SAND2021-8242PE

Robust architectures, initialization, and training for deep neural networks

Mamikon Gulian
John von Neumann Fellow
Computational Mathematics (1442)

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:

U.S. DEPARTMENT OF

dia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly ow
subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Background

- For decades, Deep Neural Networks were an academic research area, because they were too expensive to train.
- Backpropagation, based on the chain rule and dynamic programming, and GPU implementation starting in 2011 made it possible to train them efficiently.
- 2012: ImageNet challenge winners have never exceeded 74.3% top-five accuracy, and the competition has been dominated by classical approaches. A team led by Alex Krizhevsky and Geoffrey Hinton using DNNs achieve 83.6%. In more recent years, winners have achieved ∼95% top-five accuracy using improved DNNs and the ImageNet challenge is considered solved.
- Novel applications of DNNs – some of them breakthrough – have been found in various scientific applications. We consider DNNs for scientific computating tasks such as solving PDEs, surrogate modeling of high-dimensional systems, and reduced-order modeling, which involve DNNs for regression tasks.
- Here, DNNs have been utilized in leiu of established numerical methods, such as mesh-based PDE solvers and PCA.

Deep Neural Networks

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

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

  Here, $\boldsymbol{\xi}_H = \{(\boldsymbol{W}_l, \boldsymbol{b}_l)\}_{l=1}^{L}$ consists of the *hidden layer* parameters. 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

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

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

- With the hidden layer parameters $\boldsymbol{\xi}_H$ and linear layer parameters $\boldsymbol{\xi}_L = \boldsymbol{W}$ free, a DNN defines an **approximation class**, a parametrized manifold in some function space defined by the regularity of the activation function σ. With parameters fixed, a DNN is a fixed function.

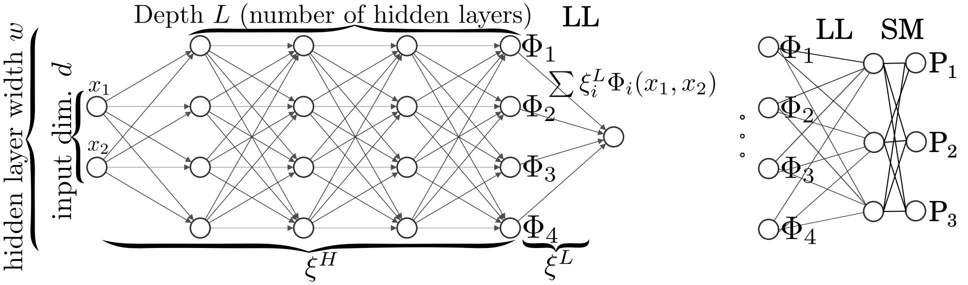


Figure: *Left:* Illustration of a fixed-width, fully-connected DNN with univariate output, typical for regression. *Right:* End-layers for a classification-type DNN for 3 classes.

- **For classification**, the DNN is typically of the form

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

where the softmax function is given by $\mathcal{F}^i_{\text{SM}}(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{j=1}^{N_c} \exp(x_j)}$, and satisfies for all x,

$$0 \leq \mathcal{F}^i_{\text{SM}}(\mathbf{x}) \leq 1, \quad \sum_i \mathcal{F}^i_{\text{SM}}(\mathbf{x}) = 1. \tag{4}$$

Approximation with DNNs

- Function approximation is at the heart of DNN applications.
- In regression, given finite samples $\{f(x_i)\}_{i=1}^N$ of a target function $f : \Omega \to \mathbb{R}^d$, where $x_i \in \Omega$, one wishes to construct a good approximation to f in some norm, such as $L^p(\Omega; \mathbb{R}^d)$ or $H^s(\Omega; \mathbb{R}^d)$.
- The purpose of training is to find parameters $(\boldsymbol{\xi}_H, \boldsymbol{\xi}_L)$ that minimize some discrete approximation to these norms.
- In a regression problem, the loss function may be of the form

  $$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{W}) = \|u(x) - \mathcal{NN}(x, \boldsymbol{\xi}, \boldsymbol{W})\|^2_{\ell_2(\mathcal{X})}, \tag{5}$$

  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, we instead try to minimize the discrepancy between the DNN and a target distribution, e.g. in the K-L divergence or Wasserstein metric sense, given some samples. Usually the cross-entropy loss is used.
- We will focus in regression for the remainder of the talk, but classification **architectures** will make an appearance.

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.

$$\mathcal{L}(\boldsymbol{\xi}, \boldsymbol{W}) = \|u(x) - \mathcal{N}\mathcal{N}(x, \boldsymbol{\xi}, \boldsymbol{W})\|^2_{\ell_2(\mathcal{X})} + \left\| f(x) - P\left[\mathcal{N}\mathcal{N}(x, \boldsymbol{\xi}, \boldsymbol{W})\right] \right\|^2_{\ell_2(\text{collocation points})}. \tag{6}$$

- 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 knowledge of the 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.

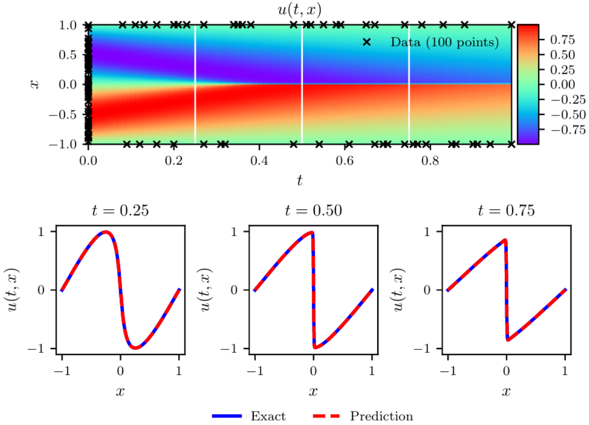


Figure: Solving the Burgers' with a PINN using only IC and BC data, with collocation points (not shown) in the interior. From Raissi et. al., "Physics Informed Deep Learning (Part I)".

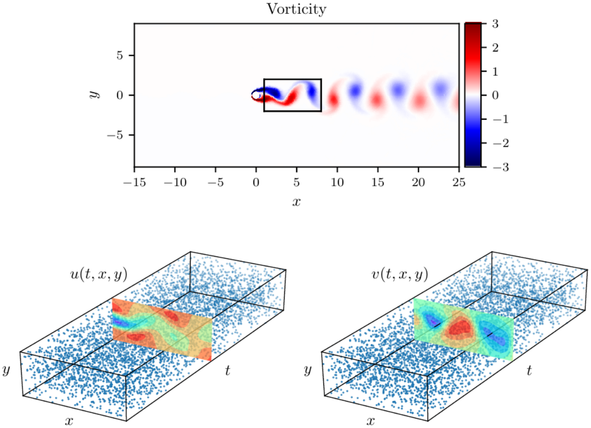


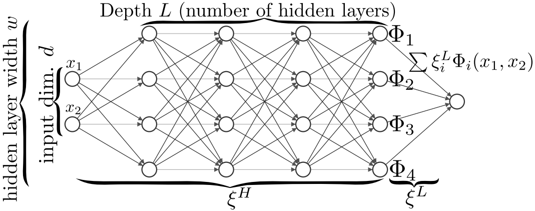
Figure: Solving the Burgers' with a PINN using IC, BC and **interior** observations, with collocation points (not shown) in the interior. From Raissi et. al., "Physics Informed Deep Learning (Part II)".

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

$$\mathcal{NN}_{\boldsymbol{\xi}}(\boldsymbol{x}) = \sum_{i=1}^{w} \xi_i^{\mathrm{L}} \Phi_i(\boldsymbol{x}; \boldsymbol{\xi}^{\mathrm{H}}) \tag{7}$$

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 $\boldsymbol{\xi}^{\mathrm{H}}$.



- This viewpoint makes it clear that $\boldsymbol{\xi}_H$ parametrize the basis (like a FEM Mesh & Shape functions), while $\boldsymbol{\xi}_L$ are just coefficients for these basis functions.

What is a PINN doing? A scientific computing perspective

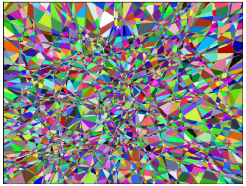
- As an approximation scheme (class & method), training DNNs is nonlinear, like adaptive FEM, free-knot splines, and best n-term wavelet approximation. (Linear methods would be Fourier series and Taylor polynomials).
- Letting an optimizer go to town on a PINN is analogous to letting all the parameters of a FEM mesh with parametrized shape functions be completely free, and letting a first-order gradient optimizer try to adapt the mesh and solve your PDE using a collocation objective function at the same time!
- This analogy can be made quite concrete; see He et al, (2018), “Relu deep neural networks and linear finite elements”, which compares ReLU DNNs to continuous piecewise linear (CPWL) finite elements.
- 

  Figure: From Hanin and Rolnick, “Complexity of Linear Regions in Deep Networks”, 2019. Output of a ReLU-activation DNN with two-dimensional input. Different colors correspond to different direction/slope of the CPWL output, to make “cut-planes” visible.

Approximation Theory & Approximation Practice

- 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.

Problems we consider and their loss functions

- We consider the following class of ℓ_2 regression problems:

$$\operatorname*{argmin}_{\boldsymbol{\xi}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \mathcal{L}_k\left[\mathcal{N}\mathcal{N}_{\boldsymbol{\xi}}\right] \right\|^2_{\ell_2(\mathcal{X}_k)} \tag{8}$$

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

$$\operatorname*{argmin}_{\boldsymbol{\xi}} \|u - \mathcal{N}\mathcal{N}_{\boldsymbol{\xi}}\|^2_{\ell_2(\mathcal{X})}. \tag{9}$$

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

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

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

  while residual networks correspond to

$$\boldsymbol{\Phi}^{\text{res}}(\boldsymbol{x}, \boldsymbol{\xi}) = (\boldsymbol{I} + \boldsymbol{\sigma} \circ \boldsymbol{T}_L) \circ \cdots \circ (\boldsymbol{I} + \boldsymbol{\sigma} \circ \boldsymbol{T}_2) \circ (\boldsymbol{\sigma} \circ \boldsymbol{T}_1), \tag{11}$$

  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 $\boldsymbol{b}$.

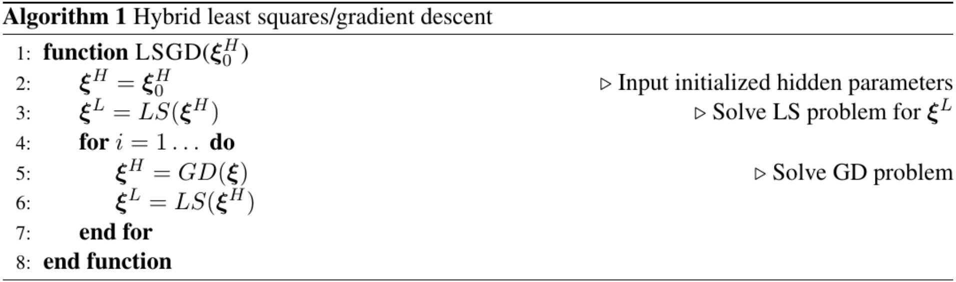
- In practice, very deep plain DNNs are not trainable. A rule of thumb is if you have more than 10 layers, you should probably use a ResNet.

Hybrid Least Squares/Gradient Descent

- We seek
$$\underset{\boldsymbol{\xi}^{\mathrm{L}},\boldsymbol{\xi}^{\mathrm{H}}}{\operatorname{argmin}} \sum_{k=1}^{K} \epsilon_k \left\| \mathcal{L}_k[u] - \sum_i \xi_i^{\mathrm{L}} \mathcal{L}_k\left[\Phi_i(\boldsymbol{x},\boldsymbol{\xi}^{\mathrm{H}})\right] \right\|_{\ell_2(\mathcal{X}_k)}^2 . \tag{12}$$
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 $K = 1$):
$$\underset{\boldsymbol{\xi}^{\mathrm{L}}}{\operatorname{argmin}} \left\| A\boldsymbol{\xi}^{\mathrm{L}} - \boldsymbol{b} \right\|_{\ell_2(\mathcal{X})}^2 \tag{13}$$
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 $\boldsymbol{x}_i \in \mathcal{X},\ i = 1,\dots,N,$ $j = 1,\dots,w$.

Hybrid Least Squares/Gradient Descent (LSGD)

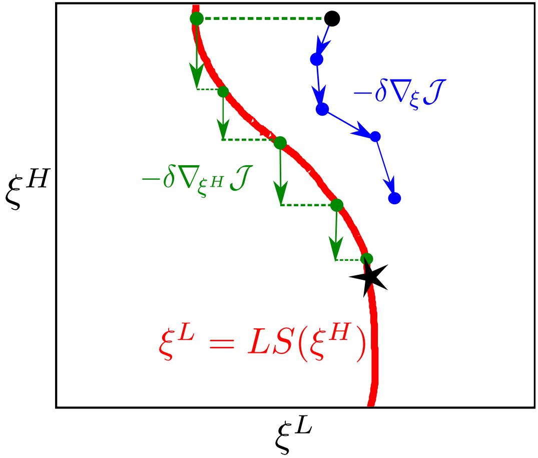
- Exposing the LS problem in this way prompts a natural modification of gradient descent.
- The LSGD algorithm proceeds by alternating between: a LS solve to update $\boldsymbol{\xi}^{\mathrm{L}}$ by a global minimum for given $\boldsymbol{\xi}^{H}$, and a GD step to update $\boldsymbol{\xi}^{\mathrm{H}}$.



Algorithm 1 Hybrid least squares/gradient descent

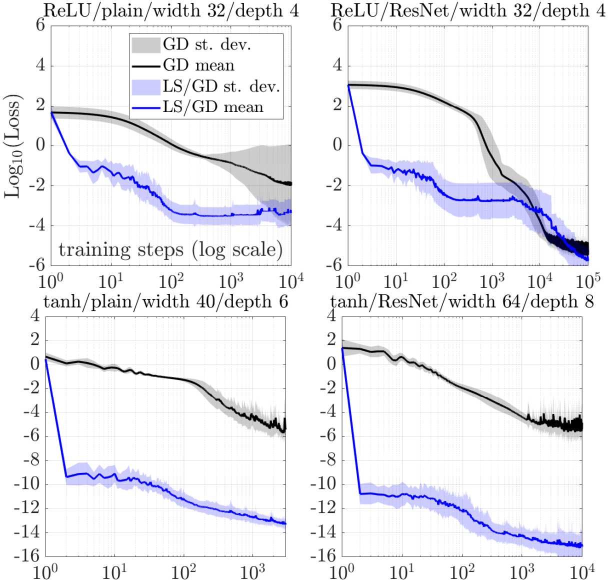
```
function LSGD($\boldsymbol{\xi}_0^H$)
    $\boldsymbol{\xi}^H = \boldsymbol{\xi}_0^H$                      ▷ Input initialized hidden parameters
    $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$                    ▷ Solve LS problem for $\boldsymbol{\xi}^L$
    for $i = 1 \ldots$ do
        $\boldsymbol{\xi}^H = GD(\boldsymbol{\xi})$                  ▷ Solve GD problem
        $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$
    end for
end function
```

Illustration of LSGD



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

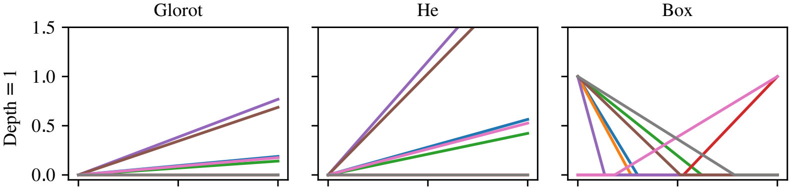


Mean of $\log_{10}(\text{Loss})$ over 16 training runs ± one standard deviation of the same quantity, for approximating $\sin(2\pi x)$ on $[0, 1]$ sampled at 256 evenly spaced points.

Sandia National Laboratories

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.

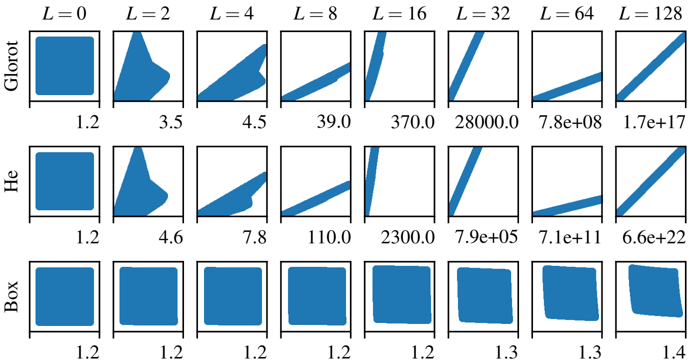


Figure: Images of the unit square $[0,1]^2$ under L initialized hidden layers of **ResNets** for Glorot (*top*), He (*center*) and Box (*bottom*) initializations. Values are presented on the square $[-0.2, H]^2$, where H is denoted to the bottom-right of each image. Collapse to a line through the origin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

Basis function plots for DNNs of width 8

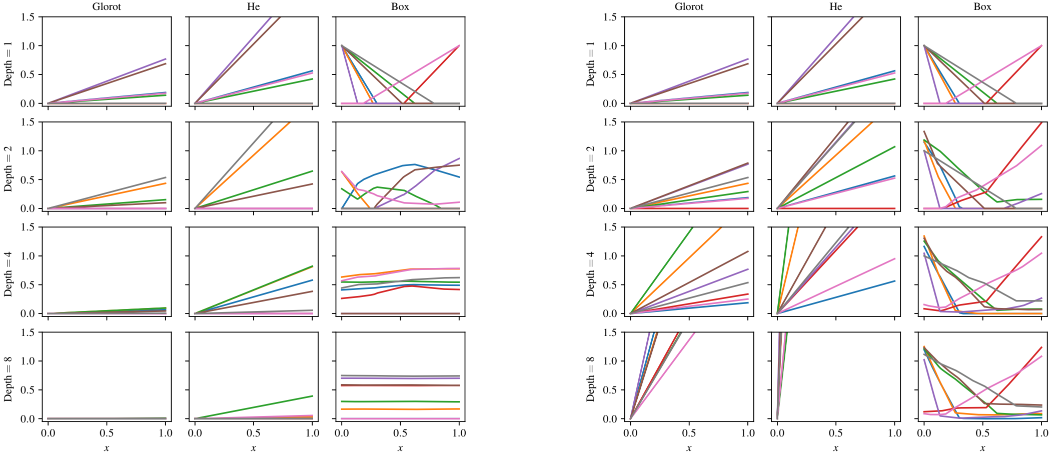


Figure: Left: Plain DNN, Right: ResNet.

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.

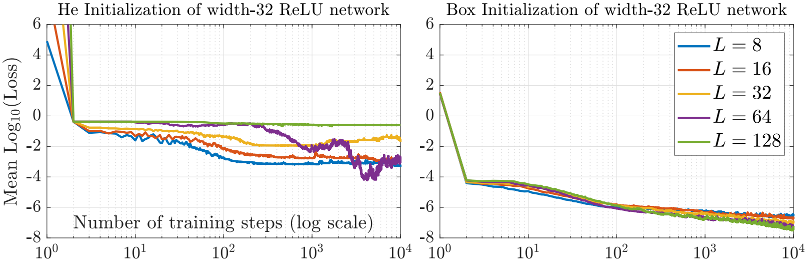


Figure: Mean of $\log_{10}$(Loss) over 16 training runs of residual width-32 ReLU network with $L = 8, 16, 32, 64$ and 128 hidden layers and training rate $2^{-(k+3)}$ for the He *(left)* and Box *(right)* initializations.

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

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

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.

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 $\mathcal{J}_1 = \mathcal{J}_2 = \mathcal{J}_3 = 0$, rendering the choice of ϵ unimportant (we set $\epsilon = 1$).

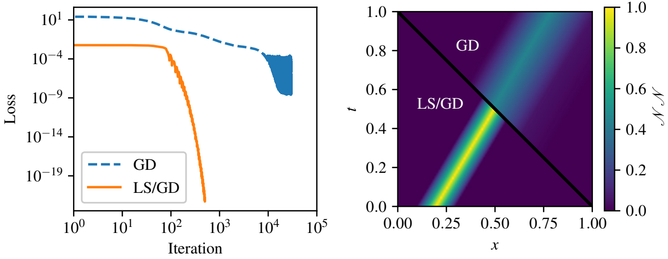
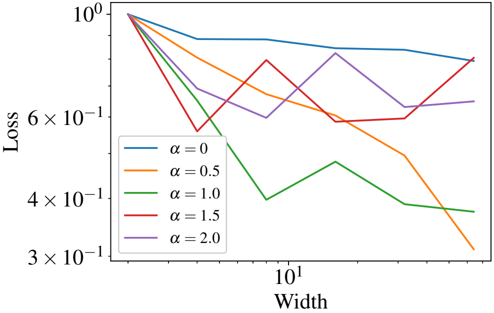


Figure: *Left:* Loss evolution over training for GD and LSGD. *Right:* Solution after 5000 iterations for GD and 500 iterations for LSGD. Setting: Box initialization, ReLU activation function, network width = 32, depth = 1, learning rate = 0.005.

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.



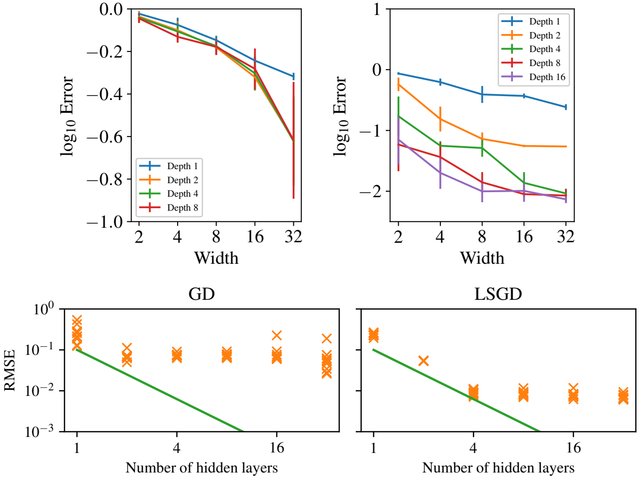


Figure: **Top:** Convergence with ReLU (left; learning rate 0.001) using $\alpha = -\frac{1}{2}$, and tanh (right; learning rate 0.01) using $\alpha = -\frac{1}{2}$. **Bottom:** Comparison of GD (*left*) and LSGD (*right*) training for tanh activation, learning rate 0.01, width 32 and 5000 steps. X's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

The POUnet Architecture

- Using practical training methods, it is not possible to achieve *hp*-approximation rates using DNNs, despite what is theoretically possible with optimal approximation rates.
- 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.

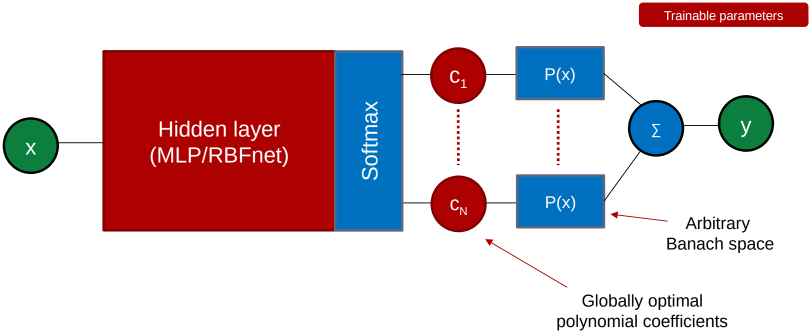


Figure: POUnet architecture.

Main idea: rather than attempt to emulate POU + monomials, build a POU directly into the architecture using softmax layers and append monomial layers.

Training with POUnets

- We fit both the localized basis coefficients $\boldsymbol{c} = [c_{\alpha,\beta}]$ and the localization itself simultaneously by solving the optimization problem

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

- 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.
- 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

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

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

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

  and

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

- In practice, we do not work with compactly supported partition functions, since globally supported functions are easier to train as they are more robust to a bad initialization.
- However, the principle behind this theorem still holds in practice: partitions that are not localized should be avoided and every partition should be "active" to minimize the size of the partitions.

Sandia National Laboratories

Two-Phase Training

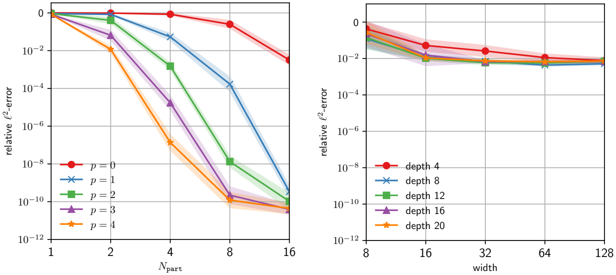
- The least-squares structure of (18) allows application of the least-squares gradient descent (LSGD) block coordinate descent strategy.
- To prevent learned partition functions ϕ_α from "collapsing" to near-zero values everywhere. we will also consider a pre-training step, which adds an ℓ_2 regularizer 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.
- In the first phase of training, we include this regularizer to the loss. After a certain number of epochs, when good partitions have been found, we perform a second phase of training without it.

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.

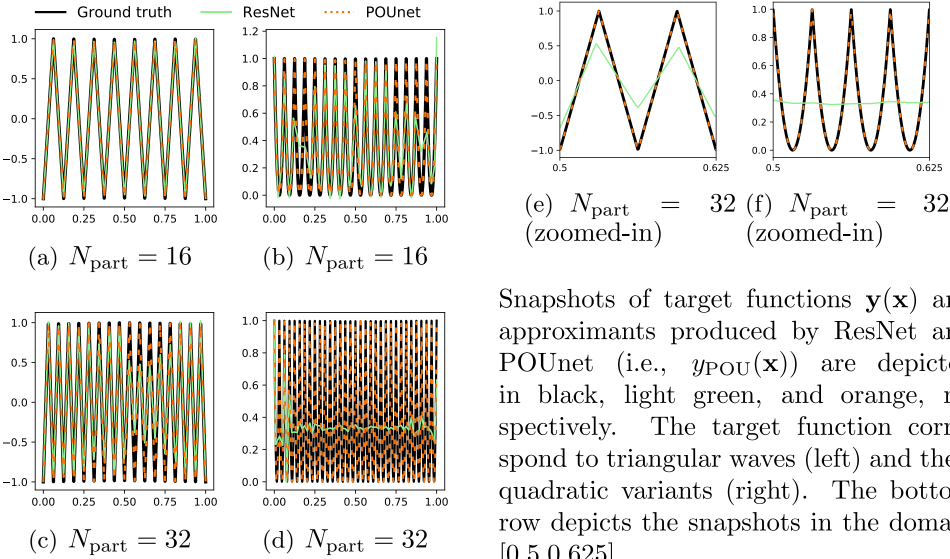


(a) POUnets

(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.



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

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

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

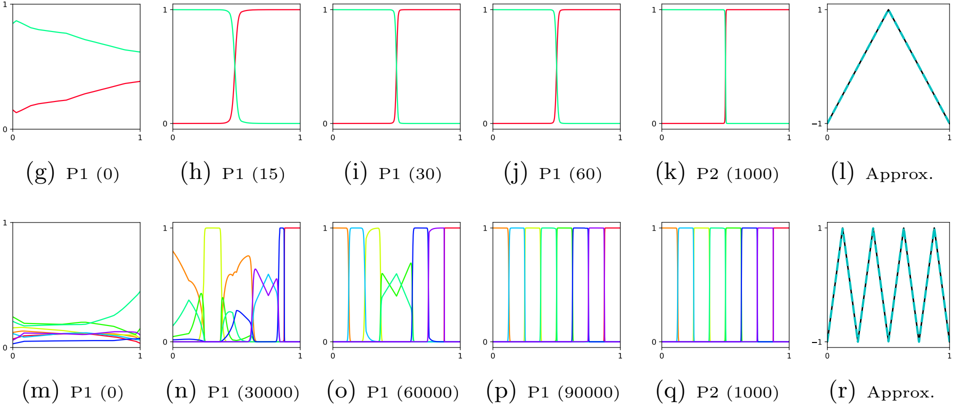
(d) $N_{\text{part}} = 32$

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

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

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 (piecewise linear waves)

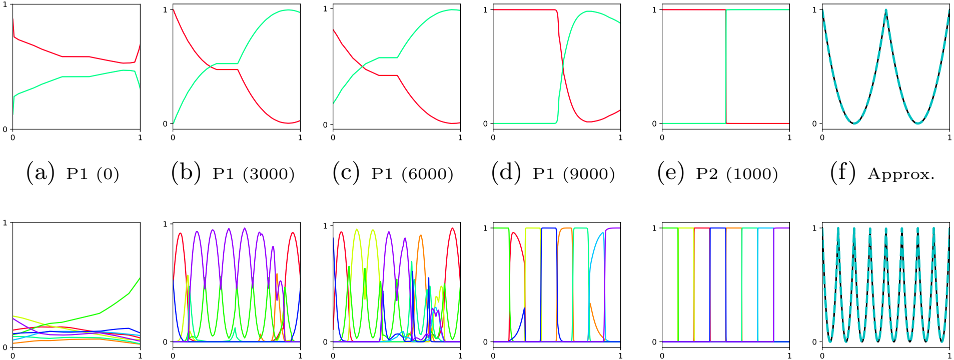


(g) P1 (0) (h) P1 (15) (i) P1 (30) (j) P1 (60) (k) P2 (1000) (l) Approx.

(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.

Two-phase training (piecewise quadratic waves)



(a) P1 (0) (b) P1 (3000) (c) P1 (6000) (d) P1 (9000) (e) P2 (1000) (f) Approx.

(g) P1 (0) (h) P1 (100000) (i) P1 (150000) (j) P1 (300000) (k) P2 (500000) (l) Approx.

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.

Probabilistic partition of unity networks (PPOUnets)

- We enrich POU-Nets with a Gaussian noise model to obtain a probabilistic generalization amenable to gradient-based minimization of a maximum likelihood loss.
- The architecture provides spatial representations of both noiseless and noisy data as Gaussian mixtures with closed form expressions for variance which provides an estimator of local error.
- The training process yields remarkably sharp partitions of input space based upon correlation of function values.
- Compared to standard deep neural networks, the framework demonstrates *hp*-convergence without the use of regularizers to tune the localization of partitions.
- The framework scales more favorably to large data sets as compared to Gaussian process regression and allows for spatially varying uncertainty, leveraging the expressive power of deep neural networks while bypassing expensive training associated with other probabilistic deep learning methods.

Generative model, mean and variances

- Now, we associate to each partition i a random variable $X_i(\theta_2)$ representing additive noise whose distribution is characterized by parameters θ_2. We define an analogous sample of the probabilistic partition on unity network (PPOU-Net) model as

$$y_{\text{ppou}}(x, \xi, \omega; \theta_1, \theta_2) = \sum_i \Phi_i(x, \xi; \theta_1)\big(p_i(x) + X_i(\omega, \theta_2)\big), \tag{24}$$

  where $X_i(\omega, \theta_2)$ denotes a sample of $X_i(\theta_2)$ with sample index ω. This leads to the following generative model: for fixed x, one selects partition i with discrete probability ϕ_i and then samples $(p_i(x) + X_i(\omega; \theta_2))$. It is clear that in the absence of the additive noise ($X_i = 0$ for all i) we recover the standard POU-Net generative model.

- The probability density function $p(y_{\text{ppou}}(x; \theta_1, \theta_2))$ governing the distribution of (24) is given by

$$p(y_{\text{ppou}}(x; \theta_1, \theta_2)) = \sum_{i=1}^{M} \phi_i(x; \theta_1) \mathcal{N}\left(y_{\text{ppou}}(x) \,\middle|\, \mu_i + Q(x), \sigma_i\right), \tag{25}$$

  where $\mathcal{N}(y \,|\, \mu, \sigma)$ denotes the normal density with mean μ and standard deviation σ.

PPOUnets: POUnets with GMM noise model

- The mean μ_y and variance σ_y of the random variable $y_{\text{ppou}}(x)$ are then given explicitly by

$$\mu_y(x) = \sum_{i=1}^{M} \phi_i(x; \theta_1)(\mu_i + Q(x)), \tag{26}$$

$$\sigma_y(x) = \sum_{i=1}^{M} \phi_i(x; \theta_1)\sigma_i^2 + \sum_{i=1}^{M} \phi_i(x; \theta_1)\mu_i^2 - \left(\sum_{i=1}^{M} \phi_i(x; \theta_1)\mu_i \right)^2 . \tag{27}$$

- Denoting by $\mathcal{N}(\mu, \sigma)$ the random variable with density $\mathcal{N}(y \,|\, \mu, \sigma)$, we have

$$y_{\text{ppou}}(x; \theta_1, \theta_2) \sim Q(x) + \sum_{i=1}^{M} \phi_i(x; \theta_1)\mathcal{N}(\mu_i, \sigma_i), \tag{28}$$

which represents y_{ppou} as a POU-Net prediction augmented by Gaussian mixture uncertainty.

Likelihood function

- For the joint statistics for $y(x_1), y(x_2), ..., y(x_N)$, we assume independence to obtain the multivariate density

$$p(y(x_1), y(x_2), ..., y(x_N)) = \prod_{\ell=1}^{N} p(y(x_\ell)) = \prod_{\ell=1}^{N} \sum_{i=1}^{M} \phi_i(x_\ell;\theta_1)\mathcal{N}(y(x_\ell);\mu_i + Q(x_\ell), \sigma_i), \tag{29}$$

where $p(y(x_\ell))$ is the density given by (25). Given data $\mathcal{D}$, we can evaluate the above density and treat it as a likelihood.

- We then define a likelihood loss

$$\begin{aligned}
\mathcal{L}(\theta_1, \boldsymbol{\mu}, \boldsymbol{\sigma}) &= -\log\left(p(y(x_1), y(x_2), ..., y(x_N))\right) & (30)\\
&= -\log\left(\prod_{\ell=1}^{N} \sum_{i=1}^{M} \phi_i(x_\ell;\theta_1)\mathcal{N}(y(x_\ell);\mu_i + Q(x_\ell), \sigma_i)\right) & (31)\\
&= -\sum_{\ell=1}^{N} \log\left(\sum_{i=1}^{M} \phi_i(x_\ell;\theta_1)\mathcal{N}(y(x_\ell);\mu_i + Q(x_\ell), \sigma_i)\right). & (32)
\end{aligned}$$

Training PPOUnets

- $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are vectors with components μ_i and σ_i; below, we refer to $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ as θ_2.
- We minimize this loss using the Adam algorithm. For all examples, a learning rate of 0.01 is used to emphasize the lack of sensitivity to model parameters.
- The parameters defining Q, i.e., the coefficients of the polynomials corresponding to each partition, can be updated by minimizing the likelihood loss. An alternative step to obtain an estimate of Q is to solve the following least squares problem for fixed θ_1,

$$p_i^*(x;\theta_1) = \underset{p_i \in \pi_m}{\operatorname{argmin}} \sum_{j=1}^{N} \sum_{i=1}^{M} \left(\phi_i(x_j;\theta_1)(p_i(x_j) - y(x_j))\right)^2 , \qquad (33)$$

and define $Q_m^* = \sum_{i=1}^{M} \phi_i(x;\theta_1) p_i^*(x)$. Note that this least square estimator amounts to solving a linear system of size $M\dim(\pi_m)$.

PCA bisection of partitions

- We observed that the optimizer partitions the y-axis by placing the μ_i and selecting the σ_i to cluster the y-values of the data. This effects a “soft” classification of the data into M nearly-disjoint classes or intervals, labelled by the POU index i. The optimizer concurrently partitions space during training by adapting the supports of the POU functions $\phi_i(x;\theta_1)$ to the boundaries of the x-values of the data that have been so labelled.
- We found that this training is excellent at producing nearly disjoint partition of domain space by the sets $\mathrm{supp}(\phi_i)$ without explicit regularization. However, it has the undesirably property that the sets $\mathrm{supp}(\phi_i)$ are not simply connected.
- To post-process partitions, after the ϕ_i are trained, we pursue a principal component analysis (PCA) based bisection strategy which reliably yields simply-connected, quasi-uniform partitions in a post-processing step.
- Given a collection of points $\mathcal{D}_i \subset \mathcal{D}$, define

$$C_i = \sum_{x_j \in \mathcal{D}_i} (x_j - \bar{x}_i) \otimes (x_j - \bar{x}_i)\,, \tag{34}$$

  where $\bar{x}_i = (\#\mathcal{D}_i)^{-1} \sum_{x_j \in \mathcal{D}_i} x_j$ is the center of mass. Performing the singular value decomposition of C_i provides a best fit of an ellipsoid to $\mathcal{D}_i$, with the axis direction given by the right singular vectors and the lengths given by the corresponding singular values. We denote the vector corresponding to the largest singular value $\boldsymbol{n}_i$.

Training with PCA bisection

- The function $F(x) = \text{argmax}\{\phi_i(x)\}$ classifies the data according into M classes. Taking $\mathcal{D}_i = F^{-1}(i)$, we obtain two new partitions $\phi_{i,+}(x) = \phi_i(x) * \mathbf{1}_{(x-\bar{x}_i)\cdot\boldsymbol{n}_i>0}(x)$ and $\phi_{i,-}(x) = \phi_i(x) * \mathbf{1}_{(x-\bar{x}_i)\cdot\boldsymbol{n}_i\leq 0}(x)$, where $\mathbf{1}_A$ denotes the indicator function of a set A.
- This decomposition preserves the POU property because $\phi_i = \phi_{i,+} + \phi_{i,-}$, and yield a total of $M_{\text{tot}} = M \times 2^{N_{\text{ref}}}$ partitions of input space.

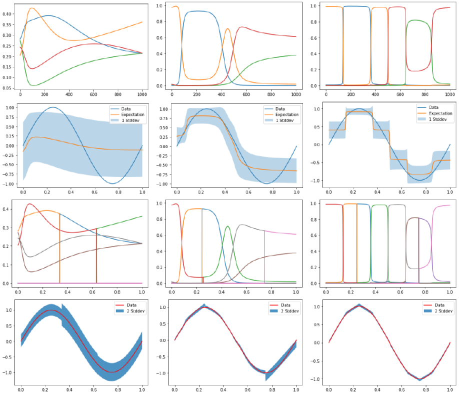
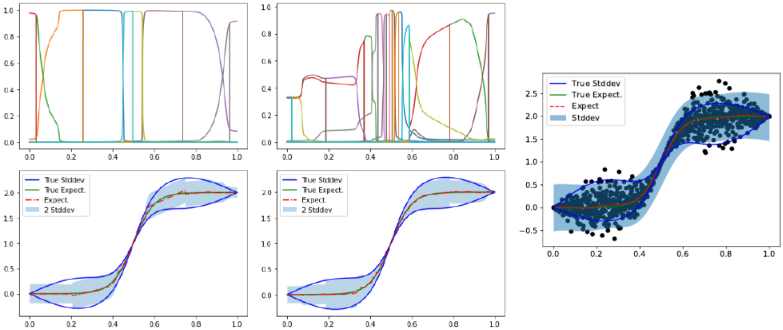


Figure: Evolution of PPOU at initialization (*left column*), 1,000 steps (*center column*) and 10,000 steps (*right column*). *Row 1:* initially random partitions evolve into approximation of indicator functions. *Row 2:* Training of MLE without polynomial contribution Q yields piecewise constant expectation on each partition with standard deviation estimating error. *Row 3:* Bisection provides simply connected partitions amenable to polynomial approximation. *Row 4:* Piecewise polynomial regression with an estimate of uncertainty with no human in the loop.

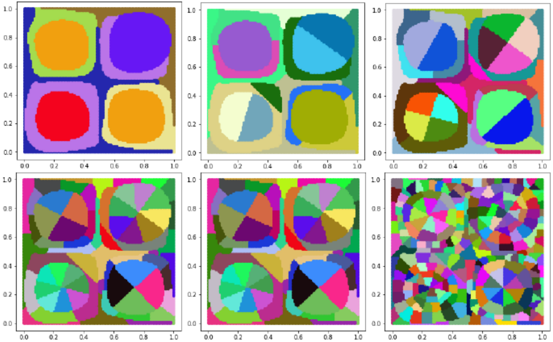
Noisy with spatially heterogeneous noise

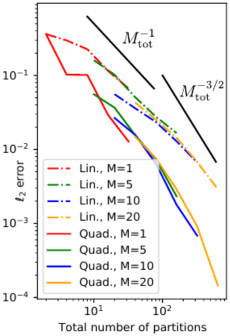
- Regression of smooth function with spatially varying noise using $M = 5$ partitions *(left column)* and $M = 10$ partitions *(center column)*. Partitions *(top)* cluster automatically near steep gradients in either function value or uncertainty to accurately estimate standard deviation. In comparison, Gaussian process regression *(right column)* is unable to provide an accurate estimate of uncertainty, since the constant noise amplitude σ_{GP} is unable to model the spatially heterogeneous noise.



Regression in 2D

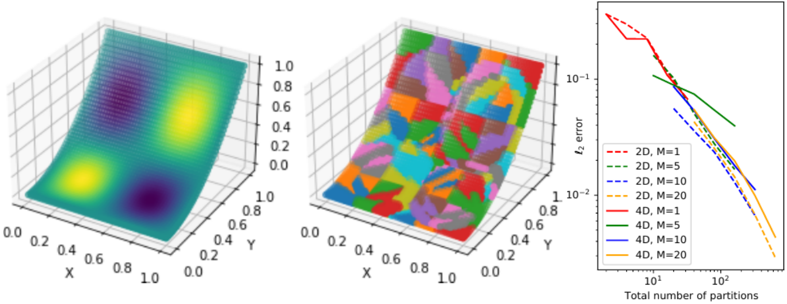
- Regressing $y = \sin 2\pi x \sin 2\pi y$ in two dimensions *(left)* provides partitions of more complex topology. Bisection is critical to obtain compactly supported sets amenable to polynomial approximation, while initial partition focuses refinement. Comparison of linear and quadratic regression for $M = 1, 5, 10, 20$ initial partitions demonstrate algebraic convergence rates under refinement consistent with FEM approximation theory.





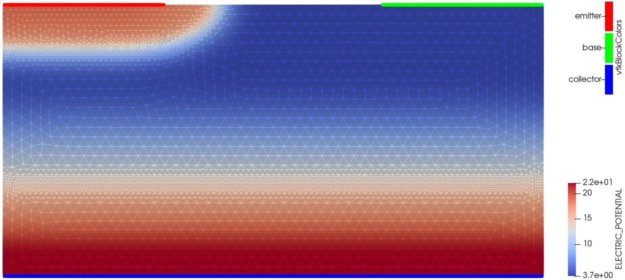
Breaking the curse of dimensionality: Regression in 4D

- Lifting the data from the previous example by mapping $(x_1, x_2) \in \mathbb{R}^2$ to $(x_1, x_2, x_2^2, 0) \in \mathbb{R}^4$ *(left)* provides qualitatively similar partitions of space *(center)* which provide similar convergence rates when comparing the two- and four-dimensional settings *(right)*. Plots depict the projection of data $(x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ into $(x_1, x_2, x_3, 0) \in \mathbb{R}^3$.



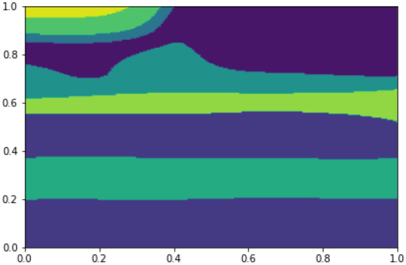
The Charon Database

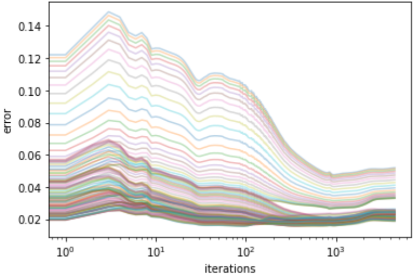
- A representative solution from the BJT database demonstrating the electric potential field on a finite element grid with 4,154 degrees of freedom (basis functions). The potential is the solution of the boundary value problem obtained by imposing a fixed voltage V_1 at the emitter (top left red boundary), V_2 at the base (top right green boundary), and V_3 at the collector (bottom blue boundary). The database is constructed by performing a three dimensional sweep over (V_1, V_2, V_3).

- 

Reduced-order model using PPOUnets for Charon

- (*Left:*) Applying the PPOU process with $m = 0$ and $N_{\text{ref}} = 0$ provides an estimate for the best set of 10 partitions which approximate any given PDE solution in the database well. The location of partitions coincide with physical intuition - the sharp gradients near "junctions" at the emitter and baseplate are the locations where the BJT provides ideal diode-like behavior.
- (*Right:*) Evolution of RMS error for least squares fit of each solution in the database as partitions evolve during training. A worst case error of $< 5\%$ is achieved.





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.

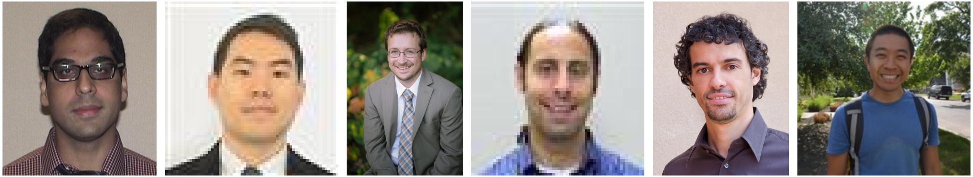


Figure: Ravi, Kookjin, Nat, Eric, Mauro, and Andy.

- 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!