ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Seminar for Applied Mathematics SAM

On the approximation of rough functions with deep neural networks

T. De Ryck and S. Mishra and D. Ray

Research Report No. 2020-07

January 2020
Latest revision: October 2020

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Funding ERC: 770880 COMANFLO

ON THE APPROXIMATION OF ROUGH FUNCTIONS WITH DEEP NEURAL NETWORKS

T. DE RYCK, S. MISHRA, AND D. RAY

Abstract. The essentially non-oscillatory (ENO) procedure and its variant, the ENO-SR procedure, are very efficient algorithms for interpolating (reconstructing) rough functions. We prove that the ENO (and ENO-SR) procedure are equivalent to deep ReLU neural networks. This demonstrates the ability of deep ReLU neural networks to approximate rough functions to high-order of accuracy. Numerical tests for the resulting trained neural networks show excellent performance for interpolating functions, approximating solutions of nonlinear conservation laws and at data compression.

1. Introduction

Rough functions i.e, functions which are at most Lipschitz continuous and could even be discontinuous, arise in a wide variety of problems in physics and engineering. Prominent examples include (weak) solutions of nonlinear partial differential equations. For instance, solutions of nonlinear hyperbolic systems of conservation laws such as the compressible Euler equations of gas dynamics, contain shock waves and are in general discontinuous, [9]. Similarly, solutions to the incompressible Euler equations would well be only Hölder continuous in the turbulent regime, [12]. Moreover, solutions of fully non-linear PDEs such as Hamilton-Jacobi equations are in general Lipschitz continuous, [11]. Images constitute another class of rough or rather piecewise smooth functions as they are often assumed to be no more regular than functions of bounded variation on account of their sharp edges, [5].

Given this context, the efficient and robust numerical approximation of rough functions is of great importance. However, classical approximation theory has severe drawbacks when it comes to the interpolation (or approximation) of such rough functions. In particular, it is well known that standard linear interpolation procedures degrade to at best first-order of accuracy (in terms of the interpolation mesh width) as soon as the derivative of the underlying function has a singularity, [1] and references therein. This order of accuracy degrades further if the underlying function is itself discontinuous. Moreover approximating rough functions with polynomials can lead to spurious oscillations at points of singularity. Hence, the approximation of rough functions poses a formidable challenge.

Artificial neural networks, formed by concatenating affine transformations with pointwise application of nonlinearities, have been shown to possess universal approximation properties, [18, 6, 8] and references therein. This implies that for any continuous (even for merely measurable) function, there exists a neural network that approximates it accurately. However, the precise architecture of this network is not specified in these universality results. Recently in [26], Yarotsky was able to construct deep neural networks with ReLU activation functions and very explicit

(T. De Ryck and S. Mishra) Seminar for Applied Mathematics, ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland

(D. Ray) University of Southern California, Los Angeles, USA

E-mail addresses: tim.deryck@sam.math.ethz.ch, siddhartha.mishra@sam.math.ethz.ch, deepray@usc.edu.

estimates on the size and parameters of the network, that can approximate Lipschitz functions to second-order accuracy. Even more surprisingly, in a very recent paper [27], the authors were able to construct deep neural networks with alternating ReLU and Sine activation functions that can approximate Lipschitz (or Hölder continuous) functions to exponential accuracy.

The afore-mentioned results of Yarotsky clearly illustrate the power of deep neural networks in approximating rough functions. However, there is a practical issue in the use of these deep neural networks as they are mappings from the space coordinate $x \in D \subset \mathbb{R}^d$ to the output $f^*(x) \in \mathbb{R}$, with the neural network f^* approximating the underlying function $f : D \to \mathbb{R}$. Hence, for every given function f, the neural network f^* has to be *trained* i.e, its weights and biases determined by minimizing a suitable loss function with respect to some underlying samples of f, [14]. Although it makes sense to train neural networks to approximate individual functions f in high dimensions, for instance in the context of uncertainty quantification of PDEs, [20] and references therein, doing so for every low-dimensional function is unrealistic. Moreover, in a large number of contexts, the goal of approximating a function is to produce an interpolant $\tilde{f}$, given the vector $\{f(x_i)\}$ at sampling points $x_i \in D$ as input. Hence, one would like to construct neural networks that map the full input vector into an output interpolant (or its evaluation at certain sampling points). It is unclear if the function approximation results for neural networks are informative in this particular context.

On the other hand, data-dependent interpolation procedures have been developed in the last decades to deal with the interpolation of rough functions. A notable example of these data dependent algorithms is provided by the essentially non-oscillatory (ENO) procedure. First developed in the context of reconstruction of non-oscillatory polynomials from cell averages in [16], ENO was also adapted for interpolating rough functions in [23] and references therein. Once augmented with a sub-cell resolution (SR) procedure of [15], it was proved in [1] that the ENO-SR interpolant also approximated (univariate) Lipschitz functions to second-order accuracy. Moreover, ENO was shown to satisfy a subtle non-linear stability property, the so-called *sign property*, [13]. Given these desirable properties, it is not surprising that the ENO procedure has been very successfully employed in a variety of contexts, ranging from the numerical approximation of hyperbolic systems of conservation laws [16] and Hamilton-Jacobi equations [24] to data compression in image processing, [17, 1, 2] and references therein.

Given the ability of neural networks as well as ENO algorithms to approximate rough functions accurately, it is natural to investigate connections between them. This is the central premise of the current paper, where we aim to reinterpret ENO (and ENO-SR) algorithms in terms of deep neural networks. We prove the following results,

- We prove that for any order, the ENO interpolation (and the ENO reconstruction) procedure can be cast as a suitable deep ReLU neural network.
- We prove that a variant of the piecewise linear ENO-SR (sub-cell resolution) procedure of [15] can also be cast as a deep ReLU neural network. Thus, we prove that there exists a deep ReLU neural network that approximates piecewise smooth (say Lipschitz) functions to second-order accuracy.
- The above theorems provide the requisite architecture for the resulting deep neural networks and we train them to obtain what we term as DeLENO (deep learning ENO) approximation procedures for rough functions. We test this procedure in the context of numerical methods for conservation laws and for data and image compression.

Thus, our results reinforce the enormous abilities of deep ReLU neural networks to approximate functions, in particular rough functions and add a different perspective to many existing results on approximation with ReLU networks.

2. Deep neural networks

In statistics, machine learning, numerical mathematics and many other scientific disciplines, the goal of a certain task can often be reduced to the following. We consider a (usually unknown) function $\mathcal{L} : D \subset \mathbb{R}^m \to \mathbb{R}^n$ and we assume access to a (finite) set of labelled data $\mathbb{S} \subset \{(X, \mathcal{L}(X)) : X \in D\}$, using which we wish to select an approximation $\hat{\mathcal{L}}$ from a parametrized function class $\{\mathcal{L}^\theta : \theta \in \Theta\}$ that predicts the outputs of $\mathcal{L}$ on D with a high degree of accuracy.

One possible function class is that of deep neural networks (DNNs). In particular, we consider multilayer perceptrons (MLPs) in which the basic computing units (neurons) are stacked in multiple layers to form a feedforward network. The input is fed into the *source layer* and flows through a number of *hidden layers* to the *output layer*. An example of an MLP with two hidden layers is shown in Figure 1.

In our terminology, an MLP of depth L consists of an input layer, $L-1$ hidden layers and an output layer. We denote the vector fed into the input layer by $X = Z^0$. The l-th layer (with n_l neurons) receives an input vector $Z^{l-1} \in \mathbb{R}^{n_{l-1}}$ and transforms it into the vector $Z^l \in \mathbb{R}^{n_l}$ by first applying an affine linear transformation, followed by a component-wise (non-linear) activation function $\mathcal{A}^l$,

$$Z^l = \mathcal{A}^l(W^l Z^{l-1} + b^l), \quad W^l \in \mathbb{R}^{n_l \times n_{l-1}},\ b^l \in \mathbb{R}^{n_l}, \quad 1 \leqslant l \leqslant L, \tag{2.1}$$

with Z^l serving as the input for the $(l+1)$-th layer. For consistency, we set $n_0 = m$ and $n_L = n$. In (2.1), W^l and b^l are respectively known as the weights and biases associated with the l-th layer. The parameter space Θ then consists of all possible weights and biases. A neural network is said to be deep if $L \geqslant 3$ and such a deep neural network (DNN) is denoted as a ReLU DNN if the activation functions are defined by the very popular rectified linear (ReLU) function,

$$\mathcal{A}^l(Z) = (Z)_+ = \max(0, Z) \quad \text{for} \quad 1 \leqslant l \leqslant L-1 \quad \text{and} \quad \mathcal{A}^L(Z) = Z. \tag{2.2}$$

Depending on the nature of the problem, the output of the ANN may have to pass through an output function $\mathcal{S}$ to convert the signal into a meaningful form. In classification problems, a suitable choice for such an output function would be the *softmax* function

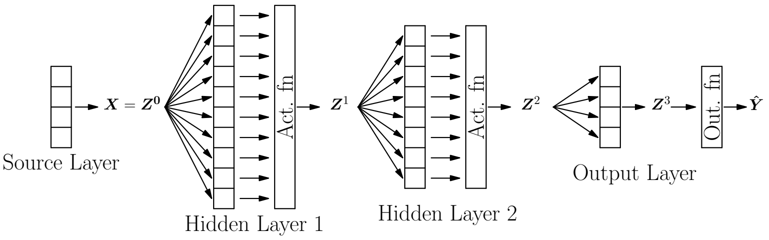


Figure 1. An MLP with 2 hidden layers. The source layer transmits the signal X to the first hidden layer. The final output of the network is $\hat{Y}$.

$$\mathcal{S}(x) : \mathbb{R}^n \to \mathbb{R}^n : x \mapsto \left(\frac{e^{x_1}}{\sum_{j=1}^n e^{x_j}}, \ldots, \frac{e^{x_n}}{\sum_{j=1}^n e^{x_j}} \right). \tag{2.3}$$

This choice ensures that the final output vector $\hat{Y} = \mathcal{S}(Z^L)$ satisfies $\sum_{j=1}^n \hat{Y}_j = 1$ and $0 \leqslant \hat{Y}_j \leqslant 1$ for all $1 \leqslant j \leqslant n$, which allows $\hat{Y}_j$ to be viewed as the probability that the input Z^0 belongs to the j-th class. Note that the class predicted by the network is $\arg\max_j \hat{Y}_j$. For regression problems, no additional output function is needed.

Remark 2.1. *It is possible that multiple classes have the largest probability. In this case, the predicted class can be uniquely defined as* $\min \arg\max_j \{\hat{Y}_j\}$*, following the usual coding conventions. Also note that the softmax function only contributes towards the interpretability of the network output and has no effect on the predicted class, that is,*

$$\min \arg\max_j \{\hat{Y}_j\} = \min \arg\max_j \{Z_j^L\}.$$

This observation will be used at a later stage.

The expressive power of ReLU neural networks, in particular their capability of approximating rough functions, has already been demonstrated in literature [22, 27]. In practice however, there is a major issue in this approach when used to approximate an unknown function $f : D \subseteq \mathbb{R} \to \mathbb{R}$ based on a finite set $\mathbb{S} \subset \{(x, f(x)) : x \in D\}$, as for each individual function f a new neural network has to be found. This network (or an approximation) is found by the process of *training* the network. The computational cost is significantly higher than that of other classical regression methods in low dimensions, which makes this approach rather impractical, at least for functions in low dimensions. This motivates us to investigate how one can obtain a neural network that takes input in D and produces an output interpolant, or rather its evaluation at certain sample points. Such a network primarily depends only on the training data $\mathbb{S}$ and can be reused for each individual function, thereby drastically reducing the computational cost. Instead of creating an entirely novel data-dependent interpolation procedure, we base ourselves in this paper on the essentially non-oscillatory (ENO) interpolation framework of [16, 15], which we introduce in the next section.

3. ENO framework for interpolating rough functions

In this section, we explore the *essentially non-oscillatory* (ENO) interpolation framework [16, 15], on which we will base our theoretical results of later sections. Although the ENO procedure has its origins in the context of the numerical approximation of solutions of hyperbolic conservation laws, this data-dependent scheme has also proven its use for the interpolation of rough functions [23].

3.1. **ENO interpolation.**

We first focus on the original ENO procedure, as introduced in [16]. This procedure can attain any order of accuracy for smooth functions, but reduces to first-order accuracy for functions that are merely Lipschitz continuous. In particular, the ENO-p interpolant is p-th order accurate in smooth regions and suppresses the appearance of spurious oscillations in the vicinity of points of discontinuity. In the following, we describe the main idea behind this algorithm.

Let f be a function on $\Omega = [c, d] \subset \mathbb{R}$ that is at least p times continuously differentiable. We define a sequence of nested uniform grids $\{\mathcal{T}^k\}_{k=0}^K$ on Ω, where

$$\mathcal{T}^k = \{x_i^k\}_{i=0}^{N_k},\ I_i^k = [x_{i-1}^k, x_i^k],\ x_i^k = c + ih_k,\ h_k = \frac{(d-c)}{N_k},\ N_k = 2^k N_0, \tag{3.1}$$

for $0 \leqslant i \leqslant N_k$, $0 \leqslant k \leqslant K$ and some positive integer N_0. Furthermore we define $f^k = \{f(x) : x \in \mathcal{T}^k\}$, $f_i^k = f(x_i^k)$ and we let $f_{-p+2}^k, ..., f_{-1}^k$ and $f_{N_k+1}^k, ...f_{N_k+p-2}^k$

be suitably prescribed ghost values. We are interested in finding an interpolation operator $\mathcal{I}^{h_k}$ such that

$$\mathcal{I}^{h_k} f(x) = f(x) \text{ for } x \in \mathcal{T}^k \quad \text{and} \quad \left\|\mathcal{I}^{h_k} f - f\right\|_\infty = O(h_k^p) \text{ for } k \to \infty.$$

In standard approximation theory, this is achieved by defining $\mathcal{I}^{h_k} f$ on I_i^k as the unique polynomial p_i^k of degree $p-1$ that agrees with f on a chosen set of p points, including x_{i-1}^k and x_i^k. The linear interpolant ($p=2$) can be uniquely obtained using the stencil $\{x_{i-1}^k, x_i^k\}$. However, there are several candidate stencils to choose from when $p > 2$. The ENO interpolation procedure considers the stencil sets

$$\mathcal{S}_i^r = \{x_{i-1-r+j}^k\}_{j=0}^{p-1}, \quad 0 \leqslant r \leqslant p-2,$$

where r is called the (left) stencil shift. The smoothest stencil is then selected based on the local smoothness of f using Newton's undivided differences. These are inductively defined in the following way. Let $\Delta_j^0 = f_{i+j}^k$ for $-p+1 \leqslant j \leqslant p-2$ and $0 \leqslant i \leqslant N_k$. We can then define

$$\Delta_j^s = \begin{cases} \Delta_j^{s-1} - \Delta_{j-1}^{s-1} & \text{for } s \text{ odd} \\ \Delta_{j+1}^{s-1} - \Delta_j^{s-1} & \text{for } s \text{ even.} \end{cases}$$

Algorithm 1 describes how the stencil shift r can be obtained using these undivided differences. Note that r uniquely defines the polynomial p_i^k. We can then write the final interpolant as

$$\mathcal{I}^{h_k} f(x) = \sum_{i=1}^{N_k} p_i^k(x) \mathbb{1}_{[x_{i-1}^k, x_i^k)}(x).$$

This interpolant can be proven to be *total-variation bounded* (TVB), which guarantees the disappearance of spurious oscillations (e.g. near discontinuities) when the grid is refined. This property motivates the use of the ENO framework over standard techniques for the interpolation of rough functions.

In many applications, one is only interested in predicting the values of f^{k+1} given f^k. In this case, there is no need to calculate $\mathcal{I}^{h_k} f$ and evaluate it on $\mathcal{T}^{k+1}$. Instead, one can use Lagrangian interpolation theory to see that there exist fixed coefficients $C_{r,j}^p$ such that

$$\begin{aligned} \mathcal{I}^{h_k} f(x_{2i-1}^{k+1}) &= \sum_{j=0}^{p-1} C_{r_i^k, j}^p f_{i-r_i^k+j}^k \text{ for } 1 \leqslant i \leqslant N_k \quad \text{and} \\ \mathcal{I}^{h_k} f(x_{2i}^{k+1}) &= f_{2i}^{k+1} = f_i^k \text{ for } 0 \leqslant i \leqslant N_k, \end{aligned} \tag{3.2}$$

where r_i^k is the stencil shift corresponding to the smoothest stencil for interval I_i^k. The coefficients $C_{r,j}^p$ are listed in Table 5 in Appendix B.

Remark 3.1. *ENO was initially introduced by* [16] *for high-order accurate piecewise polynomial reconstruction, given cell averages of a function. This allows the development of high-order accurate numerical methods for hyperbolic conservation laws, the so-called ENO schemes. ENO reconstruction can be loosely interpreted as ENO interpolation applied to the primitive function and is discussed in Appendix A.*

Remark 3.2. *The prediction of f^{k+1} from f^k can be framed in the context of multi-resolution representations of functions, which are useful for data compression* [17]*. As we will use ENO interpolation for data compression in Section 6, we refer to Appendix C for details on multi-resolution representations.*

Algorithm 1: ENO interpolation stencil selection

```
**Input:** ENO order $p$, input array $\mathbf{\Delta^0} = \{f^k_{i+j}\}^{p-2}_{j=-p+1}$, for any $0 \leqslant i \leqslant N_k$.
**Output:** Stencil shift $r$.
*Evaluate Newton undivided differences:*
for $j = 1$ to $p - 1$ do
    $\mathbf{\Delta^j} = \mathbf{\Delta^{j-1}}[2 : \text{end}] - \mathbf{\Delta^{j-1}}[1 : \text{end} - 1]$
*Find shift:*
$r = 0$
for $j = 2$ to $p - 1$ do
    if $|\mathbf{\Delta^j}[p - 2 - r]| < |\mathbf{\Delta^j}[p - 1 - r]|$ then
        $r = r + 1$
return $r$
```

3.2. **An adapted second-order ENO-SR algorithm.** Even though ENO is able to interpolate rough functions without undesirable side effects (e.g. oscillations near discontinuities), there is still room for improvement. By itself, the ENO interpolation procedure degrades to first-order accuracy for piecewise smooth functions i.e, functions with a singularity in the second derivative. However, following [15], one can use *sub-cell resolution* (SR), together with ENO interpolation, to obtain a second-order accurate approximation of such functions. We propose a simplified variant of the ENO-SR procedure from [1] and prove that it is still second-order accurate. In the following, we assume f to be a continuous function that is two times differentiable except at a single point z where the first derivative has a jump of size $[f'] = f'(z+) - f'(z-)$. We use the notation introduced in Section 3.1.

The first step of the adapted second-order ENO-SR algorithm is to label intervals that might contain the singular point z as *bad* (B), other intervals get the label *good* (G). We use second-order differences

$$\Delta^2_h f(x) := f(x-h) - 2f(x) + f(x+h) \tag{3.3}$$

as smoothness indicators. The rules of the ENO-SR detection mechanism are the following:

(1) The intervals I^k_{i-1} and I^k_i are labelled B if

$$|\Delta^2_{h_k} f(x^k_{i-1})| > \max_{n=1,2,3} |\Delta^2_{h_k} f(x^k_{i-1\pm n})|.$$

(2) Interval I^k_i is labelled B if

$$|\Delta^2_{h_k} f(x^k_i)| > \max_{n=1,2} |\Delta^2_{h_k} f(x^k_{i+n})| \quad \text{and} \quad |\Delta^2_{h_k} f(x^k_{i-1})| > \max_{n=1,2} |\Delta^2_{h_k} f(x^k_{i-1-n})|.$$

(3) All other intervals are labelled G.

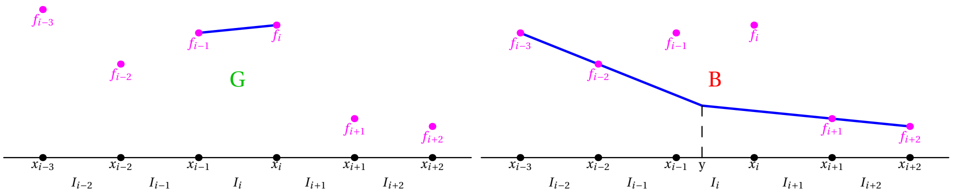


FIGURE 2. Visualization of the second-order ENO-SR algorithm in the case where an interval is labelled as good (left) and bad (right). The superscript k was omitted for clarity.

Note that neither detection rule implies the other and that an interval can be labelled B by both rules at the same time. In the following, we will denote by $p_i^k : [c,d] \to \mathbb{R}$ the linear interpolation of the endpoints of I_i^k. The rules of the interpolation procedure are stated below, a visualization of the algorithm can be found in Figure 2.

(1) If I_i^k was labelled as G, then we take the linear interpolation on this interval as approximation for f,
$$\mathcal{I}_i^{h_k} f(x) = p_i^k(x).$$

(2) If I_i^k was labelled as B, we use p_{i-2}^k and p_{i+2}^k to predict the location of the singularity. If both lines intersect at a single point y, then we define
$$\mathcal{I}_i^{h_k} f(x) = p_{i-2}^k(x) \mathbb{1}_{[c,\max\{y,c\})}(x) + p_{i+2}^k(x) \mathbb{1}_{[\min\{y,d\},d]}(x).$$
The relation between this intersection point y and the singularity z is quantified by Lemma D.3. If the two lines do not intersect, we treat I_i^k as a good interval and let $\mathcal{I}_i^{h_k} f(x) = p_i^k(x)$.

The theorem below states that our adaptation of ENO-SR is indeed second-order accurate.

Theorem 3.3. *Let f be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. The adapted ENO-SR interpolant $\mathcal{I}^h f$ satisfies*
$$\left\| f - \mathcal{I}^h f \right\|_\infty \leqslant Ch^2 \sup_{\mathbb{R}\backslash\{z\}} |f''|$$
for all $h > 0$, with $C > 0$ independent of f.

Proof. The proof is an adaptation of the proof of Theorem 1 in [1] and can be found in Appendix D. □

4. ENO as a ReLU DNN

As mentioned in the introduction, we aim to recast the ENO interpolation algorithm from Section 3.1 as a ReLU DNN. Our first approach to this end begins by noticing that the crucial step of the ENO procedure is determining the correct stencil shift. Given the stencil shift, the retrieval of the ENO interpolant is straightforward. ENO-p can therefore be interpreted as a classification problem, with the goal of mapping an input vector (the evaluation of a certain function on a number of points) to one of the $p-1$ classes (the stencil shifts). We now present one of the main results of this paper. The following theorem states that the stencil selection of p-th order ENO interpolation can be exactly obtained by a ReLU DNN for every order p. The stencil shift can be obtained from the network output by using the default output function for classification problems (cf. Remark 2.1).

Theorem 4.1. *There exists a ReLU neural network consisting of $p + \left\lceil \log_2 \binom{p-2}{\lfloor \frac{p-2}{2} \rfloor} \right\rceil$ hidden layers, that takes input $\mathbf{\Delta^0} = \{f_{i+j}^k\}_{j=-p+1}^{p-2}$ and leads to exactly the same stencil shift as the one obtained by Algorithm 1.*

Proof. We first sketch an intuitive argument why there exists a ReLU DNN that, after applying min arg max, leads to the ENO-p stencil shift. Algorithm 1 maps every input stencil $\mathbf{\Delta^0} \in [c,d]^{2p-2}$ to a certain stencil shift r. A more careful look at the algorithm reveals that the input space $[c,d]^{2p-2}$ can be partioned into polytopes such that the interior of every polytope is mapped to one of the $p-1$ possible stencil shifts. Given that every ReLU DNN is a continuous, piecewise affine function (e.g. [4]), one can construct for every $i \in \{0,\ldots,p-2\}$ a ReLU DNN

$\phi_i : [c,d]^{2p-2} \to \mathbb{R}$ that is equal to 0 on the interior of every polytope corresponding to stencil shift i and that is strictly smaller than 0 on the interior of every polytope not corresponding to stencil shift i. It is then clear that

$$\min \arg\max\{\phi_0, \dots, \phi_{p-2}\} - 1 \tag{4.1}$$

corresponds to the ENO stencil shift on the interiors of all polytopes. Thanks to the minimum in (4.1), it also corresponds to the unique stencil shift from Algorithm 3.2 on the faces of the polytopes, where multiple ϕ_i are equal to zero. The claim then follows from the fact that the mapping $\mathbf{\Delta^0} \mapsto (\phi_0(\mathbf{\Delta^0}), \dots, \phi_{p-2}(\mathbf{\Delta^0}))$ can be written as a ReLU DNN.

In what follows, we present a more constructive proof that sheds light on the architecture that is needed to represent ENO and the sparsity of the corresponding weights. In addition, a technique to replace the Heaviside function (as in Algorithm 1) is used. Recall that we look for the ENO stencil shift $r := r_i^k$ corresponding to the interval I_i^k. Let $k \in \mathbb{N}$ and define $\Delta_j^0 = f_{i+j}^k$ for $-p+1 \leqslant j \leqslant p-2$ and $0 \leqslant i \leqslant N_k$, where $f_{-p+1}^k, \dots, f_{-1}^k$ and $f_{N_k+1}^k, \dots, f_{N_k+p-2}^k$ are suitably defined ghost values. Following Section 3.1, we define $\Delta_j^s = \Delta_j^{s-1} - \Delta_{j-1}^{s-1}$ for s odd and $\Delta_j^s = \Delta_{j+1}^{s-1} - \Delta_j^{s-1}$ for s even, and with $\mathbf{\Delta^s}$ we denote the vector consisting of all Δ_j^s for all applicable j. In what follows, we use Y^l and Z^l to denote the values of the l-th layer of the neural network before and after activation, respectively. We use the notation X^l for an auxiliary vector needed to calculate Y^l.

Step 1. Take the input to the network to be

$$Z^0 = [\Delta_{-p+1}^0, \dots, \Delta_{p-2}^0] \in \mathbb{R}^{2(p-1)}.$$

These are all the candidate function values considered in Algorithm 1.

Step 2. We want to obtain all quantities Δ_j^s that are compared in Algorithm 1, as shown in Figure 3. We therefore choose the first layer (before activation) to be

$$Y^1 = \begin{bmatrix} Y_\Delta \\ -Y_\Delta \end{bmatrix} \in \mathbb{R}^{2M} \quad \text{where} \quad Y_\Delta = \begin{bmatrix} \Delta_0^2 \\ \Delta_{-1}^2 \\ \vdots \end{bmatrix} \in \mathbb{R}^M$$

is the vector of all the terms compared in Algorithm 1 and $M = \frac{p(p-1)}{2} - 1$. Note that every undivided difference is a linear combination of the network input. Therefore one can obtain Y^1 from Z^0 by taking a null bias vector and weight matrix $W^1 \in \mathbb{R}^{2M \times (2p-2)}$. After applying the ReLU activation function, we obtain

$$Z^1 = \begin{bmatrix} (Y_\Delta)_+ \\ (-Y_\Delta)_+ \end{bmatrix}.$$

Step 3. We next construct a vector $X^2 \in \mathbb{R}^L$, where $L = \frac{(p-2)(p-1)}{2}$, that contains all the quantities of the if-statement in Algorithm 1. This is ensured by setting,

$$X^2 = \begin{bmatrix} |\Delta_{-1}^2| - |\Delta_0^2| \\ |\Delta_0^3| - |\Delta_1^3| \\ |\Delta_{-1}^3| - |\Delta_0^3| \\ \vdots \end{bmatrix}.$$

Keeping in mind that $|a| = (a)_+ + (-a)_+$ for $a \in \mathbb{R}$ we see that there is a matrix $\widetilde{W}^2 \in \mathbb{R}^{L \times 2M}$ such that $X^2 = \widetilde{W}^2 Z^1$. We wish to quantify for each component of X^2 whether it is strictly negative or not (cf. the if-statement of Algorithm 1). For

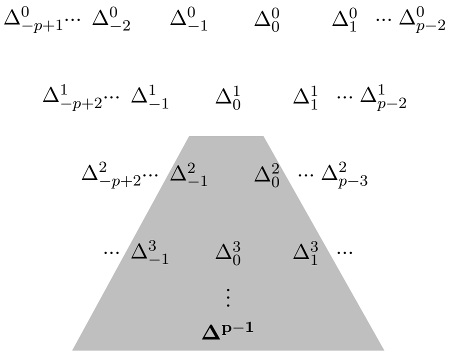

$\Delta^0_{-p+1} \cdots \Delta^0_{-2}$
Δ^0_{-1}
Δ^0_0
$\Delta^0_1 \cdots \Delta^0_{p-2}$
$\Delta^1_{-p+2} \cdots \Delta^1_{-1}$
Δ^1_0
$\Delta^1_1 \cdots \Delta^1_{p-2}$
$\Delta^2_{-p+2} \cdots \Delta^2_{-1}$
Δ^2_0
$\cdots \Delta^2_{p-3}$
$\cdots \Delta^3_{-1}$
Δ^3_0
$\Delta^3_1 \cdots$
$\vdots$
$\boldsymbol{\Delta}^{\mathbf{p-1}}$

FIGURE 3. Only undivided differences in the shaded region are compared in Algorithm 1.

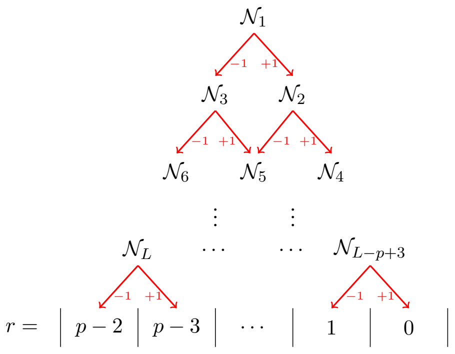


FIGURE 4. Arrangement of $N_1, \dots, N_L$ into directed acyclic graph.

this reason, we define the functions $H_1 : \mathbb{R} \to \mathbb{R}$ and $H_2 : \mathbb{R} \to \mathbb{R}$ by

$$H_1(x) = \begin{cases} 0 & x \leqslant -1 \\ x+1 & -1 < x < 0 \\ 1 & x \geqslant 0 \end{cases} \quad \text{and} \quad H_2(x) = \begin{cases} -1 & x \leqslant 0 \\ x-1 & 0 < x < 1 \\ 0 & x \geqslant 1 \end{cases}.$$

The key property of these functions is that H_1 and H_2 agree with the Heaviside function on $x > 0$ and $x < 0$, respectively. When $x = 0$ the output is respectively $+1$ or -1. Now note that $H_1(x) = (x+1)_+ - (x)_+$ and $H_2(x) = (x)_+ - (x-1)_+ - 1$. This motivates us to define

$$Y^2 = \begin{bmatrix} X^2 + 1 \\ X^2 \\ X^2 - 1 \end{bmatrix} \in \mathbb{R}^{3L},$$

which can be obtained from Z^1 by taking weight matrix $W^2 \in \mathbb{R}^{3L \times 2M}$ and bias vector $b^2 \in \mathbb{R}^{3L}$,

$$W^2 = \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \otimes \mathbb{I}_L \right) \cdot \widetilde{W}^2 \quad \text{and} \quad b^2_j = \begin{cases} 1 & 1 \leqslant j \leqslant L \\ 0 & L+1 \leqslant j \leqslant 2L \\ -1 & 2L+1 \leqslant j \leqslant 3L \end{cases}$$

where $\mathbb{I}_L$ denotes the $L \times L$ unit matrix. After activation we obtain $Z^2 = (Y^2)_+ = (W^2 Z^1 + b^2)_+$.

Step 4. We first define $X^3 \in \mathbb{R}^{2L}$ by

$$X^3_j = \begin{cases} H_1(X^2_j) = Z^2_j - Z^2_{L+j} & 1 \leqslant j \leqslant L \\ H_2(X^2_{j-L}) = Z^2_j - Z^2_{L+j} - 1 & L+1 \leqslant j \leqslant 2L. \end{cases}$$

This is clearly for every j an affine transformation of the entries of Z^2. For this reason there exist a matrix $\widetilde{W}^3 \in \mathbb{R}^{2L \times 3L}$ and a bias vector $\tilde{b}^3 \in \mathbb{R}^{2L}$ such that $X^3 = \widetilde{W}^3 Z^2 + \tilde{b}^3$. In order to visualize the next steps, we arrange the elements of X^3 in a triangular directed acyclic graph, shown in Figure 4, where every node $\mathcal{N}_j$ corresponds to the tuple $(X^3_j, X^3_{j+L}) = (H_1(X^2_j), H_2(X^2_j))$. We note that this tuple is either of the form $(+1, H_2(X^2_j))$ or $(H_1(X^2_j), -1)$. Algorithm 1 is equivalent to finding a path from the top node to one of the bins on the bottom. Starting from $\mathcal{N}_1$, we move to the closest element to the right in the row below (i.e. $\mathcal{N}_2$) if $\mathcal{N}_1$ is of the form $(+1, H_2(X^2_j))$. If $\mathcal{N}_1$ is of the form $(H_1(X^2_j), -1)$, we move to the closest element to the left in the row below (i.e. $\mathcal{N}_3$). If $\mathcal{N}_1$ is of the form $(+1,-1)$, then it is not important in which direction we move. Both paths lead to a suitable ENO stencil shift. Repeating the same procedure at each row, one ends up in one of the $p-1$ bins at the bottom representing the stencil shift r.

There are 2^{p-2} paths from the top to one of the bins at the bottom. In order to represent the path using a $(p-2)$-tuple of entries of X^3, one needs to choose between $H_1(X_j^2)$ and $H_2(X_j^2)$ at every node of the path, leading to 2^{p-2} variants of each path. At least one of these variants only takes the values $+1$ and -1 on the nodes and is identical to the path described above; this is the variant we wish to select. Counting all variants, the total number of paths is 2^{2p-4}.

Consider a path $\mathcal{P} = (X^3_{j_1}, \dots, X^3_{j_{p-2}})$ that leads to bin r. We define for this path a weight vector $W \in \{-1,0,1\}^{2L}$ whose elements are set as

$$W_j = \begin{cases} +1 & \text{if } X_j^3 = +1 \text{ and } j = j_s \text{ for some } 1 \leqslant s \leqslant p-2 \\ -1 & \text{if } X_j^3 = -1 \text{ and } j = j_s \text{ for some } 1 \leqslant s \leqslant p-2 \\ 0 & \text{otherwise.} \end{cases}$$

For this particular weight vector and for any possible $X^3 \in \mathbb{R}^{2L}$ we have $W \cdot X^3 \leqslant p-2$, with equality achieved if and only if the entries of X^3 appearing in $\mathcal{P}$ are assigned the precise values used to construct W. One can construct such a weight vector for each of the 2^{2p-4} paths. We next construct the weight matrix $\widehat{W}^3 \in \mathbb{R}^{2^{2p-4}\times 2L}$ in such a way that the first $2^{p-2}\cdot\binom{p-2}{0}$ rows correspond to the weight vectors for paths reaching $r=0$, the next $2^{p-2}\cdot\binom{p-2}{1}$ for paths reaching $r=1$ et cetera. We also construct the bias vector $\hat{b}^3 \in \mathbb{R}^{2^{2p-4}}$ by setting each element to $p-2$ and we define $\hat{X}^3 = \widehat{W}^3 X^3 + \hat{b}^3 = \widehat{W}^3(\widetilde{W}^3 Z^2 + \tilde{b}^3) + \hat{b}^3$. By construction, $\hat{X}^3_j = 2p-4$ if and only if path j corresponds to a suitable ENO stencil shift, otherwise $0 \leqslant \hat{X}^3_j < 2p-4$.

Step 5. Finally we define the final output vector by taking the maximum of all components of $\hat{X}^3$ that correspond to the same bin,

$$\hat{Y}_j = \max\left\{\hat{X}^3\left(1 + 2^{p-2}\cdot\sum_{k=0}^{j-2}\binom{p-2}{k}\right), \dots, \hat{X}^3\left(2^{p-2}\cdot\sum_{k=0}^{j-1}\binom{p-2}{k}\right)\right\},$$

for $j = 1,\dots,p-1$ and where $\hat{X}^3(j) := \hat{X}^3_j$. Note that $\hat{Y}_j$ is the maximum of $2^{p-2}\cdot\binom{p-2}{j-1}$ real positive numbers. Using the observation that $\max\{a,b\} = (a)_+ + (b-a)_+$ for $a,b \geqslant 0$, one finds that the calculation of $\hat{Y}$ requires $p-2+\left\lceil \log_2 \binom{p-2}{\lfloor\frac{p-2}{2}\rfloor}\right\rceil$ additional hidden layers. By construction, it is true that $\hat{Y}_j = 2p-4$ if and only if the $(j-1)$-th bin is reached. Furthermore, $\hat{Y}_j < 2p-4$ if the $(j-1)$-th bin is not reached. The set of all suitable stencil shifts R and the unique stencil shift r from Algorithm 1 are then respectively given by

$$R = \operatorname{argmax}_j \hat{Y}_j - 1 \quad \text{and} \quad r = \min R = \min \operatorname{argmax}_j \hat{Y}_j - 1, \tag{4.2}$$

where for classification problems, min argmax is the default output function to obtain the class from the network output (see Remark 2.1). □

Remark 4.2. *The neural network constructed in the above theorem is local in the sense that for each cell, it provides a stencil shift. These local neural networks can be concatenated to form a single neural network that takes as its input, the vector f^k of sampled values and returns the vector of interpolated values that approximates f^{k+1}. The global neural network combines the output stencil shift of each local neural network with a simple linear mapping* (3.2).

Although the previous theorem provides a network architecture for every order p, the obtained networks are excessively large for small p. We therefore present alternative constructions for ENO interpolation of orders $p = 3, 4$.

Algorithm 1 for $p = 3$ can be exactly represented by the following ReLU network with a single hidden layer, whose input is given by $X = (\Delta^0_{-2}, \Delta^0_{-1}, \Delta^0_0, \Delta^0_1)^\top$. The first hidden layer is identical to the one described in the original proof of Theorem 4.1 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{4\times4}$,

$$W^1 = \begin{pmatrix} 0 & 1 & -2 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{pmatrix}, \quad b^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.3}$$

The weights and biases of the output layer are

$$W^2 = \begin{pmatrix} -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}, \quad b^2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{4.4}$$

The resulting network output is

$$\hat{Y} = \begin{pmatrix} |\Delta^2_{-1}| - |\Delta^2_0| \\ |\Delta^2_0| - |\Delta^2_{-1}| \end{pmatrix},$$

from which the ENO stencil shift can then be determined using (4.2).

For $p = 4$, Algorithm 1 can be represented by following ReLU network with 3 hidden layers, whose input is given by $X = (\Delta^0_{-3}, \Delta^0_{-2}, \Delta^0_{-1}, \Delta^0_0, \Delta^0_1, \Delta^0_2)^\top$. The first hidden layer is identical to the one described in the original proof of Theorem 4.1 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{10\times6}$,

$$W^1 = \begin{pmatrix} \widetilde{W}^1 \\ -\widetilde{W}^1 \end{pmatrix} \in \mathbb{R}^{10\times6} \quad \text{where} \quad \widetilde{W}^1 = \begin{pmatrix} 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -3 & 1 \\ 0 & -1 & 3 & -3 & 1 & 0 \\ -1 & 3 & -3 & 1 & 0 & 0 \end{pmatrix}. \tag{4.5}$$

The second hidden layer has a null bias vector and the weight matrix

$$W^2 = \begin{pmatrix} \widetilde{W}^2 \\ -\widetilde{W}^2 \end{pmatrix} \in \mathbb{R}^{6\times10} \quad \text{where} \quad \widetilde{W}^2 = \begin{pmatrix} 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}. \tag{4.6}$$

Note that $\widetilde{W}^2 \in \mathbb{R}^{3\times10}$ is as in the original proof of Theorem 4.1 for $p = 4$. The third hidden layer and the output layer both have a null bias vector and their weights are respectively given by,

$$W^3 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ -1 & 1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad W^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.7}$$

After an elementary, yet tedious case study, one can show that the shift can again be determined using (4.2).

Remark 4.3. *Similarly, one can show that the stencil selection algorithm for ENO reconstruction (Algorithm 2 in Appendix A) for $p = 2$ can be exactly represented by a ReLU DNN with one hidden layer of width 4. The input and output dimension are 3 and 2, respectively. For $p = 3$, Algorithm 2 can be shown to correspond to a ReLU DNN with three hidden layers of dimensions* $(10, 6, 4)$. *Input and output dimension are 5 and 4, respectively.*

Remark 4.4. *After having successfully recast the ENO stencil selection as a ReLU neural network, it is natural to investigate whether there exists a pure ReLU neural network (i.e. without additional output function) input f^k and output $(\mathcal{I}^{h_k} f)^{k+1}$, as in the setting of* (3.2) *in Section 3.1. Since ENO is a discontinuous procedure and a pure ReLU neural network is a continuous function, a network with such an output does not exist. It remains however interesting to investigate to which extent we can approximate ENO using ReLU neural networks. This is the topic of Section 4.4 of the thesis* [10]*, where it is shown that there exists a pure ReLU DNN that mimics ENO to some extent such that some of the desirable properties of ENO are preserved.*

5. ENO-SR as a ReLU DNN

The goal of this section is to recast the second-order ENO-SR procedure from Section 3.2 as a ReLU DNN, similar to what we did for ENO in Section 4. Just like ENO, the crucial step of ENO-SR is the stencil selection, allowing us to interpret ENO-SR as a classification problem. In this context, we prove the equivalent of Theorem 4.1 for ENO-SR-2. Afterwards, we interpret ENO-SR as a regression problem (cfr. Remark 4.4) and investigate whether we can cast ENO-SR-2 as a pure ReLU DNN, i.e. without additional output function. In the following, we assume f to be a continuous function that is two times differentiable except at a single point z where the first derivative has a jump of size $[f'] = f'(z+) - f'(z-)$.

5.1. ENO-SR-2 stencil selection as ReLU DNN.

We will now prove that a second-order accurate prediction of f^{k+1} can be obtained given f^k using a ReLU DNN, where we use notation as in Section 3.1. Equation (3.2) shows that we only need to calculate $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ for every $1 \leqslant i \leqslant N_k$. The proof we present can be directly generalized to interpolation at points other than the midpoints of the cells, e.g. retrieving cell boundary values for reconstruction purposes. From the ENO-SR interpolation procedure it is clear that for every i there exists $r_i^k \in \{-2, 0, 2\}$ such that $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) = p_{i+r_i^k}^k(x_{2i-1}^{k+1})$. Analogously to what was described in Section 4, this gives rise to a classification problem. Instead of considering the stencil shifts as the output classes of the network, one can also treat the different cases that are implicitly described in the ENO-SR interpolation procedure in Section 3.2 as classes. This enables us to construct a ReLU neural network such that the stencil shift r_i^k can be obtained from the network output by using the default output function for classification problems (cf. Section 3.1 and Remark 2.1).

Theorem 5.1. *There exists a ReLU neural network with input f^k that leads to output $(r_1^k, \dots, r_{N_k}^k)$ as defined above.*

Proof. Instead of explicitly constructing a ReLU DNN, we will prove that we can write the output vector as a composition of functions that can be written as pure ReLU DNNs with linear output functions. Such functions include the rectifier function, absolute value, maximum and the identity function. The network architecture of a possible realisation of the network of this proof can be found after the proof. Furthermore we will assume that the discontinuity is not located in the first four or last four intervals. This can be achieved by taking k large enough, or by introducing suitably prescribed ghost values. We also assume without loss of generality that $x_i^k = i$ for $0 \leqslant i \leqslant N_k$.

The input of the DNN will be the vector $X^0 \in \mathbb{R}^{N_k+1}$ with $X_{i+1}^0 = f(x_i^k)$ for all $0 \leqslant i \leqslant N_k$. Using a simple affine transformation, we can obtain $X^1 \in \mathbb{R}^{N_k-1}$ such that $X^1_i = \Delta^2_{h_k} f(x^k_i)$ for all $1 \leqslant i \leqslant N_k - 1$. We now define the following quantities,

$$M_i = \max_{n=1,2,3} |\Delta^2_{h_k} f(x^k_{i\pm n})| = \max_{n=1,2,3} |X^1_{i\pm n}|, \quad N^{\pm}_i = \max_{n=1,2} |\Delta^2_{h_k} f(x^k_{i\pm n})| = \max_{n=1,2} |X^1_{i\pm n}|, \tag{5.1}$$

where $4 \leqslant i \leqslant N_k - 4$. Next, we construct a vector $X^2 \in \mathbb{R}^{N_k}$ such that every entry corresponds to an interval. For $1 \leqslant i \leqslant N_k$, we want $X^2_i > 0$ if and only if the interval I^k_i is labelled as B by the adapted ENO-SR detection mechanism. We can achieve this by defining

$$X^2_i = (\min\{|X^1_i| - N^+_i, |X^1_{i-1}| - N^-_{i-1}\})_+ + (|X^1_i| - M_i)_+ + (|X^1_{i-1}| - M_{i-1})_+ \tag{5.2}$$

for $5 \leqslant i \leqslant N_k - 4$. Furthermore we set $X^2_1 = X^2_2 = X^2_3 = X^2_4 = X^2_{N_k-3} = X^2_{N_k-2} = X^2_{N_k-1} = X^2_{N_k} = 0$. Note that the first term of the sum will be strictly positive if I^k_i is labelled bad by the second rule of the detection mechanism and one of the other terms will be strictly positive if I^k_i is labelled bad by the first rule. Good intervals I^k_i have $X^2_i = 0$.

Now define $n_{i,l} = l + 4(i-1)$ for $1 \leqslant i \leqslant N_k$ and $1 \leqslant l \leqslant 4$. Using this notation, i refers to the interval I^k_i. We denote by $p^k_i : [c,d] \to \mathbb{R} : x \mapsto a_i x + b_i$ the linear interpolation of the endpoints of I^k_i, where we write a_i and b_i instead of a^k_i and b^k_i to simplify notation. Define $X^3 \in \mathbb{R}^{4N_k}$ in the following manner:

$$\begin{aligned} X^3_{n_{i,1}} &= X^2_i, & X^3_{n_{i,3}} &= \left(|b_{i-2} - b_{i+2}| - x^{k+1}_{2i-1}|a_{i-2} - a_{i+2}|\right)_+, \\ X^3_{n_{i,2}} &= |a_{i-2} - a_{i+2}|, & X^3_{n_{i,4}} &= \left(-|b_{i-2} - b_{i+2}| + x^{k+1}_{2i-1}|a_{i-2} - a_{i+2}|\right)_+, \end{aligned} \tag{5.3}$$

for $5 \leqslant i \leqslant N_k - 4$. We set $X^3_{n_{i,l}} = 0$ for $1 \leqslant l \leqslant 4$ and $1 \leqslant i \leqslant 4$ or $N_k - 3 \leqslant i \leqslant N_k$. We can now define the output $\hat{Y} \in \mathbb{R}^{N_k}$ of the ReLU neural network by

$$\hat{Y}_i = \min \operatorname{argmin}_{1 \leqslant l \leqslant 4} X^3_{n_{i,l}}. \tag{5.4}$$

where we used the notation $\hat{Y}_i$ for the predicted class instead of the network output to simplify notation. It remains to prove that r^k_i can be obtained from $\hat{Y}_i$. Note that $\hat{Y}_i = 1$ if and only if I^k_i was labelled G. Therefore $\hat{Y}_i = 1$ corresponds to $r^k_i = 0$. If $\hat{Y}_i = 2$, then I^k_i was labelled B and the interpolants p^k_{i-2} and p^k_{i+2} do not intersect, leading to $r^k_i = 0$ according to the interpolation procedure. Next, $\hat{Y}_i = 3, 4$ corresponds to the case where I^k_i was labelled B and the interpolants p^k_{i-2} and p^k_{i+2} do intersect. This intersection point is seen to be $y = \frac{b_{i+2} - b_{i-2}}{a_{i-2} - a_{i+2}}$. If $\hat{Y}_i = 3$, then x^{k+1}_{2i-1} is right of y and therefore $r^k_i = 2$. Analogously, $\hat{Y}_i = 4$ corresponds to $r^k_i = -2$, which concludes the proof.

□

Now that we have established that our adaptation of the second-order ENO-SR algorithm can be written as a ReLU DNN augmented with a discontinuous output function, we can present a possible architecture of a DNN that calculates the output $\hat{Y}_i$ from f^k. The network we present has five hidden layers, of which the widths vary from 6 to 20, and an output layer of 4 neurons. The network is visualized in Figure 5. We now give some more explanation about how each layer in Figure 5 can be calculated from the previous layer, where we use the same notation as in the proof of Theorem 4.1. In addition, we define and note that

$$\gamma_{i-2,i+2}(z) = |b_{i-2} - b_{i+2}| - z|a_{i-2} - a_{i+2}|, \tag{5.5a}$$

$$\max\{x, y\} = x + (y - x)_+. \tag{5.5b}$$

A.B. It is easy to see that all quantities of the first layer are linear combinations of the input neurons. ***C.*** Application of $|x| = (x)_+ + (-x)_+$ and definition (5.5a) on

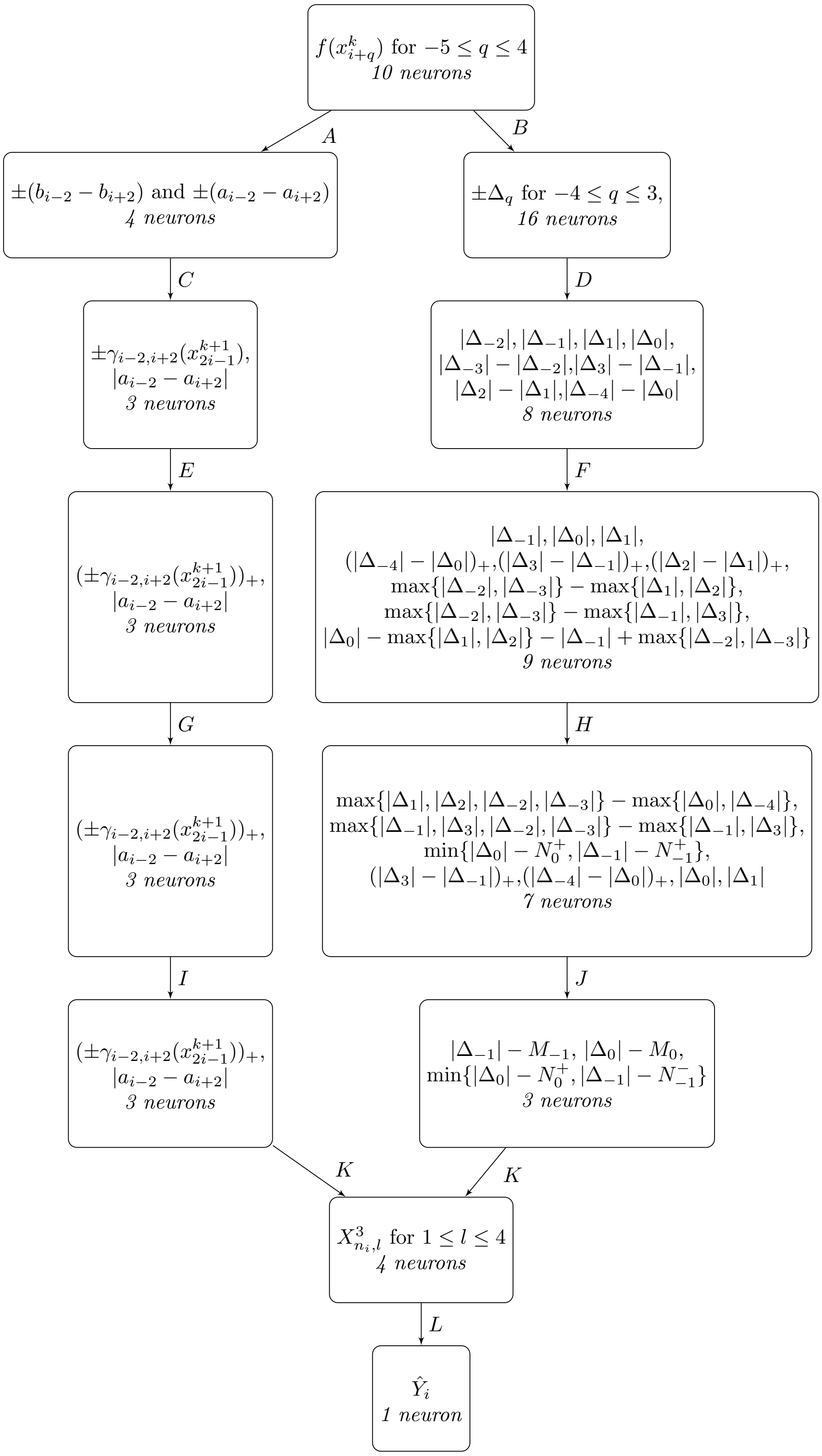

$f(x^k_{i+q})$ for $-5 \le q \le 4$
10 neurons
A
B
$\pm(b_{i-2} - b_{i+2})$ and $\pm(a_{i-2} - a_{i+2})$
4 neurons
$\pm\Delta_q$ for $-4 \le q \le 3$,
16 neurons
C
D
$\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1})$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-2}|, |\Delta_{-1}|, |\Delta_1|, |\Delta_0|$,
$|\Delta_{-3}| - |\Delta_{-2}|$,$|\Delta_3| - |\Delta_{-1}|$,
$|\Delta_2| - |\Delta_1|$,$|\Delta_{-4}| - |\Delta_0|$
8 neurons
E
F
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-1}|, |\Delta_0|, |\Delta_1|$,
$(|\Delta_{-4}| - |\Delta_0|)_+$,$(|\Delta_3| - |\Delta_{-1}|)_+$,$(|\Delta_2| - |\Delta_1|)_+$,
$\max\{|\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_1|, |\Delta_2|\}$,
$\max\{|\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_{-1}|, |\Delta_3|\}$,
$|\Delta_0| - \max\{|\Delta_1|, |\Delta_2|\} - |\Delta_{-1}| + \max\{|\Delta_{-2}|, |\Delta_{-3}|\}$
9 neurons
G
H
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$\max\{|\Delta_1|, |\Delta_2|, |\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_0|, |\Delta_{-4}|\}$,
$\max\{|\Delta_{-1}|, |\Delta_3|, |\Delta_{-2}|, |\Delta_{-3}|\} - \max\{|\Delta_{-1}|, |\Delta_3|\}$,
$\min\{|\Delta_0| - N^+_0, |\Delta_{-1}| - N^+_{-1}\}$,
$(|\Delta_3| - |\Delta_{-1}|)_+$,$(|\Delta_{-4}| - |\Delta_0|)_+$, $|\Delta_0|, |\Delta_1|$
7 neurons
I
J
$(\pm\gamma_{i-2,i+2}(x^{k+1}_{2i-1}))_+$,
$|a_{i-2} - a_{i+2}|$
3 neurons
$|\Delta_{-1}| - M_{-1}$, $|\Delta_0| - M_0$,
$\min\{|\Delta_0| - N^+_0, |\Delta_{-1}| - N^-_{-1}\}$
3 neurons
K
K
$X^3_{n_i,l}$ for $1 \le l \le 4$
4 neurons
L
$\hat{Y}_i$
1 neuron

FIGURE 5. Flowchart of a ReLU DNN to calculate $\hat{Y}_i$ from f^k.

$\pm(b_{i-2} - b_{i+2})$ and $\pm(a_{i-2} - a_{i+2})$. ***D.*** Straightforward application of the identity $|x| = (x)_+ + (-x)_+$ on $\pm\Delta_q$, followed by taking linear combinations. ***E.G.I.*** Passing by values. ***F.*** The first six quantities were passed by from the previous layer. The other ones are applications of equation (5.5b), where the order of the arguments of the maximums is carefully chosen. ***H.*** Equation (5.5b) was used, where we use that $\min\{x, y\} = -\max\{-x, -y\}$. ***J.*** Application of equation (5.1). ***K.*** The result follows from combining definitions (5.2) and (5.3). ***L.*** As can be seen in definition (5.4), $\hat{Y}_i$ is obtained by applying the output function min argmin on the output layer.

Remark 5.2. *The second-order ENO-SR method as proposed in* [1] *can also be written as a ReLU DNN, but it leads to a neural network that is considerably larger than the one presented above.*

5.2. **ENO-SR-2 regression as ReLU DNN.** After having successfully recast the ENO-SR stencil selection as a ReLU neural network, it is natural to investigate whether there exists a ReLU neural network with output $(\mathcal{I}^{h_k} f)^{k+1}$, as in the setting of (3.2) in Section 3.1. Since ENO-SR interpolation is a discontinuous procedure and a ReLU neural network is a continuous function, a network with such an output does not exist. It is however interesting to investigate to which extent we can approximate ENO-SR using ReLU neural networks. In what follows, we design an approximate ENO-SR method, based on the adapted ENO-SR-2 method of Section 3.2, and investigate its accuracy.

We first introduce for $\epsilon \geqslant 0$ the function $H_\epsilon : \mathbb{R} \to \mathbb{R}$, defined by

$$H_\epsilon(x) = \begin{cases} 0 & x \leqslant 0 \\ x/\epsilon & 0 < x \leqslant \epsilon \\ 1 & x > \epsilon. \end{cases} \tag{5.6}$$

Note that H_0 is nothing more than the Heaviside function. Using this function and the notation of the proof of Theorem 5.1, we can write down a single formula for $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$,

$$\begin{aligned} \mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) = (1-\alpha) p_i^k(x_{2i-1}^{k+1}) + \alpha\left((1-\beta) p_{i+2}^k(x_{2i-1}^{k+1}) + \beta p_{i-2}^k(x_{2i-1}^{k+1})\right), \\ \text{where } \alpha = H_0(\min\{X^3_{n_{i,1}}, X^3_{n_{i,2}}\}), \quad \beta = H_0(X^3_{n_{i,3}}), \end{aligned} \tag{5.7}$$

for $1 \leqslant i \leqslant N_k$. Observe that this formula cannot be calculated using a pure ReLU DNN. Nevertheless, we will base ourselves on this formula to introduce an approximate ENO-SR algorithm that can be exactly written as a ReLU DNN.

The first step is to replace H_0 by H_ϵ with $\epsilon > 0$, since

$$H_\epsilon(x) = \frac{1}{\epsilon}(x)_+ - \frac{1}{\epsilon}(x-\epsilon)_+, \tag{5.8}$$

which clearly can be calculated using a ReLU neural network. The now remaining issue is that the multiplication of two numbers cannot be exactly represented using a ReLU neural network. Moreover, as we aim for a network architecture that is independent of the accuracy of the final network that approximates ENO-SR-2, we cannot use the approximate multiplication networks in the sense of [26]. We therefore introduce an operation that resembles the multiplication of bounded numbers in another way. For $\lambda > 0$, we denote by $\star$ the operation on $[0,1] \times [-\lambda, \lambda]$ defined by

$$x \star y := (y + \lambda x - \lambda)_+ - (-y + \lambda x - \lambda)_+. \tag{5.9}$$

Like H_ϵ, this operation can be cast as a simple ReLU DNN. We compare $x \star y$ with $x \cdot y$ for fixed $x \in [0,1]$ and $\lambda > 0$ in Figure 6. Next, we list some properties of $\star$ that are of great importance for the construction of our approximation.

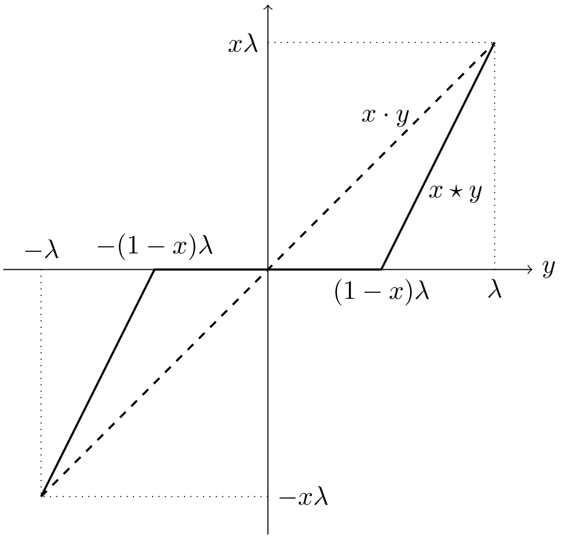


FIGURE 6. Plot of $x \star y$ and $x \cdot y$ for fixed $x \in [0,1]$ and $\lambda > 0$.

Lemma 5.3. *For $\lambda > 0$, the operation $\star$ satisfies the following properties:*

(1) *For all $x \in \{0,1\}$ and $y \in [-\lambda, \lambda]$ it holds true that $x \star y = xy$.*
(2) *For all $x \in [0,1]$ and $y \in [0,\lambda]$ we have $0 \leqslant x \star y \leqslant xy$.*
(3) *For all $x \in [0,1]$ and $y \in [-\lambda,0]$ we have $xy \leqslant x \star y \leqslant 0$.*
(4) *There exist $x \in [0,1]$ and $y_1, y_2 \in [-\lambda,\lambda]$ such that*

$$\min\{y_1, y_2\} \leqslant (1-x) \star y_1 + x \star y_2 \leqslant \max\{y_1, y_2\}$$

does not hold.

(5) *For all $x \in [0,1]$ and $y_1, y_2 \in [-\lambda,\lambda]$ it holds true that*

$$\min\{y_1, y_2\} \leqslant y_1 + x \star (y_2 - y_1) \leqslant \max\{y_1, y_2\}.$$

Proof. Properties 1,2 and 3 follow immediately from the definition and can also be verified on Figure 6. For property 4, note that $(1/2) \star (\lambda/2) + (1/2) \star (\lambda/2) = 0$. Property 5 is an application of properties 2 and 3. □

For the moment, we assume that there exists $\lambda > 0$ such that all quantities that we will need to multiply, lie in the interval $[-\lambda, \lambda]$. In view of the third property in Lemma 5.3, directly replacing all multiplications in (5.7) by the operation $\star$ will lead to a quantity that is no longer a convex combination of $p^k_{i-2}(x^{k+1}_{2i-1}), p^k_i(x^{k+1}_{2i-1})$ and $p^k_{i+2}(x^{k+1}_{2i-1})$. We therefore introduce the *approximate ENO-SR* prediction $\hat{f}^{k+1}_{i,\epsilon}$ of $f(x^{k+1}_i)$ by setting $\hat{f}^{k+1}_{2i,\epsilon} = f^{k+1}_{2i}$ for $0 \leqslant i \leqslant N_k$ and

$$\hat{f}^{k+1}_{2i-1,\epsilon} = p^k_i(x^{k+1}_{2i-1}) + \alpha \star \left(p^k_{i+2}(x^{k+1}_{2i-1}) - p^k_i(x^{k+1}_{2i-1}) + \beta \star \left(p^k_{i-2}(x^{k+1}_{2i-1}) - p^k_{i+2}(x^{k+1}_{2i-1})\right)\right), \quad \text{where } \alpha = H_\epsilon(\min\{X^3_{n_{i,1}}, X^3_{n_{i,2}}\}), \quad \beta = H_\epsilon(X^3_{n_{i,3}}), \tag{5.10}$$

for $1 \leqslant i \leqslant N_k$. The fourth property of Lemma 5.3 ensures that the two convex combinations in (5.7) are replaced by convex combinations (with possibly different weights). The theorem below quantifies the accuracy of the approximate ENO-SR predictions for $\epsilon > 0$.

Theorem 5.4. *Let $f : [c,d] \to [-1,1]$ be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a* *point z. For every k, the approximate ENO-SR predictions $\hat{f}^{k+1}_{i,\epsilon}$ satisfy for every $0 \leqslant i \leqslant N_{k+1}$ and $\epsilon \geqslant 0$ that*

$$|\mathcal{I}_i^{h_k} f(x^{k+1}_{2i-1}) - \hat{f}^{k+1}_{i,\epsilon}| \leqslant C h_k^2 \sup_{[c,d]\setminus\{z\}} |f''| + \frac{3}{2}\epsilon, \tag{5.11}$$

where $\mathcal{I}_i^{h_k} f(x^{k+1}_{2i-1})$ is the ENO-SR-2 prediction.

Proof. The proof can be found in Appendix E. □

We see that our approximation is second-order accurate up to an additional constant error, which can be made arbitrarily small. Finally, the following theorem states that the constructed approximation can indeed be represented by a ReLU DNN, i.e. there exists a pure ReLU DNN that satisfies bound (5.11).

Theorem 5.5. *Let $f : [c,d] \to [-1,1]$ be a globally continuous function with a bounded second derivative on $\mathbb{R}\setminus\{z\}$ and a discontinuity in the first derivative at a point z. For every $\epsilon > 0$, there exists a pure ReLU neural network with input $f^k \in [-1,1]^{N_k+1}$ and output $\hat{f}^{k+1}_{2i-1,\epsilon}$ for every $1 \leqslant i \leqslant N_k$, $0 \leqslant k \leqslant K$.*

Proof. Most of the work was already done in Theorem 5.1 and the discussion preceding Theorem 5.4. Indeed, we have already established that $p^k_{i-2}(x^{k+1}_{2i-1}), p^k_i(x^{k+1}_{2i-1})$, $p^k_{i+2}(x^{k+1}_{2i-1}), X^3_1, X^3_2$ and X^3_3, as well as the operation $\star$ can be represented using pure ReLU networks. It only remains to find a bound for all second arguments of the operation $\star$ in (5.10). Since the codomain of f is $[-1,1]$, one can calculate that $p^k_{i-2}(x^{k+1}_{2i-1}), p^k_i(x^{k+1}_{2i-1})$ and $p^k_{i+2}(x^{k+1}_{2i-1})$ lie in $[-4,4]$. Using Lemma 5.3, we then find that

$$p^k_{i+2}(x^{k+1}_{2i-1}) - p^k_i(x^{k+1}_{2i-1}) + \beta \star \left(p^k_{i-2}(x^{k+1}_{2i-1}) - p^k_{i+2}(x^{k+1}_{2i-1})\right) \in [-16, 16]$$

for all $\beta \in [0,1]$. We can thus use the operation $\star$ with $\lambda = 16$ in (5.9). □

We now present the network architecture of a ReLU neural network that computes the approximate ENO-SR prediction (5.10). The network we propose is visualized in Figure 7 and consists of eight hidden layers with widths 23, 13, 14, 12, 8, 7, 6 and 3. In the figure, the following notation was used,

$$\begin{aligned} m_i &= \min\{X^3_{n_i,1}, X^3_{n_i,2}\}, \\ P^k_{m,n} &= p^k_m(x^{k+1}_{2i-1}) - p^k_n(x^{k+1}_{2i-1}), \end{aligned} \tag{5.12}$$

for $1 \leqslant i, m, n \leqslant N_k$. We now give some more explanation about how all the layers can be calculated from the previous layer in Figure 7. ***A.B.*** All quantities of the first layer are linear combinations of the input neurons, where we also refer to Figure 5. From the proof of Theorem 5.5, it follows that we can take $\lambda = 16$. ***C.D.*** Linear combinations. ***E.*** We refer to (5.10) and (5.12) for the definitions of β and m_i, respectively. ***F.*** We refer to (5.9) and (5.10) for the definitions of $\star$ and α, respectively. ***G.*** From (5.10) it follows that the value of the output layer is indeed equal to the approximate second-order ENO-SR prediction $\hat{f}^{k+1}_{2i-1,\epsilon}$.

6. Numerical results

From sections 4 and 5, we know that there exist deep ReLU neural networks, of a specific architecture, that will mimic the ENO-p and the second-order ENO-SR-2 algorithms for interpolating rough functions. In this section, we investigate whether we can *train* such networks in practice and we also investigate their performance on a variety of tasks for which the ENO procedure is heavily used. We will refer to these trained networks as *DeLENO* (Deep Learning ENO) and *DeLENO-SR* networks. More details on the training procedure can be found in Section 6.1, the

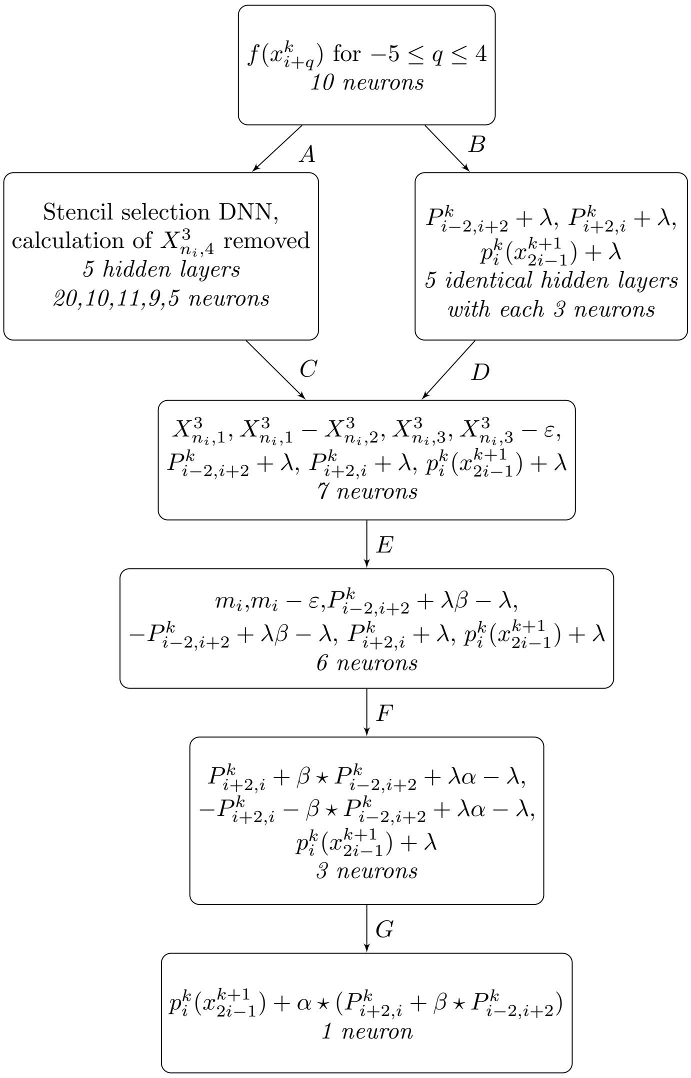

$f(x^k_{i+q})$ for $-5 \leq q \leq 4$
10 neurons
A
B
Stencil selection DNN, calculation of $X^3_{n_i,4}$ removed
5 hidden layers
20,10,11,9,5 neurons
$P^k_{i-2,i+2}+\lambda$, $P^k_{i+2,i}+\lambda$, $p^k_i(x^{k+1}_{2i-1})+\lambda$
5 identical hidden layers
with each 3 neurons
C
D
$X^3_{n_i,1}$, $X^3_{n_i,1}-X^3_{n_i,2}$, $X^3_{n_i,3}$, $X^3_{n_i,3}-\varepsilon$, $P^k_{i-2,i+2}+\lambda$, $P^k_{i+2,i}+\lambda$, $p^k_i(x^{k+1}_{2i-1})+\lambda$
7 neurons
E
m_i,$m_i-\varepsilon$,$P^k_{i-2,i+2}+\lambda\beta-\lambda$, $-P^k_{i-2,i+2}+\lambda\beta-\lambda$, $P^k_{i+2,i}+\lambda$, $p^k_i(x^{k+1}_{2i-1})+\lambda$
6 neurons
F
$P^k_{i+2,i}+\beta\star P^k_{i-2,i+2}+\lambda\alpha-\lambda$, $-P^k_{i+2,i}-\beta\star P^k_{i-2,i+2}+\lambda\alpha-\lambda$, $p^k_i(x^{k+1}_{2i-1})+\lambda$
3 neurons
G
$p^k_i(x^{k+1}_{2i-1})+\alpha\star(P^k_{i+2,i}+\beta\star P^k_{i-2,i+2})$
1 neuron

FIGURE 7. Flowchart of a ReLU DNN to calculate $\hat{f}^{k+1}_{2i-1,\epsilon}$ from f^k.

performance is discussed in Section 6.2 and illustrated by various applications at the end of this section.

6.1. **Training.** The *training* of these networks involves finding a parameter vector θ (the weights and biases of the network) that approximately minimizes a certain *loss function* $\mathcal{J}$ which measures the error in the network's predictions. To achieve this, we have access to a finite data set $\mathbb{S} = \{(X^i, \mathcal{L}(X^i))\}_i \subset D \times \mathcal{L}(D)$, where $\mathcal{L} : D \subset \mathbb{R}^m \to \mathbb{R}^n$ is the unknown function we try to approximate using a neural network $\mathcal{L}^\theta$.

For classification problems, each $Y^i = \mathcal{L}(X^i)$ is an n-tuple that indicates to which of the n classes X^i belongs. The output of the network $\hat{Y}^i = \mathcal{L}^\theta(X^i)$ is an approximation of Y^i in the sense that $\hat{Y}^i_j$ can be interpreted as the probability that X^i belongs to class j. A suitable loss function in this setting is the *cross-entropy* *function* with regularization term

$$\mathcal{J}(\theta; \mathbb{S}, \lambda) = -\frac{1}{\#\mathbb{S}} \sum_{(X^i,Y^i)\in\mathbb{S}} \sum_{j=1}^{n} Y^i_j \log(\hat{Y}^i_j) + \lambda\mathcal{R}(\theta). \tag{6.1}$$

The cross-entropy term measures the discrepancy between the probability distributions of the true outputs and the predictions. It is common to add a regularization term $\lambda\mathcal{R}(\theta)$ to prevent overfitting of the data and thus improve the generalization capabilities of the network [14]. The network hyperparameter $\lambda > 0$ controls the extent of regularization. Popular choices of $\mathcal{R}(\theta)$ include the sum of some norm of all the weights of the network. To monitor the generalization capability of the network, it is useful to split $\mathbb{S}$ into a training set $\mathbb{T}$ and a validation set $\mathbb{V}$ and minimize $\mathcal{J}(\theta; \mathbb{T}, \lambda)$ instead of $\mathcal{J}(\theta; \mathbb{S}, \lambda)$. The validation set $\mathbb{V}$ is used to evaluate the generalization error. The accuracy of network $\mathcal{L}^\theta$ on $\mathbb{T}$ is measured as

$$\mathbb{T}_{acc} = \#\left\{(X,Y) \in \mathbb{T} \mid \hat{Y} = \mathcal{L}^\theta(X), \quad \arg\max_{1\leqslant j\leqslant n} \hat{Y}_j = \arg\max_{1\leqslant j\leqslant n} Y_j\right\}/\#\mathbb{T}, \tag{6.2}$$

with a similar expression for $\mathbb{V}_{acc}$.

For regression problems, $\hat{Y}^i$ is a direct approximation of Y^i, making the *mean squared error* with regularization term

$$\mathcal{J}(\theta; \mathbb{S}, \lambda) = \frac{1}{\#\mathbb{S}} \sum_{(X^i,Y^i)\in\mathbb{S}} \left\| Y^i - \hat{Y}^i \right\|^2 + \lambda\mathcal{R}(\theta), \tag{6.3}$$

an appropriate loss function. As before, the data set $\mathbb{S}$ can be split into a training set $\mathbb{T}$ and a validation set $\mathbb{V}$, in order to minimize $\mathcal{J}(\theta; \mathbb{T}, \lambda)$ and estimate the MSE of the trained network by $\mathcal{J}(\theta; \mathbb{V}, \lambda)$.

The loss functions are minimized either with a mini-batch version of the stochastic gradient descent algorithm with an adpative learning rate or with the popular ADAM optimizer [19]. We use batch sizes of 1024, unless otherwise specified.

6.1.1. *Training DeLENO-p.* We want to construct a suitable training data set $\mathbb{S}$ to train DeLENO-p for interpolation purposes. Thanks to the results of Section 4, we are guaranteed that for certain architectures it is theoretically possible to achieve an accuracy of 100%. For any order, this architecture is given by Theorem 4.1 and its proof. For small orders $p = 3, 4$ we use the alternative network architectures described at the end of Section 4, as they are of smaller size. The network will take an input from $\mathbb{R}^m$, $m = 2p - 2$, and predicts the stencil shift r. We generate a data set $\mathbb{S}$ of size 460,200-200m using Algorithm 1 with inputs given by,

- A total of 400,000 samples $X \in \mathbb{R}^m$, with each component X_j randomly drawn from the uniform distribution on the interval $[-1, 1]$.
- The set

$$\{(u_l, ..., u_{l+m})^\top \mid 0 \leqslant l \leqslant N - m, \quad 0 \leqslant q \leqslant 39, \quad N = \{100, 200, 300, 400, 500\}\}$$

where u_l is defined as

$$u_l := \sin\left((q+1)\pi\frac{l}{N}\right), \quad 0 \leqslant l \leqslant N.$$

The input data needs to be appropriately scaled before being fed into the network, to ensure faster convergence during training. We use the following scaling for each input X,

$$\text{Scale}(X) = \begin{cases} \frac{2X-(b+a)}{b-a} & \text{if } X \neq 0 \\ (1, ..., 1)^\top \in \mathbb{R}^m & \text{otherwise} \end{cases}, \quad a = \min_j(X_j),\ b = \max_j(X_j), \tag{6.4}$$

which scales the input to lie in the box $[-1, 1]^m$.

Remark 6.1. *When the input data is scaled using formula* (6.4)*, then Newton's undivided differences are scaled by a factor* $2(b-a)^{-1}$ *as well. Therefore scaling does not alter the stencil shift obtained using Algorithm 1 or 2.*

The loss function $\mathcal{J}$ is chosen as (6.1), with an L_2 penalization of the network weights and $\lambda = 7.8 \cdot 10^{-6}$. The network is retrained using 5 times, with the weights and biases initialized using a random normal distribution for each of the retrainings. The last 20% of $\mathbb{S}$ is set aside to be used as the validation set $\mathbb{V}$. For each p, we denote by DeLENO-p the network with the highest accuracy $\mathbb{V}_{acc}$ at the end of the training. The training of the DeLENO reconstruction networks was performed entirely analogously, with the only difference that now we set $m = 2p - 1$.

6.1.2. *Training DeLENO-SR.* Next, we construct a suitable training data set $\mathbb{S}$ to train second-order DeLENO-SR for use as an interpolation algorithm. Recall that ENO-SR is designed to interpolate continuous functions f that are two times differentiable, except at isolated points, $z \in \mathbb{R}$ where the first derivative has a jump of size $[f']$. Locally, these functions can be viewed as piecewise linear functions. Based on this observation, we create a data set using functions of the form

$$f(x) = a(x-z)_- + b(x-z)_+, \tag{6.5}$$

where $a, b, z \in \mathbb{R}$. For notational simplicity we assume that the x-values of the stencil that serves as input for the ENO-SR algorithm (Section 3.2) are $0, 1, \ldots, 9$. The interval of interest is then $[4, 5]$ and the goal of ENO-SR is to find an approximation of f at $x = 4.5$. We generate 100,000 samples, where we choose a, b, z in the following manner,

- The parameters a and b are drawn from the uniform distribution on the interval $[-1, 1]$. Note that any interval that is symmetric around 0 could have been used, since the data will be scaled afterwards.
- For 25,000 samples, z is drawn from the uniform distribution on the interval $[4, 5]$. This simulates the case where the discontinuity is inside the interval of interest.
- For 75,000 samples, z is drawn from the uniform distribution on the interval $[-9, 9]$, which also includes the case in which f is smooth on the stencil.

The network architecture is described in Section 5. The network will take an input from $\mathbb{R}^{10}$ and predicts the stencil shift r. The training of DeLENO-SR was performed in a very similar fashion to the training of DeLENO-p (Section 6.1.1), only this time we retrained the DeLENO-SR network 5 times for 5000 epochs each. Furthermore we used 8-fold cross-validation on a data set of 20,000 samples to select the optimal regularization parameter, resulting in the choice $\lambda = 1 \cdot 10^{-8}$.

Remark 6.2. *Note that the detection mechanism of the ENO-SR interpolation method (Section 3.2) labels an interval as bad when* $\alpha - \beta > 0$ *for some numbers* $\alpha, \beta \in \mathbb{R}$*. This approach causes poor approximations in practice due to numerical errors. When for example* $\alpha = \beta$*, rounding can have as a consequence that* $\mathrm{round}(\alpha - \beta) > 0$*, leading to an incorrect label. This deteriorates the accuracy of the method and is very problematic for the training. Therefore we used in our code the alternative detection criterion* $\alpha - \beta > \epsilon$*, where for example* $\epsilon = 10^{-10}$*.*

6.2. **Performance.** In the previous sections, we have proven the existence of ReLU neural networks that approximate ENO(-SR) well, or can even exactly reproduce its output. However, it might be challenging to obtain these networks by training on a finite set of samples. Fortunately, Table 1 demonstrates that this is not the case for the DeLENO(-SR) stencil selection networks. For both interpolation and reconstruction, the classification accuracy (6.2) is nearly 100%. A comparison between the trained weights and biases in Appendix F and their theoretical counterparts of (4.3-4.7) reveals that there are multiple DNNs that can represent ENO. Moreover, this indicates that the weights of two DNNs (i.e. the theoretical and trained DNNs) can be very different even though the output is approximately the same (Table 1). This is in agreement with the result from [21] that the function that maps a family of weights to the function computed by the associated network is not inverse stable.

(a) DeLENO interpolation

p	hidden layer sizes	$\mathbb{T}_{acc}$	$\mathbb{V}_{acc}$
3	4	99.36%	99.32%
4	10,6,4	99.22%	99.14%
SR	20,11,12,10,6	99.74%	99.81%

(b) DeLENO reconstruction

p	hidden layer sizes	$\mathbb{T}_{acc}$	$\mathbb{V}_{acc}$
2	4	99.96%	99.97%
3	10, 6, 4	99.65%	99.65%

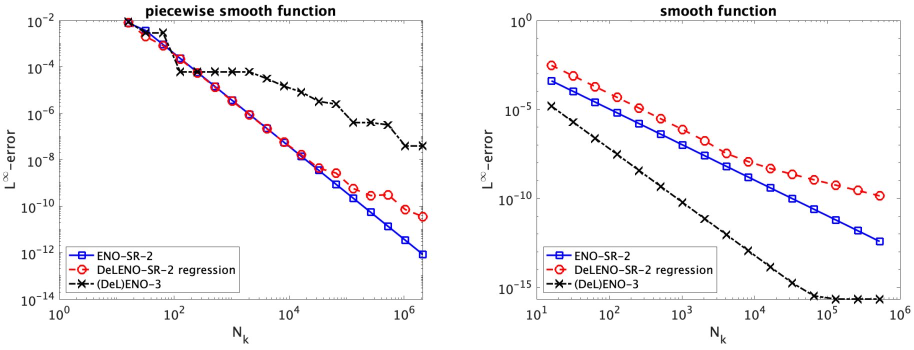
TABLE 1. Shape of DeLENO-p and DeLENO-SR networks with their accuracies for the interpolation and reconstruction problem.

Next, we investigate the order of accuracy of the DeLENO-SR regression network, for the functions

$$f_1(x) = -2\left(x - \frac{\pi}{6}\right) \mathbb{1}_{[0,\frac{\pi}{6})}(x) + \left(x - \frac{\pi}{6}\right)^2 \mathbb{1}_{[\frac{\pi}{6},1]}(x), \tag{6.6}$$

$$f_2(x) = \sin(x). \tag{6.7}$$

Note that the first derivative of f_1 has a jump at $\frac{\pi}{6}$. In Figure 8, the order of accuracy of second-order ENO-SR-2 and DeLENO-SR-2 is compared with those of ENO-3 and DeLENO-3 for both the piecewise smooth function f_1 and the smooth function f_2. In both cases, ENO-3 and DeLENO-3 completely agree, which is not



(a) Error for f_1

(b) Error for f_2

FIGURE 8. Plots of the approximation error of the DeLENO-SR-2 regression network for the piecewise smooth function f_1 and the smooth sine function f_2. The approximation errors of ENO-SR-2 and (DeL)ENO-3 are shown for comparison.

surprising given the high classification accuracies listed in Table 1. (DeL)ENO-3 is third-order accurate for the smooth function and only first-order accurate for a more rough function, in agreement with expectations. For both f_1 and f_2, DeLENO-SR-2 is second-order accurate on coarse grids, but a deterioration to first-order accuracy is seen on very fine grids. This deterioration is an unavoidable consequence of the error that the trained network makes and the linear rescaling of the input stencils (6.4). A more detailed discussion of this issue can be found in Section 6.2 of [10]. Furthermore, although the DeLENO-SR regression network is initially second-order accurate for the smooth function, the approximation error does not agree with that of ENO-SR. This is in line with Theorem 5.4, where we proved that there exists a network that is a second-order accurate approximation of ENO-SR-2 except for an error term that can be arbitrarily small, yet is fixed. A second factor that might contribute is the fact that DeLENO-SR-2 was trained on piecewise linear functions, which can be thought of as a second-order accurate approximation of a smooth function, therefore leading to a higher error.

6.3. **Applications.** Next, we apply the DeLENO algorithms in the following examples.

6.3.1. *Function approximation.* We first demonstrate the approximating ability of the DeLENO interpolation method using the function

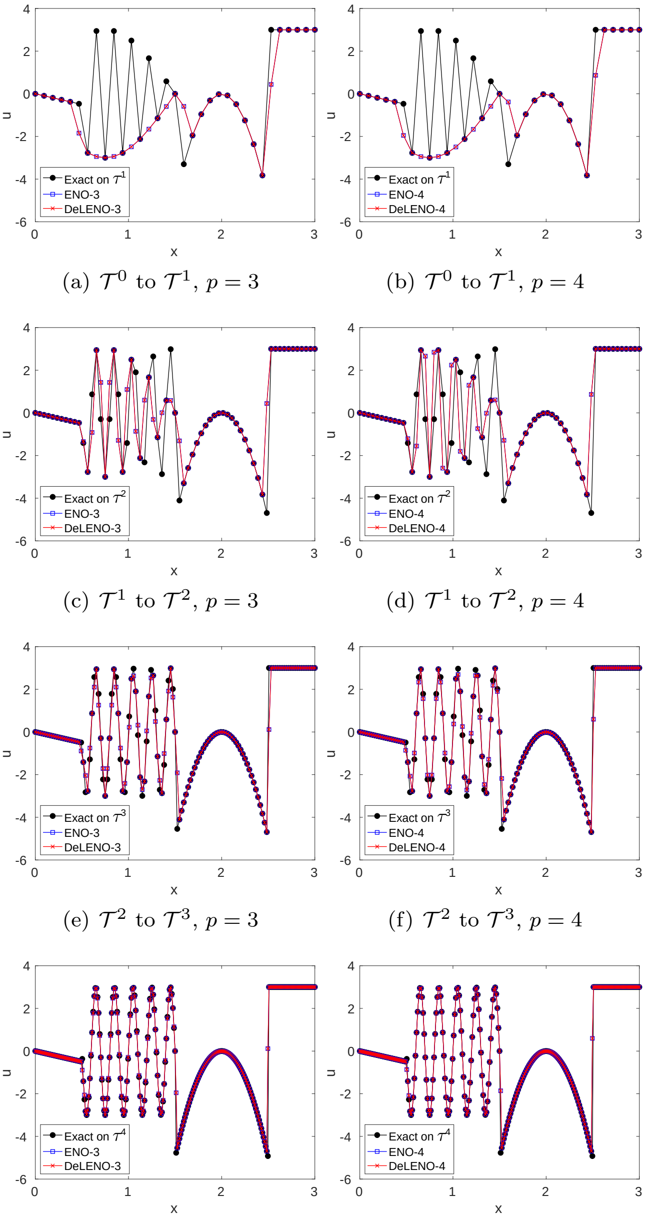
$$q(x) = \begin{cases} -x & \text{if } x < 0.5, \\ 3\sin(10\pi x) & \text{if } 0.5 < x < 1.5, \\ -20(x-2)^2 & \text{if } 1.5 < x < 2.5, \\ 3 & \text{if } 2.5 < x, \end{cases} \tag{6.8}$$

which consists of jump discontinuities and smooth high-frequency oscillations. We discretize the domain $[0,3]$ and generate a sequence of nested grids of the form (3.1) by setting $N_0 = 16$ and $K = 4$. We use the data on the grid $\mathcal{T}^k$, and interpolate it onto the grid $\mathcal{T}^{k+1}$ for $0 \leqslant k < K$. As shown in Figure 9, the interpolation with ENO-4 and DeLENO-4 is identical on all grids, for this particular function.

6.3.2. *Data compression.* We now apply the multi-resolution representation framework of Appendix C to use DeLENO to compress the function (6.8). We construct a nested sequence of meshes on $[0,3]$ by choosing $N_0 = 9$ and $K = 5$ in (C.1). We use Algorithm 3 to obtain the multi-resolution representation of the form (C.5) and decode the solution using Algorithm 4 to obtain the approximation $\widehat{q}^K$. The compression thresholds needed for the encoding procedure are set using (C.6).

Figure 10 provides a comparison of the results obtained using different values for the threshold parameters ϵ, and shows the non-zero coefficients $\widehat{d}^k$ for each mesh level k. A higher value of ϵ can truncate a larger number of $\widehat{d}^k$ components, as is evident for $p = 3$. However, there is no qualitative difference between $\widehat{q}^K$ obtained for the two ϵ values considered. Thus, it is beneficial to use the larger ϵ, as it leads to a sparser multi-resolution representation without deteriorating the overall features. The solutions obtained with ENO and DeLENO are indistinguishable. We refer to Table 2 for the errors of the two methods.

The compression ideas used for one-dimensional problems can be easily extended to handle functions defined on two-dimensional tensorized grids. We consider a sequence of grids $\mathcal{T}^k$ with $(N_k^x + 1) \times (N_k^y + 1)$ nodes, where $N_k^x = 2^k N_0^x$ and $N_k^y = 2^k N_0^x$, for $0 \leqslant k \leqslant K$. Let q^k be the data on grid $\mathcal{T}^k$ and denote by $\widehat{q}^{k+1}$ the compressed interpolation on grid $\mathcal{T}^{k+1}$. To obtain $\widehat{q}^{k+1}$, we first interpolate along the x-coordinate direction to obtain an intermediate approximation $\widetilde{q}^{k+1}$ of



(a) $\mathcal{T}^0$ to $\mathcal{T}^1$, $p = 3$

(b) $\mathcal{T}^0$ to $\mathcal{T}^1$, $p = 4$

(c) $\mathcal{T}^1$ to $\mathcal{T}^2$, $p = 3$

(d) $\mathcal{T}^1$ to $\mathcal{T}^2$, $p = 4$

(e) $\mathcal{T}^2$ to $\mathcal{T}^3$, $p = 3$

(f) $\mathcal{T}^2$ to $\mathcal{T}^3$, $p = 4$

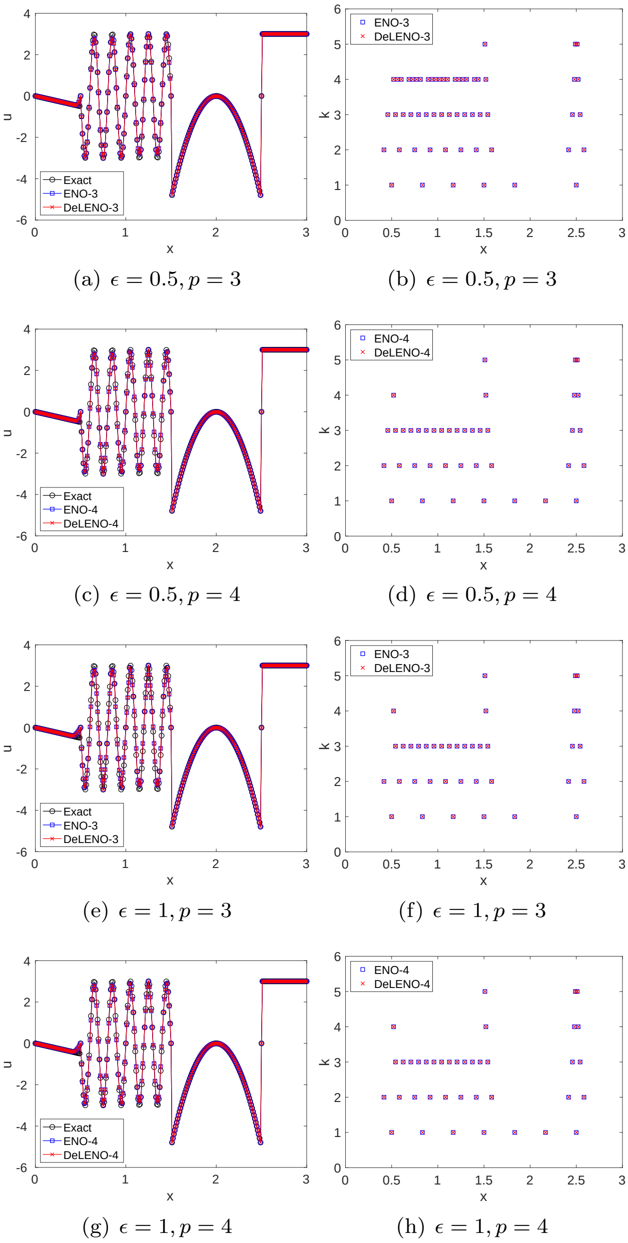
(g) $\mathcal{T}^3$ to $\mathcal{T}^4$, $p = 3$

(h) $\mathcal{T}^3$ to $\mathcal{T}^4$, $p = 4$

Figure 9. Interpolating the function (6.8) using ENO and DeLENO.

p	ϵ	$\lVert q^K - \widehat{q}^K \rVert_1$		$\lVert q^K - \widehat{q}^K \rVert_2$		$\lVert q^K - \widehat{q}^K \rVert_\infty$	
		ENO	DeLENO	ENO	DeLENO	ENO	DeLENO
3	0.5	5.125e-2	5.125e-2	8.701e-2	8.701e-2	3.281e-1	3.281e-1
	1.0	2.072e-1	2.072e-1	2.421e-1	2.421e-1	4.102e-1	4.102e-1
4	0.5	1.032e-1	1.038e-1	1.268e-1	1.274e-1	3.027e-1	3.027e-1
	1.0	1.122e-1	1.122e-1	1.356e-1	1.356e-1	3.947e-1	3.947e-1

Table 2. 1D compression errors for (6.8).



(a) $\epsilon = 0.5, p = 3$

(b) $\epsilon = 0.5, p = 3$

(c) $\epsilon = 0.5, p = 4$

(d) $\epsilon = 0.5, p = 4$

(e) $\epsilon = 1, p = 3$

(f) $\epsilon = 1, p = 3$

(g) $\epsilon = 1, p = 4$

(h) $\epsilon = 1, p = 4$

Figure 10. Data compression of (6.8) using ENO and DeLENO with N_0, $L = 5$ and $t = 0.5$. Comparison of thresholded decompressed data with the actual data on the finest level (left); Non-zero coefficients $\widehat{d}^k$ at each level (right).

size $(N^x_{k+1} + 1) \times (N^y_k + 1)$. Then we use $\widetilde{q}^{k+1}$ to interpolate along the y-coordinate direction to obtain the final approximation $\widehat{q}^{k+1}$.

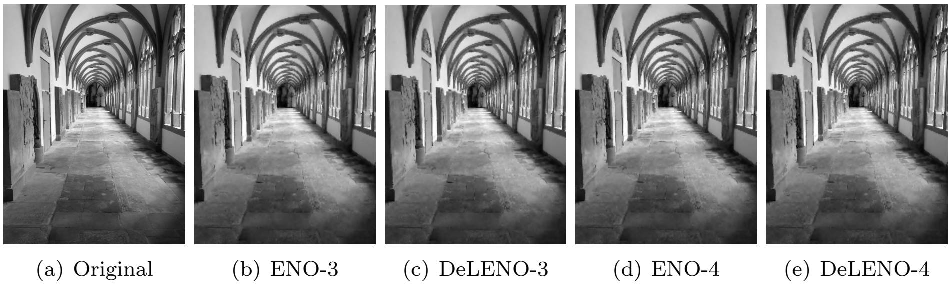
Next, we use ENO and DeLENO to compress an image with 705×929 pixels, shown in Figure 11(a). We set $K = 5$, $\epsilon = 1$, $t = 0.2$ in equation (C.6). Once again, ENO and DeLENO give similar results, as can be seen from the decompressed images in Figure 11 and the relative errors in Table 3. In this table we additionally listed the compression rate

$$c_r = 1 - \frac{\#\left\{d^k_{i,j} | d^k_{i,j} > \epsilon^k,\ 1 \leqslant k \leqslant K\right\}}{(N^x_L+1)(N^y_L+1) - (N^x_0+1)(N^y_0+1)}, \tag{6.9}$$

which represents the fraction of coefficients set to null.

p	Scheme	Rel. L^1	Rel. L^2	Rel. L^∞	c_r
3	ENO	5.346e-2	8.368e-2	5.194e-1	0.996
	DeLENO	5.343e-3	8.365e-2	5.194e-1	0.996
4	ENO	5.422e-2	8.485e-2	5.581e-1	0.996
	DeLENO	5.422e-2	8.492e-2	5.581e-1	0.996

Table 3. Image compression errors.



(a) Original (b) ENO-3 (c) DeLENO-3 (d) ENO-4 (e) DeLENO-4

Figure 11. Image compression.

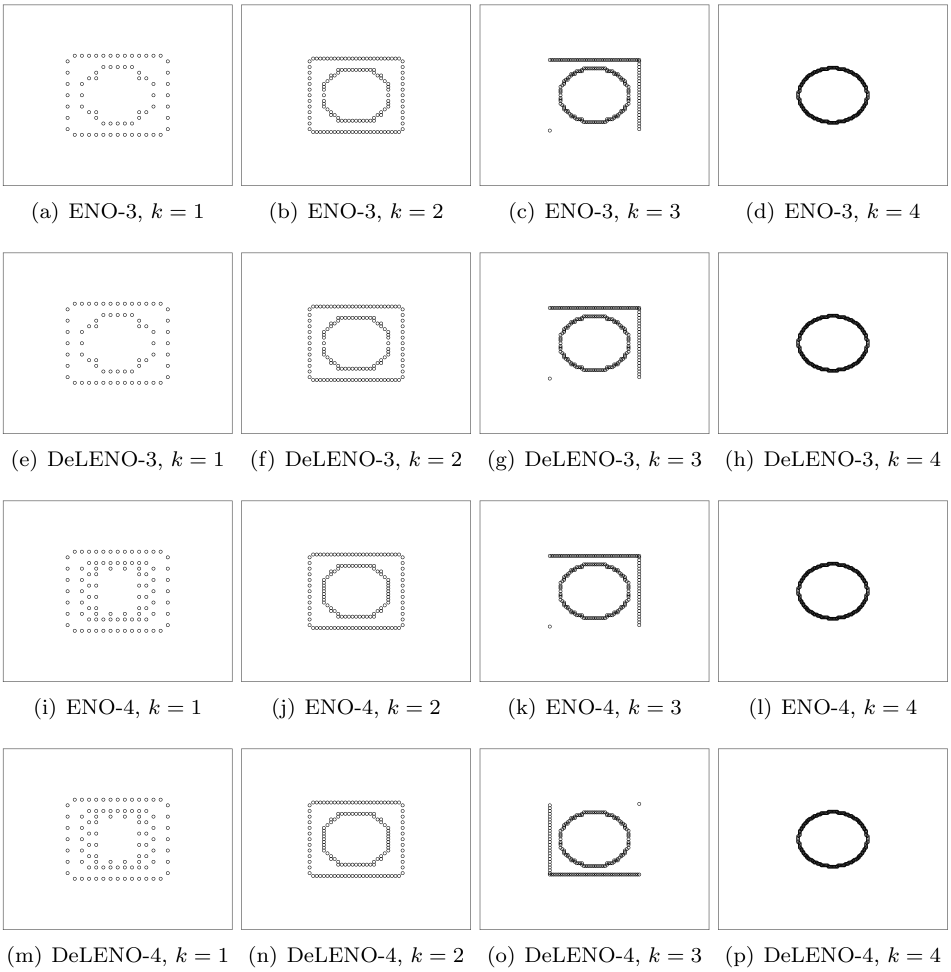
As an additional example of two-dimensional data compression, we consider the function

$$q(x,y) = \begin{cases} -10 & \text{if } (x-0.5)^2 + (y-0.5)^2 < 0.0225 \\ 30 & \text{if } |x-0.5| > 0.8 \text{ or } |y-0.5| > 0.8 \\ 40 & \text{otherwise} \end{cases}, \tag{6.10}$$

where $(x,y) \in [0,1] \times [0,1]$, and generate a sequence of meshes by setting $K = 4$, $N^x_0 = 16$ and $N^y_0 = 16$. The threshold for data compression is chosen according to (C.6), with $\epsilon = 10$ and $t = 0.5$. The non-zero $\widehat{d}^k$ coefficients are plotted in Figure 12, while the errors and compression rate (6.9) are listed in Table 4. Overall, ENO and DeLENO perform equally well, with DeLENO giving marginally smaller errors.

p	Scheme	Rel. L^1	Rel. L^2	Rel. L^∞	c_r
3	ENO	3.341e-3	2.442e-2	4.302e-1	0.989
	DeLENO	3.246e-3	2.367e-2	4.302e-1	0.989
4	ENO	3.816e-3	3.237e-2	5.876e-1	0.989
	DeLENO	3.681e-3	3.130e-2	5.876e-1	0.989

Table 4. 2D compression errors for (6.10).



(a) ENO-3, $k = 1$ (b) ENO-3, $k = 2$ (c) ENO-3, $k = 3$ (d) ENO-3, $k = 4$

(e) DeLENO-3, $k = 1$ (f) DeLENO-3, $k = 2$ (g) DeLENO-3, $k = 3$ (h) DeLENO-3, $k = 4$

(i) ENO-4, $k = 1$ (j) ENO-4, $k = 2$ (k) ENO-4, $k = 3$ (l) ENO-4, $k = 4$

(m) DeLENO-4, $k = 1$ (n) DeLENO-4, $k = 2$ (o) DeLENO-4, $k = 3$ (p) DeLENO-4, $k = 4$

FIGURE 12. Non-zero coefficients $\widehat{d}^k$ for data compression of (6.10) using ENO and DeLENO for mesh level $1 \leqslant k \leqslant 4$.

6.3.3. *Conservation laws.* We compare the performance of ENO and DeLENO reconstruction, when used to approximate solutions of conservation laws. We work in the framework of high-order finite difference schemes with flux-splitting and we use a fourth-order Runge-Kutta scheme for the time integration.

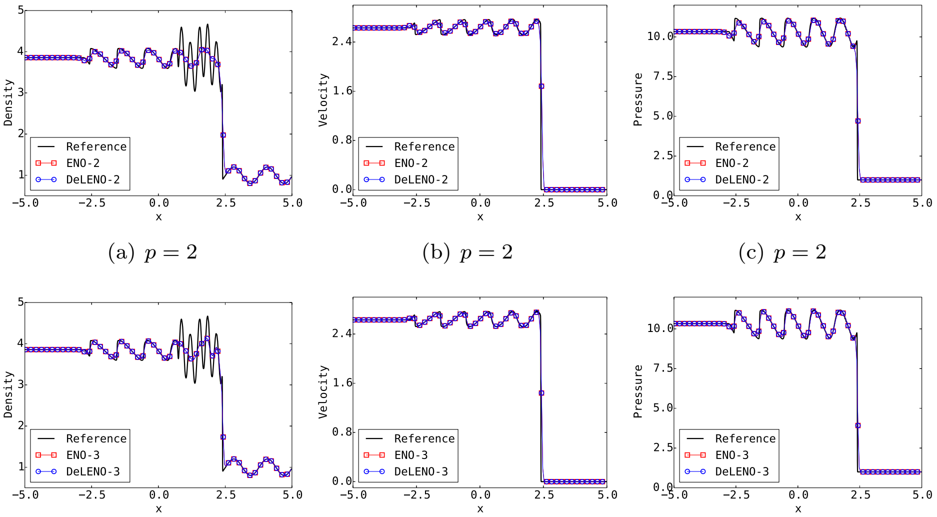
As an example, we consider the system of conservation laws governing compressible flows given by

$$\partial_t \begin{pmatrix} \rho \\ v \\ p \end{pmatrix} + \partial_x \begin{pmatrix} \rho v \\ \rho v^2 + p \\ (E+p)v \end{pmatrix} = 0, \qquad E = \frac{1}{2}\rho v^2 + \frac{p}{\gamma - 1},$$

where ρ, v and p denote the fluid density, velocity and pressure, respectively. The quantity E represents the total energy per unit volume where $\gamma = c_p/c_v$ is the ratio of specific heats, chosen as $\gamma = 1.4$ for our simulations. We consider the shock-entropy problem [23], which describes the interaction of a right moving shock with smooth oscillatory waves. The initial conditions for this test case are prescribed as

$$(\rho,\ v,\ p) = \begin{cases} (3.857143,\ 2.629369,\ 10.33333) & \text{if } x < -4 \\ (1 + 0.2\sin(5x),\ 0,\ 1) & \text{if } x > -4 \end{cases},$$

on the domain $[-5, 5]$. Due to the generation of high frequency physical waves, we solve the problem on a fine mesh with $N = 200$ cells up to $T_f = 1.8$ with CFL $= 0.5$. A reference solution is obtained with ENO-4 on a mesh with $N = 2000$ cells. As can be seen in Figure 13, ENO-p and DeLENO-p perform equally well depending on the order p.



(a) $p = 2$ (b) $p = 2$ (c) $p = 2$

(d) $p = 3$ (e) $p = 3$ (f) $p = 3$

FIGURE 13. Solution for Euler shock-entropy problem with ENO-p and DeLENO-p on a mesh with $N = 200$ cells.

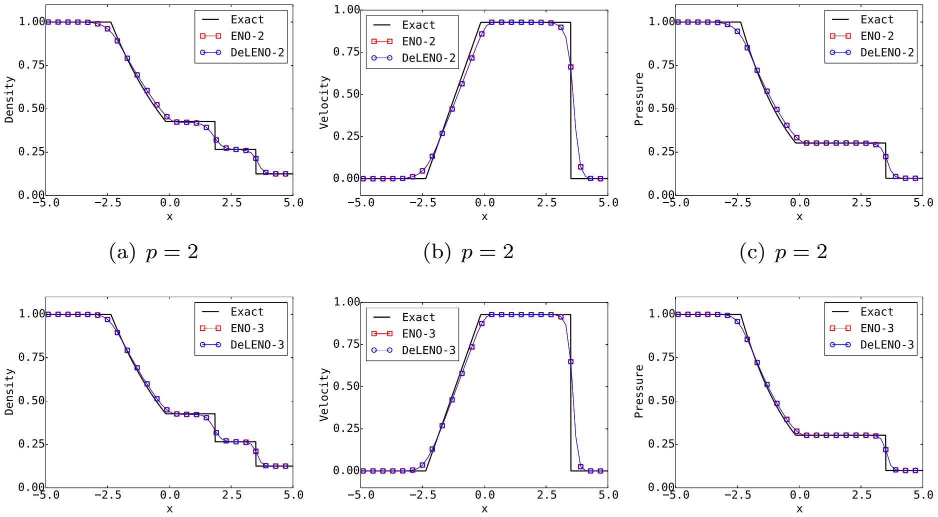
Next we solve the Sod shock tube problem [25], whose initial conditions are given by

$$(\rho,\ v,\ p) = \begin{cases} (1,\ 0,\ 1) & \text{if } x < 0 \\ (0.125,\ 0,\ 0.1) & \text{if } x > 0 \end{cases},$$

on the domain $[-5, 5]$. The solution consists of a shock wave, a contact discontinuity and a rarefaction. The mesh is descretized with $N = 50$ cells and the problem is solved till $T_f = 2$ with a CFL $= 0.5$. The solutions obtained with ENO-p and DeLENO-p are identical, as depicted in Figure 14.

7. DISCUSSION

In this paper, we considered efficient interpolation of rough or piecewise smooth functions. A priori, both deep neural networks (on account of universality) and the well-known *data dependent* ENO (and ENO-SR) interpolation procedure are able to interpolate rough functions accurately. We proved here that the ENO interpolation (and the ENO reconstruction) procedure as well as a variant of the second-order ENO-SR procedure can be cast as deep ReLU neural networks, at least for univariate functions. This equivalence provides a different perspective on the ability of neural networks in approximating functions and reveals their enormous



(a) $p = 2$

(b) $p = 2$

(c) $p = 2$

(d) $p = 3$

(e) $p = 3$

(f) $p = 3$

FIGURE 14. Solution for Euler Sod shock tube problem with ENO-p and DeLENO-p on a mesh with $N = 50$ cells.

expressive power as even a highly non-linear, data-dependent procedure such as ENO can be written as a ReLU neural network.

On the other hand, the impressive function approximation results, for instance of [26, 27], might have limited utility for functions in low dimensions, as the neural network needs to be trained for every function that has to be interpolated. By interpreting ENO (and ENO-SR) as a neural network, we provide a natural framework for recasting the problem of interpolation in terms of pre-trained neural networks such as DeLENO, where the input vector of sample values are transformed by the network into the output vector of interpolated values. Thus, these networks are trained once and do not need to retrained for every new underlying function. This interpretation of ENO as a neural network allows us to possibly extend ENO type interpolations into several space dimensions on unstructured grids.

REFERENCES

[1] F. Aràndiga, A. Cohen, R. Donat, and N. Dyn. Interpolation and approximation of piecewise smooth functions. *SIAM Journal on Numerical Analysis*, 43(1):4157, 2005.

[2] F. Arandiga, A. Cohen, R. Donat, N. Dyn, and B. Matei. Approximation of piecewise smooth functions and images by edge-adapted (ENO-EA) nonlinear multiresolution techniques. *Applied and Computational Harmonic Analysis*, 24(2):225–250, 2008.

[3] F. Aràndiga and R. Donat. Nonlinear multiscale decompositions: The approach of A. Harten. *Numerical Algorithms*, 23(2):175–216, Jun 2000.

[4] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

[5] G. Aubert and P. Kornprobst. *Mathematical problems in image processing: partial differential equations and calculus of variations*. Springer, New York, 2006.

[6] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[7] B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Math. Comp.*, 52(186):411–435, 1989.

[8] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1988.

[9] C. M. Dafermos. *Hyperbolic conservation laws in continuum physics*, volume 325 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, third edition, 2010.

[10] T. De Ryck. On the approximation of rough functions with artificial neural networks. Master's thesis, ETH Zurich, Zurich, 2020.

[11] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[12] G. L. Eyink and K. Sreenivasan. Onsager and the theory of hydrodynamic turbulence. *Rev. Modern Phys.*, 78(1):87–135, 2006.

[13] U. S. Fjordholm, S. Mishra, and E. Tadmor. ENO recontruction and ENO interpolation are stable. *Found. Comp. Math*, 13(2):139–159, 2013.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] A. Harten. ENO schemes with subcell resolution. *Journal of Computational Physics*, 83(1):148184, 1989.

[16] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 71(2):231303, 1987.

[17] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 131(1):3 – 47, 1997.

[18] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[20] K. O. Lye, S. Mishra, and D. Ray. Deep learning observables in computational fluid dynamics. Preprint, available as arXiv:1903.03040v1.

[21] P. Petersen, M. Raslan, and F. Voigtlaender. Topological properties of the set of functions generated by neural networks of fixed size.

[22] P. Petersen and F. Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

[23] C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, ii. *Journal of Computational Physics*, 83(1):32 – 78, 1989.

[24] C. W. Shu and S. Osher. High-order essentially nonoscillatory schemes for hamilton-jacobi equations. *SIAM J. Num. Anal.*, 28(4):107–122, 1991.

[25] G. A. Sod. A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of Computational Physics*, 27(1):1 – 31, 1978.

[26] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103 – 114, 2017.

[27] D. Yarotsky and A. Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. Preprint, available as arXiv:1906.09477v1.

Appendix A. ENO reconstruction

In this section we present ENO reconstruction, which is very similar to ENO interpolation as presented in Section 3.1. Let V be a function on $[c,d]$. For the sake of completeness, we repeat the main steps of the algorithm for reconstruction purposes.

We define a uniform mesh $\mathcal{T}$ on $[c,d]$ with N cells,

$$\mathcal{T} = \{I_i\}_{i=1}^N,\ I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}],\ \left\{x_i = c + (2i-1)\frac{h}{2}\right\}_{i=0}^N,\ h = \frac{(d-c)}{N}, \tag{A.1}$$

where x_i and $x_{i\pm\frac{1}{2}}$ denote the cell center and cell interfaces of the cell I_i, respectively. We are given the cell averages

$$\overline{V}_i = \frac{1}{h}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} V(\xi)\mathrm{d}\xi, \quad 1 \leqslant i \leqslant N$$

and we define $\overline{V}_{-p+2}, ..., \overline{V}_0$ and $\overline{V}_{N+1}, ..., \overline{V}_{N+p-1}$ to be ghost values that need to be suitably prescribed. The goal is to find a local interpolation operator $\mathcal{I}_i^h$ such that

$$\left\|\mathcal{I}_i^h V - V\right\|_{\infty,I_i} = O(h^p) \text{ for } h \to 0.$$

For this purpose, let $\hat{V}$ be the primitive function of V and note that we have access to the value of $\hat{V}$ at the cell interfaces,

$$\hat{V}(x_{i+\frac{1}{2}}) = h\sum_{j=0}^{i} \overline{V}_j \quad \text{where} \quad \hat{V}(x) = \int_c^x V(\xi)\mathrm{d}\xi.$$

Next let P_i be the unique polynomial of degree p that agrees with $\hat{V}$ on a chosen set of $p+1$ cell interfaces that includes $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. The ENO reconstruction procedure considers the stencil sets

$$\mathcal{S}_i^r = \{x_{i-\frac{1}{2}-r+j}\}_{j=0}^{p}, \quad 0 \leqslant r \leqslant p-1,$$

where r is called the (left) stencil shift. The smoothest stencil is then selected based on the local smoothness of f using Newton's undivided differences. Algorithm 2 describes how the stencil shift r_i corresponding to this stencil can be obtained. Note that r_i uniquely defines the polynomial P_i. We then define $\mathcal{I}_i^h V$ to be the first derivative of P_i, one can check that this polynomial is indeed a p-th order accurate approximation. Note that the interpolants on two adjacent intervals do not need to agree on the mutual cell interface.

Algorithm 2: ENO reconstruction stencil selection

```
Input: ENO order $p$, input array $\mathbf{\Delta^0} = \{\overline{V}_{i+j}\}_{j=-p+1}^{p-1}$, for any $1 \leqslant i \leqslant N$.
Output: Stencil shift $r$.
Evaluate Newton undivided differences:
for $j = 1$ to $p-1$ do
    $\mathbf{\Delta^j} = \mathbf{\Delta^{j-1}}[2 : \text{end}] - \mathbf{\Delta^{j-1}}[1 : \text{end} - 1]$
Find shift:
$r = 0$
for $j = 1$ to $p-1$ do
    if $|\mathbf{\Delta^j}[p-1-r]| < |\mathbf{\Delta^j}[p-r]|$ then
        $r = r + 1$
return $r$
```

In order to implement an ENO scheme, one only needs the values of $\mathcal{I}_i^h V$ at the cell interfaces $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. Analogously to equation (3.2), these can be directly obtained by calculating

$$\text{(A.2)} \quad \mathcal{I}_i^h V(x_{i+\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i,j}^p \overline{V}_{i-r_i+j} \quad \text{and} \quad \mathcal{I}_i^h V(x_{i-\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i-1,j}^p \overline{V}_{i-r_i+j},$$

where r_i is stencil shift corresponding to the smoothest stencil for interval I_i and with the coefficients $\tilde{C}_{r,j}^p$ listed in Table 6 in Appendix B.

Appendix B. ENO coefficients

Table 5 and Table 6 respectively list the ENO coefficients used for ENO interpolation (Section 3.1) and ENO reconstruction (Appendix A). More information on the calculation of these coefficients can be found in [7].

r	s	$j=0$	$j=1$	$j=2$	$j=3$
3	0	3/8	3/4	-1/8	-
	1	-1/8	3/4	3/8	-
4	0	5/16	15/16	-5/16	1/16
	1	-1/16	9/16	9/16	-1/16
	2	1/16	-5/16	15/16	5/16

Table 5. Coefficients for ENO interpolation for $p > 2$ used in (3.2).

p	s	$j=0$	$j=1$	$j=2$	$j=3$
2	-1	3/2	-1/2	-	-
	0	1/2	1/2	-	-
	1	-1/2	3/2	-	-
3	-1	11/6	-7/6	1/3	-
	0	1/3	5/6	-1/6	-
	1	-1/6	5/6	1/3	-
	2	1/3	-7/6	11/6	-
4	-1	25/12	-23/12	13/12	-1/4
	0	-1/4	13/12	-5/12	1/12
	1	-1/12	7/12	7/12	-1/12
	2	1/12	-5/12	13/12	-1/4
	3	-1/4	13/12	-23/12	25/12

Table 6. Coefficients for ENO reconstruction used in (A.2).

Appendix C. Multi-resolution representation of functions for data compression

To describe the multi-resolution representation of functions, we use notations and operators similar to those introduced in [3]. We define a sequence of nested uniform meshes $\{\mathcal{T}^k\}_{k=0}^K$ on $\Omega = [c,d]$, where

(C.1)
$$\mathcal{T}^k = \{I_i^k\}_{i=1}^{N_k}, \quad I_i^k = [x_{i-1}^k, x_i^k], \quad \{x_i^k = c + ih_k\}_{i=0}^{N_k}, \quad h_k = \frac{(d-c)}{N_k}, \quad N_k = 2^k N_0,$$

for $0 \leqslant k \leqslant K$ and where N_0 is some positive integer. We call $\{x_i^k\}_{i=0}^{N_k}$ the nodes of the mesh $\mathcal{T}^k$. Let $\mathcal{B}_\Omega$ be the set of bounded functions on Ω and $\mathcal{V}^n$ the space of real-valued finite sequences of length n. We define the following operators associated with the various meshes:

- The *discretizer* $D_k : \mathcal{B}_\Omega \mapsto \mathcal{V}^{N_k+1}$ defined by
$$D_k f = q^k := \{q_i^k\}_{i=0}^{N_k} = \{q(x_i^k)\}_{i=0}^{N_k}, \quad \forall\, q \in \mathcal{B}_\Omega.$$
- The *reconstructor* $R_k : \mathcal{V}^{N_k+1} \mapsto \mathcal{B}_\Omega$ satisfying $D_k R_k q^k = q^k$ for $q^k \in \mathcal{V}^{N_k+1}$. Thus, $(R_k q^k)(x)$ interpolates the members of q^k at the nodes of $\mathcal{T}^k$.
- The *decimator* $D_k^{k-1} : \mathcal{V}^{N_k+1} \mapsto \mathcal{V}^{N_{k-1}+1}$ defined by $D_k^{k-1} := D_{k-1} R_k$. For $q \in \mathcal{B}_\Omega$, we have
$$q_i^{k-1} = (D_k^{k-1} q^k)_i = q_{2i}^k, \quad 0 \leqslant i \leqslant N_{k-1}. \tag{C.2}$$
In other words, the decimator helps in extracting the function values on a given mesh from a finer one.

- The *predictor* $P^k_{k-1} : \mathcal{V}^{N_{k-1}+1} \mapsto \mathcal{V}^{N_k+1}$ defined by $P^k_{k-1} := D_k R_{k-1}$. The predictor tries to recover the function values q^k from the coarser data q^{k-1}, for $q \in \mathcal{B}_\Omega$.

The prediction error is given by

$$e^k_i = q^k_i - (P^k_{k-1} q^{k-1})_i, \quad 0 \leqslant i \leqslant N_k.$$

Clearly $e^k_{2i} = 0$ for $0 \leqslant i \leqslant N_{k-1} = N_k/2$. Thus, the interpolation error is essentially evaluated at the nodes in $\mathcal{T}^k \setminus \mathcal{T}^{k-1}$, which we denote as

$$d^k_i = e^k_{2i-1} = q^k_{2i-1} - (P^k_{k-1} q^{k-1})_{2i-1}, \quad 1 \leqslant i \leqslant N_{k-1}. \tag{C.3}$$

Given q^{k-1} and d^k, we can recover q^k using (C.2) and (C.3). By iteratively applying this procedure, the data q^k on the finest mesh can be fully encoded using the multiresolution representation

$$\{q^0, d^1, d^2, ..., d^K\}. \tag{C.4}$$

This multiresolution representation (C.4) for a function $f \in \mathcal{B}_\Omega$ is convenient to perform data compression. The easiest compression strategy [3] corresponds to setting the coefficients d^k_i in (C.3) to zero based on a suitable threshold $\epsilon^k \geqslant 0$:

$$\widehat{d}^k_i = \mathcal{G}(d^k_i; \epsilon^k) = \begin{cases} 0 & \text{if } |d^k_i| \leqslant \epsilon^k \\ d^k_i & \text{otherwise.} \end{cases}$$

The compressed representation is then given by

$$\{f^0, \widehat{d}^1, \widehat{d}^2, ..., \widehat{d}^K\}. \tag{C.5}$$

The procedures for compressed encoding and decoding are listed in Algorithms 3 and 4.

Algorithm 3: Compressed encoding [3]

```
Input: Highest resolution data $f^K$, number of levels $K$, number of points $N_0$
       on coarsest mesh, ENO order $p$, threshold parameters $\epsilon$ and $t$.
Output: Multi-resolution representation $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$.
for $k = K$ to $1$ do
    $f^{k-1} = D^{k-1}_k f^k$
$\hat{f}^0 = f^0$
for $k = 1$ to $K$ do
    $\widehat{f}^k_0 = f^K_0$
    Construct $P^k_{k-1}$ using Algorithm 1 and equation (3.2)
    $\widetilde{f}^k = P^k_{k-1} \widehat{f}^{k-1}$
    $N = N_0 2^{k-1}$
    for $i = 1$ to $N$ do
        $d^k_i = f^k_{2i-1} - \widetilde{f}^k_{2i-1}$
        $\epsilon^k = \epsilon t^{K-k}$
        $\widehat{d}^k_i = \mathcal{G}(d^k_i; \epsilon^k)$
        $\widehat{f}^k_{2i-1} = \widetilde{f}^k_{2i-1} + \widehat{d}^k_i$
        $\widehat{f}^k_{2i} = \widehat{f}^{k-1}_i$
return $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$
```

The following result is known on the error bounds for the compressed encoding in the form (C.5), the proof can be found in [3].

Algorithm 4: Decoding multi-resolution data [3]

```
Input: Multi-resolution representation $\{f^0, \widehat{d}^1, ..., \widehat{d}^K\}$, number of levels $K$,
       number of cells $N_0$ on coarsest mesh, ENO order $p$.
Output: Decoded function $\widehat{f}^K$.
$\hat{f}^0 = f^0$
for $k = 1$ to $K$ do
    Construct $P^k_{k-1}$ using Algorithm 1 and equation (3.2)
    $\widehat{f}^k = P^k_{k-1}\widehat{f}^{k-1} + \widehat{d}^k$
return $\widehat{f}^K$
```

Proposition C.1. *Let $\{\Omega^k\}_{l=0}^K$ be a sequence of nested uniform meshes discretizing the interval $[c, d]$ generated according to* (C.1) *for some positive integer $N_0 > 1$. Assume that some $f \in \mathcal{B}[c, d]$ is encoded using thresholds*

$$\epsilon^k = \epsilon t^{K-k}, \quad 0 < t < 1. \tag{C.6}$$

to give rise to the multi-resolution representation of the form (C.5)*. If $\widehat{f}^K$ is the decoded data, then*

$$\|f^K - \widehat{f}^K\|_n \leqslant C_n\epsilon \quad \textit{for } n = \infty, 1, 2, \tag{C.7}$$

where $C_\infty = (1 - t)^{-1}$, $C_1 = (b - a)(1 - t)^{-1}$ and $C_2 = \sqrt{(b - a)(1 - t^2)^{-1}}$. This estimate is independent of the interpolation procedure used to encode and decode the data.

Appendix D. Proof of Theorem 3.3

We present some properties of our adaptation of the ENO-SR algorithm. To prove the second-order accuracy, we state some results due to [1] in a slightly adapted form.

Lemma D.1. *The groups of adjacent B intervals are at most of size 2. They are separated by groups of adjacent G intervals that are at least of size 2.*

Proof. Note that our detection algorithm is the same as the one in [1] for $m = 3$. The result then follows from their Lemma 1. □

Lemma D.2. *Let f be a globally continuous function with a bounded second derivative on $\mathbb{R}\backslash\{z\}$ and a discontinuity in the first derivative at a point z. Define the critical scale*

$$h_c := \frac{|[f']|}{4 \sup_{x\in\mathbb{R}\backslash\{z\}}|f''(x)|}, \tag{D.1}$$

where $[f']$ is the jump of the first derivative f' at the point z. Then for $h < h_c$, the interval that contains z is labelled B.

Proof. See Lemma 2 in [1]. □

Lemma D.3. *There exist constants $C > 0$ and $0 < K < 1$ such that for all continuous f with uniformly bounded second derivative on $\mathbb{R}\backslash\{z\}$ and for $h < Kh_c$ with h_c defined by equation (D.1), the following holds:*

1. *The singularity z is contained in an isolated B interval I^k_i or in a B-pair (I^k_i, I^k_{i+1}).*
2. *The two polynomials p^k_{i-2} and p^k_{i+2} (or p^k_{i-1} and p^k_{i+3}) have only one intersection point y inside I^k_i or $I^k_i \cup I^k_{i+1}$, respectively.*

(3) *The distance between z and y is bounded by*

$$|z-y| \leqslant \frac{C \sup_{x\in\mathbb{R}\setminus\{z\}}|f''(x)|h^2}{|[f']|}. \tag{D.2}$$

Proof. This is a light adaptation of Lemma 3 in [1]. The proof remains the same, after one minor change. We take $I = [b, c]$ to be equal to $I_{-1} \cup I_0 \cup I_1$, which does not affect equation (38) in the proof. In fact, all other steps of their proof remain valid. It only must be noted that the constant C in equation (D.2) of this paper and equation (37) in [1] do not necessarily agree. □

We now prove Theorem 3.3 of Section 5, based on the proof of Theorem 1 in [1]. Let h_c be as in Lemma D.2 and let K be as in Lemma D.3. Note that we can write

$$f = f_-\mathbb{1}_{(-\infty,z]} + f_+\mathbb{1}_{(z,+\infty)}$$

where f_-, f_+ are C^2 on $\mathbb{R}$ such that

$$\sup_{\mathbb{R}\setminus\{z\}}|f''_\pm| \leqslant \sup_{\mathbb{R}\setminus\{z\}}|f''|.$$

Let us consider some interval $I_0 = [b, c] = [b, b + h]$. First consider the case $0 < h < Kh_c$. Suppose it was labelled as good. Lemma D.2 then guarantees that I_0 does not contain z. It then follows directly from the theory of Lagrange interpolation that

$$|f(x) - \mathcal{I}^h f(x)| \leqslant Ch^2 \sup_{\mathbb{R}\setminus\{z\}}|f''| \tag{D.3}$$

for all $x \in I_0$. Now suppose that I_0 was labelled bad. As a consequence of Lemma D.1, I_{-2} and I_2 are good intervals and therefore do not contain the discontinuity. If in addition $z \notin I_{-1} \cup I_0 \cup I_1$, then equation (D.3) holds again for all $x \in I_0$ since $\mathcal{I}^h f(x)$ is either equal to $p_{-2}(x)$, $p_0(x)$ or $p_2(x)$. On the other hand, if $z \in I_{-1} \cup I_0 \cup I_1$ then Lemma D.3 guarantees the existence of a single intersection point $y \in I_{-1} \cup I_0 \cup I_1$ of p_{-2} and p_2. Assume now without loss of generality that $z \leqslant y$. In this case, equation (D.3) holds for all $x \in [b, z] \cup [y, c]$. It thus remains to treat the case $z < x < y$. For such x, we have

$$|f(x) - \mathcal{I}^h f(x)| = |f_+(x) - p_{-2}(x)| \leqslant |f_+(x) - f_-(x)| + |f_-(x) - p_{-2}(x)|$$

where the second term is again bounded by $Ch^2 \sup_{\mathbb{R}\setminus\{z\}}|f''|$. We can use a second-order Taylor expansion for the first term to derive

$$|f_+(x) - f_-(x)| \leqslant (y-z)([f'] + 2h \sup_{\mathbb{R}\setminus\{z\}}|f''|) \leqslant \frac{3}{2}|[f']|(y-z)$$

where in the last inequality we used that $h < h_c$. By invoking the third part of Lemma D.3, we find indeed that equation (D.3) holds again. This concludes the proof for the case $h < Kh_c$.

Now suppose that $h \geqslant Kh_c$. First define

$$f_2(x) = f(x) - [f'](x-z)_+$$

for $x \in \mathbb{R}$. Furthermore, by the definition of h_c in Lemma D.2, we find that for $h \geqslant Kh_c$,

$$[f'] = 4h_c \sup_{\mathbb{R}\setminus\{z\}}|f''| \leqslant C_0 h \sup_{\mathbb{R}\setminus\{z\}}|f''|, \tag{D.4}$$

where $C_0 > 0$ does not depend on f. We distinguish two cases.

Case 1: $\mathcal{I}^h(x) = p_0(x)$ for all $x \in I_0$. If $z \notin I_0$, second-order accuracy as in equation (D.3) is immediate. If not, more work is needed. Define

$$g_1(x) = \frac{[f'](x_0-z)_+}{h}(x - x_{-1})$$

and note that $p_0 - g_1$ is the linear interpolation between $(x_{-1}, f_2(x_{-1}))$ and $(x_0, f_2(x_0))$. Since f_2 is C^2 we know that $p_0 - g_1$ is a second-order accurate approximation of f_2 on I_0, such that equation (D.3) holds. We then calculate for $x \in I_0$,

$$\begin{aligned}|f(x) - p_0(x)| &\leqslant |f_2(x) - (p_0(x) - g_1(x))| + |[f'](x-z)_+ - g_1(x)|\\ &\leqslant C_1 h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''| + [f'] \left(|(x-z)_+| + \frac{(x_0 - z)_+}{h} |x - x_{-1}| \right)\\ &\leqslant C_1 h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''| + C_0 h \sup_{\mathbb{R}\setminus\{z\}} |f''| (h+h)\\ &= C h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''|,\end{aligned}$$

where we used inequality (D.4).

Case 2: $\mathcal{I}^h(x) = p_{-2}(x)\mathbb{1}_{(-\infty,y]}(x) + p_2(x)\mathbb{1}_{(y,+\infty)}(x)$ for $x \in I_0$, where y is the intersection point of p_{-2} and p_2. If $z \notin \cup_{q=-2}^{2} I_q$, then inequality (D.3) follows immediately for $x \in I_0$. Consider now the case that $z \in \cup_{q=-2}^{2} I_q$ and assume without loss of generality $z \leqslant y$. Let $x \in I_0$ be arbitrary. It follows that equation (D.3) also holds immediately for this x if $y \leqslant x$ or $x \leqslant z$. It suffices to find a bound for when $x_{-3} \leqslant z \leqslant x \leqslant y$. Define

$$g_2(x) = \frac{[f'](x_{-2} - z)_+}{h}(x - x_{-3}).$$

Note that $p_{-2} - g_2$ is an affine function through $(x_{-3}, f_2(x_{-3}))$ and $(x_{-2}, f_2(x_{-2}))$. It follows that

$$\begin{aligned}|f(x) - p_{-2}(x)| &\leqslant |f_2(x) - (p_{-2} - g_2(x))| + |[f'](x-z)_+ - g_2(x)|\\ &\leqslant C_1 h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''| + [f'] \left(|(x-z)_+| + \frac{(x_{-2} - z)_+}{h} |x - x_{-3}| \right)\\ &\leqslant C_1 h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''| + C_0 h \sup_{\mathbb{R}\setminus\{z\}} |f''| (3h+3h)\\ &= C h^2 \sup_{\mathbb{R}\setminus\{z\}} |f''|,\end{aligned}$$

where we used again inequality (D.4) and the bounds $|x - x_{-3}| \leqslant 3h$ and $x_{-3} \leqslant z$. This concludes the proof of Theorem 3.3.

Appendix E. Proof of Theorem 5.4

In what follows, we let $x^* = x_{2i-1}^{k+1}$, $f^* = \mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$, $\hat{f}_\epsilon = \hat{f}_{2i-1,\epsilon}^{k+1}$ and $X_l^3 = X_{n_{i,l}}^3$ for $1 \leqslant l \leqslant 4$ (where $X_{n_{i,l}}^3$ is as in the proof of Theorem 5.1). We assume without loss of generality that $[c,d] \subset [0,\infty)$. Furthermore we simplify the notation by dropping the index k and setting $i = 0$. It follows from the proof of Theorem 3.3 that the results holds for $h \geqslant K h_c$, since $\hat{f}_\epsilon$ is a convex combination of $p_{-2}(x^*)$, $p_0(x^*)$ and $p_2(x^*)$ for any value of X^3. We therefore assume in the following that $h < K h_c$. The proof consists of an extensive case study, visualized in Figure 15.

Case 1: In this case $\alpha = 1$ and $\beta = 1$, therefore $\hat{f}_\epsilon = p_{-2}(x^*) = f^*$.

Case 2: We have that $\alpha = 1$ and $0 < X_3^3 < \epsilon$, therefore $\hat{f}_\epsilon = (1-\beta)p_2(x^*) + \beta p_{-2}(x^*)$ where β can take any value in $(0,1)$. From (5.3), it follows that the condition $0 < X_3^3 < \epsilon$ corresponds to

$$0 < |b_{-2} - b_2| - x^*|a_{-2} - a_2| < \epsilon, \tag{E.1}$$

where a_{-2} and a_2 are the slopes of p_{-2} and p_2, respectively. Recall that in (5.3) the assumption that $x_i^k = i$ was made. It can however be seen that X_3^3 is invariant to grid translations and scaling. Indeed, when the grid size is scaled by h one needs to replace x^* by hx^* and $a_{\pm 2}$ by $a_{\pm 2}/h$; this leaves X_3^3 unchanged. If we now observe

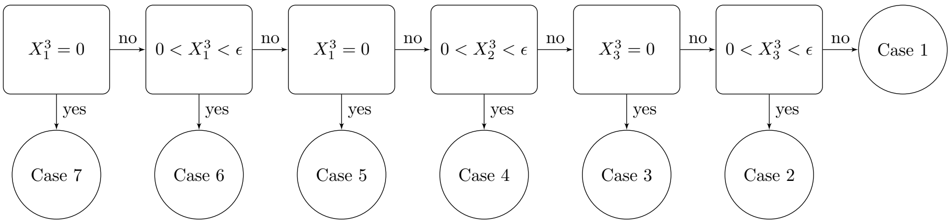


FIGURE 15. Overview of the case study done in the proof of Theorem 5.4.

that from $\alpha = 1$ follows that $|a_{-2} - a_2| \neq 0$, then we can define y as the unique intersection point of p_{-2} and p_2, and obtain

$$0 < y - x^* < \frac{\epsilon}{|a_{-2} - a_2|}. \tag{E.2}$$

Furthermore note that we can write $p_{\pm 2}(x) = a_{\pm 2}(x-y) + p_{\pm 2}(y)$ where by definition $p_{-2}(y) = p_2(y)$. This leads to the estimate

$$|p_{-2}(x^*) - p_2(x^*)| = |a_{-2} - a_2|(y - x^*) < \epsilon. \tag{E.3}$$

Case 3: In this case $\alpha = 1$ and $\beta = 0$, therefore $\hat{f}_\epsilon = p_2(x^*) = f^*$.

Case 4: By looking at the definition of X_2^3 in (5.3), we see that $0 < h|(a_{-2} - a_2)| < \epsilon$ (where the factor h was added to remove the assumption that $x_i^k = i$). Furthermore we have that $X_1^3 > 0$. If $z \notin I_{-1} \cup I_0 \cup I_1$ then Lemma D.1 and Lemma D.3 guarantee that $\hat{f}_\epsilon$ is a second-order accurate approximation of f^*. If $z \in I_{\pm 1}$, then p_0 is a second-order accurate approximation of $p_{\mp 2}$ on I_0. In addition, this leads to the bound

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_{\mp 2}(x^*)| + |p_{\mp 2}(x^*) - p_0(x^*)| \\ &= |(a_2 - a_{-2})(x^* - y)| + |p_{\mp 2}(x^*) - p_0(x^*)| \\ &\leqslant \frac{3}{2}\epsilon + Ch^2 \sup|f''|. \end{aligned} \tag{E.4}$$

Finally we treat the case where $z \in I_0$. Define p_0^* as the affine function through $(x_{-1}, p_{-2}(x_{-1}))$ and $(x_0, p_2(x_0))$. It then follows from $h|(a_{-2} - a_2)| < \epsilon$ that $h|(a_{\pm 2} - a_0^*)| < \epsilon$ where a_0^* is the slope of p_0^*. We also have that p_0^* is a second-order accurate approximation of p_0 on the interval I_0. This then leads to

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_0^*(x^*)| + |p_0^*(x^*) - p_0(x^*)| \\ &\leqslant |(a_{\pm 2} - a_0^*)\frac{h}{2}| + |p_0^*(x^*) - p_0(x^*)| \\ &\leqslant \frac{\epsilon}{2} + Ch^2 \sup|f''|. \end{aligned} \tag{E.5}$$

Case 5: In this case $\alpha = 0$ and therefore $\hat{f}_\epsilon = p_0(x^*) = f^*$.

Case 6: In this case we only know that I_0 is a bad interval, since $X_1^3 > 0$. It may or may not contain the discontinuity. If $z \notin I_{-1} \cup I_0 \cup I_1$ then Lemma D.1 and Lemma D.3 guarantee that $\hat{f}_\epsilon$ is a second-order accurate approximation of f^*. We therefore assume in the following that $z \in I_{-1} \cup I_0 \cup I_1$.

In the proof of Theorem 5.1, we introduced the quantity X_q^2 as a smoothness indicator for the interval I_q. We first investigate how the quantities X_q^2 for f are related to the same quantities for the piecewise linear function defined by $g(x) = f(z) + f'(z-)(x-z) + [f'](x-z)_+$, which we will denote by $\widehat{X}_q^2$. Let $\Delta_q = \Delta_h^2 f(x_q)$ and $\widehat{\Delta}_q = \Delta_h^2 g(x_q)$, as defined in (3.3), and define $\Phi_q = \Delta_q - \widehat{\Delta}_q$. Since g is a second-order accurate approximation of f, we obtain that

$$|\Phi_q| \leqslant C_q h^2 \sup|f''|, \tag{E.6}$$

where the constant C_q is independent of f and h. Using the triangle inequality and its reverse, we find that for $m, n \in \mathbb{N}_0$,

$$\begin{aligned} |\widehat{\Delta}_m| - |\widehat{\Delta}_n| &\leqslant |\widehat{\Delta}_m + \Phi_m| - |\widehat{\Delta}_n + \Phi_n| + |\Phi_m| + |\Phi_n| \\ &\leqslant |\Delta_m| - |\Delta_n| + (C_m + C_n) h^2 \sup|f''|. \end{aligned} \tag{E.7}$$

Now assume that $0 < X_q^2 < \epsilon$ for some index q. We can then conclude that

$$0 \leqslant \widehat{X}_q^2 \leqslant X_q^2 + Ch^2 \sup|f''| < \epsilon + Ch^2 \sup|f''|. \tag{E.8}$$

This allows us to restrict our further calculations to the piecewise linear function g. We assume that $z \in I_s$ for some index s, and that $[f'] > 0$. Furthermore we denote by $\overline{x}$ the midpoint of I_s. In Table 7 we calculate $\widehat{X}_{s-1}^2, \widehat{X}_s^2$ and $\widehat{X}_{s+1}^2$. Note that for $s \notin \{-1, 0, 1\}$, second-order accuracy is immediate.

x_q	x_{s-2}	x_{s-1}	x_s	x_{s+1}
$\lvert\widehat{\Delta}_q\rvert$	0	$[f'](x_s - z)$	$[f'](z - x_{s-1})$	0
$(\lvert\widehat{\Delta}_q\rvert - M_q)_+$	0	$2[f'](\overline{x} - z)_+$	$2[f'](z - \overline{x})_+$	0
$(\lvert\widehat{\Delta}_q\rvert - N_q^+)_+$	0	$2[f'](\overline{x} - z)_+$	$[f'](z - x_{s-1})$	0
$(\lvert\widehat{\Delta}_q\rvert - N_q^-)_+$	0	$[f'](x_s - z)$	$2[f'](z - \overline{x})_+$	0
$\widehat{X}_{s-1}^2, \widehat{X}_s^2, \widehat{X}_{s+1}^2$		$2[f'](\overline{x} - z)_+$	$[f'](2\lvert\overline{x} - z\rvert + \min\{x_s - z, z - x_{s-1}\})$	$2[f'](z - \overline{x})_+$

TABLE 7. Calculation of $\widehat{X}_{s-1}^2, \widehat{X}_s^2$, and $\widehat{X}_{s+1}^2$.

First assume that $s = 0$, in this case $x^* = \overline{x}$. From the table, it follows that $\widehat{X}_0^2 \geqslant [f']h/2$. Combining this with (E.8) leads to the bound $[f']h/2 < \epsilon + Ch^2 \sup|f''|$. We define again p_0^* as the affine function through $(x_{-1}, p_{-2}(x_{-1}))$ and $(x_0, p_2(x_0))$, which is a second-order accurate approximation of p_0. We also introduce two new second-order accurate approximations of p_0 on I_0,

$$\begin{aligned} p_0^-(x) &= p_{-2}(x) + [f'] \frac{(x_0 - z)_+}{h}(x - x_{-1}), \\ p_0^+(x) &= p_2(x) + [f'] \frac{(z - x_{-1})_+}{h}(x - x_0). \end{aligned} \tag{E.9}$$

Indeed, $p_0^-(x_{-1}) = p_0^*(x_{-1}) = p_{-2}(x_{-1})$ and $p_0^-(x_0) = p_{-2}(x_0) + [f'](x_0 - z)_+$ are both second-order accurate on I_0, therefore p_0^- is a second-order accurate approximation on I_0 of p_0^* and hence of p_0. A similar reasoning holds for p_0^+. This then leads to

$$\begin{aligned} |p_{\pm 2}(x^*) - p_0(x^*)| &\leqslant |p_{\pm 2}(x^*) - p_0^{\pm}(x^*)| + |p_0^{\pm}(x^*) - p_0(x^*)| \\ &\leqslant [f']h/2 + |p_0^{\pm}(x^*) - p_0(x^*)| \\ &\leqslant \epsilon + Ch^2 \sup|f''|. \end{aligned} \tag{E.10}$$

Next we assume that $s = 1$. Since $X_1^3 > 0$, we have that $x_0 \leqslant z < \overline{x}$. We distinguish two subcases. First, if $x_0 \leqslant z \leqslant \overline{x}/2$ then $X_1^3 = 2[f'](\overline{x} - z)_+ \geqslant [f']h/2$. Equation (E.8) is still valid and a calculation as in (E.10) leads to the bound

$$|p_{\pm 2}(x^*) - p_0(x^*)| \leqslant \frac{5}{4}\epsilon + Ch^2 \sup|f''| \tag{E.11}$$

where we used that $(z - x_{-1})_+ \leqslant 5h/4$. Second, if $\overline{x}/2 < z < \overline{x}$ then the intersection point y of p_{-2} and p_2 will be located in I_1. This result can be deduced from the proof of Lemma 3 in [1], by taking $I = I_s$ in the beginning of the proof. The second-order accuracy then follows from Case 1 or Case 2, depending on the value of X_3^3. The case $s = -1$ is completely analogous.

Case 7: In this case $\alpha = 0$ and therefore $\hat{f}_\epsilon = p_0(x^*) = f^*$.

We can now summarize all seven cases by the error bound

$$|\hat{f}_\epsilon - f^*| \leqslant Ch^2 \sup|f''| + \frac{3}{2}\epsilon, \tag{E.12}$$

which concludes the proof.

Appendix F. Weights of trained DeLENO networks

We can now compare the weights and biases of the trained networks to the theoretical ones from Section 4. As the networks do not have an accuracy of 100%, it comes as no surprise that these do not agree. We list the obtained weights and biases for the trained third-order DeLENO interpolation network.

$$\begin{aligned} W^1 &= \begin{pmatrix} 1.1951 & 2.0433 & -11.7410 & 5.6383 \\ 2.9216 & -2.8703 & -2.5077 & 2.4624 \\ -2.2775 & 7.6890 & -7.2667 & 2.4914 \\ 3.2909 & -5.8431 & -5.6085 & 3.4171 \end{pmatrix}, \quad b^1 = \begin{pmatrix} -0.1069 \\ -0.3615 \\ 0.0389 \\ 0.0605 \end{pmatrix}, \\ W^2 &= \begin{pmatrix} -11.6122 & 4.2986 & 10.7356 & 8.1240 \\ 11.5929 & -4.2767 & -10.7357 & -8.1316 \end{pmatrix}, \quad b^2 = \begin{pmatrix} -2.5193 \\ 2.4493 \end{pmatrix}. \end{aligned} \tag{F.1}$$

The theoretical counterparts can be found in equations (4.3) and (4.4). Below we list the obtained weights and biases for the trained fourth-order DeLENO interpolation network.

$$\begin{aligned} W^1 &= \begin{pmatrix} -0.0559 & -1.0026 & 1.1115 & 1.001 & 1 - 1.0569 & -0.0001 \\ -0.3547 & 0.3557 & -0.8777 & 2.0707 & -1.1983 & -0.0051 \\ -0.0060 & -0.6155 & 1.6342 & -1.4526 & 0.4599 & -0.0370 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0011 & -0.1965 & 0.7964 & -1.1817 & 0.7805 & -0.1913 \\ -0.3324 & 1.2088 & -1.6946 & 1.0632 & -0.2479 & -0.0043 \\ 0.1432 & -0.6459 & 0.9448 & -0.5434 & 0.1271 & -0.0154 \\ -0.0076 & -0.4239 & 0.8604 & -0.0248 & -0.8755 & 0.4517 \\ 0.0196 & -0.1527 & 0.6286 & -0.9194 & 0.6069 & -0.1619 \\ -0.0088 & -0.2571 & 1.0627 & -1.6288 & 1.0899 & -0.2714 \end{pmatrix}, b^1 = \begin{pmatrix} -0.0005 \\ 0.0029 \\ -0.0005 \\ -0.1217 \\ 0.0012 \\ 0.0007 \\ -0.0010 \\ 0.0010 \\ 0.0027 \\ -0.0005 \end{pmatrix}, \\ W^2 &= \begin{pmatrix} 0 & 1.6803 & 0.2910 & -3.1738 & 0 & 1.5946 \\ 0 & -1.9935 & -0.7975 & 3.2015 & 0 & -1.9124 \\ 0 & -0.3125 & 2.7085 & 0.7264 & 0 & -0.2819 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.3886 & 0.6976 & 0.6071 & 0 & -1.3514 \\ 0 & 2.1097 & 0.8511 & -3.6655 & 0 & 2.0568 \\ 0 & 0.6645 & 0.1485 & -1.2784 & 0 & 0.5052 \\ 0 & -0.0061 & -1.4376 & -0.0073 & 0 & -0.0082 \\ 0 & 0.4580 & -1.0432 & 0.0474 & 0 & 0.4542 \\ 0 & 0.6597 & -2.6379 & -0.4588 & 0 & 0.7357 \end{pmatrix}^T, \quad b^2 = \begin{pmatrix} -0.0349 \\ 0.0993 \\ 0.0463 \\ 0.0284 \\ -0.0570 \\ 0.0438 \end{pmatrix}, \end{aligned} \tag{F.2}$$

(F.3)

$$W^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8.0933 & 0.6463 & -5.7734 & 0 & 8.3502 \\ 0 & 2.0780 & -10.0148 & -0.3565 & 0 & 1.8241 \end{pmatrix}, \quad b^2 = \begin{pmatrix} -0.0316 \\ -0.0432 \\ -0.3800 \\ 0.4131 \end{pmatrix},$$

$$W^4 = \begin{pmatrix} 0 & 0 & 2.3377 & -11.5452 \\ 0 & 0 & 2.2965 & 11.1520 \\ 0 & 0 & -12.6485 & -0.8555 \end{pmatrix}, \quad b^4 = \begin{pmatrix} 1.6841 \\ -7.5807 \\ 8.2575 \end{pmatrix}.$$

Note that these matrices and vectors again differ significantly from the theoretical weights and biases from equations (4.5-4.7). This shows that there are multiple neural networks which can approximate the ENO interpolation procedure very well.