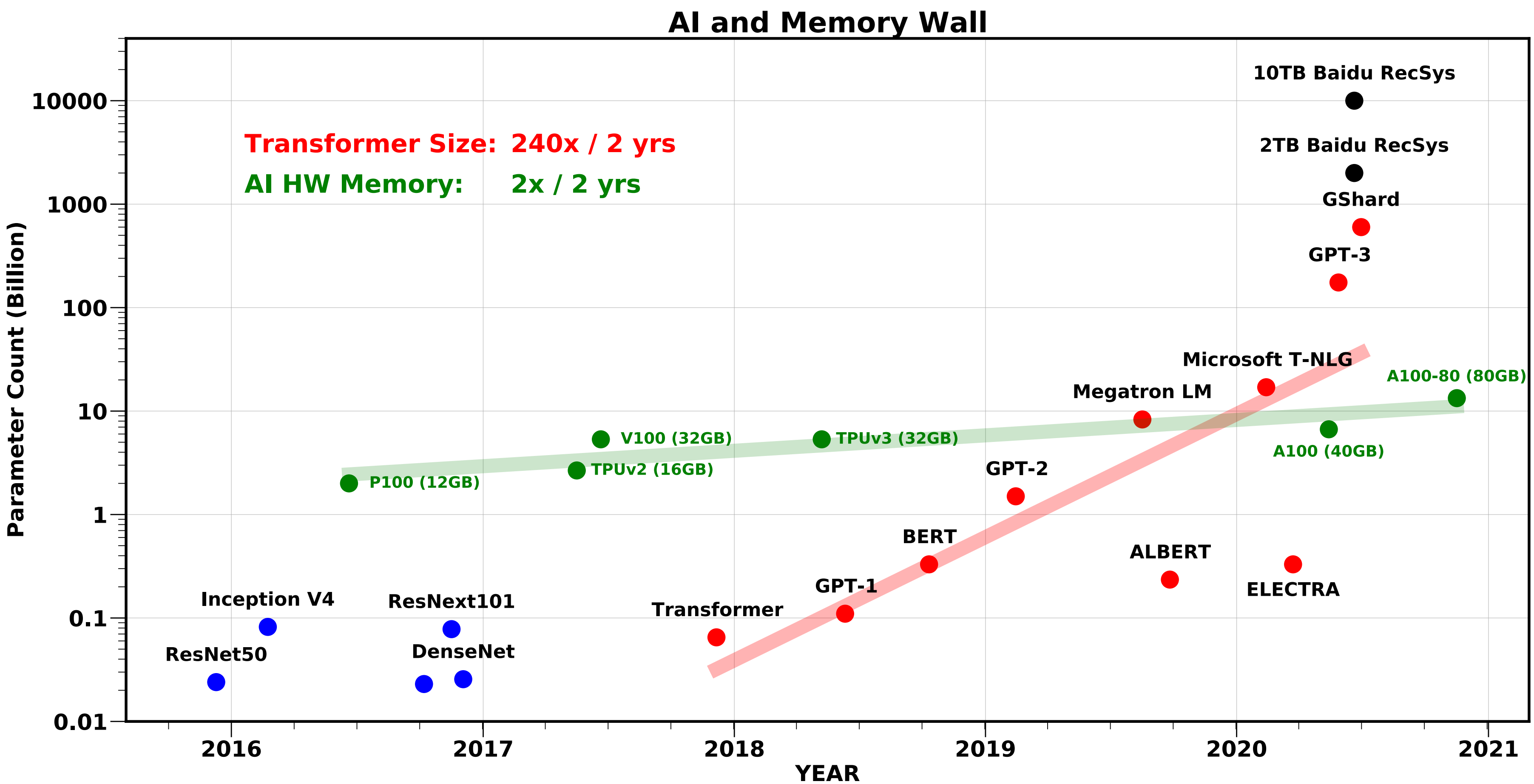
Important: Universal Function Approximation theorem only considers **approximation error,** and not trainability and/or generalizability of the NN.

We can broadly characterize NN performance into three main types:

- **Approximation error** to ground truth function
- **Generalization** to unseen data
- **Trainability** of the model

- **Universal approximation theorem only considers the first one.**
- Moreover, it provides no method to train a model to get that approximation
  - naïve method using the basis function in the previous slide can require exponentially large number of neurons even for simple functions

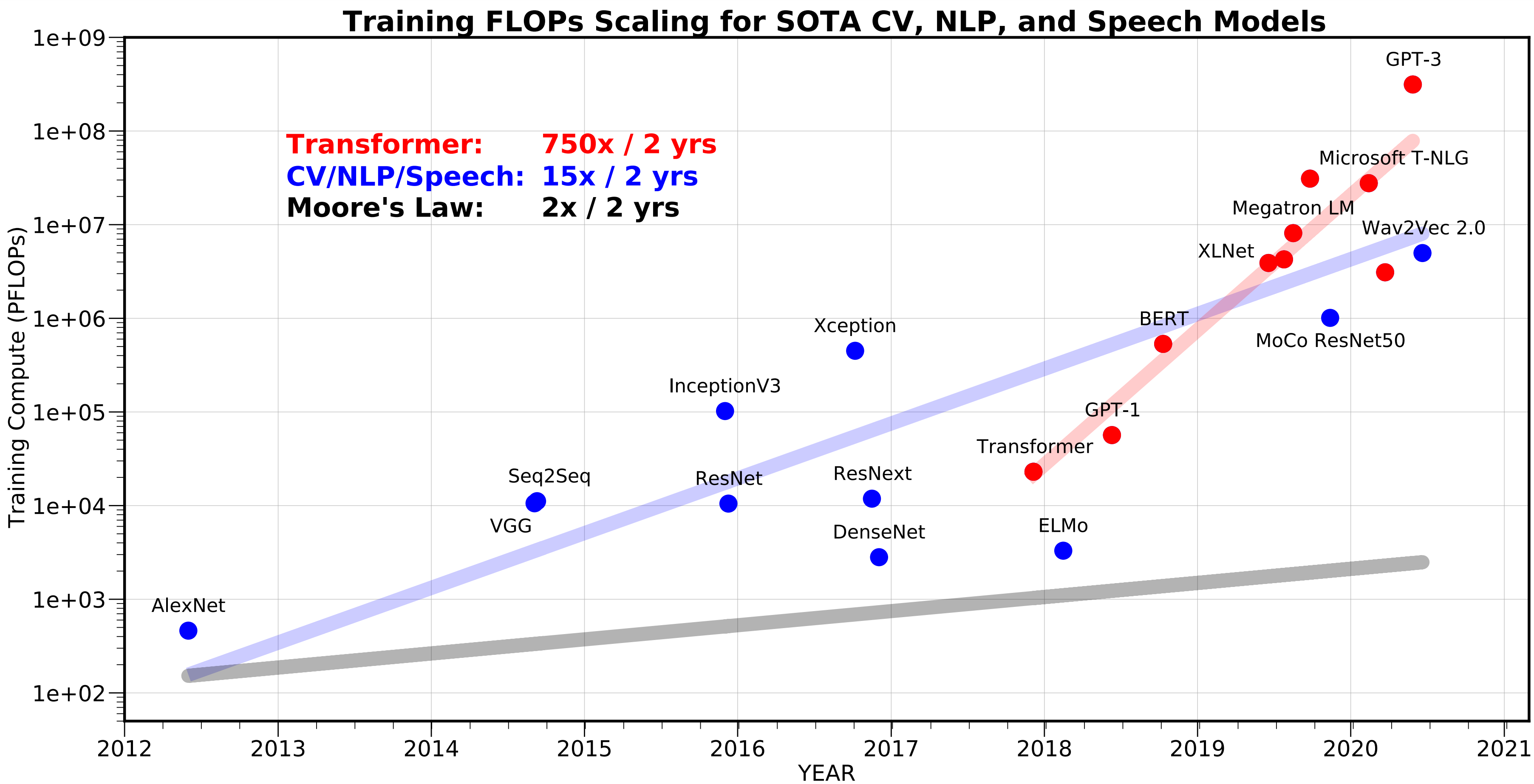
What Works in Practice?
Extremely Overparameterized Models



Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, Kurt Keutzer, [AI and Memory Wall](), Riselab Medium Blogpost, 2021.

© Pallas Group, UCB

What Works in Practice?
Exponentially Expensive Models to Train



Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W. Mahoney, Kurt Keutzer, AI and Memory Wall, Riselab Medium Blogpost, 2021.

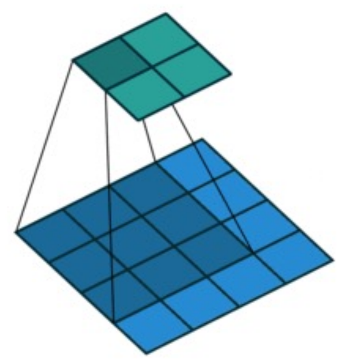
Improving Approximation Error with Invariances

- By incorporating domain specific invariances, we can significantly improve the generalization properties of NNs.

Examples in computer vision:

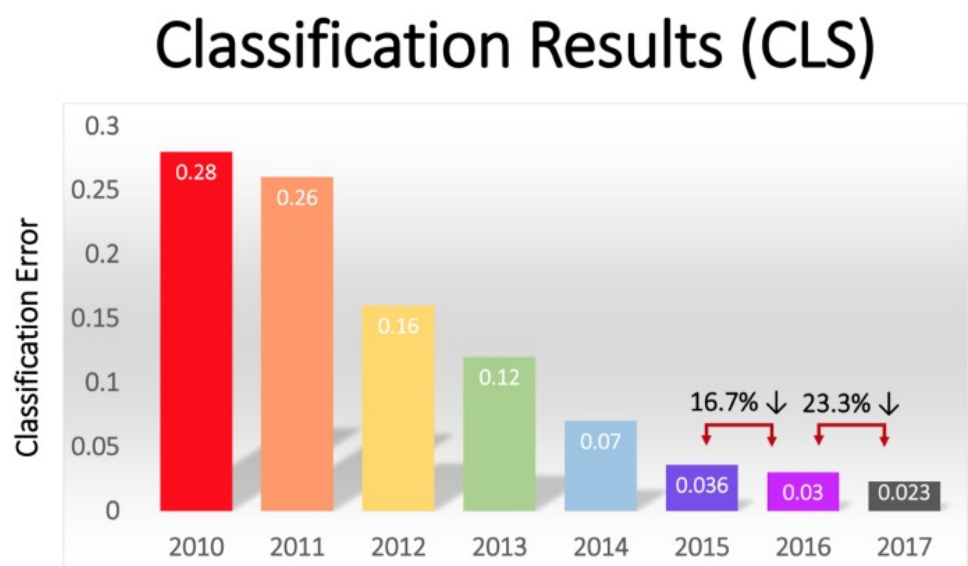
- Translational invariance => Use of convolutional layers
- Spatial invariance (be able to recognize features regardless of skew, angle, or direction)

Spatial and Translational Invariance



Improving Approximation Error with Invariances

One reason CNNs has been so successful in vision is the translational invariance that is incorporated in them by design!



ImageNet Classification Error

Chart credit: Prudhvi Gnv

Physical Laws as Additional Invariances

There are many more invariances other than translational invariance:

- Physics-based laws of nature: Conservation of mass, momentum, energy

How can these invariances help?

This is the common view of how Physics can help training. But even in the Big Data regime incorporating Physics can be helpful

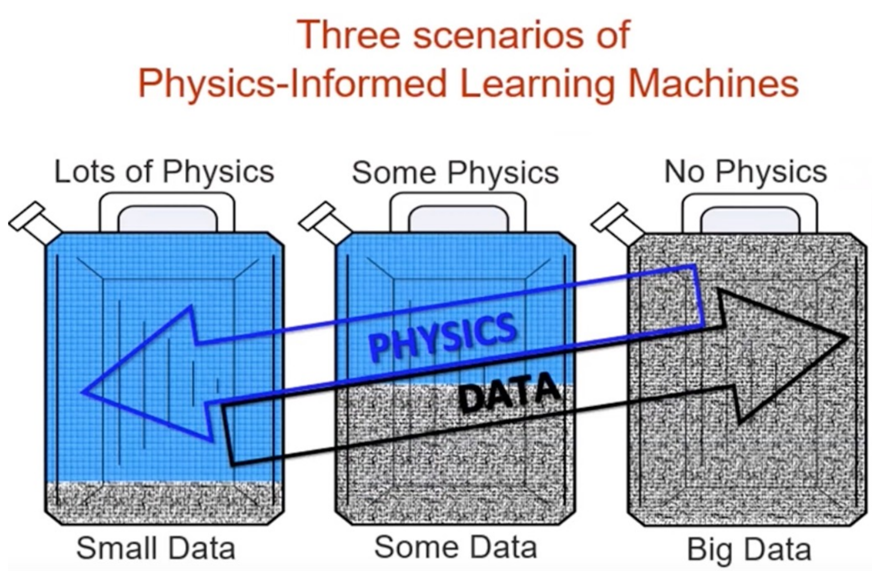


Illustration Credit: Prof. Karniadakis

Physical Laws as Additional Invariances

There are many more invariances other than translational invariance:

- Physics-based laws of nature: Conservation of mass, momentum, energy

How can these invariances help?

Less Data

More Data

Physical Invariances can help Improve Generalization or regularize training

Physical Invariances can help make the model easier to train with less parameters

The main question is how can we incorporate these invariances into learning?

Outline

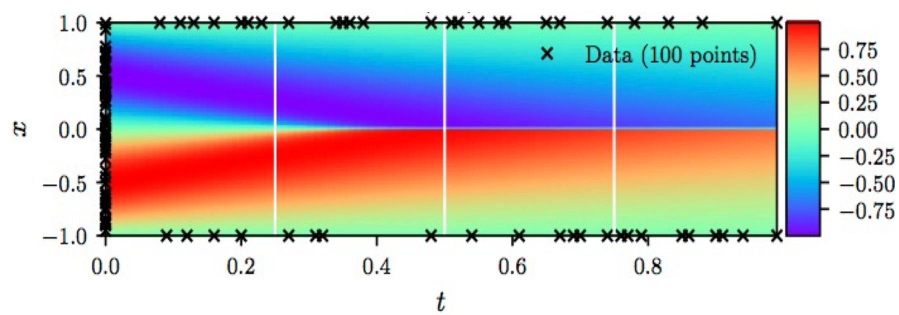
- Introduction
- **Physics Informed Neural Networks**
- Challenges associated with PINNs
- Conclusions and Future Work

Methods for Incorporating Physics into Learning

- **Method 1: Train on massive amount of data (and hope) that the NN trains with good performance/generalization**

Let's use Burgers' equation as an example (a fundamental PDE used for modeling fluid dynamics, non-linear acoustics, gas dynamics, traffic flow, etc.)

$$u_t + uu_x - u_{xx} = 0, \quad x \in (-1, 1), t \in (0, 1]$$
$$+\text{Initial/Boundary Conditions}$$

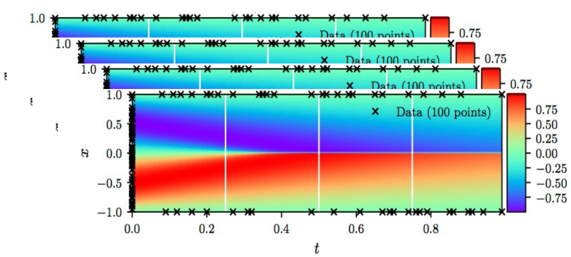


[1] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019 Feb 1;378:686-707.
[2] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks. 1998 Sep;9(5):987-1000.

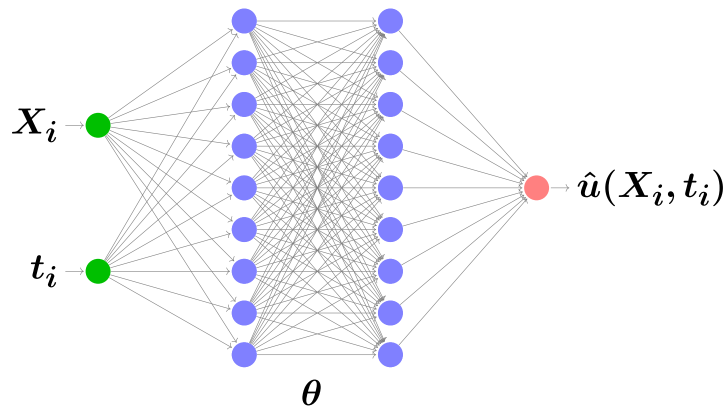
Methods for Incorporating Physics into Learning

➢ **Method 1: Train on massive amount of data (and hope) that the NN trains with good performance/generalization**

Obtain/Simulate a lot of data



Train the NN on this dataset



$$\min_{\theta} \mathcal{L} = \sum_i \|\hat{u}_i - u_i\|_2^2$$

Main problems:

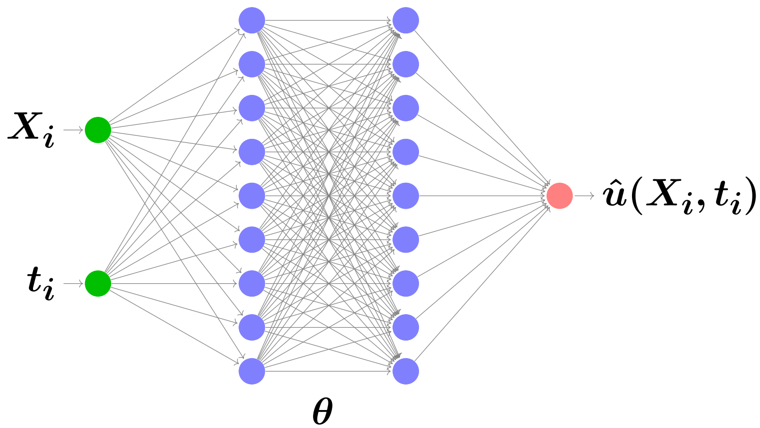
- No guarantee that the model obeys the conservation laws
- May require a lot of training data and obtaining/simulating these data is not always feasible

[1] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019 Feb 1;378:686-707.
[2] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks. 1998 Sep;9(5):987-1000.

Methods for Incorporating Physics into Learning

- **Method 2:** Enforce Physical laws as hard constraints either in:
  - NN Architecture: This is an open problem
  - Optimization: Very difficult to train the NN with such constraints

Train the NN on this dataset



$$\min_{\theta} \mathcal{L} = \sum_i \|\hat{u}_i - u_i\|_2^2,$$

$$s.t. \quad u_t + u u_x - u_{xx} = 0.$$

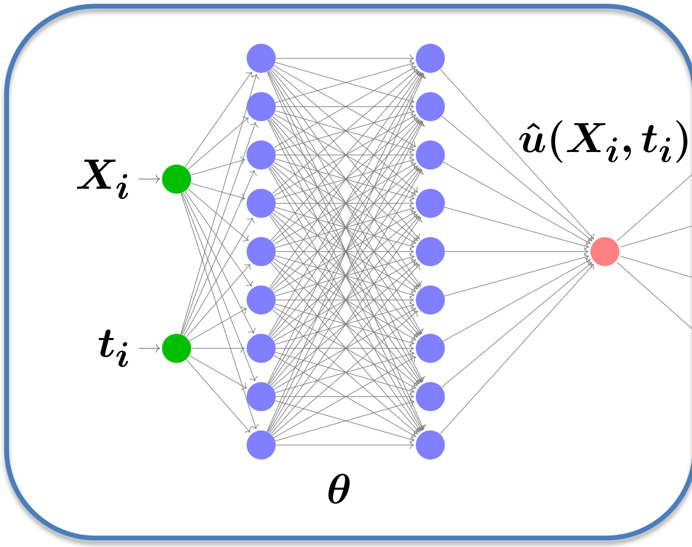
[1] Xu K, Darve E. Physics constrained learning for data-driven inverse modeling from sparse observations. arXiv preprint arXiv:2002.10521. 2020

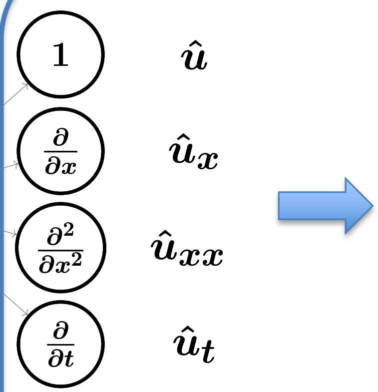
Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019 Feb 1;378:686-707.

[2] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks. 1998 Sep;9(5):987-1000. (only satisfies BCs exactly)

Methods for Incorporating Physics into Learning

- **Method 3:** Use penalty methods and add the PDE residual to the loss as a **soft constraint**.





$$\hat{u}_t + \hat{u}\hat{u}_x - \hat{u}_{xx} \in \Omega$$
$$\hat{u}(x, t = 0)$$
$$\hat{u}(x, t) \in \Gamma$$

Data Loss Function:

$$\mathcal{L}_u = \|\hat{u} - u\|_2^2$$

Physics Loss Function:

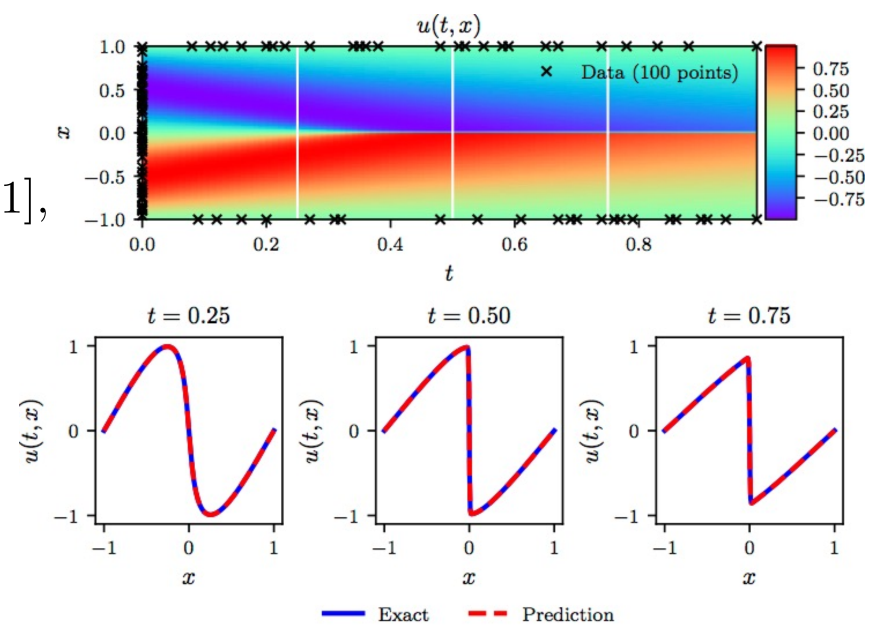
$$\mathcal{L}_{\mathcal{F}} = \|\hat{u}_t + \hat{u}\hat{u}_x - \hat{u}_{xx}\|_2^2$$

$$\min_{\theta} \mathcal{L} = \mathcal{L}_u + \lambda_{\mathcal{F}} \mathcal{L}_{\mathcal{F}}$$

Physics Informed Neural Networks

- **Method 3:** Use penalty methods and add the PDE residual to the loss as a **soft constraint**.
  - Easy to implement, and works with automatic differentiation with any NN architecture
  - Does not require a mesh or a numerical solver for the PDE
  - Can (in theory) work for high dimensional problems, and complex PDEs
    - For example, PDEs containing integral operators which are difficult to solve with finite difference methods.

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1],\ t \in [0, 1],$$
$$u(0, x) = -sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$



Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019.

Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks. 1998 Sep;9(5):987-1000.

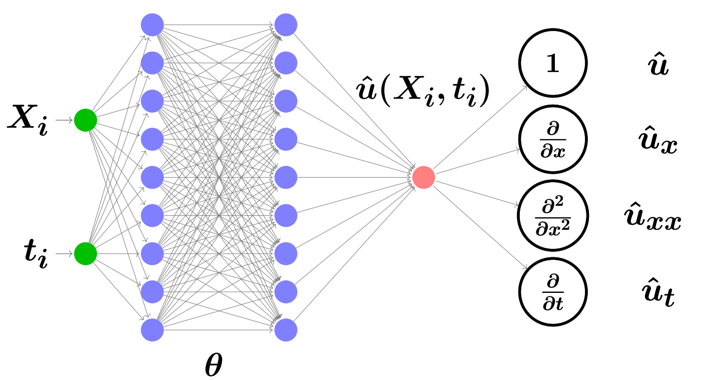
But this is not the entire story

- There are a lot of subtleties in adding PINN's soft-constraint.

To study this, we chose three families of PDEs:

- Advection (aka wave equation)
- Reaction
- Reaction-Diffusion

For all of these cases we observed that **PINNs failed to learn the relevant Physics**, since there are many moving parts in this problem, than what appears at first.

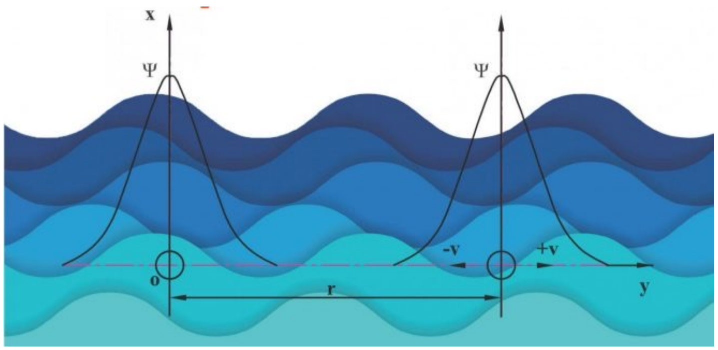


$$\min_{\theta} \mathcal{L} = \|\hat{u} - u\|_2^2 + \lambda_{\mathcal{F}} \|\hat{u}_t + \hat{u}\hat{u}_x - \hat{u}_{xx}\|_2^2$$

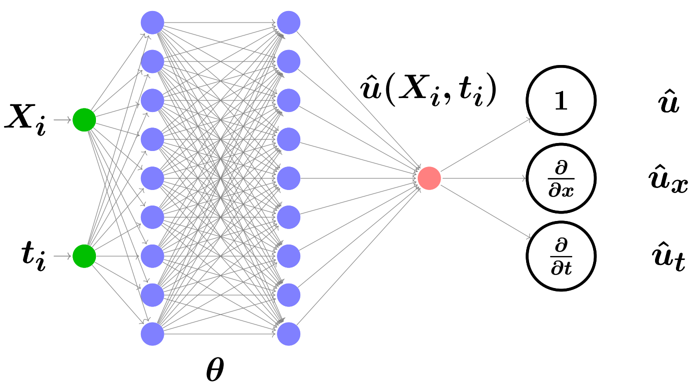
Outline

- Introduction
- Physics Informed Neural Networks
- **Challenges associated with PINNs**
- Conclusions and Future Work

Advection Equation



$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \qquad x \in \Omega,\ t \in [0, T],$$
$$u(x, 0) = sin(x), \quad x \in \Omega,$$
$$u(0, t) = u(2\pi, t) \quad t \in [0, T].$$



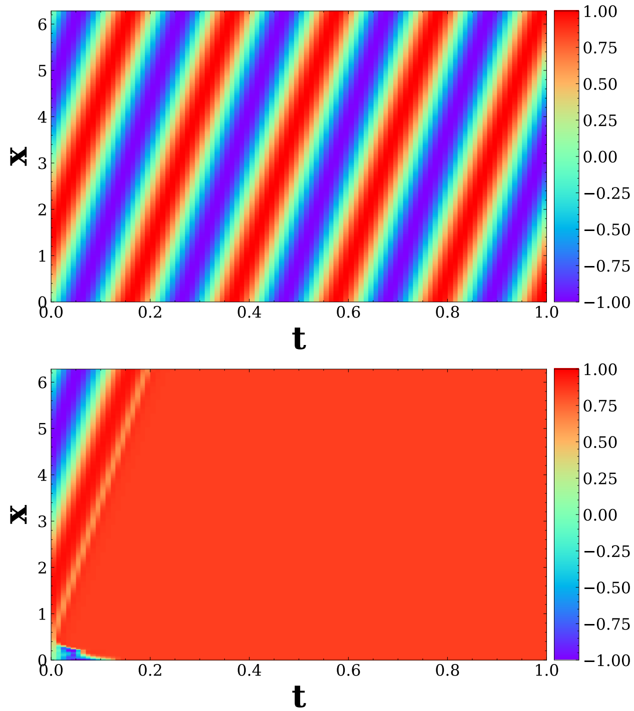
$$\min_{\theta} \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2 \quad \text{PDE Residual}$$
$$+\|\hat{u}(x, 0) - sin(x)\|_2^2 \quad \text{Initial Condition}$$
$$+\|\hat{u}(x = 2\pi) - \hat{u}(x = 0)\|_2^2 \quad \text{Boundary Condition}$$

Krishnapriyan AS, **Gholami* A**, Zhe S, Kirby RM, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. NeurIPS, 2021.*

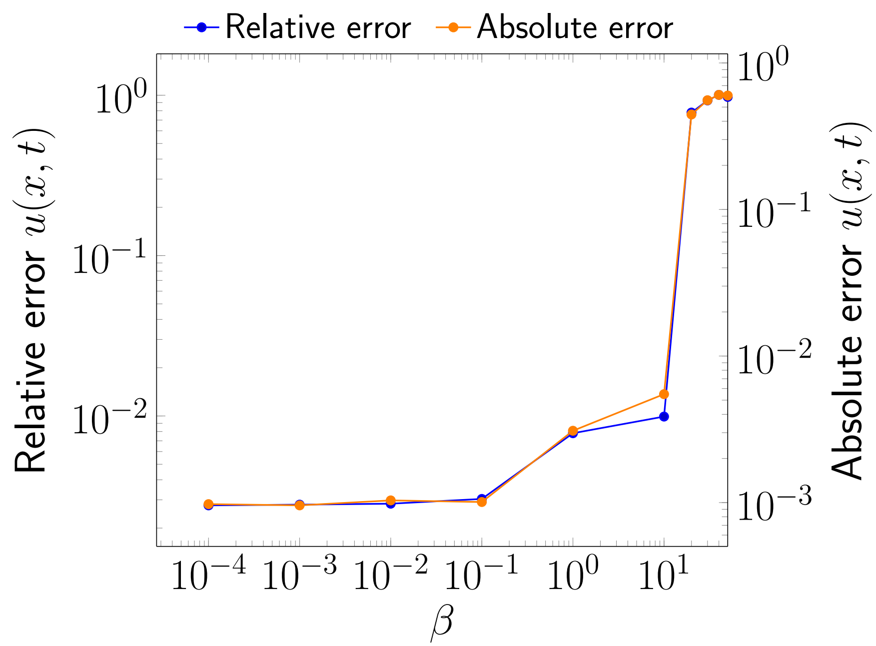
PINN can fail to learn Advection

Exact Solution

PINN Prediction

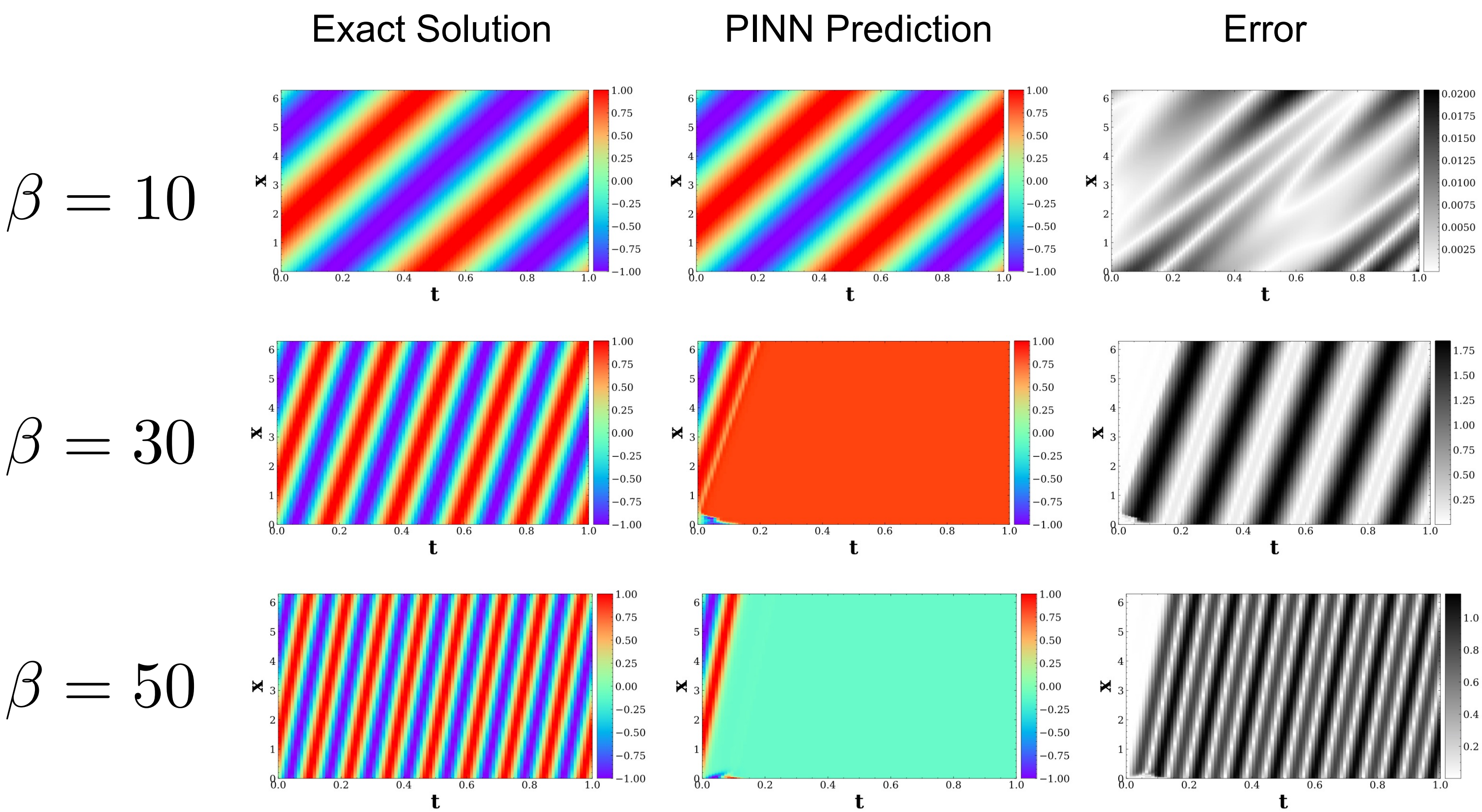


$\beta = 30$



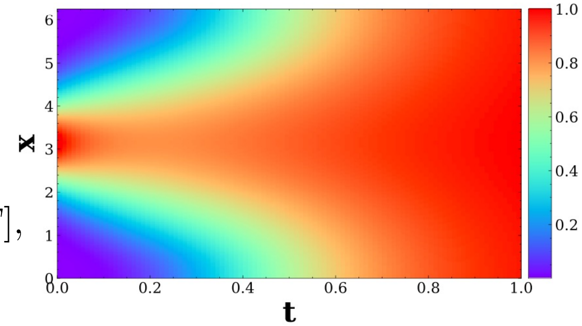
Krishnapriyan AS, **Gholami* A**, Zhe S, Kirby RM, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. NeurIPS, 2021.*

PINN can fail to learn Advection

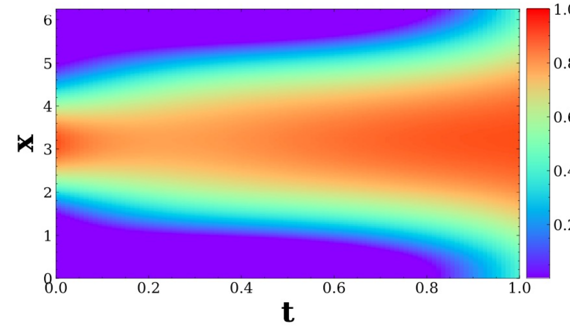


PINN can fail to Learn Fisher Equation

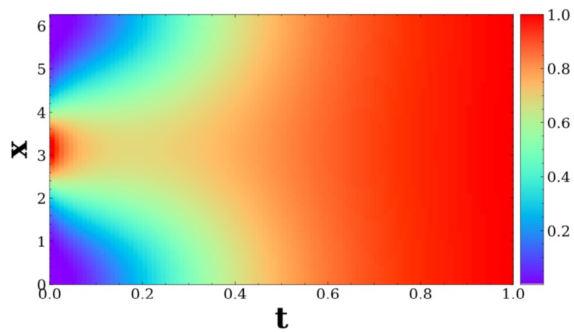
$$
\begin{aligned}
&\frac{\partial u}{\partial t} - \rho u(1-u) - \nu \frac{\partial^2 u}{\partial x^2} = 0, \quad && x \in \Omega,\ t \in (0, T], \\
&u(x, 0) = e^{-\frac{(x-\pi)^2}{2(\pi/4)^2}}, && x \in \Omega, \\
&u(0, t) = u(2\pi, t), && t \in [0, T].
\end{aligned}
$$



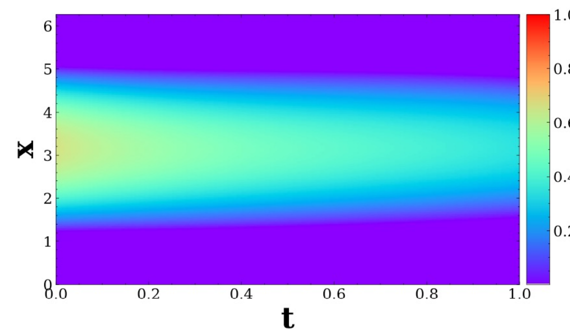
(a) *Exact solution for* $\rho = 5,\ \nu = 2$



(b) *PINN solution for* $\rho = 5,\ \nu = 2$



(c) *Exact solution for* $\rho = 5,\ \nu = 5$

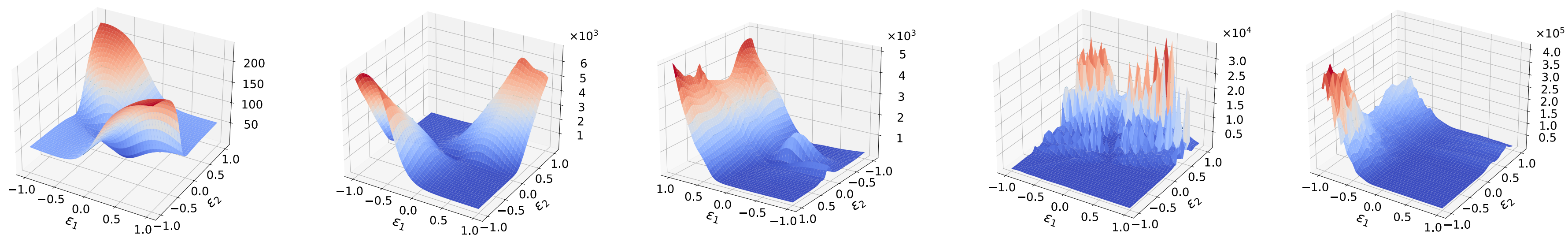


(d) *PINN solution for* $\rho = 5,\ \nu = 5$

Krishnapriyan AS, **Gholami* A**, Zhe S, Kirby RM, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. NeurIPS, 2021.*

Characterizing the Failure Points: Loss Landscape

- To better understand the problem, let's first look at the optimization loss landscape.
- The loss function that we are dealing with is quite complex and non-convex, so there is no guarantee that the optimizer can find a good solution.



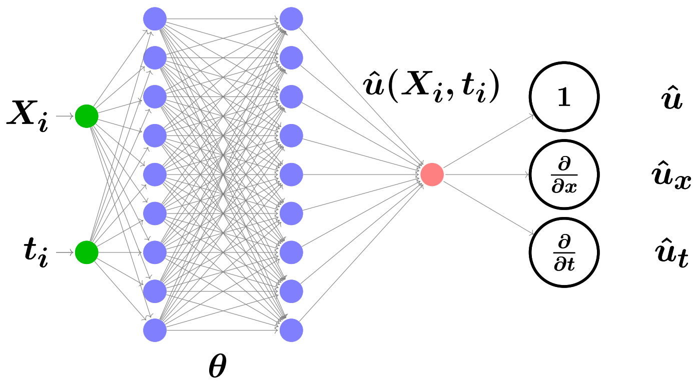
(a) $\beta = 1.0$ (b) $\beta = 10.0$ (c) $\beta = 20.0$ (d) $\beta = 30.0$ (e) $\beta = 40.0$

β	1	10	20	30	40
Relative error	7.84×10^{-3}	1.08×10^{-2}	7.50×10^{-1}	8.97×10^{-1}	9.61×10^{-1}
Absolute error	3.17×10^{-3}	6.03×10^{-3}	4.32×10^{-1}	5.42×10^{-1}	5.82×10^{-1}

As we increase the wave speed, the loss landscape becomes increasingly harder to optimize

Characterizing Failure Points: Loss Landscape

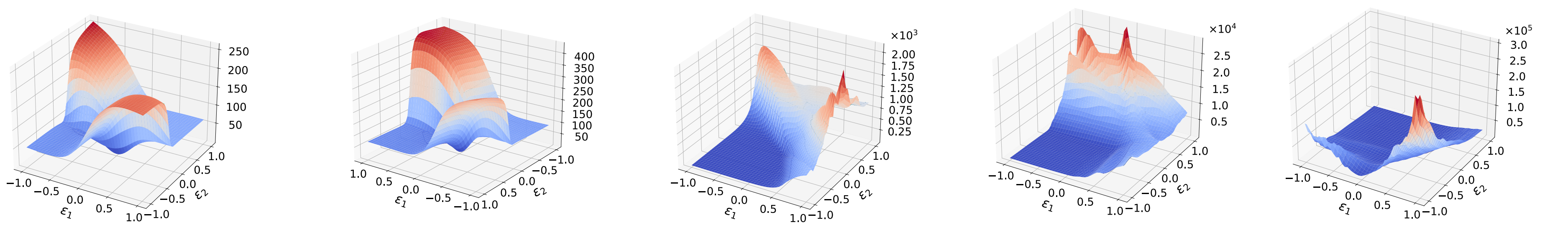
- What if we adjust the weight of the PDE loss?
- Would the loss landscape become easier to optimize if we use a smaller or larger weight?



$$\min_{\theta} \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2 \quad \text{PDE Residual}$$

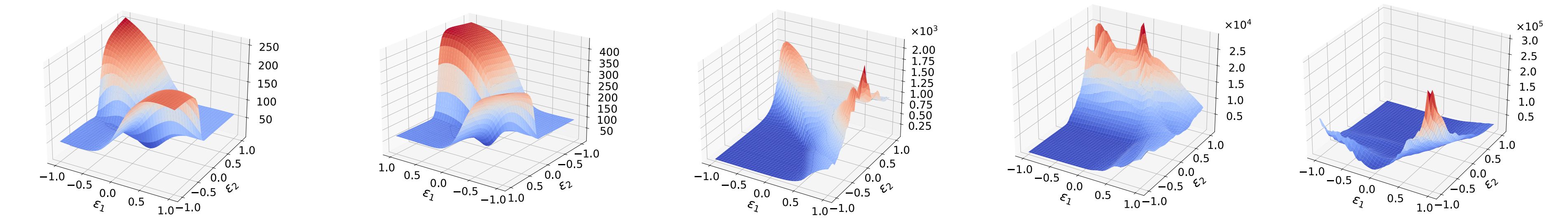
$$+\|\hat{u}(x, 0) - sin(x)\|_2^2 \quad \text{Initial Condition}$$

$$+\|\hat{u}(x = 2\pi) - \hat{u}(x = 0)\|_2^2 \quad \text{Boundary Condition}$$



(a) $\lambda = 1 \times 10^{-6}$ (b) $\lambda = 1 \times 10^{-5}$ (c) $\lambda = 1 \times 10^{-3}$ (d) $\lambda = 1 \times 10^{-1}$ (e) $\lambda = 1 \times 10^{1}$

Characterizing Failure Points: Loss Landscape



(a) $\lambda = 1 \times 10^{-6}$ (b) $\lambda = 1 \times 10^{-5}$ (c) $\lambda = 1 \times 10^{-3}$ (d) $\lambda = 1 \times 10^{-1}$ (e) $\lambda = 1 \times 10^{1}$

λ	1×10^{-6}	1×10^{-5}	1×10^{-3}	1×10^{-1}	1×10^{1}
Relative error	1.69	1.65	1.00	1.08	0.982
Absolute error	0.987	0.987	0.623	0.647	0.595

$$\min_\theta \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2 \quad \text{PDE Residual}$$

$$+\|\hat{u}(x,0) - sin(x)\|_2^2 \quad \text{Initial Condition}$$

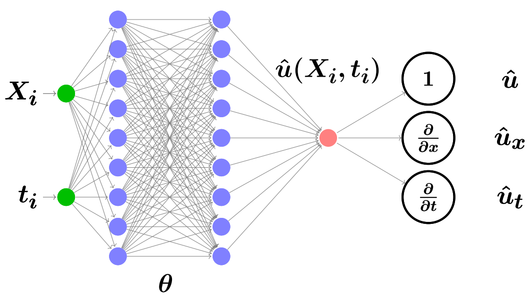
$$+\|\hat{u}(x=2\pi) - \hat{u}(x=0)\|_2^2 \quad \text{Boundary Condition}$$

As we reduce λ the optimization gets easier but PINN's solution has ~100% error

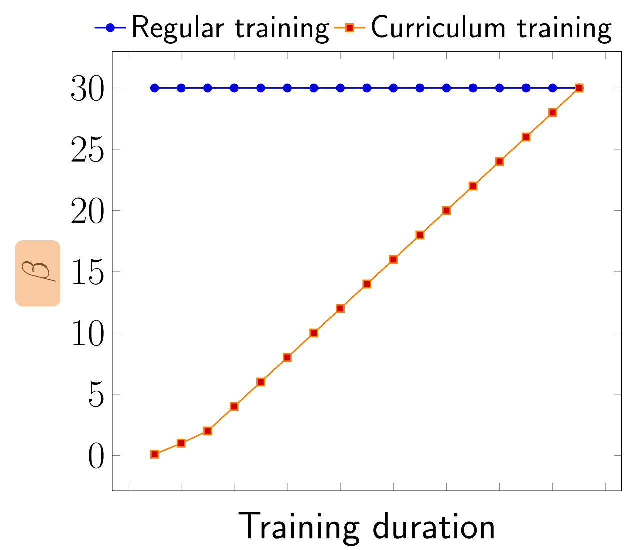
Rethinking PINNs: Curriculum Learning

- Curriculum Learning: Start with **simple physical constraints** and progressively introduce the complexities throughout learning
  - First let the NN learn the simple problems, before penalizing it for learning the target PDE

Example: Initially train the NN with small velocities, and slowly increase the velocity to the target value

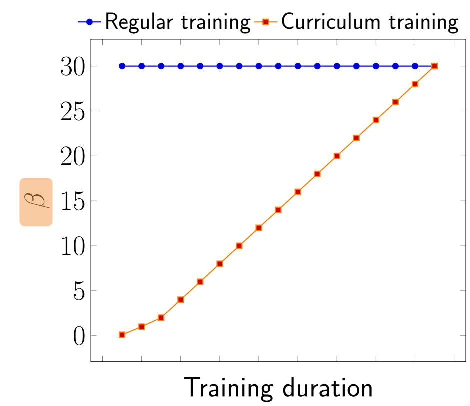


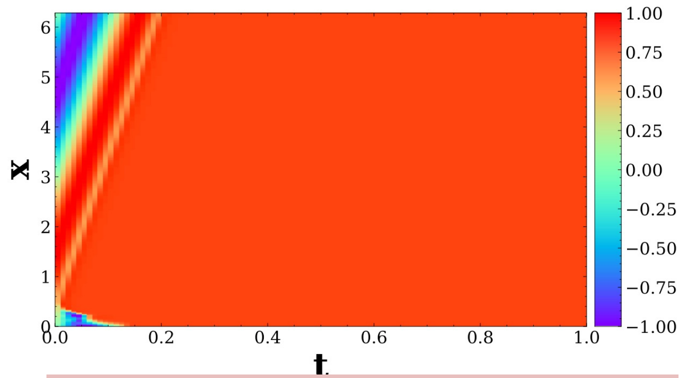
$$\min_{\theta} \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2$$
$$+\|\hat{u}(x,0) - sin(x)\|_2^2$$
$$+\|\hat{u}(x=2\pi) - \hat{u}(x=0)\|_2^2$$



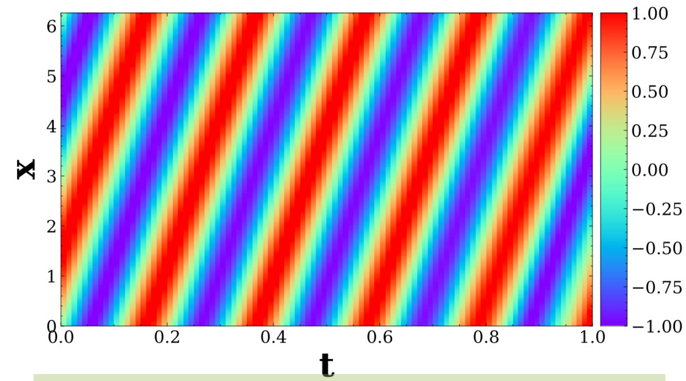
Rethinking PINNs: Curriculum Learning

$$\min_{\theta} \mathcal{L} = \lambda_{\mathcal{F}} \|\hat{u}_t + \beta \hat{u}_x\|_2^2$$
$$+ \|\hat{u}(x,0) - sin(x)\|_2^2$$
$$+ \|\hat{u}(x = 2\pi) - \hat{u}(x = 0)\|_2^2$$



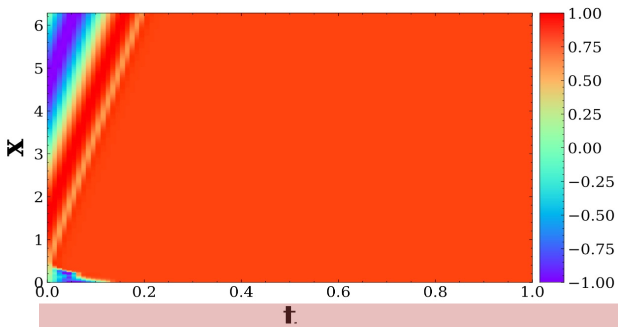


Regular training PINN solution for $\beta = 30$

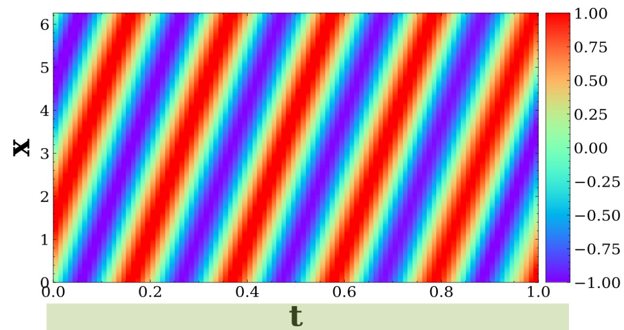


Curriculum training PINN solution for $\beta = 30$

Rethinking PINNs: Curriculum Learning



Regular training PINN solution for $\beta = 30$



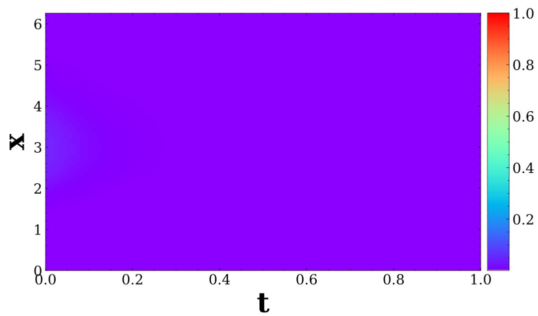
Curriculum training PINN solution for $\beta = 30$

		Regular PINN	Curriculum training
1D convection: $\beta = 20$	Relative error	7.50×10^{-1}	$\mathbf{9.84 \times 10^{-3}}$
	Absolute error	4.32×10^{-1}	$\mathbf{5.42 \times 10^{-3}}$
1D convection: $\beta = 30$	Relative error	8.97×10^{-1}	$\mathbf{2.02 \times 10^{-2}}$
	Absolute error	5.42×10^{-1}	$\mathbf{1.10 \times 10^{-2}}$
1D convection: $\beta = 40$	Relative error	9.61×10^{-1}	$\mathbf{5.33 \times 10^{-2}}$
	Absolute error	5.82×10^{-1}	$\mathbf{2.69 \times 10^{-2}}$

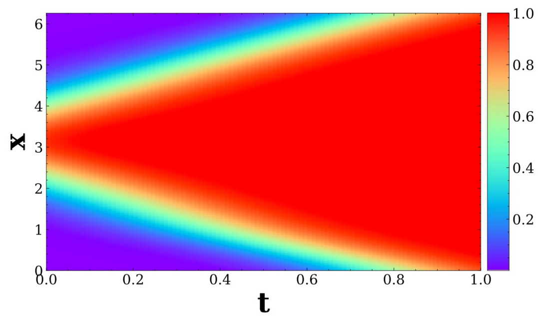
Rethinking PINNs: Curriculum Learning

- This approach works quite well for reaction problem as well

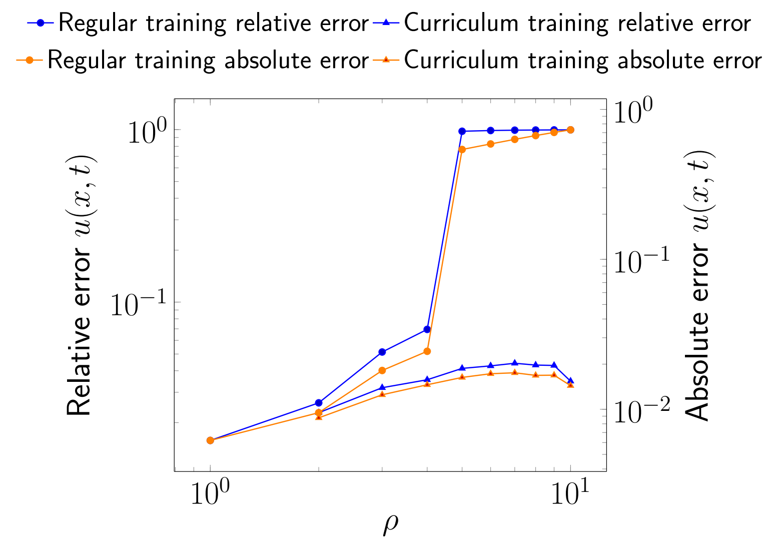
$$\frac{\partial u}{\partial t} - \rho u(1-u) = 0, \quad x \in \Omega,\ t \in (0, T],$$
$$u(x,0) = h(x), \quad x \in \Omega.$$



Regular training PINN solution for $\rho = 10$

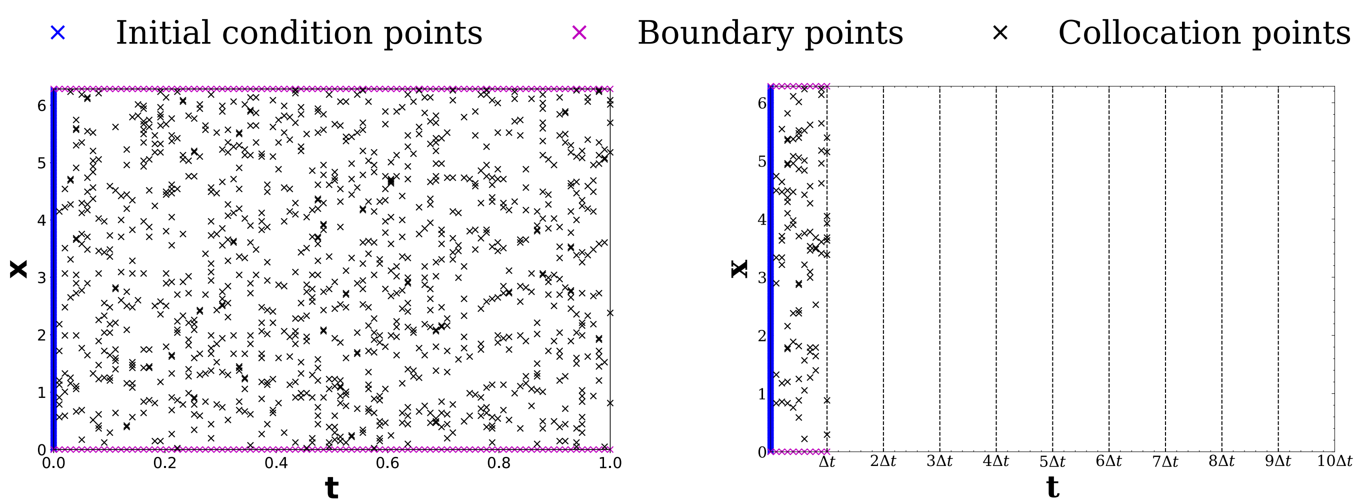


Curriculum training PINN solution for $\rho = 10$



Rethinking PINNs:
Pose the Problem as Sequence to Sequence Learning

- PINN formulation tries to predict the **entire space-time simultaneously**.
  - This is a very difficult task/function to approximate.
- An alternative is to pose the problem as sequence to sequence learning, where PINN learns to predict the solution in a finite time horizon, and iteratively predicts next time steps



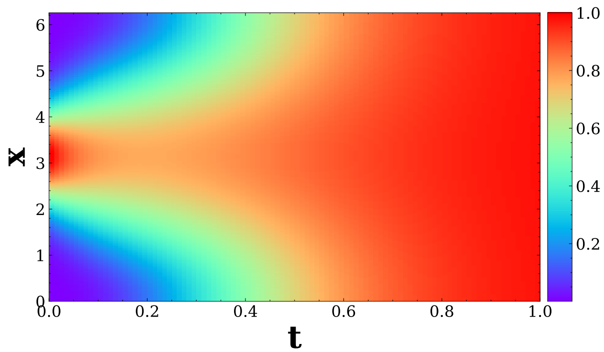
Regular PINN Training

Seq2Seq Training

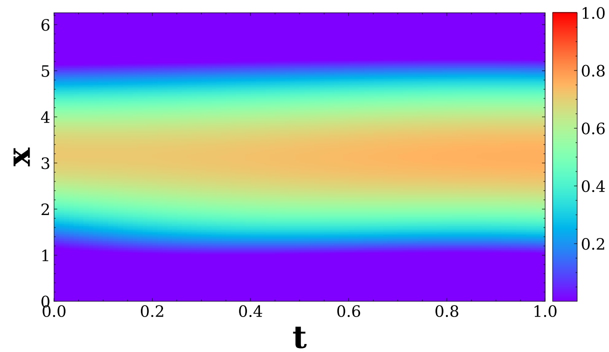
Rethinking PINNs: Seq2Seq Learning for Reaction-Diffusion Problem

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1-u) = 0, \quad x \in \Omega,\ t \in (0, T],$$

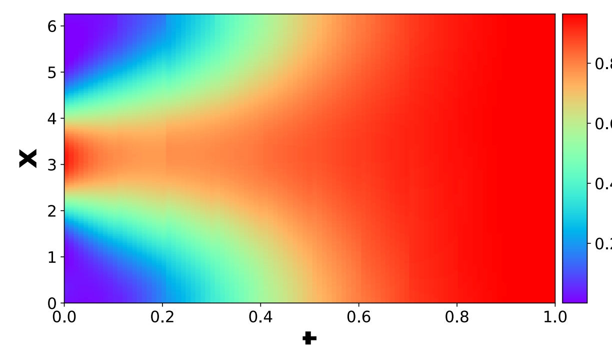
$$u(x, 0) = h(x), \quad x \in \Omega.$$



Exact solution for ρ = 5, v=3



Regular PINN solution for ρ = 5, v=3



seq2seq PINN solution for ρ = 5, v=3

Rethinking PINNs: Seq2Seq Learning for Reaction-Diffusion Problem

Seq2Seq approach can get significantly lower error than regular PINN which tries to predict the entire state space at once

		Entire state space	$\Delta t = 0.05$	$\Delta t = 0.1$
$\nu = 2,\ \rho = 5$	Relative error	5.07×10^{-1}	2.04×10^{-2}	$\mathbf{1.18 \times 10^{-2}}$
	Absolute error	2.70×10^{-1}	1.06×10^{-2}	$\mathbf{6.41 \times 10^{-3}}$
$\nu = 3,\ \rho = 5$	Relative error	7.98×10^{-1}	1.92×10^{-2}	$\mathbf{1.56 \times 10^{-2}}$
	Absolute error	4.79×10^{-1}	1.01×10^{-2}	$\mathbf{8.17 \times 10^{-3}}$
$\nu = 4,\ \rho = 5$	Relative error	8.84×10^{-1}	2.37×10^{-2}	$\mathbf{1.59 \times 10^{-2}}$
	Absolute error	5.74×10^{-1}	1.15×10^{-2}	$\mathbf{8.01 \times 10^{-3}}$
$\nu = 5,\ \rho = 5$	Relative error	9.35×10^{-1}	$\mathbf{2.36 \times 10^{-2}}$	2.39×10^{-2}
	Absolute error	6.46×10^{-1}	$\mathbf{1.09 \times 10^{-2}}$	1.15×10^{-2}
$\nu = 6,\ \rho = 5$	Relative error	9.60×10^{-1}	2.81×10^{-2}	$\mathbf{2.69 \times 10^{-2}}$
	Absolute error	6.84×10^{-1}	$\mathbf{1.17 \times 10^{-2}}$	1.28×10^{-2}

Conclusions

- PINNs are easy to implement but there are many subtle issues associated with their training
- PINNs can fail to learn simple problems such as advection, reaction, and/or reaction-diffusion problems with non-trivial coefficients
- Analyzing the problem shows that while the NN has enough capacity to learn the solution, the optimization problem with PINN's soft regularization becomes very difficult to solve

Two promising solutions are:

- **Curriculum Learning:** First train PINNs with simple constraints and progressively make it more complex
- **Sequence to Sequence Learning:** Instead of trying to predict the entire space-time at once, pose the problem as Seq2Seq and let PINN learn to predict smaller time horizons

Paper: *Krishnapriyan* AS, **Gholami* A**, Zhe S, Kirby RM, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. NeurIPS, 2021.*
Code: https://github.com/a1k12/characterizing-pinns-failure-modes

Open Problems

There are many more open problems in PINNs:

- **Optimization:**
  - Unlike all other classical ML tasks, PINNs cannot be optimized with mini-batch (SGD, ADAM, etc. all fail). The only method that works is LBFGS with full batch size
  - This makes training PINNs very slow and hard to optimize

- **NN Architecture:**
  - Classical NN architecture may not be optimal for PINNs. Need to investigate alternative architectures that are more suited for the continuous nature of the problem.
  - Need to investigate how the architecture should be changed as the underlying dynamics change
    - Elliptical vs Hyperbolic vs Parabolic PDEs may need different architectures

Thank You for Listening and thanks for inviting me to the Babuška Seminar!

amirgh@berkeley.edu






Berkeley
UNIVERSITY OF CALIFORNIA