深度之眼
deepshare.net

PINN核心技术揭秘：神经网络求解偏微分方程

导师：择木老师

深度之眼
deepshare.net

目录

深度之眼
deepshare.net

偏微分方程简介

Partial Differential Equations, PDEs

偏微分方程的定义

深度之眼 deepshare.net

偏微分方程（Partial Differential Equations, PDEs）是含有多个自变量及其偏导数的方程。它们广泛应用于物理、工程、金融等领域，用于描述随时间和空间变化的现象。

$$F\left(x_1, x_2, \ldots, x_n, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots\right) = 0,$$

其中 $u = u(x_1, x_2, \ldots, x_n)$ 是待求函数。
与常微分方程的区别： PDE 的未知函数依赖于多个变量。

常见的PDE类型

- **热传导方程（抛物型）：** 描述热量在物体中的传播。
- **波动方程（双曲型）：** 描述波动的传播。
- **拉普拉斯方程（椭圆型）：** 描述稳态下的物理现象，如静电场。

图: 达朗贝尔

偏微分方程的起源(1/2)

深度之眼 deepshare.net

达朗贝尔的波动方程：1747年，达朗贝尔在研究琴弦振动的过程中，提出了一维波动方程的数学形式：

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

其中：

- $u(x,t)$ 表示琴弦的位移，依赖于时间 t 和空间位置 x；
- c 是波速，决定了波动传播的速度；
- $\frac{\partial^2 u}{\partial t^2}$ 是时间上的二阶导数，表示加速度；
- $\frac{\partial^2 u}{\partial x^2}$ 是空间上的二阶导数，表示弯曲程度。

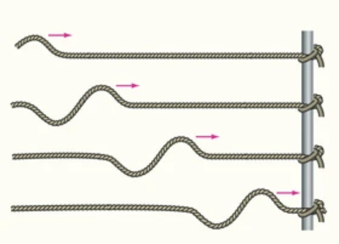


图: 波动现象

偏微分方程的起源(2/2)

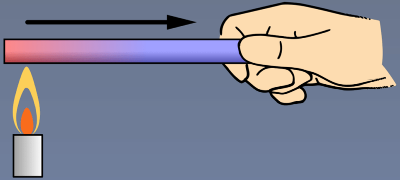
深度之眼
deepshare.net

傅里叶的热传导方程：1822年，傅里叶在研究热传导问题时，提出了著名的热传导偏微分方程：

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0,$$

其中：

- $u(x, t)$ 表示温度，依赖于时间 t 和空间位置 x；
- α 是热扩散系数，决定了热量传导的速率；
- $\nabla^2 u$ 是拉普拉斯算子，表示温度的空间分布的二阶变化，通常用于描述物体内部的热量流动。



图：热传导现象

边界条件与初始条件

深度之眼
deepshare.net

在偏微分方程（PDE）的求解过程中，边界条件和初始条件起着至关重要的作用。它们决定了问题的解的唯一性和稳定性。

边界条件（Boundary Conditions）

边界条件指定了PDE解在问题域边界上的值。常见的边界条件类型有：

- **Dirichlet边界条件：** 指定边界上的函数值，即$u(x,t)=q(x)$。
- **Neumann边界条件：** 指定边界上的导数值，即$\frac{\partial u}{\partial x}=g(x)$。
- **混合边界条件：** 结合了Dirichlet和Neumann条件，通常用来处理更复杂的物理问题。

初始条件（Initial Conditions）

深度之眼
deepshare.net

初始条件用于描述系统在时间$t = 0$时的状态。在热传导方程中，初始条件指定了初始温度分布：

$$u(x, 0) = f(x)$$

例如，在一根加热棒中，初始温度分布可能是均匀的，也可能是梯度变化的。

初始条件的重要性

初始条件对问题的解的时间演化至关重要。它提供了在$t = 0$时刻系统的“起始状态”，并决定了后续时间步的行为。

热传导方程的边界和初始条件

以一维热传导方程为例：

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2}$$

设定边界条件为：

$$u(0,t) = 100, \quad u(L,t) = 200$$

初始条件为：

$$u(x,0) = 150$$

这意味着在初始时刻，棒的温度是150℃，并且边界处的温度分别为100℃和200℃。

偏微分方程表达式

偏微分方程（PDE）： 描述物理过程，例如：

$$\mathcal{N}[u(x,t);\lambda] = s(x,t)$$

其中，$\mathcal{N}$ 是偏微分算子，$u(x,t)$ 是待求解的未知函数，λ 是系统参数，$s(x,t)$ 是已知的源项或外部激励。

边界条件（BCs）： 限制边界行为，例如：

$$u(x) = q(x) \quad \text{或} \quad \frac{\partial u}{\partial n} = g(x)$$

其中，$u(x)$ 是边界上的物理量，$g(x)$ 是已知的边界条件函数，$\frac{\partial u}{\partial n}$ 是沿边界法线方向的导数，$h(x)$ 是已知的通量边界条件函数。

初始条件（ICs）： 描述$t=0$时的状态，例如：

$$u(x,0) = f(x)$$

其中，$u(x,0)$ 是初始时刻的物理量分布，$f(x)$ 是给定的初始条件函数。

传统求解PDEs方法

传统的PDE求解方法依赖数值离散化技术，如：

- **有限差分法（FDM）：** 离散化空间和时间网格，近似求解。
- **有限元法（FEM）：** 将求解域划分为小子域，构建系统方程。
- **谱方法法（Spectral Methods）：** 使用傅里叶级数或正交基函数求解。

然而，这些方法在复杂几何和高维问题中面临挑战。

物理信息神经网络（PINNs）不依赖网格，能够处理**复杂几何和边界条件，且适用于高维问题**。通过将 PDE 转化为神经网络的损失函数，PINN 可以有效地求解多种物理问题。

PINNs的理论与损失函数构建

Physics-informed Neural Networks & Loss Functions

PINNs出现的背景

数据与物理场景的平衡：

- **数据稀缺场景：**物理知识主导。
- **大数据场景：**数据驱动方法占优势。
- **中等数据+部分物理知识：**PINN 优势显著。

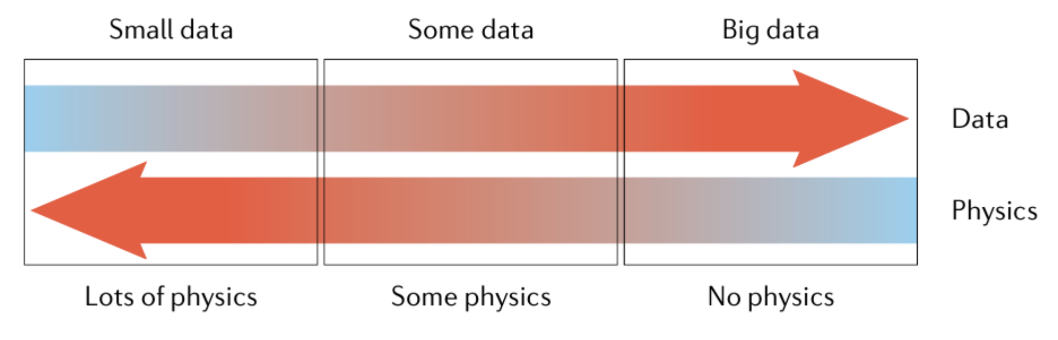


图: 数据与物理场景的平衡

理论基础的建立

PINN 的核心思想：

- 1994 年：提出用神经网络求解PDEs的研究。
- 2018 年：鄂维南等人提出用深度学习求解高维PDEs问题。
- 2019 年：Raissi 等人提出基于深度神经网络的 PINN 框架。

参考文献：

- Neural-network-based approximations for solving partial differential equations. communications in Numerical Methods in Engineering, 10(3):195–201, 1994.
- The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1):1– 12, 2018.
- Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019.

什么是物理信息神经网络（PINNs）？

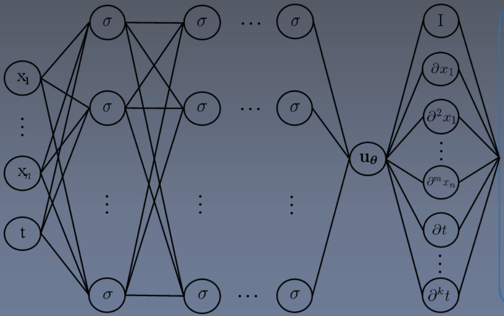
物理信息神经网络（Physics-Informed Neural Networks, PINNs）是一种结合物理规律与数据驱动学习的创新方法：

- 将偏微分方程（PDE）、边界条件和初始条件直接嵌入神经网络。
- 不仅拟合数据，还严格遵循物理规律。
- 展现出在科学计算和工程应用中的巨大潜力。

优势：

- **无需网格剖分：** 适合高维问题。
- **处理稀疏数据：** 增强模型泛化能力。
- **反问题求解：** 包括参数估计和未知规律发现。

PINNs的框架示意图



PDE residual loss:

$$\mathcal{L}_r(\Theta)=\frac{1}{N_r}\sum_{k=1}^{N_r}\left|\mathcal{F}\left[u_{\theta}\left(\boldsymbol{x}_r^k,t_r^k\right);\gamma\right]-s\left(\boldsymbol{x}_r^k,t_r^k\right)\right|^2$$

Initial condition loss:

$$\mathcal{L}_i(\boldsymbol{\theta})=\frac{1}{N_i}\sum_{k=1}^{N_i}\left|\mathcal{I}\left[u_{\theta}\left(\boldsymbol{x}_i^k,t_i^k\right)\right]-f\left(\boldsymbol{x}_i^k\right)\right|^2$$

Boundary condition loss:

$$\mathcal{L}_b(\boldsymbol{\theta})=\frac{1}{N_b}\sum_{k=1}^{N_b}\left|\mathcal{B}\left[u_{\theta}\left(\boldsymbol{x}_b^k,t_b^k\right)\right]-g\left(\boldsymbol{x}_b^k,t_b^k\right)\right|^2$$

Data loss:

$$\mathcal{L}_d(\boldsymbol{\theta})=\frac{1}{N_d}\sum_{k=1}^{N_d}\left|u_{\theta}\left(\boldsymbol{x}_d^k,t_d^k\right)-u\left(\boldsymbol{x}_d^k,t_d^k\right)\right|^2$$

Minimize:

$$\Theta^{\star}=\arg\min_{\Theta}\left[\lambda_r\mathcal{L}_r(\Theta)+\lambda_i\mathcal{L}_i(\boldsymbol{\theta})+\lambda_b\mathcal{L}_b(\boldsymbol{\theta})+\lambda_d\mathcal{L}_d(\boldsymbol{\theta})\right]$$

图: PINNs的框架：优化物理约束和数据误差

神经网络通过输出$u_\theta(x,t)$的导数计算PDE残差，同时结合边界条件、初始条件和观测数据，构建综合损失函数并优化。

PINNs正问题求解损失函数构建

PINNs的损失函数由多个部分构成：

$$\mathcal{L} = \lambda_{PDE}\mathcal{L}_{PDE} + \lambda_{BC}\mathcal{L}_{BC} + \lambda_{IC}\mathcal{L}_{IC}$$

各部分残差：

- **PDE残差：**

$$\mathcal{L}_{PDE} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|\mathcal{N}[u_\theta(x_i, t_i)] - s(x_i, t_i)\right|^2$$

- **边界条件残差：**

$$\mathcal{L}_{BC} = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}}\left|u_\theta(x_i) - q(x_i)\right|^2$$

- **初始条件残差：**

$$\mathcal{L}_{IC} = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}}\left|u_\theta(x_i, 0) - f(x_i)\right|^2$$

PINNs反问题求解损失函数构建

PINNs的损失函数由多个部分构成：

$$\mathcal{L} = \lambda_{PDE}\mathcal{L}_{PDE} + \lambda_{BC}\mathcal{L}_{BC} + \lambda_{IC}\mathcal{L}_{IC} + \lambda_{Data}\mathcal{L}_{Data}$$

数据残差：

$$\mathcal{L}_{Data} = \frac{1}{N_d}\sum_{i=1}^{N_d} |u_\theta(x_i, t_i) - u(x_i, t_i)|^2$$

深度之眼
deepshare.net

自动微分技术

Automatic Differentiation, AD

自动微分的来源和概述

自动微分（Automatic Differentiation, AD）是一种结合数值计算和符号计算的技术，用于高效计算函数导数。它广泛应用于深度学习中，尤其是在优化和梯度计算中起着关键作用。

与其他微分方法的对比：

- **数值微分：** 通过有限差分法计算导数，容易引入截断误差和舍入误差。
- **符号微分：** 基于代数操作计算导数，精确但可能导致表达式爆炸。
- **自动微分：** 结合链式法则和计算图，兼具精确性和高效性。

计算图的基本概念

自动微分依赖于构建函数的**计算图**来完成梯度计算。计算图由以下部分组成：

- **节点**：表示操作（如加法、乘法、指数等）或变量。
- **边**：表示数据流，即变量之间的依赖关系。

示例：我们以表达式

$$e = (a+b)\cdot(b+1)$$

为例，将其分解为：

$$c = a+b, \quad d = b+1, \quad e = c\cdot d.$$

计算图示意

下图展示了上述表达式的计算图：

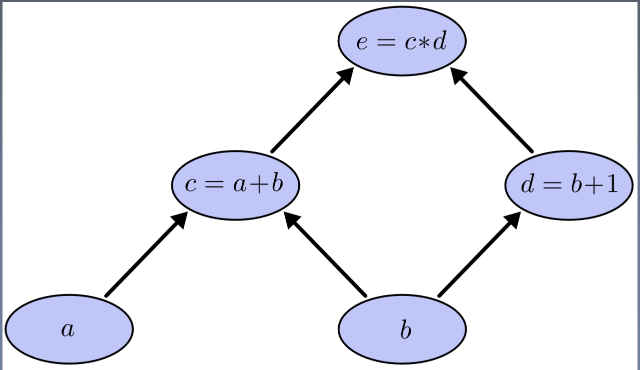


图: 表达式 $e = (a+b)\cdot(b+1)$ 的计算图

计算过程（假设 $a = 2, b = 1$）：

1. 计算 $c = a + b = 2 + 1 = 3$。
2. 计算 $d = 1 + b = 1 + 1 = 3$。
3. 最终计算
   $e = c \cdot d = 3 \cdot 2 = 6$。

基于计算图的微分

计算图不仅可以完成函数值的计算，还能通过**链式法则**高效地计算导数。例如，我们希望求 e 对 a 和 b 的偏导数 $\frac{\partial e}{\partial a}$ 和 $\frac{\partial e}{\partial b}$。

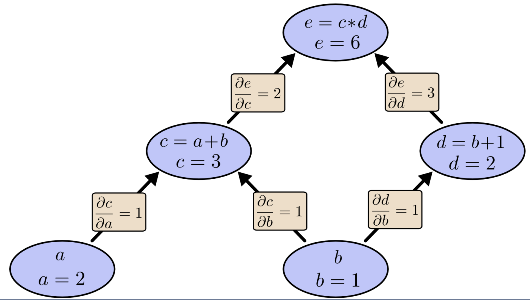


图: 计算图中的每条边标注为对应的偏导数值

链式法则:

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c_1} \cdot \frac{\partial c_1}{\partial a}, \quad \frac{\partial e}{\partial b} = \frac{\partial e}{\partial c_1} \cdot \frac{\partial c_1}{\partial b}.$$

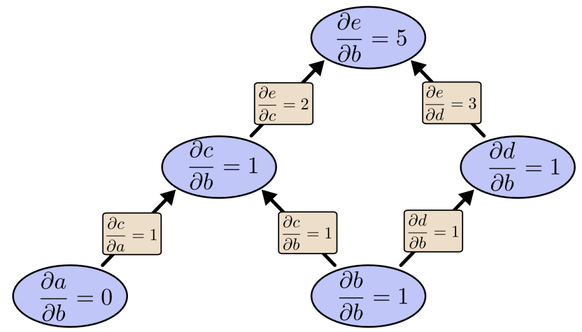
前向模式微分

前向模式微分：

- 从输入变量（如 a 和 b）开始，逐步计算每个中间节点的偏导数。
- 适合少量输入变量、多量输出变量的情况。

$\frac{\partial e}{\partial b}$计算过程：

- 初始值：a 和 b 对 b 的偏导数分别为 0 和 1。
- 从叶子节点依次向上，计算每个节点的偏导数：$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$



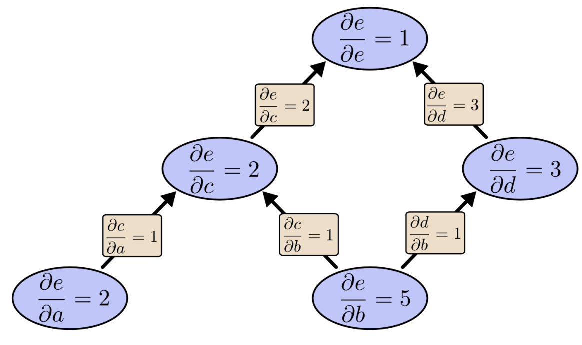
反向模式微分

反向模式微分：

- 从目标变量（如 e）开始，依次反向传播梯度。
- 适合多量输入变量、少量输出变量的情况。

计算过程：

- 初始值：$\frac{\partial e}{\partial e} = 1$。
- 从目标变量开始，逐步向下传播梯度到所有变量。



前向模式与反向模式对比

深度之眼
deepshare.net

两种模式的对比：

特性	前向模式	反向模式
计算方向	从输入到输出	从输出到输入
适用场景	少量输入，多量输出	多量输入，少量输出
深度学习中使用	较少	非常广泛

总结：

- 前向模式每次只能计算一个输入变量的偏导数。
- 反向模式一次性计算所有变量的偏导数。(DeepSeek V3：总参数量达到了6710亿)

自动微分的实现

深度之眼
deepshare.net

现代深度学习框架（如TensorFlow、PyTorch）通过动态计算图实现自动微分。以下是一个简单的示例代码：

```python
import torch

# 定义变量
x = torch.tensor(2.0, requires_grad=True)
y = x**2 + 3*x + 1

# 计算导数
y.backward()

# 输出导数
print(x.grad) # 输出为 2*x + 3, 即 7.0
```

在调用 backward() 后，框架沿计算图反向传播，计算各变量的梯度。

自动微分在 PINNs 中的作用

在PINNs中，自动微分用于高效计算PDEs中的导数。

例如，对于模型输出 $u(x,t)$，可以计算：

$$\frac{\partial u}{\partial t}, \quad \frac{\partial u}{\partial x}, \quad \frac{\partial^2 u}{\partial x^2}.$$

以下是一个具体实现示例：

```
# 定义输入
x = torch.tensor([1.0], requires_grad=True)
t = torch.tensor([2.0], requires_grad=True)
# 前向传播
u = model(x, t)
# 计算偏导数
u_t = torch.autograd.grad(u, t, create_graph=True)[0]
u_x = torch.autograd.grad(u, x, retain_graph=True, create_graph=
    True)[0]
u_xx = torch.autograd.grad(u_x, x, create_graph=True)[0]
```

PINNs将导数嵌入损失函数，以满足PDEs和边界条件的约束。

PINNs训练流程

PINNs Training Process

PINN的训练过程概述：阶段一

阶段一：计算图构建

1. **网络架构定义：** 选择神经网络结构，如全连接网络，典型结构为：
   - 输入层：空间坐标(x, t)
   - 隐藏层：4层，每层50个神经元，使用tanh激活函数
   - 输出层：物理场预测值$u_\theta(x, t)$
2. **自动微分配置：** 构建计算图跟踪机制，支持高阶导数计算，确保能够处理PDE中的导数。

PINN的训练过程概述：阶段二

深度之眼 deepshare.net

阶段二：损失函数构建与数据准备

1. **损失函数构建：** 根据正问题或反问题，构建包含PDE残差、边界条件、初始条件和数据误差的损失函数：

$$\mathcal{L} = \lambda_{PDE}\mathcal{L}_{PDE} + \lambda_{BC}\mathcal{L}_{BC} + \lambda_{IC}\mathcal{L}_{IC} + \lambda_{Data}\mathcal{L}_{Data}$$

   其中，各部分损失根据问题需求调整权重。

2. **训练数据生成：** 通过空间和时间采样，生成PDE解的样本点，同时准备边界条件、初始条件和观测数据：
   - 残差点$\{x_r^i\}_{i=1}^{N_r}$：均匀或随机采样
   - 边界点$\{x_b^i\}_{i=1}^{N_BC}$：边界区域加密采样
   - 初始时刻点$\{x_0^i\}_{i=1}^{N_IC}$：时域初始切片
   - 残差点$\{x_d^i, u_d^i\}_{i=1}^{N_d}$：均匀或随机采样

PINN的训练过程概述：阶段三

阶段三：优化策略实施

1. **优化器选择：** 通常使用Adam优化器和L-BFGS组合：
   - 前期：使用Adam快速下降，学习率通常设为$1e-3$
   - 后期：采用L-BFGS精细调优，进一步优化参数
2. **前向传播与反向传播：** 神经网络通过前向传播计算输出，并根据损失函数的残差进行反向传播优化网络参数。

PINN的训练过程概述：阶段四

深度之眼
deepshare.net

阶段四：收敛与验证

1. **收敛判断：** 根据损失函数的变化趋势，判断是否收敛。如果未收敛，则继续训练，调整训练策略。
2. **模型验证：** 通过测试集和实际数据对模型进行验证，评估泛化能力。

作业布置

给定以下欧拉梁方程及其初始和边界条件，推导PINNs求解的损失函数与Pytorch实现。

欧拉梁方程：

$$\frac{\partial^2 u(x,t)}{\partial t^2} + \alpha^2 \frac{\partial^4 u(x,t)}{\partial x^4} = 0, \quad (x,t) \in [0,1] \times [0,1]$$

其中α是与材料性质相关的参数。

初始条件：

$$u(x,0) = \sin(\pi x), \quad \frac{\partial u(x,0)}{\partial t} = 0, \quad x \in [0,1]$$

边界条件：

$$u(0,t) = u(1,t) = 0, \quad \frac{\partial^2 u(0,t)}{\partial x^2} = \frac{\partial^2 u(1,t)}{\partial x^2} = 0, \quad t \in [0,1]$$

欧拉梁PINN正问题求解

请根据给定的欧拉梁方程、初始条件和边界条件，推导出PINN模型的损失函数。

步骤：

1. 构建神经网络，输入为空间坐标x和时间t，输出为$u_\theta(x,t)$，即神经网络预测的位移。
2. 推导损失函数，包括以下几部分：
   - 需要通过神经网络输出$u_\theta(x,t)$来计算PDE残差。
   - 边界条件和初始条件也需要嵌入到损失函数中，以确保神经网络遵循这些物理规律。
3. 使用PyTorch的'torch.autograd.grad'功能，计算$u_\theta(x,t)$的高阶导数，进而构建PDE残差。

欧拉梁PINN反问题求解

通过观测数据来推断α值（与材料相关的参数），请通过PINN模型反向求解α。
步骤：

1. 构建损失函数，其中包括PDE残差、边界条件、初始条件和数据误差（如观测到的$u(x,t)$）。
2. 在神经网络中将α作为待学习的参数，反向推断α的值。
3. 使用‘torch.autograd.grad‘计算损失函数中的梯度，并优化网络来反演α。

数据误差： 请在损失函数中添加一个数据误差项，考虑观测数据与网络预测值之间的差异。

作业提交要求

- 提交包含正问题和反问题中损失函数的实现代码。
- 代码应使用PyTorch，并利用'torch.autograd.grad'计算损失函数中的高阶导数。
- 提交时请附上对损失函数推导过程的简要说明。

深度之眼

deepshare.net

公众号