Learning waveforms with partition of unity networks

Tiffany Fan
tiffan@stanford.edu *

Abstract

Using deep learning methods to approximate continuous functions and operators is a critical problem in both machine learning theory and high-fidelity real-time engineering simulation applications. In this work, we explore the partition of unity networks (POUnets) model [1] in the context of approximating waveform functions. We propose an alternative training strategy that leverages minibatch gradient descent and simplifies hyperparameter tuning. The modified POUnets models outperform the baseline fully-connected neural network models of comparable sizes in most numerical experiments.

1 Introduction

Unlike most deep learning applications that aim at minimizing classification errors, scientific computing and engineering simulations often depend on highly accurate approximations of continuous and nonlinear functions and operators. Classical methods such as Fourier analysis and orthogonal polynomial expansions have been well studied for such approximation tasks. Recently, deep learning models have been applied to physics-based engineering problems with the goal of accelerating the simulations without sacrificing the accuracy. To better inform the model selection between classical methods and deep learning methods, it is critical to understand the convergence of neural networks from the perspective of approximation theory and uncertainty quantification.

1.1 Problem Statement

We consider the partition of unity networks (POUnets), a hybrid approximation model for functions and operators that leverages both deep neural networks (DNNs) and polynomial expansions [1]. The approximant takes the form

$$y_{\mathrm{POU}}(\mathbf{x}) = \sum_{\alpha=1}^{N} \phi_\alpha(\mathbf{x}) \sum_{\beta=1}^{m} c_{\alpha,\beta} P_\beta(\mathbf{x}), \tag{1}$$

where $\{\phi_\alpha(\mathbf{x})\}_{\alpha=1}^N$ is a partition of unity (a set of non-negative functions that sum to one), V is the space of polynomials of degree $m-1$, and $\{P_\beta(\mathbf{x})\}_{\beta=1}^m$ is a basis for space V (thus $m = \dim(V)$). In particular, the partition $\{\phi_\alpha(\mathbf{x})\}_{\alpha=1}^N$ is parameterized by a DNN with output dimension N and parameters $\boldsymbol{\theta}$, i.e.,

$$\phi_\alpha(\mathbf{x};\boldsymbol{\theta}) = [\mathcal{NN}(\mathbf{x};\boldsymbol{\theta})]_\alpha, \quad 1 \le \alpha \le N.$$

We explore POUnets in the context of approximating waveforms, which are common approximation bases in engineering simulation applications. We experiment with sine functions, piecewise triangular waves, and piecewise quadratic waves with various frequency values.

*The author would like to acknowledge Prof. Eric Darve, Prof. Nathaniel Trask, Kookjin Lee, and Huizi Mao for their valuable suggestions and technical support. The implementation of the experiments is adapted from the GitHub repository https://github.com/rgp62/leastsquares_pou by Lee et al.

CS230: Deep Learning, Spring 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

1.2 Related Work

Physics informed machine learning High-fidelity engineering simulations based on traditional physical models require a large amount of memory and computing time. Recent work has demonstrated the potential of data-driven learning to facilitate efficient and accurate engineering simulations [2, 3, 4, 5]. However, many tools are developed for specific applications and cannot be easily generalized to other engineering systems [6]. This work aims at exploring a general framework that can be applied to a wide variety of approximation tasks.

Deep neural networks approximation theory Yarotsky and Opschoor et al. proved the existence of weights and biases values for ReLU-based DNNs to approximate a wide range of basis functions [7, 8, 9]. Assuming that the optimal values for weights and biases are achieved by the training strategy, the approximation rates from ReLU DNNs are shown to closely match the best available approximation rates from classical approximation by piecewise polynomial spline functions [9]. Convergence results have also been established for one-layer and two-layer networks as the width goes to infinity [8, 10]. The approximation theory of multi-layer neural networks remains incomplete.

2 Dataset

The project is conducted with simulated data. To generate a training dataset of size m_{train}, we choose m_{train} evenly spaced grid points to represent the sensor locations in the domain $D = [0, 1]$. The coordinates of the evenly spaced points are denoted as $\{\mathbf{x}_i\}_{i=1}^{m_{\text{train}}}$. Given a target function $f : [0, 1] \to \mathbf{R}$, we evaluate f on these points to generate the training outputs $\{f(\mathbf{x}_i)\}_{i=1}^{m_{\text{train}}}$. Each training example takes the form $\{\mathbf{x}_i, f(\mathbf{x}_i)\}$.

To generate a testing dataset, we randomly sample a set of m_{test} points, denoted $\{\mathbf{x}_i\}_{i=1}^{m_{\text{test}}}$, from the domain, and evaluate the model by comparing the model outputs $\{y_{\text{POU}}(\mathbf{x}_i)\}_{i=1}^{m_{\text{test}}}$ to the target function evaluated at the set of testing points $\{f(\mathbf{x}_i)\}_{i=1}^{m_{\text{test}}}$. A similar approach can be used to generate a validation dataset $\{\mathbf{x}_i, f(\mathbf{x}_i)\}_{i=1}^{m_{\text{val}}}$.

3 Methods

3.1 Model Architecture

As illustrated in Fig. 1, the model combines a DNN component with a polynomial component. The DNN component consists of a DNN model (RBF-Net or ResNet [11]) and a softmax layer, and is responsible for learning a partition of the domain. Due to the softmax layer, any input data point will be "classified" into one of the parts in the DNN-learned partition. For each part of the domain, the target function or operator is approximated by a polynomial expansion in a chosen polynomial basis with trainable coefficients (i.e., $c_{\alpha,\beta}$ in (1)). Since the derivatives of the basis polynomials can be easily computed, we can backpropogate numerical gradients through the entire model to inform parameter updates.

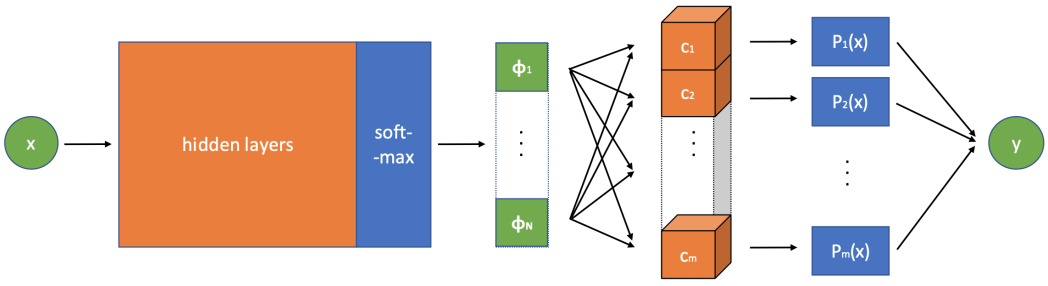


Figure 1: Partition of unity networks architecture described in [1]. Computational units containing trainable model parameters are colored in orange. Data and activations are colored in green.

Both the POUnets architecture and the alternative architecture are compared with the baseline model architecture, which is a fully-connected DNN and has $N = 1$ as its output dimension.

3.2 Training Strategy

We start with the training strategy proposed in [12] and [1] (the "LSGD algorithm"), which minimizes the loss function with a regularization term on the polynomial coefficients $\mathbf{c}$, i.e., $L_{\text{LSGD}} = \sum_{i=1}^{m_{\text{train}}} \left(y_{\text{POU}}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2 + \lambda \|\mathbf{c}\|_F^2$. The optimization algorithm alternates between (i) solving a weighted least squares problem for the optimal polynomial coefficients $\mathbf{c} = \{c_{\alpha,\beta}\}_{\alpha=1,\ldots,N;\beta=1,\ldots,m}$ while fixing the hidden layer parameters $\boldsymbol{\theta}$ and (ii) applying gradient descent with batch normalization to update the hidden layer parameters $\boldsymbol{\theta}$ while fixing the polynomial coefficients $\mathbf{c}$. The Adam optimizer [13] with different learning rates are used in step (ii) for different target functions.

Next, we propose an alternative training strategy, where we implement the polynomial component in Fig. 1 as a custom layer, treat the entire model as one DNN, and perform minibatch gradient descent with the Adam optimizer [13] to minimize the mean squared error loss

$$L_{\text{MSE}} = \frac{1}{m_{\text{train}}} \sum_{i=1}^{m_{\text{train}}} \left(y_{\text{POU}}(\mathbf{x}_i) - f(\mathbf{x}_i)\right)^2. \tag{2}$$

With the alternative training strategy, we update all the parameters, including the DNN parameters $\boldsymbol{\theta}$ and the polynomial coefficients $\mathbf{c}$, at the end of each training iteration.

4 Experiments

The experiments are implemented in Tensorflow and are adapted from the tools implemented in [14].

Data Generation For all experiments, we consider the data domain $D = [0, 1]$, and take $m_{\text{train}} = 2048$ evenly spaced points as training inputs: $\mathbf{x}_i = \frac{2048}{2047}(i-1)$ for $i = 1, \cdots, 2048$. We take $m_{\text{val}} = m_{\text{test}} = 2048$ and sample both validation and testing inputs uniformly from $[0, 1]$.

Target Functions We denote the triangular wave function on $[0, 1]$ with frequency p as

$$T(x; p) = 2\left|px - \left\lfloor px + \frac{1}{2} \right\rfloor\right| - 1, \quad p = 0, 1, 2, \ldots \tag{3}$$

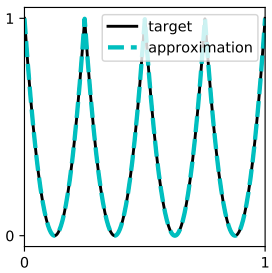


Figure 2: The piecewise quadratic target function $f(x) = T^2(x; 3)$ and its approximation $y_{\text{POU}}(x)$ by POUnet.

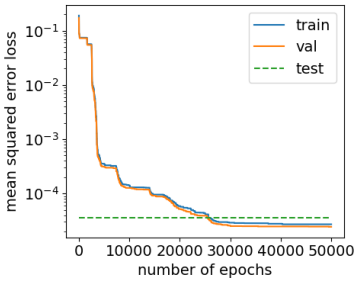


Figure 3: A typical plot of training, validation, and testing loss (2). The quantities are highly correlated due to the data generation process.

We use POUnets to approximate (i) piecewise linear target functions of the form $f(x) = T(x; p)$, for frequency $p = 1, 2, 3, 4$; (ii) piecewise quadratic target functions of the form $f(x) = T^2(x; p)$, for frequency $p = 1, 2, 3, 4$; and (iii) sine functions of the form $f(x) = 2\pi px$ on $[0, 1]$, for angular frequency $p = 1, 4$. For all target functions, larger values of p correspond to more variation in the function outputs, and higher complexity for the approximation tasks. Fig. 2 shows an example of approximation result.

Model Initialization The hidden layers are chosen as a ResNet with depth 10, width 8, output dimension 2^p. We use the Box initialization [12] for the ResNet weights. The basis polynomials are the truncated Taylor monomials: for piecewise triangular waves, the polynomials have maximum

degree 1; for piecewise quadratic waves, the polynomials have maximum degree 2; for sine functions, we experiment with different maximum polynomial degrees between 2 and 16. The polynomial coefficients $\mathbf{c}$ are initialized from a standard normal distribution.

5 Discussion

Training, Validation, and Testing Data Distribution Since the training datasets and the corresponding validation and testing datasets are sampled from the same domain and evaluated with the same target function, the training loss, validation loss, and testing loss are highly correlated (see Fig. 3). Thus, we cannot perform early stopping in the training process. It is also difficult to predict the convergence of the loss (and, in turn, the convergence of the approximation error) beyond the fixed number of epochs that we have experimented with. In the following discussions, we will only consider the training loss as a representative of the model performance.

Advantage of Minibatch Training Our alternative strategy leverages minibatch gradient descent and the Adam optimizer with learning rate $\alpha = 0.05$. The left subfigure in Fig. 4 demonstrates the advantage of minibatch gradient descent. Despite the high variance in the training loss, minibatch gradient descent helps avoid local minima in the loss landscape during loss minimization. A comparison of various minibatch sizes (see Fig. 4) leads to the choice of minibatch size 128 for our experiments.

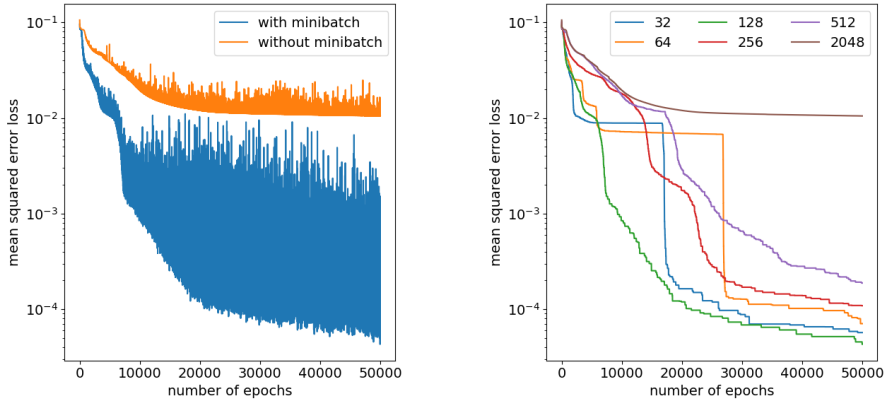


Figure 4: Training loss for target function $T^2(x; 3)$ under different minibatch training strategies. The loss curves in the right subfigure are labeled by minibatch sizes, and their y-coordinates are plotted as the minimal training loss until the current training epoch for better visualization.

Effect of Polynomial Degree With the target function $f(x) = 8\pi x$ and fixing $N = 16$, we study the effect of maximal degree for the polynomial component of POUnets on the loss convergence.

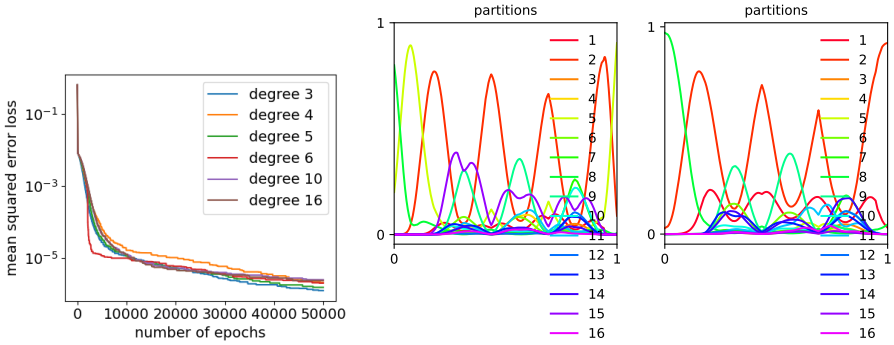
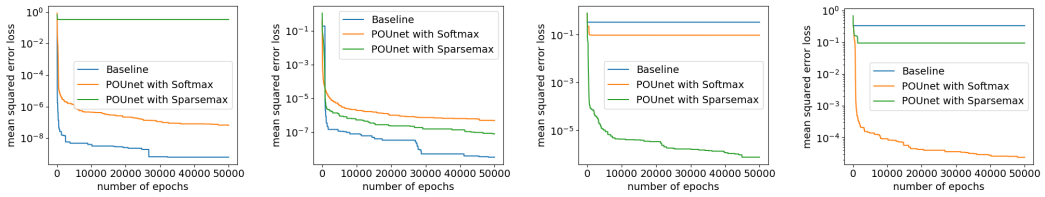


Figure 5: Training loss for $f(x) = 8\pi x$ with different maximal degrees for the polynomial component (left) and a comparison of softmax activations for maximal degree 6 (middle) and 16 (right).

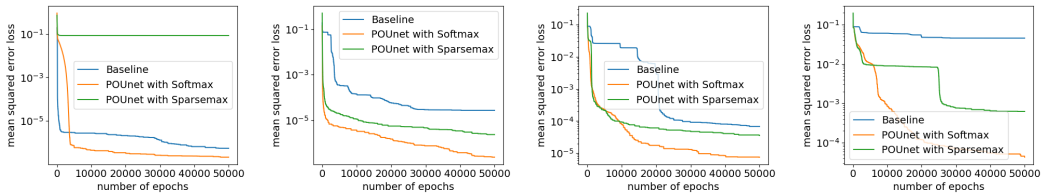
Theoretically, fixing the other model parameters and hyperparameters, a POUnet with a larger maximal polynomial degree is strictly more expressive than a POUnet with a smaller maximal polynomial degree. However, within 50000 training epochs, the models with different maximal polynomial degree have training loss curves that converge at similar rates and to similar values. Further experiments need to be conducted to evaluate the lowest loss value that each model is able to reach after much more training epochs.

The partitions (the activations of the softmax layer) are also similar among different maximal polynomial degrees. This is an ideal property of the POUnets model in order to separate the responsibilities of the DNN component and the polynomial component.

Comparison with Baseline Model We compare the training loss of the POUnets with that of a set of baseline model, where each baseline model is a fully connected network of the same size as the DNN component of the corresponding POUnet, with input dimension 1 and output dimension 1. We use the same training strategy (minibatch gradient descent with minibatch size 128) for the experiments. The training results are summarized in Fig. 6. The relative ℓ-2 errors of the trained models, as a common evaluation metric in approximation theory, are tabulated in Appendix A.



(a) $f(x) = T(x; 1)$ (b) $f(x) = T(x; 2)$ (c) $f(x) = T(x; 3)$ (d) $f(x) = T(x; 4)$



(e) $f(x) = T^2(x; 1)$ (f) $f(x) = T^2(x; 2)$ (g) $f(x) = T^2(x; 3)$ (h) $f(x) = T^2(x; 4)$

Figure 6: A comparison of training loss between the POUnets and baseline fully-connected networks, with target functions including triangular waves (top row) and quadratic waves (bottom row). The training loss is evaluated as the mean squared error (2).

We observe that the POUnets achieve lower mean squared errors than the baseline models in most experiments, except for the two simplest piecewise linear target functions. In general, POUnets result in more accurate approximations and more robust performance than the baseline models. POUnets with a softmax layer as the DNN output tend to provide more robust approximations than the alternative POUnets architecture with a sparsemax layer. This could be a result of discontinuities in the sparsemax activations and gradients.

6 Conclusion

We explored the POUnets model and numerically tested its performance in approximating common waveform functions including triangular waves, quadratic waves, and sine functions. We proposed an alternative implementation of the polynomial component as a custom layer, which allows for the use of minibatch gradient descent as an optimal training strategy, and significantly simplifies the hyperparameter tuning process. This is advantageous for the applications of POUnets in approximating a wide range of basis functions and their linear combinations. Using a consistent set of hyperparameters can increase the robustness of the model when the target functions are highly complex.

For ongoing and future work, we will experiment with variations of the current POUnets model, by (i) regularizing the partition produced by the activations of the DNN output layer; (ii) further improving the training strategy for more consistent model performance.

References

[1] Kookjin Lee, Nathaniel A Trask, Ravi G Patel, Mamikon A Gulian, and Eric C Cyr. Partition of unity networks: deep hp-approximation. *arXiv preprint arXiv:2101.11256*, 2021.

[2] Trenton Kirchdoerfer and Michael Ortiz. Data-driven computational mechanics. *Computer Methods in Applied Mechanics and Engineering*, 304:81–101, 2016.

[3] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[4] Jonathan B Freund, Jonathan F MacArt, and Justin Sirignano. Dpm: A deep learning pde augmentation method (with application to large-eddy simulation). *arXiv preprint arXiv:1911.09145*, 2019.

[5] Kailai Xu and Eric Darve. Physics constrained learning for data-driven inverse modeling from sparse observations. *arXiv preprint arXiv:2002.10521*, 2020.

[6] Jonathan R Holland, James D Baeder, and Karthikeyan Duraisamy. Field inversion and machine learning with embedded neural networks: Physics-consistent neural network training. In *AIAA Aviation 2019 Forum*, page 3200, 2019.

[7] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[8] Dmitry Yarotsky. Optimal approximation of continuous functions by very deep relu networks. In *Conference on Learning Theory*, pages 639–649. PMLR, 2018.

[9] Joost AA Opschoor, Philipp C Petersen, and Christoph Schwab. Deep relu networks and high-order finite element methods. *Analysis and Applications*, 18(05):715–770, 2020.

[10] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[12] Eric C Cyr, Mamikon A Gulian, Ravi G Patel, Mauro Perego, and Nathaniel A Trask. Robust training and initialization of deep neural networks: An adaptive basis viewpoint. In *Mathematical and Scientific Machine Learning*, pages 512–536. PMLR, 2020.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Kookjin Lee, Nathaniel A Trask, Ravi G Patel, Mamikon A Gulian, and Eric C Cyr. https://github.com/rgp62/leastsquares_pou, 2021.

A Summary of Numerical Experiments

Target function	Baseline network	POUnet with Softmax	POUnet with Sparsemax
$T(x;1)$	$\mathbf{2.739 \times 10^{-4}}$	6.091×10^{-4}	1.000
$T(x;2)$	$\mathbf{1.589 \times 10^{-3}}$	1.915×10^{-3}	1.647×10^{-3}
$T(x;3)$	1.000	5.350×10^{-1}	$\mathbf{8.953 \times 10^{-3}}$
$T(x;4)$	1.000	$\mathbf{1.556 \times 10^{-2}}$	5.329×10^{-1}
$T^2(x;1)$	1.766×10^{-3}	$\mathbf{1.576 \times 10^{-3}}$	6.452×10^{-1}
$T^2(x;2)$	1.342×10^{-2}	$\mathbf{3.433 \times 10^{-3}}$	4.122×10^{-3}
$T^2(x;3)$	3.739×10^{-2}	$\mathbf{1.058 \times 10^{-2}}$	1.840×10^{-2}
$T^2(x;4)$	4.807×10^{-1}	$\mathbf{2.500 \times 10^{-2}}$	6.200×10^{-2}
$\sin(2\pi x)$	$\mathbf{2.554 \times 10^{-3}}$	4.597×10^{-3}	N/A
$\sin(8\pi x)$	7.776×10^{-1}	$\mathbf{5.343 \times 10^{-3}}$	N/A

Table 1: A summary of relative ℓ-2 errors in approximating waveform target functions, using the baseline (fully-connected) networks, the POUnets with softmax as the last layer of the DNN component, and the POUnets with sparsemax as the last layer of the DNN component.