SAND2020-6970C

Robust training and initialization of deep neural networks: an adaptive basis viewpoint

Mamikon A. Gulian (SNL)
Eric C. Cyr (SNL)
Ravi G. Patel (SNL)
Mauro Perego (SNL)
Nathaniel A. Trask (SNL)

SAND XXXXXX

SIAM AN20. July 9 2020

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA-0003525.

Abstract

- Why do DNNs work? Why can they beat classical statistical methods for classification and classical numerical methods for certain regression algorithms, such as solving conservation laws or ROMs? Many works prove optimal approximation rates (nonlinear N-widths) of DNNs that defeat classical numerical methods as the dimension increases.
- Motivated by the **gap** between theoretical optimal approximation rates of deep neural networks (DNNs) and the accuracy realized in practice, we seek to improve the training of DNNs.
- The adoption of an adaptive basis viewpoint of DNNs leads to novel initializations and a hybrid least squares/gradient descent optimizer.
- We provide analysis of these techniques and illustrate via numerical examples dramatic increases in accuracy and convergence rate for benchmarks characterizing scientific applications where DNNs are currently used, including regression problems and physics-informed neural networks for the solution of partial differential equations.

What does approximation theory predict?

Sandia National Laboratories

- Universal approximation properties of neural networks are often touted as an explanation of the success of deep neural networks (DNNs) in applications.
- Despite their importance, such theorems offer no explanation for the advantages of neural networks, let alone *deep* neural networks, over classical approximation methods, since universal approximation properties are enjoyed by polynomials as well as single layer neural networks.
- To address this, a recent thread has emerged in the literature concerning optimal approximation with deep ReLU networks, where the error in an optimal choice of weights and biases is bounded from above using the width and depth of the neural network.
- Despite this, it remains a challenge to realize these theorized convergence rates for DNNs using practical initialization and training methods. The need is particularly acute in scientific machine learning (SciML) applications which demand greater accuracy and robustness from DNNs.

Loss functions

- We consider in this work the following class of ℓ_2 regression problems:

$$\underset{\xi}{\operatorname{argmin}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \mathcal{L}_k\left[\mathcal{NN}_\xi\right] \right\|^2_{\ell_2(\mathcal{X}_k)} \tag{1}$$

  where for each $k = 1, 2, ..., K$, $\mathcal{X}_k = \{x_i^{(k)}\}_{i=1}^{N_k}$ denotes a finite collection of data points, $\mathcal{NN}_\xi$ a neural network with parameters ξ, and $\mathcal{L}_k$ a linear operator.

- In the case where $k = 1$ and $\mathcal{L}$ is the identity, we obtain the standard regression problem

$$\underset{\xi}{\operatorname{argmin}} \left\| u - \mathcal{NN}_\xi \right\|^2_{\ell_2(\mathcal{X})}.$$

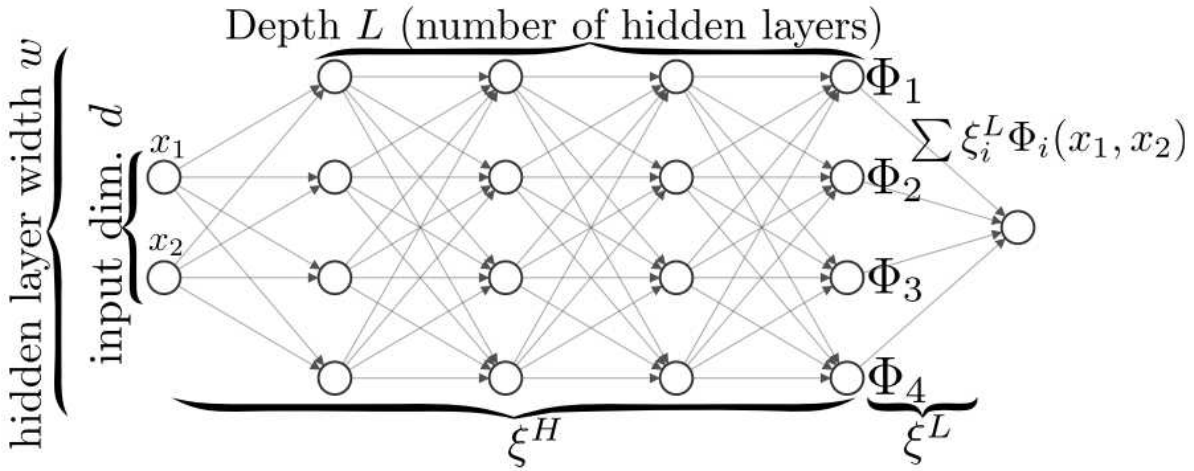
- In general (1) represents a broader class of multi-term loss functions, including those used in physics-informed neural networks for solving linear PDEs.

Adaptive basis viewpoint

- We consider the family of neural networks $\mathcal{NN}_\xi : \mathbb{R}^d \to \mathbb{R}$ consisting of L hidden layers of width w composed with a final linear layer, admitting the representation

$$\mathcal{NN}_\xi(\mathbf{x}) = \sum_{i=1}^{w} \xi_i^L \Phi_i(\mathbf{x}; \boldsymbol{\xi}^H)$$

where $\boldsymbol{\xi}^L$ and $\boldsymbol{\xi}^H$ are the parameters corresponding to the final linear layer and the hidden layers respectively, and we interpret $\boldsymbol{\xi}$ as the concatenation of $\boldsymbol{\xi}^L$ and $\boldsymbol{\xi}^H$.



DNN Architectures

Sandia National Laboratories

- A broad range of architectures admit this interpretation. In this work we consider both plain neural networks (also referred to as multilayer perceptrons) and residual neural networks (ResNets).
- Defining the affine transformation, $\mathsf{T}_l(\mathbf{x}, \boldsymbol{\xi}) = \boldsymbol{W}_l^{\boldsymbol{\xi}} \cdot \mathbf{x} + \mathbf{b}_l^{\boldsymbol{\xi}}$, and given an activation function σ, plain neural networks correspond to the choice

$$\boldsymbol{\Phi}^{\text{plain}}(\mathbf{x}, \boldsymbol{\xi}) = \boldsymbol{\sigma} \circ \mathsf{T}_L \circ \cdots \circ \boldsymbol{\sigma} \circ \mathsf{T}_1,$$

  while residual networks correspond to

$$\boldsymbol{\Phi}^{\text{res}}(\mathbf{x}, \boldsymbol{\xi}) = (\mathbf{I} + \boldsymbol{\sigma} \circ \mathsf{T}_L) \circ \cdots \circ (\mathbf{I} + \boldsymbol{\sigma} \circ \mathsf{T}_2) \circ (\boldsymbol{\sigma} \circ \mathsf{T}_1),$$

  where $\boldsymbol{\Phi}$ is the vector of the w functions Φ_i, $\boldsymbol{\sigma}$ the vector of the w activation functions σ and $\mathbf{I}$ denotes the identity. In both cases $\boldsymbol{\xi}^{\text{H}}$ corresponds to the weights and biases $\boldsymbol{W}$ and $\mathbf{b}$.
- In practice, deep plain DNNs are not trainable. A rule of thumb is if you have more than 10 layers, you should probably use a ResNet.

Hybrid Least Squares/Gradient Descent

- We seek

$$\underset{\xi^L,\,\xi^H}{\operatorname{argmin}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \sum_i \xi_i^L \mathcal{L}_k\left[\Phi_i(x,\xi^H)\right] \right\|^2_{\ell_2(\mathcal{X}_k)}.$$

  A typical approach to solving Equation 7 is to apply gradient descent with backpropagation jointly in (ξ^L, ξ^H).

- Given the adaptive basis viewpoint, an alternative is to hold the hidden weights ξ^H constant and minimize w.r.t. to ξ^L, yielding the LS problem (for simplicity focusing on $K = 1$):

$$\underset{\xi^L}{\operatorname{argmin}} \left\| A\xi^L - b \right\|^2_{\ell_2(\mathcal{X})}$$

  Here we have $b_i = \mathcal{L}[u](x_i)$ and $A_{ij} = \mathcal{L}\left[\Phi_j(x_i,\xi^H)\right]$ for $x_i \in \mathcal{X},\ i = 1,\ldots,N,\ j = 1,\ldots,w$.

Hybrid Least Squares/Gradient Descent

Sandia National Laboratories

- Exposing the LS problem in this way prompts a natural modification of gradient descent. The optimization algorithm proceeds by alternating between: a LS solve to update $\boldsymbol{\xi}^{\mathrm{L}}$ by a global minimum for given $\boldsymbol{\xi}^{\mathrm{H}}$, and a GD step to update $\boldsymbol{\xi}^{\mathrm{H}}$.

Algorithm 1 Hybrid least squares/gradient descent

1: **function** LSGD($\boldsymbol{\xi}_0^H$)
2: $\quad\boldsymbol{\xi}^H = \boldsymbol{\xi}_0^H$ ▷ Input initialized hidden parameters
3: $\quad\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$ ▷ Solve LS problem for $\boldsymbol{\xi}^L$
4: $\quad$**for** $i = 1 \ldots$ **do**
5: $\quad\quad\boldsymbol{\xi}^H = GD(\boldsymbol{\xi})$ ▷ Solve GD problem
6: $\quad\quad\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$
7: $\quad$**end for**
8: **end function**

Illustration of LSGD

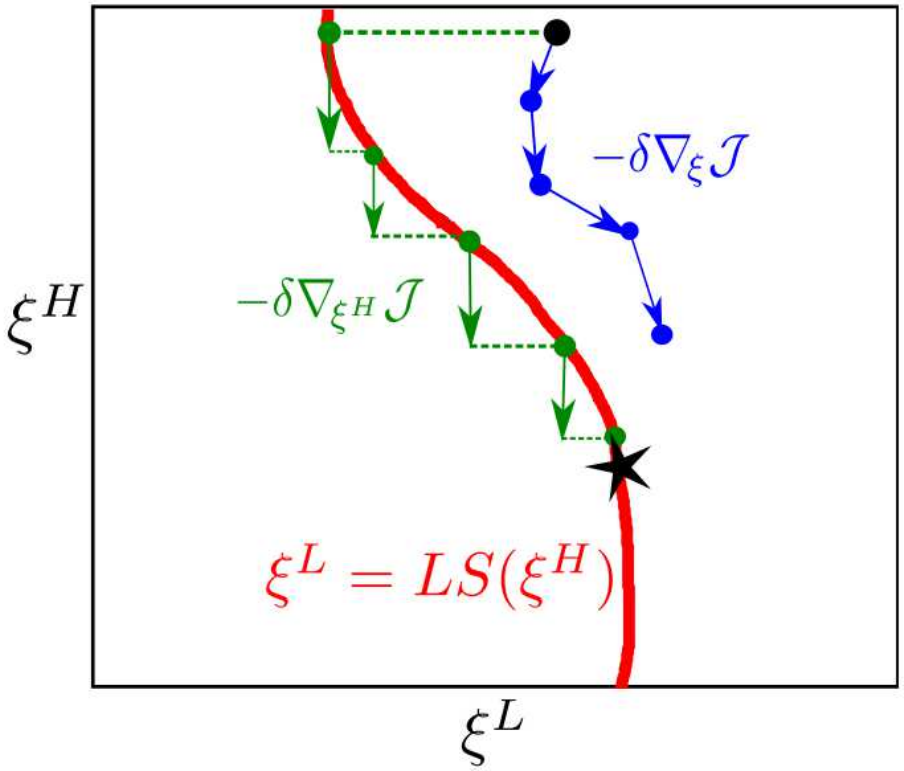


Figure: LSGD algorithm. The black dot denotes the initial guess and the black star a local minimum. The red line represents the submanifold (ξ^H, ξ^L) for which ξ^L is a solution to the least squares problem for fixed ξ^H, written $\xi^L = LS(\xi^H)$, on which $\nabla_\xi \mathcal{J} = (\nabla_{\xi^H} \mathcal{J}, 0)$. Since the black star must also be a global minimum in ξ^L, it lies on this submanifold. The blue curve represents GD, and the rectilinear green curve LSGD. Each LS solve (dashed green line) moves the parameters to the submanifold $\xi^L = LS(\xi^H)$.

LSGD vs GD

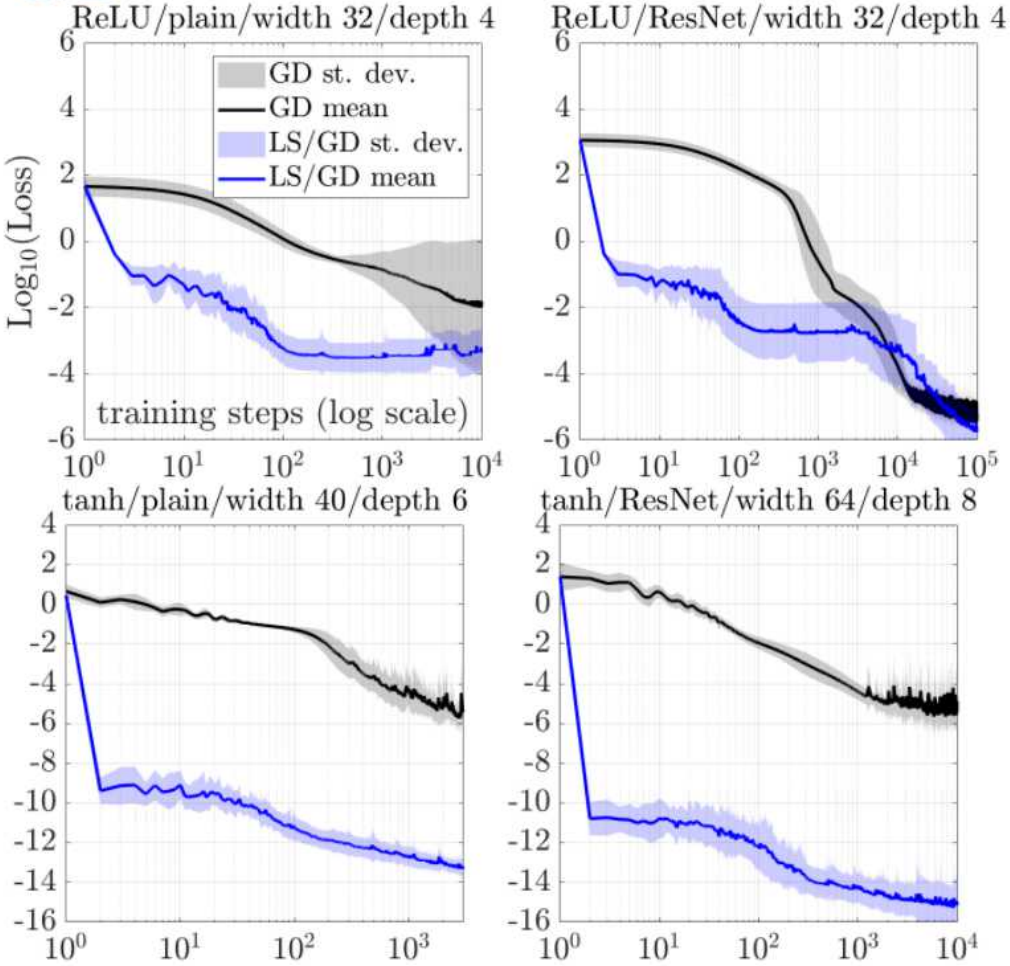


Figure: Mean of $\log_{10}(\text{Loss})$ over 16 training runs $\pm$ one standard deviation of the same quantity, for approximating $\sin(2\pi x)$ on $[0, 1]$ sampled at 256 evenly spaced points.

The Box initialization for ReLU DNNs

Sandia National Laboratories

- We demonstrate how the He/Glorot initializations, for fixed width and increasing depth, rapidly lead to a set of constant basis functions for plain networks and linearly dependent basis functions for deep ReLU network (with diverging slope). These are bad sets of initial basis functions!
- This is by design – for a shallow ReLU network, the bias (breakpoint) is chosen to be zero and the slope is sampled randomly.
- From a C^0 finite element point of view, it's better to scatter the breakpoints (in one-dimension) or cut-planes (in higher dimensions) of the ReLU functions randomly in the domain where data is available. Then, each basis function will be sensitive to local changes in parameters.
- For a deeper network, if we control the slopes (weights) based on where the breakpoints/cut-planes are, we can control the output a given layer, and then iterate the initialization through the layers. The goal is to have as high a rank as possible in the initialized basis.

Plain networks

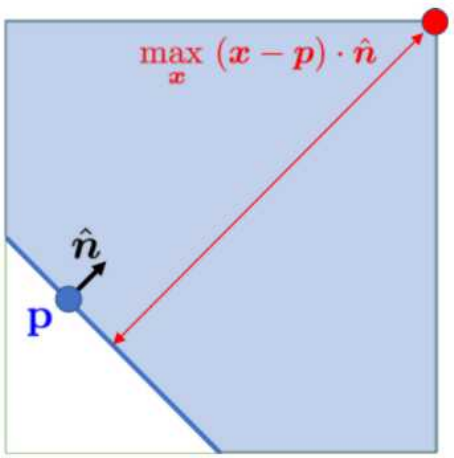


Figure: Notation used in the "Box initialization" of each node. A random point $\mathbf{p}$ with random orientation $\hat{\mathbf{n}}$ is used to define a ReLU function of form $\sigma(\mathrm{k}(\mathbf{x}-\mathbf{p})\cdot\hat{\mathbf{n}})$. One may choose the slope of the ReLU α to impose an upper bound on the output of each layer. We refer to the hyperplane normal to $\hat{\mathbf{n}}$, where the ReLU "switches on", as the *cut plane*.

For each output row $(1\ldots\mathrm{i}\ldots w)$ of the layer:

1. Select $\mathbf{p}\in[0,1]^w$ at random.
2. Select a normal $\mathbf{n}$ at $\mathbf{p}$ with random direction.
3. Choose a scaling k such that

$$\max_{\mathbf{x}\in[0,1]^w}\sigma(\mathrm{k}(\mathbf{x}-\mathbf{p})\cdot\mathbf{n})=1.$$

4. Row $\boldsymbol{w}_\mathrm{i}$ of $\boldsymbol{W}^\xi$ and $\mathbf{b}^\xi$ are selected as $\mathrm{b}_\mathrm{i}=\mathrm{k}\mathbf{p}\cdot\mathbf{n}$ and $\boldsymbol{w}_\mathrm{i}=\mathrm{k}\mathbf{n}^\mathsf{T}$.

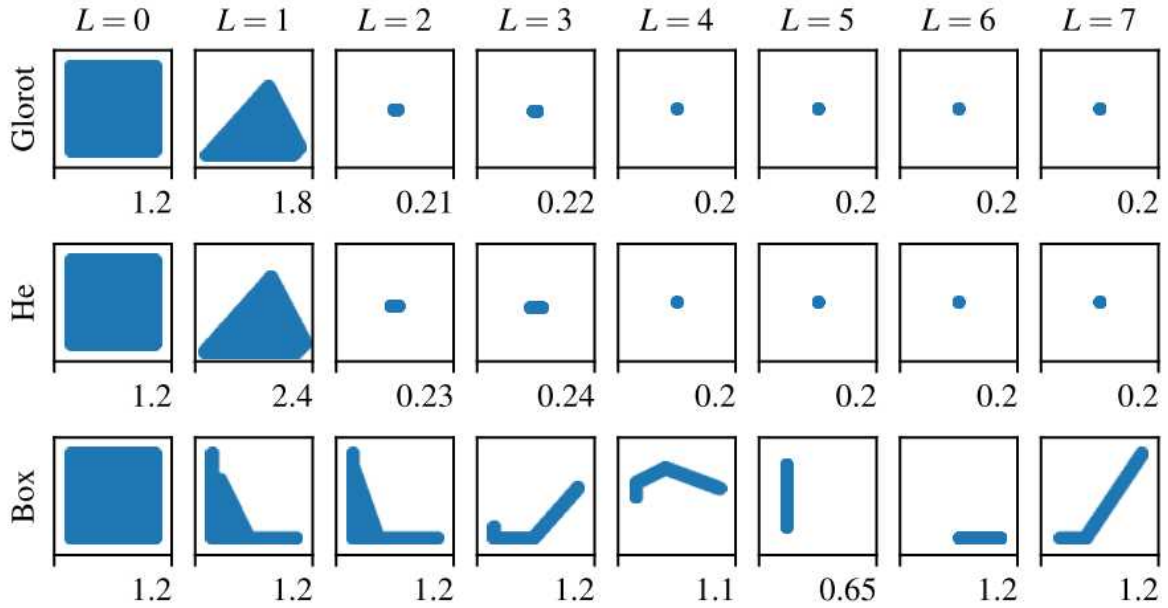


Figure: Images P_L of the unit square $[0, 1]^2$ under L initialized hidden layers of **plain** networks for He (*top*) and Box (*bottom*) initializations. Values are presented on the square $[-0.2, \mathsf{H}]^2$, where H is denoted to the bottom-right of each image. Collapse to a point corresponds to constant basis functions.

Residual Networks

Sandia National Laboratories

- For a ResNet, unless $d = w$, the first hidden layer is initialized as a plain layer. Then, for the remaining hidden layers, to initialize the neuron i, $1 \leqslant i \leqslant w$,
  1. For m specified later, select $\mathbf{p} \in [0, m]^w$ at random.
  2. Select a unit normal $\mathbf{n}$ at $\mathbf{p}$ with random direction.
  3. For δ specified later, choose a scaling k such that

$$\max_{[0,m]^w} \sigma(k(\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}) = \delta m.$$

  4. Row $\boldsymbol{w}_i$ of $\boldsymbol{W}^{\xi}$ and $\mathbf{b}^{\xi}$ is selected as $b_i = k\mathbf{p} \cdot \mathbf{n}$ and $\boldsymbol{w}_i = k\mathbf{n}^T$.

  Assuming the input into the first hidden layer is contained in $[0, 1]^w$, initializing the hidden layers with $\delta = \frac{1}{L}$ leads to a network such that the final output of the hidden layer is contained in the box $[0, e]^w$; in other words, the values of each basis function are contained in $[0, e]$.

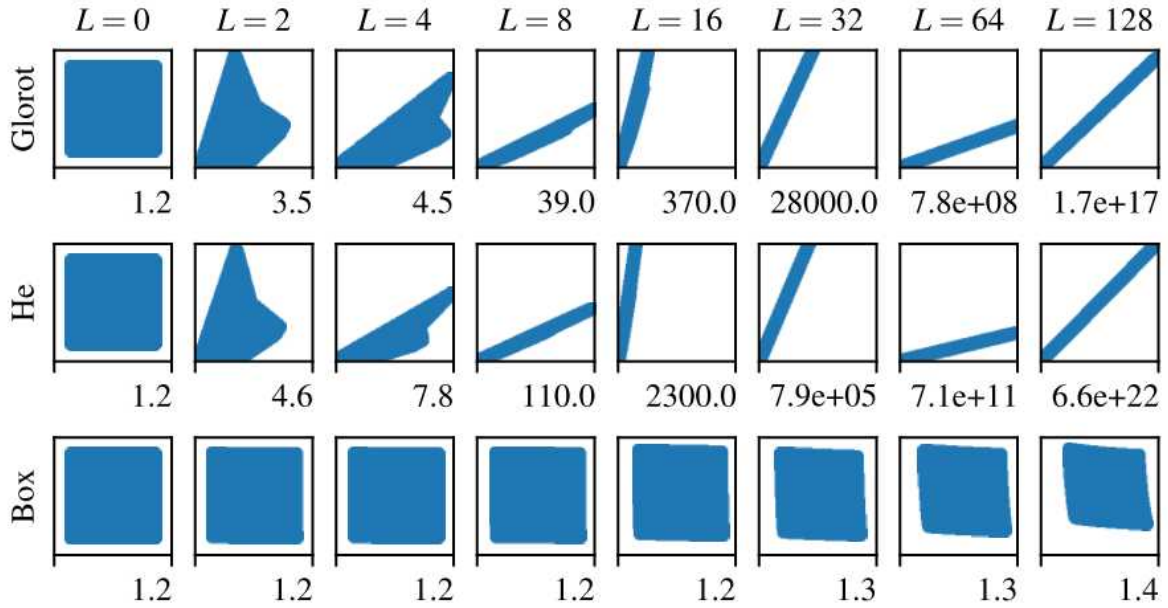


Figure: Images of the unit square $[0, 1]^2$ under L initialized hidden layers of **ResNets** for Glorot (*top*), He (*center*) and Box (*bottom*) initializations. Values are presented on the square $[-0.2, H]^2$, where H is denoted to the bottom-right of each image. Collapse to a line through the origin corresponds to linearly dependent basis functions (i.e., $\phi_1 = C\phi_2$).

Effect on training

- We compare the use of the Box initialization for a residual neural network with hidden layer width 32 against the He initialization for approximating $\sin(2\pi x)$ using 256 evenly spaced samples in $[0, 1]$. We average over 16 independent runs.

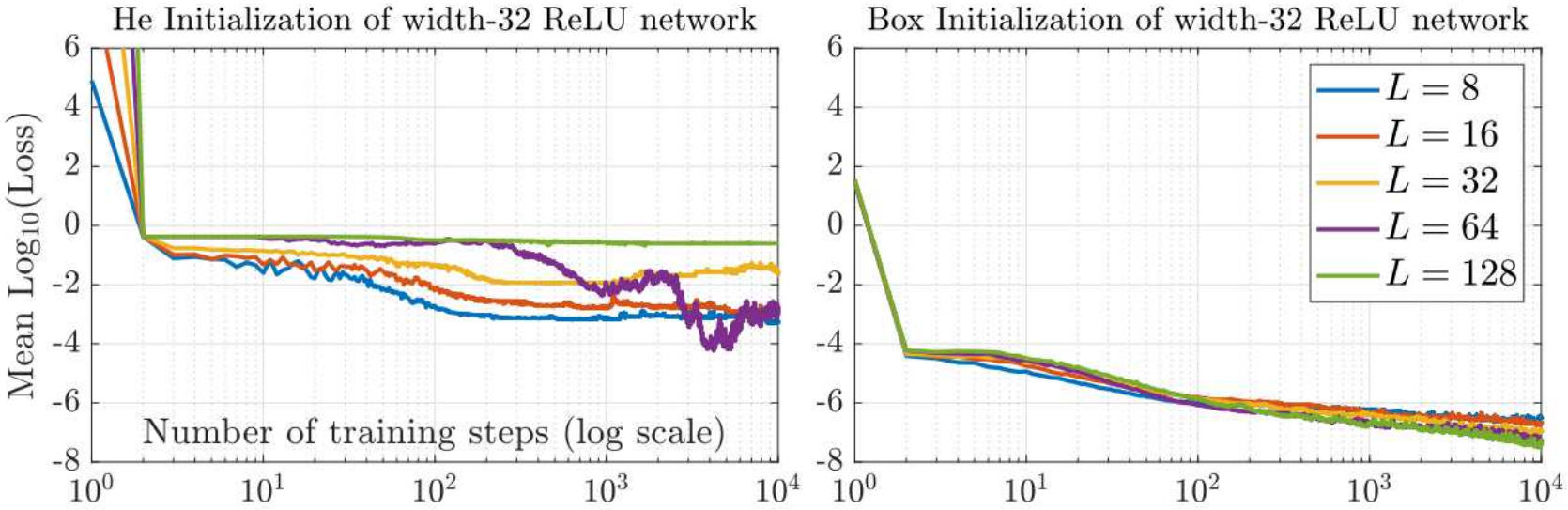


Figure: Mean of $\log_{10}(\text{Loss})$ over 16 training runs of residual width-32 ReLU network with $L = 8, 16, 32, 64$ and 128 hidden layers and training rate $2^{-(k+3)}$ for the He *(left)* and Box *(right)* initializations.

Application: PINN solver for Advection Eq.

Sandia National Laboratories

- We consider now a physics-informed neural network (PINN) solution to the linear transport equation $\partial_t u(x,t) + a(x,t)\,\partial_x u(x,t) = 0$ on the unit space-time domain $(x,t) \in [0,1]^2$, with initial condition $u(x, t=0) = u_0(x)$ and homogeneous Dirichlet boundary data $u(x=0,t) = 0$.
- The loss function considered here is

$$\mathcal{J} = \epsilon \mathcal{J}_1 + \mathcal{J}_2 + \mathcal{J}_3, \qquad \mathcal{J}_1 = \frac{1}{N_1} \sum_{i \in \mathcal{X}_1} |\partial_t \mathcal{NN}_i + \partial_x a(x,t) \mathcal{NN}_i|^2,$$

$$\mathcal{J}_2 = \frac{1}{N_2} \sum_{i \in \mathcal{X}_2} |\mathcal{NN}_i(x,0) - u_0|^2, \qquad \mathcal{J}_3 = \frac{1}{N_3} \sum_{i \in \mathcal{X}_3} |\mathcal{NN}_i(0,t)|^2$$

where $\mathcal{X}_1, \mathcal{X}_2$ and $\mathcal{X}_3$ are Cartesian point clouds with spacing Δx on the interior, left and bottom boundaries, respectively.

Constant Velocity

- For constant velocity, $a(x,t) = 1$, the analytical solution is $u(x,t) = u_0(x-t)$. We use a shallow one-layer ReLU network.
- For this case, the exact solution is in the range of the network for width $\geqslant 3$, and at this point $\mathcal{J}_1 = \mathcal{J}_2 = \mathcal{J}_3 = 0$, rendering the choice of ϵ unimportant (we set $\epsilon = 1$).

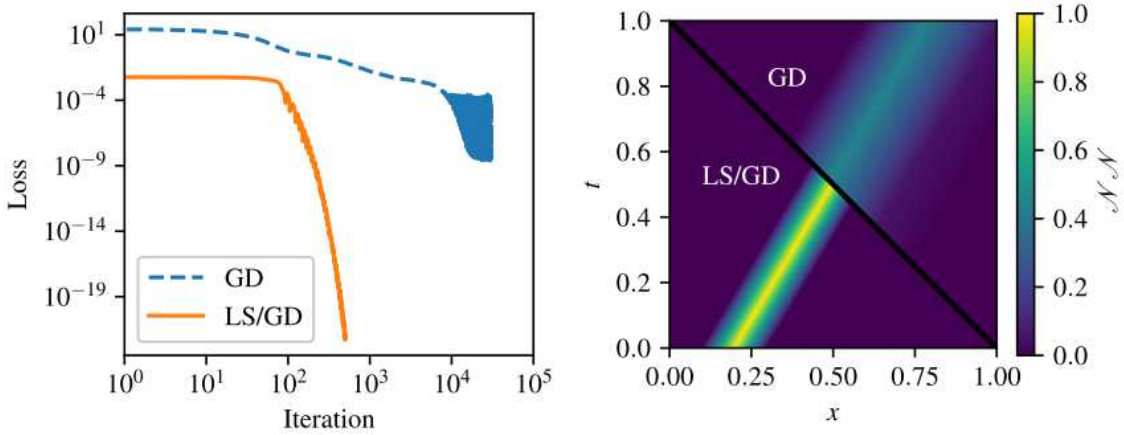


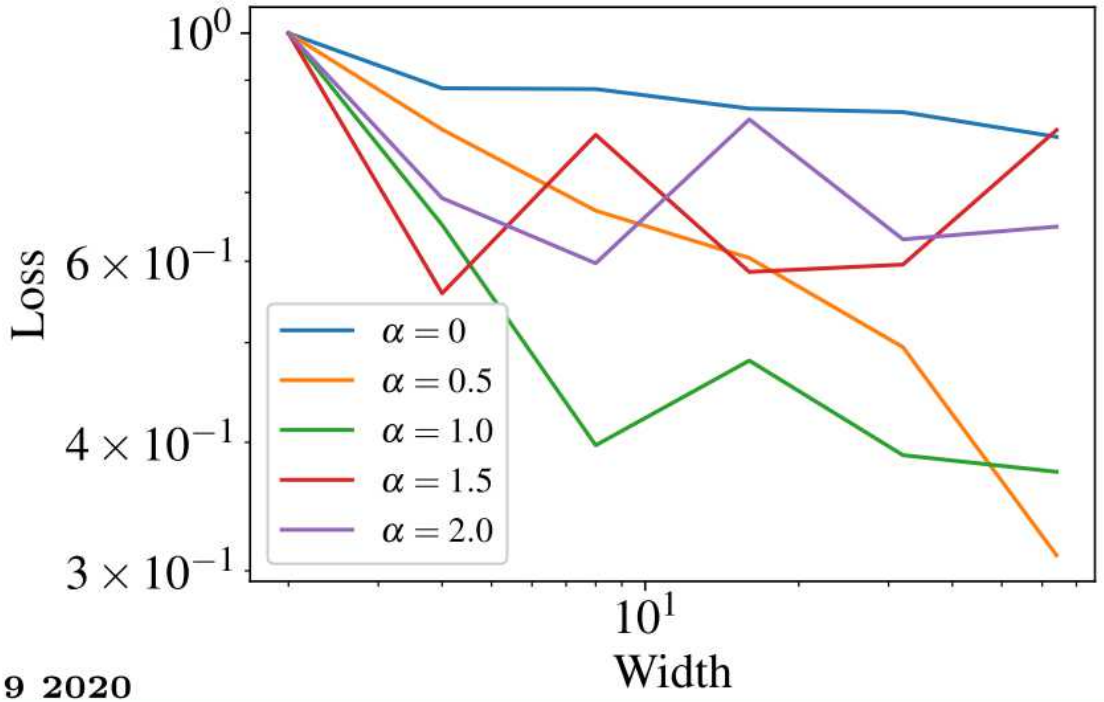
Figure: *Left:* Loss evolution over training for GD and LSGD. *Right:* Solution after 5000 iterations for GD and 500 iterations for LSGD. Setting: Box initialization, ReLU activation function, network width = 32, depth = 1, learning rate = 0.005.

Nonconstant Velocity

- We next consider nonconstant velocity, $a(x,t) = x$, with corresponding analytic solution

$$u(x,t) = u_0(x\exp(-t)).$$

- In this case we must fix ϵ independent of the neural network size to realize convergence. We hypothesized $\epsilon = W^{-\alpha}$ and identified $\alpha = 1/2$ as revealing $O(W^{\frac{1}{2}})$ convergence rate w.r.t. width.



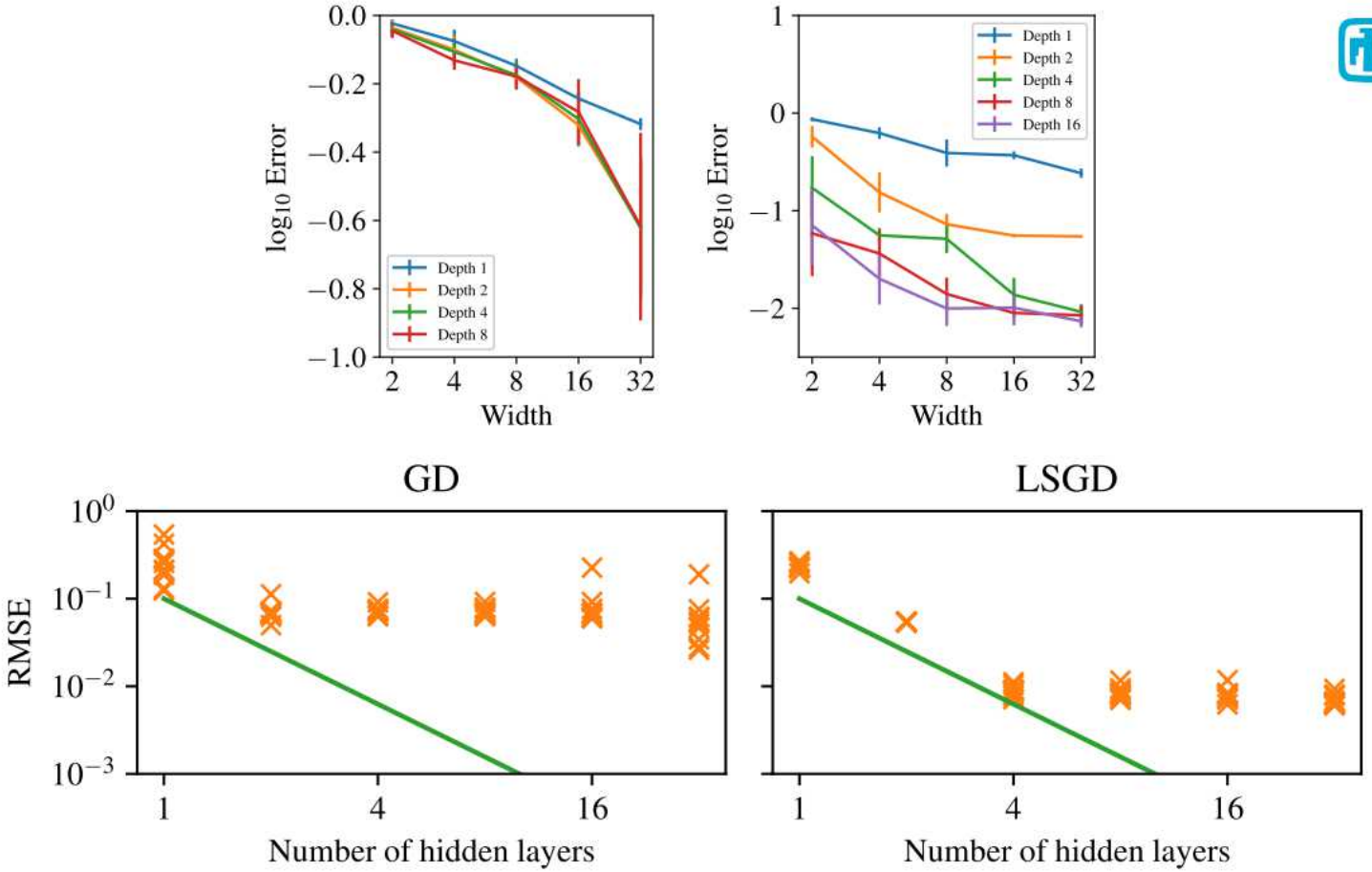


Figure: *Middle:* Convergence with ReLU (left; learning rate 0.001) using $\alpha = -\frac{1}{2}$, and tanh (right; learning rate 0.01) using $\alpha = -\frac{1}{2}$.
Bottom: Comparison of GD (*left*) and LSGD (*right*) training for tanh activation, learning rate 0.01, width 32 and 5000 steps. X's indicate errors for different realizations of the Box initialization. The line indicates second order convergence w.r.t. depth.

Acknowledgements

- The work of R. Patel, M. Perego, N. Trask, and M. Gulian is supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project.
- E. C. Cyr is supported by the Department of Energy early career program.
- M. Gulian is supported by the John von Neumann fellowship at Sandia National Laboratories.
- Thank you to the organizers and to SIAM!