

On the approximation of rough functions with deep neural networks

T. DeRyck and S. Mishra and D. Ray

Research Report No. 2020-07
January 2020

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

ON THE APPROXIMATION OF ROUGH FUNCTIONS WITH DEEP NEURAL NETWORKS

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

ABSTRACT. Deep neural networks and the ENO procedure are both efficient frameworks for approximating rough functions. We prove that at any order, the ENO interpolation procedure can be cast as a deep ReLU neural network. This surprising fact enables the transfer of several desirable properties of the ENO procedure to deep neural networks, including its high-order accuracy at approximating Lipschitz functions. Numerical tests for the resulting neural networks show excellent performance for approximating solutions of nonlinear conservation laws and at data compression.

CONTENTS

1. Introduction	1
2. Deep neural networks	3
3. ENO interpolation	3
4. ENO reconstruction	4
5. ENO as a ReLU DNN	6
6. ENO-SR interpolation	10
7. Numerical results	12
8. Discussion	23
References	24
Appendix A. ENO coefficients	25
Appendix B. Multi-resolution representation of functions for data compression	25
Appendix C. Proof of Theorem 6.1	27

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, [Daf10]. Similarly, solutions to the incompressible Euler equations would well be only Hölder continuous in the turbulent regime, [ES06]. Moreover, solutions of fully non-linear PDEs such as Hamilton-Jacobi equations are in general Lipschitz continuous, [Eva98]. Images constitute another class of rough or rather piecewise smooth functions as they are often assumed to be no

(T. De Ryck and S. Mishra) SEMINAR FOR APPLIED MATHEMATICS, ETH ZÜRICH, RÄMISTRASSE 101, 8092 ZÜRICH, SWITZERLAND

(D. Ray) DEPARTMENT OF COMPUTATIONAL & APPLIED MATHEMATICS, RICE UNIVERSITY, HOUSTON, TEXAS

E-mail addresses: `deryckt@ethz.ch`, `siddhartha.mishra@sam.math.ethz.ch`, `deep.ray@rice.edu`.

more regular than functions of bounded variation on account of their sharp edges, [AK06].

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, [ACDD05] 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, [HSW89, Bar93, Cyb88] and references therein. This implies that for any piecewise smooth function, there exists a neural network that approximates it with very high accuracy. However, the precise architecture of this network is not specified in these universality results. Recently [Yar17] was able to construct deep neural networks with ReLU activation functions that can approximate Lipschitz functions to second-order accuracy. He was able to provide very precise estimates on the type and architecture of the underlying networks. Even more surprisingly, in a very recent paper, [YZ] 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 $f : D \rightarrow \mathbb{R}$. 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 , [GBC16]. Although it makes sense to train networks for each individual function f in high dimensions, for instance in the context of uncertainty quantification of PDEs, [LMR] 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 $x_i \in D$. Hence, one would like to construct neural networks that map the input vector into an output interpolant (or its evaluation at certain sampling points). It is unclear if the networks proposed by Yarotsky can be adapted to this setting.

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 procedure is provided by the essentially non-oscillatory (ENO) procedure. First developed in the context of reconstruction of non-oscillatory polynomials from cell averages in [HEOC87], ENO was also adapted for interpolating rough functions in [SO89] and references therein. Once augmented with a sub-cell resolution (SR) procedure of [Har89], it was provided in [ACDD05] 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*, [FMT13]. Given these desirable properties, it is not surprising that the ENO procedure has been every successfully employed in the numerical approximation of hyperbolic systems of

conservation laws [HEOC87], Hamilton-Jacobi equations [SO91] and in data compression in image processing, [HEOC97, ACDD05] and references therein.

As both deep neural networks and ENO procedure seem to provide frameworks for the efficient approximation of rough functions, it is natural to explore links between these two a priori disconnected concepts. This article attempts to do so and we present the following results herein,

- 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 [Har89] can 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 numerical methods for conservation laws and for data and image compression.

2. DEEP NEURAL NETWORKS

We consider an underlying function $\mathcal{L} : \mathcal{M} \subset \mathbb{R}^m \rightarrow \mathcal{N} \subset \mathbb{R}^n$ and we assume access to a set of labelled data $\mathbb{S} \subset \{(X, \mathcal{L}(X)) : X \in \mathcal{M}\}$, 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 \mathcal{M} with a high degree of accuracy.

One possible function class is those of deep neural networks. 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 ,

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

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 $L \geq 3$ 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,

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

Additionally, the signal vector Z^L may have to pass through an output function \mathcal{S} to convert the signal into a meaningful form. For instance in classification problems [GBC16], a suitable choice is the *softmax* function

$$\mathcal{S}(Z)_i = \frac{e^{Z_i}}{\sum_{j=1}^n e^{Z_j}}, \quad i = 1, \dots, n.$$

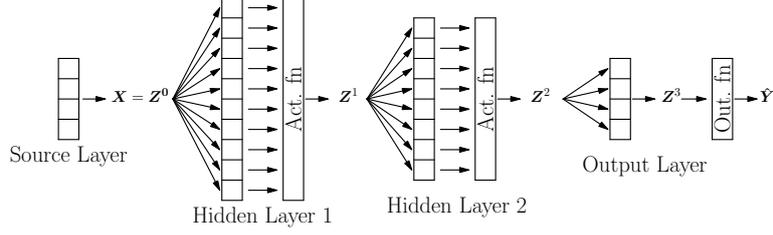


FIGURE 1. An MLP with 2 hidden layers. The source layer transmits the signal X to the first hidden layer. The final output of the network is \hat{Y} .

3. ENO INTERPOLATION

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 (for simplicity) grids $\{\mathcal{T}^k\}_{k=0}^K$ on Ω , where

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

for $0 \leq i \leq N_k$, $0 \leq k \leq K$ 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 $f_{-p+2}^k, \dots, f_{-1}^k$ and $f_{N_k+1}^k, \dots, f_{N_k+p-2}^k$ be suitably prescribed ghost values. We are interested in finding an interpolation operator \mathcal{I}^{h_k} such that

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

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

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

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. Algorithm 1 describes how the stencil shift r_i^k corresponding to this stencil can be obtained. Note that r_i^k uniquely defines the polynomial p_i^k . We can then write the final interpolant as

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

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

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

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 A.

Remark 3.1. *ENO was initially introduced by [HEOC87] for high-order accurate piecewise polynomial reconstruction, given cell averages of a function. This allows to develop 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 Section 4.*

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 [HEOC97]. As we will use ENO interpolation for data compression in Section 7, we refer to Appendix B for details on multi-resolution representations.*

Algorithm 1: ENO interpolation stencil selection

Input: ENO order p , input array $\Delta^0 = \{f_{i+j}^k\}_{j=-p+1}^{p-2}$, for any $0 \leq i \leq N_k$.

Output: Stencil shift r .

Evaluate Newton undivided differences:

for $j = 1$ **to** $p - 1$ **do**

$\Delta^j = \Delta^{j-1}[2 : \text{end}] - \Delta^{j-1}[1 : \text{end} - 1]$

Find shift:

$r = 0$

for $j = 2$ **to** $p - 1$ **do**

if $|\Delta^j[p - 2 - r]| < |\Delta^j[p - 1 - r]|$ **then**
 $r = r + 1$

return r

4. ENO RECONSTRUCTION

In this section we present ENO reconstruction, which is very similar to ENO interpolation as presented in Section 3. Let V be a function on $[c, d]$. We define a uniform mesh \mathcal{T} on $[c, d]$ with N cells,

$$(4.1) \quad \mathcal{T} = \{I_i\}_{i=1}^N, \quad I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}], \quad \left\{ x_i = c + (2i-1)\frac{h}{2} \right\}_{i=0}^N, \quad h = \frac{(d-c)}{N},$$

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

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

and we define $\bar{V}_{-p+2}, \dots, \bar{V}_0$ and $\bar{V}_{N+1}, \dots, \bar{V}_{N+p-1}$ 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

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

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,

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

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

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

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 $\Delta^0 = \{\bar{V}_{i+j}\}_{j=-p+1}^{p-1}$, for any $1 \leq i \leq N$.

Output: Stencil shift r .

Evaluate Newton undivided differences:

for $j = 1$ **to** $p - 1$ **do**

$\Delta^j = \Delta^{j-1}[2 : \text{end}] - \Delta^{j-1}[1 : \text{end} - 1]$

Find shift:

$r = 0$

for $j = 1$ **to** $p - 1$ **do**

if $|\Delta^j[p - 1 - r]| < |\Delta^j[p - r]|$ **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

$$(4.2) \quad \mathcal{I}_i^h V(x_{i+\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i, j}^p \bar{V}_{i-r_i+j} \quad \text{and} \quad \mathcal{I}_i^h V(x_{i-\frac{1}{2}}) = \sum_{j=0}^{p-1} \tilde{C}_{r_i-1, j}^p \bar{V}_{i-r_i+j},$$

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 A.

5. ENO AS A RELU DNN

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 .

Theorem 5.1. *There exists a ReLU neural network consisting of $p + \left\lceil \log_2 \left(\left\lfloor \frac{p-2}{2} \right\rfloor \right) \right\rceil$ hidden layers and a suitable output function, that takes input $\Delta^0 = \{f_{i+j}^k\}_{j=-p+1}^{p-2}$ and leads to exactly the same stencil shift as the one obtained by Algorithm 1.*

Proof. We 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 $-p+1 \leq j \leq p-2$ and $0 \leq i \leq N_k$, 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. Furthermore 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. 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 2. 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 this reason, we define the functions $H_1 : \mathbb{R} \rightarrow \mathbb{R}$ and $H_2 : \mathbb{R} \rightarrow \mathbb{R}$ by

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

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 $W^2 \in \mathbb{R}^{3L \times 2K}$ and bias vector $b^2 \in \mathbb{R}^{3L}$,

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

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

$$X_j^3 = \begin{cases} H_1(X_j^2) = Z_j^2 - Z_{L+j}^2 & 1 \leq j \leq L \\ H_2(X_{j-L}^2) = Z_j^2 - Z_{L+j}^2 - 1 & L+1 \leq j \leq 2L. \end{cases}$$

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

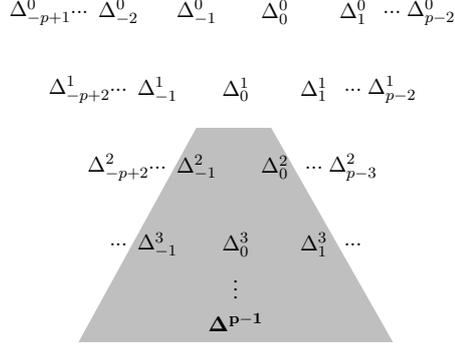


FIGURE 2. Only undivided differences in the shaded region are compared in Algorithm 1.

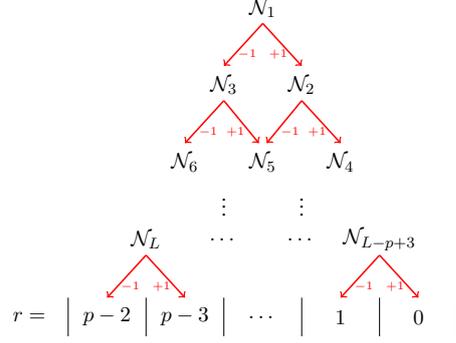


FIGURE 3. Arrangement of N_1, \dots, N_L into directed acyclic graph.

X^3 in a triangular directed acyclic graph, shown in Figure 3, where every node N_j corresponds to the tuple $(X_j^3, X_{j+L}^3) = (H_1(X_j^2), H_2(X_j^2))$. We note that this tuple is either of the form $(+1, H_2(X_j^2))$ or $(H_1(X_j^2), -1)$. Algorithm 1 is equivalent to finding a path from the top node to one of the bins on the bottom. Starting from N_1 , we move to the closest element to the right in the row below (i.e. N_2) if N_1 is of the form $(+1, H_2(X_j^2))$. If N_1 is of the form $(H_1(X_j^2), -1)$, we move to the closest element to the left in the row below (i.e. N_3). If 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 $\mathcal{P} = (X_{j_1}^3, \dots, X_{j_{p-2}}^3)$ that leads to bin r . We define for this path a weight vector $W \in \{-1, 0, 1\}^{2L}$ whose elements are set as

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

For this particular weight vector and for any possible $X^3 \in \mathbb{R}^{2L}$ we have $W \cdot X^3 \leq p - 2$, 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 $\hat{X}^3 = \widehat{W}^3 X^3 + \hat{b}^3 = \widehat{W}^3 (\widehat{W}^3 Z^2 + \hat{b}^3) + \hat{b}^3$. By construction, $\hat{X}_j^3 = 2p - 4$ if and only if path j corresponds to a suitable ENO stencil shift, otherwise $0 \leq \hat{X}_j^3 < 2p - 4$.

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 $a, b \geq 0$, one finds that the calculation of \hat{Y} requires $p-2 + \left\lceil \log_2 \left(\binom{p-2}{\lfloor \frac{p-2}{2} \rfloor} \right) \right\rceil$ 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

$$(5.1) \quad R = \operatorname{argmax}_j \hat{Y}_j - 1 \quad \text{and} \quad r = \min R = \min \operatorname{argmax}_j \hat{Y}_j - 1.$$

□

Remark 5.2. *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 architecture is given by:

- Input $X = (\Delta_{-2}^0, \Delta_{-1}^0, \Delta_0^0, \Delta_1^0)^\top$.
- The first hidden layer is identical to the one described in the original proof of Theorem 5.1 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{4 \times 4}$,

$$(5.2) \quad W^1 = \begin{pmatrix} 0 & 1 & -2 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 2 & -1 & 0 \end{pmatrix}, \quad b^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

- Output layer:

$$(5.3) \quad W^2 = \begin{pmatrix} -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}, \quad b^2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

- The shift is determined using (5.1), since

$$\hat{Y} = \begin{pmatrix} |\Delta_{-1}^2| - |\Delta_0^2| \\ |\Delta_0^2| - |\Delta_{-1}^2| \end{pmatrix}.$$

For $p = 4$, Algorithm 1 can be represented by following ReLU network with 3 hidden layers:

- Input $X = (\Delta_{-3}^0, \Delta_{-2}^0, \Delta_{-1}^0, \Delta_0^0, \Delta_1^0, \Delta_2^0)^\top$.

- The first hidden layer is identical to the one described in the original proof of Theorem 5.1 for $p = 4$, with a null bias vector and $W^1 \in \mathbb{R}^{10 \times 6}$,

$$(5.4) \quad W^1 = \begin{pmatrix} \widetilde{W}^1 \\ -\widetilde{W}^1 \end{pmatrix} \in \mathbb{R}^{10 \times 6} \quad \text{where} \quad \widetilde{W}^1 = \begin{pmatrix} 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -3 & 1 \\ 0 & -1 & 3 & -3 & 1 & 0 \\ -1 & 3 & -3 & 1 & 0 & 0 \end{pmatrix}.$$

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

$$(5.5) \quad W^2 = \begin{pmatrix} \widetilde{W}^2 \\ -\widetilde{W}^2 \end{pmatrix} \in \mathbb{R}^{6 \times 10} \quad \text{where} \quad \widetilde{W}^2 = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Note that $\widetilde{W}^2 \in \mathbb{R}^{3 \times 10}$ is as in the original proof of Theorem 5.1 for $p = 4$.

- The third hidden layer:

$$(5.6) \quad W^3 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ -1 & 1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad b^3 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

- Output layer:

$$(5.7) \quad W^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad b^4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

- After an elementary, yet tedious case study, one can show that the shift can again be determined using (5.1).

Remark 5.3. *Similarly, one can show that Algorithm 2 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.*

6. ENO-SR INTERPOLATION

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 [Har89], one can use sub-cell resolution, together with ENO interpolation, to obtain second-order accurate approximation of such functions [ACDD05]. Our goal is recast a variant of this procedure as a deep neural network in order to ensure second-order accurate approximation. 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 $[f'] = f'(z+) - f'(z-)$.

6.1. Adapted second-order ENO-SR algorithm. The first step is to label intervals that might contain the singular point z as B (bad), other intervals get the label G (good). We use second order differences $\Delta_{h_k}^2 f(x) := f(x-h) - 2f(x) + f(x+h)$ as smoothness indicators. The rules of the ENO-SR detection mechanism are the following:

- (1) The intervals I_{i-1}^k and I_i^k are labelled B if

$$|\Delta_{h_k}^2 f(x_{i-1}^k)| > \max_{n=1,2,3} |\Delta_{h_k}^2 f(x_{i-1 \pm n}^k)|.$$

(2) Interval I_i^k is labelled B if

$$|\Delta_{h_k}^2 f(x_i^k)| > \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i+n}^k)| \quad \text{and} \quad |\Delta_{h_k}^2 f(x_{i-1}^k)| > \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i-1-n}^k)|.$$

(3) All other intervals are labelled G .

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 $p_i^k : [c, d] \rightarrow \mathbb{R} : x \mapsto a_i x + b_i$ the linear interpolation of the endpoints of I_i^k , where we write a_i and b_i instead of a_i^k and b_i^k to simplify notation. The interpolation procedure is as follows:

(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

$$\mathcal{I}_i^{h_k} f(x) = p_{i-2}^k(x) \mathbb{1}_{[c,y]}(x) + p_{i+2}^k(x) \mathbb{1}_{[y,d]}(x).$$

Otherwise 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 6.1. *Let f 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 . The adapted ENO-SR interpolant $\mathcal{I}^h f$ satisfies*

$$\|f - \mathcal{I}^h f\|_\infty \leq Ch^2 \sup_{\mathbb{R} \setminus \{z\}} |f''|$$

for all $h > 0$, with $C > 0$ independent of f .

Proof. The proof is an adaptation of the proof of Theorem 1 in [ACDD05] and can be found in Appendix C. \square

6.2. Second-order ENO-SR as DNN. Assume the setting of Section 3. We will now prove that a second-order accurate prediction of f^{k+1} can be obtained given f^k using a ReLU DNN. 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. Equation (3.2) shows that we only need to calculate $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ for every $1 \leq i \leq N_k$. From the ENO-SR interpolation procedure it is clear that for every i there exists $r_i^k \in \{-2, 0, 2\}$ such that $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1}) = p_{i+r_i^k}^k(x_{2i-1}^{k+1})$. It is thus straightforward to calculate $\mathcal{I}_i^{h_k} f(x_{2i-1}^{k+1})$ from r_i^k .

Theorem 6.2. *There exists a ReLU neural network with a discontinuous output function, input f^k and 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 $0 \leq i \leq N_k$.

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 $0 \leq i \leq N_k$. Using a simple affine transformation, we can obtain $X^1 \in \mathbb{R}^{N_k-1}$ such that $X_i^1 = \Delta_{h_k}^2 f(x_i^k)$ for all $1 \leq i \leq N_k-1$. We now define the following quantities,

$$(6.1) \quad M_i = \max_{n=1,2,3} |\Delta_{h_k}^2 f(x_{i \pm n}^k)| = \max_{n=1,2,3} |X_{i \pm n}^1|, \quad N_i^\pm = \max_{n=1,2} |\Delta_{h_k}^2 f(x_{i \pm n}^k)| = \max_{n=1,2} |X_{i \pm n}^1|,$$

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

(6.2)

$$X_i^2 = (\min\{|X_i^1| - N_i^+, |X_{i-1}^1| - N_{i-1}^-\})_+ + (|X_i^1| - M_i)_+ + (|X_{i-1}^1| - M_{i-1})_+$$

for $5 \leq i \leq N_k-4$. Furthermore we set $X_1^2 = X_2^2 = X_3^2 = X_4^2 = X_{N_k-3}^2 = X_{N_k-2}^2 = X_{N_k-1}^2 = X_{N_k}^2 = 0$. Note that the first term of the sum will be strictly positive if I_i^k is labelled bad by the second rule of the detection mechanism and one of the other terms will be strictly positive if I_i^k is labelled bad by the first rule. Good intervals I_i^k have $X_i^2 = 0$.

Now define $n_{i,l} = l + 4(i-1)$ for $1 \leq i \leq N_k$ and $1 \leq l \leq 4$. Using this notation, i refers to the interval I_i^k . Define $X^3 \in \mathbb{R}^{4N_k}$ in the following manner:

$$(6.3) \quad \begin{aligned} X_{n_{i,1}}^3 &= X_i^2, & X_{n_{i,3}}^3 &= (|b_{i-2} - b_{i+2}| - x_{2i-1}^{k+1} |a_{i-2} - a_{i+2}|)_+, \\ X_{n_{i,2}}^3 &= |a_{i-2} - a_{i+2}|, & X_{n_{i,4}}^3 &= (|-b_{i-2} - b_{i+2}| + x_{2i-1}^{k+1} |a_{i-2} - a_{i+2}|)_+, \end{aligned}$$

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

$$(6.4) \quad \hat{Y}_i = \min \operatorname{argmin}_{1 \leq l \leq 4} X_{n_{i,l}}^3.$$

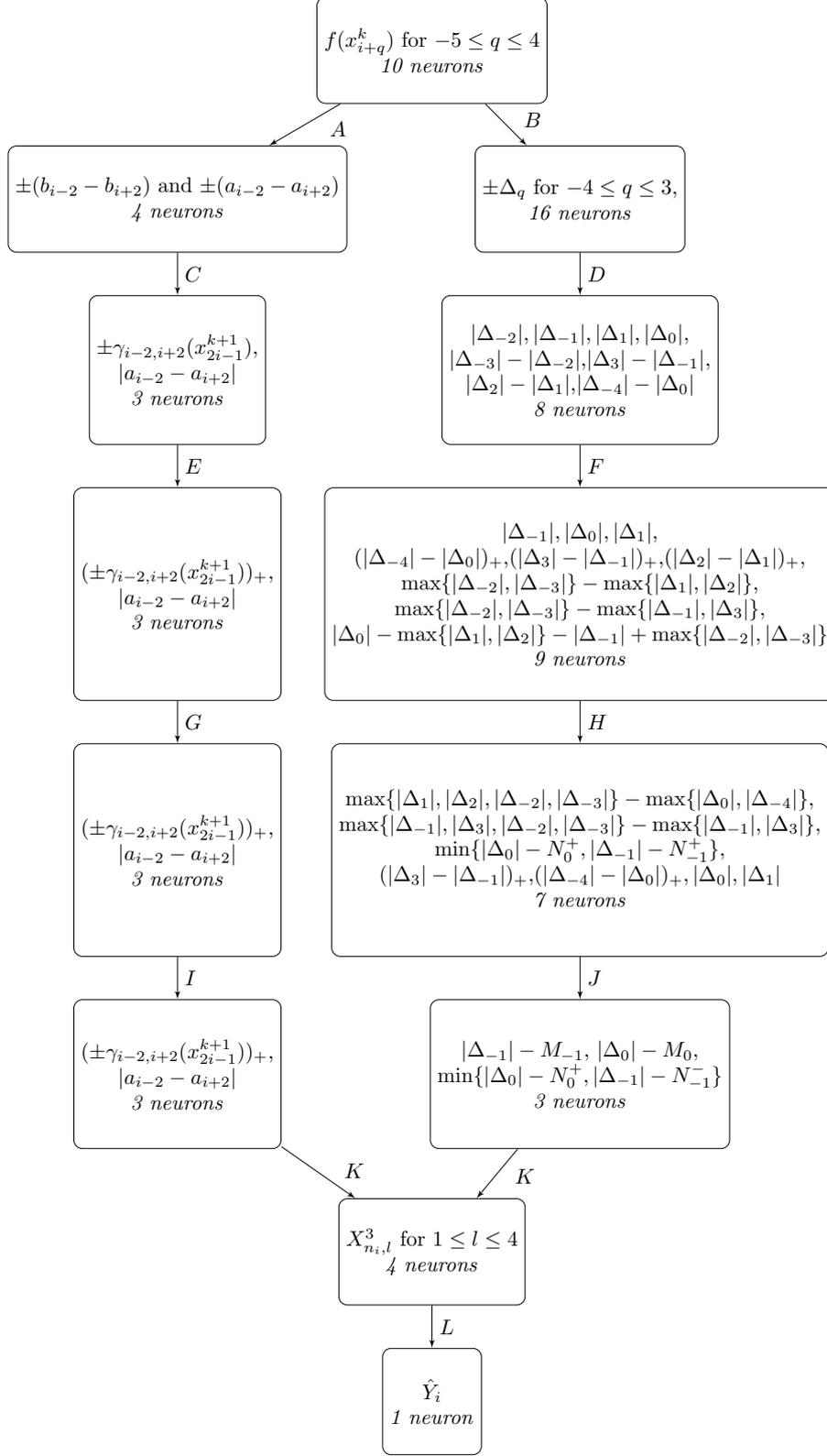
It remains to prove that r_i^k can be obtained from \hat{Y}_i . Note that $\hat{Y}_i = 1$ if and only if I_i^k was labelled G . Therefore $\hat{Y}_i = 1$ corresponds to $r_i^k = 0$. If $\hat{Y}_i = 2$, then I_i^k was labelled B and the interpolants p_{i-2}^k and p_{i+2}^k do not intersect, leading to $r_i^k = 0$ according to the interpolation procedure. Next, $\hat{Y}_i = 3, 4$ corresponds to the case where I_i^k was labelled B and the interpolants p_{i-2}^k and p_{i+2}^k 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_{2i-1}^{k+1} is right of y and therefore $r_i^k = 2$. Analogously, $\hat{Y}_i = 4$ corresponds to $r_i^k = -2$, which concludes the proof. \square

Now that we have established that our adaptation of the second-order ENO-SR algorithm can be written as a ReLU DNN 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 4. We now give some more explanation about how each layer in Figure 4 can be calculated from the previous layer, where we use the same notation as in the proof of Theorem 5.1. In addition, we define and note that

$$(6.5a) \quad \gamma_{i-2,i+2}(z) = |b_{i-2} - b_{i+2}| - z|a_{i-2} - a_{i+2}|,$$

$$(6.5b) \quad \max\{x, y\} = x + (y - x)_+.$$

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 (6.5a) on $\pm(b_{i-2} - b_{i+2})$ and $\pm(a_{i-2} - a_{i+2})$. **D.** Straightforward application of the identity

FIGURE 4. Flowchart of a ReLU DNN to calculate \hat{Y}_i from f^k .

$|x| = (x)_+ + (-x)_+$ 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 (6.5b), where the order of the arguments of the maximums is carefully chosen. **H.** Equation (6.5b) was used, where we use that $\min\{x, y\} = -\max\{-x, -y\}$. **J.** Application of equation (6.1). **K.** The result follows from combining definitions (6.2) and (6.3). **L.** As can be seen in definition (6.4), \hat{Y}_i is obtained by applying the output function $\min \operatorname{argmin}$ on the output layer.

Remark 6.3. *The second-order ENO-SR method as proposed in [ACDD05] can also be written as a ReLU DNN, but it leads to a neural network that is considerably larger than the one presented above.*

7. NUMERICAL RESULTS

From sections 5 and 6, we know that there exist deep ReLU neural networks, of a certain architecture (number of hidden layers and neurons per layer) that will mimic the ENO- p and the second-order ENO-SR-2 algorithms for interpolating rough functions. Next, we proceed to find such networks, which we label as DeLENO (deep learning ENO) and DeLENO-SR, using the following training procedure.

7.1. Training procedure. The *training* of these networks involves finding a parameter θ (the weights and biases of the network) that approximately minimizes a certain *loss function* that measures the error in the network's predictions. For classification problems, the *cross-entropy function* with regularization term

$$(7.1) \quad \mathcal{J}(\theta; \mathbb{S}, \lambda) = -\frac{1}{\#\mathbb{S}} \sum_{(X^i, Y^i) \in \mathbb{S}} \sum_{j=1}^n Y_j^i \log(\hat{Y}_j^i) + \lambda \mathcal{R}(\theta),$$

is a suitable loss function. The cross-entropy functions 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 [GBC16]. The network hyper-parameter $\lambda > 0$ controls the extent of regularization. Popular choices of θ include the sum of some norm of all the weights of the network. Details on the training of the DeLENO networks can be found in subsections 7.1.1 and 7.1.2.

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}^θ on \mathbb{T} is measured as

$$(7.2) \quad \mathbb{T}_{acc} = \frac{\#\left\{(X, Y) \in \mathbb{T} \mid \hat{Y} = \mathcal{L}^\theta(X), \arg \max_{1 \leq j \leq n} \hat{Y}_j = \arg \max_{1 \leq j \leq n} Y_j\right\}}{\#\mathbb{T}},$$

with a similar expression for \mathbb{V}_{acc} . The accuracies for DeLENO interpolation and reconstruction are listed in Table 1. It can be seen that DeLENO and ENO agree almost perfectly on the data set.

7.1.1. Training DeLENO- p . We want to construct a suitable training data set \mathbb{S} to train DeLENO- p for interpolation purposes. For small orders $p = 3, 4$ one can take the network architectures described in the beginning of Section 5, for larger orders one can use the architecture from the proof of Theorem 5.1. 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 460,200-200 m using Algorithm 1 with inputs given by the samples below.

(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.

- 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

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

where u_l is defined as

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

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

$$(7.3) \quad \text{Scale}(X) = \begin{cases} \frac{2X - (b+a)}{b-a} & \text{if } X \neq 0 \\ (1, \dots, 1)^\top \in \mathbb{R}^m & \text{otherwise} \end{cases}, \quad a = \min_j(X_j), \quad b = \max_j(X_j),$$

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

Remark 7.1. *When the input data is scaled using formula (7.3), 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 (7.1), with an L_2 penalization of the network weights and $\lambda = 7.8 \cdot 10^{-6}$. The minimization is generally performed using an iterative gradient algorithm. It is common to use a stochastic version in which the data set is shuffled and split into a number of mini-batches of size M , followed by taking an optimization step over each mini-batch. Once all the mini-batches are exhausted, the training is said to have completed one *epoch*. After each epoch, the data set is reshuffled and new mini-batches are created. Introducing such a stochasticity further assists the training algorithm to escape local minima. A single (generic) optimization step is of the form

$$(7.4) \quad \theta_{t+1} = \theta_t - \eta_t \nabla_{\theta} \mathcal{J}(\theta_t; \mathbb{B}_k, \lambda),$$

where η_t is the learning rate and \mathbb{B}_k is a mini-batch of the data set. Several stochastic optimizers have been developed, among which a popular algorithm is the ADAM optimizer [KB14]. We used the ADAM optimizer with a mini-batch size of 1024 and the learning rate adapted at the end of each epoch as

$$(7.5) \quad \eta_t = \eta_{t-1} \frac{1}{1 + \beta t}, \quad \eta_0 = 1.0 \cdot 10^{-3}, \quad \beta = 1.0 \cdot 10^{-5}.$$

The network is retrained using 5 times for 2000 epochs each, with the weights and biases initialized using a random normal distribution. 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 accuracies achieved for DeLENO interpolation networks are given in Table 1.

The training of the DeLENO reconstruction networks was performed entirely analogously, with the only difference that now $m = 2p - 1$. The accuracies on the test and validation sets can be found in Table 1.

We can now compare the weights and biases of the trained networks to the theoretical ones from Section 5. 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.

$$(7.6) \quad \begin{aligned} W^1 &= \begin{pmatrix} 1.1951 & 2.0433 & -11.7410 & 5.6383 \\ 2.9216 & -2.8703 & -2.5077 & 2.4624 \\ -2.2775 & 7.6890 & -7.2667 & 2.4914 \\ 3.2909 & -5.8431 & -5.6085 & 3.4171 \end{pmatrix}, & b^1 &= \begin{pmatrix} -0.1069 \\ -0.3615 \\ 0.0389 \\ 0.0605 \end{pmatrix}, \\ W^2 &= \begin{pmatrix} -11.6122 & 4.2986 & 10.7356 & 8.1240 \\ 11.5929 & -4.2767 & -10.7357 & -8.1316 \end{pmatrix}, & b^2 &= \begin{pmatrix} -2.5193 \\ 2.4493 \end{pmatrix}. \end{aligned}$$

The theoretical counterparts can be found in equations (5.2) and (5.3). Below we list the obtained weights and biases for the trained fourth-order DeLENO interpolation network.

$$(7.7) \quad \begin{aligned} W^1 &= \begin{pmatrix} -0.0559 & -1.0026 & 1.1115 & 1.001 & 1 - 1.0569 & -0.0001 \\ -0.3547 & 0.3557 & -0.8777 & 2.0707 & -1.1983 & -0.0051 \\ -0.0060 & -0.6155 & 1.6342 & -1.4526 & 0.4599 & -0.0370 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0011 & -0.1965 & 0.7964 & -1.1817 & 0.7805 & -0.1913 \\ -0.3324 & 1.2088 & -1.6946 & 1.0632 & -0.2479 & -0.0043 \\ 0.1432 & -0.6459 & 0.9448 & -0.5434 & 0.1271 & -0.0154 \\ -0.0076 & -0.4239 & 0.8604 & -0.0248 & -0.8755 & 0.4517 \\ 0.0196 & -0.1527 & 0.6286 & -0.9194 & 0.6069 & -0.1619 \\ -0.0088 & -0.2571 & 1.0627 & -1.6288 & 1.0899 & -0.2714 \end{pmatrix}, & b^1 &= \begin{pmatrix} -0.0005 \\ 0.0029 \\ -0.0005 \\ -0.1217 \\ 0.0012 \\ 0.0007 \\ -0.0010 \\ 0.0010 \\ 0.0027 \\ -0.0005 \end{pmatrix}, \\ W^2 &= \begin{pmatrix} 0 & 1.6803 & 0.2910 & -3.1738 & 0 & 1.5946 \\ 0 & -1.9935 & -0.7975 & 3.2015 & 0 & -1.9124 \\ 0 & -0.3125 & 2.7085 & 0.7264 & 0 & -0.2819 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1.3886 & 0.6976 & 0.6071 & 0 & -1.3514 \\ 0 & 2.1097 & 0.8511 & -3.6655 & 0 & 2.0568 \\ 0 & 0.6645 & 0.1485 & -1.2784 & 0 & 0.5052 \\ 0 & -0.0061 & -1.4376 & -0.0073 & 0 & -0.0082 \\ 0 & 0.4580 & -1.0432 & 0.0474 & 0 & 0.4542 \\ 0 & 0.6597 & -2.6379 & -0.4588 & 0 & 0.7357 \end{pmatrix}^T, & b^2 &= \begin{pmatrix} -0.0349 \\ 0.0993 \\ 0.0463 \\ 0.0284 \\ -0.0570 \\ 0.0438 \end{pmatrix}, \\ W^3 &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8.0933 & 0.6463 & -5.7734 & 0 & 8.3502 \\ 0 & 2.0780 & -10.0148 & -0.3565 & 0 & 1.8241 \end{pmatrix}, & b^2 &= \begin{pmatrix} -0.0316 \\ -0.0432 \\ -0.3800 \\ 0.4131 \end{pmatrix}, \\ W^4 &= \begin{pmatrix} 0 & 0 & 2.3377 & -11.5452 \\ 0 & 0 & 2.2965 & 11.1520 \\ 0 & 0 & -12.6485 & -0.8555 \end{pmatrix}, & b^4 &= \begin{pmatrix} 1.6841 \\ -7.5807 \\ 8.2575 \end{pmatrix}. \end{aligned}$$

Note that these matrices and vectors again differ significantly from the theoretical weights and biases from equations (5.4-5.7). This shows that there are multiple neural networks which can approximate the ENO interpolation procedure very well.

7.1.2. *Training DeLENO-SR.* Next, we construct a suitable training data set \mathbb{S} to train second-order DeLENO-SR for interpolation purposes. The network architecture is described in Section 6. The network will take an input from \mathbb{R}^{10} and predicts the stencil shift r . Recall that ENO-SR is designed to interpolate continuous functions f that are two times differentiable, except at a single point $z \in \mathbb{R}$ where the first derivative has a jump of size $[f']$. Locally, these functions look like piecewise linear functions. Based on this observation, we create a data set using functions of the form

$$(7.8) \quad f(x) = a(x-z)_- + b(x-z)_+,$$

where $a, b, c \in \mathbb{R}$. For notational simplicity we assume that the x -values of the stencil that serves as input for the ENO-SR algorithm (Section 6) are $0, 1, \dots, 9$. 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 way:

- 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 training of DeLENO-SR was performed in a very similar fashion to the training of DeLENO- p (Section 7.1.1), 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 7.2. *Note that the detection mechanism of the ENO-SR interpolation method (Section 6) 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 $\text{round}(\alpha - \beta) > 0$, 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 $\epsilon = 10^{-10}$.*

7.2. Applications.

7.2.1. *Function approximation.* As a first example, we demonstrate the approximating ability of the DeLENO interpolation method using the function

$$(7.9) \quad q(x) = \begin{cases} -x & \text{if } x < 0.5, \\ 3 \sin(10\pi x) & \text{if } 0.5 < x < 1.5, \\ -20(x-2)^2 & \text{if } 1.5 < x < 2.5, \\ 3 & \text{if } 2.5 < x, \end{cases}$$

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 $0 \leq k < K$. As shown in Figure 5, the interpolation with ENO-4 and DeLENO-4 is identical on all grids, for this particular function.

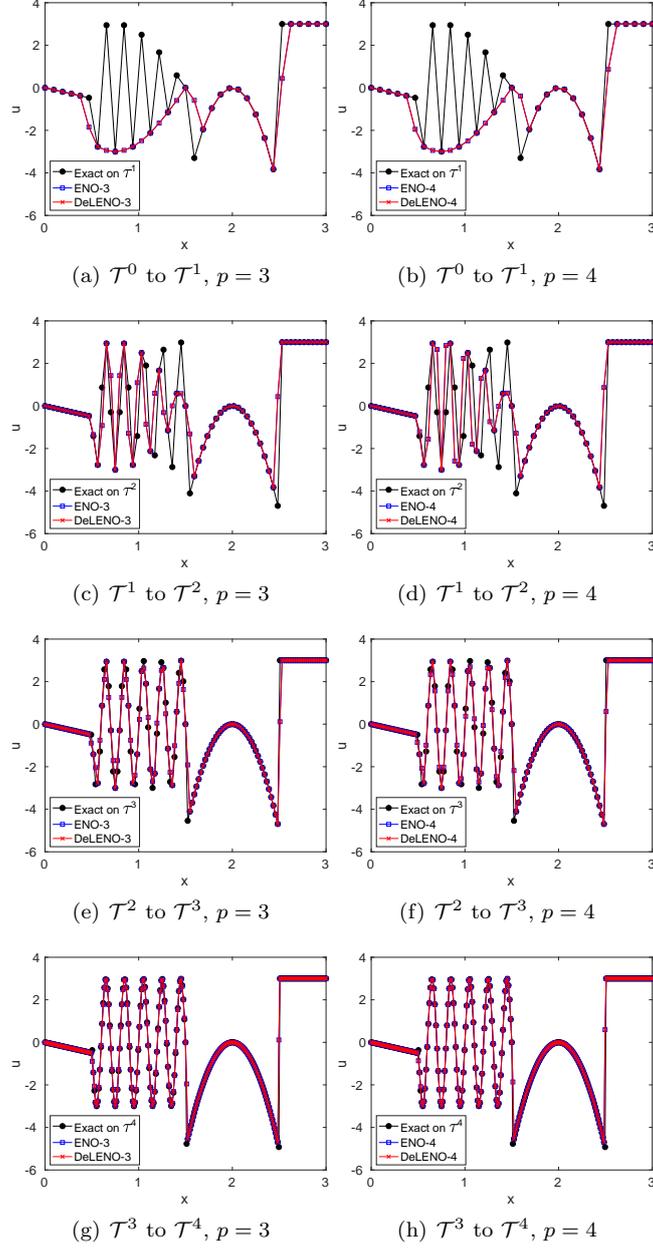


FIGURE 5. Interpolating the function (7.9) using ENO and DeLENO.

Next we investigate the order of accuracy for DeLENO and DeLENO-SR procedures, for the piecewise smooth function

$$(7.10) \quad f(x) = -2 \left(x - \frac{\pi}{6} \right) \mathbb{1}_{[0, \frac{\pi}{6})}(x) + \left(x - \frac{\pi}{6} \right)^2 \mathbb{1}_{[\frac{\pi}{6}, 1]}(x).$$

Note that the first derivative of this function has a jump at $\frac{\pi}{6}$. In Figure 6, the order of accuracy of second-order ENO-SR-2 and DeLENO-SR-2 are compared with

those of ENO-3 and DeLENO-3 for a sequence of grids with $N_0 = 16$ and $K = 11$. As shown in Theorem 6.1, ENO-SR-2 and DeLENO-SR-2 are indeed second-order accurate, whereas ENO and DeLENO reduce to first-order accuracy. Furthermore DeLENO-SR-2 performs better than ENO-SR-2 on coarse grids.

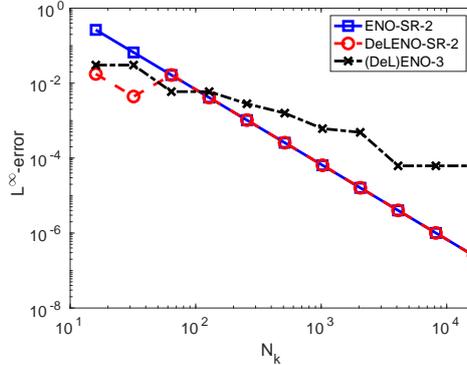


FIGURE 6. Comparison of orders of (DeL)ENO-SR-2 and (DeL)ENO-3 for function (7.10).

7.2.2. *Data compression.* We now apply the multi-resolution representation framework of Appendix B to use DeLENO to compress the function (7.9). We construct a nested sequence of meshes on $[0, 3]$ by choosing $N_0 = 9$ and $K = 5$ in (B.1). We use Algorithm 3 to obtain the multi-resolution representation of the form (B.5) and decode the solution using Algorithm 4 to obtain the approximation \hat{q}^K . The compression thresholds needed for the encoding procedure are set using (B.6).

Figure 7 provides a comparison of the results obtained using different values for the threshold parameters ϵ , and shows the non-zero coefficients \hat{d}^k for each mesh level k . A higher value of ϵ can truncate a larger number of \hat{d}^k components, as is evident for $p = 3$. However, there is no qualitative difference between \hat{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.

p	ϵ	$\ q^K - \hat{q}^K\ _1$		$\ q^K - \hat{q}^K\ _2$		$\ q^K - \hat{q}^K\ _\infty$	
		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 (7.9).

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 $N_k^y = 2^k N_0^y$, for $0 \leq k \leq K$. Let q^k be the data on grid \mathcal{T}^k and denote by \hat{q}^{k+1} the compressed interpolation on grid \mathcal{T}^{k+1} . To obtain \hat{q}^{k+1} , we first interpolate along the x -coordinate direction to obtain an intermediate approximation \tilde{q}^{k+1} of

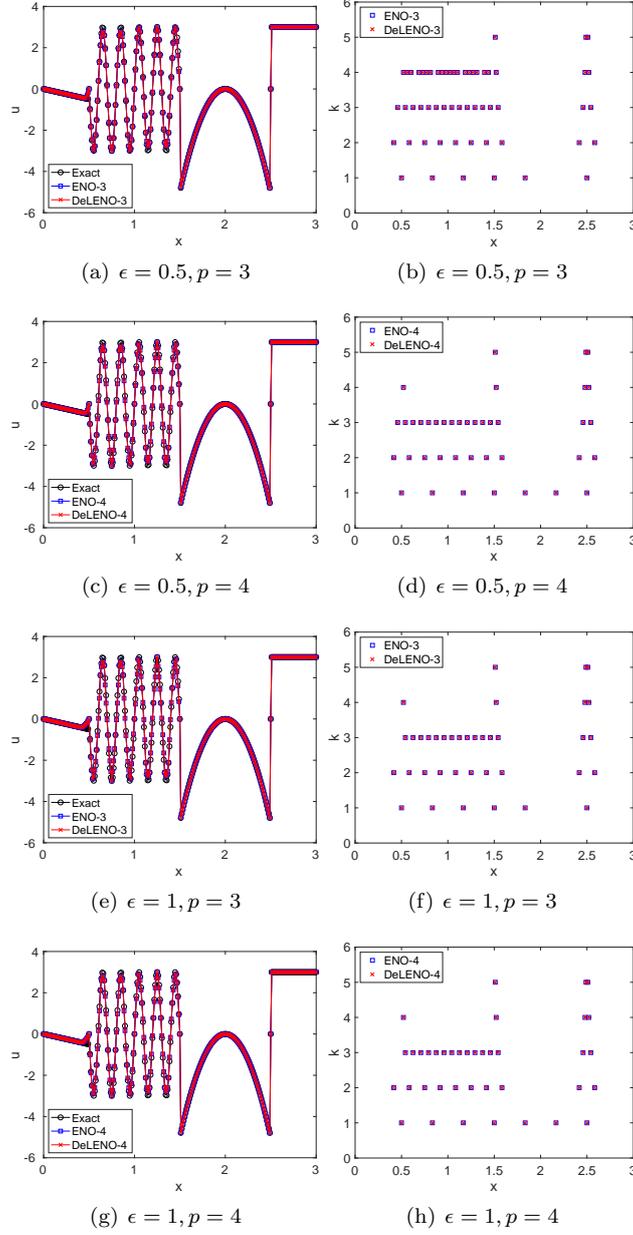


FIGURE 7. Data compression of (7.9) using ENO and DeLENO with $N_0, L = 5$ and $t = 0.5$. Comparison of thresholded decompressed data with the actual data on the finest level (left); Non-zero coefficients \hat{d}^k at each level (right).

size $(N_{k+1}^x + 1) \times (N_k^y + 1)$. Then we use \hat{q}^{k+1} to interpolate along the y -coordinate direction to obtain the final approximation \hat{q}^{k+1} .

We use ENO and DeLENO to compress an image with 705×929 pixels, shown in Figure 8(a). We set $K = 5, \epsilon = 1, t = 0.2$ in equation (B.6). Once again, ENO and DeLENO give similar results, as can be seen from the decompressed images in Figure 8 and the relative errors in Table 3. In this table we additionally listed the

compression rate

$$(7.11) \quad c_r = 1 - \frac{\#\{d_{i,j}^k | d_{i,j}^k > \epsilon^k, 1 \leq k \leq K\}}{(N_L^x + 1)(N_L^y + 1) - (N_0^x + 1)(N_0^y + 1)},$$

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.



FIGURE 8. Image compression.

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

$$(7.12) \quad q(x, y) = \begin{cases} -10 & \text{if } (x - 0.5)^2 + (y - 0.5)^2 < 0.0225 \\ 30 & \text{if } |x - 0.5| > 0.8 \text{ or } |y - 0.5| > 0.8, \\ 40 & \text{otherwise} \end{cases}$$

where $(x, y) \in [0, 1] \times [0, 1]$, and generate a sequence of meshes by setting $K = 4$, $N_0^x = 16$ and $N_0^y = 16$. The threshold for data compression is chosen according to (B.6), with $\epsilon = 10$ and $t = 0.5$. The non-zero \hat{d}^k coefficients are plotted in Figure 9, while the errors and compression rate (7.11) 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 (7.12).

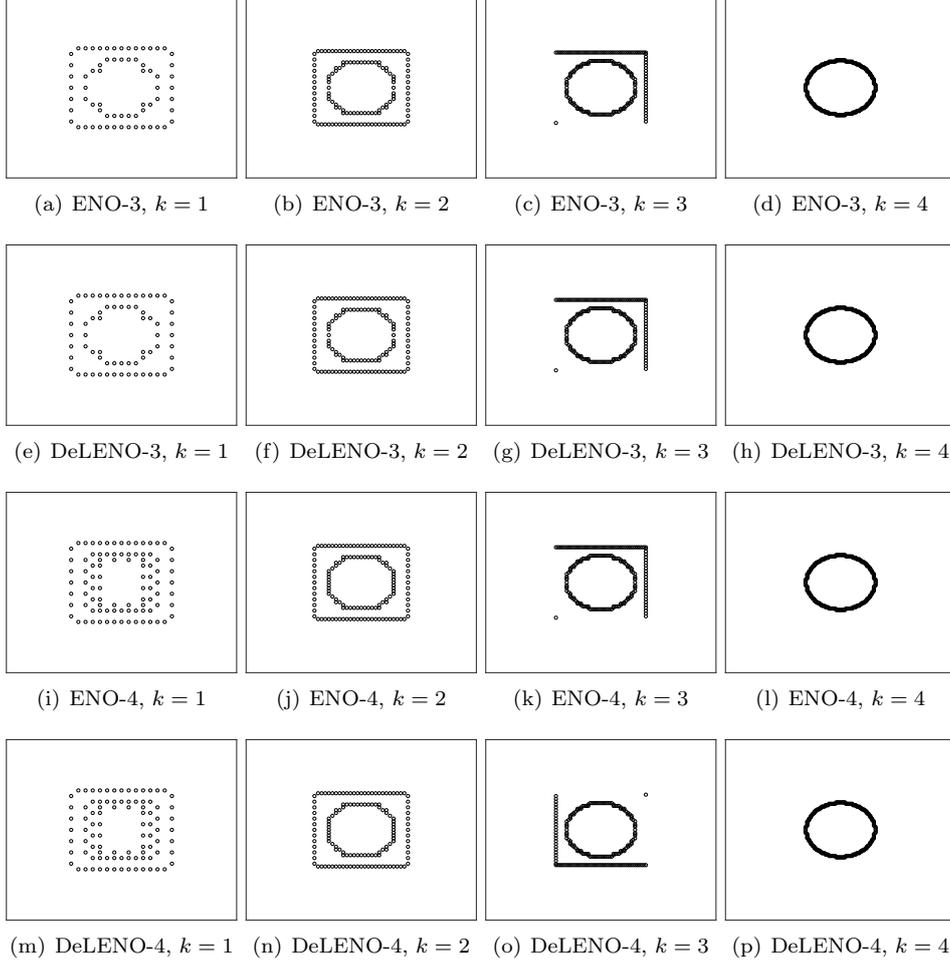


FIGURE 9. Non-zero coefficients \hat{d}^k for data compression of (7.12) using ENO and DeLENO for mesh level $1 \leq k \leq 4$.

7.2.3. 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

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

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 [SO89], which describes the interaction of a right moving shock with a 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 upto $T_f = 1.8$ 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 10, ENO- p and DeLENO- p perform equally well depending on the order p .

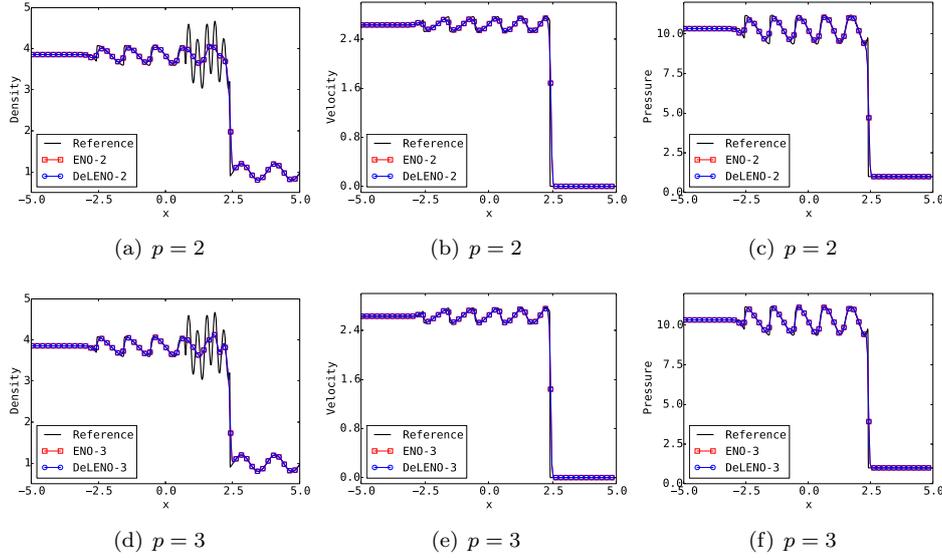


FIGURE 10. Solution for Euler shock-entropy problem with ENO- p and DeLENO- p on a mesh with $N = 200$ cells.

Next we solve the Sod shock tube problem [Sod78], 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 discretized 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 11.

8. 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 ENO (and ENO-SR) interpolation procedure (due to its data dependent nature), are able to interpolate rough functions efficiently. 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 surprising fact provides a different perspective on the ability of neural networks in approximating functions and reveals their enormous expressive power as even a highly non-linear, data dependent procedure

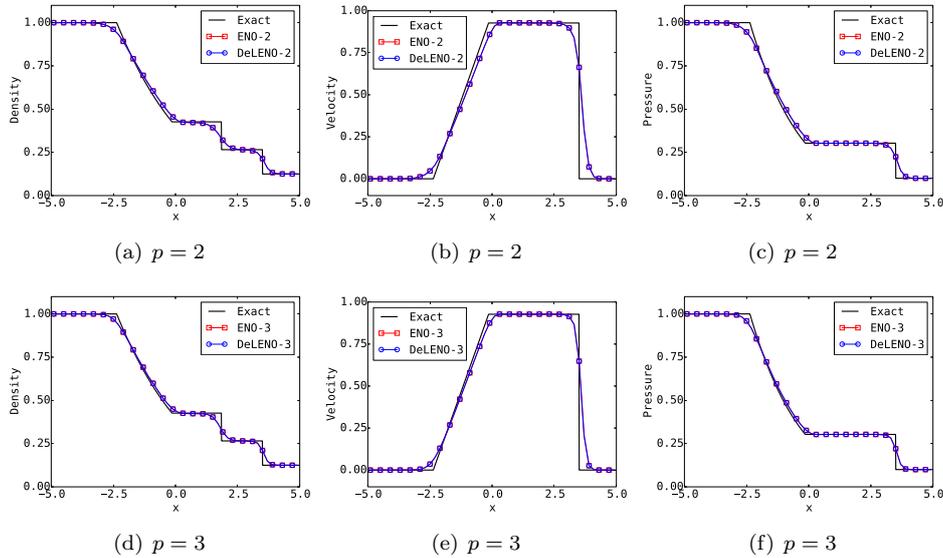


FIGURE 11. Solution for Euler Sod shock tube problem with ENO- p and DeLENO- p on a mesh with $N = 50$ cells.

such as ENO is nothing more than a ReLU neural network. On the other hand, the impressive function approximation results, for instance of [Yar17, YZ], might have limited utility in practice as the neural network might need to be trained for every function that has to be interpolated. By interpreting ENO 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 be 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

- [ACDD05] Francesc Aràndiga, Albert Cohen, Rosa Donat, and Nira Dyn. Interpolation and approximation of piecewise smooth functions. *SIAM Journal on Numerical Analysis*, 43(1):4157, 2005.
- [AD00] Francesc Aràndiga and Rosa Donat. Nonlinear multiscale decompositions: The approach of A. Harten. *Numerical Algorithms*, 23(2):175–216, Jun 2000.
- [AK06] G. Aubert and P. Kornprobst. *Mathematical problems in image processing: partial differential equations and calculus of variations*. Springer, New York, 2006.
- [Bar93] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.
- [CS89] Bernardo Cockburn and Chi-Wang Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Math. Comp.*, 52(186):411–435, 1989.
- [Cyb88] George Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1988.
- [Daf10] Constantine 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.
- [ES06] G. L. Eyink and K. Sreenivasan. Onsager and the theory of hydrodynamic turbulence. *Rev. Modern Phys.*, 78(1):87–135, 2006.

[Eva98] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[FMT13] U. S. Fjordholm, S. Mishra, and E. Tadmor. ENO reconstruction and ENO interpolation are stable. *Found. Comp. Math*, 13(2):139–159, 2013.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

[Har89] Ami Harten. ENO schemes with subcell resolution. *Journal of Computational Physics*, 83(1):148184, 1989.

[HEOC87] Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 71(2):231303, 1987.

[HEOC97] Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 131(1):3 – 47, 1997.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

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

[SO89] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, ii. *Journal of Computational Physics*, 83(1):32 – 78, 1989.

[SO91] C. W. Shu and S. Osher. High-order essentially nonoscillatory schemes for hamilton-jacobi equations. *SIAM J. Num. Anal.*, 28(4):107–122, 1991.

[Sod78] Gary 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.

[Yar17] Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103 – 114, 2017.

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

APPENDIX A. ENO COEFFICIENTS

Table 5 and Table 6 respectively list the ENO coefficients used for ENO interpolation (Section 3) and ENO reconstruction (Section 4). More information on the calculation of these coefficients can be found in [CS89].

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).

APPENDIX B. 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 [AD00]. We define a sequence of nested uniform meshes $\{\mathcal{T}^k\}_{k=0}^K$ on $\Omega = [c, d]$, where

(B.1)

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

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 (4.2).

for $0 \leq k \leq K$ 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}_Ω 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

$$(B.2) \quad q_i^{k-1} = (D_k^{k-1} q^k)_i = q_{2i}^k, \quad 0 \leq i \leq N_{k-1}.$$

In other words, the decimator helps in extracting the function values on a given mesh from a finer one.

- The *predictor* $P_{k-1}^k : \mathcal{V}^{N_{k-1}+1} \mapsto \mathcal{V}^{N_k+1}$ defined by $P_{k-1}^k := D_k R_{k-1}$. 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

$$e_i^k = q_i^k - (P_{k-1}^k q^{k-1})_i, \quad 0 \leq i \leq N_k.$$

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

$$(B.3) \quad d_i^k = e_{2i-1}^k = q_{2i-1}^k - (P_{k-1}^k q^{k-1})_{2i-1}, \quad 1 \leq i \leq N_{k-1}.$$

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

$$(B.4) \quad \{q^0, d^1, d^2, \dots, d^K\}.$$

This multiresolution representation (B.4) for a function $f \in \mathcal{B}_\Omega$ is convenient to perform data compression. The easiest compression strategy [AD00] corresponds to setting the coefficients d_i^k in (B.3) to zero based on a suitable threshold $\epsilon^k \geq 0$:

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

The compressed representation is then given by

$$(B.5) \quad \{f^0, \widehat{d}^1, \widehat{d}^2, \dots, \widehat{d}^K\}.$$

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

Algorithm 3: Compressed encoding [AD00]

Input: Highest resolution data f^K , number of levels K , number of points N_0 on coarsest mesh, ENO order p , threshold parameters ϵ and t .

Output: Multi-resolution representation $\{f^0, \widehat{d}^1, \dots, \widehat{d}^K\}$.

for $k = K$ **to** 1 **do**

$$\quad \lfloor f^{k-1} = D_k^{k-1} f^k$$

$$\widehat{f}^0 = f^0$$

for $k = 1$ **to** K **do**

$$\quad \widehat{f}_0^k = f_0^K$$

Construct P_{k-1}^k using Algorithm 1 and equation (3.2)

$$\quad \widehat{f}^k = P_{k-1}^k \widehat{f}^{k-1}$$

$$\quad N = N_0 2^{k-1}$$

for $i = 1$ **to** N **do**

$$\quad \left[\begin{array}{l} d_i^k = f_{2i-1}^k - \widehat{f}_{2i-1}^k \\ \epsilon^k = \epsilon t^{K-k} \\ \widehat{d}_i^k = \mathcal{G}(d_i^k; \epsilon^k) \\ \widehat{f}_{2i-1}^k = \widehat{f}_{2i-1}^k + \widehat{d}_i^k \\ \widehat{f}_{2i}^k = \widehat{f}_i^{k-1} \end{array} \right.$$

$$\quad \left[\begin{array}{l} \epsilon^k = \epsilon t^{K-k} \\ \widehat{d}_i^k = \mathcal{G}(d_i^k; \epsilon^k) \\ \widehat{f}_{2i-1}^k = \widehat{f}_{2i-1}^k + \widehat{d}_i^k \\ \widehat{f}_{2i}^k = \widehat{f}_i^{k-1} \end{array} \right.$$

$$\quad \left[\begin{array}{l} \widehat{f}_{2i-1}^k = \widehat{f}_{2i-1}^k + \widehat{d}_i^k \\ \widehat{f}_{2i}^k = \widehat{f}_i^{k-1} \end{array} \right.$$

$$\quad \left[\begin{array}{l} \widehat{f}_{2i}^k = \widehat{f}_i^{k-1} \end{array} \right.$$

return $\{f^0, \widehat{d}^1, \dots, \widehat{d}^K\}$

Algorithm 4: Decoding multi-resolution data [AD00]

Input: Multi-resolution representation $\{f^0, \widehat{d}^1, \dots, \widehat{d}^K\}$, number of levels K , number of cells N_0 on coarsest mesh, ENO order p .

Output: Decoded function \widehat{f}^K .

$$\widehat{f}^0 = f^0$$

for $k = 1$ **to** K **do**

Construct P_{k-1}^k using Algorithm 1 and equation (3.2)

$$\quad \left[\widehat{f}^k = P_{k-1}^k \widehat{f}^{k-1} + \widehat{d}^k \right.$$

return \widehat{f}^K

The following result is known on the error bounds for the compressed encoding in the form (B.5), the proof can be found in [AD00].

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

$$(B.6) \quad \epsilon^k = \epsilon t^{K-k}, \quad 0 < t < 1.$$

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

$$(B.7) \quad \|f^K - \widehat{f}^K\|_n \leq C_n \epsilon \quad \text{for } n = \infty, 1, 2,$$

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 C. PROOF OF THEOREM 6.1

We present some properties of our adaptation of the ENO-SR algorithm. To prove the second-order accuracy, we state some results due to [ACDD05] in a slightly adapted form.

Lemma C.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 [ACDD05] for $m = 3$. The result then follows from their Lemma 1. \square

Lemma C.2. *Let f 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 . Define the critical scale*

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

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 [ACDD05]. \square

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

- (1) *The singularity z is contained in an isolated B interval I_i^k or in a B-pair (I_i^k, I_{i+1}^k) .*
- (2) *The two polynomials p_{i-2}^k and p_{i+2}^k (or p_{i-1}^k and p_{i+3}^k) have only one intersection point y inside I_i^k or $I_i^k \cup I_{i+1}^k$, respectively.*
- (3) *The distance between z and y is bounded by*

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

Proof. This is a light adaptation of Lemma 3 in [ACDD05]. 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 (C.2) of this paper and equation (37) in [ACDD05] do not necessarily agree. \square

We now prove Theorem 6.1 of Section 6, based on the proof of Theorem 1 in [ACDD05]. Let h_c be as in Lemma C.2 and let K be as in Lemma C.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

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

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 C.2 then guarantees

that I_0 does not contain z . It then follows directly from the theory of Lagrange interpolation that

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

for all $x \in I_0$. Now suppose that I_0 was labelled bad. As a consequence of Lemma C.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 (C.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 C.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 $z \leq y$. In this case, equation (C.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

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

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

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

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

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

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

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

$$(C.4) \quad [f'] = 4h_c \sup_{\mathbb{R} \setminus \{z\}} |f''| \leq C_0 h \sup_{\mathbb{R} \setminus \{z\}} |f''|,$$

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 (C.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 (C.3) holds. We then calculate for $x \in I_0$,

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

where we used inequality (C.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 (C.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 $z \leq y$. Let $x \in I_0$ be arbitrary. It follows that equation

(C.3) also holds immediately for this x if $y \leq x$ or $x \leq z$. It suffices to find a bound for when $x_{-3} \leq z \leq x \leq y$. 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

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

where we used again inequality (C.4) and the bounds $|x - x_{-3}| \leq 3h$ and $x_{-3} \leq z$. This concludes the proof of Theorem 6.1.