深度神经网络求解偏微分方程

肖新宇
数学科学学院
北京大学
2001110035

Abstract

经典的偏微分方程求解方法包括有限差分，有限元，谱方法等。这些方法基本上都需要生成网格，而在复杂的区域或者高维问题上，这种网格方法由于维数灾难而失效。近年来，随着深度学习的兴起，一种利用深度神经网络 (DNN) 逼近解的无网格方法 [8] 也崭露头角。为了验证这种方法的有效性，在这篇报告中利用 DNN 成功求解了泊松方程（不含时的情况），Burgers 方程（含时非线性）。在泊松方程的例子中，我采取了一种改进方法避免边界惩罚项获得了比原始方案更精确的解。在含时的 Burgers 方程的例子中，我将 NN(Neural Network) 的近似解与经典的傅里叶谱方法的高精度解作为对比。发现在求解非含时和含时方程时，DNN 方法都得到了不错的结果。

1 引言

深度学习 [6] 在图像的分类、目标检测等许多任务上都取得了最好的结果，近年来迅速来应用到各个科学与计算领域。这得益于其良好的逼近能力、灵活性和解决维度灾难的能力。人们很自然地会想到一种合适的方法来使用深度学习来求解偏微分方程。[11] 提出了一种通过神经网络逼近解的方法，并使用 Ritz 方法来解决泊松型方程。[9][8] 使用了平方误差损失函数 (DGM:Deep Garlerkin Method) 解含时方程。[3][10][2] 则首先将偏微分方程转化为后向随机微分方程，然后使用神经网络或其他机器学习方法来逼近它们的解。

在本报告中，我们重点关注的是 Deep Galerkin 方法 [8]。本报告的内容组织如下：

- 在第 2 节中，介绍了 DNN 求解 PDE 的思路，并简要推导了该算法的理论，给出了一个启发性的解释.
- 在第 3 节中使用 DNN 方法求解了 Poisson 方程，并对该算法提出了一个避免惩罚项的改进，得到了更精确的解。

- 在第 4 节中使用 DNN 方法求解了 Burgers 方程，并将其结果与经典的傅里叶谱方法结果进行了对比。
- 在第 5 节中总结了这种方法的优点和缺点，并对未来的研究方向做了一些展望。

2 Deep Galerkin Method

考虑一个含时或者不含时的偏微分方程初边值问题 (不含时的情况就去掉下面公式中的 ∂_t)：

$$\begin{cases} (\partial_t + \mathcal{L})\, u(t,\boldsymbol{x}) = 0, & (t,\boldsymbol{x}) \in [0,T] \times \Omega \\ u(0,\boldsymbol{x}) = u_0(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \\ u(t,\boldsymbol{x}) = g(t,\boldsymbol{x}), & (t,\boldsymbol{x}) \in [0,T] \times \partial\Omega \end{cases} \tag{1}$$

使用一个神经网络逼近解函数：

$$u(x,t) \approx \hat{u}(x,t,\theta) \tag{2}$$

构建 DGM 损失函数：

$$\begin{aligned} J(\theta) &= \text{loss}_1 + \text{loss}_2 + \text{loss}_3 \\ \text{loss}_1 &= \left\| \tfrac{\partial \hat{u}}{\partial t}(t,x;\theta) + \mathcal{L}\hat{u}(t,x;\theta) \right\|^2_{[0,T]\times\Omega,\nu_1} \\ \text{loss}_2 &= \|\hat{u}(t,x;\theta) - g(t,x)\|^2_{[0,T]\times\partial\Omega,\nu_2} \\ \text{loss}_3 &= \|\hat{u}(0,x;\theta) - u_0(x)\|^2_{\Omega,\nu_3} \end{aligned} \tag{3}$$

$loss_1$ 代表内部方程的损失项，强制要求神经网络拟合的函数在区域内部满足方程（在使用了 Monta Carlo 采样后，即要求方程在某些配点上成立）。$loss_2$e 和 $loss_3$ 分别为边界惩罚项，初始条件惩罚项，强制要求 $\hat{u}(x,t,\theta)$ 满足初边值条件。
我们的目标是求解 $\hat{u}(x,t,\theta)$ 使得损失函数极小：

$$\begin{cases} \text{Find } \hat{u} \in H & x \in \Omega, \\ minimize\ I(\hat{u}) \end{cases} \tag{4}$$

下面考虑 Poisson 方程这个损失函数下的变分方程以说明 Poisson 方程的解是3的唯一极值点，（对 Burgers 方程的变分形式没有推出类似结果，只能说明 Burgers 方程的解是一个极小点，不能保证唯一性）。

2.1 Poisson equation 变分方程推导

考虑泊松方程 DGM 损失函数的变分：

$$\begin{aligned} I(u) &= \int_\Omega \|\Delta u + f\|^2 dx + \int_{\partial\Omega} \|u - g\|^2 ds \\ \delta I &= 2\int_\Omega (\Delta u + f)\Delta(\delta u) dx + 2\int_{\partial\Omega} (u-g)\delta u\ ds \end{aligned}$$

令 $v = \Delta u + f$，并利用 Green 第二等式可知：

$$\delta I = 2\int_\Omega \Delta(v)\delta u\ dx + 2\int_{\partial\Omega} \left[v\frac{\partial \delta u}{\partial n} + (u - g - \frac{\partial v}{\partial n})\delta u \right] ds$$

若该变分方程对任意 δu 都成立，则可知

$$\begin{cases} \Delta v = 0, & x \in \Omega, \\ v = 0, & x \in \partial\Omega. \\ u = g + \frac{\partial v}{\partial n}, & x \in \partial\Omega. \end{cases} \tag{5}$$

由5的前两式可知 $v = 0$, 所以可以推出：

$$\begin{cases} -\Delta u = f, & x \in \Omega \\ u = g, & x \in \partial\Omega. \end{cases} \tag{6}$$

这就说明该变分方程的解就是 Poisson 方程的解。也就说明了 Poisson 方程是 DGM 损失函数的唯一极值点。

2.2 DNN 网络结构

在构建 PDE 解的逼近函数2时，使用的 Deep-learning 中常用的的 Res-Net[4] 结构。

$$\begin{aligned} s_1 &= \phi(W_1 x_0 + b_0) \\ s_i &= \phi(W_{i2} \cdot \phi(W_{i1} \cdot s_{i_1} + b_{i1}) + b_{i2}) + s_{i-1}, (2 \le i \le N-1) \\ s_N &= W_N \cdot s_{N-1} + b_N \end{aligned}$$

其中 s_i 是神经网络第 i 层的输出, $W_{i1}, W_{i2} \in R^{m \times m}, b_{i1}, b_{i2} \in R^m$ 为中间层的参数。N 为网络的深度，m 为网络的宽度，这两个参数限制神经网络的规模。ϕ 是激活函数，在我们的应用中，将取为

$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

这种取法是论文 [8] 的做法。网络结构示意图可以参见1。

2.3 Monta carlo 积分近似

由于积分表达式3无法解析求出，所以这里需要采用数值积分的办法。在这里我们采用了一般的蒙特卡洛积分法：

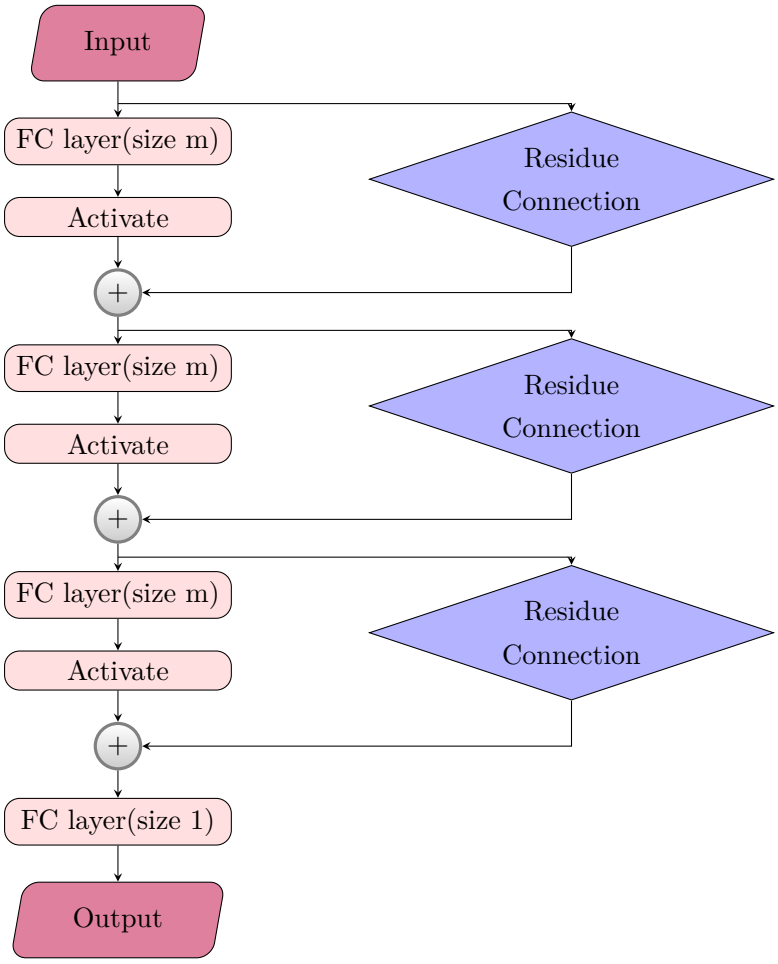
$$\int_\Omega f(x) \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

其中 $x_i \sim Uniform(\Omega)$. 采用这一技巧之后，若定义 $f(x,t) = u_t(x,t;\theta) + \mathcal{L}u(x,t;\theta)$ 我们的损失函数可以变为：

$$\begin{aligned} &MSE = MSE_f + MSE_b + MSE_0 \\ &= \frac{1}{N_f} \sum_{i=1}^{N_f} \| f(x_f^i, t_f^i)^2 \|^2 + \frac{1}{N_b} \sum_{i=1}^{N_b} \| u(x_b^i, t_b^i) - g(x_b^i, t_b^i) \|_2^2 + \frac{1}{N_0} \sum_{i=1}^{N_0} \| u(x_0^i, t_0^i) - u_0(x_0^i, t_0^i) \|_2^2 \end{aligned}$$

其中第一项在区域 $\Omega \times [0,T]$ 上均匀采样，第二项在边界上采样，第三项在初始时刻采样，实在训练中往往可以给三个项中做加权求和。

图 1: Network Structure of Deep Galerkin method



2.4 优化算法

这在深度学习中经常使用的优化算法是以 Adam 为典型的随机梯度下降算法，但是在求解 PDE 时，我们往往需要更高的计算精度。在实验中我发现使用 Adam 等梯度算法不能得到令人满意的解，能工作的优化器是 L-BFGS（有限内存的 BFGS 方法）。这是一种拟牛顿型的全梯度优化算法，也可以看成共轭梯度法的推广。更多的细节可以参考论文 [7], 在 pytorch 的 optim 类中实现了这一类的优化器，可以直接调用。

3 求解 Poisson 方程

2D Poisson 方程的 Dirichlet 边值问题,$\Omega = [-1, 1] \times [-1, 1]$

$$\begin{cases} -\Delta u = f(x), & x \in \Omega, \\ u = g(x), & x \in \partial\Omega, \end{cases} \tag{7}$$

采用 2.2 中提到的网络结构深度 $N = 4$, 宽度 $m = 20$, 损失函数在内部采样数为 $N_f = 1000$, 边界采样数 $N_b = 100$. 损失的调配参数 $\beta = 10$.

$$loss = MSE_f + \beta \cdot MSE_b$$

3.1 example 1

我们人工构造出一个在边界和区域内部变化都比较复杂的测试函数如下:

$$u_{real} = exp(x^2 + y^2) \cdot \sin(5x - 2y) + xy,\ f(x, y) = -\Delta u_{real},\ g(x, y) = u_{real}|_{\partial\Omega}(x, y)$$

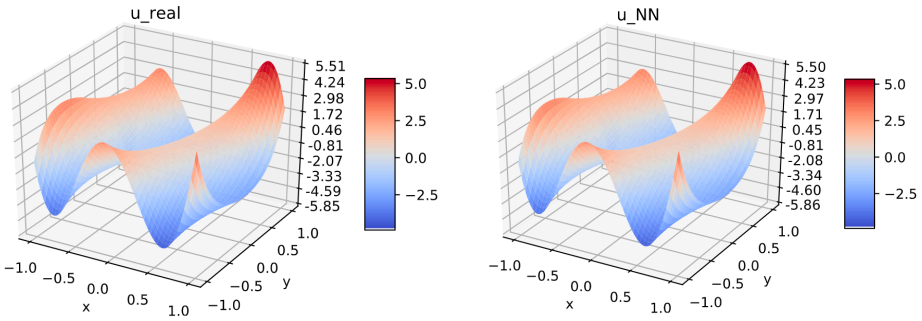


图 2: 泊松方程的解. 左边: 真解, 右边: 神经网络近似解

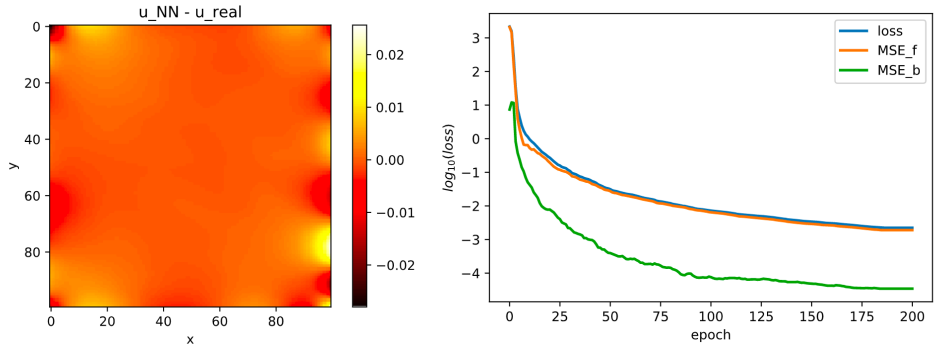


图 3: 左边: 误差图 $u_{real} - u_{NN}$ 右边: 损失函数下降曲线图

表 1: Relative Loss of Different β with Poisson Equation: error 定义见8

β	Relative Loss (%)
10	0.33
100	0.19
1000	0.48

$$error = \left(\int_\Omega \left(u^*\left(x; \theta^*\right) - u(x) \right)^2 \, \mathrm{d}x \right)^{\frac{1}{2}} \cdot \left(\int_\Omega (u(x))^2 \, \mathrm{d}x \right)^{-\frac{1}{2}} \tag{8}$$

从图2可以看到，我们得到的近似解确实和真解呈现出相同的图景，这意味着我们得到了正确的解。从3左边的误差图可以看出，拟合得不太好的地方主要分布在区域的边界附近。于是我增大了 β 到 100，使得边界项的损失具有更大的比重，发现结果确实变得更好（如表1所示）。但是当我继续增大 β 到 1000，结果反而变差。其原因可能是，

- 1. 这是一个内部误差与边界误差进行权衡的问题，过大的 β 使得内部误差项被忽略。
- 2. 过大的 β 使得问题变得奇异，难以优化。

下一小节我们便来解决这个问题：

3.2 Penalty free 的改进方案

这一小节要解决上一节中 β 的选取问题。我们采取有一种更加直接的方式避免这个调参问题：将我们的试验函数直接构造在满足边界条件的函数空间如下

$$u(x,\theta) = l(x)\cdot v(x,\theta) + \tilde{g}(x)$$

其中 $v(x,\theta)$ 是一个神经网络，$l(x)$ 是一个到边界的距离函数，比如在我们这个例子中可以取 $l(x,y) = (1-x^2)(1-y^2)$，而 $\tilde{g}(x)$ 是边界函数 $g(x)|_{\partial\Omega}$ 的延拓，. 但是问题在于如何得到 $g(x)$ 的延拓呢？如果我们直接让 $\tilde{g}(x) = u_{real}(x)$ 那么这当然存在作弊的嫌疑，因为我们是事先是不知道真解的。想法是使用另一个小型的神经网络 $h(x,\theta_1)$ 拟合 $\tilde{g}(x)$. 这样，我们的算法可以拆分为两个独立的优化问题，描述如下：

- 第一步: 优化边界损失函数

$$\theta_1 = \underset{\theta_1}{argmin}\, MSE_b(\theta_1) = \frac{1}{N_b}\sum_{i=1}^{N_b} \|h(x_i,\theta_1) - g(x_i)\|^2 \tag{9}$$

- 第二步：构建试验函数：

$$u(x,\theta) = l(x)\cdot v(x,\theta) + h(x,\theta_1) \tag{10}$$

- 第三步：优化内部损失函数

$$\theta = \underset{\theta}{argmin}\, MSE_f(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f} \|f(x_f^i,\theta)\|^2 \tag{11}$$

  其中 $f = \Delta u(x,\theta) + f$。

按照上面的方案进行求解，遗憾的是，并没有达到解得更好的结果，这是什么原因呢？将 $h(x,\theta_1)$ 的图像画出来 (如图4左) 可以知道 h 虽然拟合好了边界，但是内部延拓出来的解由于没有任何束缚而离真解非常远，甚至奇异，这毫无疑问大大提高了第二步优化的难度。所以我们应当对 h 内部加适当的约束。这个约束就不妨直接取为求解方程的内部 MSE，这样9就可以改进为12.

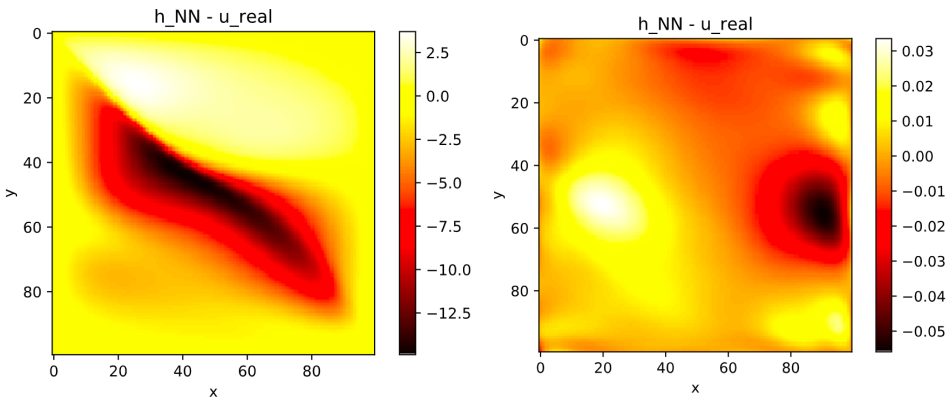


图 4: 左图：使用9的 h 与真解的误差（没有采用正则项），右图：使用12的 h 与真解的误差（采用了正则项）. 可以看到是用了正则项后的 h 与真解已经很接近了, 这会大大降低第二步优化的难度.

$$\theta_1 = \underset{\theta_1}{argmin}\, \beta MSE_b(\theta_1) + MSE_f(\theta_1) = \frac{\beta}{N_b}\sum_{i=1}^{N_b} \|h(x_i,\theta_1) - g(x_i)\|^2 + \frac{1}{N_f}\sum_{i=1}^{N_b} \|f(x_f^i,\theta_1)\|^2 \tag{12}$$

在式12中，第二项为正则项，这一项的存在使得 $h(x,\theta_1)$ 内部大致和真解相同（这样我们第二步优化11中的 $v(x,\theta)$ 将接近与 0 函数, 这是与神经网络参数零初始化相匹配的）。然后，我们取 $N_f = 100$（内部点）, $N_b = 1000$（边界点）, $\beta = 1000$，保证第一步优化精确地拟合边界函数。综合起来使用12,10,11, 并且设置参数如表2，最终的结果对比如表3。

表 2: 分步优化的参数设置

步骤	优化对象	网络宽度	网络深度	β	N_f(内部取样)	N_b（边界取样）
1	$h(x,\theta_1)$	20	3	1000	1000	100
2	$v(x,\theta)$	30	4	0	0	1000

表 3: 改进结果对比

项	Relative Loss (%)
原始形式 ($\beta = 100$)	0.19
改进分步优化	0.15

4 求解 Burgers 方程

$$\begin{cases} u_t + uu_x - \nu u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(0, x) = -\sin(\pi x) \\ u(t, -1) = u(t, 1) = 0 \end{cases} \tag{13}$$

这个方程的真解在文章 [1] 给出了:

$$u(x,t) = 4\pi\nu \left[\sum_{n=1}^{\infty} n a_n \mathrm{e}^{-n^2\pi^2 t\nu} \sin n\pi x \Big/ \left(a_o + 2\sum_{n=1}^{\infty} a_n \mathrm{e}^{-n^2\pi^2 t\nu} \cos n\pi x \right) \right]$$

在我们的求解算例中令 $\nu = 0.01/\pi$, 内部采样数 $N_f = 2000$, 边界采样数 $N_b = 100$, 初始条件采样数 $N_0 = 100$, 网络宽度 $m = 30$, 深度 $N = 5$. 求解结果如图5. 在表格4中展

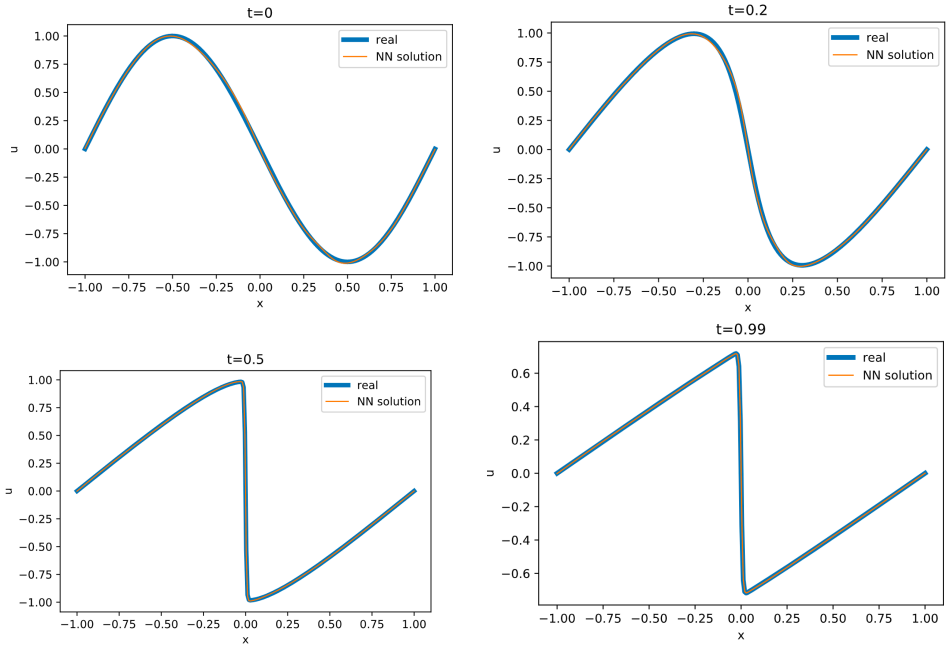


图 5: NN solution vs real solution, 相对 L_2 误差为 0.9%, 相对 L_2 误差定义见8

示了 $N_f = 1000$ 和 $N_f = 2000$ 的结果对比，发现提高内部采样数确实能够提高求解的精度。从图中结果也可以看出，DNN 求解含时非线性方程时能够捕捉到激波结构，某种程度上证明这种方案是有效的。当然，真实的情况还需要更多的计算结果来支持和验证。

表 4: 不同内部采样数的结果对比

N_f	Relative Loss (%)
1000	11
2000	0.91

5 结论和展望

在这篇报告中，我们使用 PINN[8] 的深度神经网络方法求解了 Poisson equation 和 Burgers euquation。验证了深度网络在求解 PDE（含时，不含时）的有效性。在求解 Poisson 方程时，我给出了一种 Penalty free 的改进方案，避免了参数调节，并且得到了比原始方案更好的结果。在求解 Burgers 方程时，成功用 DNN 的方法捕捉了激波结构，验证了该方案在含时方程中的有效性。

总结来说，深度网络求解 PDE 的优势在于 mesh-free, 而且依赖于神经网络强大的表达能力，能够拟合激波 (比如 burgers 方程中的 shock)。目前也已经有一系列的工作将这种方案应用于计算流体力学中，如 [5] 等，成功模拟了复杂的二维不可压流体的运动，由于时间所限制，报告的内容没有继续探索。DNN 方案的缺点在于没有良好的收敛性保证定理，这也是一个非常困难而且值得研究的方向。

6 附录

文中所有的实验代码都是亲自实现的，并且已经和报告一起提交。由于上次课堂报告时实验部分尚未完全完成，所以并没有展示自己的实验结果，显得内容有些空洞。但是这个项目其实花费了我不少的时间和心血，只是在探索过程中发现不少想法可能早已被一些文章所发表，不得已在课堂报告中呈现出一种” 综述 “的形式。希望这份期末报告能够让老师和助教看到我的一些思考和努力。

References

[1] Cea Basdevant, M Deville, P Haldenwang, JM Lacroix, J Ouazzani, R Peyret, Paolo Orlandi, and AT Patera. Spectral and finite difference solutions of the burgers equation. *Computers & fluids*, 14(1):23–41, 1986.

[2] Christian Beck, Arnulf Jentzen, et al. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *arXiv preprint arXiv:1709.05963*, 2017.

[3] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *arXiv preprint arXiv:1707.02568*, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, Feb 2021.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[7] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[8] 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.

[9] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, Dec 2018.

[10] E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[11] Bing Yu et al. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*, 2017.