Effect of Simple Operations on the Linear Regions of Continuous Piece-wise Linear Functions

Haojun Zhu(SCIPER: 312864)
Section of Mathematics, EPFL
Semester project, 8 ECTs
June, 2021

***Abstract*—Deep neural networks with ReLU activation function produce continuous piece-wise linear(CPWL) functions. Any such function can be represented by a generalized hinging hyperplane(GHH). We implement a model to generate GHH function with a grid method for the input. We study experimentally some statistics related to composed CPWL functions, especially the number of linear regions and the Lipschitz constant.**

I. Introduction

In recent years, the application of deep neural networks has brought significant improvement in terms of performance for many different machine learning tasks[1, 2, 3]. The great success of this model can be, to some extent, credited to its expressivity. A fully connected neural network using ReLU as activation function can be seen as a continuous piece-wise linear(CPWL) function which maps its input space into distinct linear regions. For one linear region in the input space, the gradient is identical for any input in this region. The number of linear regions can reflect how expressive the network is. There exists many studies on giving tighter theoretical upper and lower bounds for this number of linear regions[4, 5]. In this project, our focus is to study experimentally some statistics related to a CPWL function under particular initialization of the weights and bias in the model.

Proved by a former study[6], any CPWL function with input $\boldsymbol{x} \in \mathbb{R}^d$ can be represented by a sum of expanded hinges in the following form:

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} \epsilon_k \max\left(\boldsymbol{a}_1^{k\,\mathrm{T}}\boldsymbol{x} + b_1^k, ..., \boldsymbol{a}_{d+1}^{k\,\mathrm{T}}\boldsymbol{x} + b_{d+1}^k\right) \tag{1}$$

$$= \sum_{k=1}^{K} \epsilon_k \max\left(\mathbf{A}_k\boldsymbol{x} + \boldsymbol{b}_k\right) \tag{2}$$

where $\mathbf{A}_k \in \mathbb{R}^{(d+1)\times d}$, $\boldsymbol{b}_k \in \mathbb{R}^{d+1}$, and ϵ_k takes value in 1 and -1. This is called a generalized hinging hyperplane(GHH), which consists of K basis functions.

II. Model

A. Implementation

In order to study systematically the linear regions and other related statistics, we first implemented the generalized hinging hyperplane model based on a grid method. The input space is modelled by a sequence of equally spaced grid points and the GHH function itself is represented by a feedforward neural network implemented under the Pytorch framework. For most of the following study, we initialize the weights and bias as uniform random variables in the range [-1, 1]. And as mentioned earlier, ϵ_k will take values in $\{\pm 1\}$ with equal probability.

Given an input $\boldsymbol{x}$, we can compute now the output $f(\boldsymbol{x})$. Once the computation is finished, the gradient of $f(\boldsymbol{x})$ at $\boldsymbol{x}$ can be computed via the automatic differentiation methods from Pytorch. Each linear region of a CPWL function should have unique gradient. Therefore, we count the linear regions and partition the input space of this GHH model by the gradient. A straightforward example of a single GHH unit can be demonstrated in Figure 1. The model is also valid for 1-dimensional grid coordinate inputs.

To note that, with this implementation, we always get a scalar output if we only call once the GHH model no matter the input $\boldsymbol{x} \in \mathbb{R}$ or $\boldsymbol{x} \in \mathbb{R}^2$. For the further study on the composed models, we simply generate two times the single GHH unit to get two seperate outputs to form a 2-dimensional output. We note this as GHH_{1d}^2, if $\boldsymbol{x} \in \mathbb{R}$ or GHH_{2d}^2, if $\boldsymbol{x} \in \mathbb{R}^2$.

Limitations also come with this grid method. Since the input space is discrete, it's not possible to compute the gradient at a given position other than the coordinates in

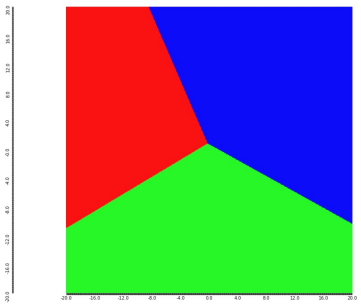

Fig. 1: Example of partition of a 2-dimensional input space into linear regions

the grid and the possibility of omission of some linear regions still exists.

B. Central nodes

With $\boldsymbol{x} \in \mathbb{R}^2$ and $K = 1$, the input space will always be divided into 3 linear regions, as shown in Figure 1. The point of intersection of the 3 linear regions is called the central node in the given grid. For a more general single GHH unit, the function is the summation of K such CPWL functions of 3 linear regions. So the distribution of the positions of central nodes can be crucial for the formation of linear regions. We investigated experimentally the coordinates of the central nodes by generating many times the weights and bias of GHH units and computing the coordinates of the central nodes through the following way.

Let $\mathbf{A}$ and $\boldsymbol{b}$ denote the matrix and vector of the form

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{a}_2{}^{\mathrm{T}} - \boldsymbol{a}_1{}^{\mathrm{T}} \\ \boldsymbol{a}_3{}^{\mathrm{T}} - \boldsymbol{a}_1{}^{\mathrm{T}} \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} b_2 - b_1 \\ b_3 - b_1 \end{bmatrix} \tag{3}$$

Then, the central node $\boldsymbol{x}_c \in \mathbb{R}^2$ can be computed as follow: $\boldsymbol{x}_c = -\mathbf{A}^{-1}\boldsymbol{b}$. We generated 1,000,000 times the weights and bias, the result is shown in the Figure 2:

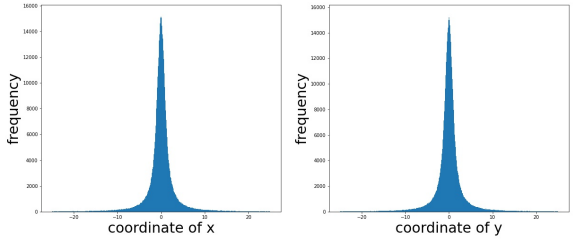

Fig. 2: Distribution of the coordinates of the central nodes

As expected, the coordinates of x and y are both centralized around 0 since the bias are genereted in an iid manner, the vector $\boldsymbol{b}$ has zero mean.

C. Norm of gradient

The norm of gradient is an important quantity to investigate, especially for the composed CPWL functions. For the inputs from two identical grids, if the norm of gradient from a GHH unit has overall higher values, the image will have a larger size and it's more likely to fall into different linear regions in the next layer of the function.

For the single GHH unit, we studied the distribution of the norm of gradient with $K = 3$, where K is the parameter in the formula (2).

The theoretical distribution and the result from the experiments are shown in the Figure 3.

This serves as a sanity check for our implementation

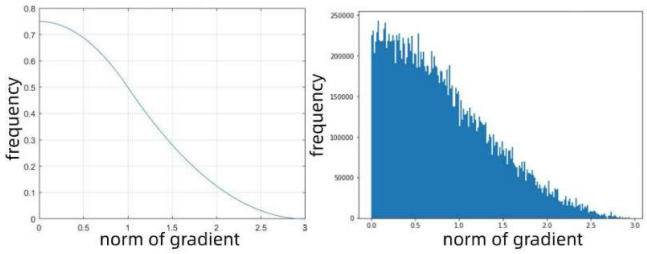

Fig. 3: Distribution of the norm of gradient

of the single GHH model.

III. Composed CPWL function

We define in advance 4 composed architectures as follow:

- architecture 1: with $\boldsymbol{x} \in \mathbb{R}$, $f(\boldsymbol{x}) = GHH^1_{1d} \circ GHH^1_{1d} \circ ... \circ GHH^1_{1d} \circ GHH^1_{1d}(\boldsymbol{x})$, which is a composition of L GHH unit for 1-dimensional input.
- architecture 2: with $\boldsymbol{x} \in \mathbb{R}^2$, $f(\boldsymbol{x}) = GHH^1_{1d} \circ GHH^1_{1d} \circ ... \circ GHH^1_{1d} \circ GHH^1_{2d}(\boldsymbol{x})$
- architecture 3: with $\boldsymbol{x} \in \mathbb{R}$, $f(\boldsymbol{x}) = GHH^1_{2d} \circ GHH^2_{2d} \circ ... \circ GHH^2_{2d} \circ GHH^1_{1d}(\boldsymbol{x})$
- architecture 4: with $\boldsymbol{x} \in \mathbb{R}^2$, $f(\boldsymbol{x}) = GHH^1_{2d} \circ GHH^2_{2d} \circ ... \circ GHH^2_{2d} \circ GHH^1_{2d}(\boldsymbol{x})$

Note that, all the 4 architectures we defined end with a layer to give a scalar output for the convenience to compute the gradient afterwards.

We are interested in these composed architectures because a single GHH unit is not enough to demonstrate the expressiveness of a deep neural network.

Activation pattern: consider a composed architecture of L layers, for each layer l in this composition, we denote the linear regions as $\{r_1^l, r_2^l, ..., r_{N_l}^l\}$. For a specific input $\boldsymbol{x}$, it will go through a set of linear regions of each GHH unit in the architecture that we note as $\{r_{n_1}^1, r_{n_2}^2, ..., r_{n_L}^L\}$. If the activation pattern of 2 inputs are different, these 2 inputs belong to 2 different linear regions under the composed architecture that they have been sent to. For a given range of input, the image of the previous part of the composition could be partitioned into different linear regions when we add an extra GHH unit after the previous part.
For a worst case estimation, the number of linear regions will grow exponentially with the increasing depth of the architectures. However, the number of linear regions hardly reach this upper bound in practice.

A. Number of linear regions

The number of linear regions is an intuitive metric to study the expressivity of composed CPWL functions. We increase the number of layers in the four pre-defined architectures and observe how the number of linear regions evolves. The result is shown in the Figure 4. From the plots, we can see that the particular value of the number of linear regions can vary a lot from one pre-defined function to another due to their different structures. However, we can observe that the number of linear regions increases less and less rapidly with larger number of layers in the function. And eventually, almost all of them come to a stage where the number of linear regions increases slowly or stops to increase.

B. 'Saturation' problem

We analyzed the possible reasons for this saturation of number of linear regions to happen and found the 3 following possibilities:

- The step size(the distance between 2 equally spaced points in the coordinate grid) is not small enough.
- The size of the window of the coordinate grid is not large enough.
- For a deep architecture with our method of initialization of the weights and bias, the image of the input $\boldsymbol{x}$ coming out of the previous layers falls into a small area so it's unlikely to fall into different linear regions of the next GHH unit that we continue to add to the composed CPWL function.

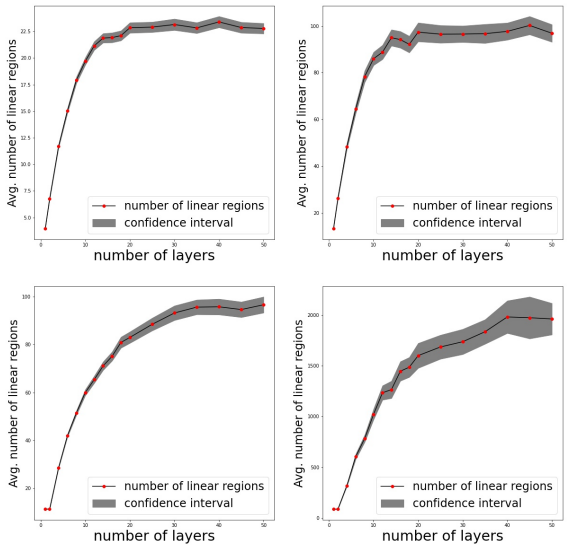

Fig. 4: Average number of linear regions with respect to the number of layers L for the 4 pre-defined composed CPWL functions(on the fisrt row: results for the architecture 1 and architecture 2; on the second row: results for the architecture 3 and architecture 4)

The first 2 listed reasons are the potential result of the drawback of this grid method. Since our grid of the coordinate of the input is discrete, it's not possible to detect a linear region if its size is smaller than the step size. Similarly, if the location of a linear region is outside the chosen window of the input, then again it won't be counted as a valid linear region.
To find out the reason why this saturation happened, we take the architecture 1 as an example and change the step size to see how the curve of the number of regions with respect to the number of layer will behave. The result is show in Figure 5. There is no significant difference between the curves with different values of step size. But there still exists the possibility that the size of undiscovered linear regions is even smaller than the smallest step size being tested. Due to the limitation of computing power, we stop at a smallest step size of 0.001 for the architecture 1.
We also change the size of the window of input and

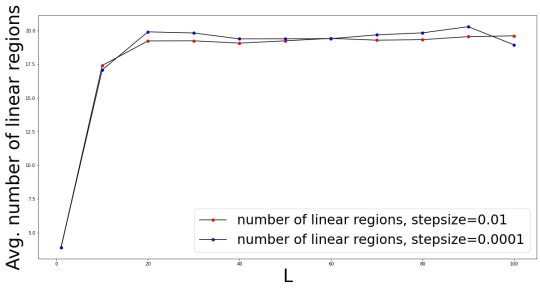

Fig. 5: Average number of linear with two different step sizes

observe the behavior of the variation of number of linear regions. In Figure 6, with a larger window of the grid, an overall increase in the number of lienar regions can be observed. However, the curve also increases slowly or stops to increase when the number of layers becomes large.

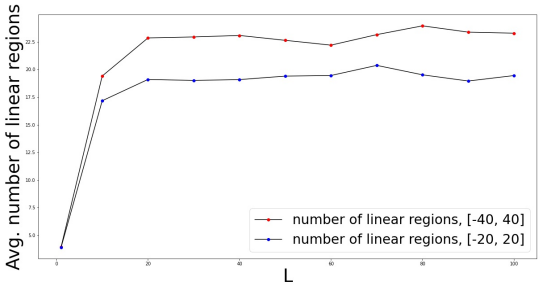

Fig. 6: Average number of linear regions with two different sizes of the window of grid

We conclude that the drawback of the grid method(step size, window size) are not the main reason for this saturation.

C. Scaling factor

If the average norm of gradient is larger, the image of a function from the same input is more likely to fall into a larger area and to reach different linear regions in the next GHH layer; otherwise, the image is less likely to reach new linear regions. To see this, we modify the GHH model by multiplying a scaling factor α. Therefore, the general form of one GHH model becomes:

$$f(\boldsymbol{x}) = \alpha \sum_{k=1}^{K} \epsilon_k \max\left(\mathbf{A}_k \boldsymbol{x} + \boldsymbol{b}_k\right) \tag{4}$$

Using the architecture 1, We test with different values of α (0.1, 0.5, 1.0, 5.0, 10.0), and check the variation of the number of linear regions. The result is shown in Figure 7.

With this result, we can see that the number of linear

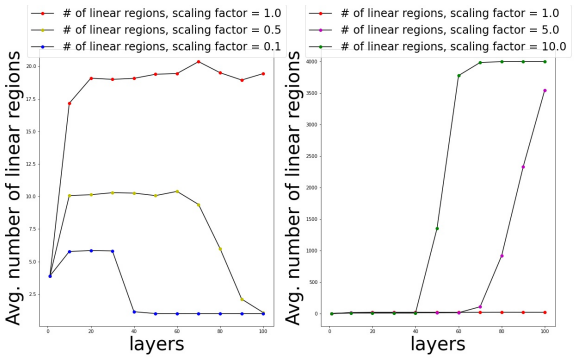

Fig. 7: Average number of linear regions with different scaling factors

regions increase drastically with a scaling factor larger than 1. We see a potential problem of this grid method as well. For a scaling factor smaller than 1, if the the number of layers of the composed model is very large, the local gradient will be close to 0. Due to the computational issue, these extremely small gradients will be counted as 0 by the machine and therefore, the linear regions defined upon these gradients which are actually different from each other won't be detected.

D. Lipschitz constant

An estimation of the Lipschitz constant of the composed CPWL functions can be defined as follow:

- For the case of 1-dimensional inputs, the Lipschitz constant $K^{1d} = \max_{\boldsymbol{x} \in \mathbb{G}} |\nabla f(\boldsymbol{x})|$, where $\mathbb{G}$ is our grid.
- For the case of 2-dimensional inputs, the Lipschitz constant $K^{2d} = \max_{\boldsymbol{x} \in \mathbb{G}^2} \|\nabla f(\boldsymbol{x})\|_2$, where $\mathbb{G}^2$ is our 2-dimensional grid.

We investigate how the Lipschitz constant of a composed CPWL function behave when the number of layers of the function is increasing. Note that, no presence of the scaling factor in the model for this part. The weights and bias are initialized as stated earlier.

For the 4 pre-defined architectures, there are 2 types of behavior of the Lipschitz constant. On the one hand, the Lipschitz constant can be decreasing continuously with respect to the increasing number of layers, see Figure 8. Otherwise, the Lipschitz constant is growing exponentially with the number of layer, see Figure 9.

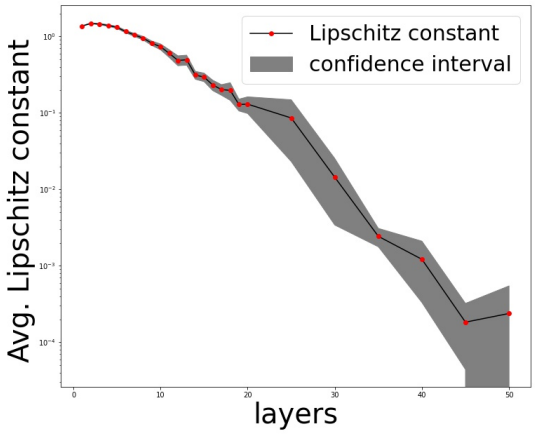

Fig. 8: Lipschitz constant for the architecture 1

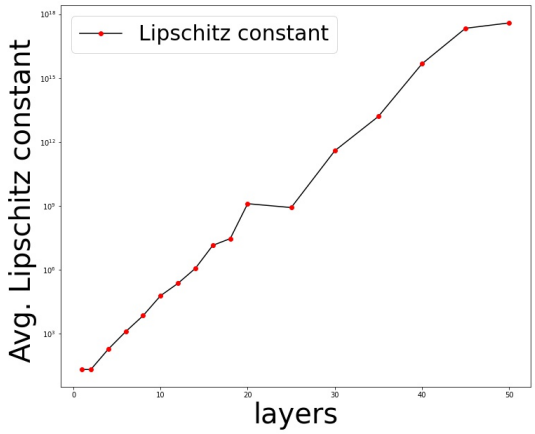

Fig. 9: Lipschitz constant for the architecture 4

IV. Conclusion and further works

In this project, we implement a grid method to generate some CPWL functions and investigate some statistics related to the composed CPWL functions, especially the number of linear regions and the Lipschitz constant. For the number of linear regions, we find that this quantity grows slowly or stops to increase after a certain number of layers which depends on the particular architecture of the composed CPWL function. For the Lipschitz constant, it has different behaviors based on the architecture of the composed CPWL function. It can either converge to 0 or increase exponentially.

For further works, we could look into the average norm of the gradient of a composed CPWL function which can better represent the overall level of the gradient, rather than the Lipschitz constant. Furthermore, we could try to find a systematic method to find the scaling factor for which the Lipschitz constant starts to increase exponentially for a specific function. This could help us understand better the characteristic of a composed CPWL function.

REFERENCES

[1] Geoffrey Hinton et al. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: 25 (2012). Ed. by F. Pereira et al. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[4] Guido Montúfar et al. “On the Number of Linear Regions of Deep Neural Networks”. In: (2014). arXiv: 1402.1869 [stat.ML].

[5] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. “Bounding and Counting Linear Regions of Deep Neural Networks”. In: (2018). arXiv: 1711.02114 [cs.LG].

[6] Shuning Wang and Xusheng Sun. “Generalization of hinging hyperplanes”. In: *IEEE Transactions on Information Theory* 51.12 (2005), pp. 4425–4431. DOI: 10.1109/TIT.2005.859246.