The GroupMax neural network approximation of convex functions.

Xavier Warin *

June 16, 2022

Abstract

We present a new neural network to approximate convex functions. This network has the particularity to approximate the function with cuts and 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.

Acknowledgement

We thank Vincent Lemaire and Gilles Pagès for helpful discussions.

1 Introduction

The approximation of a convex function using neural networks has been investigated in [AXK17] 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 [Mak+20] [Kor+19], in optimal control problems as in [CSZ18] [Agr+20], in inverse problems [Muk+20], or in general optimization problems [CSZ20] 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 SDDP method [Sha11] solve some transition problems starting from a state using some cut approximation of the Bellman function at the end of the transition period such that the transition problem can be solved using Linear Programming solvers.
The cut approximation of a convex function using regression methods has already been investigated in [BGS15] or [Gho+19] 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 [ALG19] and analyzed in [TSB21].
In a first part, we present the network supposing first 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 simple network generating simple cuts and is competitive with the ICNN.

2 The GroupMax network

Let f be a real valued convex function defined on $\mathbb{R}^d$. We have the following representation [BBV04, Chapter 3.2.3]: Define $\tilde{f}$ as:

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

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

then $\tilde{f}(x) = f(x)$ for $x \in \text{int Dom}(f)$.
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 [ALG19] 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

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

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

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

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 $MG^{K(q-1)}$.

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

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

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 [GS04, 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 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 $(x_i)_{i\in\mathbb{N}}$ 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. □

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 [AXK17] to take into account the non convexity in $\tilde{x}$: other networks tested couldn't outperform the proposed one in [AXK17].
The recursion is given by:

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

where $\circ$ denote the Hadamard product, $\otimes$ is applied between a matrix $A \in \mathbb{R}^{m\ n}$ and a vector $B \in \mathbb{R}^n$ such that $A \otimes B \in \mathbb{R}^{m\ n}$ 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, $\tilde{W}_0 \in \mathbb{R}^{m_x\ (d\ k)}$, $\tilde{W}_i \in \mathbb{R}^{m_x\ m_x}$ for $i > 0$, $W^{(zu)}_i \in \mathbb{R}^{m_y\ m_x}$ for $i > 0$, $W^{(z)}_i \in \mathbb{R}^{m_y\ m_y}$ for $i \leq q$. We don't detail the size of the different matrix $W^{(y)}$, $W^{(yu)}$, $W^{(u)}$ and the different bias that are obvious. θ is the set of all these weights and bias.
Introducing

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

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

$$(z_{i+1})_j = \max_{l \in [1,G]} [A^{i+1}(\tilde{x}) z_i + \tilde{A}^{i+1}(\tilde{x}) y + B^{i+1}(\tilde{x})]_{(j-1)G+l} \quad \text{for } j = 1, \dots, K, \text{ and } i < q-1$$

$$h^\theta(\tilde{x}, y) = \max_{l \in [1,m_y]} [A^q(\tilde{x}) z_{q-1} + \tilde{A}^q(\tilde{x}) y + B^q(\tilde{x})]_l.$$

Similarly to section 2.1 ,

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

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

3 Numerical results

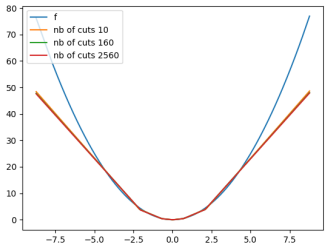
In all numerical results, we use tensorflow [Aba+16] with an ADAM gradient descent algorithm [KB14].

3.1 Some one dimensional results

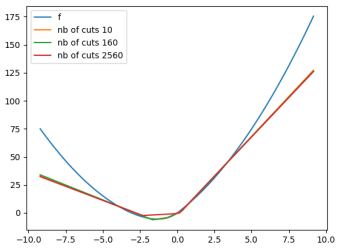
We consider the convex function $f(x) = f_i(x)$ for case $i = 1, \dots, 5$ where

1. $f_1(x) = x^2$
2. $f_2(x) = x^2 + 10[(e^x - 1)1_{x<0} + x 1_{x \geq 0}]$
3. $f_4(x) = (|x|^2 + 1)^2$
4. $f_5(x) = |x| 1_{|x| \geq 3} + \frac{x^2}{2} 1_{|x| < 3}$

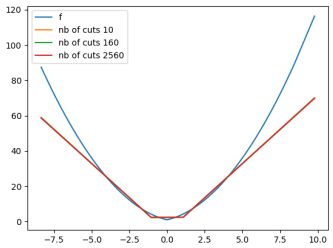
We regress $f(x) + \epsilon$ with respect to x where $\epsilon \sim \mathcal{N}(0, 1)$ using 20000 gradient iterations, a batch size equal to 300 and a learning rate equal to $1e-3$. We then solve equation (3) using $X \sim \mathcal{N}(0, 4)$. On figure 1, we see that the simple approximation (2) gives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



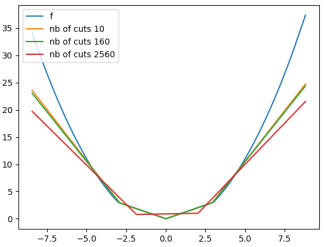
f_1



h_3



f_3



f_4

Figure 1: Solution of equation (3) using network (2) with different 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 $f(x) +$ such that both are centered with a unit standard deviation. Results are given on gure 2.

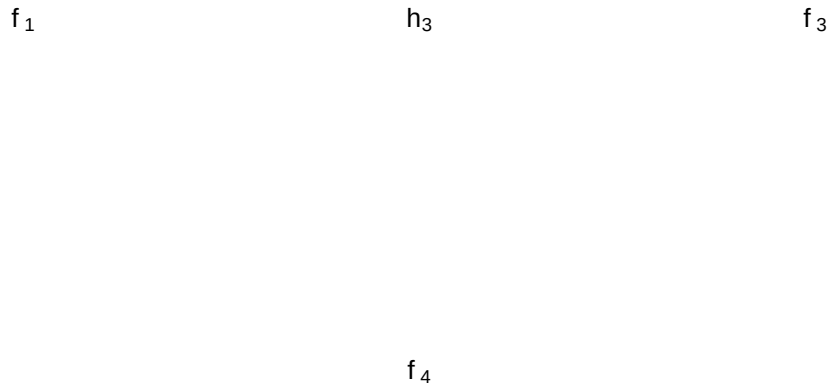


Figure 2: Solution of equation (3) using network (2) with di erent number of cuts using some input and output renormalization.

On the gure 3 we plot the solution obtained using the GroupMax network, [AXK17] 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.

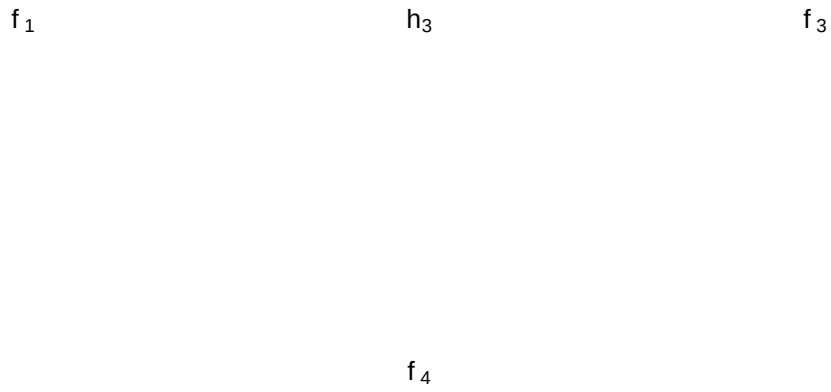


Figure 3: Solution of equation (3) using the GroupMax, the ICCN network [AXK17] and a feedforward network.

The cuts generated on the previous examples by the GroupMax network are given on gure 4.

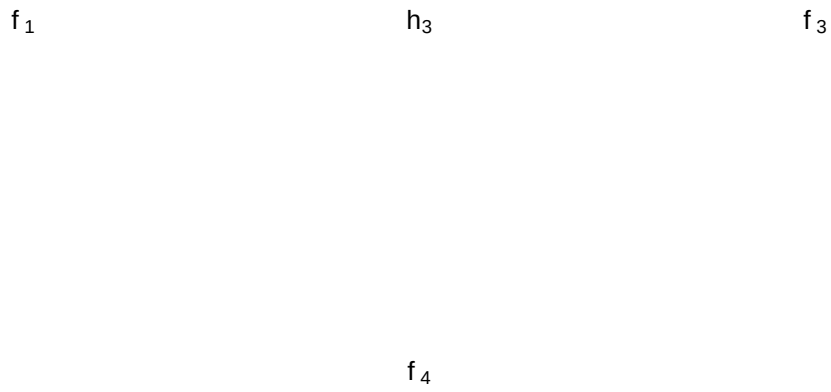


Figure 4: Cuts generated by the GroupMax network estimating solution of equation (3) using 3 layers and a group size of 5.

To see the accuracy of the GroupMax network as an interpolator, we optimize equation (3) trying to t directly the function f without any noise. Results are given on table 1. The solution obtained

by the feedforward seems to be more accurate with the chosen parameters. Between [AXK17], and the GroupMax it is hard to say which is the best.

Network	f_1	f_2	f_3	f_4
Feedfoward	0.0004	0.0031	0.0004	0.0001
[AXK17] network	0.0013	0.0050	0.0019	0.0006
GroupMax network	0.0012	0.0014	0.0029	0.0002

Table 1: L_2 interpolation error using Monte Carlo (1e6 samples). Best of 10 runs.

In table 2, we give the results obtained for di erent values of parameter$K$ using 12 neurons per layer. As before the best of 10 runs is kept.

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

Table 2: Testing in uence of the group size on the result. Best of 10 runs.

The results seem to indicate that the group size should remain rather low.

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

Table 3: Testing the in uence of the number of layers q with the GroupMax network taking 12 neurons and a group size of 2. Best of 10 runs.

At last, the in uence of the number of layers is given in table 3 taking 12 neurons per layer. The accuracy clearly improves as we increase the number of layers.

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 rst case, we suppose that$X \sim N(0;1)$, then that $X \sim U([-2;2]^2)$. We test the following functions convex in y:

1. $f_1(x;y) = |y^2 - |x + 2|x^3||$
2. $f_2(x;y) = (1 + |y|)|x + 2|x^3||$
3. $f_3(x;y) = |y^+ - |x| + x^2|$

We keep the same parameters as before for the di erent networks. For the ICNN and the GroupMax network we take the same number of neurons both for the convex and non convex part. We rst 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 and 5. Surprisingly, the feedfoward network behaves very badly especially sampling a Gaussian law.

Network	f_1	f_2	f_3
Feedforward	1.6	1.99	1.7e-3
[AXK17]	0.019	0.073	6.5e-6
GroupMax	0.057	0.21	8e-7

Table 4: Testing convexity approximation with gaussian sampling$X \sim N(0;1)$. Best of 10 runs.

Network	f_1	f_2	f_3
Feedforward	0.01	0.13	1.8e-4
[AXK17]	5.8e-4	2.7e-3	5e-7
GroupMax	3.5e-3	6e-3	4e-7

Table 5: Testing convexity approximation with uniform sampling $X \sim U([-2,2]^2)$. Best of 10 runs.

Remark 3. It was possible to get better results for the f_2 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.

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 on table 6 7. Sampling an uniform law, the error clearly decreases with the number of layers while sampling a gaussian law this decrease is not observed.

q	f_1	f_2	f_3
3	0.052	0.088	7e-7
4	0.033	0.099	3e-6
5	0.068	0.051	3e-6

Table 6: Partial convex input: Testing the influence of the number of layers q with the GroupMax network taking 12 neurons and a group size of 3 sampling $X \sim N(0,1)$. Best of 10 runs.

q	f_1	f_2	f_3
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 7: Partial convex input: Testing the influence of the number of layers q with the GroupMax network taking 12 neurons and a group size of 3 sampling $X \sim U([-2,2]^2)$. Best of 10 runs.

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.
$$f_1(x) = \|x\|^2$$
2. For the second we generate randomly a positive definite matrix in dimension d and take
$$f_2(x) = \sum_{i=1}^{d} (|x_i| + |1 - x_i|) + x^T A x$$

For the network [AXK17], 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.

Dimension	2	3	4	5
Feedforward	8e-4	1e-3	5e-3	0.016
[AXK17]	2.8e-3	1.2e-2	6.9e-2	0.197
GroupMax	5.3e-4	5.1e-3	1.3e-2	0.030

Table 8: L_2 Approximation of convex function f_1 in different dimensions (best of 10 runs) for $X \sim N(0,1)$ in equation (3).

Dimension	2	3	4	5
Feedforward	1.2e-3	6.7e-3	0.026	0.105
[AXK17]	6e-3	3.2e-2	0.141	0.361
GroupMax	3.9e-3	2.8e-2	0.075	0.228

Table 9: L_2 Approximation of convex function f_2 in different dimensions (best of 10 runs) for $X \sim N(0;1)$ in equation (3).

Dimension	2	3	4	5
Feedforward	2.7e-4	7.7e-4	1.6e-3	5.6e-3
[AXK17]	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 10: L_2 Approximation of convex function f_1 in different dimensions (best of 10 runs) for $X \sim U([-2;2]^d)$ in equation (3).

Dimension	2	3	4	5
Feedforward	7.8e-4	9.2e-3	1.2e-2	7.6e-2
[AXK17]	2.6e-3	2.3e-2	8.7e-2	0.38
GroupMax	2e-3	2.8e-2	6.3e-2	0.17

Table 11: L_2 Approximation of convex function f_2 in different dimensions (best of 10 runs) for $X \sim U([-2;2]^d)$ in equation (3).

Results are given in table 8,9 ,10, 11 either sampling using a gaussian law for X in equation (3), or sampling X uniformly in $[-2;2]^d$. As an L_2 interpolator, classical feedforward seems to be the best and the GroupMax slightly outperforms the [AXK17] network. Notice that we did not try other activation function for the [AXK17] network and did not play with the number of neurons to upgrade the results. At last we increase the number of layers in dimension 5. Clearly we see that this increase of the number of layers improves the results even if the approximation of f_2 remains not very good.

q	4	6	7	8
f_1	0.021	0.012	8.1e-3	7.7e-3
f_2	0.278	0.164	0.155	0.120

Table 12: L_2 approximation of f_1 and f_2 by the GroupMax network in dimension 5 depending on the number of layers q for $X \sim U([-2;2]^5)$. Best of 10 runs.

4 Conclusion

A new effective network has been developed to approximate convex function or partial convex functions but cuts or conditional cuts. This network gives similar results to the best networks developed giving 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 approximation of the Bellman values by cuts is necessary to use a Linear Programming solver.

References

[Aba+16] Martn Abadi et al. "{TensorFlow}: A System for {Large-Scale} Machine Learning". In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016, pp. 265{283.