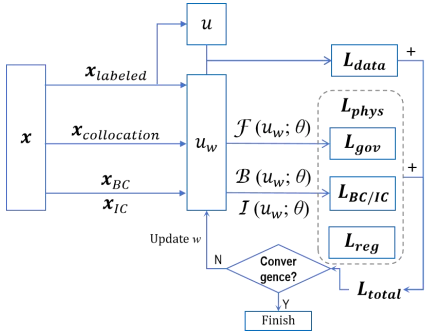
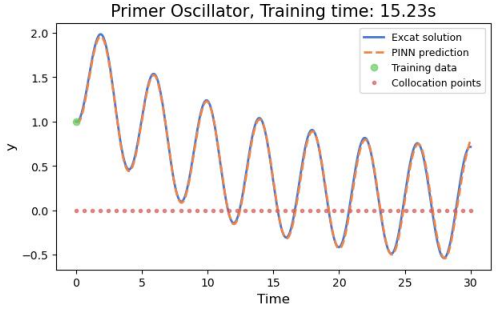
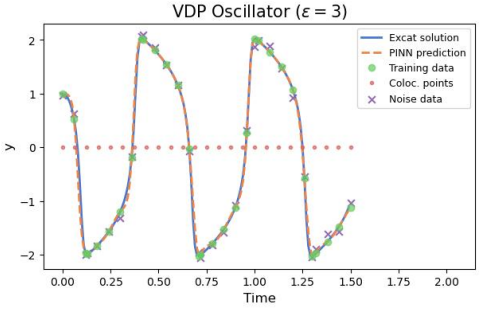
利用软约束物理信息神经网络在小数据下高性能求解谐振常微分方程

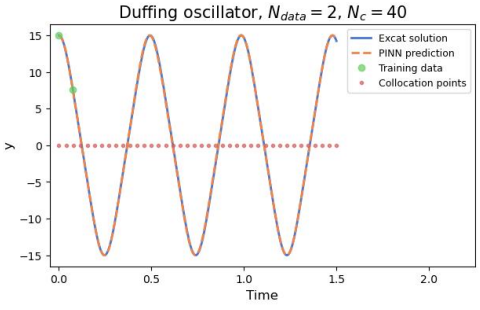
Solving Oscillator ODEs via Soft-constrained Physics-informed Neural Network (PINN) with Small Data

汇报人：卢凯良









目 录

CONTENTS

1 解微分方程的三种方法对比： PINN vs NN vs Numerical Discretization

Methods	Numerical Discretization (ND, e.g., RK, FEM) 传统数值离散化方法	Neural Network (NN) 常规神经网络	Physics-informed Neural Network (PINN) 物理信息神经网络
Features	• 理论基础完备、可理解，**具有现成的误差估计以及收敛性和稳定性保证** • 高效率 • 高精度 • 遭受维度灾难 • 网格生成仍很复杂 • 无法无缝整合噪声数据 • 解决逆问题的成本过高	• 无网格 • **一定程度上克服维度灾难**，可应对高维和非线性问题 • 对新数据的泛化能力好 • 需要大数据和强算力 • 过拟合 • “黑箱”问题	• 常规神经网络的优点 + • 小数据、不易过拟合，泛化能力强 • 可处理不完善数据和不完整模型 • 处理病态问题和逆问题时有效且高效 • 可理解性比NN好 • 网络参数少、训练和预测速度快 • 灵活性好、跨计算平台 • “黑箱”问题

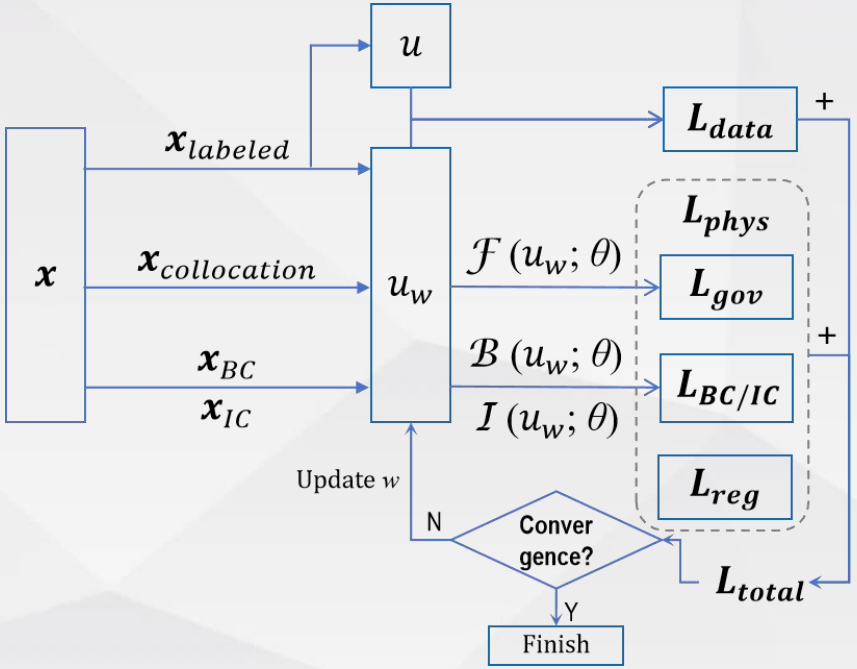
- **Why PINN: In Small Data Situations where NN can be Improved**

在现实的复杂科学和工程实践中（例如物理、生化、医学等），数据通常通过昂贵的实验或大规模仿真计算产生，数据往往是稀疏的且带有噪声；有时获取数据的成本高得令人望而却步，甚至根本无法获得大量实验数据。

- **PINN as a Complement or Alternative to ND**

需要强调的是，PINN 与 ND 并非取代关系——PINN 在解决上述的假定问题和逆问题等方面特别有效；而对于不需要任何数据同化的正向问题，现有的数值求解器 (e.g., RK4, FEM) 目前的性能优于 PINN。

2 软约束PINN方法



Soft-constrained PINN 解微分方程计算图

控制方程

$$\text{Differential equation: } \mathcal{F}(u;\theta)(\boldsymbol{x}) \equiv \mathcal{F}(u,\boldsymbol{x};\theta) = 0, x \in \Omega. \quad (1)$$

$$\text{Initial conditions: } \mathcal{I}(u;\theta)(x,t_0) = 0, x \in \Omega_0. \quad (2)$$

$$\text{Boundary conditions: } \mathcal{B}(u;\theta)(x,t) = 0, x \in \partial\Omega. \quad (3)$$

损失函数

$$\mathcal{L}_{total} = \mathcal{L}_{data} + \underbrace{\mathcal{L}_{gov} + \mathcal{L}_{IC} + \mathcal{L}_{BC} + \cdots}_{\mathcal{L}_{phys}}$$

$$= \frac{\lambda_d}{N}\sum_{i=1}^{N}\|u_w(\boldsymbol{x}_i) - u(\boldsymbol{x}_i)\|^2 + \frac{\lambda_g}{|\Omega|}\int_{\Omega}\|\mathcal{F}(u_w;\theta)(\boldsymbol{x})\|^2 d\boldsymbol{x} + \quad (4)$$

$$\frac{\lambda_i}{|\Omega_0|}\int_{\Omega_0}\|\mathcal{I}(u_w;\theta)(\boldsymbol{x})\|^2 d\boldsymbol{x} + \frac{\lambda_b}{|\partial\Omega|}\int_{\partial\Omega}\|\mathcal{B}(u_w;\theta)(\boldsymbol{x})\|^2 d\boldsymbol{x} + \cdots.$$

配置点采样

$$\mathcal{L}_{total} = \frac{\lambda_d}{N_d}\sum_{i=1}^{N}\|u_w(\boldsymbol{x}_i) - u(\boldsymbol{x}_i)\|^2 + \frac{\lambda_g}{N_g}\sum_{i=1}^{N_g}\|\mathcal{F}(u_w;\theta)(\boldsymbol{x}_i)\|^2 + \quad (5)$$

$$\frac{\lambda_i}{N_i}\sum_{i=1}^{N_i}\|\mathcal{I}(u_w;\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_b}{N_b}\sum_{i=1}^{N_b}\|\mathcal{B}(u_w;\theta)(\boldsymbol{x}_i)\|^2 + \cdots.$$

[Ref] Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nature Reviews Physics 3, 422–440 (2021).
https://www.bilibili.com/video/BV19a41167RU/?spm_id_from=333.337.search-card.all.click
Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: A deep learning library for solving differential equations. SIAM Review 63(1), 208–228 (2021)
arXiv:2408.11077v4

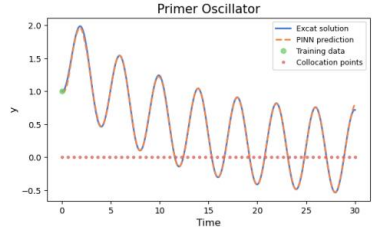
3 实验结果与讨论 — 3.1 PINN vs NN 求解线性谐振ODE

线性谐振方程（primer oscillator）

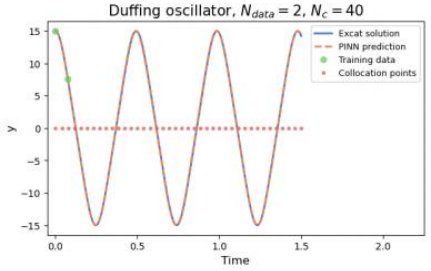
$$\frac{du}{dt} + 0.1u - \sin(\pi t/2) = 0. \tag{6}$$

式中，$t \in [0, 30]$，初始条件 $u_0 = 1$。我们使用常规神经网络和 PINN 两种方法分别求解，采用 3 层 32 个神经元的全连接隐藏层为 Base 网络，Adam 优化器、学习率 $\eta = 3 \times 10^{-3}$。精确解以龙格库塔法（3000 步，至少约 200 步）所得结果为基准。

极简数据情形 Minimalistic Training Data

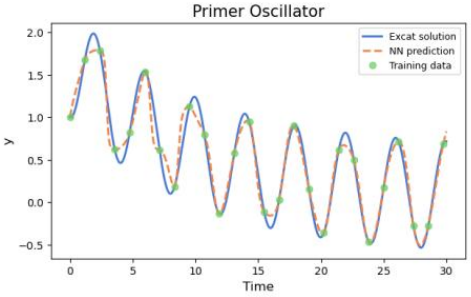


(a) $N_{data} = 1$, $N_c = 48$

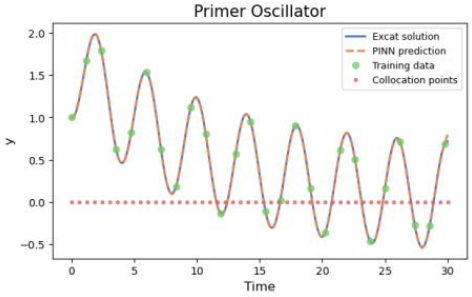


First-order **Second-order**

- （当问题的非线性较弱时，）非常少量的标注训练数据加上少量的配置点数据（仍属监督、非标注）就足以预测解
- 极简情况下，正如经典的解析或数值积分方法求解一阶和二阶微分方程仅分别需要一个和两个初始条件，PINN 方法也仅分别需要一个训练点（初始值）和含初始值的两个训练点 + 少量的配置点

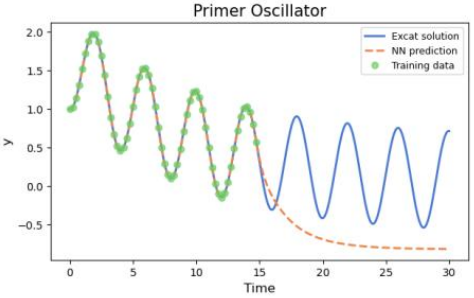


(a) $N_{data} = 26$, $N_c = 0$

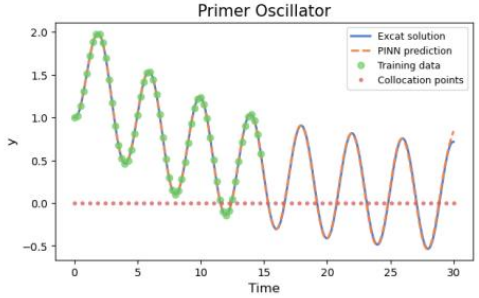


(b) $N_{data} = 26$, $N_c = 50$

图 2 Result Comparison of PINN vs NN Solving Linear Oscillator–Case 1. N_{data}–No. of training data, N_c–No. of collocation points, all the same below. (Reproduced from [26] via DeepXDE for direct comparison.)



(a) $N_{data} = 60$ (ONLY Left Interval), $N_c = 0$



(b) $N_{data} = 60$ (ONLY Left interval), $N_c = 50$

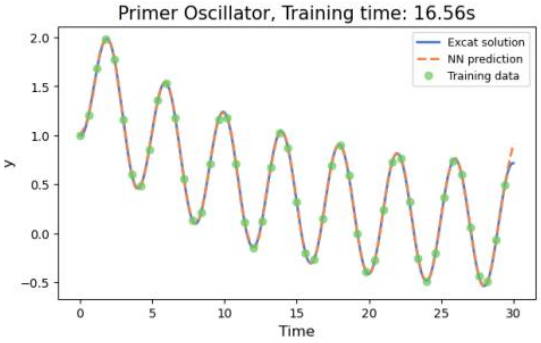
图 3 Result Comparison of PINN vs NN Solving Linear Oscillator–Case 2. (Reproduced from [26] via DeepXDE.)

3 实验结果与讨论 — 3.1 PINN vs NN 求解线性谐振ODE

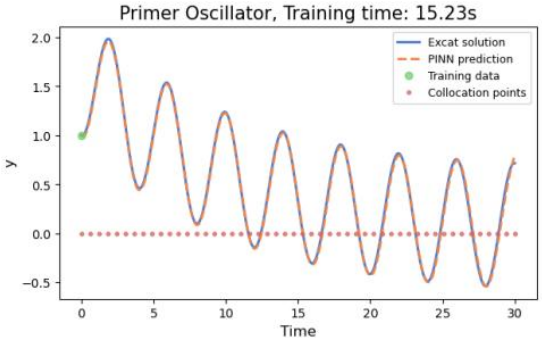
对训练时长的影响 Accelerating Training ?

- 一方面，相较于常规NN，PINN 方法因其只需要小数据量（极简的标注训练数据加上少量的配置点）故可**明显缩短训练时长**
- 另一方面，因其损失函数加上了 L_phys 变得更为复杂，在迭代时需要更多的计算、甚至有时需要更多的迭代轮数，有可能**反而拖累了训练效率**
- 当减少数据量获得的收益大于增加损失函数项引起的拖累时，软约束 PINN 方法是可以加快训练速度的

- 需要强调的是，（因为 primer oscillator 是简单的线性 ODE，方便对比起见，我们使用了相同的 Base 网络；）PINN 因其小数据量特性可以比 NN 使用参数量更少的网络，故**能从减少网络参数量上**更进一步地提高训练速度（例如，对于图 5(b)，如果 Base 网络减少为 2 × 32 也能很好收敛同时训练时长减少至 14.30s）。



(a) $N_{data} = 50$, $N_c = 0$



(b) $N_{data} = 1$, $N_c = 50$

图 5 Training Time of PINN vs NN Solving Primer Oscillator via DeepXDE: (a) 16.56s by NN with Min. 50 training points; (b) 15.23s by PINN with Min. 1 initial value training point + 50 collocation points. (Reproduced from [26] via DeepXDE.)

- 基于 DeepXDE1.11.1 在 CPU 平台 (Windows10, Intel Core i9-9900K @3.6GHz, 32GB Memory) 上对 primer oscillator 进行了验证测试，训练时长为 5 次训练的平均值（每次训练 24000 轮；通常无需 24000 轮即可收敛，公平对比起见均训练 24000 轮，因此所得训练时长是偏保守的）

3 实验结果与讨论 — 3.2 PINN求解非线性谐振子

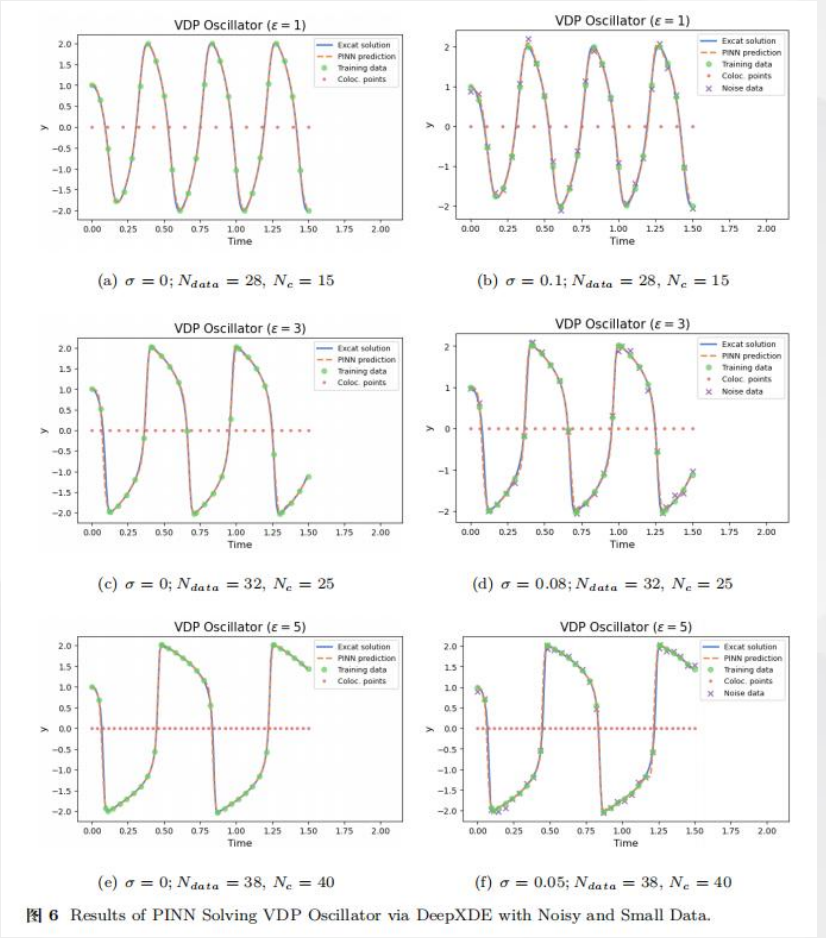
软约束 PINN 方法在求解非线性谐振 ODE 上同样具有良好的表现 [BB23]。我们基于 DeepXDE 成功求解了经典的、强非线性的 Van der Pol oscillator，并展示了 PINN 方法对噪声数据的容忍性以及 DeepXDE 在 PINN 实现上的低代码和求解速度优势。Van der Pol (VDP) oscillator ODE 为

$$\frac{d^2u}{dt^2} + \omega_0^2 u - \epsilon\omega_0(1-u^2)\frac{du}{dt} = 0. \tag{7}$$

式中，$t \in [0, 1.5]$，初始条件 $u_0 = 1$；ω_0 是归一化的角速度，取 $\omega_0 = 15$；ϵ 反映了 VDP 的非线性程度，ϵ 值越大非线性越强。

- 在同样的训练点数和配置点数下，基于有噪声数据训练得到的结果（右列）与无噪声结果（左列）基本完全一致，显示了 PINN 方法**对噪声的良好包容性**
- 随着非线性程度的增加（通过增大 ϵ）需要更多的训练点和配置点点数布置在非线性特征明显的区域才能得到精确解（当 ϵ = 1, 3, 5 时，训练点数和配置点数分别需要N_data = 28, 32, 38; N_c = 15, 25, 40）
- 并且随着非线性的增大，对噪声的容忍能力也逐渐下降：随着 ϵ 从 1 增大到 5，能容忍的 σ 却从 0.1 下降到 0.05；即使加密数据点对提升噪声容忍度的作用也十分有限

- 使用 PINN 方法求解含噪声小数据下的 VDP 非线性谐振子，与求解 primer oscillator 一样，依旧采用 3 × 32 的全连接 Base 网络，这体现了 PINN 继承了神经网络的强表达能力，相关参数设置（即 ϵ、训练点数、配置点数、正态噪声方差）见图 6 中说明，精确解由 RK4 生成



(a) $\sigma = 0; N_{data} = 28, N_c = 15$　(b) $\sigma = 0.1; N_{data} = 28, N_c = 15$

(c) $\sigma = 0; N_{data} = 32, N_c = 25$　(d) $\sigma = 0.08; N_{data} = 32, N_c = 25$

(e) $\sigma = 0; N_{data} = 38, N_c = 40$　(f) $\sigma = 0.05; N_{data} = 38, N_c = 40$

图 6 Results of PINN Solving VDP Oscillator via DeepXDE with Noisy and Small Data.

3 实验结果与讨论 — 3.2 PINN求解非线性谐振子

DeepXDE vs Pytorch of PINN Code

```python
import deepxde as dde
import numpy as np
from scipy.integrate import odeint

def VDP:
"""Define the ODE of VDP"""

# Call odeint in Scipy to generate numerical exact solutions
y = odeint(VDP)

# Sample and generate the training set from the exact
 solution, including initial point (t0, y0) and endpoint
t_data=t[0:1.5:0.005]
y_data=y[0:1.5:0.005]

geom = dde.geometry.TimeDomain(0.0, 1.5) # Define time
 domain

def boundary(t, on_boundary):
return on_boundary and dde.utils.isclose(t[0], 0)

def error_derivative(inputs, outputs, X):
return dde.grad.jacobian(outputs, inputs, i=0, j=None)

# Define initial conditions: y(0)=1 and y'(0)=0
ic1 = dde.icbc.IC(geom, lambda x : 1, lambda _, on_initial:
 on_initial)
ic2 = dde.icbc.OperatorBC(geom, error_derivative, boundary)

# Define training set and boundary conditions (including
 boundary conditions)
t_y_data = dde.PointSetBC(t_data, y_data)

data = dde.data.TimePDE(geom, ode, [ic1, ic2, t_y_data],
 num_domain=40, num_boundary=1)     # num_domain is the
 number of collocation points

net = dde.maps.FNN([1] + [32] * 3 + [1])   # Instantiate FNN
model = dde.Model(data, net)
model.train(epochs=24000)
```

Listing 2: DeepXDE Implementation of PINN to Solve VDP

```python
import torch
import numpy as np
from scipy.integrate import odeint

def VDP:
"""Define the ODE of VDP"""

# Define time domain and boundary conditions
t = np.linspace(0, 1.5, 1000)
y0 = [1, 0]     # y0 is the initial condition

# Call odeint in Scipy to generate numerical exact solutions
y = odeint(VDP, y0, t)

# Sample and generate the training set from the exact
 solution, including initial point (t0, y0) and endpoint
t_data=t[0:1.5:0.005]
y_data=y[0:1.5:0.005]

# Select collocation point
x_collocation = torch.linspace(0.,1.5,48).view(-1,1).
 requires_grad_(True)

class FNN:
"""Define neural networks"""

model = FNN(1, 1, 32, 3)    # Instantiate FNN

# The training process of PINN:
for i in range(24000):

# Data loss
yh = model(t_data)
Loss_data = 1 * torch.mean((yh - y_data)**2)

# Physical loss
yhp = model(x_collocation)
dx = torch.autograd.grad(yhp, x_collocation, torch.ones_like
 (yhp),
create_graph=True)[0]
dx2 = torch.autograd.grad(dx, x_collocation, torch.ones_like
 (dx),
create_graph=True)[0]
physics = (dx2 + 15 **2 * yhp - 5 * (1 - yhp**2) * dx)
# Calculate the mean square error of the test set
yhpp = model(t)
mse = torch.mean((yhpp - y)**2)

# Total loss of backpropagation
```

Listing 1: Pytorch Implementation of PINN to Solve VDP

- 基于 DeepXDE 或 Pytorch 实现软约束 PINN 的总体流程是一致的
- 在求解 VDP的相同设置下，DeepXDE 的代码行数（约 40 行）大概是 Pytorch 实现的 **1/3**
- DeepXDE 的训练效率优于 Pytorch？—对于图 5(b) 的例子，相同设置下，基于 Pytorch 训练耗时43.10s，比 DeepXDE 的**多 1.8 倍**左右

PINN Computational Efficiency @CPU vs @GPU

表 2 PINN Training Time (Second) @CPU vs @GPU.

PINN to ODEs via DeepXDE	@CPU	@GPU
Primer Oscillator ($N_{data} = 50, N_c = 48$)	16.02	8.73
VDP Oscillator ($N_{data} = 50, N_c = 48$)	17.60	13.15

- 我们对比了基于 DeepXDE 分别在 CPU 和GPU 上（CPU 如前，GPU 为 Nvidia GTX1080Ti）求解 primer、VDP oscillators 的训练时长，取 5 次训练（每次训练 24000 轮）的平均值，Base 网络分别采用 2×32 和 3×32的全连接隐藏层

- PINN 方法不依赖于大数据故**可以在 CPU 或 GPU 等平台上高效运算**
- GPU 对 PINN 求解 primer oscillator 和 VDP oscillator的训练加速效果明显，训练时长分别从 16.02s 和 17.60s 减小至 8.73s 和 13.15s

3 实验结果与讨论 — 3.2 PINN求解非线性谐振子

Conservation of Energy Regularization Improves the Result of Solving Duffing Oscillator

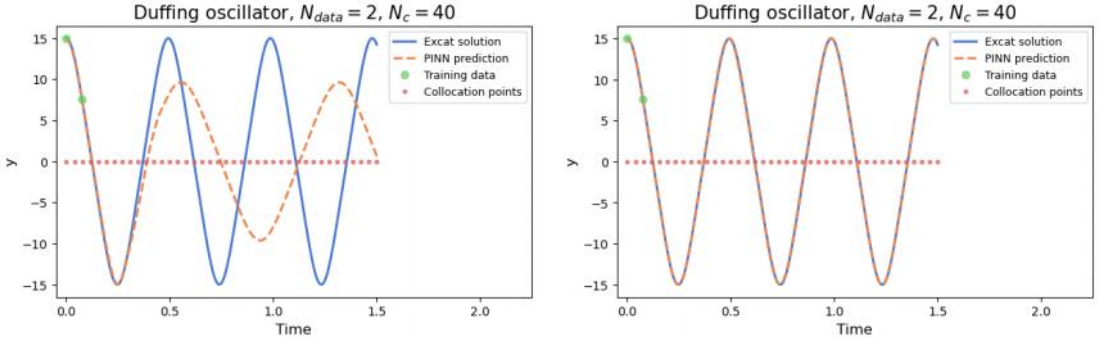
- PINN 方法的一大优势是可以充分利用物理信息或知识先验，例如能量守恒定律，软约束法通过在总损失函数中加正则化项 L_reg 来实现。我们用二阶非线性的 Duffing oscillator 来验证

$$\frac{d^2u}{dt^2} + \alpha u + \beta u^3 = 0. \tag{8}$$

式中，$t \in [0, 1.5]$，取 $\alpha = 1.0, \beta = 1.0$，初始条件为 $u_0 = 15, u_0' = 0$。同样，精确解以龙格库塔法所求得结果为准。式(8)的能量守恒项为 $E = \frac{1}{2}(\frac{du}{dt})^2 + \frac{1}{2}\alpha u^2 + \frac{1}{4}\beta u^4$，代入正则化项 $\mathcal{L}_{reg}$。

- 考虑极简数据的情况，对于二阶 ODE 只用含初始值的两个训练点，均匀选取配置点数N_c = 40。
- 使用和不使用能量守恒正则化项的 PINN 求解 Duffing oscillator 的最优调参结果分别如图 7(a) 和 (b) 所示：

- 受非线性影响，极简数据情况下，没有用能量守恒正则化的 PINN 在整个 72000 轮次的训练中都无法完成收敛；而加上能量守恒正则化的 PINN 则取得了明显的改善，与精确解完美吻合
- 需要注意的是，能量守恒正则化对Duffing 有用，因为 Duffing 遵守能量守恒；对 VDP 无用，因为 VDP 是能量耗散的



(a) w/o the Conservation of Energy Regularization (b) with the Conservation of Energy Regularization

图 7 Results of PINN Solving Duffing Oscillator with the Conservation of Energy Regularization or Not.

4 总结与应用展望

- 归纳了软约束 PINN 方法适用于求解 ODE 和 PDE 的数学框架和计算流程。通过求解典型谐振 ODE（涵盖了 Primer、VDP、Duffing 等一阶、二阶，线性、非线性各类谐振子）从实验上验证了它的工作机制及其精度和效率等定量性能。特别是，我们重点研究了在极简数据下软约束PINN 方法的有效性、高效率和对噪声的鲁棒性。

- 我们的实验验证展示了 PINN 的内在工作机制和特点：1) PINN 嵌入了物理信息和先验知识，从机制上减少了数据冗余，具有用小数据实现强泛化的独特优势。它大大减少了对标注数据的需求，甚至可以用最少的数据预测解（例如，对于线性一阶和二阶微分方程，分别只需要一个和两个包含初始值的训练点数据，外加少量的配置点数据），从而提高了计算效率。2) PINN 对噪声数据具有良好的鲁棒性，能提供准确且物理上一致的预测，甚至能在训练集的时域之外进行外推，因此增强了泛化能力。以及 3) 更好的可理解性。

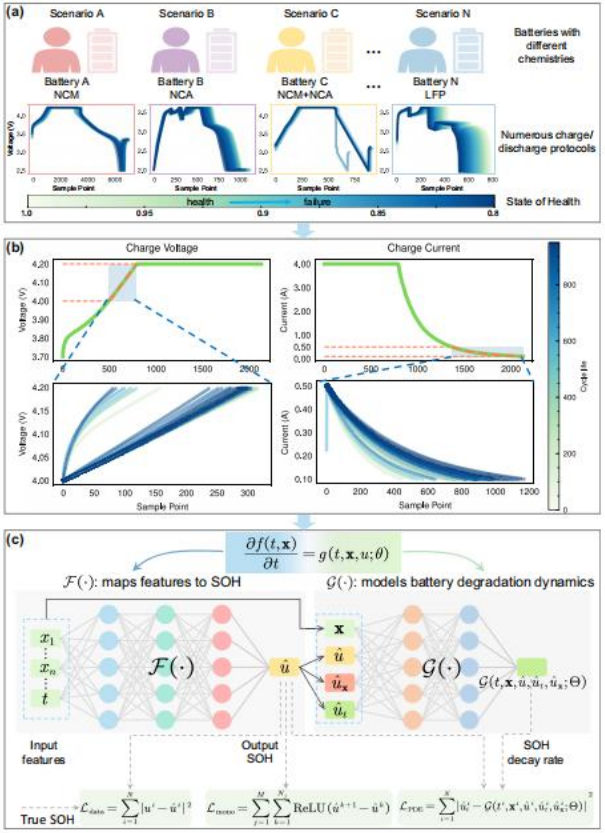
- 软约束 PINN 方法通过在总损失函数中加正则化项即可方便地施加物理定律（例如能量守恒）约束，从而改善遵守该物理定律（能量守恒）的 ODE 的求解性能，例如：二阶非线性的 Duffing 谐振子能够在极简数据（含初始值的 2 个训练点加少量配置点）下快速、精确收敛。

- 我们基于 DeepXDE 实现软约束 PINN 方法具有轻代码、训练速度快、跨CPU/GPU 平台灵活等优点。

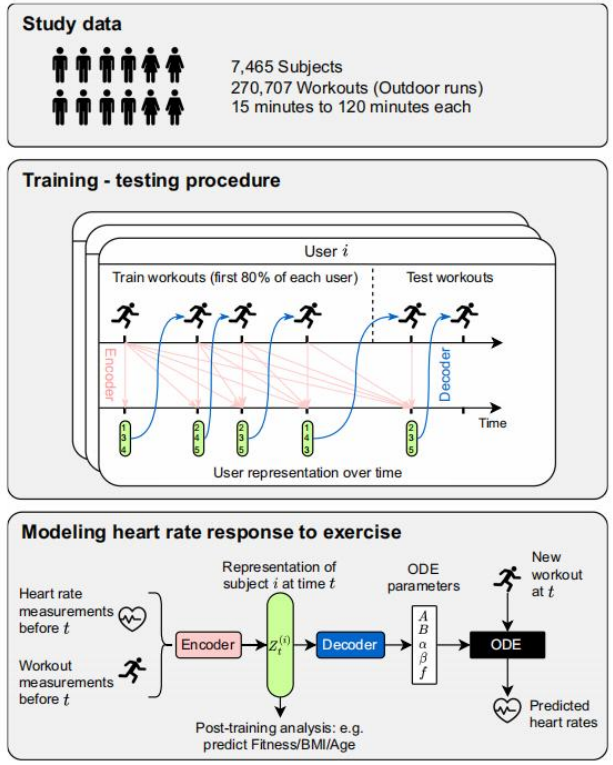
- 综上所述，PINN继承了神经网络的强表达能力，我们通过软约束的方式将物理信息和知识融入神经网络拓扑结构或学习架构以及计算流程中，从而减少了数据冗余、改进了泛化性能，进而提高了计算效率、增强了可理解性和鲁棒性。基于上述优势PINN正成为数字双胞胎（Digital Twins）时代的有利催化剂。

4 总结与应用展望

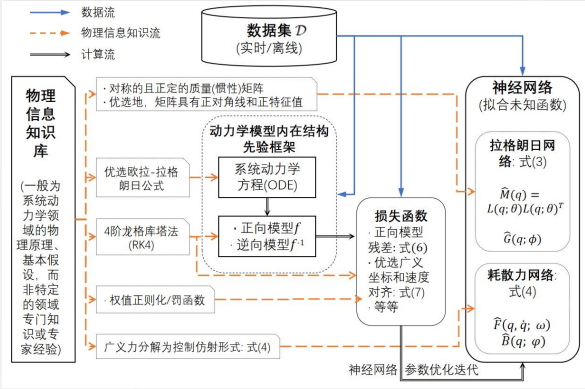
锂电池健康状态(SOH)预测[1]



基于心率图的运动健康状态预测[2]



机械臂动力学模型构建方法[3]



[1] Wang, F., Zhai, Z., Zhao, Z. et al. Physics-informed neural network for lithium-ion battery degradation stable modeling and prognosis. Nat Commun 15, 4332 (2024).

[2] Nazaret, A., Tonekaboni, S., Darnell, G. et al. Modeling personalized heart rate response to exercise and environmental factors with wearables data. npj Digit. Med. 6, 207 (2023).

[3] 卢凯良, 等. 一种基于物理信息和知识启发的机械臂动力学模型构建方法. 发明专利申请: 2024111983946.

Thank You!

感　谢！