# Partition of unity networks: deep hp-approximation

**Kookjin Lee**[1], **Nathaniel A. Trask**[2], **Ravi G. Patel**[2], **Mamikon A. Gulian**[2], **Eric C. Cyr**[2]
[1] Extreme Scale Data Science & Analytics Department, Sandia National Laboratories
[2]Center for Computing Research, Sandia National Laboratories
Albuquerque, New Mexico, 87123
{koolee, natrask, rgpatel, mgulian, eccyr}@sandia.gov

## Abstract

Approximation theorists have established best-in-class optimal approximation rates of deep neural networks by utilizing their ability to simultaneously emulate partitions of unity and monomials. Motivated by this, we propose partition of unity networks (POUnets) which incorporate these elements 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 approximation result establishes desirable properties to guide network design. Numerical results for two choices of architecture demonstrate that POUnets yield $hp$-convergence for smooth functions and consistently outperform MLPs for piecewise polynomial functions with large numbers of discontinuities.

## Overview

We consider regression over the set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{data}}}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i = y(\mathbf{x}_i)$ are point samples of a piecewise smooth function. Following successes in classification problems in high-dimensional spaces (Chollet 2017), deep neural networks (DNNs) have garnered tremendous interest as tools for regression problems and numerical analysis, partially due to their apparent ability to alleviate the *curse of dimensionality* in the presence of latent low-dimensional structure. This is in contrast to classical methods for which the computational expense grows exponentially with $d$, a major challenge for solution of high-dimensional PDEs (Bach 2017; Bengio and Bengio 2000; Han, Jentzen, and Weinan 2018).

Understanding the performance of DNNs requires accounting for both optimal approximation error and optimization error. While one may prove existence of DNN parameters providing exponential convergence with respect to architecture size, in practice a number of issues conspire to prevent realizing such convergence. Several approximation theoretic works seek to understand the role of width/depth in the absence of optimization error (He et al. 2018; Daubechies et al. 2019; Yarotsky 2017, 2018; Opschoor, Petersen, and Schwab 2019). In particular, Yarotsky and Opschoor et al. prove the existence of parameters for a deep neural network architecture that approximate algebraic operations, partitions of unity (POUs), and polynomials to exponential accuracy in the depth of the network. This shows that sufficently deep DNNs may in theory learn a spectrally convergent $hp$-element space without a hand-tailored mesh by constructing a POU to localize polynomial approximation. In practice, however, such convergent approximations are not realized when training DNNs using gradient descent optimizers, even for smooth target functions (Fokina and Oseledets 2019; Adcock and Dexter 2020). The thesis of this work is to incorporate the POU and polynomial elements directly into a deep learning architecture. Rather than attempting to force a DNN to simultaneously perform localization and high-order approximation, introducing localized polynomial spaces frees the DNN to play to its strengths by focusing exclusively on partitioning space, as in classification problems.

An attractive property of the proposed architecture is its amenability to a fast training strategy. In previous work, we developed an optimizer which alternates between a gradient descent update of hidden layer parameters and a globally optimal least squares solve for a final linear layer (Cyr et al. 2019); this was applied as well to classification problems (Patel et al. 2020). A similar strategy is applied in the current work: updating the POU with gradient descent before finding a globally optimal polynomial fit at each iteration ensures an optimal representation of data over the course of training.

While DNNs have been explored as a means of solving high-dimensional PDEs (Geist et al. 2020; Han, Jentzen, and Weinan 2018), optimization error prevents a practical demonstration of convergence with respect to size of either data or model parameters (Beck, Jentzen, and Kuckuck 2019; Wang, Teng, and Perdikaris 2020). The relatively simple regression problem considered here provides a critical first example of how the optimization error barrier may be circumvented to provide accuracy competitive with finite element methods.

## 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

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

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 $\Phi$ is parametrized as a neural network with weights and biases $\boldsymbol{\xi}$ and output dimension $N_{\text{part}}$:

$$\phi_\alpha(\mathbf{x}; \boldsymbol{\xi}) = \left[\mathcal{NN}(\mathbf{x}; \boldsymbol{\xi})\right]_\alpha, \quad 1 \leq \alpha \leq N_{\text{part}}. \quad (2)$$

We consider two architectures for $\mathcal{NN}(\mathbf{x}; \boldsymbol{\xi})$ to be specified later. Approximants of the form (1) allow a "soft" localization of the basis elements $P_\beta$ to an implicit partition of space parametrized by the $\phi_\alpha$. While approximation with broken polynomial spaces corresponds to taking $\Phi$ to consist of characteristic functions on the cells of a computational mesh, the parametrization of $\Phi$ by a DNN generalizes more broadly to differentiable partitions of space. For applications of partitions of unity in numerical analysis, see Strouboulis, Copps, and Babuška (2001); Fries and Belytschko (2010); Wendland (2002); for their use in differential geometry, see Spivak (2018); Hebey (2000).

In a traditional numerical procedure, $\Phi$ 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 (2) 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

$$\underset{\boldsymbol{\xi},\mathbf{c}}{\text{argmin}} \sum_{i \in \mathcal{D}} \left| \sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}_i, \boldsymbol{\xi}) \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta} P_\beta(\mathbf{x}_i) - y_i \right|^2. \quad (3)$$

## Error analysis and architecture design

Before specifying a choice of architecture for $\Phi^{\boldsymbol{\xi}}$ in (2), we present a basic estimate of the optimal training error using the POU-Net architecture to highlight desirable properties for the partition to have. We denote by $\text{diam}(A)$ the diameter of a set $A \subset \mathbb{R}^d$.

**Theorem 1.** *Consider an approximant $y_{POU}$ of the form* (1) *with $V = \pi_m(\mathbb{R}^d)$. If $y(\cdot) \in C^{m+1}(\Omega)$ and $\boldsymbol{\xi}^*, \boldsymbol{c}^*$ solve* (3) *to yield the approximant $y^*_{POU}$, then*

$$\|y^*_{POU} - y\|^2_{\ell_2(\mathcal{D})} \leq C_{m,y} \max_\alpha \mathit{diam}\left(\mathit{supp}(\phi^{\boldsymbol{\xi}}_\alpha)\right)^{m+1} \quad (4)$$

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

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

*and*

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

*Proof.* For each $\alpha$, take $q_\alpha \in \pi_m(\mathbb{R}^d)$ to be the $m$th order Taylor polynomial of $y(\cdot)$ centered at any point of $\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)$. Then for all $\mathbf{x} \in \text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)$,

$$|q_\alpha(\mathbf{x}) - y(\mathbf{x})| \leq C_{m,y} \,\text{diam}\left(\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)\right)^{m+1}. \quad (5)$$

Define the approximant $\widetilde{y}_{\text{POU}} = \sum_{\alpha=1}^{N_{\text{part}}} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) q_\alpha(\mathbf{x})$, which is of the form (1) and represented by feasible $(\boldsymbol{\xi}, \boldsymbol{c})$. Then by definition of $y^*_{\text{POU}}$ and (3), we have

$$\begin{aligned}\|y^*_{\text{POU}}(\mathbf{x}) - y(\mathbf{x})\|^2_{\ell_2(\mathcal{D})} &\leq \|\widetilde{y}_{\text{POU}}(\mathbf{x}) - y(\mathbf{x})\|^2_{\ell_2(\mathcal{D})} \\ &= \left\| \sum_{\alpha=1}^{N_{\text{part}}} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) q_\alpha(\mathbf{x}) - y(\mathbf{x}) \sum_{\alpha=1}^{N_{\text{part}}} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) \right\|^2_{\ell_2(\mathcal{D})} \\ &= \left\| \sum_{\alpha=1}^{N_{\text{part}}} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) \left(q_\alpha(\mathbf{x}) - y(\mathbf{x})\right) \right\|^2_{\ell_2(\mathcal{D})}.\end{aligned}$$

For each $\mathbf{x} = \mathbf{x}_i \in \mathcal{D}$, if $\mathbf{x} \in \text{supp}(\mathcal{D})$, then we apply (5); otherwise, the summand $\phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) \left(q_\alpha(\mathbf{x}) - y(\mathbf{x})\right)$ vanishes. So

$$\begin{aligned}&\|y^*_{\text{POU}}(\mathbf{x}) - y(\mathbf{x})\|^2_{\ell_2(\mathcal{D})} \\ &\leq \left\| \sum_{\alpha=1}^{N_{\text{part}}} C_{m,y} \,\text{diam}\left(\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)\right)^{m+1} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) \right\|^2_{\ell_2(\mathcal{D})} \\ &\leq C_{m,y} \max_\alpha \text{diam}\left(\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)\right)^{m+1} \left\| \sum_{\alpha=1}^{N_{\text{part}}} \phi^{\boldsymbol{\xi}}_\alpha(\mathbf{x}) \right\|^2_{\ell_2(\mathcal{D})} \\ &\leq C_{m,y} \max_\alpha \text{diam}\left(\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)\right)^{m+1}.\end{aligned}$$

Theorem 1 indicates that for smooth $y$, the resulting convergence rate of the training error is independent of the specific choice of parameterization of the $\phi^{\boldsymbol{\xi}}_\alpha$, and depends only upon their support. Further, the error does not explicitly depend upon the dimension of the problem; if the trained $\Phi^{\boldsymbol{\xi}}$ encodes a covering by $\text{supp}(\phi^{\boldsymbol{\xi}}_\alpha)$ of a low dimensional manifold containing the data locations $\mathbf{x}$ with latent dimension $d_{\mathcal{M}} \ll d$, then the approximation cost scales only with $\dim(V)$ and thus may break the curse of dimensionality, e.g. for linear polynomial approximation $\dim(V) = d + 1$. If the parameterization is able to find compactly supported quasi-uniform partitions such that the maximum diameter in (4) scales as $N_{\text{part}}^{-1/d_{\mathcal{M}}}$, the training loss will exhibit a scaling of $N_{\text{part}}^{-(m+1)/d_{\mathcal{M}}}$.

The above analysis indicates the advantage of enforcing highly localized, compact support of the POU functions. However, in practice we found that for a POU parametrized by a shallow RBF-Net networks (described below), rapidly decaying but globally supported POU functions exhibited more consistent training results. This is likely due to a higher tolerance to initializations poorly placed with respect to the data locations. Similarly, when the POU is parametrized by a deep ResNet, we obtained good results using ReLU activation functions while training with a regularizer to promote localization (Algorithm 2). Thus, the properties playing a role in the above analysis – localization of the partition functions and distribution of their support over a latent manifold containing the data – are the motivation for the architectures and training strategies we consider below.

**POU #1 - RBF-Net:** A shallow RBF-network (Broomhead and Lowe 1988; Billings and Zheng 1995) implementation of $\Phi_{\boldsymbol{\xi}}$ is given by (1) and

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

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 $C_\infty$ continuity of $\Phi_{\boldsymbol{\xi}}$ causes difficulty in the approximation of piecewise smooth functions.

**POU #2 - ResNet:** We compose a residual network architecture (He et al. 2016) with a softmax layer $\mathcal{S}$ to define (2). For the experiments considered we use a ReLU activation, allowing representation of functions with with discontinuities in their first derivative.

**Initialization:** All numerical experiments take data supported within the unit hypercube $\Omega \subset [0,1]^d$. To initialize the POU #1 architecture we use unit shape parameter and uniformly distribute centers $\mathbf{x}_1 \sim \mathcal{U}([0,1]^d)$. We initialize POU #2 with the Box initialization (Cyr et al. 2019). We found that these initializations provide an initial set of partitions sufficiently "well-distributed" throughout $\Omega$ for successful training.

## Fast optimizer

The least-squares structure of (3) allows application of the least-squares gradient descent (LSGD) block coordinate descent strategy (Cyr et al. 2019). At each epoch, one may hold the hidden parameters $\boldsymbol{\xi}$ fixed to obtain a least squares problem for the optimal polynomial coefficients $\mathbf{c}$. The step may be concluded by then taking a step of gradient descent for the POU parameters, therefore evolving the partitions along the manifold corresponding to optimal representation of data; for details see (Cyr et al. 2019).

Algorithm 1 with $\lambda = 0$ specifies the application of LSGD to Eqn. (3). We will demonstrate that while effective, several of the learned partition functions $\phi_\alpha$ may "collapse" to near-zero values everywhere. To remedy this, we will also consider a pre-training step in Algorithm 2, which adds an $\ell_2$ 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 $\phi_\alpha$ 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.

```
Data: $\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}$
Result: $\boldsymbol{\xi}_{\text{new}}, \mathbf{c}_{\text{new}}$
Function LSGD ( $\boldsymbol{\xi}_{\text{old}}$, $\mathbf{c}_{\text{old}}$, $n_{\text{epoch}}$, $\lambda$, $\rho$, $n_{\text{stag}}$ ):
    $\boldsymbol{\xi}, \mathbf{c} \leftarrow \boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}$;
    for $i \in \{1, ..., n_{\text{epoch}}\}$ do
        $\mathbf{c} \leftarrow \text{LS}(\boldsymbol{\xi}, \lambda)$ ; // solve Eqn. (1) with a regularizer $\lambda\|\mathbf{c}\|_F^2$
        $\boldsymbol{\xi} \leftarrow \text{GD}(\mathbf{c})$;
        if LSGD stagnates more than $n_{\text{stag}}$ then
            $\lambda \leftarrow \rho\lambda$
        end
    end
    $\mathbf{c}_{\text{new}} \leftarrow \text{LS}(\boldsymbol{\xi}, 0)$;
    $\boldsymbol{\xi}_{\text{new}} \leftarrow \boldsymbol{\xi}$;
```

**Algorithm 1:** The regularized least-squares gradient descent (LSGD) method. Setting $\lambda = 0$ recovers the original LSGD method (Cyr et al. 2019).

```
Data: $\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}$
Result: $\boldsymbol{\xi}_{\text{new}}, \mathbf{c}_{\text{new}}$
Function Two-phase-LSGD ( $\boldsymbol{\xi}_{\text{old}}$, $\mathbf{c}_{\text{old}}$,
  $n_{\text{epoch}}$, $n^{\text{pre}}_{\text{epoch}}$, $\lambda$, $\rho$, $n_{\text{stag}}$ ):
    $\boldsymbol{\xi}, \mathbf{c} \leftarrow$ LSGD($\boldsymbol{\xi}_{\text{old}}, \mathbf{c}_{\text{old}}, n^{\text{pre}}_{\text{epoch}}, \lambda, \rho, n_{\text{stag}}$) ;
     // 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 $\ell^2$-regularized least-squares solves.

## Numerical experiments

In this section, we assess the performance of POUnets with the two POU architectures. We implement the proposed algorithms in PYTHON, construct POUnets using TENSORFLOW 2.3.0 (Abadi et al. 2016), and employ NUMPY and SCIPY packages for generating training data and solving the least squares problem. For all considered neural networks, training is performed by batch gradient descent using the Adam optimizer (Kingma and Ba 2014). For a set of $d$-variate polynomials $\{P_\beta\}$, we choose truncated Taylor polynomials of maximal degree $m_{\text{max}}$, providing to $\frac{(m_{\max}+d)!}{(m_{\max}!)(d!)}$ polynomial coefficients per partition. The coefficients are initialized via sampling from the unit normal distribution.

### Smooth functions

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

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

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 from the domain [-1,1] for 501 samples, i.e., in total, 1001 $\{((x_1, x_2), \mathbf{y}(\mathbf{x})\}$-pairs after removing the duplicate point on the origin. We initialize centers of the RBF basis functions by sampling uniformly from the domain $[-1, 1]^2$ and initialize shape parameters as ones. We then train RBF-Nets by using the LSGD method with $\lambda = 0$ (Algorithm 1) with the initial learning rate for Adam set to $10^{-3}$. The maximum number of epochs $n_{\text{epoch}}$ is set as 100, and we choose the centers and shape parameters that yield the best result on the training loss during the specified epochs.

Figure 1(a) reports the relative $\ell^2$-errors of approximants produced by POUnets for varying $N_{\text{part}}$ and varying $p$. The results are obtained from 10 independent runs for each value of $N_{\text{part}}$ and $p$, with a single log-normal standard deviation. Algebraic convergence is observed of order increasing with polynomial degree before saturating at $10^{-10}$. We conjecture this is due to eventual loss of precision due to the ill-conditioning of the least squares matrix; however, we leave a formal study and treatment for a future work.

In comparison to the performance achieved by POUnets, we assess the performance of the standard MLPs with varying depth {4,8,12,16,20} and width {8,16,32,64,128}. The standard MLPs are trained by using Adam with an initial learning rate $10^{-3}$ and the maximum number of epochs 1000. As opposed to the convergence behavior of RBF-Net-based POUnets, the standard MLP briefly exhibits roughly first order convergence before stagnating around an error of $10^{-2}$.

## Piecewise Smooth Functions

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

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

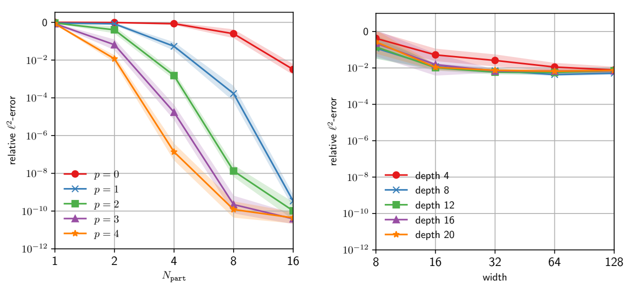
We study the introduction of increasingly many discontinuities by increasing the frequency $p$. Reproduction of such sawtooth functions by ReLU networks via both wide networks (He et al. 2018) and very deep networks (Telgarsky 2016) can be achieved theoretically via construction of carefully chosen architecture and weights/biases, but to our knowledge has not been achieved via standard training.

The smoothness possessed by the RBF-Net partition functions (POU #1) precludes rapidly convergent approximation of piecewise smooth functions, and we instead employ ResNets (POU #2) for $\Phi^{\xi}$ in this case, as the piecewise linear nature of such partition functions is better suited. As a baseline for the performance comparison, we apply regression of the same data over a mean square error using standard ResNets, $y_{\text{MLP}}(x)$. The standard ResNets share the same architectural design and hyperparameters with the ResNets used to parametrize the POU functions in the POUnets. The only exception is the output layer; the standard ResNets produce a scalar-valued output, whereas the POU parametrization produces a vector-valued output, which is followed by the softmax activation.

We consider five frequency parameters for both target functions, $p = \{1, 2, 3, 4, 5\}$, 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 $4 \times 2^p$, 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.

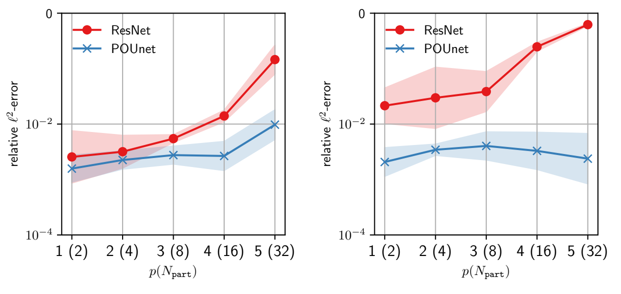
Both neural networks are trained on the same data, $\{x_i, \mathbf{y}(x_i; p)\}_{i=1}^{n_{\text{data}}}$, where $x_i$ are uniformly sampled from [0,1] and $n_{\text{data}} = 2000$. The standard ResNets are trained using the gradient descent method and the POUnets are trained using the LSGD method with $\lambda = 0$ (Algorithm 1). The initial learning rate for Adam is set as $10^{-3}$. The maximum number of epochs $n_{\text{epoch}}$ is set as 2000 and we choose the neural network weights and biases that yield the best result on the training loss during the specified epochs.

Figure 2 reports the relative $\ell^2$-errors of approximants produced by the standard ResNets and POUnets for varying $N_{\text{part}}$ and width for the piecewise linear functions (left) and the quadratic piecewise quadratic functions (right). The statistics are obtained from five independent runs. Figure 2 essentially



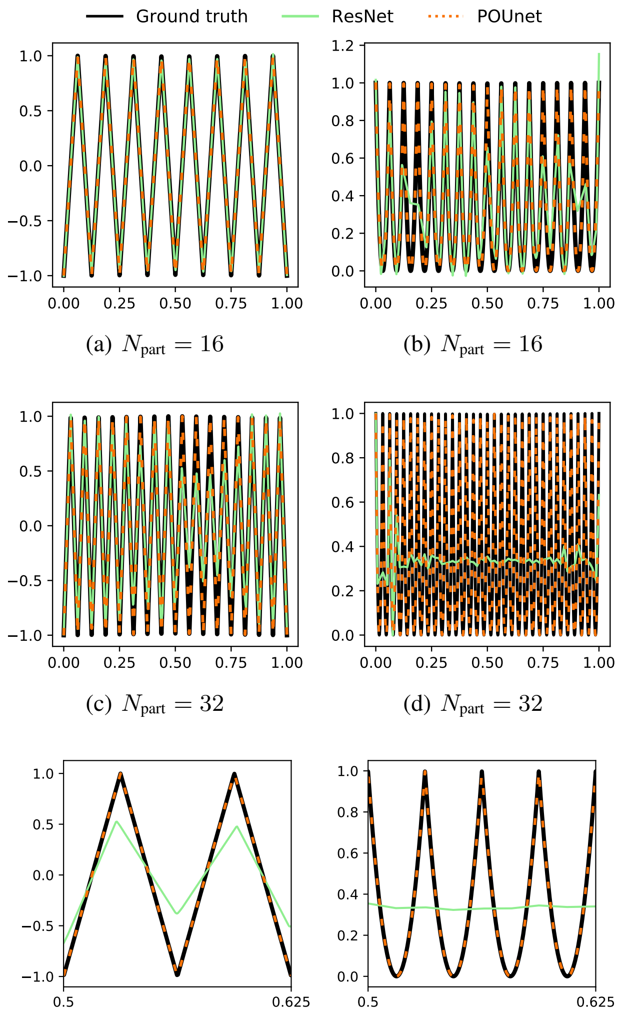
(a) POUnets (b) MLPs

Figure 1: Relative $\ell^2$-errors (log-log scale) of approximants produced by POUnets with RBF-Net partition functions for varying $N_{\text{part}}$ and varying $m_{\max}$ (left) and standard MLPs for varying width and depth (right).



(a) Triangular waves (b) Quadratic waves

Figure 2: Relative $\ell^2$-errors (log-log scale) of approximants produced by the standard ResNets and POUnets with ResNet partition functions for varying $N_{\text{part}}$ for approximating the target functions with varying $p$.



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

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

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

Figure 3: 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].

shows that the POUnets outperform the standard ResNets in terms of approximation accuracy; specifically, the standard ResNets with the ReLU activation function significantly fails to produce accurate approximantions, while POUnets obtain $< 1\%$ error for large numbers of discontinuities.

Figure 3 illustrates snapshots of the target functions and approximants produced by the two neural networks for target functions with the frequency parameter $p = \{4, 5\}$. Figure 3 confirms the trends shown in Figure 2; the benefits of using POUnets are more pronounced in approximating functions with larger $p$ and in approximating quadratic functions (potentially, in approximating high-order polynomials). Figures 3(c)–3(f) clearly show that the POUnets accurately approximate the target functions with sub $1\%$ errors, while the standard ResNets significantly fails to produce accurate approximations.

## Results of two-phase LSGD

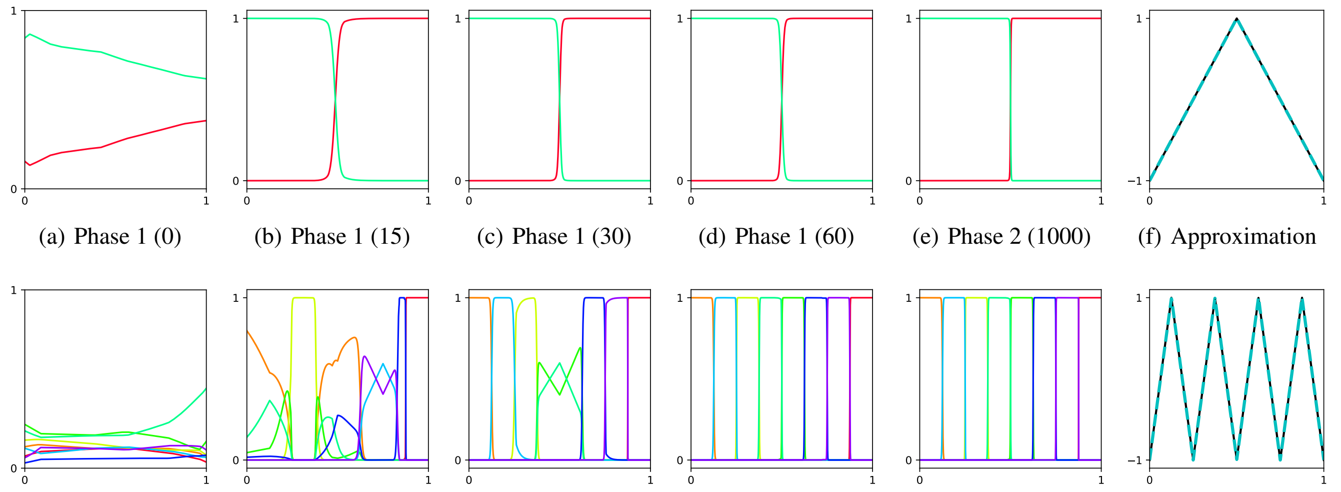
Now we demonstrate effectiveness of the two-phase LSGD method (Algorithm 2) in constructing partitions which are localized according to the features in the data and nearly disjoint, i.e., breakpoints in the target function, which we found leads to better approximation accuracy.

The first phase of the algorithm aims to construct such partitions. To this end, we limit the expressibility of the polynomial regression by regularizing the Frobenius norm of the coefficients $\|\mathbf{c}\|_F^2$ in least-squares solves. This prevents a small number of partitions dominating and other partitions being collapsed per our discussion of the fast optimizer above. These "quasi-uniform" partition are then used as the initial guess for the unregularized second phase. Finally, we study the qualitative differences in the resulting POU, particularly the ability of the network to learn a disjoint partition of space. We do however stress that there is no particular "correct" form for the resulting POU - this is primarily of interest because it shows the network may recover a traditional discontinuous polynomial approximation.

In the first phase, we employ a relatively larger learning rate in order to prevent weights and biases from getting stuck in local minima. On the other hand, in the second phase we employ a relatively smaller learning rate to ensure that the training error decreases.

**Triangular waves.** We demonstrate how the two-phase LSGD works in two example cases. The first set of example problems is the triangular wave with the frequency parameter $p = 1, 3$. We use the same POU network architecture described in the previous section (i.e., ResNet with width 8 and depth 8) and the same initialization scheme (i.e., the box initialization). We set 0.1 and 0.05 for the initial learning rates for phase 1 and the phase 2, respectively; we set the other LSGD parameters as $\lambda = 0.1$, $\rho = 0.9$, and $n_{\text{stag}} = 1000$. Figure 4 (top) depicts both how partitions are constructed during phase 1 (Figures 4(a)–4(d)), and snapshots of the partitions and the approximant at 1000th epoch of the phase 2 in Figures 4(e) and 4(f). The approximation accuracy measured in the relative $\ell^2$-norm of the error reaches down to $6.2042 \times 10^{-8}$ after 12000 epochs in phase 2.

Next we consider the triangular wave with the frequency parameter $p = 3$. We use the POU network architecture constructed as ResNet with width 8 and dept 10, and use the box initialization. We set 0.05 and 0.01 for the initial learning rates for phase 1 and phase 2, respectively; we set the other LSGD parameters as $\lambda = 0.1$, $\rho = 0.999$, and $n_{\text{stag}} = 1000$. Figure 4 (bottom) depicts again how partitions evolve during the phase 1 and the resulting approximants. Figures 4(g)–4(j) depicts that the two-phase LSGD constructs partitions that are disjoint and localized according to the features during the first phase.



(a) Phase 1 (0) (b) Phase 1 (15) (c) Phase 1 (30) (d) Phase 1 (60) (e) Phase 2 (1000) (f) Approximation

(g) Phase 1 (0) (h) Phase 1 (30000) (i) Phase 1 (60000) (j) Phase 1 (90000) (k) Phase 2 (1000) (l) Approximation

Figure 4: Triangular wave with two pieces (top) and triangular wave with eight pieces (bottom): Phase 1 LSGD constructs localized disjoint partitions (4(a)–4(d) and 4(g)–4(j)) and Phase 2 LSGD produces an accurate approximation.

**Quadratic waves.** Finally, we approximate the piecewise quadratic wave with frequency parameter $p = 1, 3$ while employing the same network architecture used for approximating the triangular waves. For the $p = 1$ case the learning rate set as 0.5 and 0.25 for phase 1 and phase 2. We use the same parameter settings (i.e., $\lambda = 0.1$, $\rho = 0.9$, and $n_{\text{stag}} = 1000$) as in the previous experiment. Again, in phase 1 disjoint partitions are constructed, and the accurate approximation is produced in the phase 2 (Figures 5(a)–5(f)). Moreover, we observe that the partitions are further refined during phase 2. For the $p = 3$ case we again employ the architecture used for the triangular wave with $p = 3$ (i.e., ResNet with width 8 and depth 10). We use the same hyperparameters as in the triangular wave with $p = 3$ (i.e., 0.05 and 0.01 for learning rates, and $n_{\text{stag}} = 1000$). The two exceptions are the weight for the regularization, $\lambda = 1$ and $\rho = 0.999$. Figures 5(g)–5(k) illustrates that the phase 1 LSGD constructs disjoint supports for partitions, but takes much more epochs. Again, the two-stage LSGD produces an accurate approximation (Figure 5(l)).

**Discussion.** The results of this section have demonstrated that it is important to apply a good regularizer during a pretraining stage to obtain approximately disjoint piecewise constant partitions. For the piecewise polynomial functions considered here, such partitions allowed in some cases recovery to near-machine precision. Given the abstract error analysis, it is clear that such localized partitions which adapt to the features of the target function act similarly to traditional $hp$-approximation. There are several challenges regarding this approach, however: the strong regularization during phase 1 requires a large number of training epochs, and we were unable to obtain a set of hyperparameters which provide such clean partitions across all cases. This suggests two potential areas of future work. Firstly, an improved means of regularizing during pretraining that does not compromise accuracy may be able to train faster; deep learning strategies for parametrizing charts as in (Schonsheck, Chen, and Lai 2019) may be useful for this. Secondly, it may be fruitful to understand the approximation error under less restrictive assumptions that the partitions are compactly supported or highly localized. We have been guided by the idea that polynomials defined on indicator function partitions reproduce the constructions used by the finite element community, but a less restrictive paradigm combined with an effective learning strategy may prove more flexible for general datasets.

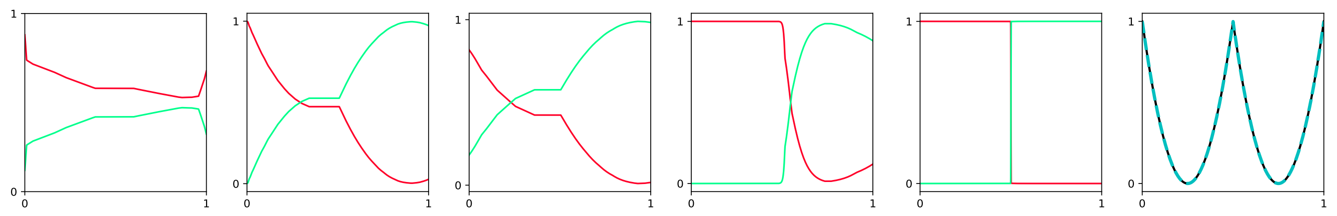
## Conclusions

Borrowing ideas from numerical analysis, we have demonstrated an novel architecture and optimization strategy the computational complexity of which need not scale exponentially with the ambient dimension and which provides high-order convergence for smooth functions and error consistently under $1\%$ for piecewise smooth problems. Such an architecture has the potential to provide DNN methods for solving high-dimensional PDE that converge in a manner competitive with traditional finite element spaces.

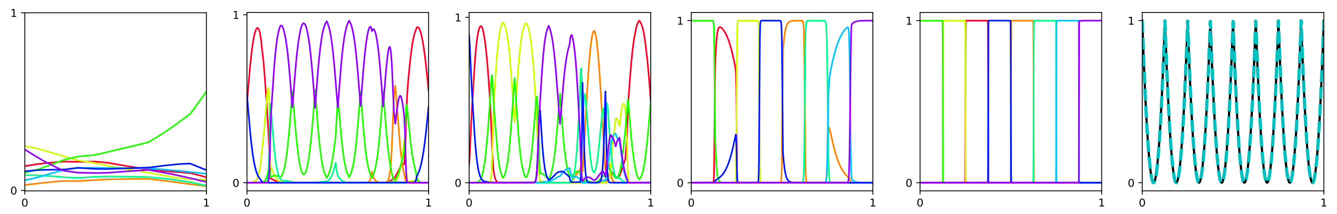
## Acknowledgements

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 Energy's National Nuclear Security Administration under contract DE-NA0003530. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. SAND Number: SAND2020-6022 J.

The work of M. Gulian, R. Patel, and N. Trask are supported by the U.S. Department of Energy, Office of Advanced



(a) Phase 1 (0) (b) Phase 1 (3000) (c) Phase 1 (6000) (d) Phase 1 (9000) (e) Phase 2 (1000) (f) Approximation



(g) Phase 1 (0) (h) Phase 1 (100000) (i) Phase 1 (150000) (j) Phase 1 (300000) (k) Phase 1 (500000) (l) Approximation

Figure 5: Quadratic wave with two pieces (top) and quadratic wave with eight pieces (bottom): Phase 1 LSGD constructs localized disjoint partitions (5(a)–5(d) and 5(g)–5(j)) and Phase 2 LSGD produces an accurate approximation.

Scientific Computing Research under the Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs) project. E. C. Cyr and N. Trask are supported by the Department of Energy early career program. M. Gulian is supported by the John von Neumann fellowship at Sandia National Laboratories.

## References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 265–283.

Adcock, B.; and Dexter, N. 2020. The gap between theory and practice in function approximation with deep neural networks. *arXiv preprint arXiv:2001.07523* .

Bach, F. 2017. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research* 18(1): 629–681.

Beck, C.; Jentzen, A.; and Kuckuck, B. 2019. Full error analysis for the training of deep neural networks. *arXiv preprint arXiv:1910.00121* .

Bengio, S.; and Bengio, Y. 2000. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks* 11(3): 550–557.

Billings, S. A.; and Zheng, G. L. 1995. Radial basis function network configuration using genetic algorithms. *Neural Networks* 8(6): 877–890.

Broomhead, D. S.; and Lowe, D. 1988. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom).

Chollet, F. 2017. *Deep learning with Python*. Manning Publications Company.

Cyr, E. C.; Gulian, M. A.; Patel, R. G.; Perego, M.; and Trask, N. A. 2019. Robust training and initialization of deep neural networks: An adaptive basis viewpoint. *arXiv preprint arXiv:1912.04862* .

Daubechies, I.; DeVore, R.; Foucart, S.; Hanin, B.; and Petrova, G. 2019. Nonlinear approximation and (deep) ReLU networks. *arXiv preprint arXiv:1905.02199* .

Fokina, D.; and Oseledets, I. 2019. Growing axons: greedy learning of neural networks with application to function approximation. *arXiv preprint arXiv:1910.12686* .

Fries, T.-P.; and Belytschko, T. 2010. The extended/generalized finite element method: an overview of the method and its applications. *International journal for numerical methods in engineering* 84(3): 253–304.

Geist, M.; Petersen, P.; Raslan, M.; Schneider, R.; and Kutyniok, G. 2020. Numerical solution of the parametric diffusion equation by deep neural networks. *arXiv preprint arXiv:2004.12131* .

Han, J.; Jentzen, A.; and Weinan, E. 2018. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences* 115(34): 8505–8510.

He, J.; Li, L.; Xu, J.; and Zheng, C. 2018. ReLU deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hebey, E. 2000. *Nonlinear Analysis on Manifolds: Sobolev Spaces and Inequalities*, volume 5. American Mathematical Soc.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Opschoor, J. A.; Petersen, P.; and Schwab, C. 2019. Deep ReLU networks and high-order finite element methods. *SAM, ETH Zürich* .

Patel, R. G.; Trask, N. A.; Gulian, M. A.; and Cyr, E. C. 2020. A block coordinate descent optimizer for classification problems exploiting convexity. *arXiv preprint arXiv:2006.10123* .

Schonsheck, S.; Chen, J.; and Lai, R. 2019. Chart Auto-Encoders for Manifold Structured Data. *arXiv preprint arXiv:1912.10094* .

Spivak, M. 2018. *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. CRC press.

Strouboulis, T.; Copps, K.; and Babuška, I. 2001. The generalized finite element method. *Computer methods in applied mechanics and engineering* 190(32-33): 4081–4193.

Telgarsky, M. 2016. Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485* .

Wang, S.; Teng, Y.; and Perdikaris, P. 2020. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536* .

Wendland, H. 2002. Fast evaluation of radial basis functions: Methods based on partition of unity. In *Approximation Theory X: Wavelets, Splines, and Applications*. Citeseer.

Yarotsky, D. 2017. Error bounds for approximations with deep ReLU networks. *Neural Networks* 94: 103–114.

Yarotsky, D. 2018. Optimal approximation of continuous functions by very deep ReLU networks. *arXiv preprint arXiv:1802.03620* .