Greedy Algorithms for solving PDEs

Jasen Lai (presenter) [1] Chunmei Wang [2] Haizhao Yang [3]

[1]The Ohio State University [2]University of Florida [3]Purdue University

Introduction

The greedy algorithm we present is a new method for training a neural network to solve PDEs. High-dimensional PDEs are of particular interest because of their modeling capabilities, but they are notoriously difficult to solve. While traditional methods struggle in high dimensions, neural networks, which are believed to overcome the "curse of dimensionality", have demonstrated empirical success in this area. Classically, the greedy algorithm is a technique for approximating functions; however, recent research has shown that it can also be applied to training neural networks. Even more recently, an error analysis was done and the greedy algorithm has been shown to be practically feasible for solving PDEs. In this poster, we present an example implementation of the greedy algorithm and give sample results.

Model problem

We first describe a general problem with linear PDEs of order $2m$.

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

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:

$$Lu = \sum_{|\alpha|=m} (-1)^m \partial^\alpha (a_\alpha(x)\partial^\alpha u) + a_0(x)u, \tag{2}$$

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

$$|\alpha| = \sum_{i=1}^n \alpha_i, \quad \partial^\alpha = \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}. \tag{3}$$

We denote the inner product on the Ω and $\partial\Omega$ as

$$\langle u, v\rangle_\Omega = \int_\Omega u(x)v(x)dx, \quad \langle u, v\rangle_{\partial\Omega} = \int_{\partial\Omega} u(x)v(x)dx. \tag{4}$$

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

$$B_D^k(u)\frac{\partial^k u}{\partial v^k}\bigg|_{\partial\Omega} \quad (0 \le k \le m-1), \tag{5}$$

Loss function

We define the loss function as

$$J(u) = \frac{1}{2N}\sum_{i=1}^N \sum_{|\alpha|=m} a_\alpha(x_i)(\partial^\alpha u(x_i))^2 + \frac{1}{2N}\sum_{i=1}^N a_0(x_i)u(x_i)^2 - \frac{1}{N}\sum_{i=1}^N f(x_i)u(x_i) \tag{6}$$

for pure Nuemann boundary conditions, and

$$J_\delta(u) = J(u) + \frac{\delta^{-1}}{2N_0}\sum_{i=1}^{N_0}\sum_{k=0}^{m-1}\left(\frac{\partial^k}{\partial v^k}u(y_i)\right)^2 \tag{7}$$

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.

Greedy algorithm

The ultimate goal is to construct our solution u as a finite linear combination of functions g,

$$u_n = \sum_{i=1}^n a_i g_i, \tag{8}$$

where $g_i \in \mathbb{D}$. Here, $\mathbb{D}$ is called a dictionary, and can represent any family of functions. For the examples, we use

$$\mathbb{D} = \{\pm\sigma(xw + b) : w \in \mathbb{R}^d, b \in \mathbb{R}\}, \tag{9}$$

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. There are several versions of the greedy algorithm, but we use the relaxed greedy algorithm (RGA). In RGA, we recursively define u_k as

$$u_0 = 0, \quad g_k = \underset{g\in\mathbb{D}}{\operatorname{argmax}}\langle \nabla J(u_{k-1}), g\rangle, \quad u_k = (1-\alpha_k)u_{k-1} - M\alpha_k g_k \tag{10}$$

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.

Functional derivative

If $J(u)$ is defined the same as equation 7, then

$$\begin{aligned} \langle \nabla J(u_{k-1}), g\rangle &= \lim_{t\to 0}\frac{J(u+tg) - J(u)}{t} \\ &= \frac{1}{N}\sum_{i=1}^N \sum_{|\alpha|=m} a_\alpha(x_i)(\partial^\alpha u(x_i))(\partial^\alpha g(x_i)) + \frac{1}{N}\sum_{i=1}^N a_0(x_i)u(x_i)g(x_i) \\ &- \frac{1}{N}\sum_{i=1}^N f(x_i)g(x_i) + \frac{\delta^{-1}}{N_0}\sum_{i=1}^{N_0}\sum_{k=0}^{m-1}\left(\frac{\partial^k}{\partial v^k}u(y_i)\right)\left(\frac{\partial^k}{\partial v^k}g(y_i)\right). \end{aligned} \tag{11}$$

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.

Generalization

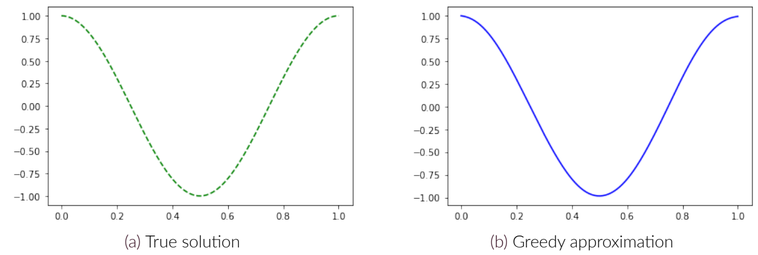
The greedy algorithm has only been shown for linear PDEs; however, we can generalize it to handle non-linear PDEs. First, we derive the PDE's weak formulation. Second, we construct the loss function by moving all of terms in the weak formulation to the left hand side and add cancellation coefficients. 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.

Example 1

The following is a second-order linear elliptic equation with Neumann boundary conditions in one dimension:

$$\begin{aligned} &-\frac{d}{dx}((1+x^2)\frac{du}{dx}) + x^2u(x) = f(x) \quad \text{for} \quad 0 < x < 1, \\ &u'(0) = u'(1) = 0 \end{aligned} \tag{12}$$

where the true solution is $u(x) = cos(2\pi x)$ and $f(x)$ is chosen to satisfy the true solution. In the results below, we obtain a mean absolute error of 0.01027.



(a) True solution (b) Greedy approximation

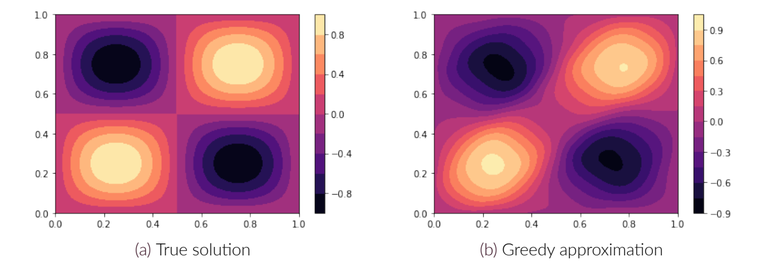
Figure 1. Example 1 comparison

Example 2

The following is a second-order semi-linear elliptic equation with Dirichlet boundary conditions in two dimensions:

$$\begin{aligned} &-\triangle u - (u(x,y)-1)^3 + (u(x,y)+2)^2 = f(x,y) \quad \text{for} \quad (x,y) \in (0,1)\times(0,1), \\ &u = 0 \quad \text{for} \quad x = (0 \text{ or } 1) \quad \text{or} \quad y = (0 \text{ or } 1) \end{aligned} \tag{13}$$

where the true solution is $u(x) = sin(2\pi x)sin(2\pi y)$ and f(x,y) is chosen to satisfy the true solution. In the results below, we obtain a mean absolute error of 0.07496.



(a) True solution (b) Greedy approximation

Figure 2. Example 2 comparison

References

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

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

[3] Jinchao Xu. Finite neuron method and convergence analysis. *Communications in Computational Physics*, 28(5):1707–1745, Jun 2020.