# A Structure-Guided Gauss-Newton Method for Shallow ReLU Neural Network

Zhiqiang Cai, Tong Ding, Min Liu, Xinyu Liu, Jianlin Xia

Tong Ding

Purdue University

Precond24

# Motivation and Background

# A difficult problem to tackle

Consider a one-dimensional piece-wise continuous function with **unknown** discontinuity as below.

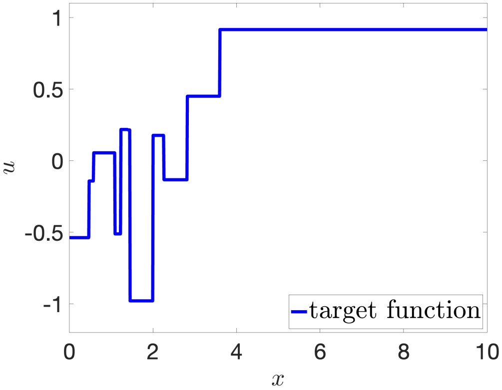
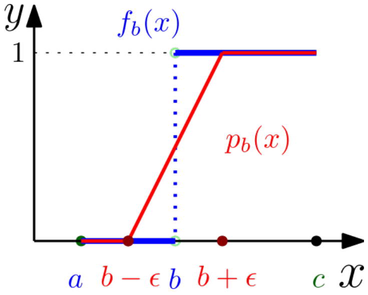



Figure: 10 jumps piecewise continuous function

Unit step function $f_b(x)$ and its continuous piecewise linear approximation $p_b(x)$:

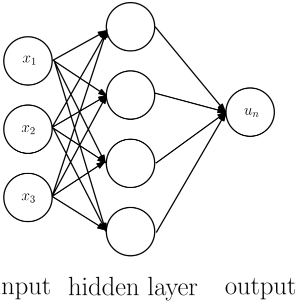
$$f_b(x) = \begin{cases} 0, & a < x < b \\ 1, & b < x < c \end{cases} \quad \text{and } p_b(x) = \begin{cases} 0 & a < x \le b - \epsilon \\ \frac{x-b+\epsilon}{2\epsilon} & b - \epsilon < x \le b + \epsilon \\ 0 & b + \epsilon < x \le c \end{cases}$$



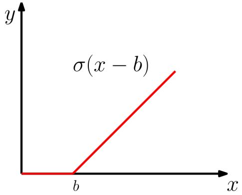

$$\|f_b - p_b\|_{L^\infty(I)} = \frac{1}{2}; \quad \|f_b - p_b\|_{L^r(I)} = \frac{\epsilon^{1/r}}{2^{1-1/r}(1+r)^{1/r}}$$

How to compute $p_b(x)$ when $b$ is **unknown**?

# Least square in Machine Learning




(a) One hidden layer neural network




(b) $\sigma(x-b)$

Let real-valued function $u$ be the target we want to approximate.

$$u_n = \sum_{i=1}^{n} c_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i) + c_0, \quad \mathcal{J}_n = \min_{u_n} \frac{1}{2} \|u - u_n\|^2 \tag{1}$$

# Gauss Newton Method with ReLU Bases

- Gauss Newton(GN) algorithm is a 'quasi-Newton method'for least squares only
- Jacobian matrix is used to approximate Hessian.

Suppose $u$ is a nonlinear, twice continuously differentiable function.

$$\nabla \mathcal{J} = \int_\Omega \nabla \left(u - u_n\right)^T \left(u - u_n\right) \tag{2}$$

$$\nabla^2 \mathcal{J} \approx \int_\Omega \nabla \left(u - u_n\right)^T \nabla \left(u - u_n\right) \tag{3}$$

Denote $J = \nabla \left(u - u_n\right)$, $r = u - u_n$, then the update is

$$u_n^{(k+1)} = u_n^{(k)} + \gamma^{(k+1)} \left(\int_\Omega J^T J\right)^{-1} \int_\Omega \left(J^T r^{(k)}\right) \tag{4}$$

# Structure-Guided Gauss-Newton method

**SgGN key ideas**:

1. Use specific basis functions for GN minimization: ReLU
   - Guaranteed **positive definiteness** of approximated hessian
2. $\sigma(x-b)-\sigma\left(x-(b+\epsilon)\right)$ captures the jump at $b$.
   - No more mesh refinements
   - No overshooting
3. Consider $b_i$ in $\sigma(x-b_i)$ as the moving mesh

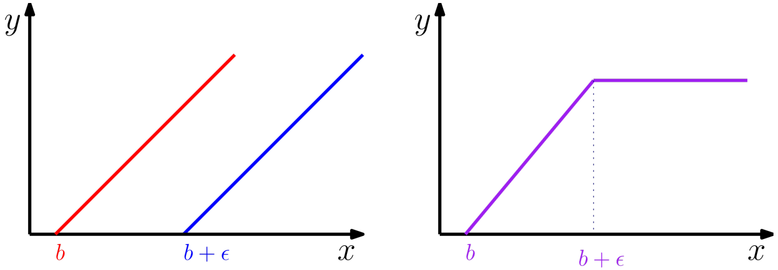
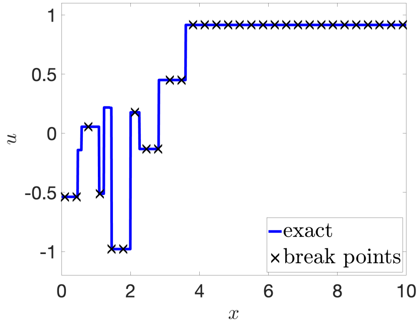



Figure: Two neurons

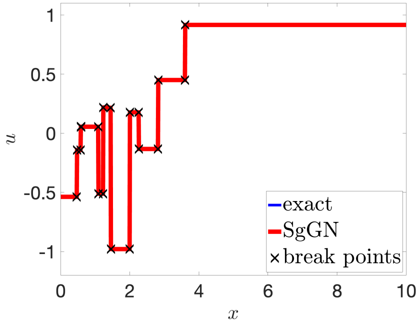
# Our result: SgGN method

30 neurons and 1000 training data.

1. Mean square error (MSE) is 8.76E-4 after 8 iterations.
2. $b_i$ is called breaking point




(a) Initial breaking points $\{b_i\}$




(b) Moved breaking points

# Algorithm & Mathematical Framework

# Problem Setup

Denote $\sigma_i = \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$, then the basis functions are:

$$\mathcal{M}_n(\Omega) = \left\{ \sum_{i=1}^{n} c_i \sigma_i(\mathbf{x}) + c_0 : \mathbf{x} \in \Omega, c_i \in \mathbb{R}, b_i \in \mathbb{R}_0^+, \mathbf{w}_i \in \mathbb{S}^{d-1} \right\}$$

The the loss function we want to minimize is:

$$\mathcal{J}(u_n) = \int_\Omega (u(\mathbf{x}) - u_n(\mathbf{x}))^2 \tag{5}$$

1. Linear parameters: **c**

$$\nabla_{\mathbf{c}} \mathcal{J}(u_n) = \mathbf{0} \tag{6}$$

2. Nonlinear parameters: $\mathbf{r}_i = (\mathbf{w}_i, b_i)$

$$\nabla_{\mathbf{r}} \mathcal{J}(u_n) = \mathbf{0} \tag{7}$$

# Linear Parameters

Equation (6) is reduced to solve a linear equation:

$$\mathcal{A}\mathbf{c} = \mathbf{F}; \quad \text{with } \mathcal{A}_{ij} = \int_\Omega \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)\sigma(\mathbf{w}_j \cdot \mathbf{x} + b_j) \tag{8}$$

with $\mathbf{F}_i = \int_\Omega f(\mathbf{x})\sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$

# Nonlinear Parameters: Gauss Newton

Use GN to solve Equation (7).

$$\nabla_{\mathbf{r}} \left( \nabla_{\mathbf{r}} \mathcal{J}_p(u_n) \right)^T \approx \hat{H} = \left( D(c) \otimes I_{d+1} \right) \mathcal{H}(\mathbf{r}) \left( D(c) \otimes I_{d+1} \right) \tag{9}$$

$$\mathcal{H}(\mathbf{r}) = \int_{\Omega} \left[ \mathbf{H}\mathbf{H}^T \right] \otimes \left[ \mathbf{y}\mathbf{y}^T \right] \tag{10}$$

It is easy to extend our analysis to the discrete least square problems.

# Summary: A block Gauss Seidel

$$\begin{pmatrix} \mathcal{A}\left(\mathbf{r}^{(k)}\right) & \\ & \hat{H}\left(\mathbf{c}^{(k+1)}\right) \end{pmatrix} \begin{pmatrix} \mathbf{c}^{(k+1)} \\ \mathbf{p}^{(k)} \end{pmatrix} = \begin{pmatrix} \mathbf{F}(\mathbf{r}^{(k)}) \\ -\nabla_{\mathbf{r}^{(k)}} \mathcal{J}\left(\mathbf{c}^{(k+1)}\right) \end{pmatrix} \tag{11}$$

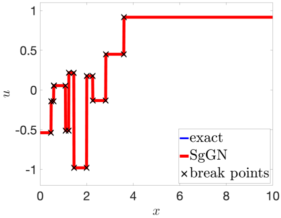
with

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \gamma^{(k+1)} \mathbf{p}^{(k)} \tag{12}$$

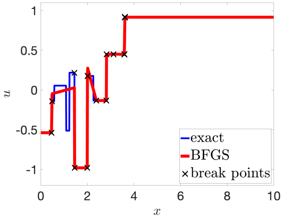
1. $\mathcal{A}$ and $\hat{H}$ are **symmetric positive definite** under mild assumptions.
2. $\mathcal{A}$ and $\hat{H}$ are highly **ill-conditioned**
   - $\mathrm{cond}(A^{(0)}) \geq O(n^4)$ in $1D$
   - More matrix analysis tomorrow 10:15-10:40 Room 1116
3. The matrix inversions are done by truncated svd consequently.

# Numerical Results

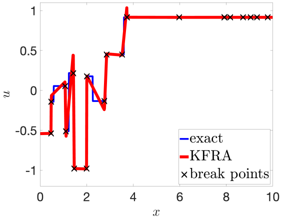
# 1D test continued



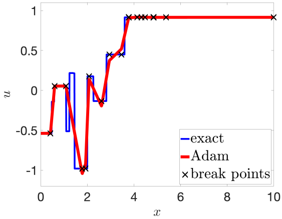

(a) SgGN(Ours)




(b) BFGS



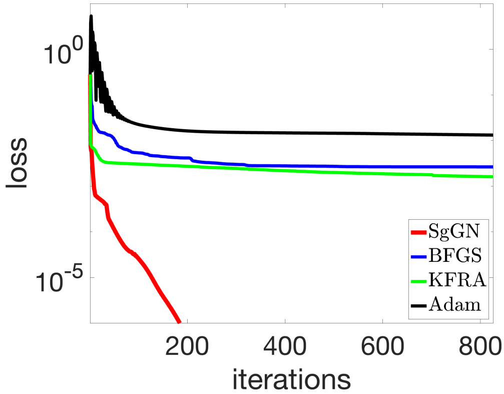

(c) KFRA



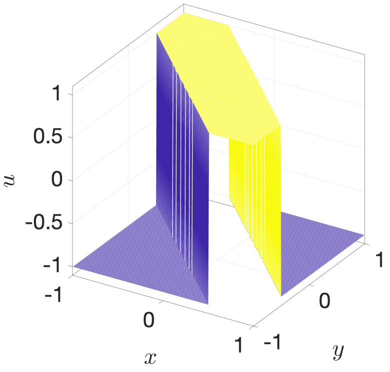

(d) Adam

$$\mathcal{J}_{30} = \frac{1}{1000}\sum_{i=1}^{1000}\left(f(x_i) - u_n(x_i)\right)^2$$

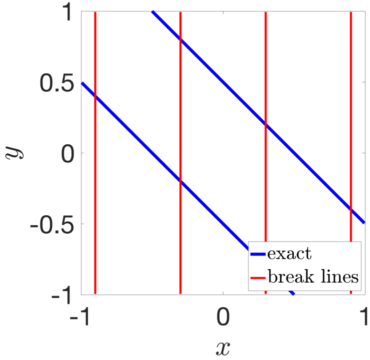
| Method | **SgGN** | BFGS | KFRA | Adam |
|---|---|---|---|---|
| Iteration | 825 | 825 | 825 | 10,000 |
| $\mathcal{J}_{30}$ | **6.56E-9** | 2.65E-3 | 1.61E-3 | 8.14E-3 |



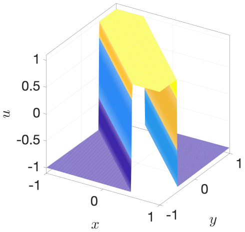
# 2D jump function: 4 neurons



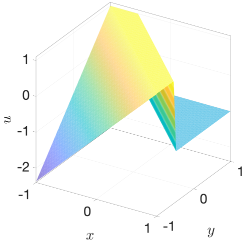

(a) Target function



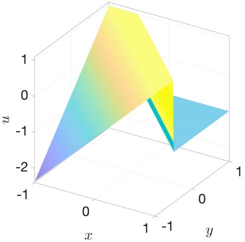

(b) $\{\mathbf{r}_i = (\mathbf{w}_i, b_i)\}$ initialization



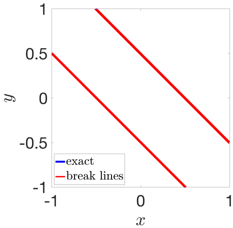

(a) SgGN(Ours)



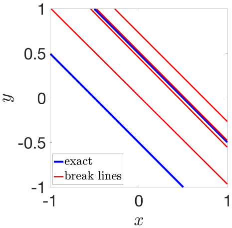

(b) KFRA



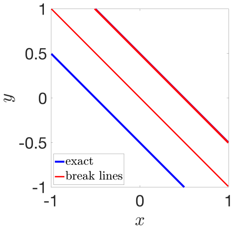

(c) BFGS




(d) SgGN



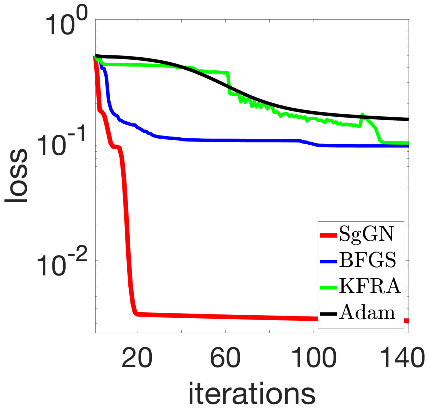

(e) KFRA



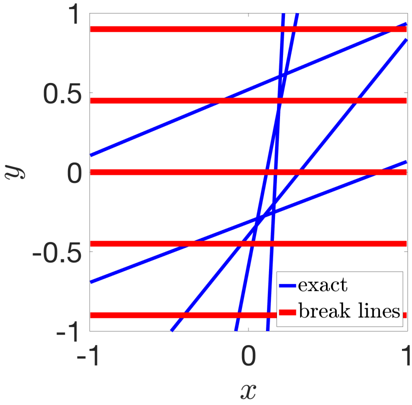

(f) BFGS

4 neurons with $200^2$ training data.

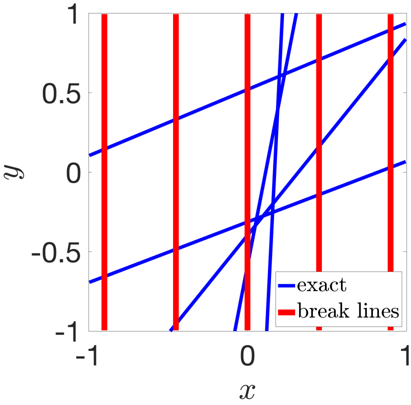
| Method | **SgGN** | BFGS | KFRA | Adam |
|---|---|---|---|---|
| Iteration | 142 | 142 | 142 | 10,000 |
| $\mathcal{J}_4$ | **3.16E-3** | 8.92E-2 | 9.40E-2 | 9.23E-2 |



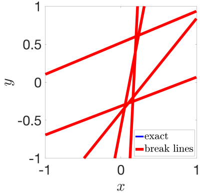
# Different Initialization for Line Approximation



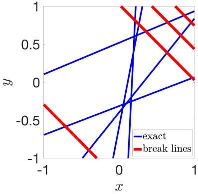

(a) Horizontal initialization (HI)



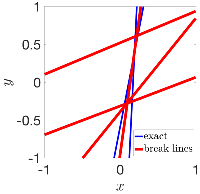

(b) Vertical initialization (VI)



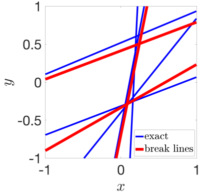

(a) BFGS (HI)



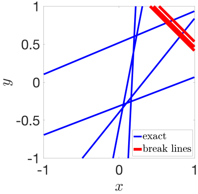

(b) KFRA (HI)



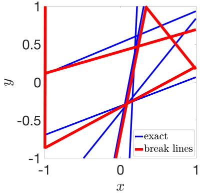

(c) Adam (HI)




(d) BFGS (VI)

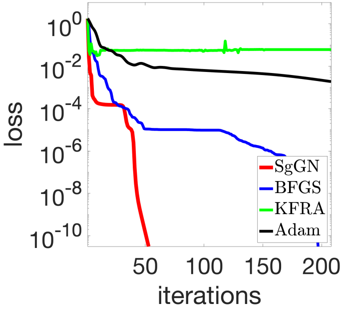



(e) KFRA (VI)

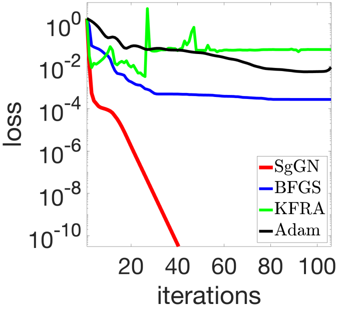



(f) Adam (VI)

| Method | **SgGN** | BFGS | KFRA | Adam |
|---|---|---|---|---|
| Iteration | 207 | 207 | 207 | 10,000 |
| $\mathcal{J}_5$ (HI) | **6.68E-27** | 7.50E-22 | 6.12E-2 | 1.17E-5 |
| Iteration | 105 | 105 | 105 | 10,000 |
| $\mathcal{J}_5$ (VI) | **4.34E-26** | 2.71E-4 | 5.56E-2 | 2.15E-4 |




(a) Loss curve for HI




(b) Loss curve for VI

# Summary

1. Adam: Richardson method
2. BFGS: Rank 2 update from the identity matrix
3. SgGN: Block Gauss Seidel
   - Combine geometric intuition and second order information
   - Least square structures and neural network structures
   - More computational cost per iteration: $O(n^2)$

**https://arxiv.org/abs/2404.05064**

# Pseudocode for SgGN

**Algorithm** A structure-guided Gauss-Newton (SgGN) method

**Require:** network parameters $\mathbf{r} = (r_1, \ldots, r_n)$, data set $\{(\mathbf{x}^i, u^i)\}_{i=1}^N$

**Ensure:** network parameters $\mathbf{c}, \mathbf{r}$

Initialize $\mathbf{r}^{(0)}$ by uniform points on the domain $\Omega$

**for** $k = 0, 1, \ldots$ **do**

  **Linear parameter c**

  Form $\mathcal{A}(\mathbf{r}^{(k)}), \mathbf{f}(\mathbf{r}^{(k)})$

  $\mathbf{c}^{(k+1)} \leftarrow \mathcal{A}^{-1}(\mathbf{r}^{(k)})\mathbf{f}(\mathbf{r}^{(k)})$

  **Nonlinear parameter r**

  Form $\mathbf{G}(\mathbf{c}^{(k+1)}, \mathbf{r}^{(k)}), \mathcal{H}(\mathbf{r}^{(k)})$

  $\mathbf{s}^{(k)} \leftarrow -\mathcal{H}^{-1}(\mathbf{r}^{(k)})\mathbf{G}(\mathbf{c}^{(k+1)}, \mathbf{r}^{(k)})$

  $\mathbf{p}^{(k)} \leftarrow (D^{-1}(\mathbf{c}^{(k+1)}) \otimes I_{d+1})\mathbf{s}^{(k)}$

  $\gamma_{k+1} \leftarrow \arg\min_{\gamma \in \mathbb{R}_0^+} \mathcal{J}_\mu(u_n(\cdot; \mathbf{c}^{(k+1)}, \mathbf{r}^{(k)} + \alpha \mathbf{p}^{(k)}))$

  $\mathbf{r}^{(k+1)} \leftarrow \mathbf{r}^{(k)} + \gamma_{k+1}\mathbf{p}^{(k)}$

  **if** a desired loss or a specified number of iterations is reached **then**

    **return** $\mathbf{c}^{(k+1)}, \mathbf{r}^{(k+1)}$

  **end if**

**end for**