SAND2022-12635C

Polynomial-Spline Networks with Exact Integrals and Convergence Rates

Jonas A. Actor
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, USA
jaactor@sandia.gov

Andy Huang
Radiation and Electrical Science
Sandia National Laboratories
Albuquerque, NM, USA
ahuang@sandia.gov

Nat Trask
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, USA
natrask@sandia.gov

***Abstract*—Using neural networks to solve variational problems, and other scientific machine learning tasks, has been limited by a lack of consistency and an inability to exactly integrate expressions involving neural network architectures. We address these limitations by formulating a polynomial-spline network, a novel shallow multilinear perceptron (MLP) architecture incorporating free knot B-spline basis functions into a polynomial mixture-of-experts model. Effectively, our architecture performs piecewise polynomial approximation on each cell of a trainable partition of unity while ensuring the MLP and its derivatives can be integrated exactly, obviating a reliance on sampling or quadrature and enabling error-free computation of variational forms. We demonstrate hp-convergence for regression problems at convergence rates expected from approximation theory and solve elliptic problems in one and two dimensions, with a favorable comparison to adaptive finite elements.**

***Index Terms*—Constructive approximation; mixture of experts; splines; integration; quadrature; scientific machine learning**

I. Introduction

Exact integration of neural networks is useful in many contexts, such as computing statistical moments of estimators, operator regression, Bayesian machine learning techniques, and scientific machine learning. Specifically concerning this last task, while deep neural networks (DNNs) have been proposed for solving partial differential equations (PDEs) and scientific machine learning, such methods fail to converge to PDE solutions in practical settings [1], [2]. While other machine learning (ML) methods (such as [3]) demonstrate $hp-$ convergence for regression problems, in terms of spatial resolution h and polynomial degree p, error analysis of these methods is complicated due to the *variational crime* [4] of using inexact quadrature. Such issues have led to the popularity of simpler to implement but more difficult to analyze collocation schemes [5], [6].

N. Trask and J. Actor acknowledge funding under the DOE ASCR PhILMS center (Grant number DE-SC001924) and the DOE Early Career program. A. Huang acknowledges function under the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-mission 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. 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.

In this work, we introduce polynomial-spline networks, a mixture-of-experts (MOE) shallow multilinear perceptron (MLP) model that combines gating functions composed of convex combinations of B-spline basis functions, with polynomial experts localized to each cell of the partition. The spline gating functions lead to a piecewise polynomial approximation with explicitly parameterized support, creating closed-form expressions for the integral of both our polynomial-spline model and its derivatives. As such, this provides a foundation for other problems requiring integration, e.g. estimation of statistical moments for probability measures, or novel loss functions and regularizers. Concurrently, our polynomial-spline networks obtain the theoretical $hp-$ convergence rates provided by classical approximation theory methods, guaranteeing convergence for practical ML and scientific machine learning problems. In Section II, we describe the construction of our polynomial-spline networks, outlining the theory of how our approach preserves $hp-$convergence and exact integration, while in Section III, we demonstrate that our proposed approach, unlike many other ML methods, achieves these proposed properties and convergence rates.

A. Related Work

Our approach admits interpretation as a mixture of experts (MOE) shallow MLP network with a gating layer constructed from a convex combination of B-spline basis functions, instead of a softmax activation. Extensive literature (and additional nomenclature) discusses such MOE models, both separately and in conjunction with MLPs [7]–[9]. Like MLPs and other neural networks, our architecture is parameterized by trainable variables in hidden layers, composing linear and nonlinear operations, where the hidden activation functions are built using a trainable piecewise-linear feature space akin to free-spline interpolation [10]. However, we explicitly enforce that our piecewise-linear encodings build a partition of unity, which in turn feeds into the final nonlinear activation function which, in our case, is multiplication and evaluation of a polynomial. Instead of using multiple layers with rectified linear units (ReLUs) to build such an intermediate piecewise linear feature space, like in a traditional deep neural network (DNN) architecture, we proceed directly using a single hidden layer of splines, as in shallow MLPs.

Other classes of MOE models using partitions of unity, such as partition of unity networks (POUNets) [3], have attempted to build $hp-$convergent solutions to regression or variational PDE problems, instead of e.g. performing kernel density estimation. The provided hp-convergence rates of POUNets are desirable, in that they are comparable to error rates from other approximation methods such as finite element analysis. Our approach differs from that of POUNets in that we build expert models out of convex combinations of B-spline basis functions, instead of the MLP or radial basis function network considered in [3]. By working with convex combinations of B-splines, our approach admits closed form expressions for integrals in terms of the B-spline knots. We thus preserve the $hp-$convergence of POUNets while enabling exact integration, even in high-dimensional settings. Compared to traditional B-spline bases the convex combination identifies an optimal compressed subspace for nonlinear approximation.

Additionally, our convex combinations of B-splines provide natural extensions of well-known results interpreting ReLU networks as continuous piecewise linear (CPWL) functions [11]. Previous works use splines to study approximation properties of DNNs, treating e.g. ReLU networks as max-affine splines [12] or as P1 finite elements [13]. In high-dimensions, the CPWL interpretation of ReLU networks is not tractable for quadrature, as the geometrically complex piecewise linear regions form non-convex polyhedral domains and do not admit a closed-form description of their support. Additionally, while approximation rates for ReLU networks have been derived [14], [15], such rates are rarely achieved in practice, especially for data-driven problems.

Relatedly, training ReLU DNNs is similar in scope to the NP-hard problem of finding an optimal spline interpolant with trainable knots [12], [16], and the similar problem of adaptively finding an optimal mesh for discretizing PDEs. In such mesh adaptivity problems, the mesh adaptivity is guided by an energy functional relating to the discretization points, whose minimizer is the optimal adaptive mesh. Such an energy function again necessitates integrating over the variational form of the PDE [17]–[19], which precludes their use in training DNNs, unless one is able and willing to commit a variational crime.

Some DNN-based approaches for solving PDEs skirt the issue of integration by adapting a collocation PDE residual (e.g.physics-informed neural networks (PINNs) [5] and related methods). While effective, this approach requires strong regularity requirements and more involved mathematical analysis beyond the standard Lax-Milgram theory [20]. Alternatively, the Deep Ritz method uses as a loss the Euler-Lagrange functional of the relevant variational problem, but ultimately resorts to sampling-based methods for integration [21]. As a result, the convergence of the loss function is dominated by the error in Monte Carlo integration, and variational crimes complicate the already complex landscape of approximation error, optimization error, and stability theory. An important practical feature of preserving the variational form is that training evolves along the manifold of optimal fits to data, similar to the least-squares gradient descent optimizer [22].

II. Formulation

Let $\Omega \subset \mathbb{R}^d$ be a closed, compact domain, where d is the spatial dimension; assume for simplicity that $\Omega = [0,1]^d$. Let $\mathbb{P}^B(\Omega)$ be the space of polynomials of degree up to B on Ω, with basis $\{p_\beta\}_{\beta=1,\dots,d_P=\dim(\mathbb{P}^B(\Omega))}$. We define a *polynomial-spline network* $y : \Omega \to \mathbb{R}$ via the expression

$$y(x) = \sum_{\alpha=1}^{N_{\text{cells}}} \left(\sum_{\gamma=0}^{N_{\text{splines}}} w_{\alpha,\gamma}\phi_\gamma(x) \right) \left(\sum_{\beta=1}^{d_P} c_{\alpha,\beta} p_\beta(x) \right). \quad (1)$$

In this expression, the functions $\phi_\gamma : \Omega \to \mathbb{R}$ are B-spline basis functions, parameterized by a set of knots $\{t_\gamma\} \subset \Omega$. Additionally the coefficients $w_{\alpha,\gamma}$ are constrained so that for all γ, $\sum_{\alpha=1}^{N_{\text{splines}}} w_{\alpha,\gamma} = 1$ and $w_{\alpha,\gamma} \geq 0$. The coefficients $c_{\alpha,\beta}$ are unconstrained. When clear, the bounds for the summations in Equation (1) are dropped for convenience.

The functions $\varphi_\alpha(x) = \sum_{\gamma=1}^{N_{\text{splines}}} w_{\alpha,\gamma}\phi_\gamma(x)$ form a partition of unity (POU) of N_{cells} partitions, following from convexity of $w_{\alpha,\gamma}$ and the fact that B-spline basis functions form a partition of unity [23]. Thus, polynomial-spline networks are MOE models, where convex combinations of B-splines serve as gating functions for $B^{th}-$ order polynomial experts. In the case that $N_{\text{cells}} = 1$, training this architecture reduces to polynomial approximation, as the polynomial-spline network becomes

$$\begin{aligned} y(x) &= \left(\sum_\gamma w_\gamma \phi_\gamma(x) \right) \left(\sum_\beta c_\beta p_\beta(x) \right) \\ &= 1 \cdot \left(\sum_\beta c_\beta p_\beta(x) \right) \\ &= \sum_\beta c_\beta p_\beta(x). \end{aligned}$$

Similarly, in the case $B = 0$ and $N_{\text{cells}} = N_{\text{splines}}$, training this architecture reduces to free-knot spline approximation, since the network becomes

$$\begin{aligned} y(x) &= \sum_\alpha \left(\sum_\gamma w_{\alpha,\gamma}\phi_\gamma(x) \right) (c_\alpha) \\ &= \sum_\alpha \sum_\gamma c_\alpha w_{\alpha,\gamma}\phi_\gamma(x), \end{aligned}$$

and setting $w_{\alpha,\gamma} = \delta_{\alpha\gamma}$ reduces the architecture to

$$y(x) = \sum_\alpha c_\alpha \phi_\alpha(x).$$

Therefore, we expect our polynomial-spline network to exhibit some form of both $h-$ and $p-$ refinement, as the number of partitions and polynomial degree increase, respectively, following the convergence rates established via numerical analysis; formal proofs of these convergence rates are identical to the proofs for polynomial and spline approximants, respectively, and can be found in most numerical analysis textbooks, such as [23].

In practice, we limit ourselves to only using B1-splines. In doing so, we make the max-affine spline interpretation of deep ReLU networks [12] and free-spline interpolation [10] connections explicit, in that we directly construct the underlying spline to partition Ω. Doing so allows us to construct closed-form expressions for the integrals (and integrals of derivatives) of the polynomial-spline network; deriving such expressions is tedious but feasible for higher-order splines.

We outline how to construct analytic expressions for the integral of y in the case of the functions ϕ_γ being B1-spline basis functions and our domain $\Omega = [0, 1]$. First, note that the B1-spline basis functions are described entirely by the set of knots $\{t_\gamma\}_{\gamma=0,\dots,N_{\text{splines}}}$, with $t_0 = 0$ and $t_{N_{\text{splines}}} = 1$, with relation to ϕ_γ in that $\phi_\gamma(t_\gamma) = 1$ and $\phi_\gamma(t_\beta) = 0$ for all $\beta \neq \gamma$. By construction, when restricted to the interval $[t_{\gamma-1}, t_\gamma]$, the polynomial-spline network y is a polynomial of degree $B+1$. Letting $\{q_i\}_{i=1,\dots,d_P+1}$ be a basis for $\mathbb{P}^{B+1}([t_{\gamma-1}, t_\gamma])$, we express y restricted to our interval in this basis, i.e.

$$\begin{aligned} y(x) &= \sum_\alpha \left(w_{\alpha,\gamma-1}\phi_{\gamma-1}(x) + w_{\alpha,\gamma}\phi_\gamma(x)\right)\left(\sum_\beta c_{\alpha\beta}p_\beta(x)\right) \\ &:= \sum_{i=1}^{d_P+1} d_i q_i(x) \end{aligned}$$

for coefficients d_i, which are closed-form expressions of the coefficients $w_{\alpha,\gamma}$, t_γ, and $c_{\alpha\beta}$. Therefore, our integral becomes

$$\begin{aligned} \int_\Omega y(x)dx &= \sum_{\gamma=1}^{N_{\text{splines}}} \int_{t_{\gamma-1}}^{t_\gamma} y(x)dx \\ &= \sum_{\gamma=1}^{N_{\text{splines}}} \sum_{i=1}^{d_P+1} d_i \int_{t_{\gamma-1}}^{t_\gamma} q_i(x)dx. \end{aligned}$$

Choosing the monomial basis $q_i(x) = x^{i-1}$,

$$\begin{aligned} \int_\Omega y(x)dx &= \sum_{\gamma=1}^{N_{\text{splines}}} \sum_{i=1}^{d_P+1} d_i \int_{t_{\gamma-1}}^{t_\gamma} x^{i-1}dx \\ &= \sum_{\gamma=1}^{N_{\text{splines}}} \sum_{i=1}^{d_P+1} \frac{d_i}{i}\left(t_\gamma^i - t_{\gamma-1}^i\right). \end{aligned}$$

Since we can explicitly calculate expressions for d_i, and all other values are known weights in our network, we can use the above formula to directly integrate y. Similar expressions are derived in the same way for ∇y, or for the calculation of moments involving y to a power, by expressing the integrand in terms of a polynomial expansion in terms of the polynomials q_i and then integrating separately upon the support of each B1-spline basis function.

III. Experiments

We demonstrate the effectiveness of our architecture on two sets of problems: regression problems and variational problems. We choose these sets of problems as to demonstrate the $hp-$convergence properties of our approach, and to highlight the capabilities of having a network possessing exact integration properties. Our implementation of polynomial-spline neural networks uses TensorFlow [24], with FEM comparisons in FEniCS [25].

A. Construction and Training

To build our polynomial-spline network, we build B1-spline basis functions for Ω as a tensor product of 1D B1-spline basis functions along each dimension. For each dimension, we construct a B1-spline layer, whose knots are parameterized to accommodate TensorFlow backwards differentiation during training. The general expression for a hat function built via ReLU functions is given in [13]. During training however, knots may become unordered, leading to a problematic inversion of elements. To prevent this concern, we track the relative position between knots rather than the locations themselves, constraining them to span the extant Ω.

For the regression problems, we train on a random uniform set of 1000 points, and we validate our model on a separate random uniform set of the same size. For the variational problems, there are no data sites involved, with the loss defined via the closed form expression for the energy. For all problems we use the Adam optimizer [26]. Our loss for the regression problems is mean squared error (MSE), while for the variational problems our loss is the Euler-Lagrange functional i.e. the Ritz energy. More details are available in Appendix A. All code is run on a 64-core Intel Xeon Gold CPU using 1 NVIDIA Tesla V100 GPU with 10GB memory.

We expedite the training of our models by using the least-squares gradient descent optimizer (LSGD) [22]. We define the function $\Phi : \mathbb{R}^d \to \mathbb{R}^{N_{\text{cells}}d_P}$ with each output component given as $\Phi_{\alpha\beta} : x \mapsto \left(\sum_\gamma w_{\alpha,\gamma}\phi_\gamma(x)\right)p_\beta(x)$; Equation 1 is then written as

$$y(x) = c^T\Phi(x)$$

for a vector of coefficients $c \in \mathbb{R}^{N_{\text{cells}}d_P}$. For regression problems, the LSGD solver adds a least-squares solve for c between each gradient step of the first-order optimizer. We wrap this least-squares solve call into a custom TensorFlow layer so that the operation is embedded in the network's TensorFlow graph directly, enforcing that all outputs of the network lie on the manifold of best-fit solutions regarding the coefficients of the outermost layer.

For regression, LSGD solves the least-squares problem, given batch data $\{x_i, y_i\}_{i=1,\dots,\text{batch size}}$ the problem

$$\min_c \left\|y - c^T\Phi(x)\right\|_F^2.$$

For variational problems, the least-squares problem that we solve is the variational problem that corresponds to the Euler-Lagrange functional; see the discussion below about our variational problems for more details.

B. Problems

To demonstrate the effectiveness of our architecture, we pose two sets of problems. First, we deploy our architecture on regression problems to evaluate consistency. After, we progress to solving variational problems.

1) Regression: We test our architecture on two 1D regression problems:

1) $f(x) = \sin(2\pi x)$ for $x \in [0, 1]$
2) $f(x) = \left|\sin(3\pi x^2)\right| + \left|\cos(5\pi x^2)\right|$ for $x \in [0, 1]$

In Problem 1, we expect to see both $h-$ and $p-$ refinement, i.e. increasing the number of partitions or increasing the degree of polynomial approximation, respectively, should improve the approximation. In Problem 2, we expect $h-$ refinement to improve our approximation but $p-$ refinement to not, since the function f in this case is only piecewise smooth. However, if we have a sufficient number of knots (i.e. at least one per piece of f), our POU cells should adaptively during training recover the locations where the $\sin$ or $\cos$ terms change sign, and that on each piece, we expect $p-$ refinement to improve our approximation.

a) Problem 1.: We perform three experiments to demonstrate the $hp-$ refinement properties of our network. First, we fix $N_{\text{cells}} = 1$ to verify that our model compares favorably to polynomial approximation with regards to $p-$ refinement. Second, we fix the polynomial degree $B = 0$ to verify that our model compares favorably to spline approximation with regards to $h-$ refinement. Third, we test our model for a variety of parameters for polynomial degree, number of cells, and number of spline knots, to verify simultaneous $hp-$ refinement and to test that our model can capture the solution to this regression problem up to machine precision.

First, we consider Problem 1 with $N_{\text{cells}} = 1$; in this case, we expect our model to return the best polynomial approximation of the specified degree. Results are shown in Figure 1; all three lines plotted in the figure nearly coincide, showing that in this limit our model maintains the $p-$ refinement properties of polynomial approximation. The staircase phenomenon is due to the function $f(x) = \sin(2\pi x)$ being an odd function, and as such the best polynomial approximation (and our model) only improves when the polynomial degree is increased to an odd power. Second, to verify the $h-$ refinement properties of our architecture, we set the polynomial degree $B = 0$ and compare our results to a piecewise linear spline approximation with uniform knots. Error plots are shown in Figure 1. In this case, we see the roughly the same rate at which the error decreases as we increase the number of knots in the network and in our spline approximation, until our model plateaus due to the instability of the backwards differentiation in the LSGD layer. Finally, when using $h-$ and $p-$ refinement simultaneously the polynomial-spline network is capable of achieving machine precision accuracy. Results are shown in Figure 2. We see that for sufficient spatial resolution and polynomial degree, we achieve mean-squared errors of order 10^{-8} or better using our proposed network, which is the precision limit given by our least-squares implementation in the LSGD layer.

b) Problem 2.: We perform two sets of experiments for Problem 2. First, we perform the same experiment as for Problem 1 to test $h-$ refinement even in the case of a non-smooth target function. Second, we test whether our network can find the discontinuities in the derivative of f in a way that is comparable to piecewise polynomial approximation. As in

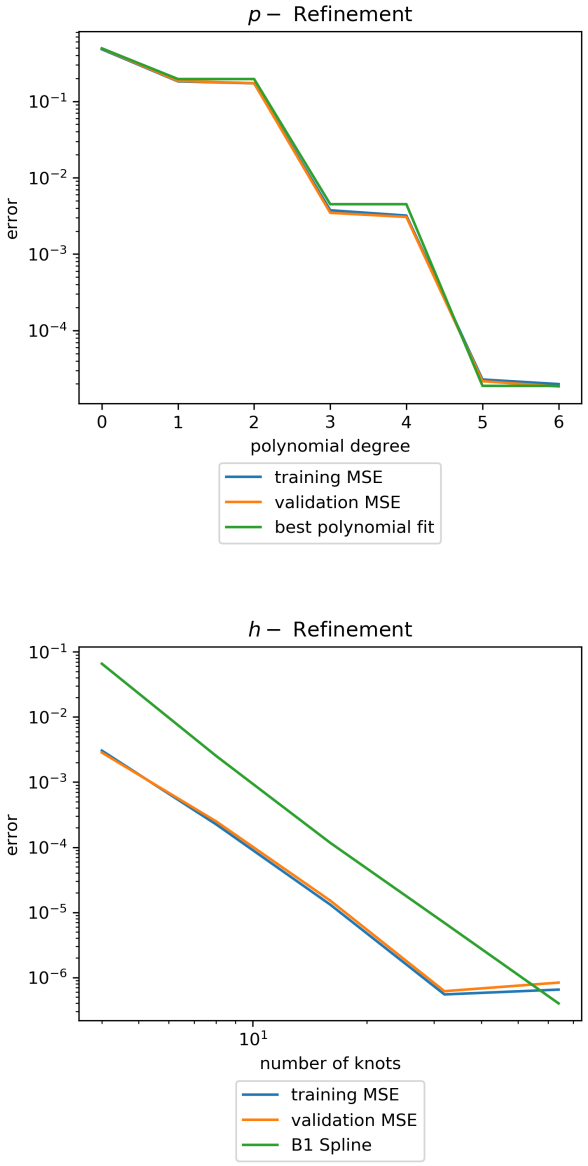

Fig. 1. Results of $p-$ refinement (top) and $h-$ refinement (bottom) for regression Problem 1. p-refinement is performed with $N_{\text{spline}} = 4$ and $N_{\text{cells}} = 1$, while h-refinement is performed with $N_{\text{cells}} = N_{\text{spline}}$ and polynomial degree $B = 0$. The errors for the training and validation sets coincide with the error for the best polynomial approximation on the p-refinement plot; the x-axis for the $h-$refinement plot maintains a logarithmic scale.

the previous problem, the training and validation error curves are extremely similar, so we only plot validation errors in the rest of the figures in this section.

First, as before, we compare our network's performance when the polynomial degree $B = 0$ and compare to a spline approximation with the same number of knots. In Figure 3, we see the error in our approximations for increasingly larger numbers of knots; we observe a roughly log-linear decrease in mean squared error as we double the number of knots (and

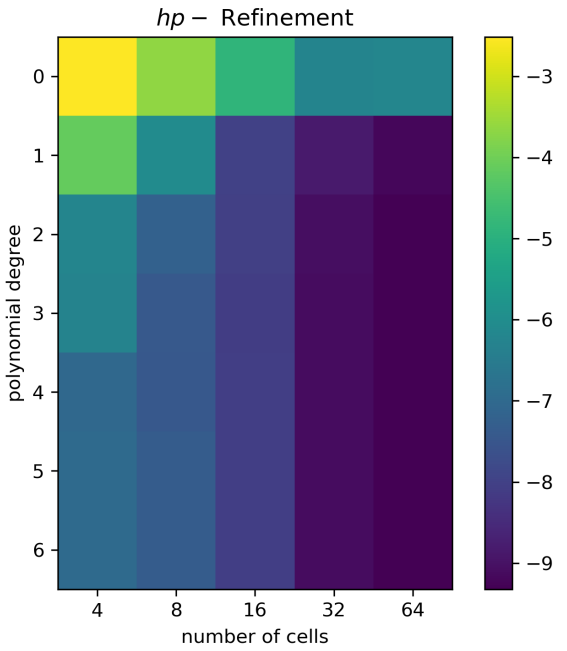

Fig. 2. Results for $hp-$ refinement on regression Problem 1. The color bar maps the $\log_{10}$ of the mean squared error for the specified number of POU cells and polynomial degree.

POU basis functions) in our network, as expected.

Second, we compare the results of our model with comparable piecewise polynomial approximation problems. We specifically compare to two piecewise polynomial approximations, one where polynomials are fit upon uniform pieces, and the other fit to pieces where the derivative of f is discontinuous. As both of these piecewise polynomial approximants fit a total of 8 polynomials, we compare these results to our model with $N_{\text{cells}} = 8$, which fits 8 polynomials in the LSGD layer. Results are shown in Figure 3.

Our polynomial-spline network outperforms these models for all polynomial degrees, efficiently capturing the discontinuous derivatives in the target function. The success of the polynomial-spline network plateaus at an accuracy of $O(10^{-6})$ due to the limitations of the numerical stability in the automatic differentiation of the least-squares solve operation in the LSGD layer.

2) Variational Problems: We test our architecture on two variational problems:

3) 1D Poisson problem with Dirichlet boundary conditions:

$$\begin{aligned} -d^2u &= 2 \quad \text{on} \quad \Omega = [0,1] \\ u &= 0 \quad \text{at} \quad \partial\Omega = \{0,1\}. \end{aligned} \tag{2}$$

4) 2D Poisson problem on a slit domain:

$$\begin{aligned} -\Delta u &= 0 \quad \text{on} \quad \Omega = [-1,1]^2 \\ u &= g(r,\theta) \quad \text{on} \quad \Gamma = \partial\Omega \cup [0,1]\times\{0\}, \end{aligned} \tag{3}$$

where $g(r,\theta) = \sqrt{r}\sin\left(\frac{\theta}{2}\right)$ is given in polar coordinates.

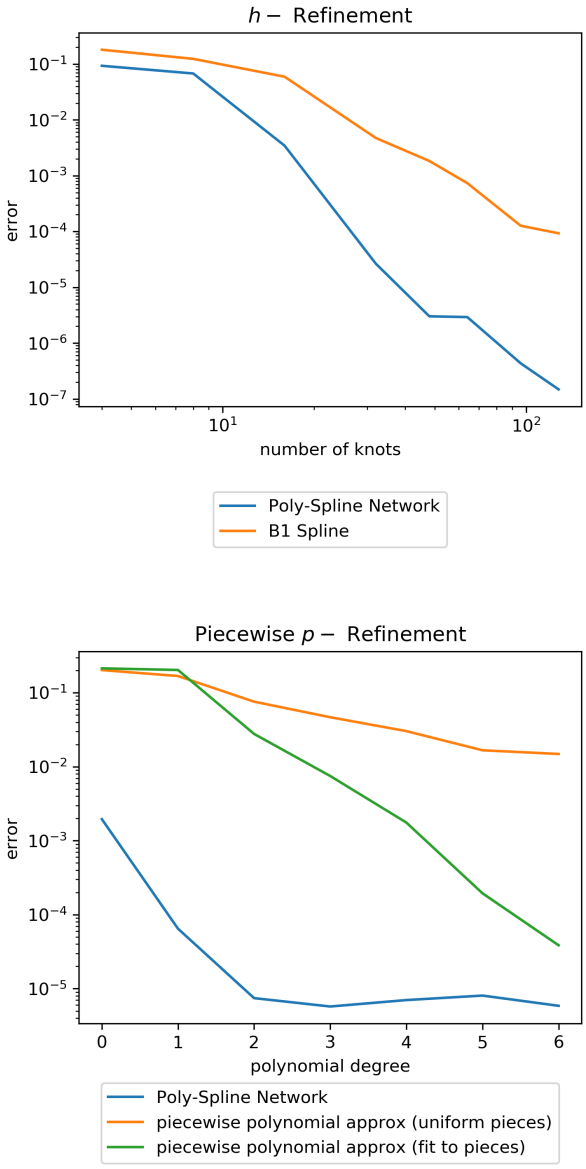

Fig. 3. Results of using h-refinement (top) and p-refinement (bottom) for Problem 2. $h-$refinement performs approximation with $B = 0$ i.e. spline approximation, with $N_{\text{spline}} = N_{\text{cells}}$ plotted against the achieved mean squared error. $p-$ refinement uses $N_{\text{cells}} = 8$

In Problem 3, we expect to recover the true solution $u(x) = x(1-x)$ exactly (up to machine precision), assuming on each POU cell we are fitting a polynomial of sufficient degree. In Problem 4 we expect our problem to recover the known singularity at the origin similar to adaptive mesh refinement methods [17]–[19]; the true solution to this problem is known to be $u = g(r,\theta)$.

For each of these variational problems, we minimize the associated Euler-Lagrange functional [27]. For example, Prob-

lem 3 is solved by minimizing the Euler-Lagrangian "loss"

$$L(u) = \int_\Omega \frac{1}{2} \|\nabla u\|^2 \, dx + \beta \left(u(0)^2 + u(1)^2\right), \qquad (4)$$

where $\beta > 0$ is a penalty parameter to enforce our solutions satisfy our boundary conditions. For Problem 4, the Euler-Lagrange functional is

$$L(u) = \int_\Omega \frac{1}{2} \|\nabla u\|^2 \, dx + \beta \int_\Gamma (u - g(r,\theta))^2 ds. \qquad (5)$$

Instead of relying on Monte Carlo or sampling-based methods to evaluate these integral (e.g. [21]), we compute these integrals exactly by virtue of our architecture construction. Rather than using the analytic expressions for the integral, we employ Gaussian quadrature of degree $B + d + 1$, which computes the integrals exactly for polynomials of degree up to $2(B + d) + 1$, which is sufficient for exactly computing the integrals of u, u^2, and $\|\nabla u\|^2$ on the support of each B1 basis function; using quadrature allows us to exploit hardware acceleration during computation, since the model evaluation at the quadrature points can more efficiently use the GPUs. We note that since the location of the knots change over the course of training, the quadrature points change as well.

a) Problem 3: 1D Poisson Boundary Value Problem: For this problem, the LSGD layer solves the linear system corresponding to the weak form of Problem 3, which is given as follows. We define the matrix A and vector b by substituting $u(x) = c^T \Phi(x)$ into Equation 4, taking the derivative of Equation 4 with respect to c, and then setting the resulting expression to zero. In doing so, we arrive at the linear problem $Ac = b$, where

$$A = \int_\Omega d\Phi d\Phi^T + \beta \left(\Phi(0)\Phi(0)^T + \Phi(1)\Phi(1)^T\right) dx$$
$$b = \int_\Omega 2\Phi \, dx.$$

The integrals in the linear problem can be calculated exactly for a given set of knots, since upon each component of the B1-spline functions, $\Phi(x)$ is a polynomial of degree $B+d$ i.e. $B+1$. We can do so by deriving the closed-form solutions for the integrands of each interval between spline knots, or (more efficiently) by using Gaussian quadrature of degree $B+d+1$ as specified earlier.

When we use the training methods described earlier, we successfully solve our problem up to machine precision (or the limits of the accuracy of the least-squares backwards differentiability) nearly immediately when $B \geq 1$, which matches our expectation, since the true solution to our problem is a polynomial of degree 2. In the case $B = 0$, we recover the best P1-finite element solution to our problem, as seen in Figure 4.

b) Problem 4: 2D Poisson Slit Domain: By symmetry, it suffices to solve this problem on a reduced domain Ω' that is half the size of Ω: note that the solution to Problem 4 is symmetric about the x-axis. As a result, we solve the following equivalent problem:

$$\begin{aligned} -\Delta u &= 0 && \text{on} \quad \Omega' = [-1,1] \times [0,1] \\ \partial_n u &= 0 && \text{on} \quad \Gamma_N = [-1,0] \times \{0\} \\ u &= g(r,\theta) && \text{on} \quad \Gamma_D = \partial\Omega' \backslash \Gamma_N. \end{aligned} \qquad (6)$$

This formulation ensures that the slit in the domain aligns with the exterior of Ω'.

For this problem, the LSGD layer solves the linear system corresponding to the weak formulation of Problem 4, which by the same process as described for Problem 3, yields the linear problem of solving $Ac = b$, where

$$A = \int_{\Omega'} D\Phi D\Phi^T \, dx + \beta \int_{\Gamma_D} \Phi\Phi^T \, ds, \quad b = \beta \int_{\Gamma_D} g\Phi \, ds.$$

Note that the matrix A can be significantly smaller than the linear system involved in other scientific computing methods e.g. the finite element method. For P1 finite elements, the linear system would be of size $d_P \left(N_{\text{splines}}\right)^d \times d_P \left(N_{\text{splines}}\right)^d$, whereas in our case A is only of size $d_P N_{\text{cells}} \times d_P \times N_{\text{cells}}$. This reduction in size (and in the cost to solve such problems) arises since we fit polynomials on each partition and not upon each B1-spline basis function. This highlights that our approach provides a nonlinear construction of a reduced finite element space providing an optimal representation of the solution.

The integrals corresponding to A can be calculated via closed-form expressions, as before. However, since g is not a polynomial, we cannot expect exact integration for the integral that defines the vector b, and we either can project g into the correct polynomial space, or over-integrate with quadrature of a higher degree to maintain sufficient accuracy.

We compare our solution to results of solving this variational problem using P1 finite elements on three different meshes: two uniform tetrahedral meshes, and an adaptive tetrahedral mesh. The first uniform mesh (FEM U3) is a 3×3 mesh, chosen so that there are 16 degrees of freedom and 18 elements, which most closely matches our polynomial-spline construction by fitting linear functions on $N_{\text{cells}} = 16$ cells. The second uniform mesh (FEM U6) is 6×6, which is the closest match to the size of the linear system that our LSGD layer solves. The adaptive mesh (FEM A) is constructed by an adaptive solver, starting with the 3×3 uniform mesh and proceeding to adaptively refine the mesh until the number of degrees of freedom is closest to the size of our linear system. Near the singularity at $r = 0$, the optimal adaptive mesh's cells should shrink at a rate of approximately $\sqrt{r}$, where r is their distance to the origin [19]. Finite element comparisons are computed using FEniCS [25].

The results of our method are shown in Figure 5, with the L_2 error listed in Table I. We outperform the finite element solver on all of the comparable meshes. The plots in Figure 5 re-scale and translate our domain of interest Ω' from $[-1,1] \times [0,1]$ to $[0,1]^2$, with the slit occurring along the line segment $[0.5,1] \times \{0\}$ along the x-axis.

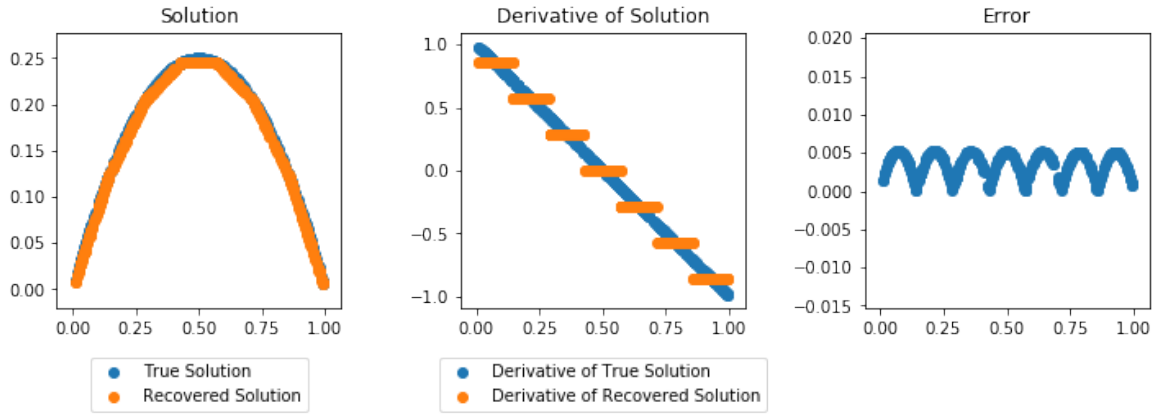

Fig. 4. Recovered solution to Problem 3 with $N_{\text{cells}} = 3$, $N_{\text{spline}} = 8$, and $B = 0$. We recover the best FEM approximation on a mesh with eight nodes i.e. 7 intervals, as seen in the plot of the derivative of the approximation in the center panel.

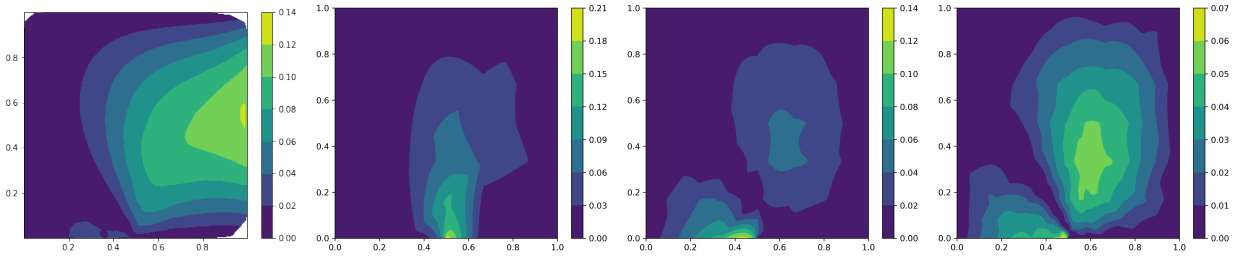

Fig. 5. Pointwise error of our model vs. finite element approximations on various meshes. From left to right: 1. Poly-Spline network; 2. FEM with uniform mesh (U3); 3. FEM with uniform mesh (U6); 4. FEM with adaptive mesh (A). Note the differences in scale for each plot.

Method	**Mesh Type**	**# Cells**	**Solve Size**	L_2 **Error**
Poly-Spline Network	Adaptive	16	48×48	0.0262
FEM U3	Uniform	18	16×16	0.0362
FEM U6	Uniform	72	49×49	0.0231
FEM A	Adaptive	120	72×72	0.0242

TABLE I
COMPARING OUR NETWORK VS. ADAPTIVE AND UNIFORM FINITE ELEMENT SOLUTIONS FOR PROBLEM 4.

The errors that arise in our solution accumulate near the boundary, whereas in the finite element approximations, the errors lie in the interior and near the singularity. This is because our Ritz loss enforcing the boundary condition via penalty, while the finite element method enforces Dirichlet conditions variationally. Still, L_2 error of our solution is lower than comparable finite element method solutions, though one could obtain better results working with a boundary conforming Galerkin framework.

IV. Discussion

Our polynomial-spline network compares favorably to traditional approximation methods, preserving the convergence rates of spline interpolation and polynomial regression. Our ability to learn the spline knots and perform free-knot spline interpolation, like MLPs and other deep learning methods, allows for adaptivity similar to adaptive mesh refinement methods. However, we obtain a reduced partition of space from the overparameterized fine B-spline grid allowing one to work with polynomials on each adaptively coarsened partition of unity; in this sense we obtain reduced-order model for the adaptive B-spline basis. As a result, the size of the linear system involved in the LSGD optimization step is smaller than if we were to depend on the spline basis functions directly. This reduction is particularly effective when $d > 1$, to avoid the curse of dimensionality. Additionally, the number of parameters in our model does not depend on the degree of the polynomial approximation, since the coefficient values for $c_{\alpha,\beta}$ are tabulated via LSGD. We thus obtain an hp-convergent variational framework for solving PDEs that circumvents variational crimes due to inexact quadrature.

References

[1] C. Beck, A. Jentzen, and B. Kuckuck, "Full error analysis for the training of deep neural networks," 2020.

[2] J. Han, A. Jentzen, and E. Weinan, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[3] K. Lee, N. A. Trask, R. G. Patel, M. A. Gulian, and E. C. Cyr, "Partition of unity networks: Deep hp-approximation," 2021.

[4] G. Strang, "Variational crimes in the finite element method," in *The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations*. Elsevier, 1972, pp. 689–710.
[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.
[6] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," 2020.
[7] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1177–1193, 2012.
[8] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.
[9] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.
[10] J. W. Siegel and J. Xu, "Approximation rates for neural networks with general activation functions," *Neural Networks*, vol. 128, pp. 313–321, 2020.
[11] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," in *International Conference on Learning Representations*, 2018.
[12] R. Balestriero and R. G. Baraniuk, "Mad max: Affine spline insights into deep learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 704–727, 2020.
[13] J. He, L. Li, J. Xu, and C. Zheng, "Relu deep neural networks and linear finite elements," *Journal of Computational Mathematics*, vol. 38, no. 3, pp. 502–527, 2020.
[14] R. DeVore, B. Hanin, and G. Petrova, "Neural network approximation," *Acta Numerica*, vol. 30, pp. 327–444, 2021.
[15] J. A. Opschoor, P. C. Petersen, and C. Schwab, "Deep relu networks and high-order finite element methods," *Analysis and Applications*, vol. 18, no. 05, pp. 715–770, 2020.
[16] D. L. Jupp, "Approximation to data by splines with free knots," *SIAM Journal on Numerical Analysis*, vol. 15, no. 2, pp. 328–343, 1978.
[17] I. Babŭska and W. C. Rheinboldt, "Error estimates for adaptive finite element computations," *SIAM Journal on Numerical Analysis*, vol. 15, no. 4, pp. 736–754, 1978.
[18] W. Gui and I. Babŭska, "The h, p and hp versions of the finite element methods in 1 dimension. part iii. the adaptive hp version." *Numerische Mathematik*, vol. 49, no. 6, pp. 659–683, 1986.
[19] A. Logg, K.-A. Mardal, G. N. Wells *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.
[20] Y. Shin, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes," *Communications in Computational Physics*, vol. 28, no. 5, p. 2042–2074, Jun 2020. [Online]. Available: http://dx.doi.org/10.4208/cicp.OA-2020-0193
[21] W. E and B. Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1–12, 2018.
[22] E. C. Cyr, M. A. Gulian, R. G. Patel, M. Perego, and N. A. Trask, "Robust training and initializatino of deep neural networks: An adaptive basis viewpoint," in *Mathematical and Scientific Machine Learning*. PMLR, 2020, pp. 512–536.
[23] M. J. D. Powell *et al.*, *Approximation theory and methods*. Cambridge university press, 1981.
[24] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/
[25] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The fenics project version 1.5," *Archive of Numerical Software*, vol. 3, no. 100, 2015.
[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.
[27] L. C. Evans, *Partial differential equations*, second edition. ed., ser. Graduate studies in mathematics ; Volume 19. Providence, R.I: American Mathematical Society, 2010.

APPENDIX

Training parameters and hyperparameters for each of the four problems are described below. For all problems, random NumPy calls are seeded by using a NumPy random number generator with a fixed seed of `seed=1234`. TensorFlow randomness is set via a global random seed of `seed=1235`, which is independent of the NumPy generator and is restarted each time a new network is trained. All problems use the Adam [26] optimizer, paired with LSGD.

A. Problem 1

Our polynomial-spline networks are constructed with spatial dimension $d = 1$, polynomial degrees $B \in \{0, 1, \ldots, 6\}$, and number of splines in the spline layer $N_{\text{splines}} \in \{4, 8, 16, 32, 64\}$. For each value of N_{splines}, we test with the number of POU cells $N_{\text{cells}} = \{1, 2, 4, 8, \ldots, N_{\text{splines}}\}$. The LSGD layer uses an L_2 regularizer of 10^{-10} as part of the least-squares solve, to protect against numerical instability in TensorFlow's backwards differentiation of the least squares function call. Optimization uses a learning rate of 5×10^{-3}, with a batch size of 1000, for 500 epochs, with MSE as the loss function.

B. Problem 2

Our polynomial-spline networks are constructed with spatial dimension $d = 1$, polynomial degrees $B \in \{0, 1, 2, 3\}$, and number of splines in the spline layer $N_{\text{splines}} \in \{4, 8, 16, 32, 64\}$. For each value of N_{splines}, we test with the number of POU cells $N_{\text{cells}} = \{2, 4, 8, \ldots, N_{\text{splines}}\}$. The LSGD layer uses an L_2 regularizer of 10^{-12} as part of the least-squares solve, to protect against numerical instability in TensorFlow's backwards differentiation of the least squares function call. All other parameter and hyperparameter choices are identical to those listed in Problem 1.

C. Problem 3

In Problem 3, we minimize the Euler-Lagrange loss

$$L(u) = \int_\Omega \frac{1}{2} \|\nabla u\|^2 \, dx + \beta \left(u(0)^2 + u(1)^2 \right). \tag{7}$$

The penalty parameter β in the Euler-Lagrange loss is set to $\beta = 1000$. For $B = 0$, we use a polynomial-spline network with $N_{\text{splines}} \in \{4, 8, \ldots, 64\}$ and with fixed $N_{\text{cells}} = 3$. We train for 1000 epochs with an initial learning rate of 0.01, reducing to 0.005 halfway through training.

D. Problem 4

In Problem 4, we minimize the Euler-Lagrange loss

$$L(u) = \int_\Omega \frac{1}{2} \|\nabla u\|^2 \, dx + \beta \int_\Gamma (u - g(r, \theta))^2 ds. \tag{8}$$

We use a polynomial-spline network with $B = 1$, with $N_{\text{splines}} = 9$ for both the x-axis splines and y-axis splines (recall our spline basis for $d > 1$ is formed via a tensor product of 1D splines), and with fixed $N_{\text{cells}} = 16$. We train for 3000 epochs with an initial learning rate of 0.01, reducing to 0.005 halfway through training and reinitialized after every 500 epochs. Otherwise, the optimizer parameters are the same as in Problem 3.