Comparative Analysis of the Linear Regions in ReLU and LeakyReLU Networks

Xuan Qi, Yi Wei, Xue Mei(✉), Ryad Chellali, and Shipin Yang

Nanjing Tech University, Nanjing 211816, China
{mx,spyang}@njtech.edu.cn, rchellali@hotmail.fr

Abstract. Networks with piecewise linear activation functions partition the input space into numerous linear regions. As such, the number of linear regions can serve as a metric to quantify the expressive capacity of networks employing ReLU (Rectified Linear Unit) and LeakyReLU activations. One notable drawback of the ReLU network lies in the potential occurrence of the "dying ReLU" issue during training, whereby the output and gradient remain zero when the input to a ReLU layer is negative. This results in ineffective weight updates and renders the affected neurons unresponsive, consequently impeding their contribution to network training. In this study, we perform statistical analysis on the actual number of linear regions expressed by ReLU and LeakyReLU networks, providing an intuitive explanation for the "dying ReLU" problem. Our findings indicate that, under consistent input distributions and network parameters, LeakyReLU networks generally exhibit stronger expressive capacity in terms of linear regions compared to ReLU networks. We hope that our research can provide inspiration for the design of activation functions and contribute to the exploration and analysis of the behaviors exhibited by piecewise linear activation functions in networks.

Keywords: Deep Networks · Linear regions · Piecewise Linear Activation Functions

1 Introduction

Activation functions are a fundamental component of neural networks, responsible for introducing non-linearity and enabling models to learn complex relationships in data. Among the commonly used activation functions, LeakyReLU (Leaky Rectified Linear Unit) [14] and ReLU (Rectified Linear Unit) [7,16] have gained significant popularity due to their simplicity and effectiveness. Network models employing the ReLU activation function, such as CFDM [6], CondenseNetV2 [21], and FATTN [4], as well as those utilizing the LeakyReLU activation function, such as EffNet [5], Scaled-YOLOv4 [18], and YOLObile [2], have achieved remarkable advancements in the field of deep learning tasks. ReLU maps negative inputs to zero and keeps positive inputs unchanged. By introducing a non-linearity, ReLU allows neural networks to learn complex patterns

B. Luo et al. (Eds.): ICONIP 2023, CCIS 1962, pp. 528–539, 2024.
https://doi.org/10.1007/978-981-99-8132-8_40

and improves their expressive power. Additionally, ReLU promotes sparsity in network activations, making computations more efficient. However, ReLU suffers from a limitation known as the "dying ReLU" problem, where neurons can become stuck at zero and cease to learn. LeakyReLU addresses the "dying ReLU" problem by modifying the activation function to allow a small non-zero output for negative inputs. Instead of mapping negative values to zero, LeakyReLU introduces a small slope for negative inputs, typically a small constant. This small slope ensures that neurons receive a gradient even for negative values, preventing them from becoming inactive. By providing a non-zero output for negative inputs, LeakyReLU promotes continuous learning and improves the stability and robustness of neural networks. Figure 1 illustrates the network unit employed in this research endeavor.

Networks harness the efficacy of piecewise linear activation functions, such as ReLU and LeakyReLU, in order to fit an extensive array of distinct linear functions. Particularly, when dealing with standard deep neural networks that employ piecewise linear activation functions, we are capable of effectuating the transformation of the input space into a diverse set of linear convex regions [9,17]. Regarding multi-classification tasks, the ultimate classification outcome relies on the hierarchical arrangement of linear functions within the networks. Figure 2 visually shows the visualization results of the linear regions in a two-dimensional input space, obtained through the multi-classification of ReLU and LeakyReLU networks. Notably, the different color blocks featured in the figure represent the distinctive linear regions formed by linear functions within the networks. Moreover, each neuron within the networks possesses the ability to partition the input space into two regions along a hyperplane. Through the collective influence of numerous neurons, a larger region can be effectively segmented into smaller, more refined subregions. Consequently, each region encompasses linear functions that symbolize the ultimate outcome of multi-classification, with the predicted label determined as the maximum value among the results of the linear functions. In ReLU and LeakyReLU networks, the neurons in subsequent layers further partition the linear regions previously established by preceding layers. As a result, the linear functions are organized in a layered fashion, leading to the generation of a substantial multitude of linear regions during network operation. Figure 3 illustrates the arrangement of internal linear convex regions within the ReLU and LeakyReLU network.

In this paper, our primary objective is to transform networks into mappings within the input space. We endeavor to explain the phenomenon of "dying ReLU" from the perspective of piecewise linear functions within the networks. Furthermore, we conduct a comparative analysis of the linear regions in both ReLU and LeakyReLU networks. The primary contributions of our study can be summarized as follows:

(1) We elucidate the evolutionary process of the linear regions in ReLU and Leaky-ReLU networks based on two-dimensional inputs.
(2) We show visualizations of the linear regions at different epochs and layer-wise in ReLU and LeakyReLU networks.

(3) Under consistent input distributions and network parameters, LeakyReLU networks generally exhibit stronger expressive capacity in terms of linear regions compared to ReLU networks. This phenomenon primarily arises from the "dying ReLU" issue, which reduces the number of piecewise linear functions within ReLU networks.

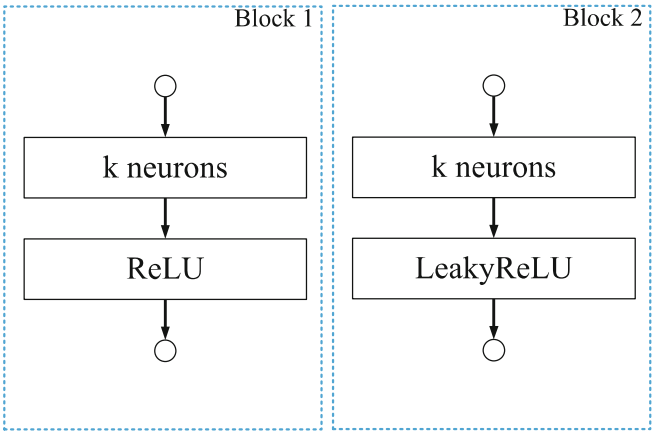

Fig. 1. The minimum network units used in this paper. (a) Block 1, a network with k neurons activated by ReLU. (b) Block 2, a network with k neurons activated by LeakyReLU.

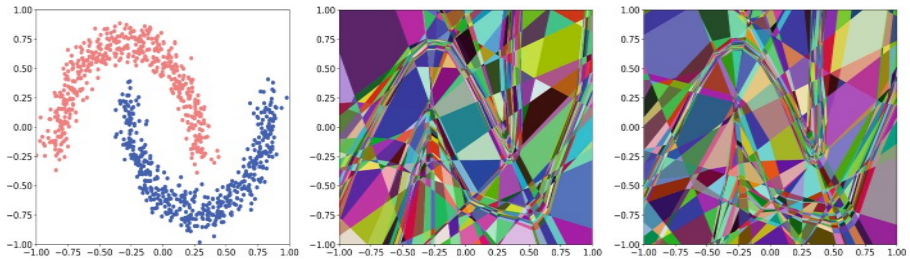

Fig. 2. The figure shows the relationship between the linear regions and input distributions of the networks under various activation states, all of which are mapped in the two-dimensional input space. Each distinct color block represents a different linear region of the networks. (a) The two-dimensional input data set *make moons*. (b) The linear regions obtained from a network utilizing architecture Fig. 3 (a) trained for 50 epochs, with each layer incorporating a Block 1 (k = 32). (c) The linear regions obtained from a network utilizing architecture Fig. 3 (a) trained for 50 epochs, with each layer incorporating a Block 2 (k = 32).

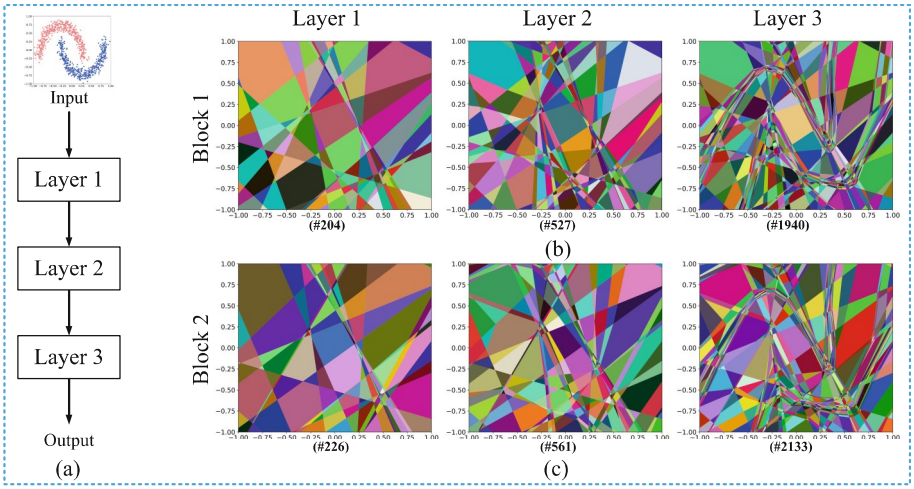

Fig. 3. During the training process with dataset *make moons* (1000 samples), we empirically observe the evolution process of linear regions and count the number of linear regions in each layer of different networks. (a) The network consists of three fully connected layers, each layer containing a minimum unit Block 1 or Block 2. (b) The evolution process of linear regions in each layer of the three-layer Block 1 network during training for 100 epochs (k = 32). (c) The evolution process of linear regions in each layer of the three-layer Block 2 network during training for 100 epochs (k = 32).

2 Related Work

In recent years, a group of scholars has made noteworthy contributions to the examination of linear regions and the operational mechanisms of networks. The research conducted in [9] presented an analysis of the correlation between the number of neurons and the count of linear regions in networks utilizing piecewise linear activation functions. Moreover, [10] established an upper bound on the number of activation regions of ReLU networks. The impact of network depth, width, and activation complexity on the quantity of linear regions was explored in [8]. Additionally, [3,12,15,19,20] investigated both upper and lower bounds of the number of linear regions of ReLU and LeakyReLU networks. By partially solving the algebraic topology problem based on ReLU networks, [11] provided an approximate analysis of how initialization affects the number of linear regions. Furthermore, [1] proposed a training approach for ReLU networks that enables the convergence of parameters to the global optimum. Notably, SplineCam [13] introduced a method for accurately calculating the geometric shape of networks.

3 Linear Regions of Networks

The ReLU [11] activation function is defined as follows:

$$ReLU(x) = \max(0, x), \tag{1}$$

when the input x is greater than 0, the output of ReLU function is x. Conversely, when x is less than or equal to 0, the output is 0 and the corresponding node is inactive and does not contribute to the output. During gradient back propagation, the derivative of ReLU is:

$$ReLU'(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise .} \end{cases} \tag{2}$$

LeakyReLU differs from ReLU by assigning a non-zero slope to all negative values. Specifically, it divides the negative portion by a value greater than 1. We define the LeakyReLU function as follows:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x >= 0, \\ x/a, & \text{if } x < 0, \end{cases} \tag{3}$$

where a is a constant greater than 1, representing the slope of the function for negative values. When the input x is greater than or equal to 0, the output of LeakyReLU function is x. Conversely, when x is less than 0, the output is x/a. During gradient back propagation, the derivative of LeakyReLU is:

$$LeakyReLU'(x) = \begin{cases} 1, & \text{if } x >= 0, \\ 1/a, & \text{otherwise .} \end{cases} \tag{4}$$

Under the activation of ReLU and LeakyReLU, let us consider a network with Z activation layers, assuming H^z is the number of neurons with z layers, where z is the number of activation layers. Therefore, the output of each H^z is

$$o^z = \left\{ o_h^z, h \in H^z \right\}, \tag{5}$$

$$o_h^z = \left(\mathbf{w}_h^z\right)^T \hat{o}^{z-1} + \mathbf{b}^z, \quad h \in H^z, z \leq Z, \tag{6}$$

where $\mathbf{b}^z$ signifies the bias term pertaining to layer z, and $\mathbf{w}_h^z$ represents the weight of neurons h within layer z. Therefore, networks with ReLU and LeakyReLU activation can be seen as a connected piecewise linear function. As a result, every neuron in a network can be represented as a piecewise linear function of the input $x \in \mathbb{R}^d$:

$$f_h^z(x) = \left(\mathbf{S}_h^z\right)^T x + \mathbf{b}_h^z, \quad h \in H^z, \tag{7}$$

where d is the dimension of the input, and the linear weight and bias of the current neuron input are denoted by $\mathbf{S}_h^z$ and $\mathbf{b}_h^z$, respectively.

Thus, each linear inequality arrangement in ReLU and LeakyReLU networks forms convex regions. We define the linear regions of networks with ReLU and LeakyReLU activation as follows:

Definition 1 (adapted from [10]) Let Q be a network with input dimension d_{inp} and fix τ, a vector of trainable parameters for Q. Define

$$K_Q(\tau) := \left\{ x \in \mathbb{R}^{d_{inp}} \mid \nabla Q(\cdot; \tau) \text{ is discontinuous at } x \right\}. \tag{8}$$

The linear regions of Q at τ are the connected parts of input without K_Q:

$$\text{linear regions } (Q, \tau) = \left\{ x \mid x \in \mathbb{R}^{d_{inp}}, x \notin K_Q(\tau) \right\}. \tag{9}$$

4 "Dying ReLU" Based on Piecewise Linear Functions

Theorem 1. *Given that A is a ReLU network and D is a LeakyReLU network, under the conditions of consistent input distribution, initialization, network structure, and other parameters, the total number of continuous and piecewise-linear (CPWL) functions generated during training for A and D can be denoted as follows:*

$$\mathbb{E}[\#\{\ CPWL\ functions\ in\ \ A\ \}] = \beta(1), \tag{10}$$

$$\mathbb{E}[\#\{\ CPWL\ functions\ in\ \ D\ \}] = \gamma(1). \tag{11}$$

Then, the relationship between the number of CPWL functions generated during training for networks A and D can be expressed as follows:

$$|\beta(1)| \leq |\gamma(1)|. \tag{12}$$

This theorem implies that under consistent input distributions and network parameters, LeakyReLU networks generally exhibit stronger expressive capacity in terms of $CPWL$ functions compared to ReLU networks. The main reason of this phenomenon is the inability of the neurons in the ReLU layer to update their weights when encountering negative inputs, which is known as the "dying ReLU" problem. We provide the proof process for Theorem 1 as follows:

Proof. Let C represent the loss function of the network, and let $x_i^{(j)}$ denote the output of the i^{th} neuron in the j^{th} layer. Suppose $f(v) = \max(0, v)$ is a ReLU neuron, and $v_i^{(j)}$ is the linear input to the $(j+1)^{th}$ layer. Then, according to the chain rule, the derivative of the loss with respect to the weight $w_i^{(j)}$ connecting the j^{th} and $(j+1)^{th}$ layers is given by:

$$\frac{\partial C}{\partial w_i^{(j)}} = \frac{\partial C}{\partial x_i^{(j+1)}} \frac{\partial x_i^{(j+1)}}{\partial w_i^{(j)}}. \tag{13}$$

Referring to (13), the first term on the right-hand side can be recursively calculated. The second term on the right-hand side is the only part directly related to $w_i^{(j)}$ and can be decomposed into:

$$\begin{aligned} \frac{\partial x_i^{(j+1)}}{\partial w_i^{(j)}} &= \frac{\partial f\left(v_i^{(j)}\right)}{\partial v_i^{(j)}} \frac{\partial v_i^{(j)}}{\partial w_i^{(j)}} \\ &= f'\left(v_i^{(j)}\right) x_i^{(j)}. \end{aligned} \tag{14}$$

Therefore, in the case of ReLU networks, if the output of the previous layer within the network is consistently negative, the weights of the neuron will not be updated, rendering the neuron non-contributory to the learning process.

Referring to (7), assuming that the number of neurons in the $(j+1)^{th}$ layer of the ReLU network whose weights $\mathbf{S}_h^z$ are not updated is denoted as M, it follows that there are M neurons in the $(j+1)^{th}$ layer of the ReLU network

that are unable to form new $CPWL$ functions. We can express the expected total number of $CPWL$ functions generated in the $(j+1)^{th}$ layer of the ReLU network as follows:

$$\mathbb{E}[\#\{\ CPWL \text{ functions in the } (j+1)^{th} \text{ layer with ReLU}\}] = \lambda(1). \tag{15}$$

However, LeakyReLU does not suffer from the issue of weights not being updated. Under the condition of consistent parameters, if we replace the activation function between the j^{th} and $(j+1)^{th}$ layers of the network with LeakyReLU, the expected total number of $CPWL$ functions generated in the $(j+1)^{th}$ layer of the LeakyReLU network can be expressed as follows:

$$\mathbb{E}[\#\{\ CPWL \text{ functions in the } (j+1)^{th} \text{ layer with LeakyReLU}\}] = \eta(1). \tag{16}$$

Therefore, without considering the generation of duplicate $CPWL$ functions in the network and the special case where the j^{th} layer of the network has no negative output, we typically have the following expression:

$$|\lambda(1)| \leq |\eta(1)|. \tag{17}$$

Referring to (10) and (11), assuming that A is a ReLU network and D is a LeakyReLU network, under the conditions of consistent input distribution, initialization, network structure, and other parameters, we easily get (12).

5 Experiments

We employed the PyTorch framework for our experiments. Each neuron in the network is represented as a linear function based on the input. The linear function of each neuron can be obtained through forward propagation. The datasets used in the experiments consist of two-dimensional data, as two-dimensional input data is easier to visualize. We used three types of two-dimensional datasets. The first dataset consists of 500 samples uniformly generated within the range of −1 to 1, with randomly generated labels dividing the data into two classes: 0 and 1. The second dataset is a binary classification dataset *make moons* with 1000 samples. The third dataset is a five-class classification dataset *make gaussian quantiles* with 1000 samples. Figure 4 illustrates the distribution of the three input data types in the network. Regarding the network parameters and configurations, we utilized network structures with different numbers of Blocks, as shown in Fig. 1, to examine the expressive capacity of the linear regions for both ReLU and LeakyReLU networks. Additionally, we employed the Adam optimizer, set the batch size to 32, used the Cross Entropy loss function, set the learning rate to 0.001, and set the value of a in (3) to 100.

Figure 5, Fig. 6, and Fig. 7 present the number of linear regions and the visualization results in ReLU and LeakyReLU networks trained on the three types of two-dimensional input samples for different epochs. Through these figures, we can visually observe a significant difference in the expressive capacity of linear regions between ReLU and LeakyReLU networks, under the same external

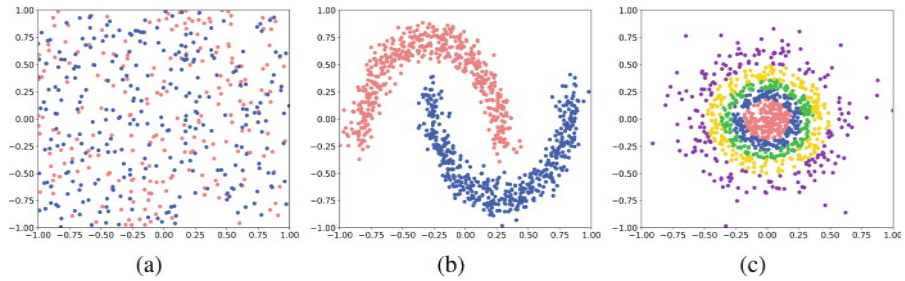

Fig. 4. (a) Two-dimensional input dataset consists of 500 samples are randomly generated within the range of -1 to +1, with randomly assigned labels for two classes. (b) Two-dimensional binary dataset *make moons* comprising 1000 samples. (c) Two-dimensional five-class dataset *make gaussian quantiles* comprising 1000 samples.

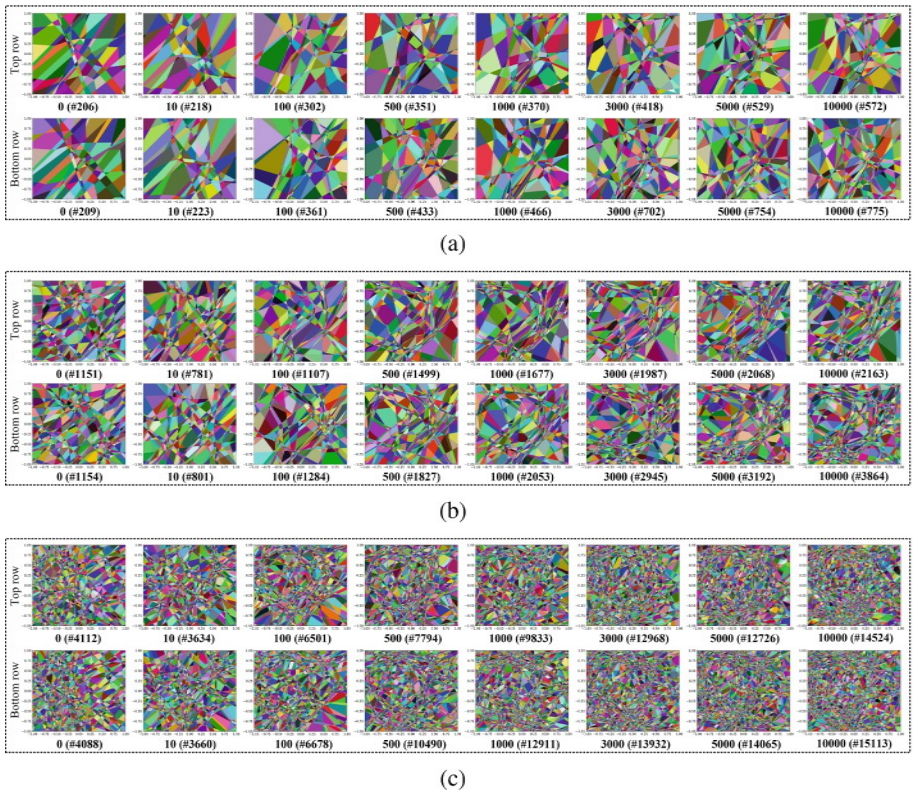

Fig. 5. The number of linear regions and the visualization results of the networks trained for r epochs with input Fig. 4 (a), for $r = 0, 10, 100, 500, 1000, 3000, 5000, 10000$. Top row (with ReLU): The network utilizes three fully connected Block 1. Bottom row (with LeakyReLU): The network employs three fully connected Block 2. (a) k = 16. (b) k = 32. (c) k = 64.

conditions. Based on our analysis of the number of linear regions and the visualization results, considering the same input distribution, number of neurons, and training parameters, in the majority of cases (while acknowledging the possibility of rare occurrences of $CPWL$ function duplication and linear region repetition during network training), LeakyReLU networks demonstrate a stronger ability to express linear regions compared to ReLU networks.

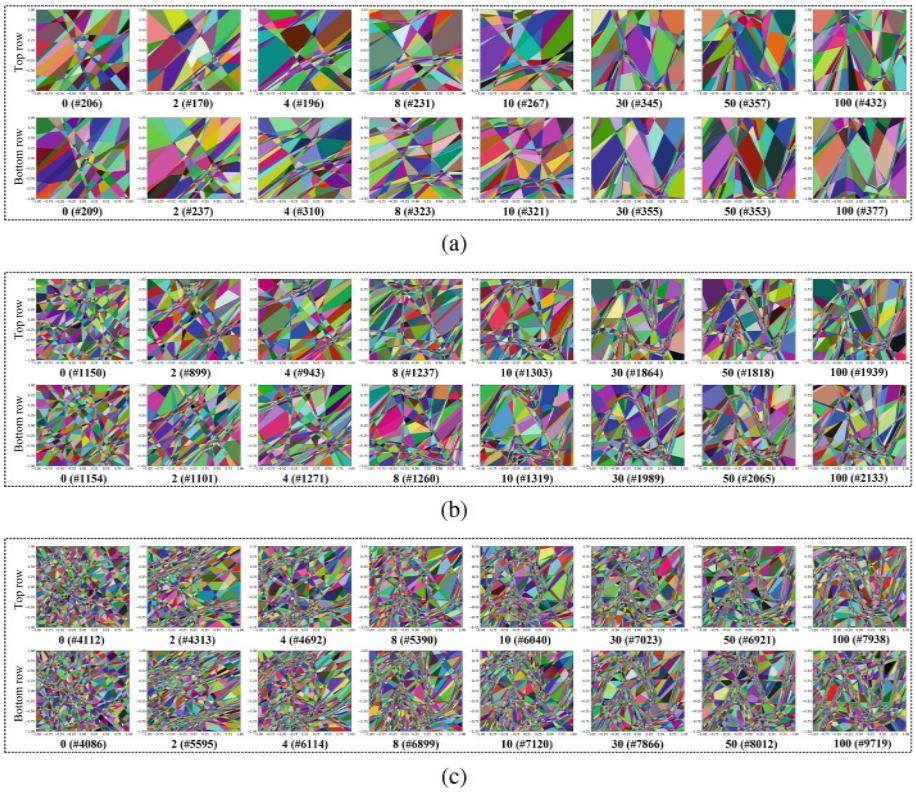

Fig. 6. The number of linear regions and the visualization results of the networks trained for r epochs with input Fig. 4 (b), for $r = 0, 2, 4, 8, 10, 30, 50, 100$. Top row (with ReLU): The network utilizes three fully connected Block 1. Bottom row (with LeakyReLU): The network employs three fully connected Block 2. (a) k = 16. (b) k = 32. (c) k = 64.

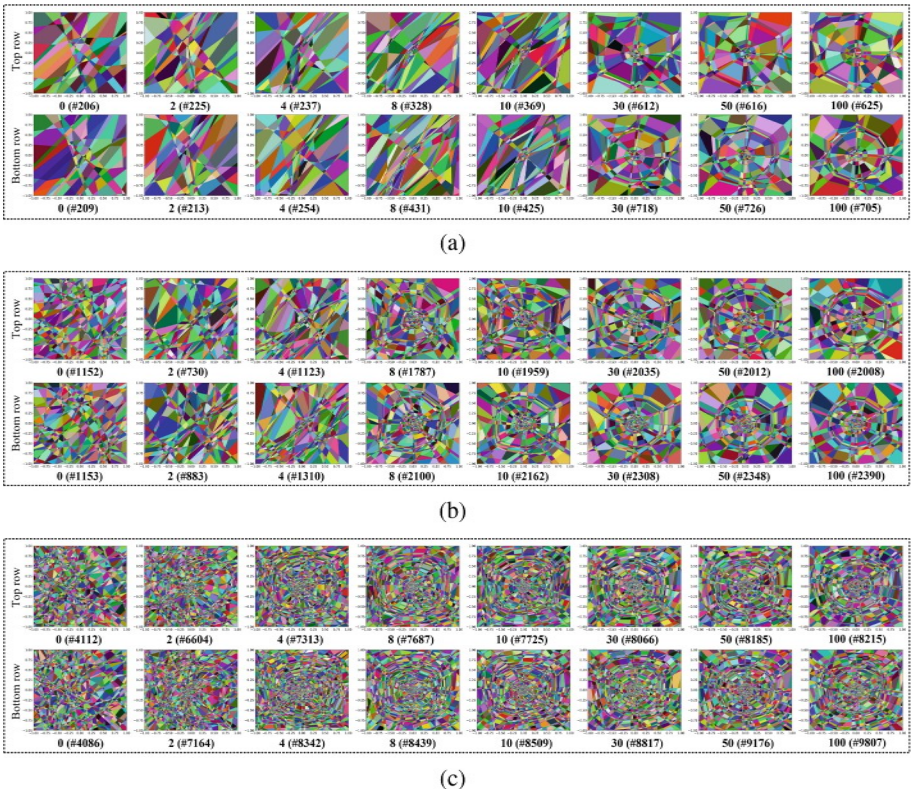

Fig. 7. The number of linear regions and the visualization results of the networks trained for r epochs with input Fig. 4 (c), for $r = 0, 2, 4, 8, 10, 30, 50, 100$. Top row (with ReLU): The network utilizes three fully connected Block 1. Bottom row (with LeakyReLU): The network employs three fully connected Block 2. (a) k = 16. (b) k = 32. (c) k = 64.

6 Conclusion

The actual expressive capacity of most $CPWL$ networks is still not well understood. In this study, we investigated a key characteristic of ReLU and LeakyReLU networks, which is their ability to express the number of linear regions. This serves as a representative measure of the complexity of functions they can effectively learn.

We conducted a series of experiments to explore how the linear regions of ReLU and LeakyReLU networks evolve during the training process. This included analyzing the evolution of linear regions layer by layer in the network and examining the evolution of linear regions across different epochs. Furthermore, we provided an intuitive explanation of the "dying ReLU" problem based on the expressive capacity of $CPWL$ functions within the network. Specifically, we found that under the same conditions of network input, network structures,

and training parameters, LeakyReLU networks generally exhibit a stronger ability to express linear regions compared to ReLU networks.

We hope that our research can provide inspiration for the design of activation functions and contribute to the exploration and analysis of the behavior exhibited by $CPWL$ functions within networks.

Acknowledgements. This work was supported by the National Natural Science Foundation of China (Grant No. 61973334).

References

1. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. arXiv preprint arXiv:1611.01491 (2016)
2. Cai, Y., et al.: Yolobile: real-time object detection on mobile devices via compression-compilation co-design. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 955–963 (2021)
3. Chen, H., Wang, Y.G., Xiong, H.: Lower and upper bounds for numbers of linear regions of graph convolutional networks. arXiv preprint arXiv:2206.00228 (2022)
4. Chen, P., Zhuang, B., Shen, C.: FATNN: fast and accurate ternary neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5219–5228 (2021)
5. Freeman, I., Roese-Koerner, L., Kummert, A.: EffNet: an efficient structure for convolutional neural networks. In: 2018 25th IEEE International Conference on Image Processing (ICIP), pp. 6–10. IEEE (2018)
6. Gao, Z., Chen, X., Xu, J., Yu, R., Zong, J.: A comprehensive vision-based model for commercial truck driver fatigue detection. In: Tanveer, M., Agarwal, S., Ozawa, S., Ekbal, A., Jatowt, A. (eds.) Neural Information Processing. ICONIP 2022. LNCS, vol. 13625, pp. 182–193. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30111-7_16
7. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323. JMLR Workshop and Conference Proceedings (2011)
8. Goujon, A., Etemadi, A., Unser, M.: The role of depth, width, and activation complexity in the number of linear regions of neural networks. arXiv preprint arXiv:2206.08615 (2022)
9. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: International Conference on Machine Learning, pp. 2596–2604. PMLR (2019)
10. Hanin, B., Rolnick, D.: Deep ReLU networks have surprisingly few activation patterns. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
11. Hinz, P.: Using activation histograms to bound the number of affine regions in ReLU feed-forward neural networks. arXiv preprint arXiv:2103.17174 (2021)
12. Hu, Q., Zhang, H., Gao, F., Xing, C., An, J.: Analysis on the number of linear regions of piecewise linear neural networks. IEEE Trans. Neural Netw. Learn. Syst. **33**(2), 644–653 (2020)
13. Humayun, A.I., Balestriero, R., Balakrishnan, G., Baraniuk, R.G.: SplineCam: exact visualization and characterization of deep network geometry and decision boundaries. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3789–3798 (2023)