GREEDY ALGORITHMS FOR SOLVING PDES

Jasen Lai
Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409, USA[*]
Department of Mathematics, The Ohio State University, Columbus, OH 43210, USA[†]
lai.474@osu.edu

Chunmei Wang
Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409, USA
Department of Mathematics, University of Florida, Gainesville, FL 32611, USA
chunmei.wang@ufl.edu

Haizhao Yang
Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA
haizhao@purdue.edu

ABSTRACT

High-dimensional PDEs are of high interest for their modeling capabilities; however, they are notoriously difficult to solve due to the "curse of dimensionality". Neural networks, which are believed to be able to overcome this problem, have been applied to solve PDEs giving rise to many different methods. Typical procedures use stochastic gradient descent for optimization, but in this paper, we train a shallow neural network using a greedy algorithm. Solving PDEs with greedy algorithms is a very recent development and is still a topic of active research. However, the goal of this paper is to provide and demonstrate an example implementation of greedy algorithms to solve PDEs.

Keywords Greedy algorithms · Linear PDEs · Nonlinear PDEs · Neural network · Optimization

AMS subject classifications 35D30 · 35G05 · 35G15 · 65D15

1 Introduction

Partial differential equations (PDEs) are prevalent in many fields, including, but not limited to, areas such as finance, science, and engineering [1, 2], because PDEs can be used to model and describe complex systems. However, in systems with many variables, PDEs become high-dimensional and much more difficult to solve due to the "curse of dimensionality". To combat this, there has been much research aimed at developing neural network based methods to solve PDEs [3, 4] because neural networks have the ability to overcome the curse of dimensionality [5].

While deep neural networks have proven to be good solvers of PDEs empirically, the theoretical approximation, generalization, and optimization of these methods have been difficult to analyze holistically [6]. However, recently, a method using the greedy algorithm to train shallow neural networks, instead of the traditional method of stochastic gradient descent (SGD) for deep neural networks, has made it possible to complete an analysis of the aforementioned areas. Additionally, this greedy algorithm has been proven to be practically feasible and has successfully demonstrated that the theoretical convergence order is attained experimentally [7].

The main goal of this paper is to provide and demonstrate an example implementation of greedy algorithms to solve PDEs. In section 2, we describe the formulation of the loss function with linear PDEs. In section 3, we define the

[*]REU Program Participant at Texas Tech
[†]Undergraduate Student at The Ohio State University

greedy algorithm and how to apply it to a PDE. And finally, in section 4, we give two examples of applying the greedy algorithm to solve PDEs.

2 Model problem

To begin, we describe a general problem with linear PDEs of order $2m$ using the notation in [6, 7, 8].

$$\begin{cases} Lu & = & f \quad \text{in} \quad \Omega, \\ B_N^k(u) & = & 0 \quad \text{on} \quad \partial\Omega \quad (0 \leq k \leq m-1), \end{cases} \tag{2.1}$$

where $B_N^k(u)$ denotes the Neumann boundary conditions and $\Omega \subset \mathbb{R}^d$ is a bounded domain with a sufficiently smooth boundary $\partial\Omega$. L is the partial differential operator defined as:

$$Lu = \sum_{|\alpha|=m} (-1)^m \partial^\alpha (a_\alpha(x)\partial^\alpha u) + a_0(x)u, \tag{2.2}$$

where α denotes a n-dimensional multi-index $\alpha = (\alpha_1, \cdots, \alpha_n)$ with

$$|\alpha| = \sum_{i=1}^n \alpha_i, \quad \partial^\alpha = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}. \tag{2.3}$$

We also assume that a_α are strictly positive and smooth functions on Ω for $|\alpha| = m$ and $\alpha = 0$, namely, $\exists \alpha_0 > 0, \alpha_1 < \infty$, such that

$$\alpha_0 \leq a_\alpha(x), a_0(x) \leq \alpha_1 \forall x \in \Omega, |\alpha| = m. \tag{2.4}$$

Additionally, for a given non-negative integer k, let

$$\mathcal{H}^k(\Omega) := \{v \in L^2(\Omega); \partial^\alpha v \in L^2(\Omega), |\alpha| \leq k\} \tag{2.5}$$

be standard Sobolev spaces with norm and semi-norm given respectively by

$$\|v\|_k := \left(\sum_{|\alpha|\leq k} \|\partial^\alpha v\|_0^2 \right)^{1/2}, \quad |v|_k := \left(\sum_{|\alpha|=k} \|\partial^\alpha v\|_0^2 \right)^{1/2}. \tag{2.6}$$

In $\|\cdot\|_0$, the subscript 0 denotes $\mathcal{H}^0(\Omega)$, which was previously defined as a subset of $L^2(\Omega)$ space. Sobolev spaces allows us to define an inner product for functions and its derivatives up to an order k, where we denote the inner product on the Ω and $\partial\Omega$ as

$$\langle u, v \rangle_\Omega = \int_\Omega u(x)v(x)dx, \quad \langle u, v \rangle_{\partial\Omega} = \int_{\partial\Omega} u(x)v(x)dx. \tag{2.7}$$

For Dirichlet boundary conditions, we replace B_N^k with B_D^k where it is defined as the directional derivative

$$B_D^k(u) := \left. \frac{\partial^k u}{\partial v^k} \right|_{\partial\Omega} \quad (0 \leq k \leq m-1), \tag{2.8}$$

with v being the outward unit normal vector on $\partial\Omega$. When $k = 0$, $B_D^k(u)$ is simply defined as u on the boundary.

The loss function for Dirichlet conditions is more difficult to construct compared to Neumann conditions, so we start with the loss in the pure Neumann case. Additionally, the loss is based of the PDE's weak formulation where we move all the terms to the left hand side and add the coefficient $\frac{1}{2}$ for reasons made clear in equation 3.4. We define the loss function as

$$J(u) = \frac{1}{2}a(u, u) - \int_\Omega fu dx, \tag{2.9}$$

where the function u is the input and

$$a(u, v) := \sum_{|\alpha|=m} \langle a_\alpha \partial^\alpha u, \partial^\alpha v \rangle_\Omega + \langle a_0 u, v \rangle_\Omega. \tag{2.10}$$

To handle Dirichlet conditions, we consider the continuous loss function

$$J_\delta(u) = \frac{1}{2}a_\delta(u, u) - \int_\Omega fu dx, \tag{2.11}$$

where we augment $a(u,v)$ to handle Dirchlet conditions by

$$a(u,v)_\delta := a(u,v) + \delta^{-1}\sum_{k=0}^{m-1}\langle B_D^k(u), B_D^k(v)\rangle_{\partial\Omega}. \tag{2.12}$$

Here, δ^{-1} acts as the weight for the boundary condition which can be set as a hyper-parameter.

We have defined the loss functions in a continuous sense. However, in practical application, only a finite number of points can be sampled from the domain, so we must discretize the loss with

$$J(u) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{|\alpha|=m} a_\alpha(x_i)(\partial^\alpha u(x_i))^2 + \frac{1}{2N}\sum_{i=1}^{N} a_0(x_i)u(x_i)^2 - \frac{1}{N}\sum_{i=1}^{N} f(x_i)u(x_i) \tag{2.13}$$

for pure Nuemann boundary conditions, and

$$J_\delta(u) = J(u) + \frac{\delta^{-1}}{2N_0}\sum_{i=1}^{N_0}\sum_{k=0}^{m-1}\left(\frac{\partial^k}{\partial v^k}u(y_i)\right)^2 \tag{2.14}$$

for Dirichlet boundary conditions. Here, we sample $x_1,...,x_N \in \Omega$ and $y_1,...,y_{N_0} \in \partial\Omega$ uniformly at random.

3 Greedy algorithm

Now that we have established a loss function to minimize, we can start explaining greedy algorithms. The ultimate goal is to construct our solution u as a finite linear combination of functions g,

$$u_n = \sum_{i=1}^{n} a_i g_i, \tag{3.1}$$

where $g_i \in \mathbb{D}$. Here, $\mathbb{D}$ is called a dictionary, and represents a family of functions. While there are many potential choices of $\mathbb{D}$, research indicates that different choices of $\mathbb{D}$ will have different convergence properties [6, 9, 10]. In this paper, we use the bounded dictionary

$$\mathbb{D} = \{\pm\sigma(xw+b) : w \in \mathbb{R}^d, b \in \mathbb{R}\}, \tag{3.2}$$

where w and b are bounded over some finite interval. $\mathbb{D}$ is analogous to activation functions used in neural networks and the structure of u_n can be interpreted as a single layer neural network. The greedy algorithm has several versions [11, 12], but in this paper, we use the relaxed greedy algorithm (RGA). In RGA, we recursively define u_k as

$$u_0 = 0, \quad g_k = \underset{g\in\mathbb{D}}{\operatorname{argmax}}\langle\nabla J(u_{k-1}), g\rangle, \quad u_k = (1-\alpha_k)u_{k-1} - M\alpha_k g_k \tag{3.3}$$

where $\alpha_k = \min(1, \frac{2}{k})$ "relaxes" new additions of g and M is a regularization parameter that controls the magnitude of g. Here, $\langle\nabla J(u_{k-1}), g\rangle$ is the functional derivative of $J(u_{k-1})$ with respect to u where we treat g as the test function. If $J(u)$ is defined the same as equation 2.14, then

$$\begin{aligned}\langle\nabla J(u_{k-1}), g\rangle &= \lim_{t\to 0}\frac{J(u+tg)-J(u)}{t} \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{|\alpha|=m} a_\alpha(x_i)(\partial^\alpha u(x_i))(\partial^\alpha g(x_i)) + \frac{1}{N}\sum_{i=1}^{N} a_0(x_i)u(x_i)g(x_i) - \frac{1}{N}\sum_{i=1}^{N} f(x_i)g(x_i) \\ &\quad + \frac{\delta^{-1}}{N_0}\sum_{i=1}^{N_0}\sum_{k=0}^{m-1}\left(\frac{\partial^k}{\partial v^k}u(y_i)\right)\left(\frac{\partial^k}{\partial v^k}g(y_i)\right).\end{aligned} \tag{3.4}$$

The final equation here no longer has the coefficient $\frac{1}{2}$ mentioned in equation 2.9 due to cancellation with other coefficients that arise when calculating the functional derivative.

The sub-optimization problem of finding the g that maximizes $\langle\nabla J(u_{k-1}), g\rangle$ has many potential solutions. One method is to first sample n random guesses of w and b, then use the best guess as the initial guess in Newton's method. Since w and b are bounded over a finite interval, we must use a bounded minimization method such as the BFGS algorithm.

In this paper, the greedy algorithm has only been shown for linear PDEs; however, we can generalize it to handle non-linear PDEs. First, we derive the PDE's weak formulation. Second, we construct the loss function by moving all of terms in the weak formulation to the left hand side and add cancellation coefficients. And finally, we calculate the functional derivative of the loss. Once we have the formula for the functional derivative, we can solve for the maximum each iteration of training and incrementally add g to our approximation u.

4 Examples

In this section, we provide example applications of the relaxed greedy algorithm (RGA). While RGA will generally step towards the minimum of the loss, a lower loss does not necessarily bring better accuracy. For the results below, we adjusted the number of nodes in the final approximation to get a better accuracy. Additionally, for both example 4.1 and 4.2, we use the following dictionary:

$$\mathbb{D} = \{\pm\sigma(xw+b) : w \in \mathbb{R}^d, b \in \mathbb{R}\}, \quad \text{with } w \text{ and } b \in [-30, 30] \tag{4.1}$$

4.1 Example 1

Consider the following second-order linear elliptic equation with Neumann boundary conditions in one dimension:

$$\begin{aligned} &-\frac{d}{dx}((1+x^2)\frac{du}{dx}) + x^2u(x) = f(x) \quad \text{for} \quad 0 < x < 1, \\ &u'(0) = u'(1) = 0 \end{aligned} \tag{4.2}$$

where the true solution is $u(x) = cos(2\pi x)$ and

$$f(x) = x^2cos(2\pi x) + 4\pi^2(x^2+1)cos(2\pi x) + 4\pi x sin(2\pi x). \tag{4.3}$$

Here, the loss function we use is the same as equation 2.13 and the corresponding functional derivative is equation 3.4 but without the boundary term. For this problem, we use the following hyper-parameters and plot the true solution and the greedy approximation side-by-side:

Hyper-parameters	
Number of nodes	1104
# of points sampled from Ω	2048
M	25
# of guesses before Newton's method	2048
Results	
Mean absolute error	0.01027

Table 1: Example 1 hyper-parameters and results

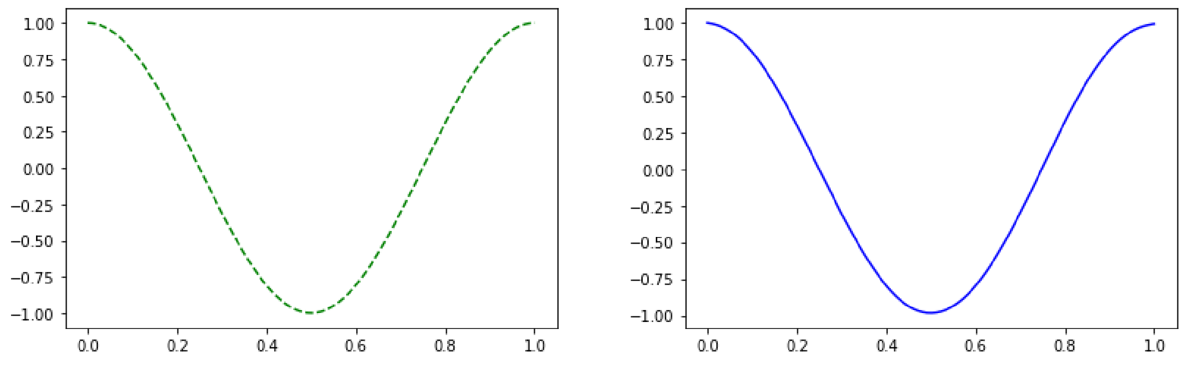

(a) True solution (b) Greedy approximation

Figure 4.1: Example 1 comparison

4.2 Example 2

Consider the following second-order semi-linear elliptic equation with Dirichlet boundary conditions in two dimensions:

$$\begin{aligned} &-\triangle u - (u(x,y)-1)^3 + (u(x,y)+2)^2 = f(x,y) \quad \text{for} \quad (x,y) \in (0,1) \times (0,1), \\ &u = 0 \quad \text{for} \quad x = (0 \text{ or } 1) \quad \text{or} \quad y = (0 \text{ or } 1) \end{aligned} \tag{4.4}$$

where the true solution is $u(x) = sin(2\pi x)sin(2\pi y)$ and

$$f(x) = 8\pi^2 sin(2\pi x)sin(2\pi y) - (sin(2\pi x)sin(2\pi y) - 1)^3 + (sin(2\pi x)sin(2\pi y) + 2)^2. \tag{4.5}$$

Here, the loss function we use is

$$J(u) = \frac{1}{2}\int_\Omega (\nabla u)^2 - \frac{1}{4}u^4 + \frac{4}{3}u^3 + \frac{1}{2}u^2 + 5udx - \int_\Omega fudx + \frac{\lambda}{2}\int_{\partial\Omega} u^2 dx \tag{4.6}$$

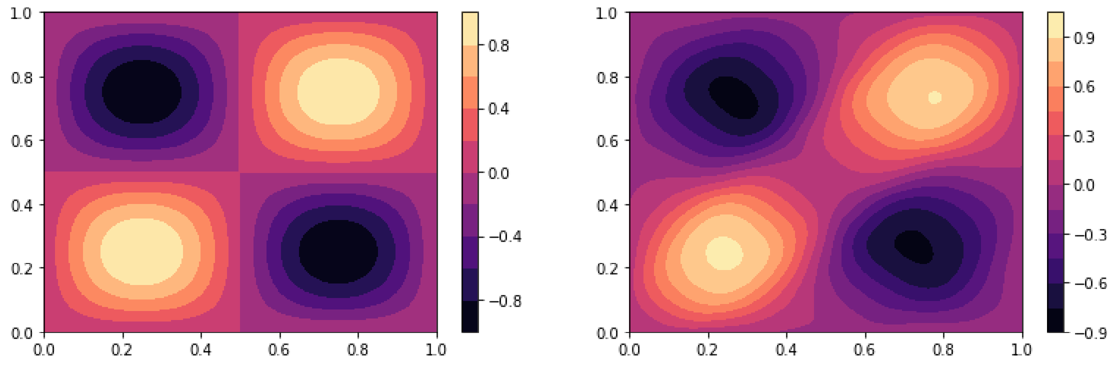
where we added the cancellation coefficients to each term of the weak formulation. Then, we get our corresponding discretized functional derivative as

$$\langle \nabla J(u_{k-1}), g\rangle = \frac{1}{N}\sum_{i=1}^{N} \nabla u(x_i)\nabla g(x_i) + \frac{1}{N}\sum_{i=1}^{N} g(x_i)(-(u(x_i)-1)^3 + (u(x_i)+2)^2) - \frac{1}{N}\sum_{i=1}^{N} f(x_i)g(x_i) + \lambda\frac{1}{N_0}\sum_{i=1}^{N_0} u(y_i)g(y_i) \tag{4.7}$$

For this problem, we use the following hyper-parameters and plot the true solution and the greedy approximation side-by-side:

Hyper-parameters	
Number of nodes	818
# of points sampled from Ω	512
Boundary weight	200
M	25
# of guesses before Newton's method	2048
Results	
Mean absolute error	0.07496

Table 2: Example 2 hyper-parameters and results



(a) True solution

(b) Greedy approximation

Figure 4.2: Example 2 comparison

5 Acknowledgements

This work is supported by the National Science Foundation under Grant No. DMS-2050133 (REU Site: Mathematical, Statistical, and Computational Methods in the Life Sciences). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

J.L. is grateful to his mentors, Dr. Chunmei Wang and Dr. Haizhao Yang, for providing guidance. Additionally, J.L. is thankful for Dr. Qingguo Hong, Dr. Jonathan Siegel, and Xianlin Jin for helpful discussions. J.L. also thanks everyone a part of the REU at Texas Tech for a wonderful experience.

Bibliography

[1] Asmat Ara, Najeeb Alam Khan, Oyoon Abdul Razzaq, Tooba Hameed, and Muhammad Asif Raja. Wavelets optimization method for evaluation of fractional partial differential equations: an application to financial modelling. *Advances in Difference Equations*, 2018(1), 2018. doi: 10.1186/s13662-017-1461-2.

[2] Leon Lapidus and George Francis Pinder. *Numerical Solution of Partial Differential Equations in Science and Engineering*. wiley, 2011.

[3] Jianguo Huang, Haoqin Wang, and Haizhao Yang. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, Oct 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109675. URL http://dx.doi.org/10.1016/j.jcp.2020.109675.

[4] Yiqi Gu, Haizhao Yang, and Chao Zhou. Selectnet: Self-paced learning for high-dimensional partial differential equations. *Journal of Computational Physics*, 441:110444, Sep 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110444. URL http://dx.doi.org/10.1016/j.jcp.2021.110444.

[5] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations, 2018.

[6] Qingguo Hong, Jonathan W. Siegel, and Jinchao Xu. A priori analysis of stable neural network solutions to numerical pdes, 2021.

[7] Wenrui Hao, Xianlin Jin, Jonathan W Siegel, and Jinchao Xu. An efficient greedy training algorithm for neural networks and applications in pdes, 2021.

[8] Jinchao Xu. Finite neuron method and convergence analysis. *Communications in Computational Physics*, 28 (5):1707–1745, Jun 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2020-0191. URL http://dx.doi.org/10.4208/cicp.OA-2020-0191.

[9] Jonathan W. Siegel and Jinchao Xu. Improved Approximation Properties of Dictionaries and Applications to Neural Networks. *arXiv e-prints*, art. arXiv:2101.12365, January 2021.

[10] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory*, 39:930–945, 1993.

[11] Andrew R. Barron, Albert Cohen, Wolfgang Dahmen, and Ronald A. DeVore. Approximation and learning by greedy algorithms. *The Annals of Statistics*, 36(1):64 – 94, 2008. doi: 10.1214/009053607000000631. URL https://doi.org/10.1214/009053607000000631.

[12] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29 (5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.