# Solving Physics Equations with Neural Networks

Brandon Ismalej, Davit Gasparyan, Stepan Karapetyan
Faculty Advisor: Dr. Yang Peng

SfS² SECURE for Student Success

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

## Introduction

### Background

**Damped Driven Pendulum**:

- Classical example of a **chaotic system**, characterized by non-linear dynamics
- Governed by differential equations that describe its motion under external periodic driving forces and damping

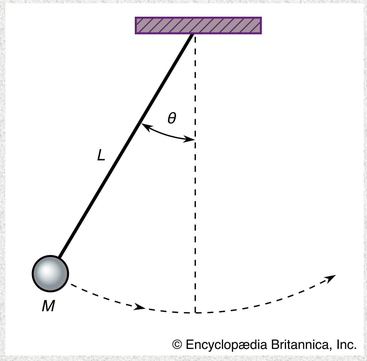


Figure 1: Image of Basic Pendulum System

**Physics-Informed Neural Networks (PINNs):**

- Incorporates physical laws (e.g. diff. eq.'s) into the loss function of neural networks
- Potential to model complex systems with reduced computational costs and higher generalization

### Objectives

**Main Goal:** To compare the performance of KAN and MLP-based PINNs against traditional numerical solvers in modeling the dynamics of the chaotic damped driven pendulum

$$\frac{d^2\theta}{dt^2} + \lambda\frac{d\theta}{dt} + \omega^2 \sin\theta - f\cos\Omega t = 0$$
$$\text{where } \lambda = \mu/m, f = \frac{F}{ml} = \gamma\omega^2, \text{ and } \omega^2 = g/l.$$

Equation 1: Damped Driven Pendulum

- **Validate claim** that **KANs outperform MLPs** in terms of accuracy and interpretability [1]

## Methodology

- **System Studied:** Main focus on damped driven pendulum with γ = 0.8
- Numerical solution benchmark generated using **SciPy's ODE Solver**
- **Modelling approach:** Implemented PINNs with MLP and KAN architectures
  - As shown in Fig. 1
- Training PINNs using **loss function** shown in Eq. 2:

$$L(\theta) = \omega_D L_D(\theta) + \omega_B L_B(\theta)$$

Equation 2: Total Loss Function

$$L_D(\theta) = \frac{1}{N_D}\sum_{\boldsymbol{x}\in\Omega}\left|D\left(\boldsymbol{x}, y_{NN}(\boldsymbol{x}), \frac{\partial y_{NN}}{\partial x_1}, \ldots, \frac{\partial y_{NN}}{\partial x_d}, \ldots\right)\right|^2$$

Equation 3: Physics Loss Function

$$L_B(\theta) = \frac{1}{N_B}\sum_{\boldsymbol{x}\in\partial\Omega}|B(\boldsymbol{x}, y_{NN}(\boldsymbol{x}))|^2$$

Equation 4: Boundary Loss Function

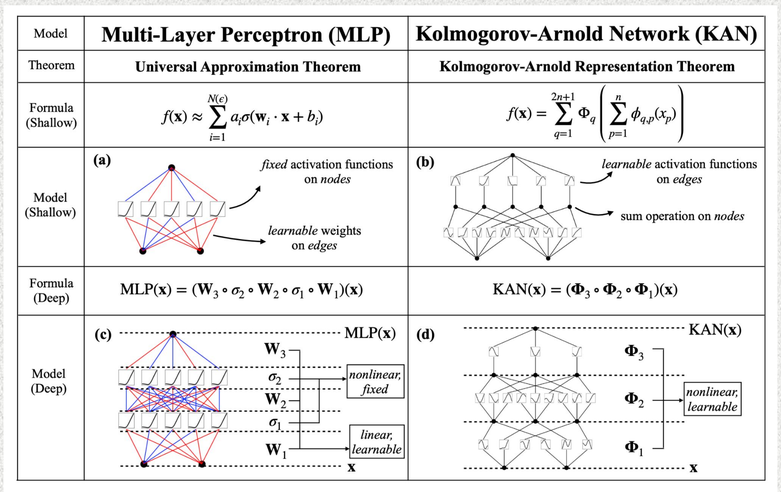


Figure 2: Comparison of MLP & KAN [1]

## Experimental Setup

### Numerical Solver (SciPy)

In our setup, we **benchmarked six ODE solvers** from SciPy, aiming to balance **computational speed and accuracy.** The Radau method emerged as our preferred choice due to its optimal trade-off between these factors.

**Why Radau?**

- Balanced Performance: Offers a good compromise between speed and accuracy.
- Stiff Problem Handling: Particularly effective for stiff ODEs.
- Global Accuracy: Maintains high accuracy across diverse problems.

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i$$

Figure 3: Radau equation 2

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij} k_j\right)$$

Figure 4: Radau equation 2

### Multilayer Perceptron (MLP)

**Theoretical Foundations:** Based on the Universal Approximation Theorem.

- Single hidden-layer MLP can approximate any continuous function with enough neurons

**Model Architecture:**

- MLP Layers: [1, 50, 50, 1]
  - 1D input & output
  - 2 layers of 50 hidden neurons/layer

**Observations:**

- Steady Accuracy increase: Steadily improves and achieves high level of accuracy over time

**Training Details:**

- γ = 0.8
- Time = 0 to 6s
- 1200 training pts.
- Optimizer: Adam
- Learning Rate: 1e-3
- Epochs: 30,000

**Results for MLP::**

- Boundary loss: 0.0001
- Physics loss: 0.0001
- Total Loss: 0.0003

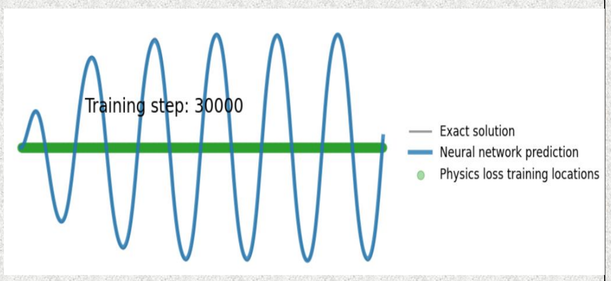


Figure 5: Final MLP vs. Numerical Solution Comparison

### Kolmogorov Arnold Network (KAN)

**Theoretical Foundations:** Based on the Kolmogorov-Arnold Representation Theorem

- Represents multivariate functions as a sum of univariate functions

**Model Architecture:**

- KAN Layers: [1, 8, 8, 8, 1]
  - 1D input & output
  - 3 layers of 8 hidden neurons/layer
- Cubic Spline: k = 3
- 15 grid intervals

**Observations:**

- Rapid Initial Accuracy: Quickly achieves a high level of accuracy in early stages of training
- Progress Plateau: Following initial training, progress slows dramatically, increasing computational costs

**Training Details:**

- γ = 0.8
- Time = 0s to 6s
- 1,600 training pts.
- Optimizer: Adam
- Learning Rate: 5e-3
- Epochs: 1,000

**Results for KAN:**

- Boundary Loss: 0.0002
- Physics Loss: 0.0045
- Total Loss: 0.0047

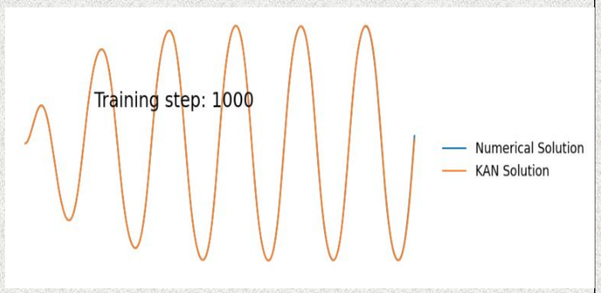


Figure 6: Final KAN vs. Numerical Solution Comparison

## Results

### Accuracy Comparison

- **Radau Numerical solver is using:**
  - Relative tolerance: $10^{-13}$
  - Absolute tolerance: $10^{-15}$
- **MLP's accuracy depends on loss**
  - Loss can be inaccurate because MLP could learn different solution to the differential equation then the one we want it to learn
  - Highly inaccurate for gamma values above 1.01
- **KAN's accuracy depends on loss**
  - Similar to MLP, the loss can be inaccurate in describing the accuracy of the solution that KAN came up with
  - Highly inaccurate for gamma values above 1.01

### Discussion

- **MLP and KAN struggled** with accuracy for chaotic systems, especially for gamma values above 1.01, likely due to limitations in their loss functions and **hyperparameter sensitivity**.
- These findings suggest that traditional **numerical solvers** currently offer more **reliable solutions** for chaotic differential equations, though advancements in PINNs, such as systematic hyperparameter tuning and new architectures, may improve their performance in the future.

## Conclusions and Future Work

### Summary

- MLP and KAN neural network architectures aren't as accurate and fast as numerical solvers for nonlinear differential equations. KAN was found to be almost same as MLP in solving nonlinear equations. MLP was only better in terms of speed of training.
- This work provides **limited evidence** in favor of the claim that KANs can outperform MLPs [1].

### Implications

- No advantage was found in using KAN and MLP to solve nonlinear differential equations instead of numerical solvers, especially in chaotic regime.

### Future Work

- Systematic tuning of hyper parameters could make neural networks better at approximating the solutions
- **MultKAN** is KAN 2.0 that introduces **multiplication layers** that could help KAN be able to better approximate the solution of nonlinear differential equations [2].

## References

[1] Z. Liu *et al.*, "KAN: Kolmogorov-Arnold Networks," Apr. 30, 2024, *arXiv*: arXiv:2404.19756. doi: 10.48550/arXiv.2404.19756.
[2] Z. Liu *et al.*, "KAN 2.0: Kolmogorov-Arnold Networks Meet Science," Aug. 19, 2024, *arXiv*: arXiv:2408.10205. doi: 10.48550/arXiv.2408.10205.

## Acknowledgement

•Supported by the SfS² Program and funded by the United States Department of Education FY 2023 Title V, Part A, Developing Hispanic-Serving Institutions Program five-year grant, Award Number P031S230232, CFDA Number 84.031S.
•However, the contents of this presentation do not necessarily represent the policy of the US Department of Education, and you should not assume endorsement by the Federal Government.