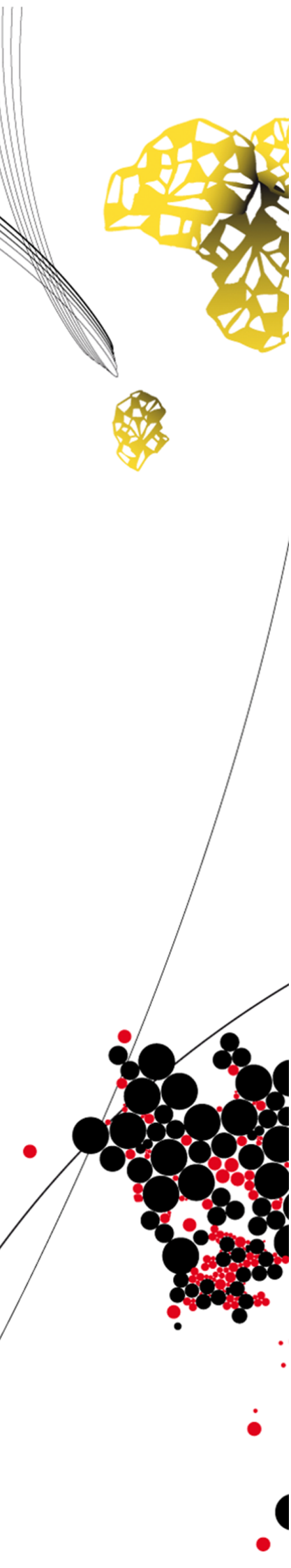


UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science

Design and Impact of Activation Functions for Sparse Neural Networks

Xuhao Zhang

M.Sc. Thesis
January 2023

Supervisors:
dr. M. POEL
dr.ir. D.C. MOCANU
dr. H. HANG

Data Management Biometrics Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Acknowledgements

I would like to express my sincerest appreciation to Professors Decebal, Mannes, and Hanyuan for their invaluable guidance and unwavering support throughout the writing of this thesis. Their expert insights and critiques have greatly contributed to the success of this work. I am also deeply grateful to Zahra for her constant assistance and support in my academic pursuits, her guidance and some of her previous works were very helpful when I was out of ideas. I would also like to thank Selima for building on her work so that I could further develop my thesis. My sincerest thanks also go to my parents and friends for their constant encouragement and support throughout this journey. I will forever be grateful for the love and support that they have shown me. Lastly, I would like to express my gratitude to Utwente University for providing me with an enriching and memorable learning experience. The knowledge and skills I have gained during my time here will stay with me throughout my life.

Summary

Inspired by biological neurons, sparse neural networks were very early proposed, meaning that the connections between neural networks should be sparse. In recent years, more and more people have begun to study sparse neural networks. Most researchers aim to train sparse neural networks directly, rather than training a dense neural network and then pruning it. There has been a lot of excellent research, but most of it has focused on training strategies, network structure, etc. Very few studies have focused on the activation function in sparse neural networks. In this paper, we propose a new activation function, GAReLU, which has a learnable parameter slope in the negative interval. We address three research questions:

- What problems do traditional activation functions, such as ReLU, face in sparse neural networks?
- How should we evaluate the activation function of a sparse neural network?
- How does GAReLU compare with other activation functions?

To answer these research questions, we first introduce background knowledge activation functions, and multilayer perceptrons. We then introduce relevant research in the field of sparse neural networks and choose Sparse Evolutionary Training as the basis for this thesis. We also introduce the activation functions commonly used in sparse neural networks.

To investigate the problems faced by traditional ReLU in sparse neural networks, we analyze ReLU sparsity and the effect of dying ReLU on SET from a theoretical perspective and find that ReLU affects the performance of sparse neural networks under certain circumstances.

To evaluate the effect of different activation functions on sparse neural networks, we are inspired by previous work to measure the activation function in terms of effective gradient flow.

Finally, we conduct experiments to test the performance of GAReLU and three other activation functions on sparse neural networks. Our model is a multilayer perceptron based on SET. We train the model on different datasets and analyze the accuracy and effective gradient flow. For the same dataset, we vary the model

parameters to change the sparsity of the network to analyze the change in accuracy. We also iterate the initial values of the slope to verify the learning ability of GAReLU for the parameter slope.

Through theoretical analysis and experiments, this thesis discusses the problems faced by traditional activation functions in sparse neural networks, how to evaluate activation functions, and proposes a new and excellent activation function.

Contents

Acknowledgements iii

Summary v

List of acronyms ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 New activation functions for SNN . . . 2
- 1.3 Research questions . . . 4
- 1.4 Report organization . . . 4

2 Background 7
- 2.1 Activation function . . . 7
  - 2.1.1 Rectified Units . . . 8
- 2.2 Multilayer Perceptron . . . 9

3 Related works 11
- 3.1 Sparse neural networks and sparsification . . . 11
- 3.2 Activation functions in sparse neural networks . . . 13

4 Dying ReLU problem 15

5 Proposed methodology 19
- 5.1 Gradient flow . . . 19
- 5.2 Experimental design . . . 20

6 Experiments 21
- 6.1 Experimental environment . . . 21
- 6.2 Structure and parameter settings . . . 22
- 6.3 Training strategies . . . 23
- 6.4 Datasets . . . 24

7 Results **25**
7.1 Performance on different datasets 25
7.2 Performance at different sparsity levels 28
7.3 Performance of different slopes on the activation function 28

8 Discussion **31**

9 Conclusion **33**

References **35**

Appendices

A AppendiA **39**
A.1 Proposed algorithm 39

List of acronyms

DNN Dense Neural Networks

SNN Sparse Neural Networks

ReLU Rectified Linear Unit

GAReLU Gradient-based Adaptive Rectified Linear Unit

SReLU S-shaped rectified linear activation unit

PReLU Parametric Rectified Linear Unit

Leaky ReLU Leaky Rectified Linear Unit

AII-ReLU Alternated left Rectified Linear Unit

SET Sparse Evolutionary Training

DSR Dynamic Sparse Reparameterization

CSR Compressed Sparse Row

EGF Effective Gradient Flow

MLP Multilayer Perceptron

SGD Stochastic Gradient Descent

Chapter 1

Introduction

1.1 Motivation

In a variety of domains [1], artificial neural networks have produced astonishing outcomes. Deep learning, based on artificial neural networks, has been successful in various applications and has also been shown to solve various artificial intelligence issues. However, training today's advanced networks typically requires large amounts of data and computational resources, and the number of model parameters is growing [2].

Network models are becoming increasingly large to learn on complex datasets with millions to billions of examples and features. For example, the BERT model is trained on two massive datasets - BookCorpus and English Wikipedia [3]. These models require extensive matrix operations to train millions of parameters. Deep learning is becoming increasingly expensive with the huge storage and computing power demand. The cost of deep learning presents several challenges for the AI community, including a large carbon footprint and the commercialization of AI research. As demand for AI capabilities on cloud servers and "edge devices" continues to increase, so does the need for cost-effective neural networks. Researchers want to find a way to reduce training costs while maintaining network performance. Some research has focused on model compression and there are many ways to reduce the computational cost of the network, such as pruning [4], hash randomization [5], and sparse training [6].

There are similarities between sparse training and pruning. Both methods end up with a Sparse Neural Networks (SNN), but the difference is that pruning techniques require training a Dense Neural Networks (DNN) before pruning, while sparse training can be trained from scratch.

Here we focus on the sparse training. Compared to the DNN, the SNN has fewer connections, significantly reducing the computational cost of forward and backward propagation while maintaining performance comparable to dense networks. How-

ever, current research on SNN is based on the experience of dense networks [7], and most of the research focuses on the network structure and training method, with little research on the activation function of SNN. The selection of activation function has a crucial impact on training deep neural networks. An appropriate activation function must be selected to match the SNN. Unfortunately, most of the current activation functions for SNN directly use the activation function of the DNN.

There is increasing evidence from non-speech deep learning research that sigmoidal nonlinearities may not be optimal for DNN. [8] found that DNN with rectifier nonlinearities in place of traditional sigmoids perform much better on image recognition and text classification tasks. Therefore, in the dense network, Rectified Linear Unit (ReLU) activation function has been the mainstream activation function for a long time [9] [10], mainly due to its excellent properties such as non-saturation and sparsity. However, it also has the severe consequence of neuron death. When we apply ReLU to SNN, we do not want the activation function to change the sparsity of the network or cause neurons to die, which could affect the performance of the network. Unlike dense neural networks, death neurons in SNN can be activated again by reassigning weights, which may significantly affect the efficiency of SNN. Therefore, we want to change the activation function so that it can keep all information rather than neglecting weights less than 0 as in ReLU. Then we propose a new activation function Gradient-based Adaptive Rectified Linear Unit (GAReLU), more details of which we will discuss in 1.2

1.2 New activation functions for SNN

As one of the core components of artificial neural networks, activation functions are continually evolving and improving. However, current research in this field primarily focuses on dense neural networks. Research on activation functions for sparse neural networks has been less successful and requires further exploration. This thesis aims to study the activation function of sparse neural networks and open new avenues for research.

However, as sparse neural networks continue to develop, traditional ReLU activation functions no longer suffice. Through experimentation, it was found that the S-shaped rectified linear activation unit (SReLU) [11] activation function designed for convolutional networks performed better than ReLU in sparse neural networks, but SReLU introduces four parameters that need to be learned. Subsequently, Alternated left Rectified Linear Unit (All-ReLU) [12] was developed, which fixed parameters to reduce overhead and accelerate convergence by breaking symmetry during training. PReLU provides an opportunity to learn parameters during training. Based on these prior studies, we propose a novel activation function for sparse

neural networks called GAReLU.

Definition Given an ANN with N layers, GAReLU is defined as follows for each layer l:

$$f_l(x) = \begin{cases} -\alpha_l x & x \leq 0 \ \& \ l\%2 == 0 \\ \alpha_l x & x \leq 0 \ \& \ l\%2 == 1 \\ x & x > 0 \end{cases} \tag{1.1}$$

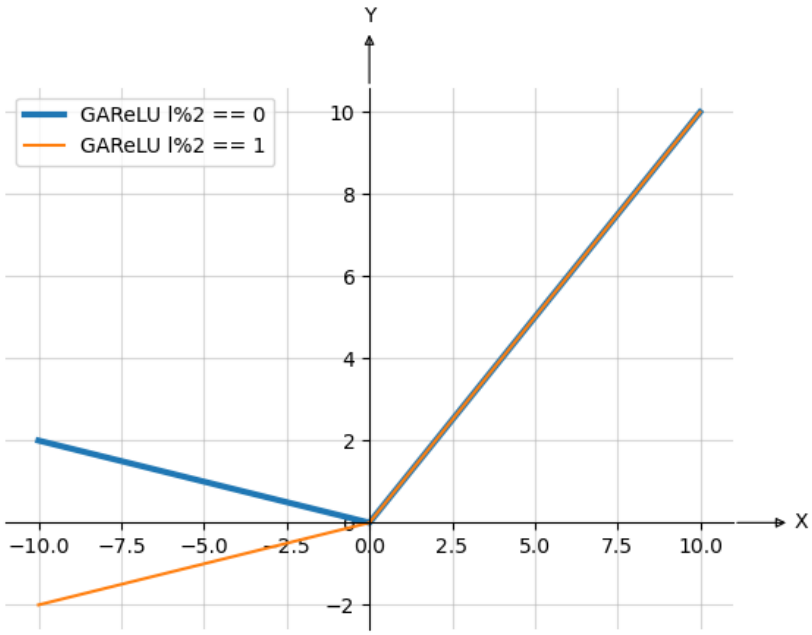


Figure 1.1: An illustration of GAReLU, there are two lines in the figure, $slope = 0.2$ for odd layers and $slope = -0.2$ for even layers

The input to the l-th layer nonlinear activation function, f_l, is x. The slope, α_l, of the negative side is determined by the gradient value of each network layer. The modulo operation is represented by %. When $\alpha_l = 0$, the activation function becomes ReLU. When α_l is fixed, it becomes All-ReLU. If the modulo operation is removed, it becomes PReLU. Additionally, by removing the modulo operation and fixing α_l, it can also become Leaky ReLU, as described in [13].

Optimization GAReLU, which can be trained using backpropagation as described in [14], modifies the slope of each layer once per round and calculates updates using the chain rule. The gradient of α_l for a single layer is as follows:

$$\frac{\partial L}{\partial \alpha_l} = \sum_i \sum_x \frac{\partial L}{\partial f_l(x)} \frac{\partial f_l(x)}{\partial \alpha_l} \tag{1.2}$$

where L represents the objective function like loss function. The total of a neuron's inputs is denoted by $\sum_{x_i}$. And $\sum_i$ adds up all of the neuron's in the layer. The gradient of the activation is given by:

$$\frac{\partial f_l(x)}{\partial \alpha_l} = \begin{cases} -x, & \text{if } x \leq 0 \ \& \ l\%2 == 0 \\ x, & \text{if } x \leq 0 \ \& \ l\%2 == 1 \\ 0, & \text{if } x > 0 \end{cases} \tag{1.3}$$

Finally, we use momentum to update α_l:

$$\alpha_l = \mu \alpha_l + \eta \frac{\partial L}{\partial \alpha_l} \tag{1.4}$$

where μ is the momentum and η is the learning rate.

From the formula, it is evident that the same slope is employed for each layer. This results in the addition of only one additional parameter per layer, thereby reducing the likelihood of overfitting. The capability to vary the slope across different layers enhances the adaptability of the function. Additionally, the use of the mode operation allows for breaking the symmetry of the activation function while preserving a better gradient flow. In summary, GAReLU can effectively regulate the parameters in conjunction with the entire model, thereby preventing overfitting and improving the gradient flow.

1.3 Research questions

Based on the situation described above, the following research questions have been addressed in this thesis:

1) What problems do traditional activation function ReLU face in sparse neural networks?

2) How should we evaluate the activation function of a sparse neural network?

3) How does GAReLU compare with other activation functions?

1.4 Report organization

- Chapter 1, introduces the motivation of the research, define the new activation function GAReLU and present the research questions of the thesis.

- Chapter 2, presents background information: the activation function and the Multilayer Perceptron (MLP)
- Chapter 3, introduces related work on sparse neural networks, outlines the strengths and weaknesses of previous work, and leads to the direction of this thesis.
- Chapter 4, describes how to answer the research question 1, theoretical analysis of the problems faced by ReLU in sparse neural networks.
- Chapter 5, describes how to answer the remaining research questions, first mentions a new method to evaluate activation function, then introduces what experiments are performed.
- Chapter 6, describes the experimental environment, structure and parameter settings, strategy, and the data set used.
- Chapter 7, presents and analyses the results.
- Chapter 8, deeper interpretation and analysis of the results and discussion of possible limitations.
- Chapter 9, do the conclusion.

Chapter 2

Background

2.1 Activation function

A neural network, mathematically speaking, is used to fit a function. The basic structure of a neuron is depicted in Figure 2.1, where it performs a linear transformation of the input data. The input data is multiplied by the weights, and then summed to obtain the input of the neuron. Following activation through the activation function, the neuron's output is produced.

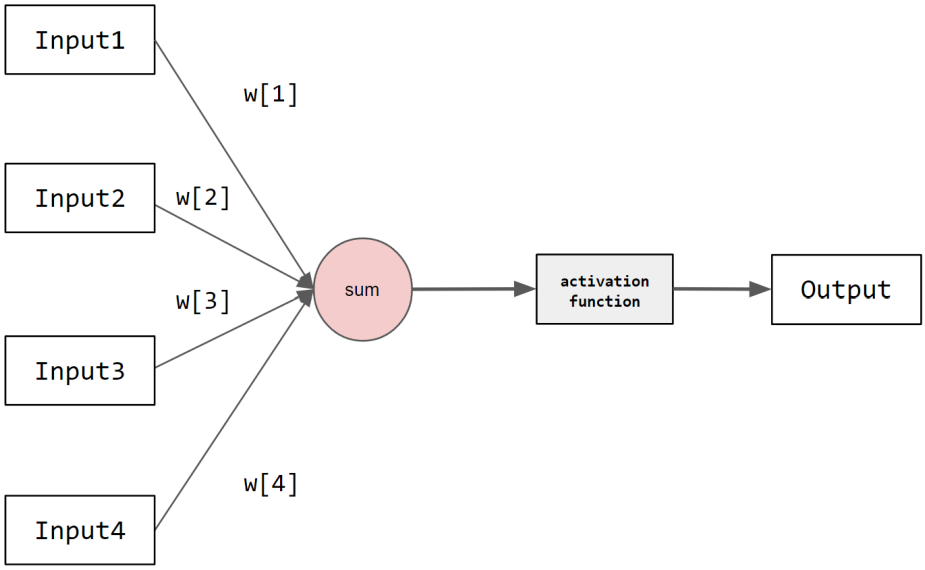


Figure 2.1: The neuron receives its inputs from the preceding layer then it adds up each signal multiplied by its corresponding weight and passes them on to an activation function.

Based on the outputs we obtain, we use them to solve various problems such as classification and regression. However, for different problems, neural networks may require different functions to be fit, including linear and nonlinear functions.

In the simplest scenario, when the neural network is tasked with fitting a linear function, the use of a linear activation function is appropriate. This is due to the fact that fitting a linear function is a simple linear transformation that can be achieved through a Multilayer Perceptron with only one hidden layer. It can be deduced that linear activation functions possess the advantage of being straightforward in their application. However, the linearity of these functions also presents a drawback. The combination of linear functions, regardless of the number of layers employed, ultimately results in a linear function. Thus, the expressiveness of a model utilizing solely linear activation functions is limited.

On the other hand, when the neural network needs to fit a nonlinear function, a specific activation function is necessary. Since linear activation functions only perform linear transformations during forward propagation, their fitting ability for nonlinear functions is poor. Therefore, nonlinear activation functions are used to fit nonlinear functions.

2.1.1 Rectified Units

Because many research papers have chosen ReLU as the activation function in the hidden layer of the neural network, we focus on the functions related to ReLU. In this section, we introduce four kinds of basic rectified units: ReLU [15] [8], Leaky Rectified Linear Unit (Leaky ReLU) [13], Parametric Rectified Linear Unit (PReLU) [16], and SReLU [11]. Our proposed activation function also belongs to the ReLU variants, which is already described in 1.2.

Rectified Linear Unit One of the keys to designing modern deep learning systems is to choose a non-saturated activation function [17]. ReLU is a non-saturated activation function proposed in [15] and is the most commonly used activation function at this stage. The unsaturated activation function has two advantages, first is to solve the problem of gradient disappearance and explosion. The second is to accelerate the convergence speed [18]. ReLU is also chosen as the activation function to compare with SReLU for sparse neural networks in Sparse Evolutionary Training (SET) [6].

The definition of the function of ReLU is as follows.

$$f(x) = \max(0, x) = \begin{cases} 0, & \text{if } \mathrm{x} < 0 \\ x, & \text{if } \mathrm{x} \geq 0 \end{cases} \tag{2.1}$$

ReLU also has the disadvantage of generating the dying ReLU problem when $x < 0$, some neurons can not activate. We suspect this is why ReLU is unsuitable for SNN, and we will explore this in 4.

Leaky Rectified Linear Unit Leaky ReLU is first introduced in [13]. It introduces a new parameter to solve the dying ReLU problem by maintaining a small gradient(slope) α when the input $x < 0$; this allows a non-zero gradient to update the parameter even when the output value of the neuron is negative, avoiding neuron inactivation. The definition of Leaky ReLU is as follows:

$$f(x) = \max(\alpha x, x) = \begin{cases} \alpha x, & \text{if x} < 0 \\ x, & \text{if x} \geq 0 \end{cases} \tag{2.2}$$

Parametric Rectified Linear Unit PReLU distinguishing from the fixed parameters of Leaky ReLU, PReLU introduces learnable parameters, which can be different for different neurons. The PReLU is defined as:

$$f(x) = \max(\alpha x, x) = \begin{cases} \alpha x, & \text{if x} < 0 \\ x, & \text{if x} \geq 0 \end{cases} \tag{2.3}$$

When α is a small constant, PReLU can be regarded as Leaky ReLU.

S-shaped Rectified Linear Unit SReLU, also known as S-shaped ReLU, is a variant of the traditional ReLU activation function. It has been shown to be capable of learning both convex and non-convex functions through the utilization of three piecewise linear functions [11], formulated using four learnable parameters. This feature enables SReLU to have a more expressive capacity, allowing it to model more complex relationships in the data. The SReLU is defined as:

$$f_{t_l,\alpha_l,t_r,\alpha_r}(x) = \begin{cases} t_l + \alpha_l \left(x - t_l\right), & \text{if} \quad x \leq t_l \\ x, & \text{if} \quad t_l < x < t_r \\ t_r + \alpha_r \left(x - t_r\right), & \text{if} \quad x \geq t_r \end{cases} \tag{2.4}$$

t_l,α_l,t_r,α_r are learnable parameters.

2.2 Multilayer Perceptron

Our experiments are based on MLP and in this section, we briefly introduce their basic structure. MLPs are simple enough and can be used in a variety of fields. Although their accuracy may not be the best for image processing, it is sufficient and fast enough for theoretical verification. A simple N-layer MLP is illustrated in Figure 2.2. The network is divided into an input, hidden, and output layer. The first layer is the input layer, the last layer is the output layer, and the middle part is the hidden layer. Assuming that the output of the Nth layer is denoted as z, the input

vector as x, and the corresponding weight matrices and bias vectors as W and b, respectively, the output of the Nth layer can be denoted as equation 2.5, where f is the activation function.

$$z = f\left(Wx + b\right) \tag{2.5}$$

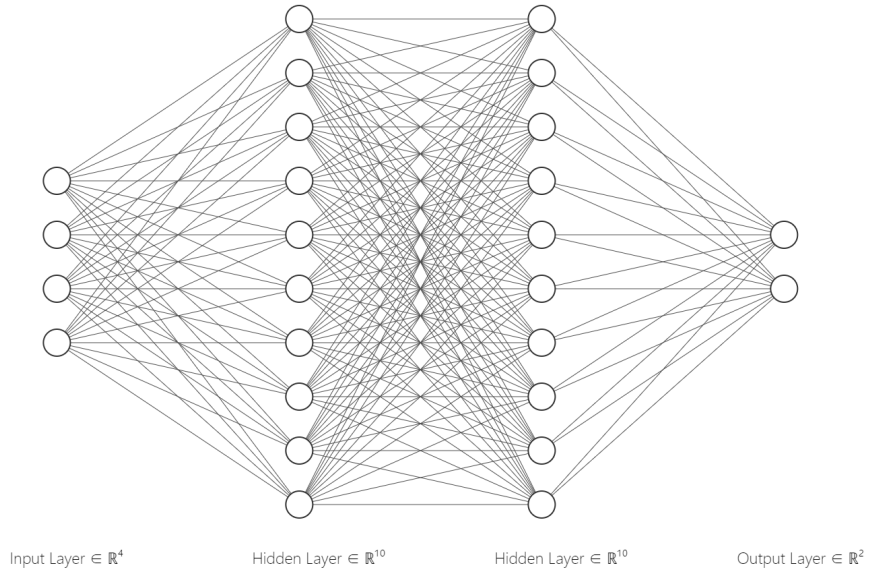


Figure 2.2: A simple MLP structure, it consists of an input layer, two hidden layers and an output layer.

The learning process of MLP is to continuously update the network weight matrix W and bias vector b based on the input data so that the network's output is as close as possible to the true output. Assuming that the loss function is L and the network aims to minimize the loss function, we use the gradient descent algorithm to update the weights equation.

$$\begin{aligned} W &\leftarrow W - \eta \frac{\partial L}{\partial W} \\ b &\leftarrow b - \eta \frac{\partial L}{\partial b} \end{aligned} \tag{2.6}$$

Where η is the learning rate, which controls the length of the iteration step, if it is too large, the training loss function will tend to oscillate, and if it is too small, the convergence rate will be slow, and it will not be easy to find the optimal solution. For classification problems, the loss function is usually Cross Entropy; for regression problems, the loss function is usually Squared Error. Our datasets are classification problems, so we use Cross Entropy in our experiments.

Chapter 3

Related works

As deep learning models grow more extensive and data-rich, sparse is essential. Especially in the traditional image, video, and other data scenarios, where the images contain a large amount of redundant information, sparse is a straightforward approach. While this thesis looks at the study of activation functions in sparse neural networks, this section will introduce what sparse neural networks are, the methods of sparsification, and introduce the activation functions used in previous research to train sparse neural networks.

3.1 Sparse neural networks and sparsification

The concept of sparse neural networks is inspired by biology [8]. The connection of biological neurons is sparse, but the energy efficiency is very high. Sparsity is key to how information is stored, and processed [19]. Therefore, the sparse neural network is different from the dense neural network. Only partial connections are reserved, which can significantly reduce the redundant parameters in the network [20] [21].

There are various methods of sparsification, as Figure 3.1, and in general they can be divided into two main categories model sparsity and ephemeral sparsity. Model sparsity means changing the structure of the model itself, such as changing weights and neurons. Ephemeral sparsity, such as dropout and ReLU, implies sparsity in the training and inference process. And it is common for model sparsity and ephemeral sparsity to be utilized together in many implementations of sparse algorithms. In recent years, many methods have been proposed to achieve model sparsity, and there are two main approaches, pruning, and sparse training.

Pruning. Obtain sparsity and compress the network by pruning some of the weights. It has commonly been assumed that reducing the redundant weights may have been an essential factor in the effectiveness of SNN. So some methods to get sparse neural networks by removing weights that have little effect on network

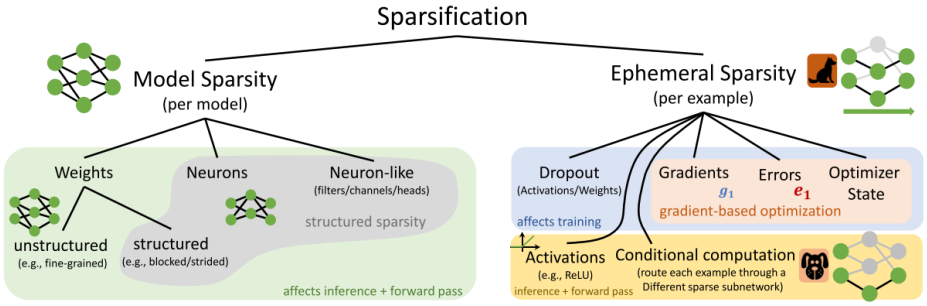


Figure 3.1: An overview of sparsification approaches in deep learning [22].

performance [23] [24].

The pruned model reduces storage requirements and improves the computational performance of inference without compromising accuracy [10] and is highly resistant to noise and interference [25].

However, one limitation of pruning is that we still need to train a dense network, so the maximum computation has not decreased. To solve this problem, researchers hope to train sparse neural networks from scratch.

Dynamic Sparse training is a class of algorithms that are utilized for training sparse neural networks from scratch. These algorithms have the ability to dynamically adjust the weights during the training process. The approach of Dynamic Sparse Training typically involves a combination of pruning and growth techniques, and various strategies may be employed depending on the specific method used. In this thesis, we present some of the more successful algorithms within this class.

The Sparse Evolutionary Training (SET) method is a dynamic approach for adjusting the connections structure of a neural network during the training process, after each training epoch, SET involves the deletion of connections with weights that are close to zero, as well as the random addition of new connections equal to the number of deletions. The utilization of SET in subsequent research within this thesis is motivated by its two main advantages: its simplicity of strategy and its capacity for achieving true sparsity as opposed to simulating sparsity through the use of binary masks.

In 2019, the Dynamic Sparse Reparameterization (DSR) [26] algorithm was introduced, extending the capabilities of the Sparse Evolutionary Training (SET) method from fully-connected layers to convolutional layers. This was achieved through the implementation of simple modifications. The DSR algorithm allows for the dynamic adjustment of sparse parameters, and the pruning of weights based on predetermined thresholds. Subsequent research has also demonstrated the importance of the free redistribution of parameters between layers for the effectiveness of dynamic sparse algorithms. This is exemplified in [27] where the weights are rearranged

according to the average momentum of each layer.

3.2 Activation functions in sparse neural networks

Most of the research focused on the network structure and training strategy of the SNN. The activation function of sparse neural networks mostly follows the default settings of dense networks, and few people explore the activation function of sparse neural networks. The unsuitable activation function can lead to the loss of input information during forward propagation, and the vanishing or exploding gradient back-propagation [28].

The available research shows that SReLU [11] performs better than ReLU on sparse neural networks [6] [29] [30]. However, SReLU requires four additional parameters, which means lots of extra computing resources. Follow-up research [12] based on SReLU, proposed an improved activation function All-ReLU, which achieves comparable performance to SReLU without introducing additional parameters. All-ReLU is defined as follows for each layer l:

$$f_l(x) = \begin{cases} -\alpha x & x \leq 0 \quad \& \quad l\%2 == 0 \\ \alpha x & x \leq 0 \quad \& \quad l\%2 == 1 \\ x & x > 0 \end{cases} \tag{3.1}$$

Nevertheless, the parameters of All-ReLU are fixed. It can only be set based on experience, which weakens the generalization ability. We have noticed that PReLU can adaptively learn and correct the parameters of the linear unit. Although a new parameter is introduced, the parameter is calculated according to the gradient and can be directly updated during the network update process. We propose a new ReLU variant based on All-ReLU and PReLU to achieve comparable performance to All-ReLU with negligible computational cost.

Chapter 4

Dying ReLU problem

In this section we answer the Research question 1 through a theoretical analysis, we explain why the dying ReLU problem arises, why ReLU is unsuitable for sparse neural networks, and the impact of the dying ReLU problem on sparse neural networks.

We assume that the weight matrix in SET is W, the input vector is x, and the output is z. After ReLU activation as a. We know the forward propagation equation as follows:

$$z = \mathrm{W} \cdot x \tag{4.1}$$

$$a = ReLU(z) \tag{4.2}$$

Let the loss function be L. Then the backpropagation equation can be obtained as:

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \tag{4.3}$$

$$\frac{\partial L}{\partial \mathrm{W}} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial \mathrm{W}} \tag{4.4}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial x} \tag{4.5}$$

When the learning rate is fixed, the larger the gradient, the more the weights are updated. It may happen that $z = \mathrm{W} \cdot x$ is less than 0, when the ReLU output $a = 0$. We will give an example to illustrate the impact of sparse neural networks.

Figure 4.1 is a three-layer sparse neural network structure. For convenience, we only study the neurons connected by the red line. The expression for z is:

$$z = \begin{bmatrix} \mathrm{W}_{11} & \mathrm{W}_{21} & \mathrm{W}_{31} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4.6}$$

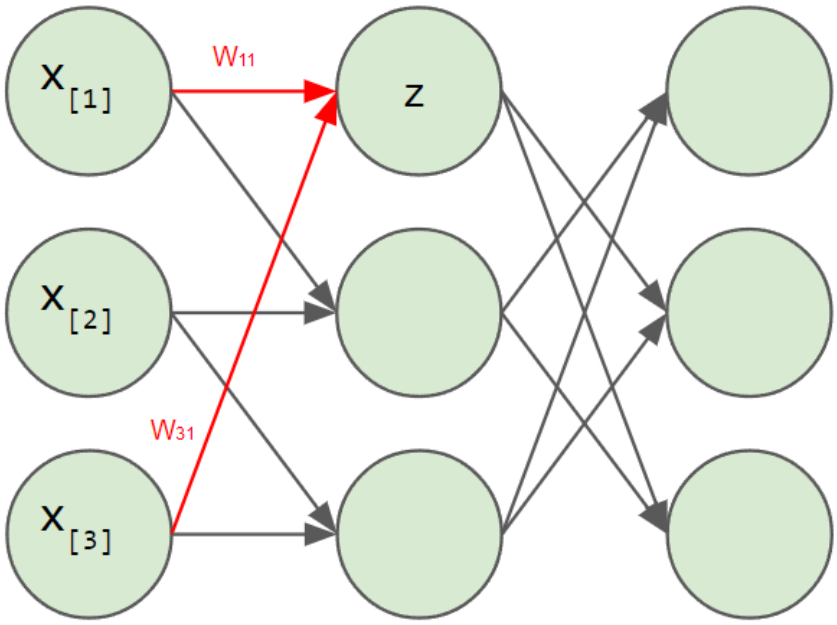


Figure 4.1: A simple three-layer sparse neural network

Where $\mathrm{W}_{21} = 0$, because there is no connection between neural, and we can also write the equation in the pure sparse equation:

$$z = \mathrm{W}^* \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4.7}$$

Where W^* is Compressed Sparse Row (CSR) matrix. CSR contains three arrays:
$data$, which stores the values of the non-zero elements of the matrix.
col, the ith element records the number of columns of the elements of $data$[i].
row, the ith element records the number of non-zero elements contained in the first i-1 rows.

Assuming that a data point, denoted as z, falls within a negative interval, the application of ReLU activation function will result in an output of 0 and a derivative of 0. This can lead to the dying ReLU problem in backpropagation, as the error cannot be passed to the weights. It is worth noting that, while a single step in optimization algorithms such as Stochastic Gradient Descent (SGD) involves multiple data points, this issue can still arise. However, as long as not all data points fall within negative intervals, the output can still be obtained from the ReLU activation function.

However, when the learning rate is too high or when the majority of the data being learned is negative, the problem of dying ReLU can arise. This makes it difficult for

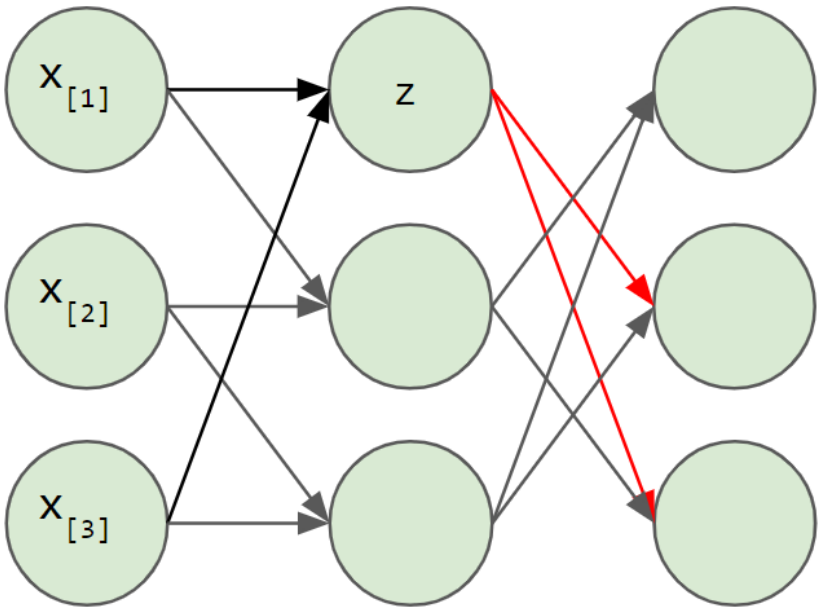


Figure 4.2: A simple three-layer sparse neural network

neurons to recover from this state without human intervention. In a dense neural network, with FashionMNIST dataset, ReLU can improve sparsity by limiting the redundant information passed through the network as long as the learning rate is set appropriately to avoid large areas of dead neurons.

In contrast, when used in a sparse neural network, the sparsity induced by ReLU can lead to a further reduction in network updates and performance. This is because when the neuron input is negative, the product of the neuron output and the weight is 0, resulting in the neuron being unable to connect to the neuron in the next layer. When the dying ReLU problem occurs, it affects the efficiency of weight updating in SET. Dead neurons need to wait for the weights to be deleted and randomly added before they can be recovered, and if the weight connected to the dead neuron is negative and has a large absolute value, it will take a long time for the weight to be removed.

In conclusion, ReLU activation function can perform well in dense neural networks with a lot of redundant information, but in sparse neural networks it can make the network even sparser when it is highly sparse, affecting network updates and performance.

Chapter 5

Proposed methodology

In this section, we describe how to answer the remaining research questions. For the Research question 2, we present a new evaluation method 5.1. For the Research question 3. We have designed some experiments to make the comparison 5.2.

5.1 Gradient flow

In previous research, many researchers have studied the performance of activation functions using metrics for evaluating models, for example, accuracy, precision, etc. However, recently some papers have shown that the gradient flow can judge the model [7] and the activation function [12], so we would like to transfer this to our thesis and evaluate the activation function in terms of whether it improves the gradient flow.

The gradient flow is used to optimize dynamics, and it is approximated by taking the norm of the gradients of the network [31], and the formula is as follows:

$$\boldsymbol{g} = \frac{\partial L}{\partial \mathrm{W}} \tag{5.1}$$

$$gf_p = \|\boldsymbol{g}\|_p \tag{5.2}$$

where L is cost function, θ is network weights, p denotes the ℓ_p norm and gf_p is the gradient flow. In this thesis we use ℓ_1 norm.

The standard gradient flow formulation could result in some weakness and loss of information and give disproportionate influence to layers with more weights [7].

To address these shortcomings we choose a new method Effective Gradient Flow (EGF) [7]. EGF computes the average, masked(active weights) gradient norm across all layers. Formulate as follow:

$$\begin{aligned} \boldsymbol{g} &= \left(\frac{\partial L}{\partial \mathrm{W}^1}, \frac{\partial L}{\partial \mathrm{W}^2}, \ldots, \frac{\partial L}{\partial \mathrm{W}^N} \right) \\ EGF_p &= \frac{\sum_{n=1}^{N} \|\boldsymbol{g}_n\|_p}{N} \end{aligned} \tag{5.3}$$

Where N is the number of layers. We will show that the activation function changes the gradient flow and thus affects the accuracy. Meanwhile, using EGF we can choose the optimal activation function of sparse neural networks.

5.2 Experimental design

To answer the second research question, we designed three experiments.

Firstly, we experimented with the accuracy and EGF of the four activation functions on three different datasets. We then reported the mean and standard deviation of the accuracy and EGF after multiple experiments, as seen in Tables 7.1, 7.2 and Figure 7.1.

Next, we experimented with the accuracy of the four activation functions on the Madelon dataset with different sparsity levels, as shown in Figure 7.2.

Finally, we experimented with the accuracy of GAReLU, All-ReLU, and Leaky ReLU for different initial slopes. We then reported the final slope learned by GAReLU, as seen in Tables 7.3 and 7.4.

Chapter 6

Experiments

This chapter describes in detail the experimental environment, structure and parameter settings, strategy, and the datasets used.

6.1 Experimental environment

We have implemented the framework on a PC with an Intel Core i7-11700 2.50GHz CPU, 32GB RAM, and RTX 3060ti 8GB GPU. Anaconda3 is used to manage packages. We use the SciPy library to implement the sparse matrix rather than a binary mask. We consider a simple scenario for the network: a five-layer MLP, one input layer, three hidden layers, and one output layer. Furthermore, train it using a momentum stochastic gradient descent algorithm to accelerate convergence. The pseudo-code for the three experiments are as follows: algorithm 1,2, 3.

Algorithm 1: Experiment 1

```
1 Initializing Parameters;
2 for each datasets do
3     for each activation function do
4         create model;
5         train the model;
6         save results;
7 produce the result;
```

Algorithm 2: Experiment 2

```
Initializing Parameters;
for each epsilon do
    for each activation function do
        create model;
        train the model;
        save results;
produce the result;
```

Algorithm 3: Experiment 3

```
Initializing Parameters;
for each slope do
    for each activation function do
        create model;
        train the model;
        save results;
produce the result;
```

6.2 Structure and parameter settings

We have created a framework to compare several activation functions and decide on a simple but effective model. The code for our sparse neural network is adapted from [12], and it is based on SET algorithm 4.

Table 6.1 gives each experiment's model structure and parameter settings.

Dataset	Architecture	Hyper-parameters					
		ϵ	learning rate	batch size	slop α	Momentum	Dropout
FashionMNIST	784-1000-1000-1000-10	20	0.01	128	0.6	0.9	0.3
CIFAR10	3072-4000-1000-4000-10	20	0.01	128	0.75	0.9	0.3
Madelon	500-400-100-400-2	10	0.01	32	0.5	0.9	0.3

Table 6.1: List of structures and hyperparameters used in the experiments

The hyperparamter ϵ determines the probability of the connection between two layers' neurons, thus controlling the sparsity of the network. And ϵ needs to satisfy: $0<\epsilon \ll n_l$ and $\epsilon \ll n_{l-1}$. Because when ϵ is 0, then $sparsity = 1$, there is no

ϵ	1st layer		2nd layer		3rd layer	
	Dimension	Sparsity	Dimension	Sparsity	Dimension	Sparsity
10	500-400	95.50%	400-100	87.50%	100-400	87.50%
20		91.00%		75.50%		75.50%
30		86.50%		62.50%		62.50%
40		82.00%		50.00%		50.00%

Table 6.2: The sparsity of each layer for different ϵ

connection in the network to train, and if ϵ is too large, sparsity will be negative and the sparse neural network will become dense neural network. The sparsity is calculated by 6.1.

$$Sparsity = 1 - \frac{\epsilon\left(n_l + n_{l-1}\right)}{n_l n_{l-1}} \tag{6.1}$$

where n_l and n_{l-1} denote the number of neurons of the lth layer and $l-1$th layer.

Table 6.2 represents the sparsity of the network under different ϵ.

6.3 Training strategies

This section describes the strategies we use in the training process.

Dropout Overfitting is a common problem in machine learning, where the prediction accuracy of a model increases in the training set but decreases in the test set, indicating that the model does not generalize well and only memorizes the features of the current data rather than generalizing. In neural networks, there are often more parameters than the training data, leading to overfitting. One solution to overfitting is Dropout [32] [33], a regularization method in which the output nodes of a layer of a neural network are randomly dropped with probability $1-p$ during training. When tested, all values are retained. The essence of this method is to create a large number of new random samples, increasing the sample size and reducing the number of features to prevent overfitting. Moreover, in our experiments, we set p to 0.3.

Stochastic Gradient Descent with Momentum The difficulty of finding neural network parameters is another central pain point of neural networks. Neural networks are not a convex optimization problem. The solutions obtained using Stochastic Gradient Descent (SGD) are usually not globally optimal but locally optimal, and the final solutions may vary greatly depending on the initial learning rate setting. It has been shown that a neural network may have multiple local optima to achieve good

classification, but the global optimum is prone to overfitting. In order to optimize the shortcomings of SGD, we use the Momentum algorithm, which borrows the concept of momentum from physics. It stimulates the inertia of an object as it moves, i.e., The update retains the direction of the previous update to some extent while fine-tuning the final update direction using the gradient of the current batch. This increases stability to a certain extent, leading to faster learning and the ability to escape from local optima. The update rule is as in Equation 6.2, which has been shown to be effective for training sparse models. [12] In our experiments, we set the momentum μ to 0.9.

$$\mathbf{w_{l+1}} = \mathbf{w_l} + \mu\left(\mathbf{w_l} - \mathbf{w_{l-1}}\right) - \eta_l \nabla \mathbf{w_l} \tag{6.2}$$

6.4 Datasets

In this section, we describe the datasets that we used in the experiments.

FashionMNIST FashionMNIST is a dataset of 28x28 greyscale images of clothing. It is more complex than MNIST and therefore provides a better representation of the actual performance of the neural network and a better representation of the dataset used in the real world. It is a 10-class classification problem with 6000 training examples and 10000 test cases, with each image having a resolution of 28x28.

CIFAR10 The CIFAR-10 dataset consists of 10 classes of 32x32 color images, containing a total of 60,000 images, with each class containing 6,000 images. Among them, 50,000 images are used as the training set, and 10,000 images are used as the test set.

Madelon Madelon is an artificial dataset that was part of the NIPS 2003 feature selection challenge. This dataset is a two-class classification problem with continuous input variables. It contains data points in 32 clusters placed on the vertices of a five-dimensional hypercube and randomly labeled +1 or -1.

Chapter 7

Results

7.1 Performance on different datasets

The experiments were conducted on three benchmark datasets to show the evaluation metrics, and we also evaluated the EGF.

All experiments were conducted on the same PC to test the maximum accuracy of the different activation functions on the three datasets and to report the corresponding EGF.

Figure 7.1 (a) shows the accuracy change curves for the four activation functions on the CIFAR10 dataset. The graph shows that the accuracy of the four activation functions rises rapidly in the first 100 epochs. Then Leaky ReLU, All-ReLU, and GAReLU increase slowly after 100 epochs. However, ReLU decreases slowly after 200 iterations, which we consider an overfitting phenomenon. Overall, in terms of accuracy, GAReLU and All-ReLU are comparable and outperform ReLU and Leaky ReLU.

Figure 7.1 (b) shows the EGF curves on the CIFAR10 dataset. All curves climb rapidly to their highest point and then converge in a slow decline. All-ReLU and GAReLU have the highest EGF and remain highest after 500 epochs. The EGF curves for All-ReLU and GAReLU are consistent with the accuracy results. For ReLU and Leaky ReLU curves, ReLU reaches its maximum around the first 100 epochs. In contrast, Leaky ReLU reaches its maximum EGF around 200 epochs, which explains the slightly slower rise in accuracy for Leaky ReLU compared to the other functions in Figure 7.1 (a). We notice that after 500 epochs, the EGF of Leaky ReLU is higher than ReLU, but the accuracy is lower than ReLU. Here, we speculate that the slope setting is the reason for the poor performance of Leaky ReLU, as in most papers and experiments, the slope of Leaky ReLU is generally set at around 0.1-0.2. This highlights that the asymmetric structure of All-ReLU and GAReLU can improve the performance of the activation function. The difference between All-ReLU and Leaky ReLU is that All-ReLU uses a negative slope at the even layers.

Figure 7.1 (c) illustrates the accuracy of the activation functions on the Fashion-MNIST dataset. The difference with the figure (a) is that ReLU does not appear to be over-fitted here.

Figure 7.1 (d) shows the EGF curve on FashionMNIST. GAReLU is slightly better than All-ReLU, both outperforming Leaky ReLU and ReLU.

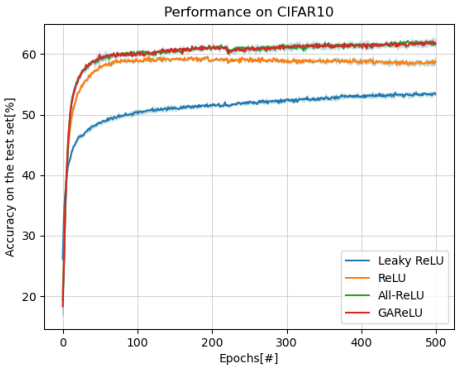
Figure 7.1 (e) shows the accuracy curve of the activation functions on the Madelon dataset. GAReLU achieves the highest accuracy, but there is overfitting after 200 epochs. All four activation functions have similar accuracy rates.

Figure 7.1 (f) shows the EGF curve on Madelon. The highest values are still GAReLU and All-ReLU, but in the end, the EGF of Leaky ReLU is higher than GAReLU and All-ReLU.

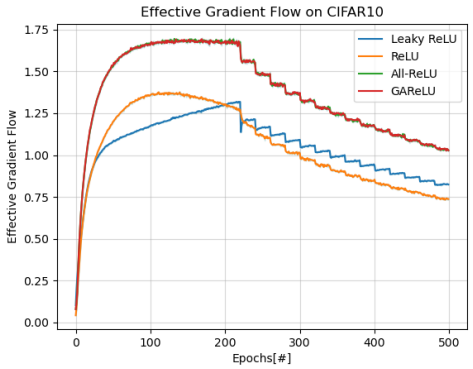
In Table 7.1, we present the maximum value of the accuracy, the maximum EGF, and the ending EGF as a supplement to Figure 7.1. For each result, we conducted three experiments and calculated the average report. The table shows that the accuracy of GAReLU outperforms the other activation functions on all datasets, as expected. On the FashionMNIST dataset, GAReLU's accuracy is slightly higher than All-ReLU, and both are greater than ReLU. On the CIFAR10 dataset, GAReLU's lead is more pronounced due to the higher dimensionality of the dataset. GAReLU also improved by almost 1% over All-ReLU on the Madelon dataset.

Dataset	Architecture	Model	Activation	Accuracy[%]	Maximum EGF	Ending EGF
FashionMNIST	784-1000-1000-1000-10	SET-MLP	GAReLU	**91.08±0.11**	**0.299±0.001**	**0.193±0.002**
			All-ReLU	90.91±0.10	0.296±0.001	0.190±0.000
			ReLU	90.70±0.20	0.222±0.001	0.126±0.001
			Leaky ReLU	88.92±0.12	0.270±0.000	0.171±0.000
CIFAR10	3072-4000-1000-4000-10	SET-MLP	GAReLU	**62.52±0.11**	1.695±0.010	**1.029±0.001**
			All-ReLU	62.29±0.07	**1.698±0.000**	1.028±0.001
			ReLU	59.75±0.00	1.375±0.01	0.735±0.002
			Leaky ReLU	53.66±0.28	1.321±0.000	0.824±0.000
Madelon	500-400-100-400	SET-MLP	GAReLU	**68.33±0.33**	1.128±0.016	**0.285±0.068**
			All-ReLU	67.50±0.31	**1.138±0.034**	0.293±0.010
			ReLU	66.00±1.25	0.933±0.020	0.139±0.015
			Leaky ReLU	65.83±0.27	0.950±0.020	0.459±0.030

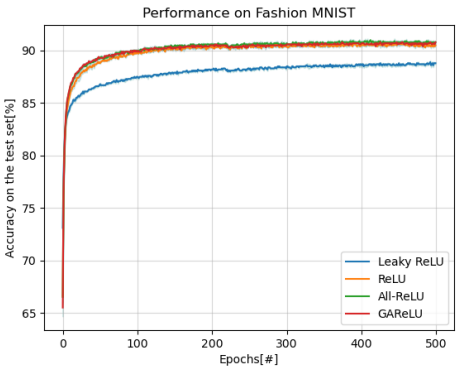
Table 7.1: Activation function results on different data sets. It shows that GAReLU achieved the highest scores on all three datasets. All parameter settings remain the same as in Table 6.1. The slope of GAReLU is updated during training as shown in Table 7.2.



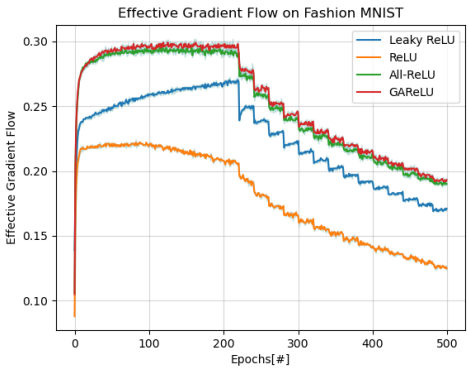
(a) Accuracy on CIFAR10



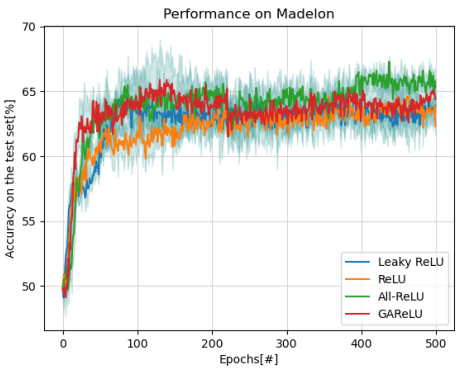
(b) Effective Gradient Flow on CIFAR10



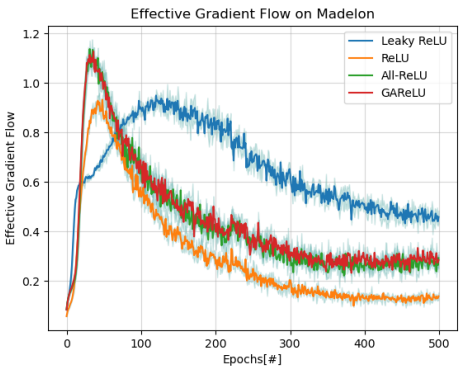
(c) Accuracy on FashionMNIST



(d) EGF on FashionMNIST



(e) Accuracy on Madelon



(f) EGF on Madelon

Figure 7.1: Activation functions performance on different datasets.

Dataset	Activation	Starting Slope			Ending slope		
		1st layer	2nd layer	3rd layer	1st layer	2nd layer	3rd layer
FashionMNIST	GAReLU	-0.6	0.6	-0.6	-0.61	0.05	-0.17
CIFAR10		-0.75	0.75	-0.75	-0.79	0.37	-0.66
Madelon		-0.5	0.5	-0.5	-0.65	-0.09	-0.51

Table 7.2: Starting slope and ending slope of GAReLU on three datasets.

7.2 Performance at different sparsity levels

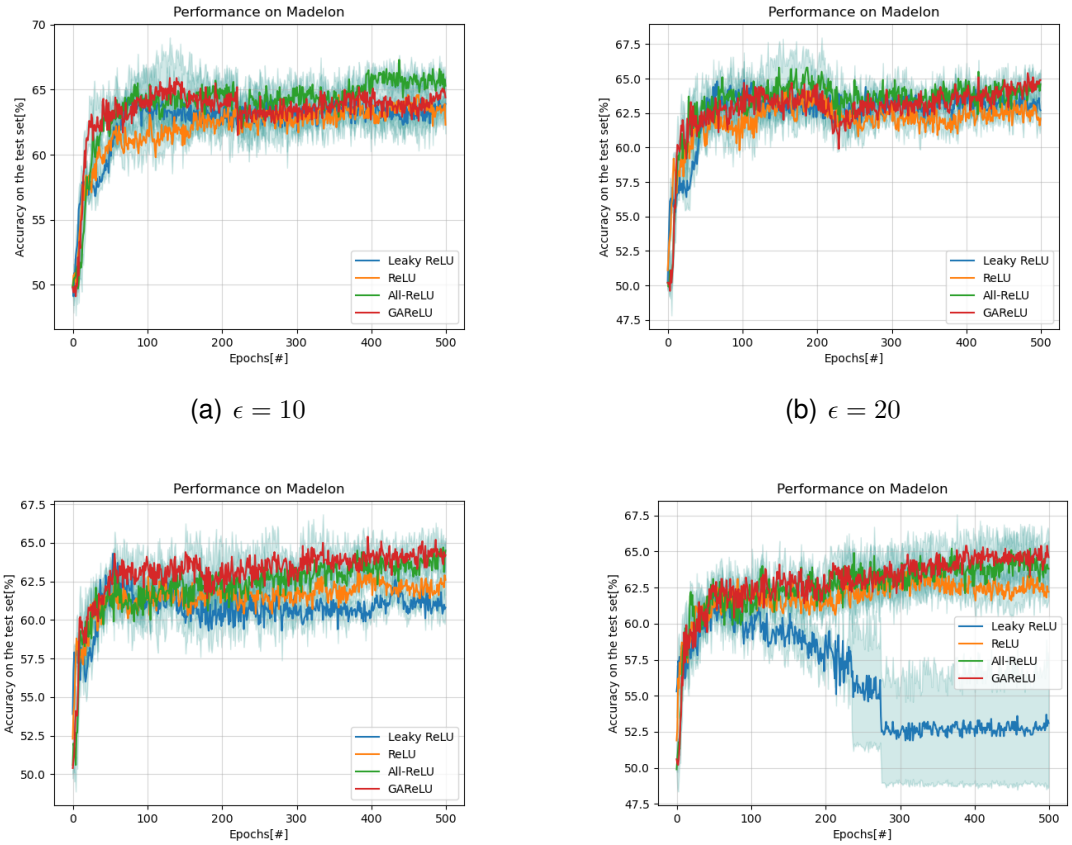
Figure 7.2 shows the accuracy curves for the four activation functions at different sparse levels. The smaller the ϵ value, the higher the sparsity. Our goal is to achieve the highest sparsity level while maintaining a reasonable level of accuracy. The four graphs show that GAReLU slightly outperforms All-ReLU. Both GAReLU and All-ReLU have higher accuracy than ReLU and Leaky ReLU. Leaky ReLU performs the worst; this may be explained by two factors: firstly, the slope setting is not suitable, and secondly, Leaky ReLU retains too much information, leading to a gradient explosion at low sparsity. Figure 7.2 (d) displays that Leaky ReLU experiences a gradient explosion after about 100 epochs.

This once again illustrates that the asymmetry of the GAReLU and All-ReLU structures helps to improve network performance.

7.3 Performance of different slopes on the activation function

Table 7.3 shows the best accuracy achieved for the three activation functions GAReLU, All-ReLU, and Leaky ReLU for different slopes. GAReLU achieves a maximum around 0.5, All-ReLU achieves a maximum around 0.8, and Leaky ReLU achieves a maximum around 0.3.

GAReLU can find the most suitable slope during the training process, so in practice, we recommend setting it at around 0.5. All-ReLU achieves the maximum value around 0.6, which may require prior experience in practice and is slightly inferior to GAReLU. Leaky ReLU achieves the maximum value when the slope is 0.3, which is close to the typical range of 0.1 - 0.2.



(a) $\epsilon = 10$ (b) $\epsilon = 20$

(c) $\epsilon = 30$ (d) $\epsilon = 40$

Figure 7.2: Comparison of activation functions with different spasity level, and the initial slope is 0.5.

Activation	Best accuracy[%]								
	α=0.1	α=0.2	α=0.3	α=0.4	α=0.5	α=0.6	α=0.7	α=0.8	α=0.9
GAReLU	67.33	67.50	67.33	66.83	**68.67**	67.83	68.17	67.33	67.50
All-ReLU	66.00	65.83	67.33	67.50	67.50	**68.17**	66.17	67.33	66.17
Leaky ReLU	65.67	65.33	**66.67**	66.17	65.50	66.50	65.00	63.00	63.00

Table 7.3: Best accuracy for different slopes(α) on Madelon, GAReLU keeps the same initial α as the other activation functions.

Dataset	Starting Slope			Ending slope		
	1st layer	2nd layer	3rd layer	1st layer	2nd layer	3rd layer
Madelon	-0.1	0.1	-0.1	-0.54	-0.46	-0.65
	-0.2	0.2	-0.2	-0.60	-0.60	-0.79
	-0.3	0.3	-0.3	-0.65	-0.51	-0.81
	-0.4	0.4	-0.4	-0.66	-0.49	-0.79
	-0.5	0.5	-0.5	-0.68	-0.47	-0.82
	-0.6	0.6	-0.6	-0.67	-0.43	-0.83
	-0,7	0.7	-0,7	-0,73	-0,34	-0.80
	-0.8	0.8	-0.8	-0.79	-0.09	-0.56
	-0.9	0.9	-0.9	-0.86	0.01	-0.72

Table 7.4: Ending slope of GAReLU on Madelon.

Chapter 8

Discussion

In this section, we explain and analyze the results of the study, provide a more in-depth analysis of the results and their implications, and discuss future work that should be considered.

Firstly, we found that the sparsity of the ReLU affects the sparsity of the sparse neural network. This was obtained through theoretical analysis. When the input of ReLU is negative, the neuron can not pass the information to the neuron in the next layer, which means that the neuron is disconnected from the neuron in the next layer and the sparsity of the network will increase. However, this analysis is only theoretical, and we are unable to experimentally determine the exact sparsity of the network, which is a drawback of this thesis. We are also unable to give an estimate of how much sparsity above which ReLU will have a negative impact. However, we can say for sure that we should not use ReLU when the highest accuracy in sparse neural networks is expected.

Secondly, this thesis uses gradient flow to show that sparse neural networks with good gradient flow have better performance. However, it is not possible to establish a deeper direct connection between the activation function and the gradient flow. In the future, a mathematical derivation will be needed to reveal the relationship between the activation function and the gradient flow. We found that All-ReLU differs only slightly from Leaky ReLU in that the slope of All-ReLU is negative in the even layers. This asymmetric structure also improves the EGF, but we cannot explain exactly why.

GAReLU performs well due to its asymmetric structure and its ability to learn a suitable slope. However, it also has a corresponding disadvantage in that an inappropriate learning rate can lead to a failure to learn a suitable slope, and choosing a suitable learning rate takes extra time.

Additionally, for Figure 7.1 (a), (c), we found that the accuracy curve of Leaky ReLU is smaller than that of ReLU. According to our theoretical analysis, Leaky ReLU does not affect network sparsity, and there is no dying ReLU problem, the ac-

curacy of Leaky ReLU should be higher than ReLU. We believe this may be related to the setting of slope, except for ReLU, the initial slope of the other three activation functions is 0.75 in CIFAR10 and 0.6 in FashionMNIST, while the slope of Leaky ReLU is usually small, so we think it may be that the slope affects the performance of Leaky ReLU.

Finally, it's worth noting that the experiments in this thesis have some limitations. The code only runs on the CPU and the large number of for loops slows down the runtime. And, we do not discuss the initialization of the weights. Our experiments focus only on multilayer perceptrons and do not extend to other types of networks such as CNNs, RNNs, and use only two types of datasets, image datasets and tabular datasets, without considering datasets such as text, audio, time-series, etc. These limitations should be considered in future work.

In summary, the main innovations in this thesis are:

1. The analysis of the problems of ReLU in sparse neural networks from a theoretical perspective, particularly the dying ReLU problem and its effect on sparse training.

2. The use of EGF as a criterion to compare the performance of different activation functions, which provides a better explanation for why the activation function performs well, and can guide future research in this area. Most previous work has only compared accuracy.

3. The proposal of a new activation function, GAReLU, which achieves better performance than other activation functions by introducing an additional learnable parameter, and the excellent classification performance of GAReLU is verified experimentally.

Chapter 9

Conclusion

Through theoretical analysis and experiments, this thesis explains why ReLU is not suitable for sparse neural networks, proposes GAReLU, a new activation function designed for sparse neural networks, and demonstrates that GAReLU outperforms the other three activation functions through a new evaluation metric, EGF. By successfully answering the proposed research questions, we arrive at the following conclusions:

For the first research question, our theoretical analysis indicate that the sparsity of ReLU results in Sparse neural networks with more sparsity than expected, and that the dying ReLU problem affects the weight update of SET. Therefore, we advocate for not using ReLU in sparse neural networks with high sparsity.

For the second research problem, we use EGF in evaluating the activation function. Previous studies have used gradient flow improvement to demonstrate the performance improvement of sparse neural networks, and we adopt this idea to evaluate the activation function. We found that the change of the activation function affects the gradient flow and thus affects the performance of sparse neural networks.

For the third research question, our proposed activation function GAReLU outperforms the other three activation functions in most experiments. GAReLU avoids the problems that exist with ReLU, it does not change the sparsity of the network, there is no dying ReLU problem, and it can learn the appropriate parameters on its own without the need for expensive grid searches of the parameters.

In conclusion, this thesis has explored the activation function for sparse neural networks, discussed the problems faced by ReLU in sparse neural networks, how to evaluate the activation function, and proposed a new excellent activation function that opens up new possibilities for further research in the field of sparse neural networks. We hope that our work will influence others and inspire future research in this field.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[5] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 445–454.

[6] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, pp. 1–12, 2018.

[7] K.-a. Tessera, S. Hooker, and B. Rosman, "Keep the gradients flowing: Using gradient flow to study sparse network optimization," *arXiv preprint arXiv:2102.01670*, 2021.

[8] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[9] H. N. Mhaskar and T. Poggio, "Deep vs. shallow networks: An approximation theory perspective," *Analysis and Applications*, vol. 14, no. 06, pp. 829–848, 2016.

[10] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[11] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[12] S. Curci, D. C. Mocanu, and M. Pechenizkiyi, "Truly sparse neural networks at scale," *arXiv preprint arXiv:2102.01732*, 2021.

[13] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Citeseer, 2013, p. 3.

[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[17] R. ZahediNasab and H. Mohseni, "Neuroevolutionary based convolutional neural network with adaptive activation functions," *Neurocomputing*, vol. 381, pp. 306–313, 2020.

[18] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[19] S. Ahmad and J. Hawkins, "How do neurons operate on sparse distributed representations? a mathematical theory of sparsity, neurons and active dendrites," *arXiv preprint arXiv:1601.00720*, 2016.

[20] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," *Advances in neural information processing systems*, vol. 26, 2013.

[21] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[22] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks." *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1–124, 2021.

[23] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[25] S. Ahmad and L. Scheinkman, "How can we be so dense? the benefits of using highly sparse representations," *arXiv preprint arXiv:1903.11257*, 2019.

[26] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4646–4655.

[27] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *arXiv preprint arXiv:1907.04840*, 2019.

[28] S. Hayou, A. Doucet, and J. Rousseau, "On the impact of the activation function on deep neural networks training," in *International conference on machine learning*. PMLR, 2019, pp. 2672–2680.

[29] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[30] A. Dubowski, "Activation function impact on sparse neural networks," *arXiv preprint arXiv:2010.05943*, 2020.

[31] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 794–803.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

Appendix A

AppendiA

A.1 Proposed algorithm

Algorithm 4: the bolded lines were added based on SET

input : An ANN model with hyper-parameters
output: Trained model and result

%Initialization;
initialize ANN model;
set ε *and* ζ;
for *each fully-connected (FC) layer* **do**
    replace FC with a Sparse Connected (SC) layer having a ER topology given by Eq;
initialize training algorithm parameters;
for *each training epoch e* **do**
    forward Propagation with new activation function;
    perform weights update;
    update α ***of GAReLU***;
    calculating gradient flow;
    for *each bipartite SC layer ofthe ANN* **do**
        remove a fraction ζ *of the smallest positive weights*;
        remove a fraction ζ *of the largest negative weights*;
        if *e is not the last training epoch* **then**
            add randomly new weights in the same amount as the ones removed previously