École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

The prediction power of an artificial neural network

Author:

Tim Valencony, École Polytechnique

Advisor:

Leo Liberti, CNRS LIX École Polytechnique, Institut Polytechnique de Paris, Palaiseau France

Academic year 2022/2023

Abstract

This thesis investigates the impact of node values during training on the performance of Artificial Neural Networks (ANNs) for function prediction tasks. To accomplish this, we analyze the number of nodes outside of range and the average distance from the node values' mean that are produced by an unseen input, compared to the network's prediction error on the input. Four different ground truth functions, from easy to hard-to-predict, were tested on relatively shallow networks. Our empirical results indicate that while the average distance from the node values' mean is correlated with the network prediction error on unseen inputs, it is hard to say the same with the number of node values out-of-range. This research has demonstrated that to obtain more conclusive results, testing on larger networks and complicated functions is necessary. Furthermore, mathematical proof of these findings would provide tangible results in this field.

Contents

1 Introduction

1.1 Basic concepts

Artificial Neural Networks (ANNs) are an essential part of this report; our aim is to focus on the prediction power of ANNs compared to their node values obtained during training. Therefore, understanding how they work and what their semantic functions are is crucial. In this section, we discuss some preliminaries and important concepts relating to ANNs, their characteristics and how they operate. See [11, 1, 7, 12] for foundational papers and textbooks.

1.1.1 Definition

An Artificial Neural Network is a type of Machine Learning algorithm that is designed to mimic a simplified structure and function of biological neurons and synapses in the human brain. They are used to approach functions $f : \mathbb{R}^n \to \mathbb{R}^m$, called *ground truth*, as closely as possible based on a limited amount of known input and output values of the ground truth. ANNs are mostly exploited for static data processing tasks such as regression, classification, and pattern recognition. In general, they have proven useful in multiple different fields, including image & speech recognition, NLP, and robotics.

In order to better depict the Mathematics behind Artificial Neural Networks, we will focus on a very simplistic case: an artificial neuron with one input & output layer, but no hidden layers, called a perceptron [15]. In this example the output layer is one dimensional, so we are predicting a ground truth function $f : \mathbb{R}^n \to \mathbb{R}$ with the perceptron. There are two principles on which a neural network is based: Forward Propagation and Backward Propagation.

1.1.2 Forward Propagation

Forward propagation is the process of feeding the input data through the ANN in order to get a prediction from it. Assume the input vector is $x = [x_1, x_2, ..., x_n]$, with $x \in \mathbb{R}^n$ & $n \in \mathbb{N}$ and the set of weights associated to the perceptron is $w = [w_1, w_2, ..., w_n]$. We define the summation (S) to be equal to the dot product of the input and weight vector of the perceptron. We have $S = (x_1 w_1) + (x_2 w_2) + \ldots + (x_n w_n) = x \cdot w$. The next step is to add bias to the summation. We take $v = S + b$, with $b \in \mathbb{R}$ and v the single node value in our Artificial Neural Network. Finally, we define an activation function σ which is used to introduce non-linearity in the ANN. We define σ as follows: $\sigma(t) = \frac{1}{1+\exp -t}$ $\forall t \in \mathbb{R}$, it can be replaced by any other type of activation function.

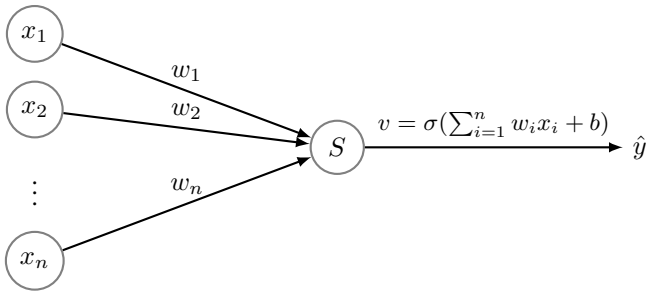

Figure 1. Graph of a Perceptron, a Neural Network with no hidden layers.

The final predicted result is thus $\hat{y} = \sigma(v)$, where $\hat{y}$ is the perceptron's output prediction of the ground truth $f(x)$. However, in order for the network to "learn" and be parameterized as accurately as possible compared to the *ground truth* to make good predictions, the perceptron has to go through a learning algorithm during what we call the training phase.

1.1.3 Backward Propagation

Backward Propagation, most commonly referred to as Backpropagation, is the learning algorithm used to compute the gradient of the loss function to readjust the trainable parameters, weights and biases, of the ANN during training. Using the *Mean Squared Error* as loss function for this example, back propagation consists in determining how far the perceptron's prediction during training $\hat{y}$ is from the predicted function's truth $f(x)$. This is computed for training input x_j as follows: $\mathsf{MSE}_j = ||f(x_j) - \hat{y}_j||^2$. Assuming there are $N \in \mathbb{N}$ training samples in the training set, the whole cost of the error on every training prediction is thus $\mathsf{MSE} = \frac{1}{N} \sum_{j=1}^{N} \mathsf{MSE}_j$.

Having computed the error MSE_j for a training sample x_j, we can now compute the gradient in order to update our

weights $w = [w_1, w_2, \ldots, w_n]$. We have to compute it for each w_i with $i \in \{1, \ldots, n\}$ and for b. Using the chain rule, we obtain:

$$\frac{\partial \mathsf{MSE}_j}{\partial w_i} = \frac{\partial \mathsf{MSE}_j}{\partial \hat{y}_j} \times \frac{\partial \hat{y}_j}{\partial v} \times \frac{\partial v}{\partial w_i}$$

We therefore have to compute the three different gradients. We have:

$\frac{\partial \mathsf{MSE}_j}{\partial \hat{y}_j} = 2(f(x_j) - \hat{y}_j)$, then $\frac{\partial \hat{y}_j}{\partial v} = \sigma(v)(1 - \sigma(v))$, and finally $\frac{\partial v}{\partial w_i} = x_j$.

Hence, $\frac{\partial \mathsf{MSE}_j}{\partial w_i} = 2(f(x_j) - \hat{y}_j)\sigma(v)(1 - \sigma(v))x_j$, while $\frac{\partial \mathsf{MSE}_j}{\partial b} = 2(f(x_j) - \hat{y}_j)\sigma(v)(1 - \sigma(v))$ as bias is considered to have a constant input value of 1. In the end, the weights and biases of the perceptron are updated as follows:

$$w_i = w_i - (\alpha \frac{\partial \mathsf{MSE}_j}{\partial w_i})$$

$\forall i \in \{1, \ldots, n\}$, and:

$$b = b - (\alpha \frac{\partial \mathsf{MSE}_j}{\partial b})$$

at each evaluation of the perceptron with input value x_j and where α is the parameter called *learning rate* of the perceptron. This process is done for every x_j in the training set.

1.1.4 Generalization to ANNs

All in all, Artificial Neural Networks are bigger perceptrons with multiple layers and nodes, enabling them to learn and recognize more complex patterns. For an objective function $f : \mathbb{R}^n \to \mathbb{R}^m$ to be predicted, we define the ANN with node values in a set V. We will classify the input nodes in $I \subset V$, and output nodes $O \subset V$, with $|I| = n$, $|O| = m$ and $I \cap O = \emptyset$. The other nodes in the network are called hidden nodes, and will be defined by a set $H = V \setminus \{I \cap O\}$.

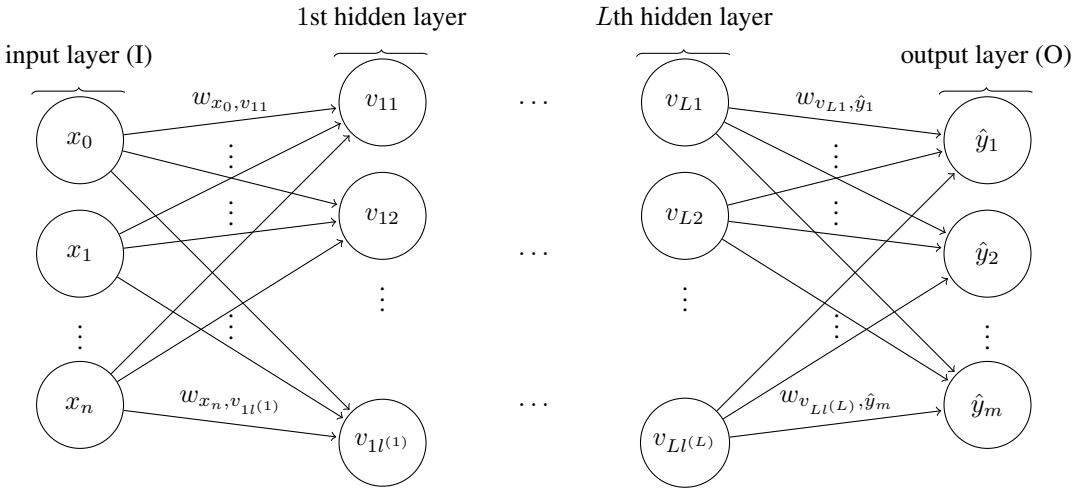

Figure 2. Graph of an Artificial Neural Network, with $l^{(L)}$ being the number of nodes in the Lth layer.

We associate biases b_i to each node $v_i \in V \setminus I$ and weights w_{ij} to each arc $(v_i, v_j) \in A$, with $A \subset V \setminus O \times V \setminus I$ being defined as the set of arcs in the ANN. Furthermore, we define the training set T as the knowledge in our possession about the ground truth f, with $|T| = N \in \mathbb{N}$. This set is composed of input and output vectors of the form $(x_j, y_j) \in \mathbb{R}^{n+m}$ with $j \in \{1, \ldots, N\}$.

Having executed the training algorithm on the ANN, and thus updated its weights and biases accordingly, the ANN can be evaluated with inputs in $\mathbb{R}^n$. We will call this the evaluation problem and assign the set of samples on which it tests the networks to be $\mathcal{E} \subset \mathbb{R}^n$. We also define the set of unseen samples on which the networks are tested as the set of sample points which are in $\mathcal{E} \setminus T$, we will call it $\mathcal{U}$.

The evaluation problem aims at computing an output vector $\hat{y} \in \mathbb{R}^m$ for input vector $x \in \mathcal{E}$. In this case, the forward propagation is defined as follows for input x:

$$\forall v_{1i} \in I \, , \, v_{1i} = x_i$$
$$\forall v_{ij} \in V \setminus I \, , \, v_{ij} = \sigma(\sum_{k=1}^{l^{i-1}} w_{v_{(i-1)k},vij} \cdot v_{(i-1)k}) + b_{v_{ij}})$$

We then assign the node values belonging to the output layer O to the network's prediction as follows: $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_m] = [v_{L1}, \ldots, v_{Ll^{(L)}}]$ with L being the last layer of the ANN, and we are done.

1.2 Motivation & Literature Review

1.2.1 Network Performance Metric

The main goal of this thesis is to ascertain what makes a "good" network. However, there are multiple definitions that apply to a "good" ANN. The first and most common one is that it performs well on inputs close to the training set T, and the second one is an ANN which performs well on all input vectors of the ground truth f belonging to $\mathbb{R}^n$.

Solely evaluating the performance of an Artificial Neural Network on its capacity to reproduce outputs from similar inputs taken in training set T has limited interest; it was established that valid generalizations can be almost certainly expected if $m > O(\frac{W}{\epsilon} log(\frac{N}{\epsilon}))$ random samples can be fed to a network of linear threshold functions with N nodes and W weights, so that at least a fraction $1 - \frac{\epsilon}{2}$ of the examples are correctly satisfied [3]. In this situation, one has confidence $1 - \epsilon$ that future test samples taken in $\mathcal{E}$ drawn from the same type of distribution as T will be correctly guessed by the ANN.

Our focus will thus be dedicated to the second performance metric of an ANN, which we believe to be more interesting and revealing in the application of Neural Networks theory.

1.2.2 Solving to Global Optima

It might seem impossible to solve an ANN to global optimality for any function, but it can be done for very simple ANNs with very large training sets [9]. Finding the right balance, however, can be quite challenging as it was proven this phenomenon often leads to overfitting [5]. It was also demonstrated that under the right conditions, the local optima found by the ANN are globally optimal and that a local descent strategy can reach global minimum from any initialization [8]. The right conditions being the requirement of the network's output and regularization to be both positively homogeneous functions of the network parameters, while controlling the network size through the regularization's design.

For other scenarios, which happen to be the majority of the cases, local optima found by the ANN using Backward Propagation methods such as Stochastic Gradient Descent are the best the ANN can do. However, the reason Artificial Neural Networks still often perform well when they are solved to local optimality is that local optima are likely to have close values to the global optimum [4].

The learning algorithm and its training problem of approximating the best values of biases b and weights w for our ANN to perform well for unseen input is central to our project. Until now, most of the research has been concentrated on obtaining fast optimization methods for achieving near-local optima on domains similar to T. In this thesis we focus on small-sized ANNs in order to determine meaningful indicators of prediction quality.

1.2.3 The role of Node Values

Besides, we have seen the crucial and intrinsic role nodes and their respective values $v \in V$ play in the mathematical definition of an Artificial Neural Network. We did not find, in the relevant literature, many references to the inter-dependence between node values, network topology, and prediction quality. However, studies on the impact of the number of nodes on an ANN's performance are numerous. Indeed, an ANN with too few or too many nodes might underfit or respectively overfit the model, making it obsolete. A study by renowned Andrew R. Barron [2] found that the estimation error of an ANN on target function f is bounded by:

$$O(\frac{C_f^2}{n}) + O(\frac{nd}{N} log(N)),$$

where n is the number of nodes, d is the input dimension of the function, N is the number of training observations, and C_f is the first absolute moment of the Fourier magnitude distribution of f. Optimization of the number of hidden nodes in the network with respect to performance was also explored, with algorithms being provided for finding the optimal number of hidden nodes by elimination [6]. This was comforted by studies in pruning methods [14] used to reduce computational time, energy, and error by replacing fully-connected ANNs with ANNs possessing alternative complex topologies [10].

Although those are crucial findings about Artificial Neural Network topology and nodes, there seems to be an area of the research with little to no investigation (in our knowledge): the impact of node values on an ANN.

1.2.4 Problem formulation

Today's trend in the composition of training data is to collect a certain amount of input-output pairs from the ground truth function f. Then, $70-80\%$ of this data is used for the learning phase, while the rest is kept for the testing phase. The testing phase is used to evaluate the prediction quality of trained networks, where evaluation normally occurs on inputs for which the output is yet unknown. Nonetheless, as the size of the set $\mathbb{R}^{m+n} \setminus T$ is infinite, having fewer in-sample data than out-of-sample data might lead to a very local prediction around T, limiting the power of prediction of the ANN.

This is why, the main goal of this thesis is an investigation on the impact that the node values $v \in V \setminus (I \cup O)$ defined during training have on the output generated by the ANN on *unseen inputs* in $\mathcal{U}$.

For example, one would expect that, for unseen input where node values are within the ranges $\mathcal{R} = [v^L, v^U]$ over which the ANN node values vary during training, the output will have smaller error than for unseen input where node values are not within $\mathcal{R}$ during evaluation. We will try to verify or deny this hypothesis over the course of this report.

2 Technical Preliminaries and parameters

2.1 Defining the conditions of observation

This section examines what parameters we are considering in order to tackle the issue at hand, i.e determining the impact that the node values defined during training have on unseen input. In order to explore as many alternatives as possible, we have to precisely determine what quantities we would like to measure, and under what conditions. This means deciding a set of ground truths we would like to focus on, how the training and testing sampling of each ground truth will be done, and the way we evaluate the changes in the node values for test samples and networks as a whole.

2.1.1 Ground Truth Selection

The first parameter concerns the type of ground truths we will try to predict, keeping in mind our original goal. As we will focus on function prediction throughout our works, we determine a range of functions on which we will collect data, from easy to predict to very hard to predict. The aim is to get a good intuition on the easy functions, and then try to generalize that intuition on hard to predict functions, which are more likely to be of use in general. Each ground truth will be defined like was done so far, i.e by $f : \mathbb{R}^n \to \mathbb{R}^m$, where $n, m \in \mathbb{N}$.

The formulae for the studied functions are spelled-out here.

$$\begin{cases} f_1(x) = \sin(x) \\ f_1 : \mathbb{R} \to [-1, 1] \end{cases}$$

$$\begin{cases} f_2(x) = \mathbb{1}_{\{x \leq 14\}}[1 + \cos(x) - 3\cos(2x) + \cos(3x)] + \mathbb{1}_{\{14 < x \leq 17\}}[\sin(1.2x) + 4.2] + \mathbb{1}_{\{x > 17\}}[\sqrt{x} + 0.6] \\ f_2 : \mathbb{R} \to \mathbb{R} \end{cases}$$

$$\begin{cases} f_3(x, y) = x + y + \sin(4xy) \\ f_3 : \mathbb{R}^2 \to \mathbb{R} \end{cases}$$

$$\begin{cases} f_4(x, y) = \mathbb{1}_{\{-1 \leq y < 1\}}[x + y] \\ \qquad\qquad + \mathbb{1}_{\{(-3 \leq x < -1) \cap (-3 \leq y < -1)\}}[(x+3)^2 \cdot 3(x+1)^2 \cdot (y+3)^2 \cdot (y+1)^2] \\ \qquad\qquad + \mathbb{1}_{\{(-3 \leq x < -1) \cap (1 \leq y \leq 3)\}}[-x^2 - y^2 - xy + 5] \\ \qquad\qquad + \mathbb{1}_{\{(-1 \leq x < 1) \cap (-3 \leq y < -1)\}}[-2x - 0.4y - 0.5xy - 1] \\ \qquad\qquad + \mathbb{1}_{\{(-1 \leq x < 1) \cap (1 \leq y \leq 3)\}}[\frac{\sin(x-2}{y+2}] \\ \qquad\qquad + \mathbb{1}_{\{(1 \leq x \leq 3) \cap (1 \leq y \leq 3)\}}[\log(xy)] \\ \qquad\qquad + \mathbb{1}_{\{(1 \leq x \leq 3) \cap (-3 \leq y < -1)\}}[(x-2)^2 - (y+2)^2 - 1] \\ f_4 : [-3, 3]^2 \to \mathbb{R} \end{cases}$$

We visualize them on Figure 3 below.

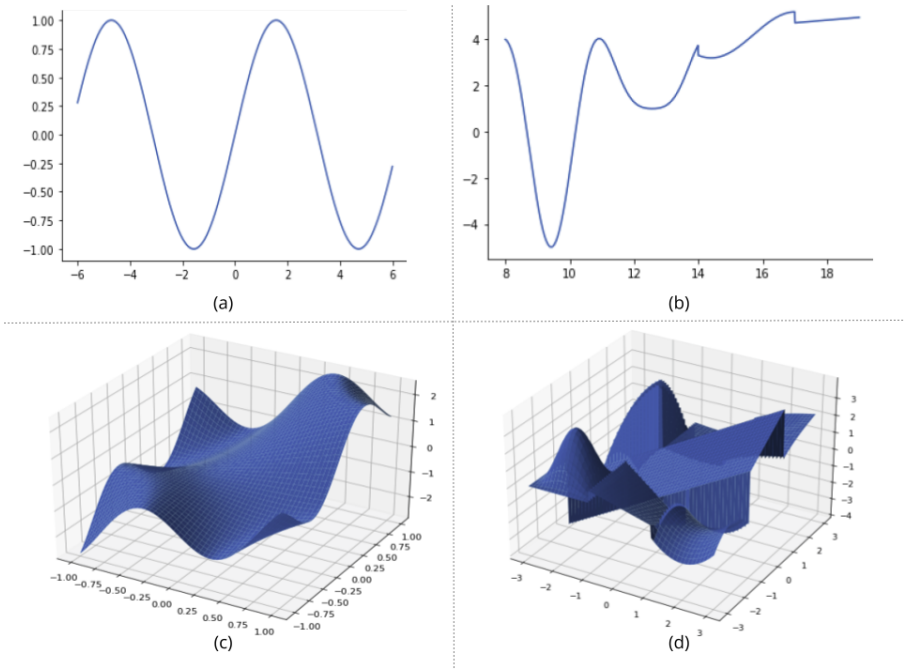

Figure 3. (a) Graph of f_1 on $[-6, 6]$. (b) Graph of f_2 on $[8, 19]$. (c) Graph of f_3 on $[-1, 1]^2$. (d) Graph of f_4 on $[-3, 3]^2$.

We have to keep in mind that our goal is to evaluate the networks on unseen inputs $\in \mathcal{U}$. In our computational experiments we will discuss node value ranges yielded by the training phase, called *training ranges* and denoted by T, as well as node value ranges yielded by the testing phase, called *testing ranges* and denoted by $\mathcal{E}$. This will be done twice per function, once where T is concentrated in one block, and a second time where T will be defined in a possibly disjoint union of different sets. $\mathcal{E}$ will always contains T, and $\mathcal{U}$ will be defined as follows: $\mathcal{U} = \mathcal{E} \setminus T$.

2.1.2 Measuring Node Values' Impact on Sample Points' Prediction Accuracy

This parameter concerns node values. In order to measure the fluctuations of node values for a test point which will be predicted well or poorly by a trained ANN, we base our analyses on two types of criteria on the node values generated by that test point.

The first criterion is *qualitative*. We collect node values over the training phase, then for each node $v_i \in V \setminus \{I \cap O\} = H$ in the network we compute the range $R_i = [v_i^L, v_i^U]$ where v_i^L (respectively v_i^U) is the lower (respectively upper) bound value of that node over the training phase. Once the range is established and the network's learning finished, when feeding the network an unseen test input $x_j \in \mathcal{U}$ we count the number of node values v_i in the network which were outside of their respective range R_i. We denote this criterion by $\mathcal{O}_j$.

The second criterion is *quantitative*. With this criterion, we are trying to understand "how far" from the node values obtained over the training phase a given unseen sample is. For this, after having collected the node values during the training phase, we calculate each node's mean value $\bar{v}_i$ and standard deviation v_i^σ. Then, we then feed the network an unseen test sample, and assign to each node of the ANN its distance from the mean in terms of standard deviations for this test sample. We will call this distance d_{v_i} for each node $v_i \in H$ and calculate it as follows; $d_{v_i} = |\frac{v_i - \bar{v}_i}{v_i^\sigma}|$. The aim is then to get the average distance from the mean for the whole network per unseen test sample x_j, which we will denote by

$$\Delta_j = \frac{\sum_{v_i \in H} d_{v_i}}{n}$$

with $n = |H| \in \mathbb{N}$ denoting the number of hidden nodes in the tested network.

2.1.3 Measuring Node Values' impact on Network Performance

This part aims at assessing, with the use of the measures on node values defined above, how the node values vary on average for a network NN_i being fed a number $N = |\mathcal{U}|$ of unseen inputs x_j. Once again, a qualitative and quantitative

approach is taken, which we will explain below.

We start with the *qualitative criterion*. Take $j \in \{1, 2, \dots, N\}$. For each j, we take the number of nodes out-of-range $\mathcal{O}_j$ associated to unseen sample x_j, and compute the mean of the $\mathcal{O}_j$s on the whole testing set for this network. We have:

$$\Omega = \frac{\sum_{j=1}^{N} \mathcal{O}_j}{N}.$$

Equivalently for the *quantitative criterion*, we simply extract information on the average distance from the node values' mean Δ_j for each of the N unseen samples x_j the network NN_i was fed, and compute the mean of the Δ_js for this network. We obtain:

$$\delta = \frac{\sum_{j=1}^{N} \Delta_j}{N}.$$

We respectively call Ω_i and δ_i the average number of nodes out-of-range and the average distance from mean per network NN_i.

2.1.4 Sparsification of the Dataset

Finally, the last parameter investigated is the sampling method for our training data-sets, which we will apply to our regression problems consisting in predicting ground truths $f : \mathbb{R}^n \to \mathbb{R}^m$. Through this parameter, we study the impact these methods have on the spreading of node values when evaluating the network, compared to the node values over training. Indeed, the intuition behind this is that if a sampling method can, on average, select samples so that the network's range $\mathcal{R}$ over training is bigger, and/or minimize the distance from the mean $\bar{v}_i$ in terms of standard deviations v_i^σ for each node $v_i \in H$, then we can generalize better and determine whether those factors actually minimize our generalization error.

In fact, assuming a function has hard-to-predict sections in the output space and sections where it is easy to predict, if a sampling method could select points accordingly, enabling the network to be fed more train samples from the hard to predict sections than the easy ones, then we would see a significant increase in network accuracy for the same amount of training points. Intuitively, we would also see the impact this has on the distribution of nodes over the different sampling methods. Thus, this thesis identifies three different sparsification methods for data sampling which will be studied:

- *Regular sampling*. This method consists in separating the wanted input space $\mathbb{R}^n$ into a grid-like structure and taking the intersecting points of the grid as training inputs, then feeding the training input to the oracle f in order to get the training output and thus define T the training set. This can be an efficient, accurate, and simple way to sample for functions which have similarly-shaped ouputs over the training input.

- *Random sampling or Uniform sampling*. This method simply sporadically picks a number $N = |T| \in \mathbb{N}$ of points in the input space $\mathbb{R}^n$. This is one of the most used sampling methods in Machine Learning nowadays. We will call it *Uniform Sampling* in this thesis as this is the name of the function used to determine those points in *python*.

- *Low-discrepancy sequence sampling or Sobol sampling*. This type of sampling is interesting as it uses low-discrepancy sequences [17] to cover the input domain of interest quickly and evenly as it uses quasi-random theory. This should yield a significant advantage over the random sampling in terms of network performance as was demonstrated in [13] for Deep Learning algorithms that are based on randomly chosen training data, with problems in moderately high dimensions. In our setting, we will only direct our attention to Sobol sequences [16] for low-discrepancy sequence sampling as it is practical to implement and a lot of documentation can be found on the subject.

We represent each sampling method in Figure 4, with the *Regular Sampling* being in (a), *Sobol Sampling* in (b), and *Uniform Sampling* in (c).

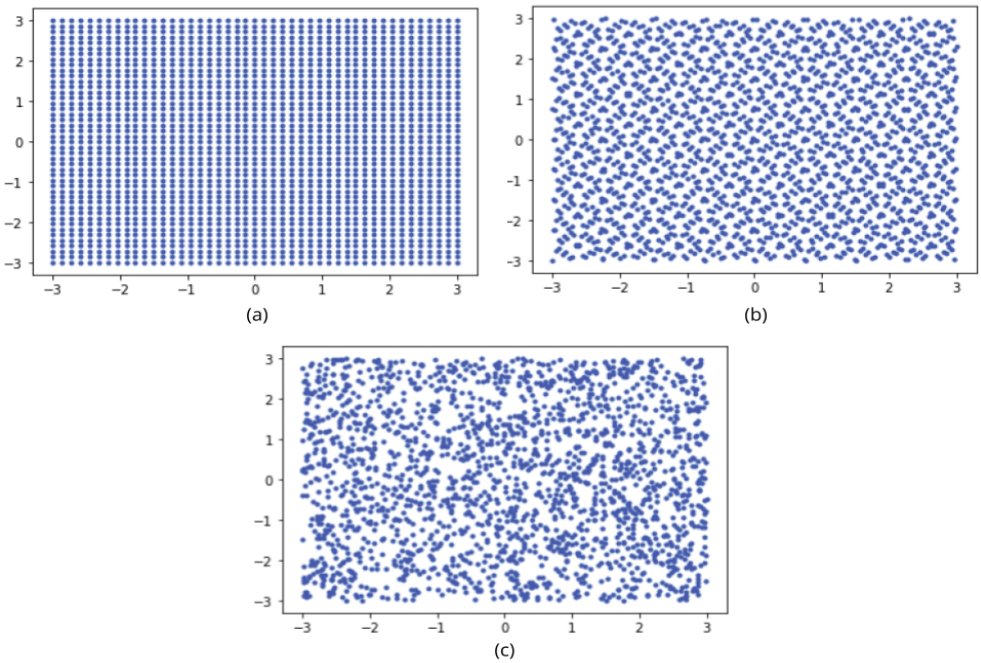

Figure 4. Sample methods represented in 2D Space with vectors $(x, y) \in [-3, 3]^2$.

2.2 Code Implementation

In this report, we coded in *Python* to implement ANNs on which we realized tests on each ground truth. Experiments were run on a separate server for performance, while the data collected after those experiments were analyzed locally using *Jupyter Notebook*. The Deep Learning Library *PyTorch* was mainly used throughout the project; the entire code work can be found on a GitHub repository.

3 Main developments and tests

Based on the studied parameters previously defined, we have to derive relevant tests for us to observe their impact on network prediction, correlating or not the changes in node values with it. As this report is almost solely based on empirical evidence, we perform lots of statistical tests. The implementation is always designed with a heuristic approach in finding the "right" network parameters corresponding to each presented function, so that the network studied isn't obsolete in the sense that it performs poorly and thus leads to unrelated results. This includes defining the correct network topology, learning rate, number of epochs (how many times the same training point is shown to the network), and activation function. Then, tests are repeated as many times as possible on the same network parameters so as to extract information on controlled environments.

3.1 A first approach with Sine

Our goal in this subsection is to execute tests with an easy function to get the first intuition on the relationship between node values and their impact on the prediction of an unseen sample. For this, we test the evolution of the number of node values outside of range as well as the distance from node values' mean when evaluating unseen inputs $\in \mathcal{U}$. Therefore, we train multiple networks with different topologies and collect data on them. The test parameters are as follows:

Test Index	Train Range	Train Size	Test Range	Test Size	Sampling Method
1	$[-3, 3]$	1,024	$[-6, 6]$	529	*Regular*
2	$[-3, 3]$	1,024	$[-6, 6]$	529	*Uniform*
3	$[-3, 3]$	1,024	$[-6, 6]$	529	*Sobol*
4	$[-6, -4] \cup [-1, 1] \cup [4, 6]$	1,024	$[-6, 6]$	529	*Regular*
5	$[-6, -4] \cup [-1, 1] \cup [4, 6]$	1,024	$[-6, 6]$	529	*Uniform*
6	$[-6, -4] \cup [-1, 1] \cup [4, 6]$	768	$[-6, 6]$	529	*Sobol*

For each test index i from $1 - 6$, the networks have *learning rates* of 0.01, *regression parameters* of 10^{-4}, *number of epochs* set to 1000, and *activation functions* defined as the σ Sigmoid function defined earlier. For each i, sixteen different topologies are tested, and seventeen iterations on each topology is carried out in order to maximize the amount of data collected. In total, this gives us 272 networks tested per test. As we proceed to feed a test sample of size 529 for each network, we then have a total of $143,888$ test points from which we can make analyses per test index i.

The topologies tested can be found in the code provided with the github above, as it is an extensive list. For simplicity issues, we defined and refer to the tested networks based on the number of hidden nodes they contain, i.e. this is determined with their *width* and *depth*, which respectively represent the number of nodes per hidden layer and the number of hidden layers in the network. For example, a network with 8 hidden nodes per layer and 3 hidden layers will be denoted as network (8, 3).

3.1.1 Focus on a single network for all inputs

We first want to concentrate on one topology only so that we get a good visualization of the impact of the the number of nodes out-of-range $\mathcal{O}_j$ and the average distance from the mean Δ_j on the accuracy of the network's prediction for each test sample $x_j \in \mathcal{E}$. We also compare the results depending on the sampling method.

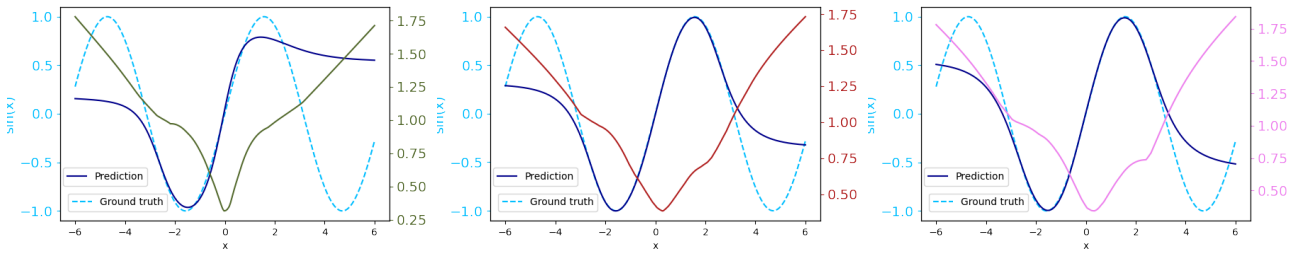

Figure 5. Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, for each sampling method (in order: Regular, Uniform, and Sobol) for tests $1 - 3$.

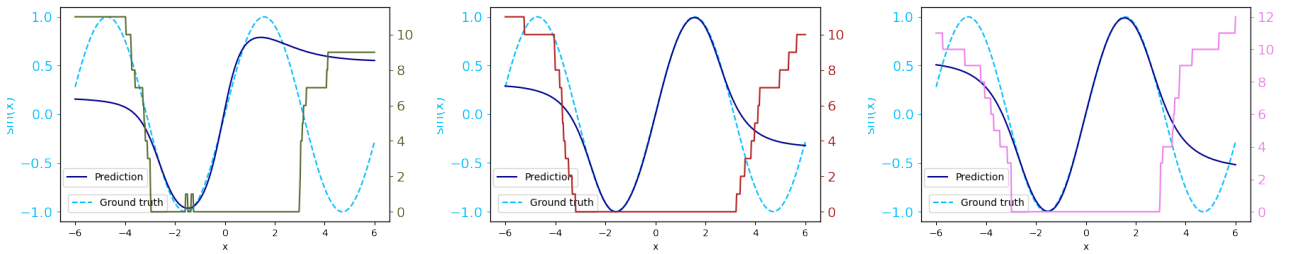

Figure 6. Number of node values out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for each sampling method (in order: Regular, Uniform, and Sobol) for tests $1 - 3$.

We have a first look with the graphs in Figure 5, focusing on a network with *width* 10 and *depth* 3, for test indices $1, 2, 3$. As the test sample $x_j \in \mathcal{E}$ is closer to the limit of the *Train Range*, the Δ_j average distance from the mean increases quite drastically, even when network predictions y_j are still quite accurate. On the other hand, for the number of nodes out-of-range $\mathcal{O}_j$ in Figure 6, it stays null over the whole *Train Range* T. As soon as the test sample x_j is outside of it, $\mathcal{O}_j$ increases drastically as well before stabilizing close to 10, which is a third of the number of hidden nodes in the network. Finally, we do not really observe a difference when it comes to sampling method except for the Sobol Sampling, where we remark that $\mathcal{O}_j$ is about 20% higher on average.

For precision on the labeling of the figures, the upper-right axes depict $\mathcal{O}_j$ and Δ_j's quantities with respect to each x_j for each plot, with their representation being the same color as the axes'.

We now double-check those observations with tests $4, 5, 6$ represented in Figures 7 & 8. The specificity here is that the *Train Range* is split into three areas defined in the table above, leaving holes. Therefore, we see how $\mathcal{O}_j$ and Δ_j "react" to this change.

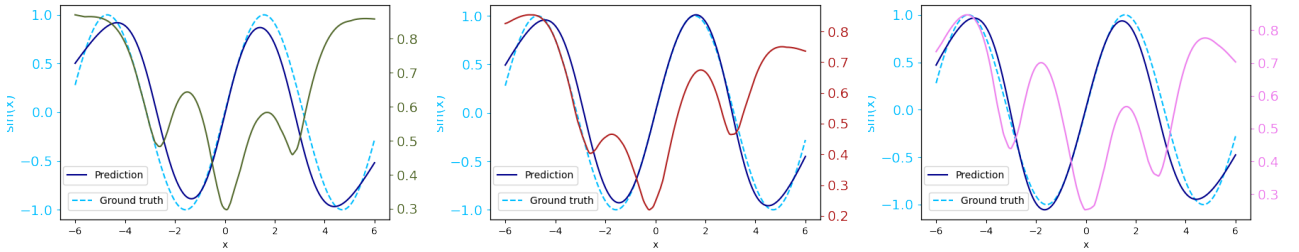

Figure 7. Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, for each sampling method (in order: Regular, Uniform, and Sobol) for tests $4 - 6$.

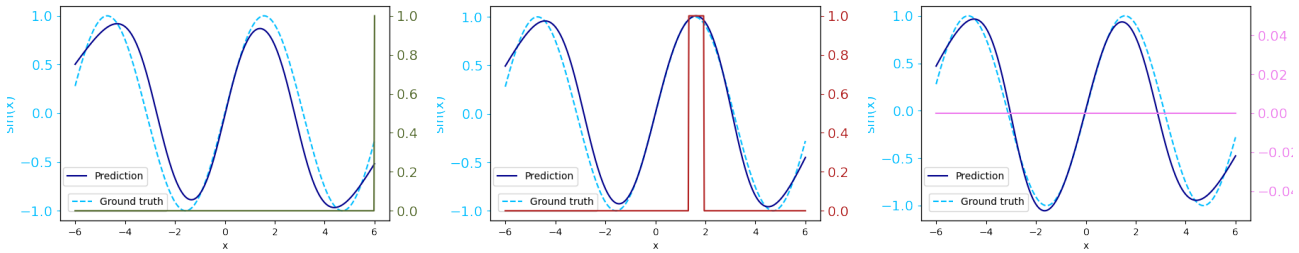

Figure 8. Number of node values out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for each sampling method (in order: Regular, Uniform, and Sobol) for tests $4 - 6$.

The phenomenon we had previously observed does not exactly confirm itself with the Δ_j parameter. We do not specifically find that within T, Δ_j decreases. Rather, at every part of the testing interval where a change of direction can be found (where the peaks are located), Δ_j is high. Furthermore, we observe that as the test samples x_j get closer to the *Train Range*'s maximal and minimal values, the average distance from mean Δ_j explodes, even when within T.

Surprisingly for tests $4 - 6$, $\mathcal{O}_j$ is close to null everywhere, even when the test samples are unseen inputs. One could guess that when the test samples are close to the training range's T suprema, only then does $\mathcal{O}_j$ start growing.

3.1.2 Generalizing with the other Network Topologies on unseen inputs

In this part, our goal is to make some sense out of what we observed previously for this example f_1 but focusing on unseen inputs in $\mathcal{U}$. For this, we collect all the data generated on the unseen inputs x_j that were tested for all different networks NN_i. This includes, their respective error MSE_j, distance from node values' mean Δ_j, and number of node values out-of-range $\mathcal{O}_j$. We then compare MSE_j to Δ_j and MSE_j to $\mathcal{O}_j$ on the same plot. For visualization purposes we show the data and the fitted function for $\mathcal{O}_j$, but only the fitted function for Δ_j.

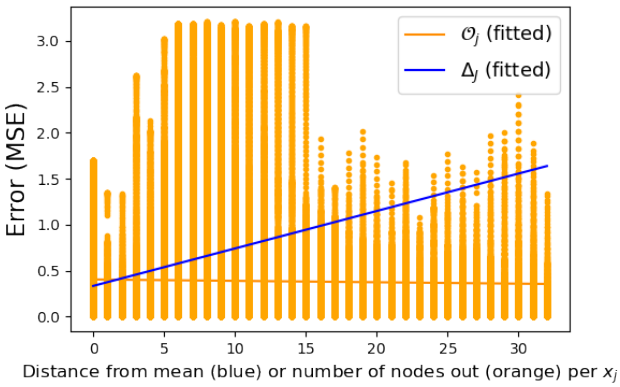

Figure 9. Plot of distance from node values' mean Δ_j vs MSE_j, and number of nodes out $\mathcal{O}_j$ vs MSE_j; per unseen test sample x_j and across all sampling methods.

We notice in Figure 9 that as the average distance from node values' mean increases, the global error for the network increases as well. For the number of nodes outside of range though, this does not seem to be the case. In the two scenarios, the slopes are very flat anyway as the fitted functions are as follows:

$$g_{\Delta}^{NN}(x) = 0.041 \cdot x + 0.333$$
$$g_{\mathcal{O}}^{NN}(x) = -0.002 \cdot x + 0.403$$

Finally, having run tests on $272 \cdot 2 = 544$ different networks NN_i on the sine function f_1 per sampling method, rather than focusing on single test point performance $x_j \in \mathcal{U}$ compared to node values impact as we previously did, we can draw some conclusions from their overall impact on network performances. This will also help us identify similarities and differences in the sampling methods and their influence on node values in the networks.

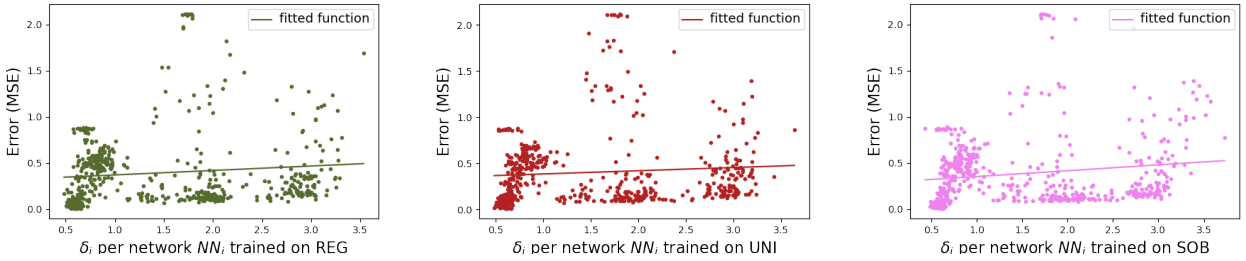

Figure 10. δ_i versus error calculated using *MSE*, per network NN_i and for each sampling method.

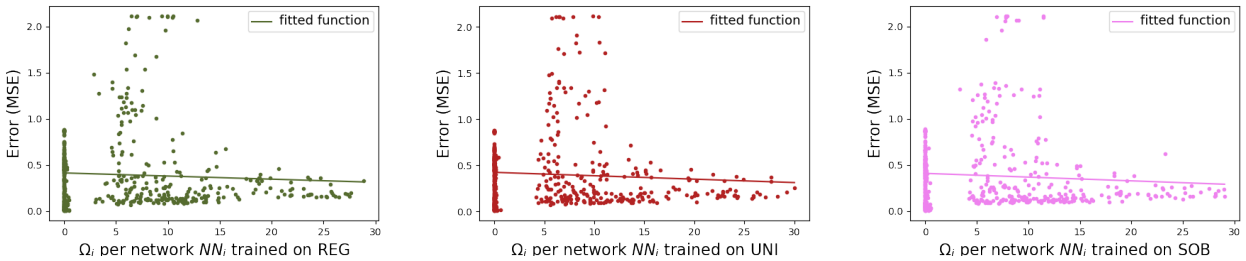

Figure 11. Ω_i versus error calculated using *MSE*, per network NN_i and for each sampling method.

The fitted functions' formulae are as follows:

$$g_{\delta}^{REG}(x) = 0.048 \cdot x - 0.323,\ g_{\delta}^{UNI}(x) = 0.035 \cdot x + 0.351,\ g_{\delta}^{SOB}(x) = 0.063 \cdot x + 0.289$$

$$g_{\Omega}^{REG}(x) = -0.003 \cdot x + 0.414,\ g_{\Omega}^{UNI}(x) = -0.004 \cdot x + 0.425,\ g_{\Omega}^{SOB}(x) = -0.004 \cdot x + 0.408$$

In those results, the slopes are so low that we cannot really conclude anything. What can be said however is that δ_i seems to have a bigger impact than Ω_i on the MSE as a whole. We can also distinguish slight differences in the slope values between sampling methods, with the Sobol sampling method having the biggest slope before the Uniform sampling method and the Regular one.

To conclude those preliminary tests, we do not yet have a good idea that our intuition was in fact well founded, at least using this example on a simple function like sine, but we are getting there. The difference between the sampling methods is not notable, the slopes and y-intercepts are extremely similar.

3.2 Testing on a Composed 1-Dimensional Input Function

Now that we have tested on a relatively easy to predict function, we have to increase the difficulty in order to get closer to real-life applications. We thus test with a composed function previously defined by f_2. The particularity with f_2 is that it is not continuous and is defined with three different functions on its domain of definition. Intuitively, we expect to see more out-of-range node values and greater distances from the mean as adding difficulty will decrease the accuracy, and thus increase Neural Networks' error. It is important to note that the definition of f_2 was chosen such that its studied range will not be centered on zero, so as to make sure that it isn't the absolute value of the x_j which have an impact on the node values.

As before, we define six different tests labeled by indices $i \in \{1, \dots, 6\}$.

Test Index	**Train Range**	**Train Size**	**Test Range**	**Test Size**	**Sampling Method**
1	$[12.5, 18.5]$	1,024	$[9.5, 21.5]$	529	*Regular*
2	$[12.5, 18.5]$	1,024	$[9.5, 21.5]$	529	*Uniform*
3	$[12.5, 18.5]$	1,024	$[9.5, 21.5]$	529	*Sobol*
4	$[9.5, 11] \cup [14, 17] \cup [20, 21.5]$	1,024	$[9.5, 21.5]$	529	*Regular*
5	$[9.5, 11] \cup [14, 17] \cup [20, 21.5]$	1,024	$[9.5, 21.5]$	529	*Uniform*
6	$[9.5, 11] \cup [14, 17] \cup [20, 21.5]$	1,024	$[9.5, 21.5]$	529	*Sobol*

For each test index i from $1 - 6$, the networks have the same initial parameters as before. They have *learning rates* of 0.01, *regression parameters* of 10^{-4}, *number of epochs* set to 1000, and *activation functions* defined as the σ Sigmoid function defined earlier. For each i, sixteen different topologies are tested, and seventeen iterations on each topology is carried out in order to maximize the amount of data collected. In total, this gives us 272 networks tested per test. As we proceed to feed a test sample of size 529 for each network, we then have a total of $143,888$ test points from which we can make analyses per test index i.

3.2.1 Focusing on a single network

First, we proceed with a local examination of the phenomenons for *Test Indices* $1 - 3$. In continuity of our analyses, we will start by focusing on the trained networks with *width* 10 and *depth* 3, similar analyses can be made using other topologies. Moreover, to reduce the number of graphs, we will only show results computed with the Uniform Sampling method in this part. The other sampling methods' results can be found in Figures 29-32 in the appendix.

In Figure 12, we remark that as soon as the test sample x_j is outside of the *Train Range*, the number of nodes outside of range increases drastically. Nonetheless, it appears that $\mathcal{O}_j$ increases less rapidly on $[18.5, 21.5]$ than on $[9.5, 12.5]$, which might be due to the fact that the predictions are extremely close to the ground truth on the first interval $[18.5, 21.5]$.

Figure 12 shows us that even though the quantitative and qualitative assessments of the node values are quite high comparatively, the error on the interval $[18.5, 21.5]$ is really low. We can guess that since by chance f_2 is defined such that it goes in a near straight line, even though the network does not know what to predict, it happens to still be correct as it also goes in a straight line, a phenomenon which can be observed on the untrained interval of $[9.5, 12.5]$.

This puts into perspective the way the evaluation of what a "good" prediction from the tested network is done, which is calculated with the Mean Squared Error. Here, even though the Mean Squared Error is low on $[18.5, 21.5]$, we can clearly

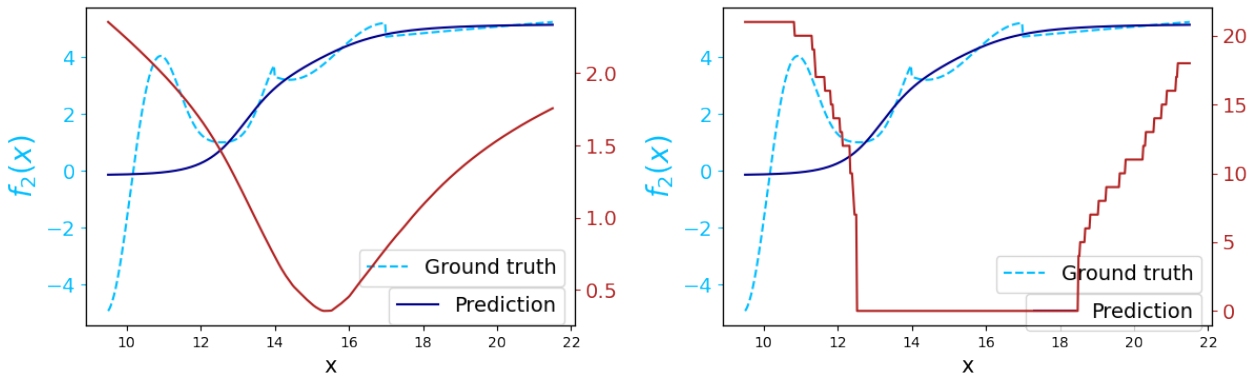

Figure 12. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 2.

notice that the network just happens to get it right. The same can be said with points around 10.5, where the network prediction crosses the ground truth. One can guess that, as the network is evaluated outside of its known ranges, it just continues in the direction the previous known inputs were going, before tending to a straight line.

We see with Figure 13 that Δ_j clearly increases near the bounds ∂T of the *Train Range* for test index 5, and even starts to gradually increase within the *Train Range*. Surprisingly, we do not necessarily see a high distance from the node values' mean at the intervals which were not trained on (i.e. $[11, 14] \cup [17, 20]$). In fact, the first interval oddly corresponds to the area where the distance from the mean is the lowest, and for all types of sampling methods.

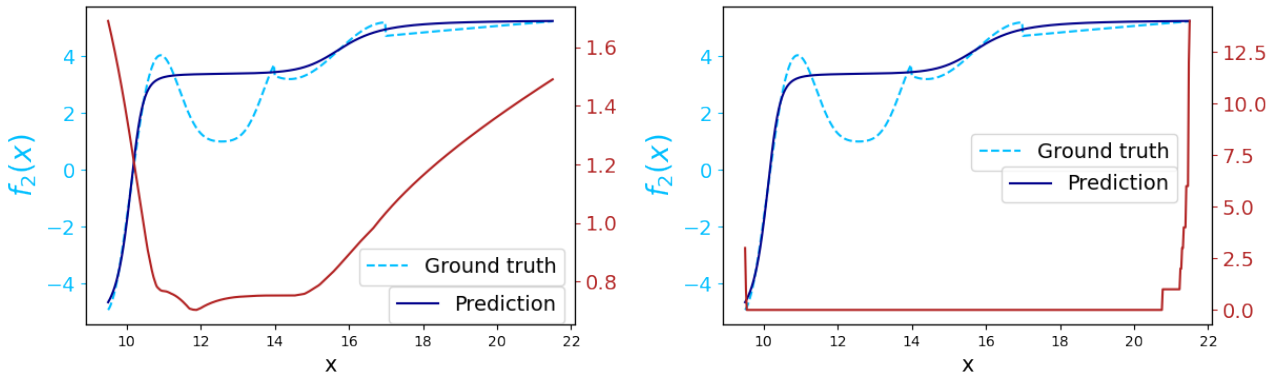

Figure 13. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 5.

We see that even though the test samples are not in the *Train Range* and the network's prediction is bad, the number of nodes outside of range is null. The prediction entirely fails to grasp the local minimum around $x = 13$ of the ground truth. This could be due to a lack of network parameters and hidden layers. However, we can declare that as the test samples x_j get closer to the extremum of the *Train Range*, $\mathcal{O}_j$ consequently increases. We can thus say that the number of nodes outside of range strongly increases as the test samples are close to the *Train Range*'s bounds ∂T for our example.

3.2.2 Generalizing with other Network Topologies on unseen inputs

Once again, we focus on unseen inputs in $\mathcal{U}$ for all the networks NN_i tested in this subsection, independently of the sampling method. Repeating the same process as in the previous subsection concerning f_1, we get a plot of the evolution

of $\mathcal{O}_j$ versus the error MSE_j, and of Δ_j versus MSE_j for each unseen sample x_j, represented in Figure 14.

The fitted functions' formulae are as follows:

$$g_{\mathcal{O}}(x) = 0.291 \cdot x + 2.300$$

$$g_{\Delta}(x) = 1.311 \cdot x + 1.677$$

As before, Δ_j seems to have a bigger impact on the error than $\mathcal{O}_j$, as its slope is greater, and this time with a comparatively big slope. We can notice that the fitted line for $\mathcal{O}_j$ is close to being flat again, which might tell us that its impact might not be as big as we thought.

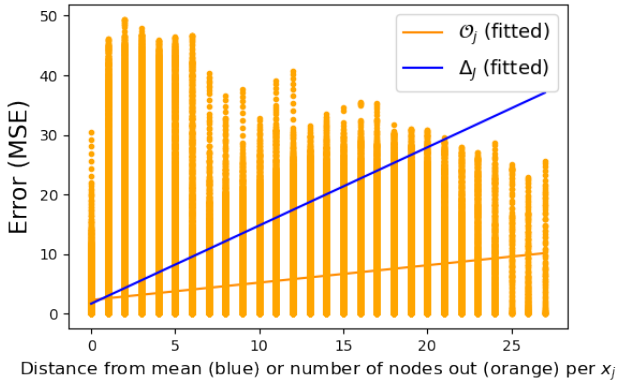

Figure 14. Plot of distance from node values' mean Δ_j vs MSE_j, and number of nodes out $\mathcal{O}_j$ vs MSE_j; per unseen test sample x_j and across all sampling methods.

Our observations for each unseen sample $x_j \in \mathcal{U}$ on Δ_j can be corroborated by the fact that if we look at δ_i corresponding to networks NN_i for each sampling method, we get a similar result.

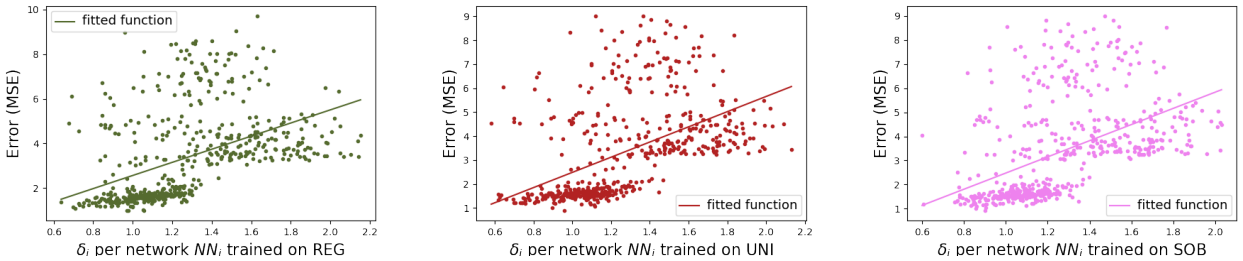

Figure 15. δ_i versus error calculated using *MSE* per network NN_i, for each sampling method.

Below are the fitted lines' formulae:

$$g_{\delta}^{REG}(x) = 2.939 \cdot x - 0.376,\ g_{\delta}^{UNI}(x) = 3.169 \cdot x - 0.684,\ g_{\delta}^{SOB}(x) = 3.356 \cdot x - 0.877$$

$$g_{\Omega}^{REG}(x) = 0.226 \cdot x + 2.544,\ g_{\Omega}^{UNI}(x) = 0.234 \cdot x + 2.455,\ g_{\Omega}^{SOB}(x) = 0.231 \cdot x + 2.543$$

Once again, δ_i seems to have a bigger impact on the Sobol sampling method, as its slope is bigger than that of the Uniform and Regular samplings methods. On the other hand, the linear regression results are extremely close between all the sampling methods for Ω_i, therefore we cannot make any differences. However, the slopes are positive for qualitative *and* quantitative assessment, which still marks the effect they have on the network performances on unseen input in our example.

Therefore, from this subsection's results on f_2, it seems that the more the qualitative & quantitative assessments on node values increase, the bigger the Mean Squared Error is for networks on unseen samples. Of course, this can only apply to this specific test.

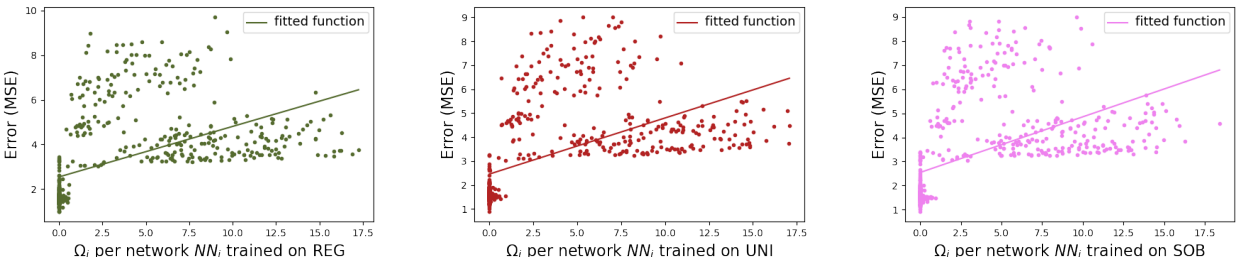

Figure 16. Ω_i versus error calculated using *MSE* per network NN_i, for each sampling method.

3.3 Testing on a Continuous 2-Dimensional Input Function

We continue our experiments, yet again increasing the difficulty of the ground truth by taking it to be f_3. Our hope is that adding some complexity might also give us more insights into the impact the sampling methods have on the node values. However, since the tests on these type of functions need much more computing power as we increase the number of samples on which the network is trained as well as the number of test samples, we decided to only collect node values in the last 10% of epochs throughout training. Even though this was mainly done for technical reasons, this approach makes sense as for complicated functions to predict, the initial network error in early epochs might be extremely high and thus the node values would not represent what the completely trained network produces when it is done learning. Accordingly, we get a refined analysis on the node values as only the ones which are closer to the end of the training are taken into account.

Below we introduce a table showing, as always, the 6 tests that were conducted on f_3.

Test Index	Train Range	Train Size	Test Range	Test Size	Sampling Method
1	$[-1,1]^2$	16,384	$[-2,2]^2$	9,216	*Regular*
2	$[-1,1]^2$	16,384	$[-2,2]^2$	9,216	*Uniform*
3	$[-1,1]^2$	16,384	$[-2,2]^2$	9,216	*Sobol*
4	T^3_{hole}	16,485	$[-2,2]^2$	9,216	*Regular*
5	T^3_{hole}	16,501	$[-2,2]^2$	9,216	*Uniform*
6	T^3_{hole}	21,847	$[-2,2]^2$	9,216	*Sobol*

We show what T^3_{hole} looks like for reasons of visualization in the appendix, Figure 33.

This time around, each network is trained with those given parameters: they have *learning rates* of 0.01, *regression parameters* of 10^{-4}, *number of epochs* set to 2000, and *activation functions* defined as the σ Sigmoid function. For each i, sixteen different topologies are tested, and eight iterations on each topology is carried out in order to maximize the amount of data collected. In total, this gives us 128 networks tested per test. As we proceed to feed a test sample of size $9,216$ for each network, we then have a total of $1,179,648$ test points from which we can make analyses per test index i.

3.3.1 Focusing on a single network

Again, we start-off with a network composed of *width* 10 and *depth* 3 hidden node architecture. Similar analyses can be made with the other network topologies, this architecture was merely chosen for facility. Furthermore, we will only show pictures generated with the *Sobol sampling* method for brevity, but the results computed with the other methods can be seen in the appendix at Figures 34 & 35.

The predictions seen in Figure 17 from network $(10,3)$ are not exactly accurate, especially when comparing test number 6's prediction to test number 3's. If we compare the distribution of $\mathcal{O}_j$ versus test samples x_j for each test index 3 & 6 on Figure 18, we observe that test 6 practically has no nodes out-of-range except right outside the extremities of the train space. On test 3's side, we see almost exactly no nodes out of range on the *Train Range* $= [-1,1]^2$, but a lot in

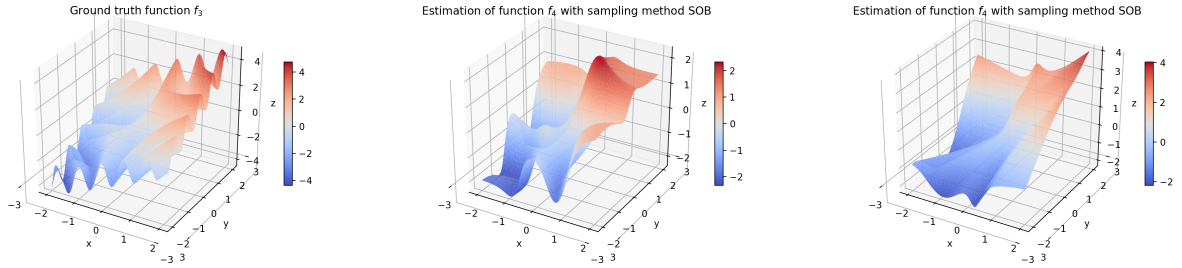

Figure 17. From left to right, plot of the ground truth f_3, prediction from network (10, 3) of f_3 for test index 3, prediction from network (10, 3) of f_3 for test index 6.

the upper right and lower left of the *Test Range* $\mathcal{E}$, where the values of f_3 are almost at their maximum and respectively minimum on $\mathcal{E}$.

Spreading the train data on the whole *Test Range* like done in test indices $4-6$ seems to reduce the number of nodes out, while when the *Train Range* T is grouped together like in test indices $1-3$, $\mathcal{O}_j$ grows rapidly outside it. This is an observation which we have already made with the previous examples, and it to confirms itself there.

Now, if we compare the distance from node values' mean Δ_j for each test 3 & 6 in Figure 19, we have two similar pictures, except that Δ_j is smaller overall for test index 6, which corroborates our reflection that spreading the *Train Range* over the *Test Range* yields better spreading of node values as a whole for unseen inputs.

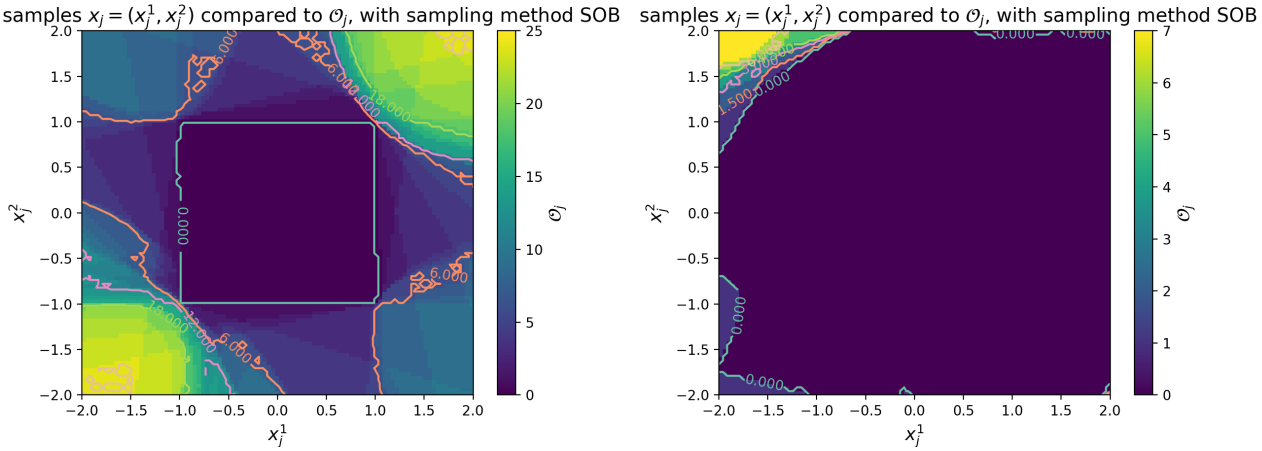

Figure 18. From left to right, $\mathcal{O}_j$ compared to test samples x_j for test index 3, $\mathcal{O}_j$ compared to test samples x_j for test index 6, each for Neural Network (10, 3).

We notice a similarity with our former analyses on Δ_j in that the distance from mean Δ_j for test 3 is low in the *Train Range* while it increases as soon as the test samples x_j are outside of it or close to its bounds. Likewise, our previous analyses suggested that Δ_j was affected by the ground truth's shape. We observe, more notably on test 6's graph in Figure 19, that it is the case when looking at peaks which coincide with the ground truth's graph in Figure 17.

Let us determine whether having better node distribution indeed yields better network performance for this example. We do this by generalizing on all of the data that we have collected for unseen inputs on the studied function: the more the data, the better the observations.

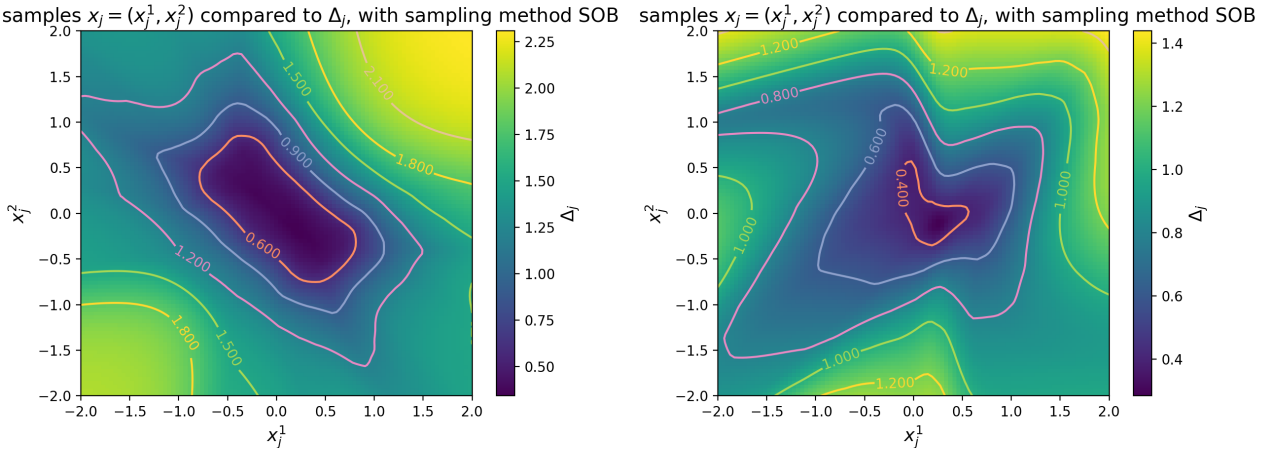

Figure 19. From left to right, Δ_j compared to test samples x_j for test index 3, Δ_j compared to test samples x_j for test index 6, each for Neural Network (10, 3).

3.3.2 Generalizing with the other Network Topologies on unseen inputs

Invariably, this section concentrates on the impact of the qualitative and quantitative parameters of node values on performance for unseen inputs. We look at all the unseen inputs $x_j \in \mathcal{U}$ throughout all the test indices $1-6$ and compare the number of out-of-range $\mathcal{O}_j$ node values they generated, then fit a function on this data. We obtain a plot found in Figure 20, which also includes the fitted function for the evolution of MSE compared to the average distance from mean Δ_j for the unseen samples.

The fitted functions are as follows:

$$g_{\mathcal{O}}(x) = 0.067 \cdot x + 0.896$$

$$g_{\Delta}(x) = 0.558 \cdot x + 0.499$$

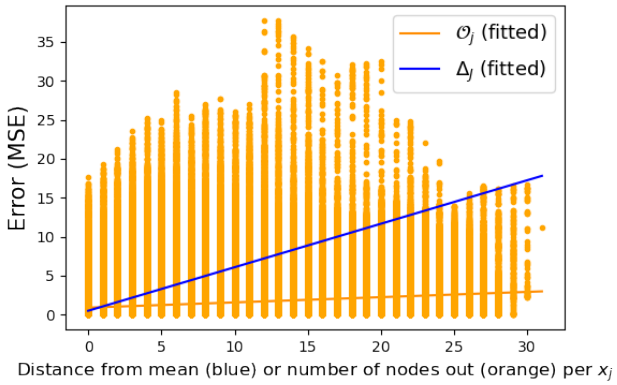

Figure 20. Plot of distance from node values' mean Δ_j vs MSE_j, and number of nodes out $\mathcal{O}_j$ vs MSE_j; per unseen test sample x_j and across all sampling methods.

In continuity of our previous observations, we again see that the average distance from mean Δ_j tends to have a bigger impact on the prediction accuracy of unseen input x_j than $\mathcal{O}_j$. Although the fitted function for $\mathcal{O}_j$ is almost flat, both fitted functions are increasing, which reassures us once more in our main hypothesis' belief.

Now, we concentrate on each network NN_i performance versus their average distance from node values' mean which we defined as δ_i. The plots on Figure 21 & 22 were generated from data only taking into account the test samples that were not seen during training and thus outside of *Train Range*, for all test indices $1-6$. We can make $128 * 2 = 256$ observations on each type of sampling methods to try and distinguish differences.

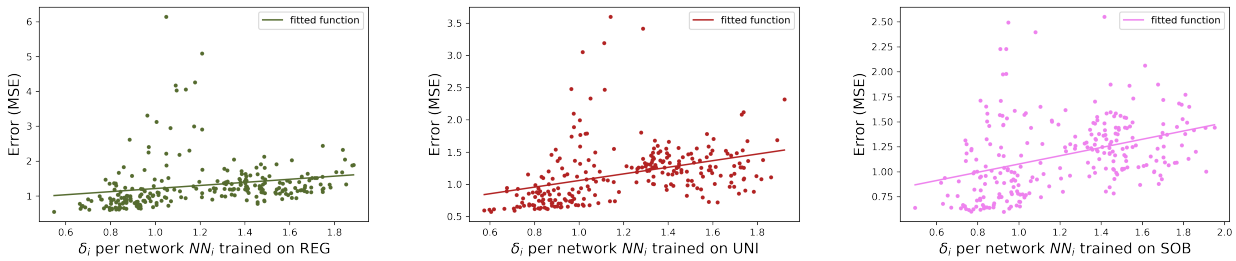

Figure 21. δ_i versus error calculated using *MSE*, per network NN_i and for each sampling method.

The fitted functions for each of the plots are as follows:

$$g_{\delta}^{REG}(x) = 0.444 \cdot x + 0.768,\ g_{\delta}^{UNI}(x) = 0.513 \cdot x + 0.545,\ g_{\delta}^{SOB}(x) = 0.413 \cdot x + 0.664$$
$$g_{\Omega}^{REG}(x) = 0.022 \cdot x + 1.226,\ g_{\Omega}^{UNI}(x) = 0.025 \cdot x + 1.066,\ g_{\Omega}^{SOB}(x) = 0.026 \cdot x + 1.065.$$

They are all increasing, showing the presence of a correlation for this example between the error MSE and the average distance from mean δ_i for each network NN_i. There is a slight difference between the slopes of the fitted functions depending on the sampling method, with the *Uniform sampling* having the largest slope this time around.

The same graphs generated with Ω_i are below, the respective fitted functions are once again extremely flat, questioning whether there really is a significantly strong relation between Ω_i and a network's error MSE on all the testing inputs.

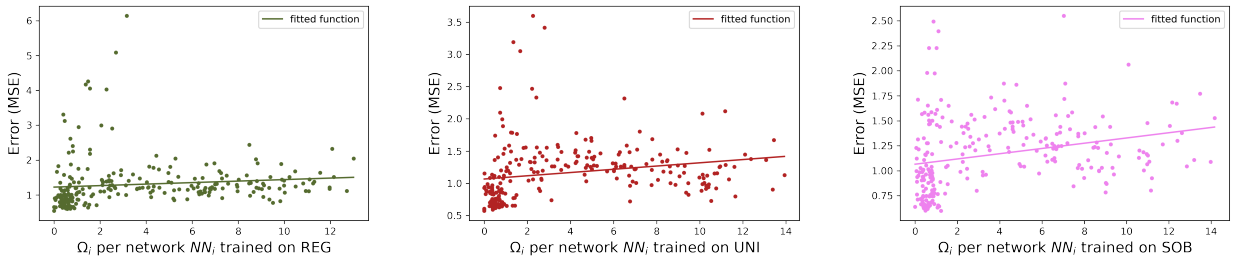

Figure 22. Ω_i versus error calculated using *MSE*, per network NN_i and for each sampling method.

3.4 Testing on a Composed 2-Dimensional Input Function

Finally, we test on a really complicated-to-predict function in comparison to the size of the ANNs we are training. f_4 is defined to be a discontinuous function which takes inputs $x \in [-3, 3]^2$ and outputs results in $\mathbb{R}$. This is the closest we are going to get in our analysis in terms of closeness to a real phenomenon's complexity. The amount of training samples has been increased accordingly, and we have focused our observations on a holed training set named T_{hole}^4 which can be visualized in the appendix at Figure 33.

We define tests with indices from $1 - 6$.

Test Index	Train Range	Train Size	Test Range	Test Size	Sampling Method
1	$[-3,3]^2$	32,768	$[-3,3]^2$	22,500	*Regular*
2	$[-3,3]^2$	32,761	$[-3,3]^2$	22,500	*Uniform*
3	$[-3,3]^2$	32,761	$[-3,3]^2$	22,500	*Sobol*
4	T^4_{hole}	44,032	$[-3,3]^2$	22,500	*Regular*
5	T^4_{hole}	43,681	$[-3,3]^2$	22,500	*Uniform*
6	T^4_{hole}	43,691	$[-3,3]^2$	22,500	*Sobol*

The same network architectures as in the previous subsection are trained with the same initial parameters. The changes are the amounts of training and testing samples, which are respectively given in the table above. As we proceed to feed a test sample of size $22,500$ for each network and we are training 128 networks per test index i, we then have a total of $2,880,00$ test points from which we can make analyses.

3.4.1 Focusing on a single network

In coherence with our previous analyses, let us focus on a network with *width* 10 and *depth* 3 hidden node architecture. We also choose to only produce pictures and analyses on test indices 1 & 4 in this subsection for clarity. Similar remarks can be made on the other sampling methods, their respective images can be found in the appendix at Figures 36 & 37.

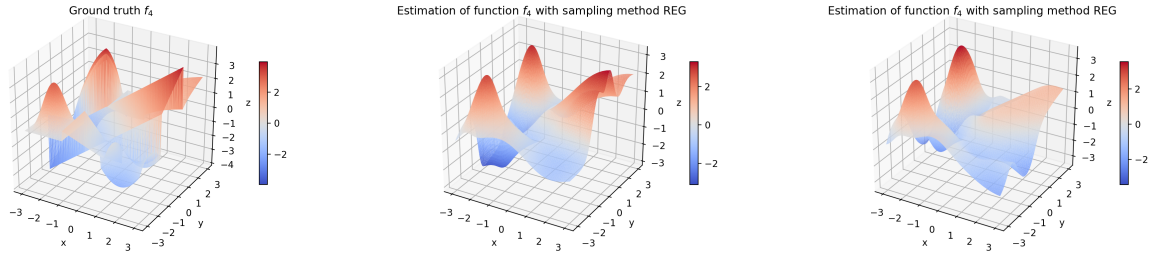

Figure 23. From left to right, plot of the ground truth f_4, prediction from network (10, 3) of f_4 for test index 1, prediction from network (10, 3) of f_4 for test index 4.

First, let us analyze with test 1 how the number of nodes out-of-range $\mathcal{O}_j$ looks like compared to the test samples x_j. This is important as we have continuously seen in the previous functions f_1, f_2, & f_3 that $\mathcal{O}_j$ is always very low when inside the *Train Range*, regardless of the error, and in this specific test case $T = \mathcal{E}$.

As we can see in Figure 24, the error is obviously increasing when on very complicated to predict areas of function f_4, but it stays low overall. Furthermore, $\mathcal{O}_j$ is null on almost the entire *Test Range*, which confirms our expectations.

Now, we focus on test 4 as test 1 does not produce data on unseen inputs by the network since the *Test Range* equals the *Train Range*. Figure 25 substantiates our findings regarding $\mathcal{O}_j$ when outside of the *Train Range*: $\mathcal{O}_j$ automatically increases when not in it.

Additionally, we recognize that the error somewhat matches the areas where $\mathcal{O}_j$ is high, except on parts of the function f_4 that are relatively flat and continuous. We can guess the network simply continued in the "direction" of the last seen sample's prediction when it did not know what to do, so that the error is minimized. We can see this is the case by looking at Figure 23's third representation of the network prediction on f_4.

As always the mean distance from node values Δ_j is really low within the center of the *Train Range* and increases when reaching the bounds of it. We can also see that it "reacts" more than $\mathcal{O}_j$, in that it increases more rapidly when the unseen samples' errors increase and it also increases depending on the aspect of the function. We can observe this on the two peaks in the lower & upper left parts of the *Testing Range*, where the actual output of f_4 are also peaks which are hard to predict for the network.

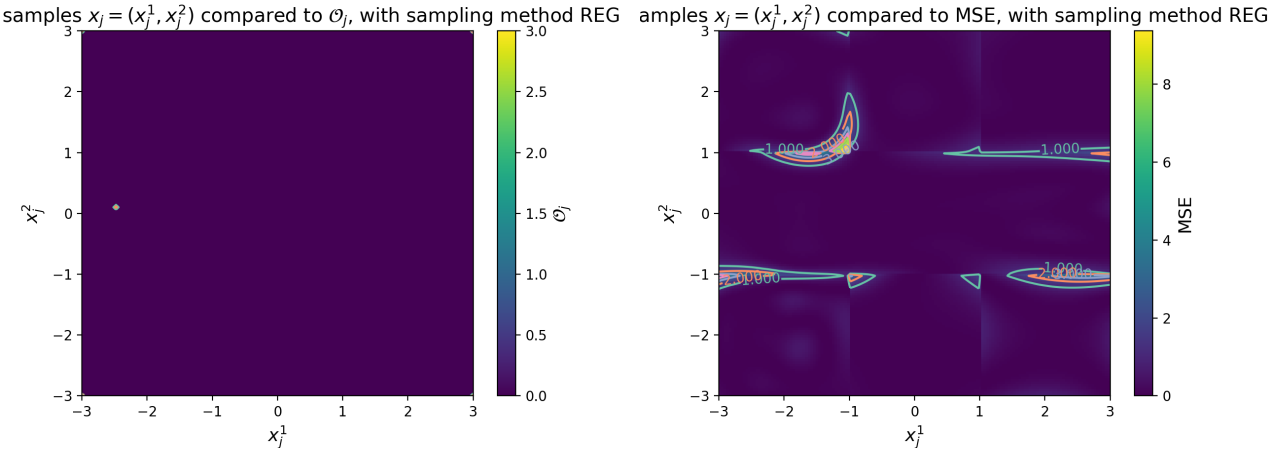

Figure 24. From left to right, $\mathcal{O}_j$ compared to test samples x_j for test index 1, MSE compared to test samples x_j for test index 1, each for Neural Network (10, 3).

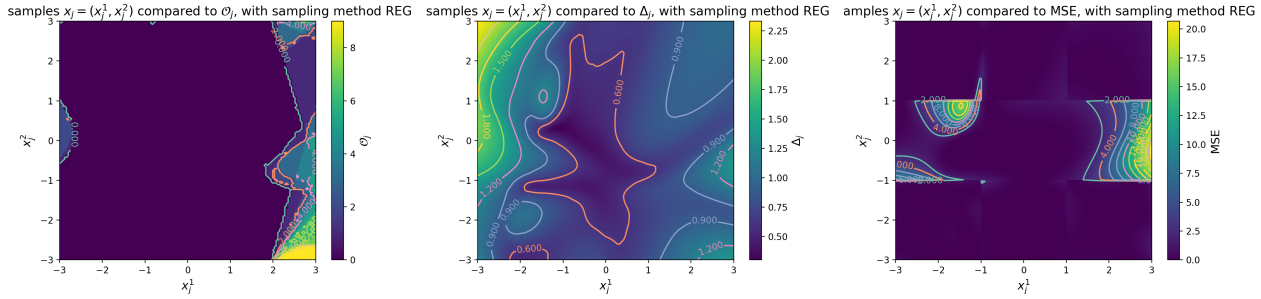

Figure 25. From left to right, $\mathcal{O}_j$ compared to test samples x_j, Δ_j compared to test samples x_j, MSE compared to test samples x_j, each for Neural Network (10, 3) and test index 4.

3.4.2 Generalizing with the other Network Topologies on unseen inputs

As the test indices from $1-3$ have the same *Train Range* and *Test Range*, we will not be considering the data collected on them for this section. Indeed, the goal of this section is to see the overall effect of $\mathcal{O}_j$ and Δ_j as well as δ_i and Ω_i for unseen inputs x_j per network NN_i, i.e. inputs not found in the *Train Range*. Therefore, we have collected all of the data for each sampling method and test index $4-6$.

Now, if we solely direct our attention towards unseen inputs and not network performance in general, we can also gain insights on the prediction quality of a single unseen sample x_j thanks to its distance from node values' mean Δ_j it generated as a whole. The respective fitted functions for Figure 26 are as follows:

$$g_\Delta(x) = 0.325 \cdot x + 2.912 \text{ and } g_\mathcal{O}(x) = -0.0782 \cdot x + 3.308$$

The fitted slope for $\mathcal{O}_j$ is negative and really small; the fitted line is almost flat and thus we conclude that not a lot can be determined from this parameter for our example.

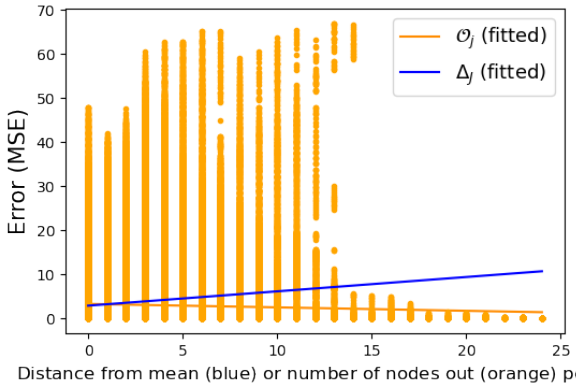

Figure 26. Plot of networks' NN_i distance from node values' mean vs MSE, and number of nodes out vs MSE; per unseen test sample x_j and across all sampling methods.

As before, we now have a look at Ω_i and δ_i. For the plots on Ω_i, the fitted functions have smaller slopes than the ones we are seeing for δ_i. In fact, here are the fitted linear functions for Ω_i and δ_i:

$$g_{\delta}^{REG}(x) = 3.102 \cdot x - 0.229,\ g_{\delta}^{UNI}(x) = 0.208 \cdot x + 3.266,\ g_{\delta}^{SOB}(x) = 3.572 \cdot x + 0.262$$

$$g_{\Omega}^{REG}(x) = 0.208 \cdot x + 2.347,\ g_{\Omega}^{UNI}(x) = 0.150 \cdot x + 3.263,\ g_{\Omega}^{SOB}(x) = 0.212 \cdot x + 3.291$$

For our example, it seems that the accuracy of the output produced by a network NN_i can be deduced by the average distance from node values' mean δ_i it produces, more than by the average number of node values out-of-range Ω_i it produces on unseen inputs. Even though this needs to be put in perspective as the slopes for δ_i are quite flat when using a uniform sampling method.

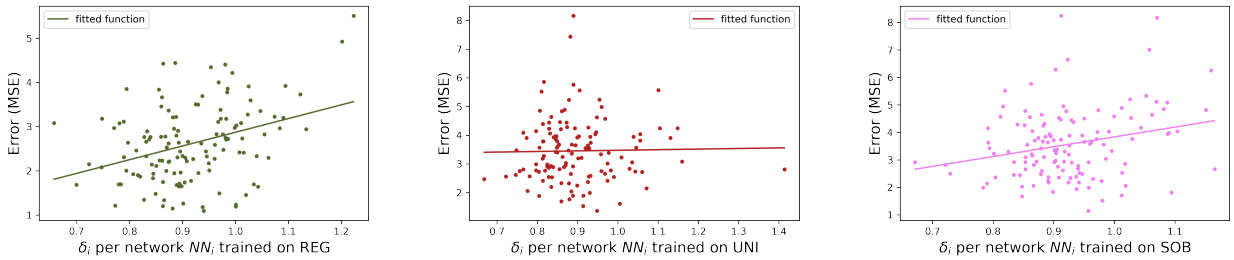

Figure 27. δ_i versus error calculated using *MSE*, per network NN_i and for each sampling method in tests $4-6$.

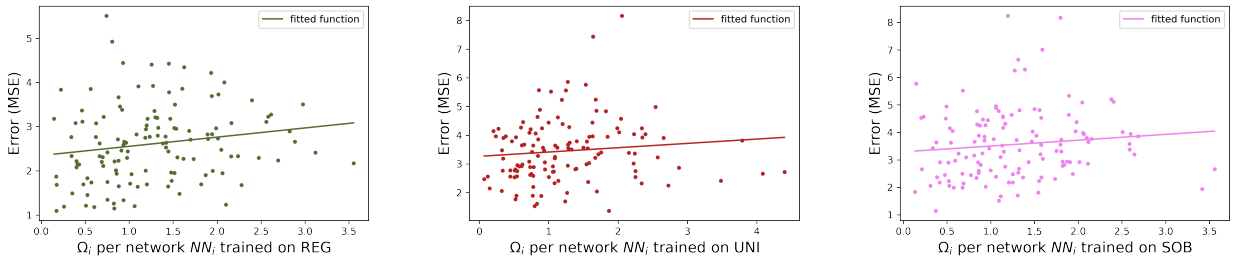

Figure 28. Ω_i versus error calculated using *MSE*, per network NN_i and for each sampling method in tests $4-6$.

4 Conclusion

Now that we have proceeded in a thorough investigation of four different ground truths, we can come to conclude on the possible generalities derived from our observations, and reflect on future research on the topic. Although, we have to keep in mind that everything done in this thesis is based on empirical evidence, itself generated on very few ground truths and trained networks which do not represent the infinite possibilities there could be with function prediction using basic ANNs. The work done here was merely a first exploration of our intuition.

4.1 Interpretation of the results

In this subsection, we discuss the extents of our investigation and what we can learn from it. We consider results that were expected and that turned out to be false, and the opposite.

Let us start with the the difference in sampling method. We selected three different sampling methods in order to choose training inputs in a disparate way so that we could eventually detect differences in node values distribution and their qualitative or quantitative associated parameters. It seemed reasonable to believe that such a difference between the Sobol Sampling method and the Regular Sampling method for example, where the Sobol method fills its space much quicker than the Regular one, could have an impact. However, from the data we studied, no striking difference could be established. Indeed, when looking at the plots made, the evolution of the node values out-of-range and the distance from node values' mean seem to, overall, behave the same way when focusing like we did on unseen inputs.

Moreover, our initial hypothesis on node values out of range is somewhat refuted. Indeed, we observed throughout our analyses and tests that the number of nodes out-of-range $\mathcal{O}_j$ per sample x_j tended to increase only when outside the suprema of the training set ∂T, while when x_j is inside it is almost always close to null. Therefore, for unseen input which tend to be more prone to generate high error, the impact was almost the same independently of the amount of nodes out-of-range, even though we found in three out of four cases that the slope of the fitted linear function to our data was still positive. The relation is simply not strong enough in the examples we produced to determine a correlation. This implies that, in our case, the impact of the number of node values out-of-range on the generated error by networks for unseen inputs is limited.

On the other hand, our data suggests that the average distance from the mean Δ_j does have an impact for network prediction on unseen inputs $x_j \in \mathcal{U}$. Even though Δ_j is generally pretty low when the tested inputs are inside T, when the inputs are taken in $\mathcal{U}$, Δ_j seems to slightly behave depending on the function's shape. When guessing hard to predict areas of the function (e.g. usually sudden peaks, or valleys), Δ_j increases, which is good as the prediction areas that are more prone to high error are those areas. But, just like $\mathcal{O}_j$, it also "reacts" depending on whether the tested input x_j is in T or $\mathcal{U}$. All those observations, and the data that was generated on networks' NN_i prediction error MSE compared to δ_i, lead us to conclude that for our examples there is a correlation between the average distance from node values' mean Δ_j and unseen samples' $x_j \in \mathcal{U}$ error MSE_j upon prediction. The more Δ_j increases, the more prone to error x_j's prediction is likely to be.

4.2 Further Works

As previously mentioned, this study did not even scratch the surface of the infinite possibilities the world of function prediction has. Therefore, to have more conclusive results and better empirical evidence to base our claims on, one would need to test on much larger networks, with much more complicated functions, and different network parameters. In this report, only analyses on somewhat simple functions and shallow Neural Networks were done.

Furthermore, tests on other types of applications for ANNs could also be derived, such as for classification using image analysis (e.g. MNIST or CIFAR-10 datasets), pattern recognition, or even clustering. Nevertheless, one cannot pursue to test every ground truth indefinitely, and thus a Mathematical proof of the results would need to come in play in order to get tangible results on the matter.

5 References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer Cham, 08 2018.

[2] Andrew Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14:115–133, 01 1994.

[3] Eric B. Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[4] Yann N. Dauphin, Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014.

[5] Thomas Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27:326–327, 09 1995.

[6] Lizelle Fletcher, Vladimir Katkovnik, François E. Steffens, and Andries Petrus Engelbrecht. Optimizing the number of hidden nodes of a feedforward artificial neural network. *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, 2:1608–1612, 06 1998.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] Benjamin D. Haeffele and René Vidal. Global optimality in neural network training. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4390–4398, 07 2017.

[9] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

[10] Sara Kaviani and Insoo Sohn. Influence of random topology in artificial neural networks: A survey. *ICT Express*, 6(2):145–150, 2020.

[11] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133, 1943.

[12] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1988.

[13] Siddhartha Mishra and T. Konstantin Rusch. Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. *CoRR*, abs/2005.12564, 2020.

[14] Chris Roadknight, Dominic Palmer-Brown, and David Al-Dabass. Simulation of correlation activity pruning methods to enhance transparency of artificial neural networks. *International Journal of Simulation: Systems, Science and Technology*, 4, 01 1997.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[16] I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[17] Bruno Tuffin. On the use of low discrepancy sequences in monte carlo methods. In *Monte Carlo Methods Appl.*, volume 2, pages 295–320, 1996.

A Appendix

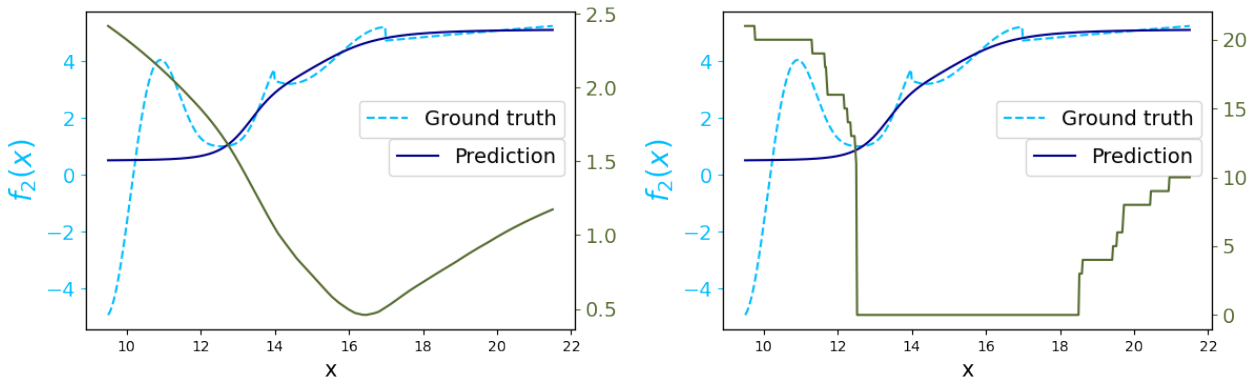

Figure 29. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 1 in f_2.

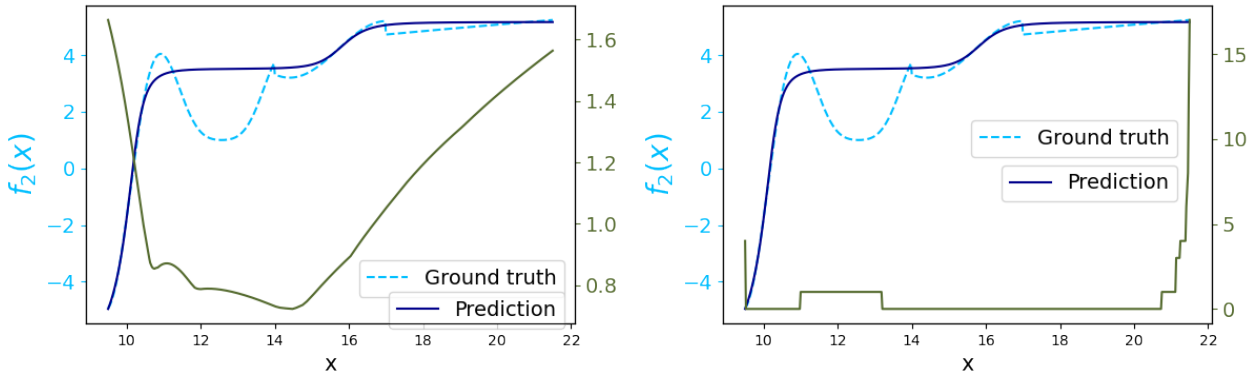

Figure 30. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 4 in f_2.

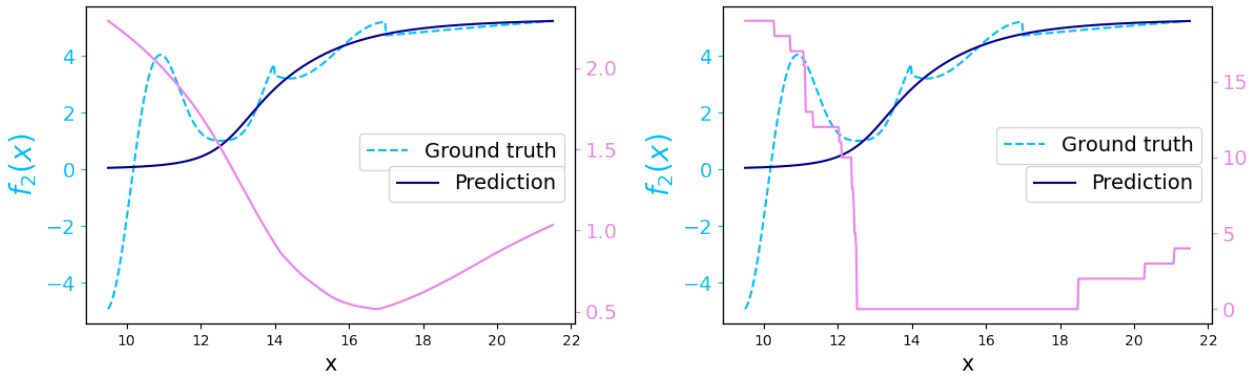

Figure 31. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 3 in f_2.

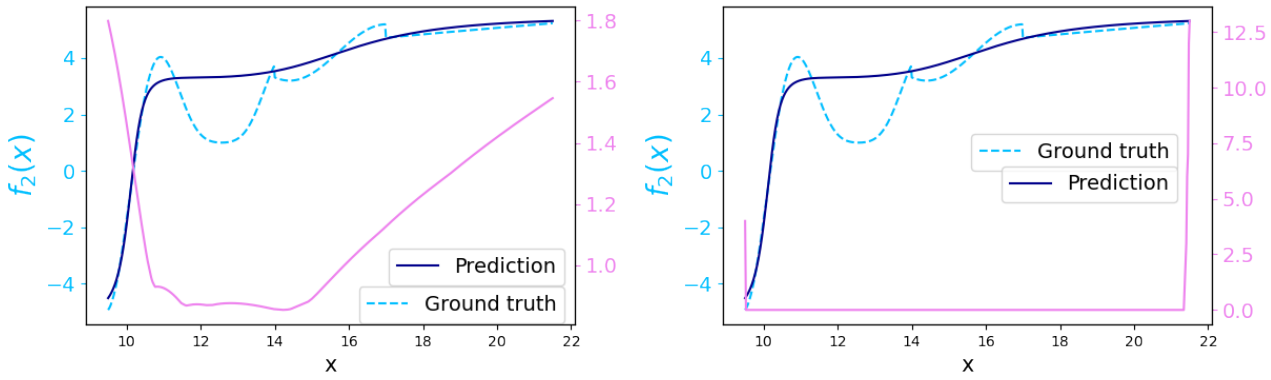

Figure 32. From left to right, Average Distance Δ_j (right vertical axis) from node values' mean per sample x_j, Number of Nodes out-of-range $\mathcal{O}_j$ (right vertical axis) per sample x_j, for test 6 in f_2.

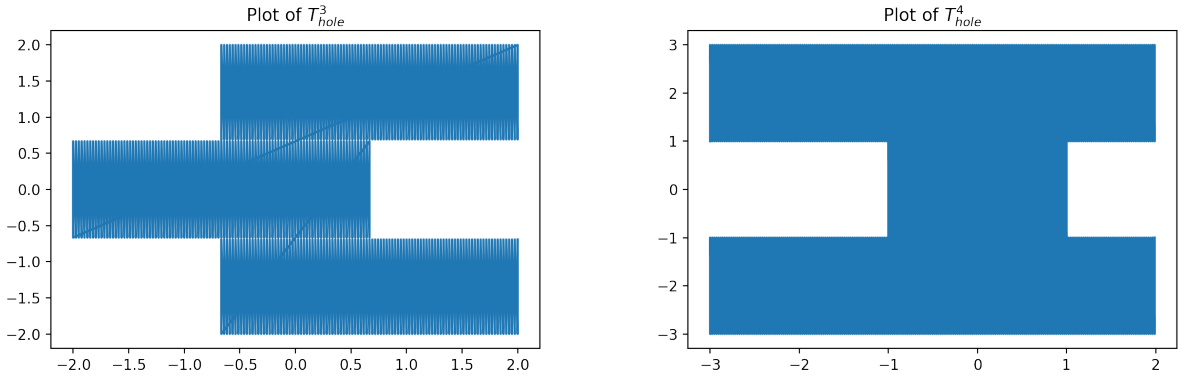

Figure 33. From left to right, T^3_{hole} and T^4_{hole}.

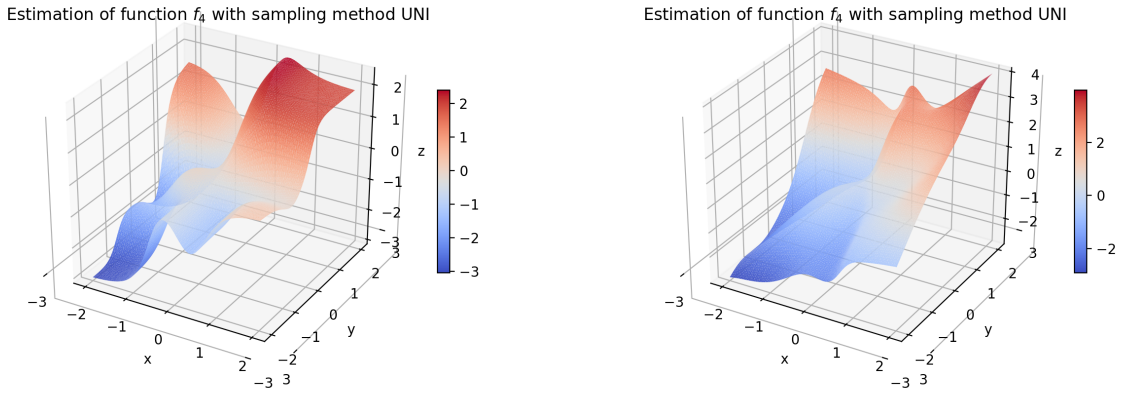

Figure 34. prediction from network (10, 3) of f_3 for test index 2, prediction from network (10, 3) of f_3 for test index 5.

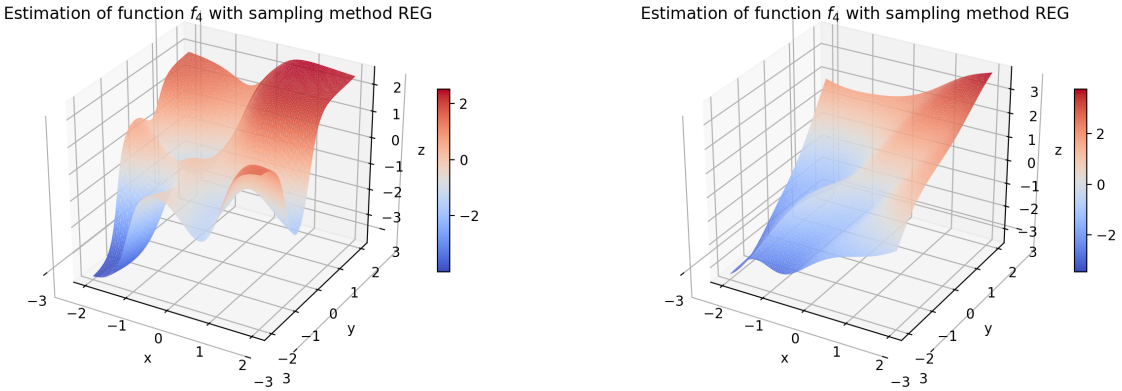

Figure 35. prediction from network (10, 3) of f_3 for test index 1, prediction from network (10, 3) of f_3 for test index 4.

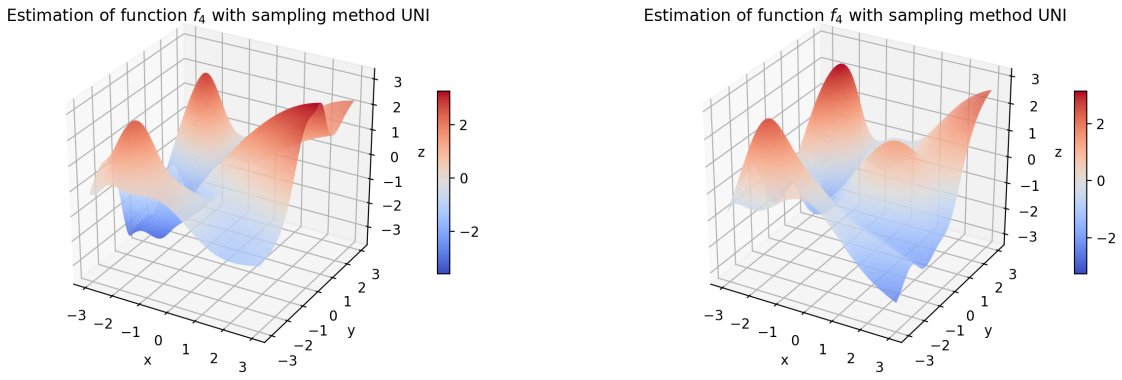

Figure 36. prediction from network (10, 3) of f_4 for test index 2, prediction from network (10, 3) of f_4 for test index 5.

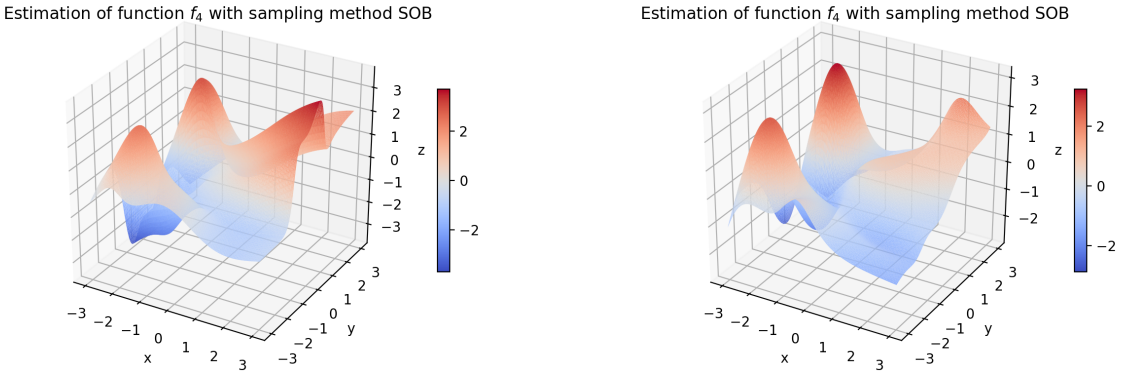

Figure 37. prediction from network (10, 3) of f_4 for test index 3, prediction from network (10, 3) of f_4 for test index 6.