Quantification of Errors Generated By Uncertain Data in a Linear Boundary Value Problem Using Neural Networks*

Vilho Halonen† and Ilkka Pölönen†

Abstract. Quantifying errors caused by indeterminacy in data is currently computationally expensive even in relatively simple PDE problems. Efficient methods could prove very useful in for example scientific experiments done with simulations. In this paper, we create and test neural networks which quantify uncertainty errors in the case of a linear one-dimensional boundary value problem. Training and testing data is generated numerically. We created three training datasets, three testing datasets and trained four neural networks with differing architectures. The performance of the neural networks is compared to known analytical bounds of errors caused by uncertain data. We find that the trained neural networks accurately approximate the exact error quantity in almost all cases and the neural network outputs are always between the analytical upper and lower bounds. The results of this paper show that after a suitable dataset is used for training even a relatively compact neural network can succesfully predict quantitative effects generated by uncertain data. If these methods can be extended to more difficult PDE problems they could potentially have a multitude of real-world applications.

Key words. PDE, Uncertainty Quantification, Machine Learning, Neural Network

MSC codes. 65G99, 68T01, 65L10

1. Introduction.

Mathematical models in real-world applications are always burdened by uncertainty in measurements. Solutions to real-world problems are usually created by numerical methods such as the finite element method [2] and the data used is never exactly known. Uncertainty in data causes errors in the outputs of mathematical models. Currently in PDE problems the methods used to quantify these errors include propabilistic methods [10], the worst case scenario method [5] and a posteriori error estimates [6].

There are two major issues with current methods. Analytical methods are not always available and even if they exist the results may be too coarse. Numerical methods can suffer from unreasonably high computational costs. If a solution seems good enough an engineer may disregard uncertainty error estimation completely.

The problem of uncertainty analysis is close to the area of scientific research where modern machine learning models such as neural networks are used. In this paper, we investigate the possibility of using neural networks to quantify errors generated by uncertain data. First we create datasets which have as inputs the uncertainty in data and as outputs the uncertainty error represented by some error metric. These datasets are used to train various different neural networks. To analyse effects of uncertain data we chose an example differential equation which is presented in section 3. For the chosen problem, analytical error bounds were derived in [4].

The uncertainty information is not a discrete set of values which can be directly input to a neural network so we need to characterize the uncertainty in an approximated way. This is

*Submitted to the editors 07.12.2022.

†University of Jyväskylä, Faculty of Information Technology. Mattilanniemi 2, 40100, Jyvaskylä, Finland.

done by scanning the set of possible values for parameters on some amount of specific points. The prototype problem we use has uncertain parameter functions which are one-dimensional, so the scanning is done on the edge of the maximum and minimum values of a function in the indeterminacy set. In our case, we used eight points for this scanning operation (see Figure 2).

The paper is organized as follows. First we introduce the uncertainty problem in an abstract setting in section 2. In section 3 we show the prototype problem used, methods to create datasets, the neural network models and how network performance is evaluated. Results are given in section 4, and the conclusions follow in section 6.

2. Problem Setting.

Consider an abstract differential problem

$$\mathcal{A}u = f, \tag{2.1}$$

where $u \in V$ is the unknown function in some suitable solution space, $\mathcal{A}$ is a differential operator and f is the source term. In practice, the differential operator $\mathcal{A}$ and source term f depend on measured data which is subject to inaccuracies in measurements. For example, if 2.1 were a linear elasticity problem the operator $\mathcal{A}$ would contain information about the Láme parameters of the deformed material and f would describe the outside forces subjected to the material. Such measured quantities are only known up to some accuracy. A solution to 2.1 is therefore not unique but rather there is a set of possible solutions.

Assume that instead of knowing $\mathcal{A}$ and f exactly we only know that

$$(\mathcal{A}, f) \in \mathcal{D} := \mathcal{D}_{\mathcal{A}} \times \mathcal{D}_f. \tag{2.2}$$

We call the set $\mathcal{D}$ the *set of admissible data*. Each element of the set $\mathcal{D}$ may produce a different solution u to the problem 2.1. For this purpose we define the set of solutions which is the image of the following solution mapping.

Definition 2.1. *The solution mapping $\mathcal{S} : \mathcal{D} \to V$ is a function such that $\mathcal{S}(\mathcal{A}, f) = u$, where u is the exact solution to 2.1. The image $\mathcal{S}(\mathcal{D})$ is called the solution set.*

If possible, we would like to know quantitatively what the solution set $\mathcal{S}(\mathcal{D})$ is like. For this purpose the quantity of interest is the diameter of this set with respect to some suitable norm.

Definition 2.2. *The diameter of the solution set is*

$$Diam(\mathcal{S}(\mathcal{D})) := \sup_{u_1, u_2 \in \mathcal{S}(\mathcal{D})} ||u_1 - u_2||,$$

where the norm $||\cdot||$ is a suitable norm in the vector space V.

The quantity $\mathrm{Diam}(\mathcal{S}(\mathcal{D}))$ is useful for two things. Firstly, if there is a priori knowledge of what this quantity is expected to be, the accuracy of measurements related to $\mathcal{D}$ can be considered suitable or unsuitable. This quantity can also give us an accuracy limit on approximation errors. If approximation errors are lower then the diameter the approximation may already be within the solution set so sharpening approximation schemes is a wasted effort (see Figure 1). Our research objective is to create neural networks which use the indeterminacy set $\mathcal{D}$ as input and outputs an approximation of the diameter $\mathrm{Diam}(\mathcal{S}(\mathcal{D}))$.

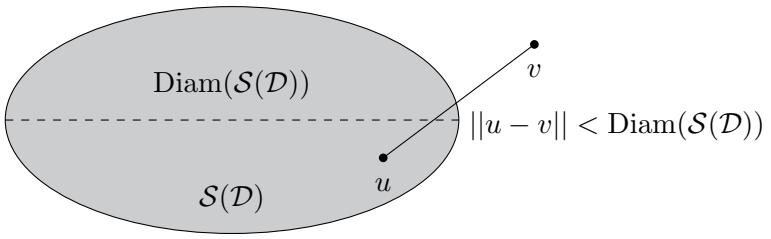

Figure 1. *Since approximation error is less then the diameter sharpening the approximation scheme is not useful.*

3. Methods. We investigate the effects of uncertain data in the context of a prototype problem which is presented in section 3.1. Our process of creating neural network models consists of three distinct steps. First, we describe and implement creation of training and test data in section 3.2. Afterwards we create and train neural networks on the made data in section 3.4. In section 3.5, we explain how network performance is tested.

3.1. Prototype problem. The effects of indeterminacy are studied in the case of the following boundary value problem. Problem $\mathcal{P}$: Find $u \in H_0^1(0,1)$, such that

$$(\alpha(x)u'(x))' - \beta(x)u(x) = f(x), \tag{3.1}$$

$$u(0) = u(1) = 0, \tag{3.2}$$

$$0 < \alpha_\ominus < a(x) < \alpha_\oplus, \quad 0 < \beta_\ominus < \beta(x) < \beta_\oplus, \quad f \in L^2(0,1). \tag{3.3}$$

Uncertainty in (3.1)-(3.3) is introduced by assuming that α, β and f are not known exactly. Instead we only know that

$$(\alpha, \beta, f) \in \mathcal{D} = \mathcal{D}_\alpha \times \mathcal{D}_\beta \times \mathcal{D}_f,$$

where

$$\begin{aligned}
\mathcal{D}_\alpha &:= \{\alpha_\circ + \delta_1 g \ : \ ||g||_\infty \leq 1\}, \\
\mathcal{D}_\beta &:= \{\beta_\circ + \delta_2 g \ : \ ||g||_\infty \leq 1\}, \\
\mathcal{D}_f &:= \{f_\circ + \delta_3 g \ : \ ||g||_\infty \leq 1\}.
\end{aligned}$$

The functions $\alpha_\circ$, $\beta_\circ$ and $f_\circ$ are referred to as the "mean" or "central" elements of the admissible data and the functions δ_1, δ_2 and δ_3 are the maximum perturbations from them. The image of the solution mapping (see 2.1) is in the space $H_0^1(0,1)$, so $\mathcal{S} : \mathcal{D} \to H_0^1(0,1)$. In this space the suitable norm that we use for the diameter Diam($\mathcal{S}(\mathcal{D})$) (see 2.2) is the energy norm

$$||u||_\circ := \left(\int_0^1 \alpha_\circ |u'|^2 + \beta_\circ |u|^2 \, dx \right)^{\frac{1}{2}}, \tag{3.4}$$

where $u_\circ = \mathcal{S}(\alpha_\circ, \beta_\circ, f_\circ)$.

For this one-dimensional problem the input for a neural network is formed by scanning each of the sets $\mathcal{D}_\alpha, \mathcal{D}_\beta$ and $\mathcal{D}_f$ on the edge of what the maximum and minimum function in these sets are (see Figure 2) on eight equidistant points on the domain $[0,1]$.

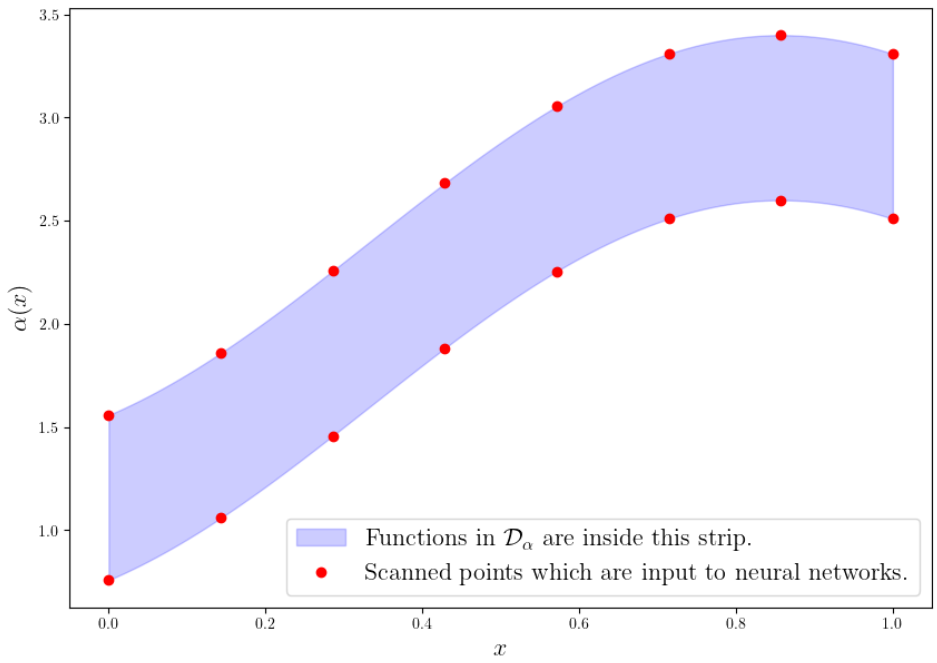

Figure 2. *Example of how the set $\mathcal{D}_\alpha$ is turned into a discrete input for a neural network. The y-coordinate of each red dot is put into a vector of length 16 in a chosen order. This vector is part of the neural network input together with similar vectors scanned from the sets $\mathcal{D}_\beta$ and $\mathcal{D}_f$.*

3.2. Creating Data. We created three training datasets and three testing datasets. The training datasets consist of examples where the admissible datasets $\mathcal{D}_\alpha, \mathcal{D}_\beta$ and $\mathcal{D}_f$ have a piecewise constant maximum and minimum element (see Figure 4) Examples in the made test datasets are less restricted in terms of smoothness and oscillation. Rather then being piecewise constant on their edges the edge is smooth and can oscillate (see Figure 3) more and more for the subsequent test datasets. This will let us see how the networks generalize when trained with a fairly constrained dataset. It is also noticeably easier and faster to generate training examples with tighter constraints. In our case, the piecewise constraints make it so that different solutions in the solution set can be found by solving relatively small linear systems of equations which is done very quickly on modern computers.

First the method of generating a single example is discussed and in the following subsections the specific characteristics of each dataset are presented. A training example consists of the input which is a discrete version of the admissible dataset $\mathcal{D}$ and the output A which is an approximation of the quantity $\mathrm{Diam}(\mathcal{S}(\mathcal{D}))$. Discrete inputs are created by scanning the admissible dataset on eight points on the domain (see Figure 2).

To approximate the diameter we randomly select a finite amount of elements from the admissible dataset and compute the solution for each choice. From the resulting finite solution set the diameter can be approximated by brute force calculation of the maximum energy distance between two elements. In our case, the finite set was chosen to have 2000 solutions. In testing different sizes it was found that after around size 1500 the diameter approximation

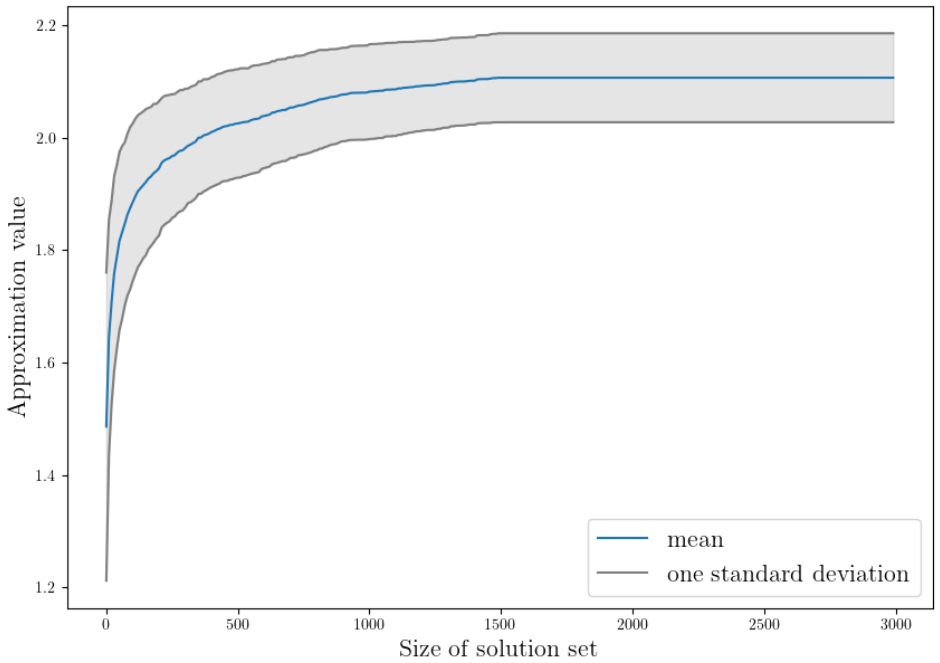

Figure 3. *Diameter approximation for solution sets of different sizes. The approximation was computed 200 times for the different sizes of solution sets* $[10, 20, \cdots, 3000]$*. After 1500 solutions are reached the approximation did not change so when making training examples we use 2000 solutions to approximate the diameter.*

tends to not increase (see Figure 3). To be safer we go a little beyond and choose 2000. Note however that if we are unlucky even with such an amount of solutions the diameter approximation may be inaccurate. For our purposes it is sufficiently accurate.

It is not trivial to decide what the random elements chosen from $\mathcal{D}$ should be. If they are too complex generating solutions may be slow. If they are too simple the finite set will not be a good representation of $\mathcal{D}$. For the training datasets we chose piecewise constant functions. For such functions the solutions can be generated very fast. The specifics of how the problem (3.1)-(3.3) becomes a linear system when using piecewise constant functions α, β and f are described in 3.2.1. The reason we chose to use these piecewise datasets for training is to see if a somewhat limited type of training data can train a network such that it generalizes to more complex functions. If the PDE at hand were more complicated it may provide a substantial increase in computational cost to make a training dataset that is somehow limited in complexity.

After the finite solution set has been generated the diameter is calculated by brute force computation which compares each pair of solutions in the solution set and computes the energy distance between them. The maximum distance found is the approximated diameter which is used as the output of the training example. This procedure is rather computationally expensive and takes almost all of the time used to generate a single example.

3.2.1. Training Datasets. In all of the datapoints in the training datasets the mean functions $\alpha_\circ$, $\beta_\circ$, $f_\circ$ and the indeterminacy functions δ_i are piecewise constant such that

$$\begin{aligned} \alpha_\circ(x) &= \alpha_k, \text{ when } x \in [x_{k-1}, x_k], \\ \beta_\circ(x) &= \beta_k, \text{ when } x \in [x_{k-1}, x_k], \\ f_\circ(x) &= f_k, \text{ when } x \in [x_{k-1}, x_k], \\ \delta_i(x) &= \delta_i^k, \text{when } x \in [x_{k-1}, x_k], \ i = 1, 2, 3. \end{aligned} \tag{3.5}$$

The amount of subintervals is the same for the first two datasets and increased for the last dataset. The indeterminacy functions have a magnitude limit given by

$$\delta_1(x) \leq 0.3\alpha_\circ(x), \quad \delta_2(x) \leq 0.3\beta_\circ(x), \quad \delta_3(x) \leq 0.3f_\circ(x). \tag{3.6}$$

From the admissible dataset with the above constraints we pick 2000 individual solutions to (3.1)-(3.3) by sampling random functions α, β and f from the admissible dataset. In every sample the functions α, β and f are also chosen to be piecewise constant. This lets us generate each solution in this set of 2000 solutions by solving linear system of equations. How this is done is described as follows:

Assume that the functions α, β, f are constant on the subintervals (x_i, x_{i+1}), $i = 0, N$.

$$\begin{aligned} \alpha(x) &= \alpha_k, \text{ when } x \in [x_{k-1}, x_k], \\ \beta(x) &= \beta_k, \text{ when } x \in [x_{k-1}, x_k], \\ f(x) &= f_k, \text{ when } x \in [x_{k-1}, x_k]. \end{aligned}$$

A differential equation of the form

$$\alpha u'' - \beta u = f$$

where α, β and f are constant has a general solution of the form

$$u = c_1 e^{\left(\sqrt{\beta/\alpha}\right)x} + c_2 e^{-\left(\sqrt{\beta/\alpha}\right)x} - \frac{f}{\beta}. \tag{3.7}$$

With n subintervals we will have n equations of the form (3.7). We denote the solutions $u_1, u_2, \cdots, u_n$ such that

$$u_k(x) = c_{k1} e^{\left(\sqrt{\beta_k/\alpha_k}\right)x} + c_{k2} e^{-\left(\sqrt{\beta_k/\alpha_k}\right)x} - \frac{f_k}{\beta_k}.$$

This gives us $2n$ unknown parameters c_{ij}. On the boundary of each subinterval the value of the solution u and the value of the flux $\alpha u'$ must coincide. On top of that we have the boundary conditions $u(a) = A$, $u(b) = B$ at the boundary of the full interval. This gives us the required $2n$ equations to solve this system. This is a linear system with the matrix form $Ac = F$. Solving this system gives us the unknowns c_{ij}. For this reason it is very efficient to create a representation of the solution cloud by sampling piecewise constant functions from the admissible data $\mathcal{D}$.

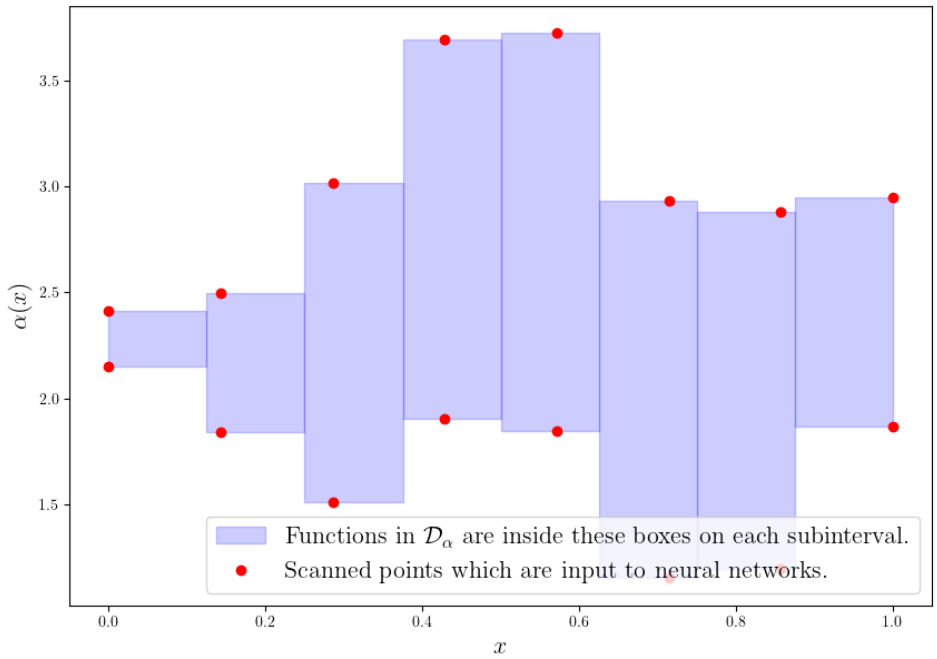

Figure 4. *Example of what admissible sets $\mathcal{D}_\alpha$ in the training datasets can look like. The constraint on each subinterval is that the functions in the admissible set are between two constants and the amount of subintervals is chosen for each training dataset. The y-coordinates of the red dots are used as inputs for neural networks.*

We created three training datasets and the examples in each dataset have the following constraints on top of the indeterminacy magnitude constraint 3.6 and the piecewise constant mean and indeterminacy function constraint 3.2.1.

Training dataset 1:

- The domain $[0, 1]$ is split into four pieces: $[x_0, x_1, x_2, x_3, x_4] = [0, 0.25, 0.5, 0.75, 1]$.
- $4 \leq \alpha_\circ(x) \leq 8$, $4 \leq \beta_\circ(x) \leq 8$, $0 \leq f_\circ(x) \leq 10$.

Training dataset 2:

- The domain $[0, 1]$ is split into four pieces: $[x_0, x_1, x_2, x_3, x_4] = [0, 0.25, 0.5, 0.75, 1]$.
- $4 \leq \alpha_\circ(x) \leq 14$, $4 \leq \beta_\circ(x) \leq 14$, $-20 \leq f_\circ(x) \leq 20$.

Training dataset 3:

- The domain $[0, 1]$ is split into eight pieces: $[x_0, x_1, x_2, \cdots, x_8] = [0, 0.125, 0.25, \cdots, 1]$.
- $4 \leq \alpha_\circ(x) \leq 14$, $4 \leq \beta_\circ(x) \leq 14$, $-20 \leq f_\circ(x) \leq 20$.

Each training dataset contains a total of 2000 datapoints. A single datapoint consists of the discrete scanned vector of length 48 generated by scanning the admissible set $\mathcal{D}$ (see Figure 4) and the approximated diameter of the solution set.

3.2.2. Test Datasets. We created three test datasets. The difference compared to the training datasets is the smoothness of the admissible datasets $\mathcal{D}$, the amount of oscillation in

the domain and the range in which the mean functions $\alpha_\circ, \beta_\circ$ and $f_\circ$ are. The first two test sets have the mean functions in the same ranges as all of the training datasets while the last test set has mean functions which have higher values then in any of the training examples. Oscillation is included in all three being the lower in the first and third set and substantially higher in the second test set. With these choices of test sets we can see if the piecewise constrained training data is sufficient to generalize to less constrained admissible data $\mathcal{D}$.

The admissible datasets for test sets are generated by randomly picking some number of points between a maximum and minimum value and fitting a spline curve to these points. This spline curve is used as the mean function of the admissible dataset. The indeterminacy functions δ_i are created in the same way. This produces smoother admissible datasets as shown in figure 5. All test sets have 130 datapoints.

The values which we can pick to create different kinds of test sets are:

1. The range of values for $\alpha_\circ(x) \in [\alpha_-, \alpha_+]$ (similarily for $\beta_\circ$ and $f_\circ$).
2. The range of values for $\delta_i \in [\delta_{i-}, \delta_{i+}]$.
3. The amount of random points used to make the splines (controls how much oscillation is possible).

These values are picked as follows for each set:

Test set 1:

- $\alpha_\circ$, $\beta_\circ \in [4, 8]$
- $f_\circ \in [4, 10]$
- $\delta_i \in [0.1, 0.3]$
- Amount of spline points is 4

Test set 2:

- $\alpha_\circ$, $\beta_\circ \in [4, 8]$
- $f_\circ \in [4, 10]$
- $\delta_i \in [0.1, 0.3]$
- Amount of spline points is 9

Test set 3:

- $\alpha_\circ$, $\beta_\circ \in [14, 24]$
- $f_\circ \in [20, 40]$
- $\delta_i \in [0.1, 0.3]$
- Amount of spline points is 4

3.3. Neural Networks. A feed-forward neural network is a machine learning model which can be used for a variety of classification and regression problems. The structure of such a model has an input layer, one or more hidden layers and an output layer. The input layer consists of as many neurons as there are dimensions in a datapoint. Hidden layers consist of a chosen number of neurons on each layer and the output layer in the case of a regression problem has a single neuron from which the approximated output value is extracted. In a dense feed-forward neural network like the ones we used each layer is "fully-connected" to the next layer such that each neuron is connected to every neuron of the next layer and adjusted by a weight parameter w. Each neuron in the hidden layers performs the operation

$$n(\mathbf{x}) = a(\mathbf{w}^T + b),$$

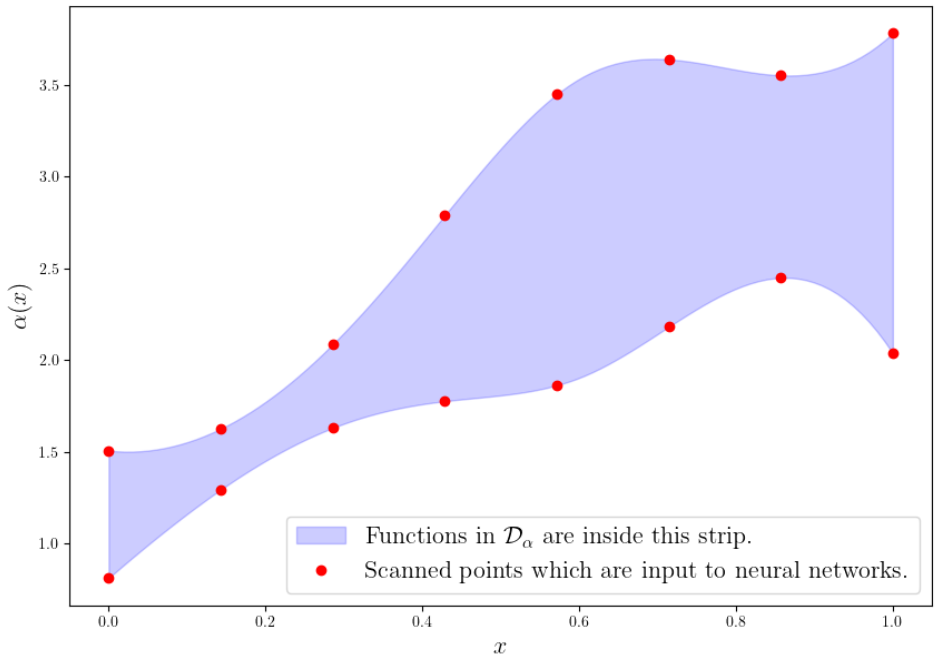

Figure 5. *Example of what admissible sets in the test datasets can look like. In the test datasets, the admissible sets $\mathcal{D}_\alpha$ have a smooth edge and oscillate randomly as fast as chosen for the particular test dataset. The y-coordinates of the red dots are used as inputs for neural networks.*

where $\mathbf{x}$ is the vector of outputs of the neurons in the previous layer, $\mathbf{w}$ are the weights of each connection from the previous layer to the given neuron, b is additional bias and the function a is the *activation function* of the neuron. The activation function may be one of many different functions but some common ones are the Sigmoid function $S(x) = \frac{1}{1+e^{-x}}$ and the ReLU (Rectified linear unit) fuction which is the piecewise defined function $R(x) = 0$ when $x < 0$ and $R(x) = x$ when $x \geq 0$.

The machine learning part in using a neural network is when the weights w and biases b of each neuron are optimized on a given dataset. In such a task when the training dataset inputs are X and outputs are Y the problem is finding

$$\inf_{w,b} ||F(X, w, b) - Y||,$$

where $F(X, w, b)$ is the output of the neural network for the entire training dataset with the weights and biases w, b. In practice the exact minimizer with respect to the weights and biases is not available. However, using numerical methods such as gradient descent such a minimization problem can be solved to a satisfactory level in many cases. For a more detailed introduction to the various different kinds of neural networks and the methods used to train them see for example [3].

3.4. Architecture. Four neural networks were created. One for each of the three training datasets and an extra one which uses all three training datasets combined as training data.

All networks have an input layer with 48 nodes. This corresponds to the admissible datasets of each parameter being scanned at eight points (see Figure 5). Implementation of the networks was done with Python version 3.8.12 [11] and Pytorch 1.9.0 [8]. Hyperparameter tuning was done with Optuna version 2.10.0 [1]. Optuna is a hyperparameter tuning package which uses various algorithms to scan the search space to find optimal parameters. In our case, the parameters being tuned are the number of hidden layers, number of nodes per hidden layer, activation function type, batch size and the hyperparameters of the Adam optimization algorithm. In Optuna, for sampling we picked the TPE (Tree-structured Parzen Estimator) algorithm, which is recommended in the documentation.

The choice of network architecture is a difficult open problem and our choices are based on intuition and trial and error. First, models with various structures were tested to give a broad idea of what kind of model could work and we found that relatively small networks with as few as two hidden layers and 20 neurons per layer are trained relatively well. We also tested ReLU and Sigmoid activation functions and found that both can give good results. In preliminary tests we used default settings of the Adam optimization algorithm but the hyperparameters of Adam will be tuned by Optuna. This gives us a good starting point to select a search space for Optuna. For the first three networks the search space given to Optuna was the following:

1. Number of hidden layers ≤ 3
2. Number of nodes in every hidden layer ≤ 100 (each hidden layer has the same amount of nodes)
3. Activation function of all nodes either Sigmoid or ReLU (all nodes have the same activation)
4. Batch size between 20 and 150.
5. Adam hyperparameters α (learning rate), β_1, β_2 and ε:
   - $1 \times 10^{-5} \leq \alpha \leq 1 \times 10^{-2}$
   - $0.01 \leq \beta_1 \leq 0.9999$
   - $0.01 \leq \beta_2 \leq 0.9999$
   - $0 \leq \varepsilon \leq 1 \times 10^{-3}$

For the final network where we use all three datasets combined to train it the search space was adjusted to allow more hidden layers and nodes in each layer. The motivation for this is the fact that a larger dataset with more varied data should require a larger model to get a good training fit. The change done was the following:

1. Number of hidden layers ≤ 5
2. Number of nodes in every hidden layer ≤ 300 (each hidden layer has the same amount of nodes)

Each dataset was used over 2000 trials to create a neural network that works well for that specific dataset. The architectures that were found best by Optuna trials can be found in table 1. The first network for the simplest dataset (Network 1) is also a lot smaller in terms of neurons and layers then the latter ones with more complex training datasets. We can see that the level of complexity of the network increases when the training datasets complexity increases. For the last three networks the ReLU activation function outperforms Sigmoid however the differences are not massive so Sigmoid or some other nonlinear activation function may work reasonably well too.

Network	1	2	3	4
Activation	Sigmoid	ReLU	ReLU	ReLU
Hidden Layers	2	3	3	5
Nodes per layer	12	82	83	159
Batch Size	24	32	20	29
Learning Rate	0.0005	0.0068	0.0047	0.0052
β_1	0.7792	0.9018	0.7822	0.9151
β_2	0.8268	0.8606	0.4533	0.8770
ε	3.9×10^{-5}	2.9×10^{-4}	1.9×10^{-4}	3.9×10^{-4}

Table 1

Best parameter values for neural networks found for each dataset over 2000 Optuna trials. These values are used in the final trained neural networks. Network 1 was optimized while training with training dataset 1, Network 2 with training dataset 2, Network 3 with training dataset 3 and Network 4 with all three training datasets (1,2,3) combined.

3.5. Tests. Neural network performance is evaluated in two ways. The first one is to simply present the mean absolute error percentage (MAPE) which is defined as follows:

$$\mathrm{MAPE}(c,p) = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{a_t - p_t}{a_t}\right|,$$

where c is the set of correct values, p is the set of predicted values and n is the amount of value-pairs.

The second way we evaluate performance is by comparing the networks outputs with known analytical upper and lower bounds. This comparison is done visually by plotting the correct approximation, predicted value and bounds for the test sets and the test portions of the training sets. These type of error bounds are well studied in [9] and [6]. The particular bounds that we use for problem (3.1-3.3) were derived in ([4], section 2.2).

For the analytical bounds we need some notations. Define

$$c_1 := \min_{x\in(0,1)} \frac{\delta_1(x)}{\alpha_\circ(x) - \delta_1(x)}, \quad c_2 := \min_{x\in(0,1)} \frac{\delta_2(x)}{\beta_\circ(x) - \delta_2(x)},$$

$$\overline{K} := \sqrt{1 + \max\{c_1, c_2\}} \qquad \text{and} \qquad \underline{K} := \sqrt{1 - \max\{c_1, c_2\}}.$$

The lower and upper bounds are defined in the next theorems.

Theorem 3.1. *The diameter of the solution set $\mathcal{S}(\mathcal{D})$ of the uncertain problem $\mathcal{P}$ with admissible dataset $\mathcal{D}$ has a guaranteed lower bound given by*

$$Diam(\mathcal{S}(\mathcal{D})) \geq \underline{K}\frac{\int_0^1 \delta_1|u_\circ'|^2 + \delta_2|u_\circ|^2 + \delta_3|u_\circ|\,dx}{\left(||u_\circ||_\circ^2 - \int_0^1 \delta_1|u_\circ'|^2 + \delta_2|u_\circ|^2\,dx\right)^{\frac{1}{2}}},$$

Network	Test portion of training set	Test set 1	Test set 2	Test set 3
1	6.17(4.40)	7.94(6.71)	9.16(7.01)	37.08(7.33)
2	13.69(13.68)	10.97(6.31)	11.49(6.67)	30.86(10.63)
3	16.16(14.58)	11.07(7.01)	10.92(5.83)	38.86(14.92)
4	14.37(13.03)	7.58(5.32)	7.34(5.53)	45.09(11.36)

Table 2

Performance of each neural network on test sets expressed as the mean absolute error percentage (Standard Deviation) rounded to two decimal digits. Each network is evaluated on their respective training datasets test split which had 200 datapoints in the first column. In the three later colums are the performances of each network on the three testing datasets.

Theorem 3.2. *The diameter of the solution set $\mathcal{S}(\mathcal{D})$ of the uncertain problem $\mathcal{P}$ with admissible dataset $\mathcal{D}$ has a guaranteed upper bound given by*

$$Diam(\mathcal{S}(\mathcal{D})) \leq 2\overline{K}\left(\left(\int_0^1 \frac{(\delta_1 u_\circ')^2}{\alpha_\circ - \delta_1}\, dx\right)^{\frac{1}{2}} + \overline{C}_F||\delta_2|u_\circ| + \delta_3||\right),$$

where $\overline{C}_F = \frac{1}{\pi}(\operatorname{ess\,sup}_{\alpha\in\mathcal{D}_\alpha, x\in[0,1]} \alpha(x)^{-1})^{\frac{1}{2}}$.

The proofs of Theorems 3.1 and 3.2 with all the required preliminaries can be found in [4] section 2. Creating such analytical bounds is based on a posteriori error estimates of the functional type which are discussed at length in [9] and [6]. Since the proofs are rather long and technical we omit them here.

4. Results. The performance of each neural network in terms of the MAPE value is shown in table 2. In figures 6-8 the performance of the networks compared to the brute force approximations and analytical bounds is compared using the test portion of the training datasets. In figures 10-14 the networks performances are similarily showcased on the test datasets.

In some cases the MAPE value is high but the figure still has some reasonability in what the network outputs look like compared to the analytical bounds (e.g. Figure 14).

On the test portion of their respective training datasets network 1 has the highest accuracy. This is expected since the dataset used to train network 1 is the most constrained. Looking at the images (6)-(9) the performance of all four networks on the test sets related to their training datasets seem decent. Each network outputs values which are always within the analytical bounds. Network 4 is the only one with a few examples where the network output is rather far away from the approximation but still within the bounds.

Performance on the three test sets varies. Network 1 and network 4 perform the best on test set 1 in terms of the MAPE value. Network 4 keeps performing similarily well on test set 2 while network 1 gets slightly worse. The added oscillation in test set 2 has this effect since network 1 has not seen such highly oscillating data while network 4 has seen more oscillating data (see Figures (10)-(12)). Networks 2 and 3 seem to suffer from the fact that the range in which their training datasets are is larger then the test sets. This means there are less training examples in the same range as the test set which explains why network 1

Dataset	Average time to create one datapoint
Training 1	1m 31s
Training 2	2m 28s
Training 3	6m 11s
Test 1	18m 27s
Test 2	29m 30s
Test 3	15m 42s

Table 3
Computational time to create datapoints in each training and testing dataset.

performs better. Still, both networks 2 and 3 are not completely lost on these test sets and keep performing within the analytical bounds. From this we can tell that piecewise constant oscillations approximate the smoother oscillations of the test sets reasonably well.

All four networks have a much higher error on test set 3 which shows us that the networks do not easily generalize to examples where the admissible data has functions which are not in the same range as the ones in the training examples. In Figure 14 the performance of network 4 on test set 3 is shown. Here we can see that the network overestimates the value of the diameter but there is still a linear growth which coincides with the growth of the correct outputs. The network outputs are still within the analytical bounds.

Depending on the size of the neural network and luck the time to train them varied from as little as five seconds to at most two minutes on a normal laptop. The total time to create the optimised models using Optuna with 2000 trials took between five and ten hours for each model. The computational time to create each dataset is shown in table 3. Training datasets with their piecewise constant restrictions are generated more quickly in all cases. Test set 2 was much slower to create because of the higher oscillation allowed.

The amount of time saved when using a neural network rather then a Monte Carlo computation is very large. Each of the neural networks created can be used to calculate an approximation in less then one millisecond. The Monte Carlo method we implemented takes an average of 56 seconds to create the solutions and another 188 seconds to compute the maximum distance between the solutions. In total the time to use the Monte Carlo algorithm to approximate the error quantity is 244 seconds.

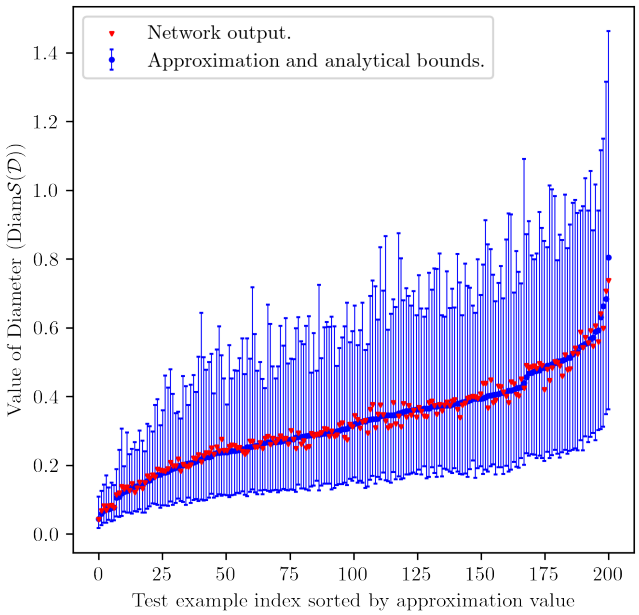

Figure 6. *Network 1 performance on test part of training dataset 1 which has 200 datapoints sorted by the ground truth approximation value. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

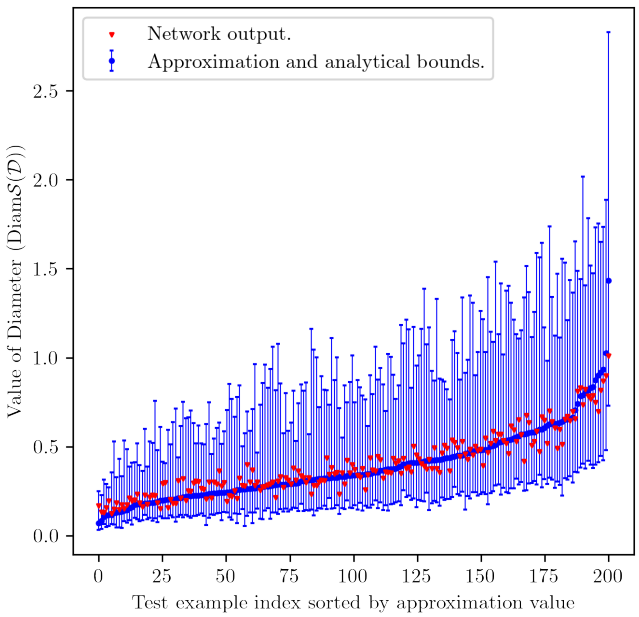

Figure 7. *Network 2 performance on test part of training dataset 2 which has 200 datapoints sorted by the ground truth approximation value. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

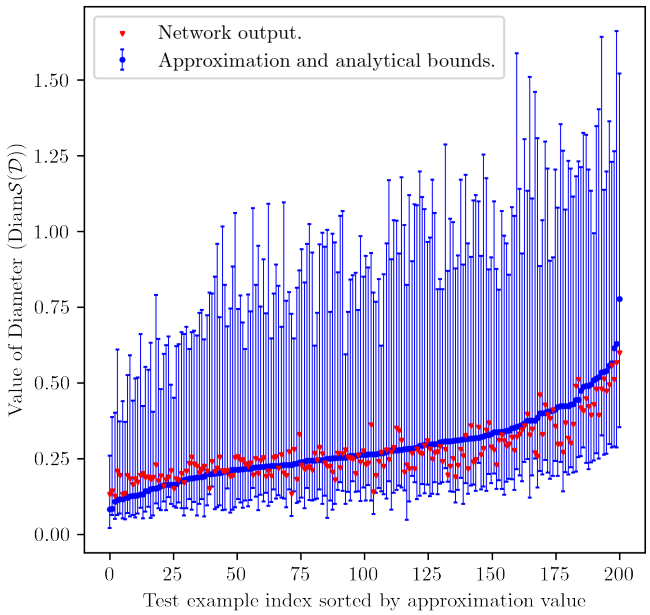

Figure 8. *Network 3 performance on test part of dataset 3 which has 200 datapoints sorted by the ground truth approximation value. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

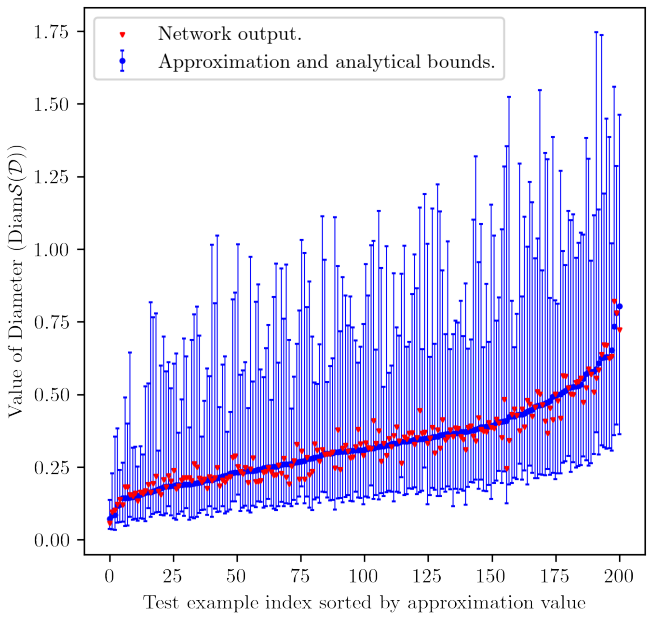

Figure 9. *Network 4 performance on 200 random samples from the test part of the combined dataset sorted by the approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

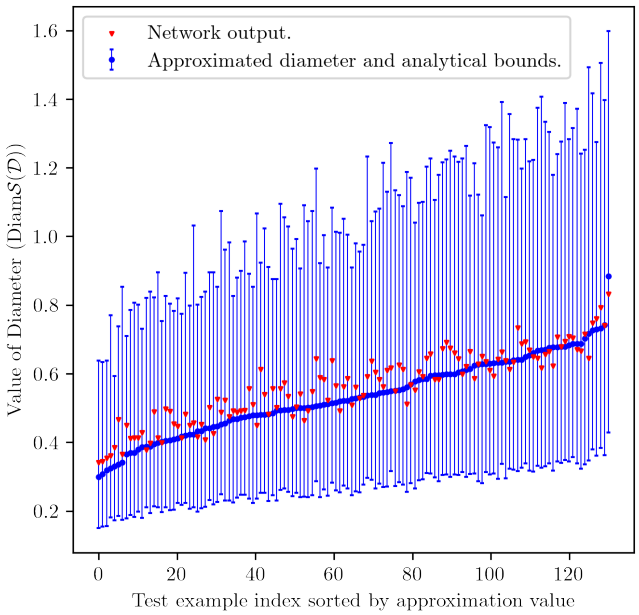

Figure 10. *Network 1 performance on test set 1. Datapoints sorted by approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

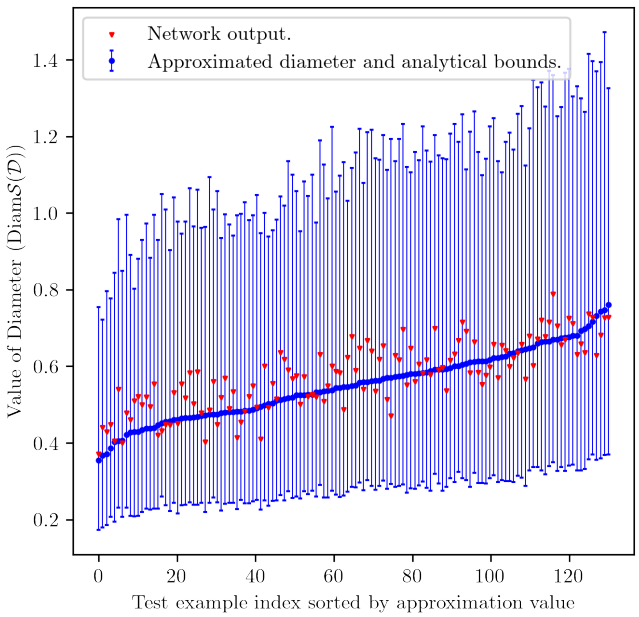

Figure 11. *Network 1 performance on test set 2. Datapoints sorted by approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

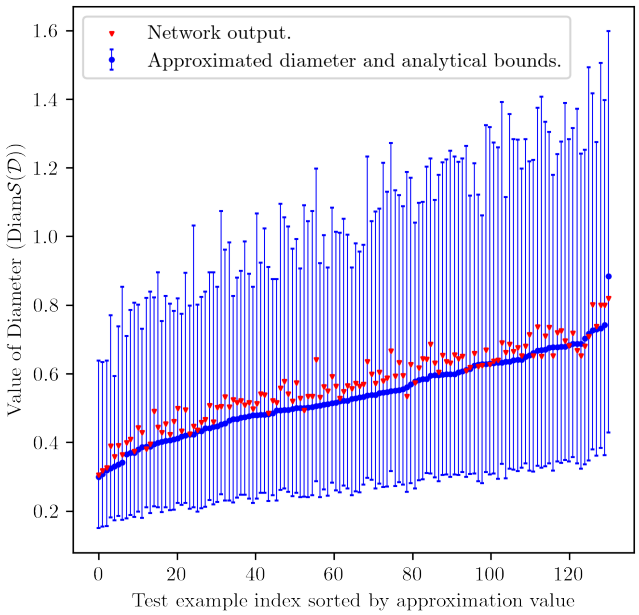

Figure 12. *Network 4 performance on test set 1. Datapoints sorted by approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

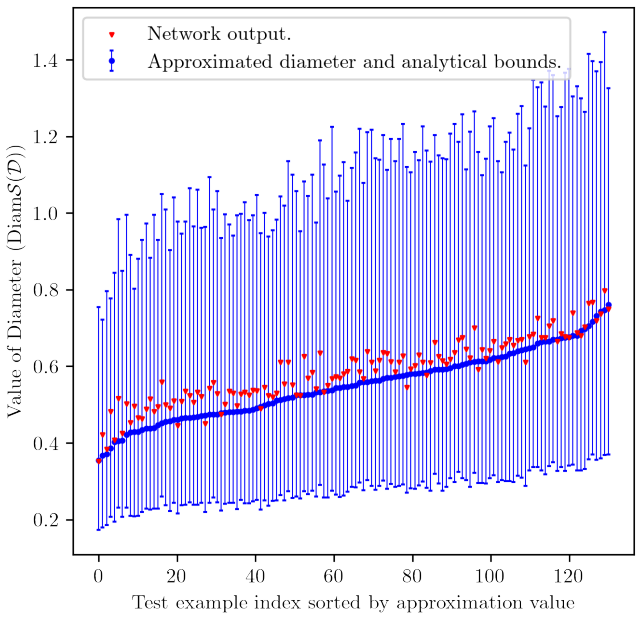

Figure 13. *Network 4 performance on test set 2. Datapoints sorted by approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

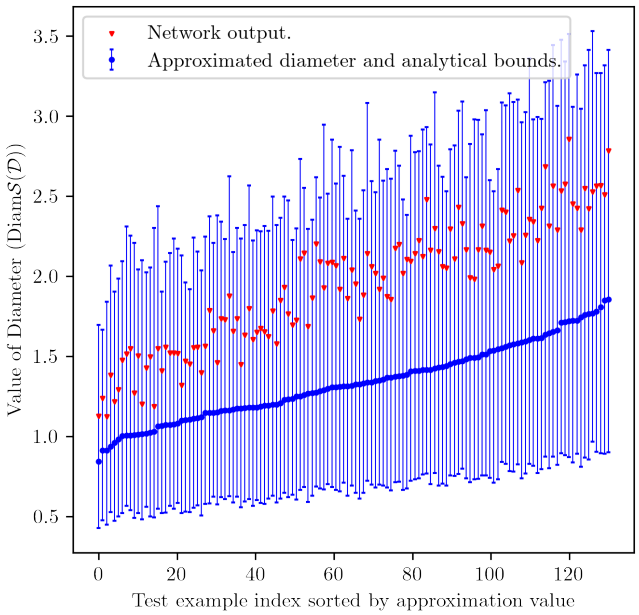

Figure 14. *Network 4 performance on test set 3. Datapoints sorted by approximation values. Blue dots are the brute-force approximation of the diameter, error bars in blue are the analytical bounds given in 3.1 and 3.2, and red triangles are the neural network outputs.*

5. Discussion. When trying to make an error quantifying machine learning model three things had to be considered. How to create data, model architecture and verifying model accuracy. In this paper the way we created data is the simplest method but it is also computationally expensive. Similar methods as the ones used to create data for training models which solve PDE's (see e.g. [7]) could be used to make this process quicker. In more difficult problems this may even be practically necessary since such brute force methods may be unreasonably expensive.

When making data for a specific real-world problem one can more carefully select the type of training examples. The problem we discussed was not related to any physical values but only serves to show that such a model can perform in some constrained situation. It is most likely important that the training data coincides with the range of values that the practical application will deal with. Choosing very constrained datapoints for training data may help with quickly generating the dataset but accuracy may be sacrificed if this is done to too large an extent. In our example, using a rather constrained dataset still produced good results for some less constrained test datasets.

Dense neural networks worked well in our example but there is no reason to think that more complicated models like CNN's can't be used for quantifying uncertainty errors in some problem. For the example we provided even simpler regression models like nonlinear support vector machines could be tried. For more difficult problems it is hard to say how sophisticated the models required to tackle this problem are.

Accuracy verification of models may also be very difficult in higher dimensional nonlinear problems. For our example problem, both analytical and numerical methods can be used to

verify results but this is not always the case. In most real-world problems analytical methods not at our disposal and one would like to avoid expensive computational methods as well.

Future research on this topic will be focused on creating machine learning models that estimate uncertainty errors of higher dimensional PDE's. A particularily interesting application of uncertainty quantification models is in simulations. Scientific experiments in the future will be done more and more by simulations and in order for a simulated experiment to be reliable error quantification has to be addressed. Complex simulations by themselves are costly and in such cases there are no efficient ways to quantify uncertainty errors quickly.

There are other interesting quantities that could be approximated with similar machine learning models as the ones described. The error metric we used was the total error described by the energy norm diameter of the solution set but depending on the use-case this is not always the most valuable information. In certain applications for example local errors are more useful then the global error. For such problems a model should be trained to approximate a local error metric instead.

6. Conclusions. The experiments show that a neural network can be used to accurately approximate errors caused by uncertainty in the case of a linear differential equation. In all of the tests conducted the neural networks work well enough to be considered as an option instead of previously known analytical methods. These results show that if a suitable training dataset can be created it is feasible to use machine learning models for uncertainty quantification. Compared to the brute-force approximation and known analytical bounds the models perform with similar accuracy. The approximation is essentially a Monte Carlo type quantity which means there is hope of replacing this type of slow and expensive methods with faster machine learning model inference. In terms of time saving the neural networks are much faster to use then a Monte Carlo method. For this simple linear problem the Monte Carlo approximation still takes around four minutes to compute while the neural network is inferenced in less then one millisecond.

In our example, the training datasets and test datasets had some differences. Even though the test data was more general, the neural networks performed in a good way. It is very useful if training examples can be picked from a more constrained class of examples since in some cases this makes the generation of the training example much more computationally fast. The only test set on which the models did not perform very well is the last one. In that case, the range of values for functions in the admissible data of test examples was outside of the range in which they were for training datasets. Higher oscillation of the admissible datasets in test examples did not have a catastrophic effect.

Creation of networks and data took a fair amount of time even for this simple problem. The data was generated on a normal laptop and it took around two minutes to generate a single training datapoint. The datapoints in the test datasets took longer at around ten minutes for a single datapoint. It is clear that the process of data generation for higher dimensional problems will have to be done by organisations with large computational resources or by using more clever methods which require less computation.

Doing the same thing for 2-D and 3-D problems will be computationally more difficult for both creation of data and training models. However neural networks even when they are much more complicated then the ones we have used in this paper are trained with relatively

small amounts of computational resources. The main computational time problem will be with creating the data. In this paper the Monte Carlo method used required creating 2000 different solutions for a simple ODE and that took one minute or so. For a complicated non-linear high dimensional PDE generating this set of solutions may take a very long time depending on what type of solvers can be used for that problem. Computing the maximum distance between solutions will also be more expensive when more dimensions are introduced since numerical integration gets more expensive with more dimensions. It may also require more datapoints to have a suitable training dataset for a network to learn properly. It is reasonable to expect that the more complicated the PDE model and the more general the restrictions on the dataset are the more training data is required.

Acknowledgments. Thanks to Professor Sergei Repin for fruitful discussions and contributions.

REFERENCES

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, in Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[2] P. G. Ciarlet, *The finite element method for elliptic problems*, SIAM, 2002.

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

[4] V. Halonen, *Accuracy analysis of uncertain variational problems with analytical and machine learning methods*, University of Jyväskylä, 2021. Master's Thesis.

[5] I. Hlavacek, J. Chleboun, and I. Babuska, *Uncertain input data problems and the worst scenario method*, Elsevier, 2004.

[6] O. Mali, P. Neittaanmäki, and S. Repin, *Accuracy verification methods: Theory and algorithms*, vol. 32, Springer Science & Business Media, 2013.

[7] A. Muzalevskiy, P. Neittaanmäki, and S. Repin, *Generation of Error Indicators for Partial Differential Equations by Machine Learning Methods*, Springer International Publishing, Cham, 2022, pp. 63–96, https://doi.org/10.1007/978-3-030-70787-3_6.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9] S. I. Repin, *A posteriori estimates for partial differential equations*, in A Posteriori Estimates for Partial Differential Equations, de Gruyter, 2008.

[10] G. Schuëller, *A state-of-the-art report on computational stochastic mechanics*, Probabilistic Engineering Mechanics, 12 (1997), pp. 197–321, https://doi.org/https://doi.org/10.1016/S0266-8920(97)00003-9, https://www.sciencedirect.com/science/article/pii/S0266892097000039.

[11] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*, CreateSpace, Scotts Valley, CA, 2009.