Comparison between finite element method and neural networks

Cyril Vallez
cyril.vallez@epfl.ch

November 4, 2021

Abstract

Contents

1 Introduction

The goal of this project is to compare the finite element method and neural networks for the approximation of functions in one and several dimensions. While doing so, we also try to gain insight on how the structure of the network impact the accuracy of the solution.

2 Description of the problem

2.1 Neural networks

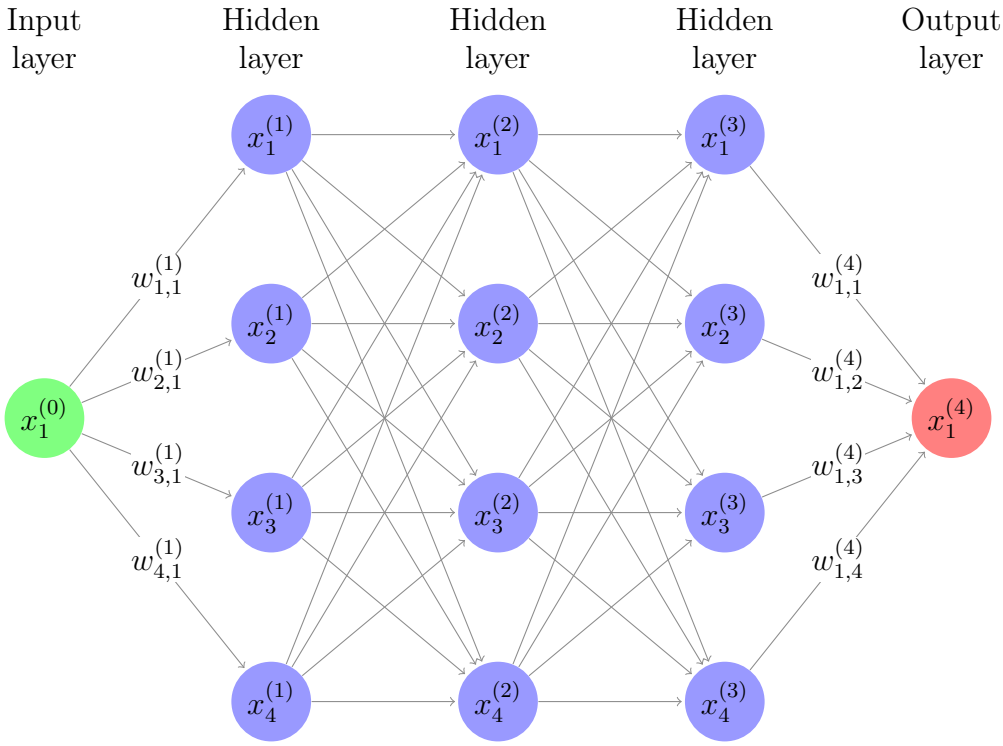


Figure 1: Example of a fully connected layer with $L = 3$, $K_1 = K_2 = K_3 = 4$ and one-dimensional input and output ($K_0 = K_4 = 1$). The biases are not represented for simplicity.

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.

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 :

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

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 (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 :

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

2.2 Finite element method

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

$$h = \frac{1}{N-1} \qquad x_i = ih \quad i = 0, 1, ...N-1 \tag{5}$$

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

$$\Pi_h u(x) = \sum_{i=1}^{N-2} u(x_i)\phi_i(x) \tag{6}$$

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 :

$$\phi_i(x) = \begin{cases} 0 & \text{if } x \notin [x_{i-1}, x_{i+1}] \\ \frac{1}{h}(x - x_{i-1}) & \text{if } x_{i-1} \leq x \leq x_i \\ \frac{1}{h}(x_{i+1} - x) & \text{if } x_i < x \leq x_{i+1} \end{cases} \tag{7}$$

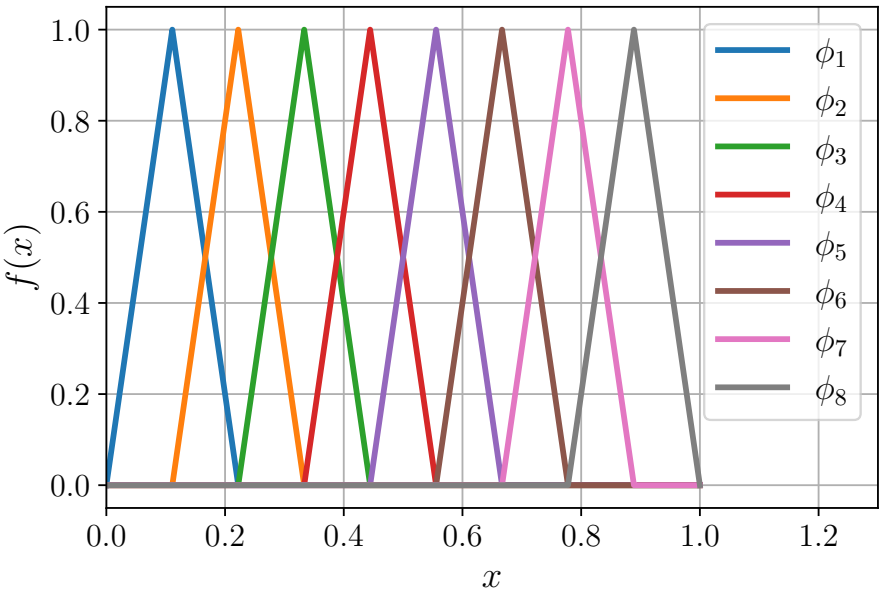


Figure 2: Basis functions for $N = 10$ points.

Note that this formulation can be easily generalized to functions with support is different from $[0, 1]$ and non-homogeneous boundary conditions.
Then, if the function $u(x)$ is unknown but is the solution of some 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 (7) can be rewritten as :

$$\phi_i(x) = \frac{1}{h}\Big[\mathrm{ReLU}(x - x_{i-1}) - 2 \cdot \mathrm{ReLU}(x - x_i) + \mathrm{ReLU}(x - x_{i+1})\Big] \tag{8}$$

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

From this, we can reconstruct (6) with the following biases and weights (noting that both the input and output are one-dimensional) :

$$W^{(1)} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^N \qquad B^{(1)} = - \begin{pmatrix} x_0 \\ \vdots \\ x_N \end{pmatrix} \in \mathbb{R}^N \tag{9}$$

$$W^{(2)} = \frac{1}{h} \begin{pmatrix} u(x_1) \\ u(x_2) - 2u(x_1) \\ u(x_1) - 2u(x_2) + u(x_3) \\ \vdots \\ u(x_{N-3}) - 2u(x_{N-2}) + u(x_{N-1}) \\ u(x_{N-2}) - 2u(x_{N-1}) \\ u(x_{N-1}) \end{pmatrix}^T \in \mathbb{R}^{1 \times N} \qquad B^{(2)} = 0 \in \mathbb{R} \tag{10}$$

where the x_i $i = 0, ..., N-1$ are defined in (5).

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 Numerical experiments and results

In this section, we take the following sharp transition function as objective $u : [0, 1] \to \mathbb{R}$:

$$u(x) = \frac{1}{2}\Big(\tanh(k(x - x_1)) - \tanh(k(x - x_2))\Big) \tag{11}$$

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 $x_2 = 2/3$ so that we have the nice symmetry property $u(x) = u(1-x)$.

The function has the following graph :

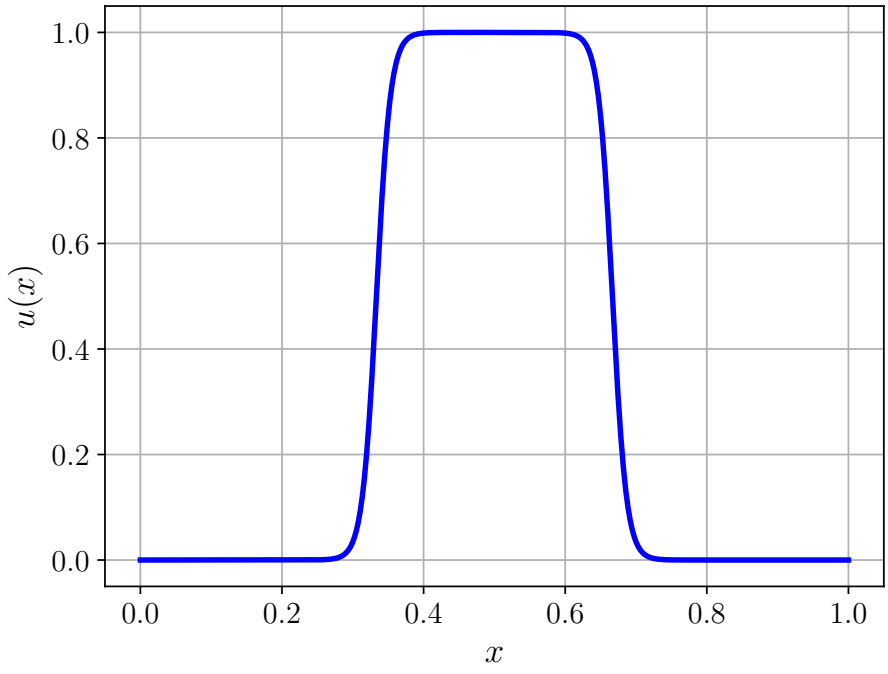


Figure 3: Graph of $u(x)$ as defined in (11)

Note that $u(0) = u(1)$ is not identically 0 for $u(x)$ as in (11). However, it is asymptotically 0, and the numerical evaluation gives $u(0) = u(1) \simeq 3.33 \cdot 10^{-15}$ which is clearly negligible for numerical purposes.

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 $M = 10^6$ points uniformly distributed at random between 0 and 1. For all of these points x, we compute $u(x)$ as in (11). 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 remaining 90000 pairs as a validation set. This means that we train the network on a total of $M_{\text{eff}} = 81000$ pairs (input, label), assess the metrics during training with 9000 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 (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 :

$$\text{MSE} = \frac{1}{M_{\text{eff}}} \sum_{i=1}^{M_{\text{eff}}} (u^i_{pred} - u^i_{true})^2 \tag{12}$$

where u^i_{pred} is the output of the network, and u^i_{true} 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 $\eta = 10^{-2}$ for the learning step.
In the same spirit, if not explicitly specified, we use a batch size of 32 for the gradient updates. We always perform 200 epochs during training, meaning that each of the M_{eff} pair (input, label) is seen exactly 200 times by the network.

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 (12). Another metric that we will sometimes use is the Mean-Absolute-Error (MAE) :

$$\text{MAE} = \frac{1}{M_{\text{eff}}} \sum_{i=1}^{M_{\text{eff}}} |u^i_{pred} - u^i_{true}| \tag{13}$$

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) :

$$\text{MaAE} = \max_{i=1,\dots,M_{\text{eff}}} |u^i_{pred} - u^i_{true}| \tag{14}$$

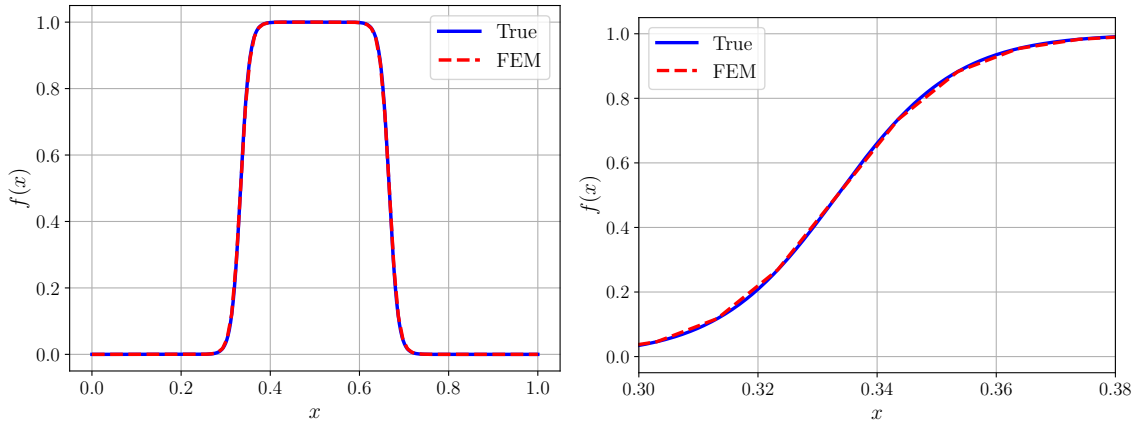
The MaAE gives a more localized feedback, telling us how the approximation behaves at worst. The worst points are likely to be around the transitions between O and 1 at $x = 1/3$ and $x = 2/3$, which are the regions of most interest.

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 (9) and (10) (or alternatively the solution directly given by (6) with $N = 100$). First we compute this interpolant and evaluate it on the testing set. The (sorted) testing set and approximation are represented in (4). The metrics are summarized in (1).

MSE	MAE	MaAE
5.8733e-06	8.8205e-04	1.1854e-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 (11) (blue line) and its interpolant (6) 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). In order to take into account the randomness of the initialization of layer weights and optimization process, we train $P = 10$ 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
1.1091e-04	9.8764e-04	6.0960e-03	5.6784e-03	1.7622e-02	5.8284e-02	5.1414e-02	6.8159e-02	1.4588e-01

Table 2: Metrics for the 1 layer network described above. The mean, min and max are applied over $P = 10$ networks trained in the same conditions

For the best of these $P = 10$ models (the model which achieves the lowest MSE, MAE and MaAE), we show the training and validation MSE during training (5), and the resulting approximation on the test set (6) :

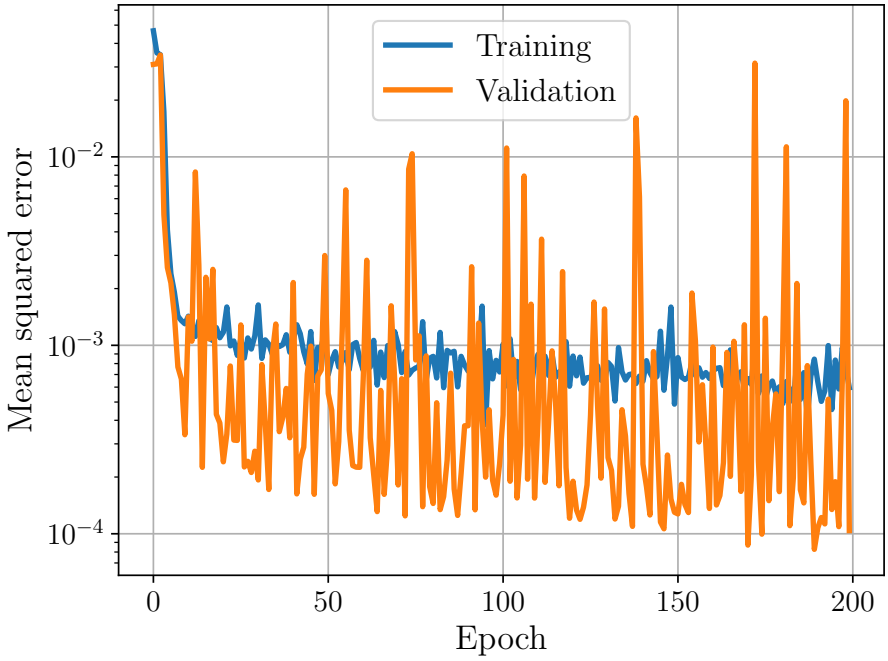
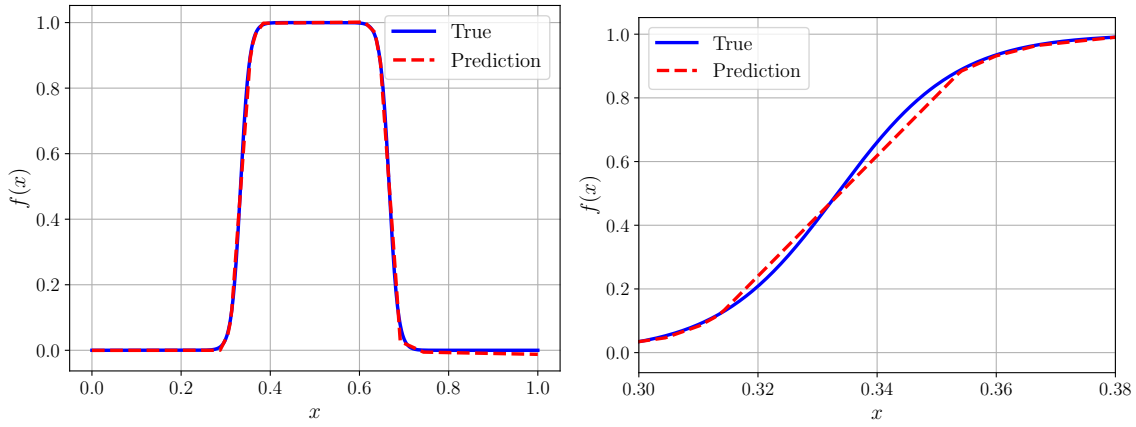


Figure 5: MSE during training for the best of the $P = 10$ models



(a) (b)

Figure 6: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 10$ 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 :

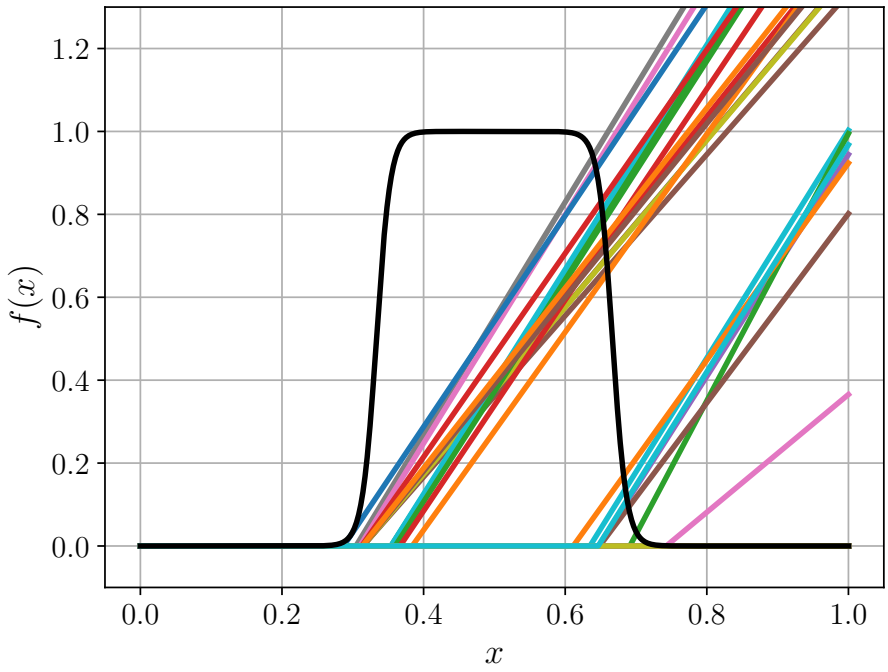


Figure 7: Visualization of the ReLU functions created by the neurons of the model. The curve in black is the ground-truth $u(x)$.

Figure (7) shows that only about 20 out 100 neurons activate on the interval of interest $[0, 1]$.
From this, one immediately sees that the approximation built by the network is very different from the interpolant given in equation (6). Thus, the idea is now to try to initialize the network with the exact weights given in equation (9) and (10) (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 $[0, 1]$ after only 20 epochs of training after initializing with the exact weights given in equation (9) and (10). 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.

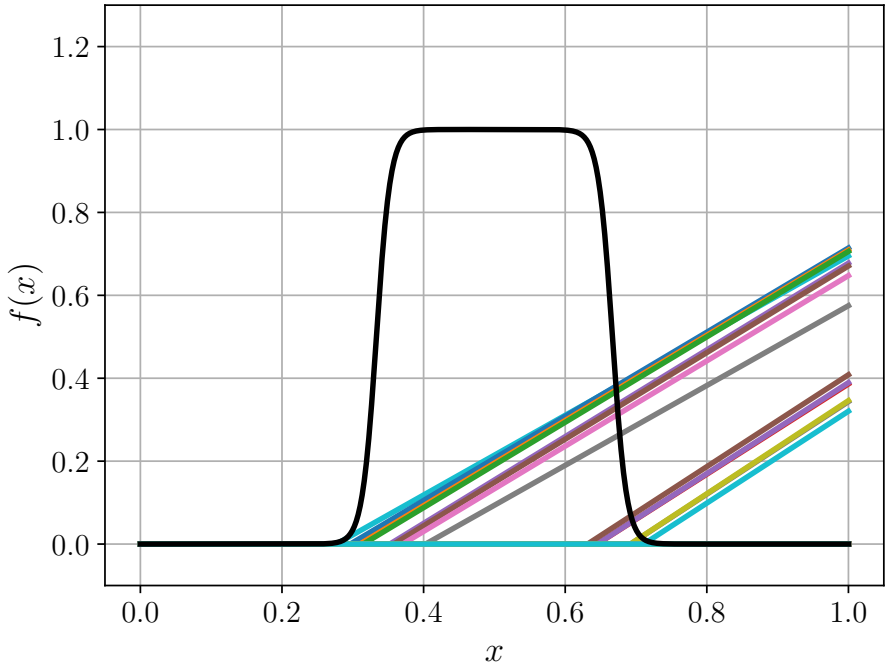


Figure 8: Visualization of the ReLU functions after training for 20 epochs using as initial weights equations (9) and (10)

MSE	MAE	MaAE
3.4056e-04	1.3699e-02	6.1593e-02

Table 3: Metrics after training for 20 epochs using as initial weights equations (9) and (10)

Table (3) shows the metrics for the same model. This is clearly worse than the metrics for the interpolant given in table (1), showing that the solution moves away from finds a worse minimum.

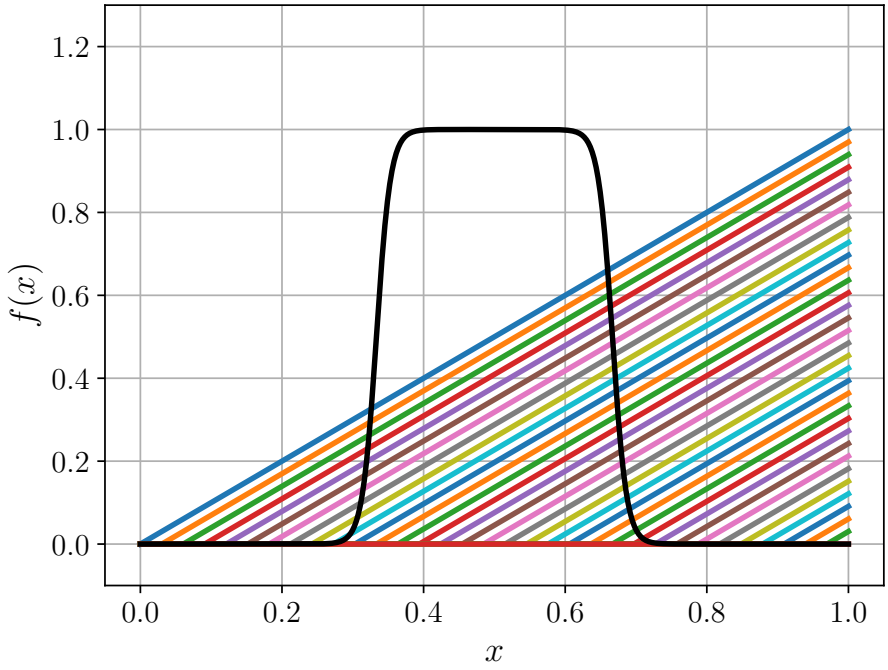
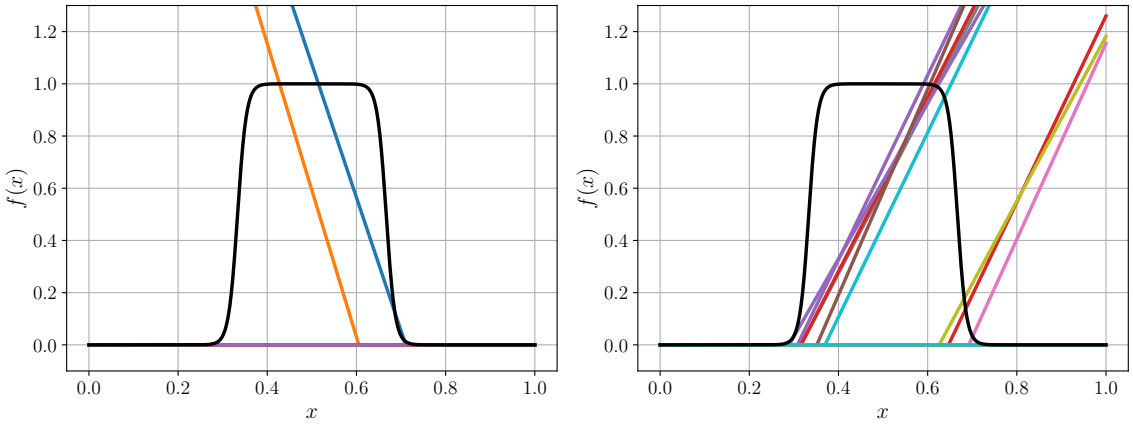


Figure 9: Visualization of the ReLU functions when initializing as in equation (9 (we represented only 1 out of 3 of the $K_1 = 100$ ReLU functions for clarity)

Figure (10) compares how the same network with different K_1 creates ReLU function in its neurons. Table (4) shows the metrics for each K_1.

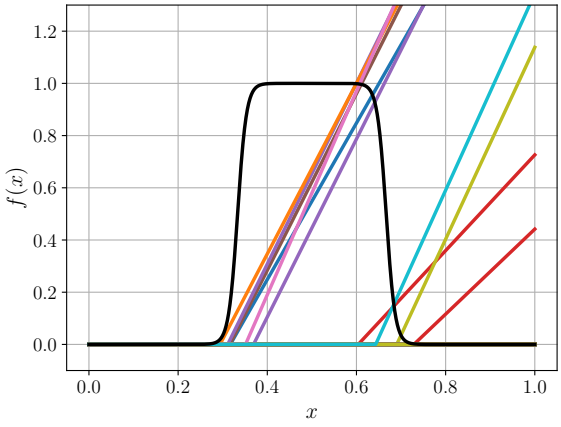
	MSE	MAE	MaAE
$K = 5$	3.2916e-02	1.2900e-01	4.7119e-01
$K = 20$	8.6222e-05	4.5529e-03	6.2654e-02
$K = 40$	1.3926e-03	2.7399e-02	9.1084e-02
$K = 60$	2.3304e-04	1.2109e-02	5.3304e-02
$K = 80$	4.3484e-03	4.9419e-02	1.2310e-01
$K = 200$	6.0840e-05	4.9979e-03	4.7195e-02

Table 4: Metrics for the different networks

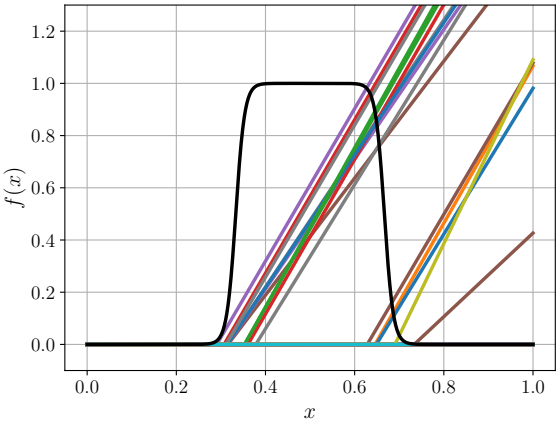


(a) $K_1 = 5$

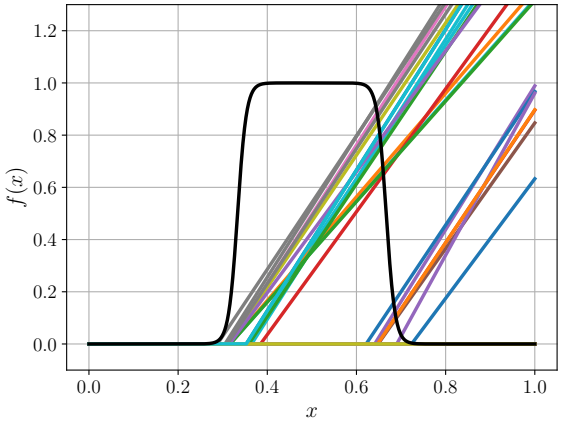
(b) $K_1 = 20$



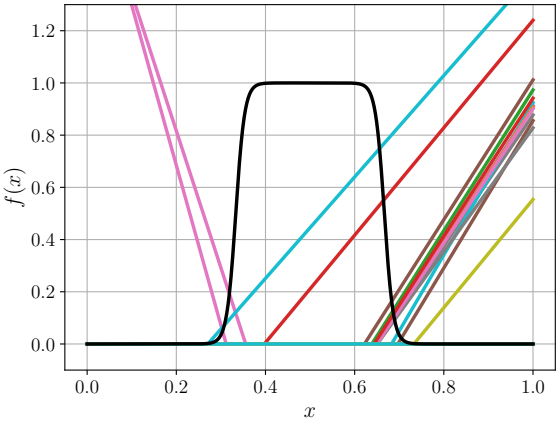
(c) $K_1 = 40$



(d) $K_1 = 60$



(e) $K_1 = 80$



(f) $K_1 = 200$

Figure 10: Visualization of the ReLU functions created by a network with $L = 1$ and different K_1

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). 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 $p = 10$ 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
4.8970e-05	1.3929e-04	2.6681e-04	4.7945e-03	6.9269e-03	1.1436e-02	3.2799e-02	6.2563e-02	9.3996e-02

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

We show in figure (11) the training and validation error, and in figure (12) the approximation on the test set for the best of the $P = 10$ models.

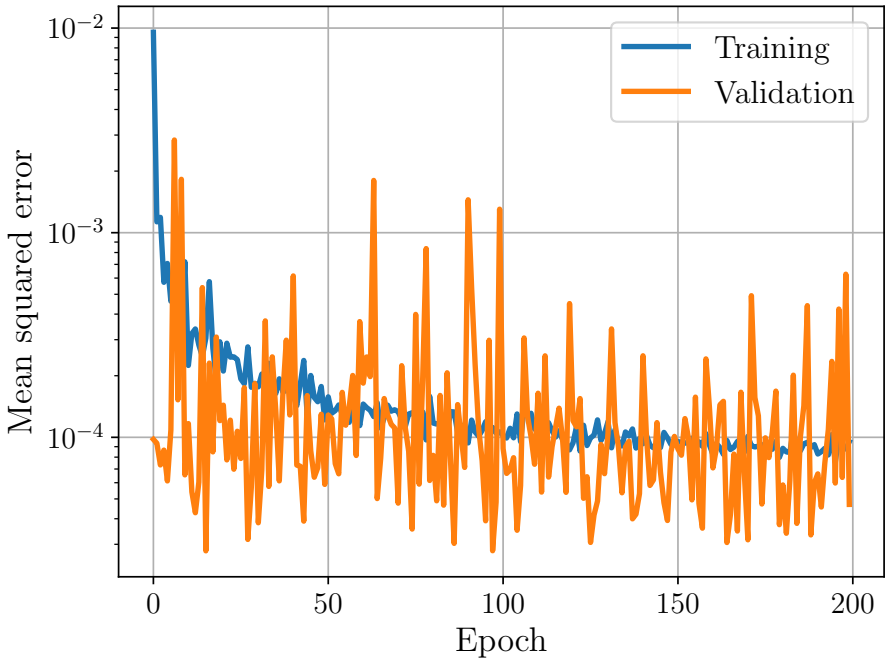
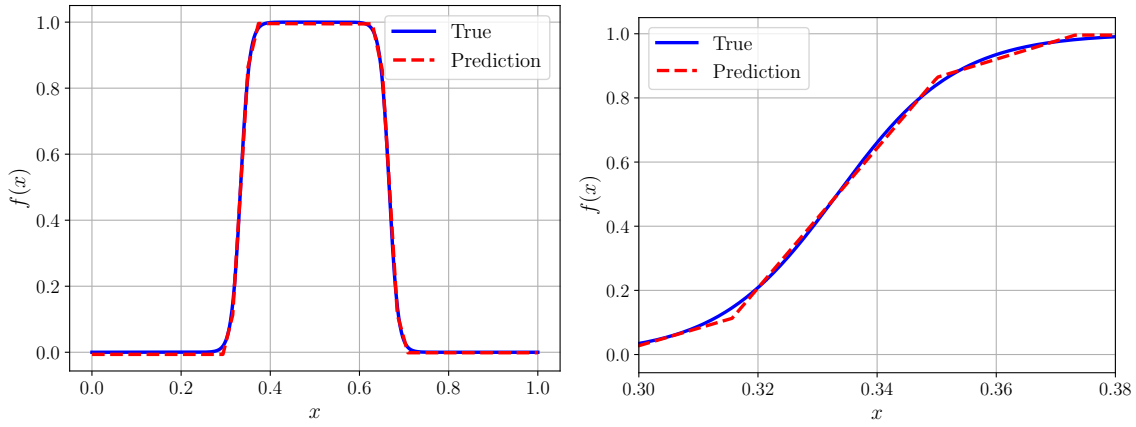


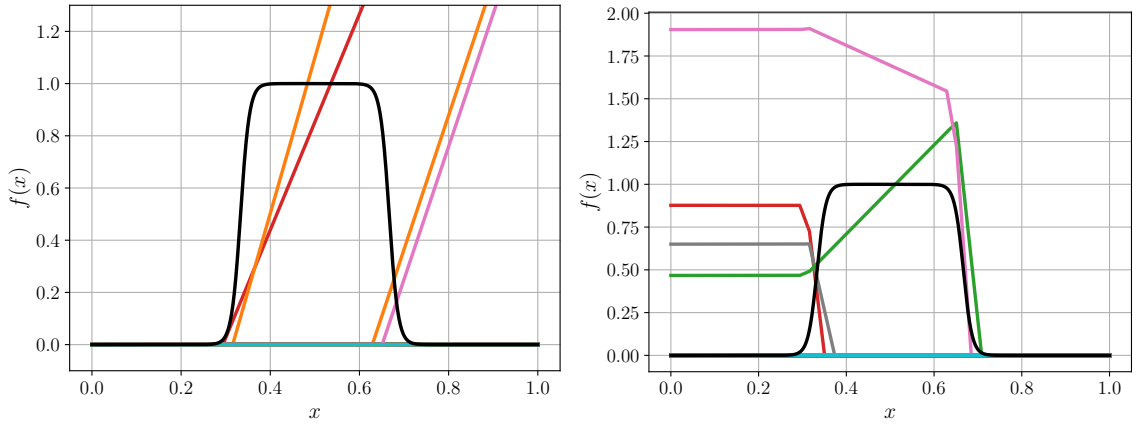
Figure 11: MSE during training for the best of the $P = 10$ models



(a) (b)

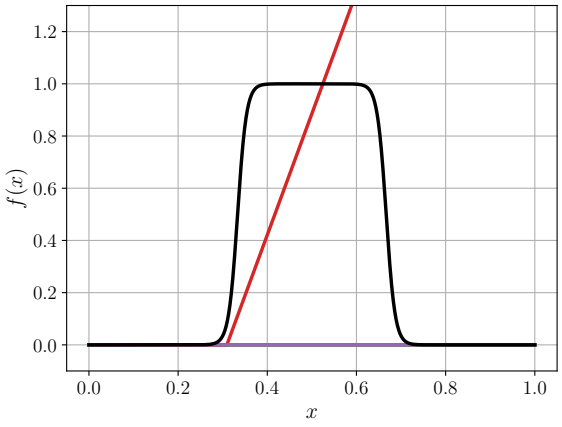
Figure 12: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 10$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition

Now we inspect the functions created in the neurons of the first and second layer. For the best of the $P = 10$ models, here are the results we got :

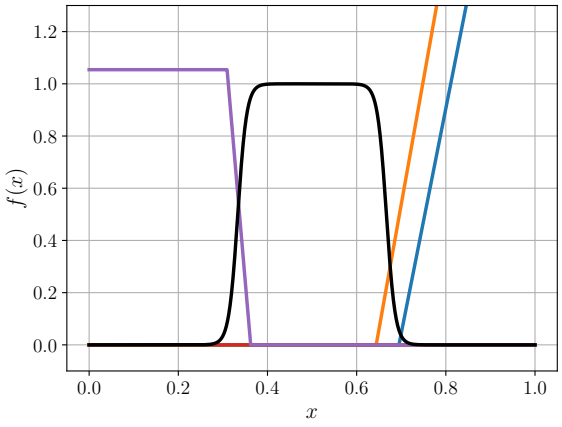


(a) First layer (b) Second layer

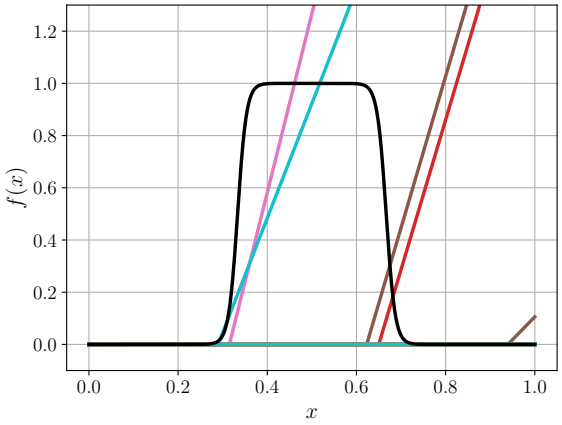
Figure 13: Functions created by the neurons of the model in the first and second layer



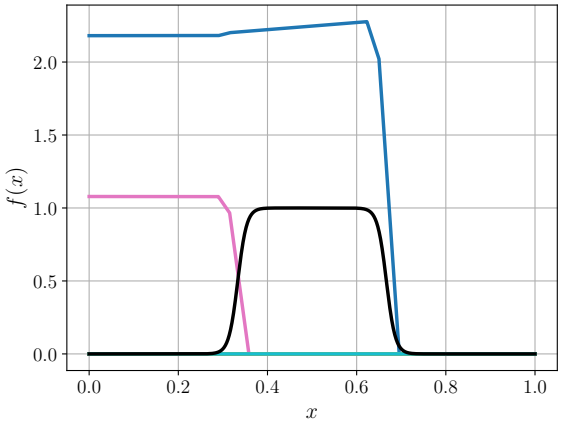
(a) $K = 5$ First layer



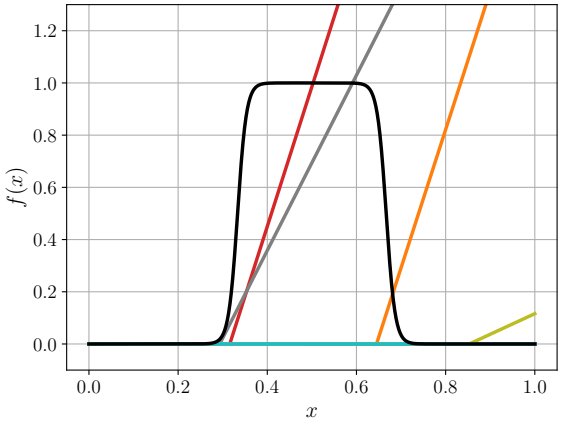
(b) $K = 5$ Second layer



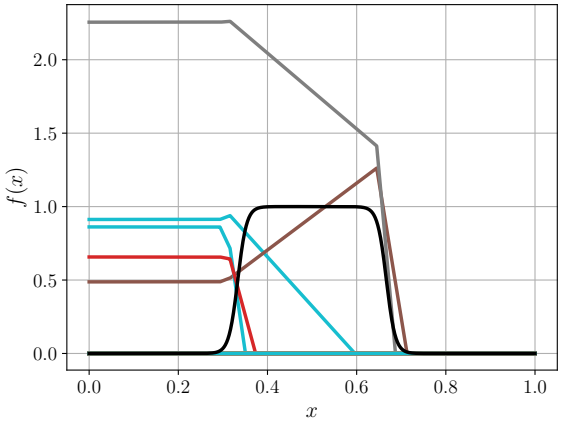
(c) $K = 20$ First layer



(d) $K = 20$ Second layer

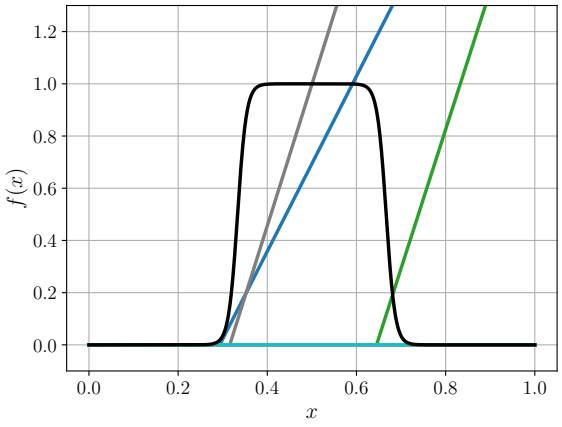


(e) $K = 40$ First layer

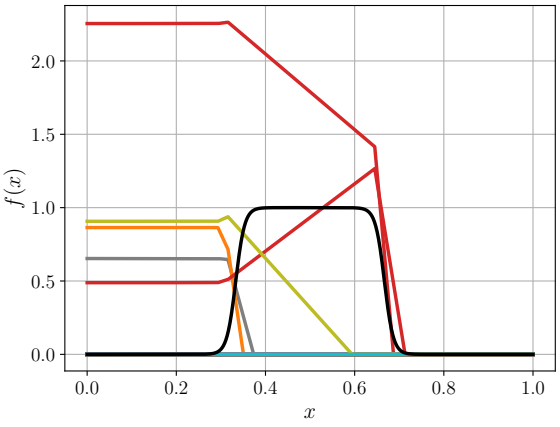


(f) $K = 40$ Second layer

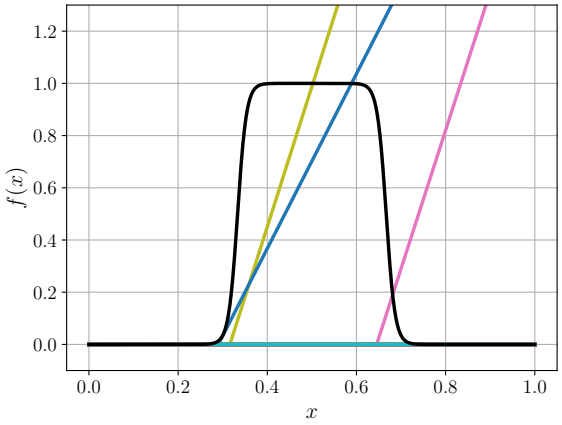
Figure 14: Visualization of the functions created by a network with $L = 2$ and different K



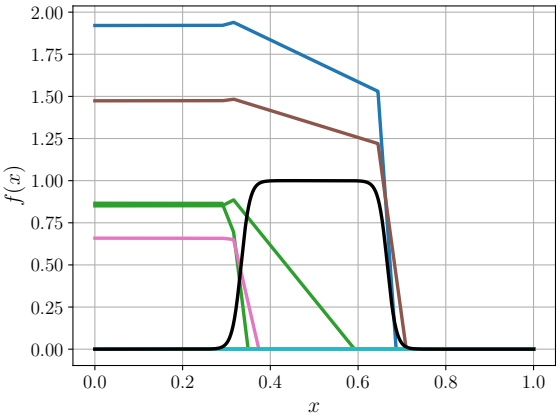
(a) $K = 60$ First layer



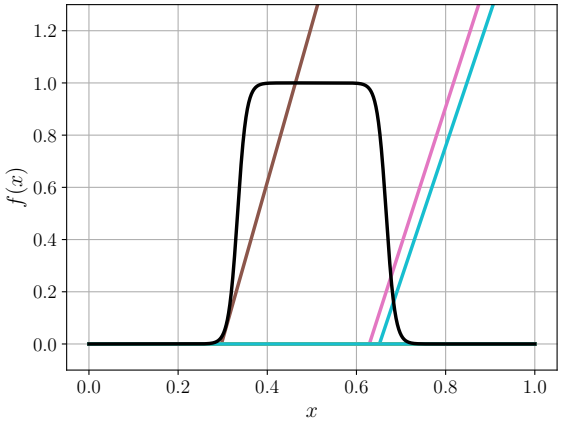
(b) $K = 60$ Second layer



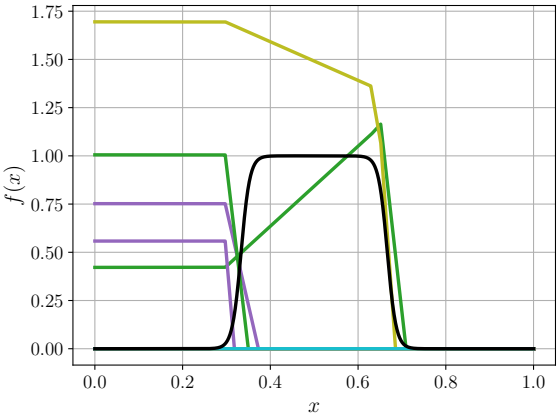
(c) $K = 80$ First layer



(d) $K = 80$ Second layer



(e) $K = 200$ First layer



(f) $K = 200$ Second layer

Figure 15: Visualization of the functions created by a network with $L = 2$ and different K

Figure (14) and (15) compare how the same network with different K creates functions in its neurons in both layers. Table (6) describes the metrics for each K.

	MSE	MAE	MaAE
$K = 5$	6.4176e-04	1.5650e-02	1.0800e-01
$K = 20$	3.5799e-04	9.4858e-03	1.0583e-01
$K = 40$	1.3155e-04	9.2261e-03	5.1318e-02
$K = 60$	1.3195e-04	9.2393e-03	5.1760e-02
$K = 80$	1.1874e-04	8.5283e-03	5.8692e-02
$K = 200$	5.7092e-05	5.0642e-03	3.9511e-02

Table 6: Metrics for the different networks

3.6 Convolutional network

The basis function (8) 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 convolutional 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 $P = 10$ identical networks, and report the metrics for the average, minimum and maximum of all P models :

MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
6.4968e-05	2.2963e-03	4.4144e-03	4.6618e-03	3.4383e-02	5.2243e-02	4.3348e-02	8.9357e-02	1.1669e-01

Table 7: Metrics for the convolutional network described above. The mean, min and max are applied over $P = 10$ networks trained in the same conditions

Figures (16) and (17) show the training and validation error, and the approximation on the test set for the best of the $P = 10$ models respectively.

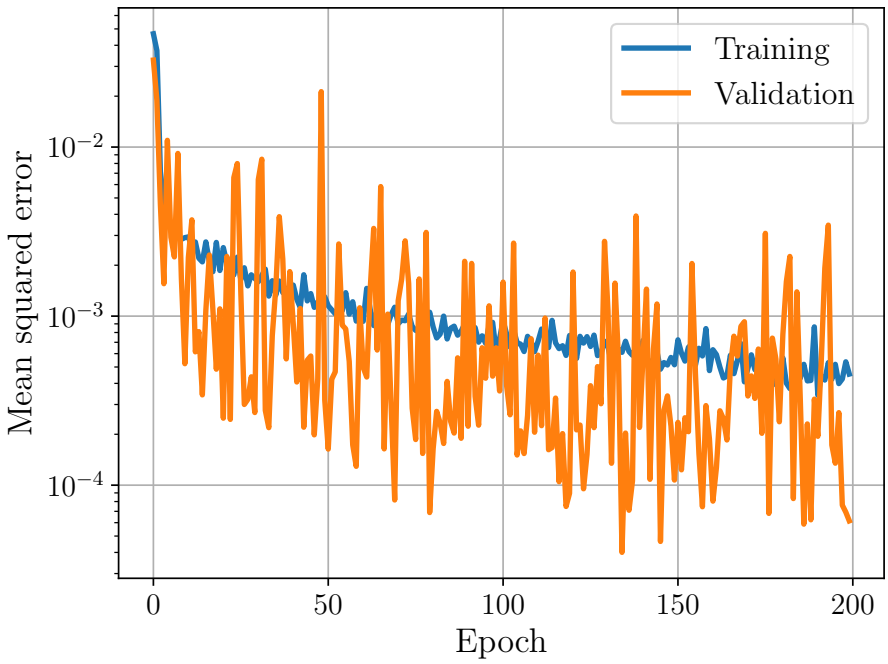
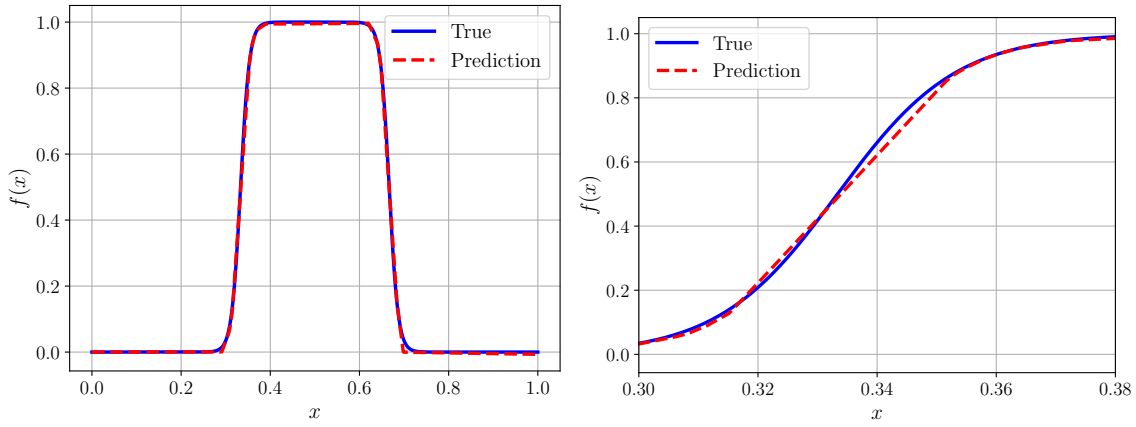
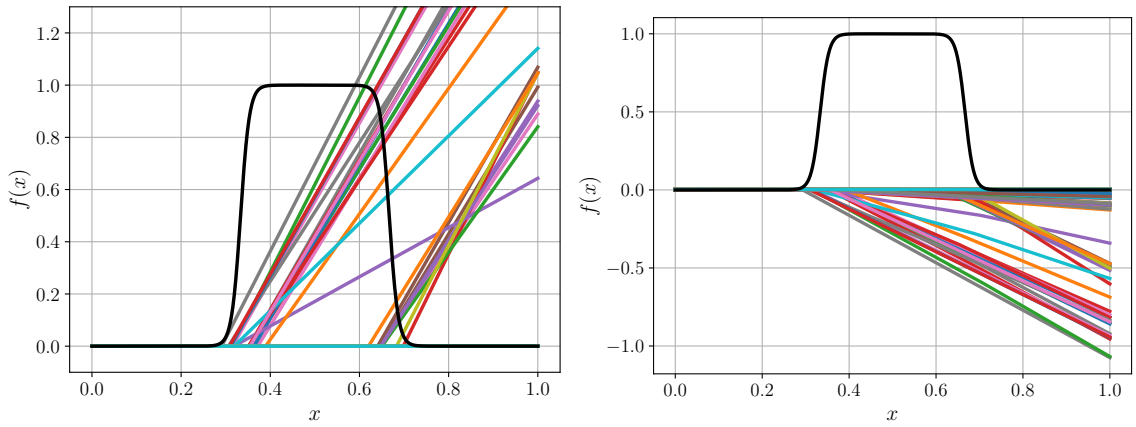


Figure 16: MSE during training for the best of the $P = 10$ models



(a) (b)

Figure 17: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 10$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition



(a) First layer

(b) Second layer

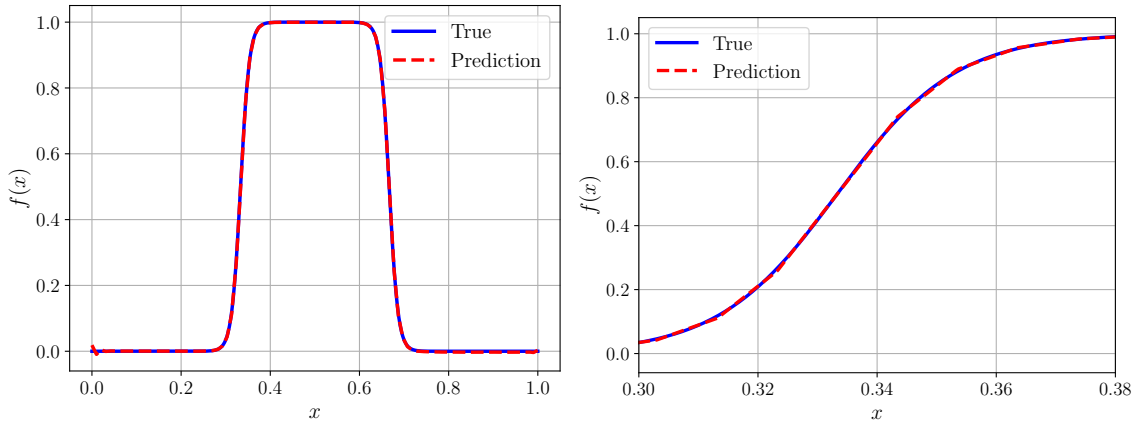
Figure 18: Functions created by the neurons of the model in the first (fully connected) layer and second (convolutional) layer for the best of the $P = 10$ models

Figure (18) shows what happens in the neurons of the best of the $P = 10$ models that we trained. This is nowhere near (2).

For this reason, we now try something new : we fix the weights of the first fully connected layer as in equation (9), 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 (8) from the nice properties of the convolutional layer.
Once again, we perform the same experiment $P = 10$ times, and report the mean, minimum and maximum in table (8),and the approximation and neurons functions for the best model in figures (19) and (20) respectively :

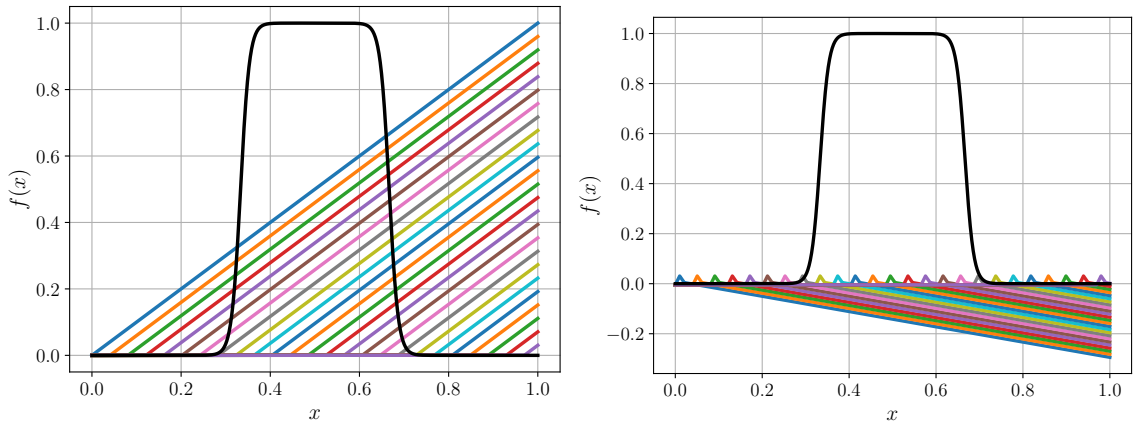
MSE			MAE			MaAE		
Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
4.0588e-06	4.1913e-05	1.4055e-04	1.3543e-03	4.2643e-03	8.8448e-03	1.1883e-02	1.6460e-02	2.4374e-02

Table 8: Metrics for the convolutional network described above where we do not train the first layer. The mean, min and max are applied over $P = 10$ networks trained in the same conditions



(a) (b)

Figure 19: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of the $P = 10$ models (red dotted-line) and (b) Zoom over the interval of the first sharp transition



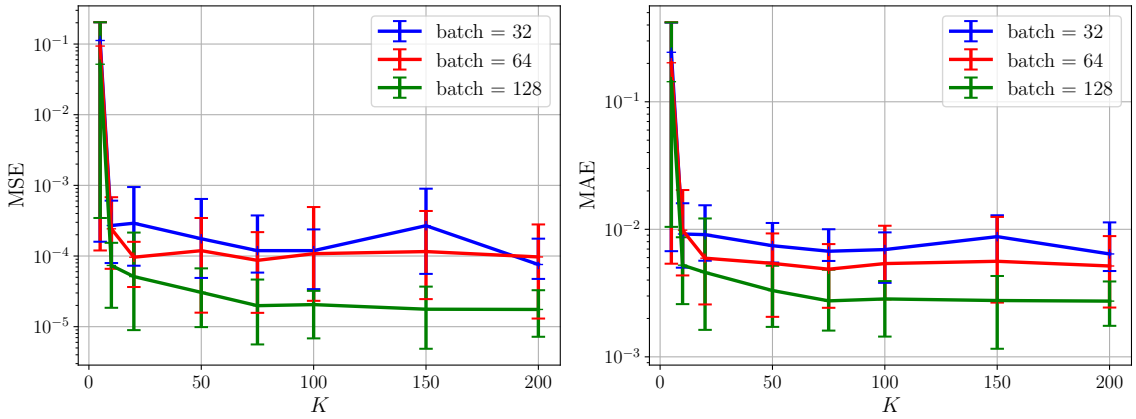
(a) First layer (b) Second layer

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 $P = 10$ models (we show only 1 out of 4 functions for readability)

As one can see in figure (20) 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 Width study

Now, we set the depth to $L = 2$ hidden layers, and wish to assess the impact of the number $K_1 = K_2 := K$ of neurons in both layers in the model performance. To account for the stochasticity, we always train $P = 10$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each parameter, we report the mean of the $P = 10$ models for a given metric, and represent as error bars the range $[\text{minimum}, \text{maximum}]$ of the $P = 10$ values for the given metric.



(a) MSE

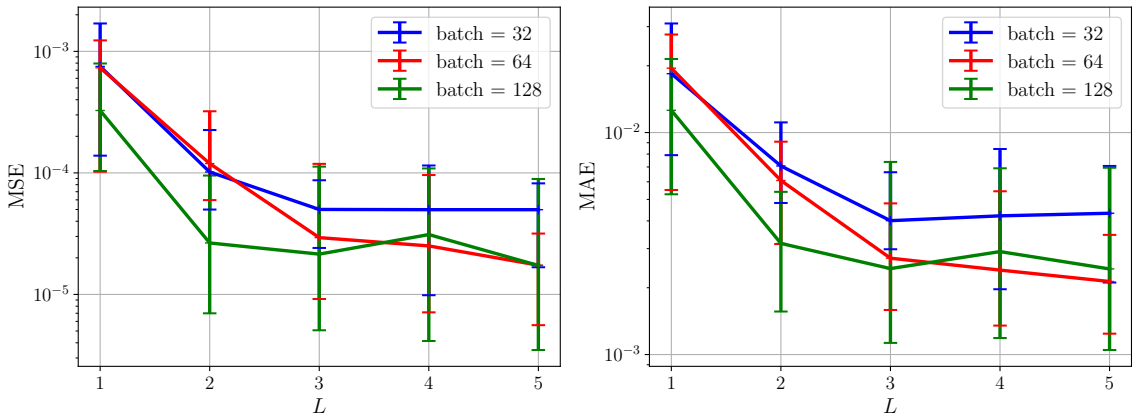
(b) MAE

(c) MaAE

Figure 21: Metrics in function of K for a network with $L = 2$ hidden layers for different batch sizes

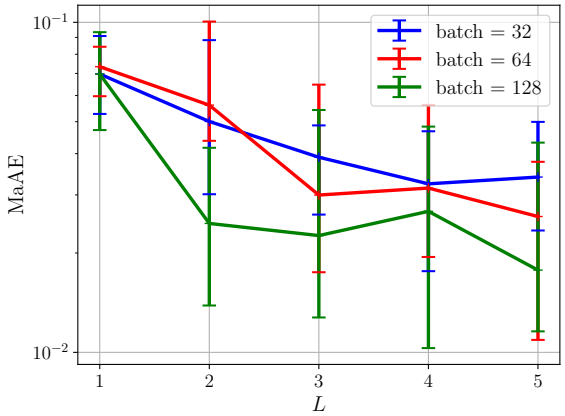
3.8 Depth study

In this part, we set the width to $K = 75$ 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 $P = 10$ different models with the same parameters. Moreover, we perform this study for different batch sizes. For each parameter, we report the mean of the $P = 10$ models for a given metric, and represent as error bars the range [minimum, maximum] of the $P = 10$ values for the given metric.



(a) MSE

(b) MAE



(c) MaAE

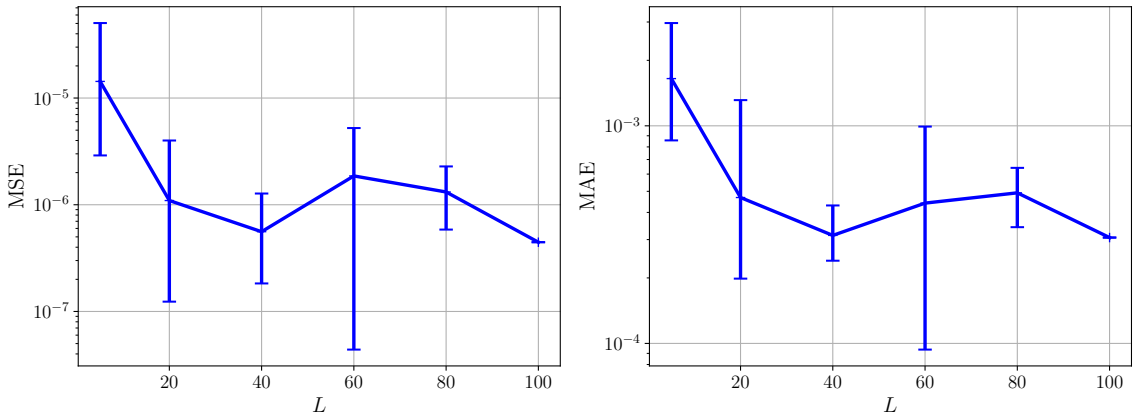
Figure 22: Metrics in function of L for a network with $K = 75$ neurons in each layer for different batch sizes

3.9 Deep depth study

According to [1], the approximation should be better for low K and large L. We wish to experiment with this result, and we perform a study of very deep networks. We set $K = 10$ and try different L up to $L = 100$. As always, we train $P = 10$ identical models to account for the stochasticity. However, for deep networks, some models are stuck from the beginning and are not able to converge towards an approximation (vanishing gradient due to the use of ReLU activations). For this reason, we have to drastically reduce the learning rate, from $\eta = 10^{-2}$ to $\eta = 10^{-5}$. However, this was not always sufficient, and some models never learned an acceptable solution. For this reason, we removed those models from the average, minimum and maximum. The exact metrics and number of models that correctly converged are summarized in table (9). For $L = 100$, only 1 out the $P = 10$ models was able to converge, thus the mean, minimum and maximum are identical. Note that the batch size is 32 here (we did not reproduce the same results for different batch sizes due to the computational time required).

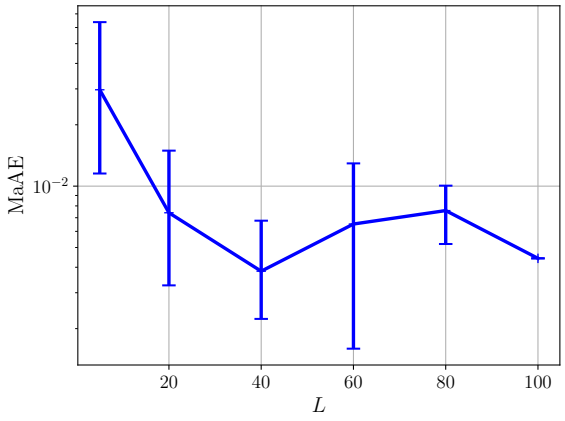
	Number of valid models	MSE			MAE			MaAE		
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
$L = 5$	10	2.8986e-06	1.4311e-05	5.0424e-05	8.5706e-04	1.6484e-03	2.9669e-03	1.1534e-02	2.9699e-02	6.3764e-02
$L = 20$	10	1.2359e-07	1.0951e-06	3.9995e-06	1.9849e-04	4.6823e-04	1.3126e-03	3.2601e-03	7.4027e-03	1.4933e-02
$L = 40$	3	1.8308e-07	5.5903e-07	1.2741e-06	2.3994e-04	3.1362e-04	4.3056e-04	2.2320e-03	3.8257e-03	6.7692e-03
$L = 60$	3	4.3922e-08	1.8623e-06	5.2328e-06	9.3670e-05	4.4136e-04	9.9213e-04	1.5947e-03	6.5184e-03	1.2931e-02
$L = 80$	4	5.8521e-07	1.3173e-06	2.2862e-06	3.4216e-04	4.9207e-04	6.4089e-04	5.2024e-03	7.5875e-03	1.0055e-02
$L = 100$	1	4.4561e-07	4.4561e-07	4.4561e-07	3.0656e-04	3.0656e-04	3.0656e-04	4.4238e-03	4.4238e-03	4.4238e-03

Table 9: Number of model that did converge towards a solution for each L, and metrics taken on those models that did converge. $K = 10$ is fixed for all L, and the learning rate is $\eta = 10^{-5}$.



(a) MSE

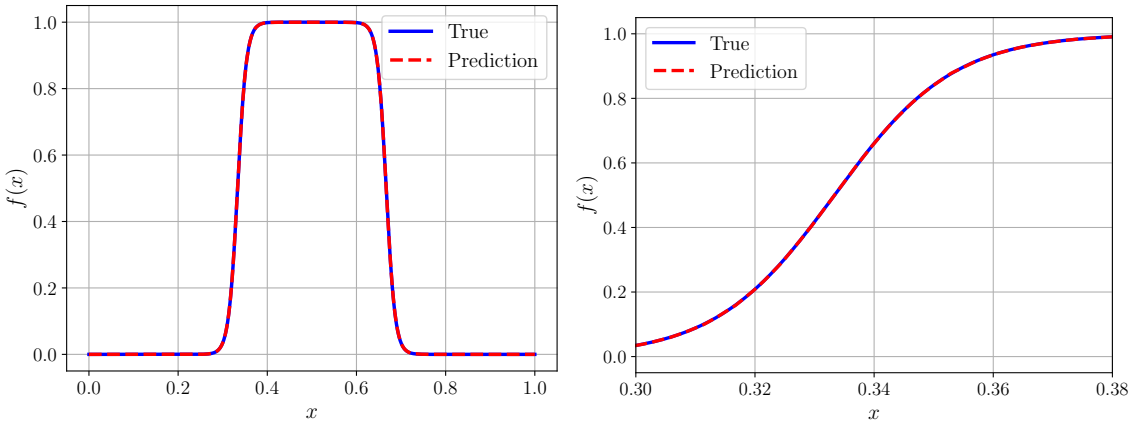
(b) MAE



(c) MaAE

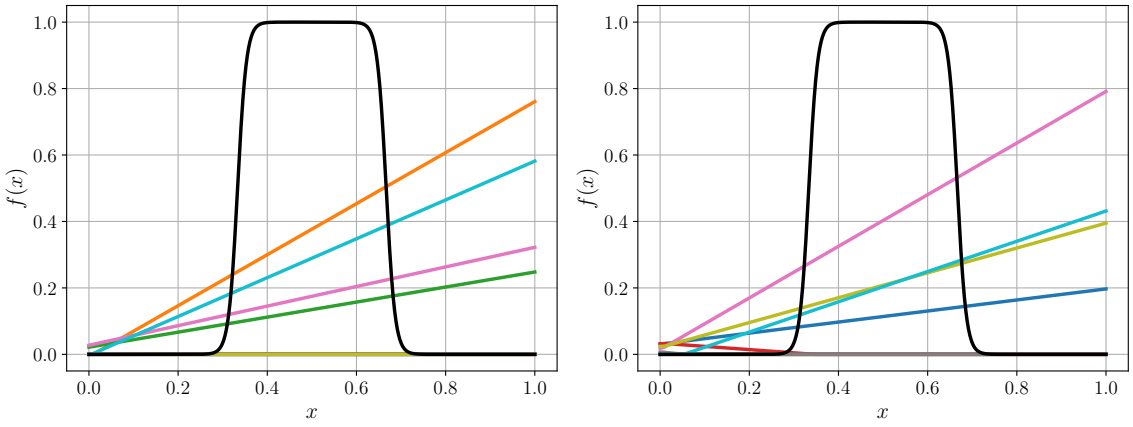
Figure 23: Metrics in function of L for a network with $K = 10$ neurons in each layer

The best of these neural network is a model with $L = 60$, which achieved an MSE of 4.3922e-08, MAE of 9.3670e-05 and MaAE of 1.5947e-03 on the testing set. We selected this model, and we show on figure (24) the approximation on the testing set, and on figure (25) and (26) the constructions of the functions in all of the $K = 10$ neurons for different layers.



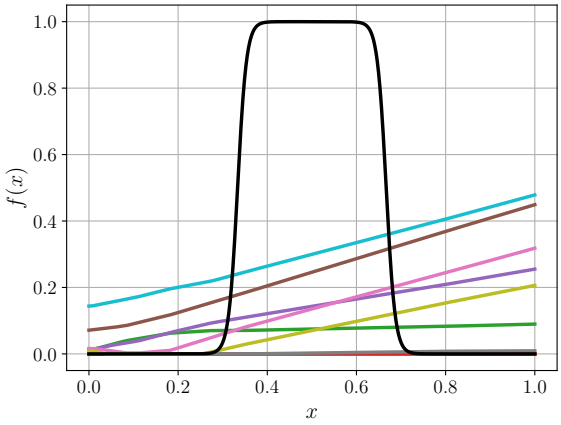
(a) (b)

Figure 24: (a) Comparison between the ground-truth (blue line) and the approximation given by the best of model (red dotted-line) and (b) Zoom over the interval of the first sharp transition

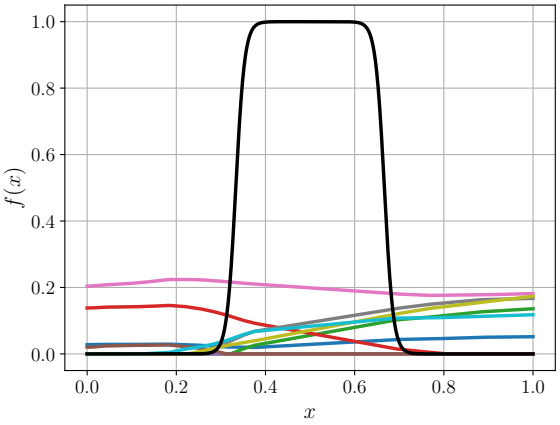


(a) 1st layer

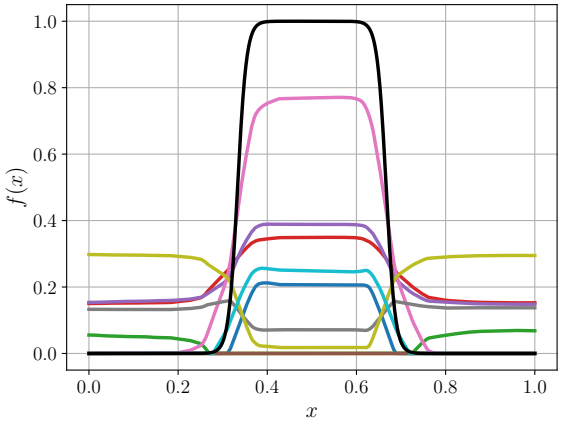
(b) 2nd layer



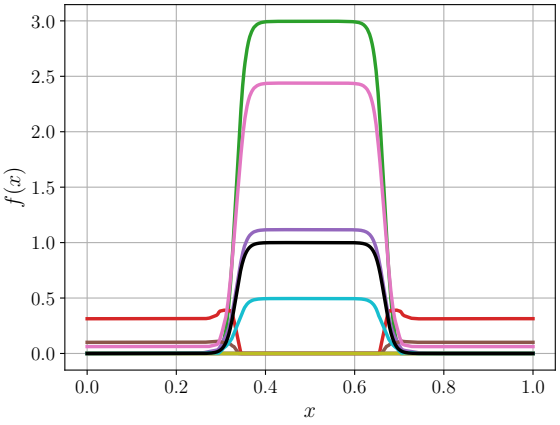
(c) 5th layer



(d) 10th layer

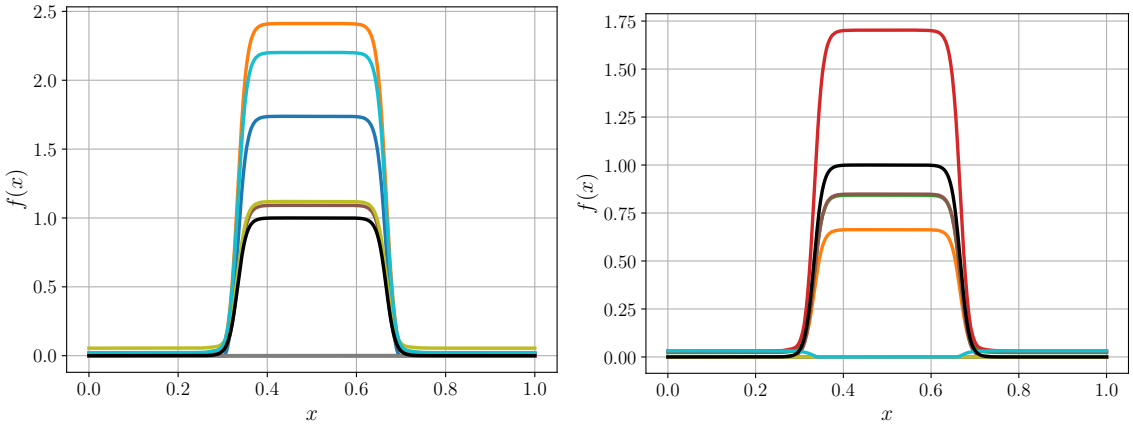


(e) 20th layer



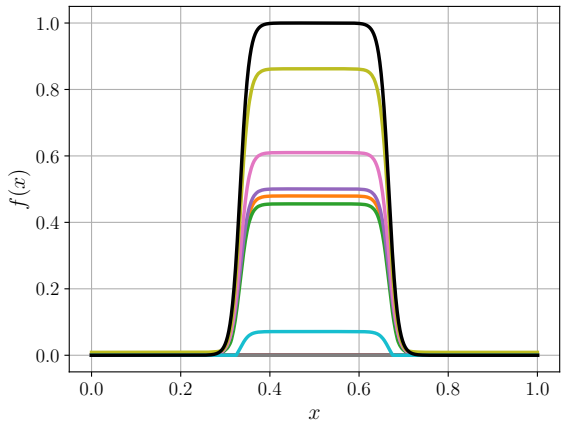
(f) 30th layer

Figure 25: Visualization of the functions created by the best of the networks. As always, the objective function is in black.



(a) 40th layer

(b) 50th layer



(c) 60th layer

Figure 26: Visualization of the functions created by the best of the networks. As always, the objective function is in black.

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