Karlsruhe Institute of Technology

wave phenomena
analysis and numerics

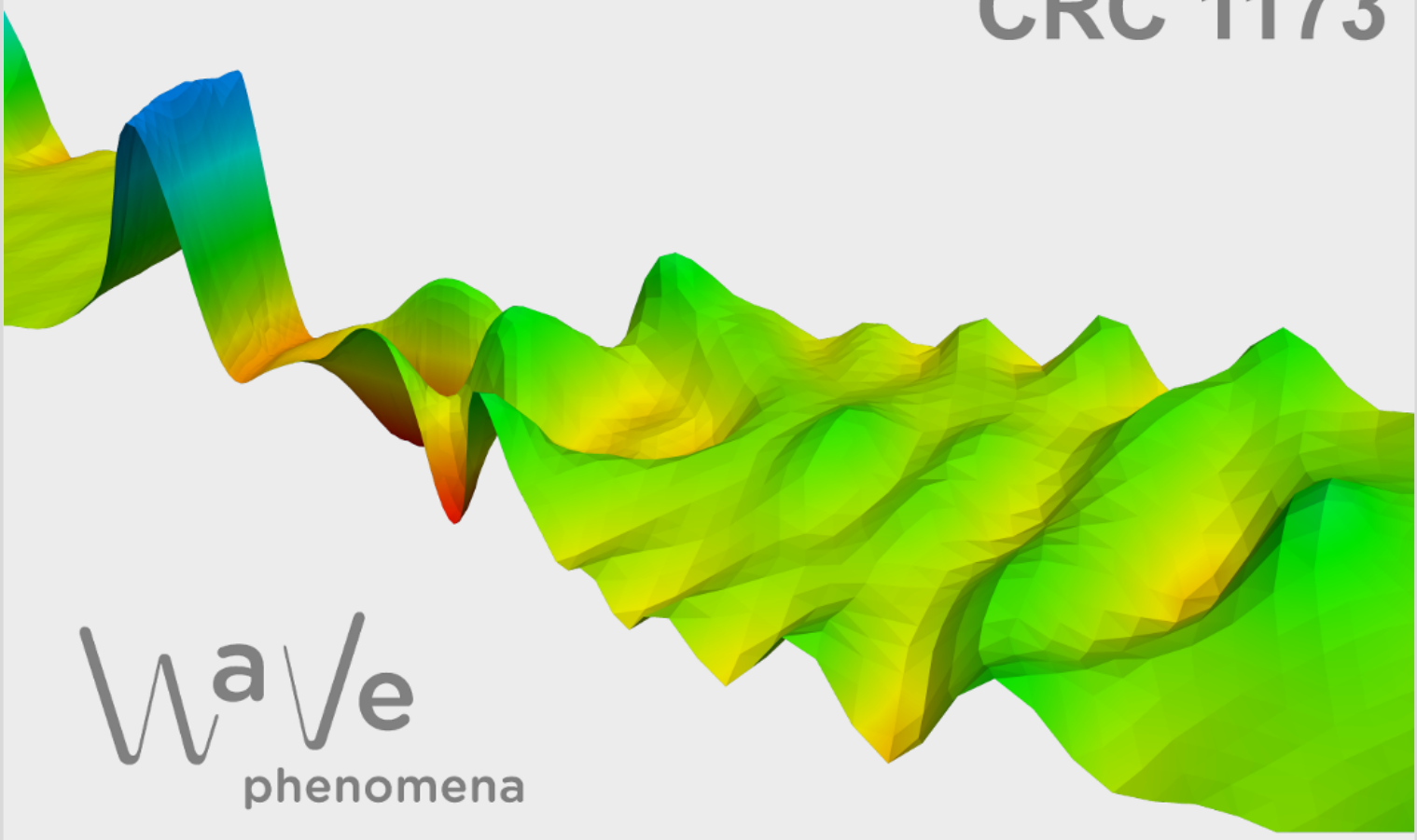
# Recurrent neural networks as optimal mesh refinement strategies

Michael Feischl, Jan Bohn

CRC Preprint 2020/33, November 2020

KARLSRUHE INSTITUTE OF TECHNOLOGY

CRC 1173



**Participating universities**

Universität Stuttgart

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN


**Funded by**


ISSN 2365-662X

# RECURRENT NEURAL NETWORKS AS OPTIMAL MESH REFINEMENT STRATEGIES*

MICHAEL FEISCHL† AND JAN BOHN‡

**Abstract.** We show that an optimal finite element mesh refinement algorithm for a prototypical elliptic PDE can be learned by a recurrent neural network with a fixed number of trainable parameters independent of the desired accuracy and the input size, i.e., number of elements of the mesh. Moreover, for a general class of PDEs with solutions which are well-approximated by deep neural networks, we show that an optimal mesh refinement strategy can be learned by recurrent neural networks. This includes problems for which no optimal adaptive strategy is known yet.

**1. Introduction.** Adaptive methods for finite element mesh refinement had tremendous impact on the scientific community both on the theoretical side as well as on the applied, engineering side.

Following the seminal works [7, 40, 13] on the adaptive finite element method, a multitude of papers extended the ideas to numerous model problems and applications, see e.g., [32, 14] for conforming methods, [36, 4, 5, 11, 33] for nonconforming methods, [15, 12, 31] for mixed formulations, and [23, 24, 2, 20, 21] for boundary element methods (the list is not exhausted, see also [10] and the references therein). Quite recently, [19, 22] also cracked non-symmetric and indefinite problems. All those works have in common that they use a standard adaptive refinement algorithm of the form

$$\boxed{\text{Solve}} \longrightarrow \boxed{\text{Estimate}} \longrightarrow \boxed{\text{Mark}} \longrightarrow \boxed{\text{Refine}}$$

where an error estimator is computed from the current solution and then used to refine certain elements of the mesh. The actual refinement of the individual elements of the mesh is usually done with an algorithm called newest-vertex bisection (see, e.g., [41]). A general drawback of adaptive mesh refinement methods is often their very specific area of application and their implementational overhead involved in the error estimation and choosing elements which to refine.

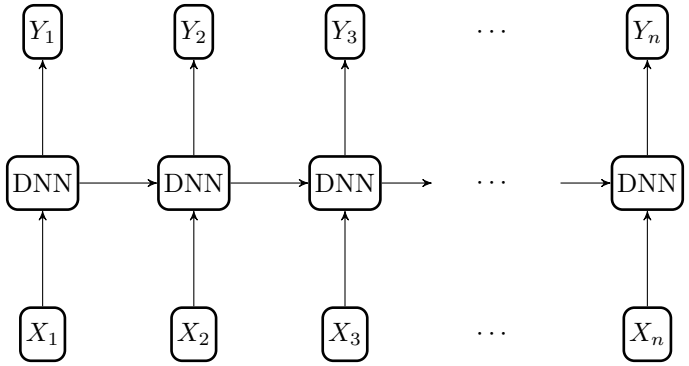
This encourages the development of black-box tools which can be adapted to a wide range of problems. In view of the huge practical success of recurrent neural networks (RNNs) in various applications and their flexibility in terms of the length of the input sequence (after all we do not want to retrain the network meshes of different sizes), they might provide exactly the required black-box tool. The most prominent examples of RNNs are Long-Term-Short-Term memory approaches proposed in [30] and since then hugely successful in practical applications, e.g., for time-series interpretation [38], speech recognition [26], speech synthesis [1], and even surgical robot control [34]. Very roughly, a recurrent neural network has the following structure

*Submitted to the editors DATE.

**Funding:** Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 258734477 – SFB 1173

†Institute of Analysis and Scientific Computing TU Wien, Wiedner Hauptstrae 8-10, 1040 Vienna (michael.feischl@tuwien.ac.at).

‡Institute for Applied and Numerical Mathematics, KIT, Englerstr. 2, 76131 Karlsruhe(jan.bohn@kit.edu).




where the $X_1, X_2, \dots, X_n$ denote a (vector valued) input sequence and $Y_1, Y_2, \dots, Y_n$ a (vector valued) output sequence. The block DNN denotes a standard deep neural network which maps the input state to the output state, but may also use hidden intermediate states from the previous iteration of the network. The major advantage of this structure compared to a fully connected DNN over all $n$ input states is that the weights of the DNNs are shared for all iterations. This means that an arbitrary long input sequence can be treated with a DNN depending only on a bounded number of trainable parameters. We will use this fact in order to construct a network whose parameter count does not depend on the number of elements of the current adaptive mesh.

The idea and question motivating this work is the following: Can we replace the steps $\boxed{\text{Estimate}} \longrightarrow \boxed{\text{Mark}}$ by a recurrent neural network $\boxed{\text{ADAPTIVE}}$ in order to achieve similar (or better) results than state of the art adaptive mesh refinement algorithms?

We answer this question in two ways: The first main result in Section 2.7 shows that an RNN ADAPTIVE can be trained to achieve at least the performance of adaptive algorithms which are known to be optimal for second order elliptic PDEs. The second main result in Section 2.8 shows for a broad class of problems that as long as the exact solution of a PDE can theoretically be efficiently approximated by a RNN, the RNN ADAPTIVE can be trained to produce optimally refined meshes. Roughly speaking, the present work shows that black-box mesh refinement by use of RNNs is at least as good as current optimal mesh refinement technology and can even achieve optimal results in areas which are not yet covered by the theory of adaptive mesh refinement.

The remainder of the work is structured as follows: Section 2 introduces the model problem, provides definitions of RNNs and optimal adaptive algorithms, and states the main results. Section 3 discusses the applicability of the main results as well as the implementation of the training process. Section 4 provides all the sub assemblies for the RNN which emulates the adaptive algorithm. Sections 4.4 and 4.5 contain the proofs of the main results. A final Section 5 underlines the theoretical findings by some numerical experiments.

**2. Model Problem & Main Results.** On the open Lipschitz domain $D \subset \mathbb{R}^d$, $d = 2, 3$, we consider a prototypical PDE of the form

$$\begin{aligned} \mathcal{L}u &= f \quad \text{in } D, \\ u &= 0 \quad \text{on } \partial D, \end{aligned} \tag{2.1}$$

where $\mathcal{L}\colon \mathcal{X} \to \mathcal{X}^\star$ is an isomorphism for some Hilbert space $\mathcal{X}$. With discrete spaces $\mathcal{X}(\mathcal{T}) \subset \mathcal{X}$ based on some triangulation $\mathcal{T}$ of $D$, this allows us to write down the discrete form of the equation: Find $U_\mathcal{T} \in \mathcal{X}(\mathcal{T})$ such that

$$\langle \mathcal{L}U_\mathcal{T}\,,\, V\rangle = f(V) \quad \text{for all } V \in \mathcal{X}(\mathcal{T}). \tag{2.2}$$

We assume that also $\mathcal{L}|_{\mathcal{X}(\mathcal{T})}$ is an isomorphism to obtain a unique discrete solution.

**2.1. Optimal mesh refinement.** We consider an initial regular and shape regular triangulation $\mathcal{T}_0$ of $D$ into compact simplices $T \in \mathcal{T}_0$. Such that $\mathcal{T}$ partitions $D$ into compact simplices such that the intersection of two elements $T \neq T' \in \mathcal{T}$ is either: a common face, a common node, or empty. In the recent literature [10, 40, 13], mesh refinement algorithms are steered by an error estimator $\rho(\mathcal{T}) = \rho(\mathcal{T}, U_\mathcal{T}, f) = \sqrt{\sum_{T\in\mathcal{T}} \rho_T^2}$ which satisfies $\rho(\mathcal{T}, U_\mathcal{T}, f) \approx \|u - U_\mathcal{T}\|_\mathcal{X}$ and have the following basic structure:

---

**Input:** Initial mesh $\mathcal{T}_0$, parameter $0 < \theta < 1$, tolerance $\varepsilon > 0$.
For $\ell = 0, 1, 2, \dots$ do:

1. Compute $U_\ell := U_{\mathcal{T}_\ell}$ from (2.2).
2. Compute error estimate $\rho_T$ for all $T \in \mathcal{T}_\ell$. If $\sum_{T\in\mathcal{T}_\ell} \rho_T^2 \le \varepsilon^2$, stop.
3. Find a set $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ of minimal cardinality such that
$$\sum_{T\in\mathcal{M}_\ell} \rho_T^2 \ge \theta \sum_{T\in\mathcal{T}_\ell} \rho_T^2. \tag{2.3}$$
4. Use newest-vertex-bisection to refine at least the elements in $\mathcal{M}_\ell$ and to obtain a new mesh $\mathcal{T}_{\ell+1}$.

**Output:** Sequence of adaptively refined meshes $\mathcal{T}_\ell$ and corresponding approximations $U_\ell \in \mathcal{X}(\mathcal{T}_\ell)$ such that $\rho(\mathcal{T}_\varepsilon) \le \varepsilon$ for final step $\mathcal{T}_\varepsilon := \mathcal{T}_L$ and $L \in \mathbb{N}$.

---

We consider the following notion of optimality of the mesh refinement algorithm: Let $\mathbb{T}$ denote the set of all possible meshes which can be generated by iterated application of newest-vertex-bisection to the initial mesh $\mathcal{T}_0$. Then, the maximal possible convergence rate $s > 0$ is defined by the maximal $s > 0$ such that

$$\sup_{N\in\mathbb{N}} \inf_{\substack{\mathcal{T}\in\mathbb{T}\\ \#\mathcal{T}-\#\mathcal{T}_0\le N}} \rho(\mathcal{T}, U_\mathcal{T}, f) N^s < \infty. \tag{2.4a}$$

We call Algorithm 2.1 optimal if it satisfies

$$\sup_{0<\varepsilon\le 1} \#\mathcal{T}_\varepsilon \rho(\mathcal{T}_\varepsilon)^{1/s} < \infty \tag{2.4b}$$

for the same rate $s$.

The main goal of this work is to prove that a particular type of neural network can be trained to perform the steps (2) and (3) of Algorithm 2.1 in an optimal way without any further knowledge about $\mathcal{L}$. We show that this is possible for second order elliptic operators $\mathcal{L}$ in Section 2.7 and for a much broader class of problems in Section 2.8.

**2.2. Definition of Deep Neural Networks.** We consider standard ReLU networks $B$ which can be defined as follows: For a given input $x \in \mathbb{R}^{s_0}$ and weight matrices $W_j \in \mathbb{R}^{s_{j+1} \times s_j}$, $j = 0, \dots, d$, we define the output $y \in \mathbb{R}^{s_{d+1}}$ as

$$y := B(x) := W_d \phi(W_{d-1} \phi(W_{d-2}(\cdots \phi(W_0 x) \cdots))),$$

where the activation function is defined as $\phi(y) := \max(y, 0)$ and is applied entry wise to vector valued inputs. A DNN is said to have depth $d$ and width $\max_{j=0,\dots,d+1} s_j$. The number of weights is given by $\sum_{j=0}^{d} s_{j+1} s_j$. We do not specify biases explicitly as we can always assume an additional constant input state $x_0$. Clearly, compositions of $+$, $-$, min, max, and $|\cdot|$ can be constructed as DNNs. Moreover, given two DNNs $B_1, B_2$, their composition is also a DNN. This is implicitly used in the following. We define the complexity of the DNN by the number of weights.

**2.3. Definition of Basic Recurrent Neural Networks.** A RNN $B$ is a deep neural network $B$ with output size $s' \in \mathbb{N}$ and input size $s+s'$, $s \in \mathbb{N}$. For reasons that become clear below, we denote this as a basic RNN. The DNN $B$ is applied to each entry of a (vector-valued) sequence $\boldsymbol{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^{s \times n}$ and returns another (vector-valued) sequence $\boldsymbol{y} = (y_1, y_2, \dots, y_n) \in \mathbb{R}^{s' \times n}$, $s, s' \in \mathbb{N}$. Additionally, the previous output state $y_{i-1}$ is fed into $B$ as an input state, i.e.,

$$y_i := B(x_i, y_{i-1}) := W_d \phi(W_{d-1} \phi(W_{d-2}(\cdots \phi(W_0 \begin{pmatrix} x_i \\ y_{i-1} \end{pmatrix}) \cdots))), \quad i = 1, \dots, n.$$

The weight matrices $W_j$ and hence the complexity of $B$ is independent of $n \in \mathbb{N}$. The number of weights of a basic RNN is just the number of entries in the weight matrices of the underlying DNN, width and depth are defined analogously. Hence, the complexity of a RNN is defined as the complexity of the underlying DNN. We also use the expression size synonymous to complexity. For $i = 1$, we always assume that $y_0 = 0 \in \mathbb{R}^{s'}$. For example, a simple summation over the sequence $\boldsymbol{x} \in \mathbb{R}^{1 \times n}$ can be realized by

$$y_i = B(x_i, y_{i-1}) := x_i + y_{i-1} = \begin{pmatrix} 1 & -1 \end{pmatrix} \phi\Big( \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} x_i \\ y_{i-1} \end{pmatrix} \Big).$$

The last entry of $\boldsymbol{y} \in \mathbb{R}^{1 \times n}$ contains the sum $y_n = \sum_{i=1}^{n} x_i$.

**2.4. Fixed number of independent weights.** As done in the previous subsection, by applying the same DNN $B$ to different parts of an input vector $x \in \mathbb{R}^{n_0}$, we may construct DNNs with arbitrary width but a fixed number of independent weights, i.e., the weights of $B$. By stacking the networks on top of each other, i.e., $B \circ B \circ \ldots \circ B(x)$ in case of $s_{d+1} = s_0$, we may create DNNs with arbitrary depth but still a fixed number of independent weights.
The distinction between number of independent weights and number of total weights is made since in the constructions below, the size of some networks grows logarithmically in the accuracy, however, they are just iterations of the same basic building block and hence the number of independent weights stays constant. This might benefit the training process, as the search space remains of constant size. On the other hand, the topology of the search space changes as the number of total weights grows. Therefore, further research is required on whether bounded number of independent weights can be used to speed up the training.

**2.5. Definition of Deep Recurrent Neural Networks.** We adopt a more general definition of RNNs in this work. We allow ourselves to deal with finite concatenations of those basic building blocks from the previous section, i.e., in our notion a RNN is a finite stack of $m$ basic RNNs $B_i$ in the sense

$$B_m \circ B_{m-1} \circ \ldots \circ B_2 \circ B_1(\boldsymbol{x}),$$

i.e., the output sequence of $B_1$ is fed into $B_2$ and so on. Additionally, we allow that the input $\boldsymbol{x}$ is initialized by the last entry of the output sequence of the previous network. So, in its most general form, the combination $\boldsymbol{y}' = B_2 \circ B_1(\boldsymbol{x})$ of two basic RNNs $B_1, B_2$ can be written as $\boldsymbol{x} \in \mathbb{R}^{s_1 \times n} \mapsto \boldsymbol{y} = B_1(\boldsymbol{x}) \in \mathbb{R}^{s_1' \times n}$ and

$$\boldsymbol{x}' = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ y_n & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{2s_1' \times n} \mapsto \boldsymbol{y}' := B_2(\boldsymbol{x}') \in \mathbb{R}^{s_2' \times n}.$$

We may write vector valued sequences $\boldsymbol{x} \in \mathbb{R}^{s \times n}$ as vectors of sequences $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_s)$. This choice of neural network class might seem arbitrary, however, it gives us much more freedom when constructing the networks and does not sacrifice the simplicity of the function class. This means that the complexity (defined as the sum over the complexities of the underlying basic RNNs) of a stacked RNN is still independent of the sequence length $n$. Similar constructions of deep RNNs (stacked RNNs) are considered in [25, 17, 37].

**2.6. Elementary operations with deep RNNs.** We illustrate some constructions which will be used implicitly in the proofs below:

• Identity-DNN: The identity function $\mathrm{id}(x) = \max(x, 0) - \max(-x, 0)$ can be emulated by a DNN of depth $d \geq 1$ with the following weight matrices (1 stands for the identity matrix of the correct size): $W_0 := \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $W_i := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ for $i = 1, \ldots, d-1$, and $W_d := \begin{pmatrix} 1 & -1 \end{pmatrix}$. Similarly, we can define Identity-RNNs.

• Matrix multiplication: The multiplication with a matrix $y = Mx$ for $M \in \mathbb{R}^{s' \times s}$ can be constructed as $W_0 := \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $W_1 := \begin{pmatrix} M & -M \end{pmatrix}$, and $W_2 := \begin{pmatrix} 1 & -1 \end{pmatrix}$.

• Applying DNN/RNNs simultaneously: If we want to compute the output of two DNNs $z_1 = A(x_1)$, $z_2 = B(x_2)$ at once, we can define the DNN $(z_1, z_2) = C(x_1, x_2) := (A(x_1), B(x_2))$ by

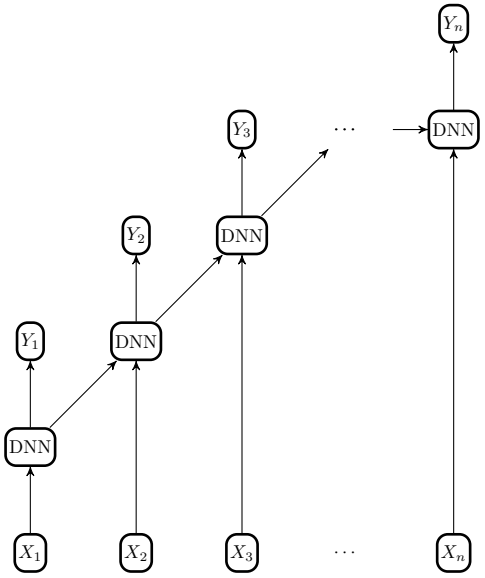
$$W_{C,i} := \begin{pmatrix} W_{A,i} & 0 \\ 0 & W_{B,i} \end{pmatrix},$$

where 0 denotes the zero matrix of appropriate size. In case the DNNs $A$ and $B$ have different depths, we use identity DNNs to extend $A$ and $B$ to equal depth (note that the resulting depth satisfies $d_c \leq \max(d_A, d_B) + 2$ since an identity DNN has at least two layers). Similarly we can apply RNNs simultaneously as long as the input lengths coincide. If we apply $m \in \mathbb{N}$ networks $B_1, \ldots, B_m$ simultaneously, the resulting network $C$ satisfies $d_C \leq \max_{i=1,\ldots,m} d_{B_i} + 3$ and the width of $C$ is bounded by the sum of the widths of the $B_i$.

• Any given deep RNN $\boldsymbol{y} = B(\boldsymbol{x})$ can be extended such that it copies an additional input variable to the output, i.e., there exists $\widehat{B}$ with comparable complexity to $B$ such that $(\boldsymbol{y}, \boldsymbol{x}_2) = \widehat{B}(\boldsymbol{x}_1, \boldsymbol{x}_2)$. In case $B$ is a basic RNN, this can be achieved by, e.g., defining $\widehat{B}(x_{1,i}, y_{i-1}, x_{2,i}) := (B(x_{1,i}, y_{i-1}), \mathrm{id}(x_{2,i}))$. If $B$ is a deep RNN, the same construction can be applied to all the basic RNNs that compose $B$.

• A basic RNN $B$ which turns a given sequence $\boldsymbol{x} = (x, 0, \dots, 0) \in \mathbb{R}^n$ into the constant sequence $\boldsymbol{y} = (x, x, \dots, x) \in \mathbb{R}^n$ can be defined by $y_i = B(x_i, y_{i-1}) = x_i + y_{i-1}$. Similarly one can add storage sequence for specific values inside an RNN.
• Composition of RNNs and DNNs:
  - (basic RNN)∘(DNN): The composition of a basic RNN with a DNN is again a RNN, by directly composing the underlying DNN of the RNN and the DNN.
  - (DNN)∘(basic-RNN): A DNN $y = B_2(x)$ and a basic RNN $y_i = B_1(x_i, y_{i-1})$ can be composed by $(z_i, y_i) = (B_2 \circ B_1(x_i, y_{i-1}), B_1(x_i, y_{i-1}))$ (note that the second entry in the output sequence is necessary for the correct evaluation of $B_1$).

• RNNs as DNNs: A RNN $B$ can be interpreted as a DNN $B'$. This means that we fix the input size $n$ of $B$ and consider the resulting neural network $B'$ which has $n$-times the width and depth of $B$ with a total number of weights of $n^3$ times the number of weights of $B$, as can be seen from:




However, the number of independent weights is determined only by the number of weights in $B$ and hence independent of $n$.
• RNNs with input $\boldsymbol{x} = (x, 0, \dots, 0) \in \mathbb{R}^n$ and output $\boldsymbol{y} = (y_1, \dots, y_n) \in \mathbb{R}^n$ that are interpreted as DNNs can be written as DNNs with one dimensional input $x \in \mathbb{R}$ and output $y_n \in \mathbb{R}$, by multiplication with the matrices $(1, 0, \dots, 0)$ and $(0, \dots, 0, 1)^T$.

**2.7. Main Result 1.** On the open Lipschitz domain $D \subset \mathbb{R}^d$, $d = 2, 3$, we consider a prototypical operator $\mathcal{L}$ of the form

$$\mathcal{L}u = -\operatorname{div}(A\nabla u) + b \cdot \nabla u + cu \tag{2.5}$$

where $\mathcal{L}$ has coefficients $A, b, c \in L^\infty(D)$ such that the associated bilinear form

$$a(u, v) := \langle \mathcal{L}u \,,\, v \rangle \quad \text{for all } u, v \in H^1_0(D)$$

satisfies $a(u, v) \leq C\|u\|_{H^1(D)}\|v\|_{H^1(D)}$ as well as $a(u, u) \geq C^{-1}\|u\|^2_{H^1(D)}$ for some constant $C > 0$. The Lax-Milgram lemma guarantees a unique solution $u \in H^1(D)$

of (2.1) and (2.2). On a triangulation $\mathcal{T}$, we define the Ansatz and test spaces

$$\mathcal{P}^p(\mathcal{T}) := \big\{v \in L^2(D) \,:\, v|_T \text{ is a polynomial of degree } \le p,\, T \in \mathcal{T}\big\}$$
$$\mathcal{X}(\mathcal{T}) := \mathcal{S}^p(\mathcal{T}) := \mathcal{P}^p(\mathcal{T}) \cap H^1_0(\mathcal{T})$$

for a polynomial degree $p \in \mathbb{N}_0$. We set $r(p,d) := \dim(\mathcal{P}^p)$, the dimension of the space of polynomials of degree $p$ in $d$ dimensions. The residual based error estimator for the given problem reads

(2.6a)
$$\rho_T^2 := \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2 := \operatorname{diam}(T)^2 \|f - \mathcal{L}U_\mathcal{T}\|^2_{L^2(T)} + \operatorname{diam}(T)\|[n \cdot A\nabla U_\mathcal{T}]\|^2_{L^2(\partial T \cap D)}$$

on each element $T \in \mathcal{T}$ with normal vector $n$ on the boundary $\partial T$ and $[\cdot]$ denoting the jump over element faces, and the overall estimator is the sum of the element wise contributions, i.e.,

$$\rho(\mathcal{T}) := \rho(\mathcal{T}, U_\mathcal{T}, f) := \sqrt{\sum_{T\in\mathcal{T}} \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2}. \tag{2.6b}$$

To avoid having to deal with data oscillations, we restrict ourselves to the simple case of $A|_T, b|_T, c|_T, f|_T \in \mathcal{P}^p(T)$ for all $T \in \mathcal{T}_0$. Obviously, the error estimator $\rho_T$ depends on the values of $U_\mathcal{T}$ on the whole patch $\omega_T := \big\{T' \in \mathcal{T} \,:\, T' \text{ shares a face with } T\big\}$. The main goal of the first part of this work is to show that RNNs of almost constant size are capable of performing optimal mesh refinement for the PDE given in (2.1). To that end, we construct a RNN which performs steps (2)–(3) of Algorithm 2.1.

Assumption 2.1. *We assume all numbers $x, y \in \mathbb{R}$ occurring in computations of the following algorithms satisfy the following: If $x \neq y$, there holds $|x - y| \ge 2^{-n_{\min}} \max\{|x|, |y|\}$ for some universal exponent $n_{\min} \in \mathbb{N}$. This assumption allows us to emulate step functions with neural networks which are continuous by construction. The assumption is satisfied in floating point number systems such as* `double`*-arithmetic, where $n_{\min}$ corresponds to the accuracy in terms of the number of digits, for* `double`*-arithmetic, it is $n_{\min} = 52$.*

Theorem 2.2. *For given $\varepsilon > 0$, there exists a deep RNN* ADAPTIVE *which takes a vector-valued input sequence $\boldsymbol{x} \in \mathbb{R}^{(2(d+1)d+(d+3)r(p,d))\times \#\mathcal{T}}$ such that $x_i$ contains the nodes of the elements $T' \in \omega_{T_i}$ for $T_i \in \mathcal{T}$ and the corresponding polynomial expansions of $U_{T'}$ and $f|_{T_i}$. The output $\boldsymbol{y} := \text{ADAPTIVE}(\boldsymbol{x}) \in \mathbb{R}^{\#\mathcal{T}}$ satisfies*

$$\sum_{\substack{T_i\in\mathcal{T}\\ y_i>0}} \widetilde{\rho}^2_{T_i} \ge \theta \sum_{T\in\mathcal{T}} \widetilde{\rho}^2_T \tag{2.7}$$

*for estimators $\widetilde{\rho}_T$ which satisfy*

$$|\rho_T^2 - \widetilde{\rho}_T^2| \le C_{\text{ada}} \frac{\varepsilon}{\#\mathcal{T}} \quad \text{for all } T \in \mathcal{T}.$$

*with a uniform constant $C_{\text{ada}} > 0$. Moreover, the number of positive entries in $\boldsymbol{y}$ is minimal in order to satisfy (2.7). The RNN has a fixed number of independent weights. The RNN can be constructed with a total number of weights of $O((n_{\min} + \log(\#\mathcal{T}) + |\log(\varepsilon)| + |\log(\|\boldsymbol{x}\|_\infty)|)^4)$ (see Figure 4.3 for the precise structure). The magnitude of the weights is $\mathcal{O}(1)$. Additionally, for a given overall tolerance $\varepsilon^2_{\text{tol}} > 0$, it holds $\boldsymbol{y} \le 0$, as soon as the tolerance $\tilde{\rho}(\mathcal{T}, U, f)^2 := \sum_{T_i\in\mathcal{T}} \widetilde{\rho}^2_{T_i} \le \varepsilon^2_{\text{tol}}$ is reached.*

**Input:** Initial mesh $\mathcal{T}_0$, tolerance $\varepsilon_{\mathrm{tol}} > 0$.
For $\ell = 0, 1, 2, \dots$ do:

1. Compute $U_\ell$ from (2.2).
2. Apply $\boldsymbol{y} = \mathrm{ADAPTIVE}(\boldsymbol{x})$ as defined in Theorem 2.2.
3. Use newest-vertex-bisection with mesh closure to refine the elements $T_i \in \mathcal{T}_\ell$ with $y_i > 0$ to obtain a new mesh $\mathcal{T}_{\ell+1}$ or stop if $\boldsymbol{y} \leq 0$.

**Output:** Sequence of adaptively refined meshes $\mathcal{T}_\ell$ and corresponding approximations $U_\ell \in \mathcal{S}^p(\mathcal{T}_\ell)$ such that $\tilde\rho(\mathcal{T}_{\varepsilon_{\mathrm{tol}}}) \leq \varepsilon_{\mathrm{tol}}$ for final step $\mathcal{T}_{\varepsilon_{\mathrm{tol}}} := \mathcal{T}_L$.

We refer to Section 4.4 for the proof of the Theorem. This result suggests the following algorithm:

From the previous theorem, we derive the following consequence.

COROLLARY 2.3. *Given $\varepsilon > 0$ in Theorem 4.14, Algorithm 2.7 is optimal in the sense*

$$\sup_{\sqrt{4C_{\mathrm{ada}}\varepsilon/\theta} \leq \varepsilon_{\mathrm{tol}} \leq 1} \#\mathcal{T}_{\varepsilon_{\mathrm{tol}}} {\varepsilon_{\mathrm{tol}}}^{1/s} \leq C < \infty$$

*with the maximal rate $s > 0$ from (2.4) and $C > 0$ independent of $\varepsilon$ and $\varepsilon_{\mathrm{tol}}$.*

*Proof.* We may assume $C_{\mathrm{ada}}\varepsilon \leq \theta\varepsilon_{\mathrm{tol}}^2/4$. Moreover, for $\widetilde\rho_\ell < \varepsilon_{\mathrm{tol}}$, we may redefine $\widetilde\rho_\ell := \rho_\ell$. There holds for $\widetilde\rho_\ell \geq \varepsilon_{\mathrm{tol}}$ that

$$|\rho_\ell^2 - \widetilde\rho_\ell^2| \leq C_{\mathrm{ada}}\varepsilon \leq \theta\varepsilon_{\mathrm{tol}}^2/4 \leq \begin{cases} \theta\widetilde\rho_\ell^2/4, \\ 2\theta\widetilde\rho_\ell^2/4 - \theta C_{\mathrm{ada}}\varepsilon \leq \theta\rho_\ell^2/2. \end{cases}$$

This implies $\widetilde\rho_\ell^2 \geq (1-\theta/2)\rho_\ell^2$ as well as $\rho_\ell^2 \geq (1-\theta/4)\widetilde\rho_\ell^2$. Assume that $\widetilde\rho_\ell$ satisfies $\sum_{T\in\mathcal{M}} \widetilde\rho_T^2 \geq \theta\widetilde\rho_\ell^2$. Then, the above shows immediately

$$\sum_{T\in\mathcal{M}} \rho_T^2 \geq \sum_{T\in\mathcal{M}} \widetilde\rho_T^2 - \theta\widetilde\rho_\ell^2/4 \geq 3\theta/4\widetilde\rho_\ell^2 \geq 3\theta/8\rho_\ell^2.$$

On the other hand, if $\rho_\ell$ satisfies $\sum_{T\in\mathcal{M}} \rho_T^2 \geq \theta\rho_\ell^2$, there holds

$$\sum_{T\in\mathcal{M}} \widetilde\rho_T^2 \geq \sum_{T\in\mathcal{M}} \rho_T^2 - \theta/2\rho_\ell^2 \geq \theta/2\rho_\ell^2 \geq 3\theta/8\widetilde\rho_\ell^2.$$

This equivalence of marking for the two error estimators $\rho$ and $\widetilde\rho$ together with the global equivalence $\widetilde\rho_\ell \simeq \rho_\ell$ allows us to apply [10, Theorem 8.4] directly to prove optimality of $\widetilde\rho$. This concludes the proof. ∎

**2.8. Main Result 2.** While the results of Section 2.7 are restricted to second order elliptic PDEs, the following statements deal with a much broader class of problems by making some assumptions on the exact solution.

Let $S := \bigcup_{k=0}^{d-1} S_k \subset D$ denote a singularity set such that $S_k$ is a finite union of compact $k$-dimensional facets (points for $k = 0$, edges for $k = 1$, etc). Define the weight

$$w(x) := \min_{k=0,\dots,d-1} \mathrm{dist}(x, S_k)^{\max\{0, d-k-\delta_{\mathrm{reg}}\}}$$

for some $\delta_{\mathrm{reg}} > 0$. This induces the weighted space $L^\infty_w(D)$ with the norm

$$\|v\|_{L^\infty_w(D)} := \|wv\|_{L^\infty(D)}.$$

Let $u\colon D \to \mathbb{R}$ denote the exact solution of some problem $\mathcal{L}u = f$ for some operator $\mathcal{L}\colon H^1_0(D) \to H^{-1}(D)$. The following result does not depend on the numerical method used to compute $U_\ell$ and hence we just assume that $\nabla\mathcal{X}(\mathcal{T}) \supseteq \mathcal{P}^0(\mathcal{T})$ for all $\mathcal{T} \in \mathbb{T}$ and that we compute some function $U_\mathcal{T} \in \mathcal{X}(\mathcal{T})$ by means of some numerical method, i.e., FEM, DG-FEM, .... Consider the following slight modification of Algorithm 2.7:

---

**Input:** Initial mesh $\mathcal{T}_0$, tolerance $\varepsilon_{\mathrm{tol}} > 0$.
For $\ell = 0, 1, 2, \dots$ do:

1. Compute discrete approximation $U_\ell$.
2. Apply $\boldsymbol{y} = \mathrm{ADAPTIVE}(\boldsymbol{x})$ as defined in Theorem 2.5.
3. Use newest-vertex-bisection to refine the elements $T_i \in \mathcal{T}_\ell \setminus \mathcal{T}_{\ell-1}$ (or $T_i \in \mathcal{T}_0$ for $\ell = 0$) with $y_i > 0$ to obtain a new mesh $\mathcal{T}_{\ell+1}$ or stop if $\boldsymbol{y} \le 0$.

**Output:** Sequence of adaptively refined meshes $\mathcal{T}_\ell$ and corresponding approximations $U_\ell \in \mathcal{S}^1(\mathcal{T}_\ell)$ with $\mathcal{T}_{\varepsilon_{\mathrm{tol}}} := \mathcal{T}_L$ for final step $L \in \mathbb{N}$.

---

For the following result, we require a slightly different definition of the maximal rate: Let $s > 0$ be maximal such that

$$\sup_{N\in\mathbb{N}} \inf_{\substack{\mathcal{T}\in\mathbb{T}\\ \#\mathcal{T}-\#\mathcal{T}_0\le N}} \max_{T\in\mathcal{T}} \inf_{v\in\mathcal{P}^0(\mathcal{T})} \|\nabla u - v\|_{L^2(T)} N^{s+1/2} < \infty. \tag{2.8}$$

We call Algorithm 2.8 optimal if it satisfies

$$\sup_{0<\varepsilon\le 1} \#\mathcal{T}_\varepsilon \Big( \max_{T\in\mathcal{T}_\varepsilon} \inf_{v\in\mathcal{P}^0(\mathcal{T}_\varepsilon)} \|\nabla u - v\|_{L^2(T)} \Big)^{1/(s+1/2)} < \infty. \tag{2.9}$$

Remark 2.4. *In general, the maximal rate in (2.8) is lower than in (2.4). However, in many practical situations, the two notions will coincide, as they are equivalent as long as there exists a quasi-best approximating triangulation $\mathcal{T}$ with $\#\mathcal{T} - \#\mathcal{T}_0 \le N$ and*

$$\max_{T\in\mathcal{T}} \inf_{v\in\mathcal{P}^0(\mathcal{T})} \|\nabla u - v\|_{L^2(T)} \lesssim \min_{T\in\mathcal{T}} \inf_{v\in\mathcal{P}^0(\mathcal{T})} \|\nabla u - v\|_{L^2(T)} + N^{-s-1/2}.$$

*This, however, is the case in many approximation results particularly those which include weighted spaces or Besov spaces (see, e.g., [8]).*

We denote by $\mathcal{T}(\varepsilon) \in \mathbb{T}$ the admissible mesh with minimal cardinality such that $\max_{T\in\mathcal{T}} \inf_{v\in\mathcal{P}^0(\mathcal{T})} \|\nabla u - v\|_{L^2(T)} \le \varepsilon$.

Theorem 2.5. *Let $u \in H^1(D)$ and $m \in \mathbb{N}$. Suppose there exists a deep RNN $v_\varepsilon$ which satisfies $\|\nabla u - v_\varepsilon\|_{L^2(D)} \le \varepsilon/(Cm)$ as well as $v_\varepsilon^2 \in L^\infty_w(D)$. Then, there exists a deep RNN* `ADAPTIVE` *such that Algorithm 2.8 produces outputs $\mathcal{T}_\varepsilon$ which satisfy*

$$\#\mathcal{T}_\varepsilon \Big( \max_{T\in\mathcal{T}_\varepsilon} \inf_{v\in\mathcal{P}^0(\mathcal{T}_\varepsilon)} \|\nabla u - v\|_{L^2(T)} \Big)^{1/(s+1/2)} \le C m^{1/(s+1/2)}$$

*with probability larger than $1 - L\#\mathcal{T}(4\varepsilon)2^{-Cm}$, where $C > 0$ depends on $D$, $\mathcal{T}_0$ and $\|v_\varepsilon\|_{L^\infty_w(D)}$ with $L$ denoting the maximal level of elements in $\mathcal{T}(4\varepsilon)$, i.e., the maximal*

*number of bisections necessary to generate each element from $\mathcal{T}_0$. The complexity of* ADAPTIVE *is bounded by $\mathcal{O}(m^2(\#v_\varepsilon + |\log(\varepsilon)| + |\log(\|v_\varepsilon\|_{L^\infty(D)})|))$, where $\#v_\varepsilon$ denotes the complexity of $v_\varepsilon$ and the number of independent weights is constant.*

REMARK 2.6. *The dependence of the complexity of* ADAPTIVE *on $\log \|v_\varepsilon\|_{L^\infty(D)}$ is usually not a problem. Any deep RNN $v_\varepsilon$ approximating $\nabla u$ up to accuracy $\varepsilon > 0$ can be capped at magnitude $C > 0$ by composition, i.e., $\widetilde{v}_\varepsilon := \max(\min(v_\varepsilon, C), -C)$. The approximation error satisfies*

$$\|\widetilde{v}_\varepsilon - \nabla u\|_{L^2(D)} \le \varepsilon + \|\nabla u\|_{L^2(D_C)},$$

*with $D_C := \{x \in D : |\nabla u| > C\}$. If $\nabla u \in L^p(D)$ for some $p > 2$, we already have $|D_C| \le \|\nabla u\|^p_{L^p(D)} C^{-p}$ and hence $\|\nabla u\|_{L^2(D_C)} \le \|\nabla u\|_{L^p(D_C)} |D_C|^{(p-2)/(2p)} \lesssim C^{1-p/2}$. This shows that $\log(\|v_\varepsilon\|_{L^\infty(D)}) \lesssim |\log(\varepsilon)|$ is possible.*

REMARK 2.7. *We note that the result in Theorem 2.5 is not completely satisfactory as the complexity of the deep RNN* ADAPTIVE *depends on the complexity of the exact solution (or rather the neural network approximating it) and not, as one would expect from a classical error estimator, on the complexity of the problem and the data. However, the numerical experiments in Section 5 show that this seems not to be an issue in practical applications and might be an artifact of the proof.*

We postpone the proof of the above theorem to Section 4.5.

## 3. Discussion of the main results.

**3.1. Theoretical results.** Theorem 2.2, Corollary 2.3, and Theorem 2.5 show that an RNN can in fact achieve optimal mesh refinement in the sense of (2.4) and (2.8). The RNN only needs to follow the fairly general structure of a deep RNN. The width and depth of the RNNs depends poly-logarithmically on the number of elements $\#\mathcal{T}$ as well as on the desired accuracy. The number of independent weights (trainable parameters) is, however, uniformly bounded and independent of the accuracy as well as of the number of elements in the mesh.

While Corollary 2.3 shows that the deep RNN approach is at least as good as current mesh refinement strategies for second order elliptic problems (2.1) which are known to be optimal, Theorem 2.5 proves that an optimal mesh refinement strategy can be learned by a deep RNN whenever the exact solution can be approximated efficiently by a deep RNN. The latter result is independent of the problem type and thus applies to problem classes for which we currently do not know optimal refinement strategies.

Such problems include non-linear PDEs (for example (2.5) with coefficients depending on $u$). For time dependent PDEs, the current setting based on the $H^1$-norm is too restrictive. However, the proofs can be transferred to any $L^2$-based norm particularly the anisotropic Bochner norms used in parabolic applications. Moreover, the method of proof for Theorem 2.5 does not depend on the numerical method used to compute the approximations $U_\ell$. Thus the result also covers non-FEM methods such as discontinuous Galerkin methods, isogeometric analysis methods, boundary element methods and more.

The only requirement is that the exact solution lies in the weighted space $L^\infty_w(D)$ and can be approximated efficiently by a deep RNN (or just a DNN). To that end, we refer to the large number of approximation results for PDEs via neural networks [27, 28, 35, 3, 29] and the references therein.

If the data-to-solution map $f \mapsto u$ can be approximated by a deep RNN, then Theorem 2.5 even provides the existence of a deep RNN ADAPTIVE which is optimal in the sense (2.8) and can be used for any right-hand side data without retraining.

But even if ADAPTIVE has to be retrained for each new instance of data, the numerical experiments in Section 5.3 show advantageous performance compared to uniform mesh refinement.

**3.2. Practical implementation.** As stated in [30], RNNs can be hard to train by gradient descent approaches since the recursive nature either dampens any gradient information or leads to blow-up. The RNNs appearing in this work are very sparsely recursive (almost all recursive connections are disabled). The existing recursive connections on the input sequence $\boldsymbol{x}$ (and also all intermediate sequences) are always multiplications by 1 or -1 as well as additions. Hence those connections do not lead to blowup or dampening. The constructions include some RNNs with multiplication by 2 or 4 in the recursive connections, but those RNNs are always transformed into DNNs and their size depends only logarithmically on the given accuracy. Including this observation into the training might improve the performance.

The training of the deep RNNs can be implemented practically in different ways. For symmetric problems, one may optimize the weights to maximize the energy of the discrete Galerkin approximation (which is equivalent to minimizing the error). This is done in the numerical experiments of Section 5.3. For more general problems, a substitute energy error is given by

$$E_\ell := \sqrt{\sum_{k=\ell}^{\infty} \|U_{k+1} - U_k\|^2}$$

It is shown in [10, 19, 18] that under quite general assumptions, there holds $E_\ell \simeq \|u - U_\ell\|$ up to higher order terms. Thus, to maximize the convergence rate $E_\ell \to 0$, it suffices to maximize $\|U_\ell - U_{\ell-1}\|$ in each adaptive step. Hence, this computable term may serve as a goal quantity for the optimization algorithm.

**4. Construction of the Neural Networks.** This section is dedicated to the construction of the basic building blocks of the RNN.

**4.1. Basic logic & algebra.** For the implementation of the RNNs below, we require a rudimentary emulation of the IF-clause.

REMARK 4.1. *We note that Assumption 2.1 is used particularly in the constructions in this particular section to guarantee that the RNN* IF *constructed below produces the correct output. The RNN* IF *is the sole part of the following constructions, where a round-off error is intentionally scaled to order* $\mathcal{O}(1)$*. Thus we provide a thorough round-off error analysis in the following Lemma 4.2. In the remaining constructions,* IF *is just used as a building block and we check that input and output of* IF *behave as expected. Thus we follow the usual convention in numerical analysis and do not treat the round-off error explicitly in the calculations outside of* IF*.*

LEMMA 4.2. *For* $\square \in \{\leq, \geq, <, >\}$ *there exists a fixed size basic RNN* IF *such that any input* $\boldsymbol{x} = ((a, b, c), 0, \dots, 0) \in \mathbb{R}^{3\times n}$ *with* $a, b, c \in \mathbb{R}$ *satisfying* $|b - c| \geq 2^{-\tilde{n}}|a|$ *results in an output* $\boldsymbol{y} := \mathrm{IF}(\boldsymbol{x}) := \mathrm{IF}(a; b \square c) \in \mathbb{R}^n$ *with*

$$y_n = \begin{cases} a & b \square c, \\ 0 & \text{else}, \end{cases}$$

*for $n \geq \tilde{n}$. If we interpret IF as a DNN, the number of weights behaves like $O(\tilde{n}^3)$, but the number of independent weights is $O(1)$. With regard to Assumption 2.1, $\tilde{n} := n_{\min}$ is a valid choice as long as $\max(|c|, |b|) \geq |a|$.*

*Proof.* We first define a basic RNN $\widehat{\mathrm{IF}}$ for which, with input $\boldsymbol{x} \in \mathbb{R}^{2\times n}$, $\boldsymbol{x} = ((a,b), 0 \dots, 0)$ with $a \geq 0$, the output $\boldsymbol{y} = \widehat{\mathrm{IF}}(\boldsymbol{x})$ satisfies

$$y_n = \begin{cases} a & b \geq a2^{-n}, \\ 0 & b \leq 0. \end{cases}$$

The RNN can be defined by

$$y_i = (y_{i,1}, y_{i,2}) := \widehat{\mathrm{IF}}(x_i, y_{i-1}) := \big(x_{i,1} + y_{i-1,1}, \min(2\max(y_{i-1,2} + x_{i,2}, 0), y_{i,1})\big).$$

(Note that the first component of $y_i$ has the sole purpose of storing the value of $a$ for later use.) Since $x_i = 0$ for all $i \geq 2$ and $y_0 = 0$, we have $y_i = (a, \min(2^i \max(b, 0), a))$. This concludes the construction of $\widehat{\mathrm{IF}}$.
Now let $a, b, c \in \mathbb{R}$ and first assume $a \geq 0$. We can see, that $\mathrm{IF}(a; b > c) := \widehat{\mathrm{IF}}(a, b-c)$ produces the expected output, as long as $\tilde{n} \geq n$ and $|b-c| \geq 2^{-\tilde{n}}a$: If $b \leq c$, then this is clear, and if $b > c$, it already holds $b - c \geq 2^{-\tilde{n}}a$ and the first case of $\widehat{\mathrm{IF}}$ occurs.

We can define $\mathrm{IF}(a; b \leq c)$ by $\mathrm{IF}(a; b \leq c) := a - \mathrm{IF}(a; b > c)$, and $\mathrm{IF}(a; b \geq c)$, $\mathrm{IF}(a; b < c)$ can be defined by changing the roles of $b$ and $c$ resulting in the condition $|b - c| \geq 2^{-\tilde{n}}a$.
For $a \in \mathbb{R}$, we set $a_+ := \max(a, 0)$ and $a_- := \max(-a, 0)$ and $\mathrm{IF}(a; b \square c) := \mathrm{IF}(a_+; b \square c) - \mathrm{IF}(a_-; b \square c)$ produces the expected output as long as $|b - c| \geq 2^{-\tilde{n}} \max(a_+, a_-) = 2^{-\tilde{n}}|a|$. ∎

Remark 4.3. *Obviously, the RNN $\widehat{\mathrm{IF}}$ could be constructed as a one layer network $\min(a, 2^n \max(b, 0))$ at the expense of allowing large weights.*

To emulate the error estimator from Section 1, we require a number of basic algebraic operations. We start with squaring. The idea that DNNs can emulate the function $x \mapsto x^2$ up to arbitrary precision first appeared in [42]. They showed that a DNN of size proportional to $|\log(\varepsilon)|$ achieves this up to some tolerance $\varepsilon$. We improve on this idea by using an RNN of fixed size to perform the same operation. The application of the network is equally expensive as the DNN from [42], however, the number of weights which need to be trained is fixed and independent of $\varepsilon$.

Theorem 4.4. *For every $n \in \mathbb{N}$, there exists a deep RNN* SQUARE *with a fixed number of weights such that the output $\boldsymbol{y} = \mathrm{SQUARE}(\boldsymbol{x})$ for an input vector $\boldsymbol{x} = (x_0, 0, \dots, 0) \in [-1, 1]^n$ satisfies*

$$y_n = |x_0| - \sum_{j=1}^{n} \frac{g^{(j)}(|x_0|)}{4^j} \quad \text{and} \quad |y_n - x_0^2| \leq 4^{-2n}.$$

*for some universal constant $C > 0$. If we interpret the basic buildings blocks of the RNN as DNNs, the concatenation of them is still a DNN, so* SQUARE *interpreted as a DNN has a total number of weights of $O(n^3)$, but the number of independent weights stays fixed.*

*Proof.* We reuse the saw-tooth function from [42]

$$G(x) := \begin{cases} 2x & x \in [0, 1/2], \\ 2 - 2x & x \in (1/2, 1], \end{cases}$$

which can also be written as $g(x) = 2\max(x, 0) - 4\max(x - 1/2, 0) + 2\max(x - 1, 0)$. We define $g := G \circ G$. From the input sequence $\boldsymbol{x} \in \mathbb{R}^n$, a first basic RNN layer generates the sequence

$$\boldsymbol{x}' = (g(|x_0|), g^{(2)}(|x_0|), \dots, g^{(n)}(|x_0|)).$$

A second basic RNN $B$ performs the following summation

$$y_i := B(x_i, y_{i-1}) = x_i + 4y_{i-1}.$$

This results in

$$\boldsymbol{y} = (g(x_0), \dots, \sum_{j=1}^{n} 4^{n-j} g^{(j)}(x_0)).$$

Finally, the basic RNN $B'$ computes

$$z_i := B'(z_{i-1}) = z_{i-1}/4.$$

Initialized with the last entry $y_n$, this operation computes the vector

$$\boldsymbol{z} = (y_n, y_n/4, \dots, y_n 4^{-n}).$$

By definition, $y_n 4^{-n} = \sum_{j=1}^{n} 4^{-j} g^{(j)}(|x_0|))$. Thus, we constructed the desired approximation to $|x_0| - |x_0|^2$.

The error estimate follows from the fact that the approximation to $x^2$ is actually the linear spline interpolation $f_n$ of $f(x) = x^2$ at $4^n$ equidistant points in $[0, 1]$ (see [42]). This shows

$$|f_n(x_0) - f(x_0)| \leq \frac{4^{-2n}}{2} \|f''\|_{L^\infty}$$

and thus concludes the proof. ∎

The new idea of the following result is that the magnitude of the input is not limited by the number of parameters, but rather by the input length only. This shows that a fixed number of trainable parameters give a network which can multiply arbitrarily large numbers.

COROLLARY 4.5. *For every $n \in \mathbb{N}$, there exists a deep RNN* SQUARE *with a fixed number of weights such that the output $\boldsymbol{y} =$ SQUARE$(\boldsymbol{x})$ for an input vector $\boldsymbol{x} = (x_0, 0, \dots, 0) \in [-2^n, 2^n]^n$ satisfies*

$$y_n = |x_0| - \sum_{j=1}^{n} \frac{g^{(j)}(|x_0|)}{4^j} \quad \text{and} \quad |y_n - x_0^2| \leq 4^{-n}.$$

SQUARE *interpreted as a DNN has a total number of weights behaving like $O(n^3)$, but the number of independent weights stays bounded.*

*Proof.* A first basic RNN performs the scaling

$$y_i = B^1(x_i) := y_{i-1}/2.$$

Note that if $|x_0| \leq 2^n$ there holds $y_n \leq 1$.

We initialize the input of SQUARE with the last entry $y_n$ to compute $z \in \mathbb{R}$ with $|z - y_n^2| \leq 4^{-2n}$. Finally, we reverse the scaling by initializing a RNN $B^2$ with $(z, 0, \dots, 0)$ and compute

$$y_i = B^2(x_i) := 4y_{i-1}$$

Hence, the final output satisfies

$$|y_n - x_0^2| = 4^n |z - (x_0 2^{-n})^2| \leq 4^n 4^{-2n} \leq 4^{-n}.$$

This concludes the proof. ∎

With the squaring operation at hand, we immediately obtain a method for multiplying two numbers by using the formula $2xy = (x+y)^2 - x^2 - y^2$.

PROPOSITION 4.6. *There exists a deep RNN* MULTIPLY *such that for all $x, y \in [-2^{n-1}, 2^{n-1}]$ the output $\boldsymbol{z} =$ MULTIPLY$(\boldsymbol{x}, \boldsymbol{y})$ ($\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ denote the sequences $\boldsymbol{x} = (x, 0, \dots, 0)$, $\boldsymbol{y} = (y, 0, \dots)$) satisfies*

$$|z_n - xy| \leq C 4^{-n},$$

*where $C > 0$ is independent of $n$ and $x, y$.* MULTIPLY *interpreted as a DNN has a total number of weights behaving like $O(n^3)$, but the number of independent weights stays bounded.*

*Proof.* As mentioned above, we construct MULTIPY from SQUARE with inputs in $[-2^n, 2^n]$. The construction is

$$\text{MULTIPLY}(\boldsymbol{x}, \boldsymbol{y}) = (\text{SQUARE}(\boldsymbol{x} + \boldsymbol{y}) - \text{SQUARE}(\boldsymbol{x}) - \text{SQUARE}(\boldsymbol{y}))/2.$$

The error estimate follows immediately from Corollary 4.5. ∎

**4.2. Error estimation.** For brevity of presentation, we restrict ourselves to the case $A = 1$ and $b = c = 0$ of (2.1). The general case can easily be implemented along the lines of this section. In the present case, the residual error estimator given in (2.6) is usually computed via quadrature. This assumes that $f$ is a piecewise polynomial of low enough order such that the quadrature is exact. For convenience, we use an equivalent definition of $\rho_T$, i.e.,

$$\begin{aligned} \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2 \simeq \operatorname{diam}_\infty(T)^{2+d} |T|^{-1} \|f + \Delta U_\mathcal{T}\|_{L^2(T)}^2 \\ + \operatorname{diam}_\infty(T)^d |\partial T|^{-1} \|[\nabla U_\mathcal{T}]\|_{L^2(\partial T \cap D)}^2, \end{aligned} \tag{4.1}$$

with $\operatorname{diam}_\infty(T) := \max_{x,y \in T} |x - y|_\infty$. Obviously, $\operatorname{diam}_\infty(T) \simeq \operatorname{diam}(T)$ depending only on the space dimension. Moreover, since $\nabla U_\mathcal{T} = n \partial_n U_\mathcal{T} + \sum_{i=1}^{d-1} t_i \partial_{t_i} U_\mathcal{T}$ for normal vector $n$ and tangential vectors $t_1, \dots, t_{d-1}$ and $[\partial_{t_i} U_\mathcal{T}] = 0$ on any interface for $U_\mathcal{T} \in \mathcal{S}^p(\mathcal{T})$, there holds

$$|[\partial_n U_\mathcal{T}]|^2 = |n[\partial_n U_\mathcal{T}]|^2 = |n[\partial_n U_\mathcal{T}] + \sum_{i=1}^{d-1} t_i [\partial_{t_i} U_\mathcal{T}]|^2 = |[\nabla U_\mathcal{T}]|^2 \quad \text{on } \partial T.$$

Note that it would certainly be possible to emulate the exact error estimator $\rho(\cdot)$, however, as shown in [10], a uniform multiplicative factor does not make any difference in the convergence behavior and hence we opted for the version which results in slightly simpler constructions.

LEMMA 4.7. *There is a fixed size DNN* DIAM *which, given the nodes of an element* $T = \mathrm{conv}(z_0, \dots, z_d)$ *computes the* $\infty$*-diameter* $\mathrm{diam}_\infty(T) := \max_{x,y\in T} |x-y|_\infty$ *of* $T$.

*Proof.* We exemplify this for $d = 2$ and $T = \mathrm{conv}(z_1, z_2, z_3)$, i.e.,

$$\mathrm{diam}_\infty(T) = \max(\max(|z_1 - z_2|_\infty, |z_1 - z_3|_\infty), |z_2 - z_3|_\infty),$$

where $|(x, y) - (x', y')|_\infty = \max(|x - x'|, |y - y'|)$ and the absolute value function is realized via

$$|x| = \max(x, 0) + \max(-x, 0).$$

Obviously, this strategy generalizes to higher dimensions. ∎

LEMMA 4.8. *Let* $U, f \in \mathcal{P}^p(T)$ *for a given element* $T \in \mathcal{T}$. *There is a deep RNN* VOL *which, given the nodes of the element* $T = \mathrm{conv}(z_0, \dots, z_d)$ *as well as the polynomial coefficients of* $U|_T$ *and* $f|_T$ *as a vector valued sequence* $\boldsymbol{x} = (x, 0, \dots, 0) \in \mathbb{R}^{(d(d+1)+2r(p,d))\times n}$, *satisfies*

$$\left|\mathrm{diam}_\infty(T)^d |T|^{-1} \|f + \Delta U\|^2_{L^2(T)} - y_n\right| \le C 2^{-n}$$

*for* $\boldsymbol{y} = \mathrm{VOL}(\boldsymbol{x})$ *as long as the coefficients of the polynomial expansion of* $(f + \Delta U)$ *and the nodes are contained in* $[-2^{\alpha n}, 2^{\alpha n}]$, *where* $0 < \alpha < 1$ *depends only on* $p$ *and* $d$. VOL *interpreted as DNN has a number of weights of* $O(n^3)$, *but the number of independent weights is fixed.*

*Proof.* We have $d + 1$ points determining the shape of $T$, so $(d + 1)d$ scalar numbers, and two input functions in $\mathcal{P}^p(T)$ with dimension $r(p, d) = \sum_{i=0}^{p} \binom{d+i-1}{i}$, which makes in total $x \in \mathbb{R}^{d(d+1)+2r(p,d)}$.

Given the polynomial coefficients of $U$, we can compute the coefficients of $\Delta U$ by multiplication with a matrix only depending on $d$ and $p$, which we construct as a DNN. Then, we stack $r(d, p)^2$ RNNs MULTIPLY from Proposition 4.6 to compute the coefficients of $(f + \Delta U)^2$ up to accuracy $\lesssim 4^{-n}$. To compute the integral of the $L^2$-norm, we note that for the basis functions of the polynomial space, $\phi_k(x) = \Pi_{j=1}^d x_j^{\alpha_j^k}$ with exponents $\alpha_j^k$, we have

$$\mathrm{diam}_\infty(T)^d |T|^{-1} \int_T \phi_k(x)\, dx = \mathrm{diam}_\infty(T)^d \int_{\widehat{T}} \phi_k(F_T(x))\, dx,$$

for the reference element $\widehat{T}$ and $F_T(x) = (z_1 - z_0, \dots, z_d - z_0)x + z_0$. The integral over $\phi_k(F_T(x))$ can be expressed as a sum over integrals over basis functions on the reference element, and from the latter we assume to have them stored in our net as weights, which are scalar numbers only depending on $d$ and $p$. The corresponding coefficients are polynomials of the nodes $z_i - z_0$ and $z_0$, which can be computed by a number of multiplications only depending on $p$ and $d$ with accuracy $\lesssim 4^{-n}$. All in all, a number only depending on $d$ and $p$ of instances of MULTIPY compute the integral with accuracy $\lesssim 4^{-n}$. By Proposition 4.6, all multiplications are computed with the stated tolerance, as long as the coefficients of $f + \Delta U$ and the nodes inserted to an polynomial depending on $d$ and $p$ is contained in $[-2^{n-1}, 2^{n-1}]$. Remark that the above constants still may depend on the magnitude of the input vector. We exemplary consider the computation of a product $\Pi_{i=1}^k x_i$, (in this situation k depending only on

$p$ and $d$). It holds (with $\odot$ denoting the approximate multiplication via MULTIPLY)

$$\begin{aligned}
&\left|\Pi_{i=1}^k x_i - x_1 \odot (\cdots \odot x_k)\right| \\
&\leq x_1 \left|\Pi_{i=2}^k x_i - x_2 \odot (\cdots \odot x_k)\right| + |x_1(x_2 \odot (\cdots \odot x_k)) - x_1 \odot (\cdots \odot x_k)| \\
&\leq |x_1| \left|\Pi_{i=2}^k x_i - x_1(x_2 \odot (\cdots \odot x_k))\right| + C4^{-n} \\
&\leq \cdots \leq Ck\Pi_{i=1}^k (1+|x_i|)4^{-n} \leq C(k)2^{-n}.
\end{aligned}$$

Here we assumed $x_i \lesssim 2^{n/k}$ and the operations $\odot$ are computed with the stated accuracy $\lesssim 4^{-n}$, as the inserted values can be shown to be bounded by $2^{-n-1}$ by induction. This concludes the proof. ∎

LEMMA 4.9. *Let $U, f \in \mathcal{P}^p(T)$ for a given element $T \in \mathcal{T}$. There is a deep RNN* JUMP *which, given the nodes of the elements $T' = \operatorname{conv}(z_0, \dots, z_d)$ as well as the polynomial coefficients of $U|_{T'}$ for all elements $T' \in \omega_T$ as a vector valued sequence $\boldsymbol{x} = (x, 0, \dots, 0) \in \mathbb{R}^{(2(d+1)d+(d+2)r(p,d))\times n}$, satisfies*

$$\left|\operatorname{diam}_\infty(T)^{d-1}|\partial T|^{-1}\|[\nabla U]\|^2_{L^2(\partial T)} - y_n\right| \leq C2^{-n}$$

*for $\boldsymbol{y} =$ JUMP$(\boldsymbol{x})$ as long as the coefficients of the polynomial expansion of $[\nabla U]$ and as long as the coefficients of the polynomial expansion of $[\nabla U]$ are contained in $[-2^{\alpha n}, 2^{\alpha n}]$, where $0 < \alpha < 1$ depends only on $p$ and $d$.* JUMP *interpreted as DNN has a number of weights behaving like $O(n^3)$, but the number of independent weights is fixed.*

*Proof.* As input, we have $d+1$ nodes determining the shape of T, and another $d+1$ elements in the patch which are determined by another $d+1$ points. So $(2d+2)d$ scalar variables for the nodes and $(d+2)r(p,d)$ for the polynomial coefficients of $U$, which results in $x \in \mathbb{R}^{2(d+1)d+(d+2)r(p,d)}$. The proof works analogously to that of Lemma 4.8, with the difference that we have to include the data on the patch of $T$ to compute $[\nabla U]$. ∎

THEOREM 4.10. *There exists a basic RNN* ESTIMATOR *which takes a vector-valued input sequence $\boldsymbol{x} \in \mathbb{R}^{(2(d+1)d+(d+3)r(p,d))\times \#\mathcal{T}}$ such that $x_i$ contains: The nodes of the elements $T' \in \omega_{T_i}$ for $T_i \in \mathcal{T}$ and the corresponding polynomial expansions of $U_{T'}$ and $f|_{T_i}$. The output $\boldsymbol{y} :=$ ESTIMATOR$(\boldsymbol{x})$ satisfies*

$$|y_i - \rho_{T_i}(\mathcal{T}, U, f)^2| \leq C2^{-n}$$

*in case $n \gtrsim \max(\log(\boldsymbol{x}))$ for a uniform hidden constant. The RNN* ESTIMATOR *has a fixed number of independent weights but width and depth proportional to $n$, so a total number of weights behaving like $O(n^3)$.*

*Proof.* Lemmas 4.7–4.9 show that there are RNNs computing all ingredients for $\rho_T(\mathcal{T}, u, f)^2$. A fixed number of applications of the RNN MULTIPLY combine the elements and output an approximation to $\rho_T(\mathcal{T}, u, f)^2$ up to an accuracy $\lesssim 2^{-n}$ as long as the magnitude of the input is bounded by $2^{\alpha n}$ (where $n \in \mathbb{N}$ is the size of the input sequence and $\alpha$ only depends on $d$ and $p$). Interpreting the resulting RNN EST which computes the approximation to $\rho_T(\mathcal{T}, u, f)^2$ as a DNN, we observe that EST is a DNN with depth and width $\mathcal{O}(n)$ composed of $n$ copies of the same net. Hence, we only have a fixed (accuracy independent) number of independent weights in EST although the width and depth of EST depends on $n$. Moreover, EST forms the building block for an RNN which takes a vector-valued sequence $\boldsymbol{x} \in \mathbb{R}^{(2(d+1)d+(d+3)r(p,d))\times \#\mathcal{T}}$ as

described in the statement. From this, EST computes the output sequence $y_i$ which satisfies

$$|y_i - \rho_{T_i}(\mathcal{T}, U, f)^2| \lesssim 2^{-n}.$$

A final application $\boldsymbol{y} = \max(\boldsymbol{y}, 0)$ guarantees the non-negativity of the estimators and this concludes the proof. ∎

**4.3. Dörfler marking.** The marking algorithm is based on the following observation: Assume $x_1, \dots, x_n \geq 0$. Consider the binary search algorithm

---

**Input:** $x_1, \dots, x_n \geq 0$, $0 < \theta \leq 1$
Set $y_1 := \max_{1\leq i\leq n} x_i/2$. For $\ell = 1, \dots, k$ do:
1. If $\sum_{x_i \geq y_\ell} x_i \geq \theta \sum_{i=1}^n x_i$, set $y_{\ell+1} = y_\ell + y_1/2^\ell$.
2. If $\sum_{x_i \geq y_\ell} x_i < \theta \sum_{i=1}^n x_i$, set $y_{\ell+1} = y_\ell - y_1/2^\ell$.

---

LEMMA 4.11. *For $x_1, \dots, x_n \geq 0$ let $y \geq 0$ be maximal such that $\sum_{x_i \geq y} x_i \geq \theta \sum_{i=1}^n x_i$. Then, there holds $|y - y_\ell| \leq 2^{-\ell} \max_{1\leq i\leq n} x_i$.*

*Proof.* The binary search nature of the algorithm immediately guarantees $|y - y_\ell| \leq 2^{-\ell} \max_{1\leq i\leq n} x_i$. ∎

THEOREM 4.12. *There exists a deep RNN* BINARY *consisting of $k$ basic RNNs of the same type (up to an fixed size input layer), which takes as input the sequence $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^{1\times n}$ and the output $\boldsymbol{y} := \mathrm{BINARY}(\boldsymbol{x}) \in \mathbb{R}$ satisfies*

$$|y - y_k| \leq 2^{-k} \max_{1\leq i\leq n} x_i,$$

*where $y \geq 0$ is maximal such that $\sum_{x_i \geq y} x_i \geq \theta \sum_{i=1}^n x_i$. The number of weights of the basic RNNs is bounded by $O(n_{\min}^3)$, the number of independent weights is fixed and as the deep RNN consists of $k$ copies of the same basic RNN, the total number of weights is bounded by $O(kn_{\min}^3)$, while the number of independent weights stays bounded independently.*

*Proof.* We may construct a basic RNN SUMY with inputs $(x_1, \dots, x_n)$ and $y$ via

$$z_i = \mathrm{SUMY}(x_i, y, z_{i-1}) := (z_{i-1} + \mathrm{IF}(x_i; x_i \geq y))$$

Lemma 4.2 shows that IF computes the exact cut-off function, as we assume $x_i$ to satisfy Assumption 2.1. This shows

$$z_n = \sum_{x_i \geq y} x_i$$

and the number of weights of SUMY behaves like $O(n_{\min}^3)$. A basic RNN gives the initial values $y_1 := \max_{1\leq i\leq n} x_i/2$, $z_1 := y_1/2$ and we construct the basic RNN that performs one iteration of Algorithm 4.3. One iteration of Algorithm 4.3 corresponds to $\boldsymbol{z}_{\ell+1} = \boldsymbol{z}_\ell/2$ and

$$\begin{aligned}\boldsymbol{y}_{\ell+1} = \boldsymbol{y}_\ell &+ \mathrm{IF}(\boldsymbol{z}_\ell; \mathrm{SUMY}(\boldsymbol{x}, y_{\ell,n}) \geq \theta \mathrm{SUMY}(\boldsymbol{x}, 0)) \\ &- \mathrm{IF}(\boldsymbol{z}_\ell; \mathrm{SUMY}(\boldsymbol{x}, y_{\ell,n}) < \theta \mathrm{SUMY}(\boldsymbol{x}, 0)).\end{aligned}$$

∎

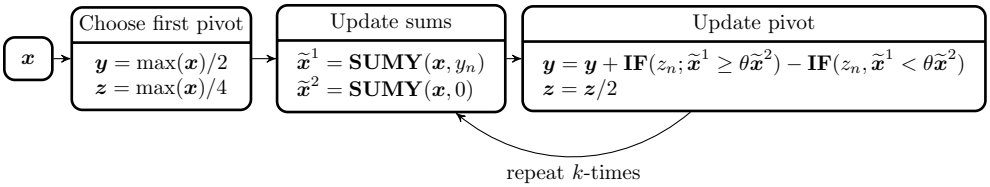



FIG. 4.1. *The structure of the RNN* BINARY *from Theorem 4.14. Variables which are not used in a particular block are copied to the output sequence. The RNN* $\boldsymbol{y} = \max(\boldsymbol{x})$ *is defined by* $y_i = \max(y_{i-1}, x_i)$ *and computes* $y_i = \max_{1\leq j\leq i} x_j$.

It is $\boldsymbol{z}_\ell = \boldsymbol{y}_1/2^\ell$ and $y_{\ell,n}$ (which is the last entry of $\boldsymbol{y}_\ell$) contains the current pivot. With Assumption 2.1, the cut-off functions in $\mathrm{SUMY}(\boldsymbol{x}, y_{\ell,n})$, $\theta\mathrm{SUMY}(\boldsymbol{x}, 0)$ are computed exactly since $\mathrm{SUMY}(\boldsymbol{x}, y_\ell) \geq \max_{1\leq i\leq n} x_i \geq \boldsymbol{z}_\ell$. Note that one mapping $(y_l, z_l) \mapsto (y_{l+1}, z_{l+1})$ corresponds to an application of a DNN of size $O(n_{\min}^3)$ to $\mathrm{SUMY}(\boldsymbol{x}, y_\ell)$ and $\mathrm{SUMY}(\boldsymbol{x}, 0)$. This can be constructed as follows (see also Figure 4.1). A basic RNN of comparable size to SUMY takes the input $(\boldsymbol{x}, y_{\ell,n})$ and computes the output $\widetilde{\boldsymbol{x}}$ with $\widetilde{x}_i := (\mathrm{SUMY}(\boldsymbol{x}, y_{\ell,n}), \mathrm{SUMY}(\boldsymbol{x}, 0))$ for all $1 \leq i \leq n$. Then a basic RNN containing the DNN IF is applied to $\widetilde{\boldsymbol{x}}$ to compute the output $\boldsymbol{y}_{\ell+1}$ and $\boldsymbol{z}_{\ell+1}$. This concludes the proof.

LEMMA 4.13. *There exists a basic RNN* ROUND *which takes as input a non-negative sequence* $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^{1\times n}$ *and values* $\overline{x} > \underline{x} \geq 0$ *and returns a sequence* $\boldsymbol{y} = (y_1, \dots, y_n) := \mathrm{ROUND}(\boldsymbol{x}) \in \mathbb{R}^{1\times n}$ *which satisfies*

$$y_i = \begin{cases} \overline{x}, & x_i \in [\underline{x}, \overline{x}] \\ x_i, & \text{else.} \end{cases}$$

*The number of weights behaves like* $O(n_{\min}^3)$*, while the number of independent weights stays bounded.*

*Proof.* The RNN can be constructed as the component-wise maximum of the sequences $\boldsymbol{x}$ and $\overline{\boldsymbol{x}}$, where

$$\overline{x}_i := \begin{cases} \overline{x}, & x_i \in [\underline{x}, \overline{x}] \\ 0, & \text{else,} \end{cases} \qquad = \overline{x} - \mathrm{IF}(\overline{x}; x_i > \overline{x}) - \mathrm{IF}(\overline{x}; x_i + \overline{x} < \underline{x} + \overline{x}).$$

The IF are interpreted as DNN's of size $O(n_{\min}^3)$, and compute the expected output, as we assume $x_i$, $\overline{x}$, $\underline{x}$ to satisfy Assumption 2.1 and it holds $\max(|x_i + \overline{x}|, |\underline{x} + \overline{x}| \geq |\underline{x} + \overline{x}|) \geq \overline{x}$. ∎

THEOREM 4.14. *There exists a deep RNN* MARK *with accepts a non-negative sequence* $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{R}^{1\times n}$*, and returns* $\boldsymbol{y} = (y_1, \dots, y_n) := \mathrm{MARK}(\boldsymbol{x})$ *with* $y_i \in \mathbb{R}$ *which satisfies*

$$\sum_{\substack{i=1 \\ y_i>0}}^{n} \tilde{x}_i \geq \theta \sum_{i=1}^{n} \tilde{x}_i$$

*such that the number of terms in the left-hand side sum is minimal and* $\tilde{\boldsymbol{x}}$ *satisfies* $|x_i - \tilde{x}_i| \leq \varepsilon/n$.*. The deep RNN consists of a fixed layer of basic RNNs, followed by*

$k \simeq \log_2(\max_{1\le i\le n} x_i)+|\log_2(\varepsilon/n)|+1$ *copies of the same basic RNN and ends with an output layer of a fixed number of basic RNNs. The overall number of weights therefore behaves like* $O(kn_{\min}^3)$*, while the number of independent weights stays bounded.*

*Proof.* See also Figure 4.2 for the following construction: As a first layer, we have the deep RNN BINARY with $k \simeq \log_2(\max_{1\le i\le n} x_i) + |\log_2(\varepsilon/n)| + 1$ repetitions. This produces the pivot $y_k$ from Algorithm 4.3 with $|y_k - y| \le 2z_k \le \varepsilon_{\text{tol}}/(2n)$ and with $y \ge 0$ maximal such that $\sum_{x_i\ge y} x_i \ge \theta \sum_{i=1}^n x_i$. Now, for $z_k$ from BINARY, it holds $y \in [y_k - 2z_k, y_k + 2z_k]$ and an application of ROUND gives the sequence $\tilde{\boldsymbol{x}}$ for $\underline{x} := y_k - 2z_k$, $\overline{x} := y_k + 2z_k$ with the stated error bound, as $2z_k \le \varepsilon/(2n)$. We now set $\overline{y} := \overline{x}$ and it holds that $\overline{y}$ is maximal such that $\sum_{\tilde{x}_i\ge\overline{y}} \tilde{x}_i \ge \theta \sum_{i=1}^n \tilde{x}_i$. This is because it holds

$$\sum_{\tilde{x}_i>\overline{y}} \tilde{x}_i = \sum_{\tilde{x}_i>\overline{y}} x_i \le \sum_{x_i>y} x_i < \theta \sum x_i \le \theta \sum \tilde{x}_i$$

and

$$\begin{aligned}\sum_{\tilde{x}_i\ge\overline{y}} \tilde{x}_i &= \sum_{\tilde{x}_i\ge\overline{y}} x_i + \sum_{\tilde{x}_i=\overline{y}} (\tilde{x}_i - x_i) \ge \sum_{x_i\ge y} x_i + \theta \sum_{\tilde{x}_i=\overline{y}} (\tilde{x}_i - x_i) \\ &\ge \theta \sum x_i + \theta \sum_{\tilde{x}_i=\overline{y}} (\tilde{x}_i - x_i) = \theta \sum \tilde{x}_i,\end{aligned}$$

where we used that $x_i \ge y$ implies $\tilde{x}_i \ge \overline{y}$. Hence we found the exact cutoff $\overline{y}$ for the sequence $\tilde{\boldsymbol{x}}$ and proceed with this new sequence. We generate a preliminary output sequence $\widetilde{y}_i := \max(\tilde{x}_i - \overline{y}, 0)$. The final output $y$ is positive whenever $\widetilde{y}_i > 0$ and additionally on a few entries with $\tilde{x}_i = \overline{y}$. To find those entries, compute the sequence

$$\widehat{x}_i = \overline{y} - \text{IF}(\overline{y}; \overline{y} > \tilde{x}_i) - \text{IF}(\overline{y}; \overline{y} < \tilde{x}_i)$$

such that $(\widehat{x}_1, \dots, \widehat{x}_n)$ is zero unless $\widehat{x}_i = \tilde{x}_i = \overline{y}$. The computations of IF are exact as before, and the complexity of this basic RNN is of $O(n_{\min}^3)$ with $n$ repetitions. Next, we compute

$$z_i := \text{IF}(\overline{y}; \sum_{j=1}^{i-1} \widehat{x}_j + \sum_{\tilde{x}_i>\overline{y}} \tilde{x}_i < \theta \sum_{i=1}^{n} \tilde{x}_i)$$

There holds $z_i = \overline{y}$ for all $1 \le i \le i_0$ and $z_i = 0$ else for minimal $i_0$ such that $\sum_{j=1}^{i_0} \widehat{x}_j + \sum_{\tilde{x}_i>\overline{y}} \tilde{x}_i \ge \theta \sum_{i=1}^n \tilde{x}_i$. Note that $\sum_{j=1}^{i-1} \widehat{x}_j$ can be computed beforehand by a basic RNN of size $\mathcal{O}(n_{\min}^3)$ as a straightforward modification of SUMY in Theorem 4.12. As usual, we use Assumption 2.1 to guarantee that the cut-off functions are computed exactly, as we assume the sums to satisfy Assumption 2.1. Finally, we generate the desired output with

$$y_i = \max(\widetilde{y}_i, \min(z_i, \widehat{x}_i)).$$

This concludes the proof. ∎

**4.4. Proof of Theorem 2.2.** The previous sections already give the necessary ingredients to build the RNN ADAPTIVE from Theorem 2.2. We use the RNN

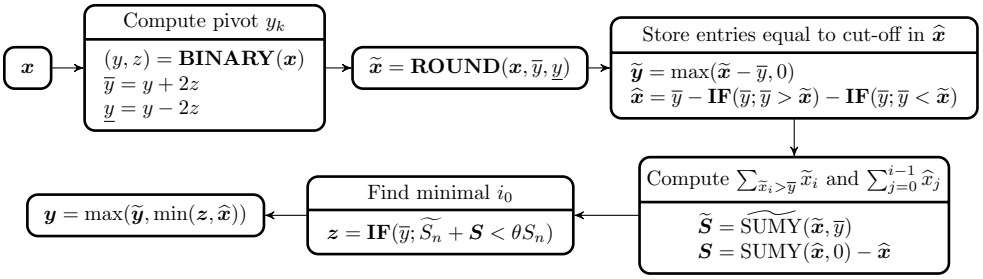



FIG. 4.2. *The structure of the RNN* MARK *from Theorem 4.14. Variables which are not used in a particular block are copied to the output sequence. Non-bold variables are implemented as constant sequences. The RNN* $\widetilde{\mathrm{SUMY}}$ *is a straightforward modification of* SUMY *from Theorem 4.12 by replacing* $\geq$ *with* $>$.

ESTIMATOR with accuracy $n \simeq |\log(\varepsilon/N)|$ from Theorem 4.10 to compute the error estimator $\widetilde{\rho}_T(\mathcal{T}, U_\mathcal{T}, f)$ such that

$$|\widetilde{\rho}_T(\mathcal{T}, U_\mathcal{T}, f)^2 - \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2| \lesssim \varepsilon/\#\mathcal{T} \tag{4.2}$$

for all $T \in \mathcal{T}$ as long as $N \geq \#\mathcal{T}$ and $|\log(\varepsilon/N)| \gtrsim \log(|x|_\infty)$. This results in a number of weights of $O(|\log(\varepsilon/N)|^3)$.
Theorem 4.14 provides the deep RNN MARK with $n = \#\mathcal{T}$, which performs the Dörfler marking and adds an additional error of $\varepsilon/\#\mathcal{T}$ to the error estimators. This results in a number of weights of

$$O\Big(\big(\log_2(\max_{1\leq i\leq \#\mathcal{T}} \tilde{\rho}_{T_i}) + |\log_2(\varepsilon/\#\mathcal{T})|\big) n_{\min}^3\Big).$$

All the basic building blocks used in the construction consist of a fixed number of independent weights but may have a total number of weights depending on $\log(N)$, $n_{\min}^3$ and $\log(\varepsilon)$. One of the basic blocks is stacked $(\log_2(\max_{1\leq i\leq \#\mathcal{T}} \tilde{\rho}_{T_i}) + |\log_2(\varepsilon/\#\mathcal{T})|)$-times. We ensure the stopping criterion of the algorithm by comparing the sum of the error estimators with the tolerance $\varepsilon_{\mathrm{tol}}$. The full structure of ADAPTIVE is given in Figure 4.3.

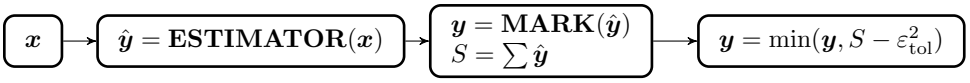



FIG. 4.3. *Structure of the RNN* ADAPTIVE. *For a given tolerance* $\varepsilon_{\mathrm{tol}}^2 > 0$*, the term* $S - \varepsilon_{\mathrm{tol}}^2$ *is non-positive whenever the prescribed tolerance has been reached by the approximate error estimator stored in* $S$ *and hence terminates the algorithm by setting* $\boldsymbol{y} \leq 0$.

**4.5. Proof of Theorem 2.5.** The proof of Theorem 2.5 requires some preparation. The first result states that for functions in certain weighted $L^\infty$ spaces, the Monte Carlo method overestimates the integral with a positive probability.

LEMMA 4.15. *For a given Lipschitz domain* $\Omega$ *(not necessarily* $D$*) and a singularity set* $S$*, let* $X\colon \Omega \to \mathbb{R}$ *be non-negative with* $X \in L^\infty_w$*. We assume* $|\Omega| = 1$*, and consider* $\Omega$ *as a probability space and* $X$ *as a random variable. There holds*

$$\mathbb{P}(X \geq q\mathbb{E}(X)) \geq C^{-1}$$

*for all* $0 < q < 1$ *and a constant* $C > 0$ *which depends on* $q$, $\Omega$, *the number of connected components in* $S_k$, $k = 0, \dots, d-1$, $\delta_{\mathrm{reg}}$, *and an upper bound for* $\|X\|_{L^\infty_w(\Omega)}$.

*Proof.* Let $p > 1$ such that $p/(p-1) < \min_{k=0,\dots,d-1} \frac{d-k}{d-k-\delta_{\mathrm{reg}}}$. Define $\Omega_{\geq} := \big\{\omega \in \Omega : X(\omega) \geq q\mathbb{E}(X)\big\}$ and observe

$$\mathbb{E}(X) = \int_{\Omega\setminus\Omega_{\geq}} X \, d\omega + \int_{\Omega_{\geq}} X \, d\omega \leq q\mathbb{E}(X) + \|w^{-1}\mathbf{1}_{\Omega_{\geq}}\|_{L^{p/(p-1)}(\Omega)} \|wX\|_{L^p(\Omega)},$$

where $w$ is defined in Section 2.8. It remains to estimate $\|w^{-1}\mathbf{1}_{\Omega_{\geq}}\|_{L^{p/(p-1)}(\Omega)}$. We aim to prove $\|w^{-1}\|_{L^r(\Omega)} < \infty$ for $r > 1$ such that $r(d-k-\delta_{\mathrm{reg}}) < d-k$ with $k = 0, \dots, d-1$. To that end, we bound the integrand $|w|^{-r}$ from above by functions of the form $x \mapsto \operatorname{dist}(E,x)^{-\alpha}$, with $0 < \alpha < d-k$ and $E$ denoting a facet of dimension $k$. All functions of this type are in $L^1(\Omega)$ and we obtain

$$\|w^{-1}\mathbf{1}_{\Omega_{\geq}}\|_{L^{p/(p-1)}(\Omega)} \leq \|w^{-1}\|_{L^r(\Omega)} \|\mathbf{1}_{\Omega_{\geq}}\|^{(p-1)/p}_{L^{r'}(\Omega)} \lesssim |\Omega_{\geq}|^{p/(r'(p-1))},$$

where $r' = r/(r - p/(p-1))$. A Hölder inequality shows

$$\|wX\|^p_{L^p(\Omega)} \leq \mathbb{E}(X) \|w^p X^{p-1}\|_{L^\infty(\Omega)} \lesssim \mathbb{E}(X) \|X\|^{p-1}_{L^\infty_w(\Omega)}$$ ∎

and hence concludes the proof.

Let $\mathcal{T}_\infty := \bigcup_{\mathcal{T}\in\mathbb{T}} \mathcal{T}$ denote the set of all possible elements which may appear as refinements of some elements in $\mathcal{T}_0$. With an error estimator $\eta(T,V)$, which for now is just a function depending on $T \in \mathcal{T}_\infty$ and $V \in L^2(D)$, we base the construction of our deep RNN ADAPTIVE on the following greedy algorithm:

---

Input: Function $V \in H^1(D)$, tolerance $\varepsilon > 0$, initial mesh $\mathcal{T}_0$
For $\ell = 0, 1, 2, \dots$ do:
- (i) Compute $\eta(T,V)$ for all $T \in \mathcal{T}_\ell$, if $\eta(T,V) \leq \varepsilon$ for all $T \in \mathcal{T}_\ell$, stop.
- (ii) Find $T_0 \in \mathcal{T}_\ell$ with maximal $\eta(T_0,V)$.
- (iii) Bisect $T_0$ with newest-vertex-bisection to generate $\mathcal{T}_{\ell+1}$ and goto (i) (no mesh closure at this point).

Output: In case algorithm terminates at $\ell \in \mathbb{N}$, it produces a mesh $\mathcal{T}_\varepsilon := \mathcal{T}_\ell$ with $\eta(T,V) \leq \varepsilon$ for all $T \in \mathcal{T}_\varepsilon$.

---

The following lemma states that Algorithm 4.5 produces the minimal mesh to satisfy the given tolerance $\varepsilon$ in the maximum norm. Since Algorithm 4.5 does not perform mesh-closure, we define $\mathbb{T}_{\mathrm{nc}} \supset \mathbb{T}$ as the set of meshes which can be generated from $\mathcal{T}_0$ by iterated newest-vertex-bisection without mesh-closure.

LEMMA 4.16. *Let* $\mathcal{T} \in \mathbb{T}_{\mathrm{nc}}$ *denote a mesh with* $\max_{T\in\mathcal{T}} \eta(T,V) \leq \varepsilon$, *then* $\mathcal{T}$ *is a refinement of* $\mathcal{T}_\varepsilon$ *generated by Algorithm 4.5. In this case, Algorithm 4.5 terminates.*

*Proof.* First, assume that Algorithm 4.5 terminates and produces some mesh $\mathcal{T}_\varepsilon$. Assume that $\mathcal{T}$ is not a refinement of $\mathcal{T}_\varepsilon$. By the binary tree structure of newest-vertex-bisection, this implies the existence of a descendant $T \in \mathcal{T}_\varepsilon \setminus \mathcal{T}$ of some element $T' \in \mathcal{T}$. In this case, however, $\eta(T',V)$ must have been picked for refinement in Step (ii) of Algorithm 4.5 in some intermediate step $k \in \mathbb{N}$ (otherwise it would not have been refined). This, however, implies $\eta(T',V) > \varepsilon$ and hence contradicts the definition of $\mathcal{T}$.

Second, if Algorithm 4.5 does not terminate, we obtain a sequence of meshes $\mathcal{T}_\ell$ which is refined arbitrarily often. This implies that we may define $\mathcal{T}_\varepsilon := \mathcal{T}_\ell$ for sufficiently large $\ell \in \mathbb{N}$ such that $\mathcal{T}$ is not a refinement of $\mathcal{T}_\varepsilon$. Then, the arguments of the first part of the proof apply analogously. ∎

In the following, we emulate Algorithm 4.5 with a deep RNN. The major obstacle is that we usually can not compute the required error estimator $\eta(T,V)$ exactly only using deep RNNs ($\eta(T,V)$ will contain some $L^2$-norm and hence must be approximated). To circumvent this, we assume the existence of a random variable $\rho(T,V) \geq 0$ (to be constructed later by means of Monte Carlo sampling) with $\mathbb{E}(\rho(T,V)) \leq q^{-1}\eta(T,V)$ and $\mathbb{P}(\rho(T,V) \geq q\eta(T,V)) \geq \gamma > 0$ for some $q > 0$ and $\gamma > 0$. The following algorithm is similar to Algorithm 4.5 with high probability as shown in Lemma 4.17 below.

---

Input: Function $V \in H^1(D)$, tolerance $\varepsilon > 0$, initial mesh $\mathcal{T}_0$
For $\ell = 0, 1, 2, \dots$ do:
- (i) For all $T \in \mathcal{T}_\ell$ do:
  - (a) Sample $\rho(T,V)$ $K$-times.
  - (b) If all samples satisfy $\rho(T,V) \leq \varepsilon$ add $T$ to $\mathcal{T}_{\text{stop}}$.
  - (c) Otherwise, add $T$ to $\mathcal{M}_\ell$.
- (ii) Remove all elements $T$ from $\mathcal{M}_\ell$ for which there exists $T' \in \mathcal{T}_{\text{stop}}$ with $T \subseteq T'$.
- (iii) Generate $\mathcal{T}_{\ell+1}$ by refining the marked elements $\mathcal{M}_\ell$ with newest vertex bisection and mesh closure.

Output: The algorithm terminates if $\mathcal{T}_\ell = \emptyset$ and outputs $\mathcal{T}_\varepsilon$.

---

LEMMA 4.17. *Let $\mathcal{T}_\varepsilon$ denote the output of Algorithm 4.5 and $\mathcal{T}'_\delta$ denote the output of Algorithm 4.5 with $\delta > 0$. Then, there holds*

$$\mathbb{P}\big(\forall T \in \mathcal{T}_\varepsilon : \eta(T,V) \leq \varepsilon/q\big) \geq 1 - (L+1)\#\mathcal{T}'_{\varepsilon/q} 2^{-m},$$

*where $L \in \mathbb{N}$ is the number of iterations of Algorithm 4.5 needed to produce $\mathcal{T}'_{\varepsilon/q}$ and $m = K/|\log(1-\gamma)|$. Furthermore, there holds $\mathbb{P}(\#\mathcal{T}_\varepsilon \leq C\max\{m, \#\mathcal{T}'_\delta\}) \geq 1 - 2^{-C\max\{m,\#\mathcal{T}'_\delta\}}$, where $C$ depends only on the shape regularity of $\mathcal{T}_0$ and $\delta \simeq \varepsilon/K$ with hidden constants depending additionally on $q$.*

*Proof.* In the following, meshes with a dash, e.g., $\mathcal{T}'$, always denote meshes generated by Algorithm 4.5. To compare Algorithm 4.5 with Algorithm 4.5, we want to bound the probability of

$$\text{(E1):} \quad \eta(T,V) > \varepsilon/q \quad \text{but } T \text{ is added to } \mathcal{T}_{\text{stop}}.$$

If (E1) does not occur during the runtime of the algorithm, then the output $\mathcal{T}_\varepsilon$ satisfies

$$\eta(T,V) \leq \varepsilon/q \quad \text{for all } T \in \mathcal{T}_\varepsilon.$$

The sampling procedure in Step (i) of Algorithm 4.5 ensures that (E1) happens in step $\ell$ with probability less than $\#\mathcal{T}_\ell(1-\gamma)^K$. Thus, to ensure that (E1) does not occur with high probability over the runtime of the algorithm, we choose $K = |\log(1-\gamma)|m$. For $L \in \mathbb{N}$ iterations of Algorithm 4.5, this guarantees that (E1) occurs with probability less than

$$\sum_{\ell=0}^{L} \#\mathcal{T}_\ell 2^{-m} \leq (L+1)\#\mathcal{T}_L 2^{-m}.$$

If we set $L$ to the number of iterations of Algorithm 4.5 to produce $\mathcal{T}'_{\varepsilon/q}$, then we conclude $\mathbb{P}\big(\forall T \in \mathcal{T}_\varepsilon : \eta(T,V) \le \varepsilon/q\big) \ge 1 - (L+1)\#\mathcal{T}_L 2^{-m}$ since without (E1), $\mathcal{T}_\varepsilon$ is a refinement of $\mathcal{T}'_{\varepsilon/q}$.

To estimate the number of additional refinements compared to Algorithm 4.5, we want to bound the probability of

$$\text{(E2):}\quad \eta(T,V) \le q\varepsilon/C \qquad \text{but } T \text{ is not added to } \mathcal{T}_{\mathrm{stop}}.$$

Note that (E2) occurs if $\rho(T,V) > \varepsilon$ for at least one of $K$ independent samples despite $\eta(T,V) \le q\varepsilon/C$. Markov's inequality shows for all $C > 0$

$$\mathbb{P}(\rho(T,V) \ge Cq^{-1}\eta(T,V)) \le 1/C.$$

Therefore, the probability that (E2) occurs for an element $T \in \mathcal{T}_\ell$ is bounded by $1-(1-1/C)^K$. Since $\mathcal{T}'_{q\varepsilon/C}$ is the coarsest mesh to satisfy $\eta(T,V) \le q\varepsilon/C$ for all its elements, (E2) can only occur on elements of $\mathcal{T}'_{q\varepsilon/C}$ or refinements of them.

Assume that (E2) occurs on $r \in \mathbb{N}$ elements before Algorithm 4.5 terminates. Then, together with the mesh-closure estimate from [41], the final mesh contains less than $C_{\mathrm{cl}}(\#\mathcal{T}'_{q\varepsilon/C} + r)$ elements, where $C_{\mathrm{cl}} > 0$ depends only on the shape regularity $\mathcal{T}_0$.

We will now consider refinement forests $\mathcal{F}$, which are rooted in $\mathcal{T}'_{q\varepsilon/C}$. These are $\#\mathcal{T}'_{q\varepsilon/C}$ binary trees with root nodes corresponding to the elements of $\mathcal{T}'_{q\varepsilon/C}$. We denote the total number of leaves of $\mathcal{F}$ by $N(r)$. The leaves of a forest $\mathcal{F}$ correspond to a refinement $\mathcal{T}$ of $\mathcal{T}'_{q\varepsilon/C}$. Not every binary forest corresponds to a conforming mesh without hanging nodes, but every newest-vertex-bisection mesh can be represented by at least one ordered binary forest (ordered in the sense that every parent node of every tree has exactly zero children or exactly one *left* and one *right* child). The number of different binary trees with $n$ leaves is given by the Catalan number $C_{n-1}$, where $C_n := \binom{2n}{n}/(n+1)$, see, e.g., [39, Example 5.3.12]. Thus, the total number $M(r,s)$ of different forests $\mathcal{F}$ with $s$ roots and $N(r) \ge s$ leaves is given by

$$M(r,s) := \sum_{i_1+\ldots+i_s = N(r)-s} C_{i_1} C_{i_2} \cdots C_{i_s}.$$

A combinatorial identity (see, e.g. [9]) shows

$$M(r,s) = \begin{cases} \frac{s(N(r)-s+1)(N(r)-s+2)\cdots(N(r)-s/2-1)}{2(N(r)-s/2+2)(N(r)-s/2+3)\cdots N(r)} C_{N(r)-s/2} & s \text{ is even} \\ \frac{s(N(r)-s+1)(N(r)-s+2)\cdots(N(r)-(s+1)/2)}{2(N(r)-(s-3)/2)(N(r)-(s-3)/2+1)\cdots N(r)} C_{N(r)-(s+1)/2} & s \text{ is odd} \end{cases}$$
$$\le \frac{s}{2} C_{N(r)-\lceil s/2\rceil} \le \frac{s}{2N(r)-s+1} \binom{2(N(r)-\lceil s/2\rceil)}{N(r)-\lceil s/2\rceil} \le \frac{s(2e)^{N(r)-\lceil s/2\rceil}}{2N(r)-s+1} \le (2e)^{N(r)},$$

where we used $\binom{n}{k} \le (en/k)^k$ in the last estimate.

The probability $\mathbb{P}(\mathcal{F})$ that one particular forest $\mathcal{F}$ occurs can be calculated as follows: Due to Step (ib) of Algorithm 4.5, there is exactly one opportunity for (E2) to happen at each node of each tree $\mathcal{R}$ of $\mathcal{F}$ (if the element does not get refined by (E2), it never will be refined again by (E2)).

Thus, $\mathbb{P}(\mathcal{F})$ is bounded by the probability that $r$ instances of (E2) occur at some nodes of $\mathcal{F}$. Since a binary tree with at most $N(r)$ leaves has at most $2N(r)-1$ nodes, a binary forest with $s$ roots and $N(r)$ leaves has $2N(r)-s$ nodes. This implies

$\mathbb{P}(\mathcal{F}) \leq \binom{2N(r)-s}{r}(1-(1-1/C)^K)^r$. Thus, the probability $\mathbb{P}(r)$, that Algorithm 4.5 produces an arbitrary forest with $N(r)$ leaves, is bounded by

$$\begin{aligned}\mathbb{P}(r) &\leq M(r,s)\binom{2N(r)-s}{r}(1-(1-1/C)^K)^r\\ &\leq \binom{2N(r)-s}{r}(2e)^{N(r)}(1-(1-1/C)^K)^r.\end{aligned}$$

Choosing $r \geq \#\mathcal{T}'_{q\varepsilon/C}$, and $C = \kappa K$, we obtain $(1-(1-1/C)^K) \leq 1/\kappa$ as well as $\binom{2N(r)-s}{r} \leq (4C_{\mathrm{cl}}e)^r$. Since $N(r) \leq 2C_{\mathrm{cl}}r$, this shows $\mathbb{P}(r) \leq (8C_{\mathrm{cl}}e^2)^{2C_{\mathrm{cl}}r}\kappa^{-r}$. Finally, the choice $\kappa = (16C_{\mathrm{cl}}e^2)^{2C_{\mathrm{cl}}}$ bounds the probability by

$$\mathbb{P}(r) \leq 2^{-2C_{\mathrm{cl}}r}.$$

Thus, the probability of $\mathcal{T}_\varepsilon$ having more than $\max\{C_{\mathrm{cl}}(\#\mathcal{T}'_{q\varepsilon/C}+m), 2C_{\mathrm{cl}}\#\mathcal{T}'_{q\varepsilon/C}\}$ elements is thus bounded by

$$\sum_{r=\max\{m,\#\mathcal{T}_{q\varepsilon/C}\}}^{\infty} 2^{-N(r)} \lesssim 2^{-\max\{C_{\mathrm{cl}}(\#\mathcal{T}'_{q\varepsilon/C}+m),2C_{\mathrm{cl}}\#\mathcal{T}'_{q\varepsilon/C}\}}.$$

This concludes the proof. ∎

LEMMA 4.18. *For $V \in L^2(D)$ and $T \in \mathcal{T}$, define*

$$\rho(T,V) := \Big(\frac{|T|}{N}\sum_{i=1}^N \big(V(\phi_T(x_i)) - \frac{1}{N}\sum_{j=1}^N V(\phi_T(y_j))\big)^2\Big)^{1/2},$$

*where $x_1,\dots,x_N$ and $y_1,\dots,y_N$ are uniformly i.i.d. points on the reference element $T_{\mathrm{ref}}$ and $\phi_T\colon T_{\mathrm{ref}} \to T$ is the affine transformation. Then, there holds*

$$\mathbb{E}\rho(T,V)^2 = (1+1/N)\eta(T,V)^2 := (1+1/N)\|V-\Pi_T^0 V\|^2_{L^2(T)}.$$

*Assume that $V^2 \in L^\infty_w(D)$. Then, there exists $C,\gamma > 0$ such that each $T \in \mathcal{T}_\infty$ satisfies*

$$\mathbb{P}(\rho(T,V)^2 \geq (1+1/N)\eta(T,V)^2 \geq \gamma.$$

*The constants $\gamma$ and $C$ depend only on $N$, $\|V^2\|_{L^\infty_w(D)}$, and the constant $C$ from Lemma 4.15.*

*Proof.* Expansion of $\rho(T,V)$ as well as the independence of the Monte Carlo points show

$$\begin{aligned}&N\mathbb{E}\rho(T,V)^2\\ &= \sum_{i=1}^N \mathbb{E}\Big(|T|V(\phi_T(x_i))^2 - \frac{2|T|}{N}V(\phi_T(x_i))\sum_{j=1}^N V(\phi_T(y_j))\\ &\qquad + \frac{|T|}{N^2}\sum_{j,k=1}^N V(\phi_T(x_j))V(\phi_T(x_k))\Big)\\ &= \sum_{i=1}^N \Big(\int_T V^2\,dx - 2|T|^{-1}(\int_T V\,dx)^2 + |T|^{-1}(1-\frac{1}{N})(\int_T V\,dx)^2 + \frac{1}{N}\int_T V^2\,dx\Big)\\ &= N\Big((1+1/N)\int_T V^2\,dx - |T|(1+1/N)(\Pi_T^0 V)^2\Big) = N(1+1/N)\|V-\Pi_T^0 V\|^2_{L^2(T)}.\end{aligned}$$

To show the second statement, we employ Lemma 4.15. To that end, note that $X := \rho(T,V)^2$ is a non-negative random variable on $\Omega := \bigcup_{i=1}^N T_{\mathrm{ref},i} \subset \mathbb{R}^d$, where $T_{\mathrm{ref},i}$ are pairwise disjoint copies of the reference element scaled to $|T_{\mathrm{ref},i}| = 1/N$. We define $w_\Omega$ analogously to $w$ on $\Omega$ with respect to the transformed singularity set $S_\Omega := \bigcup_{i=1}^N \phi_{T,i}^{-1}(S \cap T)$, where $\phi_{T,i}\colon T_{\mathrm{ref},i} \to T$ are the affine element mappings. Note that there holds $\operatorname{diam}(T)^d w_\Omega(x) \lesssim w \circ \phi_{T,i}(x)$ by definition of the weight $w$. By assumption, we have $V^2 \in L^\infty_w(D)$ and hence

$$\|w_\Omega X\|_{L^\infty(\Omega)} \lesssim |T| \|w_\Omega V^2 \circ \phi_T\|_{L^\infty(\Omega)} \lesssim \|V^2\|_{L^\infty_w(D)},$$

where we used $|T| \simeq \operatorname{diam}(T)^d$. Hence, Lemma 4.15 applies and proves $\mathbb{P}(X \geq q\mathbb{E}(X)) \gtrsim 1$. This concludes the proof. ∎

Finally, we complete Algorithm 4.5 by replacing the theoretical error estimator $\rho(T,V)$ by the concrete DNN approximation $\rho_T$.

---

Input: Function $V \in H^1(D)$, tolerance $\varepsilon > 0$, initial mesh $\mathcal{T}_0$
For $\ell = 0, 1, 2, \dots$ do:
- (i) For all $T \in \mathcal{T}_\ell$ do:
  - (a) Sample $\rho_T$ $K$-times.
  - (b) If all samples satisfy $\rho_T \leq \varepsilon$ add $T$ to $\mathcal{T}_{\mathrm{stop}}$.
  - (c) Otherwise, add $T$ to $\mathcal{M}_\ell$.
- (ii) Remove all elements $T$ from $\mathcal{M}_\ell$ for which there exists $T' \in \mathcal{T}_{\mathrm{stop}}$ with $T \subseteq T'$.
- (iii) Generate $\mathcal{T}_{\ell+1}$ by refining the marked elements $\mathcal{M}_\ell$ with newest vertex bisection and mesh closure.

Output: The algorithm terminates if $\mathcal{M}_\ell = \emptyset$ and outputs $\mathcal{T}_\varepsilon := \mathcal{T}_\ell$.

---

THEOREM 4.19. *Let $u \in H^1(D)$. Given $\varepsilon > 0$, let $v_\varepsilon$ denote an approximation to $\nabla u$ which satisfies $\|\nabla u - v_\varepsilon\|_{L^2(D)} \leq \varepsilon$ as well as $v_\varepsilon^2 \in L^\infty_w(D)$. We assume that the refinement indicator $\rho_T$ satisfies $|\rho_T - \rho(T, v_\varepsilon)| \leq \varepsilon/4$ with $\rho(T,\cdot)$ from Lemma 4.18. We denote by $\mathcal{T}_\varepsilon$ the output of Algorithm 4.5 as well as by $\mathcal{T}'_\delta$ the output of Algorithm 4.5 (with $V := v_\varepsilon$). Then, there holds*

$$\mathbb{P}\big(\forall T \in \mathcal{T}_\varepsilon : \|\nabla u - \Pi^0_{\mathcal{T}_\varepsilon} \nabla u\|_{L^2(T)} \leq (1 + 2/q)\varepsilon\big) \geq 1 - (L+1)\#\mathcal{T}'_{2\varepsilon/q} 2^{-m},$$

*where $L \in \mathbb{N}$ is the number of iterations of Algorithm 4.5 needed to produce $\mathcal{T}'_{2\varepsilon/q}$ and $m = K/|\log(1-\gamma)|$. Furthermore, there holds $\mathbb{P}(\#\mathcal{T}_\varepsilon \leq C \max\{m, \#\mathcal{T}'_\delta\}) \geq 1 - 2^{-C\max\{m, \#\mathcal{T}'_\delta\}}$, where $C$ depends only on the shape regularity of $\mathcal{T}_0$, $N$ in the definition of $\rho(T,\cdot)$, and $\|v_\varepsilon^2\|_{L^\infty_w(D)}$. There holds $\delta \simeq \varepsilon/K$ with hidden constants depending additionally on $q$.*

*Proof.* We define

$$\widetilde{\rho}_T := \begin{cases} \rho(T, v_\varepsilon) & \rho(T, v_\varepsilon) < \varepsilon/2, \\ \rho_T & \text{else.} \end{cases}$$

Note that $\widetilde{\rho}_T \leq \varepsilon$ if and only if $\rho_T \leq \varepsilon$. Hence, Algorithm 4.5 produces exactly the same output when we replace $\rho_T$ with $\widetilde{\rho}_T$. We prove

$$\widetilde{\rho}_T/2 \leq \rho(T, v_\varepsilon) \leq 2\widetilde{\rho}_T. \tag{4.3}$$

To see this, we only need to consider the case $\rho(T,v_\varepsilon)\geq\varepsilon/2$. There holds $\widetilde{\rho}_T=\rho_T\leq\rho(T,v_\varepsilon)+\varepsilon/4\leq 2\rho(T,v_\varepsilon)$ as well as $\rho(T,v_\varepsilon)\leq\rho_T+\varepsilon/4\leq\rho_T+\rho(T,v_\varepsilon)/2$. This concludes (4.3).

Lemma 4.18 confirms that $\mathbb{E}(\widetilde{\rho}_T)\leq 2q^{-1}\eta(T,v_\varepsilon)$ as well as $\mathbb{P}(\widetilde{\rho}_T\geq q/2\eta(T,v_\varepsilon))\geq\gamma$ are satisfied for all elements $T\in\mathcal{T}_\infty$. Thus, Lemma 4.17 applies and concludes

$$\mathbb{P}\big(\forall T\in\mathcal{T}_\varepsilon:\,\eta(T,v_\varepsilon)\leq 2/q\varepsilon\big)\geq 1-(L+1)\#\mathcal{T}'_{2\varepsilon/q}2^{-m},$$

as well as $\mathbb{P}(\#\mathcal{T}_\varepsilon\leq C\#\mathcal{T}'_\delta)\geq 1-2^{-C\#\mathcal{T}'_\delta}$. Note that $\mathcal{T}'_\delta$ is generated by Algorithm 4.5 with $V=v_\varepsilon$. By assumption $\|\nabla u-v_\varepsilon\|_{L^2(D)}\leq\varepsilon$, we have $\|\nabla u-\Pi^0_{\mathcal{T}_\varepsilon}\nabla u\|_{L^2(T)}\leq\eta(T,v_\varepsilon)+\varepsilon$ and hence conclude the proof. ∎

*Proof of Theorem 2.5.* Again we denote the meshes generated by Algorithm 4.5 by $\mathcal{T}'_\delta$. We aim to apply Theorem 4.19 with $N=1$. Under the assumption that the input $v_\varepsilon$ satisfies $\|\nabla u-v_\varepsilon\|_{L^2(D)}\leq\varepsilon$, we use summation and taking the absolute value to construct a DNN $\rho_T$ which computes

$$\rho_T:=|T|^{1/2}\odot\big|v_\varepsilon(x)-v_\varepsilon(y)\big|.$$

The DNN $\rho_T$ takes as input two samples of uniform i.i.d. points $x_T,y_T\in T$ as well as the square root of the element area $|T|^{1/2}$ (this could also be computed via DNNs but we aim to keep the construction simple). The approximate multiplication $\odot$ is realized via MULTIPLY and achieves accuracy $\mathcal{O}(\varepsilon')$ with a DNN of the size $\mathcal{O}(|\log(\varepsilon')|+|\log(\|v_{\varepsilon'}\|_{L^\infty(D)})|)$ (see Proposition 4.6). Choosing $\varepsilon'\simeq\varepsilon$ sufficiently small, we ensure $|\rho_T-\rho(T,v_\varepsilon)|\leq\varepsilon/4$.

To denote the independent samples required from $\rho_T$ we define $\rho_{T,k}$ as the $k$-th sample computed with input $x_{T,k},y_{T,k}\in T$. We construct the RNN ADAPTIVE via

$$y_{T_i}=\mathrm{ADAPTIVE}(\boldsymbol{x}_{T_i}):=\max\{\max_{k=1,\dots,K}\rho_{T_i,k},\varepsilon\}-\varepsilon,$$

where $\boldsymbol{x}_{T_i}:=(x_{T,k},y_{T,k},|T|^{1/2},\varepsilon)_{k=1,\dots,K}$ and $K\simeq m$.

This already proves the complexity bound in Theorem 2.5. Furthermore, Algorithm 2.8 with ADAPTIVE is equivalent to Algorithm 4.5 with $\rho_T$. Hence Theorem 4.19 applies with $q=1/2$ and proves

$$\mathbb{P}\big(\forall T\in\mathcal{T}_\varepsilon:\,\|\nabla u-\Pi^0_{\mathcal{T}_\varepsilon}\nabla u\|_{L^2(T)}\leq 5\varepsilon\big)\geq 1-(L+1)\#\mathcal{T}'_{4\varepsilon}2^{-m},$$

as well as $\mathbb{P}(\#\mathcal{T}_\varepsilon\leq C\max\{m,\#\mathcal{T}'_\delta\})\geq 1-2^{-C\max\{m,\#\mathcal{T}'_\delta\}}$ with $\mathcal{T}'_\delta$ denoting the output of Algorithm 4.5 with $V=v_\varepsilon$ and $\delta\simeq\varepsilon/m$. Let $\mathcal{T}$ denote a mesh which satisfies (2.8) for minimal $N\in\mathbb{N}$ with $N^{-s-1/2}\leq\delta/2$, i.e.

$$\eta(T,\nabla u)\leq N^{-s-1/2}\leq\delta/2\quad\text{for all }T\in\mathcal{T}.$$

Then, we have $\eta(T,v_\varepsilon)\leq\delta/2+\varepsilon/(Cm)\leq\delta$ and Lemma 4.16 implies that $\mathcal{T}$ is a refinement of $\mathcal{T}'_\delta$, i.e., $\#\mathcal{T}'_\delta\leq\#\mathcal{T}\leq N$. This shows that with probability larger than $(1-(L+1)\#\mathcal{T}'_{4\varepsilon}2^{-m})(1-2^{-C\max\{m,\#\mathcal{T}'_\delta\}})$, there holds

$$\#\mathcal{T}_\varepsilon\max_{T\in\mathcal{T}_\varepsilon}\|\nabla u-\Pi^0_{\mathcal{T}_\varepsilon}\nabla u\|^{1/(s+1/2)}_{L^2(T)}\lesssim\max\{m,N\}(5\varepsilon)^{1/(s+1/2)}.$$

Since $N-1\lesssim(\varepsilon/m)^{-1/(s+1/2)}$, we conclude for $m\leq\#\mathcal{T}_\varepsilon$

$$\#\mathcal{T}_\varepsilon\max_{T\in\mathcal{T}_\varepsilon}\|\nabla u-\Pi^0_{\mathcal{T}_\varepsilon}\nabla u\|^{1/(s+1/2)}_{L^2(T)}\lesssim m^{1/(s+1/2)}$$

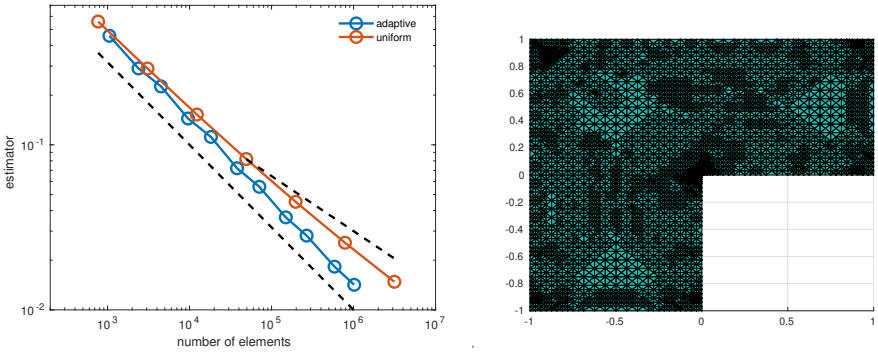



FIG. 5.1. *(left) Comparison of the performance of the RNN* ADAPTIVE *versus uniform mesh refinement. The dashed lines indicate the expected rates for uniform/adaptive refinement* $\mathcal{O}(N^{-1/3})$ *and* $\mathcal{O}(N^{-1/2})$. *(right) Adaptive mesh generated by the RNN* $\widetilde{\text{MARK}}$ *found by stochastic gradient descent.*

with probability larger than

$$(1-(L+1)\#\mathcal{T}'_{4\varepsilon}2^{-m})(1-2^{-C\max\{m,\#\mathcal{T}'_\delta\}}) \geq 1-((L+1)\#\mathcal{T}'_{4\varepsilon}+1)2^{-Cm}.$$

Note that $\mathcal{T}(4\varepsilon)$ as defined in Section 2.8 is conforming and thus a refinement of $\mathcal{T}'_{4\varepsilon}$ (according to Lemma 4.16). Since $L$ is the number of iterations of Algorithm 4.5, we know that the maximal level of elements in $\mathcal{T}'_{4\varepsilon}$ is equal to $L$. This concludes the proof. ∎

## 5. Numerical Experiments.

**5.1. Hardcoded deep RNN.** As a first experiment, we implement the RNN ADAPTIVE from Theorem 2.2 exactly as shown in the proofs of Section 4. We run Algorithm 2.7 on an L-shaped domain shown in Figure 5.2. We choose a constant right-hand side $f = 1$ and start from a coarse triangulation with six elements. Figure 5.1 shows that the adaptive method reaches the expected convergence rate of $\mathcal{O}(N^{-1/2})$, while the uniform approach only achieves a suboptimal rate due to the singularity at the re-entrant corner of the domain. Figure 5.2 compares the adaptive meshes generated by ADAPTIVE to a standard adaptive mesh generated by Algorithm 2.1. This experiment's main purpose is to show that round-off errors do not spoil the theoretically shown performance.

The more interesting experiment would be to find the the weights of ADATIVE by means of computational optimization (machine learning) as described in Section 3. We do not cover this topic in its entirety, because the training of RNNs is a challenging topics on itself. However, we achieve some intermediate goals in the following two sections.

**5.2. Learning the maximum strategy.** First, we try to find an RNN $\widetilde{\text{MARK}}$ which, given the exact residual based error estimator from (2.6), marks elements and achieves the optimal order of convergence. To that end, we use the smallest possible blue print for an RNN such that it can represent the maximum strategy (which is much simpler than Dörfler marking). The maximum strategy defines the set of marked elements as

$$\mathcal{M} := \big\{T \in \mathcal{T} : \rho_T > (1-\theta)\max_{T'\in\mathcal{T}} \rho_{T'}\big\}.$$

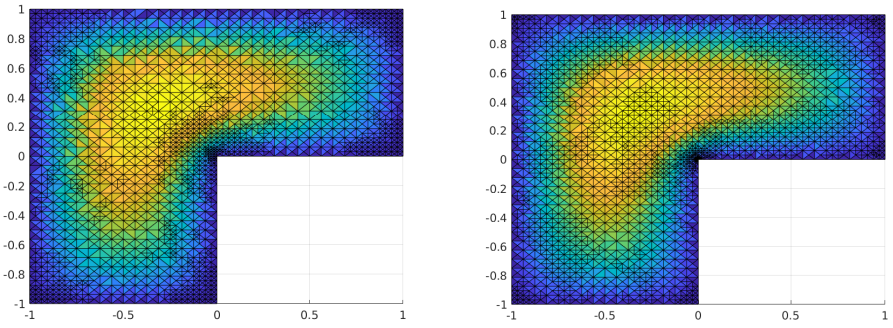


FIG. 5.2. *Comparison of the adaptive meshes generated after 9 steps of adaptive refinement with the RNN* ADAPTIVE *(left) and the standard residual error estimator* (2.6) *with Dörfler marking (right).*

Although it is not known whether the maximum strategy leads to optimal convergence in the sense of (2.4), it is usually observed in practice and [16] even shows optimality for a slight variation of this strategy. The maximum strategy can be realized by the combination of two basic RNNs. First, $B_1$ is defined for an input $\boldsymbol{x} \in \mathbb{R}^{\#\mathcal{T}}$, $x_i = \rho_{T_i}$ and output $\boldsymbol{y} \in \mathbb{R}^{2\times\#\mathcal{T}}$ by

$$y_i = B_1(x_i, y_{i-1}) = (x_i, \max(x_i, y_{i-1,2})) = \begin{pmatrix} 1 & -1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \phi\left(\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_{i-1,2} \end{pmatrix}\right).$$

Initialization with $y_{0,2} = 0$ results in $y_{i,1} = x_i$ as well as $y_{i,2} = \max_{1\le j\le i} x_i$ for all $1 \le i \le \#\mathcal{T}$. Then, we initialize a second basic RNN $B_2$ with $\boldsymbol{x} \in \mathbb{R}^{2\times\#\mathcal{T}}$, $x_{i,1} = y_{i,1}$ and $x_{i,2} = y_{\#\mathcal{T},2}$ for $i = 1, \dots, \#\mathcal{T}$ (note that if we insist on the initialization as described in Section 2.5, we need a third RNN to copy the value of $y_{\#\mathcal{T},2}$ to the entire vector). We filter the marked elements by

$$y_i = B(x_i) = \max(x_{i,1} - (1-\theta)x_{i,2}, 0)$$

and observe that $\mathcal{M} = \{T_i \in \mathcal{T} : y_i > 0\}$. Now, we know the structure necessary to represent the maximum strategy.

To find $\widetilde{\text{MARK}}$ by machine learning, we set up a simple optimization algorithm to compute the necessary weights. We initialize an RNN with the structure as given above with random weights, run Algorithm 2.1 with Step (3) replaced by our RNN as long as $\#\mathcal{T}_\ell \le 2 \cdot 10^4$, and apply simultaneous perturbation stochastic approximation (SPSA) to maximize the energy norm of the finest computed solution (note that due to Galerkin orthogonality, maximizing the energy norm is equivalent to minimizing the error).

The SPSA approach is basically a stochastic gradient descent algorithm which replaces the gradient by a finite difference in a random direction (see, e.g. [6] for details). As discussed in the previous section, this is necessary since marking is not a continuous procedure. As discussed in Section 3, we limited the values of the recursive weights to the set $\{-1, 0, 1\}$ to avoid blow-up or dampening. The weights found by

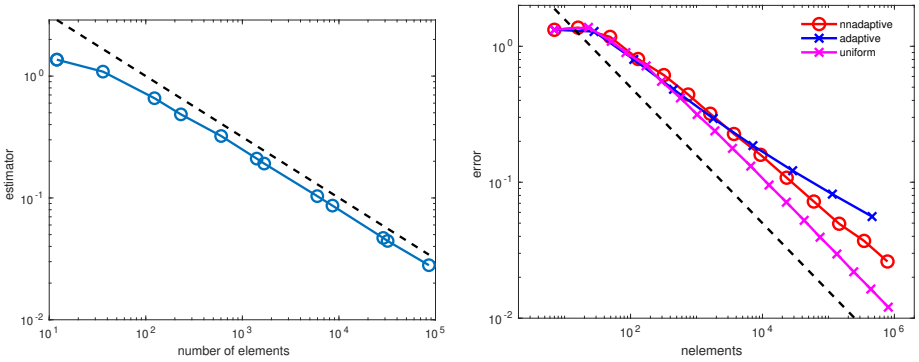



FIG. 5.3. *(left) Performance of the RNN* $\widetilde{\text{MARK}}$ *as a marking strategy in Algorithm 2.1. The dashed line marks the optimal rate of convergence* $\mathcal{O}(N^{-1/2})$*. (right) Result (in terms of the residual error estimator plotted over the number of elements) of the* training on the job *Algorithm 5.3 compared with uniform refinement and (optimal) adaptive refinement via the residual error estimator. We observe that the deep RNN* ADAPTIVE *(which is trained on the fly) clearly beats the uniform refinement and almost achieves optimal rate of convergence. Even with the very crude learning method described in Remark 5.1, the deep RNN approach leads to an advantage over uniform refinement without using any information about the problem such as error estimators. This improved rate implies particularly that as long as the training cost is proportional to the cost of the solve step, this approach will eventually also be more cost effective than plain uniform refinement.*

the algorithm for $B_1$ and $B_2$ are

$$\begin{pmatrix} -0.2471 & 0.1095 & -0.2358 \\ -0.1868 & 0.3123 & -0.9564 \end{pmatrix}, \quad \begin{pmatrix} 0.4394 & 0 \\ -0.6591 & 0 \\ -0.6466 & -1 \end{pmatrix}, \quad \begin{pmatrix} -0.1585 & 0.2804. \end{pmatrix}$$

While we cannot offer a meaningful explanation of the marking strategy found, we observe in Figure 5.1 (right) and Figure 5.3 (left) that it behaves in an empirically optimal fashion and also the generated meshes look reasonable.

**5.3. Training on the job.** Finally, we try to learn the full deep RNN ADAPTIVE for the Poisson problem

$$\begin{aligned} -\Delta u &= f \quad \text{in } D, \\ u &= 0 \quad \text{on } \partial D. \end{aligned}$$

We choose a $Z$-shaped domain $D$ to increase the rate difference between optimal and uniform refinement, i.e., $D := [-1,1]^2 \setminus \operatorname{conv}\{(0,0),(-1,0),(-1,-1/5)\}$. There are many different plausible methods of how to train the network. We chose to set up a deep RNN which consists of two RNNs $B$ and $B'$ with the layer structure $(s_0,\dots,s_3) = (16,10,10,10)$ and $(s'_0,s'_1,s'_2) = (11,10,1)$. The two RNNs are only mildly recursive in the sense that $y_i = B(x_i, y_{i-1,1})$ (and analogously for $B'$) only has access to the first component of the previous vector valued output. The input data of $B$ is the sequence $x_1,\dots,x_{\#\mathcal{T}}$, where each $x_i$ contains $\nabla U_{\mathcal{T}}|_T \in \mathbb{R}^2$ for $T = T_i$ as well as for each neighbor element $T$ which shares an edge with $T_i$. Moreover, $x_i$ contains the coordinates of the nodes of $T$ as well as the midpoint evaluation of the right-hand side $f$. This results in $x_i \in \mathbb{R}^{15}$. Since the RNN $B$ also processes the first component of the previous output, the first layer has to consist of 16 nodes. For the same reason, the first layer of $B'$ has one more node than the final layer of $B$. We

call this section *training on the job* because the training of the neural network is part of the adaptive algorithm: The result of this algorithm is shown in Figure 5.3.

---

**Input:** Initial mesh $\mathcal{T}_0$, number of training steps $n_{\text{train}} \in \mathbb{N}$.
For $\ell = 0, 1, 2, \dots$ do:

1. Compute discrete approximation $U_\ell$.
2. For $k = 1, \dots, n_{\text{train}}$
   (a) Apply $\boldsymbol{y} = \text{ADAPTIVE}(\boldsymbol{x})$.
   (b) Use newest-vertex-bisection to refine $\#\mathcal{T}_\ell/5$ elements $T_i \in \mathcal{T}_\ell$ with largest $y_i$ to obtain $\widetilde{\mathcal{T}}_{\ell+1}$.
   (c) Optimize weights of $\text{ADAPTIVE}(\boldsymbol{x})$ with the goal to maximize $\|\nabla \widetilde{U}_{\ell+1}\|_{L^2(D)}$.
3. Apply $\boldsymbol{y} = \text{ADAPTIVE}(\boldsymbol{x})$.
4. Use newest-vertex-bisection to refine $\#\mathcal{T}_\ell/5$ elements $T_i \in \mathcal{T}_\ell$ with largest $y_i$ to obtain $\mathcal{T}_{\ell+1}$.

---

REMARK 5.1. *We note that Algorithm 5.3 is different to Algorithm 2.7 or 2.8 due to the fact that we always refine 20% of the elements and, in case of Algorithm 2.8 we we do not exclude elements which are not refined in a specific step. This is largely to simplify the training process of the network and to minimize the implementational overhead. The optimization step (2c) consists of trying $n_{\text{train}} = 50$ random perturbations of the network and choosing the best one. To avoid to optimize towards networks which always refine all elements (this would give the largest increase in energy $\|\nabla \widetilde{U}_{\ell+1}\|_{L^2(D)}$), we restrict ourselves to refining exactly 20 % of the elements. We are confident that a more sophisticated training method would improve the results. This, however, is beyond the scope of this work.*

## REFERENCES


[1] Gopala K. Anumanchipalli, Josh Chartier, and Edward F. Chang. Speech synthesis from neural decoding of spoken sentences. *Nature*, 568:493–498, 2019.

[2] Markus Aurada, Michael Feischl, Thomas Führer, Michael Karkulik, and Dirk Praetorius. Efficiency and optimality of some weighted-residual error estimator for adaptive 2D boundary element methods. *J. Comput. Appl. Math.*, 255:481–501, 2014.

[3] Christian Beck, Lukas Gonon, and Arnulf Jentzen. Overcoming the curse of dimensionality in the numerical approximation of high-dimensional semilinear elliptic partial differential equations. *2003.00596*, 2020.

[4] Roland Becker and Shipeng Mao. Quasi-optimality of adaptive nonconforming finite element methods for the Stokes equations. *SIAM J. Numer. Anal.*, 49(3):970–991, 2011.

[5] Roland Becker, Shipeng Mao, and Zhongci Shi. A convergent nonconforming adaptive finite element method with quasi-optimal complexity. *SIAM J. Numer. Anal.*, 47(6):4639–4659, 2010.

[6] S. Bhatnagar, H. L. Prasad, and L. A. Prashanth. *Stochastic recursive algorithms for optimization*, volume 434 of *Lecture Notes in Control and Information Sciences*. Springer, London, 2013. Simultaneous perturbation methods.

[7] Peter Binev, Wolfgang Dahmen, and Ronald DeVore. Adaptive finite element methods with convergence rates. *Numer. Math.*, 97(2):219–268, 2004.

[8] Peter Binev, Wolfgang Dahmen, Ronald DeVore, and Pencho Petrushev. Approximation classes for adaptive methods. volume 28, pages 391–416. 2002. Dedicated to the memory of Vassil Popov on the occasion of his 60th birthday.

[9] Douglas Bowman and Alon Regev. Counting symmetry classes of dissections of a convex regular polygon. *Advances in Applied Mathematics*, 56:35 – 55, 2014.

[10] Carsten Carstensen, Michael Feischl, Marcus Page, and Dirk Praetorius. Axioms of adaptivity. *Comput. Math. Appl.*, 67(6):1195–1253, 2014.

[11] Carsten Carstensen, Daniel Peterseim, and Hella Rabus. Optimal adaptive nonconforming FEM for the Stokes problem. *Numer. Math.*, 123(2):291–308, 2013.
[12] Carsten Carstensen and Hella Rabus. The adaptive nonconforming FEM for the pure displacement problem in linear elasticity is optimal and robust. *SIAM J. Numer. Anal.*, 50(3):1264–1283, 2012.
[13] J. Manuel Cascon, Christian Kreuzer, Ricardo H. Nochetto, and Kunibert G. Siebert. Quasi-optimal convergence rate for an adaptive finite element method. *SIAM J. Numer. Anal.*, 46(5):2524–2550, 2008.
[14] J. Manuel Cascon and Ricardo H. Nochetto. Quasioptimal cardinality of AFEM driven by nonresidual estimators. *IMA J. Numer. Anal.*, 32(1):1–29, 2012.
[15] Long Chen, Michael Holst, and Jinchao Xu. Convergence and optimality of adaptive mixed finite element methods. *Math. Comp.*, 78(265):35–53, 2009.
[16] Lars Diening, Christian Kreuzer, and Rob Stevenson. Instance optimality of the adaptive maximum strategy. *Found. Comput. Math.*, 16(1):33–68, 2016.
[17] S. El Hihi and Y Bengio. Hierarchical recurrent neural networks for long-term dependencies. *In NIPS 8. MIT Press*, 1996.
[18] Michael Feischl. Optimal adaptivity for non-symmetric fem/bem coupling. *arxiv-preprint 1710.06082*.
[19] Michael Feischl. Optimality of a standard adaptive finite element method for the Stokes problem. *SIAM J. Numer. Anal.*, 57(3):1124–1157, 2019.
[20] Michael Feischl, Thomas Führer, Michael Karkulik, Jens Markus Melenk, and Dirk Praetorius. Quasi-optimal convergence rates for adaptive boundary element methods with data approximation, part I: weakly-singular integral equation. *Calcolo*, 51(4):531–562, 2014.
[21] Michael Feischl, Thomas Führer, Michael Karkulik, Jens Markus Melenk, and Dirk Praetorius. Quasi-optimal convergence rates for adaptive boundary element methods with data approximation, part II: Hypersingular integral equation. *Electron. Trans. Numer. Anal.*, 44:153–176, 2015.
[22] Michael Feischl, Thomas Führer, and Dirk Praetorius. Adaptive FEM with optimal convergence rates for a certain class of nonsymmetric and possibly nonlinear problems. *SIAM J. Numer. Anal.*, 52(2):601–625, 2014.
[23] Michael Feischl, Michael Karkulik, J. Markus Melenk, and Dirk Praetorius. Quasi-optimal convergence rate for an adaptive boundary element method. *SIAM J. Numer. Anal.*, 51:1327–1348, 2013.
[24] Tsogtgerel Gantumur. Adaptive boundary element methods with convergence rates. *Numerische Mathematik*, 124(3):471–516, 2013.
[25] A. Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.
[26] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.
[27] Philipp Grohs and Lukas Herrmann. Deep neural network approximation for high-dimensional elliptic pdes with boundary conditions. *arXiv:2007.05384*, 2020.
[28] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv:1809.02362*, 2018.
[29] L. Herrmann, Ch. Schwab, and J. Zech. Deep relu neural network expression rates for data-to-qoi maps in bayesian pde inversion. Technical Report 2020-02, Seminar for Applied Mathematics, ETH Zürich, Switzerland, 2020.
[30] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[31] Huang Jian Guo and Xu Yi Feng. Convergence and complexity of arbitrary order adaptive mixed element methods for the Poisson equation. *Sci China Math*, 55(5):1083–1098, 2012.
[32] Christian Kreuzer and Kunibert G. Siebert. Decay rates of adaptive finite elements with Dörfler marking. *Numer. Math.*, 117(4):679–716, 2011.
[33] Shipeng Mao, Xuying Zhao, and Zhongci Shi. Convergence of a standard adaptive nonconforming finite element method with optimal complexity. *Appl. Numer. Math.*, 60:673–688, July 2010.
[34] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 543–548, Oct 2006.
[35] Joost A.A. Opschoor, Christoph Schwab, and Jakob Zech. Exponential relu dnn expression of holomorphic maps in high dimension. Technical report, Zurich, 2019-07.
[36] Hella Rabus. A natural adaptive nonconforming FEM of quasi-optimal complexity. *Comput.*

*Methods Appl. Math.*, 10(3):315–325, 2010.

[37] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4:234–242, 1992.

[38] Jürgen Schmidhuber, Daan Wierstra, and Faustino Gomez. Evolino: Hybrid neuroevolution / optimal linear search for sequence learning. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 853–858, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[39] Richard P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999. With a foreword by Gian-Carlo Rota and appendix 1 by Sergey Fomin.

[40] Rob Stevenson. Optimality of a standard adaptive finite element method. *Found. Comput. Math.*, 7(2):245–269, 2007.

[41] Rob Stevenson. The completion of locally refined simplicial partitions created by bisection. *Math. Comp.*, 77(261):227–241, 2008.

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