The GroupMax Neural Network Approximation of Convex Functions

Xavier Warin

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 its efficiency. The network is competitive with the most efficient convexity-preserving neural networks and can be used to approximate functions in high dimensions.

Index Terms—Approximation, Benders cuts, convex function, neural network, partially convex function.

I. INTRODUCTION

Neural networks are effective tools to approximate function and numerically generally outperform classical regression using an expansion on a function basis. Some classical Universal Approximation theorems for neural network with bounded depth are given in [1] and [2] when the activation function is nonpolynomial. A “dual” result is given in [3] where the number of the hidden layers can be taken arbitrarily large with bounded widths if the activation function is nonaffine, continuous and twice continuously differentiable. Recently, [4] slightly improves the results with only nonaffine, 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] and [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 is not sufficient. For example, in multistage stochastic linear optimization, some methods such as the stochastic dual dynamic programming (SDDP) method [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 brief proposes a different approach using a new network permitting to efficiently approximate of a convex function by cuts using some ideas similar to the GroupSort network developed in [17] and analyzed in [18].

In the 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. The 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.

II. GROUPMAX NETWORK

Let \( f \) be a real-valued convex function defined on \( \mathbb{R}^d \). We have the following representation [19, Ch. 3.2.3]: Define \( \hat{f} \) as

\[
\hat{f}(x) = \sup \{ g(x) | g \text{ affine}, g(z) \leq f(z) \}.
\]

Then \( \hat{f}(x) = f(x) \) 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 (1), we may think of a first single layer network \( h \)

\[
h^q(x) = \max_{i=1}^N A_i \cdot x + b_i
\]

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 \( D \) we solve

\[
\theta^* = \arg \min \mathbb{E}((f(X) - h^q(X))^2)
\]

where \( X \) is a random variable for example uniformly distributed in \( D \). 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.

A. Network for Fully Convex Functions

1) 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 \( K = \frac{N}{G} \) is an integer corresponding to the number of groups. For \( x \in \mathbb{R}^d \), the network is defined by recurrence as

\[
\zeta^j = \rho(A^j x + B^j)
\]

\[
\zeta^j = \rho((A^j)^\dagger \zeta^j - 1 + B^j)1 < i < q
\]

\[
h^q(x) = \tilde{\rho}((A^q)^\dagger \zeta^q - 1 + B^q)
\]

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

The activation function \( \rho \) and \( \tilde{\rho} \) are defined as follows.

1) \( \tilde{\rho} \) is a \( \mathbb{R} \) valued function defined on \( \mathbb{R}^d \) where

\[
\tilde{\rho}(x) = \max(x_1, \ldots, x_M)
\]

for \( x \in \mathbb{R}^d \).

2) \( \rho \) is a function from \( \mathbb{R}^M \) in \( \mathbb{R}^K \) such that

\[
\rho(x_i) = \max(x_{(i-1)G+1}, \ldots, x_{iG})
\]

for \( i = 1, \ldots, K \).

And an example of the structure of the network is given in Fig. 1.
This network gives an approximation of $f$ by some cuts: clearly by positivity of the $(A_i)^+$, we get that

$$h^0(\hat{x}) = \max_{i\in[1,M]} \max_{j\in[1,G]} \sum_{k=1}^{K} (A_i)^+ \xi_{j,k-1}^{\epsilon}\left(\hat{x}\right),$$

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 $MG^0(q-1)$.

Remark 1: Using $M$ neurons on the first layer and $\hat{M} \geq \sum_{i=1}^{d} d_i$ neurons on the following layers, we notice that if we take $B_i = 0$ for $1 < j \leq q$, and $A_i$ null except $(A_i)^j = 1$, for $i = 1, \ldots, M$, 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 (5), it is possible to reconstruct the underlying cuts which are, for example, necessary to solve stochastic linear optimization problems using Benders cuts.

2) Universal Approximation Theorem: We denote $\mathcal{N}(M_i, \ldots, M_q, K)$ the set of generated functions $h^{M_i,\ldots, M_q, K}$ by the network (4) with $\theta = (\theta_1, \ldots, \theta_M, K) \in \mathbb{R}^{M(1)+\sum_{i=1}^{d} M_i+K}$ where the number of neurons for layer $q$ is $M_q$. We introduce the space of functions generated letting the number of neurons vary

$$\mathcal{N}(K, q) = \bigcup_{i=1,\ldots,M_q} \mathcal{N}(M_1, \ldots, M_q, K).$$

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 $K$ for the sup norm.

Proof: Due to (1), it is sufficient to prove that for a given function $f$, for each $\epsilon$, 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 $\sup f(x) - g(x) \leq \epsilon$.

Let us suppose the contrary. There exist $\epsilon$, such that for $n_0$ chosen arbitrarily, and whatever the cuts we generate 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 $f(x) = A x + B$ and $f(x) \geq A x + B$ on $K$. Let use define, $g(x) = max(g(x), A x + B)$. Due to the hypothesis, there exist $\epsilon$ such that $g(x) - g^0(x) > \epsilon$ and we can build a sequence $(x_i, g_i)$, $i \geq 0$ such that $g_i$ is below $g$ and $f(x_i) - g_i(x_i) \geq \epsilon$ for all $j \leq i$. We do not detail the size of the different matrices $W_i^{(m)}$, $W_i^{(m)}$ and the different biases that are obvious. $\theta$ is the set of all these weights and bias. Introducing

$$A^\prime(\hat{x}) = \left[ W_i^{(m)} \otimes W_i^{(m)} u_{i-1} + b_i^{(m)} \right]_{i=1,\ldots,K}$$

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

$$h^0(\hat{x}, y) = \max_{j=1,\ldots,K} \left( A^\prime(\hat{x}) y + B(\hat{x}) \right)$$

Similar to Section II-A

$$h^0(\hat{x}, y) = \max_{j=1,\ldots,K} \left( A(\hat{x}) y + B(\hat{x}) \right)$$

where $A(\hat{x}) \in \mathbb{R}^{m_\theta e^{K(i-1)}x_k}$ is a function of the $(A(\hat{x}))_{i=1,\ldots,\theta}$ and $(A(\hat{x}))_{i=1,\ldots,\theta}$ matrices. Similarly $B(\hat{x}) \in \mathbb{R}^{m_\theta e^{K(i-1)}x_k}$ is a function of the $(B(\hat{x}))_{i=1,\ldots,\theta}$, $(A(\hat{x}))_{i=1,\ldots,\theta}$ and $(A(\hat{x}))_{i=1,\ldots,\theta}$. Then it is clear that this recursion permits to define some cuts conditional to $\hat{x}$.

III. NUMERICAL RESULTS

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

A. Some 1-D Results

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

1) $f_1(x) = x^2$.
2) $f_2(x) = x^2 + 10((e^x - 1) 1_{x<0} + x 1_{x \geq 0})$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
TABLE I

| Network    | $f_1$  | $f_2$  | $f_3$  | $f_4$  |
|------------|--------|--------|--------|--------|
| Feedforward| 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 |

TABLE II

| $K$ | $f_1$  | $f_2$  | $f_3$  | $f_4$  |
|-----|--------|--------|--------|--------|
| 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  | 0.0007 | 0.0071 | 0.0137 | 0.00085|

TABLE III

| $q$ | $f_1$  | $f_2$  | $f_3$  | $f_4$  |
|-----|--------|--------|--------|--------|
| 2   | 0.0073 | 0.0054 | 0.062  | 6.1e-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 regress $f(x) + \epsilon$ with respect to $x$ where $\epsilon \sim \mathcal{N}(0,1)$ using 20 000 gradient iterations, a batch size equal to 300 and a learning rate equal to $1e-3$. We then solve (3) using $X \sim \mathcal{N}(0, 4)$. In Fig. 2, we see that the simple approximation (2) gives visually very good results and that the solution does not 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 in Table III. In Fig. 3, we plot the solution obtained using the GroupMax network, [6] network and the feedforward network. For the GroupMax, we use three layers with ten neurons on each layer and the group size $G = 5$. As for the two other networks, we use three hidden layers with ten neurons and the ReLU activation function. 50,000 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 (3) trying to fit directly 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 II, we give the results obtained (best of ten runs) for different values of parameter $K$ using 12 neurons per layer. As before the best of ten 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 III taking 12 neurons per layer and a group size of 2 in the GroupMax network. The best of ten runs is given. The accuracy clearly improves as we increase the number of layers.

B. Testing Partial Convexity in 2-D

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 $X \sim U([-2, 2]^2)$. We test the following functions convex in $y$.

1) $f_3(x, y) = y^2[x + 2x^3]$.
2) $f_4(x, y) = (1 + |y|)[x + 2x^3]$.
3) $f_5(x, y) = y^4|x| + x^2$.

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 nonconvex part. We first keep a learning rate equal to $10^{-3}$ and a batch size equal to 300. We take 50,000 gradient iterations. Results are given in Table IV sampling $X \sim \mathcal{N}(0, 1)^2$ and Table V with $X \sim U([-2, 2^2])$. Best of ten runs are given. Surprisingly, the feedforward network behaves very badly especially sampling a Gaussian law.
Fig. 5. Cuts generated by the GroupMax network estimating solution of (3) using three layers and a group size of 5. (a) $f_1$. (b) $f_2$. (c) $f_3$. (d) $f_4$.

TABLE IV
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  |

TABLE V
MSE WITH $X \sim \mathcal{U}([-2, 2]^2)$

| 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 VI
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 | 1e-7  |
| 4   | 0.033 | 0.099 | 3e-6  |
| 5   | 0.063 | 0.051 | 3e-6  |

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.

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 Tables VI and VII. The best of ten runs is given. In sampling a uniform law, the error clearly decreases with the number of layers while sampling a Gaussian law this decrease is not observed.

C. Convexity in Higher Dimension

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

1) First, it is the square of the $L_2$ norm

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

2) For the second, we generate randomly a positive definite matrix in dimension $d$ and take

$$f_9(x) = \sum_{i=1}^{d} (|x_i| + |1 - x_i|) + x^T Ax.$$ 

Rem for the network [6], we keep on using three hidden layers with ten neurons. As for the feedforward network we also take three layers with ten neurons. For the GroupMax, we take five layers of ten 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 Tables VIII–XI either sampling using a Gaussian law for $X$ in (3), or sampling $X$ uniformly in $[-2, 2]^d$. The best of ten 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. In the same cases, we then increase the number of layers in dimension 5 and give the results in Table XII using the GroupMax network depending on the number of layers $q$. The best of ten runs is given. Clearly, we see that this increase of the number of layers improves the results even if the approximation of $f_8$ remains not very good. At last, we try to approximate a function in a very high dimension

$$f_{10}(x, y) = -\frac{1}{2n} x^T x + \frac{1}{2m} y^T y.$$
TABLE XIII
MSE FOR FUNCTION $f_{10}$ DEPENDING ON THE NUMBER OF LAYERS $q$

| $q$ | Feedforward | [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  |

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 $X \sim (N(0, 1))^n$ and $Y \sim (N(0, 1))^m$. We estimate the mse for each method in Table XIII depending on the number of layers $q$ taking ten neurons for [6] network, 20 neurons for the feedforward 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.

IV. CONCLUSION

A new effective network has been developed to approximate convex functions 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 applications where convexity of the function is required or could be used in multistage linear stochastic optimization where the Bellman values by cuts 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 applications where convexity of the function is required or could be used in multistage linear stochastic optimization where the Bellman values by cuts 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 applications where convexity of the function is required or could be used in multistage linear stochastic optimization where the Bellman values by cuts or partially convex functions by cuts or conditional cuts.

Y. Chen, Y. Shi, and B. Zhang, “Optimal control via neural networks: A convex approach,” 2018, arXiv:1805.11835.

A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, “Learning convex optimization control policies,” in Proc. 2nd Conf. Learn. Dyn. Control, Zürich, Switzerland, 2020, pp. 361–373.

S. Mukherjee, S. Dittmer, Z. Shumaylov, S. Lunz, O. Öktem, and C.-B. Schölnie, “Learned convex regularizers for inverse problems,” 2020, arXiv:2008.02839.

Y. Chen, Y. Shi, and B. Zhang, “Input convex neural networks for optimal voltage regulation,” 2020, arXiv:2002.08684.

A. Shapiro, “Analysis of stochastic dual dynamic programming method,” Eur. J. Oper. Res., vol. 209, no. 1, pp. 63–72, 2011.

G. Balázs, A. György, and G. Szepesvári, “Near-optimal max-affine estimators for convex regression,” in Proc. 18th Int. Conf. Artif. Intell. Statist., San Diego, CA, USA, 2015, pp. 56–64.

A. Ghosh, A. Pananjady, A. Guntuboyina, and K. Ramchandran, “Max-affine regression: Provably tractable, and near-optimal statistical estimation,” 2019, arXiv:1906.09255.

A. Ghosh, A. Pananjady, A. Guntuboyina, and K. Ramchandran, “Max-affine regression: Parameter estimation for Gaussian designs,” IEEE Trans. Inf. Theory, vol. 68, no. 3, pp. 1851–1885, Mar. 2022.

C. Anil, J. Lucas, and R. Grosse, “Sorting out Lipschitz function approximation,” in Proc. Int. Conf. Mach. Learn., Long Beach, CA, USA, 2019, pp. 291–301.

U. Tanielian, M. Sangnier, and G. Biou, “Approximating Lipschitz continuous functions with groupsort neural networks,” in Proc. Int. Conf. Artif. Intell. Statist., 2021, pp. 442–450.

S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press, 2004.

A. Gueesb and G. Schmeiser, “Convexity results and sharp error estimates in approximate multivariate integration,” Math. Comput., vol. 73, no. 247, pp. 1365–1384, 2004.

M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” in Proc. 12th USENIX Symp. Operat. Syst. Design Implement. (OSDI), Savannah, GA, USA, 2016, pp. 265–283.

P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv:1412.6980.