IMA journal title (to be updated during typesetting) (2021) **00**, 1–32
https://doi.org/DOI HERE

Equidistribution-based training of Univariate Free Knot Splines and ReLU Neural Networks: Revised

SIMONE APPELLA
Mathematical Sciences, University of Bath, Bath, BA2 7AY, UK

SIMON ARRIDGE
Computer Science, University College London, London, WC1E 6BT, UK

CHRIS BUDD*
Mathematical Sciences, University of Bath, Bath, BA2 7AY, UK

TEO DEVENEY
Computer Science, University of Bath, Bath, BA2 7AY, UK

YAN DU
Computer Science, University College London, London, WC1E 6BT, UK

AND

LISA MARIA KREUSSER
Mathematical Sciences, University of Bath, Bath, BA2 7AY, UK
*Corresponding author: mascjb@bath.ac.uk

[Received on Date October 2025]

Keywords: ReLU Neural networks; free knot splines; equidistribution; preconditioning, approximation theory.

We consider the problem of improving the accuracy, convergence, and conditioning of univariate nonlinear function approximations using (mainly) shallow neural networks (NN) with a rectified linear unit (ReLU) activation function. The standard L_2 based approximation problem is ill-conditioned and the behaviour of the optimisation algorithms used in training these networks degrades rapidly as the width of the network increases. This can lead to significantly poorer approximation in practice than we would expect from the theoretical expressivity of the ReLU NN architecture. Univariate shallow ReLU NNs and traditional approximation methods, such as univariate Free Knot Splines (FKS) span the same function space, and thus have the same theoretical expressivity. However, the FKS representation, both remains well-conditioned as the number of knots increases, and can be highly accurate if the knots are correctly placed. We leverage the theory of optimal piecewise linear interpolants to improve the training procedure for both a FKS and a ReLU NN. For the FKS we propose a novel two-level training procedure. First solving the nonlinear problem of finding the optimal knot locations of the interpolating FKS using an equidistribution approach. Then solving the nearly linear, well-conditioned, problem of finding the optimal weights and knots of the FKS. The training of the FKS gives insights into how we can train a ReLU NN effectively to give an equally accurate approximation. To do this we combine the training of the ReLU NN with an equidistribution based loss to find the breakpoints of the ReLU functions, this is then combined with preconditioning the ReLU NN approximation (to

take an FKS form) to find the scalings of the ReLU functions. This procedure leads to a fast, well-conditioned and reliable method of finding an accurate shallow ReLU NN approximation to a univariate target function. This method avoids spectral bias and is highly effective for a wide variety of functions. We test this method on a series of regular, singular, and rapidly varying target functions and obtain good results, realising the expressivity of the shallow ReLU network in all cases. We conclude that in the shallow case to gain full expressivity for the ReLU NN we must both find the optimal breakpoints (by equidistribution) *and* precondition the problem of finding the optimal coefficients. We then extend our results to more general activation functions, and to deeper networks.

1. Introduction

1.1. *Overview*

Approximation of a univariate target function $u(x)$, $x \in \Omega$ is an important application of *neural nets* (NNs), either in its own right, or as part of the solution of a differential equation. Indeed, a NN (regardless of its architecture) can be regarded as a function generator for instance in the context of neural operators [22], and can be used as means for solving both ordinary and partial differential equations [10], [20]. In general, for an vector x to a NN, the output of the NN is a nonlinear function $y(x)$ where the corresponding function y is defined on a suitable domain and has the same degree of smoothness as the NN activation functions. NNs approximate the target function $u(x)$ through a combination of linear operations and nonlinear/semilinear activation functions which results in function approximations which are nonlinear in their coefficients. The NN output $y(x)$ is trained to approximate $u(x)$. A key feature of NNs is that they are nonlinear approximators with provable expressivity, see [6], [12]. By the Universal approximation theorem [17] any continuous function on a compact set can be approximated by a certain neural network, with a continuous activation function, to any accuracy. A NN approximation also has the attractive feature that it can be 'self-adaptive' [10], so that it may be able to account for singularities and rapid variations in the target function. A commonly used activation function is the globally supported rectified linear unit (ReLU), defined as $\mathrm{ReLU}(x) = \max(x,0) = (x)_+$. This is often used in practice due to its efficiency and simplicity. ReLU NNs have a globally continuous and piecewise linear input-output relation. They are thus in one-dimension formally equivalent to a piecewise linear free knot spline (FKS), with the ReLU NN and FKS describing the same set of functions [6], [14]. Similar results hold in higher dimensions although the mapping one from one to another is significantly more subtle [6]. For a ReLU NN, the approximation error can be explicitly linked to the depth L and width W of the network, with exponential rates of (error) convergence with increasing depth L [28].

However, even with theoretical expressivity guarantees, this does not guarantee that the optimal solution is found by the training algorithm with the standard L_2^2 loss [16], [24], [26]. This is because the resulting approximation problem is both badly conditioned (leading to spectral bias), and highly non-convex. As a result when training we may either not see convergence, or convergence to a sub-optimal solution, with errors which are many orders of magnitude worse than they should be [16], [26], [29]. This is either because the optimal approximation with high expressivity is not found, or the training process is far too slow and is terminated too early. Similar behaviour is seen in the physics informed neural networks (PINNs) used to approximate the solution of differential equations [13][20]. In this paper we will draw on results from classical adaptive approximation theory to show how the training approach for such a NN can be improved significantly by using a two-level approach. Firstly we use an equidistribution based loss function to find the *breakpoints* of the ReLU NN where the derivative of the output of the

NN changes discontinuously. This is essential to give high expressivity for singular or rapidly varying target functions. Then we solve a preconditioned form of the NN representation to find the scaling coefficients to overcome the ill-conditioning and spectral bias of the usual training procedure [16], [24], [29]. We show in this paper that by doing these two steps we can achieve the theoretical expressivity of the NN for many challenging functions in cases where other training method fails to deliver a good approximation.

1.1.1. Classical approximation theory

Classical approximation theory has long considered the problem of efficiently approximating a univariate target function $u(x), x \in \Omega$ [8], [23]. Examples include the use of combinations of basis functions $\phi_i(x)$ (such as a hat-function) with *localised support*, for function interpolation or least squares approximation, such as the Galerkin methods used in the Finite Element Method. Such classical function approximations are usually *linear* in their coefficients, as the weights $w_i \in \mathbf{w}$ of a given set of basis functions are optimised and the function approximation lies in a linear function space. Typically the basis functions themselves are piece-wise polynomials (B-splines), defined over a *fixed* set of *knots* $k_i \in \mathbf{k}$ in one-dimension, where the form of the local polynomial approximation changes. This gives an approximation of the form

$$y(x) = \sum_{i=0}^{N-1} w_i \, \phi_i(x), \quad \text{where} \quad \phi_i(k_i) = 1, \quad \phi_i(k_j) = 0, \quad j \neq i. \tag{1.1}$$

These approximations have provable accuracy [7] and there exist efficient, and well-conditioned, algorithms for calculating the optimal weights (inverting a well conditioned Gram matrix to find the weights w_i) that reliably converge to a unique solution, often either solving well-conditioned linear systems of equations (directly or by iteration), or by a simple quadratic minimisation problem. However, such linear methods often lack accuracy, and may require a large number of coefficients to achieve a good level of approximation for a function with complex, small scale, or singular behaviour. An important example of such an approximation in one dimension, is the piecewise linear interpolant $y = \Pi_1 u$ defined over a *uniform mesh* with N points. If $u \in H^2(\Omega)$, then as $N \to \infty$ the following estimate is given in [19]

$$N^2 \|\Pi_1 u - u\|_2 \leq C_1 \|u\|_{H^2(\Omega)}. \tag{1.2}$$

Here $H^2(\Omega)$ is the usual Sobolev space of functions with square integrable second derivative [1], and the constant C_1 depends only on Ω. If u is not in $H^2(\Omega)$ this estimate fails and we see a sub optimal rate of convergence. Even if $u \in H^2(\Omega)$ when approximating a rapidly varying function good convergence is only seen when the local knot spacing is smaller than the smallest length scale of the target function. It is thus *essential* to change the knot locations to achieve the best approximation.

1.1.2. Free Knot Splines (FKS)

Often very significantly greater accuracy, with far fewer numbers of coefficients, is achieved by using an (adaptive) nonlinear approximation method [5] such as a Free Knot Spline (FKS) in one-dimension, or adaptive mesh methods [19], [27] in higher dimensions. In such approximations both the weights $\mathbf{w}$ *and* the knot locations $\mathbf{k}$/mesh are optimised. Even though such methods are nonlinear in the coefficients, effective a-priori and a-posteriori error estimates and interpolation error estimates have been established linked to the regularity of the meshes over which the approximations are defined [19]. A key advantage

of using such an adaptive approximation over a standard spline approximation, is that for a fixed number N of knots a much higher accuracy can be achieved by exploiting the extra degrees of freedom given by moving the knots $\mathbf{k}$ [7], [19], [27], which is of importance when approximating functions with rapid variation and/or singularities. A special case is the linear interpolating FKS (IFKS). In this approximations the weights w_i are evaluations of the target function $u(k_i)$ at the knot locations. As such the optimisation reduces to finding the knot locations only. Tight convergence results in various norms have been developed in a general dimension d for the best interpolating piecewise linear approximation $y = \Pi_1 u$ to a general target function u as $N \to \infty$. Summaries are given in [7], [19], and later in this paper. In particular, in one-dimension, the optimal approximation error between a general target function u (even one of limited smoothness) and a linear interpolating FKS $\Pi_1 u$ with an *optimal choice of N knots* satisfies (for a suitable constant C_2) the bound [19]

$$N^2 \|\Pi_1 u - u\|_2 \leq C_2 \left(\int_\Omega \left(u''\right)^{2/5} \, dx \right)^{5/2}. \tag{1.3}$$

The estimate (1.3) is in general often very much better than that in (1.2) and applies for functions which are not in $H^2(\Omega)$. Whilst the IFKS is, in general, far from the best approximation, the optimal knots for the best IFKS are usually close to those for the best FKS. Training a general FKS approximation then becomes a *two stage process*. The first being the (nonlinear) problem of finding the optimal knot locations k_i, and the second the well conditioned problem of finding the correct weights w_i. We show in this paper that it is this *combination* which gives excellent approximations with high accuracy.

1.1.3. Comparing ReLU NN with classical adaptive approximations

The complementary advantages of FKS and ReLU NNs, and also their formal equivalence, motivates us to exploit the relationship between shallow ReLU NNs and B-spline approximations. In comparison to (1.1) a (suitably scaled) shallow ReLU NN with *breakpoints* k_i and *coefficients* c_i can take the form

$$y(x) = \sum_{i=0}^{N-1} c_i \, \mathrm{ReLU}(x - k_i). \tag{1.4}$$

We study both the direct correspondence between the knot points of the FKS and the breakpoints k_i (where the derivative changes discontinuously) of the shallow ReLU NN, and also the linear correspondence between the weights w_i of the FKS and the coefficients c_i of the shallow ReLU NN. Using the spline lens, such shallow univariate ReLU NNs have been investigated in terms of their initialization, loss surface, Hessian, conditioning and gradient flow dynamics, [14], [16], [24], [25]. A ReLU NN representation for continuous piecewise linear functions has been studied in [15], with a focus on continuous piecewise linear functions from the finite element method with at least two spatial dimensions. A theoretical study on the expressive power of NNs with a ReLU activation function in comparison to linear B-spline methods is provided in [11], [14], [16], [24]. The connection between NNs and splines have also been considered in other contexts, for instance when learning activation functions in deep spline NNs [2]. The ill-conditioning and spectral bias of finding the scaling coefficients of shallow ReLU NN networks, and a comparison with 'hat-function' linear B-spline approximations has been investigated analytically in [16], [24], [26], [29], with empirical studies also given for deeper networks. We consider this comparison further in this paper.

1.1.4. Motivating example

To motivate this comparison we consider the approximation of the target function

$$u(x) = \tanh(100(x - 1/4)), \quad x \in [0,1].$$

This function has a narrow boundary layer around $x = 1/4$ which must be resolved to give a good approximation. This requires an adaptive approach. We consider various methods of approximating this function.

1. Regular spline and ReLU NN approximations (1.1,1.4) where knots/breakpoints k_i are *fixed* to be uniformly spread in the interval, and *only* the weights/coefficients w_i, c_i are trained using the standard L_2 loss function.
2. A (standard) FKS and a shallow ReLU NN with *all* of k_i, c_i, w_i trained using the standard L_2 loss function.
3. A two-level approach with the FKS (1.1) first trained to find the optimal knots k_i using an equidistribution based loss and then using optimising a regularised L_2 loss to train the weights w_i. Similarly training a ReLU NN (1.4) to find the optimal breakpoints k_i and then optimising the L_2 loss *without preconditioning* to find the optimal coefficients c_i. Note that in this case a preconditioned ReLU NN is completely equivalent to the FKS.

In Figure 1 we plot (left) the resulting L_2^2 loss of the trained networks as a function of the number N of knots/width of the network. We can see from this figure that the ReLU NN and regular spline with *fixed* knots k_i, but with optimised weights/coefficients both perform poorly, with large errors. The (standard) ReLU NN trained on the L_2 loss function also performs poorly. The (standard) FKS trained on the L_2 loss function (equivalent to a preconditioned ReLU NN) performs rather better than the standard ReLU NN, but still behaves sub-optimally with poor convergence for large N. The FKS trained using the two-level approach performs optimally on this test function with rapid convergence. The ReLU NN traoned using the two-level method gives a much worse result if no preconditioning is used to find the coefficients c_i. The difference in approximation error between the ReLU NN approximation and the correctly trained FKS is striking with a difference in error of many orders of magnitude. Similarly the difference between the regular spline error (with uniform knots) and the FKS error (with optimised knots) is also striking. However, a ReLU NN trained to find the optimal breakpoints k_i and then preconditioned to find the optimal coefficients is identical to the optimal FKS and gives similarly good results. We conclude from this example that to get the best approximation (by some orders of magnitude) we must find *both* the optimal knots *and* the optimal weights/coefficients using a well-conditioned optimisation method.

To see the role that ill-conditioning (and related spectral bias) plays in the problem of finding the optimal coefficients of the ReLU NN network we consider using both a FKS with $N = 64$ knots and the formally equivalent shallow ReLU NN of width $W = 64$ which has N breakpoints. In Figure 1 (right) we compare the training, using the L_2^2 error, of both in the case where the knots/breakpoints are fixed to the optimal values (for the IFKS). It is clear from Figure 1 that with this set of knots the FKS trains very much more quickly, and to a much more accurate approximation than the ReLU NN for this target function. This, is due to the poor conditioning of the ReLU NN approximation problem for large N associated with the condition number of the normal equations of the training process. This was noted in [16], [26], [24], [29] and is due to the 'global' nature of the ReLU functions. Methods such as norm weighting do not improve this optimisation problem and a much more careful preconditioning is needed. However,

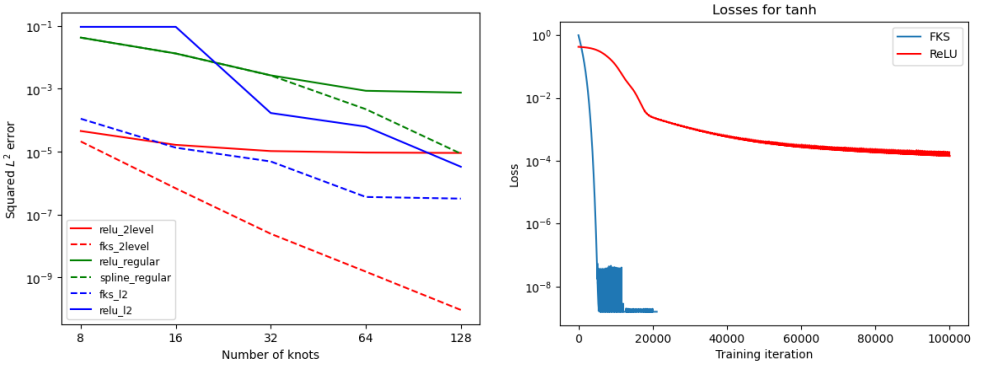


FIG. 1. (Left) A comparison of the convergence of the approximation of the target function $u(x) = \tanh(100(x-1/4))$ with various architectures and varying N This shows the poor performance of the ReLU NN trained using the regular L_2^2 loss function with regularly spaced knots k_i, the better, but still poor, performance of the regular spline with the same loss function (green, green dashed). We see the still poor performance of the ReLU NN and the better, but still sub-optimal performance of the (standard) FKS trained using the usual L_2^2 loss function (blue,blue dashed). We also see the far better performance of the FKS trained with a two level approach (red dashed) but the much poorer performance of the ReLU NN using the two-level approach without preconditioning (red). A preconditioned ReLU NN architecture trained in the two-level approach shows the same optimal performance to the FKS. (Right) A comparison of the training times and convergence of the weights/coefficients of the FKS and a shallow ReLU NN for the target function $u(x) = \tanh(100(x-1/4))$ with $N = 64$, following the calculation of the optimal knots of the interpolating FKS. The FKS trains very rapidly, whereas the ReLU NN trains slowly due to ill-conditioning.

this is only one factor of the approximation problem. Only by finding the optimal breakpoints k_i of the ReLU network first and then preconditioning the problem of finding the optimal coefficients c_i, can the same accuracy as the optimal FKS be realised.

1.2. *Contributions*

We analyse the problem of function approximation from the perspective of training both a ReLU NN and a FKS, and hence improve the conditioning, training and accuracy of the ReLU NN architecture

Our initial theoretical analysis considers the most simple case of univariate approximation using a shallow ReLU NN and an FKS. Whilst this is removed from the usual practice of using deep networks on higher dimensional problems, the simplicity of this problem allows us to conduct a careful mathematical analysis, which gives much insight into the numerical results that we obtain for more general (deep and higher dimensional) problems.

We show rigorously that in the univariate, shallow case the ReLU NN approximation problem is poorly conditioned as the network width N increases. As a result the training process is very slow, leading either to a poor approximation of the target function, or simply to no convergence. We then propose a novel two-level approach to the nonlinear problem of training the univariate FKS approximation with weights $\mathbf{w}$ and knots $\mathbf{k}$). This problem is tackled by constructing an effective loss function based on the equidistribution principle for the knot locations of the interpolating FKS, and minimising this using the Adam optimiser. We then solve the provably well conditioned problem of finding the weights w_i and (close-by) knots of the FKS. We then show that this approach can be used to train the shallow ReLU NN, firstly by finding breakpoints of the ReLU NN (as the knots of the interpolating FKS), and then the

coefficients c_i of the ReLU functions. We show that this latter problem can be made well conditioned for large N by a linear preconditioning of the (weights of the) ReLU NN, transforming it into an FKS, and then training the resulting transformed system. Using a variety of (regular, singular and rapidly varying) target functions, we demonstrate numerically that by using this *two-stage training procedure*, the shallow univariate ReLU NN can be trained reliably to give a highly accurate approximation, achieving the theoretical expressivity of these networks. Other related papers such as [16], [24], [26],[29] have considered the question of the ill-conditioning (and then preconditioning) the problem of finding the coefficients c_i of the shallow ReLU NN and our proof of this is similar to theirs. However, we believe that the *combination* of preconditioning the system to find the weights with the separate (equidistribution based method) of finding the knots/breakpoints which we present is novel. It is also *essential* if we are to realise the full expressivity of such networks when used to approximate very challenging target functions.

We then extend our results to univariate approximations using more general activation functions, and also give a short study of the accuracy, and the preconditioning, of deeper networks. The complexity of these systems makes an analytical investigation difficult. However we show, by numerical experiments, that the training of the ReLU NN is still ill-conditioned and (if it converges at all) gives a sub-optimal solution. In contrast methods based on spline approximation can be derived which have high accuracy and we give experimental evidence of the effectiveness of a preconditioner in the deep case.

1.3. *Outline*

The remainder of this paper is structured as follows. In Section 2 we describe univariate ReLU NNs, as well as free knot splines (FKS) as nonlinear function approximators. In Section 3 we show the formal equivalence of shallow ReLU NN and the FKS representations and the links between the breakpoints/knots and the parameters/weights of these respective architectures. Section 4 defines the loss functions often used based on the mean-squared error and discusses the difficulty of optimising the approximation in using these functions to train the knots/breakpoints and weights together. In Section 5 we focus on the problem of finding the optimal knots for an interpolating FKS and introduce a loss function based on equidistribution as a way of guiding the training. In Section 6 we consider the problem of finding the optimum weights having found the knots (as in Section 5). We show that this problem is well-conditioned for the FKS representation of the solution, but is ill-conditioned for the shallow ReLU NN architecture. In Section 7 we then introduce the two-level training procedure for firstly the knots and then the weights of an FKS. Then we show how, after preconditioning, it leads to an effective optimisation approach for the shallow ReLU NN. Numerical results for the one dimensional approximation using both a FKS and a shallow network for a range of challenging target functions are shown in Section 8 where we investigate the impact of the choice of loss function, the quality of the function approximation, the evolution of the knot locations as well as the conditioning and the convergence of the loss training. Brief extensions of this work to different activation functions, and deep networks, are discussed in Section 9. Finally, in Section 10 we will draw our conclusions based on the results, and suggest areas of further investigation.

2. ReLU NN and free knot splines (FKS) as nonlinear function approximators

In this section, we define univariate ReLU Neural Networks (NN) and free knot splines (FKS).

2.1. ReLU *Neural Networks (*ReLU *NN)*

We start by defining a standard feed forward neural network with a one-dimensional input $x_0 \in \mathbb{R}$ and one-dimensional output $y(x_0, \theta)$ for some vector of parameters θ, described further below. We assume that this network has a width W and a depth L and uses a rectified linear unit (ReLU) $\sigma(x) = (x)_+$ as activation function which acts element wise on the input vector. Defining the network input by $x_0 \in \mathbb{R}$, the first hidden layer is defined by $x_1 = \sigma(A_0 x_0 + b_0) \in \mathbb{R}^W$ for $A_0, b_0 \in \mathbb{R}^W$ and the NN with L hidden layers is then iteratively defined by

$$x_{k+1} = \sigma(A_k x_k + b_k) \in \mathbb{R}^W \tag{2.1}$$

for $k \in \{1, \ldots, L-1\}$, with $A_k \in \mathbb{R}^{W \times W}$ and $b_k \in \mathbb{R}^W$. The NN output is given by

$$y(x, \theta) = A_L x_L + b_L \in \mathbb{R}, \tag{2.2}$$

with $A_L \in \mathbb{R}^{1 \times W}$ and $b_L \in \mathbb{R}$. In the simplest case we can consider a *shallow* ReLU network with width W and depth $L = 1$ as in (1.4), otherwise the network is *deep*. The case $L = 1$ is closest to standard approximation, but the power and expressivity of the NN often arises when $L > 1$, although such networks are harder to train and to analyse. In this representation we define the set of parameters $\theta = \{A_k, b_k \colon \ k = 0, \ldots, L\}$. For a given range of parameters θ we define $\Gamma_{W,L}$ to be the set of all functions $y(x)$ which can be represented by the NN. Observe that $\Gamma_{W,L}$ is *not* a linear vector space and that A_k and b_k are not sparse in general. We apply this NN to approximate a function $u(x) \colon [0,1] \to \mathbb{R}$ by finding an appropriate choice of θ. We assume that there exists a batch of training data $\{y_{x_i}\}_{i=0}^{s-1} \subset \mathbb{R}^s$ at training points $\{x_i\}_{i=0}^{s-1} \subset [0,1]$ of size s. Typically this uses the standard means of training a NN to do this, including the use of a squared-error L_2 loss function given by $L = \|y - u\|_2^2$ and an optimiser such as Adam [21]. However, as we will see, crucial to the effectiveness of the training procedure is the use of a good initialisation to start the approximation, an effective (convex) loss function, and a careful preconditioning of the problem. We shall demonstrate this using the examples in this paper, and will also show how an effective two-level training method can be constructed, using a suitably convex loss function combined with preconditioning.

2.2. *Linear spline approximations*

Let $N \in \mathbb{N}$ be given and let $0 = k_0 < k_1 < \ldots k_i < \ldots < k_{N-1} = 1$ be a set of N *knots* in the interval $[0,1]$. For $i = 0, \ldots, N-1$, we define a locally supported *linear spline* (a hat-function) $\phi_i(x)$ to be the unique piecewise linear function so that $\phi_i(k_i) = 1$, and $\phi_i(k_j) = 0$ if $i \neq j$. Each spline ϕ_i is weighted by some weight w_i, and we denote by $\psi_{\text{FKS}} = \{(w_i, k_j) \colon \ i = 0, \ldots, N-1, \ j = 1, \ldots, N-2\}$ the set of parameter values. Note that we exclude k_0 and k_{N-1} in the definition of ψ_{FKS} as they are set as 0 and 1, respectively.

Definition 1 (Free knot splines (FKS)) *A* free knot spline (FKS) *is a piecewise linear approximation to the function $u(x)$ defined over the set of linear splines and (as in (1.1) is given by*

$$y(x, \psi_{\text{FKS}}) = \sum_{i=0}^{N-1} w_i \, \phi_i(x), \tag{2.3}$$

where we assume that we are free to choose both the knots k_i, $i = 1, \ldots, N-2$, in the linear splines ϕ_i and the weights $w_i, i = 0 \ldots N-1$.

We denote by $\Sigma_{N,1}$ the set of all piecewise linear functions of the form (2.3). Observe that the FKS has $2N-2$ degrees of freedom ($N-2$ for the knots and N for the weights). Like the NN space $\Gamma_{W,L}$, the approximation space $\Sigma_{N,1}$ is nonlinear. Similarly to the NN we can train a FKS to approximate a function. This is a nonlinear, non-convex problem to find the full set of weights $\mathbf{w}_i$ and knots $\mathbf{k}_i$.

In the more usual case of piecewise linear approximation, the values of the knots k_i are *fixed* and we refer to such approximations as either *fixed knot piecewise linear approximations* or *regular/standard approximations*. The best function approximation is given by the well-conditioned problem of finding the optimal choice of *weights* w_i for $i = 0,\dots,N-1$ and we denote by $\psi_{\mathrm{PWL}} = \{w_i \colon i = 0,\dots,N-1\}$ the associated parameter values.

Definition 2 (Fixed knot piecewise linear interpolant (PWL)) *Such a* fixed knot piecewise linear approximation *is a piecewise linear approximation to the function $u(x)$ defined over the set of linear splines and is given by*

$$y(x,\psi_{\mathrm{PWL}}) = \sum_{i=0}^{N-1} w_i\,\phi_i(x) \tag{2.4}$$

where we assume that we are free to choose the coefficients w_i, but the knots k_i, $i = 1,\dots,N-2$, in the linear splines ϕ_i are assumed to be fixed.

Note that (2.4) is a linear function of the coefficients w_i and the set of all such possible functions is a linear vector space. The values of w_i can be found simply by minimising the least squares loss. Equivalently, a linear matrix problem of the form $Gw = q$ for the coefficients $w = (w_i)_{i=0}^{N-1} \in \mathbb{R}^N$ can be solved, where the (well conditioned) sparse Gram matrix $G \in \mathbb{R}^{n\times n}$ and vector $q \in \mathbb{R}^n$ are given. Whilst this is a simple process and widely used, for example in the Finite Element Method, such approximations lack expressivity, especially if the target function $u(x)$ is singular, or has natural length scales which are smaller than the smallest separation between the knots. The additional degrees of freedom given by choosing the knots k_i in the FKS leads to a very significant increase in the accuracy of the approximation.

A distinguished subset of the set of free knot splines are the *interpolating free knot splines* where the set of knots k_i for $i = 1,\dots,N-2$ can be freely chosen, but the weights are given by $w_i = u(k_i)$ We denote the associated parameter values by $\psi_{\mathrm{IFKS}} = \{k_i \colon i = 1,\dots,N-2\}$.

Definition 3 (Interpolating free knot splines (FKS)) *The* interpolating free knot splines (FKS) $y(x,\psi_{\mathrm{IFKS}}) = \Pi_1 u(x)$ *for the target function $u(x)$ is given by*

$$y(x,\psi_{\mathrm{IFKS}}) \equiv \Pi_1 u(x) = \sum_{i=0}^{N-1} u(k_i)\phi_i(x), \tag{2.5}$$

where we assume that we are free to choose the knots k_i, $i = 1,\dots,N-2$, in the linear splines ϕ_i.

The interpolating FKS is easier to construct than the best FKS approximation for $u(x)$ as the coefficients w_i of the FKS are given directly by $w_i = u(k_i)$ and only the knots need to be determined. It is thus not as expressive as the best FKS, but has the same rate of convergence as $N \to \infty$ for classes of singular functions [19], and it is a much better approximation in general (especially for smaller

values of N) than a fixed knot approximation. Observe that finding the interpolating FKS still requires solving a nonlinear problem for the knots. We will make extensive use of this function as a way of guiding the approximation procedure and also assessing the accuracy of a more general FKS/ReLU NN approximation to u. In particular $\Pi_1 y$ is a very good function to take as the initial start of the optimisation procedure for finding either the best ReLU NN approximation or the best FKS approximation to u. This is because the *optimal* set K of knot points is constructed for $\Pi_1 u$ appears from extensive calculations to be very close to the optimal set of knot points for the FKS, and equivalently the breakpoints of the ReLU NN representations.

3. Relations between the ReLU NN and the FKS representations

Any ReLU NN gives a function $y(x)$ which is piecewise linear, and is smooth (linear) everywhere apart from at a set of breakpoints where it changes gradient and has undefined curvature. These points are equivalent to the knot points k_i of the FKS if they lie in the range of values of $x \in [0,1]$ over which the target function $u(x)$ is defined. In this sense the ReLU NN and the FKS are formally equivalent. For a deep NN the map between θ and $\{k_i : i = 0,\ldots,N-1\}$ is complex and is related to the solution of a system of algebraic equations [6],[14]. More precisely, the authors show that a NN of width W and depth L is formally equivalent to a FKS with N knots for $N < (W+1)^L$. Hence they can have the same expressivity, even though this is rarely seen in practice. Due to the linear dependencies between the linear pieces, the NN space $\Gamma_{W,L}$ is usually much smaller than the FKS space $\Sigma_{N,1}$ with $N \ll (W+1)^L$.
.

As shown in [6],[14] both a ReLU NN and an FKS are piecewise linear functions of x and we can use the methodology of training one to inform the training of the other. For a shallow network with $L = 1$ and $W = N$, we have breakpoints in the ReLU NN representation, which are the same as the knots in the FKS at

$$k_i = -(b_0)_i/(A_0)_i, \quad i = 0,\ldots,N-1 \quad \text{if} \quad k_i \in [0,1], \tag{3.1}$$

where $(b_0)_i$ and $(A_0)_i$ denote the ith components of vectors b_0 and A_0. For any ordered set of knots k_i, with $0 = k_0 < k_1 < \ldots < k_{N-2} < k_{N-1} = 1$ we can represent the piece-wise linear basis functions $\phi_i(x)$ of the FKS using the formula presented in [6] so that

$$\phi_i(x) = \alpha_i \mathrm{ReLU}(x - k_{i-1}) - \beta_i \mathrm{ReLU}(x - k_i) + \gamma_i \mathrm{ReLU}(x - k_{i+1}) \tag{3.2}$$

for $i = 1,\ldots,N-2$, where

$$\alpha_i = \frac{1}{k_i - k_{i-1}}, \quad \beta_i = \frac{k_{i+1} - k_{i-1}}{(k_{i+1} - k_i)(k_i - k_{i-1})}, \quad \gamma_i = \frac{1}{k_{i+1} - k_i}. \tag{3.3}$$

Similarly, $\phi_0(x) = \mathrm{ReLU}(k_1 - x)/k_1, \quad \phi_{N-1}(x) = \mathrm{ReLU}(x - k_{N-2})/(1 - k_{N-2})$. This implies that FKS representation in terms of the basis functions can also be given in terms of ReLU functions as follows

$$y(x, \psi_{\mathrm{FKS}}) = \sum_{i=0}^{N-1} w_i \phi_i(x) \equiv w_0 \mathrm{ReLU}(k_1 - x)/k_1 + \sum_{i=0}^{N-2} c_i \mathrm{ReLU}(x - k_i) \tag{3.4}$$

where the ReLU NN coefficients c_i are rational functions of the knots k_i and weights w_i of the FKS. Taking careful note of the representation of the basis functions at the two ends of the interval we obtain:

$$c_0 = \alpha_1 w_1, \quad c_1 = \gamma_1 w_2 - \beta_1 w_1, \quad c_i = \gamma_i w_{i+1} - \beta_i w_i + \alpha_i w_{i-1}, \quad i = 2,\ldots,N-2. \tag{3.5}$$

Hence the two representations of the functions by the ReLU NN and by FKS are formally equivalent, with the equivalence of the ReLUbreakpoints with knot points, and with the *tri-diagonal linear* map (3.5) between the weights $\mathbf{w}$ and coefficients $\mathbf{c}$. Hence both networks are equally *expressive*. However, the information from the respective parameter sets θ and ψ_{FKS} is encoded differently, and is critical for their training and in particular for the well-posedness of the solution representation and the associated training algorithm.

4. Training the ReLU NN and the FKS

Having seen the expressivity of the ReLU NN and of the FKS and the formal equivalence of the (shallow) ReLU NN and the FKS approximation in Section 2, in this and the following two sections we consider how to train these to find the best approximation of the given function u. In all cases the training will be done through a (first order) gradient-based approach using the Adam optimiser with a suitable initial start and with an appropriate loss function. This procedure is chosen as it is a very common way of training a machine learning network. We find in general that a direct training of all of the parameters is problematic, and that a more effective approach is to divide the training procedure into first training the knots/breakpoints and then the weights of the network. Different loss functions are used in these two cases, and preconditioning is needed for the ReLU NN network. To unify notation, we denote the ReLU NN or the FKS approximation by $y(x)\colon [0,1] \to \mathbb{R}$ and the associated parameter values by θ. Further, we denote the target function by $u\colon [0,1] \to \mathbb{R}$. We initially consider the loss function $\mathscr{L}_2^2$ based directly on the *mean-squared approximation error* of the approximation. This is the default loss function which is typically used in training a NN approximation.

As both $y(x)$ and $u(x)$ are given for all $x \in [0,1]$ we have that the $\mathscr{L}_2^2$-error of the approximation is given by

$$\mathscr{L}_2^2(\theta) = \int_0^1 (y(x,\theta) - u(x))^2 \, dx. \tag{4.1}$$

In practice, we cannot evaluate (4.1) for a general function and we have to consider various approximations to it. For this, we consider a set of *quadrature points* $\{x_k\}_{k=0}^{s-1} \subset [0,1]$ for some large parameter $s \in \mathbb{N}\backslash\{0\}$ which can either be randomly generated with uniform distribution or regularly distributed over the domain $[0,1]$. Assume that points $\{x_k\}_{k=0}^{s-1} \subset [0,1]$ are ordered so that $x_i < x_{i+1}$. An approximation to $\mathscr{L}_2^2$ in (4.1) is given by

$$L_2^2(\theta) = \sum_{i=0}^{s-1} (x_{i+1} - x_i) \Big(y(x_i,\theta) - u(x_i) \Big)^2. \tag{4.2}$$

Note that minimising (4.2) is a highly non-convex problem, and the minimisation process generally leads to sub-optimal solutions. A similar loss function is often used in the *Deep Ritz Method (DRM)* [10]. Similarly to the DRM, we will consider both the case where $\{x_k\}_{k=0}^{s-1}$ are sampled at each iteration or fixed during the training. Observe that the minimisation of L_2^2 in (4.2) reduces to the pointwise error at sample points $\{x_i\}_{i=0}^{s-1}$ and neglects global properties of the function, such as its curvature. This is important when considering equidistributed knots in regions of highest curvature, which contribute significantly to the approximation error.

Whilst simple and widely used, for example in general ReLU NN training or in linear spline approximation, the use of the loss function (4.2) *is problematic* when used to train either a ReLU NN

or an FKS network. We can see this clearly in the results presented in Figure 1 where we see poor accuracy of both the FKS and ReLU NN networks trained using this loss function. Similar behaviour can be seen in many other examples and we will give a series of numerical calculations giving evidence for this is Section 8. There are a number of reasons for this poor behaviour. Firstly all of the terms in the parameter set θ are treated equally, whereas the knots/breakpoints k_i play a very different role in the approximation from the weights w_i, c_i. This leads to a highly non-convex minimisation problem. Secondly, in the the case of the ReLU NN approximation it leads to an ill-conditioned problem for the weights (see Section 6 and [16],[29]).

Accordingly we now consider an alternative procedure for finding the best approximation which takes into account the distinguished role played by the k_i. This firstly solves the *nonlinear* problem of finding a close approximation to the knots and then the (nearly) *linear* problem of finding the weights and the optimal knots of the network. To do this we make use of the interpolating FKS

5. Finding the optimal knots of the interpolating FKS using equidistribution based loss functions

Whilst the interpolating FKS (IFKS) $\Pi_1 u$ is a sub-optimal approximation, it is an excellent first guess for general optimisation procedures. Finding the IFKS only involves determining the knots k_i which leads to a simplified approximation process approach for choosing the optimal knots which can be adapted for training the breakpoints of the ReLU NN. Having found the knots, the problem of finding the weights w_i of the optimal FKS is a well conditioned nearly linear problem and we consider this in the next section together with the ill=conditioned problem of finding the coefficients c_i of the ReLU NN. To find the optimal knots for the IFKS we consider optimising a different loss function from the standard L_2 loss. This loss function is based on the concept of equidistributing the IFKS solution error across the knot locations.

5.1. *The equidistribution loss function*

Provided that the target function $u(x)$ is twice differentiable in (k_i, k_{i+1}), the local interpolation error of the linear interpolant y of u on $[k_i, k_{i+1}]$ satisfies

$$y(x,\theta) - u(x) \approx \frac{1}{2}(x-k_i)(k_{i+1}-x)u''(x_{i+1/2}) \tag{5.1}$$

for any $x \in [k_i, k_{i+1}]$ and some $x_{i+1/2} \in (k_i, k_{i+1})$. Note that this result even holds for the functions such as $u(x) = x^{2/3}$ for which $u''(0)$ is singular. This yields

$$\int_{k_i}^{k_{i+1}} (y(x,\theta) - u(x))^2 \, dx \approx \frac{1}{120}(k_{i+1}-k_i)^5 |u''(x_{i+1/2})|^2,$$

implying

$$\mathscr{L}_2^2(\theta) = \int_0^1 (y(x,\theta) - u(x))^2 \, dx \approx \frac{1}{120} \sum_{i=0}^{N-2} (k_{i+1}-k_i)^5 |u''(x_{i+1/2})|^2. \tag{5.2}$$

Motivated by (5.2), we define

$$L_I(\theta) = \frac{1}{120} \sum_{i=0}^{N-2} (k_{i+1}-k_i)^5 \, |u''(k_{i+1/2})|^2 \tag{5.3}$$

where $k_{i+1/2} = k_i + k_{i+1}$ and we obtain $\mathscr{L}_2^2(\theta) \approx L_I(\theta)$. Note that L_I only depends on the free knot locations for the interpolating FKS, and thus can be used to train the knots. However, unlike other loss functions L_I requires knowledge of u'' rather than point values of u and we will assume for this paper that u'' is known. In many contexts, such as the use of (moving mesh) approximations to solve differential equations, [19] we have direct access to u''. In other contexts an appropriate estimate of u'' can be obtained by gradient recovery, with an iteration between fiding the solution and updating the knots [19].

A powerful result in the form of the equidistribution principle, first introduced by de Boor [3] for solving boundary value problems for ordinary differential equations, gives a way of finding θ so that L_I in (5.3) is minimised. More precisely, minimising L_I in (5.3) can be expressed by equidistributing the error over each cell which results in the following lemma, given in [19, Chapter 2]:

Lemma 5.1 [Equidistribution] *The loss L_I in (5.3) is minimised over all knots k_i if given*

$$\rho_{i+1/2} = (k_{i+1} - k_i)\,|u''(k_{i+1/2})|^{2/5} \quad \textit{for all } i = 0,\dots,N-2 \tag{5.4}$$

there is a constant $\sigma > 0$ so that

$$\rho_{i+1/2} = \sigma \quad \textit{for all } i = 0,\dots,N-2. \tag{5.5}$$

The equidistribution principle in Lemma 5.1 replaces a non-convex optimisation problem for finding all of the terms of θ by one of just finding the knots $\mathbf{k}$ with better convexity properties. A set of knot points satisfying these conditions is said to be *equidistributed*. The degree of equidistribution can then be used as a quality measure of the solution. The algebraic equations (5.5) can in principle be solved directly [4], or iteratively using moving mesh techniques such as [18] for example. We propose instead training the knots of the IFKS directly using an optimisation approach enforcing the equidistribution condition (5.5) directly through minimising an equidistribution-based loss function. We do this using Adam with an initially uniform distribution of the knots as a start. We demonstrate that this procedure is both effective, and can be generalised to the training of the ReLU NN. To do this, we compute $\rho_{i+1/2}$ from knots k_i by (5.4) and set σ as their mean, i.e. $\sigma = \frac{1}{N-1}\sum_{i=0}^{N-2}\rho_{i+1/2}$. We define the equidistribution loss function L_E by

$$L_E(\theta) = \sum_{i=0}^{N-2} (\rho_{i+1/2} - \sigma)^2. \tag{5.6}$$

Remark 5.2 In practice, to ensure regularity when $|u''|$ is small we replace the definition of $\rho_{i+1/2}$ in (5.4) by the regularised version

$$\rho_{i+1/2} = (k_{i+1} - k_i)\,(\varepsilon^2 + u''(k_{i+1/2}))^{1/5} \quad \text{for all } i = 0,\dots,N-2.$$

A value of $\varepsilon^2 = 0.1$ works well in practice, and we will use it for further calculations. However, abalation studies show that the results are largely insensitive to the value of ε provided that it is not too small.

5.2. *Convergence of the IFKS*

To emphasise the importance of finding the *optimal* knots of the IFKS (and hence the FKS), we show the excellent accuracy $\mathscr{O}(1/N^4)$ of this approximation for a wide class of target functions, including

singular functions. The optimal knots $k_i,\ i=0,\dots,N-1$, of L_I for $\Pi_1 u$ are given analytically by using quadrature applied to (5.4) in the one-dimensional setting, and we will consider this approach in Section 8.3. Similar to Moving Mesh PDEs [19], we consider the *physical interval* $x\in[0,1]$ to be a map from a *computational interval* $\xi\in[0,1]$ so that $x=x(\xi)$ with $x(0)=0$ and $x(1)=1$. For $i=0,\dots,N-1$, the ith knot point k_i is given by $k_i=x(i/(N-1))$. Provided that N is large we can then use the approximations

$$k_{i+1}-k_i=\frac{1}{N-1}\frac{dx}{d\xi}(\xi_{i+1/2})\quad\text{and}\quad x_{i+1/2}=x(\xi_{i+1/2})$$

for some $\xi_{i+1/2}\in(i/(N-1),(i+1)/(N-1))$, where $x_{i+1/2}\in(k_i,k_{i+1})$ is defined by (5.1). The equidistribution condition for L_I requires that $(k_{i+1}-k_i)^5|u''(x_{i+1/2})|^2$ is constant for $i=0,\dots,N-1$ which yields

$$\left(\frac{dx}{d\xi}(\xi_{i+1/2})\right)^5\left(u''(x(\xi_{i+1/2}))\right)^2=D^5 \tag{5.7}$$

for a suitable constant D. This results in a differential equation for x given by

$$\frac{dx}{d\xi}(\xi)=D\,\left(u''(x(\xi))\right)^{-2/5},\quad x(0)=0,\quad x(1)=1, \tag{5.8}$$

where value of D is fixed by the boundary condition so that

$$D=\int_0^1(u''(x(\xi)))^{2/5}\,d\xi. \tag{5.9}$$

The optimal knots $k_i=x(i/(N-1))$ of a piecewise linear IFKS on an equidistributed mesh can then be determined from the solution of (5.8) using (5.9). We substitute (5.7) into (5.3) to obtain

$$L_I(\theta)\approx\sum_{i=0}^{N-2}\left(\frac{dx}{d\xi}(\xi_{i+1/2})\right)^5\frac{(u''(x(\xi_{i+1/2})))^2}{120(N-1)^5}=\sum_{i=0}^{N-2}\frac{D^5}{120(N-1)^5}=\frac{D^5}{120(N-1)^4}$$

up to leading order. We deduce that provided $D=\mathcal{O}(1)$, the discretisation L_I of $\mathscr{L}_2^2$ for an interpolating FKS is then $\mathcal{O}(1/N^4)$ as required.

As an example, in the supplementary material we consider the singular target function $u(x)=x^\alpha$ for some $\alpha\in(0,1)$.

6. Training the coefficients of the shallow ReLU NN and FKS representations

6.1. *Overview*

Having found the optimal knots of the interpolating FKS we now optimise the ReLU NN/FKS representations given by (3.4). We find empirically that the best knots for the FKS are very close to those of the IFKS. For a given set of *fixed* knots this problem then reduces in the case of spline approximation to that of the well-conditioned least minimisation problem of finding the optimal weights w_i. In contrast determining of the best coefficients c_i for the ReLU NN *directly* is (as we will now prove and is shown in [16],[29]) an ill-posed problem with very slow convergence. However, by a simple preconditioning step we can transform the ReLU NN network into one which can be readily optimised to accurately approximate complex functions. Accordingly, we now consider the problem of using the L_2 loss function to determine w_i and c_i for *known* k_i which we assume have been pre-calculated by optimising the equidistribution loss .

6.2. *Conditioning of the training*

Using the results of Section 3 we compare the conditioning of the problem of calculating the coefficients $c_i, i = 0 \ldots N-2$ and the weights $w_i, i = 1 \ldots N-1$ of the shallow ReLU NN and the FKS representations respectively (to give the best L_2 approximation of the function $u(x)$) on the assumption that the knot locations k_i are known. We show that, as expected, the FKS problem (with the localised support basis functions) is well conditioned for all N whereas the ReLU NN problem (for which basis functions have global support) is increasingly ill-conditioned for larger values of N. In particular we establish the following

Lemma 6.1 *For fixed knots k_i, $i = 1..N$, the normal equations for finding the weights w_i of the FKS have condition number $\kappa = \mathcal{O}(1)$. In contrast the normal equations for finding the coefficients c_i of the* ReLU *NN have condition number $\kappa = \mathcal{O}(N^4)$*

NOTE A similar result, obtained using different methods, is given in [16], [29], without reference to the calculation of the optimal k_i. Our method simplifies, extends this earlier calculation.

To prove Lemma 6.1, we observe from (3.5) that

$$c = Aw \tag{6.1}$$

for a linear operator A and we compare the conditioning of the two optimisation problems by studying the properties of the matrix A. The matrix A has a specific structure which we can exploit. It follows from (3.5) that

$$c_i = \frac{w_{i+1} - w_i}{k_{i+1} - k_i} - \frac{w_i - w_{i-1}}{k_i - k_{i-1}} \tag{6.2}$$

for $i = 2, \ldots, N-2$. Hence, $c = (c_i)_{i=1}^{N-2}, \mathbf{w}^* = (w_i)_{i=1}^{N-2} \in \mathbb{R}^{N-2}$ are related via

$$\mathbf{c} = T\mathbf{w}^* + \gamma_{N-2} w_{N-1} \mathbf{e}_{N-2}, \tag{6.3}$$

where $\mathbf{e}_{N-2} = (0,0,0,\ldots 1)^T \in R^{N-2}$ and $T \in R^{(N-2)\times(N-2)}$ is the symmetric tri-diagonal matrix with β_i on the leading diagonal, and α_i and γ_i on the upper and lower first diagonals. Observe that to find $\mathbf{w}$ from $\mathbf{c}$ (and hence to invert the linear operator A) we can let w_{N-1} be initially unknown, invert T using the Thomas algorithm, to find $\mathbf{w}^*$ in terms of w_{N-1} and then fixing w_{N-1} from the relation $w_1 = c_0/\alpha_1$. Evidently the conditioning of the operator A is completely determined by the conditioning of T.

6.3. *Uniformly spaced knots*

We can make precise estimates of the conditioning of the problem of finding $\mathbf{w}$ and $\mathbf{c}$ in this case. Suppose initially that we have uniformly spaced knots $k_i = i/(N-1)$, then (6.2) reduces to $c_i = (N -$

$1)(w_{i+1}-2w_i+w_{i-1})$. The linear operator T then becomes the tri-diagonal Toeplitz matrix

$$T=(N-1)\begin{pmatrix} -2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & -2 & 1 & 0 & & & & \vdots \\ 0 & 1 & -2 & 1 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & -2 \end{pmatrix} \in \mathbb{R}^{N-2\times N-2}.$$

The standard theory of $M\times M$ Toeplitz matrices of the tri-diagonal form taken by T implies that its eigenvalues are given by

$$\lambda_k=-2(M-1)+2(M-1)\cos(\pi k/(M+1))\in(-4(N-1),0)$$

for $k=1,\ldots,M$. In particular, we have $|\lambda_1|\leq\ldots\leq|\lambda_M|$, and for large M we have

$$\begin{aligned}\lambda_1&=-2(M-1)+2(M-1)\cos\left(\frac{\pi}{M+1}\right)\\ &\approx -2(M-1)+2(M-1)\left(1-\frac{\pi^2}{2(M+1)^2}\right)=-\frac{\pi^2(M-1)}{(M+1)^2}.\end{aligned}$$

Similarly, for large M,

$$\lambda_M=-2(M-1)+2(M-1)\cos\left(\frac{M\pi}{M+1}\right)\approx -4(M-1),$$

implying that the condition number $\kappa(T)$ satisfies

$$\kappa(T)=|\lambda_M/\lambda_1|\approx 4(M+1)^2/\pi^2+1,\quad \text{with}\quad M=N-2. \tag{6.4}$$

To find the weights $\mathbf{w}$ of the FKS approximation to the target function, we must solve the linear problem $Hw=u$, where the Gram matrix $H_{i,j}=\langle\phi_i,\phi_j\rangle$, is given by

$$H=\frac{1}{6(N-1)}\begin{pmatrix} 4 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 4 & 1 & 0 & & & & \vdots \\ 0 & 1 & 4 & 1 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & 0 & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 4 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

This is also a tri-diagonal Toeplitz matrix and by standard theory has eigenvalues given by

$$6(N-1)\mu_k=4+2\cos(\pi k/(N+1))\in(2,6)\quad k=1,\ldots,N.$$

Note that $\mu_N\leq\ldots\leq\mu_1$ with $6(N-1)\mu_N>2$, $6(N-1)\mu_1<6$ and condition number $\kappa(H)=\mu_1/\mu_N<3$. As $N\to\infty$ we have $6(N-1)\mu_N\to 2$, $6(N-1)\mu_1\to 6$ and $\kappa(M)\to 3$. The normal equations for

$\mathbf{w}$ are given by $H^T H w = H^T u$. The boundedness of $\kappa(H)$ in the limit of large N lies at the heart of the regularity, and eases the construction of the FKS approximation from the normal equations. We now consider the related problem of finding the coefficients $\mathbf{c}$ of the ReLU approximation. This is of course closely linked to the problem of finding the FKS coefficients $\mathbf{w}$. For simplicity we will consider the case where $w_0 = w_{N-1} = 0$ so that the operators A and T are the same. From (6.1), we then obtain $H^T H T^{-1} c = H^T u$ and multiplication by $(T^{-1})^T$ yields $\tilde{H}^T \tilde{H} c = \tilde{H}^T u$ for the matrix $\tilde{H} = HT^{-1}$. To calculate the condition number of the matrix $\tilde{H}$ we note, that as the matrices T and H have exactly the same tri-diagonal Toeplitz structure, it follows from standard theory that they have *identical eigenvectors* $\mathbf{e}_k$, $k = 1 \ldots N$ with the eigenvector $\mathbf{e}_k$ having corresponding eigenvalues λ_k and μ_k for the matrices T and H respectively. It follows immediately, that the eigenvectors of the matrix $\tilde{M}$ are also given by the vectors $\mathbf{e}_k$ with corresponding eigenvalues $\nu_k = \mu_k/\lambda_k$. Observe that for large N we have $\nu_1 \to -6N/\pi^2$ and $\nu_N \to -1/2N$. Hence we deduce that as $N \to \infty$, $\kappa(\tilde{H}) = \nu_1/\nu_N \to 12\,N^2/\pi^2$. In particular, this shows that $\kappa(\tilde{H}) = \mathscr{O}(N^2)$ for $N \gg 1$. When solving the normal equations for the weights c of the ReLU network, we must consider the condition number κ of the matrix $\tilde{H}\tilde{H}^T$ which from the above calculation satisfies $\kappa = \mathscr{O}(N^4)$. This proves Lemma 6.1 in this case.

The value of κ for the ReLU optimisation problem is very large even for moderate values of N such as $N = 64$, and is huge compared with the $\mathscr{O}(1)$ condition number for the FKS approximation. Intuitively this is so large because the integral of the overlap of two ReLUfunctions is a quadratic function and thus its integral grows rapidly with the support size. Classical optimisation methods (such as BFGS) have convergence rate dependent on $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1) = 1 - \mathscr{O}(1/N^2)$. This leads to very slow convergence for large N. We see presently that the performance of the Adam optimiser is similarly very slow on this problem.

6.4. *Non-uniformly spaced knots*

In this case the matrices T and H are still symmetric and tri-diagonal, but are not constant along their diagonals. The Gram matrix H takes values $(k_{i+1} - k_{i-1})/3$ along its leading diagonal, and values $(k_i - k_{i-1})/6$ and $(k_{i+1} - k_i)/6$ along its lower and upper diagonals, and thus is diagonally dominant. A simple application of Gershgorin's circle theorem to estimate the location of its eigenvalues λ_j implies that

$$\frac{1}{6}\min(k_{i+i} - k_{i-1}) \le \lambda_j \le \frac{1}{2}\max(k_{i+i} - k_{i-1}).$$

Hence (consistent with the estimate on a uniform mesh)

$$\kappa(H) < 3\,\frac{\max(k_{i+i} - k_{i-1})}{\min(k_{i+i} - k_{i-1}).}$$

Note that this estimate for the condition number is smallest for uniform knots, and increases if the knot spacing is non-uniform. In contrast, each row of the matrix T has the respective coefficients:

$$\alpha_i = \frac{1}{k_i - k_{i-1}}, \quad -\beta_i = -\frac{1}{k_i - k_{i-1}} - \frac{1}{k_{i+1} - k_i}, \quad \gamma_i = \frac{1}{k_{i+1} - k_i}.$$

Observe that $\alpha_i - \beta_i + \gamma_i = 0$ for each i. It follows that the matrix T is near singular; indeed if $\mathbf{f} = (1,1,1,1,,\ldots 1)^T$ then $T\mathbf{f} = \mathbf{g} = (\gamma_1 - \beta_1, 0, \ldots, 0, \alpha_{N-2} - \beta_{N-2})^T$. Hence $\|T^{-1}\| > \|f\|/\|g\| \to \infty$ as $N \to \infty$. The condition number of T, and hence of HT^{-1} thus increases without bound as N increases. Numerical experiments strongly indicate that, as in the case of the matrix H, the condition number

increases more rapidly for non-uniform knots than uniform knots. In Figure 2 we show the condition number for the normal equations used to optimise the coefficients of the ReLU NN for the test function $u(x) = \tanh(100(x-1/4))$ considered in Section 1, with the (non-uniform) breakpoints pre-trained to be the optimal values for the IFKS. We can see in this figure that the condition number scales as $\mathcal{O}(N^4)$ as expected

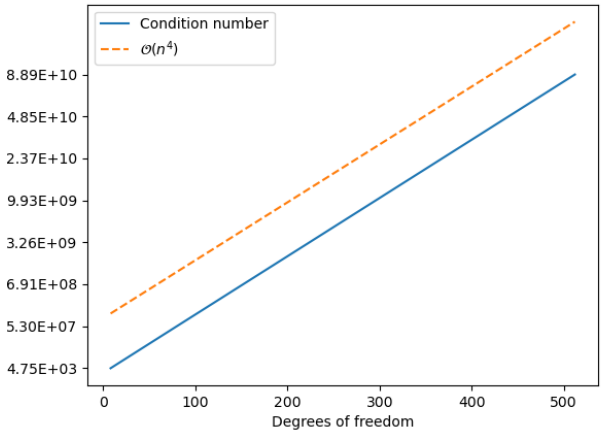


FIG. 2. The condition number of the normal equations for the ReLU NN network of width N, showing that it is $O(N^4)$.

7. Two-level training of ReLU NN and FKS networks

Given the results on the loss for the IFKS in Section 5, and the condition number estimates above, we consider two procedures to train either a shallow ReLU NN or a general FKS which combine the usual L_2 optimisation with equidistribution and the calculation of the best IFKS. These lead to procedures which allow the expressivity of these methods to be better realised with close to optimal approximations. We note that both of these methods require extra knowledge about the higher derivatives of the target function. We introduce the *combined loss function* which combines the L_2^2-approximation error with the equidistribution condition and is given by

$$L_{\text{comb}} \equiv L_2^2 + \beta L_E. \tag{7.1}$$

Observe that if β is small the equidistribution condition acts to regularise the approximation error loss. Conversely when β is larger then the approximation error acts to regularise the equidistribution condition. The latter case is surprisingly effective. Often when invoking the equidistribution condition directly in adaptive methods it is found that the resulting mesh is irregular. Adding the approximation error indirectly as a regulariser appears to smooth the resulting knot location.

Two-level training In this approach we first find the knots/breakpoints of the FKS/ReLU NN. We then find the weights/coefficients. This method is motivated by the observation that whilst the interpolating FKS $y(x) = \Pi_1 u(x)$ has quite a large error compared with the optimised solution, it is still quite close

to the final solution, has a knot distribution very close to optimal, and has the correct asymptotic convergence properties.

1. Suppose that we have either a FKS with knots k_i given directly, or a shallow ReLU NN with breakpoints given indirectly by $-b_i/a_i$. We determine the knots k_i of the Interpolating FKS $u(x) = \Pi_1 u(x)$ by minimising the equidistribution loss (5.6) for the knots k_i. This is done by directly minimising the loss function L_{comb} using Adam with large β (say $\beta = 10$ although ablation studies show that the results are fairly insensitive to the values of β.). Typically we start with a uniform distribution of knots in the interval $[0,1]$.
2. Use $\Pi_1 u(x)$ with the calculated knots as the *initial guess* for an optimising procedure based on the loss function L_2^2 in (4.2) to find the coefficients of either the FKS or the shallow ReLU NN. Or more simply, observing that the optimal knots for the FKS are very close indeed to those for the IFKS, freeze the knot locations and then solve the simple linear problem of finding the optimal weights w_i or, after preconditioning (see below), the coefficients c_i.

We see presently, by looking at a range of examples, that this method applied as above is effective and well-conditioned in giving an expressive FKS approximation. In contrast when applied to the problem of finding the ReLU NN coefficients *without* preconditioning, we find that whilst part (i) of the two-level training correctly locates the knot points k_i, applying part (ii) is still very slow for large N due to ill conditioning of the problem of finding the coefficients c_i identified in Section 6. A simple resolution for this stage of the optimisation is to precondition the problem of finding the ReLUcoefficients c_i by applying the transformation (6.1) after step (i). Hence, given a ReLU NN we consider *preconditioned two-level training of a* ReLU *NN:*

1. Fix $\mathbf{c}$ and use an equidistribution based loss function to find k_i
2. Apply T^{-1} to determine $\mathbf{w}$ from $\mathbf{c}$ using the procedure outlined in Section 6 in which we find T^{-1} efficiently in $\mathcal{O}(N)$ operations using the well known *Thomas Algorithm*. This preconditions the shallow ReLU NN optimisation problem to that of finding the optimal FKS coefficients w_i, k_i.
3. Apply (ii) to locate these coefficients.
4. Finally we transform back to find c_i from w_i using (6.1). As the matrix L is tri-diagonal the operation of multiplication by L is $\mathcal{O}(N)$.

8. Numerical Results

8.1. *Overview*

We now give a series of numerical calculations for the training and convergence of shallow univariate ReLU NNs and different linear spline approximations, including the FKS. These calculations support the theoretical results of the previous sections, and confirm the viability of the two-level algorithm we propose. The calculations are structured to first demonstrate the problematic issues with use of the standard L_2 loss based training for the ReLU NN architecture. We then look at the optimal case where we solve the equidistribution equation for the knots accurately using quadrature, and then find the weights of the optimal FKS. This gives a 'gold standard' for the accuracy which can be achieved using an adaptive linear spline approximation. We then compare the results from both of these sections with those achieved for the ReLU NN and FKS using a variety of optimisation methods, including the two-level method described in Section 7.

We do this by considering the approximation of a set of target functions using the different loss functions and training procedures described earlier. To make the comparisons between the FKS approximations and the ReLU NN comparable we assume that we have N knots k_i for $i = 0, \ldots, N-1$, with $k_0 = 0$ and $k_{N-1} = 1$. Accordingly, we consider the approximation of five different target functions $u_i(x)$ for $u(x)$ on the interval $[0,1]$ of increasing complexity:

$$u_1(x) = x(1-x), \quad u_2(x) = \sin(\pi x), \quad u_3(x) = x^{2/3},$$
$$u_4(x) = \tanh(100(x-1/4)), \quad u_5(x) = \exp(-500(x-3/4)^2)\sin(20\pi x).$$

Note that the first two target functions are very smooth, the third is singular as $x \to 0$, and the last two examples represent a smoothed step function and an oscillatory function, respectively, both with a mix of small and large length scales. Our interest will be in the importance and speed of the training, the role of the initial start, and the convergence of the resulting approximation as $N \to \infty$. Our extensive experiments on different optimisation methods such as Adam (with and without weighting, and other stabilisation methods such as weight normalisation), Gauss-Newton, and Nelder-Mead optimisers, all led to similar solutions, computation times and training behaviour. Consequently, we focus on the standard Adam optimiser for the remainder of this section.

As a summary of the results of this section, we find (as was commented on in the introduction)for all the examples, that with the usual L_2 loss (4.2) that the ReLU NN trains slowly, if at all, leading to poor accuracy in the approximation. This is due to the strong non-convexity of the loss function, and the ill-conditioning problems identified in Section 6. Better, but still sub-optimal results are obtained when training the FKS with this loss function. In contrast, optimal results, with high accuracy, are obtained when training a FKS with the equidistribution-based loss function L_{comb} and two-level training of first finding the knots and then the weights. Similar results are found when using the same method, combined with preconditioning, to train the ReLU NN.

8.2. *Poor convergence and ill-conditioning of the* ReLU *NN training*

In Section 4 we claimed that the training of ReLU NN with the standard L_2 loss function led to poor results, and in Section 6 proved the ill-conditioning of the training of the scaling coefficients c_i of this representation. We now present numerical evidence for these statements. We consider first the training of a shallow ReLU NN with the usual L_2^2 loss function given in (4.2), with width $W = 16$, and consider the approximation of the target functions $u_i(x), i = 2,3,5$. For this calculation the default PyTorch parameter initialisation is used, and the location of the implicit knot locations over 50,000 optimisation iterations using Adam with a learning rate of 10^{-3} are illustrated in the top part of Figure 3. The lower part of Figure 3 shows the quality of the function approximation, with the breakpoints/implicit knot locations plotted. We see that during the course of the training procedure the knots can cross over, merge and leave the domain $[0,1]$.

We next repeat the above calculation, but with the constraint on the starting parameters that the initial implicit knot locations given by (3.1) are all required to lie in the interval $[0,1]$. The corresponding results are shown in Figure 4. We can see that the results are better than those presented in Figure 3 but are still far from optimal, and show evidence of knot crossing and merging during training.

The resulting values of the L_2^2 loss function in the respective cases are:

$$\text{Random}: \quad u_2 : 3.948 \times 10^{-3}, \quad u_3 : 2.611 \times 10^{-5}, \quad u_5 : 8.524 \times 10^{-3},$$

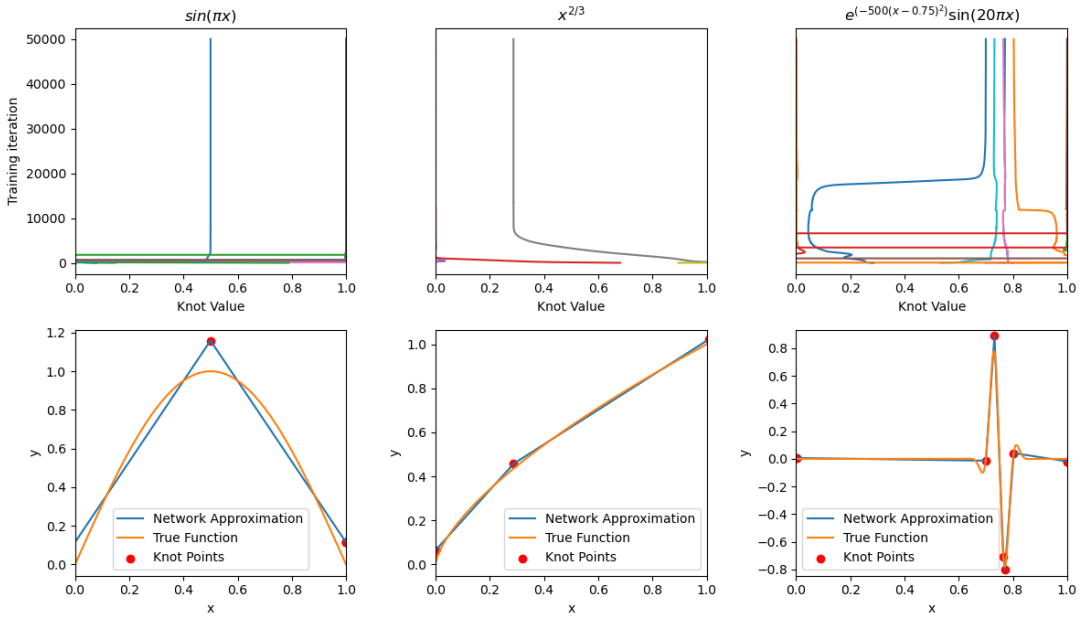


FIG. 3. Comparison of (top) Knot evolution with standard Gaussian initialisation and (bottom) the trained approximation (in blue) with the knot points indicated for different target functions (in orange).

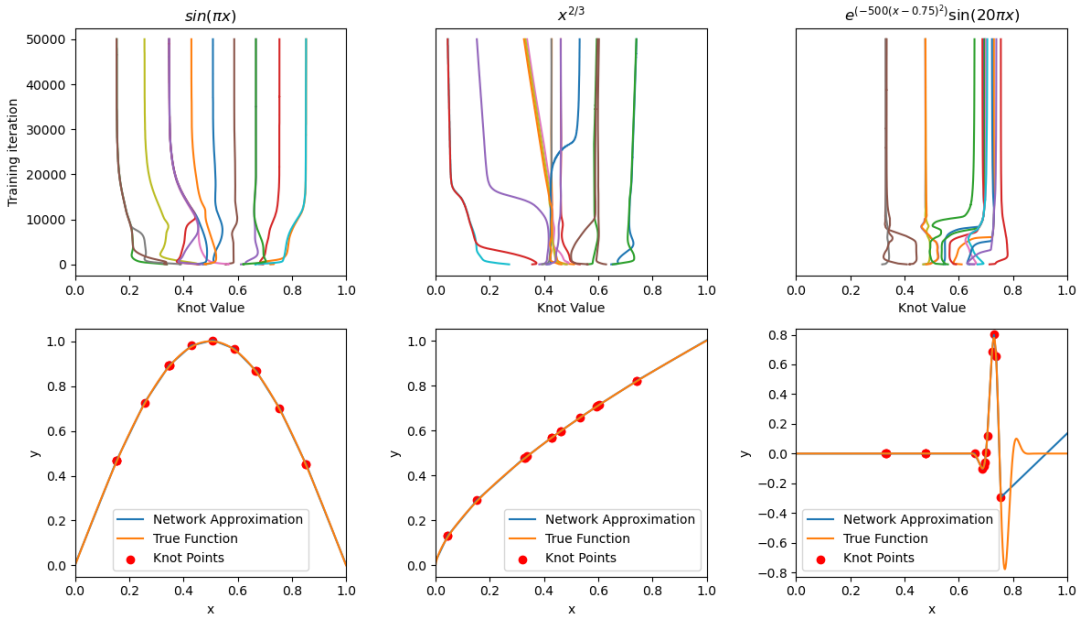


FIG. 4. Comparison of (top) Knot evolution, with initialisation in $[0,1]$ and (bottom) the trained approximation with the knot points indicated for different target functions.

$$\text{Constrained}: \quad u_2 : 4.77 \times 10^{-6}, \quad u_3 : 2.30 \times 10^{-6}, \quad u_5 : 8.24 \times 10^{-3}.$$

We note that the loss values for the constrained start are in general better than the large loss values we obtain for the random start. However, we show presently that they are much larger than the loss function values given by the FKS.

These calculations demonstrate that the standard machine learning based techniques, when used in a simple example of functional approximation, can lead to a significantly sub-optimal solution in each of the three cases of the target functions. This can be seen by the distribution of the knots which is far from optimal. Note that they are irregular and unevenly spaced whereas the optimal knots would be expected to be symmetric, and regularly spaced. Furthermore the knots for the target functions u_3 and u_5 show no sign of clustering close to the singularity at $x = 0$ (in the case of u_3) or the points of rapid variation (in the case of u_5). In all cases the training of the knots is erratic. We conclude, as said in Section 4, that the commonly used training methods and loss functions for training a shallow ReLU NN perform badly on even simple target functions. We now see how we can improve on this performance.

As proven in Section 6, the problem with training a ReLU NN lies deeper than finding the optimal knots, and is also impacted by the poor conditioning of the problem of finding the associated coefficients c_i even when the optimal knots for the IFKS are known. The case of taking $N = 64$ and the target function $u_4(x)$ has already been illustrated in Figures 1 (right) and 2 where we use Adam with an equidistribution based loss function in the first stage of the two-level method to train both an FKS and a ReLU NN to find the optimal knots k_i. Starting from these knot locations and with an initially random distribution of the scaling coefficients/weights, we then apply the second stage to further train both networks to find the optimal values of the coefficients c_i for the ReLU NN and w_i for the FKS, respectively. It is clear from Figure 1 that the FKS trains much more quickly to a much more accurate approximation than the ReLU NN for this target function. This is due to the poor conditioning of the problem for large N associated with the condition number $\mathcal{O}(N^4)$ of the normal equations identified in Section 6 and in Figure 2. For example if $N = 64$ then a direct calculation for this specific approximation problem gives $\kappa_{\text{ReLU}} = 6.3 \times 10^7$ and $\kappa_{\text{FKS}} = 3.7$. Similar poor conditioning is observed for all of the target functions considered.

8.3. *Optimal FKS approximations using MMPDEs*

By way of contrast, we now show that very high approximation accuracy can be achieved for a correctly trained FKS obtained by using moving mesh PDE (MMPDE) based methods described in Section 5.2 to find the optimal knots, and then least-squares optimisation to find the weights. This gives a 'gold standard' result that can be used for comparison with optimisation based methods. To do this calculation we consider solving a regularised form of equations 5.8,5.9, and define a monitor function $m_j = (\varepsilon_j^2 + u_j''^2)^{1/5}$. The precise value of the regulariser ε_j is not toocritical, but has to be chosen with some care when the value of u_j'' varies a lot over the domain to ensure that regions where u_j'' is relatively small still have some knots within them. We take $\varepsilon_j = 0$ for $j = 1,2$ and $\varepsilon_j = 1$ for $j = 3,4,5$. From 5.8,5.9 it follows that the optimal knot points $k_i, i = 0,\ldots,N-1$, are given by $x(\xi)$ where $\xi = i/(N-1)$ and $x(\xi)$ satisfies

$$\frac{dx}{d\xi} = \frac{\int_0^1 m_j(\tilde{x})\, d\tilde{x}}{m_j(x)}, \quad x(0) = 0. \tag{8.1}$$

To find the best knot points k_i for the IFKS we solve (8.1) and evaluate the integral in the expression by using a high-order Gear solver with a high tolerance. We then find the optimal weights by solving

the L_2 minimisation problem. Note that for the target functions $u_1(x) = x(1-x)$ and $u_3(x) = x^{2/3}$ that the optimal IFKS knot points are given analytically (see the results in the Supplementary Material) by $k_i = i/(N-1)$ and $k_i = (i/(N-1))^{15/7}$, respectively.

We present results for each of the target functions in Figure 5. For each target function we study the convergence of the approximation by plotting the L_2^2 error as a function of N in the three cases of (i) (blue) the PWL interpolant defined over a uniform mesh, (ii) (orange) the optimal IFKS interpolant, and (iii) (green) the fully trained FKS. In each case the training was rapid.

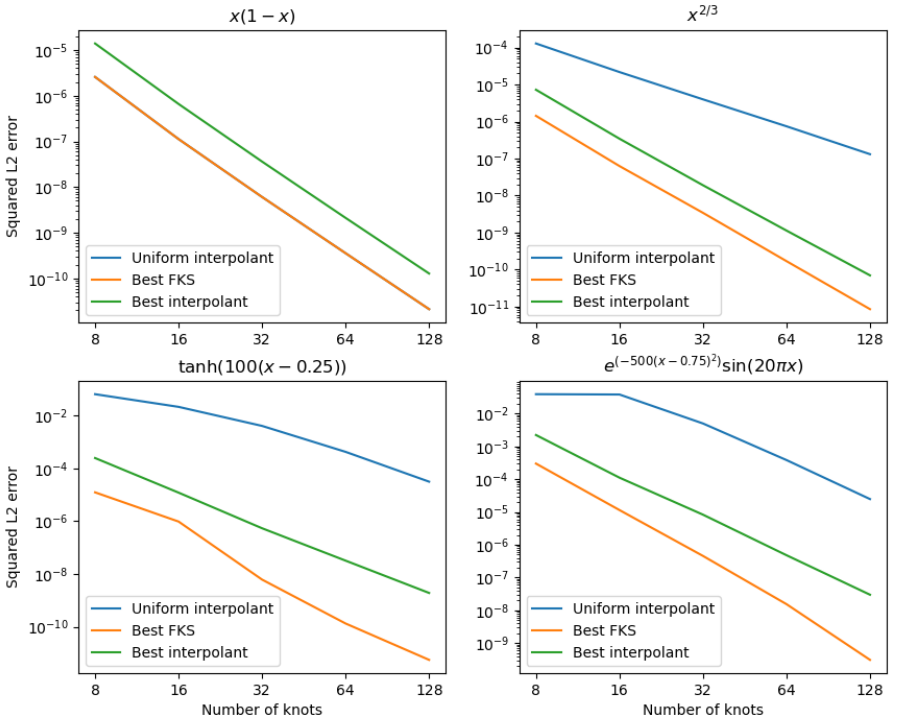


FIG. 5. Comparison of the L_2^2 loss for linear spline approximations of four different target functions with different numbers of knots: blue - PWL interpolant on a uniform mesh, orange- optimal IFKS, green - optimal FKS

We see that the optimal IFKS and FKS both show $\mathcal{O}(1/N^4)$ convergence of the loss function. The FKS gives the best results as expected, often significantly better than the optimal IFKS. In this context the 'optimal' FKS achieves the best expressivity of piecewise linear approximations which includes the ReLU approximation, with errors of around 10^{-10} when $N = 64$. The error of the best L_2^2 PWL approximation on a uniform mesh is always much poorer, by several order of magnitude, than the IFKS or FKS. But, as we have seen, it is much better than the naively trained ReLU network. For the *smooth* target functions u_1, u_2, u_4, u_5 we see $\mathcal{O}(1/N^4)$ convergence of the uniform PWL approximation

for values of N so that $h = 1/N$ is smaller than the smallest length scale of the target function, but with a much larger constant of proportionality than either the optimal FKS or the optimal IFKS. In the case of the *singular* target function $u_3(x)$ we see a slower convergence of the uniform PWL approximation at the theoretical rate of $\mathcal{O}(1/N^{7/3})$.

8.4. ReLU *NN and FKS training using optimisation*

In this subsection, we present a set of numerical results for directly training a ReLU NN, a linear spline (without free knots), and a FKS for each of the target functions using an *optimisation approach*. In all case we use the Adam optimiser with a learning rate of 10^{-3}. We consider the use of different loss functions, including the equidistribution-linked loss function L_{comb} in (7.1) and using the two-level training procedure described in Section 7 where we first train the knots of the approximation and then the weights, in the case of the ReLU NN with and without preconditioning.

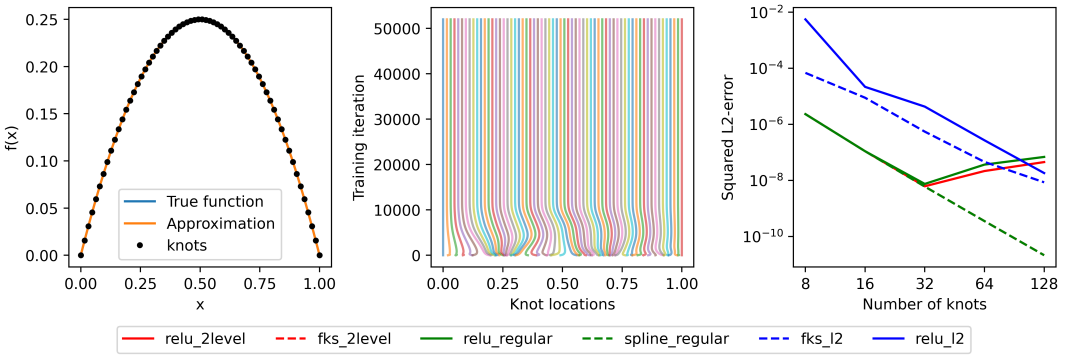
We present the results as follows. For each target function $u_i(x)$ we consider using the following methods: (a) *spline-regular*: the best linear-spline least squares approximation, which takes a uniform knot distribution and optimises the weights. (b) *ReLU-regular*: A standard ReLU NN. This is the usual implementation of the ReLU NN approximation in PyTorch as considered in the previous subsection, with random initialisation of the Adam optimiser but with the knot locations constrained to lie in $[0,1]$. (c) *ReLU-L2*: a ReLU NN and *FKS-l2*: a FKS trained using the standard L_2^2 loss function. (d) *ReLU-2level*: a ReLU NN trained in the two-level manner described in Section 7.1 to first to locate the breakpoints k_i and then the coefficients c_i *without preconditioning*, *FKS-2level*: a FKS trained using the two-level method which is equivalent to the ReLU NN trained using the two-level method *with preconditioning*. For cases (c) and (d), we take the initialisation for the Adam optimiser to be the piecewise linear interpolant of the target function with uniform knots $k_i = i/(N-1), i = 0 \ldots N-1$ represented either in the form of the ReLU NN or as a piecewise linear spline approximation.

We show our numerical results for different target functions in Figure 6 in terms of different aspects: (i) The function approximation of the FKS trained using the two-level approach method when $N = 64$. (ii) The convergence of the knot values during training when using the equidistribution based loss function, where we plot (on the x-axis) the location of the knots as a function of time t on the y-axis. This figure is nearly identical for both the FKS and the ReLU NN approximations and the smooth evolution of the knots should be compared to erratic behaviour seen in Figures 3 and 4. (iii) The convergence of the L_2^2 approximation error of each method as a function of N,

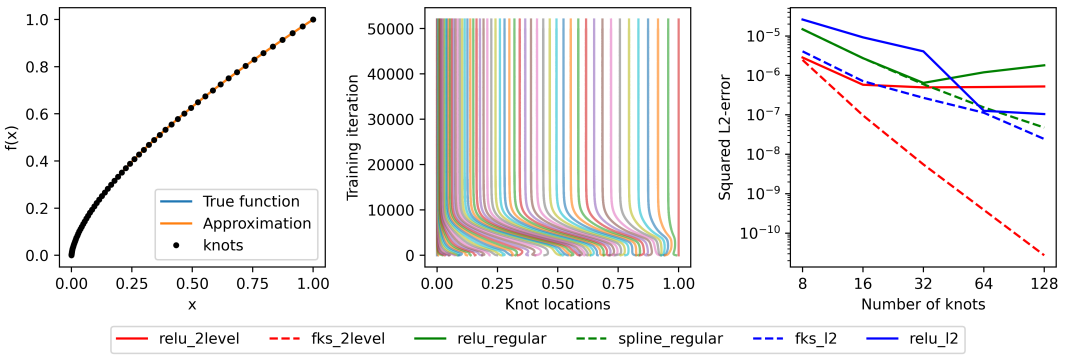
The *regular classical least squares linear spline approximation on a uniform mesh of mesh spacing* $h = 1/N$ so that there is no training of the knots, works well for the first two smooth target functions with convergence $\mathcal{O}(N^{-4})$. For the singular target function $u_3(x)$ it converges but at the sub-optimal rate of $\mathcal{O}(N^{-7/3})$ described in the Supplementary Material. (Exactly the same behaviour is seen for the piecewise linear interpolant on a uniform mesh.) In the case of the smooth target functions $u_4(x)$ and $u_5(x)$ we see sub optimal convergence when $h = 1/N$ is greater than the smallest length scale of the problem (which is $1/100$ for u_3 and $1/\sqrt{500} = 1/22.3$. For larger values of N (and smaller values of h) we see $\mathcal{O}(1/N^4)$ convergence. In all cases the error of the regular spline is much larger than that of the best FKS approximation, but in general better than that of the ReLU NN without preconditioning.

The *basic* ReLU *NN* trained using the direct L_2^2 loss function consistently performs badly. Whilst the approximation error does in general decrease with N, it is usually the worst of all the approximations

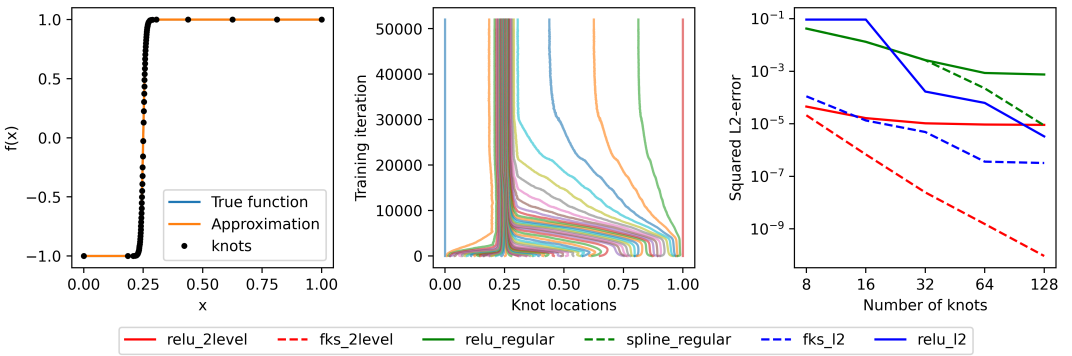
(a) Target function $u_1(x) = x(1-x)$



(b) Target function $u_3(x) = x^{2/3}$



(c) Target function $u_4(x) = \tanh(100(x-1/4))$



(d) Target function $u_5(x) = \exp(500(x-3/4))^2)\sin(x)$.

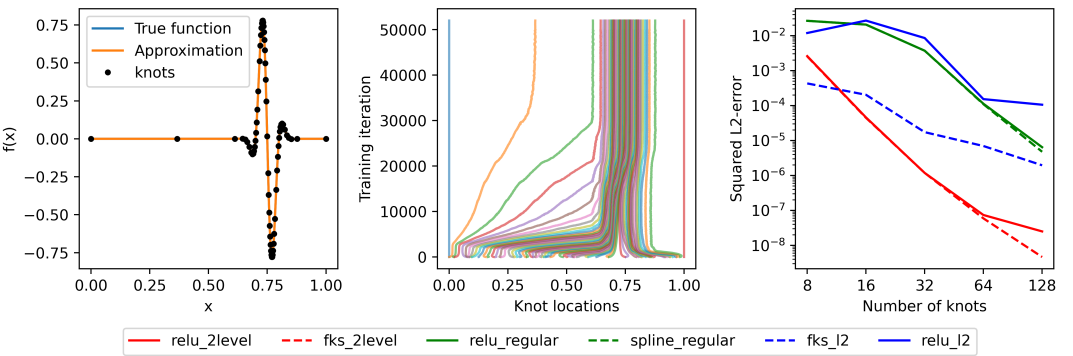


FIG. 6. Comparison of (i) Function approximation, (ii) knot evolution, (iii) Convergence for different target functions

by a significant margin. It also takes a much longer time to train than any other method due to the ill-conditioning of the system.

The ReLU *NN trained using the direct two-level approach without preconditioning* in which the breakpoints k_i are first trained and then the coefficients are calculated *without* using preconditioning, works in general rather better than the basic ReLU NN above, particularly for the lower values of N where conditioning issues are less severe, but is easily outperformed by the FKS, particularly for larger N where ill-conditioning issues arise. In the two-level approach the knot locations are found accurately in stage (i) of the training using the equidistribution based loss function and are presented in the middle figure. Again, contrast this smooth evolution with the erratic convergence of the knots of the basic ReLU NN presented earlier in Figures 3 and 4. However, having found the knot locations correctly, finding the resulting weights c_i is ill-conditioned, as expected from the results of Section 6. Consequently the convergence to the correct solution is very slow, as was illustrated in Figure 1 (right).

The *FKS trained using the direct* L_2^2 *loss* function performs better than the above approximations, but a lot worse than the optimal FKS (see below). We do not in general see $\mathcal{O}(1/N^4)$ convergence in this case.

The *full FKS (with free knots and weights) trained using the two-level training (first training the knots using equidistribution)* then the weights, is consistently the best performing method, alongside the near identical example of the *preconditioned two-level training of the* ReLU *NN* with similar results in both cases. The results show excellent and consistent convergence at a rate of $\mathcal{O}(1/N^4)$. The convergence of the knots in all cases is regular and monotone, indicating a degree of convexity to the problem. The resulting knots are regularly spaced, showing symmetry and clustering where needed. The training converges rapidly in all cases. Note that pre-training the knots using the 'gold standard' MMPDE as discussed in Section 8.3 gives a slightly smaller error after optimisation than the best trained FKS/ReLU NN (although the difference is not large). This is to be expected as the MMPDE has employed an accurate solver to determine the knot points.

8.5. *Conclusions from the results*

A naively (in the sense of using the usual training procedures) trained shallow ReLU NN, always gives a poor approximation of the target functions. The knots/breakpoints are generally badly placed, especially if we start from a random parameter set. Starting from a uniform set of knots gives an improvement, but still leads to a poor approximation. The training method is also very ill-conditioned.

Two-level training (using equidistribution to find the knots/breakpoints but without using preconditioning) of a ReLU NN is significantly better than non-equidistribution based training, and the breakpoints are close to optimal locations. However the training of the coefficients $c_i$is far from optimal due to ill-conditioning for larger values of N, leading to an approximation that is still far from being as expressive as it could be.

Two-level training of an FKS (using the optimisation approach) is better than non-equidistribution based training as it is faster and more accurate and gives a good approximation which is very expressive and close to the optimal theoretical result (and that obtained using the MMMPDE approach).

Preconditioning the ReLU NN in the two-level training (effectively turning it into an FKS) gives almost identical results to using an FKS directly. We conclude that to deliver the theoretical expressivity of the

ReLUit is necessary (and sufficient) to find both the optimal breakpoints k_i , and to precondition the optimisation procedure for finding the coefficients c_i.

9. Generalisations

We have so far confined the theory, and experiments, to the case of the shallow ReLU NN. This is done to allow a direct theoretical and numerical comparison with the linear spline FKS. We now make some preliminary investigations of extensions of these results to more general machine learning examples.

9.1. *More general activation functions*

In applications activation functions other than ReLU are often used, such as leaky ReLU, ReLU-cubed, tanh and sigmoid. As a first calculation we consider the standard training, using Adam, of a shallow NN with tanh activation on the same target functions $u_i(x)$ as before, with the L_2^2 loss. In Table 1 we present the L_2^2 error when training this network with N knots and P parameters. The results for the smooth functions u_1, u_2 and mildly singular function u_3 are rather better than those for the shallow ReLU NN without preconditioning and this problem appears to be better conditioned than the ReLUapproximation in this case. However we see poor performance on the rapidly varying test function u_4, and the highly oscillatory test function u_5. If we consider the optimally trained ReLU NN (with the preconditioned two-level approach), for example when $N = 64$, as seen in the results presented in Section 8, we see errors of 10^{-10} or smaller in all of the examples. These errors are consistently much smaller than those in Table 1. It is worth noting that a shallow ReLU-cubed NN is formally equivalent (in a similar manner to the ReLU NN) to a *cubic* B-spline FKS. In this case it is possible to precondition the coefficients of the ReLU-cubed network by mapping them onto the coefficients of the cubic B-spline FKS. This recovers the high expressivity of the latter case [9].

N	u_1	u_2	u_3	u_4	u_5	P
8	4.6×10^{-9}	1.70×10^{-6}	3.55×10^{-4}	1.08×10^{-3}	1.77×10^{-2}	25
16	1.56×10^{-9}	1.20×10^{-8}	1.62×10^{-7}	8.01×10^{-4}	1.53×10^{-2}	49
32	1.21×10^{-8}	1.41×10^{-8}	1.50×10^{-7}	6.17×10^{-4}	1.64×10^{-2}	97
64	5.25×10^{-9}	8.88×10^{-9}	2.15×10^{-7}	5.54×10^{-4}	1.77×10^{-2}	193
128	2.21×10^{-8}	8.00×10^{-9}	2.15×10^{-7}	4.40×10^{-4}	1.90×10^{-2}	385

TABLE 1 *The L_2^2 error of training a shallow network with the* tanh *activation function, without preconditioning, to approximate the test functions $u_i(x)$.*

9.2. *Deeper networks*

In general approximations deep networks are usually considered instead of shallow ones. We now attempt to extend the shallow univariate approximation framework to two hidden layers (often called a three layer network) and describe a block–structured preconditioning strategy that stabilises and accelerates training.

As observed earlier, a deep network using ReLUactivation is again a piecewise linear function of its input, however, unlike the shallow case, the link between the knots of a FKS and the breakpoints of the

deep network approximation is very subtle [6] and it is much harder to develop a direct analytical theory in this case. Instead we briefly present some preliminary numerical comparisons and draw comparisons with the earlier analysed case of shallow networks. For this we use a two hidden-layer (depth–3) ReLU network with widths W_1 and W_2:

$$z^{(1)}(x) = A^{(1)}x + b^{(1)}, \qquad h^{(1)}(x) = \mathrm{ReLU}\big(z^{(1)}(x)\big), \tag{9.1}$$

$$z^{(2)}(x) = A^{(2)}h^{(1)}(x) + b^{(2)}, \qquad h^{(2)}(x) = \mathrm{ReLU}\big(z^{(2)}(x)\big), \tag{9.2}$$

$$\widehat{f}(x;\Theta) = b^{(3)} + w^{\top}h^{(2)}(x), \tag{9.3}$$

with parameters $\Theta = \{A^{(1)}, b^{(1)}, A^{(2)}, b^{(2)}, w, b^{(3)}\}$. In component–wise form,

$$\widehat{f}(x;\Theta) = b^{(3)} + \sum_{j=1}^{W_2} w_j \,\mathrm{ReLU}\left(b_j^{(2)} + \sum_{i=1}^{W_1} A_{ji}^{(2)}\,\mathrm{ReLU}\big(b_i^{(1)} + A_i^{(1)\top}x\big)\right). \tag{9.4}$$

On $[a,b]$, $\widehat{f}$ remains continuous piecewise–linear (CPWL); depth controls how first–layer breakpoints are mapped through the second layer to create additional affine regions. Including the final bias $b^{(3)}$ is important to represent targets with nonzero mean or persistent negative lobes; empirically it enables a faithful approximation of negative portions of certain targets that are missed otherwise.

9.2.1. Comparison with a preconditioned shallow network

To make a comparison with the shallow ReLU NN we compare a 2 hidden-layer ReLUNN of width $W_1 = W_2 = W = 5$ and 46 trainable parameters, with an FKS with 16 knots and 49 trainable parameters. The 2 hidden-layer NN was trained on the functions $u_i(x)$ using Adam with the L_2^2 loss function and no preconditioning. Results are presented in Table 2 in which we show the variance of results obtained with a number of initial choices of the parameters. In Table 2 we see poor convergence, similar to that observed earlier for the shallow ReLU NN. In all cases the errors are very much larger than the comparable two-level trained FKS. The behaviour of the deep ReLU NN is rather erratic and is critically dependent on the initial values of the parameters, suffering from the same issues with ill-conditioning as were observed earlier for the shallow ReLU NN. Similar issues with ill-conditioning of two hidden-layer ReLU NN networks with uniform breakpoints for the first layer, have been seen empirically in the computational results presented in [16].

	u_1	u_2	u_3	u_4	u_5
L_2^2 error	4.2×10^{-3}	5.47×10^{-2}	3.45×10^{-2}	1.8×10^{-1}	1.85×10^{-2}
Variance	9.71×10^{-5}	1.42×10^{-2}	1.35×10^{-2}	5.40×10^{-2}	1.09×10^{-4}

TABLE 2 *Results of training a two hidden layer network with 5 nodes on the test functions.*

9.2.2. Preconditioning a deep network

Clearly, preconditioning deep networks is important for effective optimisation. One approach used in [16] is to replace *all* of the activation functions by linear splines, hence essentially creating a deep FKS. The latter network is shown empirically in [16] to be well conditioned. As an alternative approach,

we consider here whether we can precondition the original deep ReLU NN. There is no obvious map between the coefficients of a deep ReLU NN network and a FKS, and hence the earlier form of preconditioning in which we mapped the shallow ReLu NN coefficients to the FKS weights, cannot be applied directly. However, it is possible to consider preconditioning the linear operator applied to the linear operator in each gradient descent step of the optimisation procedure for the coefficients of the deep ReLU NN. As a preliminary study (with more details in [9]) we consider a 2-hidden layer ReLU NN with a block preconditioner B for the linear operator in the gradient descent, where B is based on the linear operator T used for the shallow network. To use this we group the parameters as blocks $\theta = (A^{(1)}, b^{(1)}, A^{(2)}, b^{(2)}, w, b^{(3)})^T$ We then use a block–diagonal operator B whose blocks mirror these groups. Accordingly we consider two potential deep NN preconditioners $B = B_{1,2}$ inspired by the recursive structure in the expression (9.4). However, it is difficult to mathematically prove the improvement on the condition number after using this preconditioner, and further research is needed to find a good preconditioner.

$$B_1 \;=\; \mathrm{blkdiag}(I,\, I,\, (T)\otimes(T),\, I,\, I,\, 1)\,, \quad B_2 \;=\; \mathrm{blkdiag}(I,\, I,\, (T)\otimes(T),\, I,\, T,\, 1)\,. \tag{9.5}$$

where $\otimes$ denotes the Kronecker product. With such a symmetric positive definite preconditioner $B \succ 0$, and for a loss function L, we update the coefficients via the preconditioned gradient descent step

$$\theta_{k+1} = \theta_k - \eta B \nabla L(\theta_k) \equiv \theta_k - \eta B H \theta_k,$$

which converges if $0 < \eta < 2/\lambda_{\max}(BH)$.

In Figure 7 we present the results of using the preconditioner B_1 for approximating the function u_4 described earlier. The figure shows the loss and function approximation both with and without preconditioning (Vanilla) for a variety of different learning rates. For these tests we take $W_1 = W_2 = 10$ and use the ReLU activation. A Kaiming initialisation of the parameters is used for all of the tests.
Our observations are as follows:

Convergence speed: The preconditioned runs show large oscillations for higher learning rates and a slower descent overall. In contrast the vanilla case (especially with a learning rate $= 0.1/0.05$) drops quickly and reaches its plateau earlier. No speed up from using the B_1 preconditioner is observed.
Error magnitude. Under a comparison between the two methods using the optimal learning rate for each , the vanilla method achieves a lower terminal loss (deeper log-loss band) than the preconditioned model, whose best run settles to a higher error and is more sensitive to the learning rate.
Visual match. Both methods learn the sharp step close to the ground truth; and the final curves are near–indistinguishable for their best learning rates.

We conclude that for this deep ReLU NN setup the preconditioner B_1 does **not** deliver an improvement. Indeed the vanilla method is faster and attains a slightly lower peak error, while visual fidelity is essentially tied. The results from using the B_2 preconditioner were similar. Clearly further work is required to find effective preconditioners for this deep case.

10. Conclusions and future work

The results of this paper have shown that it is hard to train a shallow ReLU NN to give a particularly accurate approximation for either smooth or singular target functions if a standard method is used. In contrast, using a two-level equidistribution based training approach, *combined* with preconditioning

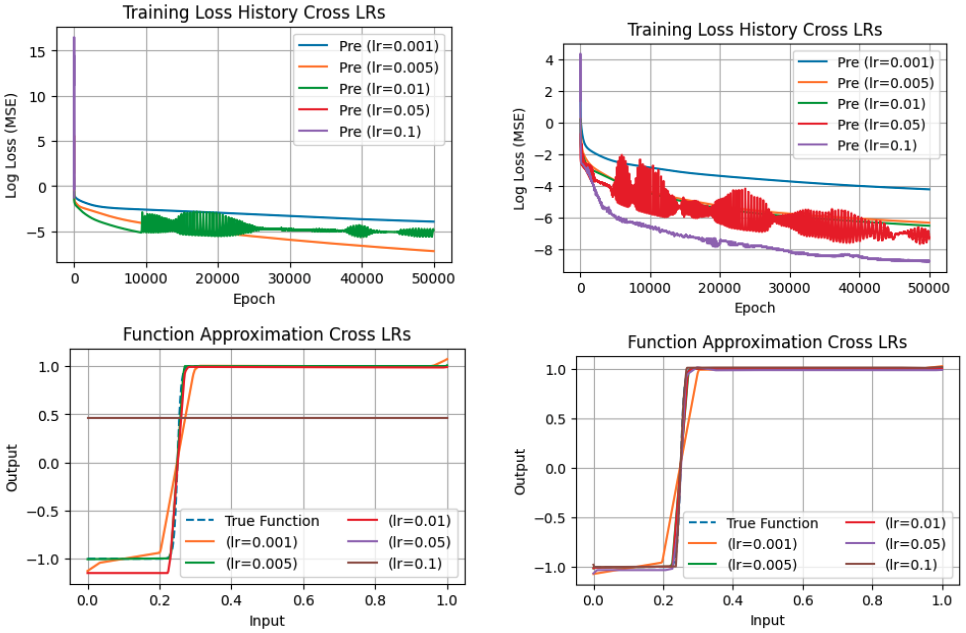


FIG. 7. Target function= u4 , Deep ReLU NN Width=10, init=Kaiming (left) preconditioned using B_1 (right) Vanilla

it is possible to train a ReLU NN formally equivalent FKS to give very accurate approximations on the same set of target functions. Hence the high level of expressivity theoretically possible for the ReLU NN approximation is achieved in training by the FKS and the preconditioned ReLU NN, but not, in practical training using standard methods, by the standard ReLU NN. The reason for this IS two-fold. Firstly, it is necessary to have (in both the FKS and the ReLU NN) control over the knot location using (for example) an equidistribution based loss function. Secondly, the process of training the weights of the FKS is much better conditioned than that of finding the scaling coefficients of the ReLU functions which have a global support. As we have seen, the combination of these two factors leads to poor ReLU NN approximations. Both issues have been overcome when training a ReLU NN by using an equidistribution based loss function to find k_i and then preconditioning the ReLU NN problem to give it the same structure as an FKS before finding the coefficients c_i. By doing this we can then reliably realise the full expressivity of the ReLU NN approximation.

In this paper we have deliberately mainly confined ourselves to piecewise linear function approximations in one spatial dimension by shallow networks. This is because we can then make a direct comparison between a shallow ReLU NN and an FKS, and can develop both effective loss functions, and preconditioning strategies, which ensure that the optimisation approaches work well and delivers a highly accurate approximation. Preliminary work on deep ReLU NN, show that preconditioning them is needed (to have the same performance as the preconditioned shallow network), but that finding an effective preconditioner is not straightforward and requires more work.

Future work developing the results of this paper include (i) developing preconditioning strategies for deep univariate ReLU NN networks, (ii) extending the results to higher dimensions, (iii) using these

results to understand, and improve, the convergence of (neural net based) PINN approximations [20] for the solution of differential equations (see preliminary work in [9]), and other aspects of (scientific) machine learning.

REFERENCES

1. R. ADAMS AND J. FOURNIER, *Sobolev Spaces*, Academic Press, 2003.
2. P. BOHRA, J. CAMPOS, H. GUPTA, S. AZIZNEJAD, AND M. UNSER, *Learning activation functions in deep (spline) neural networks*, IEEE Open Journal of Signal Processing, 1 (2020), pp. 295–309.
3. C. DE BOOR, *Good Approximation by Splines with Variable Knots*, Birkhäuser Basel, Basel, 1973, pp. 57–72.
4. C. DE BOOR, *Good approximation by splines with variable knots. ii*, in Conference on the Numerical Solution of Differential Equations, G. A. Watson, ed., Berlin, Heidelberg, 1974, Springer Berlin Heidelberg, pp. 12–20.
5. C. DE BOOR, *A practical guide to splines*, Springer-Verlag, 2001.
6. R. DEVORE, B. HANIN, AND G. PETROVA, *Neural network approximation*, Acta Numerica, 30 (2021), p. 327–444.
7. R. A. DEVORE, *Nonlinear approximation*, Acta Numerica, 7 (1998), p. 51–150.
8. R. A. DEVORE AND G. G. LORENTZ, *Constructive Approximation*, Grundlehren der mathematischen Wissenschaften, Springer Berlin Heidelberg, 1993.
9. Y. DU, *Preconditioning of Neural Networks with applications to PINNS*, master's thesis, Computer Science, UCL, 2025.
10. W. E AND B. YU, *The Deep Ritz Method: A deep learning-based numerical algorithm for solving variational problems*, Communications in Mathematics and Statistics, 6 (2018), pp. 1–12.
11. K. ECKLE AND J. SCHMIDT-HIEBER, *A comparison of deep networks with ReLU activation function and linear spline-type methods*, Neural Networks, 110 (2019), pp. 232–242.
12. P. GROHS AND G. KUTYNIOK, *Mathematical Aspects of Deep Learning*, Cambridge University Press, 2022.
13. T. GROSSMAN, U. KOMOROWSKA, J. LATZ, AND C.-B. SCHOENLIEB, *Can physics-informed neural networks beat the finite element method?*, IMA Journal of Applied Mathematics, 89 (2024), pp. 143—-174.
14. M. HANSSON AND C. OLSSON, *Feedforward neural networks with ReLU activation functions are linear splines*, PhD thesis, Mathematical Sciences, Lund, 2017.
15. J. HE, L. LI, J. XU, AND C. ZHENG, *ReLU deep neural networks and linear finite elements*, Journal of Computational Mathematics, 38 (2020), pp. 502–527.
16. Q. HONG, J. SIEGEL, Q. TAN, AND J. XU, *On the activation function dependence of the spectral bias of neural networks*, (2022), https://arxiv.org/abs/2208.04924v3.
17. K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.
18. W. HUANG, Y. REN, AND R. D. RUSSELL, *Moving mesh partial differential equations (MMPDES) based on the equidistribution principle*, SIAM Journal on Numerical Analysis, 31 (1994), pp. 709–730.
19. W. HUANG AND R. RUSSELL, *Adaptive Moving Mesh Methods*, Applied Mathematical Sciences, Springer New York, 2012.
20. G. E. KARNIADAKIS, I. G. KEVREKIDIS, L. LU, P. PERDIKARIS, S. WANG, AND L. YANG, *Physics-informed machine learning*, Nature Reviews Physics, 3 (2021), pp. 422–440.
21. D. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in International Conference on Learning Representations (ICLR), San Diega, CA, USA, 2015.
22. N. KOVACHKI, Z. LI, B. LIU, K. AZIZZADENESHELI, K. BHATTACHARYA, A. STUART, AND A. ANANDKUMAR, *Neural Operators: Learning maps between function spaces with applications to PDEs*, Journal of Machine Learning Research, 24 (2023), pp. 1–97.
23. M. J. D. POWELL, *Approximation Theory and Methods*, Cambridge University Press, 1981.
24. N. RAHAMAN, A. BARATIN, D. ARPIT, F. DRAXLER, M. LIN, F. HAMPRECHT, Y. BENGIO, AND A. COOURVILLE, *On the spectral bias of neural networks*, ICML, (2019).

25. J. SAHS, R. PYLE, A. DAMARAJU, J. O. CARO, O. TAVASLIOGLU, A. LU, F. ANSELMI, AND A. B. PATEL, *Shallow univariate relu networks as splines: Initialization, loss surface, Hessian, and gradient flow dynamics*, Frontiers in Artificial Intelligence, 5 (2022).
26. J. SHENOUDA, Y. ZHOU, AND R. NOWAK, *ReLUs are sufficient for learning implicit neural represeantations*, (2024), https://arxiv.org/abs/2406.02529v2.
27. W. SONG, M. ZHANG, J. WALLWORK, J. GAO, Z. TIAN, F. SUN, M. PIGGOTT, J. CHEN, Z. SHI, X. CHEN, AND J. WANG, *M2N: Mesh movement networks for PDE solvers*, Advances in Neural Information Processing Systems, 35 (2022).
28. D. YAROTSKY, *Error bounds for approximations with deep ReLU networks*, Neural Networks, 94 (2017), pp. 103–114.
29. S. ZHANG, H. ZHAO, Y. ZHONG, AND H. ZHOU, *Why shallow networks struggle to approximate and learn high frequencies*, (2025), https://arxiv.org/abs/2306.17301v3.