Fast Iterative Solver for Neural Network Method: 1D Diffusion Problems

César Herrera[1]
Zhiqiang Cai[1] Anastassia Doktorova[1] Robert D. Falgout[2]

[1]Department of Mathematics, Purdue University
[2]Lawrence Livermore National Laboratory

Preliminary Exam, Purdue University
March 2024

Table of Contents

1. Poisson's Equation and Neural Network Approximation
2. Systems of algebraic equations
3. Coefficient matrix
4. A Damped Block Newton (dBN) Method
5. Numerical results

Table of Contents

1. Poisson's Equation and Neural Network Approximation
2. Systems of algebraic equations
3. Coefficient matrix
4. A Damped Block Newton (dBN) Method
5. Numerical results

1D Diffusion Problem

Consider the Poisson equation

$$\begin{cases} -(a(x)u'(x))' = f(x), & x \in I = (0,1), \\ u(0) = \alpha, \quad u(1) = \beta \end{cases}$$

Ritz formulation: find $u \in H^1(I)$ such that

$$u = \underset{\substack{v \in H^1(I) \\ v(0)=\alpha, v(1)=\beta}}{\arg\min} \left\{ \frac{1}{2} \int_0^1 a(x)(v'(x))^2 dx - \int_0^1 f(x)v(x)dx \right\}$$

Let

$$\mathcal{M}_n(I) \quad = \quad \left\{ c_{-1} + \sum_{i=0}^{n} c_i \sigma(x - b_i) \ : \ c_i \in \mathbb{R}, 0 \leq b_i \leq 1, b_i < b_{i+1} \right\}$$

Modified Ritz formulation

Given $\gamma > 0$, let $J : H^1(I) \to \mathbb{R}$ be the modified energy functional given by

$$J(v) = \frac{1}{2}\int_0^1 a(x)(v'(x))^2 dx - \int_0^1 f(x)v(x)dx + \frac{\gamma}{2}(v(b) - \beta)^2$$

Ritz neural network approximation: find $u_n(x) \in \mathcal{M}_n(I)$ such that

$$J(u_n) = \min_{\substack{v \in \mathcal{M}_n(I) \\ v(0)=\alpha}} J(v)$$

Error estimate

Let $\Sigma_n(I)$ be the set of functions $S : \mathbb{R} \to \mathbb{R}$, where S is a CPwL function with at most n distinct breakpoints in I. We have the following inclusion[1]:

$$\Sigma_{n-1}(I) \subset \mathcal{M}_n(I) \subset \Sigma_n(I)$$

Theorem

Let u be the exact solution of the diffusion problem and $u_n \in \mathcal{M}_n(I)$ be the Ritz neural network approximation. If $u \in H^2(I)$, then

$$\|u - u_n\|_a \leq \frac{C}{n}|u|_{H^2(I)} + \frac{2\sqrt{2}}{\sqrt{\gamma}}|a(1)u'(1)|,$$

where $\|v\|_a^2 = \int_0^1 a(x)(v'(x))^2 dx + \gamma(v(1))^2$.

[1] I. Daubechies et al. "Nonlinear Approximation and (Deep) ReLU Networks". English (US). in: *Constructive Approximation* 55.1 (Feb. 2022), pp. 127–172. ISSN: 0176-4276. DOI: 10.1007/s00365-021-09548-z.

Table of Contents

1 Poisson's Equation and Neural Network Approximation

2 Systems of algebraic equations

3 Coefficient matrix

4 A Damped Block Newton (dBN) Method

5 Numerical results

Systems of algebraic equations

Let

$$u_n = u_n(x) = u_n(x; \mathbf{c}, \mathbf{b}) = \alpha + \sum_{i=0}^{n} c_i \sigma(x - b_i)$$

be a solution of the previous minimization problem. Then the linear and nonlinear parameters

$$\mathbf{c} = (c_0, \ldots, c_n)^T \quad \text{and} \quad \mathbf{b} = (b_0, \ldots, b_n)^T$$

satisfy the following system of algebraic equations

$$\nabla_{\mathbf{c}} J(u_n) = \mathbf{0} \quad \text{and} \quad \nabla_{\mathbf{b}} J(u_n) = \mathbf{0}.$$

Linear parameters $\mathbf{c}$

The equation $\nabla_{\mathbf{c}} J(u_n) = 0$ has the form

$$\left(A(\mathbf{b}) + \gamma \mathbf{d}\mathbf{d}^T\right)\mathbf{c} = \mathbf{f}(\mathbf{b}) + \gamma(\beta - \alpha)\mathbf{d},$$

where

- $A(\mathbf{b}) = \displaystyle\int_0^1 a(x) \nabla_{\mathbf{c}} u_n'(x) \left(\nabla_{\mathbf{c}} u_n'(x)\right)^T dx$
- $\mathbf{f}(\mathbf{b}) = \displaystyle\int_0^1 f(x) \nabla_{\mathbf{c}} u_n(x) dx$
- $\mathbf{d} = (b - b_0, \dots, b - b_n)^T$

Nonlinear parameters **b**

Lemma

For $j = 0, 1, \ldots, n$, the j^{th} equation of $\nabla_{\mathbf{b}} J(u_n) = \mathbf{0}$ is given by

$$\frac{\partial}{\partial b_j} J(u_n) = c_j \left(\int_{b_j}^{1} f(x)\, dx - \gamma u_n(1) - a(b_j) \left(\sum_{i=0}^{j-1} c_i + \frac{c_j}{2} \right) + c_j \gamma \beta \right) = 0.$$

Nonlinear parameters $\mathbf{b}$

Lemma

For $j = 0, 1, \ldots, n$, let

$$g(b_j) = f(c_j) + a'(b_j)\left(\sum_{i=0}^{j-1} c_i + \frac{c_j}{2}\right).$$

Then the Hessian matrix $\nabla^2_{\mathbf{b}} J(u_n)$ has the form

$$\mathcal{H}(\mathbf{c}, \mathbf{b}) = \mathbf{B}(\mathbf{c}, \mathbf{b}) + \gamma \mathbf{c}\mathbf{c}^T,$$

where $\mathbf{B}(\mathbf{c}, \mathbf{b}) := \mathrm{diag}(-c_0 g(b_0), \ldots, -c_n g(b_n))$

Table of Contents

1. Poisson's Equation and Neural Network Approximation
2. Systems of algebraic equations
3. Coefficient matrix
4. A Damped Block Newton (dBN) Method
5. Numerical results

Coefficient matrix

$$A(\mathbf{b}) = \begin{pmatrix} \int_{b_0}^1 a(x)dx & \int_{b_1}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \int_{b_1}^1 a(x)dx & \int_{b_1}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \int_{b_2}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \int_{b_2}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \int_{b_n}^1 a(x)dx & \int_{b_n}^1 a(x)dx & \int_{b_n}^1 a(x)dx & \cdots & \int_{b_n}^1 a(x)dx \end{pmatrix},$$

particularly, when $a(x) = 1$

$$A(\mathbf{b}) = \begin{pmatrix} 1-b_0 & 1-b_1 & 1-b_2 & \cdots & 1-b_n \\ 1-b_1 & 1-b_1 & 1-b_2 & \cdots & 1-b_n \\ 1-b_2 & 1-b_2 & 1-b_2 & \cdots & 1-b_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1-b_n & 1-b_n & 1-b_n & \cdots & 1-b_n \end{pmatrix}$$

Condition number

Lemma

Let $a(x) = 1$, then the condition number of the coefficient matrix $A(\mathbf{b})$ is bounded by $\mathcal{O}(n/h_{\text{min}})$.

Inverse of the coefficient matrix

Lemma

The coefficient matrix is invertible and its inverse is given by

$$A(\mathbf{b})^{-1} = \begin{pmatrix} \frac{1}{s_1} & -\frac{1}{s_1} & 0 & 0 & \cdots & 0 & 0 \\ -\frac{1}{s_1} & \frac{1}{s_1}+\frac{1}{s_2} & -\frac{1}{s_2} & 0 & \cdots & 0 & 0 \\ 0 & -\frac{1}{s_2} & \frac{1}{s_2}+\frac{1}{s_3} & -\frac{1}{s_3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \frac{1}{s_{n-1}}+\frac{1}{s_n} & -\frac{1}{s_n} \\ 0 & 0 & 0 & 0 & \cdots & -\frac{1}{s_n} & \frac{1}{s_n}+\frac{1}{s_{n+1}} \end{pmatrix},$$

where $s_i := \int_{b_{i-1}}^{b_i} a(x)dx$.

Table of Contents

1 Poisson's Equation and Neural Network Approximation

2 Systems of algebraic equations

3 Coefficient matrix

4 A Damped Block Newton (dBN) Method

5 Numerical results

A Damped Block Newton (dBN) Method

We want to solve

$$\nabla_{\mathbf{c}} J(u_n) = \mathbf{0} \quad \text{and} \quad \nabla_{\mathbf{b}} J(u_n) = \mathbf{0}.$$

The equation $\nabla_{\mathbf{c}} J(u_n) = 0$ has the form

$$\left(A(\mathbf{b}) + \gamma \mathbf{d}\mathbf{d}^T \right) \mathbf{c} = \mathbf{f}(\mathbf{b}) + \gamma(\beta - \alpha)\mathbf{d},$$

The Hessian matrix $\nabla^2_{\mathbf{b}} J(u_n)$ has the form

$$\mathcal{H}(\mathbf{c}, \mathbf{b}) = \mathbf{B}(\mathbf{c}, \mathbf{b}) + \gamma \mathbf{c}\mathbf{c}^T,$$

where $\mathbf{B}(\mathbf{c}, \mathbf{b})$ is a diagonal matrix.

A Damped Block Newton (dBN) Method

Let $\left(\mathbf{c}^{(k)}, \mathbf{b}^{(k)}\right)$ be the previous iterate. We then compute the current state $\left(\mathbf{c}^{(k+1)}, \mathbf{b}^{(k+1)}\right)$ by doing the following:

(i) Compute the current linear parameters $\mathbf{c}^{(k+1)}$ using

$$\left(A(\mathbf{b}^{(k)}) + \gamma \mathbf{d}\mathbf{d}^T\right) \mathbf{c} = \mathbf{f}(\mathbf{b}^{(k)}) + \gamma(\beta - \alpha)\mathbf{d},$$

(ii) Assume that the Hessian matrix $\mathcal{H}(\mathbf{c}^{(k+1)}, \mathbf{b}^{(k)})$ is invertible. Set the search direction

$$\mathbf{p}^{(k)} = -\mathcal{H}(\mathbf{c}^{(k+1)}, \mathbf{b}^{(k)})^{-1} \nabla_{\mathbf{b}} J(u_n(x; \mathbf{c}^{(k+1)}, \mathbf{b}^{(k)})).$$

(iii) Compute the stepsize η_k

$$\eta_k = \underset{\eta \in \mathbb{R}_+}{\operatorname{argmin}}\, J(u_n(x; \mathbf{c}^{(k+1)}, \mathbf{b}^{(k)} + \eta \mathbf{p}^{(k)})).$$

Set the current nonlinear parameters by

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \eta_k \mathbf{p}^{(k)}.$$

Some remarks

- We notice that we can compute all the matrix inverses exactly.
- If $c_i^{(k+1)} g(b_i^{(k)})$ vanishes for some $i \in \{0, \ldots, n\}$, then the corresponding nonlinear parameter will remain unchanged, i.e., $\mathbf{b}_i^{(k+1)} = \mathbf{b}_i^{(k)}$, and be removed at the step *(ii)* of the method.
- The computational cost of the matrix operations in our algorithm is $\mathcal{O}(n)$.

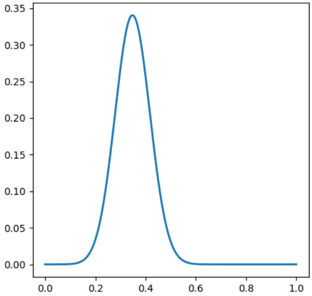
Table of Contents

1. Poisson's Equation and Neural Network Approximation
2. Systems of algebraic equations
3. Coefficient matrix
4. A Damped Block Newton (dBN) Method
5. Numerical results

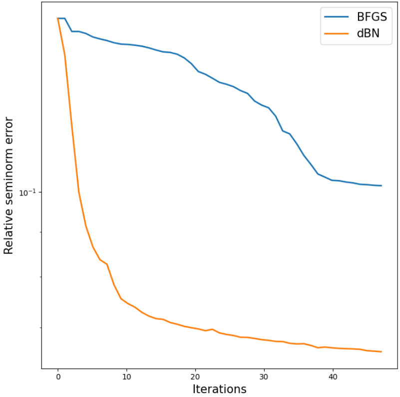
The first test problem involves the function

$$u(x) = x\left(\exp\left(-\frac{(x-\frac{1}{3})^2}{0.01}\right) - \exp\left(-\frac{4}{9\times 0.01}\right)\right)$$

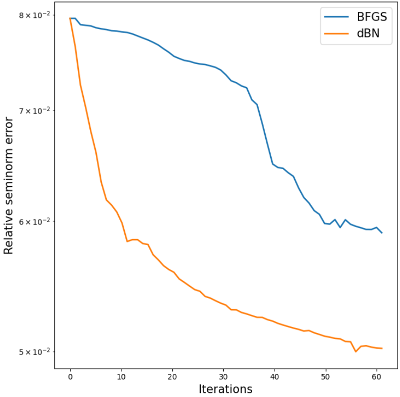
Figure: Graph of the test function $u(x)$



dBN vs BFGS



(a) $\frac{|u-u_n|_{H^1(I)}}{|u|_{H^1(I)}}$ vs number of iterations using 32 neurons, the ratio between the final errors is 0.673



(b) $\frac{|u-u_n|_{H^1(I)}}{|u|_{H^1(I)}}$ vs number of iterations using 64 neurons, the ratio between the final errors is 0.783

Convergence rate

n	e_n	r
60	4.65×10^{-2}	0.749
90	3.41×10^{-2}	0.751
120	2.49×10^{-2}	0.771
150	1.97×10^{-2}	0.783
180	1.78×10^{-2}	0.775
210	1.59×10^{-2}	0.775
240	1.27×10^{-2}	0.796
270	1.22×10^{-2}	0.787
300	1.09×10^{-2}	0.792
330	1.01×10^{-2}	0.792
360	9.94×10^{-3}	0.783
390	9.54×10^{-3}	0.780
420	8.07×10^{-3}	0.798

Table: Relative errors $e_n = \frac{|u_n^{(k)} - u|_{H^1(I)}}{|u|_{H^1(I)}}$ and rates r for n neurons after 1000 iterations.

Adaptivity

Let $\mathcal{K} = [c, d] \subseteq [0, 1]$, define the indicator of $\mathcal{K}$ using the ZZ-estimator

$$\xi_{\mathcal{K}} = \|a^{-1/2}\left(G(au_n') - au_n'\right)\|_{L^2(\mathcal{K})},$$

where $G(au_n')$ is the projection of au_n' onto the continuous piecewise linear space of functions.

Given $u_n \in \mathcal{M}_n(I)$, we determine a partition of $[0, 1]$

$$\mathcal{K}_n = \{[b_{i-1}, b_i]\}_{i=0}^{n+1} = \{\mathcal{K}^i\}_{i=0}^{n+1},$$

where $b_{-1} := 0$.

Adaptivity

Then, we define a subset $\widehat{\mathcal{K}_n} \subset \mathcal{K}_n$ using the average marking strategy[2]:

$$\widehat{\mathcal{K}_n} = \left\{ K \in \mathcal{K}_n : \xi_{\mathcal{K}} \geq \frac{1}{\#\mathcal{K}_n} \sum_{K \in \mathcal{K}_n} \xi_{\mathcal{K}} \right\}$$

where $\#\mathcal{K}_n$ is the number of elements in $\mathcal{K}_n$. For a refinement step, a meshpoint gets added at the midpoint within each element of $\widehat{\mathcal{K}_n}$.

[2] Min Liu and Zhiqiang Cai. "Adaptive two-layer ReLU neural network: II. Ritz approximation to elliptic PDEs". In: *Computers Mathematics with Applications* 113 (2022), pp. 103–116. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2022.03.010.

Adaptivity

Given a tolerance ϵ, start with a small number of neurons n_0, then

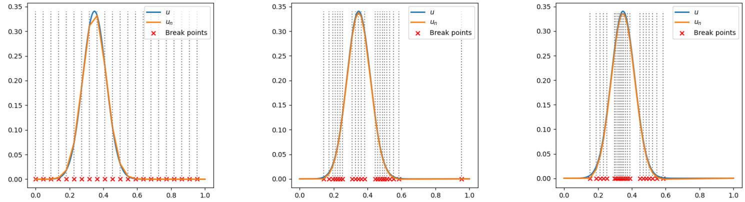
(i) Compute the solution u_n

(ii) Estimate the total error by computing $\xi = \left(\sum_{K \in \mathcal{K}} \xi_K^2 \right)^{1/2} / |u_n|_{H^1(I)}$

(iii) If $\xi \leq \epsilon$, then stop; otherwise go to step (iv)

(iv) Add new neurons to the network by using the network enhancement strategy, then go to step (i)

dBN + Adaptivity



(a) Initial NN model with 22 uniform breakpoints, $\frac{|u-u_n|_1}{|u|_1} = 0.227$

(b) Optimized NN model with 22 breakpoints, 500 iterations, $\frac{|u-u_n|_1}{|u|_1} = 0.100$

(c) Adaptive NN model with 22 breakpoints: 10 initial breakpoints, 2 refinements (13, 22 neurons), $|u - u_n|_1/|u|_1 = 0.083$

Figure: Results of using ReLU networks for approximating function u(x)

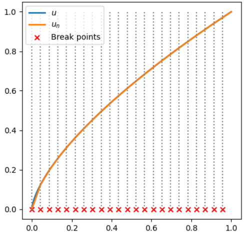
dBN + Adaptivity

NN (n neurons)	e_n	ξ_n	r
Adaptive (19)	9.77×10^{-2}	0.953	0.790
Adaptive (29)	7.30×10^{-2}	0.730	0.807
Adaptive (46)	4.19×10^{-2}	0.563	0.829
Adaptive (77)	2.52×10^{-2}	0.430	0.847
Adaptive (130)	1.53×10^{-2}	0.328	0.859
Adaptive (204)	1.05×10^{-2}	0.265	0.857
Adaptive (272)	8.02×10^{-3}	0.226	0.861
Fixed (204)	1.78×10^{-2}	0.396	0.757
Fixed (272)	1.28×10^{-2}	0.336	0.777

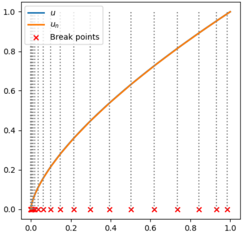
Table: Comparison of an adaptive network with fixed networks for relative error $e_n = \frac{|u_n^{(k)} - u|_{H^1(I)}}{|u|_{H^1(I)}}$, relative error estimators ξ_n, and rates r

dBN + Adaptivity

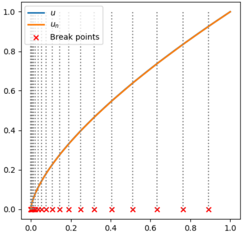
Figure: Results of using ReLU networks for approximating $u(x) = x^{2/3}$



(a) Initial NN model with 23 uniform breakpoints, $\frac{|u-u_n|_1}{|u|_1} = 0.284$



(b) Optimized NN model with 23 breakpoints, 500 iterations, $\frac{|u-u_n|_1}{|u|_1} = 0.056$



(c) Adaptive NN model with 23 breakpoints: 9 initial break points, 2 refinements (14, 23 neurons), $\frac{|u-u_n|_1}{|u|_1} = 0.042$

Conclusions

- **Key compotents of dBN:** decoupling and the use of exact inverses for the matrices
- **Performance:** dBN outperforms state of the art methods such as BFGS
- **Convergence rate:** adaptivity improves the convergence rate
- **Drawback:** dBN as defined here only applies to the one dimensional setting

Current and possible future work

- Current work
  - Mass matrix, 1D difussion-reaction and datta fitting problems
  - Convergence of NN Gauss-Newton methods
- Future work
  - 2D problems
  - Deep NN methods