11530C

Robust architectures, initialization, and training for deep neural networks via the adaptive basis interpretation

Mamikon Gulian
Sandia National Laboratories

Exceptional service in the national interest

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 Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND number: XXXXXXXXXX

Part I: Introduction & Background

Overview

- DNNs have proven successful in very high-dimensional image classification problems.
- In practice, this is due to highly parallel and optimized algorithms for training DNNs.
- Approximation theory for DNNs has been developed in an attempt to explain the success of DNNs.
- At the same time, physics-informed neural networks have shown intriguing potential for fusing data and physics knowledge in setups that resemble traditional numerical methods.
- Significant gap between what approximation theory says is optimal and the results when used with standard architectures and optimization algorithms.
- This talk addresses issues about robustness, reproducibility, and accuracy of DNNs when used for numerical tasks.

Deep Neural Networks

- Defining the affine transformation, $\boldsymbol{T}_l(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{W}_l^{\boldsymbol{\xi}} \cdot \boldsymbol{x} + \boldsymbol{b}_l^{\boldsymbol{\xi}}$, and given an activation function σ, a feedforward neural network "hidden layer" architecture may be

$$\mathcal{F}_{\text{HL}}(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{\sigma} \circ \boldsymbol{T}_L \circ \cdots \circ \boldsymbol{\sigma} \circ \boldsymbol{T}_1. \tag{1}$$

  A typical activation function σ is ReLU, defined by $\text{ReLU}(x) = x$ if $x \geq 0$ and $\text{ReLU}(x) = 0$ else.

- For regression, the DNN is typically of the form

$$\mathcal{N}\mathcal{N}(\boldsymbol{x}; \boldsymbol{\xi}, \boldsymbol{W}) = \mathcal{F}_{\text{LL}}(\boldsymbol{x}; \boldsymbol{W}) \circ \mathcal{F}_{\text{HL}}(\boldsymbol{x}, \boldsymbol{\xi}) \tag{2}$$

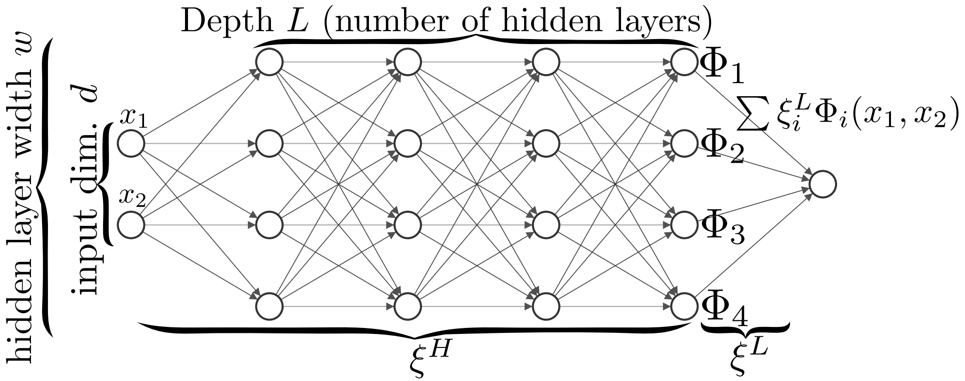
  where the linear layer is given by $\mathcal{F}_{\text{LL}}(\boldsymbol{x}; \boldsymbol{W}) = \mathbf{W}\mathbf{x}$.

- For example, a DNN of one-dimensional input and output, with one-hidden layer and ReLU activation, is CPWL function $\sum_{i=1}^{\text{width}} c_i \sigma(w_i x + b_i)$ with breakpoints characterized by hidden layer parameters.

- For classification, the DNN is typically

$$\mathcal{N}\mathcal{N}(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{W}) = \mathcal{F}_{\text{SM}} \circ \mathcal{F}_{\text{LL}}(\cdot; \mathbf{W}) \circ \mathcal{F}_{\text{HL}}(\mathbf{x}; \xi), \tag{3}$$

  where the softmax function is given by $\mathcal{F}_{\text{SM}}^i(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{N_c} \exp(x_j)}$.



Depth L (number of hidden layers)
hidden layer width w
input dim. d
x_1
x_2
Φ_1
Φ_2
Φ_3
Φ_4
$\sum \xi_i^L \Phi_i(x_1, x_2)$
ξ^H
ξ^L

Loss functions and gradient descent

- In a regression problem, the loss function may be of the form

$$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{W}) = \|u(x) - \mathcal{NN}(x, \boldsymbol{\xi}, \boldsymbol{W})\|^2_{\ell_2(\mathcal{X})}, \tag{4}$$

  where $\mathcal{X}$ is a finite set of points x in $\mathbb{R}^d$ over which $u(x)$ is known. This is your training set.

- In classification problems, the loss function is typically a cross-entropy loss, which measures the deviation of the (tentative) learned probability distribution parametrized by $\mathcal{NN}$ from the distribution of classes over a fixed, finite set of training points.

- Using the chain rule, the gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\xi}$ can be expressed in terms of the values and derivative values of nodes of the DNN through the layers. This can be calculated in an efficient, parallelizable way from the graph structure of the DNN, an algorithm known as backpropagation or automatic differentiation.

- DNNs are typically trained using some variety of gradient descent, such as stochastic gradient descent.

Physics-informed neural networks

- This involves a simple modification to the loss.
- Suppose in addition to having some data for a field u, you know that $Pu = f$, where P is some (possibly nonlinear) differential operator.

$$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{W}) = \|u(x) - \mathcal{N}\mathcal{N}(x, \boldsymbol{\xi}, \boldsymbol{W})\|^2_{\ell_2(\mathcal{X})} + \left\| f(x) - P\left[\mathcal{N}\mathcal{N}(x, \boldsymbol{\xi}, \boldsymbol{W})\right] \right\|^2_{\ell_2(\text{collocation points})}. \tag{5}$$

- Can work with this loss function exactly as before, using off-the-shelf tools such as Tensorflow, Pytorch, etc.
- Typically, the first term is broken up into data on the interior of a domain and data on the boundaries (ICs and BCs).
- Fuses data on u and PDE for u; can use both.
- Warning: BCs, ICs, and Source term must be treated as scattered data. Question of what weights to put in front of each term in the loss.

Approximation Theory for DNNs

- Approximation theory for DNNs has made several recent strides.
- **Important Result:** DNNs can approximate partitions-of-unity and monomials to an accuracy which decays exponentially with the depth of the network, i.e., there exist parameters for such DNNs that give such error rates w.r.t. depth.
- Therefore, DNNs can emulate *hp*-FEM approximation, among other approximation classes.
- However, these theoretical results are only about existence of parameters giving such approximation. Such optimal parameters cannot be found using practical training methods! They do not address optimization error and generalization error.
- In practice, DNNs exhibit **severe** issues related to bad initializations ("dead gradients"), unstable and irreproducible training, and hyperparameter optimization.
- Just getting DNNs to perform well for basic numerical tasks requires a lot of tuning and art.

Part II: Least Squares/Gradient Descent Training and Box Initialization

Problems & Loss functions

- We consider the following class of ℓ_2 regression problems:

$$\underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \mathcal{L}_k \left[\mathcal{NN}_{\boldsymbol{\xi}}\right] \right\|_{\ell_2(\mathcal{X}_k)}^2 \tag{6}$$

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

$$\underset{\boldsymbol{\xi}}{\operatorname{argmin}} \left\| u - \mathcal{NN}_{\boldsymbol{\xi}} \right\|_{\ell_2(\mathcal{X})}^2. \tag{7}$$

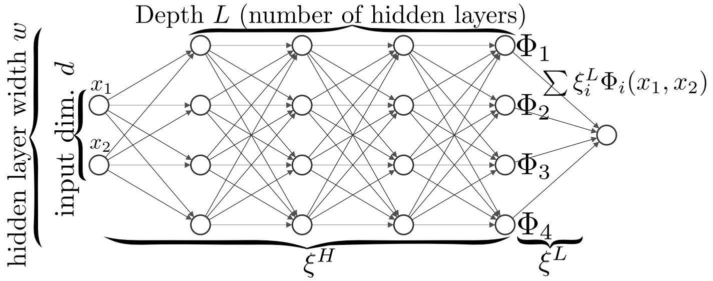
- In general, the multi-term loss is used, e.g., in physics-informed neural networks for solving linear PDEs.

Adaptive basis viewpoint

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

$$\mathcal{NN}_{\boldsymbol{\xi}}(\boldsymbol{x}) = \sum_{i=1}^{w} \xi_i^{\mathrm{L}} \Phi_i(\boldsymbol{x}; \boldsymbol{\xi}^{\mathrm{H}}) \tag{8}$$

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



DNN Architectures

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

$$\boldsymbol{\Phi}^{\text{plain}}(\boldsymbol{x},\boldsymbol{\xi}) = \boldsymbol{\sigma} \circ \boldsymbol{T}_L \circ \cdots \circ \boldsymbol{\sigma} \circ \boldsymbol{T}_1, \tag{9}$$

  while residual networks correspond to

$$\boldsymbol{\Phi}^{\text{res}}(\boldsymbol{x},\boldsymbol{\xi}) = (\boldsymbol{I} + \boldsymbol{\sigma} \circ \boldsymbol{T}_L) \circ \cdots \circ (\boldsymbol{I} + \boldsymbol{\sigma} \circ \boldsymbol{T}_2) \circ (\boldsymbol{\sigma} \circ \boldsymbol{T}_1), \tag{10}$$

  where $\boldsymbol{\Phi}$ is the vector of the w functions Φ_i, $\boldsymbol{\sigma}$ the vector of the w activation functions σ and $\boldsymbol{I}$ denotes the identity. In both cases $\boldsymbol{\xi}^{\text{H}}$ corresponds to the weights and biases $\boldsymbol{W}$ and $\boldsymbol{b}$.
- In practice, very 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

$$\underset{\boldsymbol{\xi}^{\mathrm{L}}, \boldsymbol{\xi}^{\mathrm{H}}}{\operatorname{argmin}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \sum_i \xi_i^{\mathrm{L}} \mathcal{L}_k \left[\Phi_i(\boldsymbol{x}, \boldsymbol{\xi}^{\mathrm{H}}) \right] \right\|_{\ell_2(\mathcal{X}_k)}^2 . \tag{11}$$

  A typical approach to solving this problem is to apply gradient descent with backpropagation jointly in $(\boldsymbol{\xi}^{\mathrm{L}}, \boldsymbol{\xi}^{\mathrm{H}})$.

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

$$\underset{\boldsymbol{\xi}^{\mathrm{L}}}{\operatorname{argmin}} \left\| A\boldsymbol{\xi}^{\mathrm{L}} - \boldsymbol{b} \right\|_{\ell_2(\mathcal{X})}^2 \tag{12}$$

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

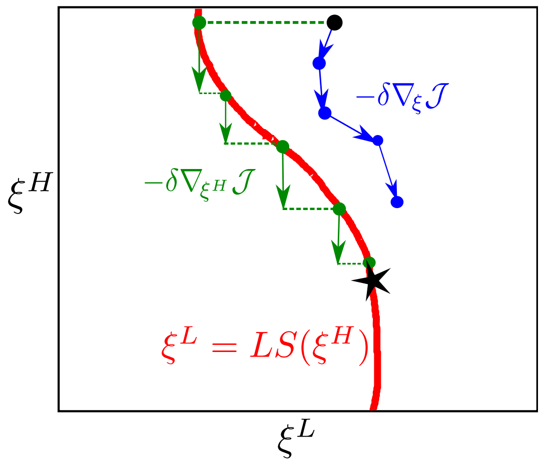
Hybrid Least Squares/Gradient Descent (LSGD)

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

Algorithm 1 Hybrid least squares/gradient descent

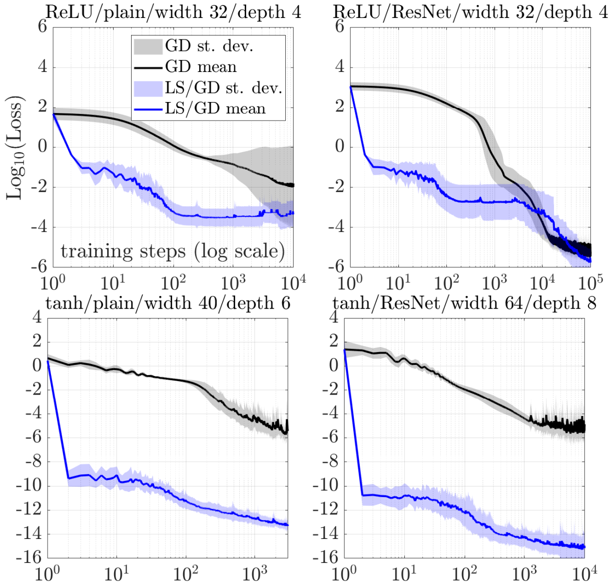
```
1: function LSGD($\boldsymbol{\xi}_0^H$)
2:     $\boldsymbol{\xi}^H = \boldsymbol{\xi}_0^H$          ▷ Input initialized hidden parameters
3:     $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$          ▷ Solve LS problem for $\boldsymbol{\xi}^L$
4:     for $i = 1 \ldots$ do
5:         $\boldsymbol{\xi}^H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

Illustration of LSGD



$-\delta\nabla_{\xi}\mathcal{J}$
$-\delta\nabla_{\xi^H}\mathcal{J}$
$\xi^L = LS(\xi^H)$
ξ^H
ξ^L

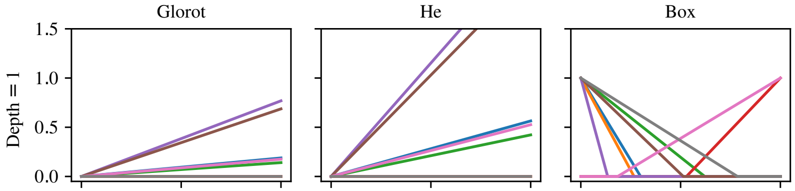
Figure: LSGD algorithm. The black dot denotes the initial guess and the black star a local minimum. The red line represents the submanifold $(\boldsymbol{\xi}^H, \boldsymbol{\xi}^L)$ for which $\boldsymbol{\xi}^L$ is a solution to the least squares problem for fixed $\boldsymbol{\xi}^H$, written $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$, on which $\nabla_{\boldsymbol{\xi}}\mathcal{J} = (\nabla_{\boldsymbol{\xi}^H}\mathcal{J}, \mathbf{0})$.
Since the black star must also be a global minimum in $\boldsymbol{\xi}^L$, 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 $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$.



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

- LSGD ensures optimal representation of data in terms of the basis. Thus, we want the initial basis to have maximal rank.
- 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.



- Imagine a DNN with one hidden layer. From a C^0 finite element point of view, it is 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.

Box Initialization for Plain networks

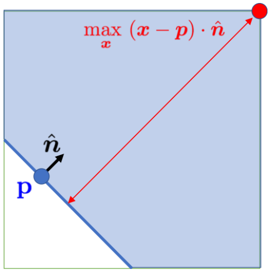

$\max_{\boldsymbol{x}} (\boldsymbol{x}-\boldsymbol{p})\cdot\hat{\boldsymbol{n}}$
$\hat{\boldsymbol{n}}$
p

Figure: Notation used in the "Box initialization" of each node. A random point $\boldsymbol{p}$ with random orientation $\hat{\boldsymbol{n}}$ is used to define a ReLU function of form $\sigma(k(\boldsymbol{x}-\boldsymbol{p})\cdot\hat{\boldsymbol{n}})$. 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 $\hat{\boldsymbol{n}}$, where the ReLU "switches on", as the *cut plane*.

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

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

$$\max_{\boldsymbol{x}\in[0,1]^w} \sigma(k(\boldsymbol{x}-\boldsymbol{p})\cdot\boldsymbol{n}) = 1. \tag{13}$$

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

Sandia National Laboratories

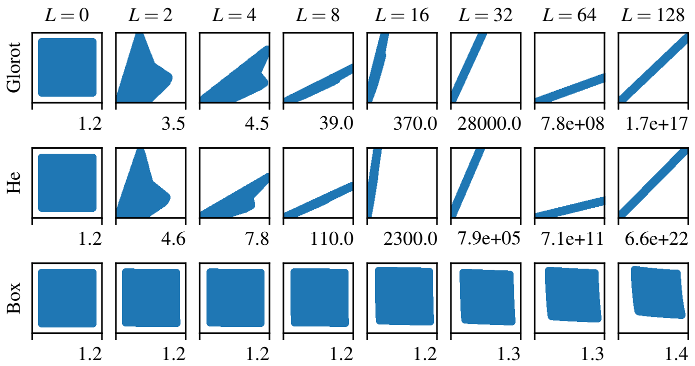
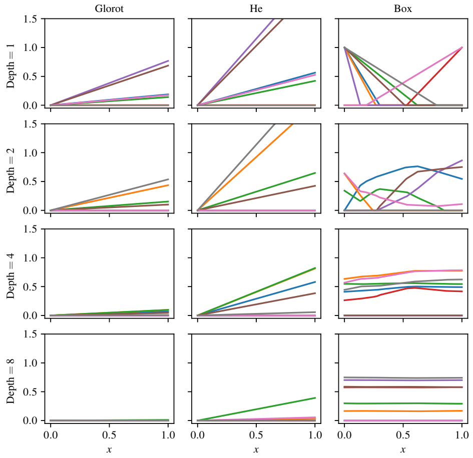


Figure: Images of the unit square $[0,1]^2$ under L initialized hidden layers of **ResNets** for Glorot (*top*), He (*center*) and Box (*bottom*) initializations. Values are presented on the square $[-0.2, H]^2$, where H is denoted to the bottom-right of each image. Collapse to a line through the origin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

Basis function plots for DNNs of width 8



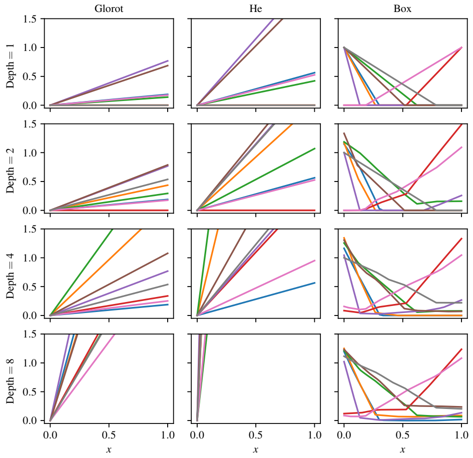


Figure: Left: Plain DNN, Right: ResNet.

Effect of Box initialization 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.

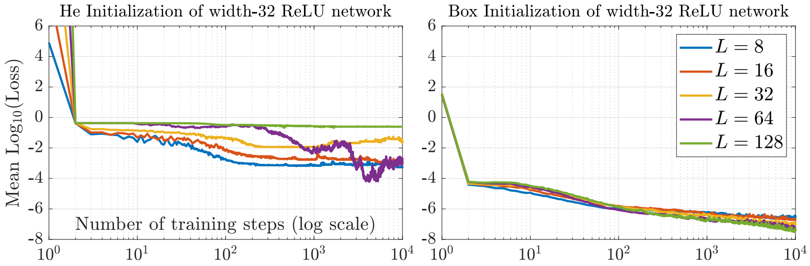


Figure: Mean of $\log_{10}(\text{Loss})$ over 16 training runs of residual width-32 ReLU network with $L = 8, 16, 32, 64$ and 128 hidden layers and training rate $2^{-(k+3)}$ for the He *(left)* and Box *(right)* initializations.

Application: PINN solver for Advection Eq.

- 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

$$\begin{aligned}\mathcal{J} &= \epsilon\mathcal{J}_1 + \mathcal{J}_2 + \mathcal{J}_3, \qquad \mathcal{J}_1 = \frac{1}{N_1}\sum_{i\in\mathcal{X}_1} |\partial_t \mathcal{NN}_i + \partial_x a(x,t)\mathcal{NN}_i|^2, \\ \mathcal{J}_2 &= \frac{1}{N_2}\sum_{i\in\mathcal{X}_2} |\mathcal{NN}_i(x,0) - u_0|^2, \qquad \mathcal{J}_3 = \frac{1}{N_3}\sum_{i\in\mathcal{X}_3} |\mathcal{NN}_i(0,t)|^2\end{aligned} \tag{14}$$

  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.

Sandia National Laboratories

Advection Equation with 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 ≥ 3, and at this point $\mathscr{J}_1 = \mathscr{J}_2 = \mathscr{J}_3 = 0$, rendering the choice of ϵ unimportant (we set $\epsilon = 1$).

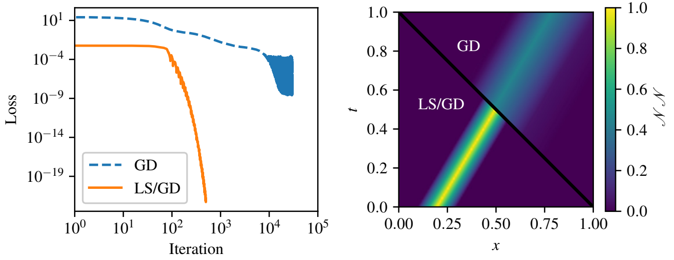
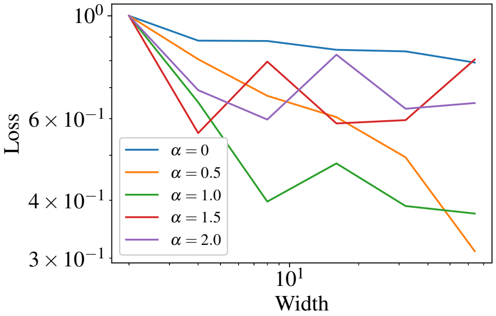


Figure: *Left:* Loss evolution over training for GD and LSGD. *Right:* Solution after 5000 iterations for GD and 500 iterations for LSGD. Setting: Box initialization, ReLU activation function, network width = 32, depth = 1, learning rate = 0.005.

Advection Equation with Nonconstant Velocity

- We next consider nonconstant velocity, $a(x,t) = x$, with corresponding analytic solution
$$u(x,t) = u_0(x \exp(-t)). \tag{15}$$
- 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.



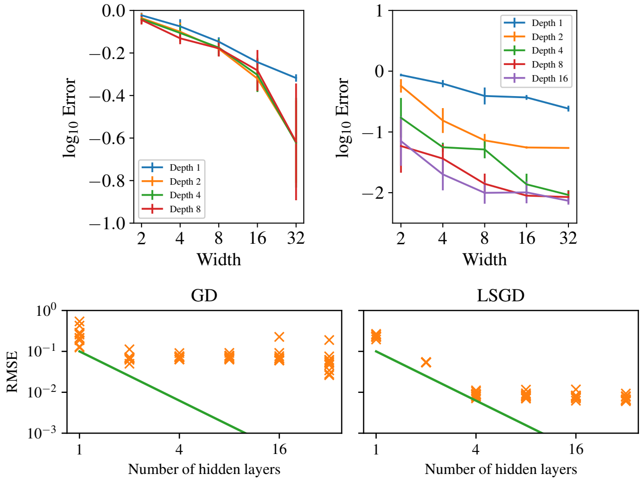


Figure: **Top:** Convergence with ReLU (left; learning rate 0.001) using $\alpha = -\frac{1}{2}$, and tanh (right; learning rate 0.01) using $\alpha = -\frac{1}{2}$. **Bottom:** Comparison of GD (*left*) and LSGD (*right*) training for tanh activation, learning rate 0.01, width 32 and 5000 steps. X's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

Part III: Partition of Unity Networks (POUnets)

The POUnet Architecture

- Using practical training methods, it is not possible to achieve *hp*-approximation rates using DNNs, despite what is theoretically possible.
- On the other hand, DNNs have proven ability to partition space in very high dimensions when used for classification problems.
- We propose partition of unity networks (POUnets) which incorporate *hp*-approximation directly into the architecture.
- Classification architectures of the type used to learn probability measures are used to build a meshfree partition of space, while polynomial spaces with learnable coefficients are associated to each partition.
- The resulting *hp*-element-like approximation allows use of a fast least-squares optimizer, and the resulting architecture size need not scale exponentially with spatial dimension, breaking the curse of dimensionality.

An abstract POU network

- Consider a *partition of unity* $\Phi = \{\phi_\alpha(\mathbf{x})\}_{\alpha=1}^{N_{\text{part}}}$ satisfying $\sum_\alpha \phi_\alpha(\mathbf{x}) = 1$ and $\phi_\alpha(\mathbf{x}) \geq 0$ for all $\mathbf{x}$. We work with the approximant

$$y_{\text{POU}}(\mathbf{x}) = \sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}) \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta} P_\beta(\mathbf{x}), \tag{16}$$

  where $V = \text{span}\,\{P_\beta\}$.
- For this work, we take V to be the space $\pi_m(\mathbb{R}^d)$ of polynomials of order m, while Φ is parametrized as a neural network with weights and biases $\boldsymbol{\xi}$ and output dimension N_{part}:

$$\phi_\alpha(\mathbf{x};\boldsymbol{\xi}) = \left[\mathcal{NN}(\mathbf{x};\boldsymbol{\xi})\right]_\alpha, \quad 1 \leq \alpha \leq N_{\text{part}}. \tag{17}$$

- We consider two architectures for $\mathcal{NN}(\mathbf{x};\boldsymbol{\xi})$ to be specified later. Approximants of the form (16) allow a "soft" localization of the basis elements P_β to an implicit partition of space parametrized by the ϕ_α.

Sandia National Laboratories

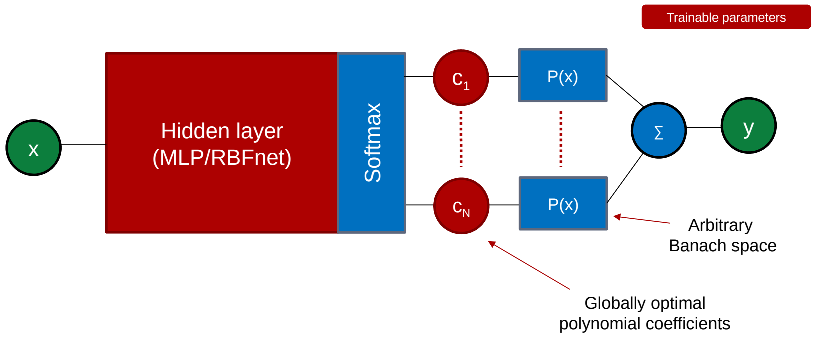


Figure: POUnet architecture.

Main idea: rather than attempt to emulate POU + monomials, build them directly into architecture

Training with POUnets

- While approximation with broken polynomial spaces corresponds to taking Φ to consist of characteristic functions on the cells of a computational mesh, the parametrization of Φ by a DNN generalizes more broadly to differentiable partitions of space.
- In a traditional numerical procedure, Φ is constructed prior to fitting $c_{\alpha,\beta}$ to data through a geometric "meshing" process. We instead work with a POU $\Phi^{\boldsymbol{\xi}}$ in the form of a DNN (17) in which the weights and biases $\boldsymbol{\xi}$, which are trained to fit the data.
- We therefore fit both the localized basis coefficients $\boldsymbol{c} = [c_{\alpha,\beta}]$ and the localization itself simultaneously by solving the optimization problem

$$\underset{\boldsymbol{\xi},\mathbf{c}}{\operatorname{argmin}} \sum_{i\in\mathcal{D}} \left| \sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}_i, \boldsymbol{\xi}) \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta} P_\beta(\mathbf{x}_i) - y_i \right|^2 . \tag{18}$$

POU-block Architecture

- A **shallow** RBF-network implementation of $\Phi_{\boldsymbol{\xi}}$ is given by (16) and

$$\phi_\alpha = \frac{\exp\left(-|x - \boldsymbol{\xi}_{1,\alpha}|^2 / \boldsymbol{\xi}_{2,\alpha}^2\right)}{\sum_\beta \exp\left(-|x - \boldsymbol{\xi}_{1,\beta}|^2 / \boldsymbol{\xi}_{2,\beta}^2\right)}. \tag{19}$$

- Here, $\boldsymbol{\xi}_1$ denotes the RBF centers and $\boldsymbol{\xi}_2$ denotes RBF shape parameters, both of which evolve during training. A measure of the localization of these functions can be taken to be the magnitude of $\boldsymbol{\xi}_1$.
- Such an architecture works well for approximation of smooth functions, but the C_∞ continuity of $\Phi_{\boldsymbol{\xi}}$ causes difficulty in the approximation of piecewise smooth functions.
- We also consider a **deep** architecture for $\Phi_{\boldsymbol{\xi}}$ given by a residual network architecture composed with a softmax layer $\mathcal{S}$ to define (17).

Optimal training error estimate

Consider an approximant y_{POU} of the form (16) with $V = \pi_m(\mathbb{R}^d)$. If $y(\cdot) \in C^{m+1}(\Omega)$ and $\boldsymbol{\xi}^*, \boldsymbol{c}^*$ solve (18) to yield the approximant y^*_{POU}, then

$$\|y^*_{\text{POU}} - y\|^2_{\ell_2(\mathcal{D})} \leq C_{m,y} \max_{\alpha} \operatorname{diam}\left(\operatorname{supp}(\phi^{\boldsymbol{\xi}}_{\alpha})\right)^{m+1} \tag{20}$$

where $\|y^*_{\text{POU}} - y\|_{\ell_2(\mathcal{D})}$ denotes the root-mean-square norm over the training data pairs in $\mathcal{D}$,

$$\|y^*_{\text{POU}} - y\|_{\ell_2(\mathcal{D})} = \sqrt{\frac{1}{N_{\text{data}}} \sum_{(\mathbf{x},y)\in\mathcal{D}} \left(y^*_{\text{POU}}(\mathbf{x}) - y(\mathbf{x})\right)^2}, \tag{21}$$

and

$$C_{m,y} = \|y\|_{C^{m+1}(\Omega)}. \tag{22}$$

Training

- The least-squares structure of (18) allows application of the least-squares gradient descent (LSGD) block coordinate descent strategy.
- To prevent learned partition functions ϕ_α from "collapseing" to near-zero values everywhere. we will also consider a pre-training step,, which adds an ℓ_2 regularizer to the polynomial coefficients.
- The intuition behind this is that a given partition regresses data using an element of the form $c_{\alpha,\beta}\phi_\alpha P_\beta$.
- If ϕ_α is scaled by a small $\delta > 0$, the LSGD solver may pick up a scaling $1/\delta$ for $c_{\alpha,\beta}$ and achieve the same approximation. Limiting the coefficients thus indirectly penalizes this mode of partition function collapse, promoting more quasi-uniform partitions of space.

Sandia National Laboratories

Two-phase algorithm

```
Data: $\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}$
Result: $\boldsymbol{\xi}_{\text{new}}, \mathbf{c}_{\text{new}}$
Function Two-phase-LSGD( $\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}},$
 $n_{\text{epoch}}, n_{\text{epoch}}^{\text{pre}}, \lambda, \rho, n_{\text{stag}}$ ):
    $\boldsymbol{\xi}, \mathbf{c} \leftarrow$ LSGD$(\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}, n_{\text{epoch}}^{\text{pre}}, \lambda, \rho, n_{\text{stag}})$ ;
     // Phase 1: LSGD with a regularizer
    $\boldsymbol{\xi}_{\text{new}}, \mathbf{c}_{\text{new}} \leftarrow$ LSGD$(\boldsymbol{\xi}, \mathbf{c}, n_{\text{epoch}}, 0, 0, n_{\text{epoch}})$ ;
     // Phase 2: LSGD without a
     regularizer
```

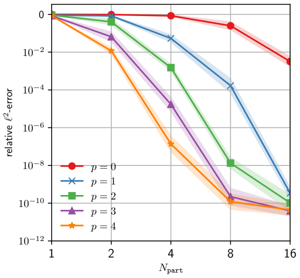
Algorithm 2: The two-phase LSGD method with the ℓ^2-regularized least-squares solves.

Smooth example using RBFnets for POU

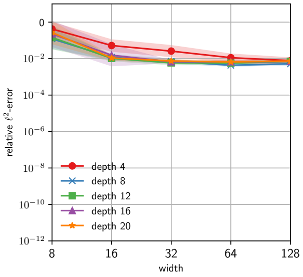
We consider an analytic function as our first benchmark, specifically the sine function defined on a cross-shaped one-dimensional manifold embedded in $[-1,1]^2$

$$\mathbf{y}(\mathbf{x}) = \begin{cases} \sin(2\pi x_1), & \text{if } x_2 = 0, \\ \sin(2\pi x_2), & \text{if } x_1 = 0. \end{cases}$$

We test RBF-Nets for varying number of partitions, $N_{\text{part}} = \{1,2,4,8,16\}$ and the maximal polynomial degrees $\{0,1,2,3,4\}$. For training, we collect data $x_i, i = 1,2$ by uniformly sampling 501 $\{((x_1,x_2),\mathbf{y}(\mathbf{x})\}$-pairs on each axis. We initialize centers of the RBF basis functions by sampling uniformly from the domain $[-1,1]^2$ and initialize shape parameters as ones.



(a) POUnets



(b) MLPs

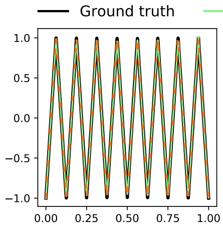
Piecewise smooth functions

- We next consider piecewise linear and piecewise quadratic functions: triangle waves with varying frequencies, i.e., $\mathbf{y}(x) = \mathrm{TRI}(x; p)$, and their quadratic variants $\mathbf{y}(x) = \mathrm{TRI}^2(x; p)$, where

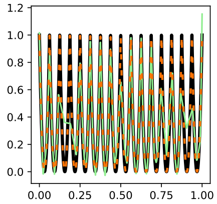
$$\mathrm{TRI}(x; p) = 2\left|px - \left\lfloor px + \frac{1}{2} \right\rfloor\right| - 1. \tag{23}$$

- We study the introduction of increasingly many discontinuities by increasing the frequency $p = \{1, 2, 3, 4, 5\}$, which results in piecewise linear and quadratic functions with 2^p pieces.
- Based on the number of pieces in the target function, we scale the width of the baseline neural networks and POUnets as 4×2^p, while fixing the depth as 8, and for POUnets the number of partitions are set as $N_{\text{part}} = 2^p$.
- For POUnets, we choose the maximal degree of polynomials to be $m_{\max} = 1$ and $m_{\max} = 2$ for the piecewise linear and quadratic target functions, respectively.
- Reproduction of such sawtooth functions by ReLU networks via both wide networks and very deep networks can be has been discussed theoretically, but to our knowledge has not been achieved via standard training.

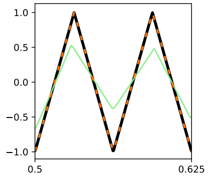
Sandia National Laboratories



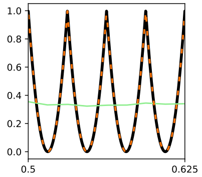
(a) $N_{\text{part}} = 16$



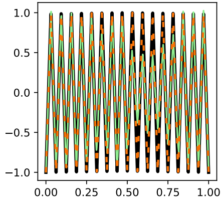
(b) $N_{\text{part}} = 16$



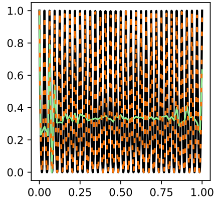
(e) $N_{\text{part}} = 32$ (zoomed-in)



(f) $N_{\text{part}} = 32$ (zoomed-in)



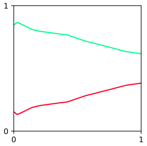
(c) $N_{\text{part}} = 32$

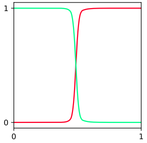


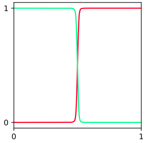
(d) $N_{\text{part}} = 32$

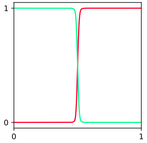
Snapshots of target functions $\mathbf{y}(\mathbf{x})$ and approximants produced by ResNet and POUnet (i.e., $y_{\text{POU}}(\mathbf{x})$) are depicted in black, light green, and orange, respectively. The target function correspond to triangular waves (left) and their quadratic variants (right). The bottom row depicts the snapshots in the domain [0.5,0.625].

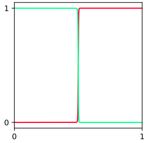
Two-phase training (quadratic waves)

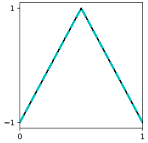

(g) P1 (0)

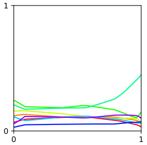

(h) P1 (15)

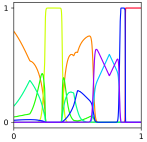

(i) P1 (30)

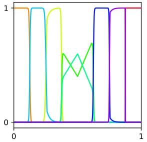

(j) P1 (60)

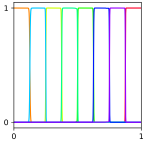

(k) P2 (1000)

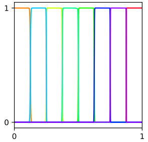

(l) Approx.


(m) P1 (0)


(n) P1 (30000)


(o) P1 (60000)


(p) P1 (90000)


(q) P2 (1000)

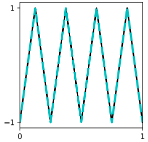

(r) Approx.

Figure: Triangular wave with two pieces (top) and triangular wave with eight pieces (bottom): Phase 1 LSGD constructs localized disjoint partitions and Phase 2 LSGD produces an accurate approximation.

Title

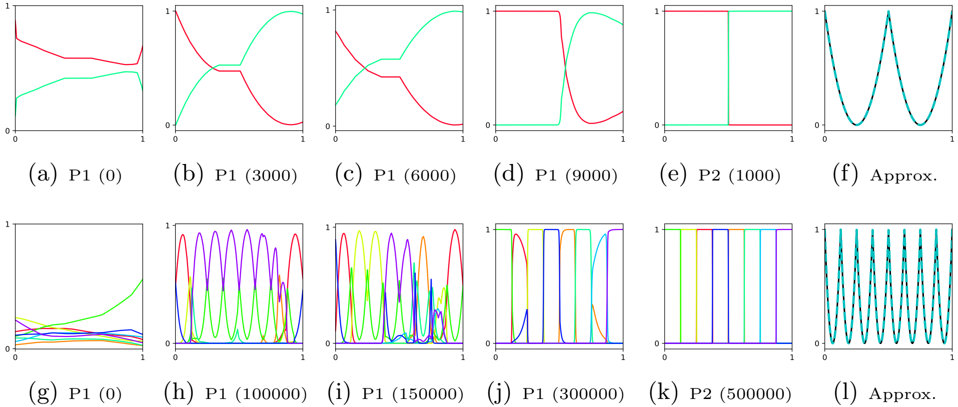


Figure: Quadratic wave with two pieces (top) and quadratic wave with eight pieces (bottom): Phase 1 LSGD constructs localized disjoint partitions and Phase 2 LSGD produces an accurate approximation.

Acknowledgements

- The work of R. Patel, K. Lee, 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.

Figure: Ravi, Kookjin, Nat, Mauro, and Eric.

- 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!