TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Adaptive mesh refinement for finite element methods with machine learning

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik – Schwerpunkt Angewandte Mathematik

eingereicht von

Benedikt Moser
Matrikelnummer 01527052

ausgeführt am Institut für Analysis und Scientific Computing
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung
Betreuer: Univ.Prof. Dipl.-Ing. Dr.techn. Michael Feischl BSc

Wien, 28.08.2023 [redacted] [redacted]

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Contents

Abstract

In [1] it has been shown that neural networks exist that provide at least as good results for adaptive mesh refinement in finite element methods as current optimal mesh refinement strategies and that they are problem independent. This master thesis is dedicated to the experimental exploration of these theoretical results. Additionally, the work includes a theoretical overview of neural networks, also covering the ability of neural networks to approximate splines and polynomials effectively, as well as an explanation of the finite element method.

The primary objective of this work is the design and implementation of a neural network suitable for adaptive mesh refinement. While the outcomes are not entirely satisfactory, they suggest that with additional effort, advancements could be achieved.

Another emphasis is placed on evaluating whether a neural network can effectively learn the residual error estimator for the Poisson problem, a goal that has been successfully realized.

The thesis also addresses the formulation of an optimization approach for the neural network, tailored to mesh refinement. The outcomes of this optimization endeavor are partially affirmative, indicating that a similar approach might hold the potential to enhance mesh refinement procedures.

Overall, this master's thesis contributes to expanding the comprehension of neural network applications in numerical mathematics. The experimental scrutiny of the presented methodologies and the partial successes attained in the neural network's development underscore the promise of further research and optimization within this domain.

Chapter 1

Introduction

One method to solve partial differential equations (PDEs) numerically is the so-called finite element method (FEM). In this method the domain on which the PDE should be solved is discretized with elements, e.g. triangles in two dimensional applications. Then the numerical solution is computed as a linear combination of ansatzfunctions in a suitable approximating space. For many problems it can be shown that the finer the discretization, i.e. the smaller the elements, the more accurate is the numerical solution. (cf. [5])

But the finer the mesh gets the more expensive the calculation of the numerical solution becomes. Hence, the best possible accuracy is determined by the computing power. In order to make the most efficient use of the available computing power, one utilizes adaptive mesh refinement. Unlike the static grid structure of traditional FEM, the adaptive method allows grid points to be added or removed at specific locations in the simulation domain to produce more accurate results in areas of high variation or importance while spending less effort on less important areas. The adaptive approach allows to refine the grid where high gradients or strong changes occur in the numerical solution and to use coarser grids where the changes are slower or less relevant. This allows simulations to be performed with higher accuracy, and often faster, by concentrating computational effort in the important areas. This adaptive finite element method (AFEM) is particularly useful when the solution to a differential equation has strong local variations or discontinuities, as it allows the grid to be adapted to these conditions without having to maintain unnecessarily high resolutions throughout the whole domain. It can be shown that adaptive mesh refinement can achieve optimality in regards to computational cost. (cf. [4])

Most AFEM algorithms share a common framework, employing a standard adaptive refinement procedure that iterates the steps:

1. **Solve**
2. **Estimate**

3. **Mark**

4. **Refine**

In this process, a problem-specific error estimator is computed based on the current solution and is subsequently utilized to refine specific elements of the mesh. (cf. [4])

However, a significant issue arises when dealing with algorithms of this nature. Generally, the error estimators are limited in their applicability to a specific problem type and a particular numerical method.

This suggests the development of black-box mesh refinement algorithms. These algorithms aim to be provably optimal for a wide range of problems and require minimal user intervention beyond the ability to compute numerical approximations on a given mesh. In essence, the objective is to replace the **Estimate** and **Mark** modules in the aforementioned adaptive loop with a problem-independent black-box **Adaptive**. This black-box **Adaptive** serves as a module that takes the current numerical approximation as input and provides guidance on how to refine the mesh in the most effective manner, aiming to minimize the approximation error while considering the computational work involved. The goal is to create a versatile method that can be applied across various problems without requiring specific knowledge about the problem or the need for custom implementation. By employing this black-box approach, users can focus on generating numerical approximations while relying on Adaptive to handle the mesh refinement process and optimize the overall computational performance. (cf. [1])

In [1] the theoretical framework is provided and it is shown that black-box mesh refinement using recurrent neural networks is at least as good as current optimal methods and is problem independent. It can even achieve optimal results in areas not covered by existing adaptive mesh refinement theory. The purpose of this work is to experimentally demonstrate some of these results.

First, Chapter 2 describes neural networks and demonstrates the capability of neural networks to approximate both splines and polynomials. Chapter 3 is a short introduction into finite element methods and a posteriori error estimation. Chapter 4 examines whether a neural network can effectively learn the residual error estimator for the Poisson problem. Chapter 5 focuses on exploring optimization techniques for training a neural network suitable for adaptive mesh refining and on an approach which uses machine learning for AFEM.

All the code used in this thesis can be found in `https://github.com/benmoser-23/ML-in-AFEM`. In the directory `learn_err_est` is the code from Chapter 4, the code used in Section 5.2 is in the directory `optimizer` and the code used in Section 5.3 is in `AFEM_with_NN`.

Chapter 2

Neural network

Neural networks are a class of machine learning models inspired by the structure and functioning of the human brain's neural networks. They are used for various tasks, such as image and speech recognition, natural language processing, and more.

At its core, a neural network consists of interconnected nodes, or "neurons", organized in layers. These layers typically include an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to neurons in adjacent layers through weighted connections.

The process of training a neural network involves feeding it a large dataset with input data and corresponding desired outputs. The network adjusts its internal weights through a process called "backpropagation", which involves calculating the difference between the predicted outputs and the actual outputs, and then updating the weights to minimize this difference, thus improving the network's ability to make accurate predictions.

Neural networks are powerful because they can automatically learn complex patterns and representations from data, enabling them to generalize and perform well on unseen examples. Deep neural networks, with multiple hidden layers, are particularly skilled at capturing hierarchical and abstract features from the input data, making them suitable for tackling complicated tasks.

In recent years, advancements like convolutional neural networks (CNNs) for image data and recurrent neural networks (RNNs) for sequential data have further improved the performance of neural networks in specialized domains. Overall, neural networks have become a foundational technology in the realm of artificial intelligence, enabling machines to learn from data and perform tasks that were previously challenging for traditional rule-based approaches. (cf. [6], [9])

The first section of this chapter focuses on the the ability of neural networks to approximate splines and polynomials. The purpose of this theoretical part is that if it is possible to efficiently approximate polynomials

then it is possible to approximate a wide variety of functions with very few degrees of freedom. And hence, it is plausible that neural networks can learn an error estimator. We will experimentally examine this ability in Chapter 4.

In the second section we will quickly go through layers, activation functions and optimizers used in machine learning with neural networks.

2.1 Approximating splines and polynomials with neural networks

All the results of this section can be found in [11, Sections 2-4], [12, Section 2] or [16, Section 3].

2.1.1 Definitions

In accordance with common convention, we make a distinction between a neural network (NN) in terms of its parameters and what is known as its realization. The realization refers to the actual map realized by the network, which is achieved by iteratively applying affine linear transformations, determined by the parameters, along with an activation function. All following definitions are according to [11, Section 2].

Definition 2.1. Let $d, L \in \mathbb{N}$, $N_0 = d$, $N_1, \dots, N_L \in \mathbb{N}$. For $\ell = 1, \dots, L$ let $W_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{N_\ell}$. Then a *neural network* $\mathcal{N}$ *with* L *Layers and input dimension* d is defined as

$$\mathcal{N} = ((W_1, b_1), \dots, (W_L, b_L)) \tag{2.1}$$

We see that a neural network is just a sequence of tupels consisting of a matrix and a vector. The matrices, denoted as W_ℓ, are referred to as the weights, while the vectors, denoted as b_ℓ, are known as the biases of the ℓ-th layer of the neural network.

Definition 2.2. Let $\varrho : \mathbb{R} \to \mathbb{R}$ be an activation function which acts component-wise if its input is a vector, i.e. $\varrho(x) := (\varrho(x^1), \dots, \varrho(x^m))$ if $x = (x^1, \dots, x^m) \in \mathbb{R}^m$. Let $x_L \in \mathbb{R}^{N_L}$ be determined by

$$\begin{aligned} x_0 &:= x, \\ x_\ell &:= \varrho(W_\ell x_{\ell-1} + b_\ell) \quad \text{for } \ell = 1, \dots, L-1 \\ x_L &:= W_L x_{L-1} + b_L \end{aligned} \tag{2.2}$$

Then the *realization* R *of the neural network* $\mathcal{N}$ is

$$R(\mathcal{N}) : \mathbb{R}^d \to \mathbb{R}^{N_L} : x \mapsto x_L =: R(\mathcal{N})(x). \tag{2.3}$$

The realization applied to an input x can also be expressed as

$$R(\mathcal{N})(x) = W_L \varrho(W_{L-1} \dots \varrho(W_2 \varrho(W_1 x + b_1) + b_2) \cdots + b_{L-1}) + b_L. \quad (2.4)$$

In this representation, we observe that each layer of the neural network entails a matrix-vector multiplication, an addition operation and an evaluation of the activation function.

In this section, we exclusively focus on a specific activation function:

Definition 2.3. The so-called *standard rectified linear unit (ReLU) function* is defined as

$$\varrho : \mathbb{R} \to \mathbb{R} : x \mapsto \max\{0, x\}. \quad (2.5)$$

Several significant numbers play a crucial role within the neural network framework. We collect them in the following Definition.

Definition 2.4. Let $\mathcal{N}$ be a neural network. Let $d, L, N_0, \dots, N_L$ be as in Definition 2.1. Let $\|A\|_{\ell^0} := \#\{A_{i,j} \neq 0\}$ denote the number of non-zero entries of a matrix $A \in \mathbb{R}^{m \times n}$ and analogously $\|x\|_{\ell^0}$ the number of non-zero entries of a vector $x \in \mathbb{R}^m$. Then we call

- $N(\mathcal{N}) := d + \sum_{j=1}^{L} N_j$ the *number of neurons*,
- $L(\mathcal{N}) := L$ the *number of layers* or *depth*
- $M_j(\mathcal{N}) := \sum_{j=1}^{L} (\|W_j\|_{\ell^0} + \|b_j\|_{\ell^0})$ the *number of weights in the ℓ-th layer* and
- $M(\mathcal{N}) := \sum_{j=1}^{L} M_j(\mathcal{N})$ the *number of weights* or *size*

of the neural network $\mathcal{N}$.

2.1.2 Fundamental operations on neural networks

In this part, we examine some results regarding fundamental operations within neural networks which we will need in the remainder of the section.

Lemma 2.5 (Emulation of identity, [12, Lemma 2.3, Remark 2.4])**.** *Let $d, L \in \mathbb{N}$. Then there exists a neural network $\mathcal{N}_{d,L}^{\mathrm{id}}$ with L layers such that*

$$R(\mathcal{N}_{d,L}^{\mathrm{id}}) = \mathrm{Id}_d, \quad (2.6)$$

where Id_d is the identity on $\mathbb{R}^d$.

Proof. For $L \geq 2$ define

$$\mathcal{N}_{d,L}^{\mathrm{id}} := \left(\left(\begin{pmatrix} \mathrm{Id}_d \\ -\mathrm{Id}_d \end{pmatrix}, 0 \right), \underbrace{(\mathrm{Id}_{2d}, 0), \dots, (\mathrm{Id}_{2d}, 0)}_{L-2 \text{ times}}, \left(\begin{pmatrix} \mathrm{Id}_d & -\mathrm{Id}_d \end{pmatrix}, 0 \right) \right).$$

Obviously, this network has L Layers. For $x \in \mathbb{R}^d$ and with the fact that $\varrho \circ \varrho = \varrho$ there holds

$$\begin{aligned} R(\mathcal{N}_{d,L}^{\mathrm{id}})(x) &= \begin{pmatrix} \mathrm{Id}_d & -\mathrm{Id}_d \end{pmatrix} \varrho \left(\dots \mathrm{Id}_{2d} \varrho \left(\mathrm{Id}_{2d} \varrho \left(\begin{pmatrix} \mathrm{Id}_d \\ -\mathrm{Id}_d \end{pmatrix} x \right) \right) \dots \right) \\ &= \begin{pmatrix} \mathrm{Id}_d & -\mathrm{Id}_d \end{pmatrix} \begin{pmatrix} \varrho(x) \\ -\varrho(x) \end{pmatrix} \\ &= \varrho(x) - \varrho(-x) = x. \end{aligned}$$

For $L = 1$, $\mathcal{N}_{d,1}^{\mathrm{id}} := ((\mathrm{Id}_d, 0))$ has the desired properties. □

Lemma 2.6 (Concatenation, [12, Remark 2.6])**.** *Let $L_1, L_2 \in \mathbb{N}$. For two neural networks $\mathcal{N}^1 = ((W_1^1, b_1^1), \dots, (W_{L_1}^1, b_{L_1}^1))$, $\mathcal{N}^2 = ((W_1^2, b_1^2), \dots, (W_{L_2}^2, b_{L_2}^2))$ where the input layer of $\mathcal{N}^1$ has the same dimension as the output layer of $\mathcal{N}^2$ there exists a neural network $\mathcal{N}^1 \bullet \mathcal{N}^2$ called the* concatenation of $\mathcal{N}^1$ and $\mathcal{N}^2$ *with $L_1 + L_2$ layers,*

$$R(\mathcal{N}^1 \bullet \mathcal{N}^2) = R(\mathcal{N}^1) \circ R(\mathcal{N}^2) \tag{2.7}$$

and $M(\mathcal{N}^1 \bullet \mathcal{N}^2) \leq 2M(\mathcal{N}^1) + 2M(\mathcal{N}^2)$. Furthermore, if $\mathcal{N}^1, \dots, \mathcal{N}^n$ are $n \in \mathbb{N}$ neural networks with suitable input and output dimensions then

$$M(\mathcal{N}^1 \bullet \dots \bullet \mathcal{N}^n) \leq 4^{n-1} \max\{M(\mathcal{N}^1), \dots, M(\mathcal{N}^n)\}. \tag{2.8}$$

Proof. If we define

$$\begin{aligned} \mathcal{N}^1 \bullet \mathcal{N}^2 := \Bigg(&(W_1^2, b_1^2), \dots, (W_{L_2-1}^2, b_{L_2-1}^2), \left(\begin{pmatrix} W_{L-2}^2 \\ -W_{L-2}^2 \end{pmatrix}, \begin{pmatrix} b_{L-2}^2 \\ -b_{L-2}^2 \end{pmatrix} \right), \\ &\left(\begin{pmatrix} W_1^1 & -W_1^1 \end{pmatrix}, \begin{pmatrix} b_1^1 & -b_1^1 \end{pmatrix} \right) (W_2^1, b_2^1), \dots, (W_{L_1}^1, b_{L_1}^1) \Bigg). \end{aligned}$$

(2.7) can be verified directly. It is also easy to see that the estimate $M(\mathcal{N}^1 \bullet \mathcal{N}^2) \leq 2M(\mathcal{N}^1) + 2M(\mathcal{N}^2)$ holds is also easy to see and (2.8) follows from induction and the fact that $a + b \leq 2 \max\{a, b\}$. □

Remark 2.7. Let $m \in \mathbb{N}$ and $\mathcal{N}_m^+ := ((W_1^+, b_1^+), (W_2^+, b_2^+))$ be a neural network with

$$\begin{aligned} W_1^+ &:= \begin{pmatrix} \mathrm{Id}_d & \cdots & \mathrm{Id}_d \\ -\mathrm{Id}_d & \cdots & -\mathrm{Id}_d \end{pmatrix} \in \mathbb{R}^{2d \times md}, \quad b_1^+ := 0 \in \mathbb{R}^{2d}, \\ W_2^+ &:= \begin{pmatrix} \mathrm{Id}_d & -\mathrm{Id}_d \end{pmatrix} \in \mathbb{R}^{d \times 2d}, \quad b_2^+ := 0 \in \mathbb{R}^d, \end{aligned}$$

where $\mathrm{Id}_d \in \mathbb{R}^d$ is the identity matrix. Then for $x_1, \dots x_m \in \mathbb{R}^d$ there holds

$$R(\mathcal{N}^+) \left(\begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} \right) = \varrho \left(\sum_{k=1}^m x_k \right) - \varrho \left(-\sum_{k=1}^m x_k \right) = \sum_{k=1}^m x_k \tag{2.9}$$

Indeed, we have built a neural network capable of performing vector addition.

Another operation which will be helpful in the following is multiplying a vector with a fixed scalar $a \in \mathbb{R}^n$. To this end let $\mathcal{N}_a^{\text{scalar}} := ((W_1^a, b_1^a), (W_2^a, b_2^a))$ be a neural network with

$$W_1^a := \begin{pmatrix} a\mathrm{Id}_d \\ -a\mathrm{Id}_d \end{pmatrix} \in \mathbb{R}^{2d \times d}, \quad b_1^a := 0 \in \mathbb{R}^{2d},$$
$$W_2^a := \begin{pmatrix} \mathrm{Id}_d & -\mathrm{Id}_d \end{pmatrix} \in \mathbb{R}^{d \times 2d}, \quad b_2^a := 0 \in \mathbb{R}^d.$$

Then for $x \in \mathbb{R}^d$ there holds

$$R(\mathcal{N}_a^{\text{scalar}})(x) = \varrho(ax) - \varrho(-ax) = ax. \tag{2.10}$$

Lemma 2.8 (Parallelization, [12, Definition 2.7, Remark 2.8])**.** *Let $d \in \mathbb{N}$, $L_1, \dots, L_m \in \mathbb{N}$ and let $\mathcal{N}^1, \dots, \mathcal{N}^m$ be neural networks with input dimension d and $L_1, \dots, L_m$ layers respectively. Then there exists a neural Network $\mathcal{N}^{\text{par}}(\mathcal{N}^1, \dots, \mathcal{N}^m)$ such that*

$$R(\mathcal{N}^{\text{par}}(\mathcal{N}^1, \dots, \mathcal{N}^m)) = \begin{pmatrix} R(\mathcal{N}^1) \\ \vdots \\ R(\mathcal{N}^m) \end{pmatrix} \in \mathbb{R}^{md} \tag{2.11}$$

Proof. Let $\mathcal{N}^1 = ((W_1^1, b_1^1), \dots, (W_{L_1}^1, b_{L_1}^1))$, $\mathcal{N}^2 = ((W_1^2, b_1^2), \dots, (W_{L_2}^2, b_{L_2}^2))$ be two neural networks with input dimension d and $L_1 = L_2 = L$ layers. Define $\widetilde{\mathcal{N}}^{\text{par}}(\mathcal{N}^1, \mathcal{N}^2) := ((\widetilde{W}_1, \widetilde{b}_1), \dots, (\widetilde{W}_L, \widetilde{b}_L))$ with

$$\widetilde{W}_1 := \begin{pmatrix} W_1^1 \\ W_1^2 \end{pmatrix}, \quad \widetilde{b}_1 := \begin{pmatrix} b_1^1 \\ b_1^2 \end{pmatrix} \quad \text{and}$$
$$\widetilde{W}_\ell := \begin{pmatrix} W_\ell^1 & 0 \\ 0 & W_\ell^2 \end{pmatrix}, \quad \widetilde{b}_\ell := \begin{pmatrix} b_\ell^1 \\ b_\ell^2 \end{pmatrix} \quad \text{for } \ell = 2, \dots, L.$$

Then it follows directly that

$$R(\widetilde{\mathcal{N}}^{\text{par}}(\mathcal{N}^1, \mathcal{N}^2)) = \begin{pmatrix} R(\mathcal{N}^1) \\ R(\mathcal{N}^2) \end{pmatrix}.$$

Now consider two networks $\mathcal{N}^1$, $\mathcal{N}^2$ with different sizes $L_1 < L_2$. Then we define $\mathcal{N}^{\text{par}}(\mathcal{N}^1, \mathcal{N}^2) := \widetilde{\mathcal{N}}^{\text{par}}(\mathcal{N}^1 \bullet \mathcal{N}_{d, L_2 - L_1}^{\text{id}}, \mathcal{N}^2)$.

If we want the parallelization of three networks $\mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3$ then we define $\mathcal{N}^{\text{par}}(\mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3) := \mathcal{N}^{\text{par}}(\mathcal{N}^{\text{par}}(\mathcal{N}^1, \mathcal{N}^2), \mathcal{N}^3)$. If we repeat this process then we showed the claim of the lemma. □

Remark 2.9 (Linear combination of neural networks)**.** Let $m \in \mathbb{N}$, $\boldsymbol{a} = (a_k)_{k\in\{1,\dots,m\}} \subset \mathbb{R}$ and $\boldsymbol{N} = (\mathcal{N}^k)_{k\in\{1,\dots,m\}}$ be a family of neural networks with input dimension $d \in \mathbb{N}$ and output dimension $n \in \mathbb{N}$. With the concatenation from Lemma 2.6, $\mathcal{N}_m^+$ and $\mathcal{N}_a^{\text{scalar}}$ from Remark 2.7 and $\mathcal{N}^{\text{par}}$ from Lemma 2.8 we define

$$\mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\text{LC}} := \mathcal{N}_m^+ \bullet \mathcal{N}^{\text{par}}(\mathcal{N}_{a_1}^{\text{scalar}} \bullet \mathcal{N}^1, \dots, \mathcal{N}_{a_m}^{\text{scalar}} \bullet \mathcal{N}^m).$$

Then with (2.7), (2.9), (2.10) and (2.11) there holds

$$\begin{aligned}
R(\mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\text{LC}})(x) &= R(\mathcal{N}_m^+ \bullet \mathcal{N}^{\text{par}}(\mathcal{N}_{a_1}^{\text{scalar}} \bullet \mathcal{N}^1, \dots, \mathcal{N}_{a_m}^{\text{scalar}} \bullet \mathcal{N}^m))(x) \\
&= R(\mathcal{N}_m^+) \circ R(\mathcal{N}^{\text{par}}(\mathcal{N}_{a_1}^{\text{scalar}} \bullet \mathcal{N}^1, \dots, \mathcal{N}_{a_m}^{\text{scalar}} \bullet \mathcal{N}^m))(x) \\
&= R(\mathcal{N}_m^+)(R(\mathcal{N}^{\text{par}}(\mathcal{N}_{a_1}^{\text{scalar}} \bullet \mathcal{N}^1, \dots, \mathcal{N}_{a_m}^{\text{scalar}} \bullet \mathcal{N}^m))(x)) \\
&= R(\mathcal{N}_m^+) \begin{pmatrix} \begin{pmatrix} R(\mathcal{N}_{a_1}^{\text{scalar}} \bullet \mathcal{N}^1)(x) \\ \vdots \\ R(\mathcal{N}_{a_m}^{\text{scalar}} \bullet \mathcal{N}^m)(x) \end{pmatrix} \end{pmatrix} \\
&= R(\mathcal{N}_m^+) \begin{pmatrix} \begin{pmatrix} R(\mathcal{N}_{a_1}^{\text{scalar}}) \circ R(\mathcal{N}^1))(x) \\ \vdots \\ R(\mathcal{N}_{a_m}^{\text{scalar}}) \circ R(\mathcal{N}^m))(x) \end{pmatrix} \end{pmatrix} \\
&= R(\mathcal{N}_m^+) \begin{pmatrix} \begin{pmatrix} R(\mathcal{N}_{a_1}^{\text{scalar}})(R(\mathcal{N}^1)(x)) \\ \vdots \\ R(\mathcal{N}_{a_m}^{\text{scalar}})(R(\mathcal{N}^m)(x)) \end{pmatrix} \end{pmatrix} \\
&= R(\mathcal{N}_m^+) \begin{pmatrix} \begin{pmatrix} a_1(R(\mathcal{N}^1)(x)) \\ \vdots \\ a_m(R(\mathcal{N}^m)(x)) \end{pmatrix} \end{pmatrix} \\
&= \sum_{k=1}^{m} a_k R(\mathcal{N}^k).
\end{aligned}$$

2.1.3 Emulating affine splines with neural networks

First we define the right approximation spaces.

Definition 2.10. Let $N \in \mathbb{N}$, $0 = x_0 < x_1 < \cdots < x_{N-1} < x_N = 1$. Then $\mathcal{T} := \{(x_{i-1}, x_i) : i = 1, \dots, N\}$ is called a *partition of the interval* $I := (0,1)$ with *elements* $I_i := (x_{i-1}, x_i)$ and *element sizes* $h_i := x_i - x_{i-1}$ for $i = 1, \dots, N$.

On such a partition we can now define the space of continuous, piecewise polynomial functions also known as splines.

Definition 2.11. Let $\mathcal{T}$ be a partition of $I = (0,1)$ with $N \in \mathbb{N}$ elements. Let $\mathcal{P}^p$ be the space of polynomials with maximal degree $p \in \mathbb{N}$. We define

$$\mathcal{S}^p(I, \mathcal{T}) := \{v \in C(I) : v|_{I_i} \in \mathcal{P}^p(I_i) \text{ for } i = 1, \dots, N\}. \tag{2.12}$$

In the following theorem we construct a network that accurately emulates splines with polynomial degree 1 on a partition of I.

Theorem 2.12 ([11, Lemma 3.1])**.** *Let $\mathcal{T}$ be a partition of $I = (0,1)$ with $N \in \mathbb{N}$ elements and nodes $0 = x_0 < x_1 < \cdots < x_{N-1} < x_N = 1$. Let $v \in \mathcal{S}^1(I, \mathcal{T})$. Then there exists a neural network $\mathcal{N}^v$ with*

(i) $R(\mathcal{N}^v) = v$

(ii) $L(\mathcal{N}^v) = 2$

(iii) $M(\mathcal{N}^v) \leq 3N + 1$

Proof. Let $\mathcal{N}^v := ((W_1^v, b_1^v), (W_2^v, b_2^v))$ with the weights and biases

$$W_1^v := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{N \times 1}, \quad b_1^v := \begin{pmatrix} -x_0 \\ \vdots \\ -x_{N-1} \end{pmatrix} \in \mathbb{R}^N,$$
$$W_2^v \in \mathbb{R}^{1 \times N}, \quad b_2^v := v(x_0) \in \mathbb{R},$$

where the entries of W_2^v are given by

$$(W_2^v)_{1,i} := \begin{cases} \dfrac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}} & \text{if } i = 1, \\ \dfrac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}} - \dfrac{v(x_{i-1}) - v(x_{i-2})}{x_{i-1} - x_{i-2}} & \text{if } i > 1. \end{cases}$$

Obviously, $\mathcal{N}^v$ satisfies claim *(ii)*. Since $\|W_1^v\|_{\ell^0} = N$, $\|b_1^v\|_{\ell^0} \leq N$, $\|W_2^v\|_{\ell^0} \leq N$ and $\|b_2^v\|_{\ell^0} \leq 1$, claim *(iii)* also follows directly.

To show claim *(i)* let $x \in I$. Then there exists $i \in \{1, \ldots, N\}$ such that $x \in (x_{i-1}, x_i]$. The realization of $\mathcal{N}$ is given by

$$R(\mathcal{N})(x) = W_2^v \varrho(W_1^v x + b_1^v) + b_2^v.$$

Since ϱ is the standard ReLU activation function there holds

$$\varrho(W_1^v x + b_1^v) = \begin{pmatrix} \varrho(x - x_0) \\ \vdots \\ \varrho(x - x_{N-1}) \end{pmatrix} = \begin{pmatrix} x - x_0 \\ \vdots \\ x - x_{i-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

If $x \in (x_0, x_1]$ then

$$R(\mathcal{N})(x) = W_2^v \varrho(W_1^v x + b_1^v) + b_2^v = \frac{v(x_1) - v(x_0)}{x_1 - x_0}(x - x_0) + v(x_0).$$

If $x \in (x_{i-1}, x_i]$, $i \in \{2, \dots, N\}$ then

$$W_2^v \varrho(W_1^v x + b_1^v) = \frac{v(x_1) - v(x_0)}{x_1 - x_0}(x - x_0) + v(x_0) + \sum_{j=2}^{i} \left(\frac{v(x_j) - v(x_{j-1})}{x_j - x_{j-1}}(x - x_{j-1}) - \frac{v(x_{j-1}) - v(x_{j-2})}{x_{j-1} - x_{j-2}}(x - x_{j-1}) \right).$$

After a short calculation, one obtains

$$W_2^v \varrho(W_1^v x + b_1^v) = \sum_{j=1}^{i-1} \big(v(x_j) - v(x_{j-1})\big) + \frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}(x - x_{i-1}).$$

Since the first term is a telescoping sum, we get

$$W_2^v \varrho(W_1^v x + b_1^v) = v(x_{i-1}) - v(x_0) + \frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}(x - x_{i-1}).$$

These considerations yield

$$R(\mathcal{N})(x) = v(x_{i-1}) + \frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}(x - x_{i-1}).$$

Note that since $v \in \mathcal{S}^1(I, \mathcal{T})$, for $x \in (x_{i-1}, x_i]$ there holds

$$v(x) = \frac{v(x_i) - v(x_{i-1})}{x_i - x_{i-1}}(y - x_{i-1}) + v(x_{i-1}).$$

This observation concludes the proof. □

2.1.4 Approximating polynomials with neural networks

The main result of this section is

Theorem 2.13 ([11, Proposition 4.2])**.** *Let $\varepsilon > 0$, $m \in \mathbb{N}_0$ and $p = \sum_{k=0}^{m} a_k x^k \in \mathcal{P}^m([-1,1])$ be a polynomial of maximal degree m. Then there exists a neural network $\mathcal{N}_\varepsilon^p$ with input and output dimension 1 which satisfies*

(i) $\max_{x \in [-1,1]} |p(x) - R(\mathcal{N}_\varepsilon^p)(x)| < \varepsilon$,

(ii) $R(\mathcal{N}_\varepsilon^p)(0) = p(0)$,

(iii) $M(\mathcal{N}_\varepsilon^p) \lesssim m \log_2(C_0/\varepsilon) + m \log_2(m) + (1 + \log_2(m))^2 \log_2(C_0/\varepsilon) + m$, *where* $C_0 := \sum_{k=2}^{m} \|a_k\|$.

To prove this theorem we need some preparations. The first lemma shows that there exists a neural network which can approximate the function $f(x) = x^2$.

Lemma 2.14 ([16, Proposition 2])**.** *Let $f(x) = x^2$. For all ε there exists a neural network $\mathcal{N}_\varepsilon^{sq}$ with*

(i) $\max_{x\in[0,1]} |f(x) - R(\mathcal{N}_\varepsilon^{sq})(x)| < \varepsilon$,

(ii) $M(\mathcal{N}_\varepsilon^{sq}) = \mathcal{O}(\log_2(1/\varepsilon))$

Proof. First Step: In this step we show that $f(x) = x^2$ can be approximated by linear combinations of saw-tooth functions.

Let $g : [0,1] \mapsto [0,1]$ be the so-called "tooth"-function given by

$$g(x) = \begin{cases} 2x & \text{if } x < \frac{1}{2}, \\ 2(1-x) & \text{if } x \geq \frac{1}{2}. \end{cases}$$

For $s \in \mathbb{N}$ consider the iterated function

$$g_s(x) = \underbrace{g \circ g \circ \cdots \circ g}_{s}.$$

In [15] it is shown that this function satisfies

$$g_s(x) = \begin{cases} 2^s \left(x - \frac{2k}{2^s}\right) & \text{if } x \in \left[\frac{2k}{2^s}, \frac{2k+1}{2^s}\right], k = 0, 1, \ldots, 2^{s-1} - 1, \\ 2^s \left(\frac{2k}{2^s} - x\right) & \text{if } x \in \left[\frac{2k-1}{2^s}, \frac{2k}{2^s}\right], k = 1, 2, \ldots, 2^{s-1} - 1. \end{cases}$$

Now we will show that $f(x) = x^2$ can be approximated by linear combinations of the functions g_s. To this end, we show that the piecewise linear interpolation f_m of $f(x) = x^2$ with nodes $\frac{k}{2^m}$, $k = 0, \ldots, 2^m$ satisfies

$$f_m(x) = x - \sum_{s=1}^{m} \frac{g_s(x)}{2^{2s}}. \tag{2.13}$$

We show this by induction. Clearly, for $m = 0$ the claim (2.13) holds. Now we suppose that $f_{m-1}(x) = x - \sum_{s=1}^{m-1} \frac{g_s(x)}{2^{2s}}$. We will show that

$$f_{m-1}(x) - f_m(x) = \frac{g_m(x)}{2^{2m}} \tag{2.14}$$

then (2.13) follows immediately. To show (2.14) first note that

$$f_m(x) = \frac{\left(\frac{k+1}{2^m}\right)^2 - \left(\frac{k}{2^m}\right)^2}{\frac{k+1}{2^m} - \frac{k}{2^m}} \left(x - \frac{k}{2^m}\right) + \left(\frac{k}{2^m}\right)^2 \quad \text{for } x \in \left[\frac{k}{2^m}, \frac{k+1}{2^m}\right].$$

If $x \in \left[\frac{2k}{2^m}, \frac{2k+1}{2^m}\right]$ for some $k \in \{0, 1, \dots, 2^{m-1}-1\}$, then $x \in \left[\frac{k}{2^{m-1}}, \frac{k+1}{2^{m-1}}\right]$ and with some calculations we see that

$$f_{m-1}(x) - f_m(x) = \frac{x}{2^m} - \frac{2k}{2^{2m}} = \frac{1}{2^{2m}}\left(2^m\left(x - \frac{2k}{2^m}\right)\right) = \frac{g_m(x)}{2^{2m}}.$$

Analogously, we get that for $x \in \left[\frac{2k-1}{2^m}, \frac{2k}{2^m}\right]$ for some $k \in \{1, 2, \dots, 2^{m-1}\}$ there holds

$$f_{m-1}(x) - f_m(x) = \frac{2k}{2^{2m}} - \frac{x}{2^m} = \frac{1}{2^{2m}}\left(2^m\left(\frac{2k}{2^m} - x\right)\right) = \frac{g_m(x)}{2^{2m}}.$$

Thus we showed 2.14 and therefore also 2.13.

Second Step: We show that the interpolation error of f_m is 2^{-2m-2}. Let $x \in \left[\frac{k}{2^m}, \frac{k+1}{2^m}\right]$. Then after some calculations we obtain

$$f_m(x) - f(x) = -\frac{k^2+k}{2^{2m}} + \frac{x(2k+1)}{2^m} - x^2.$$

The derivative of this expression is given by

$$\frac{d}{dx}(f_m(x) - f(x)) = \frac{2k+1}{2^m} - 2x,$$

the root of this derivative is $x_0 := \frac{2k+1}{2^{2m}}$. Hence,

$$\max_{x \in \left[\frac{k}{2^m}, \frac{k+1}{2^m}\right]} |f_m(x) - f(x)| = |f_m(x_0) - f(x_0)| = 2^{-2m-2}.$$

Since this maximum is independent of k, we also get that

$$\max_{x \in [0,1]} |f_m(x) - f(x)| = 2^{-2m-2}.$$

Third Step: In the last step we construct a neural network which emulates f_m. To this end we define the neural network $\mathcal{N}^g := ((W_1^g, b_1^g), (W_2^g, b_2^g))$ with

$$W_1^g := \begin{pmatrix} 1 \\ 1 \end{pmatrix} \in \mathbb{R}^{2\times 1}, \quad b_1^g := \begin{pmatrix} 0 \\ -\frac{1}{2} \end{pmatrix} \in \mathbb{R}^2,$$
$$W_2^g := \begin{pmatrix} 2 & -4 \end{pmatrix} \in \mathbb{R}^{1\times 2}, \quad b_2^g := 0 \in \mathbb{R}.$$

It it is easy to check that this neural network satisfies $R(\mathcal{N}^g) = g$. Now we set

$$\mathcal{N}^{g_s} = \underbrace{\mathcal{N}^g \bullet \cdots \bullet \mathcal{N}^g}_{s}.$$

Then Lemma 2.6 yields that $R(\mathcal{N}^{g_s}) = g_s$.

Let $\varepsilon > 0$. Choose $m \in \mathbb{N}$ such that $2^{-2m-2} < \varepsilon$. Then we set $\mathcal{N}_\varepsilon^{\mathrm{sq}} := \mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\mathrm{LC}}$ from Remark 2.9 where $\boldsymbol{a} = (a_s)_{s\in\{0,1,\dots,m\}}$ with $a_0 = 1$, $a_s = -2^{2s}$

for $s = 1, \dots, m$ and $\boldsymbol{N} = (\mathcal{N}^s)_{s\in\{0,1,\dots,m\}}$ with $\mathcal{N}^0 = \mathcal{N}^{\mathrm{Id}}_{1,1}$ from Lemma 2.5, $\mathcal{N}^s = \mathcal{N}^{g_s}$ for $s = 1, \dots, m$. Then

$$R(\mathcal{N}^{\mathrm{sq}}_\varepsilon)(x) = R(\mathcal{N}^{\mathrm{LC}}_{\boldsymbol{a},\boldsymbol{N}})(x) = x - \sum_{s=1}^{m} \frac{g_s(x)}{2^{2s}} = f_m(x).$$

Hence, we got that

$$\max_{x\in[0,1]} |f(x) - R(\mathcal{N}^{\mathrm{sq}}_\varepsilon)(x)| = 2^{-2m-2} < \varepsilon$$

which concludes the proof of claim *(i)*.

The proof of claim *(ii)* can be found in [16, Proposition 2]. □

Due to the fact that

$$xy = \frac{1}{2}((x+y)^2 - x^2 - y^2), \tag{2.15}$$

we can now construct a neural network which is able to multiply two numbers.

Lemma 2.15 ([16, Proposition 3])**.** *Let $M > 0$ and $\varepsilon \in (0,1)$, then there exists a neural network $\mathcal{N}^{\times}_{M,\varepsilon}$ such that*

(i) $\left| xy - R(\mathcal{N}^{\times}_{M,\varepsilon}) \left(\begin{pmatrix} x \\ y \end{pmatrix} \right) \right| < \varepsilon$ *for $x, y \in \mathbb{R}$ with $|x|, |y| < M$,*

(ii) if $x = 0$ or $y = 0$, then $R(\mathcal{N}^{\times}_{M,\varepsilon}) \left(\begin{pmatrix} x \\ y \end{pmatrix} \right) = 0$,

(iii) $M(\mathcal{N}^{\times}_{M,\varepsilon}) \leq C_1 \log_2(1/\varepsilon) + C_2$*, where C_2 is an absolute constant and C_2 depends on M.*

Proof. Let $\mathcal{N}^{\mathrm{sq}}_\delta$ be the neural network from Lemma 2.14 for some $\delta > 0$. We

define some neural networks:

$$\mathcal{N}^x := (((1 \quad 0), 0)) \quad \text{then } R(\mathcal{N}^x)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = x,$$

$$\mathcal{N}^y := (((0 \quad 1), 0)) \quad \text{then } R(\mathcal{N}^y)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = y,$$

$$\mathcal{N}^{|\cdot|} := \left(\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix}, 0\right), ((1 \quad 1), 0)\right) \quad \text{then } R(\mathcal{N}^{|\cdot|})(x) = \varrho(x) + \varrho(-x) = |x|,$$

$\mathcal{N}_a^{\text{scalar}}$ as in Remark 2.7 with $R(\mathcal{N}_a^{\text{scalar}})(x) = ax$,

$\mathcal{N}^+$ as in Remark 2.7 with $R(\mathcal{N}^+)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = x + y$,

$$\mathcal{N}^1 := \mathcal{N}_\delta^{\text{sq}} \bullet \mathcal{N}_{\frac{1}{2M}}^{\text{scalar}} \bullet \mathcal{N}^{|\cdot|} \bullet \mathcal{N}^+ \quad \text{then } R(\mathcal{N}^1)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x+y|}{2M}\right),$$

$$\mathcal{N}^2 := \mathcal{N}_\delta^{\text{sq}} \bullet \mathcal{N}_{\frac{1}{2M}}^{\text{scalar}} \bullet \mathcal{N}^{|\cdot|} \bullet \mathcal{N}^x \quad \text{then } R(\mathcal{N}^2)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x|}{2M}\right),$$

$$\mathcal{N}^3 := \mathcal{N}_\delta^{\text{sq}} \bullet \mathcal{N}_{\frac{1}{2M}}^{\text{scalar}} \bullet \mathcal{N}^{|\cdot|} \bullet \mathcal{N}^y \quad \text{then } R(\mathcal{N}^3)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|y|}{2M}\right).$$

Let $\mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\text{LC}}$ be the neural network from Remark 2.9, $\boldsymbol{a} = (2M^2, -2M^2, -2M^2)$, $\boldsymbol{N} = (\mathcal{N}^1, \mathcal{N}^2, \mathcal{N}^3)$, $\delta := \frac{\varepsilon}{6M^2}$ and define $\mathcal{N}_{M,\varepsilon}^\times := \mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\text{LC}}$. Then there holds

$$R(\mathcal{N}_{M,\varepsilon}^\times)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = 2M^2\Bigg(R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x+y|}{2M}\right) - R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x|}{2M}\right) - R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|y|}{2M}\right)\Bigg)$$

With the expansion (2.15) and the triangle inequality we get that

$$\left|xy - R(\mathcal{N}_{M,\varepsilon}^\times)\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\right| \leq A_1 + A_2 + A_3$$

where

$$A_1 := \left|2M^2 R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x+y|}{2M}\right) - \frac{(x+y)^2}{2}\right|$$

$$A_1 := \left|2M^2 R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|x|}{2M}\right) - \frac{x^2}{2}\right|$$

$$A_1 := \left|2M^2 R(\mathcal{N}_\delta^{\text{sq}})\left(\frac{|y|}{2M}\right) - \frac{y^2}{2}\right|$$

Now we estimate A_1. First note that

$$\frac{(x+y)^2}{2} = 2M^2\left(\frac{x+y}{2M}\right)^2,$$

therefore we get with Lemma 2.14 that

$$A_1 = 2M^2 \left| R(\mathcal{N}_\delta^{\mathrm{sq}}) \left(\frac{|x+y|}{2M} \right) - \left(\frac{x+y}{2M} \right)^2 \right| < 2M^2\delta.$$

Analogously we obtain $A_2, A_3 < 2M^2\delta$. Since $\delta = \frac{\varepsilon}{6M^2}$ we have constructed a neural network $\mathcal{N}_{M,\varepsilon}^{\times}$ such that

$$\left| xy - R(\mathcal{N}_{M,\varepsilon}^{\times}) \left(\begin{pmatrix} x \\ y \end{pmatrix} \right) \right| < 6M^2\delta = \varepsilon.$$

It remains to show that *(ii)* holds. Since $g_s(0) = 0$ for all $s \in \mathbb{N}$, this follows directly from the proof of Lemma 2.14.

Claim *(iii)* is shown in [16, Proposition 3]. □

Now we have gathered all the results we need to prove the main result of this section Theorem 2.13.

Proof of Theorem 2.13. Let $\mathcal{N}^0 = ((0,1))$, $\mathcal{N}^1 = \mathcal{N}_{1,1}^{\mathrm{Id}}$ from Lemma 2.5 and

$$\mathcal{N}^k = \mathcal{N}_{M,\delta}^{\times} \bullet \mathcal{N}^{\mathrm{par}}(\mathcal{N}^{k-1}, \mathcal{N}^1).$$

with $\mathcal{N}^{\mathrm{par}}$ from Lemma 2.8 and $\mathcal{N}_{M,\delta}^{\times}$ from Lemma 2.15 for some $\delta > 0$ and $M > 0$. To make the following easier to read, we introduce some notations:

$$g^k(x) := R(\mathcal{N}^k)(x) \quad \text{and}$$
$$\times(x,y) := R(\mathcal{N}_{M,\delta}^{\times}) \left(\begin{pmatrix} x \\ y \end{pmatrix} \right).$$

Let $\varepsilon > 0$, $m \in \mathbb{N}_0$ and $p \in \mathcal{P}^m([-1,1])$ with $p(x) = \sum_{k=0}^m a_k x^k$. We define

$$\mathcal{N}_\varepsilon^p := \mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\mathrm{LC}},$$
$$f^p(x) := R(\mathcal{N}_\varepsilon^p)(x)$$

with $\boldsymbol{a} = (a_k)_{k\in\{1,\dots,m\}}$, $\boldsymbol{N} = (\mathcal{N}^k)_{k\in\{1,\dots,m\}}$ and $\mathcal{N}_{\boldsymbol{a},\boldsymbol{N}}^{\mathrm{LC}}$ from Remark 2.9. Then

$$f^p(x) = \sum_{k=0}^m a_k g^k(x) \quad \text{and}$$
$$g^k(x) = \times(g^{k-1}(x), x)$$

Now note that $g^0(x) = 1$ and $g^1(x) = x$. Therefore, if $m \in \{0,1\}$, $\mathcal{N}_\varepsilon^p$ emulates p exactly.

If $m \geq 2$, set

$$\delta := \frac{2\varepsilon}{\max\limits_{k=2,\dots,m} |a_k|(m-1)m}$$

and let $M > 1 + (m-2)\delta$. We will show that for $x \in [0,1]$ there holds

$$\max_{x\in[0,1]} |x^k - g^k(x)| < (k-1)\delta \quad \text{for all } k \in \{2,\dots,m\}. \tag{2.16}$$

With Lemma 2.15 and since $|x| \leq 1 < M$ we obtain for $k = 2$

$$|x^2 - g^2(x)| = |x^2 - \times(x,x)| < \delta.$$

Now suppose that (2.16) holds for some $k \in \{2,\dots,m-1\}$. Then $|g^k(x)| < 1 + (k-1)\delta < M$ and again Lemma 2.15 yields that

$$\begin{aligned}
\left|x^{k+1} - g^{k+1}(x)\right| &= \left|x^{k+1} - \times(x^k, x)\right| \\
&= \left|x^{k+1} - g^k(x)x + g^k(x)x - \times(x^k, x)\right| \\
&\leq |x| \left|x^k - g^k(x)\right| + \left|g^k(x)x - \times(x^k, x)\right| \\
&< \delta + (k-1)\delta \\
&= k\delta.
\end{aligned}$$

Hence, we showed (2.16) by induction. Now we look at the approximation error of $\mathcal{N}_\varepsilon^p$:

$$\begin{aligned}
|p(x) - f^p(x)| &= \left|\sum_{k=0}^{m} a_k(x_k - g^k(x))\right| = \left|\sum_{k=2}^{m} a_k(x_k - g^k(x))\right| \\
&\leq \sum_{k=2}^{m} |a_k||x_k - g^k(x)| < \sum_{k=2}^{m} |a_k|(k-1)\delta \\
&\leq \delta \max_{k\in\{2,\dots,m\}} |a_k| \sum_{k=2}^{m}(k-1) = \delta \max_{k\in\{2,\dots,m\}} |a_k| \frac{(m-1)m}{2} = \varepsilon.
\end{aligned}$$

This shows *(i)* of the Theorem. Claim *(ii)* follows directly from Lemma 2.15 *(ii)*.

For the proof of claim *(iii)* you can refer to [11, Proposition 4.2]. □

2.2 Neural networks in machine learning

2.2.1 Different layer types

The layers described in Section 2.1 are called *fully connected layers* or *dense layers*. In this type of layer each neuron is connected to every neuron in the previous and subsequent layer.

But there are also other types of layers, for example the *convolutional layer*. Models which use these type of layers are called convolutional neural networks and are often used for processing grid-like data, such as images.

Mathematically, a two-dimensional convolutional layer can be described as follows:
Let's consider an input volume or tensor I of size $H_{\text{in}} \times W_{\text{in}} \times D_{\text{in}}$, where $H_{\text{in}} \in \mathbb{N}$ is the height, $W_{\text{in}} \in \mathbb{N}$ is the width, and $D_{\text{in}} \in \mathbb{N}$ is the depth (number of input channels).

Let K be a set of filters or kernels, each of size $F \times F \times D_{\text{in}}$, where $F \in \mathbb{N}$ is the filter size.

The output of the convolutional layer, which forms the next layer's input, can be denoted as O, with dimensions $H_{\text{out}} \times W_{\text{out}} \times D_{\text{out}}$, where

$$H_{\text{out}} = H_{\text{in}} - F + 1,$$

$$W_{\text{out}} = W_{\text{in}} - F + 1,$$

and D_{out} is the number of filters or kernels used in the layer.

The mathematical operation of a single filter K being convolved with the input I at a specific position (i, j) can be represented as follows:

$$O_{i,j} = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} \sum_{d=0}^{D_{\text{in}}-1} K_{m,n,d} I_{i+m,j+n,d},$$

where $O_{i,j}$ is the output value at position (i, j) in the output tensor, $K_{m,n,d}$ is the weight (kernel value) at position (m, n) in the filter and depth d, $I_{i+m,j+n,d}$ is the input value at position $(i + m, j + n)$ in the input tensor and depth d. This operation is performed for each position (i, j) in the output tensor and for each filter K. The resulting values are stacked along the depth dimension to form the output tensor O.

It's important to note that the output dimensions H_{out} and W_{out} are reduced compared to the input dimensions H_{in} and W_{in}. If you want to retain the same output dimensions, you can use padding. Padding refers to the addition of extra dimensions to the input tensor before applying the convolutional operation.

The so called *pooling layers* are also used in convolutional neural networks to downsample input data by dividing it into small regions (e.g., 2x2) and summarizing the information within each region through operations like max or average pooling. This downsampling reduces computational load and helps in abstracting essential features. Pooling maintains spatial invariance, making the network more tolerant to variations in input position. It's commonly applied after convolutional layers to reduce feature map dimensions while preserving important patterns for subsequent processing.

A *softmax layer* is a component commonly used in neural networks, particularly in classification tasks. It takes a vector $\boldsymbol{z} = (z_1, \ldots, z_k)$ of arbitrary real numbers (often referred to as logits) as input and transforms them into a probability distribution over multiple classes. The output of

the softmax layer assigns a probability P to each class, ensuring that the probabilities sum up to 1. This is achieved by

$$P(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}.$$

The class with the highest probability after the softmax operation is often chosen as the predicted class by the neural network. The softmax function helps convert raw scores into a form that's suitable for interpreting the model's confidence in various class predictions.

2.2.2 Different activation functions

In Definition 2.3 we already defined the ReLU activation function. One potential issue of ReLU neurons is the so called *dying ReLU problem*: Occasionally, neurons can be driven into states where they remain inactive for virtually all input data. During this state, gradients do not flow backward through the neuron, causing the neuron to become trapped in a permanent inactive state. In certain instances, large numbers of neurons in a neural network can get trapped in these non-functional states, effectively diminishing the overall model capacity. This complication usually arises when the learning rate is set excessively high.

One potential issue of ReLU neurons is the so called *dying ReLU problem*: This refers to a situation where a significant portion of the neurons in a network using the ReLU activation function consistently output zero, effectively becoming inactive. This can happen during training when the weights of neurons are updated in such a way that they consistently produce negative values for their inputs, causing the ReLU to output zero and preventing any further learning or gradient flow through those neurons. To describe this problem mathematically consider a neural network layer (W_ℓ, b_ℓ) with $W_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$, $b_\ell \in \mathbb{R}^{N_\ell}$ (cf. Definition 2.1) and the ReLU activation function. Then, according to Definition 2.2 the output of the i-th neuron is given by

$$x_\ell^i = \max\left\{ \sum_{j=1}^{N_{\ell-1}} W_\ell^{i,j} x_{\ell-1}^i + b_\ell^i, 0 \right\}.$$

The dying ReLU problem occurs when the values of $\sum_{j=1}^{N_{\ell-1}} W_\ell^{i,j} x_{\ell-1}^i + b_\ell^i$ become consistently negative or zero. In this case, the ReLU activation function will always output zero, resulting in the gradient of the loss function with respect to the weights (needed for weight updates during backpropagation) also being zero. This effectively stops the neuron from learning, as its weights are not updated.

To address this, one approach is to use the *leaky ReLU* activation function, which introduces a slight positive slope for inputs less than zero:

$$\varrho(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases} = \max\{x, 0.01x\} \tag{2.17}$$

However, this adjustment may lead to a reduction in performance.

A very similar activation function is the *parametric ReLU* (PReLU). It is given by

$$\varrho(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{otherwise,} \end{cases} = \max\{x, ax\} \tag{2.18}$$

where $a \in [0, 1]$ is a learnable parameter. This allows the network to learn the slope that works best for a particular task.

We see that these ReLU functions are continuous but not continuously differentiable. The following two activation functions are examples of smooth activation functions. The first one is the *sigmoid* function:

$$\varrho(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

Because its outputs are between 0 and 1 the sigmoid function is useful for binary classification problems, but it suffers from vanishing gradient problems during training, i.e. the gradients of the loss function with respect to the network's parameters become extremely small as they are propagated backward through the network during the process of gradient descent optimization. This can lead to slow or stalled learning and hinder the network's ability to effectively learn from the data.

The *swish* activation function is given by

$$\varrho(x) = \frac{x}{1 + e^{-\beta x}}, \tag{2.20}$$

where β is a trainable parameter. It combines a linear behavior for positive inputs with a saturating behavior for negative inputs, thanks to the sigmoid component. It has shown to perform well in certain neural network architectures, often providing improvements in training convergence and generalization compared to other activation functions like ReLU.

2.2.3 Stochastic gradient descent

An optimizer plays a crucial role in training neural networks by adjusting the model's parameters to minimize the difference between predicted and actual outputs. One of the most prominent examples is the stochastic gradient descent method or its variants like Adam or RMSprop. This section gives a brief overview of the stochastic gradient descent and some of its variants.

The content stated in this section and a lot more can be found in [2], [3], [7], [8], [10], [13] and [14].

To minimize the difference between the predicted and the actual output one has to quantify it. This is done by the so-called *loss function*. This is a function mapping the weights of the neural network to a real value which measures the quality of the predictions for a given training set. The goal of training is now to find weights which minimize the loss function.

Suppose we want to train a neural network $\mathcal{N}$ with input dimension $d \in \mathbb{N}$, $L \in \mathbb{N}$ layers and output dimension N_L with the data $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$, $\mathcal{Y} = \{y_1, y_2, \dots, y_n\}$ where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}^{N_L}$ for $i = 1, \dots, n$. We denote the vector of learnable paramaters of $\mathcal{N}$ by

$$w = \begin{pmatrix} \operatorname{vec}(W_1) \\ b_1 \\ \operatorname{vec}(W_2) \\ b_2 \\ \vdots \\ \operatorname{vec}(W_L) \\ b_L \end{pmatrix}$$

where $\operatorname{vec}(W)$ denotes the vectorization of a matrix W. Then in many cases the loss function $\mathcal{L}$ can be written as

$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(w), \tag{2.21}$$

where $\mathcal{L}_i(w)$ is the value of the loss function at the i-th sample in the training data. One example of such a loss function is the *mean squared error* given by $\mathcal{L}_i(w) = (R(\mathcal{N})(x_i) - y_i)^2$ where $R(\mathcal{N})$ is the realization of $\mathcal{N}$ described in Definition 2.2.

With such a loss function at hand, the learning of a neural network is minimizing the loss function in regards of its weights. One way to minimize a loss function given by (2.21) is the *gradient descent* method which performs the following iterations:

$$w_{\ell+1} := w_\ell - \eta \nabla \mathcal{L}(w_\ell) = w_\ell - \frac{\eta}{n} \sum_{i=1}^{n} \nabla \mathcal{L}_i(w_\ell), \tag{2.22}$$

where w_0 is a randomly chosen starting guess and η is a step size often referred to as *learning rate*.

Calculating the sum of all gradients could demand costly computations. This becomes especially inconvenient when dealing with vast training datasets lacking straightforward formulas. The process of evaluating the sum of gradients becomes highly expensive in such cases, as each gradient

computation requires evaluating gradients for all individual functions being summed. To address this computational challenge iteratively, *stochastic gradient descent* adopts a strategy of selecting a subset of these individual functions to compute gradients for at each step. This approach proves highly efficient for tackling large-scale machine learning problems.

Stochastic gradient descent involves approximating the true gradient of the function $\mathcal{L}(w)$ by computing the gradient using a single sample. This approximation is achieved through the following update:

$$w_{\ell+1} := w_\ell - \eta \nabla \mathcal{L}_i(w_\ell). \tag{2.23}$$

During the execution of the algorithm, it iterates through the training set, applying the above update for each individual training sample. This process can be repeated multiple times over the training set until the algorithm reaches convergence, these repetitions are often called *epochs*. To prevent patterns from emerging, the data can be shuffled before each pass. To enhance convergence, adaptive learning rates are often employed in practical implementations.

The following theorem shows that under fairly mild assumptions and with an appropriately decreasing learning rate η_ℓ stochastic gradient descent converges almost surely to a point where the gradient is 0. In particular, for strongly convex objective functions it converges to the global minimum. The proof of this theorem is omitted but can be found in [2].

Theorem 2.16. *Suppose that the objective function $\mathcal{L}$ is three times continuously differentiable and bounded from below, the learning rate η_ℓ satisfies*

$$\sum_{\ell=1}^{\infty} \eta_\ell^2 < \infty \quad \text{and} \quad \sum_{\ell=1}^{\infty} \eta_\ell = \infty.$$

Suppose there holds

$$\frac{1}{n} \sum_{i=1}^{n} \|\mathcal{L}_i(w)\|^2 \leq A + B\|w\|^2$$

for all w and some $A, B \geq 0$. Furthermore, suppose there exist constants $E > D > 0$ and $K > 0$ such that

$$\inf_{\|w\|^2 > D} w \cdot \nabla \mathcal{L}(w) > 0$$

and

$$\sup_{\|w\|^2 < E} \|\nabla \mathcal{L}_i(w)\| \leq K \quad \text{for all } i \in \{1, \ldots, n\}.$$

Then the gradient $\nabla \mathcal{L}(w_\ell)$ converges almost surely to 0 for ℓ to ∞.

Numerous enhancements to the fundamental stochastic gradient descent technique have been suggested and applied. One of such a variant is the so-called *stochastic gradient descent with momentum*. The idea of this method is that the update Δw is remembered in each iteration and then the next update to the weights is determined as a linear combination of the gradient in this step and the previous update:

$$\begin{aligned} \Delta w_{\ell+1} &:= \alpha \Delta w_\ell - \eta \nabla \mathcal{L}_i(w_\ell) \\ w_{\ell+1} &:= w_\ell + \Delta w_{\ell+1}. \end{aligned}$$

Hence, the method is given by

$$w_{\ell+1} := w_\ell - \eta \nabla \mathcal{L}_i(w_\ell) + \alpha \Delta w_\ell \tag{2.24}$$

where $\alpha \in [0, 1]$ is an exponential decay factor.

Another variant of the stochastic gradient descent method is the so called *RMSProp* which stands for root mean square propagation. In this method the learning rate is adapted for each of the parameters. This is done by dividing the learning rate for a parameter by a running average of the magnitudes of recent gradients for that parameter. This running average for the ℓ-th iteration is calculated by

$$v_\ell := \gamma v_{\ell-1} + (1 - \gamma)(\nabla \mathcal{L}_i(w_\ell))^2$$

where γ is a forgetting factor. The parameters are then updated by

$$w_{\ell+1} := w_\ell - \frac{\eta}{\sqrt{v_\ell}} \nabla \mathcal{L}_i(w_\ell) \tag{2.25}$$

The combination of the stochastic gradient descent with momentum and RMSProp is called *Adam* which is short for adaptive moment estimation. The iteration for this method is given by

$$\begin{aligned} m_{\ell+1} &= \beta_1 m_\ell + (1 - \beta_1) \nabla \mathcal{L}(w_\ell) \\ v_{\ell+1} &= \beta_2 v_\ell + (1 - \beta_2)(\nabla \mathcal{L}(w_\ell))^2 \\ \widehat{m} &= \frac{m_{\ell+1}}{1 - \beta_1} \\ \widehat{v} &= \frac{v_{\ell+1}}{1 - \beta_2} \\ w_{\ell+1} &= w_\ell - \eta \frac{\widehat{m}}{\sqrt{\widehat{v}} + \varepsilon} \end{aligned}$$

where ℓ denotes the iteration, $\varepsilon > 0$ is small and used to prevent division by 0 and β_1, β_2 are forgetting factors.

All the mentioned methods are implemented in major machine learning toolboxes such as TensorFlow or PyTorch.

Chapter 3

Finite element method and a posteriori error estimation

The finite element method (FEM) is a computational technique used to solve partial differential equations. In this chapter, we present the fundamental principles for elliptic problems based on the Riesz theorem. To illustrate these concepts, we focus on the Poisson equation with homogeneous Dirichlet boundary conditions as a standard example. This chapter follows the lecture notes [5].

3.1 Notation

In this chapter we will use the following notations:

- $\mathcal{D}(\Omega) := C_c^\infty(\Omega) = \{v \in C^\infty(\Omega) \mid \text{supp}(v) \text{ is a compact subset of } \Omega\}$
- $L^1_{loc}(\Omega) = \{v : \Omega \to \mathbb{R} \text{ measureable} \mid \forall K \subset \Omega \text{ compact} : v \in L^1(K)\}$
- $C^k(\overline{\Omega}) = \{v|_{\overline{\Omega}} \mid v \in C^k(\mathbb{R}^d)\}$
- $C^k_0(\overline{\Omega}) = \{v \in C^k(\overline{\Omega}) \mid v|_{\partial\Omega} = 0\}$
- $H^0(\Omega) = L^2(\Omega)$, and for $m \in \mathbb{N}$,
  $H^m(\Omega) = \{v \in L^2(\Omega) \mid u \text{ weakly differentiable}, \nabla v \in H^{m-1}(\Omega)\}$
  with the norms $\|u\|_{H^m(\Omega)} := (\|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{H^{m-1}(\Omega)})^{1/2}$
- $H^1_0(\Omega) = \overline{\mathcal{D}(\Omega)}^{\|\cdot\|_{H^1}}$

3.2 Galerkin schemes

The finite element method is a specific type of Galerkin scheme. In this section, we will outline the fundamental properties of Galerkin schemes. We

use the following notations: H is a Hilbert space, $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ is an equivalent scalar product on H, i.e. its induced norm $|||v||| := \langle\!\langle v, v \rangle\!\rangle^{1/2}$ is an equivalent norm

$$\alpha \|v\|_H \leq |||v||| \leq \beta \|v\|_H \quad \text{for all } v \in H. \tag{3.1}$$

A fundamental result for Galerkin schemes is the Riesz theorem which reads as

Theorem 3.1 (Riesz)**.** *The mapping*

$$I_H : H \to H^* : v \mapsto \langle v, \cdot \rangle_H \tag{3.2}$$

is linear, isometric and bijective. Hence, for all $F \in H^*$ there exists a unique $u \in H$ with $\langle u, v \rangle_H = F(v)$ for all $v \in H$ and $\|u\|_H = \|F\|_{H^*}$.

This theorem proves the existence and uniqueness of a solution $u \in H$ of

$$\langle\!\langle u, \cdot \rangle\!\rangle = F \in H^* \tag{3.3}$$

Now we replace H by a finite dimensional subspace $X_h \subset H$. Since X_h is finite dimensional it is closed and thus also a Hilbert space. Because of this the following definition makes sense.

Definition 3.2. We define the *Galerkin projection* $\mathbb{G}_h : H \to X_h$ where $\mathbb{G}_h u \in X_h$ solves

$$\langle\!\langle \mathbb{G}_h u, \cdot \rangle\!\rangle = \langle\!\langle u, \cdot \rangle\!\rangle \in X_h^* \quad . \tag{3.4}$$

We call $u_h := \mathbb{G}_h u$ *Galerkin solution* and the property

$$\langle\!\langle u - \mathbb{G}_h u, v_h \rangle\!\rangle = 0 \quad \text{for all } v_h \in X_h \tag{3.5}$$

Galerkin orthogonality.

The following theorem shows that if the scalar product $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ and the right hand side $F \in H^*$ are known, the Galerkin solution $\mathbb{G}_h u \in X_h$ can be computed by solving a linear system of equations, without knowing u.

Theorem 3.3. *Let $X_h \subset H$ be a finite dimensional subspace of H and $\{\phi_1, \ldots, \phi_N\}$ be a basis of X_h. Let $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$ defined by*

$$A_{jk} := \langle\!\langle \phi_k, \phi_j \rangle\!\rangle \quad \text{and} \quad b_j := F(\phi_j).$$

Then A is a symmetric, positive definite and, in particular, regular matrix. Furthermore, the Galerkin solution is given by $\mathbb{G}_h u = \sum_{j=1}^N x_j \phi_j$ where $x \in \mathbb{R}^N$ is the solution of $Ax = b$.

Proof. Since $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ is symmetric, the matrix A is symmetric as well. For $x \in \mathbb{R}^N \setminus \{0\}$ and $v_h := \sum_{j=1}^N x_j \phi_j$ there holds

$$x^T A x = \sum_{j=1}^N \sum_{k=1}^N x_j x_k \langle\!\langle \phi_j, \phi_k \rangle\!\rangle = \langle\!\langle v_h, v_h \rangle\!\rangle = |||v_h|||^2 > 0.$$

This proves that A is positive definite and thus regular.

Let $\mathbb{G}_h u$ be the Galerkin solution, then there exists $x \in \mathbb{R}^n$ such that $\mathbb{G}_h u = \sum_{j=1} x_j \phi_j$. Since $\langle\!\langle \cdot, \cdot \rangle\!\rangle$ is linear (3.4) can be written as

$$b_k = F(\phi_k) = \langle\!\langle \mathbb{G}_h u, \phi_k \rangle\!\rangle = \sum_{j=1}^N x_j \langle\!\langle \phi_j, \phi_k \rangle\!\rangle = (Ax)_k \quad \text{for all } k = 1, \dots, N.$$

Hence, $x \in \mathbb{R}^N$ satisfies $Ax = b$. □

Now, let's move forward with the abstract analysis of Galerkin schemes. In this context, we will examine two lemmas that reveal some fundamental properties of the Galerkin projection. The purpose of the first lemma is to demonstrate the stability of the method.

Lemma 3.4. *The Galerkin projection $\mathbb{G}_h$ is a linear and continues projection onto X_h satisfying*

$$\|\mathbb{G}_h u\|_H \leq \frac{\beta}{\alpha} \|u\|_H \quad \textit{for all } u \in H, \tag{3.6}$$

with $\alpha, \beta > 0$ from (3.1) and furthermore $\mathbb{G}_h$ is the orthogonal projection onto X_h with respect to $\langle\!\langle \cdot, \cdot \rangle\!\rangle$.

Proof. Let $u_h \in X_H$ then the Galerkin orthogonality (3.5) implies $\mathbb{G}_h u_h = u_h$. Thus, $\mathbb{G}_h$ is projection onto X_h. Now let $u, v \in H$, $\lambda \in \mathbb{R}$ then (3.5) yields for $w_h \in X_h$

$$\begin{aligned} \langle\!\langle \mathbb{G}_h(u + \lambda v) - (u + \lambda v), w_h \rangle\!\rangle &= \langle\!\langle \mathbb{G}_h u - u, w_h \rangle\!\rangle + \lambda \langle\!\langle \mathbb{G}_h v - v, w_h \rangle\!\rangle \\ &= \langle\!\langle \mathbb{G}_h u + \lambda \mathbb{G}_h v - u + \lambda v), w_h \rangle\!\rangle, \end{aligned}$$

which shows the linearity of $\mathbb{G}_h$.

Let $u \in H$, then with the Cauchy-Schwarz inequality we get that

$$|||\mathbb{G}_h u|||^2 = \langle\!\langle \mathbb{G}_h u, \mathbb{G}_h u \rangle\!\rangle = \langle\!\langle u, \mathbb{G}_h u \rangle\!\rangle \leq |||u||| \, |||\mathbb{G}_h u|||.$$

Hence, $|||\mathbb{G}_h u||| \leq |||u|||$ and therefore

$$\alpha \|\mathbb{G}_h u\|_H \leq |||\mathbb{G}_h u||| \leq |||u||| \leq \beta \|u\|_H.$$

This proves (3.6). The orthogonality follows immediately from the Galerkin orthogonality. □

The following result is called Céa lemma and it yields that the *Galerkin error* $\|u - \mathbb{G}_h u\|_H$ is quasi-optimal, i.e. up to multiplicative constants it behaves like the best approximation of u in X_h. These constants depend solely on the continuous setting and not on the finite dimensional subspace X_h.

Lemma 3.5 (Céa)**.** *For $\alpha, \beta > 0$ from (3.1) and all $u \in H$ there holds*

$$\|u - \mathbb{G}_h u\|_H \leq \frac{\beta}{\alpha} \min_{v_h \in X_h} \|u - v_h\|_H, \tag{3.7}$$

$$|||\, u - \mathbb{G}_h u|||\, = \min_{v_h \in X_h} |||u - v_h|||\,. \tag{3.8}$$

Proof. Let $v_h \in X_h$, then (3.5) and the Cauchy-Schwarz inequality imply

$$|||u - \mathbb{G}_h u|||^2 = \langle\!\langle u - \mathbb{G}_h u, u - v_h \rangle\!\rangle \leq |||u - \mathbb{G}_h u|||\, |||u - v_h|||\,.$$

Now we can take the infimum over all $v_h \in X_h$ on the right hand side. Obviously, this infimum is attained for $v_h = \mathbb{G}_h u$. This shows (3.8).

With the same arguments as in the last prove we get that

$$\alpha \|u - \mathbb{G}_h u\|_H \leq |||u - \mathbb{G}_h u||| \leq |||u - v_h||| \leq \beta \|u - v_h\|_H.$$

Again, we take the infimum over all $v_h \in X_h$ on the right hand side. If $\Pi_h : H \to X_h$ is the orthogonal projection onto X_h with respect to $\|\cdot\|_H$ then the minimum is attained for $v_h = \Pi_h u$. This concludes the proof. □

The now following last result in this section states that if a dense subspace of X can be approximated well by finite dimensional subspaces then the Galerkin method converges.

Proposition 3.6. *Let $D \subset H$ be a dense subspace of H. For all $h > 0$ let X_h be a finite dimensional subspace of H. If*

$$\lim_{h \to 0} \min_{v_h \in X_h} \|v - v_h\|_H = 0 \quad \text{for all } v \in D, \tag{3.9}$$

then

$$\lim_{h \to 0} \|u - \mathbb{G}_h u\|_H = 0. \tag{3.10}$$

Proof. First we state that for $v \in D$ with (3.7) and the triangle inequalitiy there holds

$$\|u - \mathbb{G}_h u\|_H \leq \frac{\beta}{\alpha} \min_{v_h \in X_h} \|u - v_h\|_H \leq \frac{\beta}{\alpha} \Big(\|u - v\|_H + \min_{v_h \in X_h} \|v - v_h\|_H \Big).$$

Now let $\varepsilon > 0$ and define $\delta := \frac{\alpha \varepsilon}{2\beta}$. Since D is dense in H, there exists $v \in D$ such that $\|u - v\|_H < \delta$. The approximation assumption (3.9) yields that there exists an $h_0 > 0$ such that for all $h \in (0, h_0)$ there holds $\min_{v_h \in X_h} \|v - v_h\|_H < \delta$. Altogether, we have that

$$\forall \varepsilon > 0 \exists h_0 > 0 \forall h \in (0, h_0) : \|u - \mathbb{G}_h u\|_H > 0.$$

This proves the claim of the proposition. □

3.3 Finite element method for the Poisson equation

Let $\Omega \subset \mathbb{R}^d$ be a *Lipschitz domain*, i.e. Ω is a bounded, open and connected subset of $\mathbb{R}^d$, Ω is locally on one side of the boundary $\Gamma := \partial\Omega$ and Γ can locally be parametrized by Lipschitz continuous functions. For $v \in C^2(\overline{\Omega})$ we define the Laplace operator as

$$\Delta v(x) := \sum_{j=1}^{d} \frac{\partial^2 u}{\partial x_j^2}(x).$$

Let $f : \Omega \to \mathbb{R}$, then the *strong form* of the Poisson equation with homogeneous Dirichlet boundary conditions reads as: Find $u \in C^2(\overline{\Omega})$ such that

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{3.11}$$

If $u \in C^2(\overline{\Omega})$ solves (3.11) then we call u a *strong solution* of (3.11).

First we look at the so-called Friedrichs inequality.

Proposition 3.7 (Friedrichs inequality)**.** *For all $v \in H_0^1(\Omega)$ it holds that*

$$\|v\|_{L^2(\Omega)} < C_F \|\nabla v\|_{L^2(\Omega)}, \tag{3.12}$$

where $C_F > 0$, solely depends on the Ω.

Proof. We assume that there is no C_F such that (3.12) holds. Then there exists a sequence $(v_n) \subset H_0^1(\Omega)$ such that for $n \in \mathbb{N}$

$$\frac{1}{n}\|v_n\|_{L^2(\Omega)} > \|\nabla v_n\|_{L^2(\Omega)}.$$

We define $w_n = v_n / \|v_n\|_{L^2(\Omega)}$. Then we obtain a sequence $(w_n) \subset H_0^1(\Omega)$ with $\|v_n\|_{L^2(\Omega)} = 1$ and $\|\nabla w_n\|_{L^2(\Omega)} < 1/n$. Since (w_n) is bounded and $H_0^1(\Omega)$ is reflexive the Banach-Alaoglou theorem gives existence of a weakly convergent subsequence. Hence, we may assume $w_n \rightharpoonup w \in H_0^1(\Omega)$. Since $\|\nabla v\|_{L^2(\Omega)} \leq \|v\|_{H^1(\Omega)}$ for all $v \in H_0^1(\Omega)$, the mapping $v \mapsto \|\nabla v\|_{L^2(\Omega)}$ is continuous and therefore there holds

$$\|w\|_{L^2(\Omega)} \leq \liminf_{n\to\infty} \|\nabla w_n\|_{L^2(\Omega)}.$$

This implies that w is constant and thus $w = 0$.

However, the Rellich theorem yields that $w_n \to w \in L^2(\Omega)$. Hence, $\|w\|_{L^2(\Omega)} = \lim_{n\to\infty} \|w_n\|_{L^2(\Omega)} = 1$. So our assumption leads to a contradiction. This proves the proposition. □

The following proposition provides the weak form of the boundary value problem which is uniquely solvable and, in some sense, equivalent to the strong form.

Proposition 3.8. *(i) Let $u \in C^2(\overline{\Omega})$ be strong solution of (3.11) with a given source term $f \in C(\overline{\Omega})$. Then $u \in H_0^1(\Omega)$ and*

$$\langle \nabla u, \nabla v \rangle_{L^2(\Omega)} = \langle f, v \rangle_{L^2(\Omega)} \quad \text{for all } v \in H_0^1(\Omega). \tag{3.13}$$

(ii) Let $f \in L^2(\Omega)$ then the weak form *(3.13) has a unique solution $u \in H_0^1(\Omega)$ and it holds that*

$$\|u\|_{H^1(\Omega)} \leq C \sup_{v \in H_0^1(\Omega)\setminus\{0\}} \frac{\langle f, v \rangle_{L^2(\Omega)}}{\|v\|_{H^1(\Omega)}} \leq C \|f\|_{L^2(\Omega)}. \tag{3.14}$$

(iii) Let $f \in C(\overline{\Omega})$ and $u \in H_0^1(\Omega)$ be the weak solution *of (3.13). If $u \in C^2(\overline{\Omega})$ then u solves the strong form (3.11).*

Proof. ad (i): First note that $u \in C^2(\overline{\Omega})$ and $u|_\Gamma = 0$ of course implies that $u \in H_0^1(\Omega)$. First we will show that (3.13) holds for $v \in C_0^1(\overline{\Omega})$ and then use density arguments to show the claim. To this end, let $u \in C^2(\overline{\Omega})$ be a strong solution of (3.11) and $v \in C_0^1(\overline{\Omega})$. We multiply $-\Delta u = f$ by v, integrate over Ω and use integration by parts. We obtain

$$\int_\Omega f v dx = -\int_\Omega (\Delta u) v dx = \int_\Omega \nabla u \cdot \nabla v dx - \int_\Gamma \frac{\partial u}{\partial n} v ds.$$

Since $v|_\Gamma = 0$, we see that

$$\langle \nabla u, \nabla v \rangle_{L^2(\Omega)} = \langle f, v \rangle_{L^2(\Omega)} \quad \text{for all } v \in C_0^1(\overline{\Omega}).$$

Now let $v \in H_0^1(\Omega)$. Since $C_0^1(\overline{\Omega})$ is dense in $H_0^1(\Omega)$, there exists a sequence $(v_n) \subset C_0^1(\overline{\Omega})$ with $v_n \xrightarrow{H^1} v$. Therefore, $v_n \xrightarrow{L^2} v$, $\nabla v_n \xrightarrow{L^2} \nabla v$ and

$$\langle \nabla u, \nabla v \rangle_{L^2(\Omega)} = \lim_{n\to\infty} \langle \nabla u, \nabla v_n \rangle_{L^2(\Omega)} = \lim_{n\to\infty} \langle f, v_n \rangle_{L^2(\Omega)} = \langle f, v \rangle_{L^2(\Omega)}.$$

ad (ii): With the Friedrichs inequality, $\|v\|_{H^1(\Omega)} \leq C_F \|\nabla v\|_{L^2(\Omega)}$ for all $v \in H_0^1(\Omega)$, it holds that

$$\|\nabla v\|_{L^2(\Omega)}^2 \leq \|v\|_{H^1(\Omega)}^2 \leq (1 + C_F^2) \|\nabla v\|_{L^2(\Omega)}^2 \quad \text{for all } v \in H_0^1(\Omega).$$

Hence, $\langle \nabla u, \nabla v \rangle_{L^2(\Omega)}$ defines an equivalent scalar product on $H_0^1(\Omega)$. Thus, the Riesz theorem yields the existence of a unique solution $u \in H_0^1(\Omega)$ of (3.13). If one plugs in $v = u \in H_0^1(\Omega)$ into (3.13) then

$$\begin{aligned}(1 + C_F^2)^{-1} \|u\|_{H^1(\Omega)}^2 \leq \|\nabla u\|_{L^2(\Omega)}^2 = \langle f, u \rangle_{L^2(\Omega)} &= \frac{\langle f, u \rangle_{L^2(\Omega)}}{\|u\|_{H^1(\Omega)}} \|u\|_{H^1(\Omega)} \\ &\leq \sup_{v \in H_0^1(\Omega)\setminus\{0\}} \frac{\langle f, v \rangle_{L^2(\Omega)}}{\|v\|_{H^1(\Omega)}} \|u\|_{H^1(\Omega)}\end{aligned}$$

which proves the first estimate of (3.14). The second one is a consequence of the Cauchy inequality

$$\langle f, v\rangle_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)}\|v\|_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)}\|v\|_{H^1(\Omega)}.$$

ad (iii): Since we assume that the weak solution u is in $C^2(\overline{\Omega})$, integration by parts yields

$$\langle f + \Delta u, v\rangle_{L^2(\Omega)} = 0 \quad \text{for all } v \in H_0^1(\Omega).$$

Since $\mathcal{D}(\Omega) \subset H_0^1(\Omega)$, the Fundamental Theorem of Calculus of Variations yields that $f + \Delta u = 0$. And $u \in H_0^1(\Omega) \cap C^2(\overline{\Omega})$ implies $u|_\Gamma = 0$. □

This weak form fits into the framework of Section 3.2. For $u, v \in H_0^1(\Omega)$ we define $a(u, v) := \langle \nabla u, \nabla v\rangle_{L^2(\Omega)}$ and $F(v) := \langle f, v\rangle_{L^2(\Omega)}$ for $f \in L^2(\Omega)$. Then (3.13) is equivalent to: Find $u \in H_0^1(\Omega)$ such that:

$$a(u, \cdot) = F \in H_0^1(\Omega)^*.$$

To apply the results of Section 3.2 it is left to show that $a(\cdot, \cdot)$ is an equivalent scalar product on $H_0^1(\Omega)$ and we need to construct finite dimensional subspaces with an approximation property at least on smooth functions.

Remark 3.9. The Friedrichs inequality yields that $a(v, v)^{1/2} = \|\nabla v\|_{L^2(\Omega)}$ is an equivalent norm on $H_0^1(\Omega)$:

$$C_F^{-1}\|v\|_{H^1(\Omega)} \leq \|\nabla v\|_{L^2(\Omega)} \leq \|v\|_{H^1(\Omega)}.$$

In the following we discuss one example of finite dimensional subspaces $X_h \subset H_0^1(\Omega)$ with an approximation property similar to (3.9). First we define a mesh on the domain Ω.

Definition 3.10. We call $\mathcal{T}$ a *triangulation of* Ω if

- $\mathcal{T}$ is finite set of non-degenerate triangles,
- $\overline{\Omega} = \bigcup_{T\in\mathcal{T}} T$ and
- for all $T, T' \in \mathcal{T}$ with $T \neq T'$, it holds that the overlap $T \cap T'$ is a set of measure zero.

For $T = \operatorname{conv}(\{x_T, y_T, z_T\}) \in \mathcal{T}$ we denote by $\mathcal{K}_T := \{x_T, y_T, z_T\}$ the *set of nodes of the triangle* T, by $\mathcal{E}_T := \{\operatorname{conv}(\{x_T, y_T\}), \operatorname{conv}(\{y_T, z_T\}), \operatorname{conv}(\{x_T, z_T\})\}$ the *set of edges of the triangle* T and by $h_T := \operatorname{diam}(T) := \max\{|x - y| : x, y \in T\}$ *the diameter of the triangle* T.

Furthermore, we denote by $\mathcal{K} := \bigcup_{T\in\mathcal{T}} \mathcal{K}_T$ the *set of nodes of the triangulation* $\mathcal{T}$, by $\mathcal{E} := \bigcup_{T\in\mathcal{T}} \mathcal{E}_T$ the *set of edges of the triangulation* $\mathcal{T}$, by $\mathcal{E}_\Gamma := \{E \in \mathcal{E} \mid E \subset \Gamma\}$ the *set of boundary edges of the triangulation* $\mathcal{T}$ and by $\mathcal{E}_\Omega = \mathcal{E} \setminus \mathcal{E}_\Gamma$ the *set of interior edges of the triangulation* $\mathcal{T}$.

A triangulation $\mathcal{T}$ is called a *conforming* or *regular* triangulation if for $T, T' \in \mathcal{T}$ with $T \neq T'$ the intersection $T \cap T'$ is either empty ($T \cap T' = \emptyset$), a joint node ($T \cap T' \in \mathcal{K}_T \cap \mathcal{K}_{T'}$) or a joint edge ($T \cap T' \in \mathcal{E}_T \cap \mathcal{E}_{T'}$).

Definition 3.11. Let $\mathcal{T}$ be a conforming triangulation of Ω then we define the *discrete space of all $\mathcal{T}$-piecewise affine and globally continuous functions* by

$$\mathcal{S}^1(\mathcal{T}) := \{v_h \in C(\Omega) \mid \forall T \in \mathcal{T} : v_h|_T \text{ affine}\}.$$

Moreover, we define

$$\mathcal{S}^1_0(\mathcal{T}) := \{v_h \in \mathcal{S}^1(\mathcal{T}) \mid \forall z \in \mathcal{K} \cap \Gamma : v_h(z) = 0\}.$$

Proposition 3.12. *Let $\mathcal{T}$ be a conforming triangulation of Ω and $N = \#\mathcal{K}$. Then*

(i) $\mathcal{S}^1(\mathcal{T})$ is an N-dimensional subspace of $H^1(\Omega)$.

(ii) For each node $z \in \mathcal{K}$, there exists a unique hat function *$\zeta_z \in \mathcal{S}^1(\mathcal{T})$ with $\zeta_z(z') = \delta_{zz'}$.*

(iii) The set $\mathcal{B} := \{\zeta_z \mid z \in \mathcal{K}\}$ is a basis of $\mathcal{S}^1(\mathcal{T})$. We call $\mathcal{B}$ the nodal basis.

Proof. For $T \in \mathcal{T}$ an affine function $v_h : T \to \mathbb{R}$ is uniquely defined by the nodal values $v_h(z)$, $z \in \mathcal{K}_T$. Hence, for $z \in \mathcal{K}$ the $\mathcal{T}$-piecewise affine function $\zeta_z : \Omega \to \mathbb{R}$ is uniquely defined by $\zeta_z(z') = \delta_{zz'}$. It is left to show that $\zeta_z \in C(\Omega)$. To this end, we have to show that ζ_z is continuous on the interior edges. Since $\mathcal{T}$ is conforming, for two triangles $T, T' \in \mathcal{T}$ there holds that either $T = T'$, $T \cap T' = \{z'\}$ is a joint point or $T \cap T' = E$ is a joint edge. If the last case occurs, then $\mathrm{tr}_E(\zeta_z|_T)$ and $\mathrm{tr}_E(\zeta_z|_{T'})$ are uniquely defined by $\zeta_z(x_E)$ and $\zeta_z(y_E)$, where $E = \mathrm{conv}(\{x_E, y_E\})$. Hence, $\mathrm{tr}_E(\zeta_z|_T) = \mathrm{tr}_E(\zeta_z|_{T'})$ and thus $\zeta_z \in C(\Omega)$. This shows *(ii)* and *(iii)*.

(iii) implies that $\dim(\mathcal{S}^1(\mathcal{T})) = \#\mathcal{K}$. It remains to show that $\mathcal{S}^1(\mathcal{T}) \subset H^1(\Omega)$. Let $v_h \in \mathcal{S}^1(\mathcal{T})$. Since $v_h \in C(\Omega)$ and Ω is bounded, $v_h \in L^2(\Omega)$. Moreover, $v_h|_T$ is smooth for $T \in \mathcal{T}$ which implies that $\nabla(v_h|_T)$ exists in the classical sense. For v_h to be weakly differentiable, it has to hold that for all $j = 1, \dots, d$ there exists $\partial_j v_h$ such that for all $w \in \mathcal{D}(\Omega)$ there holds

$$\int_\Omega v_h(\partial_j w)dx = \int_\Omega (\partial_j v_h)w dx.$$

Let's denote the jump of a function v across an edge E by $[\![v]\!]_E$. Then since $v_h \in C(\Omega)$, there holds that $[\![v_h]\!]_E = 0$ for all $E \in \mathcal{E}_\Omega$. We define $\partial_j v_h$ by

$(\partial_j v_h)|_T = \partial_j(v_h|_T)$ then $\nabla v_h \in L^2(\Omega)$ and with integration by parts

$$\begin{aligned}\int_\Omega (\partial_j v_h) w dx &= \sum_{T\in\mathcal{T}} \int_T (\partial_j v_h) w dx \\ &= \sum_{T\in\mathcal{T}} \left(-\int_T v_h(\partial_j w) dx + \int_{\partial T} v_h w n_j ds \right) \\ &= -\int_\Omega v_h(\partial_j w) dx + \sum_{E\in\mathcal{E}} \left(\int_E [\![v_h]\!]_E w ds \right) \\ &= -\int_\Omega v_h(\partial_j w) dx\end{aligned}$$

Hence, $v_h \in H^1(\Omega)$ and thus $\mathcal{S}^1(\mathcal{T}) \subset H^1(\Omega)$. □

Corollary 3.13. *Let $\mathcal{T}$ be a conforming triangulation of Ω then $\mathcal{S}^1_0(\mathcal{T})$ is a finite dimensional subspace of $H^1_0(\Omega)$ of dimension $\#\{z \in \mathcal{K} \mid z \notin \Gamma\}$.*

Proof. We need to show that for $v_h \in \mathcal{S}^1_0(\mathcal{T})$ there holds $v_h|_\Gamma = 0$. Let $x \in \Gamma$. Since $\mathcal{T}$ is conforming, there exists an edge $E = \mathrm{conv}(\{x_E, y_E\}) \in \mathcal{E}_\Gamma$ such that $x \in E$. The trace $v_h|_E$ is affine and therefore, it is uniquely determined by the nodal values $v_h(x_E) = v_h(y_E) = 0$. That implies that $v_h|_E = 0$ for all $v_h \in \mathcal{S}^1_0(\mathcal{T})$. □

Now it is left to show functions can be approximated by $\mathcal{T}$-piecewise affine functions. The following theorem states this approximation property. But first we need some definitions.

Definition 3.14. Let $\mathcal{T}$ be a triangulation of Ω. We denote by δ_T the height over the longest side of the triangle $T \in \mathcal{T}$. Then, we define the *local mesh-width functions* $h, \delta \in L^\infty(\Omega)$ by

$$h|_T := h_T = \mathrm{diam}(T) \quad \text{and} \quad \delta|_T := \delta_T \quad \text{for all } T \in \mathcal{T}.$$

Moreover, we define the *shape regularity constant* σ of an triangle $T \in \mathcal{T}$ respectively the triangulation $\mathcal{T}$ by

$$\sigma(T) := \frac{h_T}{\delta_T} \quad \text{and} \quad \sigma(\mathcal{T}) := \|h/\delta\|_{L^\infty(\Omega)} = \max_{T\in\mathcal{T}} \frac{h_T}{\delta_T} \geq 1.$$

We say that a conforming triangulation $\mathcal{T}$ is *γ-shape regular* if there exists $\gamma < \infty$ such that $\sigma(\mathcal{T}) \leq \gamma$.

Definition 3.15. For $u \in H^2(\Omega)$ and a conforming triangulation $\mathcal{T}$ we define the *nodal interpolant* as

$$I_h u := \sum_{z\in\mathcal{K}} u(z)\zeta_z \in \mathcal{S}^1(\mathcal{T}), \tag{3.15}$$

where ζ_z are the hat functions defined in Proposition 3.12.

Due to the Sobolev theorem, which states the continuous inclusion $H^m(\Omega) \subset C(\overline{\Omega})$, if Ω is a Lipschitz domain in R^d and $m > d/2$, each function $u \in H^2(\Omega)$ is continuous for $d = 2, 3$. Hence, the above definition makes sense.

Theorem 3.16 (Approximation Theorem)**.** *Let $u \in H^2(\Omega)$. Then, for each $T \in \mathcal{T}$ there holds*

$$\|u - I_h u\|_{L^2(T)} \leq C\|h^2 D^2 u\|_{L^2(T)}, \tag{3.16}$$

$$\|\nabla(u - I_h u)\|_{L^2(T)} \leq C\sigma(T)\|hD^2 u\|_{L^2(T)}, \tag{3.17}$$

where the generic constant $C > 0$ does not depend on $u, \mathcal{T}$ and Ω. In particular, there holds

$$\|h^\alpha u - I_h u\|_{L^2(T)} \leq C\|h^{2+\alpha} D^2 u\|_{L^2(\Omega)}, \tag{3.18}$$

$$\|h^\alpha \nabla(u - I_h u)\|_{L^2(T)} \leq C\sigma(\mathcal{T})\|h^{1+\alpha} D^2 u\|_{L^2(T)}, \tag{3.19}$$

for all $\alpha \in \mathbb{R}$.

We will not go through the proof of this theorem because the proof needs some preparation and this would exceed the scope of this thesis. The proof can be found in [5]. With this Theorem we can prove the following corollary which states convergence of FEM with $\mathcal{T}$-piecewise affine functions.

Corollary 3.17. *If $u \in H^2(\Omega) \cap H_0^1(\Omega)$, then $I_h u \in \mathcal{S}_0^1(\mathcal{T})$ and*

$$\min_{v_h \in \mathcal{S}_0^1(\mathcal{T})} \|u - v_h\|_{H^1(\Omega)} \leq \|u - I_h u\|_{H^1(\Omega)} \leq C\sigma(\mathcal{T})\|hD^2 u\|_{L^2(\Omega)}, \tag{3.20}$$

where $C > 0$ depends only on Ω.

Proof. Let $u \in H^2(\Omega) \cap H_0^1(\Omega)$. Then $u(z) = 0$ for all $z \in \Gamma$ and thus $I_h u \in \mathcal{S}_0^1(\mathcal{T})$. With C_{apx} being the constant from Theorem 3.16 and C_F from Proposition 3.7 there holds

$$\begin{aligned}
\|u - I_h u\|_{H^1(\Omega)}^2 &= \|u - I_h u\|_{L^2(\Omega)}^2 + \|\nabla(u - I_h u)\|_{L^2(\Omega)}^2 \\
&\leq (C_F^2 + 1)\|\nabla(u - I_h u)\|_{L^2(\Omega)}^2 \\
&\leq C_{\text{apx}}^2 \sigma(\mathcal{T})^2 (C_F^2 + 1)\|hD^2 u\|_{L^2(\Omega)}
\end{aligned}$$

and therefore

$$\min_{v_h \in \mathcal{S}_0^1(\mathcal{T})} \|u - v_h\|_{H^1(\Omega)} \leq \|u - I_h u\|_{H^1(\Omega)} \leq C_{\text{apx}}\sigma(\mathcal{T})(C_F^2 + 1)^{1/2}\|hD^2 u\|_{L^2(\Omega)}.$$

□

For exact solutions $u \in H^2(\Omega)$ this corollary shows that FEM with $\mathcal{T}$-piecewise affine functions leads to convergence order $\mathcal{O}(h)$. Since $C_0^\infty(\overline{\Omega})$ is dense in $H_0^1(\Omega)$, Proposition 3.6 also provides convergence of FEM with $\mathcal{T}$-piecewise affine functions without any regularity assumptions on u.

3.4 A posteriori error estimation

In most applications one does not know the exact solution u of the PDE one wants to solve. Hence, one can not compute the error $\|u - u_h\|_{H^1(\Omega)}$ of the numerical solution. But for a lot of problems there exist numerically computable bounds $\eta = \eta(u_h, f, \mathcal{T})$ for the error which may depend on the numerical solution u_h, the triangulation $\mathcal{T}$ and the right hand side f. We call such a bound η an error estimator. One important property of an error estimator is reliability: There exists $C_{\text{rel}} > 0$ such that

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{\text{rel}}\eta. \tag{3.21}$$

This estimate ensures that one can make conclusions on the accuracy of the numerical solution with the value of the error estimator. However, it may occur that the numerical solutions u_h converge to u but η does not tend to zero. To avoid this one wants an error estimator which is also efficient: There exists $C_{\text{eff}} > 0$ such that

$$C_{\text{eff}}\eta \leq \|u - u_h\|_{H^1(\Omega)}. \tag{3.22}$$

In this section we consider a more general FEM space, namely

$$\mathcal{S}_0^p(\mathcal{T}) := \{v_h \in C(\Omega) \mid \forall T \in \mathcal{T} : v_h|_T \in \mathcal{P}^p(\Omega) \text{ and } v_h|_\Gamma = 0\}. \tag{3.23}$$

One error estimator for the Poisson problem 3.11 which is reliable and efficient is the so called residual error estimator. Let $\mathcal{T}$ be a conforming triangulation of Ω. For an interior facet E of $\mathcal{T}$, i.e. $E = T \cap T'$, $T, T' \in \mathcal{T}$ let n_E and n'_E be the outward pointing normal vectors on E with regard to ∂T and $\partial T'$ respectively. The jump of the normal derivative for a smooth function v across E is defined as $[\![\partial_n v]\!]|_E = \nabla v|_T \cdot n_E + \nabla v|_{T'} \cdot n'_E$.

Then the residual error estimator for the Dirichlet - Poisson problem (3.11) is defined by

$$\begin{aligned} \eta_h(T)^2 &= h_T^2 \|f + \Delta u_h\|^2_{L^2(T)} + h_T \|[\![\partial_n u_h]\!]\|^2_{L^2(\partial T \cap \Omega)}, \\ \eta_h &= \left(\sum_{T \in \mathcal{T}} \eta_h(T)^2 \right)^{1/2}. \end{aligned} \tag{3.24}$$

The following two propositions state the reliability and the efficiency of the residual error estimator.

Proposition 3.18 (Reliability)**.** *Let $\mathcal{T}$ be a conforming and γ-shape regular triangulation. Then there exists C_{rel} depending only on Ω and γ such that*

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{\text{rel}}\eta_h. \tag{3.25}$$

To proof this we need two lemmas. We will omit the proofs of them. They can be found in [5].

Lemma 3.19 (Clement-type operator)**.** *Let $\mathcal{T}$ be a conforming and γ-shape regular triangulation. We define J_h by*

$$J_h v := \sum_{z \in \mathcal{K} \setminus \Gamma} \left(\frac{1}{|\Omega[z]|} \int_{\Omega[z]} v dx \right) \zeta_z \quad \text{for } v \in H_0^1(\Omega), \tag{3.26}$$

where ζ_z are the hat functions from Proposition 3.12 and the patch $\Omega[z]$ of a node $z \in \mathcal{K}$ is defined by $\Omega[z] := \bigcup_{\{T \in \mathcal{T} | z \in T\}} T$.

Then $J_h : H_0^1(\Omega) \to \mathcal{S}_0^1(\mathcal{T})$ and for all $v \in H_0^1(\Omega)$ and $T \in \mathcal{T}$ there holds

$$\|v - J_h v\|_{L^2(T)} \leq C h_T \|\nabla v\|_{L^2(\Omega[T])}, \tag{3.27}$$

$$\|\nabla(v - J_h v)\|_{L^2(T)} \leq C \|\nabla v\|_{L^2(\Omega[T])}, \tag{3.28}$$

where $C > 0$ only depends on γ and the patch $\Omega[T]$ of an element $T \in \mathcal{T}$ is defined by $\Omega[T] := \bigcup_{\{T' \in \mathcal{T} | T' \cap T \neq \emptyset\}} T'$.

Lemma 3.20 (Trace inequality)**.** *Let $\mathcal{T}$ be a conforming and γ-shape regular triangulation. For $T \in \mathcal{T}$ and $v \in H^1(T)$ there holds*

$$\|v\|_{L^2(\partial T)}^2 \leq C \left(h_T^{-1} \|v\|_{L^2(T)}^2 + \|v\|_{L^2(T)} \|\nabla v\|_{L^2(T)} \right), \tag{3.29}$$

where $C > 0$ only depends on γ.

Proof of Proposition 3.18. With the Friedrichs inequality 3.12 we get that

$$\begin{aligned} \|u - u_h\|_{H^1(\Omega)} &\leq (C_F^2 + 1)^{1/2} \|\nabla(u - u_h)\|_{L^2(\Omega)} \\ &= (C_F^2 + 1)^{1/2} \frac{a(u - u_h, u - u_h)}{\|\nabla(u - u_h)\|_{L^2(\Omega)}} \\ &\leq (C_F^2 + 1)^{1/2} \sup_{v \in H_0^1(\Omega) \setminus \{0\}} \frac{a(u - u_h, v)}{\|\nabla v\|_{L^2(\Omega)}} \end{aligned}$$

Let $v \in H_0^1(\Omega) \setminus \{0\}$ and $v_h \in \mathcal{S}_0^p(\mathcal{T})$ then the Galerkin orthogonality 3.5 yields $a(u - u_h, v) = a(u - u_h, v - v_h)$. Now we show $a(u - u_h, v - v_h) \leq \eta_h \|\nabla v\|_{L^2(\Omega)}$ for an appropriate v_h. To abbreviate the notation we define $w := v - v_h$. With integration by parts, the Cauchy-Schwarz inequality and

since u is a solution of (3.13) we get that

$$\begin{aligned}
a(u-u_h,w) &= \langle f,w\rangle_{L^2(\Omega)} - a(u_h,w)\\
&= \int_\Omega f w dx - \sum_{T\in\mathcal{T}}\int_T \nabla u_h\cdot\nabla w dx\\
&= \int_\Omega f w dx - \sum_{T\in\mathcal{T}}\left(-\int_T \Delta u_h w dx + \int_{\partial T}\partial_n u_h w ds\right)\\
&= \sum_{T\in\mathcal{T}}\int_T (f+\Delta u_h) w dx + \sum_{T\in\mathcal{T}}\int_{\partial T\cap\Omega}\partial_n u_h w ds\\
&= \sum_{T\in\mathcal{T}}\int_T (f+\Delta u_h) w dx + \frac{1}{2}\sum_{T\in\mathcal{T}}\int_{\partial T\cap\Omega}[\![\partial_n u_h]\!] w ds\\
&= \sum_{T\in\mathcal{T}}\left(\langle f+\Delta u_h, w\rangle_{L^2(T)} + \frac{1}{2}\langle [\![\partial_n u_h]\!], w\rangle_{L^2(\partial T\cap\Omega)}\right)\\
&\le \sum_{T\in\mathcal{T}}\Bigg(h_T\|f+\Delta u_h\|_{L^2(T)} h_T^{-1}\|w\|_{L^2(T)}\\
&\qquad + h_T^{1/2}\|[\![\partial_n u_h]\!]\|_{L^2(\partial T\cap\Omega))}\frac{h_T^{-1/2}}{2}\|w\|_{L^2(\partial T\cap\Omega))}\Bigg)\\
&\le \eta_h\left(\sum_{T\in\mathcal{T}}\left(h_T^{-2}\|w\|^2_{L^2(T)} + \frac{h_T^{-1}}{4}\|w\|^2_{L^2(\partial T\cap\Omega))}\right)\right)^{1/2}
\end{aligned}$$

With J_h from Lemma 3.19 choose $v_h = J_h v$ then

$$\|w\|_{L^2(T)} = \|v - J_h v\|_{L^2(T)} \le C h_T \|\nabla v\|_{L^2(\Omega[T])}.$$

And with the trace inequality (3.29) there holds

$$\begin{aligned}
\|w\|^2_{L^2(\partial T\cap\Omega)} &\le C\left(h_T^{-1}\|v-J_h v\|^2_{L^2(T)} + \|v-J_h v\|_{L^2(T)}\|\nabla(v-J_h v)\|_{L^2(T)}\right)\\
&\le C h_T\|\nabla v\|^2_{L^2(\Omega[T])}.
\end{aligned}$$

In both previous estimates C is a generic constant which only depends on γ and these estimates yield

$$\sum_{T\in\mathcal{T}}\left(h_T^{-2}\|w\|^2_{L^2(T)} + \frac{h_T^{-1}}{4}\|w\|^2_{L^2(\partial T\cap\Omega))}\right) \le C\sum_{T\in\mathcal{T}}\|\nabla v\|^2_{L^2(\Omega[T])} \le C\|\nabla v\|^2_{L^2(\Omega)}.$$

Here, we used that for a conforming triangulation the overlap of different element patches is finite. This last estimate concludes the proof. □

Proposition 3.21 (Efficiency)**.** *Let $\mathcal{T}$ be a conforming and γ-shape regular triangulation. Define* $\operatorname{osc}_h := (\sum_{T\in\mathcal{T}}\operatorname{osc}(T)^2)^{1/2}$ *where*

$$\operatorname{osc}_h(T)^2 = h_T^2 \min_{f_T\in\mathcal{P}^{p-1}(T)}\|f-f_T\|^2_{L^2(T)}.$$

Then there exists a constant C_{eff} depending only on γ and the polynomial degree p such that

$$\eta_h \leq C_{\text{eff}}\big(\|u-u_h\|^2_{H^1(\Omega)} + \operatorname{osc}_h^2\big)^{1/2}. \tag{3.30}$$

Remark 3.22. This proposition shows efficiency for the residual error estimator but with oscillations. However, if f is sufficiently smooth the oscillations will decay faster than the error if the mesh size h_T tends to zero.

We will need the following lemma to prove the efficiency. We omit the proof of this lemma. It can be found in [5].

Lemma 3.23 (Inverse estimate)**.** *For all $m,k,r \in \mathbb{N}$ with $k > r$ exists a constant $C > 0$ such that for all $v_h \in \mathcal{P}^m(\mathcal{T}) := \{v_h : \Omega \to \mathbb{R} \mid \forall T \in \mathcal{T} : v_h|_T \in \mathcal{P}^m(T)\}$ and for all $T \in \mathcal{T}$ there holds*

$$\|D^k v_h\|_{L^2(T)} \leq C\sigma(\mathcal{T}) h_T^{r-k} \|D^r v_h\|_{L^2(T)}. \tag{3.31}$$

Proof of Proposition 3.21. If constants appear in any estimates in this proof we will omit them and use the symbol $\lesssim$ instead of $\leq$. Note that any hidden constants in this proof at most depend on γ and the polynomial degree p.

First step: For $T \in \mathcal{T}$ we define the element bubble function

$$b_T := \prod_{z \in \mathcal{K}_T} \zeta_z \in H^1_0(\Omega) \cap \mathcal{P}^3(T).$$

With a scaling argument one gets that $\|v\|_{L^2(T)} \simeq \|v b_T^{1/2}\|_{L^2(T)}$ are equivalent norms on $\mathcal{P}^q(T)$, $q \in \mathbb{N}_0$ with constants depending only on q and the reference element. Now let $q \geq p-2$ and Π_T be defined by $\Pi_T f \in \mathcal{P}^q(T)$ such that

$$\|f - \Pi_T f\|_{L^2(T)} = \min_{g_T \in \mathcal{P}^q(T)} \|f - g_T\|_{L^2(T)} \quad \text{for all } f \in L^2(T),$$

i.e. $\Pi_T : L^2(T) \to \mathcal{P}^q(T)$ is the orthogonal projection.

Now set $f_T := \Pi_T(f + \Delta u_h)$. We note that since $\Delta u_h \in \mathcal{P}^{p-2}(T) \subset \mathcal{P}^q(T)$, there holds $\Pi_T \Delta u_h = \Delta u_h$ and thus

$$f_T - (f + \Delta u_h) = f - \Pi_T f. \tag{3.32}$$

The equivalence $\|v\|_{L^2(T)} \simeq \|v b_T^{1/2}\|_{L^2(T)}$ on $\mathcal{P}^q(T)$ yields

$$\begin{aligned} \|f_T\|^2_{L^2(T)} &\lesssim \|f_T b_T^{1/2}\|^2_{L^2(T)} \\ &= \langle f_T, f_T b_T \rangle_{L^2(\Omega)} \\ &= \big(\langle f_T - (f+\Delta u_h), f_T b_T\rangle_{L^2(\Omega)} \\ &\qquad + \langle (f+\Delta u_h), f_T b_T\rangle_{L^2(\Omega)}\big). \end{aligned} \tag{3.33}$$

With $u \in H_0^1(\Omega)$ being the solution of the weak form (3.13), with integration by parts, the Cauchy-Schwarz inequality and the inverse estimate (3.31) there holds

$$\begin{aligned}\langle f + \Delta u_h, f_T b_T\rangle_{L^2(\Omega)} &= \langle \nabla u, \nabla(f_T b_T)\rangle_{L^2(\Omega)} + \langle \Delta u, f_T b_T\rangle_{L^2(\Omega)} \\ &= \langle \nabla(u - u_h), \nabla(f_T b_T)\rangle_{L^2(\Omega)} \\ &\leq \|\nabla(u - u_h)\|_{L^2(T)} \|\nabla(f_T b_T)\|_{L^2(T)} \\ &\lesssim h_T^{-1} \|\nabla(u - u_h)\|_{L^2(T)} \|f_T b_T\|_{L^2(T)} \\ &\leq h_T^{-1} \|\nabla(u - u_h)\|_{L^2(T)} \|f_T\|_{L^2(T)}.\end{aligned} \tag{3.34}$$

Moreover, the Cauchy-Schwarz inequality and (3.32) yield

$$\begin{aligned}\langle f_T - (f + \Delta u_h), f_T b_T\rangle_{L^2(\Omega)} &\leq \|f_T - (f + \Delta u_h)\|_{L^2(\Omega)} \|f_T b_T\|_{L^2(\Omega)} \\ &= \|f - \Pi_T f\|_{L^2(\Omega)} \|f_T b_T\|_{L^2(\Omega)} \\ &\leq \|f - \Pi_T f\|_{L^2(\Omega)} \|f_T\|_{L^2(\Omega)}\end{aligned} \tag{3.35}$$

Plugging (3.34) and (3.35) into (3.33) yields

$$\|f_T\|_{L^2(T)} \lesssim C \left(h_T^{-1} \|\nabla(u - u_h)\|_{L^2(T)} + \|f - \Pi_T f\|_{L^2(T)} \right), \tag{3.36}$$

With (3.32) and (3.33) there holds

$$\begin{aligned}h_T \|f + \Delta u_h\|_{L^2(T)} &\leq h_T \|f_T\|_{L^2(T)} + h_T \|f + \Delta u_h - f_T\|_{L^2(T)} \\ &= h_T \|f_T\|_{L^2(T)} + h_T \|f - \Pi_T f\|_{L^2(T)} \\ &\lesssim \left(\|\nabla(u - u_h)\|_{L^2(T)} + h_T \|f - \Pi_T f\|_{L^2(T)} \right).\end{aligned} \tag{3.37}$$

Second step: Let $T \in \mathcal{T}$ and E an edge of T. Consider the reference element $T_{\text{ref}} = \{(0,0), (1,0), (0,1)\}$ and $E_{\text{ref}} = \{(0,0), (0,1)\}$. For $\underline{r} \in \mathcal{P}^q(T_{\text{ref}})$, define $\widehat{\underline{r}} \in \mathcal{P}^q(T_{\text{ref}})$ by $\widehat{\underline{r}}(s,t) = \underline{r}(s,0)$, then $\widehat{\underline{r}}|_{E_{\text{ref}}} = \underline{r}|_{E_{\text{ref}}}$.

Since $\mathcal{P}^q(T_{\text{ref}})$ is finite dimensional, $\|\underline{r}\|_{L^2(E_{\text{ref}})} \simeq \|\underline{r}\| := \|\widehat{\underline{r}}\|_{L^2(T_{\text{ref}})}$ are equivalent norms on $\mathcal{P}^q(T_{\text{ref}})$.

Let $\Phi_T : T_{\text{ref}} \to T$ be an affine diffeomorphism such that $\Phi_T(E_{\text{ref}}) = E$. For $r \in \mathcal{P}^q(T)$, choose $\widehat{r} := \widehat{r \circ \Phi_T} \circ \Phi_T^{-1}$. Clearly, $r|_E = \widehat{r}|_E$ and with the Transformation theorem we get

$$\begin{aligned}\|\widehat{r}\|_{L^2(T)}^2 &= \|\widehat{r \circ \Phi_T} \circ \Phi_T^{-1}\|_{L^2(T)}^2 \\ &= |\det(D\Phi_T)| \|\widehat{r \circ \Phi_T}\|_{L^2(T_{\text{ref}})}^2 \\ &= \frac{|T|}{|T_{\text{ref}}|} \|r \circ \Phi_T\|^2\end{aligned}$$

and

$$\|r\|_{L^2(E)}^2 = |E| \|r \circ \Phi_T\|_{L^2(E_{\text{ref}})}^2 \simeq |E| \|r \circ \Phi_T\|^2$$

This yields

$$\|\widehat{r}\|^2_{L^2(T)} \simeq \frac{|T|}{|E||T_{\mathrm{ref}}|}\|r\|^2_{L^2(E)} \simeq h_T\|r\|^2_{L^2(E)}.$$

Third step: Let $E = T \cap T'$ be an interior edge, $T, T' \in \mathcal{T}$. We define the edge bubble function

$$b_E := \prod_{z \in \mathcal{K}_T \cap \mathcal{K}_{T'}} \zeta_z \in H^1_0(T \cup T') \cap \mathcal{P}^2(\{T, T'\})$$

A scaling argument shows that $\|v\|_{L^2(E)} \simeq \|v b_E^{1/2}\|_{L^2(E)}$ are equivalent norms on $\mathcal{P}^{p-1}(E)$ where the constants only depend on γ and the reference element.

Define $r := [\![\partial_n u_h]\!]|_E \in \mathcal{P}^{p-1}(E)$. Then, according to step 2 there exists $\widehat{r} \in \mathcal{P}^{p-1}(T, T')$ with $\widehat{r}|_E = r$ and

$$\begin{aligned} \|\widehat{r}\|_{L^2(T)} &\simeq h_T^{1/2}\|r\|_{L^2(E)}, \\ \|\widehat{r}\|_{L^2(T')} &\simeq h_{T'}^{1/2}\|r\|_{L^2(E)}. \end{aligned} \tag{3.38}$$

Moreover, $\widehat{r}$ is continuous and hence $\widehat{r} b_E \in H^1_0(T \cup T') \cap \mathcal{P}^{p+1}(\{T, T'\})$.

With the previous norm equivalence, integration by parts, the weak form (3.13), the Cauchy-Schwarz inequality, the inverse estimate (3.31) and (3.38) we get that

$$\begin{aligned} \|[\![\partial_n u_h]\!]\|^2_{L^2(E)} &\lesssim \|\widehat{r} b_E^{1/2}\|_{L^2(E)} \\ &= \langle [\![\partial_n u_h]\!], \widehat{r} b_E\rangle_{L^2(E)} \\ &= \langle \partial_n u_h, \widehat{r} b_E\rangle_{L^2(\partial T)} + \langle \partial_n u_h, \widehat{r} b_E\rangle_{L^2(\partial T')} \\ &= \langle \nabla u_h, \nabla(\widehat{r} b_E)\rangle_{L^2(T)} + \langle \Delta u_h, \widehat{r} b_E\rangle_{L^2(T)} + \\ &\qquad \langle \nabla u_h, \nabla(\widehat{r} b_E)\rangle_{L^2(T)} + \langle \Delta u_h, \widehat{r} b_E\rangle_{L^2(T)} \\ &= \langle \nabla(u_h - u), \nabla(\widehat{r} b_E)\rangle_{L^2(\Omega)} + \langle f + \Delta u_h, \widehat{r} b_E\rangle_{L^2(\Omega)} \\ &\le \|\nabla(u_h - u)\|_{L^2(T\cup T')}\|\nabla(\widehat{r} b_E)\|_{L^2(T\cup T')} + \\ &\qquad \|f + \Delta u_h\|_{L^2(T\cup T')}\|\widehat{r} b_E\|_{L^2(T\cup T')} \\ &\lesssim \|\nabla(u_h - u)\|_{L^2(T\cup T')} h_T^{-1}\|\widehat{r} b_E\|_{L^2(T\cup T')} + \\ &\qquad \|f + \Delta u_h\|_{L^2(T\cup T')}\|\widehat{r} b_E\|_{L^2(T\cup T')} \\ &\le \|\widehat{r}\|_{L^2(T\cup T')}\big(h_T^{-1}\|\nabla(u_h - u)\|_{L^2(T\cup T')} + \\ &\qquad \|f + \Delta u_h\|_{L^2(T\cup T')}\big) \\ &\lesssim h_T^{1/2}[\![\partial_n u_h]\!]\|_{L^2(E)}\big(h_T^{-1}\|\nabla(u_h - u)\|_{L^2(T\cup T')} + \\ &\qquad \|f + \Delta u_h\|_{L^2(T\cup T')}\big). \end{aligned}$$

This estimate implies

$$\begin{aligned} \|[\![\partial_n u_h]\!]\|_{L^2(E)} \lesssim \|\nabla(u_h - u)\|_{L^2(T\cup T')} + h_T\|\nabla(u_h - u)\|_{L^2(T)} + \\ h_{T'}\|\nabla(u_h - u)\|_{L^2(T')} \end{aligned} \tag{3.39}$$

Fourth step: We combine the first and the third step. For $T \in \mathcal{T}$ there holds

$$\begin{aligned}
\eta_h(T)^2 &= h_T^2 \, \|f + \Delta u_h\|_{L^2(T)}^2 + h_T \, \|[\![\partial_n u_h]\!]\|_{L^2(\partial T \cap \Omega)}^2 \\
&= h_T^2 \, \|f + \Delta u_h\|_{L^2(T)}^2 + \sum_{E \in \mathcal{E}_T \cap \mathcal{E}_\Omega} h_T \, \|[\![\partial_n u_h]\!]\|_{L^2(E)}^2 \\
&\lesssim h_T^2 \, \|f + \Delta u_h\|_{L^2(T)}^2 + \sum_{\substack{T' \in \mathcal{T} \setminus \{T\} \\ T' \cap T \neq \emptyset}} \Big(\|\nabla(u - u_h)\|_{L^2(T \cup T')}^2 + \\
&\qquad\qquad h_T^2 \, \|f + \Delta u_h\|_{L^2(T)}^2 + \\
&\qquad\qquad h_{T'}^2 \, \|f + \Delta u_h\|_{L^2(T')}^2 \Big) \\
&\lesssim \sum_{\substack{T' \in \mathcal{T} \\ T' \cap T \neq \emptyset}} \|\nabla(u - u_h)\|_{L^2(T')}^2 + h_{T'}^2 \|f + \Delta u_h\|_{L^2(T')}^2 \\
&\lesssim \sum_{\substack{T' \in \mathcal{T} \\ T' \cap T \neq \emptyset}} \|\nabla(u - u_h)\|_{L^2(T')}^2 + h_{T'}^2 \|f - \Pi_{T'} f\|_{L^2(T')}^2.
\end{aligned}$$

Since $\mathcal{T}$ is conforming, $\#\{T' \in \mathcal{T} \mid T' \cap T \neq \emptyset\}$ is finite and thus

$$\eta_h^2 = \sum_{T \in \mathcal{T}} \eta_h(T)^2 \lesssim \|u - u_h\|_{H^1(\Omega)}^2 + \operatorname{osc}_h^2.$$

□

Chapter 4

Error estimation with neural networks

The first experiment in this thesis is to check whether a machine learning algorithm can learn to predict the error estimator for the numerical solution on a given mesh. To this end we make some numerical experiments. The idea is to generate a set of random meshes on which a PDE is solved numerically and the error of this approximate solution is estimated. This is done with a standard Galerkin finite element method and the so called residual error estimator. With data of this type we train a neural network and then compare the error estimates which are predicted by the trained neural network with the error estimates determined by the residual error estimator to assess the performance of the machine learning model.

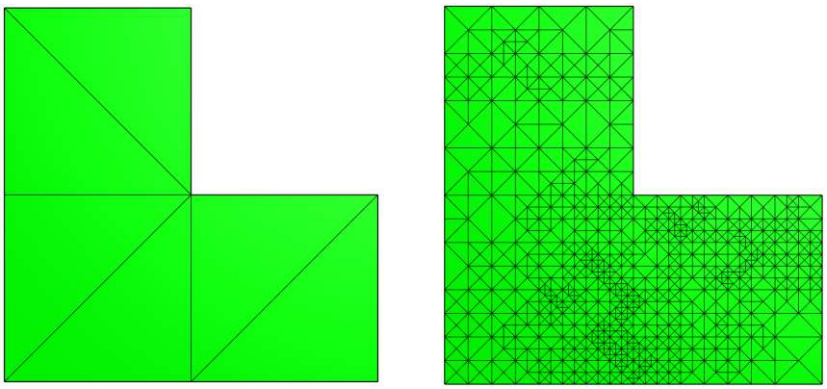
4.1 Setting

4.1.1 Meshes

To train the neural network one needs training data. For us this means different meshes. The idea to generate these different meshes is to refine a given initial mesh in different ways. To this end we refine the initial mesh 2, 3, 4 or 5 times. For each mesh this number of refinements is chosen randomly. Which elements are refined in each refinement step is also chosen randomly. The only restriction for this meshes is that the number of elements is less than or equal to 1,000.

In the numerical experiments we use two different kind of domains. The first domain is an L-shape domain. In Figure 4.1 one can see the initial mesh on this domain and one example of these random generated meshes.

The second type of domain is a somewhat random domain. We consider 8 points: $(0,0)$, $(1,0)$, $(2,0)$, $(2,1)$, $(2,2)$, $(1,2)$, $(0,2)$, $(0,1)$. The convex hull of these points is, of course, a square. Now we add a different perturbation



(a) (b)

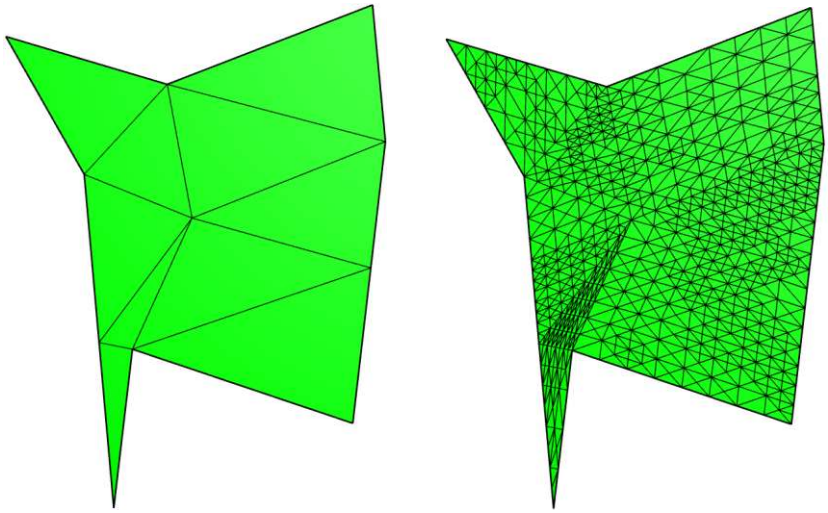
Figure 4.1: Initial and randomly refined mesh on L-shape geometry

vector $p = (p_1, p_2)$ to each point. The entries p_1 and p_2 are uniformly distributed in $(-0.5, 0.5)$. With this procedure one obtains random domains. One example of such a domain with the initial mesh and an example of a random mesh is pictured in Figure 4.2.

4.1.2 Neural network

We use a neural network with four dense layers and the ReLU activation function. The output dimension of the first layer is 1,000, of the second layer 100, of the third layer 10 and of the last layer 1.

The input varies in the different numerical experiments. There are four cases. The first case is that only the coordinates of the vertices of the elements are given to the neural network. If the mesh has 1,000 elements the input dimension is 6,000 because there are three vertices per element and two coordinates per vertex. If the mesh has less than 1,000 elements the missing entries to reach an input dimension of 6,000 are filled up with zeros. In the second case the input also contains the value of the right hand side f at every vertex. Whereas in the third case the input also contains the value of the discrete solution $u_{\mathcal{T}}$ at every vertex. The fourth case is the combination of the second and the third case. Hence, in the second and third case the dimension of the input is 9,000 and in the last case it is 12,000. Let $\boldsymbol{x}_i = (x_{i,1}, x_{i,2})^T, \boldsymbol{y}_i = (y_{i,1}, y_{i,2})^T, \boldsymbol{z}_i = (z_{i,1}, z_{i,2})^T$ be the coordinates of the vertices of the i-th element and k the index of the last element. Then



(a) (b)

Figure 4.2: Initial and randomly refined mesh on random geometry

the four different input vectors look like this:

$$6\,000\left\{\overbrace{\begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{y}_1\\ \boldsymbol{z}_1\\ \vdots\\ \boldsymbol{x}_k\\ \boldsymbol{y}_k\\ \boldsymbol{z}_k\\ 0\\ \vdots\\ 0\end{pmatrix}}^{\text{case 1}}\right., 9\,000\left\{\overbrace{\begin{pmatrix}\boldsymbol{x}_1\\ f(\boldsymbol{x}_1)\\ \boldsymbol{y}_1\\ f(\boldsymbol{y}_1)\\ \boldsymbol{z}_1\\ f(\boldsymbol{z}_1)\\ \vdots\\ \boldsymbol{x}_k\\ f(\boldsymbol{x}_k)\\ \boldsymbol{y}_k\\ f(\boldsymbol{y}_k)\\ \boldsymbol{z}_k\\ f(\boldsymbol{z}_k)\\ 0\\ \vdots\\ 0\end{pmatrix}}^{\text{case 2}}\right., \overbrace{\begin{pmatrix}\boldsymbol{x}_1\\ u_h(\boldsymbol{x}_1)\\ \boldsymbol{y}_1\\ u_h(\boldsymbol{y}_1)\\ \boldsymbol{z}_1\\ u_h(\boldsymbol{z}_1)\\ \vdots\\ \boldsymbol{x}_k\\ u_h(\boldsymbol{x}_k)\\ \boldsymbol{y}_k\\ u_h(\boldsymbol{y}_k)\\ \boldsymbol{z}_k\\ u_h(\boldsymbol{z}_k)\\ 0\\ \vdots\\ 0\end{pmatrix}}^{\text{case 3}}, \overbrace{\left\{\begin{pmatrix}\boldsymbol{x}_1\\ f(\boldsymbol{x}_1)\\ u_h(\boldsymbol{x}_1)\\ \boldsymbol{y}_1\\ f(\boldsymbol{y}_1)\\ u_h(\boldsymbol{y}_1)\\ \boldsymbol{z}_1\\ f(\boldsymbol{z}_1)\\ u_h(\boldsymbol{z}_1)\\ \vdots\\ \boldsymbol{x}_k\\ f(\boldsymbol{x}_k)\\ u_h(\boldsymbol{x}_k)\\ \boldsymbol{y}_k\\ f(\boldsymbol{y}_k)\\ u_h(\boldsymbol{y}_k)\\ \boldsymbol{z}_k\\ f(\boldsymbol{z}_k)\\ u_h(\boldsymbol{z}_k)\\ 0\\ \vdots\\ 0\end{pmatrix}\right\}}^{\text{case 4}}\right\}12\,000$$

To train the neural network we use 4,000 different meshes which are generated as described in 4.1.1. Then the neural network is tested with another 1,000 of these meshes.

As loss function we use the mean squared relative error

$$\frac{1}{n}\sum_{i=0}^{n}\left(\frac{\eta_i - \eta_i^*}{\eta_i}\right)^2 \tag{4.1}$$

where n is the number of meshes, η_i is the residual error estimator of mesh $\mathcal{T}_i$ and η_i^* is the corresponding error estimator predicted by the neural network.

The optimizer used is the Adam optimizer from 2.2.3 and it gets 10 epochs to train the neural network. An epoch in machine learning refers to a complete iteration through the entire training dataset.

4.2 Results

For the numerical experiments we use two different right hand sides f. The first one is $f(x, y) = x$ and the second one is a random discrete function defined on the nodes of the mesh. The values are uniformly distributed between 0 and 1.

For each of the above described settings we build 20 independent neural networks and train them with the same 4,000 training meshes. Then we determine the accuracy of the predictions of these neural networks using the 1,000 test meshes by calculating the mean squared relative error as stated in (4.1). Then we compare the mean accuracy of each of these settings.

Since the training of one such neural network takes approximately 30 seconds for the smallest input and 70 seconds for the largest input we only trained 20 networks for each scenario. Therefore the sample size is rather low and consequently the calculated mean accuracy is not very reliable but it shows some tendencies.

In the Tables 4.1, 4.2, 4.3 and 4.4 we can see the accuracies for the four different inputs described in 4.1.2 and 20 independently trained neural networks. The corresponding histograms are pictured in Figures 4.3, 4.4, 4.5 and 4.6.

4.2.1 L-shape domain, fixed right hand side

For the L-shape domain and the right hand side $f(x, y) = x$ the accuracies are almost the same for all four inputs and on average the discrepancy between the predicted error estimates and the actual ones is within one percent of the value of the actual error estimates (see Table 4.1 and Figure 4.3). This might be explained by the network learning only the fixed vector of error estimates.

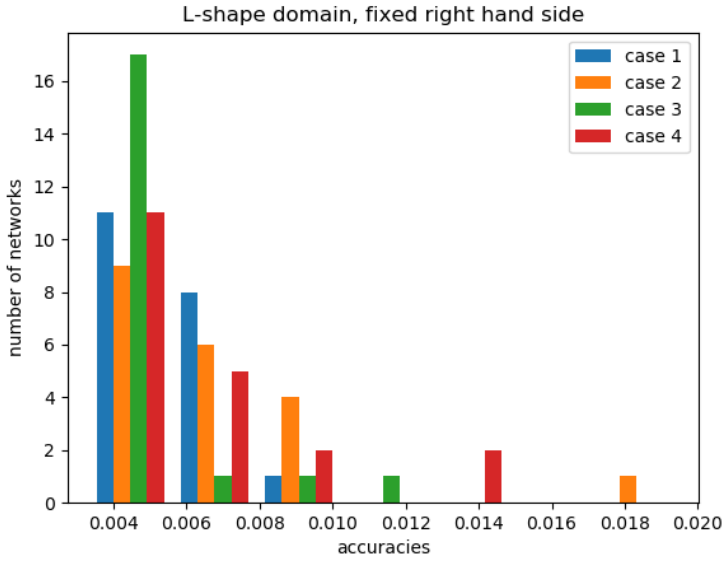


Figure 4.3: Histogram of accuracies for L-shape domain and fixed right hand side for the four different inputs.

4.2.2 L-shape domain, random right hand side

For a random right hand side (see Table 4.2 and Figure 4.4) the neural networks which only got the coordinates as input perform worse than the others and overall the predictions are more inaccurate than with the fixed right hand side. The deviation between the predicted estimate and the actual ones is in the scale of three percent.

4.2.3 Random domain, fixed right hand side

If the domain is random the performance for all inputs decrease (see Table 4.3 and Figure 4.5). This is not surprising, because it seems easier to learn the error estimator for a domain which stays the same than for changing domains. And we see that the neural networks with inputs including the numerical solution outperform the networks with the other two inputs.

4.2.4 Random domain, random right hand side

The case where not only the domain is random but also the right hand side f shows a similar behaviour as with the fixed right hand side. And in the best case which is the input including the numerical solution the error between predicted error estimates and actual ones is in the scale of 5 percent. This can be seen in Table 4.4 and Figure 4.6.

input	accuracies				mean accuracies
coordinates of elements	0.00551	0.00615	0.00511	0.00534	0.00586
	0.00533	0.00445	0.00500	0.00678	
	0.00694	0.00556	0.00602	0.00592	
	0.00492	0.00485	0.00527	0.00760	
	0.00904	0.00507	0.00585	0.00659	
coordinates of elements + RHS	0.00625	0.00458	0.00464	0.00628	0.00701
	0.00656	0.00943	0.00537	0.00835	
	0.00537	0.00651	0.00538	0.00652	
	0.00507	0.01946	0.00632	0.00938	
	0.00896	0.00516	0.00549	0.00510	
coordinates of elements + numerical solution	0.00388	0.00844	0.01170	0.00362	0.00508
	0.00401	0.00455	0.00517	0.00358	
	0.00670	0.00389	0.00421	0.00559	
	0.00481	0.00331	0.00399	0.00461	
	0.00459	0.00403	0.00541	0.00546	
coordinates of elements + RHS and numerical solution	0.00540	0.01431	0.00605	0.01331	0.00654
	0.00472	0.00476	0.00500	0.00641	
	0.00531	0.00447	0.00508	0.00494	
	0.00421	0.00476	0.00587	0.00565	
	0.00727	0.00925	0.00890	0.00518	

Table 4.1: Accuracies for L-shape domain and fixed right hand side.

4.2.5 Conclusion

Summing up, we see that for a fixed domain there is only little difference between the inputs. If the domain is random the neural networks with inputs including the numerical solution outperform the others. In the last, most interesting case the neural networks with the coordinates and the numerical solution as input were able to predict the error estimates best. The deviation in this case is about 5 percent of the actual estimate.

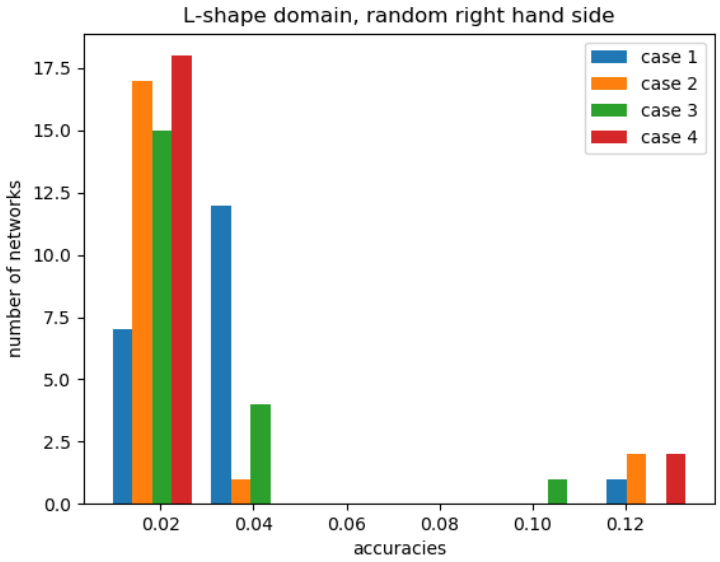


Figure 4.4: Histogram of accuracies for L-shape domain and a random right hand side for the four different inputs.

input	accuracies				mean accuracies
coordinates of elements	0.03098	0.13516	0.02785	0.03612	0.03557
	0.02928	0.03771	0.03112	0.02846	
	0.02918	0.02839	0.02878	0.03081	
	0.03064	0.03059	0.02897	0.02786	
	0.02685	0.02848	0.03386	0.03029	
coordinates of elements + RHS	0.02603	0.13515	0.00867	0.13515	0.02647
	0.01304	0.01420	0.01500	0.00753	
	0.00831	0.01011	0.01620	0.01995	
	0.00962	0.03114	0.00888	0.00854	
	0.01164	0.01101	0.02500	0.01416	
coordinates of elements + numerical solution	0.02879	0.02086	0.02039	0.03433	0.02847
	0.01969	0.02018	0.01980	0.03091	
	0.02448	0.02199	0.11304	0.02156	
	0.02259	0.02029	0.02175	0.02901	
	0.02576	0.02172	0.02894	0.02334	
coordinates of elements + RHS and numerical solution	0.02108	0.01136	0.00785	0.01020	0.02431
	0.00940	0.01229	0.13517	0.01986	
	0.01389	0.00904	0.00800	0.00999	
	0.13514	0.01087	0.01020	0.01539	
	0.01384	0.00904	0.00922	0.01438	

Table 4.2: Accuracies for L-shape domain and random right hand side.

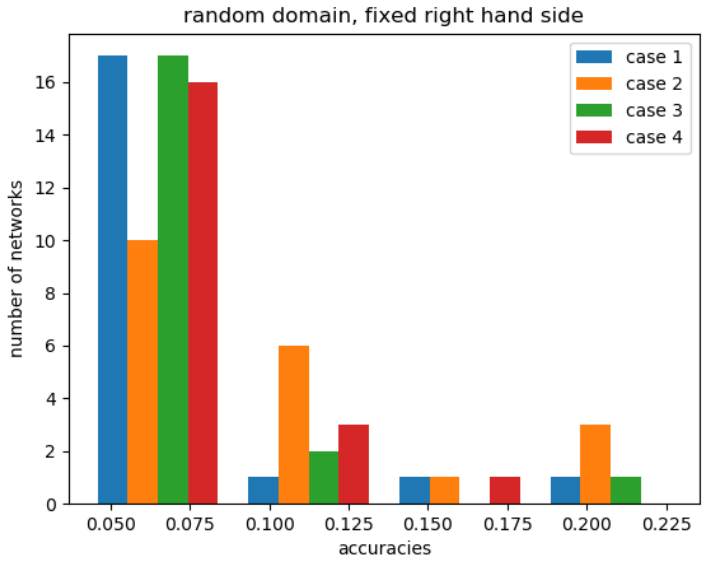


Figure 4.5: Histogram of accuracies for a random domain and fixed right hand side for the four different inputs.

input	accuracies				mean accuracies
coordinates of elements	0.08465	0.07705	0.15690	0.07965	0.08849
	0.06955	0.07601	0.09266	0.07790	
	0.07177	0.07158	0.07842	0.08013	
	0.07357	0.07596	0.07120	0.07647	
	0.07851	0.08786	0.20293	0.08696	
coordinates of elements + RHS	0.07713	0.22418	0.22080	0.07166	0.10963
	0.18212	0.07910	0.08017	0.10320	
	0.09427	0.07220	0.08853	0.07513	
	0.19977	0.08237	0.09370	0.11119	
	0.09092	0.07805	0.08975	0.07833	
coordinates of elements + numerical solution	0.04630	0.04198	0.07403	0.04116	0.06261
	0.04336	0.05442	0.04385	0.04640	
	0.06140	0.04491	0.09272	0.04442	
	0.23123	0.05223	0.04424	0.04323	
	0.08870	0.05448	0.04918	0.05400	
coordinates of elements + RHS and numerical solution	0.05795	0.04677	0.04476	0.05636	0.06569
	0.09167	0.04902	0.04519	0.04721	
	0.06967	0.04757	0.04575	0.11555	
	0.05457	0.05346	0.04714	0.12041	
	0.04123	0.04826	0.18346	0.04777	

Table 4.3: Accuracies for random domain and fixed right hand side.

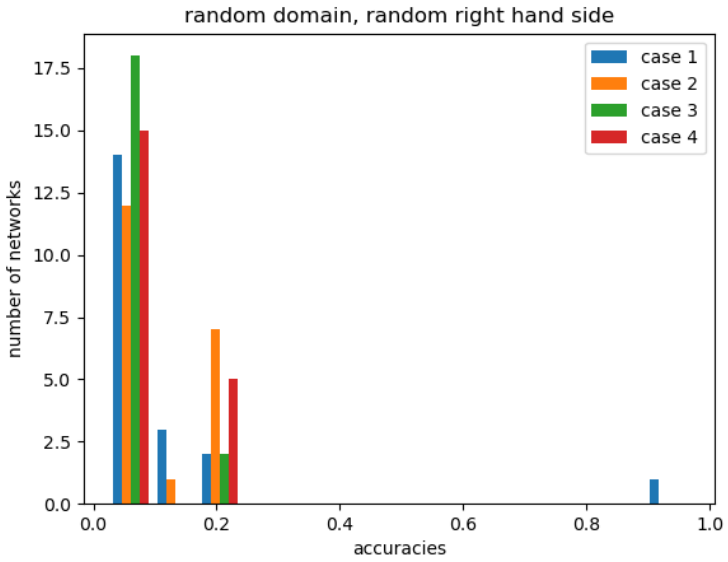


Figure 4.6: Histogram of accuracies for a random domain and a random right hand side for the four different inputs.

input	accuracies				mean accuracies
coordinates of elements	0.05582	0.06871	0.06474	0.96866	0.12873
	0.06387	0.07371	0.06346	0.08016	
	0.17318	0.06858	0.09838	0.08261	
	0.17425	0.13993	0.06321	0.05825	
	0.09951	0.05839	0.06128	0.05800	
coordinates of elements + RHS	0.05708	0.17868	0.04399	0.04344	0.10043
	0.19151	0.17322	0.17561	0.05561	
	0.04497	0.07048	0.04175	0.17599	
	0.04283	0.04795	0.17318	0.17314	
	0.04504	0.04435	0.06803	0.16173	
coordinates of elements + numerical solution	0.17362	0.03639	0.02767	0.03412	0.05815
	0.03180	0.07107	0.18508	0.03713	
	0.04753	0.04186	0.07590	0.08415	
	0.02827	0.05760	0.03384	0.03383	
	0.04002	0.03774	0.04970	0.03565	
coordinates of elements + RHS and numerical solution	0.03125	0.17360	0.02895	0.02958	0.07596
	0.02504	0.02386	0.04882	0.09483	
	0.02361	0.03651	0.04327	0.17649	
	0.04503	0.05392	0.09379	0.03567	
	0.03500	0.17315	0.17327	0.17355	

Table 4.4: Accuracies for random domain and random right hand side.

Chapter 5

AFEM with NN

The objective of AFEM with machine learning is to utilize a neural network to process a given mesh, denoted as $\mathcal{T}$, and the corresponding numerical solution $u_{\mathcal{T}}$ on this mesh, in order to obtain a set of marked elements M. Let's denote such a neural network as $\mathcal{A}$. The output is then determined by $M = \mathcal{A}(w, \mathcal{T}, u_h)$, where w represents the vector of learnable parameters as in Section 2.2.3.

To evaluate the quality of the obtained marking set, it is necessary to define a loss function $\mathcal{L}$ that takes a marking set as input and calculates a numerical value, $\mathcal{L}(M)$, representing the quality of the set of marked elements. The goal is to minimize this loss function by adjusting the weights w of the neural network. In formal terms, we aim to minimize the mapping $w \to \mathcal{L}(\mathcal{A}(w, \mathcal{T}, u_h))$.

The output of the neural network has only two values for each element of the mesh: either an element is marked for refinement or not. Only slight changes in the weights can lead to a different set of marked elements. So the mapping $w \to \mathcal{A}(w, \mathcal{T}, u_h)$ is not continuous and therefore $w \to \mathcal{L}(\mathcal{A}(w, \mathcal{T}, u_h))$ is not continuous either. And thus traditional learning methods as stated in Section 2.2.3 are not suitable for optimizing this type of neural network. In the following, we will attempt to develop an approach to minimize the loss function and evaluate its performance through numerical experiments.

5.1 Loss function

Let $\mathcal{T}$ be a mesh and $M \subset \mathcal{T}$ be a marking set. Let $\mathcal{T}_{\text{uniform}}$ and $\mathcal{T}_M$ be the uniformly refined mesh and the mesh refined according to M, respectively. Furthermore, let $u_{\mathcal{T}_{\text{uniform}}} \in \mathcal{S}^1_0(\mathcal{T}_{\text{uniform}})$ and $u_{\mathcal{T}_M} \in \mathcal{S}^1_0(\mathcal{T}_M)$ be the numerical solutions on these meshes. Then, we define the loss function $\mathcal{L}$ of a set of marked elements M by

$$\mathcal{L}(M) := \left\| \nabla \left(u_{\mathcal{T}_{\text{uniform}}} - u_{\mathcal{T}_M} \right) \right\|_{L^2(\Omega)} . \tag{5.1}$$

5.2 Optimizer

To optimize the weights we consider two approaches. These two are a random search and a method similar to gradient descent but with an approximated gradient.

5.2.1 Random search

The idea of the random search is that one adds a random perturbation to the weights and then checks whether the neural network with the modified weights performs better. This is done iteratively:

Algorithm 1 Random Search

Require: neural network model $\mathcal{A}$, loss function $\mathcal{L}$, initial weights w, mesh $\mathcal{T}$, numerical solution $u_{\mathcal{T}}$

```
L_min ← L(A(w, T, u_T))
for i = 0 to N do
    w_tmp ← w
    for j = 0 to M do
        w_new ← w_tmp + random vector
        L_new = L(A(w_new, T, u_T))
        if L_new < L_min then
            w ← w_new
            L_min ← L_new
        end if
    end for
end for
```

To validate this method we perform some numerical experiments. We consider the Poisson equation (3.11) with right hand side $f(x, y) = xy$ on an L-shape domain with a given mesh $\mathcal{T}$ (see Figure 5.1) and we use the Algorithm 1 to minimize the loss on that mesh. As number of iterations we use $N = 40$ and as number of directions $M = 30$. We use a neural network with two layers. As input we use the third case from Section 4.1.2, i.e. the input contains the coordinates of the nodes of the elements and the values of the numerical solution at these nodes. Hence, the input dimension is nine times the number of elements. The first layer is a dense layer with the ReLU activation function and the output dimension is 1.5 times the number of elements. The second layer is a dense layer with the softmax activation function. The output of this layer is a probability distribution with one value for each element. Then the set of marked elements consists of the $\#\mathcal{T}/3$ elements with the highest value. This restriction is necessary since the optimal refinement in regards to the accuracy of the numerical solution would be to refine all elements.

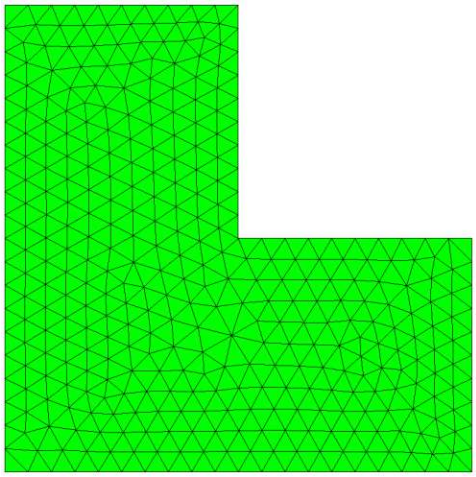
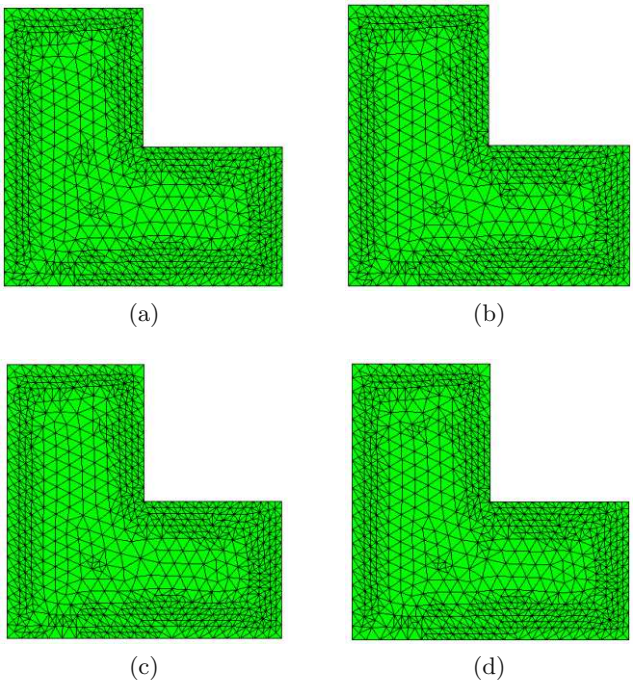


Figure 5.1: Initial mesh for assessment of optimizers



(a) (b) (c) (d)

Figure 5.2: Refined meshes determined with random search

In Figure 5.2 we see the meshes which are refined according to the marking set provided by 4 independently optimized neural networks. One can see that mostly the elements on the boundary are refined which is not really what we would expect because a strong refinement towards the reentrant corner would be expected.

5.2.2 Gradient descent with approximated gradient

If one supposes that the loss function $\mathcal{L}$ and the neural network $\mathcal{A}$ are continuous, then with Taylor expansion one gets

$$\mathcal{L}(\mathcal{A}(w+\delta,\mathcal{T},u_{\mathcal{T}})) \approx \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) + \nabla_w \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}}))\cdot\delta \tag{5.2}$$

for $w,\delta \in \mathbb{R}^{M(\mathcal{A})}$. Let's denote $\nabla_w \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) \in \mathbb{R}^{M(\mathcal{A})}$ with X and let $\delta_1,\delta_2,\ldots,\delta_n \in \mathbb{R}^{M(\mathcal{A})}$ be random vectors. Then with (5.2) one gets the following system of equations

$$\begin{aligned}
\mathcal{L}(\mathcal{A}(w+\delta_1,\mathcal{T},u_{\mathcal{T}})) &= \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) + X\cdot\delta_1\\
\mathcal{L}(\mathcal{A}(w+\delta_2,\mathcal{T},u_{\mathcal{T}})) &= \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) + X\cdot\delta_2\\
&\vdots\\
\mathcal{L}(\mathcal{A}(w+\delta_n,\mathcal{T},u_{\mathcal{T}})) &= \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) + X\cdot\delta_n
\end{aligned} \tag{5.3}$$

With

$$B = \begin{pmatrix} \mathcal{L}(\mathcal{A}(w+\delta_1,\mathcal{T},u_{\mathcal{T}})) - \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) \\ \mathcal{L}(\mathcal{A}(w+\delta_2,\mathcal{T},u_{\mathcal{T}})) - \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) \\ \vdots \\ \mathcal{L}(\mathcal{A}(w+\delta_n,\mathcal{T},u_{\mathcal{T}})) - \mathcal{L}(\mathcal{A}(w,\mathcal{T},u_{\mathcal{T}})) \end{pmatrix} \in \mathbb{R}^n \tag{5.4}$$

and

$$D = \begin{pmatrix} \delta_1^T \\ \delta_2^T \\ \vdots \\ \delta_n^T \end{pmatrix} \in \mathbb{R}^{n\times M(\mathcal{A})} \tag{5.5}$$

(5.3) can be rewritten as

$$DX = B. \tag{5.6}$$

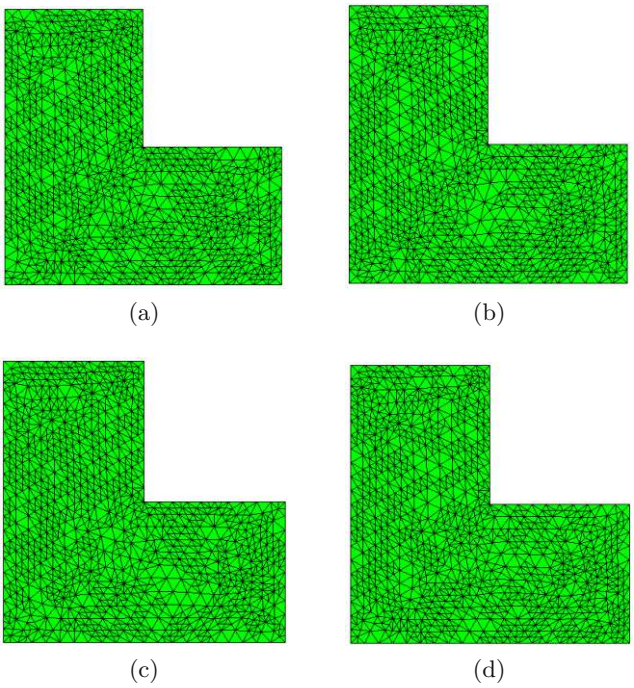
It can be assumed that $M(\mathcal{A}) > n$ and hence, there does not exist a unique solution of (5.6) but one solution is given by $X = D^T Y$ with Y being the solution of

$$DD^T Y = B. \tag{5.7}$$

This can be verified if we insert $X = D^T(DD^T)^{-1}B$ in (5.6):

$$DD^T(DD^T)^{-1}B = B \tag{5.8}$$

This leads to



(a) (b) (c) (d)

Figure 5.3: Refined meshes determined with approximated gradient descent

Algorithm 2 Random Search

Require: neural network model $\mathcal{A}$, loss function $\mathcal{L}$, initial weights w

for $i = 0$ to N **do**

$$D \leftarrow \begin{pmatrix} \delta_1^T \\ \delta_2^T \\ \vdots \\ \delta_n^T \end{pmatrix} \qquad \triangleright \text{ for } i = 1, \ldots, n\text{: } \delta_i \in R^{M(\mathcal{A})} \text{ random vector}$$

$$B \leftarrow \begin{pmatrix} \mathcal{L}(\mathcal{A}(w+\delta_1, \mathcal{T}, u_\mathcal{T})) - \mathcal{L}(\mathcal{A}(w, \mathcal{T}, u_\mathcal{T})) \\ \mathcal{L}(\mathcal{A}(w+\delta_2, \mathcal{T}, u_\mathcal{T})) - \mathcal{L}(\mathcal{A}(w, \mathcal{T}, u_\mathcal{T})) \\ \vdots \\ \mathcal{L}(\mathcal{A}(w+\delta_n, \mathcal{T}, u_\mathcal{T})) - \mathcal{L}(\mathcal{A}(w, \mathcal{T}, u_\mathcal{T})) \end{pmatrix}$$

$Y \leftarrow (DD^T)^{-1}B$

$X \leftarrow D^T Y$

$w \leftarrow w - \frac{1}{\|X\|} X$

end for

To assess the quality of this algorithm we make the same experiments as in 5.2.1. In figure 5.3 we see that the results are also not ideal and more research is required. Different network architectures and more training time might lead to better results.

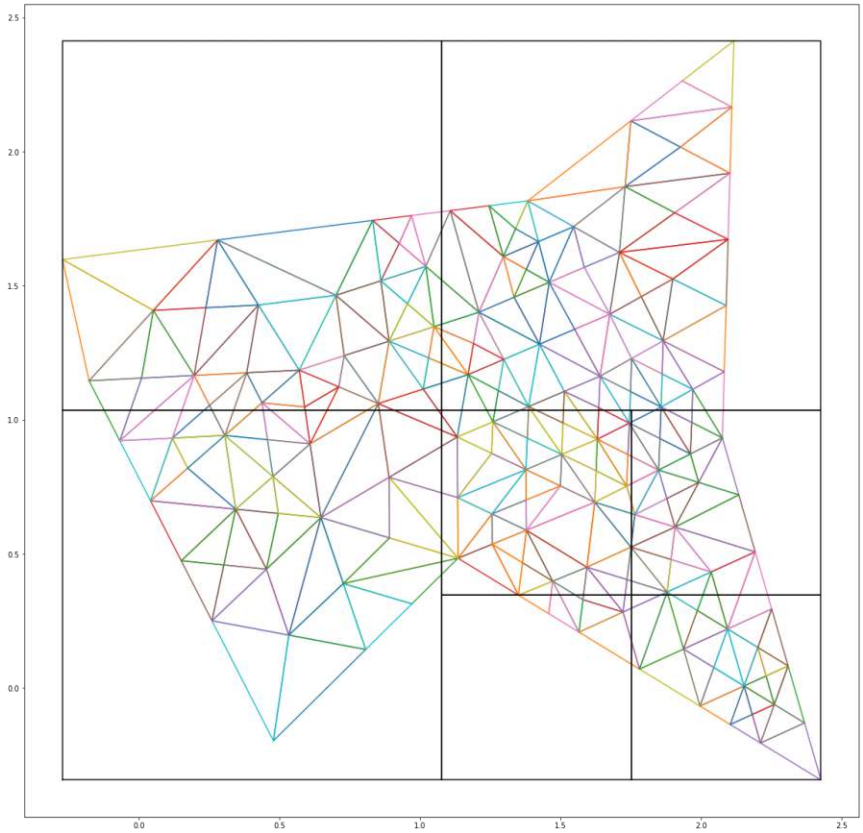


Figure 5.4: Partition of a random mesh with $n_{\max} = 100$

5.3 AFEM on arbitrary meshes

In the previous section we assessed the performance of the optimizers for one given mesh. But as described in the introduction the goal of including machine learning into AFEM is that one finds a neural network model which is able to determine the right elements to refine for arbitrary meshes and PDEs. In this section we will discuss one approach to tackle this problem.

The idea of this approach is that we partition a mesh into submeshes with a given maximal number of elements. One each of these submeshes a neural network determines the elements which should be refined. The reason for this partitioning is that we only need one neural network with a fixed number of inputs. If the mesh gets finer and therefore the number of elements increase the number of partitions also increase but we can use the same neural network for each of this partition.

Let Ω be domain, $\mathcal{T}$ a triangulation on this domain and $n_{\max} \in \mathcal{N}$ the maximum number of elements in one partition. To partition such a mesh we first check if the number of elements $\#\mathcal{T}$ is less than or equal to $n_{\max}$.

If this is the case then we are done. Otherwise, we determine the bounding rectangle of Ω and split it in into four rectangles of the same size. This yields 4 regions of the domain, let's denote them with $\Omega_1, \dots, \Omega_4$. Now we go through all the elements of $\mathcal{T}$ and assign them to the submeshes $\mathcal{T}_1, \dots, \mathcal{T}_4$ if the element is in the corresponding subdomain. Note that one element can be assigned to more than one submesh. Now we continue iteratively: for each of the submeshes we do the same as for the original mesh. This procedure terminates once we partitioned the mesh such that in every submesh there are less than $n_{\max}$ elements. (cf. Figure 5.4)

We use the same neural network as described in 5.2.1 but with input dimension $9n_{\max}$ and output dimension $n_{\max}$. This neural network is used on each of the parts in the partitioned mesh. If an element is in more than one submesh then the mean values of the predictions are used. With these predictions again the $\mathcal{T}/3$ elements with the highest predicted value are put in the marking set. With this marking set one can calculate the loss function from Section 5.1.

To create the training data data we use the same approach as in Section 4.1.1. We create n_{meshes} random domains, refine them randomly and then calculate the numerical solution of the Poisson problem on these meshes. This yields a set of random meshes and their solutions. We denote this set of meshes with $\mathbb{T} = \{\mathcal{T}_1, \dots, \mathcal{T}_{n_{\text{meshes}}}\}$ and the corresponding set of numerical solutions with $U = \{u_1, \dots, u_{n_{\text{meshes}}}\}$.

The loss function for such a set of meshes is now given by

$$\mathcal{L} := \frac{1}{n} \sum_{k=1}^{n_{\text{meshes}}} \mathcal{L}_k$$

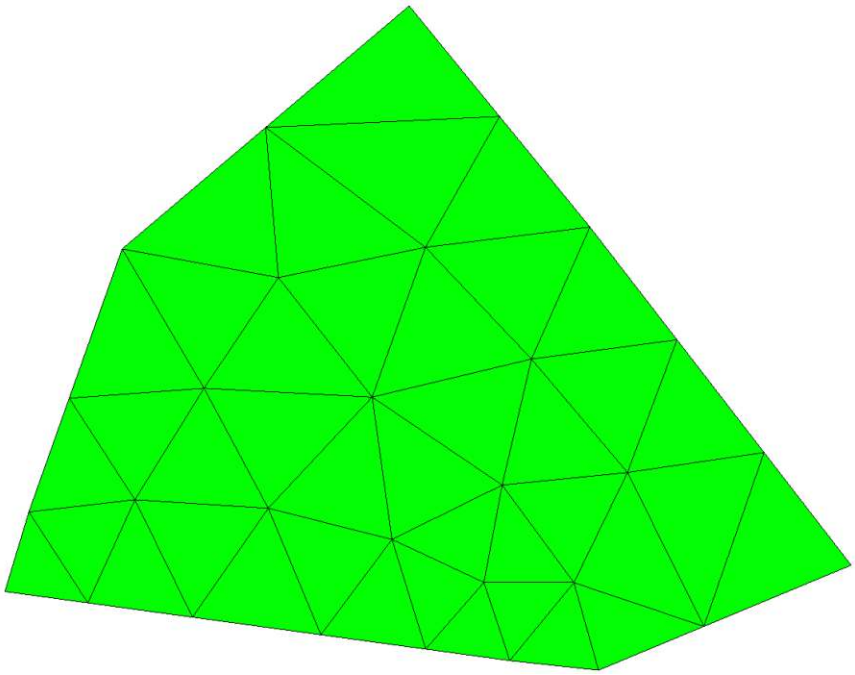
where $\mathcal{L}_k$ is the loss given in Section 5.1 for the k-th mesh.

To test if this approach works we train a neural network as described above with 30 training meshes and the random search from Section 5.2.1 with 20 iterations and 30 directions. The reason why the number of meshes is rather small is that the computation of the loss is pretty expensive. Even for only 30 meshes one calculation of the loss takes about 7 seconds. And therefore the training altogether takes approximately 70 minutes.

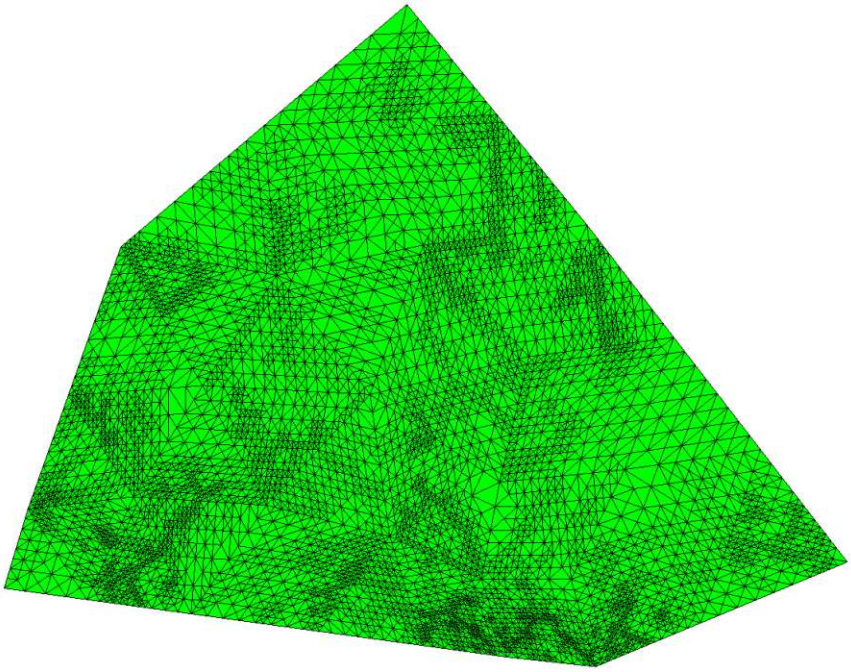
Eventually, we use this trained neural network in an adaptive algorithm which follows the steps described in the Introduction but uses the neural network to determine the elements which should be refined. This process is iterated until the number of degrees of freedoms of the mesh exceed 3,000.

Two examples of such refined meshes are pictured in Figures 5.5 and 5.6. In Figure 5.5 we see that the refinement looks kind of random. Figure 5.6 shows some refinement towards reentrant corners which would be expected but the overall result is not ideal either. Therefore, we see that this approach does not work well. Maybe tuning of the neural network, a larger training set or more training iterations might improve the performance of the network.

This is not done here because of the lack of computational power.

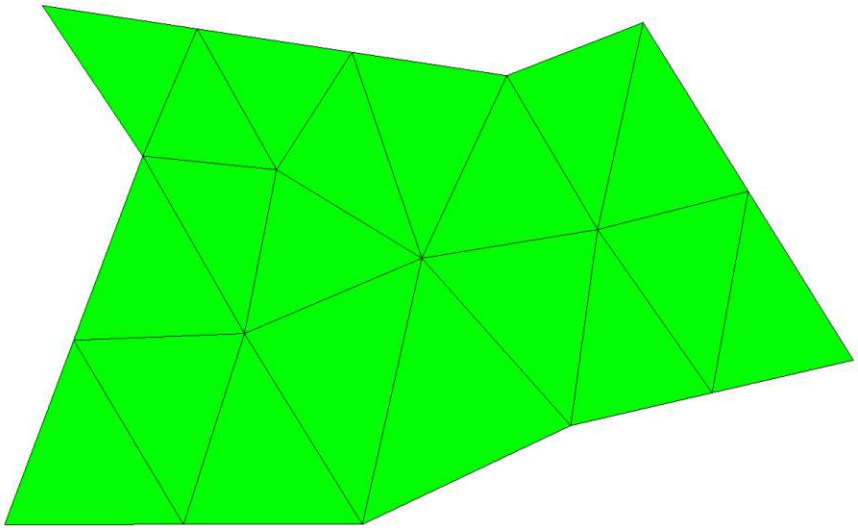


(a)

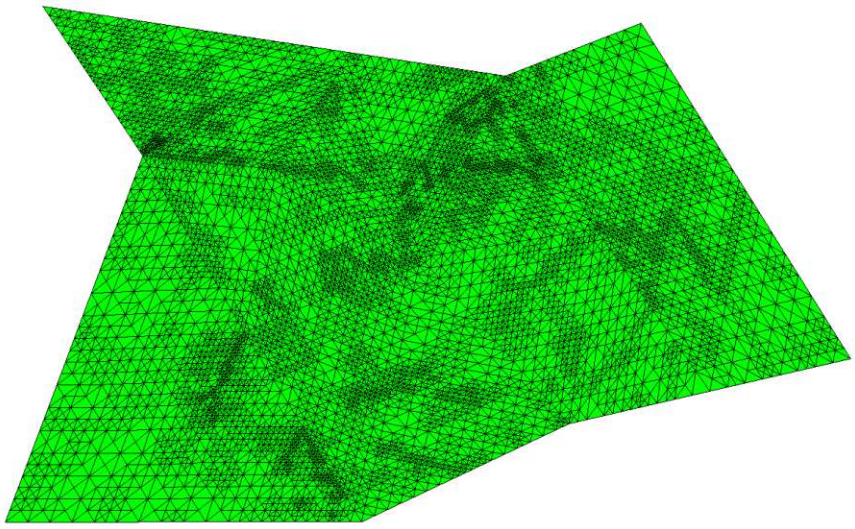


(b)

Figure 5.5: Initial and adaptively refined mesh 1



(a)



(b)

Figure 5.6: Initial and adaptively refined mesh 2

Bibliography

[1] Jan Bohn and Michael Feischl. "Recurrent neural networks as optimal mesh refinement strategies". In: *Computers & Mathematics with Applications* 97 (2021), pp. 61–76.

[2] Leon Bottou. "Online learning and stochastic approximations". In: *Online learning in neural networks* 17.9 (1998), p. 142.

[3] Léon Bottou and Olivier Bousquet. "The tradeoffs of large scale learning". In: *Advances in neural information processing systems* 20 (2007).

[4] Carsten Carstensen, Michael Feischl, Marcus Page, and Dirk Praetorius. "Axioms of adaptivity". In: *Computers & Mathematics with Applications* 67.6 (2014), pp. 1195–1253.

[5] Michael Feischl and Dirk Praetorius. *Numerics of Partial Differential Equations: Stationary Problems*. Lecture Notes. 2020.

[6] Neha Gupta et al. "Artificial neural network". In: *Network and Complex Systems* 3.1 (2013), pp. 24–28.

[7] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[8] Krzysztof C Kiwiel. "Convergence and efficiency of subgradient methods for quasiconvex minimization". In: *Mathematical programming* 90 (2001), pp. 1–25.

[9] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2.

[10] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[11] Joost AA Opschoor, Philipp C Petersen, and Christoph Schwab. "Deep ReLU networks and high-order finite element methods". In: *Analysis and Applications* 18.05 (2020), pp. 715–770.

[12] Philipp Petersen and Felix Voigtlaender. "Optimal approximation of piecewise smooth functions using deep ReLU neural networks". In: *Neural Networks* 108 (2018), pp. 296–330.

[13] David Saad. *On-line learning in neural networks*. Cambridge University Press, 1999.

[14] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. PMLR. 2013, pp. 1139–1147.

[15] Matus Telgarsky. "Representation benefits of deep feedforward networks". In: *arXiv preprint arXiv:1509.08101* (2015).

[16] Dmitry Yarotsky. "Error bounds for approximations with deep ReLU networks". In: *Neural Networks* 94 (2017), pp. 103–114.

List of Figures

List of Tables