The GroupMax neural network approximation of convex functions.

Xavier WARIN [*]

January 27, 2023

Abstract

We present a new neural network to approximate convex functions. This network has the particularity to approximate the function with cuts which is, for example, a necessary feature to approximate Bellman values when solving linear stochastic optimization problems. The network can be easily adapted to partial convexity. We give an universal approximation theorem in the full convex case and give many numerical results proving it efficiency. The network is competitive with the most efficient convexity preserving neural networks and can be used to approximate functions in high dimension.

1 Introduction

Neural networks are an effective tools to approximate function and numerically generally outperform classical regression using an expansion on a function basis. Some classical Universal Approximation theorem for neural network with bounded depth are given in [1, 2] when the activation function is non polynomial. A "dual" result is given in [3] where the number of the hidden layers can be taken arbitrary large with bounded widths if the activation function is non affine, continuous and twice continuously differentiable. Recently, [4] slightly improves the results with only non affine, continuous and continuously differentiable activation functions.

The approximation of a convex function has recently been theoretically investigated for example in [5] for a one-layer feedforward neural network with exponential activation functions in the inner layer and a logarithmic activation at the output. Numerically this problem has been investigated in [6] developing the Input Convex Neural Network (ICNN) methodology. This approach is effective and has been widely used in many applications where the convexity of the approximation is required, for example in optimal transport problems [7,8], in optimal control problems as in [9,10], in inverse problems [11], or in general optimization problems [12] just to quote some of them.

In some cases, a simple convex approximation of the function in not sufficient. For example, in multi stage stochastic linear optimization, some methods such as the Stochastic Dual Dynamic Programming method (SDDP) [13] solve transition problems starting from a state using an approximation by cuts of the Bellman function at the end of the transition period. This transition problem is solved using a Linear Programming solver. The cut approximation of a convex function using regression methods has already been investigated in [14,15] or [16] leading to some max affine approximation.

This article proposes a different approach using a new network permitting to efficiently approximate a convex function by cuts using some ideas similar to the GroupSort network developed in [17] and analyzed in [18].

In a first part, we present the network supposing that the function to approximate is convex with respect to the input. An universal approximation theorem is then given for this full convex case. Then, supposing that the function is only convex with respect to a part of the input, we extend our representation using conditional cuts. In a second part some numerical examples are given showing that the network is clearly superior to a network generating simple cuts and is competitive with the ICNN method. At last a conclusion is given.

---

[*]EDF R&D & FiME xavier.warin at edf.fr

2 The GroupMax Network

Let f be a real valued convex function defined on $\mathbb{R}^d$. We have the following representation [19, Chapter 3.2.3]: Define $\tilde{f}$ as:

$$\tilde{f}(x) = \sup\{g(x)\,|\,g(x) \text{ affine}, g(z) \leq f(z)\} \tag{1}$$

then $\tilde{f}(x) = f(x)$ for $x \in \text{int Dom}(f)$. In the sequel an approximation by cuts of a convex function will refer to a max-affine affine representation (1). A cut is one of the affine functions in this max-affine representation.

From equation (1), we may think of a first single layer network h:

$$h^\theta(x) = \max_{i=1}^{N} A_i.x + b_i \tag{2}$$

where $A_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$, and $\theta = (A_i)_{i=1,N} \cup (B_i)_{i=1,N}$ is the set of parameters to optimize. In this single layer network, N is the number of neurons and the max function is not applied componentwise but on the global vector. Then trying to approximate the convex function f on $\mathcal{D}$ we solve

$$\theta^* = \operatorname*{argmin}_{\theta} \mathbb{E}((f(X) - h^\theta(X))^2) \tag{3}$$

where X is a random variable for example uniformly distributed in $\mathcal{D}$. As we will see later, this first network is too simple and leads to bad approximations. We then develop a new network generating cuts with a given number of layers.

2.1 A Network for Fully Convex Functions

2.1.1 The Network with q Layers

We impose to simplify that M, the number of neurons, is kept constant for all layers and we define the group size G as in [17] such that $K = \frac{M}{G}$ is an integer corresponding to the number of groups. For $x \in \mathbb{R}^d$, the network is defined by recurrence as

$$\begin{aligned}
z^1 &= \rho(A^1 x + B^1) \\
z^i &= \rho((A^i)^+ z^{i-1} + B^i) 1 < i < q \\
h^\theta(x) &= \hat{\rho}((A^q)^+ z^{q-1} + B^q)
\end{aligned} \tag{4}$$

where q is the number of layers, $y^+ = \max(y, 0)$, $A^1 \in \mathbb{R}^{M,d}$, $B^1 \in \mathbb{R}^M$, $A^j \in \mathbb{R}^{M \times K}$, $B^j \in \mathbb{R}^M$, $j = 2, \ldots, q$, defining the set of parameters as $\theta = (A^j)_{j=1,q} \cup (B^j)_{j=1,q}$.

The activation function ρ and $\tilde{\rho}$ are defined as follows:

- $\hat{\rho}$ is a $\mathbb{R}$ valued function defined on $\mathbb{R}^M$ where

$$\hat{\rho}(x) = \max(x_1, \ldots, x_M) \text{ for } x \in \mathbb{R}^M$$

- ρ is a function from $\mathbb{R}^M$ in $\mathbb{R}^K$ such that

$$\rho(x)_i = \max(x_{1+(i-1)G}, \ldots, x_{iG}) \text{ for } i = 1, \ldots, K$$

and an example of the structure of the network given in Fig. 1.

This network gives an approximation of f by some cuts: clearly by positivity of the $(A_i)^+$, we get that

$$\begin{aligned}
h^\theta(x) = \max_{i_q \in [1,M]} \quad &\max_{\substack{i_j \in [1, G], \\ j = 1, \ldots, q-1}} \quad \sum_{\substack{k_j = 1 \\ j = 1, q-1}}^{K} (A^q)^+_{i_q, k_{q-1}} \\
&\prod_{n=1}^{q-2} (A^{n+1})^+_{i_{n+1}+(k_{n+1}-1)G, k_n} \sum_{m=1}^{d} A^1_{i_1+(k_1-1)G, m} x_m \\
&+ C
\end{aligned} \tag{5}$$

where C is function of the B^i, $i = 1, \ldots, q$ and A^i, $i = 1, \ldots, q-1$. Then the number of cuts can be calculated as $MG^{K(q-1)}$.
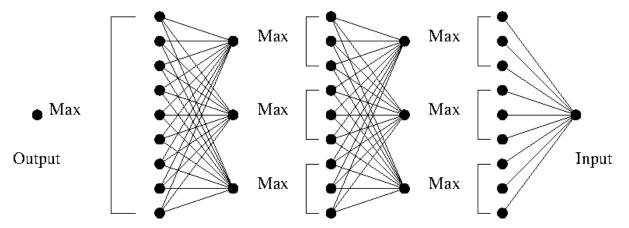
Figure 1: Network structure in dimension 1, group size $G = 3$, $K = 3$ groups and 3 layers of $M = 9$ neurons.

Remark 1. *Using M neurons on the first layer and $\tilde{M} \geq \frac{M}{G}$ neurons on the following layers, we notice that if we take $B^j = 0$ for $1 < j \leq q$, and A^j null except $(A^j)_{i,i} = 1$, for $i = 1, \ldots, \frac{M}{G}$ for $1 < j \leq q$, then the cuts generated by the one layer network are the same as the cuts generated by the $q > 1$ layers network. Then cuts generated by the network (2) can be reached with the network (4).*

Remark 2. *Using equation (5), it is possible to reconstruct the underlying cuts which is, for example, necessary to solve stochastic linear optimization problems using Benders cuts.*

2.1.2 Universal Approximation Theorem

We denote $\mathcal{N}(M_1, \ldots, M_q, K)$ the set of generated functions $h^{\theta_{M_1, \ldots, M_q, K}}$ by the network (4) with $\theta_{M_1, \ldots, M_q, K} \in \mathbb{R}^{M_1(d+1) + \sum_{i=2}^{q} M_i(K+1)}$ where the number of neurons for layer q is M_q. We introduce the space of functions generated letting the number of neurons vary:

$$\mathcal{N}(K, q) = \cup_{(M_1, \ldots, M_q) \in \mathbb{N}^q} \mathcal{N}(M_1, \ldots, M_q, K)$$

Proposition 1. *Let f be a convex function on $\mathbb{R}^d$, then $\mathcal{N}(K, q)$ approximates arbitrarily well f by below on every compact $\hat{K}$ for the sup norm.*

Proof. Due to remark 1, it is sufficient to prove that for a given function f, for each ϵ, there exists a finite number of cuts $(A_i, B_i)_{i=1,M}$ such that $g(x) = \max_{i=1,M} A_i x + B_i$ and such that $g(x) \leq f(x)$ and $\sup_{x \in \hat{K}} f(x) - g(x) \leq \epsilon$.

Let suppose the converse. There exist ϵ, such that for n_0 being chosen arbitrary, and whatever the cuts we take generating a function g^0 below f, then there exists x_0 such that $f(x_0) - g^0(x_0) > \epsilon$. Using [20, Proposition A], we can generate a cut (A, B) such that $f(x_0) = Ax_0 + b$ and $f(x) \geq Ax + B$ on $\hat{K}$. Let use define, $g^1(x) = max(g^0(x), Ax + B)$. Due to the hypothesis, there there exist x_1 such that $f(x_1) - g^1(x_1) > \epsilon$ and we can build a sequence (x_i, g^i), $i \geq 0$ such that g^i is below f and $f(x_i) - g^j(x_i) \geq \epsilon$ for all $j \leq i$ and $f(x_i) = g^j(x_i)$ for $j > i$.

We can extract a sequence $\tilde{x}^i$ from $(x_i)_{i \geq 0}$ that converges to $\tilde{x} \in \hat{K}$. As f is convex on $\mathbb{R}^d$ it is continuous on $\hat{K}$. Then letting i go to infinity in $f(\tilde{x}_i) - g^j(\tilde{x}_i) \geq \epsilon$ for j fixed, and using g^j continuity we get $f(\tilde{x}) - g^j(\tilde{x}) \geq \epsilon$ for all j. Defining $g = \sup_{i > 0} g_i$ which is convex so continuous, we get that $f(\tilde{x}) - g(\tilde{x}) \geq \epsilon$.

Starting from $f(x_i) = g^j(x_i)$ for $j > i$ and first letting j to infinity, we get $f(x_i) = g(x_i)$. Letting i go to infinity and using the continuity of g and f we get the contradiction. $\square$

2.2 An Extension for Partial Convex Functions

We suppose that $x = (\tilde{x}, y) \in \mathbb{R}_d$ where we have convexity in $y \in \mathbb{R}^k$, We simply modify our network as done in [6] to take into account the non convexity in $\tilde{x}$: other networks tested could not outperform the

one proposed in [6]. The recursion is given by:

$$
\begin{aligned}
u_0 =& \tilde{x}, \quad z_0 = 0 \\
u_{i+1} =& \tilde{\rho}(\tilde{W}_{i+1}u_i + \tilde{b}_{i+1}) \\
z_{i+1} =& \rho([W_{i+1}^{(z)} \otimes (W_{i+1}^{(zu)}u_i + b_{i+1}^{(z)})]^+ z_i + \\
& W_{i+1}^{(y)}(y \circ (W_{i+1}^{(yu)}u_i + b_{i+1}^{(y)})) + W_{i+1}^{(u)}u_i + b_{i+1}) \\
& \text{for } i < q - 1 \\
h^\theta(\tilde{x}, y) =& \hat{\rho}([W_q^{(z)} \otimes (W_q^{(zu)}u_{q-1} + b_q^{(z)})]^+ z_q + \\
& W_q^{(y)}(y \circ (W_q^{(yu)}u_{q-1} + b_q^{(y)})) + W_q^{(u)}u_{q-1} + b_q)
\end{aligned}
\tag{6}
$$

where $\circ$ denote the Hadamard product, $\otimes$ is applied between a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $B \in \mathbb{R}^n$ such that $A \otimes B \in \mathbb{R}^{m \times n}$ and $(A \otimes B)_{i,j} = A_{i,j}B_j$.

In recursion (6), $\tilde{\rho}$ is a classical activation function such as the ReLU function. Using m_x neurons for the non convex part of the function and m_y neural networks for the convex part, $\tilde{W}_0 \in \mathbb{R}^{m_x \times (d-k)}$, $\tilde{W}_i \in \mathbb{R}^{m_x \times m_x}$ for $i > 0$, $W_i^{(zu)} \in \mathbb{R}^{m_y \times m_x}$ for $i > 0$, $W_i^{(z)} \in \mathbb{R}^{m_y \times m_y}$ for $i \leq q$. We do not detail the size of the different matrices $W^{(y)}$, $W^{(yu)}$, $W^{(u)}$ and the different biases that are obvious. θ is the set of all these weights and bias. Introducing

$$
\begin{aligned}
A^i(\tilde{x}) =& [W_i^{(z)} \otimes (W_i^{(zu)}u_{i-1} + b_i^{(z)})]^+ \\
\tilde{A}^i(\tilde{x}) =& W_i^{(y)} \otimes (W_i^{(yu)}u_{i-1} + b_i^{(y)}) \\
B^i(\tilde{x}) =& W_i^{(u)}u_{i-1} + b_i
\end{aligned}
$$

where we can note by recurrence that u_i is a non linear function of $\tilde{x}$, equation (6) can be rewritten as

$$
\begin{aligned}
(z_{i+1})_j =& \max_{l \in [1,G]} [A^{i+1}(\tilde{x})z_i + \tilde{A}^{i+1}(\tilde{x})y + B^{i+1}(\tilde{x})]_{(j-1)G+l} \\
& \text{for } j = 1, \ldots, K, \text{ and } i < q - 1 \\
h^\theta(\tilde{x}, y) =& \max_{l \in [1,m_y]} [A^q(\tilde{x})z_{q-1} + \tilde{A}^q(\tilde{x})y + B^q(\tilde{x})]_l
\end{aligned}
$$

Similarly to section 2.1 ,

$$
h^\theta(\tilde{x}, y) = \max_{l=1,\ldots,m_y G^{K(q-1)}} [A(\tilde{x})y + B(\tilde{x})]
$$

where $A(\tilde{x}) \in \mathbb{R}^{m_y G^{K(q-1)} \times k}$ is a function of the $(A^i(\tilde{x}))_{i=1,\ldots,q}$ and $(\tilde{A}^i(\tilde{x}))_{i=1,\ldots,q}$ matrices. Similarly $B(\tilde{x}) \in \mathbb{R}^{m_y G^{K(q-1)}}$ is a function of the $(B^i(\tilde{x}))_{i=1,\ldots,q}$, $(A^i(\tilde{x}))_{i=1,\ldots,q-1}$ and $(\tilde{A}^i(\tilde{x}))_{i=1,\ldots,q-1}$. Then it is clear that this recursion permits to define some cuts conditional to $\tilde{x}$.

3 Numerical Results

In all numerical results, we use tensorflow [21] with an ADAM gradient descent algorithm [22].

3.1 Some One Dimensional Results

We consider the convex function $f(x) = f_i(x)$ for case $i = 1, \ldots, 5$ where

1. $f_1(x) = x^2$

2. $f_2(x) = x^2 + 10[(e^x - 1)1_{x<0} + x1_{x \geq 0}]$

3. $f_3(x) = (|x|^2 + 1)^2$

4. $f_4(x) = |x|1_{|x| \leq 3} + \frac{x^2 - 3}{2}$

We regress $f(x) + \epsilon$ with respect to x where $\epsilon \sim \mathcal{N}(0, 1)$ using 20000 gradient iterations, a batch size equal to 300 and a learning rate equal to $1e - 3$. We then solve equation (3) using $X \sim \mathcal{N}(0, 4)$. On Fig. 2, we see that the simple approximation (2) gives visually not very good results and that the solution does not
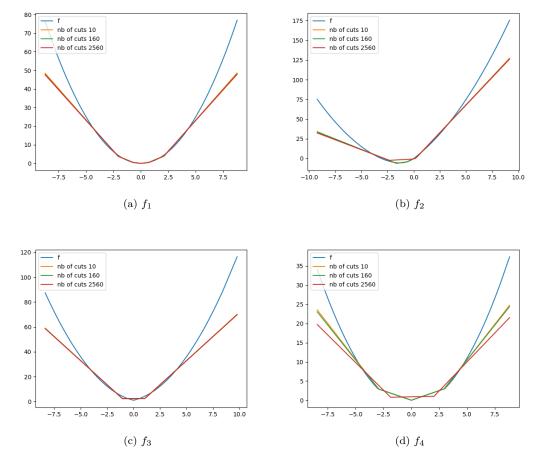
(a) f_1

(b) f_2

(c) f_3

(d) f_4

Figure 2: Solution of equation (3) using network (2) with different number of cuts.

Table 1: MSE for the Different Networks.

Network	f_1	f_2	f_3	f_4
Feedfoward	0.0004	0.0031	0.0004	0.0001
[6] network	0.0013	0.0050	0.0019	0.0006
GroupMax network	0.0012	0.0014	0.0029	0.0002

improve as we increase the number of cuts. It was however possible to get some rather accurate solutions except in the tails using network (2) by renormalizing **both the input x and the output $f(x) + \epsilon$** such that both are centered with a unit standard deviation. Results are given on Fig. 3. On the Fig. 4 we plot the solution obtained using the GroupMax network, [6] network and the feedforward network. For the GroupMax we use 3 layers with 10 neurons on each layer and the group size $G = 5$. As for the two other networks, we use 3 hidden layers with 10 neurons and the ReLU activation function. 50000 gradient descent iterations are used. The cuts generated on the previous examples by the GroupMax network are given on Fig. 5. To see the accuracy of the GroupMax network as an interpolator, we optimize equation (3) trying to fit directly the function f without any noise. Table 1 gives the best MSE on 10 test runs using Monte Carlo (1e6 samples). The solution obtained by the feedforward seems to be more accurate with the chosen parameters. Between [6], and the GroupMax it is hard to say which is the best. In table 2, we give the results obtained (best of 10 runs) for different values of parameter K using 12 neurons per layer. As before the best of 10 runs is kept. The results seem to indicate that the group size should remain rather low. At last, the influence of the number of layers q is given in table 3 taking 12 neurons per layer and a group size of 2 in the GroupMax network. The best of 10 runs is given. The accuracy clearly improves as we increase the number of layers.

(a) f_1 (b) f_2
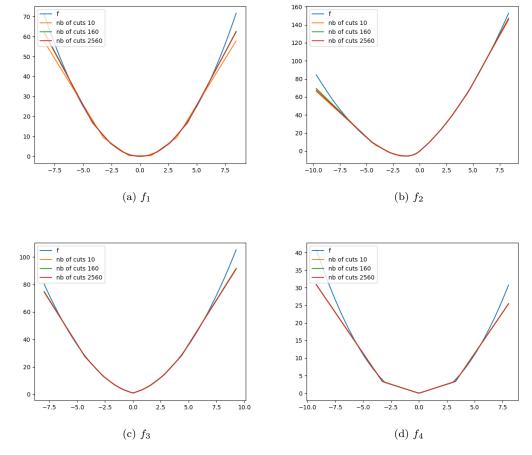
(c) f_3 (d) f_4

Figure 3: Solution of equation (3) using network (2) with different number of cuts using some input and output renormalization.

Table 2: Influence of the Group Size on the MSE.

K	f_1	f_2	f_3	f_4
2	0.0010	0.0015	0.0019	0.00028
4	0.0007	0.0012	0.0014	0.00023
6	0.0010	0.0018	0.0032	0.00038
12	0.00071	0.0071	0.0137	0.00085

3.2 Testing Partial Convexity in 2D

We suppose that we want to interpolate a function which is convex only in its second dimension and test the error obtained in solving (3) in two cases. In the first case, we suppose that $X \sim \mathcal{N}(0,1)^2$, then that $X \sim \mathbf{U}([-2,2]^2)$. We test the following functions convex in y:

1. $f_5(x,y) = y^2|x + 2x^3|$

2. $f_6(x,y) = (1 + |y|)|x + 2x^3|$

3. $f_7(x,y) = y^+|x| + x^2$

We keep the same parameters as before for the different networks. For the ICNN and the GroupMax network we take the same number of neurons both for the convex and non convex part. We first keep a learning rate equal to 10^{-3} and a batch size equal to 300. We take 50000 gradient iterations. Results are given in table 4 sampling $X \sim \mathcal{N}(0,1)^2$ and 5 with $X \sim \mathbf{U}([-2,2]^2)$. Best of 10 runs are given. Surprisingly, the feedforward network behaves very badly especially sampling a Gaussian law.

Remark 3. *It was possible to get better results for the f_6 function using the feedforward network using up to 80 neurons or increasing the number of layers. The results obtained were not as good as the ones obtained by the other networks.*

(a) f_1 (b) f_2
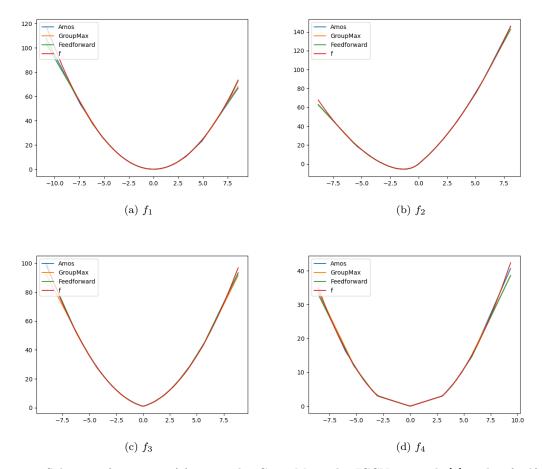
(c) f_3 (d) f_4

Figure 4: Solution of equation (3) using the GroupMax, the ICCN network [6] and a feedforward network.

Table 3: Influence of the Number of Layers q on the MSE.

q	f_1	f_2	f_3	f_4
2	0.0073	0.0054	0.062	6.7e-4
3	0.0017	0.0010	0.021	5.9e-4
4	7.5e-4	0.0011	0.001	9.5e-5
5	2.7e-4	7e-4	4e-4	3.5e-5

We test the GroupMax network on the two cases with 12 neurons on each layer, with a group size K equal to 3, with different numbers of layers q on table 6 7. The best of 10 runs is given. Sampling an uniform law, the error clearly decreases with the number of layers while sampling a gaussian law this decrease is not observed.

3.3 Convexity in Higher Dimension

We try to interpolate here a function in higher dimension. We take two cases:

1. First, it is the square of the L_2 norm.

$$f_8(x) = ||x||^2$$

2. For the second we generate randomly A a positive definite matrix in dimension d and take

$$f_9(x) = \sum_{i=1}^{d} (|x_i| + |1 - x_i|) + x^\intercal A x$$
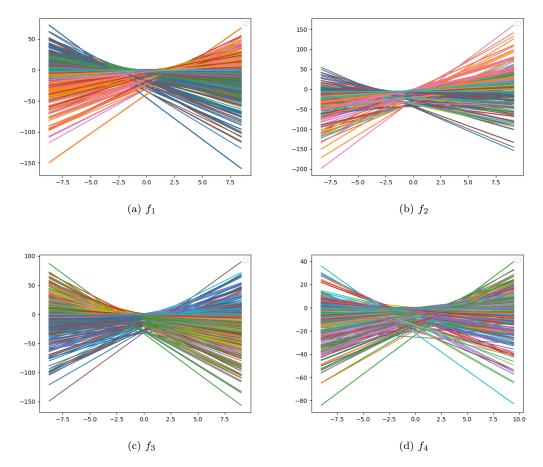
7

(a) f_1

(b) f_2

(c) f_3

(d) f_4

Figure 5: Cuts generated by the GroupMax network estimating solution of equation (3) using 3 layers and a group size of 5.

Table 4: MSE with $X \sim \mathcal{N}(0,1)^2$.

Network	f_5	f_6	f_7
Feedforward	1.6	1.99	1.7e-3
[6]	0.019	0.073	6.5e-6
GroupMax	0.057	0.21	8e-7

For the network [6], we keep on using 3 hidden layers with 10 neurons. As for the feedforward network we also take 3 layers with 10 neurons. For the GroupMax we take 5 layers of 10 neurons with a group size of 2. The learning rate is still equal to 10^{-3} and we take 100000 gradient iterations. Results depending on the dimension of the problem are given in table 8,9 ,10, 11 either sampling using a gaussian law for X in equation (3), or sampling X uniformly in $[-2,2]^d$. The best of 10 runs is given. As an L_2 interpolator, classical feedforward seems to be the best and the GroupMax slightly outperforms the [6] network. Notice that we did not try other activation functions for the [6] network and did not play with the number of neurons to upgrade the results. On the same cases we then increase the number of layers in dimension 5 and give the results in table 12 using the GroupMax network depending on the number of layers q. The best of 10 runs is given. Clearly we see that this increase of the number of layers improves the results even if the approximation of f_9 remains not very good.

At last we try to approximate a function in very high dimension :

$$f_{10}(x,y) = -\frac{1}{2n}x^\intercal x + \frac{1}{2m}y^\intercal y$$

where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$, with $n = 376$, $m = 17$. We train the different methods to approximate f_{10} using $X \sim (\mathcal{N}(0,1))^n$ and $Y \sim (\mathcal{N}(0,1))^m$. We estimate the MSE for each method in table 13 depending on the number of layers q taking 10 neurons for [6] network, 20 neurons for the feedforward

Table 5: MSE with $X \sim \mathbf{U}([-2,2]^2)$.

Network	f_5	f_6	f_7
Feedforward	0.01	0.13	1.8e-4
[6]	5.8e-4	2.7e-3	5e-7
GroupMax	3.5e-3	6e-3	4e-7

Table 6: Influence on the MSE of the Number of Layers q Sampling $X \sim \mathcal{N}(0,1)^2$.

q	f_5	f_6	f_7
3	0.052	0.088	7e-7
4	0.033	0.099	3e-6
5	0.068	0.051	3e-6

Table 7: Influence on the MSE of the Number of Layers q Sampling $X \sim \mathbf{U}([-2,2]^2)$.

q	f_5	f_6	f_7
3	2.3e-3	4.8e-3	1.9e-7
4	9.6e-4	3.9e-3	5e-7
5	5.9e-4	1.8e-3	5e-7

Table 8: MSE for f_8 Sampling $X \sim \mathcal{N}(0,1)^d$.

Dimension (d)	2	3	4	5
Feedforward	8e-4	1e-3	5e-3	0.016
[6]	2.8e-3	1.2e-2	6.9e-2	0.197
GroupMax	5.3e-4	5.1e-3	1.3e-2	0.030

Table 9: MSE for f_9 Sampling $X \sim \mathcal{N}(0,1)^d$.

Dimension	2	3	4	5
Feedforward	1.2e-3	6.7e-3	0.026	0.105
[6]	6e-3	3.2e-2	0.141	0.361
GroupMax	3.9e-3	2.8e-2	0.075	0.228

Table 10: MSE for f_8 Sampling $X \sim \mathbf{U}([-2,2]^d)$.

Dimension	2	3	4	5
Feedforward	2.7e-4	7.7e-4	1.6e-3	5.6e-3
[6]	7.5e-4	5.5e-3	1.8e-2	4.5e-2
GroupMax	3.9e-4	1.5e-3	4.4e-3	1.2e-2

Table 11: MSE for f_9 Sampling $X \sim \mathbf{U}([-2,2]^d)$.

Dimension	2	3	4	5
Feedforward	7.8e-4	9.2e-3	1.2e-2	7.6e-2
[6]	2.6e-3	2.3e-2	8.7e-2	0.38
GroupMax	2e-3	2.8e-2	6.3e-2	0.17

Table 12: MSE for f_8 and f_9 Depending on the Number of Layers q Sampling $X \sim \mathbf{U}([-2,2]^5)$.

q	4	6	7	8
f_8	0.021	0.012	8.1e-3	7.7e-3
f_9	0.278	0.164	0.155	0.120

Table 13: MSE for Function f_{10} Depending on the Number of Layers q.

q	Feed forward	[6]	GroupMax
3	0.0025	0.0041	0.0073
5	0.0025	0.0029	0.0037
7	0.0024	0.0022	0.0024
9	0.0025	0.0019	0.0019

and 12 neurons with a group size of 2 for the GroupMax network. As previously, we take the best result of ten runs. The MSE obtained is small for all methods. The feedforward network's results are nearly independent of the number of layers while the two other methods get better results as the number of layers increases and tend to outperform the feedforward network.

4 Conclusion

A new effective network has been developed to approximate convex function or partially convex functions by cuts or conditional cuts. This network gives similar results to the best networks developed giving a convex or partially convex solution. This approximation by cuts can be used in many application where convexity of the approximation is required or could be used in multistage linear continuous stochastic optimization where of the Bellman values by cuts is necessary to use a Linear Programming solver.

5 Acknowledgement

References

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[3] G. Gripenberg, "Approximation by neural networks with a bounded number of nodes at each level," *Journal of approximation theory*, vol. 122, no. 2, pp 260–266, 2003.

[4] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," *Proceedings of Machine Learning Research*, Graz, Austria, vol. 125, pp. 230–2327, 2020.

[5] G. C. Calafiore, S. Gaubert, and C. Possieri, "Log-sum-exp neural networks and posynomial models for convex and log-log-convex data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 827–838, 2020.

[6] B. Amos, L. Xu, and J.Z. Kolter, "Input convex neural networks," *International Conference on Machine Learning*, Sydney NSW, Australia, pp. 146–155, 2017.

[7] A. Makkuva, A. Taghvaei, and S. Oh, "Optimal transport mapping via input convex neural networks," *International Conference on Machine Learning*, Vienna, Austria, pp. 6672–6681, 2020.

[8] A. Korotin, V. Egiazarian, A. Asadulaev, A. Safin, and E. Burnaev, "Wasserstein-2 generative networks," 2019. [Online]. Available: *arXiv:1909.13082*.

[9] Y. Chen, Y. Shi, and B. Zhang, "Optimal control via neural networks: A convex approach," 2018. [Online]. Available: *arXiv:1805.11835*.

[10] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," *Learning for Dynamics and Control*, Zurich, Switzerland, pp. 361–373, 2020.

[11] S. Mukherjee, S. Dittmer, Z. Shumaylov, S. Lunz, O. Öktem, and C.B. Schönlieb, "Learned convex regularizers for inverse problems," 2020. [Online]. Available: *arXiv:2008.02839*.

[12] Y. Chen, Y. Shi, and B. Zhang, "Input convex neural networks for optimal voltage regulation," 2020. [Online]. Available: *arXiv:2002.08684*.

[13] A. Shapiro, "Analysis of stochastic dual dynamic programming method, " *European Journal of Operational Research*, vol. 209, no. 1, pp. 63–72, 2009.

[14] G. Balázs, A. György, and G. Szepesvári, "Near-optimal max-affine estimators for convex regression," *Artificial Intelligence and Statistics*, San Diego, California USE, pp. 56–64, 2015.

[15] A. Ghosh, A. Pananjady, A. Guntuboyina, and K. Ramchandran, "Max-affine regression: Provable, tractable, and near-optimal statistical estimation, " 2019. [Online]. Available: *arXiv:1906.09255*.

[16] A. Ghosh, A. Pananjady, A. Guntuboyina, and K. Ramchandran, "Max-affine regression: parameter estimation for gaussian designs," *IEEE Transactions on Information Theory*, vol. 68, no. 3, pp. 1851–1885, 2022.

[17] C. Anil, J. Lucas, and R. Grosse, "Sorting out Lipschitz function approximation," *International Conference on Machine Learning*, Long Beach, California, USA, pp. 291–301, 2019.

[18] U. Tanielian, M. Sangnier, and G. Biau, "Approximating Lipschitz continuous functions with Group-Sort neural networks," *International Conference on Artificial Intelligence and Statistics*, Virtual Conference, pp. 442–450, 2021.

[19] S. Boyd, and L. Vandenberghe, "Convex optimization," Cambridge university press, 2004.

[20] A. Guessab, and G. Schmeisser, "Convexity results and sharp error estimates in approximate multivariate integration," *Mathematics of computation*, vol. 73, no. 247, pp. 1365–1384, 2004.

[21] M. Abadi and al, "TensorFlow: A System for Large-Scale Machine Learning," *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, Savannah, GA, USA, pp. 265–283, 2016.

[22] D. Kingma, J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: *arXiv:1412.6980*.