# 神经网络应用：求解偏微分方程

神经网络在数学上是参数可学习的一个函数（映射）/ 泛函。在统计学习领域使用的较多，之前已经接触过它的拟合任务。分类任务可以看作目标集合离散的映射学习。

将神经网络看作是输入的函数，可以作为一个含参待定函数作为偏微分方程（PDE）的解，用在自动求解偏微分方程上。

## 如何求解复杂的物理系统（PDE）？

偏微分方程是描述自然界众多现象的语言，比如流体的流动、热量的传导、电磁场的分布等等。无论是数学上还是工程上，各类偏微分方程的理论求解和数值求解技术都是非常重要的研究领域。

目前，除了少数简单的情况（抛物方程、椭圆方程、双曲方程），绝大多数PDE都很难求出其解析解。

因此，数值求解算法，特别是适用于多种方程的数值算法应用前景广泛，目前常用的数值算法包括：

- **有限差分法**： 把连续的空间和时间离散化成网格，用差分代替微分。
- **有限元法**：将复杂的区域分解成许多小单元，在每个单元上用简单的函数来近似解。
- **有限体积法**：基于守恒律。将求解区域划分为一系列互不重叠的“控制体积”。对每一个控制体积直接积分PDE（通常是守恒形式的），将体积分转化为通过控制体积表面的通量积分。该方法保证在离散层面上，物理量（如质量、动量、能量）在每一个控制体积内都是守恒的。
- **谱方法**：将解在整个求解区域内用一组全局性的、光滑的基函数（如傅里叶级数、切比雪夫多项式）的级数来表示。通过确定基函数的系数来求解PDE。
- **边界元法、粒子方法**等

传统方法在各自适应的领域非常成功，是现代科学计算的基石。但它们也有一些共同的缺点：

1. **网格依赖：** 非常多的方法都需要生成计算网格，对于复杂几何形状，网格生成本身就是一个难题。
2. **维数灾难：** 当问题维度升高时（比如描述一个多粒子系统），网格点的数量会指数级增长，计算成本变得无法承受。
3. **数据驱动难题：** 如果我们有一些实验测量数据，想用于辅助求解PDE或识别参数，把这些数据融入到传统的网格方法中较为困难。

深度学习求解偏微分方程技术尝试绕开网格，用神经网络作为一个强大的 “万能函数” 来直接表示PDE的解。可以有效的应对上面的挑战。

## 理论基础

传统方法（FDM, FEM, FVM）在离散空间（网格或单元）上构建代数方程组来“逼近”PDE的解。而神经网络方法则采用了一种完全不同的哲学：**用一个神经网络本身作为PDE解的泛函表达形式。**一个自然的问题是：神经网络能近似好一个解吗？

更数学 / 一般的提法是：神经网络可以在何种程度上近似一个给定的函数空间？

这个问题的答案是**万能逼近定理。**万能逼近定理（Universal Approximation Theorem）实际上是一系列定理，最早的工作可以追溯到20世纪50~60年代的Kolmogorov–Arnold表示定理，而目前使用较多的是以下三个结果：

1. 足够宽神经网络可以以任意精度逼近连续函数空间。

   1989年，George Cybenko证明了sigmoid激活函数的通用逼近性。同年，Kurt Hornik、Maxwell Stinchcombe和Halbert White证明，即使只有一个隐藏层的多层前馈神经网络也是通用逼近器。Hornik还在1991年证明，赋予神经网络通用逼近能力的并非激活函数的具体选择，而是多层前馈架构本身。1993年，Moshe Leshno等人以及后来的Allan Pinkus在1999年证明，通用逼近性质等价于使用非多项式激活函数。

   Matlab神经网络工具箱正是基于这一理论结果，只需要指定网络宽度，就可以完成拟合任务。打开神经网络拟合工具箱，深度被固定为2层，较早版本的Matlab只使用一层隐藏层。

**定理 (宽网络)**: 令 $\sigma$ 是一个非多项式的激活函数。对于定义在 $\mathbb{R}^n$ 的一个紧集 $\mathbb{X}$ 上的任意连续函数 $f : \mathbb{X} \to \mathbb{R}$，以及任意误差容忍度 $\epsilon > 0$，都存在一个单隐藏层的神经网络 $f_{\text{NN}}$，其形式为：

$$f_{\text{NN}}(\mathbf{x}) = \sum_{j=1}^{M} c_j \sigma(\mathbf{w}_j \cdot \mathbf{x} + b_j)$$

使得对于所有 $\mathbf{x} \in \mathbb{X}$，都有 $|f(\mathbf{x}) - f_{\text{NN}}(\mathbf{x})| < \epsilon$ 成立。

2. 足够深神经网络可以在积分意义下以任意精度逼近可积函数空间。

   任意深度的情况也曾被许多学者研究过，例如 Gustaf Gripenberg、 Dmitry Yarotsky、 Zhou Lu、Boris Hanin 和 Mark Sellke他们主要研究了使用 ReLU 激活函数的神经网络。2020 年，Patrick Kidger 和 Terry Lyons将这些结果扩展到了使用一般激活函数（例如 tanh 或 GeLU） 的神经网络。

   **定理 (深网络)**: 对于任意勒贝格可积函数 $f : \mathbb{R}^n \to \mathbb{R}$ 以及任意 $\epsilon > 0$，都存在一个使用 ReLU 作为激活函数的神经网络 $f_{\text{NN}}$，其每一层的宽度 $W$ 满足 $W \leq n + 4$（其中 $n$ 是输入维度），使得：

$$\int_{\mathbb{R}^n} |f(\mathbf{x}) - f_{\text{NN}}(\mathbf{x})| d\mathbf{x} < \epsilon$$

3. 有限深度和有限宽度的ReLU网络可以逼近连续函数空间。

   Maiorov 和 Pinkus 于 1999 年首次研究了有界深度和有界宽度的情况。他们证明存在一个解析的 sigmoid 激活函数，使得隐藏层单元数量有界的两个隐藏层神经网络是通用逼近器。

   2018年，Guliyev和Ismailov构建了一个平滑的sigmoid激活函数，该函数为具有较少隐藏层单元的双隐藏层前馈神经网络提供了通用逼近特性。同年，他们还构建了具有有界宽度的单隐藏层网络，这些网络对于单变量函数仍然是通用逼近器。然而，这并不适用于多变量函数。

   2022 年，Shen等人获得了关于深度和宽度的 ReLU 神经网络近似目标函数所需的深度和宽度的精确定量信息。

   **Theorem 1.1**

   *Given a continuous function $f \in C([0,1]^d)$, for any $N \in \mathbb{N}^+$, $L \in \mathbb{N}^+$, and $p \in [1, \infty]$, there exists a function $\phi$ implemented by a ReLU network with width $C_1 \max\{d\lfloor N^{1/d} \rfloor, N + 2\}$ and depth $11L + C_2$ such that $\|f - \phi\|_{L^p([0,1]^d)} \leq 131\sqrt{d}\,\omega_f\left(\left(N^2 L^2 \log_3(N+2)\right)^{-1/d}\right)$, where $C_1 = 16$ and $C_2 = 18$ if $p \in [1, \infty)$; $C_1 = 3^{d+3}$ and $C_2 = 18 + 2d$ if $p = \infty$.*

PDE的解 $u(x,t)$ 本身就是一个函数。万能逼近定理告诉我们，必然存在一个神经网络 $\hat{u}(x,t;\theta)$，其输出可以无限接近这个真实解。这就为使用神经网络作为解的表达形式提供了坚实的数学基础。我们的任务就从“求解网格点上的值” 转变为 “寻找一组最优的网络参数 $\theta$，使得 $\hat{u}(x,t;\theta)$ 最好地满足PDE和定解条件” 。

需要说明的是，上面的两个定理实际上不会对实际计算带来多大帮助，因为对于一个解函数未知的问题，通常只知道其连续性（甚至这个也不知道），而定理无法告诉我们多深 / 多宽的网络就可以较好的满足我们的精度需求。类似于不知道余项公式的泰勒逼近定理。

最后一个定理给出了网络超参数的定量结果，但是其界是非常宽的，也没有实际上的指导意义。例如，近似一个二维空间中的一个Lipschitz连续函数 (设连续系数为1) 到 $L^2$ 积分误差小于 $10^{-3}$. 需要：

$$\frac{131\sqrt{2}}{\sqrt{N^2 L^2 \log_3(N+2)}} \leq 10^{-3}$$

随意取一组值：$L = 131, N = 590$ 是较接近等于的满足条件的参数。此时宽度为9472个神经元、深度为1459个隐藏层。约1300亿的参数，需要450GB以上的存储空间。

# 如何使用神经网络求解偏微分方程（PINN）

在前面的学习中，我们大量练习了神经网络拟合函数的过程，因此直接使用神经网络在拟合的思想下求解PDE是自然的想法：

> “数据驱动”（Data Driven）的神经网络求解PDE算法：
> 在域内和边界上采集大量样本点 $(x_i, t_i)$，并知道这些点上解的真值 $u_i$（可能是通过高精度仿真或实验得到），然后用神经网络去做一个纯粹的监督学习，拟合这些数据。拟合完成后，神经网络函数就是方程解函数的一个近似。

$$Loss = \sum_{i \in I} |u(x_i, t_i; \theta) - u_i|^2$$

神经网络求解偏微分方程领域早期有非常多这方面的工作，随着各类问题的数据集都“消耗殆尽”，目前纯粹使用数据驱动方法来求解PDE的技术文章已经不多，而且主要集中在算子学习或者反问题领域。

这种方法有一个巨大的问题：它严重依赖大量高保真的训练数据。如果我们已经有了全场的精确解**信息**，那还求解PDE干什么呢？我们的目标恰恰是在**没有**或**仅有少量**数据的情况下，也能找到满足物理规律的解。

如何在不依赖大量数据的前提下，引导神经网络去寻找那个符合物理规律的解呢？2019年，Raissi等人提出了一个划时代的想法：将物理规律本身，也就是PDE，作为惩罚项加入到神经网络的损失函数中！

PDE告诉了我们解函数`u`的各阶导数之间应该满足的关系，例如：$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0$。这个关系在域内**处处**都应该成立。

而神经网络是可微的，我们可以利用自动微分精确地计算出网络输出`u`对输入`(x, t)`的导数，`∂u/∂t`，`∂²u/∂x²`。然后，我们要求这些导数代入PDE后，结果尽可能地接近零。

$$Loss = \sum_{i \in I} \| \frac{\partial u}{\partial t}(x_i, t_i) - \alpha \frac{\partial^2 u}{\partial^2 x}(x_i, t_i) \|^2$$

以上面的损失函数为优化目标训练出来的神经网络，自然就会(近似)满足方程的等式约束。除此之外把PDE求解所用到的初边值条件代入，可以得到(近似)满足初边值和方程的函数，即解的近似。

> PINN的损失函数构造：PDE残差 + 初始条件 + 边界条件

$$Loss = \sum_{i \in I_1} \| \frac{\partial u}{\partial t}(x_i, t_i) - \alpha \frac{\partial^2 u}{\partial^2 x}(x_i, t_i) \|^2 + \lambda_1 \sum_{i \in I_2} \|u(x_i, 0) - u_{init}(x_i, 0)\|^2 + \lambda_2 \sum_{i \in I_3} \|u(x_i, t_i) - u_{bd}(x_i, t_i)\|^2$$

Raissi等人将物理规律的描述（方程、初边值）嵌入神经网络的损失函数，提出了Physics-Informed Neural Networks（PINN），是目前使用最广泛的神经网络解PDE技术之一。

对于一个一般的偏微分方程：

$$\begin{cases} L[u(x)] = f(x), & x \in \Omega \\ B[u(x)] = 0, & x \in \Gamma \end{cases}$$

PINN的损失函数可以写为：

$$Loss = \|L[u(x)] - f(x)\|_{L^2(\Omega)} + \lambda \|B[u(x)]\|_{L^2(\Gamma)}$$

离散化之后可以用于训练。

几个关键问题：

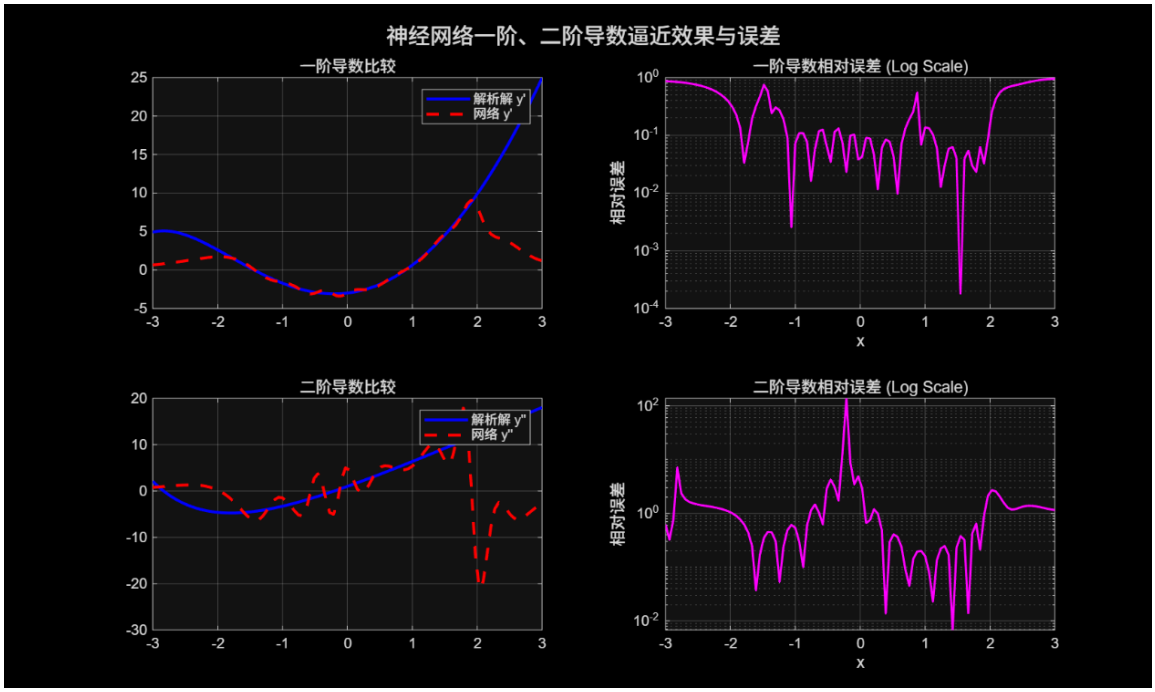
1. 离散化的数据集应该怎么采样？

   这是目前的热门研究方向之一，除了最简单的 $\Omega$ 内均匀采样，还有在解近奇异部分多采样（重要性采样）策略。在高维PDE问题上如何确保采样均匀（正交拉丁方采样策略）也是研究方向之一。

2. 权重超参数 $\lambda$ 如何选择？

在很多问题上都有实验表明PINN对权重超参数非常敏感，两部分损失函数收敛速度是否协同、是否先收敛复杂方程再考虑简单边界、多种归一化策略和自适应权重等等问题都是研究的热点方向。

3. 自动微分缓慢，高阶梯度变化剧烈如何解决（正则化）？




考虑PDE的弱形式（二阶梯度将为一阶梯度），增加正则化技术（归一化、梯度裁剪、引入正则化项）等方向也有很多研究。

## 实例

使用PINN求解Burger's方程。

ref：使用物理信息神经网络求解 PDE

## 作业

使用PINN求解如下的一维波动方程：

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, & x \in [0,1], t \in [0,2] \\ u(x,0) = \sin(\pi x), & (\mathrm{IC1})\ x \in [0,1] \\ \frac{\partial u}{\partial t}(x,0) = 0, & (\mathrm{IC2})\ x \in [0,1] \\ u(0,t) = u(1,t) = 0, & (\mathrm{BC})\ t \in [0,2] \end{cases}$$

上述PDE有解析解：$u(x,t) = \sin(\pi x)\cos(c\pi t)$. **波速 $c$ 设置为 0.5**

当你训练好网络后，使用如下代码生成测试点，并将你的网络在测试点的预测结果保存为工作区变量。

```
% 定义x和t坐标向量
x = 0:0.01:1;     % x从0到1，步长0.01
t = 0:0.01:2;     % t从0到2，步长0.01

% 生成网格坐标
[X, T] = meshgrid(x, t);
```

你保存的文件应该为 $201 \times 101$ 的double类型矩阵，以`学号.mat`命名。

如果使用Python，请同样导出为`.mat`格式。

提交：

1. 代码文件
2. 简单的报告（docx/pdf），仅需要包含训练截图和绘图截图。
3. 以`学号.mat`命名的数据，将用于评测机评分。
4. 提交截至：周一晚23:59

# 参考


1. https://ww2.mathworks.cn/help/deeplearning/ug/solve-partial-differential-equations-with-lbfgs-method-and-deep-learning.html
2. https://en.wikipedia.org/wiki/Universal_approximation_theorem
3. https://zhuanlan.zhihu.com/p/81933892473
4. https://www.cambridge.org/core/journals/acta-numerica/article/abs/approximation-theory-of-the-mlp-model-in-neural-networks/18072C558C8410C4F92A82BCC8FC8CF9