RECURRENT NEURAL NETWORKS AS OPTIMAL MESH REFINEMENT STRATEGIES

JAN BOHN AND MICHAEL FEISCHL

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.

1. Introduction

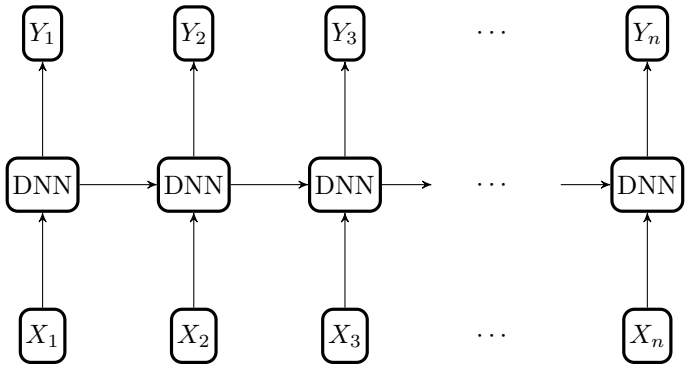
Adaptive methods for finite element mesh refinement had tremendous impact on the scientific community both on the theoretical side as well as on the applied, engineering side.

Following the seminal works [6, 29, 10] on the adaptive finite element method, a multitude of papers extended the ideas to numerous model problems and applications, see e.g., [23, 11] for conforming methods, [27, 3, 4, 8, 24] for nonconforming methods, [12, 9, 22] for mixed formulations, and [18, 19, 2, 15, 16] for boundary element methods (the list is not exhausted, see also [7] and the references therein). Quite recently, [14, 17] 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., [30]). A general drawback of adaptive mesh refinement methods is the implementational overhead involved in the error estimation and choosing elements which to refine, which might prevent a more widespread use in engineering and industry.

This encourages the use 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, they might provide exactly the required black-box tool. The most prominent examples of RNNs are Long-Term-Short-Term memory approaches proposed in [21] and since then hugely successful in practical applications, e.g., for time-series interpretation [28], speech recognition [20], speech synthesis [1], and even surgical robot control [25]. Very roughly, a recurrent neural network has the following structure



where the $X_1, X_2, \dots, X_n$ denote a (vector valued) input sequence and the $Y_1, Y_2, \dots, Y_n$ 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.

Both authors are supported by the Deutsche Forschungsgemeinschaft (DFG) through CRC 1173.

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 one replace the steps Estimate $\longrightarrow$ Mark by a recurrent neural network ADAPTIVE in order to achieve similar (or better) results than state of the art adaptive mesh refinement algorithms?

While this question is very ambitious and this work is only a step towards a final answer, we show that a recurrent neural network can (in theory) learn to refine the mesh in an optimal fashion. If implemented in the right way, this approach could improve the current state of adaptive mesh refinement in several ways.

First, while asymptotic optimal error estimators are known for some problems, their preasymptotic performance is largely unknown. It is the hope of the author that automated machine learning could come up with an estimate–refine scheme which drastically improves the preasymptotic behavior of known error estimators.

Second, for many problems there are no optimal adaptive algorithms known, or sometimes there are not even error estimators available. By training a neural net built from blue prints laid out in this work, one might be able to obtain optimal mesh refinement algorithms for those hard problems. Examples for those problems are, e.g., time-dependent equations like the Landau-Lifshitz-Gilbert equations or the Navier-Stokes equations.

Finally, although we only deal with h-refinement in this work, it is not hard to adapt the neural nets constructed below to also output p-refinement information. This approach might lead to robust hp-adaptive algorithms.

The present work shows that a mildly complex RNN can indeed emulate an optimal adaptive mesh refinement strategy. The major difficulty is that the number of trainable parameters of the RNN should not depend on the input size, i.e., the number of mesh elements of the current iteration of the adaptive algorithm. This ensures that, once trained, a RNN emulating the adaptive algorithm can be used for a variety of problem sizes and input parameters.

The remained 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 result. Section 3 provides all the sub assemblies for the RNN which emulates the adaptive algorithm. Section 4 offers some conclusions and topics for further research, while a final Section 5 underlines the theoretical findings by some numerical experiments.

2. Model Problem & Main Result

On the open Lipschitz domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, we consider a prototypical PDE of the form

$$\begin{aligned} \mathcal{L}u := -\mathrm{div}(A\nabla u) + b \cdot \nabla u + cu = f \quad &\text{in } \Omega, \\ u = 0 \quad &\text{on } \partial\Omega, \end{aligned} \tag{1}$$

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

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

satisfies $a(u, v) \leq C\|u\|_{H^1(\Omega)}\|v\|_{H^1(\Omega)}$ as well as $a(u, u) \geq C^{-1}\|u\|^2_{H^1(\Omega)}$ for some constant $C > 0$. The Lax-Milgram lemma guarantees a unique solution $u \in H^1(\Omega)$ of (1) which we can approximate with a finite element method. To that end, define a triangulation $\mathcal{T}$ of Ω 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. On this triangulation, we define the Ansatz and test spaces

$$\begin{aligned} \mathcal{P}^p(\mathcal{T}) &:= \big\{ v \in L^2(\Omega) \,:\, v|_T \text{ is a polynomial of degree } \leq p,\, T \in \mathcal{T} \big\} \\ SS^p(\mathcal{T}) &:= \mathcal{P}^p(\mathcal{T}) \cap H^1_0(\mathcal{T}). \end{aligned}$$

for a polynomial degree $p \in \mathbb{N}_0$. This allows us to write down the discrete form of the equation: Find $U_\mathcal{T} \in SS^p(\mathcal{T})$ such that

$$a(U_\mathcal{T}, V) = \int_D fV\,dx \quad \text{for all } V \in SS^p(\mathcal{T}). \tag{2}$$

Again, the Lax-Milgram lemma guarantees existence and uniqueness. For simplicity of presentation, we assume $f \in \mathcal{P}^{p-1}(\mathcal{T})$ to avoid having to deal with data oscillations.

The residual based error estimator for the given problem reads

$$\rho_T^2 := \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2 := \operatorname{diam}(T)^2 \|f - \mathcal{L}U_\mathcal{T}\|_{L^2(T)}^2 + \operatorname{diam}(T)\|[n \cdot A\nabla U_\mathcal{T}]\|_{L^2(\partial T \cap \Omega)}^2 \tag{3a}$$

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

$$\rho := \rho(\mathcal{T}, U_\mathcal{T}, f) := \sqrt{\sum_{T \in \mathcal{T}} \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2}. \tag{3b}$$

Obviously, the error estimator ρ_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 error estimator is reliable and efficient in the sense

$$C_{\text{rel}}^{-1}\|u - U_\mathcal{T}\|_{H^1(\Omega)} \le \rho(\mathcal{T}, U_\mathcal{T}, f) \le C_{\text{eff}}\|u - U_\mathcal{T}\|_{H^1(\Omega)},$$

for constants $C_{\text{rel}}, C_{\text{eff}} > 0$ which depend only on the shape regularity of $\mathcal{T}$ and on p.

2.1. **Optimal mesh refinement.** Given the error estimate, it makes sense to refine elements with large estimated error. There are many ways to do this, but to obtain some sort of optimality of the procedure, algorithms of the following general form are needed:

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

(1) *Compute $U_\ell := U_{\mathcal{T}_\ell}$ from (2).*
(2) *Compute error estimate ρ_T for all $T \in \mathcal{T}_\ell$.*
(3) *Find a set $\mathcal{M}_\ell \subseteq \mathcal{T}_\ell$ of minimal cardinality such that*

$$\sum_{T \in \mathcal{M}_\ell} \rho_T^2 \ge \theta \sum_{T \in \mathcal{T}_\ell} \rho_T^2. \tag{4}$$

(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 $U_\ell \in SS^p(\mathcal{T}_\ell)$.*

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

$$\sup_{N \in \mathbb{N}} \inf_{\substack{\mathcal{T} \in \mathbb{T} \\ \#\mathcal{T} - \#\mathcal{T}_0 \le N}} \rho(\mathcal{T}, U_\mathcal{T}, f) N^s < \infty. \tag{5a}$$

We call a mesh refinement algorithm optimal if it satisfies

$$\sup_{\ell \in \mathbb{N}} \rho(\mathcal{T}_\ell, U_\ell, f) N^s < \infty \tag{5b}$$

for the same rate s. The main theorem of interest for the present work is the following. Its proof can be found in [17, 7] and for $b = 0$ also in [10, 29].

Theorem 2. *Algorithm 1 applied to (1)–(2) with constant coefficients A, b, c and error estimator (3) is optimal in the sense of (5) provided that θ is sufficiently small.*

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 $\max_{j=0,\dots,d+1} s_j$. We do not specify biases explicitely as we can always assume an additional constant input state x_0.

2.3. **Fixed number of distinct weights.** 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 distinct 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 distinct weights. The distinction between number of distinct weights and number of total weights is important as in the constructions below, some networks grow logarithmically in the accuracy, however, they are just iterations of the same basic building block and hence the number of distinct weights stays constant.

2.4. **Definition of Basic Recurrent Neural Networks.** Basically, a RNN B is a deep neural network B with output size $s' \in \mathbb{N}$ and input size $s + s'$, $s \in \mathbb{N}$. 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 inpute state, i.e.,

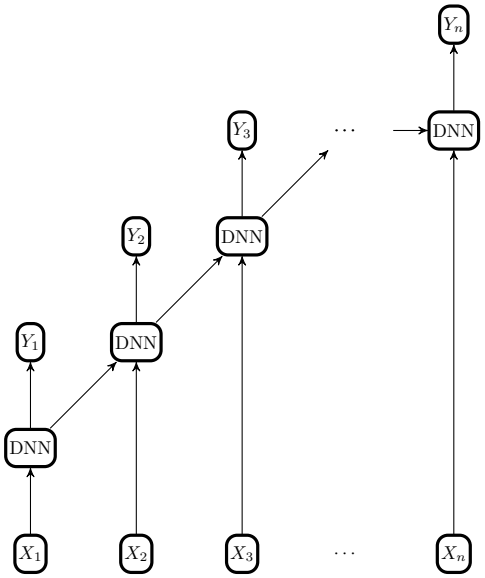
$$y_i := B(x_i, y_{i-1}) := W_d \phi(W_{d-1} \phi(W_{d-2}(\cdots \phi(W_1 \begin{pmatrix} x_i \\ y_{i-1} \end{pmatrix}) \cdots))).$$

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. For $j = 1$, we allways assume that $y_0 = 0 \in \mathbb{R}^{s'}$. For example, a simple summation over the seqeuence $\boldsymbol{x} \in \mathbb{R}^{n \times 1}$ 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 $\boldsymbol{y} \in \mathbb{R}^{n \times 1}$ contains the sum $y_n = \sum_{i=1}^{n} x_i$.

2.5. **RNNs as DNNs.** In the following, we will sometimes interpret a RNN B as a DNN B'. This means that we fix the input size n of B and consider the resulting neural network B' which has a n-times the width and depth of B with a total number of weights of n^2 as can be seen from:



However, the number of distinct weights is determined only by the number of weights in B and hence independent of n.

2.6. **Definition of General 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 $B_2 \circ B_1$ of two basic RNNs B_1, B_2 can be written as

$$\boldsymbol{x} \in \mathbb{R}^{s_1 \times n} \mapsto \boldsymbol{y} = B_1(\boldsymbol{x}) \in \mathbb{R}^{s_1' \times n} \quad \text{and} \quad \boldsymbol{x}' = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \\ y_n & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{2s_1' \times n} \mapsto \boldsymbol{y}' := B_2(\boldsymbol{x}') \in \mathbb{R}^{s_2' \times n}.$$

This choice might seem arbitrary, however, it gives us much more freedom when constructing the neural networks and does not sacrifice the simplicity of the function class. This means that the complexity of a stacked RNN is still independent of the sequence length n. For training on arbitrary sequence lengths $n \in \mathbb{N}$, this means that a bounded number of parameters (independent of n) has to be optimized, only.

To simplify notation, we will often write $B(\boldsymbol{x}) = B(\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^s)$ when dealing with vector valued input sequences $\boldsymbol{x} \in \mathbb{R}^{s \times n}$. Moreover, we will not explicitely write down the weight matrices W_j whenever their construction is clear from the formulas (i.e., concatenations of addition, substraction, and ϕ).

2.7. **Main Result.** The main goal of this work is to show that RNNs of almost constant size are capable of performing optimal mesh refinement for the PDEs as given in (1). To that end, we construct a RNN which performs steps (2)–(3) of Algorithm 1. The idea is that those steps are the only parts of the algorithm not determined by the problem setting, i.e., we want a finite-element approximation of the exact solution (determines step (1)) and we want to use a given mesh refinement routine (determines step (4)).

Theorem 3. *For a given accuracy $\varepsilon > 0$ and $\alpha \in \mathbb{N}$, there exists a RNN* ADAPTIVE *which takes a vector-valued input sequence $\boldsymbol{x} \in \mathbb{R}^{(d+1)(2(p+1)+d+1)\times \#\mathcal{T}}$ 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 $f|_{T'}$. Assuming a shuffled input sequence $\boldsymbol{x}' = (x_{\pi(1)}, \ldots, x_{\pi(\#\mathcal{T})})$ for some randomly chosen permutation π, the output $\boldsymbol{y} :=$ ADAPTIVE$(\boldsymbol{x}') \in \mathbb{R}^{\#\mathcal{T}}$ satisfies*

$$\sum_{\substack{T_i \in \mathcal{T} \\ y_{\pi(i)} > 0}} \widetilde{\rho}_{T_i}^2 \geq \theta \sum_{T \in \mathcal{T}} \widetilde{\rho}_T^2 \tag{6}$$

with probability $1 - C\exp(-\alpha)$ for some independent constant $C > 0$ and some estimators $\widetilde{\rho}_T$ which satisfy

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

Moreover, the number of positive entries in $\boldsymbol{y}$ is minimal in order to satisfy (6)*. The RNN has a fixed number of distinct weights. The RNN can be constructed with a total number of weights $N \in \mathbb{N}$ as long as $N \geq C(\alpha \log(\#\mathcal{T}) + |\log(\varepsilon)| + |\log(\|x\|_\infty)| + |\log(\theta)|)^2$ (see Figure 2 for the precise structure). The basic RNNs contained in* ADAPTIVE *have vector valued input and output sequences of dimension $\#\mathcal{T} \times 4$ at most. The magnitude of the weights is bounded by $\mathcal{O}(1)$.*

We refer to Section 3.4 for the proof of the Theorem. This result suggests the following algorithm:

Algorithm 4. ***Input:*** *Initial mesh $\mathcal{T}_0$.*
For $\ell = 0, 1, 2, \ldots$ do:

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

Output: *Sequence of adaptively refined meshes $\mathcal{T}_\ell$ and corresponding approximations $U_\ell \in SS^p(\mathcal{T}_\ell)$.*

From the previous theorem, we derive the following consequence.

Corollary 5. *Under the assumptions of Theorem 3 and with probability* $(1-C\exp(-\alpha))^{C|\log(\varepsilon)|}$ *Algorithm 4 is optimal up to accuracy* $2\sqrt{C\varepsilon/\theta} > 0$ *in the sense*

$$\sup_{\substack{\ell\in\mathbb{N}\\ \rho(\mathcal{T}_\ell,U_\ell,f)\geq 2\sqrt{C\varepsilon/\theta}}} \rho(\mathcal{T}_\ell,U_\ell,f)N^s \leq C$$

for a constant $C > 0$ *which depends only on* $\mathcal{L}$, Ω, *and* $\mathcal{T}_0$ *and the maximal rate* $s > 0$ *from* (5).

Proof. Theorem 3 and $\rho_\ell := \rho(\mathcal{T}_\ell, U_\ell, f) \geq 2\sqrt{C\varepsilon/\theta}$ imply

$$\sum_{\substack{T_i\in\mathcal{T}_\ell\\ y_{\pi(i)}>0}} \rho_{T_i}^2 \geq \frac{\theta}{2}\sum_{T\in\mathcal{T}_\ell}\rho_T^2.$$

Hence, [7, Proposition 4.10] implies linear convergence

$$\rho_{\ell+k}^2 \leq Cq^k\rho_\ell^2 \quad \text{for all } k,\ell\in\mathbb{N},$$

with constants $C > 0$ and $0 < q < 1$ depending only on $\mathcal{L}$, Ω, and $\mathcal{T}_0$. This shows that there occur at most $\mathcal{O}(|\log(\varepsilon)|)$-steps of Algorithm 4 before the tolerance ε is reached. In combination with Theorem 3, this leads to the probability estimate in the statement.

Suppose the set $\mathcal{M}\subseteq\mathcal{T}_\ell$ satisfies

$$\sum_{T\in\mathcal{M}}\rho_T^2 \geq 2\theta\sum_{T\in\mathcal{T}_\ell}\rho_T^2,$$

and $\rho_\ell \geq 2\sqrt{C\varepsilon/\theta} \geq \sqrt{2C\varepsilon/\theta}$. Then, using the same argument as above, it also satisfies

$$\sum_{T\in\mathcal{M}}\widetilde{\rho}_T^2 \geq \theta\sum_{T\in\mathcal{T}_\ell}\widetilde{\rho}_T^2.$$

and hence, by the minimality condition on y in Theorem 3, there holds $\#\mathcal{M} \geq \#\{i\in\mathbb{N} : y_i \neq 0\}$. These two properties enable us to apply the optimality proof of [7, Proposition 4.15] almost verbatim and hence conclude the proof. □

3. Construction of the Neural Networks

This section is dedicated to the construction of the basic building blocks of the RNN.

3.1. **Basic logic & algebra.** For the implementation of the RNNs below, we require a rudimentary IF operation. In the following, we use the notation $\mathbb{R}_{\delta+} := \{0\}\cup\{x\in\mathbb{R} : x\geq\delta\}$ as well as $\mathbb{R}_\delta := \{0\}\cup\mathbb{R}\setminus(-\delta,\delta)$. Moreover, we are going to directly exploit the round off errors of floating point arithmetic. For simplicity, we restrict to `double` arithmetic, however, it would be possible to transfer the proofs to any floating point system. To that end, we restrict the possible tolerances to $2^{\mathcal{E}}$ with $\mathcal{E}\subset-\mathbb{N}$. For `double` arithmetic, this set is defined by $\mathcal{E} := \{-1023+53,\dots,0\}$.

Lemma 6. *There exists a fixed size RNN* IF *such that any input* $\boldsymbol{x} = (x,0,\dots,0)\in\mathbb{R}_\delta^n$ *results in an output* $\boldsymbol{y} := \mathrm{IF}(\boldsymbol{x})\in\mathbb{R}_+^n$ *with*

$$y_n = \begin{cases} 1 & x\geq 2^{-n},\\ 0 & x\leq 0.\end{cases}$$

Proof. The RNN can be defined by

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

where $\min(x,1) = -\max(-x+1,0)+1$. Since $x_i = 0$ for all $i\geq 1$ and $y_0 = 0$, we have $y_i = \min(2^i\max(x,0),1)$. This concludes the proof. □

Remark 7. *The goal of Lemma 6 is to emulate a non-continuous function with a (inherently continuous) RNN. To that end, we may actually use the round-off error of* `double` *variables to our advantage. It is well-known that the precision of* `double` *numbers $x \approx 1$ is limited by 2^{-52}. Hence, for $x < 2^{-52}$ the (inexact) calculation $\widetilde{x} = (x +_{\text{double}} 1) - 1$ will yield $\widetilde{x} \leq 0$. This implies that the application of* IF *in the following way*

$$\boldsymbol{y} = \mathrm{IF}((\boldsymbol{x} +_{\text{double}} 1) - 1)$$

with an input size of $n = 52$ satisfies

$$y_n = \begin{cases} 1 & x \geq 2^{-52} \\ 0 & x < 2^{-52}. \end{cases}$$

Lemma 8. *Assuming* `double` *arithmetic and a given precission $\delta \in 2^{\mathcal{E}}$, there exists a DNN $\widehat{\mathrm{IF}}$ such that any input $x \in \mathbb{R}_\delta$ and $x' \in [0, M]$ results in an output $y := \widehat{\mathrm{IF}}(x, x')$ with*

$$y = \begin{cases} x' & x > 0, \\ 0 & x \leq 0. \end{cases}$$

The total number of weights of the DNN is bounded by $(\log_2(M) + |\log_2(\delta)|)^2$ and the number of distinct weights is independent of M.

Proof. We construct $\widehat{\mathrm{IF}}$ as a RNN by using

$$y_i = \mathrm{IF}(x_i, x') := \min(2\max(y_{i-1} + x_i, 0), x'),$$

with input size $n \in \mathbb{N}$ and interpreting this as a DNN. We define $\widehat{\mathrm{IF}}(x, x') := \mathrm{IF}((x +_{\text{double}} +\delta 2^{52}) - \delta 2^{52}, x')$. As in the discussion of Remark 7, we see that the finite precission of `double` implies $(x +_{\text{double}} \delta 2^{52}) - \delta 2^{52} \leq 0$ for all $x < \delta$. For $x \geq \delta$, we have $x +_{\text{double}} \delta 2^{52} \geq \delta + \delta 2^{52}$ and hence $\widehat{\mathrm{IF}}$ has the expected behavior as long as $n \geq 52 + |\log_2(\delta)| + \log_2(M)$. This concludes the proof.

□

Lemma 9. *Assuming* `double` *arithmetic and a given precission $\delta \in 2^{\mathcal{E}}$, there exists a DNN $\widetilde{\mathrm{IF}}$ such that any input $x \in \mathbb{R}_\delta$ and $x' \in [0, M]$ results in an output $y := \widetilde{\mathrm{IF}}(x, x')$ with*

$$y = \begin{cases} x' & x \geq 0, \\ 0 & x < 0. \end{cases}$$

The total number of weights of the DNN is bounded by $(\log_2(M) + |\log_2(\delta)|)^2$ and the number of distinct weights is independent of M.

Proof. The construction is given by $\widetilde{\mathrm{IF}}(x, x') = x' - \widehat{\mathrm{IF}}(-x, x')$. □

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 precission first appeared in [31]. They showed that a DNN of size proportional to $|\log(\varepsilon)|$ achieves this up to some tolerance ε. 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 [31], however, the number of weights which need to be trained is fixed and independent of ε.

Theorem 10. *For every $n \in \mathbb{N}$, there exists a RNN* SQUARE *with a fixed number of weights such that the output $\boldsymbol{y} = \mathrm{SQUARE}(\boldsymbol{x})$ for an input vector $\boldsymbol{x} = (x_0, 0, \ldots, 0) \in [0, 1]^n$ satisfies*

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

Proof. We reuse the saw-tooth function from [31]

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

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

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

A second 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 RNN B' computes

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

Initialized with the last entry of 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 $x - x^2$.

The error estimate follows from the fact that the approximation to x^2 is actually the linear interpolation f_n of $f(x) = x^2$ at 4^n equidistant points in $[0,1]$ (see [31]). This shows

$$|f_n(x) - f(x)| = |\int_z^{x_0} f'(s)\,ds \le 4^{-n} \max_{0 \le x \le x_0} f'(x) = 2 \cdot 4^{-n} x_0,$$

where we used z as the largest interpolation point smaller than x_0 as well as $|z - x_0| \le 4^{-n}$. □

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

Proposition 11. *There exists a RNN* MULTIPLY *such that for all $x, y \in [-2^{n/2}, 2^{n/2}]$ 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*

$$|z_n - xy| \le C 2^{-n/2},$$

where $C > 0$ is independent of n and x, y.

Proof. As mentioned above, we construct MULTIPY from SQUARE with inputs in $[0,1]$. Thus, we first assume that $x, y \ge 0$ and scale the input to the appropriate size. Since we do not want to magnify the rounding errors produced in SQUARE, it is essential that the input does not get scaled $\ll 1$.

Hence, we first use a RNN B constructed from the if-clauses from Lemma 8–9 with tolerance $\delta = 2^{-52}$

$$(x_i', r_i) := B(x_i) = \begin{pmatrix} \widetilde{\mathrm{IF}}(\max(x_{i-1}, y_{i-1}) - 1/4, x_{i-1}'/2) + \widehat{\mathrm{IF}}(1/4 - \max(x_{i-1}, y_{i-1}), x_{i-1}') \\ \widetilde{\mathrm{IF}}(\max(x_{i-1}, y_{i-1}) - 1/4, 1) \end{pmatrix}$$

to reduce the input magnitude of x to a value below $1/4 + 2^{-52}$ (as we did not restrict the minimal positive value of the input, see Remark 7). Since we have the reverse the scaling in the end, we need to log the number of times B divides by 2 in the vector $\boldsymbol{r}$. We apply B also to $\boldsymbol{y}$ to obtain the sequence $(y_i', s_i) = B(y_i)$.

Then, we initialize SQUARE with the last entries of the output sequences of B, to calculate $(x2^{-d_x} + y2^{-d_y})^2$ as well as $(x2^{-d_x})^2$ and $(y2^{-d_y})^2$, with $d_x = \sum \boldsymbol{r}$ and $d_y = \sum \boldsymbol{s}$. Some linear transformations yield the approximation to $xy2^{-d_x - d_y}$. With B' defined by

$$x_i' = B'(x_i) := \widehat{\mathrm{IF}}(r_i, 2x_{i-1}') + \widehat{\mathrm{IF}}(s_i, 2x_{i-1}'), \tag{7}$$

we reverse the scaling and return the correct approximation.

The error estimate follows by noticing that

$$|2^{2n}\mathrm{SQUARE}(x2^{-n}) - x^2| = 2^{2n}|\mathrm{SQUARE}(x2^{-n}) - (x2^{-n})^2| \le 2^{2n} 4^{-n} x 2^{-n} = 2^{-n/2}.$$

This concludes the proof for $x, y \geq 0$. For general input x, y, we use the formula

$$xy = (x_+ - x_-)(y_+ - y_-) = x_+y_+ - x_+y_- - x_-y_+ + x_-y_-$$

for $x_+ = \max(x, 0)$ and $x_- = \max(-x, 0)$. □

Remark 12. *It might seem odd that we actually use the magnification of the round-off error in the construction of the if-clauses in Lemma 8–9 but ignore it in Lemma 11. The reason we chose to do this is that in* MULTIPLY*, the magnification of the round-off error only happens in* (7) *if the magnitude of the input is large. This means that any rounding error $\varepsilon_{\text{square}}$ coming from the application of* SQUARE *gets magnified up to $\simeq \varepsilon_{\text{square}}|1 + x||1 + y|$. This kind of error, however, has to be expected of any arithmetic operation in floating point arithmetic with large inputs and is not a particular flaw of the presented multiplication algorithm (see $\widehat{\text{IF}}$ for the contrary case, where small inputs lead to a large, intended round-off error). Hence, to simplify presentation, we believe it is justified to take the usual approach in the numerical analysis of high-level algorithms and to ignore the round-off error as long as scales relatively with the input-output size. The numerical examples in Section 5 underline this argument.*

3.2. **Error estimation.** For brevity of presentation, we restrict ourselves to the case $A = 1$ and $b = c = 0$ of (1). The general case can easily be implemented along the lines of this section. In the present case, the residual error estimator given in (3) 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 ρ_T, i.e.,

$$\rho_T(\mathcal{T}, U_\mathcal{T}, f)^2 \simeq \operatorname{diam}_\infty(T)^{2+d}|T|^{-1}\|f + \Delta U_\mathcal{T}\|^2_{L^2(T)} + \operatorname{diam}_\infty(T)^d|\partial T|^{-1}\|[\nabla U_\mathcal{T}]\|^2_{L^2(\partial T\cap\Omega)}, \tag{8}$$

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 $[\partial_{t_i} U] = 0$ on any interface for $U \in SS^p(\mathcal{T})$, there holds

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

Note that it would certainly be possible to emulate the exact error estimator $\rho(\cdot)$, however, as shown in [7], 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 13. *There is a fixed size DNN* DIAM *which, given the nodes of an element $T = \operatorname{conv}(z_0, \dots, z_d)$ computes the ∞-diameter $\operatorname{diam}_\infty(T) := \max_{x,y\in T}|x - y|_\infty$ of T.*

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

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

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

$$\left|\operatorname{diam}_\infty(T)^d|T|^{-1}\|f + \Delta U\|^2_{L^2(T)} - y_n\right| \leq C2^{-n/2}$$

for $\boldsymbol{y} = \text{VOL}(\boldsymbol{x})$ as long as the coefficients of the polynomial expansion of $(f + \Delta U)^2$ are contained in $[-2^{n/2}, 2^{n/2}]$.

Proof. We compute ΔU as a linear function of the polynomial coefficients. Then, we stack p^2 RNNs MULTIPLY from Proposition 11 to compute the coefficients of $(f + \Delta U)^2$ up to accuracy $\lesssim 2^{-n/2}$. To compute the integral of the L^2-norm and hence conclude the construction, we note that

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

for the reference element $\widehat{T}$ and $F_T(x) = (z_1 - z_0, \dots, z_d - z_0)x + z_0$ can be computed via linear operations on the nodes of T. Another application of $\mathcal{O}(p^2)$ instances of MULTIPY computes the integral with accuracy $\lesssim 2^{-n/2}$. This concludes the proof. □

Lemma 15. *Let $U, f \in \mathcal{P}^p(T)$ for a given element $T \in \mathcal{T}$. There is an RNN* JUMP *which, given the nodes of the element $T' = \operatorname{conv}(z_0, \dots, z_d)$ as well as the polynomial coefficients of $U|_{T'}$ for all elements $T' \in \omega_T$ as a vector valued sequence $\boldsymbol{x} = (x, 0, \dots, 0) \in \mathbb{R}^{(d+1)(2(p+1)+d+1)\times n}$, satisfies*

$$\big|\operatorname{diam}_\infty(T)^{d-1} |\partial T|^{-1} \|[\nabla U]\|^2_{L^2(\partial T)} - y_n\big| \le C 2^{-n/2}$$

for $\boldsymbol{y} = \mathrm{JUMP}(\boldsymbol{x})$ as long as the coefficients of the polynomial expansion of $[\nabla U]$ are contained in $[-2^{n/2}, 2^{n/2}]$.

Proof. The proof works analogously to that of Lemma 14, with the difference that we have to include the data on the patch of T to compute $[\nabla U]$. □

Theorem 16. *There exists an RNN* ESTIMATOR *which takes a vector-valued input sequence $\boldsymbol{x} \in \mathbb{R}^{(d+1)(2(p+1)+d+1)\times \#\mathcal{T}}$ 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 $f|_{T'}$. The output $\boldsymbol{y} := \mathrm{ESTIMATOR}(\boldsymbol{x})$ satisfies*

$$|y_i - \rho_{T_i}(\mathcal{T}, U, f)^2| \le C 2^{-n/2}$$

in case $n \gtrsim \max(\log(\boldsymbol{x}))$ for a uniform hidden constant. The RNN ESTIMATOR *has a fixed number of distinct weights but width and depth proportional to n.*

Proof. Lemmas 13–15 show that there are RNNs computing all ingredients for $\rho_T(\mathcal{T}, u, f)^2$. Two applications of the RNN MULTIPLY combine the elements and output an approximation to $\rho_T(\mathcal{T}, u, f)^2$ up to an accuracy $\lesssim 2^{-n/2}$ as long as the magnitude of the input is bounded by $2^{n/2}$ (where $n \in \mathbb{N}$ is the size of the input sequence). 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 distinct 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 $x \in \mathbb{R}^{\#\mathcal{T}\times(d+1)(2(p+1)+d+1)}$ as described in the statement. From this, EST computes the output sequence y_i which satisfies

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

This concludes the proof. □

3.3. **Dörfler marking.** The backbone of all rate-optimal adaptive algorithms is the Dörfler marking criterion (4). The aim of this section is to construct a RNN which emulates this marking criterion.

Lemma 17. *Assuming* `double` *arithmetic and a given precissions $\delta \in 2^{\mathcal{E}+52}$, there exists a RNN* CHOOSE *such that every input $\boldsymbol{x} \in \mathbb{R}^n_{\delta+} \cap [0, M]^n$ and $z \in \mathbb{R}_\delta$, $n \in \mathbb{N}$ results in an output $\boldsymbol{y} = \mathrm{CHOOSE}(\boldsymbol{x}, z) \in \mathbb{R}^n_{\delta+}$ with*

$$y_i = \begin{cases} x_i & z \ge 0, \\ 0 & z < 0. \end{cases}$$

The RNN has a fixed number of distinct weights and the number of weights is $\mathcal{O}(|\log(M)| + |\log(\delta)|)$.

Proof. We use the DNN $\widetilde{\mathrm{IF}}$ from Lemma 9 with precision $\delta 2^{-52}$ and construct

$$F(x, y) := \max(x - y, 0) + \widetilde{\mathrm{IF}}(x - y, y) = \begin{cases} x & x \ge y, \\ 0 & x < y. \end{cases} \tag{9}$$

This works since in `double` arithmetic, $x, y \in \mathbb{R}_{\delta+}$ implies that $x - y = 0$ or $|x - y| \geq \delta 2^{-52}$. This follows from the fact that a `double` variable $x \geq \delta$ actually saves a number $m_x \in [1, 2 - 2^{-52}]$ with precision of 2^{-52} and scales it using a separately stored exponent $e_x \in [-1023, 1022] \cap \mathbb{Z}$. This implies that `double` subtraction of two numbers x, y can not be more accurate than $2^{\min(e_x, e_y)} 2^{-52}$.

With this building block, we construct CHOOSE by

$$y_i := F(x_i + z, x_i) - \max(z, 0).$$

This concludes the proof. □

Lemma 18. *Assuming `double` arithmetic and a given precissions $\delta \in 2^{\mathcal{E}+52}$, there exists a RNN* $\widetilde{\text{CHOOSE}}$ *such that every input $\boldsymbol{x} \in \mathbb{R}^n_{\delta+} \cap [0, M]^n$ and $z \in \mathbb{R}_\delta$, $n \in \mathbb{N}$ results in an output $\boldsymbol{y} = \widetilde{\text{CHOOSE}}(\boldsymbol{x}, z) \in \mathbb{R}^n_{\delta+}$ with*

$$y_i = \begin{cases} x_i & z > 0, \\ 0 & z \leq 0. \end{cases}$$

The RNN has a fixed number of distinct weights and the number of weights is $\mathcal{O}(|\log(M)| + |\log(\delta)|)$.

Proof. The proof works analogously to that of Lemma 17. □

Lemma 19. *Assuming `double` arithmetic and a given precission $\delta \in 2^{\mathcal{E}}$, there exists a RNN* NONZERO *such that for each input $\boldsymbol{x}, \boldsymbol{x}' \in (\mathbb{R}_{\delta+} \cap [0, M])^n$ the output of $[\boldsymbol{y}, \boldsymbol{y}'] = \text{NONZERO}(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^n_{\delta+}$ satisfies $y_j = x_i$, $j = i, \ldots, n$ for the first element x_i of $\boldsymbol{x}$ such that $x_i \neq 0$ and $x'_i = 0$. Moreover, there holds $y_j = 0$ for all $j < i$. The RNN has a fixed number of weights and fixed depth but width $\mathcal{O}(|\log(M)| + |\log(\delta)|)$.*

Proof. We reuse the function F from (9) and construct the net with the block

$$y_i := B(x_i, x'_i, y_{i-1}) := y_{i-1} + F(x_i - y_{i-1}, x_i) - \widehat{\text{IF}}(F(x'_i - y_{i-1}, x'_i), y_{i-1} + F(x_i - y_{i-1}, x_i))$$

The net NONZERO initializes $B(x_1, 0)$ and the applies $B(\cdot)$ to the whole sequence of $\boldsymbol{x}$, $\boldsymbol{x}'$, and $\boldsymbol{y}$. Let x_i denote the first non-zero element and assume $x'_i = 0$. This implies that $x_j = 0 = y_j$ for all $1 \leq j \leq i$. Then, $B(x_i, y_{i-1}) = B(x_i, 0) = x_i$. We prove by induction that $B(x_j, y_{j-1}) = x_i$ for all $j \geq i$. The case $j = i$ is done already. Assume the statement is true for some $j > i$. Then, since $y_j \neq 0$, we have

$$B(x_{j+1}, x'_{j+1}, y_j) = y_j = x_i.$$

This concludes the proof if $x'_i = 0$. In case $x'_i \neq 0$, we have $y_i = 0$ since $F(x'_i - y_{i-1}, x'_i) = F(x'_i - 0, x'_i) = x'_i$. Thus, the situation is as if $x_i = 0$ the net net will search for the next non-zero entry in x. This proof also shows that the input to F, namely $x_i - y_{i-1}$ is either x_i or zero and hence in $\mathbb{R}_\delta$. The input $x'_i - y_{i-1}$ is in $\mathbb{R}_\delta$ since x'_i is either zero or one. □

Lemma 20. *There exists a fixed size RNN* SHIFT *such that any input $\boldsymbol{x} \in \mathbb{R}^n$ results in an output $\boldsymbol{y} =$ SHIFT$(\boldsymbol{x}) \in \mathbb{R}^n$ with*

$$y_1 = 0, \quad \text{and} \quad y_i = x_{i-1} \quad \text{for all } i = 2, \ldots, n.$$

Proof. The RNN uses an internal state which can be formalized by a vector-valued output $\boldsymbol{y} \in \mathbb{R}^{n \times 2}$ of which, however, only $y_{i,1}$ is of interest for the user. The construction

$$(y_{i,1}, y_{i,2}) = B(x_i, y_{i-1,2}) = (y_{i-1,2}, x_i)$$

satisfies $y_{i,2} = x_i$ and $y_{i,1} = x_{i-1}$ for $i \geq 2$. This concludes the proof. □

Theorem 21. *We assume `double` arithmetic. For all $C, \delta > 0$ and $0 < \theta \leq 1$, there exists a RNN* MARK *such that the following holds: Assume an input $\boldsymbol{x} \in \mathbb{R}^n_{\delta+}$ which is randomly permuted and satisfies $x_i \neq x_j$ for all $1 \leq i \neq j \leq n$ with $x_i \neq 0$ and $x_j \neq 0$, $\boldsymbol{y} :=$ MARK$(\boldsymbol{x})$ satisfies with probability $1 - C\exp(-\alpha)$ that*

$$\sum_{\substack{i=1 \\ y_i > 0}}^{n} x_i \geq \theta \sum_{i=1}^{n} x_i \tag{10}$$

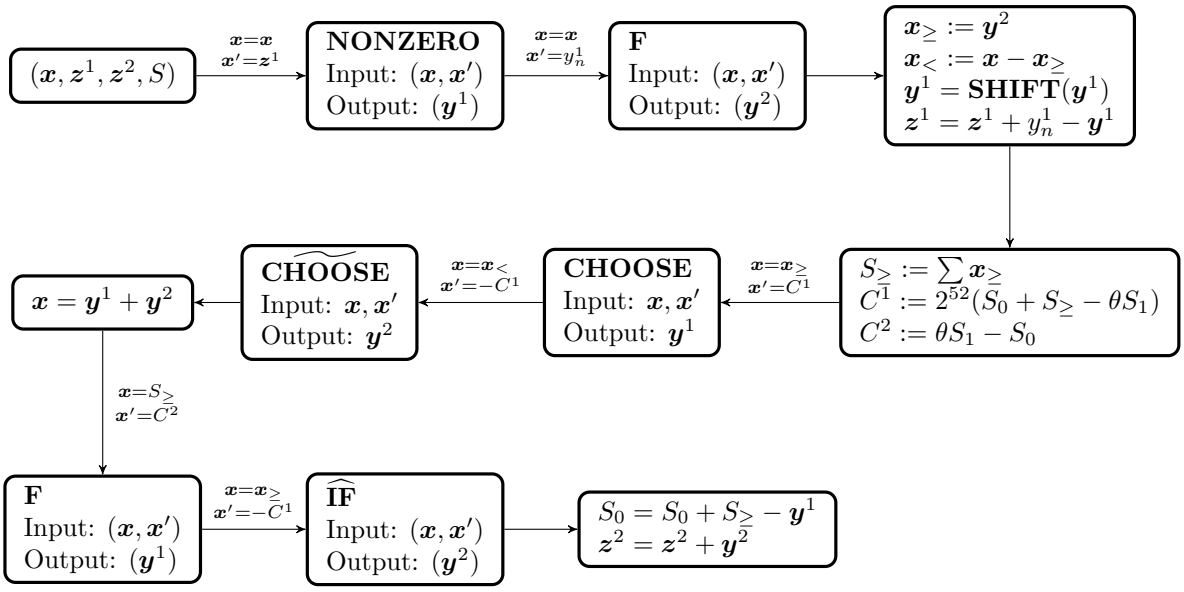


FIGURE 1. Structure of the RNN B as defined in the proof of Theorem 21. Each block represents a basic RNN and the annotations over the arrows indicate the initialization of the input vectors (whenever a sequence is initialized with a scalar, e.g., $\boldsymbol{x} = y_n$, this means $\boldsymbol{x} = (y_n, 0, \ldots, 0)$). Output variables which are not mentioned are simply passed through to the following block.

with a minimal number of terms in the left-hand side sum. The constant $C > 0$ is independent of $\boldsymbol{x}$, α, θ, and n. The network has a fixed number of distinct weights, but its total number of weights depends on $\mathcal{O}(\alpha \log(n) + (|\log(\delta)| + |\log(\theta)|)^2)$

Proof. We aim to recreate the quick-mark algorithm proposed in [26]. In the case where $\boldsymbol{x}$ has no two equal entries, the algorithm works as follows: Given a pivot element x_1, the vector $\boldsymbol{x}$ is partitioned into smaller elements $\boldsymbol{x}_<$ and larger-or-equal elements $\boldsymbol{x}_\geq$. If the elements in $\boldsymbol{x}_\geq$ are sufficient to satisfy the marking criterion, $\boldsymbol{x}_<$ is dropped and the procedure is repeated on $\boldsymbol{x} = \boldsymbol{x}_\geq$. In the other case, we know that all elements of $\boldsymbol{x}_\geq$ have to be marked and we may drop $\boldsymbol{x}_\geq$ to repeat the procedure on $\boldsymbol{x}_<$.

With the DNN F from (9), an RNN to emulate the quick-mark algorithm can be constructed as follows: Instead of dropping the entries of $\boldsymbol{x}$, we set them to zero in this construction. We define the basic building block B in Figure 1. The input sequences are $\boldsymbol{x}$ (error estimates), $\boldsymbol{z}^1$ (keeps track of previously used pivots), $\boldsymbol{z}^2$ (keeps track of marked elements), and $S = (S_0, S_1)$ (S_1 contains the overall sum of $\boldsymbol{x}$ before the algorithm is started while S_0 contains the sum of already marked elements). The pivot element y_n^1 is just the first non-zero element of the vector $\boldsymbol{x}$ which has not already been used as pivot. To see this, assume that x_i is that element. Then, after the application of $\boldsymbol{y}^1 = \text{NONZERO}(\boldsymbol{x}, \boldsymbol{z}^1)$, $y_j^1 = x_i$ for all $j \geq i$ and after the application of SHIFT, we have $y_j^1 = x_i$ for all $j \geq i+1$. Hence $\boldsymbol{z}^1 = \boldsymbol{z}^1 + y_n^1 - \boldsymbol{y}^1$ is equal to $x_i \neq 0$ up to the i-th entry. By the definition of NONZERO, this prevents x_i to be picked again as pivot in the subsequent steps.

We show that the block B performs one iteration of the quick-mark algorithm: First, we modify the accuracy δ to be smaller than $\delta\theta$. This will ensure that all the quantities in the construction above are in $\mathbb{R}_{\delta+}$. To see this, note that all building blocks NONZERO, F, SHIFT, CHOOSE, F, and $\widetilde{\text{IF}}$ produce outputs in $\mathbb{R}_\delta$ (as shown in Lemma 17–Lemma 20). Apart from these blocks, there occur only four subtractions in B which could lead to results not in $\mathbb{R}_{\delta+}$. Those are, cf. Figure 1,

(1) $\boldsymbol{z}^1 = \boldsymbol{z}^1 + y_n^1 - \boldsymbol{y}^1$: As shown above, y_j^1 is either zero or equal to x_i for all $j \geq i$. Hence, $y_n^1 - \boldsymbol{y}^1$ is either zero or equal to x_i and thus is contained in $\mathbb{R}_\delta$.

(2) $C^1 = 2^{52}(S_0 + S_{\geq} - \theta S_1)$: As explained in the proof of Lemma 17, the result of $S_0 + S_{\geq} - \theta S_1$ is in $\mathbb{R}_{\delta 2^{-52}}$. Hence, the scaling ensures $C^1 \in \mathbb{R}^\delta$ (note that only the sign of C^1 counts for the computation).
(3) $C^2 = \theta S_1 - S_0$: By definition in the proof of Lemma 17, $F(x, y)$ performs correctly as long as the input satisfies $x - y \in \mathbb{R}_{\delta 2^{-52}}$. Here, we have $x - y = S_{\geq} - C^2 \in \mathbb{R}_{\delta 2^{-52}}$, since $S_{\geq} \in \mathbb{R}_\delta$.
(4) $S_0 = S_0 + S_{\geq} - \boldsymbol{y}^1$: By definition, y^1 has either the value zero or equals $S_{\geq}$. Hence S_0 remains unchanged or is updated to $S_0 + S_{\geq} \in \mathbb{R}_\delta$.

This shows that the output of B is again contained in $\mathbb{R}_\delta$ in all cases and the computations are performed correctly.

Given a sequence $\boldsymbol{x} \in \mathbb{R}^n_{\delta+}$ which satisfies the assumptions in the statement, Lemma 9 shows that the application of F produces vectors $\boldsymbol{x}_{\geq}, \boldsymbol{x}_{<} \in \mathbb{R}^n$ such that for $\circ \in \{\geq, <\}$

$$(x_\circ)_i = \begin{cases} x_i & x_i \circ x_1, \\ 0 & \text{else.} \end{cases}$$

Next, we note that $S = (S_0, S_1)$ is such that S_1 will be initialized with $S_1 = \sum \boldsymbol{x}$ and will not change during the application of B. S_0 on the other hand contains the current sum of all marked elements of $\boldsymbol{x}$.

Case 1: Consider the case $S_0 + \sum \boldsymbol{x}_{\geq} - \theta S_1 \geq 0$. This is the case if $\boldsymbol{x}_{\geq}$ already contains enough elements to satisfy the marking criterion. Hence, we may drop $\boldsymbol{x}_{<}$ completely. To that end, Lemma 17 shows that the application of CHOOSE implies that $\boldsymbol{y}^1$ contains $\boldsymbol{x}_{\geq}$. In the following, S remains unchanged (since F returns $\sum \boldsymbol{x}_{\geq} + S_0$) and also $\boldsymbol{z}^2$ remains unchanged (since $\boldsymbol{y}^2 = 0$).

Case 2: In the case of $S_0 + \sum \boldsymbol{x}_{\geq} - \theta S_1 < 0$, all elements of $\boldsymbol{x}_{\geq}$ are needed to satisfy the marking criterion. Consequently, we want $\boldsymbol{x} = \boldsymbol{x}_{<}$, set the corresponding entries of $\boldsymbol{y}^2$ to some non-zero value and update S_0. Lemma 17 shows that the application of $\widetilde{\text{CHOOSE}}$ results in $\boldsymbol{y}^2 = \boldsymbol{x}_{<}$ and Lemma 6 shows that the next step updates $S_0 = S_0 + \sum \boldsymbol{x}_{\geq}$ as well as $\boldsymbol{z}^2 = \boldsymbol{z}^2 + \boldsymbol{x}_{\geq}$.

After sufficiently many steps, the vector $\boldsymbol{x}$ is either zero or it contains one non-zero element. In case no more elements are available as pivots, NONZERO will always pick the pivot element zero. Thus, any subsequent application of B will not change anything. The remaining element has to be added to the marked elements by a final application of $\boldsymbol{z}^2 = \boldsymbol{z}^2 + \boldsymbol{x}$ as well as an update $S_0 = S_0 + \sum \boldsymbol{x}$. Hence, S_0 contains the sum of all marked elements and $\boldsymbol{z}^2$ is non-zero only at the indices of marked elements.

Choosing the optimal pivot would guarantee that $\log_2(n)$ applications of B suffice to find the result. However, we essentially choose a random pivot by shuffling the input sequence. This means that we potentially need more iterations.

The pivot z_n of the reduced vector $\boldsymbol{x}$ is always the first non-zero element which has not been a pivot before. Let $z_n^1, z_n^2, \ldots, z_n^s$ denote the previously chosen pivots and $R := \{x_i : x_i \neq 0,\, x_i \neq z_n^j,\, j = 1, \ldots, s\}$ the set of possible pivots. Since the input is assumed to be randomly shuffled, the probability $\mathbb{P}(z_n = x_i | z_n^1, z_n^2, \ldots, z_n^s) = 1/\#R$ for all $x_i \in R$. This implies that the probability of x_i being in the $(\lambda, 1-\lambda)$-quantile of R is given by

$$\mathbb{P}\Big(\#\{z \in R : z > x_i\} \geq \lambda \#R \text{ and } \#\{z \in R : z < x_i\} \geq \lambda \#R\Big) \geq 1 - 2\lambda - \frac{1}{\#R}.$$

In this case, at least $\lceil \lambda \#R \rceil$ elements of $\boldsymbol{x}$ are set to zero during the application of B. The pivot is always either the smallest remaining non-zero element in $\boldsymbol{x}$ ($\boldsymbol{x} = \boldsymbol{x}_{\geq}$), or it has been set to zero already ($\boldsymbol{x} = \boldsymbol{x}_{<}$). Hence, there is at most one previously used pivot in the non-zero elements of x. This means that $|\#R - \#\{i \in \mathbb{N} : x_i \neq 0\}| \leq 1$.

We first assume $\#R \geq \lambda^{-1}$: To reduce the size of R to λ^{-1}, the pivot element must lie in the $(\lambda, 1-\lambda)$-quantile at least $k = \lceil \log_{1/(1-\lambda)}(n) \rceil$-times. The binomial distribution shows that K applications of B will achieve this with a probability of

$$P = \sum_{j=k}^{K} \binom{K}{j} (1 - 3\lambda)^j (3\lambda)^{K-j}.$$

A tail bound for the binomial distribution based on Hoeffding's inequality gives for $p = 1 - 3\lambda$

$$P \geq 1 - \exp(-2\frac{(Kp-k)^2}{K}).$$

Hence, for $K = \alpha/pk$ and with $k \geq 1$, we obtain

$$P \geq 1 - \exp(-2\frac{(\alpha-1)^2 kp}{\alpha}) \geq 1 - C\exp(-2\alpha p).$$

On the other hand, if $\#R < \lambda^{-1}$, we observe that at least every second application of B reduces the size of R by one. This follows from the fact that if the pivot happens to be the smallest element of R, it gets eliminated in the next step.

Altogether, we conclude the with probability $P \geq 1 - C\exp(-2\alpha/p)$, $K = 2 + 2\lambda^{-1} + \alpha/p\lceil \log_{1/(1-\lambda)}(n)\rceil$ applications of B will conclude the quick-mark algorithm and hence yield the correct result. The choice $\lambda = 1/6$ concludes the proof. □

Corollary 22. *We assume* `double` *arithmetic. For all $C, \delta > 0$ and $0 < \theta \leq 1$, there exists a RNN* MARK *such that the following holds: Assume an input $\boldsymbol{x} \in \mathbb{R}^n_{\delta+}$ which is randomly permuted, $\boldsymbol{y} :=$ MARK$(\boldsymbol{x})$ satisfies with probability $1 - C\exp(-\alpha)$ that*

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

with a minimal number of terms in the left-hand side sum. The constant $C > 0$ is independent of $\boldsymbol{x}$, α, θ, and n. The network has a fixed number of distinct weights, but its total number of weights depends on $\mathcal{O}(\alpha\log(n) + (|\log(\delta)| + |\log(\theta)|)^2)$

Proof. With some minor changes in the RNN B of Figure 1 we can drop the assumption $x_i \neq x_j$ for all $1 \leq i \neq j \leq n$ with $x_i \neq 0$ and $x_j \neq 0$ in Theorem 21: With similar constructions as in Lemma 17 and Figure 1 we can build $\boldsymbol{x}_>$ with elements larger than the pivot and set $\boldsymbol{x}_= := \boldsymbol{x}_{\geq} - \boldsymbol{x}_>$ and update $\boldsymbol{z}^1 = \boldsymbol{z}^1 + y^1_n - \boldsymbol{y}^1 + \boldsymbol{x}_=$. in the definition of B in Figure 1. This prevents other elements of $\boldsymbol{x}$ with same value to be chosen again as pivot and therefore the probability estimates in the proof of Theorem 21 remain valid.

After the application of MARK from Theorem 21 with the modifications described above, the output $\boldsymbol{z}^2$ contains a set of non-zero indices (with probability $1 - C\exp(-\alpha)$) that fulfills the optimality condition (10). However, it might fail the minimality condition in Theorem 21 up to a set of equal, non-zero entries of $\boldsymbol{x}$. Note that after all iterations of B from Figure 1, $\boldsymbol{x}$ contains only those entries (and zeros else).

We now sketch how to remove these unnecessary elements: We construct an RNN B' by

$$y_i = B'(x_i, z^2_i, y_{i-1,2}) = \begin{cases} (0, y_{i-1,2} - x_i) & \text{if } x_i > 0 \text{ and } y_{i-1,2} + x_i \geq \theta S_1, \\ (z^2_i, y_{i-1,2}) & \text{else,} \end{cases}$$

where the *and*-statement can be realized by two applications of IF from Lemma 8, i.e, $a > 0$ and $b > 0$ is represented as $\widehat{\mathrm{IF}}(b, \widehat{\mathrm{IF}}(a, \cdot))$. Initialized with $y_{-1,2} = S_0 + \sum \boldsymbol{x}$ with S_0, $\boldsymbol{x}$, and $\boldsymbol{z}^2$ being the output after the final iteration of B from the proof of Theorem 21, we remove the (equally) smallest elements of $\boldsymbol{x}$ one by one as long as the remaining marked elements still satisfy the marking criterion. This concludes the proof. □

3.4. **The complete mesh-refinement RNN.** The previous sections already give the necessary ingredients to build the RNN ADAPTIVE from Theorem 3. Figure 2 shows the construction of the RNN. The proof of Theorem 3 follows from the previous sections. We use the RNN ESTIMATOR with accuracy $n \simeq 2|\log(\varepsilon/N)|$ from Theorem 16 as the building block for ADAPTIVE as shown in Figure 2 which computes the error estimator $\widetilde{\rho}_T(\mathcal{T}, U_\mathcal{T}, f)$ such that

$$|\widetilde{\rho}_T(\mathcal{T}, U_\mathcal{T}, f)^2 - \rho_T(\mathcal{T}, U_\mathcal{T}, f)^2| \lesssim \varepsilon/\#\mathcal{T} \tag{11}$$

for all $T \in \mathcal{T}$ as long as $N \geq \#\mathcal{T}$. The application of $\widehat{\mathrm{IF}}$ with tolerance $\delta \leq \varepsilon/N$ ensures $\boldsymbol{x} \in \mathbb{R}^{\#\mathcal{T}}_{\delta+}$ and increases the error in (11) only by at most a multiple of δ. Corollary 22 with the lower tolerance $\delta \leq \varepsilon/N$ provides the block B and hence MARK. All the building blocks used in the construction consist of a fixed number of distinct weights but may have a total number of weights depending on $\log(N)$ and $\log(\varepsilon)$.

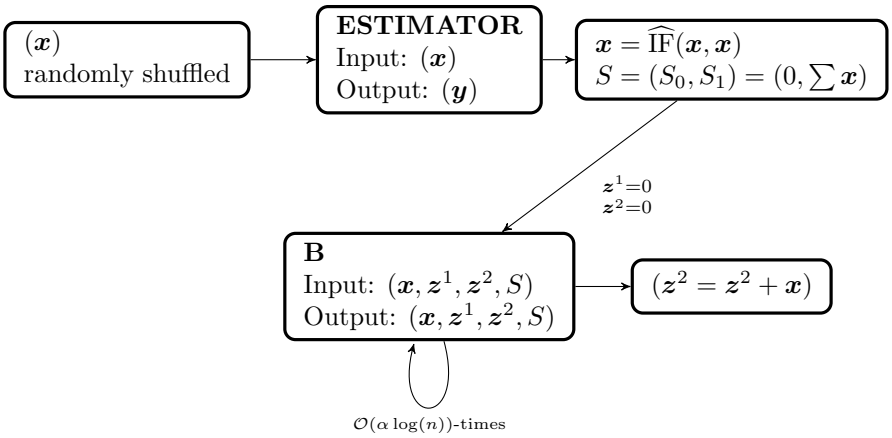


FIGURE 2. Structure of the RNN ADAPTIVE.

4. Conclusions

4.1. **Theoretical result.** Theorem 3 and Corollary 5 show that an RNN can in fact achieve optimal mesh refinement in the sense of (5). The RNN only needs to follow a fairly general structure in the sense that it consists of a fixed number of basic RNNs of which one block has to be repeated $\mathcal{O}(\log(\#\mathcal{T}))$ times. The width and depth of the basic RNNs depends logarithmically on $\#\mathcal{T}$ as well as on the desired accuracy. The number of distinct weights (trainable parameters) is, however, uniformly bounded and independent of the accuracy as well as of the number of elements in the mesh.

Despite the fact that for the given model problem (1), an optimal mesh refinement strategy is known (in terms of the given error estimator and Dörfler marking), we believe that this result encourages the developement of artificially intelligent mesh refinement algorithms based on RNNs which show superior performance on harder problems for which we currently do not know optimal refinement strategies. Such problems include non-linear PDEs, time dependent PDEs, and non-Galerkin based methods. Moreover, it would not be a problem to adapt the RNN to also yield *hp*-refinement information, such that the approach might also shed some light on optimal adaptive *hp*-FEM.

4.2. **Practical implementation.** One has to rely on standard optimization techniques to actually construct the RNN. The idea is to set up a generic RNN ADAPTIVE with sufficient complexity to accommodate the structure given in Theorem 3 and to apply an optimization algorithm which minimizes the error for the given problem

$$\|u - U_L\| \simeq \rho_L = \rho_L(\mathcal{T}_L, U_L, f, \Omega, \dots)$$

after a maximal number $L \in \mathbb{N}$ of steps such that $\#\mathcal{T}$ is smaller than a prescribed bound (here, u is the exact solution of the given problem). As training data can serve randomly generated domains Ω, right-hand sides f, as well as coefficients of the PDE. One could even think of randomly generating partial differential operators and train a network to optimally refine the mesh for a class of operators. A basic optimization algorithm could look as follows:

Algorithm 23. *Randomly generate a sequence of problems denoted by $M_1, M_2, \dots, M_k$ (random domains, coefficients, right-hand sides, ...).*
While $G >$ tol do:

1. *Run Algorithm 4 for the current problems M_i, $i = 1, \dots, k$ and compute an estimate of the error $\rho_L^{(i)}$.*
2. *Compute $G := \sum_{i=1}^k (\rho_L^{(i)})^2$*
3. *Optimize parameters of* ADAPTIVE.

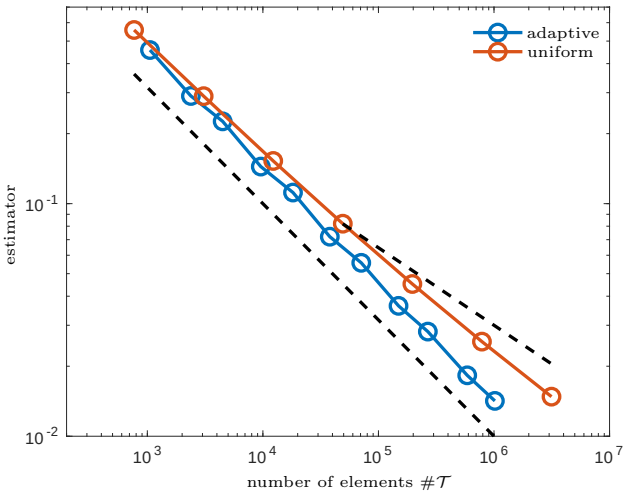


FIGURE 3. Comparision of the performance of the RNN ADAPTIVE versus uniform mesh refinement. The dashed lines indicate the expected rates for uniform/adaptive refinement $\mathcal{O}(N^{-1/3})$ and $\mathcal{O}(N^{-1/2})$.

The standard approach for step (3) of Algorithm 23 would probably be a gradient descent algorithm. However, the adaptive mesh refinement problem is discrete in its nature (an element gets refined or not) and hence also the goal quantity G can not be expected to change continuously with the parameters of the RNN (much less to be differentiable). Therefore, one probably has to apply methods of discrete optimization as well as subgradient or interior point methods. While this would certainly be an interesting direction of research, the optimization (theoretical and practically) of an RNN to perform optimal adaptive mesh refinement must be postponed to future work.

As stated in [21], 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.

5. Numerical Experiments

As a first experiment, we implement the RNN ADAPTIVE from Theorem 3 exactly as shown in the proofs of Section 3. We run Algorithm 4 on an L-shaped domain shown in Figure 4. We choose a constant right-hand side $f = 1$ and start from a coarse triangulation with six elements. Figure 3 shows that the adaptive method reaches the expected convergence rate of $\mathcal{O}(N^{-1/3})$, while the uniform approach only achieves a suboptimal rate due to the singularity at the re-entrant corner of the domain. Figure 4 compares the adaptive meshes generated by ADAPTIVE to a standard adaptive mesh generated by Algorithm 1. This experiment's main purpose is to show that round-off errors do not spoil the theoretically shown performance. Moreover, our implementation only uses $\alpha = 3$ in Theorem 3 and hence shows that the probability estimate is conservative. The tolerance used in the implementation was set to $\delta = 2^{-52}$.

The more interesting experiment would be to find the the weights of ADATIVE by means of computational optimization (machine learning) as described in the previous section in Algorithm 23. This, however, is way beyond the means and purpose of this paper. Therefore, we restrict ourselves to a much simpler setting. We try to find an RNN $\widetilde{\text{MARK}}$ which, given the exact residual based error estimator from (3), marks elements and achieves the optimal order of convergence. To that end, we use the smallest possible blue print for an

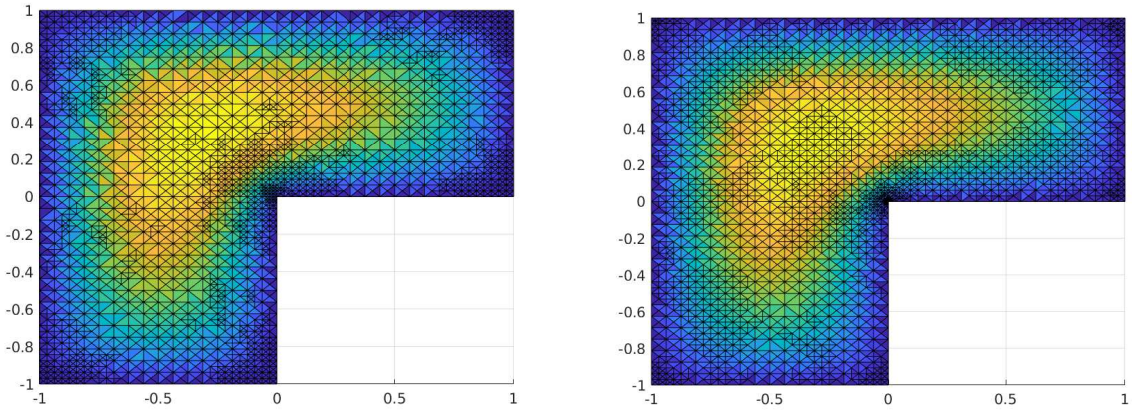


FIGURE 4. Comparision of the adaptive meshes generated after 9 steps of adaptive refinement with the RNN ADAPTIVE (left) and the standard residual error estimator (3) with Dörfler marking (right).

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

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

Although it is not known wether the maximum strategy leads to optimal convergence in the sense of (5), it is usually observed in practice and [13] 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 by 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 inizialization as discribed in Section 2.6, 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 $\mathcal{M} = \big\{T_i \in \mathcal{T} : y_i > 0\big\}$. Now, we know the structure necessary to represent the maximum strategy.

To find $\widetilde{\text{MARK}}$ by machine learning, we set up an optimization algorithm, which is a simplification of Algorithm 23 to compute the necessary weights. We initialize an RNN with the structure as given above with random weights, run Algorithm 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). Note that we additionally applied the random shuffling of the input from Theorem 3 both for training and evaluation.

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. [5] for details). As discussed in the previous section, this is necessary since marking is not a continuous procedure. As discussed in Section 4, we limited the values of the recursive weights to the set $\{-1, 0, 1\}$ to avoid blow-up or dampening. The weights found by the

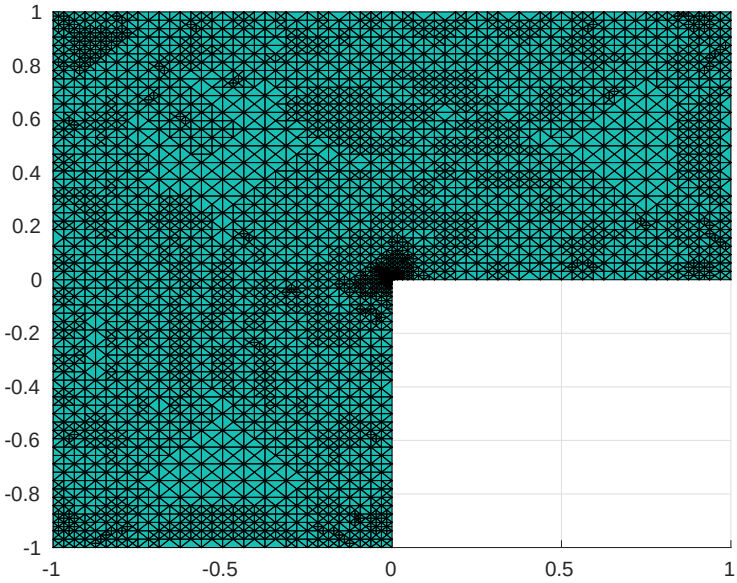


FIGURE 5. Adaptive mesh generated by the RNN $\widetilde{\text{MARK}}$ found by stochastic gradient descent.

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 and Figure 6 that it behaves in an empirically optimal fashion and also the generated meshes look reasonable.

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] 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.

[4] 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.

[5] 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.

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

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

[8] Carsten Carstensen, Daniel Peterseim, and Hella Rabus. Optimal adaptive nonconforming FEM for the Stokes problem. *Numer. Math.*, 123(2):291–308, 2013.

[9] 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.

[10] 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.

[11] J. Manuel Cascon and Ricardo H. Nochetto. Quasioptimal cardinality of AFEM driven by nonresidual estimators. *IMA J. Numer. Anal.*, 32(1):1–29, 2012.

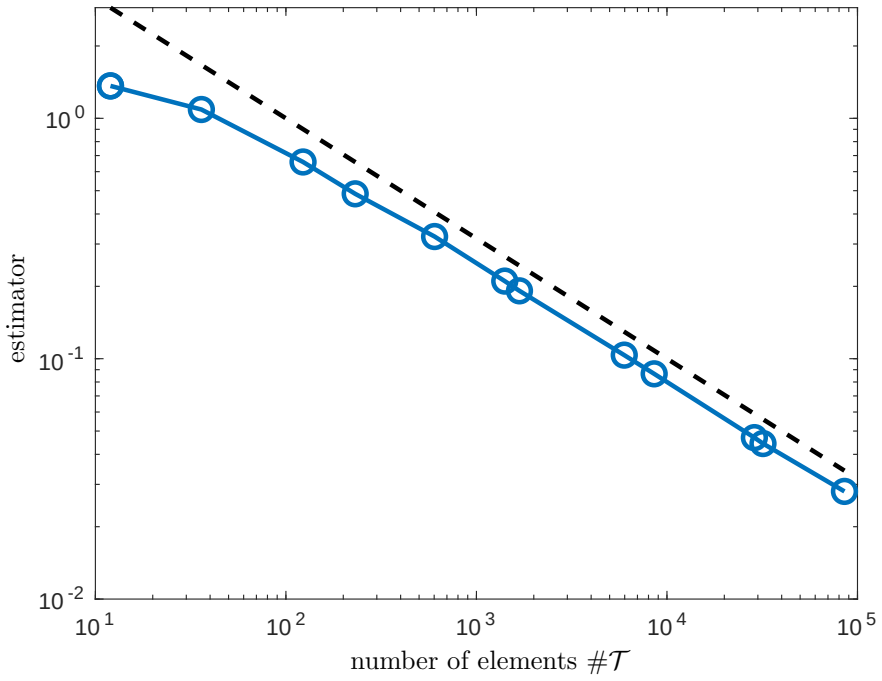


FIGURE 6. Performance of the RNN $\widetilde{\text{MARK}}$ as a marking strategy in Algorithm 1. The dashed line marks the optimal rate of convergence $\mathcal{O}(N^{-1/2})$.

[12] Long Chen, Michael Holst, and Jinchao Xu. Convergence and optimality of adaptive mixed finite element methods. *Math. Comp.*, 78(265):35–53, 2009.

[13] Lars Diening, Christian Kreuzer, and Rob Stevenson. Instance optimality of the adaptive maximum strategy. *Found. Comput. Math.*, 16(1):33–68, 2016.

[14] Michael Feischl. Optimal adaptivity for a standard finite element method for the stokes problem. *to appear SIAM. J. Numer. Anal.*, 2019.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Tsogtgerel Gantumur. Adaptive boundary element methods with convergence rates. *Numerische Mathematik*, 124(3):471–516, 2013.

[20] 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.

[21] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[22] 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.

[23] Christian Kreuzer and Kunibert G. Siebert. Decay rates of adaptive finite elements with Dörfler marking. *Numer. Math.*, 117(4):679–716, 2011.

[24] 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.

[25] 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.

[26] Carl-Martin Pfeiler and Dirk Praetorius. Drfler marking with minimal cardinality is a linear complexity problem. *Eprint arXiv:1907.13078*, 2019.

[27] Hella Rabus. A natural adaptive nonconforming FEM of quasi-optimal complexity. *Comput. Methods Appl. Math.*, 10(3):315–325, 2010.

[28] 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.

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

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

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