Annals of Operations Research
https://doi.org/10.1007/s10479-023-05315-4

ORIGINAL RESEARCH

Neural network expression rates and applications of the deep parametric PDE method in counterparty credit risk

Kathrin Glau[1] · **Linus Wunderlich**[1]

Accepted: 20 March 2023

Abstract
The recently introduced deep parametric PDE method combines the efficiency of deep learning for high-dimensional problems with the reliability of classical PDE models. The accuracy of the deep parametric PDE method is determined by the best-approximation property of neural networks. We provide (to the best of our knowledge) the first approximation results, which feature a dimension-independent rate of convergence for deep neural networks with a hyperbolic tangent as the activation function. Numerical results confirm that the deep parametric PDE method performs well in high-dimensional settings by presenting in a risk management problem of high interest for the financial industry.

Keywords Deep neural networks · Deep parametric PDE method · DNN approximation theory · DNN expression rates · High-dimensional partial differential equations · Option pricing · Exposure calculation

1 Introduction

In this article, we expand the analysis of the deep parametric PDE method, provide further numerical results and demonstrate performance gains in a core computational problem in counterparty credit risk. The deep learning-based solver introduced in Glau and Wunderlich (2022) offers efficient approximations to high-dimensional parametric partial differential equations (PDEs) and builds on Sirignano and Spiliopoulos (2018). For instance to comply with regulatory policies, financial institutions have to regularly evaluate a massively large number of high-dimensional PDEs. Particularly, large portfolios and scenario calculations demand for PDE solutions for different parameters in one stochastic model. In order to advance these calculations in risk management, efficient high-dimensional parametric PDE solvers are required. The efficiency of classical PDE solvers deteriorates for high dimensions due to the curse of dimensionality. We show that the approximation rate of deep neural

✉ Linus Wunderlich
L.Wunderlich@qmul.ac.uk

Kathrin Glau
K.Glau@qmul.ac.uk

[1] School of Mathematical Sciences, Queen Mary University of London, Mile End Road, London E1 4NS, UK

Published online: 13 April 2023

networks (DNNs) used in the deep parametric PDE solver is independent of the dimension of the problem.

The deep parametric PDE method has a strong mathematical foundation as it was shown to be converging to the exact solution, provided perfect numerical minimisation. However, there are currently no approximation rates available for the underlying DNN which reflect the efficiency in high dimensions. While the approximation theory for DNN is an active field, see, e.g., Elbrächter et al. (2021), Gühring et al. (2020) most results are only available for rectified linear units (ReLU) as the activation functions. Their lack of smoothness makes them unsuitable for the approximation of PDEs in the given framework. We therefore extend the approximation theory for neural networks with smooth activation functions and prove for the first time (to the best of our knowledge) an approximation rate which is independent of the dimensionality of the problem, thus lessening the curse of dimensionality. This result is shown by providing best-approximation rates for the L^2-norm which can be applied flexibly to the approximation over a variable state space, a time-state space as well as to the full parametric solutions.

In order to confirm this theoretical insight in a numerical example of practical relevance, we consider credit exposure calculations. Here financial institutions have to evaluate the risk they face in case counterparties of a derivative deal default. The common use of Monte-Carlo simulations to generate risk factors in exposure calculations requires frequent calls of the pricer. In particular for derivatives depending on several risk factors, this leads to unsustainably long run-times, rendering e.g. real-time nowcasting infeasible. As a prototypical example, we consider multi-asset option pricing and evaluate the expected exposure as well as the potential future exposure with up to ten underlying assets.

1.1 Literature review

The mathematical analysis of deep neural networks has significantly advanced in the recent years, see, e.g. Berner et al. (2021). In parts this is due to the increased attention of the topic thanks to impressive results of deep learning (LeCun et al., 2015) .

Applications in deep learning and DNN-based PDE solver share the need for results on the best-approximation property of neural networks. For an overview over current results (mainly based on ReLU networks) see Elbrächter et al. (2021); Gühring et al. (2020). Approximation results for ReLU networks are shown in Yarotsky (2017) and were subsequently improved, e.g. specifically for solutions of PDEs (Grohs et al., 2019; Beck et al., 2020; Elbrächter et al., 2021), or in Sobolev spaces which include the first derivative (Gühring et al., 2020). Errors in Sobolev spaces for similar activations functions, rectified cubic units, were considered in Abdeljawad and Grohs (2022). For Kolmogorov PDEs it has furthermore been shown that the solution in the state-space can be approximated without the curse of dimensionality, see for example (Jentzen et al., 2021; Grohs et al., 2018; Reisinger & Zhang, 2020). We note that in this article, we have chosen the more general approach of best-approximation rates, as the deep parametric PDE method can be used to solve non-Kolmogorov PDEs. Exponential convergence rates for analytic functions were shown in low dimensions (E & Wang, 2018) and in high dimensions (Opschoor et al., 2021). Also constructive approaches based on Chebyshev interpolation were investigated in Tang et al. (2019); Herrmann et al. (2022).

The deep parametric PDE method was first introduced in Glau and Wunderlich (2022) and is based on the deep Galerkin method (Sirignano & Spiliopoulos, 2018). For a detailed literature review, we refer the reader to these articles. Related ideas led to the development of physics-informed neural networks (Raissi et al., 2019; Karniadakis et al., 2021). The

interest for DNN-based methods is large, few recent examples are applications to Hamilton-Jacobi-Bellman equations (Grohs & Herrmann, 2021) and stochastic control in finance (Germain et al., 2021). In addition the current research into high-dimensional problems has inspired approaches beyond neural networks, see, e.g. Antonov and Piterbarg (2021).

Due to the second order derivative terms, some PDE solvers (including the deep parametric PDE method) require smooth activation functions. However, for smooth activation functions, there are less results available than for ReLU networks. For single-layer networks (Barron, 1993; Siegel & Xu, 2020) have shown approximation estimates in the L^2-norm which do not exceed those of standard Monte-Carlo methods. Expression rates for deep neural networks are investigated in Ohn and Kim (2019); Langer (2021), but these rates deteriorate in high-dimensions due to the remaining curse of dimensionality.

To the best of our knowledge, this article presents the first approximation rates for DNNs with smooth activation functions which are not deteriorating with the number of dimensions. The analysis is based on a sparse grid approach (Bungartz & Griebel, 2004; Garcke & Griebel, 2013). It is motivated by related results for ReLU activation functions (Montanelli & Du, 2019) .

While the approximative power is a major contribution to the numerical error, research into the remaining factors is highly important, too. Some gaps between theory and practice are demonstrated in Adcock and Dexter (2021). Approximation results which include the sampling error are considered, e.g., in Shin et al. (2020). First investigations into the convergence of the optimiser are also available, see, e.g. Jentzen and Riekert (2021) and the references therein.

For an overview on counterparty credit risk and exposure calculations, see Gregory (2010); Green (2015). Several articles have proposed techniques to speeding up its calculation. For low-dimensions, the dynamic Chebyshev methods introduced in Glau et al. (2019) has proven highly suited for exposure calculations (Glau et al., 2021). For higher dimensions some machine learning algorithms have been considered: a proof of concept for a supervised learning-based approach in Welack (2019), calculations based on Deep BSDE solver in Gnoatto et al. (2020), and based on the deep optimal stopping algorithm for Bermudan options in Andersson and Oosterlee (2021a, b). Besides DNN-based approaches for example sparse grid methods have recently been proposed for use in exposure calculations (Grzelak, 2021). In this article, we demonstrate the performance of the deep parametric PDE method for exposure calculations including high-dimensional pricing problems.

1.2 Structure of the article

In Sect. 2, we briefly recall the deep parametric PDE method. The theoretical analysis of the approximation through neural networks with smooth activation functions is presented in Sect. 3. Then in Sect. 4, we present the performance of the deep parametric PDE method in exposure calculations. Finally, we summarise and conclude the article in Sect. 5.

2 The deep parametric PDE method for option pricing

The deep parametric PDE method was recently introduced in Glau and Wunderlich (2022) as a neural network-based method to simultaneous compute the option price at all time, state and parameter value of interest. For convenience of the reader, we recall its main properties.

Expressing the option price in logarithmic asset variables, $u(\tau, x; \mu)$ denotes the fair price of an option at time to maturity τ for the asset prices $s_i = e^{x_i}$:

$$u(\tau, x; \mu) = \text{Price}(T - \tau, e^x; \mu),$$
$$\text{Price}(t, s; \mu) = e^{-r(T-t)}\mathbb{E}(G(S_T(\mu)) \mid S_t(\mu) = s),$$

with d underlyings $S_t(\mu) = (S_t^1(\mu), \ldots, S_t^d(\mu))$ and the physical time $t = T - \tau$. G denotes the payoff function at maturity and the assets S are modelled by a multivariate geometric Brownian motion. The parameter μ can describe model parameters as well as option parameters. In our setting the parameter vector μ contains the risk-free rate of return, volatilities and correlations, each with a smooth parameter dependency.

The option price solves the Black-Scholes PDE

$$\begin{aligned} \partial_\tau u(\tau, x; \mu) + \mathcal{L}_x^\mu u(\tau, x; \mu) &= 0, & (\tau, x) &\in \mathcal{Q} = (0, T) \times \Omega, \\ u(0, x; \mu) &= g(x), & x &\in \Omega, \\ u(\tau, x; \mu) &= u_\Sigma(t, x; \mu), & (\tau, x) &\in \Sigma = (0, T) \times \partial\Omega, \end{aligned}$$

with $\Omega = (x_{\min}, x_{\max})^d$, $g(x_1, \ldots, x_d) = G(e^{x_1}, \ldots, e^{x_d})$ and

$$\mathcal{L}_x^\mu u(\tau, x; \mu) = r(\mu)u(\tau, x; \mu) - \sum_{i=1}^{d} \left(r(\mu) - \frac{\sigma_i(\mu)^2}{2} \right) \partial_{x_i} u(\tau, x; \mu)$$
$$- \sum_{i,j=1}^{d} \frac{\rho_{ij}(\mu)\sigma_i(\mu)\sigma_j(\mu)}{2} \partial_{x_i x_j} u(\tau, x; \mu).$$

For a given function $u \colon \mathcal{Q} \times \mathcal{P} \to \mathbb{R}$ of sufficient smoothness, we define the loss based on the PDE's residuals:

$$\mathcal{J}(u) = \mathcal{J}_{\text{int}}(u) + \mathcal{J}_{\text{ic}}(u).$$

The interior residual is defined as

$$\mathcal{J}_{\text{int}}(u) = |\mathcal{Q} \times \mathcal{P}|^{-1} \int_{\mathcal{P}} \int_{\mathcal{Q}} \left(\partial_\tau u(\tau, x; \mu) + \mathcal{L}_x^\mu u(\tau, x; \mu) \right)^2 \, \mathrm{d}(\tau, x) \, \mathrm{d}\mu,$$

with $|\mathcal{Q} \times \mathcal{P}|$ the size of the domain, and the initial residual as

$$\mathcal{J}_{\text{ic}}(u) = |\Omega \times \mathcal{P}|^{-1} \int_{\mathcal{P}} \int_{\Omega} (u(0, x; \mu) - g(x))^2 \, \mathrm{d}x \, \mathrm{d}\mu.$$

The deep parametric PDE method minimises the residual over the space of deep neural networks of a given size:

$$u_{\text{DPDE}} = \arg\min_{u_{\text{DNN}}} \mathcal{J}(u_{\text{DNN}}).$$

The integrals are numerically evaluated by Monte-Carlo quadrature, which yields a similarity to mean squared error-residuals often used in machine learning:

$$\mathcal{J}_{\text{int}}(u) \approx \sum_{i=1}^{N} \left(\partial_\tau u\big(\tau^{(i)}, x^{(i)}; \mu^{(i)}\big) + \mathcal{L}_x^\mu u\big(\tau^{(i)}, x^{(i)}; \mu^{(i)}\big) - f\big(\tau^{(i)}, x^{(i)}; \mu^{(i)}\big) \right)^2 / N,$$
$$\mathcal{J}_{\text{ic}}(u) \approx \sum_{i=1}^{N} \left(u\big(0, \hat{x}^{(i)}; \hat{\mu}^{(i)}\big) - g\big(\hat{x}^{(i)}; \hat{\mu}^{(i)}\big) \right)^2 / N,$$

where $(\tau^{(i)}, x^{(i)}, \mu^{(i)}) \in \mathcal{Q} \times \mathcal{P}$ and $(\hat{x}^{(i)}, \hat{\mu}^{(i)}) \in \Omega \times \mathcal{P}$ for $i = 1, \dots, N$ are chosen randomly with a uniform distribution. In our experiment, we choose $N = 10,000$ as we observed less accurate approximations with smaller values of N.

In (Glau & Wunderlich, 2022, Theorem 1), it was shown that for sufficiently large networks, the L^2-error of the deep parametric PDE method can be arbitrarily small. However, no estimates on the required network size were given. In this article, we investigate this question in more details and provide new results on the approximation of functions by neural networks with smooth activation functions. In particular, we provide (to the best of our knowledge) the first approximation rates which do not deteriorate for higher dimensions, thus reducing the curse of dimensionality. We then discuss the consequences of these results to the deep parametric PDE method in more details.

3 Function approximation by neural networks

We consider dense neural networks with d input nodes, a single output node and the hyperbolic tangent as the activation function. Results for several output nodes can be shown in an analogue way.

In this chapter, we are going to improve the available approximation rates for a smooth activation function, based on available results for ReLU networks. There are two alternative pathways: based on a given ReLU approximation or by a direct construction of a network with smooth activation functions. For a proof based on a given ReLU approximation, we would first construct an auxiliary ReLU network which can then be approximated by a network with a smooth activation function. This was shown, for example in Boulle et al. (2020); Telgarsky (2017) using rational activation functions. For the direct construction, we construct the network with a smooth activation function without the auxiliary network, but in a similar way to the construction of the ReLU network. This way is technically more challenging, but results in sharper bounds. For this reason, we are going to directly construct the neural network with a smooth activation function. In addition to sharper bounds, this way also allows us to provide bounds of the magnitude of the parameters.

3.1 Deep neural networks

We consider neural networks with input $h^0 \in \mathbb{R}^n$, output dimension m, width w, depth L and activation function $\sigma : \mathbb{R}^w \to \mathbb{R}^w$. Here σ is the component-wise evaluation of the hyperbolic tangent. The mapping from one layer to the next is an affine transformation combined with the activation function:

$$h^l = \sigma(W^l h^{l-1} + b^l) \in \mathbb{R}^w.$$

The output is given by an affine transformation of the last hidden layer:

$$W^{L+1} h^L + b^{L+1} \in \mathbb{R}^m.$$

Here the weights are $W^1 \in \mathbb{R}^{w \times n}$, $W^l \in \mathbb{R}^{w \times w}$, $l = 2, \dots, L$, $W^{L+1} \in \mathbb{R}^{m \times w}$ and the biases are $b^l \in \mathbb{R}^w$, $l = 1, \dots, L$ and $b^{L+1} \in \mathbb{R}^m$. Their values will be set by a numerical optimisation.

We are particularly interested in deep neural networks and their ability of function approximation. While the exact threshold between deep and shallow networks is rather arbitrary, we consider networks with $L \geq 4$ as deep neural networks.

We denote by $\Theta(L, w, n, m)$ the set of all weights corresponding to a neural network of depth L and width w mapping $\mathbb{R}^n$ to $\mathbb{R}^m$:

$$\begin{aligned}\Theta(L, w, n, m) = \{ \Big(&W^1, \ldots, W^{L+1}, b^1, \ldots b^{L+1}\Big) : \\ &W^1 \in \mathbb{R}^{w\times n}, W^l \in \mathbb{R}^{w\times w}, l = 2, \ldots, L, W^{L+1} \in \mathbb{R}^{m\times w}, \\ &b^l \in \mathbb{R}^w, l = 1, \ldots, L, b^{L+1} \in \mathbb{R}^m.\}\end{aligned}$$

Note that we will frequently drop the arguments n, m when they are clear from the context. Typically we will consider an input dimension $n = d$ and an output dimension $m = 1$.

Next to the dimension of the network, also the magnitude of the weights and biases is important. Too large parameters can cause numerical instabilities and may lead to slow convergence by an iterative solver. Therefore, we will also show bounds for the maximal parameter value of $\theta \in \Theta(L, w, n, m)$ as

$$|\theta|_\infty = \max \left\{ |W^l_{i,j}|, |b^l_i| \colon i, j, l \right\}.$$

The evaluation of the neural network is denoted as $N(\cdot \mid \theta)$:

$$N(x \mid \theta) = W^{L+1}\sigma\Big(W^L\sigma\Big(\cdots\sigma\big(W^2\sigma(W^1x + b^1) + b^2\big) + \cdots\Big) + b^L\Big) + b^{L+1}.$$

Lemma 1 *We can manipulate neural networks in the following ways:*

(a) For $N \geq n$, we can trivially embed $\Theta(L, w, n, m)$ in $\Theta(L, w, N, m)$. I.e. for an injective index mapping $\pi \colon \{1, \ldots, n\} \to \{1, \ldots, N\}$, we can extend a neural network $\theta \in \Theta(L, w, n, m)$ to a neural network $\tilde{\theta} \in \Theta(L, w, N, m)$, such that

$$N(x \mid \theta) = N(\tilde{x} \mid \tilde{\theta}),$$

for all $x \in \mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^N$ with $x_i = \tilde{x}_{\pi(i)}$, $i = 1, \ldots, n$. In particular

$$\left|\tilde{\theta}\right|_\infty = |\theta|_\infty .$$

(b) Two neural networks can be evaluated in parallel. There is an operator $\biguplus \colon \Theta(L, w_1, n, m_1) \times \Theta(L, w_2, n, m_2) \to \Theta(L, w_1 + w_2, n, m_1 + m_2)$ producing the evaluation of two networks of at the same time, i.e. such that for $\theta_\mathrm{u} = \theta_1 \uplus \theta_2$:

$$N(x \mid \theta_\mathrm{u}) = \begin{pmatrix} N(x \mid \theta_1) \\ N(x \mid \theta_2) \end{pmatrix}$$

for all $x \in \mathbb{R}^n$. It furthermore holds

$$|\theta_\mathrm{u}|_\infty = \max\{|\theta_1|_\infty , |\theta_2|_\infty\}.$$

(c) There exists an addition operator $\bigoplus \colon \Theta(L, w_1, n, m) \times \Theta(L, w_2, n, m) \to \Theta(L, w_1 + w_2, n, m)$ producing the sum of two networks with the same output size, i.e. such that for $\theta_+ = \theta_1 \oplus \theta_2$:

$$N(x \mid \theta_+) = N(x \mid \theta_1) + N(x \mid \theta_2)$$

for all $x \in \mathbb{R}^d$. It furthermore holds

$$|\theta_+|_\infty \leq \max\{|\theta_1|_\infty , |\theta_2|_\infty , \left|b^{L+1}_1\right|_\infty + \left|b^{L+1}_2\right|_\infty\},$$

where b_1^{L+1} and b_2^{L+1} are the final biases of θ_1 and θ_2, respectively. The input weights and biases W^1, b^1 of θ_+ can be bound more strictly by $\max\{|W_1^1|_\infty, |b_1^1|_\infty, |W_1^2|_\infty, |b_1^2|_\infty\}$, *where W_1^1, b_1^1 and W_2^1, b_2^1 are the input weights and biases of θ_1 and θ_2, respectively.*

(d) There exists a composition operator $\bigodot\colon \Theta(L_1, w_1, m, l) \times \Theta(L_2, w_2, n, m) \to \Theta(L_1 + L_2, \max\{w_1, w_2\}, n, l)$ *producing the composition of two networks, i.e. such that for* $\theta_\odot = \theta_1 \odot \theta_2$*:*

$$N(x \mid \theta_\odot) = N(N(x \mid \theta_2) \mid \theta_1)$$

for all $x \in \mathbb{R}^d$. We can estimate the new weights:

$$|\theta_\odot|_\infty \le \max\left\{|\theta_1|_\infty, |\theta_2|_\infty, |W_2^1|_\infty \left|W_1^{L_1+1}\right|_\infty, |W_2^1|_\infty \left|b_1^{L_1+1}\right|_\infty + |b_2^1|_\infty\right\},$$

where $W_1^{L_1+1}$ and $b_1^{L_1+1}$ are the final layer weights and biases of θ_1 and W_2^1, b_2^1 are the first layer weights and biases of θ_2.

The weights of the first and final layer of $\theta_\odot$ are equal to the weights of the first layer of θ_1 and the final layer of θ_2, respectively:

$$W^1 = W_1^1, \quad b^1 = b_1^1, \quad W^{L_1+L_2+1} = W_2^{L_2+1}, \quad b^{L_1+L_2+1} = b_2^{L_2+1}.$$

Proof The four statements are shown by constructing the respective networks.

Part a Adding columns of zero and permuting all columns according to π allows us to construct weights $\widetilde{W}^1 \in \mathbb{R}^{w \times N}$ such that

$$W^1 x = \widetilde{W}^1 \widetilde{x}$$

for $x \in \mathbb{R}^n, \widetilde{x} \in \mathbb{R}^N$ with $x_i = \tilde{x}_{\pi(i)}, i = 1, \ldots, n$. Leaving all remaining weights and biases equal, yields the desired network.

Part b Let $\theta_1 \in \Theta(L, w_1, n, m_1)$ with weights W_1^l and biases b_1^l, as well as $\theta_2 \in \Theta(L, w_2, n, m_2)$ with weights W_2^l and biases b_2^l be given. We construct the new weights block-wise as

$$W^1 = \begin{pmatrix} W_1^1 \\ W_2^1 \end{pmatrix} \in \mathbb{R}^{w_1+w_2 \times n},$$

$$b^l = \begin{pmatrix} b_1^l \\ b_2^l \end{pmatrix} \in \mathbb{R}^{w_1+w_2} \text{ for } l = 2, \ldots, L,$$

$$W^l = \begin{pmatrix} W_1^l & 0 \\ 0 & W_2^l \end{pmatrix} \in \mathbb{R}^{w_1+w_2 \times w_1+w_2} \text{ for } l = 2, \ldots, L,$$

$$b^{L+1} = \begin{pmatrix} b_1^{L+1} \\ b_2^{L+1} \end{pmatrix} \in \mathbb{R}^{m_1+m_2},$$

$$W^{L+1} = \begin{pmatrix} W_1^{L+1} & 0 \\ 0 & W_2^{L+1} \end{pmatrix} \in \mathbb{R}^{m_1+m_2 \times w_1+w_2}.$$

Clearly the absolute value of the new weights and biases are bounded by the maximal value in the networks θ_1 and θ_2.

For this new network we can show that the first hidden layer indeed combines the values of both networks as the activation function is applied element-wise and

$$W^1 h^0 + b^1 = \begin{pmatrix} W_1^1 h^0 + b_1^1 \\ W_2^1 h^0 + b_2^1 \end{pmatrix}.$$

Inductively the same argument shows the final result.
Part c We follow the construction from part (b), but change the final layer to obtain the sum of the output values:

$$W^{L+1} = \begin{pmatrix} W_1^{L+1} & W_2^{L+1} \end{pmatrix} \in \mathbb{R}^{m \times w_1 + w_2},$$
$$b^{L+1} = b_1^{L+1} + b_2^{L+1} \in \mathbb{R}^m.$$

Clearly the newly constructed weights can be bounded by the maximum of the original network weights and the sum of the output biases.

We consider the final hidden layers of both networks $h_1^L \in \mathbb{R}^{w_1}$ and $h_2^L \in \mathbb{R}^{w_2}$ and for the newly constructed network:

$$h^L = \begin{pmatrix} h_1^L \\ h_2^L \end{pmatrix} \in \mathbb{R}^{w+1}.$$

Then the output of the network is given as

$$\begin{aligned} W^{L+1}h^L + b^{L+1} &= W_1^{L+1}h_1^L + b_1^{L+1} + W_2^{L+1}h_2^L + b_2^{L+1} \\ &= N(x \mid \theta_1) + N(x \mid \theta_2). \end{aligned}$$

Part d We consider the case where $w_1 = w_2 = w$ as similar to the construction in (a) the networks can trivially be extended. The first L_1 and last L_2 weights can be set equal to the networks θ_1 and θ_2 respectively:

$$\begin{aligned} W^l &= W_1^l \text{ for } l = 1, \ldots, L_1, \\ b^l &= b_1^l \text{ for } l = 1, \ldots, L_1, \\ W^{L_1+l} &= W_2^l \text{ for } l = 2, \ldots, L_2 + 1, \\ b^{L_1+l} &= b_2^l \text{ for } l = 2, \ldots, L_2 + 1, \end{aligned}$$

and it remains to construct the interface of both networks W^{L_1+1}, b^{L_2+1}:

$$W^{L_1+1} = W_2^1 W_1^{L_1+1} \in \mathbb{R}^{w \times w}$$
$$b^{L_1+1} = W_2^1 b_1^{L_1+1} + b_2^1 \in \mathbb{R}^w.$$

Note that as $W_1^{L_1+1} \in \mathbb{R}^{m \times w}$, $b_1^{L_1+1} \in \mathbb{R}^m$ and $W_2^1 \in \mathbb{R}^{w \times m}$, the products are well-defined. The weights are clearly bounded by the maximum of the weights of θ_1 and θ_2 and the products at the interface.

The hidden layer h^{L_1} coincides with the final hidden layer of θ_1, thus

$$N(x \mid \theta_1) = W_1^{L_1+1}h^{L_1} + b_1^{L_1+1}.$$

Having $N(x \mid \theta_1)$ as the input to the network θ_2 yields the first hidden layer as

$$\begin{aligned} h_2^1 &= \sigma\left(W_2^1 N(x \mid \theta_1) + b_2^1\right) \\ &= \sigma\left(W_2^1 (W_1^{L_1+1}h^{L_1} + b_1^{L_1+1}) + b_2^1\right) \\ &= \sigma\left(W^{L_1+1}h^{L_1} + b^{L_1+1}\right) = h^{L_1+1}. \end{aligned}$$

This means that the value of the hidden layer $L_1 + 1$ equals the first layer of θ_2 with the input $N(x \mid \theta_1)$. As the remaining weights are equal, the output of θ equals the composition of the two networks θ_1 and θ_2. □

3.2 Sparse grid theory

We show approximation results for neural networks in two steps: first, we approximate a function by a sparse grid function; then, the sparse grid function is approximated by a neural network. Therefore, we briefly recall basic results from sparse grid theory. For further information, we refer the reader to Bungartz and Griebel (2004); Garcke and Griebel (2013).

We consider a tensor-product space of $W^{2,\infty}([0,1])$ functions:

$$X_0^{\infty,2} = \Big\{ f \in L^\infty\left([0,1]^d\right) : D_\alpha f \in L^\infty\left([0,1]^d\right) \text{ for } \alpha \in \mathbb{N}^d_{\geq 0},\ |\alpha|_\infty \leq 2,\ u|_{\partial[0,1]^d} = 0 \Big\}$$

with the semi-norm

$$|f|_{2,\infty} = \|D_{(2,\ldots,2)} f\|_{L^2(\Omega)}.$$

Here D_α refers to the derivative with respect to a multi-index $\alpha \in \mathbb{N}^d_{\geq 0}$, i.e. for each $i = 1, \ldots, d$ the derivative with respect to the variable x_i is taken α_i-times.

Sparse grids are based on a tensor grid with hierarchical refinements in each direction. In contrast to a full grid, only a reduced number of combinations are considered. For a multi-index $\mathbf{l} = (l_1, \ldots, l_d) \in \mathbb{N}^d_{>0}$, we consider the grid created by the points $\mathbf{x}_{\mathbf{l},\mathbf{i}} = (i_1 h_{l_1}, \ldots, i_d h_{l_d}) \in [0,1]^d$ for $\mathbf{i} \in \mathcal{I}_\mathbf{l}$, where $h_l = 2^{-l}$ and

$$\mathcal{I}_\mathbf{l} = \left\{ \mathbf{i} = (i_1, \ldots, i_d) \colon i_j = 2k_j - 1, k_j = 1, \ldots, 2^{l_j - 1} \right\}.$$

The corresponding basis functions $\Psi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{j=1}^d \psi_{l_j,i_j}(x_j)$ are created as a tensor product of the univariate basis functions $\psi_{l,i}(x) = (h_l - |x - x_{l,i}|)_+$ with $x_{l,i} = i h_l$. We also define the multivariate mesh-size $h_\mathbf{l} = 2^{-|\mathbf{l}|_1}$ and note that $|\mathcal{I}_\mathbf{l}| = 2^{|\mathbf{l}|_1 - d}$.

On refinement level n, we consider the basis functions of all levels $\mathbf{l}$ with $|\mathbf{l}| \leq n + d - 1$:

$$\left\{ \Psi_{\mathbf{l},\mathbf{i}} \colon\ \mathbf{i} \in \mathcal{I}_\mathbf{l},\ |\mathbf{l}| \leq n + d - 1 \right\}.$$

Given coefficients $v_{\mathbf{l},\mathbf{i}}$, this yields a sparse grid function

$$f_n = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}} \Psi_{\mathbf{l},\mathbf{i}}.$$

Note that often a different scaling of the basis functions is used:

$$f_n = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} \widetilde{v}_{\mathbf{l},\mathbf{i}} \varphi_{\mathbf{l},\mathbf{i}},$$

where $\varphi_{\mathbf{l},\mathbf{i}} = \psi_{\mathbf{l},\mathbf{i}} / h_\mathbf{l}$ and $\widetilde{v}_\mathbf{l} = h_\mathbf{l} v_{\mathbf{l},\mathbf{i}}$. This is the scaling used for example in Bungartz and Griebel (2004); Garcke and Griebel (2013).

For $f \in X_0^{\infty,2}$ with $|f|_{2,\infty} = 1$, Bungartz and Griebel (2004) construct coefficients $v_{\mathbf{l},\mathbf{i}}$, such that

$$|f(x) - f_n(x)| \leq \varepsilon,$$

for the number of degrees of freedom $M = \mathcal{O}\left(\varepsilon^{-1/2} |\log_2 \varepsilon|^{3/2(d-1)}\right)$, see (Bungartz & Griebel, 2004, Lemma 3.13). In addition (Bungartz & Griebel, Bungartz and Griebel (2004), Lemma 3.3) provides a bound of the coefficients:

$$|v_{\mathbf{l},\mathbf{i}}| = h_\mathbf{l}^{-1} |\widetilde{v}_{\mathbf{l},\mathbf{i}}| \leq h_\mathbf{l}^{-1} 2^{-d} 2^{-2|\mathbf{l}|_1} = 2^{-d-|\mathbf{l}|_1}, \tag{1}$$

where we used $h_\mathbf{l} = 2^{-|\mathbf{l}|_1}$.

3.3 Improved approximation result using sparse grids

We show approximation results for neural networks in two stages. First, we show that we can approximate each basis function $\Psi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ by a neural network to the desired accuracy. Then, we replace the basis functions in the sparse grid formulation by the constructed neural networks. We show that this network approximates the function up to the accuracy ε and that the size of the network grows at a rate that does not depend on the dimension. This reduces the curse of dimensionality as the influence of the dimension is confined to the constant of the estimate.

To approximate the basis functions, we first need to approximate some elementary functions: the identity, a multiplication and the absolute value. Recall that throughout this article, we consider the hyperbolic tangent as the activation function.

Lemma 2 *For $a \geq 1$, $1 > \delta > 0$, we can construct the following neural networks with $\sigma = \tanh$. The generic constant c is independent of a and δ.*

(a) There is $\theta_{\mathrm{id},a} \in \Theta(1,1)$ with $|\theta_{\mathrm{id,a}}|_\infty \leq \delta^{-1/2} a^{3/2}$, such that

$$\sup_{x\in[-a,a]} |x - N(x \mid \theta_{\mathrm{id},a})| \leq \delta/3.$$

(b) There is $\theta_{\times,a} \in \Theta(1,4)$ with $|\theta_{\times,a}|_\infty \leq \frac{9\sqrt{3}}{4} a^6 \delta^{-2}$, such that

$$\sup_{x\in[-a,a]} |x_1 x_2 - N((x_1,x_2) \mid \theta_{\times,a})| \leq \delta.$$

(c) There is $\theta_{\mathrm{abs,a}} \in \Theta(2,4)$ with $|\theta_{\mathrm{abs,a}}|_\infty \leq c a^3 \delta^{-2}$, such that

$$\sup_{x\in[-a,a]} \big||x| - N(x \mid \theta_{\mathrm{abs,a}})\big| \leq \delta.$$

Proof The proof is partially based on Ohn and Kim (2019), but shows improved results and overcomes some technical difficulties.[1] In the following, let $1 > \delta > 0$ and $a \geq 1$.

Part a To construct the identity network with a single node, we consider for $t > 0$

$$N(x \mid \theta) = \frac{1}{t}\sigma(tx).$$

We use the Taylor expansion with Lagrange remainder to represent $\sigma(tx)$ as

$$\sigma(tx) = \sigma(0) + \sigma'(0)tx + \frac{1}{2}\sigma''(0)(tx)^2 + \frac{1}{6}\sigma'''(\xi)(tx)^3 = tx + \frac{1}{6}\sigma'''(\xi)(tx)^3,$$

for some $\xi \in \mathbb{R}$.

Using $\sigma(0) = 1$ and $|\sigma'''(\xi)| \leq 2$ for all $\xi \in \mathbb{R}$, we have for $|x| \leq a$

$$\begin{aligned} |N(x \mid \theta) - x| &= \left| \frac{1}{t}\left(tx + \frac{1}{6}\sigma'''(\xi)(tx)^3 \right) - x \right| \\ &= \left| \frac{\sigma'''(\xi) t^2 x^3}{6} \right| \leq \frac{a^3}{3} t^2 \end{aligned}$$

We choose $t_a = \delta^{1/2} a^{-3/2}$ and have $\theta_{\mathrm{id},a} \in \Theta(1,1)$ with

$$\sup_{x\in[-a,a]} |N(x|\theta_{\mathrm{id},a}) - x| \leq \delta/3.$$

[1] In the proof of (Ohn & Kim, 2019, Lemma A3d), the Taylor series of the exponential function is considered, instead of the Taylor series of the square root function.

Using $t_a < 1$, we also have

$$|\theta_{\mathrm{id},a}|_\infty = \max\left\{t_a, t_a^{-1}\right\} = \delta^{-1/2}a^{3/2}.$$

Part b.1 Using the polarisation identity, we can evaluate a product based on squares: $xy = (\frac{x+y}{2})^2 - (\frac{x-y}{2})^2$. The square function can be approximated in a similar way to the identity function.

For $t > 0$ and $x_0 = \log(2+\sqrt{3})/2 \approx 0.66$[2], we consider the network θ as

$$N(x|\theta) = \frac{1}{t^2\sigma''(x_0)}\sigma(x_0 + tx) + \frac{1}{t^2\sigma''(x_0)}\sigma(x_0 - tx) - \frac{2\sigma(x_0)}{t^2\sigma''(x_0)}.$$

As before, we can use a Taylor expansion at point x_0 to estimate (with $\xi, \xi' \in \mathbb{R}$)

$$|N(x|\theta) - x^2| = \left|\frac{1}{6\sigma''(x_0)}\sigma'''(\xi)tx^3 - \frac{1}{6\sigma''(x_0)}\sigma'''(\xi')tx^3\right| \leq \frac{\|\sigma'''\|_\infty}{3|\sigma''(x_0)|}a^3t.$$

We choose $t_a = \frac{3}{2}\delta a^{-3}\frac{|\sigma''(x_0)|}{\|\sigma'''\|_\infty} = \frac{\sqrt{3}}{3}\delta a^{-3}$ and have $\theta_{2,a} \in \Theta(1,2)$ with

$$|N(x \mid \theta_{2,a}) - x^2| \leq \delta/2.$$

We have used that, $|\sigma''(x_0)| = 4\sqrt{3}/9$ and $\|\sigma'''\|_\infty = 2$.

Part b.2 Duplicating $\theta_{2,a}$ for the inputs $\frac{x_1+x_2}{2}$ and $\frac{x_1-x_2}{2}$ yields according to Lemma 1(c) a neural network $\theta_{\times,a} \in \Theta(1,4)$ as

$$\begin{aligned}
N((x_1,x_2) \mid \theta_{\times,a}) &= N\left(\frac{x_1+x_2}{2} \mid \theta_{2,a}\right) - N\left(\frac{x_1-x_2}{2} \mid \theta_{2,a}\right) \\
&= \frac{1}{t_a^2\sigma''(x_0)}\sigma(x_0 + t_a/2x_1 + t_a/2x_2) + \frac{1}{t_a^2\sigma''(x_0)}\sigma(x_0 - t_a/2x_1 - t_a/2x_2) \\
&- \frac{1}{t_a^2\sigma''(x_0)}\sigma(x_0 + t_a/2x_1 - t_a/2x_2) - \frac{1}{t_a^2\sigma''(x_0)}\sigma(x_0 - t_a/2x_1 + t_a/2x_2). \qquad (2)
\end{aligned}$$

We have

$$|\theta_{\times,a}|_\infty = \max\left\{\frac{1}{t_a^2|\sigma''(x_0)|}, x_0, t_a/2\right\} = \frac{9\sqrt{3}}{4}\delta^{-2}a^6 \approx 3.90\delta^{-2}a^6,$$

For $(x_1,x_2) \in [-a,a]^2$, we note that $\frac{x_1+x_2}{2}, \frac{x_1-x_2}{2} \in [-a,a]$ and have

$$\begin{aligned}
&|N(x_1,x_2) \mid \theta_{\times,a}) - x_1x_2| \\
&\leq \left|N\left(\frac{x_1+x_2}{2} \mid \theta_{2,a}\right) - \left(\frac{x_1+x_2}{2}\right)^2\right| + \left|N\left(\frac{x_1-x_2}{2} \mid \theta_{2,a}\right) - \left(\frac{x_1-x_2}{2}\right)^2\right| \leq \delta,
\end{aligned}$$

which concludes the proof of part b.

Part c We first show the statements for $a = 1$, where we omit the index a. Then a scaling argument yields the results.

We use the representation $|x| = \mathrm{sign}(x) \cdot x$. First, we approximate the sign-function and then multiply it by the identity to obtain the absolute value.

Set $\theta_{\mathrm{sign}} \in \Theta(1,1)$ as $N(x|\theta_{\mathrm{sign}}) = \sigma(\alpha x)$ with $\alpha = \delta^{-1}\log(2/\delta - 1)/2$ for $0 < \delta < 1$. Taking into account point symmetry and monotonicity of the hyperbolic tangent, from $N(\delta \mid$

[2] While any choice of $x_0 \neq 0$ is possible, this particular choice optimises the resulting constants.

$\theta_{\text{sign}}) = \tanh(\log(2/\delta - 1)/2) = 1 - \delta$ it follows that

$$\sup_{x \notin (-\delta, \delta)} |N(x|\theta_{\text{sign}}) - \text{sign}(x)| \leq \delta.$$

Now we combine $\theta_{\text{sign}} \in \Theta(1, 1)$, $\theta_{\text{id}} \in \Theta(1, 1)$ and $\theta_{\times, 1+\delta/3} \in \Theta(1, 4)$ to $\theta_{\text{abs}} \in \Theta(2, 4)$ as

$$\begin{aligned} N(x|\theta_{\text{abs}}) &= N((N(x|\theta_{\text{id}}), N(x|\theta_{\text{sign}})) \mid \theta_{\times, 1+\delta/3}). \\ &= \sum_{s_1, s_2 \in \{-1, 1\}} \frac{s_1 s_2}{t_{1+\delta/3}^2 \sigma''(x_0)} \sigma\left(x_0 + s_1 \frac{t_{1+\delta/3}}{2\delta^{1/2}} \sigma(\delta^{1/2} x) + s_2 \frac{t_{1+\delta/3}}{2} \sigma(\alpha x)\right), \end{aligned}$$

with $t_{1+\delta/3} = \frac{\sqrt{3}\delta}{3(1+\delta/3)^3}$.

For $x \in [-1, 1]$ we have

$$\begin{aligned} \left|N(x \mid \theta_{\text{abs}}) - |x|\right| \leq\ & |N((N(x \mid \theta_{\text{id}}), N(x \mid \theta_{\text{sign}})) \mid \theta_{\times, 1+\delta/3}) - N(x \mid \theta_{\text{id}}) \cdot N(x \mid \theta_{\text{sign}})| \\ & + |N(x \mid \theta_{\text{id}}) \cdot N(x \mid \theta_{\text{sign}}) - \text{sign}(x) \cdot x|. \end{aligned}$$

The first term can directly be estimated by δ as $N(x|\theta_{\text{id}}) \in [-1 - \delta/3, 1 + \delta/3]$ and $N(x \mid \theta_{\text{sign}}) \in (-1, 1)$.

Estimating the second term, we need to consider the cases $x \in [-\delta, \delta]$ and $x \notin [-\delta, \delta]$ separately. For $\delta < |x| \leq 1$ we have

$$\begin{aligned} |N(x \mid \theta_{\text{id}}) N(x \mid \theta_{\text{sign}}) - \text{sign}(x) \cdot x| \leq\ & |(N(x|\theta_{\text{id}}) - x)(N(x|\theta_{\text{sign}}) - \text{sign}(x))| \\ & + |x(N(x|\theta_{\text{sign}}) - \text{sign}(x))| \\ & + |\text{sign}(x)(N(x \mid \theta_{\text{id}}) - x)| \leq 4/3\delta + \delta^2/3 < 2\delta. \end{aligned}$$

For $x \in [-\delta, \delta]$, we get with $|N(x|\theta_{\text{sign}})| < 1$:

$$\left|N(x \mid \theta_{\text{id}}) \cdot N(x \mid \theta_{\text{sign}}) - |x|\right| \leq |N(x \mid \theta_{\text{id}})| \cdot |N(x|\theta_{\text{sign}})| + |x| \leq 7/3\delta < 3\delta.$$

In conclusion, we have for all $x \in [-1, 1]$:

$$\left|N(x \mid \theta_{\text{id}}) N(x \mid \theta_{\text{sign}}) - |x|\right| \leq 3\delta,$$

with

$$|\theta_{\text{abs}}| \leq \max\left\{\frac{9\sqrt{3}}{4}\delta^{-2}(1 + \delta/3)^6, \delta^{-1}\log(2/\delta - 1)/2\right\} \leq c\,\delta^{-2}.$$

With a scaling argument, we can extend the result to include $x \in [-a, a]$:

$$N(x|\theta_{\text{abs},a}) = aN(a^{-1}x \mid \theta_{\text{abs}})$$

satisfies

$$\sup_{x \in [-a,a]} \left|N(x \mid \theta_{\text{abs},a}) - |x|\right| \leq 3\delta a \quad \text{and} \quad |\theta_{\text{abs}}| \leq ca\delta^{-2}.$$

□

These three basic approximations are sufficient to construct a simple univariate hat function in the following lemma. Using the multiplication operator, we will later extend the result to multivariate basis functions.

Lemma 3 *For $1 > \delta > 0$, $\sigma = \tanh$, $x_0 \in [0, 1]$ and $1 > h > 0$, it holds with c independent of δ, x_0, h_0 that there is $\theta_{h,x_0} \in \Theta(4, 5)$, such that*

$$\sup_{x\in[0,1]} |\psi_{h,x_0}(x) - N(x \mid \theta_{h,x_0})| \leq \delta,$$

with $|\theta_{h,x_0}|_\infty \leq c\delta^{-2}$, where $\psi_{h,x_0}(x) = (h - |x - x_0|)_+$.

Proof We first approximate the rectified linear unit $(x)_+ = (x + |x|)/2$ up to an accuracy of δ by a neural network θ_{relu}:

$$N(x \mid \theta_{\text{relu}}) = 0.5 \left(N(N(x \mid \theta_{\text{id}}) \mid \theta_{\text{id},1+\delta/3}) + N(x \mid \theta_{\text{abs}})\right).$$

To add the networks approximating x and $|x|$ using Lemma 1(c), both networks need to have the same depth. With $\theta_{\text{id}} \in \Theta(1, 1)$ and $\theta_{\text{abs}} \in \Theta(2, 4)$, we compose the identity network with another identity network using Lemma 1(d). With Lemmas 1 and 2, we then have $\theta_{\text{relu}} \in \Theta(2, 5)$ and $|\theta_{\text{relu}}|_\infty \leq c\delta^{-2}$.

The error is given by

$$|N(x \mid \theta_{\text{relu}}) - (x)_+| \leq |N(N(x \mid \theta_{\text{id}}) \mid \theta_{\text{id},1+\delta/3}) - x|/2 + |N(x \mid \theta_{\text{abs}}) - |x||/2.$$

We start with the first term and show using Lemma 2 (a) for any $x \in [-1, 1]$

$$\begin{aligned} |N(N(x \mid \theta_{\text{id}}) \mid \theta_{\text{id},1+\delta/3}) - x| &\leq |N(x \mid \theta_{\text{id}}) - x| \\ &\quad + |N(N(x \mid \theta_{\text{id}}) \mid \theta_{\text{id},1+\delta/3}) - N(x \mid \theta_{\text{id}})| \\ &\leq \delta/3 + \delta/3 < \delta, \end{aligned}$$

as $N(x \mid \theta_{\text{id}}) \in [-1 - \delta/3, 1 + \delta/3]$. The second term is directly approximated by $\delta/2$ using Lemma 2 (c), which yields

$$|N(x \mid \theta_{\text{relu}}) - (x)_+| \leq \delta$$

With this approximated rectified linear unit, we can finally approximate univariate basis functions

$$\psi_{h,x_0}(x) = (h - |x - x_0|)_+ = (1 + \delta) \left(\frac{h - |x - x_0|}{1 + \delta}\right)_+$$

by θ_{h,x_0} as

$$N(x \mid \theta_{h,x_0}) = (1 + \delta) N \left(\frac{h - N(x - x_0 \mid \theta_{\text{abs}})}{1 + \delta} \mid \theta_{\text{relu}}\right).$$

The construction is a composition of the networks $\theta_{\text{relu}} \in \Theta(2, 5)$ with $\theta_{\text{abs}} \in \Theta(2, 4)$, which yields $\theta_{h,x_0} \in \Theta(4, 5)$ by Lemma 1(c).

For $x \in [0, 1]$, we have $x - x_0 \in [-1, 1]$ and $h - N(x - x_0 \mid \theta_{\text{abs}}) \in [-1 - \delta, 1 + \delta]$. Then we have

$$\begin{aligned} &|\psi_{h,x_0}(x) - N(x \mid \theta_{h,x_0})| \\ &\quad \leq \left| (h - |x - x_0|)_+ - (h - N(x - x_0 \mid \theta_{\text{abs}}))_+ \right| \\ &\qquad + (1 + \delta) \left| \left(\frac{h - N(x - x_0 \mid \theta_{\text{abs}})}{1 + \delta}\right)_+ - N \left(\frac{h - N(x - x_0 \mid \theta_{\text{abs}})}{1 + \delta} \mid \theta_{\text{relu}}\right) \right|. \end{aligned}$$

Both terms can be estimated easily as $|(x)_+ - (y)_+| \leq |x - y|$:

$$\left| (h - |x - x_0|)_+ - (h - N(x - x_0|\theta_{\text{abs}}))_+ \right| \leq \left||x - x_0| - N(x - x_0 \mid \theta_{\text{abs}})\right| \leq \delta,$$

using Lemma 2(c) and

$$(1+\delta)\left|\left(\frac{h-N(x-x_0 \mid \theta_{\text{abs}})}{1+\delta}\right)_+ - N\left(\frac{h-N(x-x_0 \mid \theta_{\text{abs}})}{1+\delta} \mid \theta_{\text{relu}}\right)\right| \leq 2\delta.$$

All coefficients still fulfil $\left|\theta_{h,x_0}\right|_\infty \leq c\delta^{-2}$, which finishes the proof. □

Successive applications of the multiplication operator yield an approximation of the multivariate basis function. As in the previous lemmas, we are interested in the network's width, depth and parameter norm, which is required for a specific accuracy.

Lemma 4 *For $d \in \mathbb{N}_{>0}$, $\mathbf{h} = (h_1, \ldots, h_d) \in [0,1]^d$ and $\mathbf{x}^0 = (x_1^0, \ldots, x_d^0) \in [0,1]^d$, let*

$$\Psi_{\mathbf{h},\mathbf{x}^0}(\mathbf{x}) = \prod_{i=1}^{d} \psi_{h_i,x_i^0}(x_i).$$

For $1 > \varepsilon > 0$ and $\sigma = \tanh$, there exists a neural network $\theta_{\mathbf{h},\mathbf{x}^0} \in \Theta(4 + \lceil \log_2 d \rceil, 5d)$ such that

$$\sup_{\mathbf{x}\in[0,1]^d} \left|\Psi_{\mathbf{h},\mathbf{x}^0}(\mathbf{x}) - N(\mathbf{x} \mid \theta_{\mathbf{h},\mathbf{x}^0})\right| \leq \varepsilon,$$

and

$$|\theta_{\mathbf{h},\mathbf{x}^0}|_\infty \leq c d^{2\log_2 3} \varepsilon^{-2} \leq c d^4 \varepsilon^{-2}.$$

Proof The statement is similar to (Rolnick & Tegmark, 2018, Proposition 4.6) and (Schwab & Zech, 2019, Proposition 3.7) as in all cases a d-dimensional product is approximated. We refine these proofs and in particular include a bound on the network's weights and biases.

We start with the neural network approximations of the univariate basis functions ψ_{h_i,x_i^0} by a neural network $\theta_i^{\text{hat}} \in \Theta(4,5)$ up to accuracy $\delta = 2/9d^{-\log_2 3}\varepsilon < 1$, see Lemma 3. Using Lemma 1(b), we can create a network $\theta^0 \in \Theta(4, 5d, d, d)$, which evaluates these networks in parallel:

$$N\left(\mathbf{x} \mid \theta^0\right) = \left(N(x_1 \mid \theta_1^{\text{hat}}), N(x_2 \mid \theta_2^{\text{hat}}), \ldots, N(x_d \mid \theta_d^{\text{hat}})\right) \in \mathbb{R}^d$$

Formally we extend the d terms of the output by constant ones until we reach $2^{\lceil \log_2 d \rceil}$ terms:

$$\left(N(x_1 \mid \theta_1^{\text{hat}}), N(x_2 \mid \theta_2^{\text{hat}}), \ldots, N(x_d \mid \theta_d^{\text{hat}}), 1, \ldots 1\right) \in \mathbb{R}^{2^{\lceil \log_2 d \rceil}}$$

Then we use the multiplication operator in a binary tree structure to multiply these terms. In the first row of the tree structure, we have θ^1 with

$$N\left(\mathbf{x} \mid \theta^1\right) = \Bigg(N\Big(\left(N(x_1 \mid \theta_1^{\text{hat}}), N(x_2 \mid \theta_2^{\text{hat}})\right) \mid \theta_{\times,1+\delta}\Big), \\ N\Big(\left(N(x_3 \mid \theta_3^{\text{hat}}), N(x_4 \mid \theta_4^{\text{hat}})\right) \mid \theta_{\times,1+\delta}\Big), \cdots\Bigg) \in \mathbb{R}^{2^{\lceil \log_2 d \rceil - 1}}.$$

Using Lemmas 1(a, b) and 2(b), $2^{\lceil \log_2 d \rceil - 1} < d$ multiplications can be approximated in parallel in $\Theta(1, 4d, d, 2^{\lceil \log_2 d \rceil - 1})$. To compose the multiplications and the basis functions, we use Lemma 1(d) which shows $\theta^1 \in \Theta(5, 5d, d, 2^{\lceil \log_2 d \rceil - 1})$.

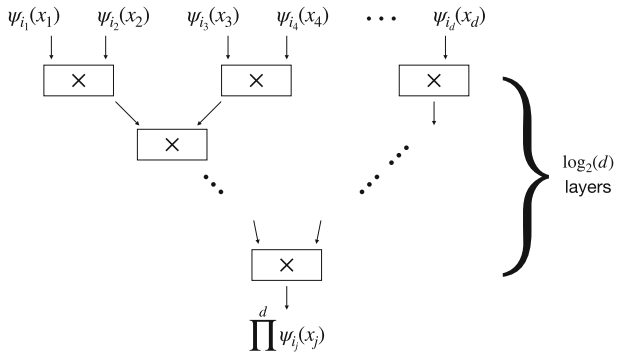

Fig. 1 Illustration of the multiplication stricture. Displayed in the special case, where $\log_2 d \in \mathbb{N}$

Adding the second row analogously, we define θ^2 with

$$N\left(\mathbf{x} \mid \theta^2\right) = \Big(N\Big(\big(N(\mathbf{x} \mid \theta^1)_1, N(\mathbf{x} \mid \theta^1)_2\big) \mid \theta_{\times,1+4\delta}\Big), \\ N\Big(\big(N(\mathbf{x} \mid \theta^1)_3, N(\mathbf{x} \mid \theta^1)_4\big) \mid \theta_{\times,1+4\delta}\Big), \dots\Big) \in \mathbb{R}^{2^{\lceil \log_2 d \rceil - 2}}$$

where Lemma 1(d) yields $\theta^2 \in \Theta(6, 5d, d, 2^{\lceil \log_2 d \rceil - 2})$. We continue the construction for $\theta^l \in \Theta(4 + l, 5d, d, 2^{\lceil \log_2 d \rceil - l})$ to construct $\theta_{\mathbf{h},\mathbf{x}^0} = \theta^{\lceil \log_2 d \rceil} \in \Theta(4 + \lceil \log_2 d \rceil, 5d, d, 1)$, see Fig. 1 for an illustration of the multiplicative tree.

In the following, we inductively show that the error after each layer is bounded by $\tau_l = 1/2(3^{l+1} - 1)\delta < 1$. With $\tau_0 = \delta$, the induction start holds.

Let us assume the approximation error on layer l is bounded by τ_l. We denote the exact product approximated by $N(\mathbf{x} \mid \theta^l)_i$ by $f_i^l(\mathbf{x})$, $i = 1, \dots, 2^{\lceil \log_2 d \rceil - l}$. As f_i^l is constructed as a multiplication of univariate hat functions (each of which are bounded by 1), we have $|f_i^l(\mathbf{x})| \leq 1$. By the induction hypothesis it holds

$$\sup_{\mathbf{x} \in [0,1]^d} \left| N\left(\mathbf{x} \mid \theta_i^l\right) - f_i^l(\mathbf{x}) \right| \leq \tau_l, \quad \text{for } i = 1, \dots, n.$$

We consider an arbitrary multiplication on layer $l + 1$: for some $i \in \{1, \dots, 2^{\lceil \log_2 d \rceil - (l+1)}\}$ the approximation of $f_i^{l+1}(\mathbf{x}) = f_{2i-1}^l(\mathbf{x}) \cdot f_{2i}^l(\mathbf{x})$ by

$$N\left(\mathbf{x} \mid \theta^{l+1}\right)_i = N\Big(\big(N(\mathbf{x} \mid \theta^l)_{2i-1}, N(\mathbf{x} \mid \theta^l)_{2i}\big) \mid \theta_{\times,1+\tau_l}\Big).$$

Using Lemma 2(b), we have for $\mathbf{x} \in [0, 1]^d$:

$$\begin{aligned}
&|N\left(\mathbf{x} \mid \theta^{l+1}\right)_i - f_{2i-1}^l(\mathbf{x}) \cdot f_{2i}^l(\mathbf{x})| \\
&\quad \leq \left| N\Big(\big(N(\mathbf{x} \mid \theta^l)_{2i-1}, N(\mathbf{x} \mid \theta^l)_{2i}\big) \mid \theta_{\times,1+\tau_l}\Big) - N(\mathbf{x} \mid \theta^l)_{2i-1} \cdot N(\mathbf{x} \mid \theta^l)_{2i} \right| \\
&\qquad + \left| N(\mathbf{x} \mid \theta^l)_{2i-1} \cdot N(\mathbf{x} \mid \theta^l)_{2i} - f_{2i-1}^l(\mathbf{x}) \cdot f_{2i}^l(\mathbf{x}) \right| \\
&\quad \leq \delta + \left| N(\mathbf{x} \mid \theta^l)_{2i-1} - f_{2i-1}^l(\mathbf{x}) \right| \cdot \left| N(\mathbf{x} \mid \theta^l)_{2i} \right| + \left| N(\mathbf{x} \mid \theta^l)_{2i} - f_{2i}^l(\mathbf{x}) \right| \cdot \left| f_{2i-1}^l(\mathbf{x}) \right| \\
&\quad \leq \delta + \tau_l(1 + \tau_l) + \tau_l \leq \delta + 3\tau_l = \tau_{l+1}.
\end{aligned}$$

After $\lceil \log_2 d \rceil$ layers, we arrive at a single network $\theta_{\mathbf{h},\mathbf{x}^0}$, which approximates $\Psi_{\mathbf{h},\mathbf{x}^0}$ up to an accuracy of

$$\tau_{\lceil \log_2 d \rceil} = \frac{1}{2}(3^{\lceil \log_2 d \rceil + 1} - 1)\delta \leq \frac{9}{2} 3^{\log_2 d} \delta = \varepsilon.$$

Since the absolute values of the input weights for $\theta_{\times,a}$ are given by $t_a/2 = \sqrt{3}/6\delta a^{-3} \leq \sqrt{3}/6$, see (2), Lemma 1(d) shows that by concatenating the networks, the asymptotic bounds of the weights remain in place. Thus we have $|\theta_{\mathbf{h},\mathbf{x}^0}|_\infty \leq c\delta^{-2} = c d^{2\log_2 3} \varepsilon^{-2}$. □

With the approximation of each basis function, we show approximations of sparse grid functions.

Lemma 5 *Let a sparse grid function be given:*

$$f_n = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathbf{l}} v_{\mathbf{l},\mathbf{i}} \, \Psi_{\mathbf{l},\mathbf{i}},$$

with bound (1): $|v_{\mathbf{l},\mathbf{i}}| \leq 2^{-d-|\mathbf{l}|_1}$. *With M the number of sparse grid basis functions, there exists* $\theta_{f,n} \in \Theta(4 + \lceil \log_2 d \rceil, 5dM)$ *with* $\sigma = \tanh$, *such that*

$$\sup_{\mathbf{x} \in [0,1]^d} |f_n(\mathbf{x}) - N(\mathbf{x} \mid \theta_{f,n})| \leq \varepsilon,$$

and $|\theta_{f,n}|_\infty \leq \max\{1, c d^{\log_2 3} 2^{-4d} M^2 \varepsilon^{-2}\}$, *where c is independent of n, f_n, d and ε.*

Proof The proof is closely related to (Montanelli & Du, 2019, Theorem 1), where only ReLU activation functions were used. ReLU neural networks can exactly represent the univariate basis functions, but are less efficient when approximating multiplications. As a consequence, estimating the approximation error for smooth activation functions differs even though the construction is analogue.

We denote the neural network approximations of each basis function $\Psi_{\mathbf{l},\mathbf{i}}$ with an accuracy of δ as $\theta_{\mathbf{l},\mathbf{i}} = \theta_{\mathbf{h}_{\mathbf{l}},\mathbf{x}_{\mathbf{l},\mathbf{i}}}$. Using Lemma 1(c) to add these M networks, we have $\theta_{f,n} \in \Theta(4 + \lceil \log_2 d \rceil, 5dM)$ such that

$$N(\mathbf{x} \mid \theta_{f,n}) = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}} \, N(\mathbf{x} \mid \theta_{\mathbf{l},\mathbf{i}}).$$

By construction we have for $\mathbf{x} \in [0,1]^d$:

$$\begin{aligned} |f_n(\mathbf{x}) - N(\mathbf{x} \mid \theta_{f,n})| &\leq \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}} \, |\Psi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) - N(\mathbf{x} \mid \theta_{\mathbf{l},\mathbf{i}})| \\ &\leq \delta \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}}. \end{aligned}$$

Using $|v_{\mathbf{l},\mathbf{i}}| \leq 2^{-d-|\mathbf{l}|_1}$, as given by (1), and $|\mathcal{I}_\mathbf{l}| = 2^{|\mathbf{l}|_1 - d}$ we have

$$\sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}} \leq |\mathcal{I}_\mathbf{l}| 2^{-d-|\mathbf{l}|_1} = 2^{-2d},$$

and thus

$$\sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathcal{I}_\mathbf{l}} v_{\mathbf{l},\mathbf{i}} \leq 2^{-2d} |\{\mathbf{l} : |\mathbf{l}|_1 \leq n + d - 1, \mathcal{I}_\mathbf{l} \neq \emptyset\}|.$$

The number of layers with a non-empty index-set is in any case bounded by the number of basis functions M, which yields

$$|f_n(\mathbf{x}) - N(\mathbf{x} \mid \theta_{f,n})| \leq \delta 2^{-2d} M.$$

Setting $\delta = \min\{1, 2^{2d}\varepsilon/M\}$, we get

$$|f_n(\mathbf{x}) - N(\mathbf{x} \mid \theta_{f,n})| \leq \varepsilon.$$

The coefficients can then be bounded by

$$|\theta_{f,n}|_\infty \leq c d^{\log_2 3} \delta^{-2} = c d^{\log_2 3} \max\{1, 2^{-4d} M^2 \varepsilon^{-2}\}.$$

□

Now we can summarise our theory.

Theorem 1 *For $f \in X_0^{\infty,2}([0,1]^d)$ with $|f|_{2,\infty} = 1$, there exists a deep neural network $\theta_{f,\varepsilon} \in \Theta(L, w, d, 1)$ with $\sigma = \tanh$, $L = 4 + \lceil \log_2 d \rceil$ and $w = \mathcal{O}\left(\varepsilon^{-1/2} |\log \varepsilon|^{3/2(d-1)}\right)$, such that*

$$\sup_{\mathbf{x} \in [0,1]^d} |f(\mathbf{x}) - N(\mathbf{x} \mid \theta_f)| \leq \varepsilon$$

with

$$|\theta_f|_\infty \leq \mathcal{O}\left(\varepsilon^{-3} |\log \varepsilon|^{3(d-1)}\right).$$

Proof Fist, we construct the sparse grid approximation

$$f_n = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{\mathbf{i} \in \mathbf{l}} v_{\mathbf{l},\mathbf{i}} \, \Psi_{\mathbf{l},\mathbf{i}},$$

up to the accuracy $\varepsilon/2$. From the approximation theory of sparse grids (Bungartz & Griebel, 2004, Lemma 3.13), we have

$$M = \mathcal{O}\left(\varepsilon^{-1/2} |\log \varepsilon|^{3/2(d-1)}\right)$$

and (Bungartz & Griebel, 2004, Lemma 3.3) shows (1).

This shows that we can apply Lemma 5 to construct $\theta_f = \theta_{f,n}$, choosing the same accuracy $\varepsilon/2$. Then for $\mathbf{x} \in [0,1]^d$,

$$|f(\mathbf{x}) - N(\mathbf{x} \mid \theta_f)| \leq |f(\mathbf{x}) - f_n(\mathbf{x})| + |f_n(\mathbf{x}) - N(\mathbf{x} \mid \theta_f)| \leq \varepsilon.$$

□

3.4 Application of the results to the deep parametric PDE method

In this chapter, we discuss how the result can be applied to the deep parametric PDE method. We note that the theoretical results are necessary but not sufficient for an efficient approximation in high dimensions. The reasons are elaborated in the following.

Theorem 1 in Glau and Wunderlich (2022) shows that the L^2-error of the deep parametric PDE method is bounded by the value of the loss function, which is minimised:

$$\|u - u_{\mathrm{DPDE}}\|_{L^2(\mathcal{Q})} \leq c \mathcal{J}(u_{\mathrm{DPDE}}) = c \operatorname*{arg\,min}_{u_{\mathrm{DNN}}} \mathcal{J}(u_{\mathrm{DNN}}).$$

As the loss includes terms of the second derivative, further estimates require approximation results for the first and second derivative. Also the reduced regularity of most option pricing problems prevent a direct application of the results shown in the previous section. Both points are subject of future research.

On contrast, the results in this article show necessary conditions. For the deep parametric PDE method to be efficient in high dimensions, the best approximation in the L^2 norm must not deteriorate for high dimensions, such as shown here with a dimension-independent rate. Without these results, the best-approximation in the H^2 norm would have an approximation rate which decreases for higher dimensions, making the method inefficient. Thus our results are necessary for an efficient high-dimensional method.

While we acknowledge this remaining gap in the theory, the numerical results provide further insights into the efficiency of the method for higher dimensions.

4 Applications in credit risk management

As an illustrative example of the practical performance of the deep parametric PDE method, we investigate the evaluation of credit exposure, see Gregory (2010); Green (2015) for further information.

We consider the exposure $V_t(X_t; \mu)$ at a given parameter μ, where X_t is the random variable describing the underlying assets/risk-factors. In our case, we have $V_t(x; \mu) = \text{Price}(t, e^x; \mu) > 0$ (otherwise, only the positive part is considered).

For a given real-world measure $\mathbb{P}$, we consider the expected exposure

$$\text{EE}^{\text{risk}}(t) = \mathbb{E}^{\mathbb{P}}(V_t)$$

and the potential future exposure for risk levels α:

$$\text{PFE}^{\text{risk}}{}_{\alpha}(t) = \inf\{y : \mathbb{P}(V_t > y) \leq 1 - \alpha\}.$$

In the numerical experiment we consider three different risk levels: $\alpha = 95\%, 97.5\%$ and 99%.

In addition, we compute the expected exposure in the riskfree measure $\mathbb{Q}$ as we know its exact value:

$$\text{EE}^{\text{price}}(t) = e^{-rt}\mathbb{E}^{\mathbb{Q}}(V_t) = V_0$$

To evaluate these exposure measures, we use a full re-evaluation. On a grid of time-evaluations $0 < t_1 < t_2 < \ldots < t_n \leq T$:

1. Sample M random paths for X_t according to $\mathbb{P}$ or $\mathbb{Q}$, each evaluated at t_i: $\widehat{X}_i^j$, $j = 1, \ldots, M$;
2. Compute the asset prices in each scenario $v_{i,j} = V_{t_i}(X_i^j; \mu)$;
3. Evaluate the mean value for the expected exposure:
   - In the real-world measure: $\widehat{\text{EE}^{\text{risk}}}(t_{\hat{1}}) = \sum_{j=1}^{M} v_{\hat{1},j}/M$;
   - For the risk-neutral measure, include the discount factor: $\widehat{\text{EE}^{\text{price}}}(t_{\hat{1}}) = e^{-rt_{\hat{1}}} \sum_{j=1}^{M} v_{\hat{1},j}/M$.
4. For each time $t_{\hat{1}}$ sort the values $(v_{\hat{1},j})_j$ and return the value at the position $\lceil \alpha M \rceil$ as the potential future exposure $\widehat{\text{PFE}^{\text{risk}}}_{\alpha}(t_{\hat{1}})$.

We note that this exposure calculation has two sources of a numerical error: the Monte-Carlo approximation of the risk factors as well as the accuracy of the solver.

We use $M = 150\,000$ simulations to calculate the exposure. When we compare to a reference solution, we use $M = 1\,500\,000$ simulations and evaluate the price using the reference pricer based on Bayer et al. (2018). More precisely: we use the dimensionality reduction described in their article and solve the remaining auxiliary problem using a tensorised Gauss-Hermite quadrature. Due to the smoothness of the auxiliary problem as small number of 9 quadrature points per dimension is sufficient for a relative error of only 0.13%. For two underlying assets, the auxiliary problems in one-dimensional, so we can use more quadrature points. With 33 Gauss-Hermite points we observed an accuracy close to machine precision. Further details can also be found in Glau and Wunderlich (2022).

For the deep parametric PDE method, we use the implementation developed in Glau and Wunderlich (2022). A sample code for two assets is available on GitHub.[3] The computational domain covers $s^i \in [21, 460]$, $t \in [0, 4]$, $\sigma_i \in [0.1, 0.3]$, $r \in [0.01, 0.03]$ and $\rho_{i,i+1} \in [0.2, 0.8]$. Unless noted otherwise, we consider a riskfree rate $r = 0.01$ and a drift (not used in pricing) of $\mu = 0.02$. For asset values outside of the computational domain, we return the value of the payoff as the asymptotics. We compare the deep parametric PDE method to a Monte-Carlo solver with 1 000 samples, the recently proposed sparse grid stochastic collocation method (Grzelak, 2021) and the reference pricer.

We note that, although the approximation results in Theorem 1 were shown using sparse grids, there is no direct connection of our theoretical work to the sparse grid stochastic collocation method. The main motivation to consider this method is to compare the deep parametric PDE method to another modern method intended for medium to high dimensions.

4.1 Evaluations at fixed parameter

We start our investigation on fixed parameters with two and five underlying assets. As the deep parametric PDE method solves for a whole range of parameter values, the training phase was not specific to these parameters.

4.1.1 Two asset case

We consider two assets with the initial value $s_0 = (50, 125)$, individual volatilities $\sigma = (0.1, 0.3)$ and a correlation of $\rho = 0.2$. We evaluate the exposure at 41 points in time.

In the two-dimensional case, we also investigate the need for high-order models. Therefore, we include a fitted one-dimensional model in our comparison. For $S_t^i/s_0^i \sim \mathcal{LN}((r - \sigma_i^2/2)t, \sigma_i^2 t)$, $i = 1, 2$, $\mathrm{corr}(\log(S_t^1), \log(S_t^2)) = \rho$, we consider an independent univariate process $\bar{S}_t/\bar{s}_0 \sim \mathcal{LN}(\bar{\mu}t, \bar{\sigma}^2 t)$, with the same expectation and variance as the basket: $\mathbb{E}(\bar{S}_t) = \mathbb{E}((S_t^1 + S_t^2)/2)$ and $\mathrm{var}(\bar{S}_t) = \mathrm{var}(S_t^1 + S_t^2)/2)$. Using this fitted model, the Black-Scholes formula provides an approximation of the asset price.

Our first comparison considers the expected exposure in the riskfree measure. In this case, we know that the exposure profile is constant during the lifetime of the option. In Fig. 2 we see the expected exposure as well as its relative error. At maturity $t = 4$ all methods deliver the same exposure, as the option price coincides with the payoff. With the exception of the one-dimensional model, all methods yield an acceptable exposure profile over the lifetime of the option. The fitted one-dimensional model exhibits a significantly larger error than

[3] https://github.com/LWunderlich/DeepPDE/tree/main/TwoAssetsExample.

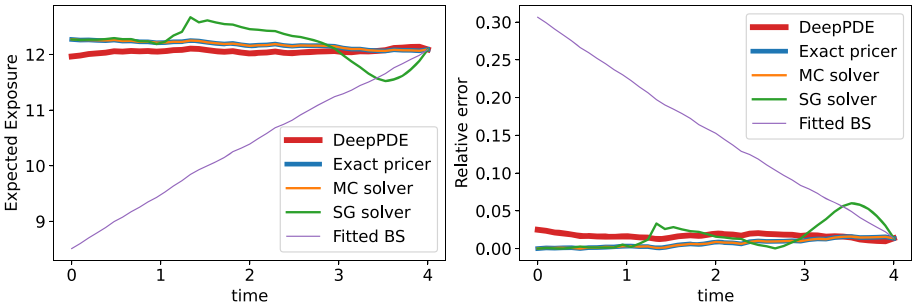

Fig. 2 Expected exposure in pricing measure (left) and its relative error (right). Two underlying assets with maturity $T = 4$

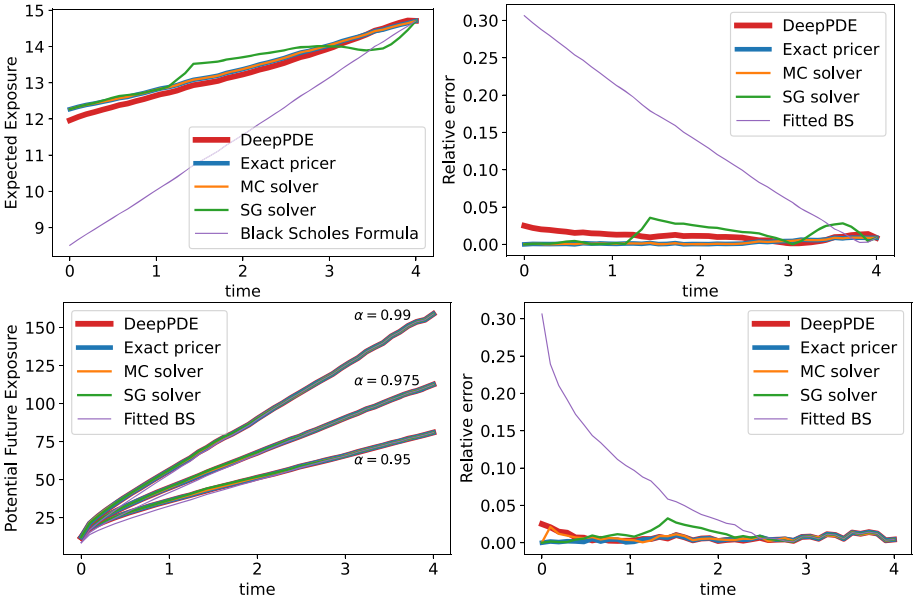

Fig. 3 Expected exposure (top) and potential future exposure for $\alpha = 95\%, 97.5\%$ and 99% (bottom) in real-world measure (left) and their relative error (right). For the error in the potential future exposure, the maximal error of the three risk levels is measured. Two underlying assets

the multivariate models. This shows the importance to consider multivariate models as low-dimensional fits do not produce accurate results. The sparse grid solver with 4 refinements shows more reasonable results, but still exhibits a notable error. The deep parametric PDE method only exhibits a slightly larger error than the reference pricer and the Monte-Carlo method. This confirms the accuracy of the deep parametric PDE method.

In Fig. 3, we see the expected exposure as well as the potential future exposure for three different risk levels in the real-world measure. We see a similar picture as before, with the multivariate pricer consistently showing relative errors below 5%, while the fitted one-dimensional method shows up to 30% error. The results are similar for the expected exposure and the potential future exposure. In this situation we note that the deep parametric PDE method and the sparse grid solver have a similar maximal error. While the deep parametric

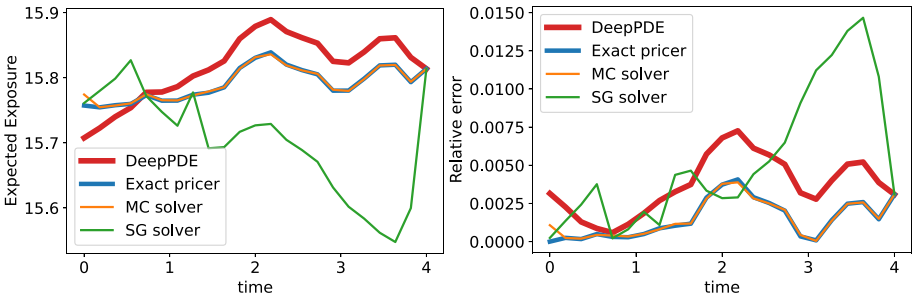

Fig. 4 Expected exposure in pricing measure (left) and its relative error (right). Five underlying assets

Table 1 Runtime comparison for the evaluation of the expected exposure and the potential future exposure for three risk levels at 41 points in time

	Monte-Carlo	Reference pricer	Sparse grid	Deep parametric PDE
Two asset	21.3 min	55.6 min	0.6 min	7.0 min
Five assets	37.4 min	184.0 min	6.6 min	7.1 min

PDE method shows the largest error in the near future, the sparse grid method has its largest error in the middle of the time interval.

Due to the large error of the fitted one-dimensional model, we will not consider it in the remaining comparisons.

4.1.2 Five asset case

For five assets, we consider initial values $s_0 = (125, 100, 75, 150, 80)$, volatilities $\sigma = (0.1, 0.3, 0.25, 0.15, 0.2)$ and pairwise correlations $(\rho_{i,i+1})_i = (0.3, 0.6, 0.4, 0.5)$. We note that due to the parameters, the pricing problem is already 16-dimensional. Due to the longer runtimes of the reference solver, we only consider 21 points in time.

In Fig. 4the expected exposure in the riskfree measure and its error are shown. Again, we see similar error values in all four cases with the error due to the Monte-Carlo sampling of the risk-factors significantly contributing to the overall error. In the five-dimensional case, we note that the deep parametric PDE method results in about half the maximal error when compared to the sparse grid approach.

A similar picture is seen when considering the real-world measure, as shown in Fig. 5. In all cases a good level of approximation is observed, with relative errors below 2% for the expected exposure and below 1% for the potential future exposure. In both cases we see more accurate results using the deep parametric PDE method than using the sparse grid solver and the deep parametric PDE method has the same accuracy as the reference pricer.

4.2 Runtime comparisons

With three of the four methods showing similar error values, the computational effort is decisive for the comparison. We note that the sparse grid approach exhibits larger error values, especially for five underlying assets. In Table 1the runtimes for the presented examples are given. The runtimes measure the computation of the expected exposure and the three potential

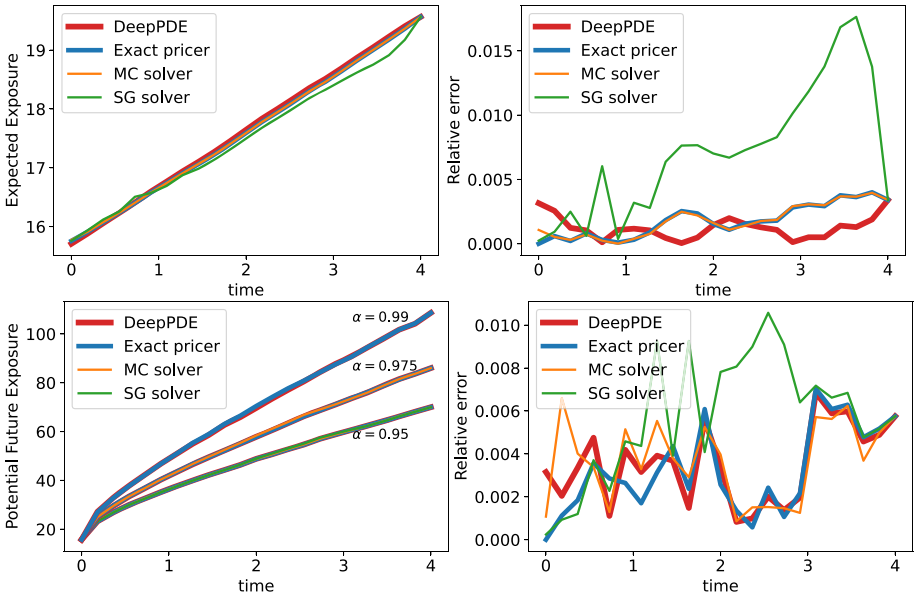

Fig. 5 Expected exposure (top) and potential future exposure for $\alpha = 95\%, 97.5\%$ and 99% (bottom) in real-world measure (left) and their relative error (right). For the error in the potential future exposure, the maximal error of the three risk levels is measured. Five underlying assets

future exposures at considered points in time. The observed runtimes for five assets were scaled for comparability to account for the smaller number of time-steps in our examples. The code was evaluated on a typical end-user device, a 2015 MacBook Pro with a 2.7GHz dual-core CPU and 8GB RAM.

In all cases, the evaluation of exposures using the deep parametric PDE methods was significantly faster than the traditional methods, due to the use of neural networks. In comparison to the Monte-Carlo solver the gain was up to a factor of 5, while the speed-up factor compared to the reference pricer was up to 25. We note that the evaluation using the sparse grid approach was significantly faster for two assets with only a slightly larger error. This confirms that both methods are well suited for the task. However for five assets, the sparse grid approach takes almost as long as the deep parametric PDE method, but shows about twice the error. This highlights an important feature of the deep parametric PDE method: the stability of the run-time with a growing number of dimensions. The deep parametric PDE method does not take significantly longer to evaluate exposures for five assets than it takes to evaluate two assets. In contrast, the Monte-Carlo method, the sparse grid method and the reference pricer take significantly longer for higher-dimensions. This confirms our theoretical results, showing that the deep parametric PDE method significantly reduces the curse of dimensionality.

4.3 Parametric evaluations

The parametric PDE method does not only return the solution for a single parameter value, but for a whole range of option and market parameters. To demonstrate the full performance

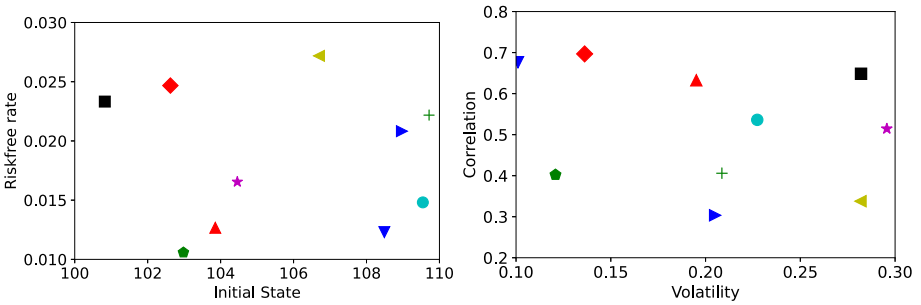

Fig. 6 The ten random parameter values used in the experiments: initial state $\bar{s}_0$, the riskfree rate r, the volatility $\bar{\sigma}$ and the correlation $\bar{\rho}$

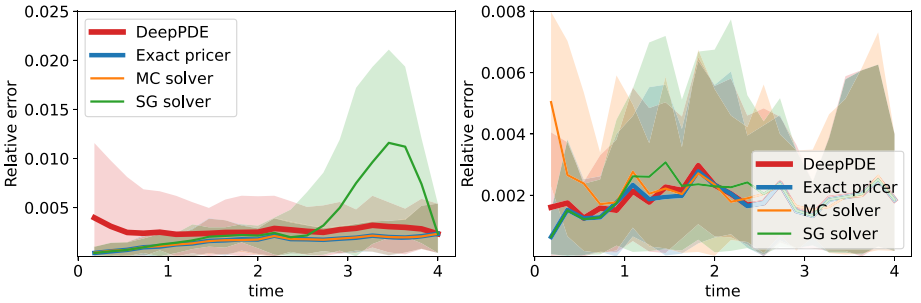

Fig. 7 Average error for expected exposure (left) and PFE (right) in the real-world measure $\mathbb{P}$. Area between maximal and minimal observed errors shaded

of the deep parametric PDE method, we present examples for exposure calculations in a parametric setting. For scenario calculations and stress-tests, this is a highly relevant task.

To limit the amount of random states, we consider equal parameters for all five assets: we vary the initial state $s_0 = (\bar{s}_0, \dots, \bar{s}_0) \in \mathbb{R}^5$, the volatilities $\sigma = (\bar{\sigma}, \dots, \bar{\sigma}) \in \mathbb{R}^5$ and pairwise correlations $(\rho_{i,i+1})_i = (\bar{\rho}, \dots, \bar{\rho}) \in \mathbb{R}^4$. In addition, the riskfree rate of return is chosen randomly. We consider ten different random parameters, which are displayed in Fig. 6. In each case, we evaluate the expected exposure and the potential future exposure and estimate their error.

To be able to evaluate the reference pricer, we had to reduce the complexity of the problem. Therefore, we only evaluate the exposure at 21 points in time and use $M = 750\,000$ for the calculation of reference values.

Figure 7 shows the relative errors in all ten examples. We see the average error over the ten scenarios as well as the maximal and minimal error. Again, we see a similar level of error for all four methods. For the expected exposure and $t < 0.5$, we see a slightly larger error for the deep parametric PDE method, while for the potential future exposure, we see a slightly larger error for the Monte Carlo solver. Nevertheless, in both cases the error remains small, almost always below 1%. The sparse grid solver again shows a larger maximal error than the deep parametric PDE method. We note that again the error for the deep parametric PDE method is largest in the near future, while the sparse grid error is larger towards the maturity of the option.

In summary, in all considered cases, the deep parametric PDE method provides accurate results with a speed-up factor of up to 25 to the reference pricer and 5 to the Monte-Carlo

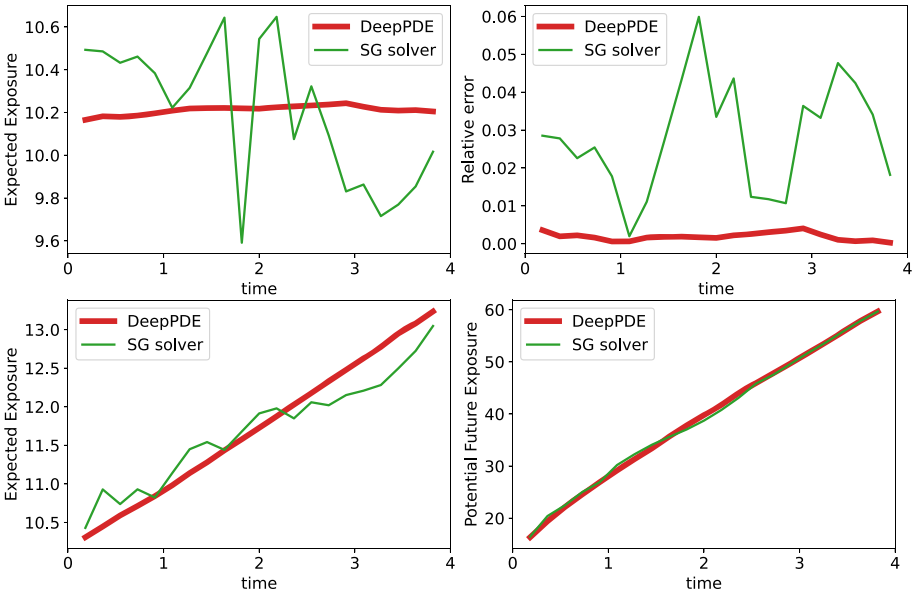

Fig. 8 Exposures for 10 underlying assets. Top row: Expected exposure in the pricing measure $\mathbb{Q}$ (left) and its relative error (right). Bottom row: Expected exposure (left) and potential future exposure (right) in the real-world measure $\mathbb{P}$

method. A calculation which used to take almost an hour can now be performed in under 10 min. In comparison to a state-of-the-art sparse grid solver we observed half the error with a comparable runtime.

4.4 Higher dimensional calculation

We have seen a clear performance gain of the deep parametric PDE method in comparison to a full re-evaluation with a Monte-Carlo solver as well as the reference pricer. In comparison to another modern high-dimensional solver the performance gain for two and five assets was less pronounced, but we have observed a trend that the deep parametric PDE method performs better for higher dimensions. In this section we continue this investigation and consider a case of ten underlying assets. We highlight that the deep parametric PDE method solves the option price for all time, state and parameter values at the same time, which overall is a 31-dimensional problem.

We evaluate the expected exposure and the potential future exposure (for risk-level $\alpha = 97.5\%$) at 21 points in time. We consider an initial value of 100 and a volatility of 0.2 for each of the ten assets. The pairwise correlation is set to 0.5 for all pairs, the riskfree rate is set to 0.01 and the drift of the real-world measure is 0.02.

Due to the high dimensionality we have not computed reference values except for the constant expected exposure in the riskfree measure. As the expected exposure in this situation is equal to the current option price we could use a Monte-Carlo method with 1 000 000 evaluations to compute the single value. Note that we had to reduce the number of refinements of the sparse grid method to three to obtain reasonable run-times.

In Fig. 8, the evaluated expected exposures as well as their relative error are shown. We

see a good accuracy of below 0.5% for the deep parametric PDE method, while the sparse grid method shows larger errors of up to 6%. A similar picture can be seen for the quantities in the real-world measure $\mathbb{P}$, which are also shown in the figure. Even though we do not have a reference value available, we can see that the deep parametric PDE method provides significantly more accurate results.

In addition to the improved accuracy, the deep parametric PDE method features faster runtimes. While the sparse grid method took 9.9min for the calculation of the expected exposures and the potential future exposure, the deep parametric PDE method took only about half of the time with 4.3min. Note that in comparison to Table 1 we only considered 21 evaluations in time instead of 41. To compare the values, we need to roughly double the observed runtimes. We then notice only a small increase in the runtime for the deep parametric PDE method between 2 and 10 dimensions.

5 Conclusion

In this article, we advanced the theory of the deep neural networks with smooth activation functions and demonstrated the practical use of the parametric PDE method in a situation of interest for the financial industry.

We have shown (as far as we are aware) the first approximation results for deep neural networks with smooth activation functions that exhibits an approximation rate independent of the dimension, therefore reducing the curse of dimensionality. The proofs employs valuable sparse grid estimates and arithmetic operations on DNNs.

We confirm the efficiency in high dimensions which is implied by these results in an example from credit exposure. We have found that the deep parametric PDE method consistently outperforms our reference method and a Monte-Carlo solver. While for two underlying assets the sparse grid stochastic collocation method has a competing performance, also this method was outperformed for five and especially ten underlying assets. We have observed speed-up factors of up to 25 on a standard office laptop.

Both the theoretical and the practical results show the strong performance of DNN-based methods, especially in finance.

Funding This work was funded by the EPSRC grant no. EP/T004738/1. This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. Doi: https://doi.org/10.5281/zenodo.438045.

References

Abdeljawad, A., & Grohs, P. (2022). Approximations with deep neural networks in Sobolev time-space. *Analysis and Applications, 20*(03), 499–541.

Adcock, B., & Dexter, N. (2021). The gap between theory and practice in function approximation with deep neural networks. *SIAM Journal on Mathematics of Data Science, 3*(2), 624–655.

Andersson, K., & Oosterlee, C. W. (2021). A deep learning approach for computations of exposure profiles for high-dimensional Bermudan options. *Applied Mathematics and Computation, 408*, 126332.
Andersson, K., & Oosterlee, C. W. (2021). Deep learning for CVA computations of large portfolios of financial derivatives. *Applied Mathematics and Computation, 409*, 126399.
Antonov, A., & Piterbarg, V. (2021). Alternatives to deep neural networks for function approximations in finance. https://papers.ssrn.com/sol3/papers.cfm?abstractid=3958331.
Barron, A. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory, 39*(3), 930–945.
Bayer, C., Siebenmorgen, M., & Tempone, R. (2018). Smoothing the payoff for efficient computation of basket option prices. *Quant. Finance, 18*(3), 491–505.
Beck, C., Hutzenthaler, M., Jentzen, A., & Kuckuck, B. (2020). An overview on deep learning-based approximation methods for partial differential equations. arXiv:https://arxiv.org/abs/2012.12348.
Berner, J., Grohs, P., Kutyniok, G., & Petersen, P. (2021). The modern mathematics of deep learning. arXiv:https://arxiv.org/abs/2105.04026.
Boulle, N., Nakatsukasa, Y., & Townsend, A. (2020). Rational neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 14243–14253). Curran Associates, Inc.
Bungartz, H.-J., & Griebel, M. (2004). Sparse grids. *Acta Numerica, 13*, 147–269.
Elbrächter, D., Grohs, P., Jentzen, A., & Schwab, C. (2021). DNN expression rate analysis of high-dimensional PDEs: Application to option pricing. Constructive Approximation.
Elbrächter, D., Perekrestenko, D., Grohs, P., & Bölcskei, H. (2021). Deep neural network approximation theory. *IEEE Transactions on Information Theory, 67*(5), 2581–2623.
Garcke, J., & Griebel, M. (Eds.). (2013). *Sparse grids and applications.* Berlin: Springer.
Germain, M., Pham, H., & Warin, X. (2021). Neural networksbased algorithms for stochastic control and PDEs in finance. arXiv:https://arxiv.org/abs/2101.08068 .
Glau, K., Mahlstedt, M., & Pötz, C. (2019). A new approach for American option pricing: The dynamic Chebyshev method. *SIAM Journal on Scientific Computing, 41*(1), B153–B180.
Glau, K., Pachon, R., & Pötz, C. (2021). Speed-up credit exposure calculations for pricing and risk management. *Quantitative Finance, 21*(3), 481–499.
Glau, K., & Wunderlich, L. (2022). The deep parametric PDE method and applications to option pricing. *Applied Mathematics and Computation, 432*, 127355.
Gnoatto, A., Picarelli, A., & Reisinger, C. (2020). Deep xVA solver - a neural network based counterparty credit risk management framework. arXiv:https://arxiv.org/abs/2005.02633 .
Green, A. (2015). *Xva?: Credit, funding and capital valuation adjustments*. Chichester: Wiley.
Gregory, J. (2010). *Counterparty credit risk: The new challenges for global financial markets*. Chichester: Wiley.
Grohs, P., & Herrmann, L. (2021). Deep neural network approximation for high-dimensional parabolic Hamilton-Jacobi-Bellman equations. arXiv:https://arxiv.org/abs/2103.05744 .
Grohs, P., Hornung, F., Jentzen, A., & von Wurstemberger, P. (2018). A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. arXiv:https://arxiv.org/abs/1809.02362 .
Grohs, P., Hornung, F., Jentzen, A., & Zimmermann, P. (2019). Space-time error estimates for deep neural network approximations for differential equations. arXiv:https://arxiv.org/abs/1908.03833 .
Grzelak, L.A. (2021). Sparse grid method for highly efficient computation of exposures for xVA. arXiv:https://arxiv.org/abs/2104.14319 .
Gühring, I., Raslan, M., & Kutyniok, G. (2020). Expressivity of deep neural networks. arXiv:https://arxiv.org/abs/2007.04759 .
Gühring, I., Kutyniok, G., & Petersen, P. (2020). Error bounds for approximations with deep ReLU neural networks in Ws, p norms. *Analysis and Applications, 18*(05), 803–859.
Herrmann, L., Opschoor, J., & Schwab, C. (2022). Constructive deep ReLU neural network approximation. *Journal of Scientific Computing, 90*(75), 1–37.
Jentzen, A., & Riekert, A. (2021). A proof of convergence for the gradient descent optimization method with random initializations in the training of neural networks with ReLU activation for piecewise linear target functions. arXiv:https://arxiv.org/abs/2108.04620 .
Jentzen, A., Salimova, D., & Welti, T. (2021). A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *Communications in Mathematical Sciences, 19*(5), 1167–1205.
Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics, 3*, 422–440.

Langer, S. (2021). Approximating smooth functions by deep neural networks with sigmoid activation function. *Journal of Multivariate Analysis, 182*, 104696.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444.
Montanelli, H., & Du, Q. (2019). New error bounds for deep ReLU networks using sparse grids. *SIAM Journal on Mathematics of Data Science, 1*(1), 78–92.
Ohn, I., & Kim, Y. (2019). Smooth function approximation by deep neural networks with general activation functions. *Entropy, 21*(7), 627.
Opschoor, J., Schwab, C., & Zech, J. (2021). Exponential ReLU DNN expression of holomorphic maps in high dimension. Constr Approx.
Raissi, M., Perdikaris, P., & Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics, 378*, 686–707.
Reisinger, C., & Zhang, Y. (2020). Rectified deep neural networks overcome the curse of dimensionality for non-smooth value functions in zero-sum games of nonlinear stiff systems. *Analysis and Applications, 6*(18), 951–999.
Rolnick, D., & Tegmark, M. (2018). The power of deeper networks for expressing natural functions. International conference on learning representations.
Schwab, C., & Zech, J. (2019). Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ. *Analysis and Applications, 17*(01), 19–55.
Shin, Y., Darbon, J., & Em Karniadakis, G. (2020). On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. *Communications in Computational Physics, 28*(5), 2042–2074.
Siegel, J., & Xu, J. (2020). Approximation rates for neural networks with general activation functions. *Neural Networks, 128*, 313–321.
Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics, 375*, 1339–1364.
Tang, S., Li, B., & Yu, H. (2019). ChebNet: efficient and stable constructions of deep neural networks with rectified power units using Chebyshev approximations. arXiv:https://arxiv.org/abs/1911.05467 .
Telgarsky, M. (2017). Neural networks and rational functions. In D. Precup & Y.W. Teh (Eds.), Proceedings of the 34th international conference on machine learning (Vol. 70, pp. 3387-3393). PMLR.
Wang, Q. (2018). Exponential convergence of the deep neural network approximation for analytic functions. *SCIENCE CHINA Mathematics, 61*, 1733–1740.
Welack, S. (2019). Artificial neural network approach to counterparty credit risk and XVA. https://papers.ssrn.com/sol3/papers.cfm?abstractid=3312944 .
Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks, 94*, 103–114.