理解深度学习

许志钦

2023 年 9 月 13 日

目录

1 深度学习解微分方程 **2**

1.1 引言 2
1.2 传统的迭代算法 2
1.3 传统算法的困难 4
1.4 深度学习优势与不足 5
1.5 深度学习解 PDE 的常用方法 6
1.6 用深度学习参数化 PDE 解的方法 7
1.6.1 Physics-informed neural network (PINN) 7
1.6.2 Deep Ritz method 9
1.6.3 深度学习方法与迭代算法的收敛速度差异 10
1.6.4 多尺度神经网络解 PDE 12
1.7 用深度学习参数化 PDE 解算子 15
1.7.1 基于格林函数的神经网络算子 15
1.7.2 DeepONet 15
1.7.3 用神经网络参数化算子 15

Chapter 1

深度学习解微分方程

1.1 引言

微分方程是许多领域用来描述基本原理的语言。因此，解决许多科学问题或者工程问题的核心是解微分方程。比如牛顿第二定律就是用微分方程描述宏观物体运动的基本规律，麦克斯韦方程描述电磁场的演化规律等。一般的微分方程可以分成常微分方程（ODE）和偏微分方程（PDE）。ODE 中仅考虑状态量关于时间的导数信息，可以包含高阶的时间导信息。PDE 不仅考虑时间导数信息，还考虑了状态量关于空间的导数，同样是可以包含高阶导的信息。若 PDE 中不包含时间导数，则是稳态方程。计算数学领域针对不同的方程发展了很多数值算法，是近几十年发展速度很快的领域。常用的数值算法有欧拉法、有限元、有限体积等。一方面计算数学这个领域已经很成功也很成熟，另一方面也体现该领域进入了发展的瓶颈期，很多困难的问题长期无法解决。我们将在后面的章节具体讨论这些困难。

最近几年，以神经网络形式出现的深度学习，在计算机视觉、语音识别、自然语言处理等诸多领域的成功引起了计算数学家的关注。人们逐渐意识到深度学习方法有可能成为提升研究传统科学问题能力的一个非常有潜力的工具。使用深度学习求解偏微分方程吸引了越来越多的注意力 [16]，并被寄予了很大的希望可以突破应用数学领域的瓶颈 [15]。

1.2 传统的迭代算法

在介绍神经网络求解偏微分方程的方法之前，我们先来回顾一下传统的数值解法。以求解 Poisson 方程为例 [7]，

一维 Poisson 方程:

$$-\Delta u(x) = g(x), \quad x \in \Omega = (-1,1) \tag{1.1}$$
$$u(x) = 0, \quad x = -1, 1.$$

我们考虑一种简单常用的传统方法–有限差分方法。首先离散求解区间, 取网格间距为 $h = 2/n$, 得到 $[-1,1]$ 上 $n+1$ 个等距节点。其次使用中心差分格式求解 Eq. (1.1),

$$-\Delta u_i = -\frac{u_{i+1} - 2u_i + u_{i-1}}{(\delta x)^2} = g(x_i), \quad i = 1, 2, \cdots, n, \tag{1.2}$$

可以写成如下线性系统,

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{g}, \tag{1.3}$$

其中

$$\boldsymbol{A} = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \cdots & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}_{(n-1)\times(n-1)}, \tag{1.4}$$

$$\boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix}, \quad \boldsymbol{g} = (\delta x)^2 \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{pmatrix}, \quad x_i = 2\frac{i}{n}. \tag{1.5}$$

有很多方法可以用来求解这个线性系统，我们这里用雅可比迭代 (Jacobi iteration) 方法。

令 $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, 其中 $\boldsymbol{D}$ 是 $\boldsymbol{A}$ 的对角矩阵, $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 分别是 $-\boldsymbol{A}$ 的严格下三角和上三角部分，则我们有

$$\boldsymbol{u} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{u} + \boldsymbol{D}^{-1}\boldsymbol{g}. \tag{1.6}$$

在第 $t \in \mathbb{N}$ 步, 雅可比迭代公式如下,

$$\boldsymbol{u}^{t+1} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{u}^t + \boldsymbol{D}^{-1}\boldsymbol{g}. \tag{1.7}$$

我们对上述迭代过程进行标准误差分析。用 $\boldsymbol{u}^*$ 表示直接对 Eq. (1.3) 中的 $\boldsymbol{A}$ 求逆得到的真解。第 $t+1$ 步的误差是 $\boldsymbol{e}^{t+1} = \boldsymbol{u}^{t+1} - \boldsymbol{u}^*$. 则可以得到 $\boldsymbol{e}^{t+1} = \boldsymbol{R}_J\boldsymbol{e}^t$, 其中 $\boldsymbol{R}_J = \boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$. 因此 $\boldsymbol{e}^t$ 的收敛速度是由 $\boldsymbol{R}_J$ 的特征值决定的, 也就是,

$$\lambda_k = \lambda_k(\boldsymbol{R}_J) = \cos\frac{k\pi}{n}, \quad k = 1, 2, \cdots, n-1, \tag{1.8}$$

相应的特征向量 $\boldsymbol{v}_k$ 的分量是

$$v_{k,i} = \sin\frac{ik\pi}{n}, i = 1, 2, \cdots, n-1. \tag{1.9}$$

所以我们可以写出

$$\boldsymbol{e}^t = \sum_{k=1}^{n-1} \alpha_k^t \boldsymbol{v}_k, \tag{1.10}$$

其中 α_k^t 可以被理解为 $\boldsymbol{e}^t$ 在 $\boldsymbol{v}_k$ 方向的幅值。则

$$\boldsymbol{e}^{t+1} = \sum_{k=1}^{n-1} \alpha_k^t \boldsymbol{R}_J \boldsymbol{v}_k = \sum_{k=1}^{n-1} \alpha_k^t \lambda_k \boldsymbol{v}_k. \tag{1.11}$$

$$\alpha_k^{t+1} = \lambda_k \alpha_k^t.$$

因此，$\boldsymbol{e}^t$ 在 $\boldsymbol{v}_k$ 方向上的收敛速度是被 λ_k 控制的. 因为

$$\cos\frac{k\pi}{n} = -\cos\frac{(n-k)\pi}{n}, \tag{1.12}$$

频率 k 和 $(n-k)$ 是紧密相关的并且以相同的速率收敛的. 因此只需要考虑频率 $k < n/2$, 可以看到频率越低，λ_k 越大。因此, 在雅可比迭代方法中低频收敛慢，也就是说先学到高频。为了解决低频收敛慢的问题，一系列的工作发展了多重网格算法 [1]。

1.3 传统算法的困难

上一节的例子是一维线性的二阶稳态方程，传统的算法可以非常快速且以高精度得到它的数值解。但实际问题一般会有很多困难，在这一节中，我们扼要地举一些传统算法面临的困难。

高维问题。传统的数值算法都需要对问题的区域打网格。当问题的维度变高时，打网格是一件变得非常困难的事情。现在有商业软件是专门用来给问题区域打网格的。另一方面，网格的数量随着维度增加会指数增加，使得高维问题的处理面临维数灾难。举一个均匀网格的例子，假设每个维度取 100 个等距点，3 维的情况就有 100 万个点。对于像 Boltzmann 方程，有七个维度，需要 10^{14} 个点，像多体薛定谔方程，每个粒子的空间位置有三个维度，动量有三个维度，n 个粒子就有 $6n$ 个维度，而实际关心的系统的粒子数目量级都非常高，比如阿伏加德罗常 10^{23} 这样的大小。

多尺度特征。实际系统在空间和时间上都会呈现出很多特征尺度，不同尺度的现象会相互作用，各个尺度可能都非常重要，比如湍流。为了在模拟中抓住小尺度的物理演化，数值模拟中需要取非常小的时间或者空间步长，使得模拟对计算量要求非常大，而且非常慢。

编程。前面的一维例子看起来很简单，但当考虑有限元或者有限体积时，编程的难度就变得非常高，特别地，当问题的区域不是简单的几何形状（如方形）时，对边界的处理会变得非

常繁琐。这些编程困难劝退了很多学习者或者应用的人，他们已经习惯调用商业软件来解决各类问题，比如 Fluent 等。

复杂的几何。在偏微分方程中，当问题的几何区域非常复杂时，比如有孔的问题或者不规则的边界，在传统算法的处理过程中会变得很麻烦，比如打网格时就要照顾很多细节。

间断。在偏微分方程中，实际的问题可能会出间断的情况，比如击波，或者光在不同介质中传播，这些情况都会出现解的间断或者导数的间断，使得解具有人奇异。

刚性。在常微分方程中，刚性指的是关心的状态量在非常小的时间尺度内有较大的变化，比如燃烧化学反应问题。

反问题。反问题的求解是重要但困难的问题。有一些反问题的算法涉及到频繁地求解正问题。因此，正问题求解的速度对求反问题至关重要。

1.4 深度学习优势与不足

深度学习的成功吸引了各行各业的注意，它的优势在解方程中也是重要的。

没有维数灾难。深度学习成功的实际例子绝大多数都是非常高维的例子。尽管目前还没有严格理论完全解释清楚神经网络对于哪一些高维问题的能做得特别好，但大量的实践暗示着当碰到高维问题时，神经网络常常是一个可能的解决方案。它的优势可能在于特征学习的能力，以及在拟合高维问题时，网络的大小不会随模型的维度指数增长，也就是没有维数灾难。

无网格。神经网络解 PDE 时，一般可以在区域内对所有的点进行随机采样，然后用自动微分，最小优方程左右端的差值。因此，可以避免打网格的操作。

编程容易。由于现在神经网络已经有很多成熟的软件包，现在用神经网络解相关的问题的代码的门槛变得非常低。

计算高效。由于可以充分利用矩阵的并行和 GPU 的架构，现在神经网络的计算非常高效，并且现在神经网络非常流行，大量的软硬件会使计算越来越高效。

自动微分。自动微分的技术避免了数值微分，可以使精度更高，同时使很多问题变得很方便。对任意参数的优化都可以用几行命令完成。

但神经网络也有非常明显地不足。

训练数据很重要。神经网络的泛化性非常依赖于训练数据的质量。对于高维问题，随机采样在大部分情况是难以满足要求的的，比如一个 100 维的问题，即使每个维度仅有两个数据点，总的数据数目就有 2^{100} 个。即使随机采 10 亿个数据，也不到全部数据的百亿分之一。而高维的问题一般都有特定的结果，如果采到的数据刚好能够抓住这些特殊的结构，实际上采样数据可以大大减少。但如何进行合适地采样却往往是非常困难的问题。

精度不高。用神经网络解 PDE 时，许多人常常会根据传统的计算数学方法问这个神经网络算法具有几阶精度。而实际训练网络时谈几阶精度往往是奢侈的，因为经常训练到 loss 为

10^{-3} 左右后，训练很难再往下，更不用谈几阶精度的概念。因此，神经网络算法有时也被诟病其精度不高。工程应用对精度的要求常常不会特别高，所以神经网络算法尽管精度不高，但仍然有它的市场。

高频灾难。正如频率原则指出的，神经网络在学习高频时速度很慢，也就是高频灾难。在一些 PDE 的问题中，高频是重要的，因此，高频灾难是不容忽视的。

缺乏理论。使用神经网络常常遇到的一个问题是不理解它的工作原理，让使用者不能放心或者不能估计该方法可能出错的地方。这就是缺乏理论的方法所面临的问题。

训练代价大。虽然神经网络在推断时速度非常快，但其训练时需要大量的计算，特别是对于网络规模比较大的情况。

1.5 深度学习解 PDE 的常用方法

现在，用神经网络来解 PDE 吸引了越来越多的注意力 [5]。我们简单来看一下有哪些常用的方法。

第一类常用的方法是用神经网络来参数化 PDE 的解。基本的做法是用一个带参数的神经网络表示 PDE 的解，然后通过调整参数使得神经网络所表示的函数满足一些特定的目标函数的最小值。而这些目标函数的最小值在数学上能够被保证是对应 PDE 的解。最简单的目标函数可以是方程左端和右端的差的平方 [4, 14]，物理驱动的神经网络（Physics-informed neural network, PINN），或者 PDE 的变分形式 [6, 11]，比如 Deep Ritz 方法。

参数化解的方法可以解非常高维的 PDE 并且不需要任何事先计算的标签，但它每次训练只能解一个特定的方程。也就是说，当方法有一点点变化，比如源项发生变化时或者边界条件发生变化时，我们必须要重新训练神经网络。高频灾难在这种情况经常被观察到，为了缓解高频困难，一系列的多尺度神经网络方法也被相应地提出来 [3, 12]。

第二类常用的方法是用神经网络参数化 PDE 的算子，以解决前面提到的重复训练的问题。这类方法有两种常见的情况，第一种是直接学习所关心的函数到解之间的映射，一般是考虑固定网格上这些函数的值，比如从源项在网格上的值，也就是一个向量，到解在网格上的解，也就是一个向量的映射。这种情况需要事先解很多次 PDE，然后获得一系列标签，再训练神经网络。另一种情况是利用 PDE 解的结构，用神经网络只参数化结构中的一部分 [10, 9]。比如对于线性的 PDE，其解可以表示成格林函数和源项的卷积的结构，因此神经网络只需要参数化格林函数，每给一个新的源项后，只需要将源项和神经网络做卷积就可以得到新的解。再比如 DeepOnet [13] 利用解关于源项的特殊的表示形式构造（包含非线性的情况）对应的算法结构。一般情况下，这些算子方法都需要事先解一系列 PDE 做为标签，这些使算子方法在使用时受到很大的限制。

还一些混合的方法，比如我们也可以用物理驱动的想法，利用方程本身来构造损失函数，

减少对标签的依赖。尽管理论上可行，但实际使用时，纯物理驱动的做法很难训练，有数据的训练会比较容易。实验上发现，添加少量的数据标签（可能是精度不高的标签）对算子使用物理驱动的损失函数会有较大的帮助 [17]。

1.6 用深度学习参数化 PDE 解的方法

1.6.1 Physics-informed neural network (PINN)

1994 年，Dissanayake 和 Phan-Thien 的文章 [4] 提出使用神经网络解一般 PDE 的想法。考虑如下方程

$$\mathcal{L}u = f, \quad x \in \Omega, \tag{1.13}$$

$$\mathcal{B}u = g, \quad x \in \partial\Omega. \tag{1.14}$$

考虑一个通用逼近函数 $u_a(x, \beta_i)$，其中 β_i 表示参数，通过如下优化问题可以寻找方程的近似解

$$h = \int_\Omega ||\mathcal{L}u_a - f||^2 dV + \int_{\partial\Omega} ||\mathcal{B}u_a - g||^2 dS. \tag{1.15}$$

注意，这里的符号是为了保持和原文 [4] 保持一致。该文章发表后引用率一直很低，从谷歌的数据查询，九十年代的引用只有十来篇。这阶段恰好也是深度学习进入第二阶低谷期。

2018 年, 布朗的大学的 Karniadakis 教授和学生发表的文章 [14] 题目为《Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations》。在方法上，2018 年的文章和 1994 年的文章几乎没有差异，但 2018 年的文章的数值实验更丰富，也提出了解反问题的观点。但注意，2018 年的文章并没引用 1994 年的文章，同时，2018 年的文章的影响范围已经远远超出数学领域，特别是在工程领域取得了非常大的影响。截止 2023 年 1 月 9 日，2014 年的文章引用数为 271，2018 年的文章引用数为 4282。可以看到，在科学研究上，第一次提出方法的工作固然非常重要，但推广方法的工作也是非常重要。例如 1986 年, David E. Rumelhart，Geoffrey E. Hinton 和 Ronald J. Williams 在 Nature 发表了一篇用向后传播方法训练神经网络的文章，这篇文章成为深度学习发展历史上的一个里程碑，它明确指出向后传播的算法不是在这篇文章里首次提出的，但它是让向后传播算法起到很大影响的工作。下面我们来看一下 2018 年的文章是怎么描述方法的。

PINNs (physical-informed neural networks，物理信息神经网络) 接受空间和时间坐标值作为输入，并通过微分方程构建损失函数来更新网络，而不是纯粹的数据驱动。在这种情况下，网络充当通用函数近似器，并且可以使用自动微分的概念来计算相对于空间和时间坐标的偏导数。

考虑如下一般形式的偏微分方程

$$u_t + \mathcal{N}[u] = 0, x \in \Omega,\ t \in [0, T],$$

这里 $u(t, x)$ 是方程的解函数 (latent solution, 形式表达) , $\mathcal{N}[\cdot]$ 是一个非线性的微分算子, $\Omega \subset \mathbb{R}^D$.

PINN 利用

$$f(t, x) := u_t + \mathcal{N}[u],$$

定义微分方程的残差 (residual) 并用一个有大量参数的深度神经网络来逼近 $u_\theta(t, x)$。损失函数为

$$MSE = MSE_{eq} + MSE_{ibc},$$

其中

$$MSE_{eq} = \frac{1}{N_{eq}} \sum_{i=1}^{N_{eq}} |f(t^i, x^i)|^2,$$

还有

$$MSE_{ibc} = \frac{1}{N_{ibc}} \sum_{j=1}^{N_{ibc}} |u_\theta(t^j, x^j) - u^j|^2.$$

这里, $\{t^j, x^j, u^j\}_{j=1}^{N_{ibc}}$ 表示初始或者边界条件的训练数据, $\{t^i, x^i\}_{i=1}^{N_{eq}}$ 是为 $f(t, x)$ 指定的配点 (collocations points)。

为说明 PINNs 的效果, 我们考虑如下 Burgers 方程,

$$q_t + \frac{1}{2}(q^2)_x = 0.01/\pi q_{xx} \text{ in } [-1, 1] \times [0, 1]. \tag{1.16}$$

其中初始条件和边界条件是,

$$q(x, 0) = -\sin(\pi x),\ q(-1, t) = q(1, t) = 0.$$

用 PINNs 学到的结果如图 1.1所示, 图 1.1的第一行是解析解的可视图, 颜色反应的是函数值的大小, 其中黑色的 cross 代表初始条件和边界田间的训练点 (采样点)。第二行的三张图是为了看的更加清楚神经网络学到的解和真实解析解之间的差距, 从左到右分别对应 $t = 0.25, 0.50, 0.75$ 时的解, 可以看到这三个时刻神经网络的预测值和真实解很接近。因此我们可以相信 PINNs 方法的确学到了这个方程的解函数。

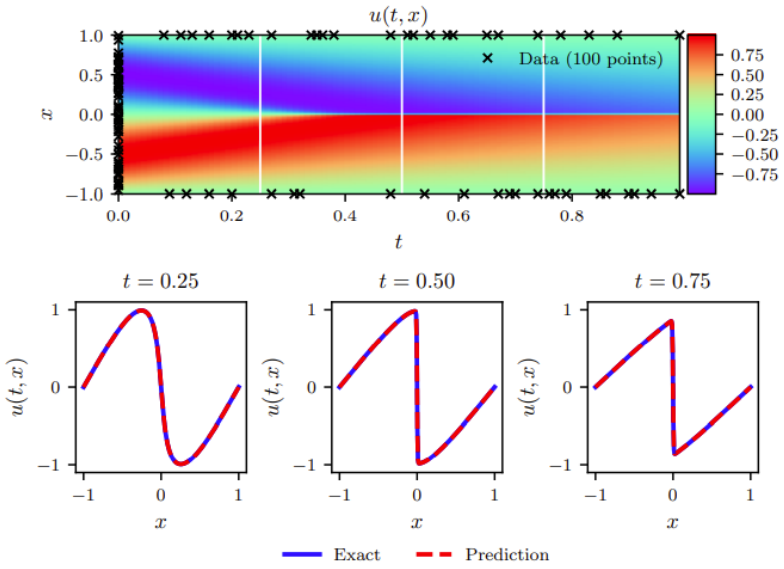


图 1.1: 比较 PINN 和有限差分在 Burgers 方程的解，时间步为 $t = 0.25, 0.50, 0.75$. 图片来源于 [14]。

1.6.2 Deep Ritz method

用神经网络学习偏微分方程的解，关键步骤是以最小化偏微分方程的残差来约束神经网络。除了像 PINNs 中用方程构建均方误差，也可以采用其他方式。E.&Yu 提出了一种基于偏微分方程的变分形式来构造损失函数来约束神经网络的方法 [6]，Deep ritz method。

对于一个 PDE 问题，考虑如下的变分问题，

$$\min_{u \in H} I(u)$$

其中 H 代表可行函数的集合 (也可以称为试验函数 trial function, 用 u 来表示)，变分问题的最小值如果满足 PDE 的解，那我们可以把解 PDE 的问题转化为求解变分问题的最小值问题。我们来看文章 [6] 中一个泊松方程的例子，

$$\begin{aligned} -\Delta u(x) &= 1, \quad x \in \Omega, \\ u(x) &= 0, \quad x \in \partial\Omega, \end{aligned}$$

其中 $\Omega = (-1,1) \times (-1,1) \backslash [0,1) \times \{0\}$.

这个问题的解受到众所周知的“角奇异性（corner singularity）”的影响，这一奇性是由定义域的特点引起的。我们用修正形式的泛函

$$I(u) = \int_{\Omega} (\frac{1}{2}|\nabla u(x)|^2 - f(x)u(x))dx + \beta \int_{\partial\Omega} u(x)^2 dx$$

来作为训练神经网络的损失函数。注意右端的第二个积分是采用惩罚项的想法来限制边界条件，针对这个问题用的 $\beta = 500$，这个参数的值要根据实际问题进行调节。为了比较传统数值解与 Deep ritz method 的区别，实验中还用到了有限差分方法求解这个微分方程。结果如图 1.2所示，左边 Deep Ritz method 学到的解，右边是有限差分方法的结果。对比两种方法的结果，可以看到，Deep ritz method 的解是不错的。

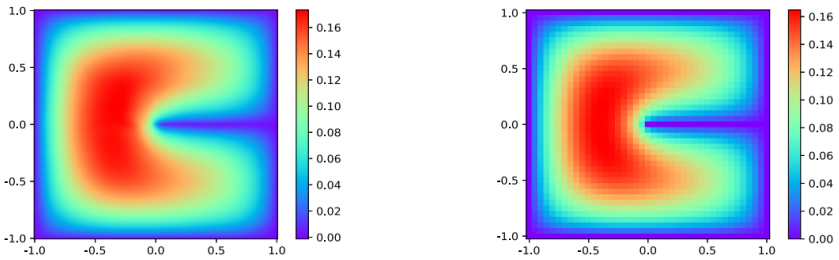


图 1.2: Solutions computed by two different methods. On the left is Deep Ritz with 811 parameters. On the right is the solution of the finite difference method on a uniform grid with 1681 parameters.

Deep Ritz 这篇文章有一个 100 维的例子

$$-\Delta u(x) = -200, \quad x \in (0,1)^{100},$$
$$u(x) = \sum_k x_k^2, \quad x \in \partial(0,1)^{100}.$$

用神经网络学习后，相对误差可以降到 2.2%。这个例子说明神经网络有潜力可以克服传统算法中遇到的“高维灾难”。

1.6.3 深度学习方法与迭代算法的收敛速度差异

传统的数值算法一般最后会转化成求解线性方程组的问题。当方程的数目太大时，比如网格点很多的情况下，一般会采用迭代的算法求解。根据前边的分析，雅可比迭代这种传统方法是低频收敛极慢，那大家不妨猜一下用神经网络解方程是否有类似的特点呢？事实上正相反，其收敛正如频率原则揭示一样，神经网络先抓住低频的东西，而高频收敛较慢。

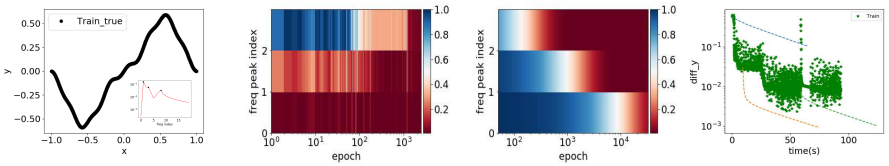
下面我们通过实验来展示雅可比迭代与神经网络求解偏微分方程在频率角度的差异。我们考虑一个具体的泊松方程，令 $g(x) = \sin(x) + 4\sin(4x) - 8\sin(8x) + 16\sin(24x)$, 则 Possion 方程有解析解 $u_{\text{ref}}(x) = g_0(x) + c_1 x + c_0$, 其中 $g_0 = \sin(x) + \sin(4x)/4 - \sin(8x)/8 + \sin(24x)/36$, $c_1 = (g_0(-1) - g_0(1))/2$, $c_0 = -(g_0(-1) + g_0(1))/2$. 等间距的在 $[0, 1]$ 取 1001 训练样本 $\{x_i\}_{i=0}^n$, 其中网格间距是 δx。雅可比迭代的结果见图. 1.3(c)，可以看到高频收敛快。为了对比，这里我们也用神经网络的输出 $h(x;\theta)$, 去拟合方程的解 $u_{\text{ref}}(x)$ (图. 1.3(a))。注意这里神经网络求解方程是通过定义如下的损失函数 [6],

$$I_{\text{emp}} = \sum_{i=1}^{n-1} \left(\frac{1}{2} |\nabla_x h(x_i)|^2 - g(x_i) h(x_i) \right) \delta x + \beta \left(h(x_0)^2 + h(x_n)^2 \right). \tag{1.17}$$

$I_{\text{emp}}(h)$ 中的第二项是罚项 (penalty), 权重系数 β 是取一个固定常数, 这一项是根据 Dirichlet 边界条件得来的 (1.1). 这里其实用到的就是下一节要介绍的神经网络求解偏微分方程中的 Deep ritz 方法。大家可以先忽略神经网络求解方法的细节，具体的我们将在下一节介绍。不过具体训练的细节，最终神经网络的输出和真解 u_{ref} 很接近。不仅如此，我们将注意力放到观察对 u_{ref} 做傅里叶变换后得到的函数中三个峰的收敛 (见图 1.3(a))，在图 1.3(b) 中, 可以看到神经网络的训练过程中正如频率原则所说一样，低频比高频收敛的快。

可以看到从频率角度来看，传统方法和神经网络表现出完全不同的收敛特点：传统方法高频收敛快，低频收敛慢，而神经网络低频收敛快，高频收敛慢。那如果两者结合在一起，会有什么表现呢？我们可以来看下面这个实验。

我们首先用神经网络来解 Eq. (1.1) 中的 Poisson 方程，其中 $g(x) = \sin(x) + 4\sin(4x) - 8\sin(8x) + 16\sin(24x)$。注意这里我们要仔细选择停止训练时的步数 M (or epochs) 以确保得到一个比较好的初始化 (initial guess), 即此时神经网络已经学到了低频部分 (即对应了前边分析中 $\boldsymbol{R}_J$ 大的特征值)。接着，我们用这个神经网络在离散格点上的函数值作为新的初始值用于雅可比迭代方法来进行接下来的迭代。实验中我们用 $\|h - u_{\text{ref}}\|_\infty \triangleq \max_{x\in\Omega} |h(x) - u_{\text{ref}}(x)|$ 来衡量得到的结果。正如图. 1.3(d) 所示，绿色的星标记代表只使用神经网络，误差值在迭代一些时间后 $\|h - u_{\text{ref}}\|_\infty$ 来回波动。而虚线表示以相应 M 步的神经网络输出值为初始化数据集，用雅可比迭代后得到结果。如果 M 太小 (也就是生成初始数据集时神经网络训练停止的太早,)，此时等价于只使用雅可比迭代，其结果在图上对应的是左边的虚线, 可以看到需要经过长时间才能收敛到一个小的误差值，原因在于低频收敛的仍很慢。反之，如果 M 太大，(停止的太晚,) 此时等价于只使用神经网络，其结果在图上对应的是右边的虚线, 可以看到高频的收敛速度慢，会浪费很多时间。选择了合适的 M 将得到的神经网络的输出值作为雅可比迭代的初始点，其对应结果由橙色的虚线表示, 可以看到低频先迅速被神经网络捕捉到，紧接着高频部分通过雅可比迭代方法迅速收敛。



(a) (b) (c) (d)

图 1.3: 泊松方程. (a) $u_{\text{ref}}(x)$. 插入图: $|\hat{u}_{\text{ref}}(k)|$ 关于频率 k 的函数图像. 黑色的点对应频率的峰值。(b,c) 针对所选的几个频率，对 DNN(b) 和 Jacobi 方法 (c) 分别计算在不同 epoch 处得到的训练数据上的 $\Delta_F(k)$. (d) $\|h - u_{\text{ref}}\|_\infty$ 在不同运行时间上的值。绿色的星代表单独使用神经网络的 $\|h - u_{\text{ref}}\|_\infty$. 虚线代表使用雅可比迭代方法的 $\|h - u_{\text{ref}}\|_\infty$，不同颜色的虚线代表以经过不同训练时间得到的神经网络值作为初始点的演化。

这个例子给我们带来了启发，有时在解决问题时单独使用 DNN 可能不是最佳选择，因为它在高频下具有缓慢收敛的限制。利用 DNN 和传统方法的优势来设计更快的方案可能是科学计算问题的一个有希望的方向，比如 [8] 用神经网络做为迭代算法的初始值在很多问题上加速了求解。

1.6.4 多尺度神经网络解 PDE

基于对频率原则的理解，我们为了解决高频学习慢的特点，可以对神经网络进行一些特殊设计。有一些方法直接给高频成分加更多的权重，有一些方法是把频率空间的每个高频成分都平移到低频（如 PhaseDNN[2]）。这些工作的问题在于对频率空间的显示操作或者逐个平移都会面临维数灾难。随着维数上升，傅里叶变换的计算也会非常困难。

为了避免在频域上进行显式操作，在前面第?? 章中，我们设计了一个非常简单的多尺度神经网络（MscaleDNN 结构）[12]。基本的想法是，对于一个函数，如果你把它拉伸一下，它的频率就会变低。对不同频率的成分，我们可以做不同的拉伸，使得它们变成频率差不多的函数。下面，我们来看看几个数值算例。

宽频域的例子

首先，我们看一下二维的宽频域的例子，来源于 [12]。考虑定义在 $\Omega = [-1, 1]^d$ 上的 Poisson equation,

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{1.18}$$

其中

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} 4\mu^2 x_i^2 \sin(\mu x_i^2) - 2\mu \cos(\mu x_i^2). \tag{1.19}$$

这个方程有解析解，

$$u(\boldsymbol{x}) = \sum_{i=1}^{d} \sin(\mu x_i^2), \tag{1.20}$$

这个解析解也可以为方程 (1.18) 提供了边界条件。

我们使用 Deep Ritz 方法进行解，这样可以减少对激活函数可导性的要求。在每个训练时期，我们在定义域内采样 5000 点，在边界上采样 4000 点。我们研究以下两个结构，

1. 一个全连接神经网络 (fully-connected DNN) 的规模 **1-1000-1000-1000-1** (**normal**).
2. 一个 MscaleDNN 具有 5 个子网络,其规模是 **1-200-200-200-1**, 和标量系数 $\{1, 2, 4, 8, 16\}$. (**Mscale**).

正常的全连接网络使用 ReLU 激活函数，MscaleDNN 采样一个有限支撑的激活函数（图1.4），

$$\phi(x) = \mathrm{ReLU}(x)^2 - 3\mathrm{ReLU}(x-1)^2 + 3\mathrm{ReLU}(x-2)^2 - \mathrm{ReLU}(x-3)^2. \tag{1.21}$$

事实上，关于激活函数的选择是一个比较关键的因素，我们后面做一些讨论。这个问题没有一

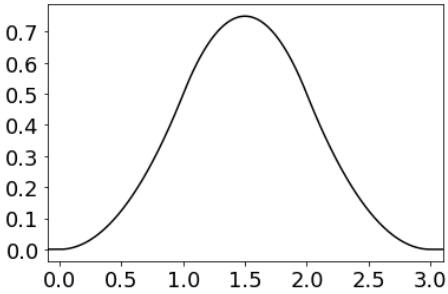


图 1.4: 激活函数 $\phi(x)$。图片来源于 [12]。

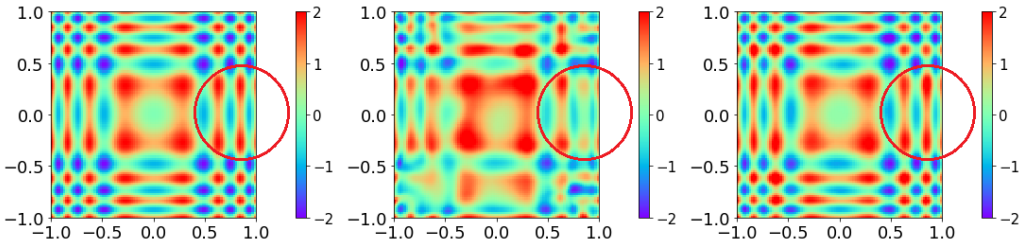
个固定的频率，而是一个广泛的频率范围。一个常用的全连接 DNN 将很难解决这个问题。对于 $\mu = 15$，二维情况下问题 (1.18) 的精确解如图1.5 (a) 所示为一个高度振荡的函数。图1.5 (b) 中普通 DNN 得到的解无法捕捉到振荡结构，而图1.5 (c) 中 MscaleDNN 得到的解很好地捕捉到了不同尺度的振荡。例如，在红色圆圈所标记的区域，正常网络的解中预期的振荡几乎消失，而 MscaleDNN 解很好地解决了振荡。四个角处的振荡也有类似的行为差异。

误差随迭代进行的变化展示在图1.6中，显然，MscaleDNN 误差下降比较快.

多孔二维的例子

接下来，我们看一下多孔二维的例子，来源于 [12]。考虑问题 (1.18) 带有如下源项：

$$f(\boldsymbol{x}) = 2\mu^2 \sin \mu x_1 \ \sin \mu x_2, \mu = 7\pi. \tag{1.22}$$



(a) 精确解　　(b) 正常神经网络　　(c) MscaleDNN

图 1.5: 二维情况下的问题 (1.18)。例如，当正常的全连接网络无法准确抓住振荡时，MscaleDNN 很好地捕获了振荡。图片来源于 [12]。

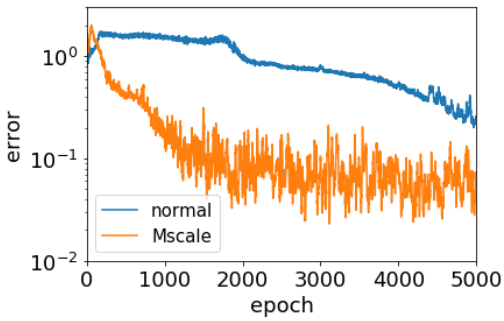
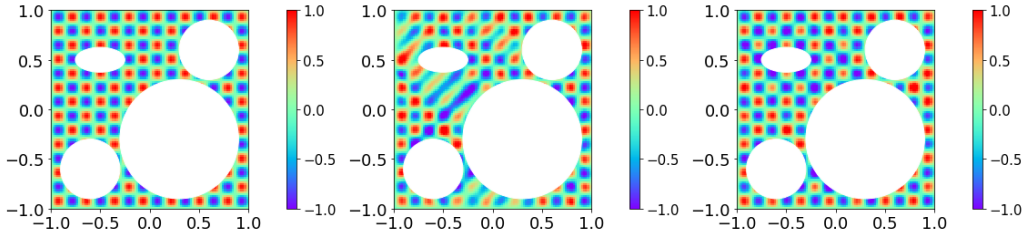


图 1.6: 误差下降图。图片来源于 [12]。

精确解为

$$u(\boldsymbol{x}) = \sin \mu x_1 \sin \mu x_2. \tag{1.23}$$

我们同样用精确解来给边界条件。采样和神经网络结构和前面的一样。如图1.7所示，多尺度神经网络可以准确地抓住精确解中的振荡。



(a) 精确解　(b) 正常神经网络　(c) MscaleDNN

图 1.7: 多孔二维的例子。图片来源于 [12]。

1.7 用深度学习参数化 PDE 解算子

1.7.1 基于格林函数的神经网络算子

1.7.2 DeepONet

1.7.3 用神经网络参数化算子

传统方法与直接参数化解的神经网络方法，有一个相同的问题，只能求解单个方程，那找一个例子（源项或者边界条件改变，要解一族方程的），如果每次都要重新求解计算代价很大，有没有其他的方法?

事实上可以，引入目前的参数化算子的方法。

FNO

MOD-Net

学算子，实际上 Green 函数可以，但只有极少数的情况可以写出解析解。通过前边的学习知道 DNN 有很强的拟合能力，那能否用 DNN 来学习格林函数呢?

参考文献

[1] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*, volume 72. Siam, 2000.

[2] Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency wave equations in inhomogeneous media. *Arxiv preprint, arXiv:1909.11759*, 2019.

[3] Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing*, 42(5):A3285–A3312, 2020.

[4] MWMG Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[5] Weinan E, Jiequn Han, Arnulf Jentzen, et al. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. *arXiv preprint arXiv:2008.13333*, 2020.

[6] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[7] Lawrence C Evans. Partial differential equations. 2010.

[8] Jianguo Huang, Haoqin Wang, and Haizhao Yang. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, 2020.

[9] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[10] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[11] Yulei Liao and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2021.

[12] Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *Communications in Computational Physics*, 28(5):1970–2001, 2020.

[13] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[14] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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.

[15] E Weinan. The dawning of a new era in applied mathematics. *Notices of the American Mathematical Society*, 68(4):565–571, 2021.

[16] E Weinan, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional pdes: from nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

[17] L. Zhang, T. Luo, Y. Zhang, W. E, Z. Q. J. Xu, and Z. Ma. MOD-Net: A machine learning approach via model-operator-data network for solving pdes. *Communications in Computational Physics*, 32(2):299–335, 2022.