四川师范大学

Sichuan Normal University

GitHub

PINNuclear-Neutrons 项目笔记

刘洋[1]

1. 四川师范大学数学科学学院, 四川成都 610066
E-mail: mathliuyang@163.com

通信作者简介：刘洋（1997-）, 男, 研究生, 主要研究方向：偏微分方程与数学物理

摘要 PINNuclear-Neutrons 项目旨在学习和复现核反应堆中子相关问题的解决方法, 使用物理反应神经网络 (PINN) 和深度机器学习技术. 的主要目标是重新实现刘东老师团队发表的“基于 PINN 深度机器学习技术求解多维中子学扩散方程”, 以更准确地模拟中子传输. 鼓励对核能技术和深度学习感兴趣的研究人员和学生参与到这个项目中. 在各个子目录中, 您将找到有关每个主题的详细说明和示例.

关键词 基于物理信息指引的神经网络模型; 深度学习; 核反应堆; 中子学扩散方程

目录

引言

随着人工智能技术的迅猛发展, 深度学习神经网络（DNNs）在解决复杂偏微分方程（PDE）的领域引起了广泛的兴趣. 在核反应堆设计和工程领域, 中子学扩散模型是核反应堆工程设计的重要组成部分. 传统的中子扩散模型采用有限差分法（FDM）和有限元法（FEM）等数值方法进行求解, 这些方法在工程实践中取得了显著的成功. 然而, 随着深度学习技术的不断发展, 一种新的方法已经崭露头角, 即基于物理信息指导的神经网络模型（PINN）, 它在解决微分方程方面展现出巨大潜力.DNNs 首次用于解微分方程是在 1998 年由 Lagaris 提出的 [**?**], 后来由 Raissi 和 Karniadakis 进一步发展成为基于物理信息的神经网络（PINNs）[**?**], 将方程和边界条件等物理信息的知识作为损失函数引入到神经网络训练当中, 通过这种方法将问题的物理信息考虑进去, 不再仅仅依靠数据驱动的方法来使得网络拟合偏微分方程的解函数; 同时, 此研究也指出如果将方程中的系数当作未知参数也可以完成发现方程的任务. 此后, 便有许多研究围绕基于物理信息的神经网络展开, 而这些研究通常可以总结为调整神经网络结构, 更改损失函数的形式, 挑选不同的激活函数, 采用不同的优化策略. PINN 模型充分利用了深度学习神经网络的通用逼近能力, 同时能够处理多维高阶复杂微分方程. 与传统数值方法相比,PINNs 具有多方面的优势, 包括无需事先收集大量训练数据、适用于正向和反向问题、无空间几何限制等. 因此,PINNs 已经引起了广泛的研究兴趣, 并在热传递、结构动力学、流体力学、固体力学以及核反应堆动力学等多个领域取得了重要突破.

本 PINNuclear-Neutrons 项目笔记旨在深入理解 PINN 模型的原理和应用, 通过推导和复现刘东老师的研究工作 [**?**], 进一步探讨 PINN 模型在核反应堆设计和工程中的潜在应用. 将重点关注 PINN 模型的工作原理、关键参数的影响以及可能的优化方向, 以便更好地理解和应用这一创新方法. 通过本学习笔记, 希望能够为核反应堆设计和工程领域的研究人员提供有关 PINNs 的深入见解, 以应对复杂问题和挑战.

1 学习笔记：基于 PINN 深度机器学习技术求解多维中子学扩散方程

在前面的学习中，已经掌握了有关物理反应神经网络（PINN）和核反应堆物理的知识。现在，将深入学习刘东老师的论文，该论文使用 PINN 深度学习技术来解决多维中子学扩散方程的问题。通过对论文内容的分析，将理解深度学习在核反应堆物理中的具体应用，包括模型构建、关键技术、数值验证等方面。

1.1 理论原理

1.1.1 中子学扩散方程模型

核反应堆工程堆芯中子学设计计算中采用中子扩散模型的方程形式如式 (1.1) 所示:

$$\begin{aligned}\frac{1}{v}\frac{\partial\phi(r,E,t)}{\partial t} &= \nabla\cdot D\nabla\phi(r,E,t) - \Sigma_{\mathrm{t}}(r,E)\phi(r,E,t)\\ &+ x(E)\int_0^\infty \vartheta\left(E'\right)\Sigma_{\mathrm{f}}\left(r,E'\right)\phi\left(r,E',t\right)\mathrm{d}E'\\ &+ S(r,E,t) + \int_0^\infty \Sigma_{\mathrm{s}}\left(r,E'\to E\right)\phi\left(r,E',t\right)\mathrm{d}E'\end{aligned} \tag{1.1}$$

方程 (1.1) 是核反应堆中描述中子行为的重要方程，其各项的物理意义 (表 1) 对于理解中子守恒和核反应堆的性能至关重要。

方程项	物理含义
$\frac{1}{v}\frac{\partial\phi(r,E,t)}{\partial t}$	通量密度变化率
$\nabla\cdot D\nabla\phi(r,E,t)$	泄漏项
$-\Sigma_{\mathrm{t}}(r,E)\phi(r,E,t)$	吸收项
$x(E)\int_0^\infty \vartheta\left(E'\right)\Sigma_{\mathrm{f}}\left(r,E'\right)\phi\left(r,E',t\right)\mathrm{d}E'$	裂变项
$S(r,E,t)$	源项
$\int_0^\infty \Sigma_{\mathrm{s}}\left(r,E'\to E\right)\phi\left(r,E',t\right)\mathrm{d}E'$	散射项

表 1 多群中子学扩散方程模型各项解释

式中, $\phi(r,E,t)$ 是 t 时刻在 r 位置处能量为 E 的中子注量率[1]; v 为中子速率; D 为中子扩散系数; ϑ 为每次裂变放出的平均中子数; $x(E)$ 是裂变中子的能谱分布; Σ_{t} Σ_{s} 和 Σ_{f} 分别是总的反应截面、散射截面和裂变截面; $S(r,E,t)$ 为外源。

其单能群存在独立的外中子源 $S_0(\boldsymbol{r},t)$ 的扩散方程[2]就可以写成

$$\frac{1}{Dv}\frac{\partial\phi(\boldsymbol{r},t)}{\partial t} = \nabla^2\phi(\boldsymbol{r},t) + \frac{k_\infty-1}{L^2}\phi(\boldsymbol{r},t) + \frac{1}{D}S_0(\boldsymbol{r},t) \tag{1.2}$$

其单能群无外源扩散方程[3]形式可写为:

$$\frac{1}{Dv}\frac{\partial\phi(r,t)}{\partial t} = \nabla^2\phi(r,t) + \frac{k_\infty-1}{L^2}\phi(r,t) \tag{1.3}$$

式中, L 为扩散长度; k_∞ 为无限介质增殖系数。

1) 在研究辐射和与能量沉积有关的问题时, 相应通量密度的总照射量常用注量 (fluence) 来度量。根据国际辐射单位和测量委员会定义, 在空间 r 处单位时间内进入以该点为中心的单位横截面的小球体内的中子数称为该点的中子注量率 (neutron fluence rate), 一般以 ϕ 表示, 中子 / $\left(\mathrm{m}^2\cdot\mathrm{s}\right)$ 。因而 Δt 时间内的注量 $F(\boldsymbol{r})$ 为

$$F(\boldsymbol{r}) = \int_{t_1}^{t_1+\Delta t}\phi(\boldsymbol{r},t)\mathrm{d}t$$

根据中子通量密度定义, 显然中子注量率就等于中子通量密度。在我国国家标准 (GB/T 3102.10-1993 和 GB/T 4960.2-1996) 中, 这两个名词都是允许使用的。

2) 推导细节见第??节式(??)

3) 推导细节见第5节式(??)

当系统处于稳态[4)]时, 式 (1.3) 形式为:

$$\nabla^2\phi(r) + \frac{k_\infty/k_{\text{eff}} - 1}{L^2}\phi(r) = 0 \tag{1.4}$$

当系统处于临界状态 ($k_{\text{eff}} = 1$) 时, 式 (3) 的形式为:

$$\nabla^2\phi(r) + B_g^2\phi(r) = 0 \tag{1.5}$$

式中, B_g^2 为系统临界时的几何曲率, 与系统的几何特性相关，临界时等于材料曲率。

式 (1.1) ∼ 式 (1.4) 可描述为统一的两阶偏微分方程形式:

$$F\left[\vec{x}, \phi(\vec{x}), \nabla\phi(\vec{x}), \nabla^2\phi(\vec{x})\right] = 0 \tag{1.6}$$

式中，$\vec{x}$ 为扩散方程几何描述空间向量。

1.1.2 利用 PINN 求解扩散方程的关键技术

利用 PINN 基本模型 (PINN 网络结构见图 **??**) 求解微分方程具有一定的普适性，但将其直接应用于中子学扩散方程，精度与收敛性上会出现一定的困难，需要针对扩散方程的特点进行改进与完善。

1.1.3 扩散方程特征向量加速收敛方法

当扩散方程是稳态临界状态时，扩散方程为式 (1.4)、式 (1.5) 形式，这在数理方法上属于特征值方程。对同一特征值来说，特征向量是无穷多个[5)], 对应于其中的任一 B_n 值, 都可以给出满足微分方程 (1.3) 及边界条件 (**??**) 的解

$$\varphi_n(x) = A_n \cos B_n x = A_n \cos\frac{(2n-1)\pi}{a}x \tag{1.7}$$

从上面的求解过程可以得出结论, 齐次方程 (1.3) 只对某些特定的特征值 B_n 才有解. 相应的解 $\varphi_n(x)$ 称为此问题的特征函数.

实验证明，这种特征值方程解的不确定性会对神经网络收敛过程造成大幅度振荡，收敛时间会大幅度增加。同时，神经网络初始值的不同会造成收敛特征向量的不同，而且，在同样的初始值下，因为学习率、批处理值的差异等因素，收敛的特征向量也可能不同，相同的初始值难以重现计算结果。

一种改进的方法是将扩散方程的特殊空间点、多个区域平均值/积分值与预设值的均方差作为的额外加权因素，采用与初始条件、边界条件相似的处理方法进行加权处理，将大幅度提高机器学习收敛效率，即便采用不同的神经网络初始值参数也会收敛到相近的结果。

特定空间点/区域理论上可选择任意注量率不为零的点/区域，实践中推荐选取求解几何域的中心处，或者具有相同注量率对称分布的不同几何点/区域，预设值可设定为任意固定的归一化注量率值。

此外, 采用已经收敛的神经网络作为类似问题的神经网络初始值, 将很大程度加速收敛过程, 这是由于深度神经网络具备一定的迁移学习特性所导致的。

4) 推导细节见第**??**节式(**??**)

5) 细节说明见第节式(1.7)

1.1.4 k_{eff} 并行搜索方法

稳态方程式 (1.4) 需要引人 k_{eff}, 并进行搜索达到临界。由于式 (1.4) 是特征值方程, 常用的是源迭代方法。基于 PINN 模型的深度学习方程求解方法, 可以直接作为其内迭代环节进行 k_{eff} 搜索计算。但由于内迭代次数较多, 若每一次内迭代均需要完整的机器学习, 迭代计算量比较大, k_{eff} 的搜索效率很低。

本文采用式 (1.3) 进行 k_{eff} 的搜索, 将式 (1.4) 写成式 (1.3) 的形式:

$$\frac{1}{Dv}\frac{\partial\phi(r,t)}{\partial t}=\nabla^2\phi(r,t)+\frac{k_\infty/k-1}{L^2}\phi(r,t) \tag{1.8}$$

给出 k 与 $\phi\left(r,t_0\right)$ 的任意初始值, 考察经过一定时间 t_t, 即 $t>t_\tau$ 时, $\phi(r,t)$ 对 t 的偏导数是否接近 0 , 从而进行临界判断。可设定合适的 ε, 若 $\partial\phi(r,t)/\partial t>\varepsilon$, 判断为超临界状态, 需要增大 k 值; 若 $\partial\phi(r,t)/\partial t<-\varepsilon$, 为次临界状态, 需要减小 k 值。当 $-\varepsilon<\partial\phi(r,t)/\partial t<\varepsilon$, 则认为系统已达临界, 此时 $k=k_{\text{eff}},\phi\left(r,t>t_{\text{r}}\right)$ 即为稳态时系统的 $\phi(r)$ 分布 (式 (1.4))。

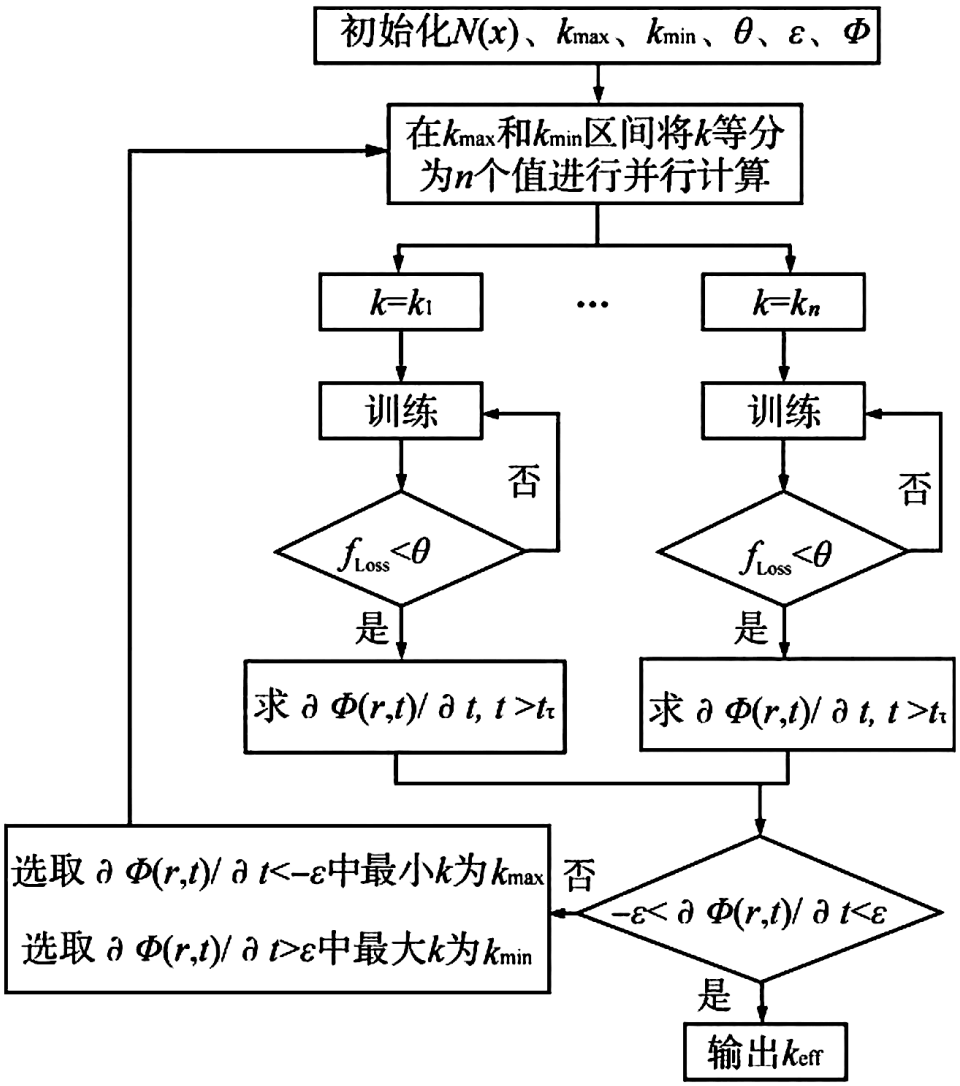

图 1.1 k_{eff} 并行搜索流程图

Figure 1.1 Flow chart of k_{eff} parallel search

t_τ 的取值应大于注量率从任意初始状态到达“稳定”变化状态的时间，根据扩散方程中反应堆类型 [表现为式 (1.8) 中的 Dv] 与初始边界条件形式会有所不同；k 值的搜索方法可采用二分法或者将 k 等分成一定数量后，大规模并行搜索，可有效提高效率。k_{eff} 的并行搜索流程如图 1.1 所示。

1.1.5 几何空间网格点密度不均匀分布策略

式 (1.1) ∼ 式 (1.4) 中空间网格几何点 i 的分布与数量, 对模型精度与机器学习效率的影响很大。通常来说, 网格点的数量越多, 计算精度越高，但计算量越大，收敛效率降低。同时，初始条件、边界条件及 $\phi(\vec{x})$ 变化很大的地方网格点密度对模型的精度与收敛效率影响很大。

为了平衡机器学习训练精度与效率, 推荐的策略是在相同网格点数量情况下，$\phi(\vec{x})$ 变化大的区

域网格点相对紧密, 变化平滑的区域相对稀疏, 以 $\phi(\vec{x})$ 在几何空间区域网格点偏导数的变化值作为网格点密度设定的判别依据。

据此, 在方程的边界条件和初始条件附近, 或者系统 $\phi(\vec{x})$ "畸性"比较大处 (即区域偏导较大处), 应加密网格点提高精度, 而在变化平缓处（即几何区域点偏导较小处），应减小网格密度以提高收敛效率。

1.2 数值验证程序复现

为了便于计算和讨论，本文选择了结构相对简单的平板几何形式的扩散方程进行数值验证。这样的选择使我们能够更清晰地理解和分析神经网络在特定几何条件下的性能，为进一步的讨论提供基础。

1.2.1 深度机器学习的超参数设置

为了验证计算结果，我们选择具有解析解的特定几何形式的扩散方程进行数值验证。这些验证结果不仅对当前方程形式有所验证，同时也为其他形式的方程和几何形状提供了参考。以下是用到的神经网络的具体超参数设置，如表 2 所示：

超参数		设置
网络结构	$\mathcal{N}(\vec{x})$	全连接神经网络
深度	l	16
中间层神经元数量	s	20
激活函数	σ	双曲正切函数 $\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
损失函数	$Loss$	$Loss: MSE_u + MSE_{\{R,BC,IC\}}$
边界权重	P_{b}	100
学习率	lr	0.001
优化器		Adam
网络初始化方法		高斯分布随机采样
训练停止条件		达到设定的训练次数

表 2 神经网络超参数及相关设置

1.2.2 临界条件下稳态扩散方程的验证

选取平板几何结构的扩散方程进行验证，该几何结构的解析解为 $C\cdot\cos(x\cdot\pi/a)$，其中 $C=0.5$ 为常数，$a=1$ 为平板的长度。我们验证计算神经网络的超参数设置见表 2。这部分将逐步讲解上述描述的泊松方程求解器的实现。作为示例，此处将使用 `Python` 语言和深度学习框架 `DeepXDE` [6] 进

6) `DeepXDE` 是一个用于求解微分方程的 `Python` 包，其主要特点是使用深度学习技术求解微分方程，其官方网站为 https://deepxde.readthedocs.io。

行求解。具体代码如下所示：

首先，导入 DeepXDE 和 numpy 等必要模块

```
import deepxde as dde
import numpy as np
import os
import matplotlib.pyplot as plt
```

初始化相关参数

```
# 初始化参数
k_eff = 1  # 有效增殖系数
a = 1  # 平板的宽度
B2 = (np.pi / a) ** 2  # 系统临界时的几何曲率
l = 16  # 神经网络的深度
s = 20  # 神经网络的中间层隐藏神经单元数量
Pb = 100  # 边界权重
C = 0.5  # 解析解参数
```

定义解析解 $C \cdot \cos(x \cdot \pi / a)$

```
# 定义解析解
def phi_analytical(x):
    return C * np.cos(x * np.pi / a)
```

使用内置的 Interval 类定义一个几何求解区域 $[-a/2, a/2]$，如下所示：

```
# 定义几何网格
geom = dde.geometry.Interval(-a / 2, a / 2)
```

接下来，表达扩散方程(1.5)的 PDE 残差函数 $\nabla^2\phi(r) + B_g^2\phi(r)$，如下所示：

```
# 定义微分方程
def pde(x, phi):
    dphi_xx = dde.grad.hessian(phi, x, i=0, j=0)
    return dphi_xx + B2 * phi
```

pde 的第一个参数是网络输入，即横坐标。第二个参数是网络输出，即解，但这里我们使用 y 作为变量的名称。

接下来，考虑平板反应堆的外推边界条件 (**??**)，即 $\phi(\pm\frac{a}{2}) = 0$。我们使用一个简单的 Python 函数，返回一个布尔值，来定义平板反应堆的外推边界条件的子域 ($\{-a/2, a/2\}$)。该函数应该对子域内的点返回 True，对于外部的点返回 False。在我们的情况下，平板反应堆的外推边界条件的点 x 是 $-a/2$ 和 $a/2$。

```
# 定义边界条件
bc = dde.icbc.DirichletBC(geom, lambda x: 0, lambda _, on_boundary: on_boundary)
```

现在，我们已经指定了几何形状、扩散方程的 PDE 残差和外推边界条件。然后，我们定义 PDE 问题：

```
# 定义数据
data = dde.data.PDE(geom, pde, bc, num_domain=898, num_boundary=2, solution=phi_analytical
    , num_test=100)
```

数字 **num_domain=898** 是在域内采样的训练残差点的数量，数字 **num_boundary=2** 是在边界上采样的训练点的数量。参数 **solution=phi_analytical** 是用于计算我们解的误差的参考解，如果我们没有参考解，可以忽略它。我们使用 **num_test=100** 个残差点来测试 PDE 残差。

接下来，我们选择网络。在这里，我们使用深度为 $l = 16$（即 16 个隐藏层）宽度为 $s = 20$ 的全连接神经网络, 其网络初始值权重采用高斯分布随机采样，并选用 $tanh$ 函数作为激活函数：

```
# 定义神经网络
layer_size = [1] + [s] * l + [1]
activation = "tanh"
# 网络初始值权重采用高斯分布随机采样
initializer = "Glorot uniform"
net = dde.nn.PFNN(layer_size, activation, initializer)
```

现在，有了 PDE 问题和网络。下面构建一个 Model，并选择 Adam 作为优化器和初始学习率 **lr=0.001**：

```
# 定义模型
model = dde.Model(data, net)
# 定义求解器
model.compile("adam", lr=0.001, metrics=["l2 relative error"], loss_weights=[1, Pb])
```

这里通过 **loss_weights=[1, Pb]** 将边界处的损失权重调整为 100，同时还选用 **metrics=["l2 relative error"]** 计算在测试点上训练好的神经网络输出与对应解析解的 L_2 相对误差作为训练过程中的一个指标。然后，对模型进行 3500 次迭代训练, 并使用 **dde.saveplot** 回调函数在训练期间保存模型和图片, 最后输出在 $x = 0$ 处的值 (即 C), 以此来显示特征值方程解 (特征向量) 的不确定性。

```
# 训练模型
losshistory, train_state = model.train(epochs=3500)
# 保存和可视化训练结果
dde.saveplot(losshistory, train_state, issave=True, isplot=True)

# 输出在 x=0 处的值(即 C)
print("Predicted value at x=0:", model.predict(np.array([0])))
```

为了深入了解模型的训练过程、监控关键指标的变化，并直观地呈现训练效果，我们采用了以下可视化策略：

```
#
    ----------------------------------------------------------------------------------------


# 可视化
# 创建保存图像的文件夹
output_folder = "figure/算例1"
os.makedirs(output_folder, exist_ok=True)

# 设置中文字体
```

```
8
9       # 替换为您的字体文件路径
10      font_path = '/System/Library/Fonts/STHeiti Light.ttc'
11
12      # 添加字体路径
13      font_properties = FontProperties(fname=font_path)
14      plt.rcParams['font.sans-serif'] = [font_properties.get_name()]
15      plt.rcParams['axes.unicode_minus'] = False
16
17      # 绘制损失函数变化图
18      loss_train = np.sum(losshistory.loss_train, axis=1)
19      loss_test = np.sum(losshistory.loss_test, axis=1)
20
21      plt.figure(figsize=(10, 6))  # 设置图像大小
22      plt.semilogy(losshistory.steps, loss_train, label="训练损失")
23      plt.semilogy(losshistory.steps, loss_test, label="测试损失")
24      for i in range(len(losshistory.metrics_test[0])):
25          plt.semilogy(
26              losshistory.steps,
27              np.array(losshistory.metrics_test)[:, i],
28              label="测试指标",
29          )
30      plt.xlabel("# 步骤")
31      plt.legend()
32      # 保存为png
33      plt.tight_layout()
34      plt.savefig(os.path.join(output_folder, "losshistory.png"), format="png", dpi=200)
35
36
37      # 绘制训练状态变化图
38      def _pack_data(train_state):
39          def merge_values(values):
40              if values is None:
41                  return None
42              return np.hstack(values) if isinstance(values, (list, tuple)) else values
43
44          y_train = merge_values(train_state.y_train)
45          y_test = merge_values(train_state.y_test)
46          best_y = merge_values(train_state.best_y)
47          best_ystd = merge_values(train_state.best_ystd)
48          return y_train, y_test, best_y, best_ystd
49
50
51      if isinstance(train_state.X_train, (list, tuple)):
52          print(
53              "错误：网络有多个输入，尚未实现绘制此类结果。"
54          )
55
56      y_train, y_test, best_y, best_ystd = _pack_data(train_state)
57      y_dim = best_y.shape[1]
58
59      # 回归分析图
```

```
    if train_state.X_test.shape[1] == 1:
        idx = np.argsort(train_state.X_test[:, 0])
        X = train_state.X_test[idx, 0]
        plt.figure(figsize=(12, 5))  # 设置图像大小
        for i in range(y_dim):
            if y_train is not None:
                plt.plot(train_state.X_train[:, 0], y_train[:, i], "ok", label="训练点分布")
            if y_test is not None:
                plt.plot(X, y_test[idx, i], "-k", label="真实")
            plt.plot(X, best_y[idx, i], "--r", label="预测")
            if best_ystd is not None:
                plt.plot(
                    X, best_y[idx, i] + 2 * best_ystd[idx, i], "-b", label="95% 置信区间"
                )
                plt.plot(X, best_y[idx, i] - 2 * best_ystd[idx, i], "-b")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.legend()
        plt.savefig(os.path.join(output_folder, "solution_state.png"), format="png", dpi=200)
    # 神经网络输出的残差
    if y_test is not None:
        plt.figure(figsize=(6, 5))  # 设置图像大小
        residual = y_test[:, 0] - best_y[:, 0]
        plt.plot(best_y[:, 0], residual, "o", zorder=1)
        plt.hlines(0, plt.xlim()[0], plt.xlim()[1], linestyles="dashed", zorder=2)
        plt.xlabel("预测值")
        plt.ylabel("残差 = 观测 - 预测")
        plt.tight_layout()
        plt.savefig(os.path.join(output_folder, "y_test.png"), format="png", dpi=200)
    # 神经网络输出的标准差
    if best_ystd is not None:
        plt.figure(figsize=(6, 5))  # 设置图像大小
        for i in range(y_dim):
            plt.plot(train_state.X_test[:, 0], best_ystd[:, i], "-b")
            plt.plot(
                train_state.X_train[:, 0],
                np.interp(
                    train_state.X_train[:, 0], train_state.X_test[:, 0], best_ystd[:, i]
                ),
                "ok",
            )
        plt.xlabel("x")
        plt.ylabel("std(y)")
        plt.tight_layout()
        plt.savefig(os.path.join(output_folder, "best_ystd.png"), format="png", dpi=200)

    # 绘制数值解与解析解的图像并比较误差
    x = np.linspace(-a / 2, a / 2, 100).reshape((-1, 1))
    y_pred = model.predict(x)
    y_exact = phi_analytical(x)

    plt.figure(figsize=(12, 5))
```

```
112     plt.subplot(1, 2, 1)
113     plt.plot(x, y_pred, label="神经网络解", color="tab:blue")
114     plt.plot(x, y_exact, label="解析解", color="tab:orange")
115     plt.title("数值解与解析解的比较")
116     plt.xlabel("x")
117     plt.ylabel("$\phi(x)$")
118     plt.legend()
119
120     plt.subplot(1, 2, 2)
121     plt.plot(x, np.abs(y_pred - y_exact), label="绝对误差", color="tab:red")
122     plt.title("数值解与解析解的绝对误差")
123     plt.xlabel("x")
124     plt.ylabel("绝对误差")
125     plt.legend()
126
127     plt.tight_layout()
128     plt.savefig(os.path.join(output_folder, "solution_comparison.png"), format="png", dpi=200)
129
130     plt.show()
```

通过深度学习网络训练的结果可视化展示了在训练过程和最终效果。首先，我们关注损失函数变化图，如图 1.2 所示。图中横轴表示训练步骤，纵轴表示误差数量级，分别展示了训练损失和测试损失的变化情况。从图中可以清晰地观察到随着训练的进行，训练损失逐渐降低，而测试损失也呈现下降趋势，这表明网络在学习过程中取得了良好的效果。此外，图中还展示了测试阶段的指标（数值解与解析解比较）变化，图中显示，随着训练的进行，测试指标几乎保持稳定，且其值相对较大，这表明数值解与解析解始终保持较大误差。这为我们提供了对网络性能的更全面认识。

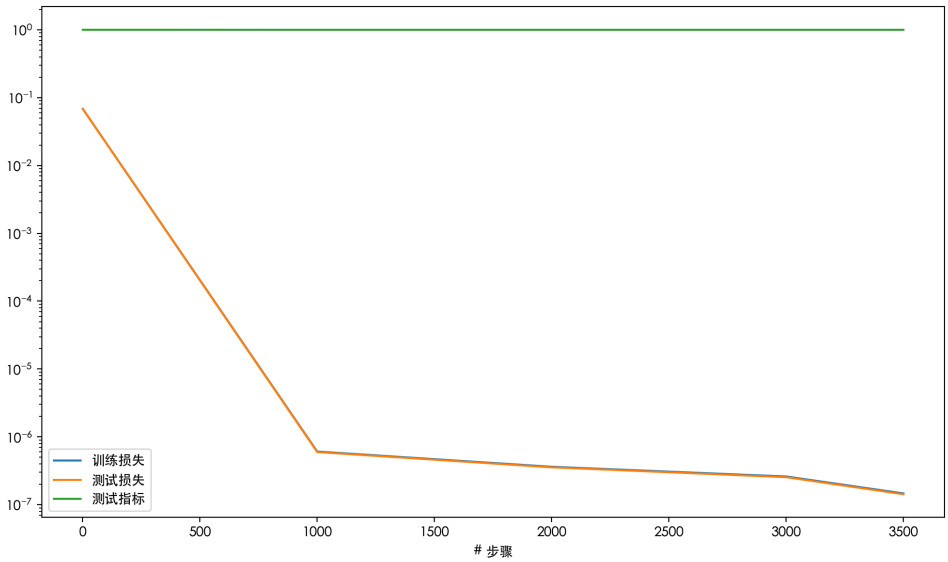

图 1.2　损失函数变化图

Figure 1.2　Loss function change

接下来，我们关注回归分析图，图 1.3 展示了神经网络训练点的采样分布情况，真实数据的曲

线以及网络的预测曲线。从图中可以看出，网络预测曲线与真实曲线趋势相差很大。

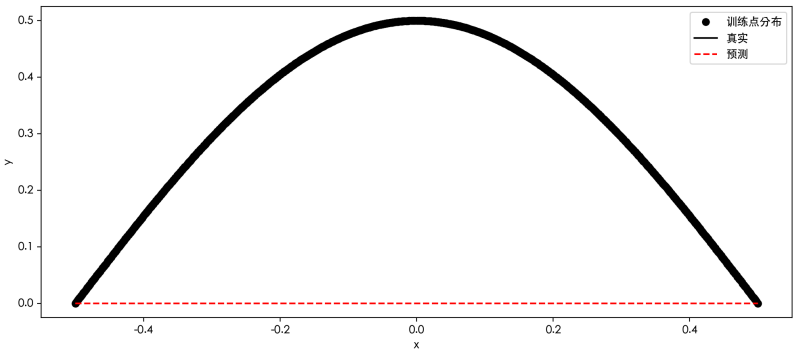

图 1.3　回归分析图
Figure 1.3　Regression analysis

另外，我们还通过图 1.4 展示了针对测试数据的残差分析。从图中可以看出，残差的分布情况较为随机，且残差的绝对值较大，这表明网络的预测效果并不理想。这些现象归因于由于特征值方程的特征向量不确定性，收敛得到的特征向量是不同的（表现为收敛时 C 值不同），且不易收敛。同时可以发现，通过神经网络的输出值几乎全为 0，这表明神经网络从众多解中选择了一个近似为 0 的解，这复合神经网络优先偏好学习低频解的特性。

为了解决此问题，我们采用 1.1.3 节中提到的改进方法：将扩散方程的特殊空间点、多个区域平均值/积分值与预设值的均方差作为的额外加权因素，采用与初始条件、边界条件相似的处理方法进行加权处理。

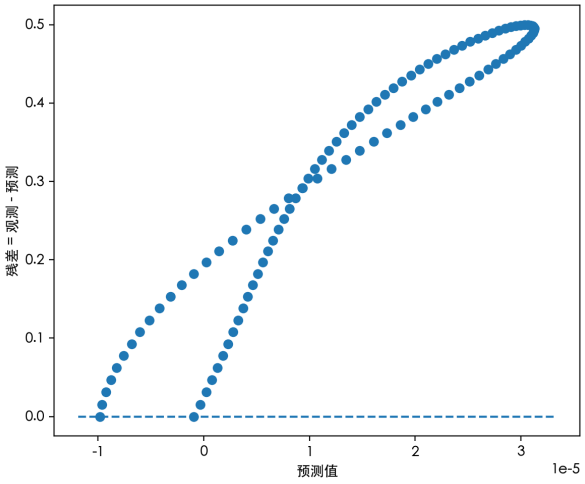

图 1.4　残差分析图
Figure 1.4　Residual analysis

1.2.3　扩散方程特征向量加速收敛方法验证

为了验证扩散方程特征向量加速收敛方法的有效性，我们选取了平板几何结构的扩散方程进行验证，该几何结构的解析解为 $C\cdot\cos(x\cdot\pi/a)$，其中 $C=0.5$ 为常数，a 为平板的长度。我们验证计算神经网络的超参数设置见表 2。

验证计算的目标是：统计并分析不同初始权重值 $\{\vec{w},\vec{b}\}$ 的神经网络 $\mathcal{N}(\vec{x})$，在训练到相似精度时，所需要的收敛时间。

算例 1、算例 2 网络初始值权重 $\{\vec{w},\vec{b}\}$ 随机选择；算例 3 也选择随机初始值，但将 $x=0$ 时，$\phi(0)$ 值 [(**??**) 解析解中的 C 值] 设定为 0.5. 算例 4、算例 5 、算例 6 初始状态为具有不同 C_0 值、

已经事先训练好的精度小于 10^{-7} 网络, 训练方式与算例 3 类似, 将 $\phi(0)=0.5$ 作为加权损失函数组成部分进行训练。各算例训练精度小于 10^{-7} 即停止, 记录所需要的训练次数及精度等相关参数, 结果如表 3 所示。

算例	网络初始值 /C_0 值	目标 C 值设定	$\sigma_{\mathrm{MSE},1}/10^{-8}$	学习次数 (n)	学习完成时 C 值
1	随机/随机	无	9.9998	2405	0.015
2	随机/随机	无	9.1095	1606	0.0022
3	随机/随机	0.5	9.6124	379	0.4995
4	已训练 /$C_0=0.01$	0.5	9.1010	211	0.4996
5	已训练 /$C_0=0.05$	0.5	9.3309	55	0.4996
6	已训练 /$C_0=0.1$	0.5	9.5475	39	0.4995

表 3 神经网络初始值敏感性分析结果

可见，算例 1、算例 2 分别收敛到不同的 C 值, 训练次数较多, 这说明了若不给定目标 C 值作为训练的加权 $Loss$, 由于特征值方程的特征向量不确定性, 收敛得到的特征向量是不同的 (表现为收敛时 C 值不同), 且不易收敛。算例 3 训练次数大幅度下降, 而算例 4、算例 5 、算例 6 由于有预训练初始值, 最终训练次数进一步降低, 且初始 C 值越接近目标值 $C=0.5$, 收敛越快, 这表明了采用已训练好的神经网络作为相似问题神经网络初始值, 具有良好的迁移学习特性, 可大幅度加快收敛进程。

2 结束语

本文主要介绍了 PINNuclear-Neutrons 项目的学习笔记, 主要是对 PINNuclear-Neutrons 项目的学习过程进行了总结, 复习了论文中的算例代码, 并对项目中的一些关键代码进行了解读, 对项目中的一些关键参数进行了调整, 以便更好地理解论文中的算法.

PINNuclear-Neutrons Project Notes

Liu Yang[1]

1. *School of Mathematical Sciences, Sichuan Normal University, Chengdu* 610068*, Sichuan*
E-mail: mathliuyang@163.com

Abstract PINNuclear-Neutrons project aims to learn and reproduce the solution of neutron related problems in nuclear reactor, using physical reaction neural network (PINN) and deep machine learning technology. Our main goal is to re-implement the "Solving Multidimensional Neutron Diffusion Equation Based on PINN Deep Machine Learning Technology" published by Liu Dong's team, so as to simulate neutron transport more accurately.

Keywords PINN; Deep Learning; Nuclear Reactor; Neutron Diffusion Equation

Liu Yang was born in Sichuan in 1997. He is currently a master's student in the School of Mathematical Sciences, Sichuan Normal University. His research interests include partial differential equations and mathematical physics.