Characterisation of neural networks and comparison with finite element methods

Cyril Vallez
cyril.vallez@epfl.ch

Supervised by Prof. Marco Picasso

December 31, 2021

Abstract

In this report, we study how neural networks compute approximation of real function in one and several dimensions. We put an emphasis on how the approximation is constructed, meaning that we try to visualize and understand what happens in the neurons in the hidden layers of the networks. We compare this to finite element methods, for which analytic bounds and convergence order of the approximations are well known. Moreover, we try to understand what parameters are most important when computing such approximations with neural networks, and the impact they have on the global and local quality of the approximation.

Contents

1 Introduction

Over the last years, machine learning and deep neural networks proved that they were capable of mind blowing achievements that no other numerical methods could hope to match in many fields, including image classification (sometimes exceeding human performance [3]), text mining, or recommender systems. However, if neural networks are so good in practice, it seems that they still lack a clear theoretical interpretation, and a clear description on the how and why they work this good. Indeed, they rely on a large set of hyper-parameters (from the structure of the network, to the learning step used, passing by the activation functions used,...) which are not always well understood, and which rely on trial and error. There are no general cases on the impact of one parameter or another on the performances. For this reason, the goal of this project is to provide insights on what parameters impact the performances when using simple neural networks for a relatively easy task : the approximation of a real function in one and several dimensions. We also compare this with the structure of finite element methods for creating an approximation of a real function. However, finite element methods are bound to work only with a few dimensions, as we have to create a mesh of the domain, and the size increase exponentially with the number of dimensions.

2 Description of the problem

2.1 Neural networks

In our discussion of neural networks, we refer as L for the depth of the network, or number of hidden layers, and as K_l $l = 1, ..., L$ for its width, or number of neurons in layer l. The input layer is the 0-th layer with dimension K_0 and the output is the $(L+1)$-th layer with dimension K_{L+1}. We denote as $w_{i,j}^{(l)}$ the weight connecting neuron j in layer $l-1$ to neuron i in layer l.
The value of neuron i in layer l satisfy the following recursion :

$$x_i^{(l)} = \sigma\left(\sum_{j=1}^{K_{l-1}} w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}\right) \tag{1}$$

where $\sigma(x)$ is the activation function and $b_i^{(l)}$ is a bias term.

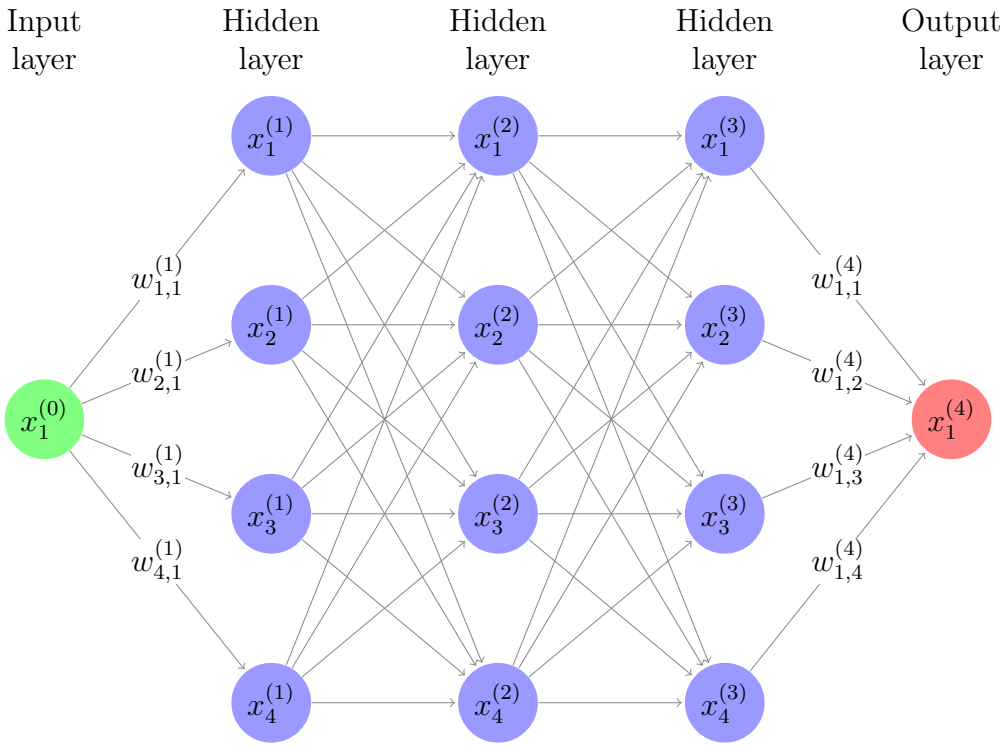


Figure 1: Example of a fully connected layer with $L = 3$, $K_1 = K_2 = K_3 = 4$ and one-dimensional input and output ($K_0 = K_4 = 1$). The biases are not represented for simplicity.

We introduce the output vectors $X^{(l)} \in \mathbb{R}^{K_l}$, the bias vectors $B^{(l)} \in \mathbb{R}^{K_l}$ and the weight matrices $W^{(l)} \in \mathbb{R}^{K_l \times K_{l-1}}$ with the following relations :

$$X^{(l)} = \begin{pmatrix} x_1^{(l)} \\ \vdots \\ x_{K_l}^{(l)} \end{pmatrix} \qquad W^{(l)} = \begin{pmatrix} w_{1,1}^{(l)} & \cdots & w_{1,K_{l-1}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{K_l,1}^{(l)} & \cdots & w_{K_l,K_{l-1}}^{(l)} \end{pmatrix} \qquad B^{(l)} = \begin{pmatrix} b_1^{(l)} \\ \vdots \\ b_{K_l}^{(l)} \end{pmatrix} \tag{2}$$

With this convention, we are immediately able to describe the value of each K_l neurons of layer l with the relation :

$$X^{(l)} = \sigma\left(W^{(l)} X^{(l-1)} + B^{(l)}\right) \tag{3}$$

where the activation function σ operates component-wise, i.e on all components of the vector.

We will almost always use fully connected neural networks whose hidden layer widths are the same, i.e $K_1 = K_2 = ... = K_L \coloneqq K$. Moreover, we will also almost always use ReLU activation functions (Equation 4) in all the layers except the output layer where we do not use any activation (i.e linear activation, $\sigma(x) = x$. The ReLU activation function is defined as :

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Note that for a neural network with L hidden layers, same number of neurons K in each layer, input dimension X_0 and output dimension 1, the total number of trainable parameters N_{params} is given by :

$$\begin{aligned} N_{\text{params}} &= (X_0 + 1)K + (L-1)(K^2 + K) + K + 1 \\ &= (L-1)K^2 + (X_0 + L + 1)K + 1 \end{aligned} \tag{5}$$

2.2 Finite element method

We want to approximate a function $u : [-1, 1] \to \mathbb{R}$ with homogeneous boundary conditions $u(-1) = u(1) = 0$. The usual finite element formulation for this is to divide the interval $[-1, 1]$ into N points and $N - 1$ sub-intervals of length h such that :

$$h = \frac{2}{N-1} \qquad x_i = -1 + ih \quad i = 0, 1, ...N - 1 \tag{6}$$

We then define the approximation of $u(x)$ or interpolant of $u(x)$ as $\Pi_h u(x)$:

$$\Pi_h u(x) = \sum_{i=1}^{N-2} u(x_i)\phi_i(x) \tag{7}$$

where we used the fact that $u(x)$ has homogeneous boundary conditions.

The basis functions or "hat" functions $\phi_i(x)$ $i = 1, ...N - 2$ are defined as :

$$\phi_i(x) = \begin{cases} 0 & \text{if } x \notin [x_{i-1}, x_{i+1}] \\ \dfrac{1}{h}(x - x_{i-1}) & \text{if } x_{i-1} \leq x \leq x_i \\ \dfrac{1}{h}(x_{i+1} - x) & \text{if } x_i < x \leq x_{i+1} \end{cases} \tag{8}$$

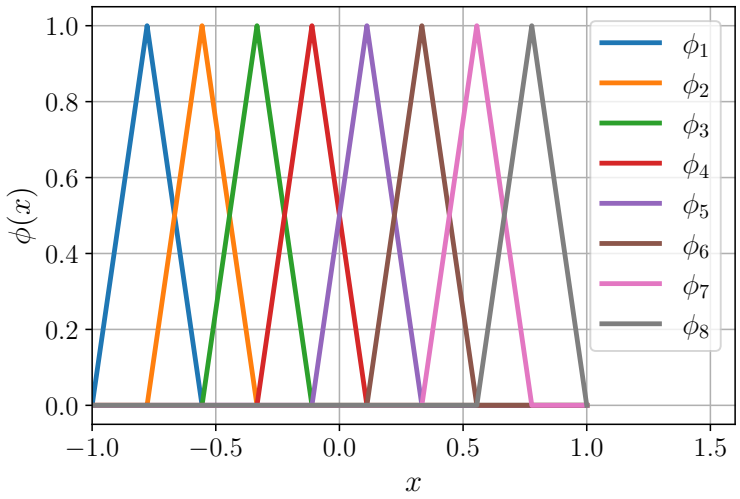


Figure 2: Basis functions for $N = 10$ points.

Note that this formulation can be easily generalized to functions with support different from $[-1, 1]$ and non-homogeneous boundary conditions.
Then, if the function $u(x)$ is unknown but is the solution of some (differential) equation, one is able to use this formulation to approximate the coefficients $u(x_i)$ and reconstruct the function.

2.3 Link between neural networks and finite element

From this description of neural networks and finite element method, one may note that the interpolant $\Pi_h u(x)$ of $u(x)$ can be obtained from a neural network with one hidden layer of $K_1 = N$ neurons with ReLU activation in the first layer, and no activation in the output. Indeed, we first note that the basis functions (Equation 8) can be rewritten as :

$$\phi_i(x) = \frac{1}{h}\Big[\mathrm{ReLU}(x - x_{i-1}) - 2 \cdot \mathrm{ReLU}(x - x_i) + \mathrm{ReLU}(x - x_{i+1})\Big] \tag{9}$$

where $\mathrm{ReLU}(x)$ is defined in Equation 4.

From this, we can reconstruct Equation 7 with the following biases and weights (noting that both the input and output are one-dimensional) :

$$W^{(1)} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^N \qquad B^{(1)} = - \begin{pmatrix} x_0 \\ \vdots \\ x_N \end{pmatrix} \in \mathbb{R}^N \tag{10}$$

$$W^{(2)} = \frac{1}{h} \begin{pmatrix} u(x_1) \\ u(x_2) - 2u(x_1) \\ u(x_1) - 2u(x_2) + u(x_3) \\ \vdots \\ u(x_{N-3}) - 2u(x_{N-2}) + u(x_{N-1}) \\ u(x_{N-2}) - 2u(x_{N-1}) \\ u(x_{N-1}) \end{pmatrix}^T \in \mathbb{R}^{1 \times N} \qquad B^{(2)} = 0 \in \mathbb{R} \tag{11}$$

where the $x_i \;\; i = 0, ..., N-1$ are defined in Equation 6.

Thus the finite element formulation with N points ($N-1$ intervals) is included in the family of functions that a neural network with $L = 1$ hidden layer and $K_1 = N$ neurons is able to reconstruct.

Therefore, we study in the following if a neural network with such characteristics does reconstruct a function in a "finite element way" or not.

3 The 1D case

In this section, we take the following sharp transition function as objective $u : [-1, 1] \to \mathbb{R}$:

$$u(x) = \frac{1}{2}\Big(\tanh(k(x - x_1)) - \tanh(k(x - x_2))\Big) \tag{12}$$

The parameter k controls the steepness of the transition, and the parameters x_1 and x_2 control the location of the two transitions. In the following we consider $k = 50$, $x_1 = -1/3$, and $x_2 = 1/3$, so that the function is symmetric.
The function has the following graph :

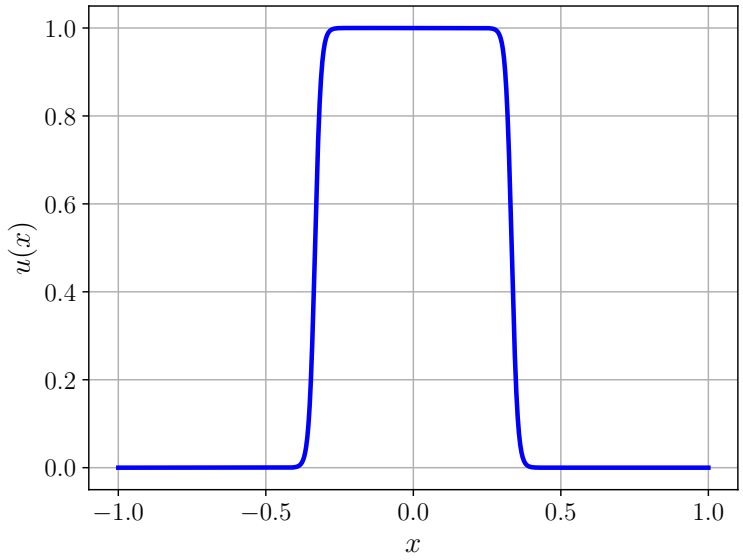


Figure 3: Graph of $u(x)$ as defined in Equation 12

3.1 Dataset

The training process of a neural network requires a large amount of training data consisting of input/output pairs. For this reason, we take $M = 10^5$ points uniformly distributed at random between -1 and 1. For all of these points x, we compute $u(x)$ as in Equation 12. The pairs $(x, u(x))$ represent the pairs (input, label) or (feature, label) which we will use for training. We separate 10% of the M pairs as a testing set. During training, we also monitor the metrics with 10% of the M pairs as a validation set. This means that we train the network on a total of $M_{\text{eff}} = 80000$ pairs (input, label), assess the metrics during training with 10000 pairs, and still have a testing set of 10000 pairs to compare the accuracy between different models/architectures of networks.

3.2 Loss and optimization process

In the following we will consider different architectures of neural networks. However, as represented in Figure 1, the network will always have 1-dimensional input and output, representing x and the approximation of $u(x)$ respectively. Only the hidden layers will change.
We will always train the network using the Mean-Squared-Error (MSE) loss function :

$$\text{MSE} = \frac{1}{M_{\text{eff}}} \sum_{i=1}^{M_{\text{eff}}} (u^i_{pred} - u(x_i))^2 \tag{13}$$

where u^i_{pred} is the output of the network, and $u(x_i)$ is the label corresponding to the input of the network.

We always use the Adam algorithm [2] for the optimization problem, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. If not explicitly specified in the experiment, we also use $\eta = 10^{-4}$ for the learning step.
In the same spirit, if not explicitly specified, we use a batch size of 32 for the gradient updates. We almost always perform 250 epochs during training, meaning that each of the M_{eff} pair (input, label) is seen exactly 250 times by the network. We then keep the model which minimized the validation loss during training.

3.3 Metrics

To assess the quality of the approximation we make, we use some metrics that we evaluate on the testing/validation set. The first metric is the MSE, or loss, which is defined in Equation 13. Another metric that we will sometimes use is the Mean-Absolute-Error (MAE) :

$$\text{MAE} = \frac{1}{M_{\text{eff}}} \sum_{i=1}^{M_{\text{eff}}} |u^i_{pred} - u(x_i)| \tag{14}$$

However, the MSE and MAE reflect the same information, only the order of magnitude (square or absolute value) will change. Indeed, both show how each prediction is far from the ground-truth value in average. For this reason, we define another metric, the Maximum-Absolute-Error (MaAE) :

$$\text{MaAE} = \max_{i=1,\dots,M_{\text{eff}}} |u^i_{pred} - u(x_i)| \tag{15}$$

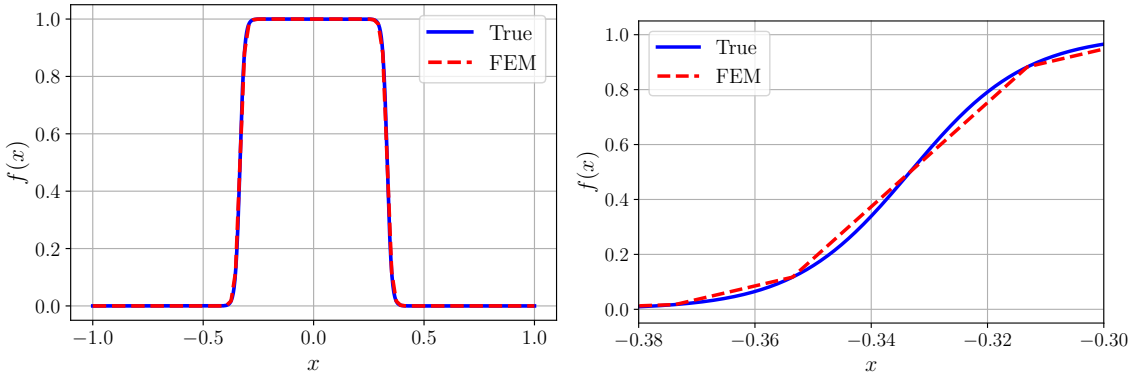
The MaAE gives a more localized feedback, telling us how the approximation behaves at worst. The worst points are likely to be around the transitions between 0 and 1 at $x = -1/3$ and $x = 1/3$, which are the regions of most interest.

3.4 1 layer network

In this section we focus on network with $L = 1$ hidden layer, and $K_1 = 100$ to try to replicate the solution given in Equations 10 and 11 (or alternatively the solution directly given by Equation 7 with $N = 100$). First we compute this interpolant and evaluate it on the testing set. The (sorted) testing set and approximation are represented in Figure 4. The metrics are summarized in Table 1.

MSE	MAE	MaAE
4.4188e-05	1.7255e-03	4.1793e-02

Table 1: Metrics on the testing set for the interpolant of $u(x)$ with $N = 100$ points



(a) (b)

Figure 4: (a) Comparison between $u(x)$ as in Equation 12 (blue line) and its interpolant (Equation 7) with $N = 100$ points (red dotted-line) and (b) Zoom over the interval of the first sharp transition

Now, we train a neural network with $L = 1$, $K_1 = 100$ and ReLU activation function in the first layer. No activation is used at the output. The optimization parameters are described in Section 3.2. The only difference is that we use a learning step $\eta = 3 \cdot 10^{-4}$. In order to take into account the randomness of the initialization of

layer weights and optimization process, we train $P = 5$ similar networks. We report the metrics on the testing set for the average, minimum, and maximum of all P networks :

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
1.1337e-06	1.5296e-06	2.7364e-06	4.6878e-04	5.2059e-04	5.7629e-04	9.8791e-03	1.1573e-02	1.7971e-02

Table 2: Metrics for the 1 layer network described above. The mean, min and max are applied over $P = 5$ networks trained in the same conditions

For the best of these $P = 5$ models (the model which achieves the lowest MSE), we show the training and validation MSE during training in Figure 5, and the resulting approximation on the test set in Figure 6 :

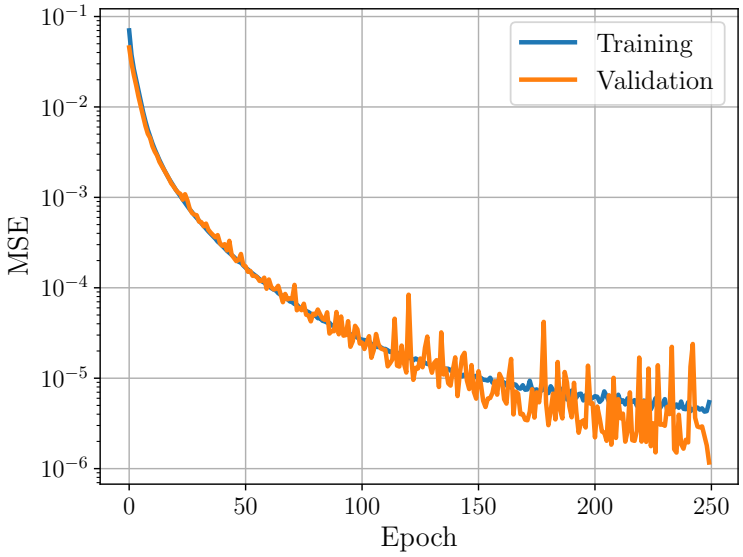
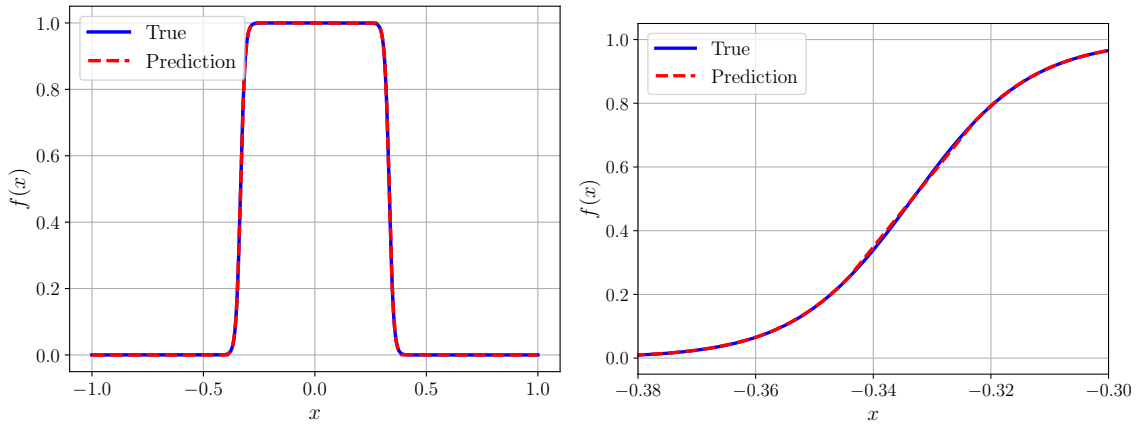


Figure 5: MSE during training for the best of the $P = 5$ models



(a) (b)

Figure 6: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 5$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition

We now inspect the ReLU functions created on all of the $K_1 = 100$ neurons :

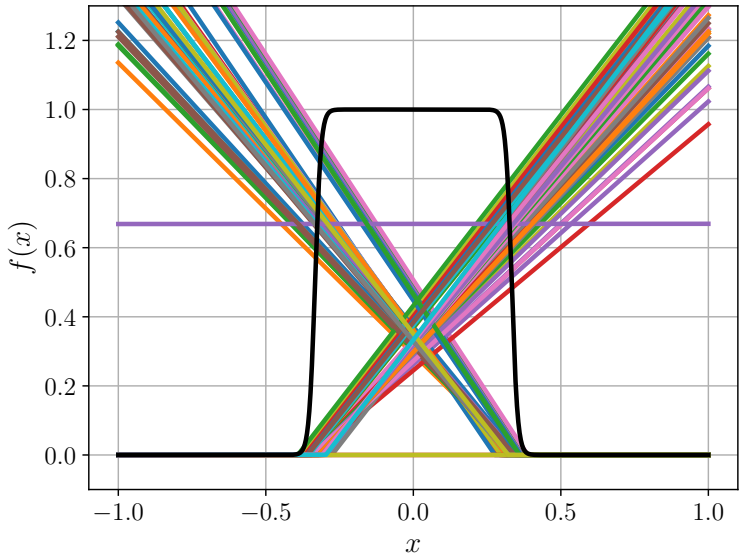


Figure 7: Visualization of the ReLU functions created by the neurons of the model. The curve in black is the ground-truth $u(x)$.

On Figure 7, one immediately sees that the approximation built by the network is

very different from the interpolant given in Equation 7. Thus, the idea is now to try to initialize the network with the exact weights given in Equations 10 and 11 (or weights very close to this) which corresponds to the interpolant, and see if the solution stays in this local minimum or moves away from it.

Figure 8 shows the ReLU functions created by the network on interval $[-1, 1]$ after only 10 epochs of training after initializing with the exact weights given in Equations 10 and 11. This is obviously very far from the structure of the ReLU functions represented in Figure 9, which shows that even after being initialized with the exact weights from the interpolant solution, the solution diverges from this given minimum in just a few epochs.

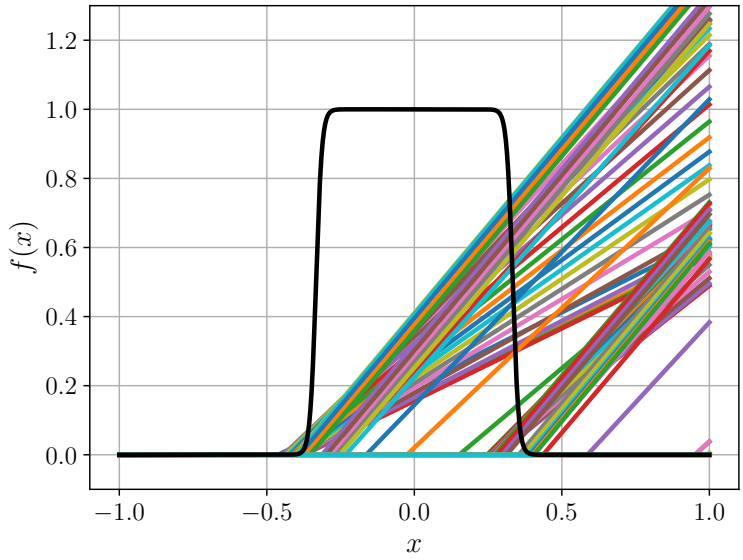


Figure 8: Visualization of the ReLU functions after training for 10 epochs using as initial weights Equations 10 and 11

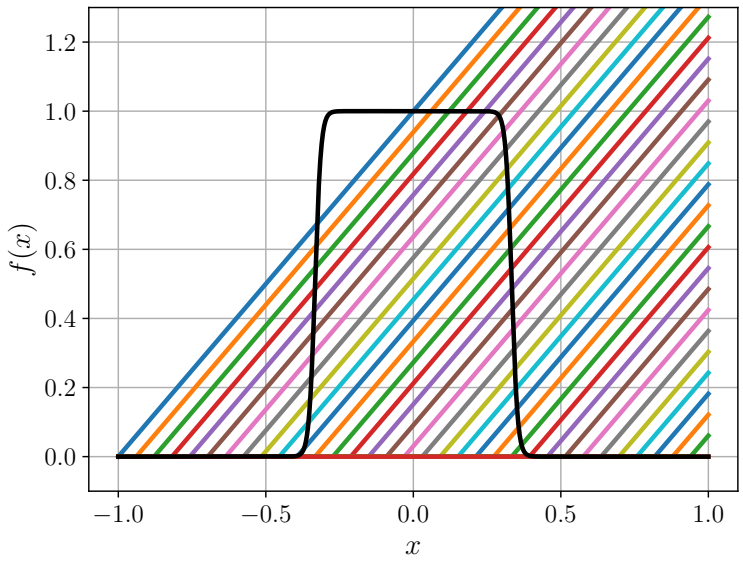
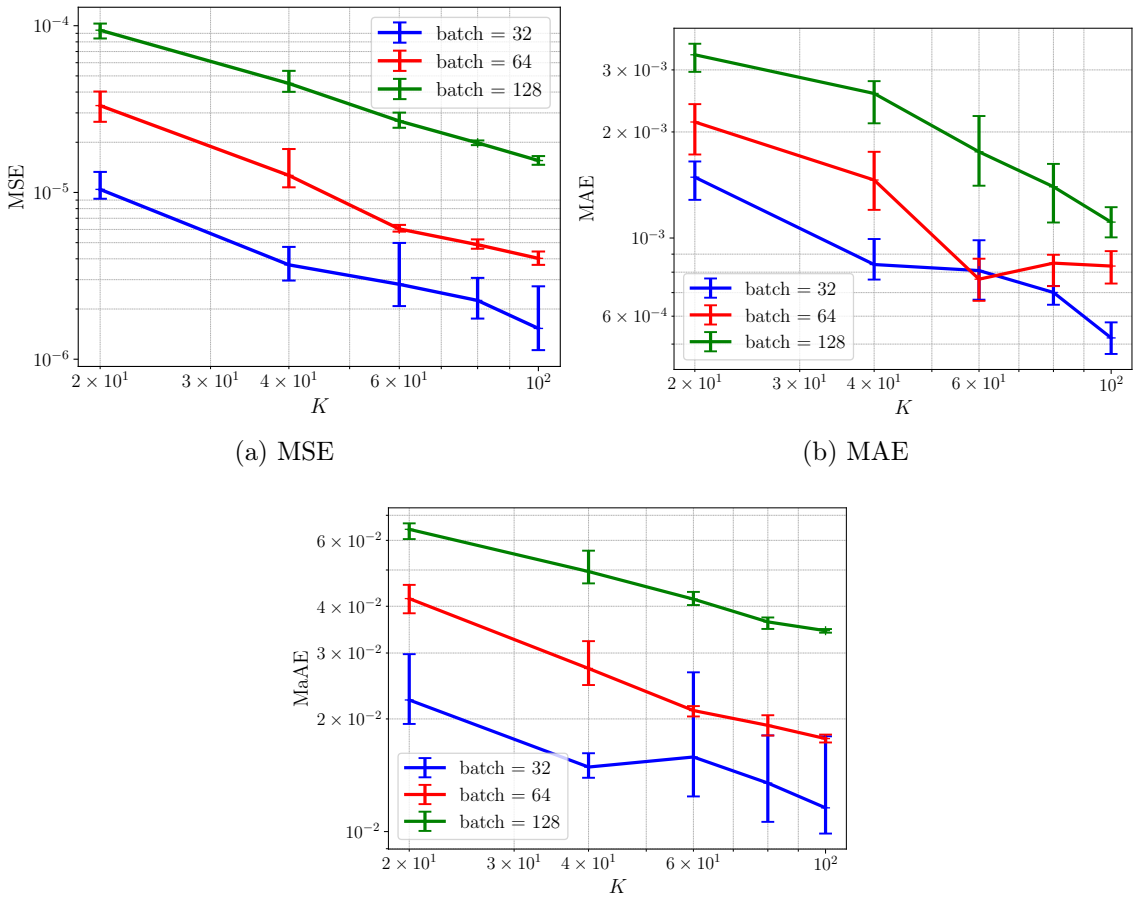


Figure 9: Visualization of the ReLU functions when initializing as in Equation 10 (we represented only 1 out of 3 of the $K_1 = 100$ ReLU functions for clarity)

Now, we wish to assess the impact of the number K_1 of neurons in the hidden layer in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each K_1, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range [minimum,maximum] of the $P = 5$ values for the given metric. The learning step $\eta = 3 \cdot 10^{-4}$ is fixed for all K_1 and batch sizes.



(a) MSE

(b) MAE

(c) MaAE

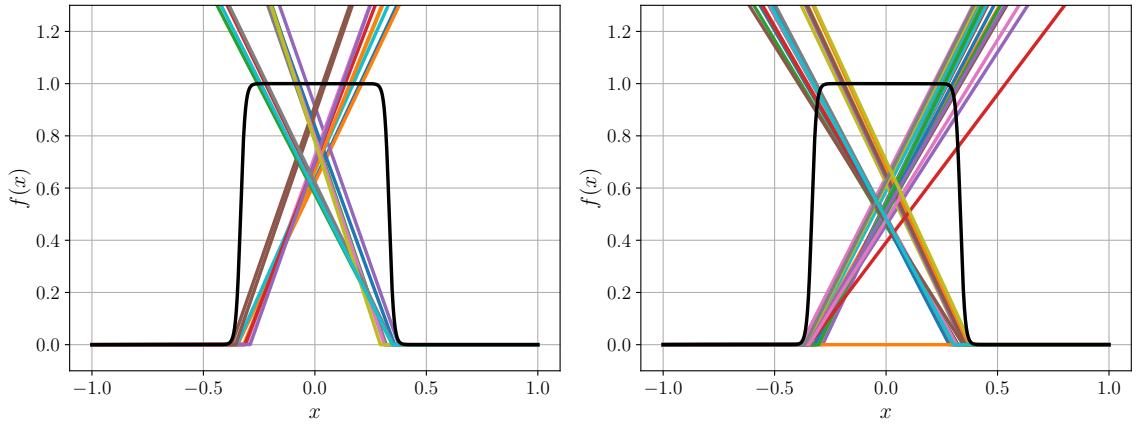
Figure 10: Metrics in function of $K_1 := K$ for a network with $L = 1$ hidden layer

One is able to see on Figure 10 (a) that the MSE is almost a straight line as a function of K in the log-log scale. This means that we can easily perform a linear fit and obtain an approximation of the order of convergence as a function of K. By doing so, we obtain the following :

	Slope	Intercept
batch=32	-1.13 ± 0.10	-3.55 ± 0.18
batch=64	-1.35 ± 0.10	-2.74 ± 0.17
batch=128	-1.13 ± 0.02	-2.55 ± 0.04

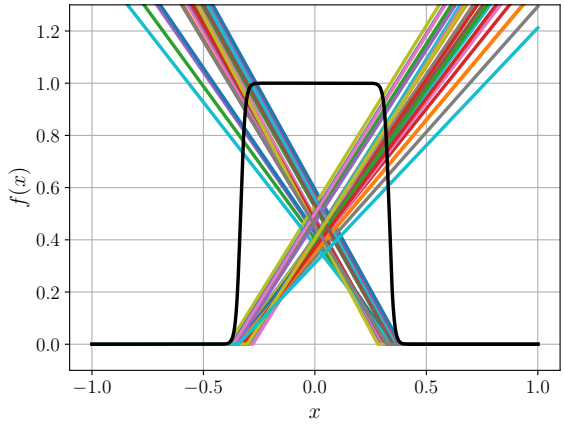
Table 3: Results of the linear fit made on the log-MSE and log-K

From Table 3 and Figure 10, we are able to approximate that for our settings (dataset and optimization process), the MSE decreases approximately at order 1 as K_1 increases for a network with $L = 1$ (at least in the interval $20 \leq K_1 \leq 100$). Moreover, this seems independant of the batch size for small batches, since the three curves are parallels. However, the smaller the batch size, the smaller the intercept and thus the MSE will be always smaller.
We do not perform the same analysis for the MAE and MaAE because the curves are less smooth, and we are particularly interested in the MSE since this is the loss function we use. However, the behavior is similar.

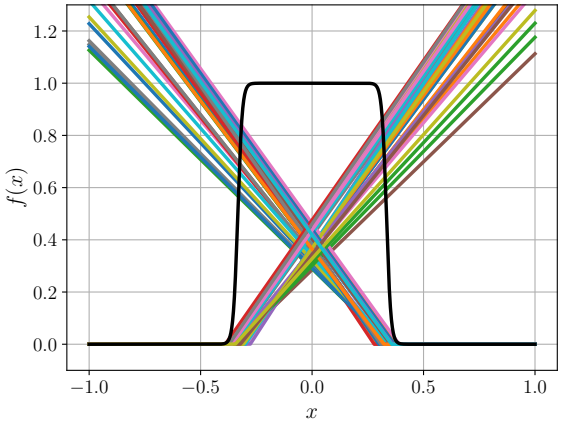


(a) $K_1 = 20$

(b) $K_1 = 40$



(c) $K_1 = 60$



(d) $K_1 = 80$

Figure 11: ReLU functions created in the first layer for the best model of the different K_1 for batch size 32

Figure 11 shows the ReLU functions created in the first layer of the best of the $P = 5$ models with batch size 32 for each K_1 we tested. The structure is always similar to that of Figure 7.

3.5 2 layers network

We now focus on neural networks with $L = 2$ hidden layers and $K_1 = K_2 := K = 100$ neurons in each layer. As always, ReLU activation functions are used in both hidden layers, and we do not use any activation in the output layer. The optimization parameters are described in Section 3.2 with the exception of the learning step set to $\eta = 10^{-5}$. The goal is to try to find a better approximation than with only 1 hidden layer. In order to take into account the randomness of the initialization of layer weights and optimization process, we train $P = 5$ similar networks. We report the metrics on the testing set for the average, minimum, and maximum of all P networks :

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
9.8597e-09	1.8178e-08	3.6362e-08	4.5042e-05	6.6100e-05	9.1782e-05	1.3444e-03	1.7136e-03	2.7757e-03

Table 4: Metrics for the 2 layers network described above. The mean, min and max are applied over $P = 5$ networks trained in the same conditions

We show in Figure 12 the training and validation error, and in Figure 13 the approximation on the test set for the best of the $P = 5$ models.

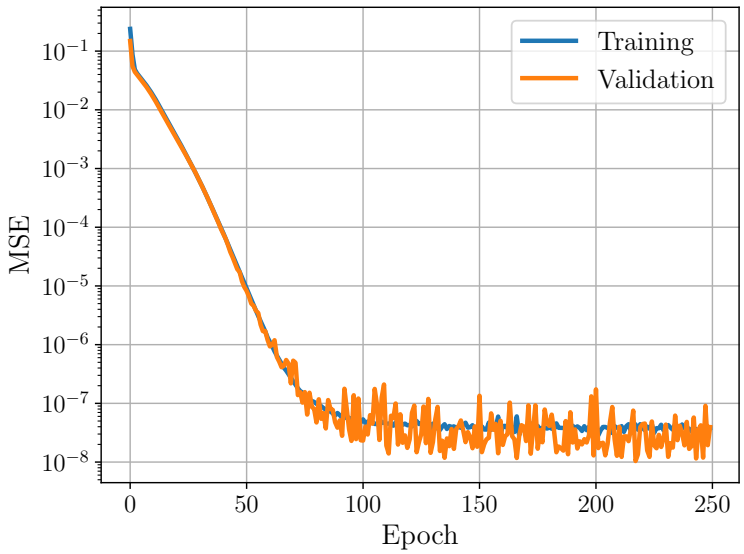
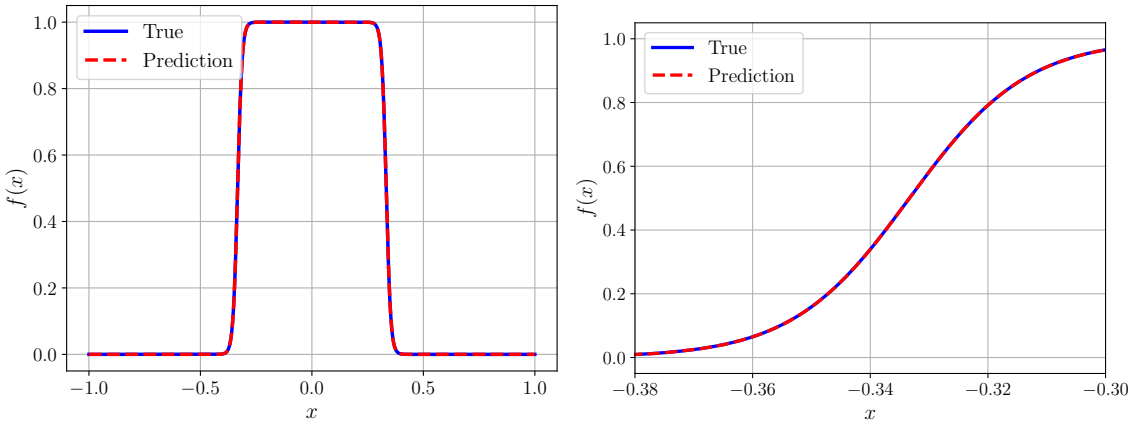


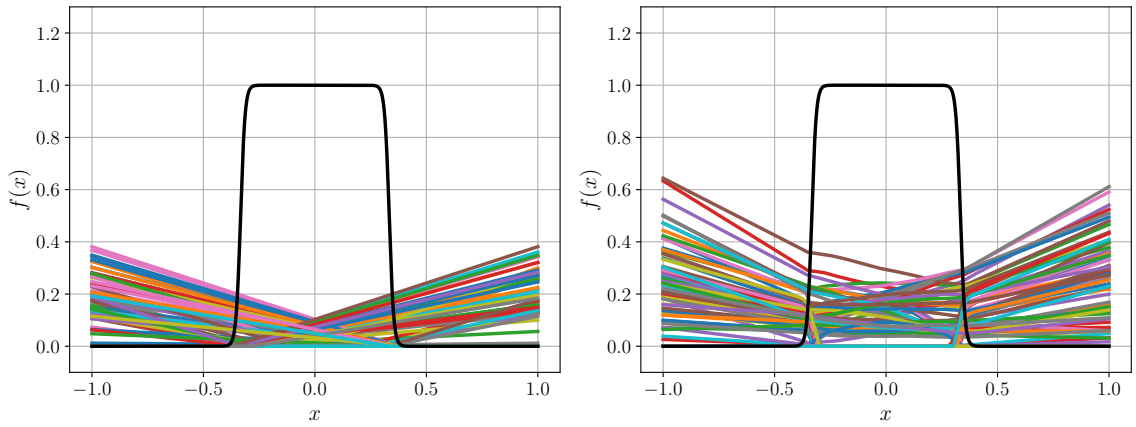
Figure 12: MSE during training for the best of the $P = 5$ models



(a) (b)

Figure 13: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 5$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition

Now we inspect the functions created in the neurons of the first and second layer. For the best of the $P = 5$ models, here are the results we got :

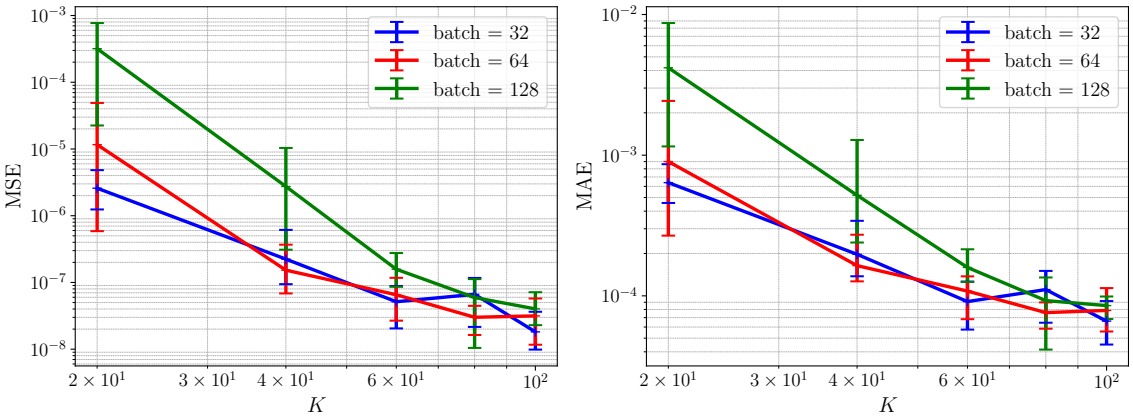


(a) First layer

(b) Second layer

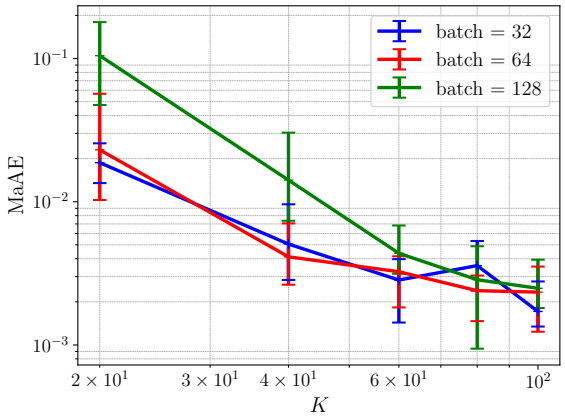
Figure 14: Functions created by the neurons of the model in the first and second layer

Now, as before we wish to assess the impact of the number $K_1 = K_2 \coloneqq K$ of neurons in the hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each K, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range [minimum,maximum] of the $P = 5$ values for the given metric. The learning step $\eta = 10^{-5}$ is fixed for all K and batch sizes.



(a) MSE

(b) MAE



(c) MaAE

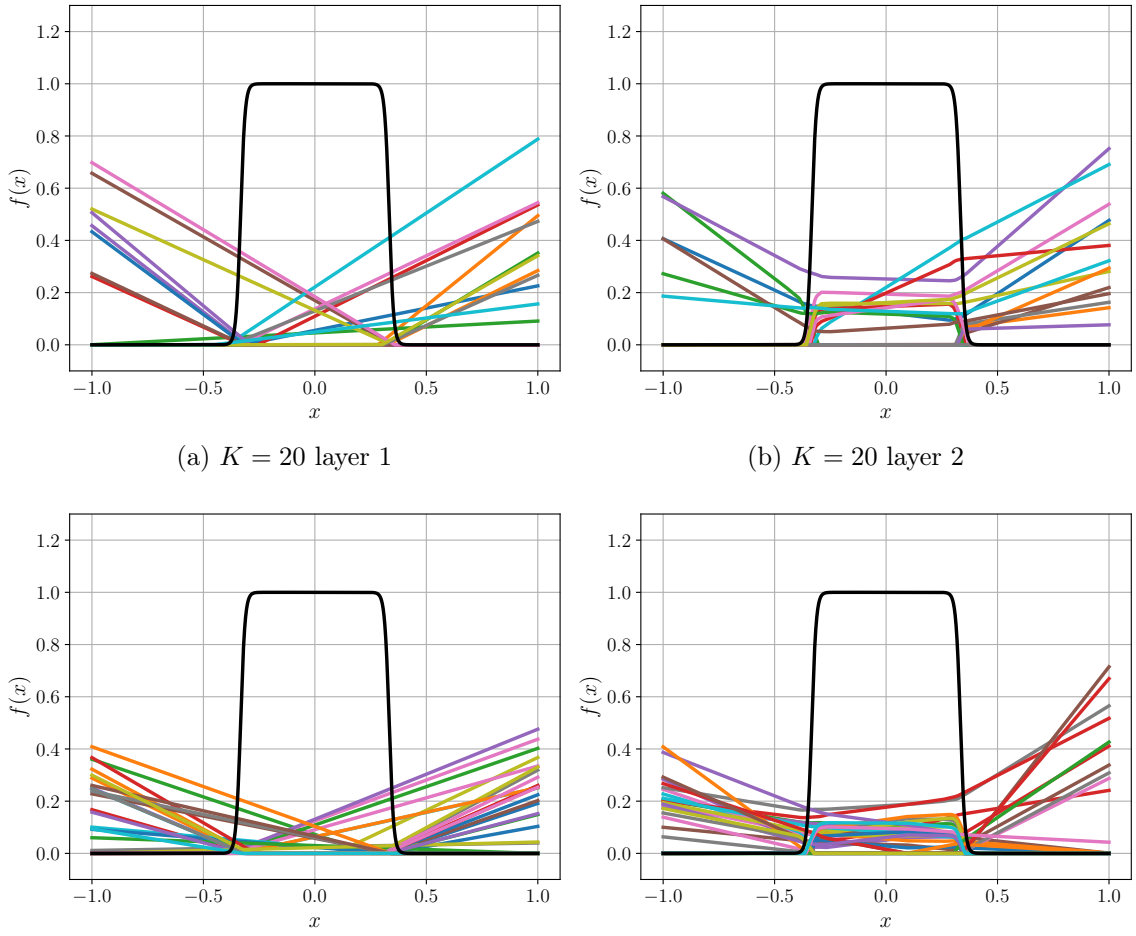
Figure 15: Metrics in function of $K_1 = K_2 \coloneqq K$ for a network with $L = 2$ hidden layers

As one is able to see on Figure 15 (a), the MSE as a function of K are not parallels straight lines as it was for only $L = 1$ hidden layers. However, for each batch size, the curve is relatively straight, and we can still perform a linear fit as before; we just expect to see a bigger difference in the slopes.

	Slope	Intercept
batch=32	$-2.91 + \pm 0.38$	-1.89 ± 0.66
batch=64	-3.71 ± 0.70	-0.42 ± 1.21
batch=128	-5.79 ± 0.57	3.84 ± 0.98

Table 5: Results of the linear fit made on the log-MSE and log-K

As we expected, the slopes are quite different for each batch sizes, and the confidence interval is much bigger than in the case with only $L = 1$ hidden layer. However, it is clear from the slopes that the convergence is way faster with $L = 2$ than with $L = 1$, as we increase K.



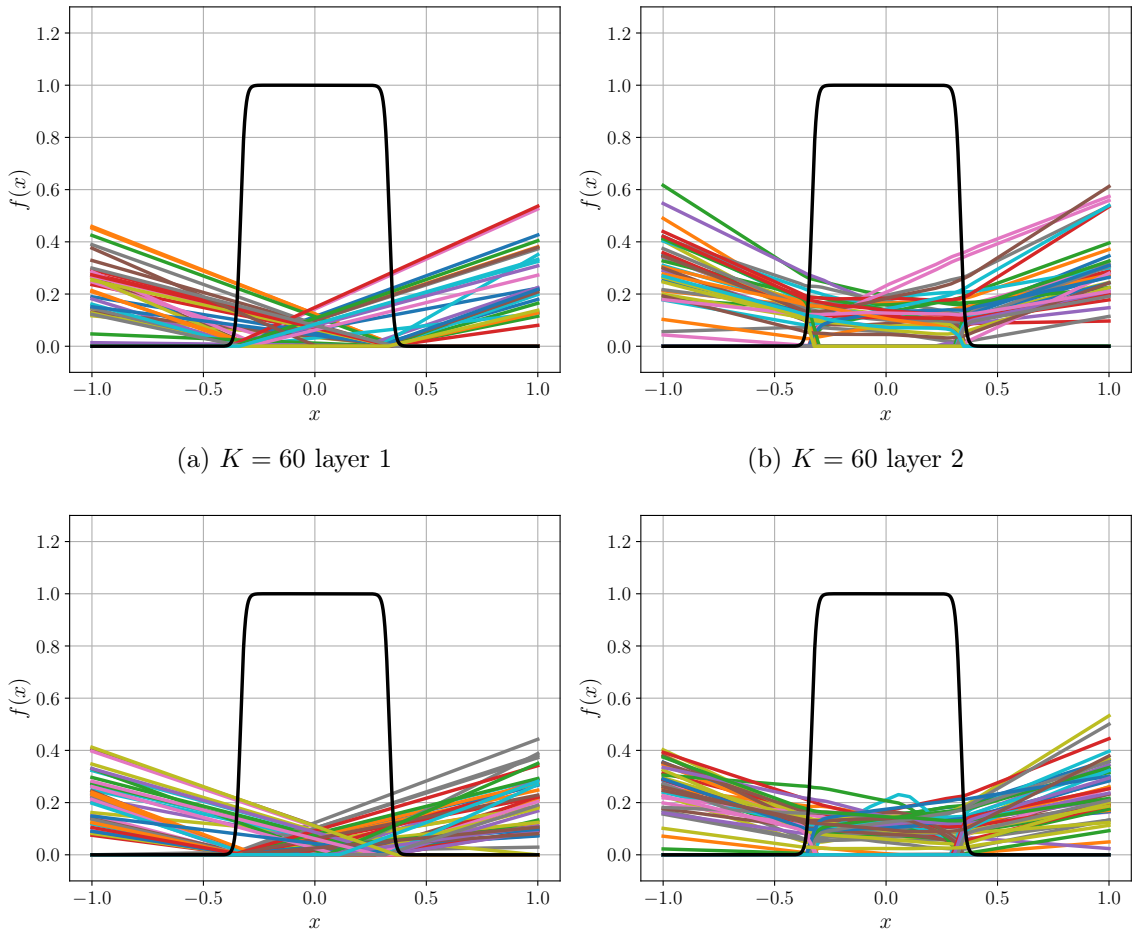
(a) $K = 20$ layer 1

(b) $K = 20$ layer 2

(c) $K = 40$ layer 1

(d) $K = 40$ layer 2

Figure 16: ReLU functions created in the first and second layer for the best model of the different K_1 for batch size 32



(a) $K = 60$ layer 1

(b) $K = 60$ layer 2

(c) $K = 80$ layer 1

(d) $K = 80$ layer 2

Figure 17: ReLU functions created in the first and second layer for the best model of the different K_1 for batch size 32

3.6 Convolutional network

The basis function (Equation 9) are linear combinations of 3 ReLU functions. Thus if these ReLU functions are created by a first dense layer, a convolutional layer with a kernel of 3 and stride 1 could recreate the basis functions. This observation is the key point of this part.

The architecture of the networks considered in this section is the following : First, a fully connected layer with K neurons, with ReLU activation. Then a 1D convo-

lutional layer with 1 filter, a kernel of size 3, stride 1, padded with 0 such that the output shape is the same as the input shape (K neurons), and without any activation function. And finally, the output of dimension 1, still without any activation function. The idea is to try to "force" the model to recreate the basis functions depicted in Figure 2 before combining them for the output.

As before, to account for the randomness during the training process, we train $P = 5$ identical networks, and report the metrics for the average, minimum and maximum of all P models. Note that we use a learning rate of $\eta = 5 \cdot 10^{-5}$:

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
3.7612e-07	5.3576e-07	8.7218e-07	3.1774e-04	3.7064e-04	4.9573e-04	4.9139e-03	6.1872e-03	7.7638e-03

Table 6: Metrics for the convolutional network described above. The mean, min and max are applied over $P = 5$ networks trained in the same conditions

Figures 18 and 19 show the training and validation error, and the approximation on the test set for the best of the $P = 5$ models respectively.

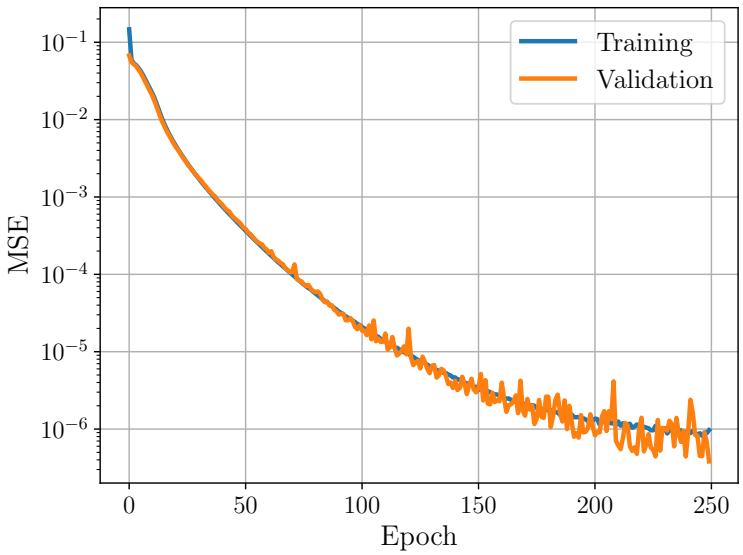
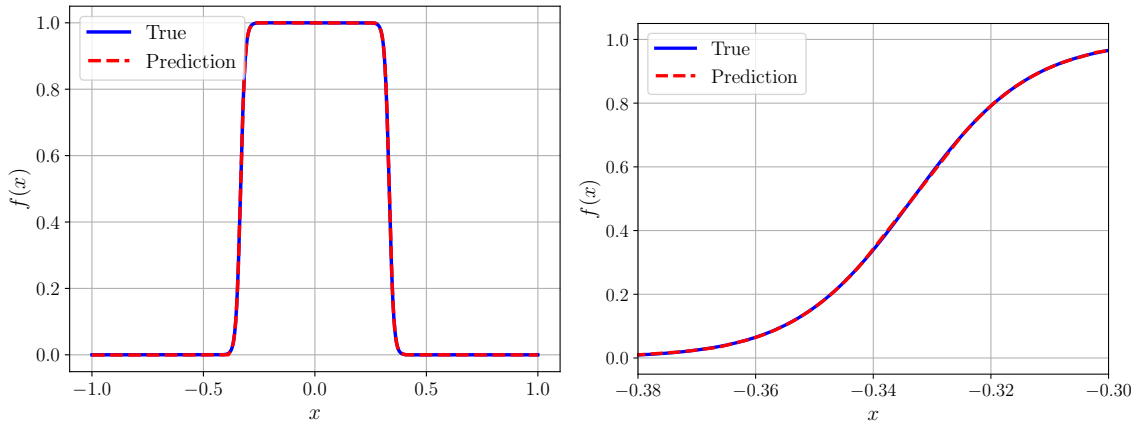
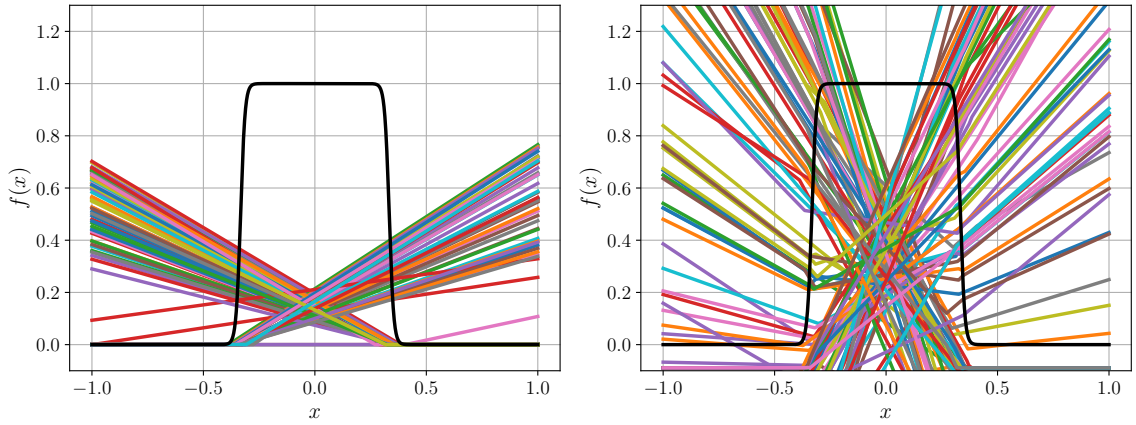


Figure 18: MSE during training for the best of the $P = 5$ models



(a) (b)

Figure 19: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 5$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition



(a) First layer (b) Second layer

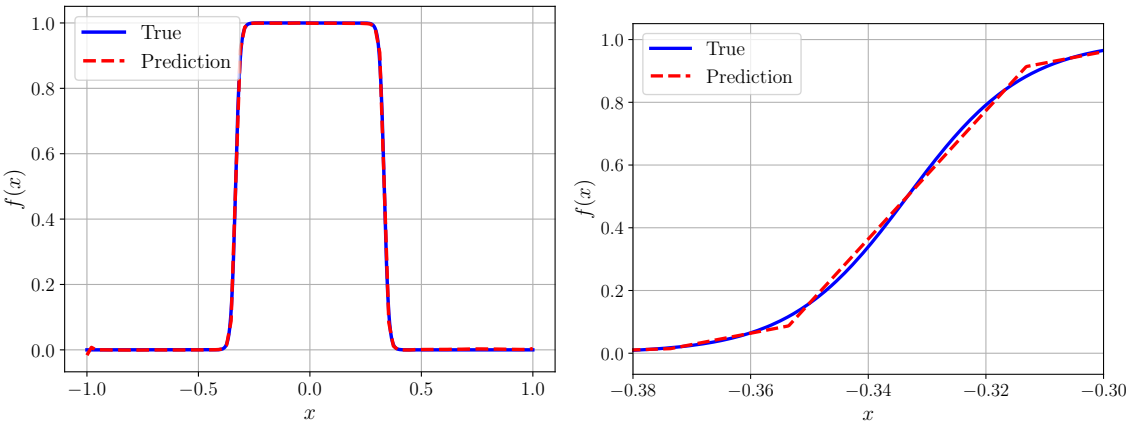
Figure 20: Functions created by the neurons of the model in the first (fully connected) layer and second (convolutional) layer for the best of the $P = 5$ models

Figure 20 shows what happens in the neurons of the best of the $P = 5$ models that we trained. This is nowhere near Figure 2.

For this reason, we now try something new : we fix the weights of the first fully connected layer as in Equation 10, and we do not train those weights. This means that the neural network will only update the weights of the convolutional layer, and the output layer. Using this, we expect the network to be able to reconstruct the basis functions (Equation 9) from the nice properties of the convolutional layer.
Once again, we perform the same experiment $P = 5$ times, and report the mean, minimum and maximum in Table 7,and the approximation and neurons functions for the best model in Figures 21 and 22 respectively :

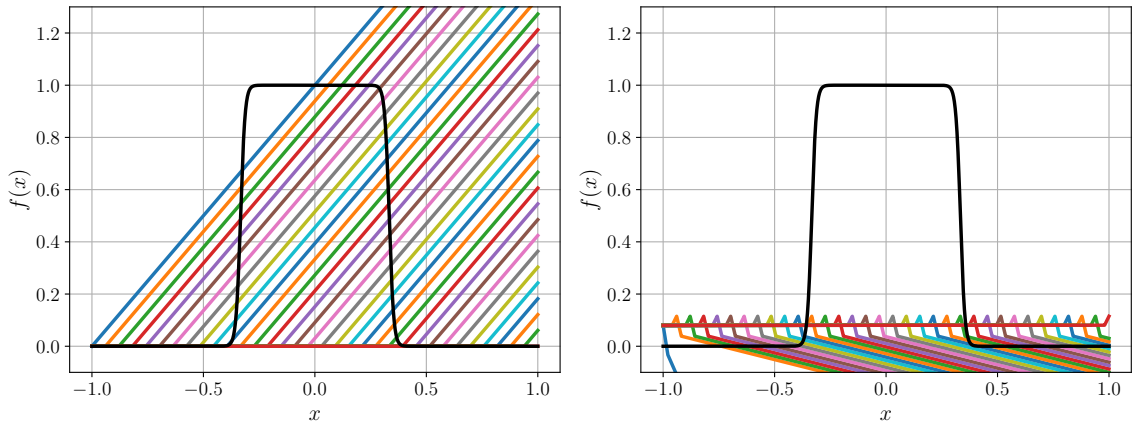
MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
2.0458e-05	2.1271e-05	2.2541e-05	1.8562e-03	1.9749e-03	2.2063e-03	2.9386e-02	3.1176e-02	3.1964e-02

Table 7: Metrics for the convolutional network described above where we do not train the first layer. The mean, min and max are applied over $P = 5$ networks trained in the same conditions



(a) (b)

Figure 21: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 5$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition



(a) First layer

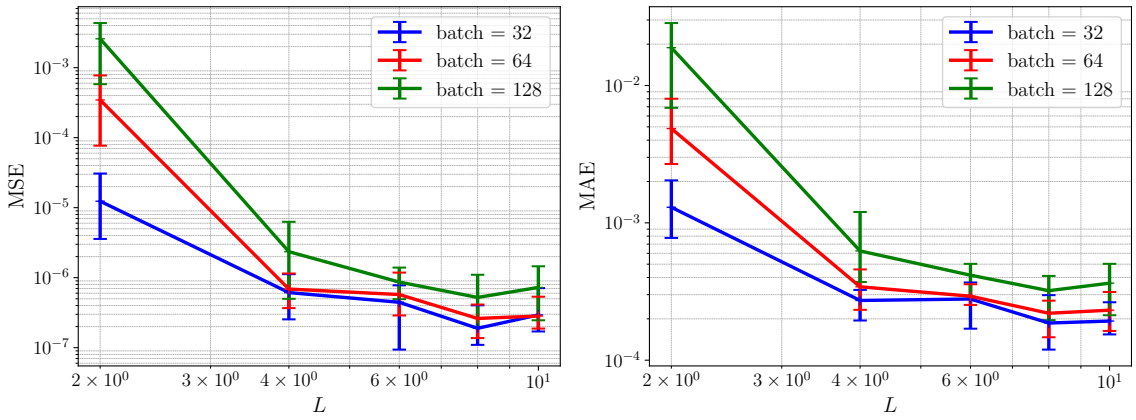
(b) Second layer

Figure 22: Functions created by the neurons of the model in the first (fully connected) layer and second (convolutional) layer for the best of the $P = 5$ models (we show only 1 out of 3 functions for readability)

Figure 21 show the approximation on the testing set.
As one can see in Figure 22 (b), the network does find similar basis functions as what we expected (Figure 2), but the amplitude is lower, and they do not stay exactly at 0 after the peak.:

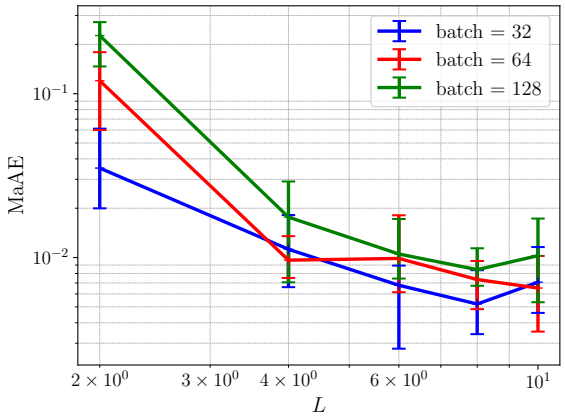
3.7 Depth study

In this part, we set the width to $K = 10$ neurons in each layers, and wish to assess the impact of the depth L of hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each parameter, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range $[\text{minimum}, \text{maximum}]$ of the $P = 5$ values for the given metric. Note that we used $\eta = 10^{-5}$ for the learning step.



(a) MSE

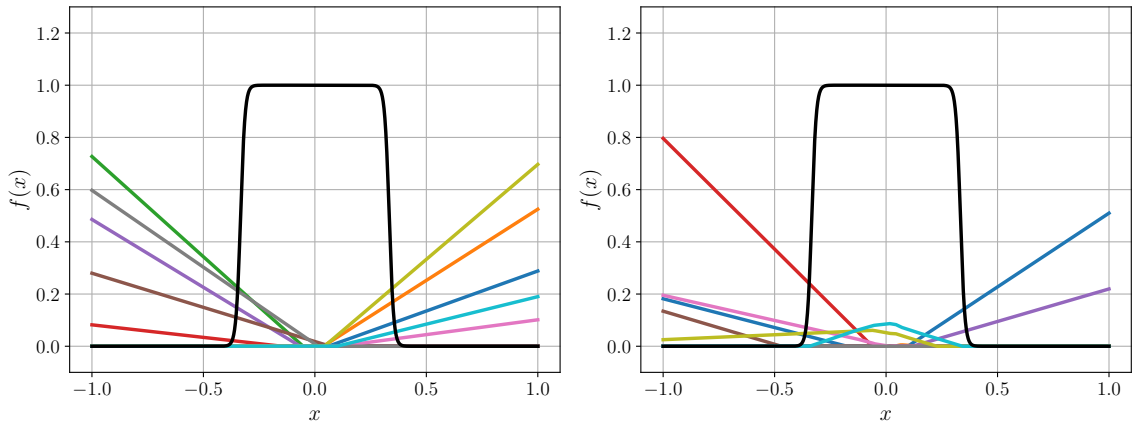
(b) MAE



(c) MaAE

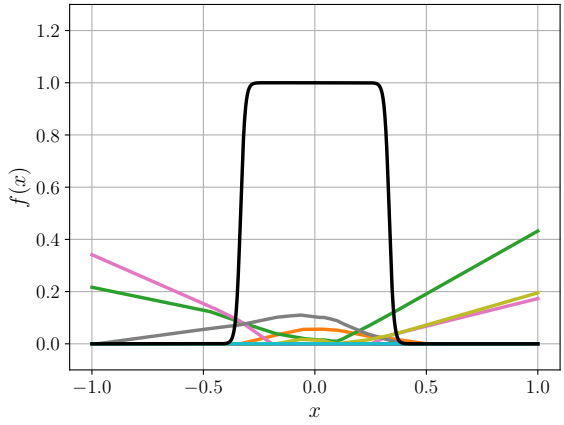
Figure 23: Metrics in function of L for a network with $K = 10$ neurons in each layer for different batch sizes

As can be seen on Figure 23, the metrics don't really appear as a straight line in function of the depth L. For this reason, it does not make sense to perform a linear regression as before to get an effective order of convergence.
However, note that for $K = 10$ and $L = 10$, the total number of parameters to optimize is only 1021 (see Equation 5), whereas for $K = 100$ and $L = 2$ (the best MSE we obtained so far) it was 10401. For $L = 1$ and $K = 100$, the number of parameters was 301.

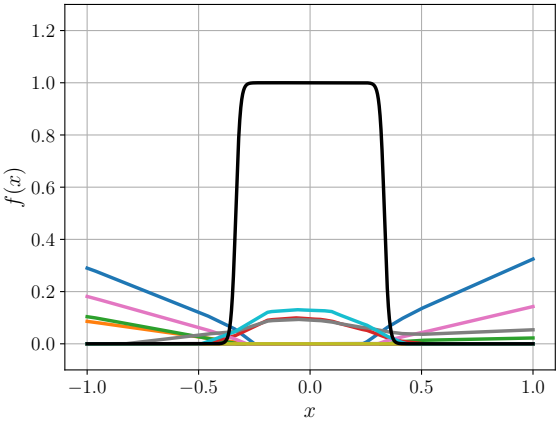


(a) Layer 1

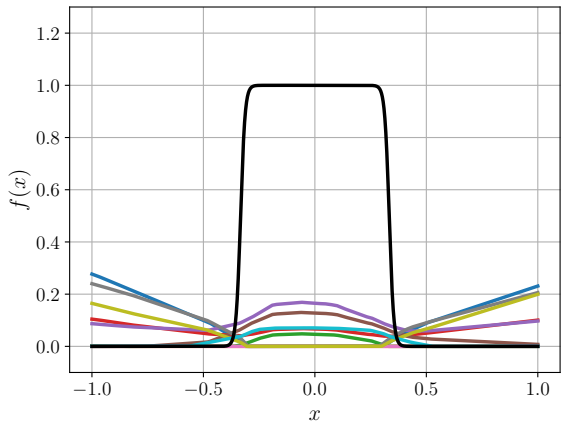
(b) Layer 2



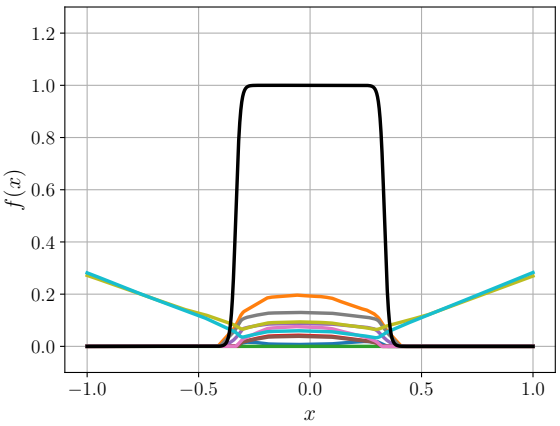
(c) Layer 3



(d) Layer 4

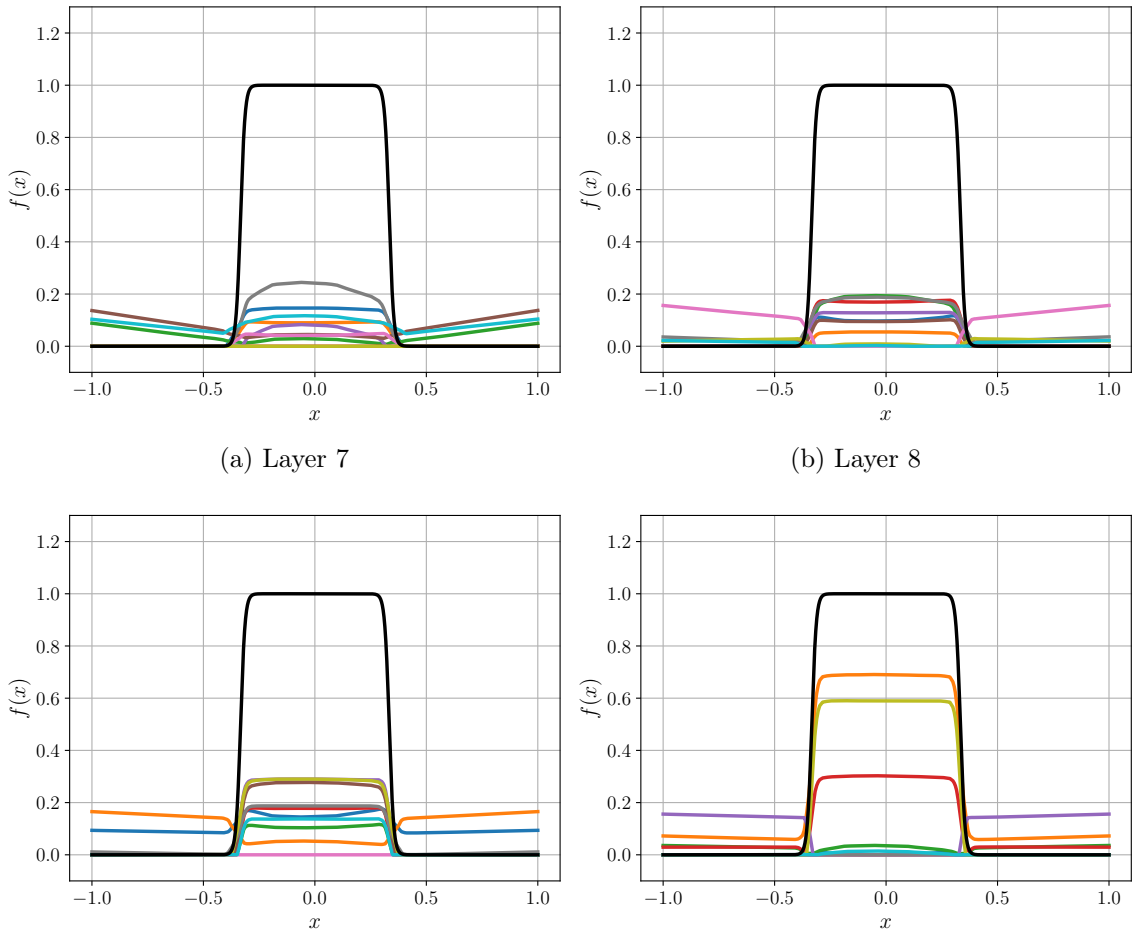


(e) Layer 5



(f) Layer 6

Figure 24: ReLU functions created in each layer for the best model with $L = 10$ and batch 32



(a) Layer 7

(b) Layer 8

(c) Layer 9

(d) Layer 10

Figure 25: ReLU functions created in each layer for the best model with $L = 10$ and batch 32

Figures 24 and 25 show how a network with $L = 10$ uses each layer to approximate the objective function.

4 The 2D case

We are now interested in 2D functions. In this section, we use the following objective function $u : [-1, 1]^2 \to \mathbb{R}$:

$$u(x,y) = \frac{1}{2}\left[1 + \tanh\left(-k\left(\sqrt{(x-x_0)^2 + (y-y_0)^2} - r_0\right)\right)\right] \tag{16}$$

where $k = 50$ is the steepness, $x_0 = y_0 = 0$ are the coordinates of the center of the circle, and $r_0 = 1/2$ is the radius of the circle.

The function has the following contour :

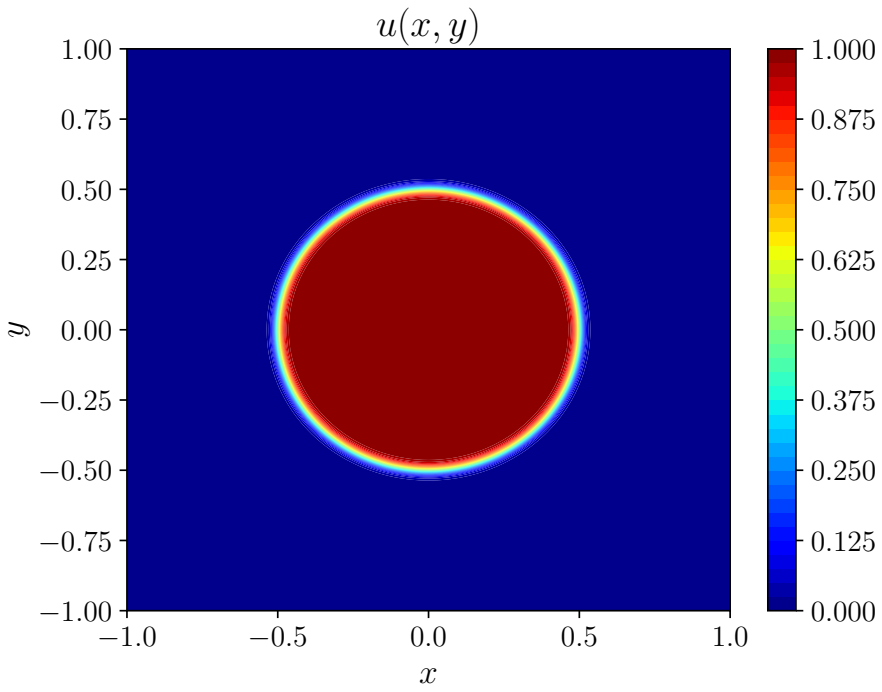


Figure 26: Objective function in 2D

4.1 Dataset and optimization process

As in the 1D case, we take $M = 10^5$ pairs (x, y), where both x and y are uniformly distributed at random between -1 and 1. For all of these pairs (x, y), we compute $u(x, y)$ as in Equation 16. The pairs $((x, y), u(x, y))$ represent the pairs (input, label) or (feature, label) which we will use for training. We separate 10% of the M pairs as a testing set. During training, we also monitor the metrics with 10% of the M pairs as a validation set. This means that we train the network on a total of $M_{\text{eff}} = 80000$ pairs (input, label), assess the metrics during training with 10000 pairs, and still

have a testing set of 10000 pairs to compare the accuracy between different models/architectures of networks.

The loss and optimization process used are the same as in the 1D case (see Section 3.2), unless specifically stated otherwise.

4.2 2 layers network

First, we focus on neural networks with $L = 2$ hidden layers and $K_1 = K_2 := K = 100$ neurons in each layer. As always, ReLU activation functions are used in both hidden layers, and we do not use any activation in the output layer. The optimization parameters are described in Section 3.2 with the exception of the learning step set to $\eta = 5 \cdot 10^{-5}$. In order to take into account the randomness of the initialization of layer weights and optimization process, we train $P = 5$ similar networks. We report the metrics on the testing set for the average, minimum, and maximum of all P networks :

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
4.9175e-06	6.6474e-06	9.7614e-06	1.0634e-03	1.2579e-03	1.5500e-03	2.6920e-02	3.7421e-02	5.0015e-02

Table 8: Metrics for the 2 layers network described above. The mean, min and max are applied over $P = 5$ networks trained in the same conditions

Figures 27 and 28 show respectively the loss during training, and the contour of the approximation and absolute error for the best of the P models. Note that the grid used to display this solution is artificial, and thus has not been trained on. However, the model is able to generalize quite well to this grid.

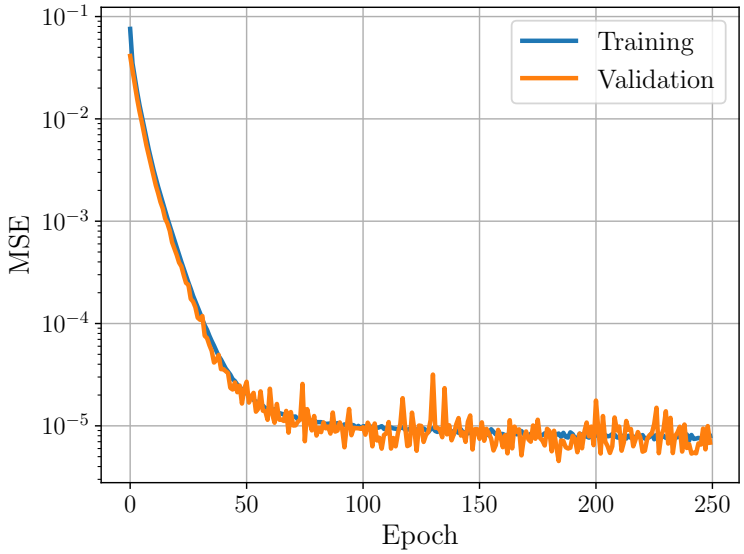
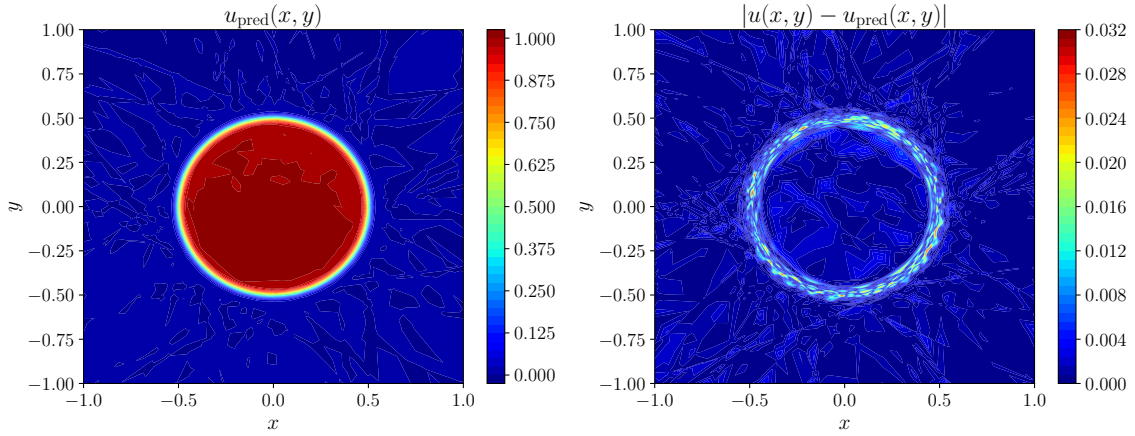


Figure 27: MSE during training for the best of the $P = 5$ models

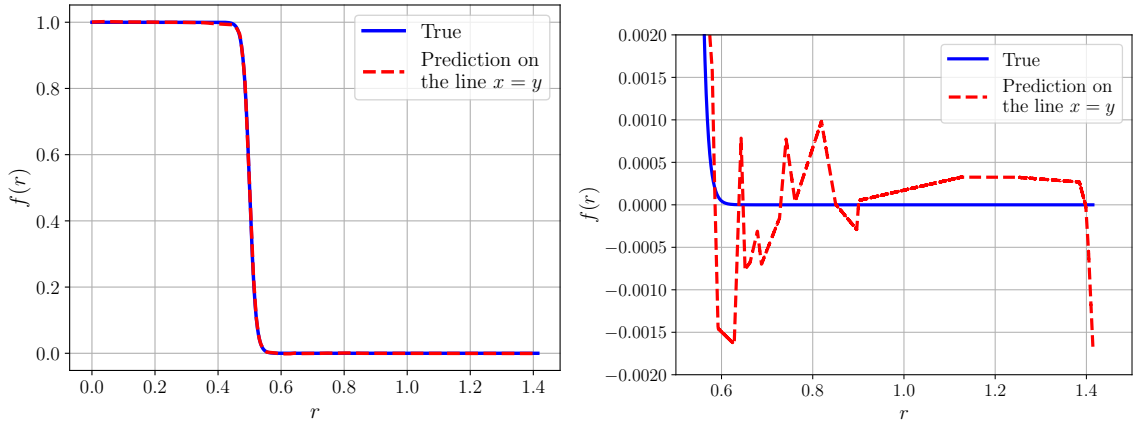


(a) (b)

Figure 28: (a) Approximation given by the best of the $P = 5$ models and (b) Absolute difference between the approximation and the ground-truth

Now, note that the objective is a radial function, but that the approximation is not perfectly radial (see for example Figure 28). This means that the approximation depends on the radius line we choose. In the following, we represent the approximation

on the radius line given by $x = y$, $x \in [0, 1]$, i.e we choose the main diagonal line. It is possible (and almost surely the case) that the approximation is better or worse on other lines from the center. However, looking at Figure 28, the difference between radius lines does not seem too big, motivating the representation on the diagonal.

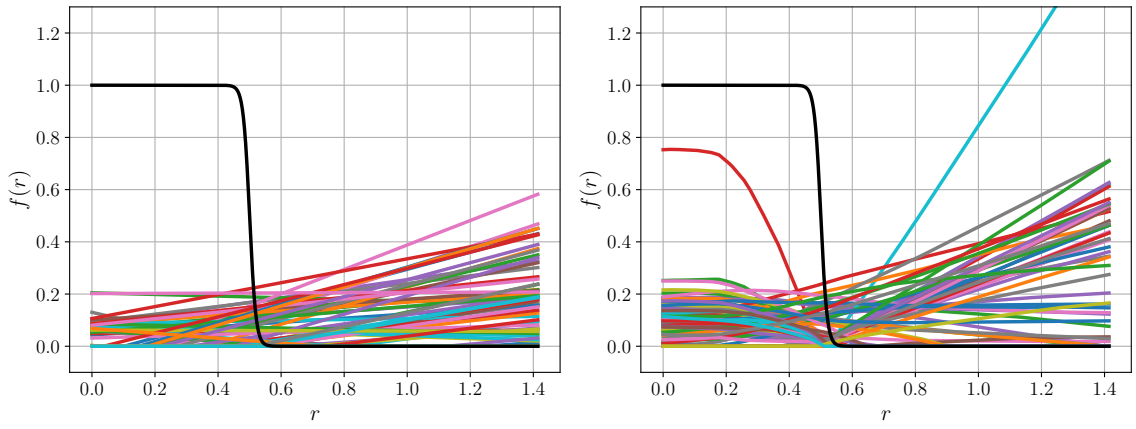


(a) (b)

Figure 29: (a) Approximation given by the best of the $P = 5$ models as a radial function on the line $x = y$ and (b) zoom on the constant part

Figure 29 show the approximation as a radial function as described above. We can see in (b) that the part of the function which should be constant at 0 is not smooth and oscillates around 0, thus explaining the behavior in Figure 28. Indeed, in this figure one can see that the two regions which are 0 and 1 in the objective are not perfectly constant anymore in the approximation.

Figure 30 show the functions created by the first and second layer as a radial function on the line $x = y$. Once again, the black curve represent the objective function. As one can see, it is now very different from what we had in the 1D case (Figure 14).

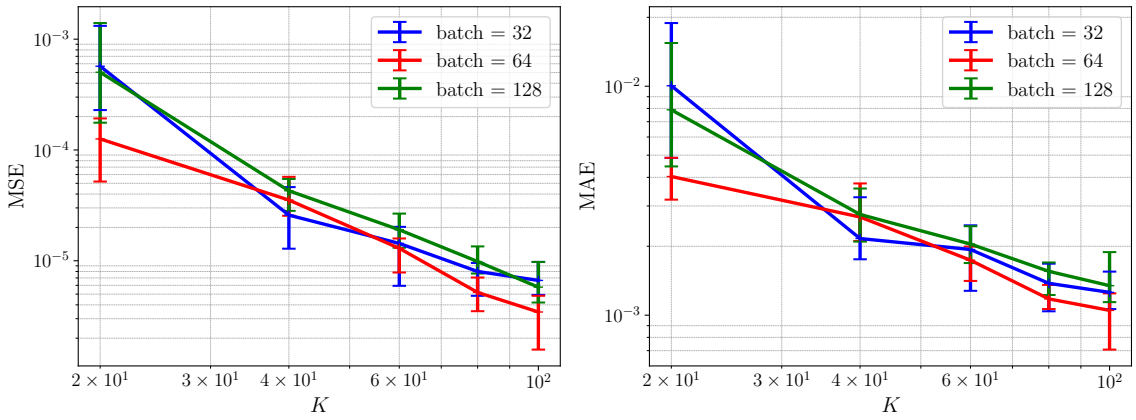


(a) First layer

(b) Second layer

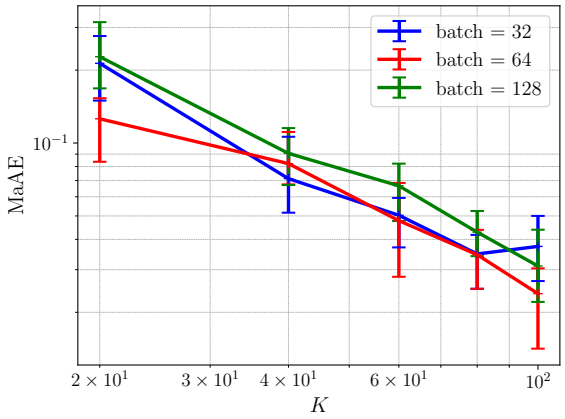
Figure 30: Functions created by the neurons of the model in the first and second layer for the best of the $P = 5$ models

Now, as in 1D, we wish to assess the impact of the number $K_1 = K_2 \coloneqq K$ of neurons in the hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each K, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range [minimum,maximum] of the $P = 5$ values for the given metric. The learning step $\eta = 5 \cdot 10^{-5}$ is fixed for all K and batch sizes.



(a) MSE

(b) MAE



(c) MaAE

Figure 31: Metrics in function of $K_1 = K_2 \coloneqq K$ for a network with $L = 2$ hidden layers

Figure 31 (a) show that the MSE as a function of K are really close to straight lines in the log-log plot. Thus, as before, we perform a linear fit to get an approximation of the order of convergence as a function of K.

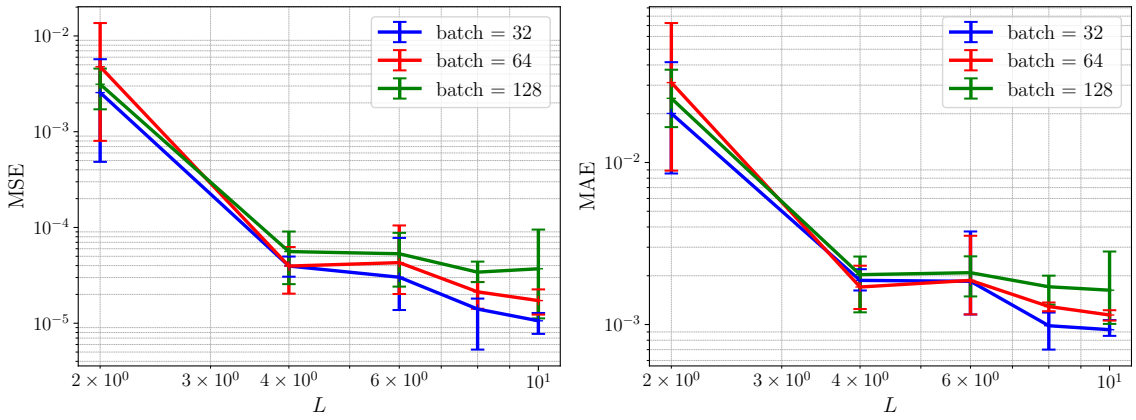
	Slope	Intercept
batch=32	-2.74 ± 0.45	0.12 ± 0.78
batch=64	-2.29 ± 0.13	-0.87 ± 0.23
batch=128	-2.73 ± 0.20	0.16 ± 0.35

Table 9: Results of the linear fit made on the log-MSE and log-K

Table 9 show the results of the linear fit. The approximation quickly becomes better as we increase K, as the slopes of about -2.7 show us.

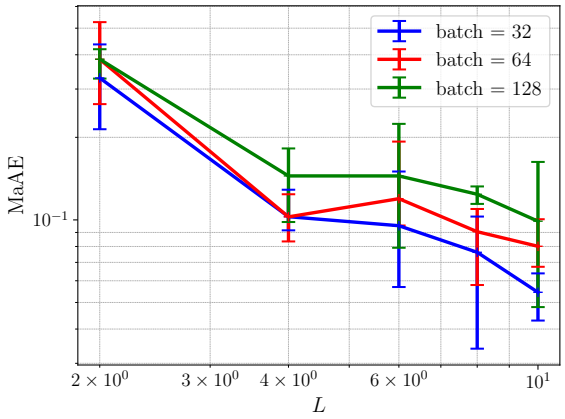
4.3 Depth study

We now set the width to $K = 10$ neurons in each layers, and wish to assess the impact of the depth L of hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each parameter, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range $[\text{minimum}, \text{maximum}]$ of the $P = 5$ values for the given metric. Note that we used $\eta = 5 \cdot 10^{-5}$ for the learning step.



(a) MSE

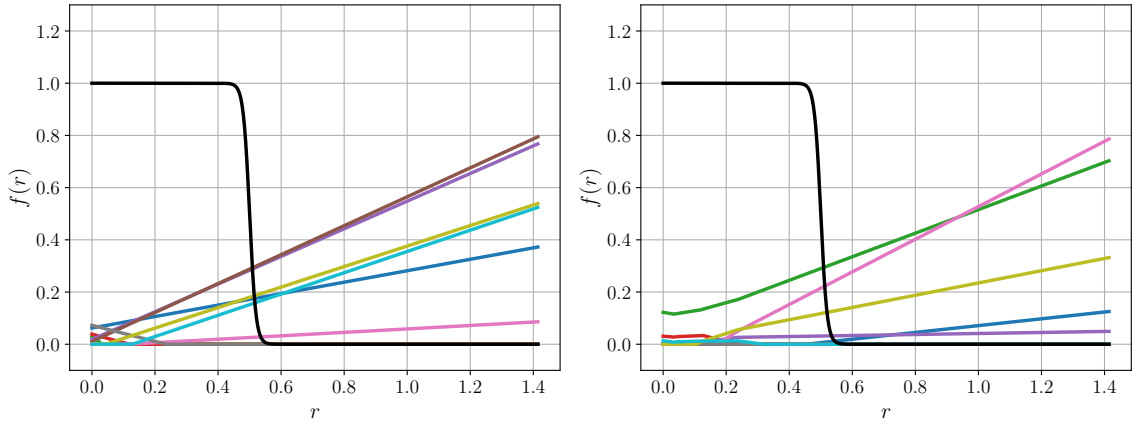
(b) MAE



(c) MaAE

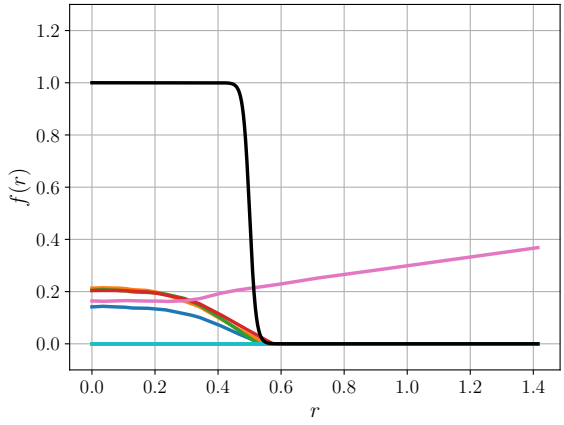
Figure 32: Metrics in function of L for a network with $K = 10$ neurons in each layer for different batch sizes

As in the 1D case, Figure 32 show that the metrics do not really appear as a straight line in function of the depth L. For this reason, it does not make sense to perform a linear regression as before to get an effective order of convergence.
However, note that for $K = 10$ and $L = 10$, the total number of parameters to optimize is only 1031 (see Equation 5), whereas for $K = 100$ and $L = 2$ (the best MSE we obtained so far) it was 10501. Thus in term of number of parameters, we obtain a fairly good approximation with $K = 10$ and $L = 10$ with 10 times less parameters as with $L = 2$ and $K = 100$.

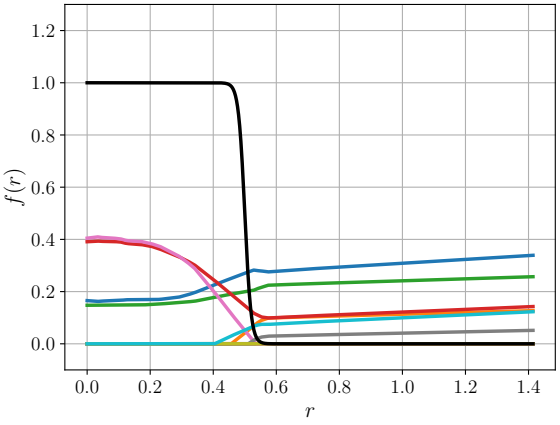


(a) Layer 1

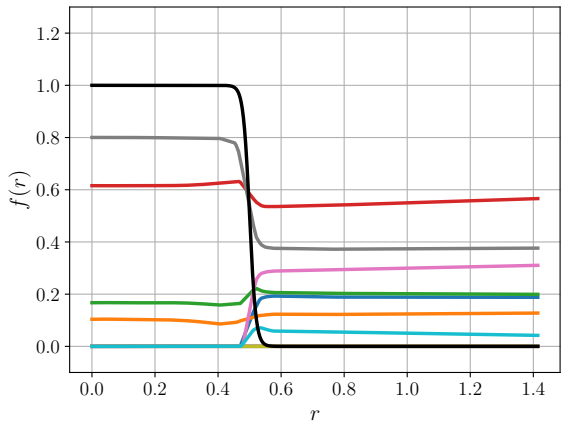
(b) Layer 2



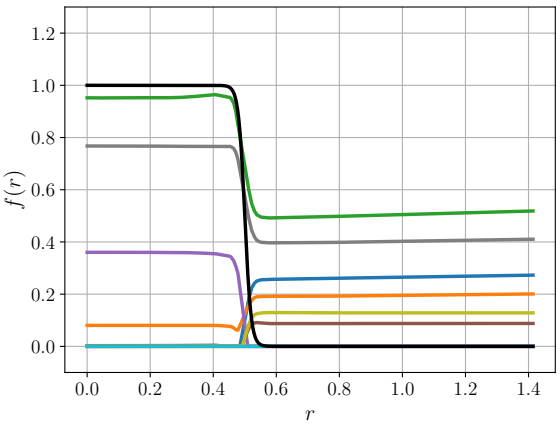
(c) Layer 5



(d) Layer 6



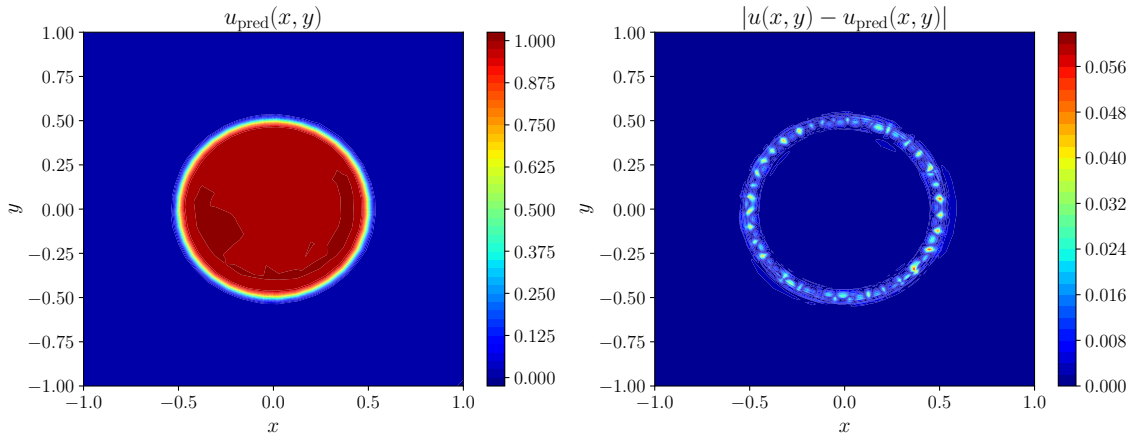
(e) Layer 9



(f) Layer 10

Figure 33: ReLU functions created in each layer for the best model with $L = 10$ and batch 32

Figure 33 shows the functions created in some of the layers of the best model with batch 32 and $L = 10$, on the diagonal line $x = y$. Figure 34 show the approximation and absolute error for that same model. As one can see, this model does generalize better on the constant portion where the function should be 0 before the transition to 1. However, the MSE on the testing set is still higher (7.7401e-06).



(a) (b)

Figure 34: (a) Approximation given by the best of the models and (b) Absolute difference between the approximation and the ground-truth

5 The 10D case

It is well known that finite element methods fail in high dimensions, due to the exponential increase of grid points needed. For example, if we want to discretize the interval $[-1, 1]$ with N points, and we want a uniform grid in d dimensions over $[-1, 1]^d$, then one would need N^d points, which very quickly becomes way too much to computationally handle.

This is where neural networks show all of their effectiveness, as they are not troubled by the increase in dimensions. To investigate this phenomenon, we now go into 10D. In the same spirit as in 2D, we use the following objective function $u : [-1, 1]^{10} \to \mathbb{R}$:

$$u(\boldsymbol{x}) = \frac{1}{2}\left[1 + \tanh\left(-k\left(r - r_0\right)\right)\right] \tag{17}$$

where $k = 50$ is the steepness, $r = ||\boldsymbol{x}||_2$ is the radius (centered at 0 in each direction), and $r_0 = \sqrt{10}/2$ is the radius of the transition (center of the interval for r).

The graph is represented in Figure 17 :

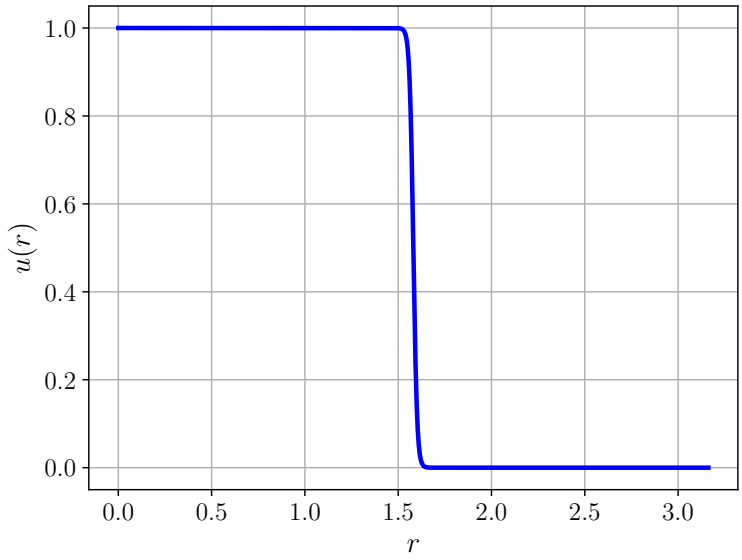
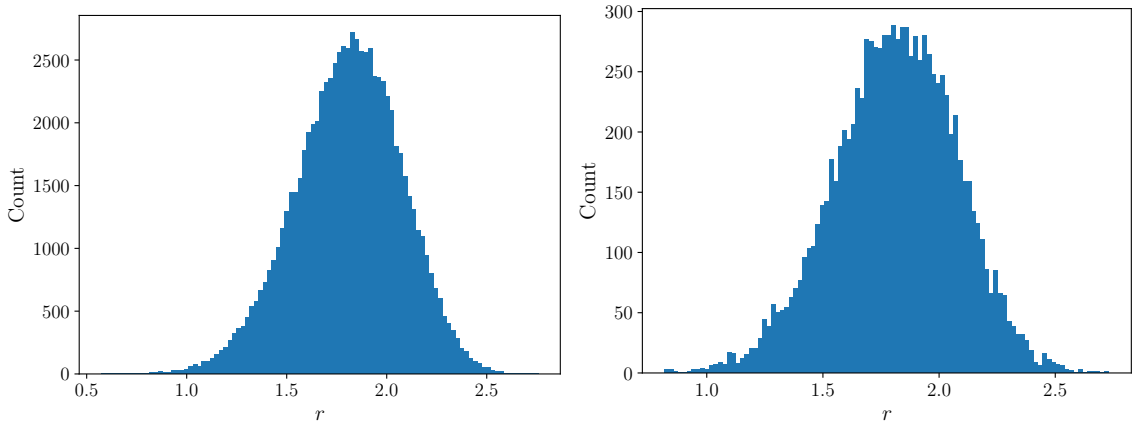


Figure 35: Objective function in 10D

5.1 Dataset and optimization process

The dataset is constructed exactly in the same way as in the 1D and 2D case, except that now the input consist of points $(x_1, ..., x_{10})$ where all x_i $i = 1, ...10$ are uniformly distributed at random between -1 and 1. However, it is well known that as we go into higher dimensions, points are further and further apart. For this reason, we begin by checking the repartition of radii in the dataset :



(a) Training set (b) Testing set

Figure 36: Distribution of radii in the dataset. (a) training set, and (b) testing set

As one can see on Figure 36, the radii in the dataset are far from uniformly distributed between 0 and $\sqrt{10} \simeq 3.16$. Indeed, almost all values are in the interval $[1.0, 2.5]$. Thus, we have absolutely no representative of very low radius (near 0), and high radius (near maximum, $\sqrt{10}$). One has to keep that in mind during training, and when setting the radius of transition r_0 in the objective function. Indeed, this is likely that the model will have a hard time to generalize to low and high radii since it cannot train on such data.

The loss and optimization process used are the same as in the 1D and 2D cases (see Section 3.2), unless specifically stated otherwise.

5.2 2 layers network

As usual, we start by focusing on networks with $L = 2$ hidden layers, but this time we set $K = 200$ neurons in each layer. This is motivated by the fact that the task in now harder in 10D. We use ReLU activation in both layers, and the learning step is $\eta = 5 \cdot 10^{-5}$. We train $P = 5$ similar networks, and we report the metrics on the testing set for the average, minimum, and maximum of all P networks :

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
5.3231e-03	5.6457e-03	6.0836e-03	2.4421e-02	2.6226e-02	2.7019e-02	7.8892e-01	8.4377e-01	9.0651e-01

Table 10: Metrics for the 2 layers network described above. The mean, min and max are applied over $P = 5$ networks trained in the same conditions

As for the 2D case, the objective is a radial function, but not the approximation. For this reason, and in the same spirit as before, we represent all results on the main diagonal line $x_1 = x_2 = ... = x_{10}$. Moreover, note that when we represent the approximation as a function of the radius, we do so on an artificial dataset, consisting of points on the diagonal (thus representing all radii from 0 to $\sqrt{10}$). However, the reader should not forget Figure 36 : the approximation for low and high radii is very likely to be erroneous, due to the lack of training data.

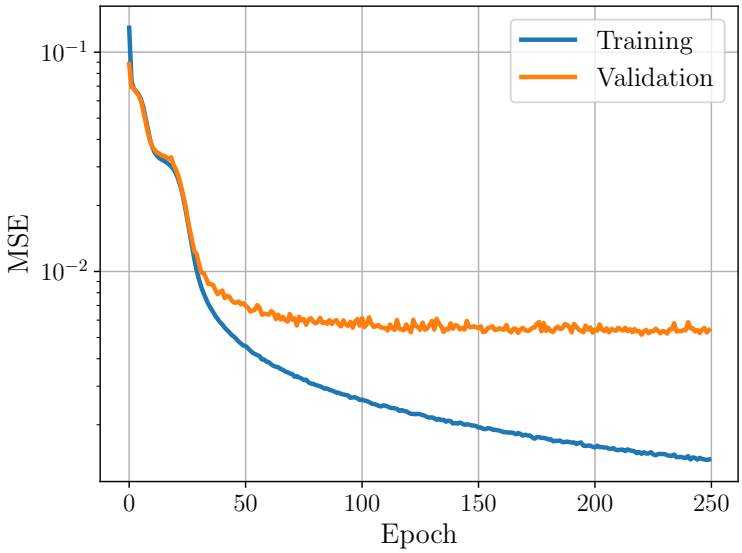
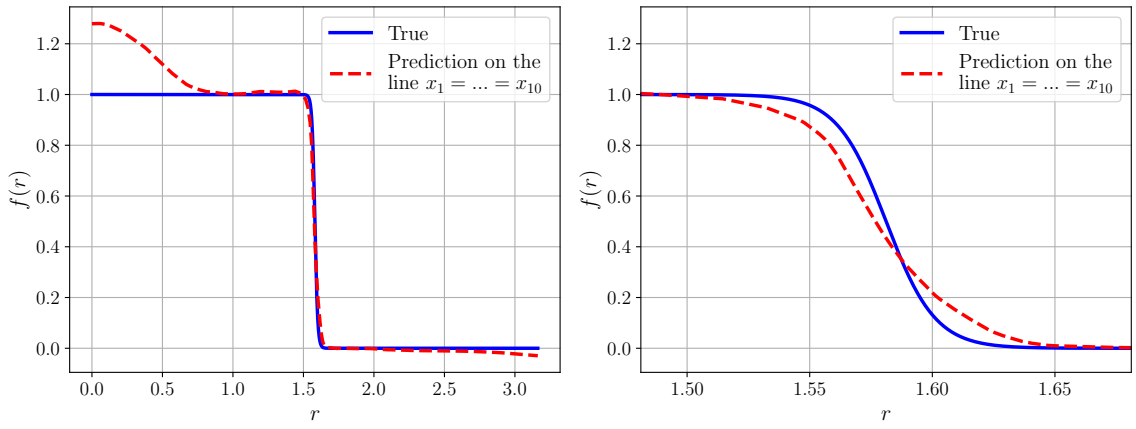
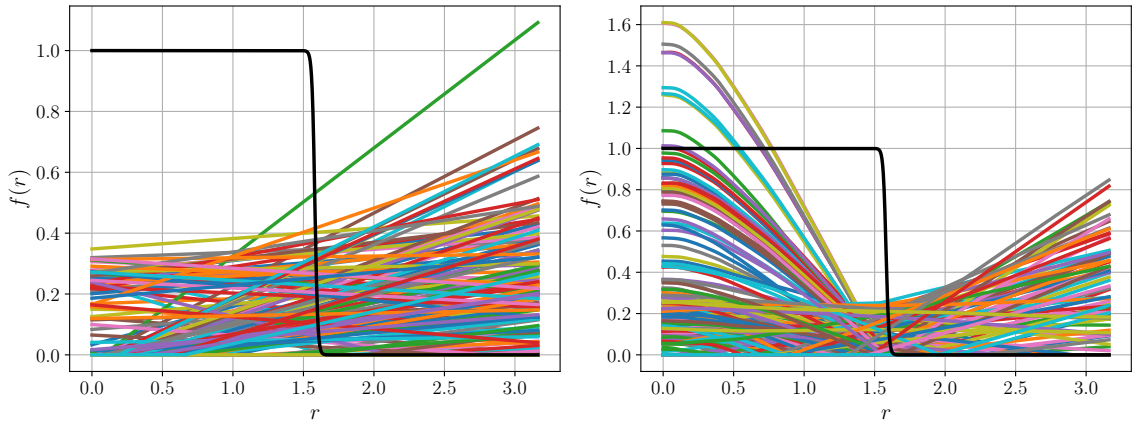


Figure 37: MSE during training for the best of the $P = 5$ models



(a) (b)

Figure 38: Approximation of the objective function by the best of the $P = 5$ models. a) Approximation on the line $x_1 = x_2 = ... = x_{10}$, $x_i \in [0, 1]$ $i = 1, ...10$, and b) zoom around the transition



(a) First layer (b) Second layer

Figure 39: Functions created by the neurons on the line $x_1 = x_2 = ... = x_{10}$, $x_i \in [1/2, 1]$ $i = 1, ...10$. a) First layer, and b) Second layer

Figures 37, 38 and 39 show the MSE during training, the approximation on the diagonal line and the functions created in each neurons on the diagonal line.
As suspected, the network performs very badly for low radii. For high radii, the

approximation is however decent. Still, when zooming on the transition, one may see that the approximation is not that good. Table 10 does confirm this fact, with MSE of the order of 10^{-3} and very high MaAE of about $8 \cdot 10^{-1}$.

The reason for this lack of precision is that the network struggles to approximate the radial function, $r = ||\boldsymbol{x}||_2$. Indeed, if one uses the same objective function (Equation 17) but replaces $r = ||\boldsymbol{x}||_2$ by $r = \frac{1}{10}\sum_i x_i$ (i.e the mean of the components) and thus try to approximate the objective (Equation 18), then the approximation is way better since r is linear in the input x_i.

$$u(\boldsymbol{x}) = \frac{1}{2}\left[1 + \tanh\left(-k\left(\frac{1}{10}\sum_{i=1}^{10} x_i - r_0\right)\right)\right] \tag{18}$$

with $k = 50$ as the steepness and $r_0 = 0$ (the center of the interval for $r = \frac{1}{10}\sum_i x_i$).

This fact is demonstrated with a model with $L = 2$ hidden layers and $K = 100$ neurons in each layer. In the following Figures 40 and 41, one is able to see that the approximation is very accurate (MSE of the order of 10^{-7}, and MaAE of the order of 10^{-3}). The model is even able to generalize for low r (around -1) and high r (around 1), even though the training set lacks this kind of data (see Figure 42).

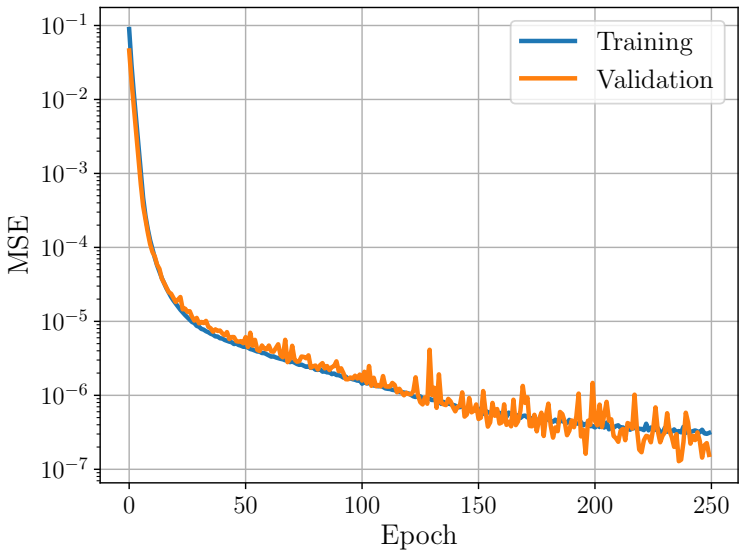
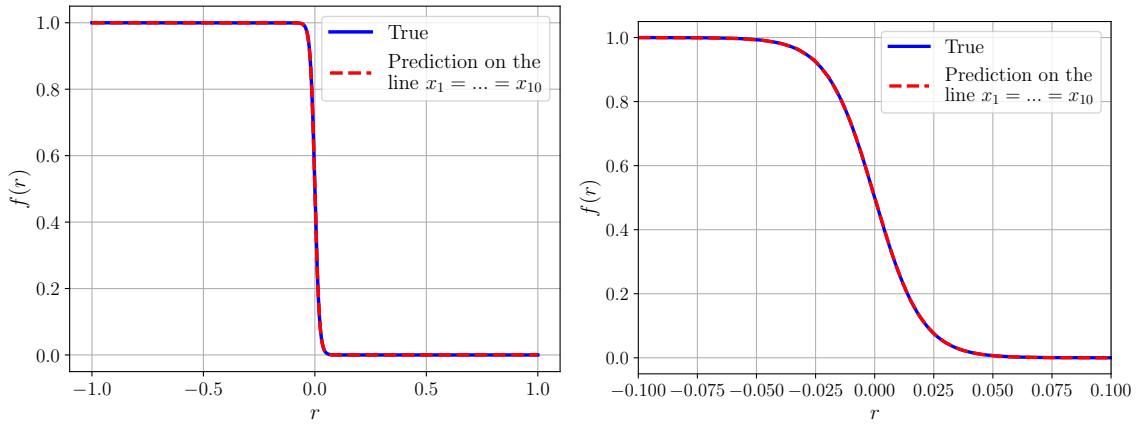
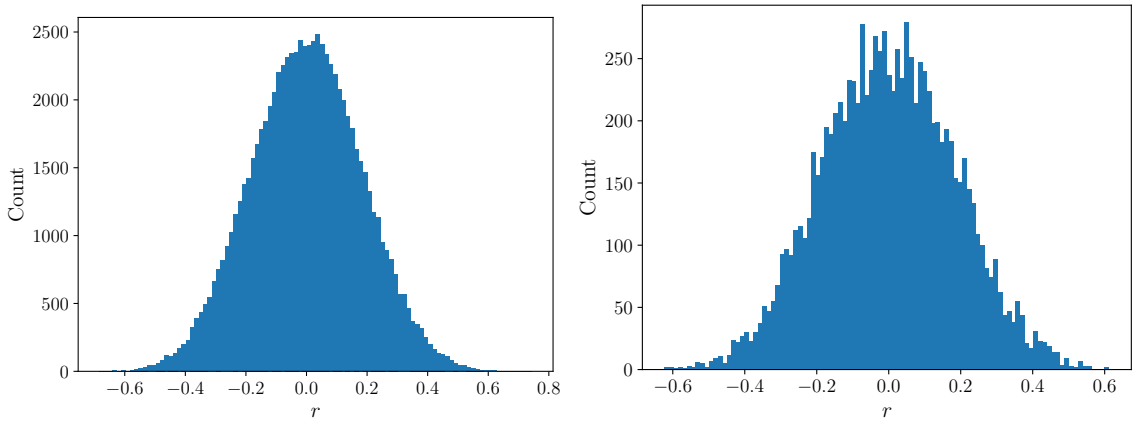


Figure 40: MSE during training for a model approximating the objective Equation 18



(a) (b)

Figure 41: Approximation of the objective function (Equation 18) (a) Approximation on the line $x_1 = x_2 = ... = x_{10}$, $x_i \in [0, 1]$ $i = 1, ...10$, and (b) zoom around the transition

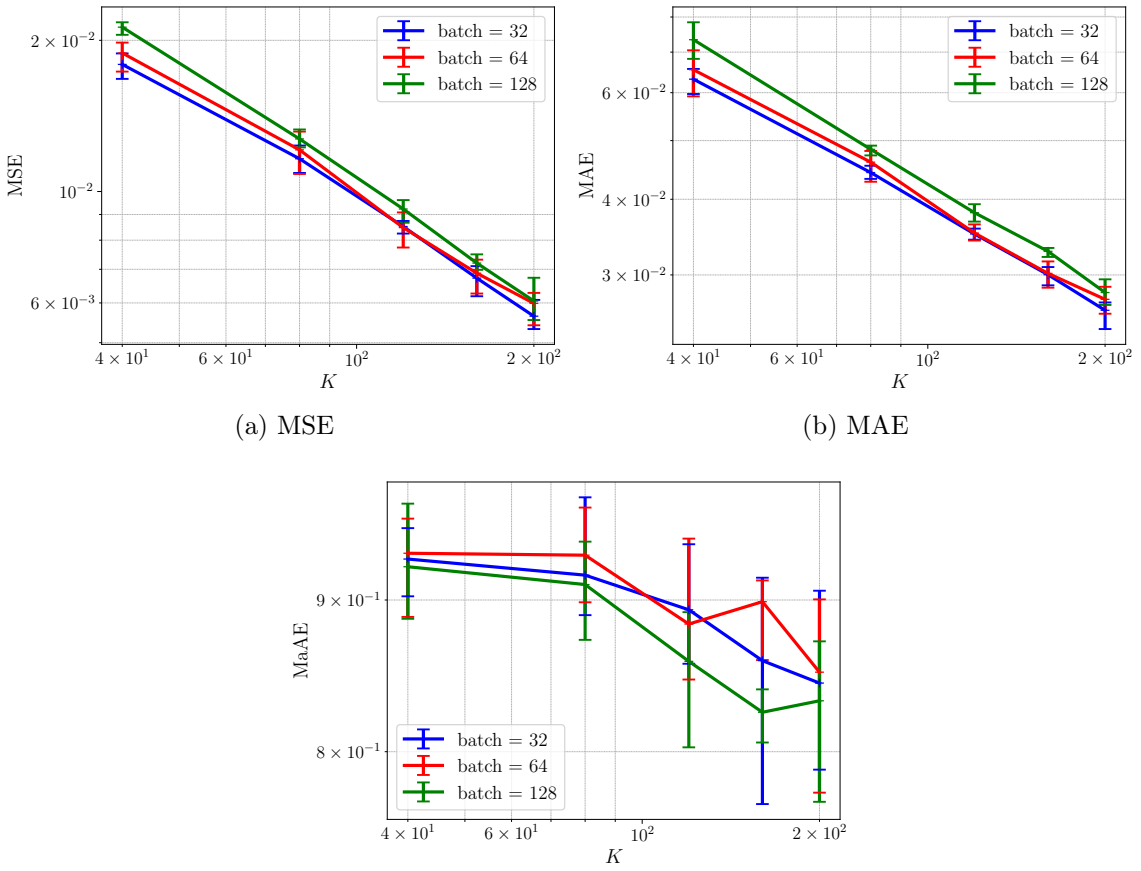


(a) Training set (b) Testing set

Figure 42: Distribution of $r = \frac{1}{10}\sum_i x_i$ in the dataset using the objective (Equation 18). (a) training set, and (b) testing set

We now go back to approximate the radial objective (Equation 17). We still wish to assess the impact of the number $K_1 = K_2 := K$ of neurons in the hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$

different models with the same parameters. Moreover, we perform this study for different batch sizes. For each K, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range [minimum,maximum] of the $P = 5$ values for the given metric. The learning step $\eta = 5 \cdot 10^{-5}$ is fixed for all K and batch sizes.



(a) MSE

(b) MAE

(c) MaAE

Figure 43: Metrics in function of $K_1 = K_2 \coloneqq K$ for a network with $L = 2$ hidden layers

Figure 43 (a) show that the MSE as a function of K are really close to straight lines in the log-log plot. Thus, as before, we perform a linear fit to get an approximation of the order of convergence as a function of K.

	Slope	Intercept
batch=32	-0.72 ± 0.03	-0.58 ± 0.05
batch=64	-0.73 ± 0.02	-0.55 ± 0.05
batch=128	-0.78 ± 0.01	-0.41 ± 0.03

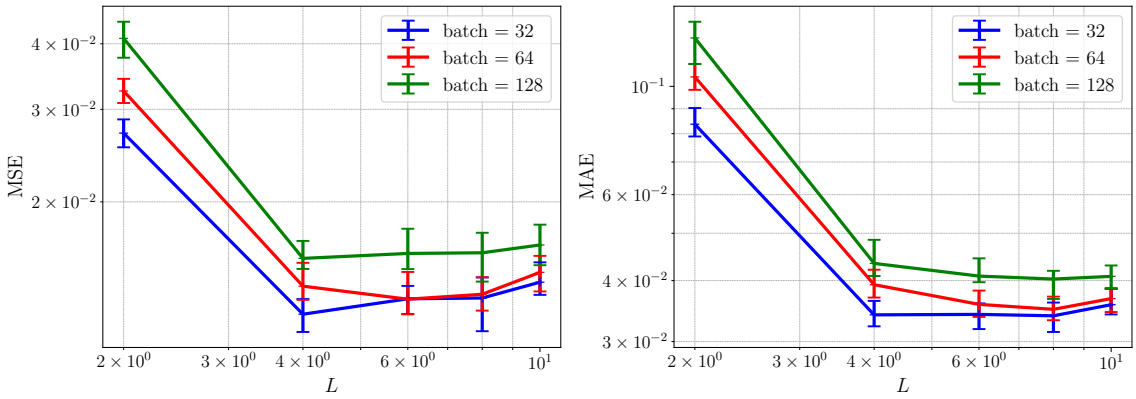
Table 11: Results of the linear fit made on the log-MSE and log-K

Table 11 reveal a slow increase in performance (MSE) as K increases (slope of about -0.7). However, Figure 43 (c) reveals that the MaAE almost do not decreases as K increases. Thus the approximation stays quite far from the objective at some points.

5.3 Depth study

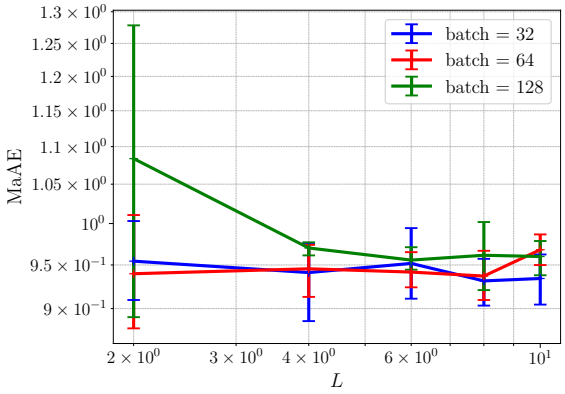
We now set the width to $K = 20$ neurons in each layers (instead of 10 in 1D and 2D), and wish to assess the impact of the depth L of hidden layers in the model performance. To account for the stochasticity, we always train $P = 5$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each parameter, we report the mean of the $P = 5$ models for a given metric, and represent as error bars the range $[\text{minimum}, \text{maximum}]$ of the $P = 5$ values for the given metric. Note that we used $\eta = 5 \cdot 10^{-5}$ for the learning step.

Figure 44 show that the metrics do not decrease as we increase L after $L = 4$. For this reason, it seems that in higher dimensions (here 10D), it is better to use more neurons than more layers.



(a) MSE

(b) MAE



(c) MaAE

Figure 44: Metrics in function of L for a network with $K = 20$ neurons in each layer for different batch sizes

6 Conclusion

In this report, we studied how neural networks with ReLU activation functions create an approximation of real functions in one and several dimensions. We began by comparing the structure with finite element methods, but it appeared that neural networks create their approximation in a completely different way. Even when "forced" to reproduce the structure of finite element with convolutional layers and fixed first dense layer, the network did not quite correctly reproduce the structure. Then in 1D and 2D, we found that increasing the number of layers instead of the

number of neurons in a fixed number of layers was better in the sense that the approximation is quickly as good but with fewer parameters to optimise. However, in 10D, increasing the number of layers was not significant, and it was therefore better to increase the number of neurons in each layer. Moreover, in 10D the approximation of a radial function was quite difficult, whereas the approximation of a function depending only on the mean of components of the input was very easy and accurate.

Ressources

All the code and tools (including all the trained models) used in this project are made available in the following public GitHub repository https://github.com/Cyrilvallez/Semester_project_NN

References

[1] Ronald DeVore, Boris Hanin, Guergana Petrova, *Neural Network Approximation.* Available on https://arxiv.org/pdf/2012.14501.pdf

[2] Diederik P. Kingma, Jimmy Lei Ba, *Adam: A Method for Stochastic Optimization.* Available on https://arxiv.org/pdf/1412.6980.pdf

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* Available on https://arxiv.org/pdf/1502.01852.pdf

[4] Yann LeCun, Y. Bengio, and Geoffrey Hinton, *Deep learning*

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning, volume 1*